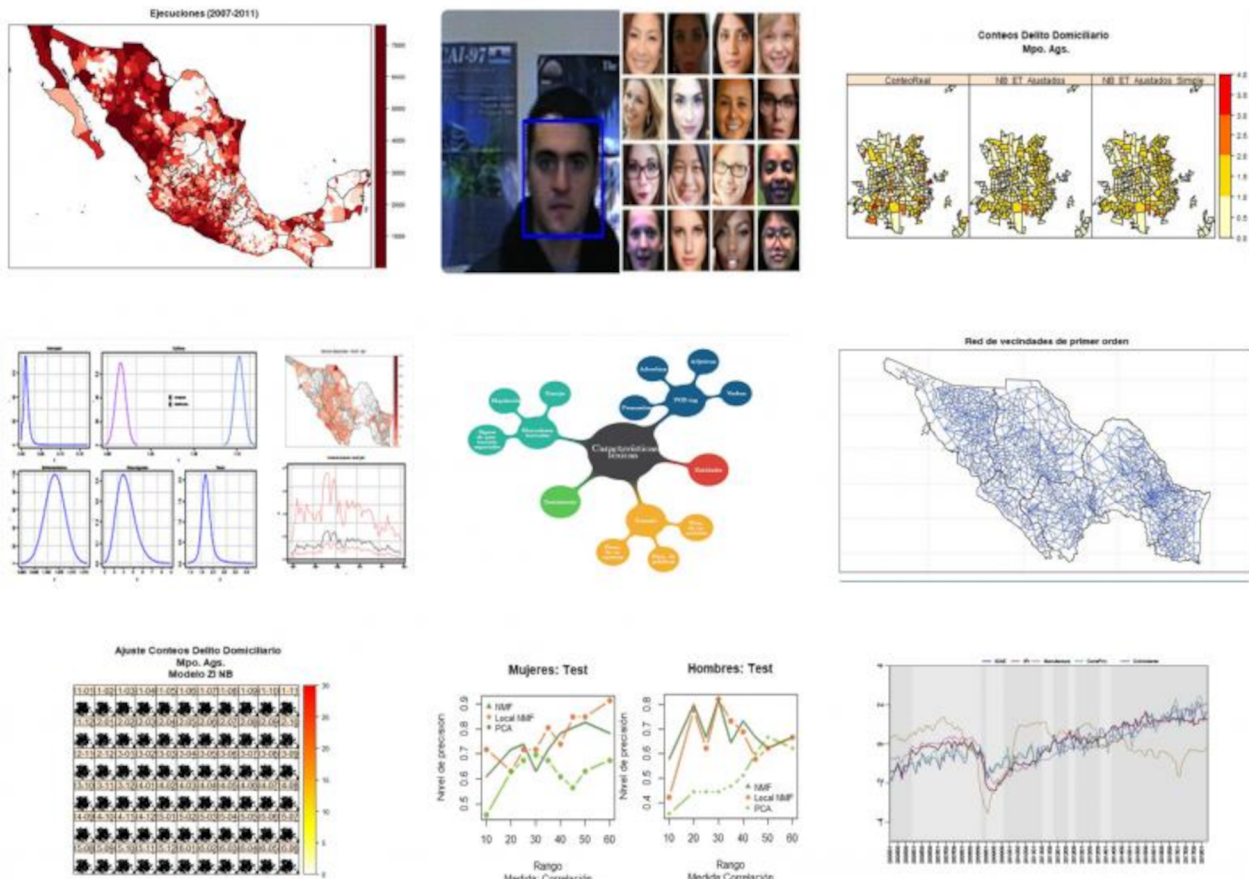


Introducción al Cómputo Científico



Antonio Carrillo Ledesma

Introducción al Cómputo Científico

Antonio Carrillo Ledesma

Facultad de Ciencias, UNAM

<http://academicos.fcencias.unam.mx/antoniocarrillo>

La última versión de este trabajo se puede descargar de la página:

<https://sites.google.com/ciencias.unam.mx/acl/en-desarrollo>

<http://132.248.181.216/acl/EnDesarrollo.html>

2025, Versión 1.0 α ¹

¹El presente trabajo está licenciado bajo un esquema Creative Commons Atribución CompartirIgual (CC-BY-SA) 4.0 Internacional. Los textos que componen el presente trabajo se publican bajo formas de licenciamiento que permiten la copia, la redistribución y la realización de obras derivadas siempre y cuando éstas se distribuyan bajo las mismas licencias libres y se cite la fuente. ¡Copia este libro! ... Compartir no es delito.

Índice

1	Introducción	4
1.1	Algo de Programación	5
1.2	Cómputo Científico	11
1.3	FLOPS	16
1.4	Agradecimientos	18
2	Aritmética de Punto Flotante	20
2.1	Aritmética de Punto Flotante IEEE	25
2.2	Trabajando con Punto Flotante	43
2.3	Aritmética de Baja Precisión	72
3	Entornos de Desarrollo y Herramientas de Programación	76
3.1	Java	79
3.2	Python	88
3.3	Julia	104
3.4	C y C++	107
3.5	Fortran	114
3.6	R	119
3.7	Herramientas de Programación	121
3.7.1	¿Qué es eso de ASCII, ISO-8859-1 y UTF-8?	127
3.7.2	Uso de Espacios o Tabuladores en Fuentes	131
3.7.3	Comparar Contenido de Fuentes	133
3.7.4	Astyle	134
3.7.5	Compilación y la Optimización del Ejecutable	135
3.7.6	Análisis de Rendimiento y Depuración	140
3.7.7	Mejora del Rendimiento en Python	144
3.7.8	Git	150
3.7.9	GitLab vs GitHub	169
3.7.10	Otras opciones	174
3.8	Programando Desde la Nube	176
4	Visualización Gráfica de Datos	180
4.1	Python	182
4.2	SAC	212
4.3	Sistema de Información Geográfica	213
4.4	Generar Mallas con GMSH	216

5	Programación en Paquetes de Cálculo Numérico	222
5.1	Cálculo Numérico con Octave	225
5.2	Trabajando con Octave	226
5.3	Desde la Nube	235
6	Programación en Paquetes de Cálculo Simbólico	236
6.1	Cálculo Simbólico con Maxima	238
6.2	Trabajando con Maxima	240
6.3	Desde la Nube	246
7	Programación en Paquetes Estadísticos	247
7.1	Cálculo Estadístico con R	249
7.2	Trabajando con R	250
7.2.1	R y Python	254
7.3	Desde la Nube	255
8	El Cómputo en Paralelo	256
8.1	Computadoras Actuales	260
8.2	¿Que es Computación Paralela?	303
8.3	Clasificación Clásica de Flynn	309
8.4	Categorías de Computadoras Paralelas	312
8.4.1	Equipo Paralelo de Memoria Compartida	313
8.4.2	Equipo Paralelo de Memoria Distribuida	316
8.4.3	Equipo Paralelo de Memoria Compartida-Distribuida	317
8.4.4	Cómputo Paralelo en Multihilos	323
8.4.5	Cómputo Paralelo en CUDA	324
8.5	Métricas de Desempeño	330
8.6	Programación de Cómputo de Alto Rendimiento	334
8.6.1	Programando con OpenMP para Memoria Compartida	336
8.6.2	Programando con MPI para Memoria Distribuida	339
8.6.3	Esquema de Paralelización Principal-Subordinados	344
8.6.4	Opciones de Paralelización Híbridas	346
8.7	Programando Desde la Nube	348
9	Ejemplitos	350
9.1	Precisión Arbitraria	353
9.2	Sistemas Dinámicos Discretos	360
9.3	Fractales	364

9.4	Derivadas	371
9.5	Integrales	378
9.6	Matrices y Vectores	387
9.7	Resolución de Sistemas Lineales	404
9.8	Ecuaciones Diferenciales Ordinarias	433
9.9	Ecuaciones Diferenciales Parciales	445
10	Apéndice A: Software Libre y Propietario	460
10.1	Software Propietario	463
10.2	Software Libre	465
10.3	Seguridad del Software	472
10.4	Tipos de Licencias	475
10.4.1	Licencias Creative Commons	481
10.4.2	Nuevas Licencias para Responder a Nuevas Necesidades	483
10.5	Implicaciones Económico-Políticas del Software Libre	486
10.5.1	Software Libre y la Piratería	486
10.5.2	¿Cuánto Cuesta el Software Libre?	487
10.5.3	La Nube y el Código Abierto	490
10.5.4	El Código Abierto como Base de la Competitividad	493
10.5.5	Software Libre en Empresas y Corporaciones	494
10.6	Código Abierto y las Organizaciones Internacionales	502
10.6.1	Las Naciones Unidas y el Código Abierto	503
10.6.2	La Comisión Europea se Compromete a Liberar Todo el Software que Pueda Beneficiar a la Sociedad	504
11	Bibliografía	507

1 Introducción

Para muchas personas en Ciencias e Ingenierías, el cómputo científico y las matemáticas aplicadas son principalmente física matemática. La física matemática es principalmente ecuaciones diferenciales. La solución numérica de ecuaciones diferenciales se reduce al álgebra lineal. Por lo tanto, el corazón de las matemáticas aplicadas es el álgebra lineal.

Si bien, hay mucho de verdad en el resumen anterior. El álgebra lineal es muy importante y gran parte de las matemáticas aplicadas dependen en última instancia de soluciones eficientes de grandes sistemas lineales. El eslabón más débil del argumento puede ser el primero: las matemáticas aplicadas son mucho más que la física matemática. La física matemática no ha decaído, pero otras áreas han crecido. Aun así, las áreas de las matemáticas aplicadas fuera de la física matemática y fuera de las ecuaciones diferenciales a menudo dependen críticamente del álgebra lineal.

Sin duda recomendaría que alguien interesado en las matemáticas aplicadas se familiarice con el álgebra lineal numérica. Si quieres ser un experto en ecuaciones diferenciales, optimización o en muchos otros campos, necesitas estar al menos familiarizado con el álgebra lineal numérica si vas a calcular algo. Como señala Stephen Boyd en su clase de optimización convexa, muchos de los avances en optimización de los últimos 20 años tienen en su núcleo avances en el álgebra lineal numérica. Los algoritmos mejorados han acelerado la solución de sistemas muy grandes más que la ley de Moore.

Puede parecer cuestionable decir que el álgebra lineal está en el corazón de las matemáticas aplicadas porque es lineal. ¿Qué pasa con las aplicaciones no lineales, como las ecuaciones diferenciales parciales no lineales? Las ecuaciones diferenciales no lineales conducen a ecuaciones algebraicas no lineales cuando se discretizan. Pero estos sistemas no lineales se resuelven mediante iteraciones de sistemas lineales, por lo que volvemos al álgebra lineal.

En el estudio de los sistemas continuos -sistemas físicos macroscópicos-, tales como los yacimientos petroleros, la atmósfera, los campos electromagnéticos, los océanos, el aparato circulatorio de los seres humanos, la corteza terrestre y muchos otros sistemas de interés en Ciencia y en Ingeniería, al modelarse contienen un gran número de grados de libertad¹.

¹El número de grados de libertad en un sistema físico se refiere al número mínimo de números reales que es necesario especificar para determinar completamente el estado físico.

1.1 Algo de Programación

Tipos de datos Dependiendo del lenguaje (véase [12], [13], [14] y [15]), la implementación del mismo y la arquitectura en la que se compile/ejecute el programa, se tienen distintos tipos de datos, pero los básicos son²:

- char, 8 bits, rango de -128 a 127
- unsigned char, 8 bits, rango de 0 a 255
- int, 16 bits, rango de -32,768 a 32,767
- unsigned int, 16 bits, rango de 0 a 65,535
- long int, 32 bits, rango de -2,147,483,648 a 2,147,483,647
- long, 64 bits, rango de -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
- float, 32 bits, rango de -3.4×10^{38} a 3.4×10^{38} , con 14 dígitos de precisión
- double, 64 bits, rango de -1.7×10^{308} a 1.7×10^{308} , con 16 dígitos de precisión
- long double, 80 bits, 3.4×10^{-4932} a 3.4×10^{4932} , con 18 dígitos de precisión
- boolean, 1 bit, rango de True o False

La conversión entre los diferentes tipos de datos esta en función del lenguaje y existe una pérdida de datos si se hace de un tipo de dato que no "quepa" en el rango del otro; estos y otros errores de programación han generado graves errores en el sistema automatizado de control que han desembocado por ejemplo en, misiones fallidas de cohetes espaciales y su carga en numerosas ocasiones³, la quiebra de la empresa Knight Capital 2012, sobredosis de radioterapia en 1985 a 1987, la desactivación de servidores de Amazon en el 2012, el apagón en el noreste de EE.UU. en el 2003, fallo en el sistema de reserva de vuelos de American Airlines en el 2013, fallo en el sistema de misiles estadounidense Patriot en Dhahran en el 1991, etc.

²La precedencia de los operadores básicos son: = el de mayor precedencia, siguen - y + unario, luego *, / , % y finalmente los operadores + y -.

³Una de las pérdidas, fue el satélite mexicano Centenario en mayo del 2015.

Aritmética de Punto Flotante La aritmética de punto flotante es considerada un tema esotérico para muchas personas. Esto es sorprendente porque el punto flotante es omnipresente en los sistemas informáticos. Casi todos los lenguajes de programación tienen un tipo de datos de punto flotante, i.e. los números no enteros como 1.2 ó $1e + 45$.

Los problemas de precisión del punto flotante son una preocupación crítica en la informática y la computación numérica, donde la representación y manipulación de números reales puede conducir a resultados inesperados y erróneos. Estos obstáculos surgen debido a las limitaciones inherentes de la aritmética de punto flotante, que aproxima números reales con una precisión finita.

Esto puede causar problemas importantes en diversas aplicaciones, desde simulaciones científicas hasta cálculos financieros, donde incluso las imprecisiones menores pueden propagarse y amplificarse, dando lugar a errores sustanciales. Comprender estos obstáculos es esencial para que los desarrolladores, ingenieros y científicos implementen algoritmos numéricos sólidos y confiables, garantizando que las matemáticas realizadas por las computadoras sigan siendo confiables y precisas.

Comprensión de los Errores de Redondeo en Aritmética de Punto Flotante El meollo de la cuestión es la representación de números de punto flotante. Las computadoras utilizan una cantidad finita de bits para almacenar estos números, y generalmente cumplen con el estándar IEEE 754⁴. Este estándar define cómo se almacenan los números en formato binario, con un número fijo de bits asignados para el signo, el exponente y la mantisa. Si bien esto permite representar una amplia gama de valores, también impone limitaciones a la precisión. No todos los números decimales se pueden representar exactamente en forma binaria, lo que genera pequeñas discrepancias conocidas como errores de redondeo.

Una de las primeras cosas que uno encuentra con sorpresa cuando hace cálculos en una computadora es que por ejemplo, si usamos números muy grandes y seguimos incrementando su valor eventualmente el resultado será negativo ... ¿qué pasó? Esto se llama desbordamiento aritmético al intentar crear un valor numérico que está fuera del rango que puede representarse

⁴El estándar IEEE 754-2008 define varios tamaños de números de punto flotante: media precisión (binary16), precisión simple (binary32), precisión doble (binary64), precisión cuádruple (binary128), etc., cada uno con su propia especificación.

con un número dado de dígitos, ya sea mayor que el máximo o menor que el mínimo valor representable. Algo similar pasa al restar al menor número representable en la máquina, el resultado será positivo y se denomina subdesbordamiento.

Por otro lado, tenemos errores de redondeo, estos ocurren porque el sistema binario no puede representar con precisión ciertas fracciones. Por ejemplo, el número decimal 0.1 no se puede representar exactamente en binario, lo que da como resultado una aproximación, por ejemplo al sumar $0.1 + 0.2$ en una computadora usando por ejemplo el lenguaje Python obtenemos:

```
print(0.1+0.2)
0.30000000000000004
```

que no es exactamente lo que esperábamos⁵. Cuando se utilizan tales aproximaciones en los cálculos, los errores pueden acumularse. Este fenómeno es particularmente problemático en procesos iterativos, donde el mismo cálculo se realiza repetidamente y los errores se acumulan con el tiempo.

Estos errores de precisión se vuelven particularmente problemáticos cuando se realizan controles de igualdad. En un mundo ideal, comparar la igualdad de dos números de punto flotante sería sencillo. Sin embargo, debido a los pequeños errores introducidos durante las operaciones aritméticas, dos números que deberían ser iguales pueden no ser exactamente iguales en su representación binaria.

Por ejemplo, el resultado de sumar 0.1 y 0.2 puede no ser exactamente igual a 0.3 debido a la forma en que estos números se representan en binario, por ejemplo:

```
if 0.1 + 0.2 == 0.3:
    print("si")
else:
    print("no")
```

la respuesta será "no". Esta discrepancia puede provocar que fallen las comprobaciones de igualdad, lo que provocará un comportamiento inesperado en los programas.

⁵Lo mismo obtenemos si usamos la multiplicación, por ejemplo:

```
print(0.1 * 3)
0.30000000000000004
```

Para mitigar estos problemas, los desarrolladores suelen utilizar una técnica conocida como "comparación épsilon". En lugar de verificar la igualdad exacta, verifican si la diferencia absoluta entre dos números de punto flotante es menor que un valor pequeño predefinido, conocido como épsilon. Este enfoque reconoce la imprecisión inherente de la aritmética de punto flotante y proporciona una forma más sólida de comparar números. Sin embargo, elegir un valor épsilon apropiado puede resultar complicado, ya que depende del contexto específico y del rango de valores involucrados.

El problema se ve exacerbado por la precisión finita de los números de punto flotante. Al realizar operaciones aritméticas, el resultado a menudo debe redondearse para que se ajuste a los bits disponibles. Este redondeo puede introducir más imprecisiones. Por ejemplo, sumar dos números de punto flotante de magnitudes muy diferentes puede provocar una pérdida de precisión, ya que el número más pequeño puede ignorarse de hecho. Esto se conoce como cancelación catastrófica y puede afectar gravemente a la precisión de los cálculos.

Por ejemplo, ¿qué podría tener de interesante la humilde fórmula cuadrática?. Después de todo, es una fórmula. Simplemente le pones números. Bueno, hay un detalle interesante. Cuando el coeficiente lineal b es grande en relación con los otros coeficientes, la fórmula cuadrática puede dar resultados incorrectos cuando se implementa en aritmética de punto flotante. Eso es cierto, pero veamos qué sucede cuando tenemos $a = c = 1$ y $b = 10e8$ en

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \text{ y } x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \quad (1.1)$$

regresa

$$x_1 = -7.450580596923828e - 09 \text{ y } x_2 = -100000000.0$$

La primera raíz está equivocada en aproximadamente un 25%, aunque la segunda es correcta.

¿Qué pasó? La ecuación cuadrática violó la regla cardinal del análisis numérico: evitar restar números casi iguales. Cuanto más similares sean dos números, más precisión puedes perder al restarlos. En este caso $(b^2 - 4ac)$ es casi igual a b . Si evaluamos, obtenemos $1.49e - 8$ cuando sería la respuesta correcta $2.0e - 8$.

Si usamos la otra formula⁶

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}} \text{ y } x_2 = \frac{-2c}{b - \sqrt{b^2 - 4ac}} \quad (1.2)$$

obtenemos

$$x_1 = -1e - 08 \text{ y } x_2 = -134217728.0$$

Entonces, ¿cuál formula cuadrática es mejor? Da la respuesta correcta para la primera raíz, exacta dentro de la precisión de la máquina. Pero ahora la segunda raíz está equivocada en un 34%. ¿Por qué la segunda raíz es incorrecta? La misma razón que antes: ¡prestamos dos números casi iguales!

La versión familiar de la fórmula cuadrática calcula correctamente la raíz más grande y la otra versión calcula correctamente la raíz más pequeña. Ninguna versión es mejor en general. No estaríamos ni mejor ni peor usando siempre la nueva fórmula cuadrática que la anterior. Cada uno es mejor cuando evita restar números casi iguales. La solución es utilizar ambas fórmulas cuadráticas, utilizando la apropiada para la raíz que estás intentando calcular.

Otro error común es la suposición de que la aritmética de punto flotante es asociativa y distributiva, como lo es en matemáticas puras. En realidad, el orden de las operaciones puede afectar significativamente el resultado debido a errores de redondeo. Por ejemplo, la expresión $(a + b) + c$ puede producir un resultado diferente que $a + (b + c)$ cuando se trata de números de punto flotante. Esta no asociatividad puede provocar errores sutiles, especialmente en algoritmos complejos que se basan en cálculos precisos.

Además, la aritmética de punto flotante puede introducir problemas en los algoritmos que requieren comparaciones exactas, como los que se utilizan en la clasificación o la búsqueda. Cuando los números de punto flotante se utilizan como claves en estructuras de datos como tablas hash o árboles de búsqueda binarios, los errores de precisión pueden provocar un comportamiento incorrecto, como no encontrar un elemento existente o insertar incorrectamente un duplicado. Para solucionar este problema, es posible que los desarrolladores necesiten implementar funciones de comparación personalizadas que tengan en cuenta la imprecisión del punto flotante.

Pasando a implicaciones prácticas, estos errores de redondeo pueden tener consecuencias de gran alcance. En las simulaciones científicas, pequeñas imprecisiones pueden dar lugar a predicciones incorrectas, lo que podría socavar

⁶Para obtener la segunda fórmula, multiplique el numerador y denominador de x_1 por $-b - \sqrt{b^2 - 4ac}$ (y de manera similar para x_2).

la validez de la investigación. En aplicaciones financieras, los errores de redondeo pueden dar lugar a importantes discrepancias monetarias, afectando todo, desde el análisis del mercado de valores hasta las transacciones bancarias. Incluso en aplicaciones cotidianas como la representación de gráficos, los errores de redondeo pueden causar artefactos visuales que restan valor a la experiencia del usuario.

Además, comprender las limitaciones de la aritmética de punto flotante puede ayudar a diseñar sistemas más robustos. Al anticipar dónde es probable que se produzcan errores de redondeo, los desarrolladores pueden implementar controles y equilibrios para detectar y corregir imprecisiones. Por ejemplo, la aritmética de intervalos puede proporcionar límites a los posibles valores de un cálculo, ofreciendo una manera de cuantificar la incertidumbre introducida por los errores de redondeo.

Estrategias para Mitigar los Problemas de Precisión del Punto Flotante en el Desarrollo de Software Los problemas de precisión del punto flotante son un desafío común en el desarrollo de Software y a menudo conducen a resultados inesperados y errores sutiles. Estos problemas surgen debido a las limitaciones inherentes a la representación de números reales en formato binario, lo que puede provocar errores de redondeo y pérdida de precisión. Para mitigar estos obstáculos, los desarrolladores deben emplear una variedad de estrategias que garanticen la precisión numérica y la confiabilidad en sus aplicaciones.

Una estrategia eficaz es utilizar tipos de datos de mayor precisión cuando sea necesario. Si bien los tipos de punto flotante estándar como "flotante" y "doble" son suficientes para muchas aplicaciones, es posible que no proporcionen la precisión requerida para cálculos más sensibles. En tales casos, el uso de tipos de precisión extendidos, como "long double" en C++ o bibliotecas de precisión arbitraria como GMP (Biblioteca aritmética de precisión múltiple GNU), puede reducir significativamente el riesgo de pérdida de precisión.

Sin embargo, es importante equilibrar la necesidad de precisión con consideraciones de rendimiento, ya que los tipos de mayor precisión pueden resultar costosos desde el punto de vista computacional. Otro enfoque consiste en implementar algoritmos numéricos que sean inherentemente más estables. Algunos algoritmos son más susceptibles a errores de redondeo que otros y elegir el algoritmo correcto puede marcar una diferencia significativa.

Por ejemplo, al resolver sistemas de ecuaciones lineales, utilizar méto-

dos como la descomposición LU o la descomposición QR puede ser más estable que la eliminación gaussiana. Además, se pueden emplear técnicas de refinamiento iterativo para mejorar la precisión de la solución corrigiendo iterativamente los errores introducidos por la aritmética de punto flotante.

Los desarrolladores deben tener en cuenta el orden de las operaciones en sus cálculos. Las propiedades asociativas y distributivas de la aritmética no siempre se cumplen en la aritmética de punto flotante debido a errores de redondeo. Por lo tanto, reorganizar el orden de las operaciones a veces puede conducir a resultados más precisos. Por ejemplo, al sumar una gran cantidad de números de punto flotante, sumarlos en orden de magnitud ascendente puede minimizar la acumulación de errores de redondeo. Esta técnica, conocida como suma de Kahan, ayuda a preservar la precisión al compensar los pequeños errores que ocurren durante el proceso de suma.

Además de estas estrategias, es fundamental realizar pruebas y validaciones exhaustivas del Software numérico. Las pruebas unitarias deben diseñarse para cubrir una amplia gama de valores de entrada, incluidos casos extremos que probablemente expongan problemas de precisión. Comparar los resultados de los cálculos de punto flotante con soluciones analíticas conocidas o utilizar aritmética de mayor precisión como referencia puede ayudar a identificar discrepancias.

Además, se puede realizar un análisis de sensibilidad para evaluar cómo pequeños cambios en los valores de entrada afectan la salida, proporcionando información sobre la estabilidad y confiabilidad de los algoritmos numéricos.

Por último, la documentación y la comunicación desempeñan un papel fundamental a la hora de mitigar los problemas de precisión del punto flotante. Los desarrolladores deben documentar las limitaciones y suposiciones de sus algoritmos numéricos, así como cualquier fuente potencial de error. Esta información es invaluable para otros desarrolladores que necesiten mantener o ampliar el Software. Además, una comunicación clara con las partes interesadas sobre la precisión esperada del Software puede ayudar a gestionar las expectativas y evitar mal entendidos.

1.2 Cómputo Científico

Los sistemas lineales densos más grandes que se están resolviendo en la actualidad en un sólo equipo de cómputo son de orden $n = 10^{12}$ y los futuros sistemas informáticos a exaescala podrán abordar problemas aún mayores. El análisis del error de redondeo muestra que la solución calculada satisface un

límite de error hacia atrás por componentes que -bajo supuestos favorables- es de orden $n\mu$, donde μ es el redondeo unitario de la aritmética de punto flotante: $\mu = 10^{-16}$ para doble precisión y $\mu = 10^{-8}$ para precisión simple. Este límite de error hacia atrás no puede garantizar ninguna estabilidad para la solución de precisión simple de los problemas más grandes de hoy y sugiere una pérdida de la mitad de los dígitos en el error hacia atrás para precisión doble.

La aritmética de punto flotante de media precisión ahora está disponible en Hardware, tanto en el formato IEEE binary16 como en el formato bfloat16, y se utiliza cada vez más en el aprendizaje automático y en el cómputo científico en general. Para el cálculo del producto interno de dos n -vectores el límite de error hacia atrás es nuevamente de orden $n\mu$, y este límite excede 1 por $n > 864$ para ambos formatos de precisión media, lo que sugiere una pérdida potencialmente completa de estabilidad numérica. Sin embargo, productos internos con $n > 864$ se utilizan con éxito en cálculos de precisión media en la práctica.

Los límites de error a los que me he referido son límites superiores y, por lo tanto, limitan el peor de los casos sobre todos los posibles errores de redondeo. Su objetivo principal es revelar posibles inestabilidades en lugar de proporcionar estimaciones de error realistas. Sin embargo, necesitamos conocer los límites de lo que podemos calcular y, para las aplicaciones de misión crítica, debemos poder garantizar un cálculo exitoso.

¿Podemos entender el comportamiento de los algoritmos de álgebra lineal a escala extrema y en aritmética de punto flotante de baja precisión?

En gran medida, la respuesta es sí si aprovechamos tres características diferentes para obtener límites de error más pequeños.

Algoritmos Bloqueados muchos algoritmos se implementan en forma bloqueada. Por ejemplo, un producto interior $x^T y$ de dos n -vectores x y y se puede calcular como

$$\begin{aligned} s_i &= x((i-1)b+1 : ib)^T y((i-1)b+1 : ib), i = 1 : k, \\ s &= s_1 + s_2 + \dots + s_k \end{aligned}$$

dónde $n = kb$ y $b \ll n$ es el tamaño del bloque. El producto interior se ha partido en k productos internos más pequeños de tamaño b , que se calculan de forma independiente y luego se suman. Muchos algoritmos de álgebra lineal se bloquean de forma análoga, donde el bloqueo se realiza en submatrices con

b filas o b columnas (o ambas). Un análisis cuidadoso del análisis de errores muestra que un algoritmo bloqueado tiene un límite de error alrededor de un factor de b más pequeño que el del algoritmo desbloqueado correspondiente. Los tamaños de bloque prácticos para algoritmos matriciales suelen ser 128 o 256, por lo que el bloqueo trae una reducción sustancial de los límites de error.

De hecho, uno puede hacerlo incluso mejor que un límite de error de orden $(n/b)\mu$. Calculando la suma $s = s_1 + s_2 + \dots + s_k$ con un método de suma más preciso, la constante de error se reduce aún más a $b\mu + O(\mu^2)$ (este es el método FABsum de Blanchard et al. 2020).

Características Arquitectónicas los procesadores Intel x86 admiten un formato de precisión extendido de 80 bits con un significado de 64 bits, que es compatible con el especificado en el estándar IEEE. Cuando un compilador usa este formato con registros de 80 bits para acumular sumas y productos internos, está trabajando efectivamente con un redondeo unitario de 2^{-64} en vez de 2^{-53} para precisión doble, dando límites de error más pequeños en un factor de hasta $2^{11} = 2048$.

Algunos procesadores Intel y AMD tienen una operación fusionada de multiplicación y suma (FMA), que calcula una multiplicación y una suma combinadas $x + yz$ con un error de redondeo en lugar de dos. Esto da como resultado una reducción en los límites de error por un factor 2.

Las operaciones FMA de bloques de precisión mixta $D = C + AB$, con matrices A, B, C y D de tamaño fijo, están disponibles en las unidades de procesamiento tensorial de Google, las GPU NVIDIA y en la arquitectura ARMv8-A. Para entradas de precisión media, estos dispositivos pueden producir resultados de calidad de precisión simple, lo que puede proporcionar un aumento significativo en la precisión cuando los bloques FMA se encadenan para formar un producto matricial de dimensión arbitraria.

Límites probabilísticos los límites del error de redondeo en el peor de los casos sufren el problema de que no se pueden alcanzar para la mayoría de los conjuntos de datos específicos y es poco probable que se alcancen casi por completo. Stewart (1990) señaló que:

Para ser realistas, debemos eliminar lo improbable. Lo que queda es necesariamente un enunciado probabilístico.

Theo Mary y Nick Higham han desarrollado recientemente un análisis de error de redondeo probabilístico, que hace suposiciones probabilísticas sobre los errores de redondeo y deriva límites que se mantienen con cierta probabilidad. La característica clave de los límites es que son proporcionales a $\sqrt{n\mu}$ cuando un límite correspondiente en el peor de los casos es proporcional a $n\mu$. En la forma más general del análisis (Connolly, Higham y Mary, 2021), se supone que los errores de redondeo son independientes de la media y de media cero, donde la independencia media es un supuesto más débil que la independencia.

Poniendo las Piezas Juntas las diferentes características que hemos descrito se pueden combinar para obtener límites de error significativamente más pequeños. Si usamos un algoritmo bloqueado con tamaño de bloque $b \ll n$ luego, en un producto interno, el límite de error estándar de orden $n\mu$ se reduce a un límite probabilístico de orden $\left(\sqrt{n/b}\right)\mu$, lo que es una reducción significativa. Los bloques FMA y los registros de precisión extendidos proporcionan reducciones adicionales.

Por ejemplo, para un sistema lineal de orden 10^7 resuelto en precisión simple con un tamaño de bloque de 256, el límite de error probabilístico es de orden 10^{-5} versus 1 para el límite estándar del peor de los casos. Si se utiliza FABsum, el límite se reduce aún más.

Los autores concluyen que podemos resolver con éxito problemas de álgebra lineal de mayor tamaño y menor precisión de lo que sugiere el análisis de error de redondeo estándar. Sin embargo, los límites a priori siempre serán pesimistas. Se deben calcular los residuos a posteriori o los errores hacia atrás (según el problema) para evaluar la calidad de una solución numérica.

En el caso de los modelos matemáticos de los sistemas continuos, estos son sistemas de ecuaciones diferenciales, las cuales son parciales -con valores iniciales y condiciones de frontera- para casi todos los sistemas de mayor interés en la Ciencia y la Ingeniería. Salvo por los problemas más sencillos, no es posible obtener por métodos analíticos las soluciones de tales ecuaciones, que son las que permiten predecir el comportamiento de los sistemas continuos y realizar las simulaciones requeridas.

La capacidad para formular los modelos matemáticos de sistemas continuos complicados y de gran diversidad, es sin duda una contribución fundamental para el avance de la Ciencia y sus aplicaciones, tal contribución

quedaría incompleta y, debido a ello, sería poco fecunda, si no se hubiera desarrollado simultáneamente su complemento esencial: los métodos numéricos y la computación electrónica.

Sin embargo, la solución numérica y las simulaciones computacionales de problemas concomitantes en Ciencias e Ingenierías han llevado al límite nuestra actual capacidad de predicción, por la gran cantidad de grados de libertad que necesitan nuestros modelos para tratar de representar a la realidad.

Con el desarrollo de nuevas herramientas numéricas y computacionales, la diversidad y complejidad de problemas que pueden ser tratados de forma satisfactoria y eficiente es impresionante. Pero hay que destacar, que todavía hay una gama de problemas que hasta la fecha no es posible resolver satisfactoriamente o con la precisión deseada -como la predicción climática a largo plazo o simulación de recuperación mejorada en yacimientos petroleros, entre otros-.

La solución numérica de ecuaciones diferenciales parciales por los esquemas tradicionales -tipo Diferencias Finitas (que trataremos brevemente en el apéndice), Volumen Finito y Elemento Finito- reducen el problema a la generación y solución de un -cada vez más grande- sistema algebraico de ecuaciones. La factorización directa de sistemas de gran escala $O(10^6)$ con toda su eficacia, no es, en general una opción viable, y el uso de métodos iterativos básicos resultan en una convergencia bastante lenta con respecto a otras formas de solución.

El desarrollo de métodos numéricos para sistemas algebraicos grandes no densos, es central en el desarrollo de códigos eficientes para la solución de problemas en Ciencia e Ingeniería, actualmente cuando se necesita implementar una solución computacional, se dispone de bibliotecas optimizadas⁷ para la solución de sistemas lineales que pueden correr en ambientes secuenciales y/o paralelos. Estas bibliotecas implementan métodos algebraicos robustos para muchos problemas prácticos, pero sus discretizaciones no pueden ser construidas por sólo técnicas algebraicas simples, tales como aproximaciones a la inversa o factorización incompleta.

En la actualidad, los sistemas computacionales paralelos son ubicuos. En ellos es posible encontrar más de una unidad de procesamiento, conocidas como Núcleo (Core). El número de Cores es creciente conforme avanza la

⁷Como pueden ser las bibliotecas ATLAS —<http://math-atlas.sourceforge.net>— y HYPRE —<https://computation.llnl.gov/projects/hypre-scalable-linear-solvers-multigrid-methods>— entre muchas otras.

tecnología, esto tiene una gran importancia en el desarrollo eficiente de algoritmos que resuelvan sistemas algebraicos en implementaciones paralelas. Actualmente la gran mayoría de los algoritmos desarrollados son algoritmos secuenciales y su implantación en equipos paralelos no es óptima, pero es una práctica común usar diversas técnicas de pseudoparalelización -a veces mediante la distribución de una gran matriz en la memoria de los múltiples Cores y otras mediante el uso de directivas de compilación-, pero la eficiencia resultante es pobre y no escalable a equipos masivamente paralelos por la gran cantidad de comunicación involucrada en la solución.

1.3 FLOPS

Una medida relativamente objetiva para analizar el rendimiento de un dispositivo suele ser medir sus operaciones de punto flotante por segundo o más conocidas como FLOPS. Hay que tener en cuenta que la medición de FLOPS es muy compleja porque las diferentes operaciones en punto flotante llevan diferentes cantidades de tiempo para ejecutarse. Y no todo el mundo utiliza las mismas operaciones para establecer los cálculos.

Por ejemplo, una división simple como $1/5$, toma significativamente menos tiempo que el cálculo del logaritmo de 5. Por eso, se estableció el algoritmo de Linpack como un estándar representativo con el que poder medir todos los sistemas bajo el mismo baremo de FLOPS.

Es importante señalar que el algoritmo de Linpack utiliza el formato en punto flotante de doble precisión (64-bit). Sin embargo, como veremos la mayoría de los valores que dan los fabricantes son con precisión simple (32-bit). Además, los valores que dan los fabricantes suelen ser teóricos y en la práctica suelen ser inferiores debido a otros factores limitantes como la frecuencia de reloj o la velocidad de las memorias ROM y RAM.

Por tanto, aunque todos hemos acabado midiendo el rendimiento en FLOPS⁸, no es una medida absoluta de la potencia del CPU ni de una GPU. Por ejemplo para algunos dispositivos tenemos:

Móviles El SoC Snapdragon 821 que monta una GPU Adreno 530 tiene una potencia de 519.2 gigaFLOPS (0.52 TFLOPS), y los Chips Apple A9X

⁸El Cray-1 fue puesto marcha en 1975 y utilizaba una CPU a 80 MHz y llevaba integrada una unidad SIMD de 64 bits de precisión de punto flotante, lo cual fue un salto de gigante que permitió un salto de los 3 MFLOPS de potencia del CDC 6600 a los 160 MFLOPS en el Cray-1.

del iPad Pro alcanzan los 345.6 gigaFLOPS (0.35 TFLOPS), todos ellos medidos con precisión simple de 32-bits.

CPU

- Intel Xeon W-3245: 1.4 TFLOPS
- Intel Core i9-9900X: 1.2 TFLOPS
- AMD Ryzen 9 3950X: 1.1 TFLOPS

Los procesadores de gama media-alta rondan el medio TFLOPS:

- AMD Ryzen 7 3700X: 546.0 GFLOPS - 0.55 TFLOPS
- Intel Core i9-9900: 499.0 GFLOPS - 0.50 TFLOPS
- AMD Ryzen 5 3600X: 461.0 GFLOPS - 0.46 TFLOPS

Tarjetas Gráficas Ojo: La tabla está ordenada por los valores en precisión simple (32-bit) primer columna:

<i>GPU</i>	<i>FP32 TFLOPS</i>	<i>FP64 TFLOPS</i>
<i>TITANV</i>	13.8	6.9
<i>RadeonRXVega64</i>	12.7	0.8
<i>GeForceGTX1080Ti</i>	11.3	0.4

Como podemos ver, las tarjetas gráficas de Nvidia, normalmente, tienen una potencia muy alta en precisión simple, pero muy mala en precisión doble. La precisión simple es la que se usa en los juegos, pero la precisión doble es la que se utiliza en los cálculos complejos científicos y en el minado de muchas criptomonedas.

Consolas Todas ellas son en valores de precisión simple (32-bit)

- PlayStation 4: 1.3 TFLOPS
- Xbox One: 1.8 TFLOPS
- PlayStation 4 Pro: 4.2 TFLOPS

- Nintendo Switch: entre 0.4 y 0.5 TFLOPS
- PlayStation 5 promete una GPU con 10.28 TFLOPS
- La Xbox Series X promete una GPU de 12 TFLOPS

SuperCómputo Se dio a conocer en mayo del 2022 la publicación de la 59ª edición del ranking de las 500 computadoras de mayor rendimiento del mundo. En la nueva edición, el top ten incluyó tres nuevos clústeres, donde el Frontier pasará a la historia por ser el primero en romper oficialmente la barrera de la exaescala:

- El primer lugar fue para el nuevo Frontier, ubicado en el Laboratorio Nacional de Oak Ridge del Departamento de Energía de los Estados Unidos. El clúster tiene 9.400 CPUs y 37.000 GPUs (CPU AMD EPYC 64C de 2 GHz, acelerador AMD Instinct MI250X) y proporciona un rendimiento de 1102 petaflops, que es casi tres veces más que el líder anterior. Al mismo tiempo, el consumo de energía de Frontier es un 30 % más bajo que el líder anterior, con una inversión de más de 600 millones de dólares
- El clúster japonés Fugaku construido con procesadores ARM (158976 nodos basados en el SoC Fujitsu A64FX, equipado con una CPU Armv8.2-A SVE de 2,2 GHz de 48 núcleos) pasó al segundo lugar. El clúster Fugaku está alojado en el Instituto RIKEN de Investigación Física y Química y proporciona un rendimiento de 442 petaflops. El sistema operativo es Red Hat Enterprise Linux.
- El tercer lugar lo ocupó el nuevo clúster LUMI, alojado en el Centro Europeo de Supercomputación (EuroHPC) en Finlandia y que ofrece un rendimiento de 151 petaflops. El clúster se basa en la misma plataforma HPE Cray EX235a que el nuevo líder de la clasificación, pero incluye 1,1 millones de núcleos de procesador (AMD EPYC 64C 2GHz, acelerador AMD Instinct MI250X, red Slingshot-11).

1.4 Agradecimientos

Este texto es una recopilación de múltiples fuentes, mi aportación -si es que puedo llamarla así- es plasmarlo en este documento, en el que trato de dar coherencia a mi visión de los temas desarrollados.

En la realización de este texto se han revisado -en la mayoría de los casos indicó la referencia, pero pude omitir varias de ellas, por lo cual pido una disculpa- múltiples páginas Web, artículos técnicos, libros, entre otros materiales bibliográficos, los más representativos y de libre acceso los pongo a su disposición en la siguiente liga:

Herramientas

<http://132.248.181.216/Herramientas/>

Además, la documentación y los diferentes ejemplos que se presentan en este trabajo, se encuentran disponibles en dicha liga, para que puedan ser copiados desde el navegador y ser usados. En aras de que el interesado pueda correr dichos ejemplos y afianzar sus conocimientos, además de que puedan ser usados en diferentes ámbitos a los presentados aquí.

2 Aritmética de Punto Flotante

La aritmética de punto flotante es considerada un tema esotérico para muchas personas. Esto es sorprendente porque el punto flotante es omnipresente en los sistemas informáticos. Casi todos los lenguajes de programación tienen un tipo de datos de punto flotante, i.e. los números no enteros como 1.2 ó $1e + 45$.

Los problemas de precisión del punto flotante son una preocupación crítica en la informática y la computación numérica, donde la representación y manipulación de números reales puede conducir a resultados inesperados y erróneos. Estos obstáculos surgen debido a las limitaciones inherentes de la aritmética de punto flotante, que aproxima números reales con una precisión finita.

Esto puede causar problemas importantes en diversas aplicaciones, desde simulaciones científicas hasta cálculos financieros, donde incluso las imprecisiones menores pueden propagarse y amplificarse, dando lugar a errores sustanciales. Comprender estos obstáculos es esencial para que los desarrolladores, ingenieros y científicos implementen algoritmos numéricos sólidos y confiables, garantizando que las matemáticas realizadas por las computadoras sigan siendo confiables y precisas.

Comprensión de los Errores de Redondeo en Aritmética de Punto Flotante El meollo de la cuestión es la representación de números de punto flotante. Las computadoras utilizan una cantidad finita de bits para almacenar estos números, y generalmente cumplen con el estándar IEEE 754⁹. Este estándar define cómo se almacenan los números en formato binario, con un número fijo de bits asignados para el signo, el exponente y la mantisa. Si bien esto permite representar una amplia gama de valores, también impone limitaciones a la precisión. No todos los números decimales se pueden representar exactamente en forma binaria, lo que genera pequeñas discrepancias conocidas como errores de redondeo.

Una de las primeras cosas que uno encuentra con sorpresa cuando hace cálculos en una computadora es que por ejemplo, si usamos números muy grandes y seguimos incrementando su valor eventualmente el resultado será

⁹El estándar IEEE 754-2008 define varios tamaños de números de punto flotante: media precisión (binary16), precisión simple (binary32), precisión doble (binary64), precisión cuádruple (binary128), etc., cada uno con su propia especificación.

negativo ... ¿qué pasó? Esto se llama desbordamiento aritmético al intentar crear un valor numérico que está fuera del rango que puede representarse con un número dado de dígitos, ya sea mayor que el máximo o menor que el mínimo valor representable. Algo similar pasa al restar al menor número representable en la máquina, el resultado será positivo y se denomina sub-desbordamiento.

Por otro lado, tenemos errores de redondeo, estos ocurren porque el sistema binario no puede representar con precisión ciertas fracciones. Por ejemplo, el número decimal 0.1 no se puede representar exactamente en binario, lo que da como resultado una aproximación, por ejemplo al sumar $0.1 + 0.2$ en una computadora usando por ejemplo el lenguaje Python obtenemos:

```
print(0.1+0.2)
0.30000000000000004
```

que no es exactamente lo que esperábamos¹⁰. Cuando se utilizan tales aproximaciones en los cálculos, los errores pueden acumularse. Este fenómeno es particularmente problemático en procesos iterativos, donde el mismo cálculo se realiza repetidamente y los errores se acumulan con el tiempo.

Estos errores de precisión se vuelven particularmente problemáticos cuando se realizan controles de igualdad. En un mundo ideal, comparar la igualdad de dos números de punto flotante sería sencillo. Sin embargo, debido a los pequeños errores introducidos durante las operaciones aritméticas, dos números que deberían ser iguales pueden no ser exactamente iguales en su representación binaria.

Por ejemplo, el resultado de sumar 0.1 y 0.2 puede no ser exactamente igual a 0.3 debido a la forma en que estos números se representan en binario, por ejemplo:

```
if 0.1 + 0.2 == 0.3:
    print("si")
else:
    print("no")
```

¹⁰Lo mismo obtenemos si usamos la multiplicación, por ejemplo:

```
print(0.1 * 3)
0.30000000000000004
```

la respuesta será "no". Esta discrepancia puede provocar que fallen las comprobaciones de igualdad, lo que provocará un comportamiento inesperado en los programas.

Para mitigar estos problemas, los desarrolladores suelen utilizar una técnica conocida como "comparación épsilon". En lugar de verificar la igualdad exacta, verifican si la diferencia absoluta entre dos números de punto flotante es menor que un valor pequeño predefinido, conocido como épsilon. Este enfoque reconoce la imprecisión inherente de la aritmética de punto flotante y proporciona una forma más sólida de comparar números. Sin embargo, elegir un valor épsilon apropiado puede resultar complicado, ya que depende del contexto específico y del rango de valores involucrados.

El problema se ve exacerbado por la precisión finita de los números de punto flotante. Al realizar operaciones aritméticas, el resultado a menudo debe redondearse para que se ajuste a los bits disponibles. Este redondeo puede introducir más imprecisiones. Por ejemplo, sumar dos números de punto flotante de magnitudes muy diferentes puede provocar una pérdida de precisión, ya que el número más pequeño puede ignorarse de hecho. Esto se conoce como cancelación catastrófica y puede afectar gravemente a la precisión de los cálculos.

Por ejemplo, ¿qué podría tener de interesante la humilde fórmula cuadrática?. Después de todo, es una fórmula. Simplemente le pones números. Bueno, hay un detalle interesante. Cuando el coeficiente lineal b es grande en relación con los otros coeficientes, la fórmula cuadrática puede dar resultados incorrectos cuando se implementa en aritmética de punto flotante. Eso es cierto, pero veamos qué sucede cuando tenemos $a = c = 1$ y $b = 10e8$ en

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \text{ y } x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \quad (2.1)$$

regresa

$$x_1 = -7.450580596923828e - 09 \text{ y } x_2 = -100000000.0$$

La primera raíz está equivocada en aproximadamente un 25%, aunque la segunda es correcta.

¿Qué pasó? La ecuación cuadrática violó la regla cardinal del análisis numérico: evitar restar números casi iguales. Cuanto más similares sean dos números, más precisión puedes perder al restarlos. En este caso $(b^2 - 4ac)$ es casi igual a b . Si evaluamos, obtenemos $1.49e - 8$ cuando sería la respuesta correcta $2.0e - 8$.

Si usamos la otra formula¹¹

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}} \text{ y } x_2 = \frac{-2c}{b - \sqrt{b^2 - 4ac}} \quad (2.2)$$

obtenemos

$$x_1 = -1e - 08 \text{ y } x_2 = -134217728.0$$

Entonces, ¿cuál formula cuadrática es mejor? Da la respuesta correcta para la primera raíz, exacta dentro de la precisión de la máquina. Pero ahora la segunda raíz está equivocada en un 34%. ¿Por qué la segunda raíz es incorrecta? La misma razón que antes: ¡prestamos dos números casi iguales!

La versión familiar de la fórmula cuadrática calcula correctamente la raíz más grande y la otra versión calcula correctamente la raíz más pequeña. Ninguna versión es mejor en general. No estaríamos ni mejor ni peor usando siempre la nueva fórmula cuadrática que la anterior. Cada uno es mejor cuando evita restar números casi iguales. La solución es utilizar ambas fórmulas cuadráticas, utilizando la apropiada para la raíz que estás intentando calcular.

Otro error común es la suposición de que la aritmética de punto flotante es asociativa y distributiva, como lo es en matemáticas puras. En realidad, el orden de las operaciones puede afectar significativamente el resultado debido a errores de redondeo. Por ejemplo, la expresión $(a + b) + c$ puede producir un resultado diferente que $a + (b + c)$ cuando se trata de números de punto flotante. Esta no asociatividad puede provocar errores sutiles, especialmente en algoritmos complejos que se basan en cálculos precisos.

Además, la aritmética de punto flotante puede introducir problemas en los algoritmos que requieren comparaciones exactas, como los que se utilizan en la clasificación o la búsqueda. Cuando los números de punto flotante se utilizan como claves en estructuras de datos como tablas hash o árboles de búsqueda binarios, los errores de precisión pueden provocar un comportamiento incorrecto, como no encontrar un elemento existente o insertar incorrectamente un duplicado. Para solucionar este problema, es posible que los desarrolladores necesiten implementar funciones de comparación personalizadas que tengan en cuenta la imprecisión del punto flotante.

Pasando a implicaciones prácticas, estos errores de redondeo pueden tener consecuencias de gran alcance. En las simulaciones científicas, pequeñas im-

¹¹Para obtener la segunda fórmula, multiplique el numerador y denominador de x_1 por $-b - \sqrt{b^2 - 4ac}$ (y de manera similar para x_2).

precisiones pueden dar lugar a predicciones incorrectas, lo que podría socavar la validez de la investigación. En aplicaciones financieras, los errores de redondeo pueden dar lugar a importantes discrepancias monetarias, afectando todo, desde el análisis del mercado de valores hasta las transacciones bancarias. Incluso en aplicaciones cotidianas como la representación de gráficos, los errores de redondeo pueden causar artefactos visuales que restan valor a la experiencia del usuario.

Además, comprender las limitaciones de la aritmética de punto flotante puede ayudar a diseñar sistemas más robustos. Al anticipar dónde es probable que se produzcan errores de redondeo, los desarrolladores pueden implementar controles y equilibrios para detectar y corregir imprecisiones. Por ejemplo, la aritmética de intervalos puede proporcionar límites a los posibles valores de un cálculo, ofreciendo una manera de cuantificar la incertidumbre introducida por los errores de redondeo.

Estrategias para Mitigar los Problemas de Precisión del Punto Flotante en el Desarrollo de Software Los problemas de precisión del punto flotante son un desafío común en el desarrollo de Software y a menudo conducen a resultados inesperados y errores sutiles. Estos problemas surgen debido a las limitaciones inherentes a la representación de números reales en formato binario, lo que puede provocar errores de redondeo y pérdida de precisión. Para mitigar estos obstáculos, los desarrolladores deben emplear una variedad de estrategias que garanticen la precisión numérica y la confiabilidad en sus aplicaciones.

Una estrategia eficaz es utilizar tipos de datos de mayor precisión cuando sea necesario. Si bien los tipos de punto flotante estándar como "flotante" y "doble" son suficientes para muchas aplicaciones, es posible que no proporcionen la precisión requerida para cálculos más sensibles. En tales casos, el uso de tipos de precisión extendidos, como "long double" en C++ o bibliotecas de precisión arbitraria como GMP (Biblioteca aritmética de precisión múltiple GNU), puede reducir significativamente el riesgo de pérdida de precisión.

Sin embargo, es importante equilibrar la necesidad de precisión con consideraciones de rendimiento, ya que los tipos de mayor precisión pueden resultar costosos desde el punto de vista computacional. Otro enfoque consiste en implementar algoritmos numéricos que sean inherentemente más estables. Algunos algoritmos son más susceptibles a errores de redondeo que otros y elegir el algoritmo correcto puede marcar una diferencia significativa.

Por ejemplo, al resolver sistemas de ecuaciones lineales, utilizar métodos como la descomposición LU o la descomposición QR puede ser más estable que la eliminación gaussiana. Además, se pueden emplear técnicas de refinamiento iterativo para mejorar la precisión de la solución corrigiendo iterativamente los errores introducidos por la aritmética de punto flotante.

Los desarrolladores deben tener en cuenta el orden de las operaciones en sus cálculos. Las propiedades asociativas y distributivas de la aritmética no siempre se cumplen en la aritmética de punto flotante debido a errores de redondeo. Por lo tanto, reorganizar el orden de las operaciones a veces puede conducir a resultados más precisos. Por ejemplo, al sumar una gran cantidad de números de punto flotante, sumarlos en orden de magnitud ascendente puede minimizar la acumulación de errores de redondeo. Esta técnica, conocida como suma de Kahan, ayuda a preservar la precisión al compensar los pequeños errores que ocurren durante el proceso de suma.

Además de estas estrategias, es fundamental realizar pruebas y validaciones exhaustivas del Software numérico. Las pruebas unitarias deben diseñarse para cubrir una amplia gama de valores de entrada, incluidos casos extremos que probablemente expongan problemas de precisión. Comparar los resultados de los cálculos de punto flotante con soluciones analíticas conocidas o utilizar aritmética de mayor precisión como referencia puede ayudar a identificar discrepancias.

Además, se puede realizar un análisis de sensibilidad para evaluar cómo pequeños cambios en los valores de entrada afectan la salida, proporcionando información sobre la estabilidad y confiabilidad de los algoritmos numéricos.

Por último, la documentación y la comunicación desempeñan un papel fundamental a la hora de mitigar los problemas de precisión del punto flotante. Los desarrolladores deben documentar las limitaciones y suposiciones de sus algoritmos numéricos, así como cualquier fuente potencial de error. Esta información es invaluable para otros desarrolladores que necesiten mantener o ampliar el Software. Además, una comunicación clara con las partes interesadas sobre la precisión esperada del Software puede ayudar a gestionar las expectativas y evitar mal entendidos.

2.1 Aritmética de Punto Flotante IEEE

La aritmética de punto flotante es considerada un tema esotérico para muchas personas. Esto es sorprendente porque el punto flotante es omnipresente en los sistemas informáticos. Casi todos los lenguajes de programación tienen

un tipo de datos de punto flotante, i.e. los números no enteros como 1.2 ó $1e + 45$.

Un número de punto flotante de 64 bits (*double* en lenguaje C) tiene aproximadamente 16 dígitos decimales de precisión con un rango del orden de 1.7×10^{-308} a 1.7×10^{308} de acuerdo con el estándar 754 de IEEE -El estándar IEEE 754-2008 define varios tamaños de números de punto flotante: media precisión (binary16), precisión simple (binary32), precisión doble (binary64), precisión cuádruple (binary128), etc., cada uno con su propia especificación-, la implementación típica de punto flotante¹².

Una de las primeras cosas que uno encuentra con sorpresa cuando hace cálculos en una computadora es que por ejemplo, si usamos números muy grandes y seguimos incrementando su valor eventualmente el resultado será negativo ... ¿qué pasó? Esto se llama desbordamiento aritmético al intentar crear un valor numérico que está fuera del rango que puede representarse con un número dado de dígitos, ya sea mayor que el máximo o menor que el mínimo valor representable. Algo similar pasa al restar al menor número representable en la máquina, el resultado será positivo y se denomina subdesbordamiento.

Las computadoras, desde las PC hasta las supercomputadoras, tienen aceleradores de punto flotante; la mayoría de los compiladores deberán compilar algoritmos de punto flotante de vez en cuando y prácticamente todos los sistemas operativos deben responder a excepciones de punto flotante como el desbordamiento.

Desde ya hace mucho tiempo, los procesadores Intel x86 y todos los procesadores de las siguientes generaciones de todas las marcas admiten un formato de precisión extendido de 80 bits con un significado de 64 bits, que es compatible con el especificado en el estándar IEEE. Cuando un compilador usa este formato con registros de 80 bits para acumular sumas y productos internos, está trabajando efectivamente con un redondeo unitario de 2^{-64} en vez de 2^{-53} para precisión doble, dando límites de error más pequeños en un factor de hasta $2^{11} = 2048$.

¿Dieciséis lugares decimales es mucho? Casi ninguna cantidad medida se conoce con tanta precisión. Por ejemplo: en la constante en la ley de gravedad de Newton sólo se conoce con seis cifras significativas. En la carga

¹²Además, existe el número de 80 bits (*long double* en lenguaje C) que tiene 18 dígitos decimales de precisión con un rango del orden de 3.4×10^{-4096} a 1.1×10^{4096} . Y no podemos olvidar, el número de 32 bits (*float* en lenguaje C) que tiene 14 dígitos decimales de precisión con un rango del orden de 3.4×10^{-38} a 3.4×10^{38} .

de un electrón se conoce con 11 cifras significativas, mucha más precisión que la constante gravitacional de Newton, pero aún menos que un número de punto flotante¹³.

Entonces, ¿cuándo no son suficientes 16 dígitos de precisión? Un área problemática es la resta. Las otras operaciones elementales (suma, multiplicación, división) son muy precisas. Siempre que no se presenten desbordamientos y subdesbordamientos, estas operaciones suelen producir resultados que son correctos hasta el último bit. Pero la resta puede ser desde exacta hasta completamente inexacta. Si dos números concuerdan con n cifras, puede perder hasta n cifras de precisión en su resta. Este problema puede aparecer inesperadamente en medio de otros cálculos.

¿Qué pasa con el desbordamiento o con el subdesbordamiento? ¿Cuándo se necesitan números mayores que 10^{308} ? No tan a menudo, pero en los cálculos de probabilidad, por ejemplo, se usan todo el tiempo a menos que se haga un uso inteligente.

Es común en probabilidad calcular un número de tamaño mediano que es el producto de un número astronómicamente grande y un número infinitesimalmente pequeño. El resultado final encaja perfectamente en una computadora, pero es posible que los números intermedios no se deban a un desbordamiento o un subdesbordamiento. Por ejemplo, el número máximo de punto flotante en la mayoría de las computadoras está entre 170 factorial y 171 factorial. Estos grandes factoriales aparecen frecuentemente en aplicaciones, a menudo en proporciones con otros grandes factoriales.

¹³¿Cuántos dígitos de π necesitamos?

- 3.1415 para diseñar los mejores motores.
- 3.1415926535 para obtener la circunferencia de la Tierra dentro de una fracción de pulgada.
- 3.141592653589793 para los cálculos de navegación interplanetaria de la NASA-JPL.
- 3.1415926535897932384626433832795028842 para medir el radio del universo con una precisión igual al tamaño de un átomo de hidrógeno.

En el 2022 se calcularon los primeros 100 billones de decimales del número π , para lograr este hito necesitó 157 días, 23 horas, 31 minutos y 7,651 segundos de cálculos, 515 Terabytes de almacenamiento y desplegar un abanico de tecnologías de computación en Compute Engine, un servicio de computación de Google Cloud.

Anatomía de un Número de Punto Flotante Un número de punto flotante de 64 bits codifica un número de la forma $\pm p \times 2^e$. El primer bit codifica el signo, 0 para números positivos y 1 para números negativos. Los siguientes 11 bits codifican el exponente e , y los últimos 52 bits codifican la precisión p .

El exponente se almacena con un sesgo de 1023. Es decir, los exponentes positivos y negativos se almacenan todos en un solo número positivo almacenando $e + 1023$ en lugar de almacenarlos directamente. Once bits pueden representar números enteros desde 0 hasta 2047. Restando el sesgo, esto corresponde a valores de e de -1023 a $+1024$. Definimos $e_{min} = -1022$ y $e_{max} = +1023$. Los valores $e_{min} - 1$ y $e_{max} + 1$ están reservados para usos especiales.

Los números de punto flotante se almacenan normalmente en forma normalizada. En base 10, un número está en notación científica normalizada si el significando es ≥ 1 y < 10 . Por ejemplo, 3.14×10^2 está en forma normalizada, pero 0.314×10^3 y 31.4×10^2 no lo están.

En general, un número en base β está en forma normalizada si tiene la forma $p \times \beta^e$ donde $1 \leq p < \beta$. Esto dice que para expresar un número binario, es decir, $\beta = 2$, el primer bit del significado de un número normalizado es siempre 1. Dado que este bit nunca cambia, no es necesario almacenarlo. Por lo tanto, podemos expresar 53 bits de precisión en 52 bits de almacenamiento. En lugar de almacenar el significado directamente, almacenamos f , la parte fraccionaria, donde el significado es de la forma $1.f$.

El esquema anterior no explica cómo almacenar 0. Es imposible especificar valores de f y e de modo que $1.f \times 2^e = 0$. El formato de punto flotante hace una excepción a las reglas establecidas anteriormente. Cuando $e = e_{min} - 1$ y $f = 0$, los bits se interpretan como 0. Cuando $e = e_{min} - 1$ y $f \neq 0$, el resultado es un número desnormalizado. Los bits se interpretan como $0.f \times 2^{e_{min}}$. En resumen, el exponente especial reservado debajo de e_{min} se usa para representar 0 y números de punto flotante desnormalizados.

El exponente especial reservado arriba de e_{max} se usa para representar ∞ y NaN . Si $e = e_{max} + 1$ y $f = 0$, los bits se interpretan como ∞ . Pero si $e = e_{max} + 1$ y $f \neq 0$, los bits se interpretan como un NaN o "no es un número" (Not a Number).

Dado que el exponente más grande es 1023 y el significativo más grande es $1.f$ donde f tiene 52 unidades, el número de punto flotante (en C y C++, esta

constante se define como *DBL_MAX* definido en *<float.h>*, en Python¹⁴ en *sys.float_info*) más grande es $2^{1023}(2 - 2^{-52}) = 2^{1024} - 2^{971} \approx 2^{1024} \approx 1.8 \times 10^{308}$. Los números mayores que 2^{1024} i.e. $(2 - 2^{-52})$ producen un desbordamiento conocido como Overflow.

Dado que el exponente más pequeño es -1022 , el número normalizado positivo más pequeño es $1.0 \times 2^{-1022} \approx 2.2 \times 10^{-308}$ (en C y C++, esta constante se define como *DBL_MIN* definido en *<float.h>*, en Python en *sys.float_info*). Sin embargo, no es el número positivo más pequeño representable como un número de punto flotante, solo el número de punto flotante normalizado más pequeño. Los números más pequeños se pueden expresar en forma desnormalizada, aunque con una pérdida de significado. El número positivo desnormalizado más pequeño ocurre con *f* que tiene 51 números 0 seguidos de un solo 1. Esto corresponde a $2^{-52} * 2^{-1022} = 2^{-1074} \approx 4.9 \times 10^{-324}$.

Los números que aparecen en los cálculos y tienen magnitud menor que 2^{-1023} i.e. $(1 + 2^{-52})$ producen un desbordamiento de la capacidad mínima o subdesbordamiento también conocido como Underflow y, por lo general se igualan a cero.

C y C++ da el nombre de *DBL_EPSILON* al número positivo más pequeño ϵ tal que $1 + \epsilon \approx 1$, también llamado la precisión de la máquina. Dado que el significativo tiene 52 bits, está claro que *DBL_EPSILON* = $2^{-52} \approx 2.2 \times 10^{-16}$. Por eso decimos que un número de punto flotante tiene entre 15 y 16 cifras significativas (decimales).

Formatos de Punto Flotante Se han propuesto varias representaciones diferentes de números reales, pero por mucho la más utilizada es la representación de punto flotante. Las representaciones de punto flotante tienen una base (que siempre se asume que es par) y una precisión *p*. Si $\beta = 10$ y $p = 3$, entonces el número 0.1 se representa como 1.00×10^{-1} . Si $\beta = 2$ y

14

```
import sys
print(sys.float_info)

sys.float_info(max=1.7976931348623157e+308,      max_exp=1024,
max_10_exp=308,  min=2.2250738585072014e-308,    min_exp=-1021,
min_10_exp=-307, dig=15, mant_dig=53, epsilon=2.220446049250313e-
16, radix=2, rounds=1)
```

$p = 24$, entonces el número decimal 0.1 no se puede representar exactamente, pero es aproximadamente $1.10011001100110011001101 \times 2^{-4}$.

En general, un número de punto flotante se representará como $\pm d.dd\dots d \times \beta^e$, donde $d.dd\dots d$ se llama mantisa y tiene p dígitos. De forma más precisa $\pm d_0.d_1d_2\dots d_{p-1} \times \beta^e$ representa el número

$$\pm (d_0 + d_1\beta^{-1} + \dots + d_{p-1}\beta^{-(p-1)})\beta^e, (0 \leq d_i < \beta). \quad (2.3)$$

El término número de punto flotante se utilizará para referirse a un número real que se puede representar exactamente en el formato en discusión. Otros dos parámetros asociados con las representaciones de punto flotante son los exponentes más grandes y más pequeños permitidos, e_{max} y e_{min} . Dado que hay β^p posibles significados, y $e_{max} - e_{min} + 1$ posibles exponentes, un número de punto flotante se puede codificar en

$$[\log_2(e_{max} - e_{min} + 1)] + [\log_2(\beta^p)] + 1$$

bits, donde el +1 final es para el bit de signo. La codificación precisa no es importante por ahora.

Hay dos razones por las que un número real podría no ser exactamente representable como un número de punto flotante. La situación más común se ilustra con el número decimal 0.1. Aunque tiene una representación decimal finita, en binario tiene una representación repetida infinita. Por lo tanto, cuando $\beta = 2$, el número 0.1 se encuentra estrictamente entre dos números de punto flotante y ninguno de ellos lo puede representar exactamente.

Una situación menos común es que un número real esté fuera de rango, es decir, su valor absoluto sea mayor que $\beta \times \beta^{e_{max}}$ o menor que $1.0 \times \beta^{e_{min}}$. La mayor parte de esta sección analiza cuestiones debidas a la primera razón.

Las representaciones de punto flotante no son necesariamente únicas. Por ejemplo, tanto 0.01×10^1 como 1.00×10^{-1} representan 0.1. Si el dígito inicial es distinto de cero ($d_0 \neq 0$ en la Ec. (2.3)), se dice que la representación está normalizada. El número de punto flotante 1.00×10^{-1} está normalizado, mientras que 0.01×10^1 no lo está.

¿Cuándo es Exacta la Conversión de Base de Punto Flotante de Ida y Vuelta? Suponga que almacenamos un número de punto flotante en la memoria, lo imprimimos en base 10 legible por humanos y lo regresamos a memoria. ¿Cuándo se puede recuperar exactamente el número original?

Suponga que comenzamos con la base β con p lugares de precisión y convertimos a la base γ con q lugares de precisión, redondeando al más cercano, luego volvemos a convertir a la base original β . El teorema de Matula dice que si no hay enteros positivos i y j tales que

$$\beta^i = \gamma^j$$

entonces una condición necesaria y suficiente para que la conversión de ida y vuelta sea exacta (suponiendo que no haya desbordamiento o subdesbordamiento) es que

$$\gamma^{q-1} > \beta^p.$$

En el caso de números de punto flotante (por ejemplo doble en C) tenemos $\beta = 2$ y $p = 53$. (ver Anatomía de un Número de Punto Flotante). Estamos imprimiendo a base $\gamma = 10$. Ninguna potencia positiva de 10 también es una potencia de 2, por lo que se mantiene la condición de Matula en las dos bases.

Si imprimimos $q = 17$ decimales, entonces

$$10^{16} > 2^{53}$$

por lo que la conversión de ida y vuelta será exacta si ambas conversiones se redondean al más cercano. Si q es menor, algunas conversiones de ida y vuelta no serán exactas.

También puede verificar que para un número de punto flotante de precisión simple ($p =$ precisión de 24 bits) necesita $q = 9$ dígitos decimales, y para un número de precisión cuádruple (precisión de $p = 113$ bits) necesita $q = 36$ dígitos decimales¹⁵.

Mirando hacia atrás en el teorema de Matula, claramente necesitamos

$$\gamma^q \geq \beta^p.$$

¿Por qué? Porque el lado derecho es el número de fracciones de base β y el lado izquierdo es el número de fracciones de base γ . No puede tener un mapa

¹⁵El número de bits asignados para la parte fraccionaria de un número de punto flotante es 1 menos que la precisión: la cifra inicial es siempre 1, por lo que los formatos IEEE ahorran un bit al no almacenar el bit inicial, dejándolo implícito. Entonces, por ejemplo, un doble en C tiene una precisión de 53 bits, pero 52 bits de los 64 bits en un doble se asignan para almacenar la fracción.

uno a uno de un espacio más grande a un espacio más pequeño. Entonces, la desigualdad anterior es necesaria, pero no suficiente. Sin embargo, es casi suficiente. Solo necesitamos una base γ con más cifras significativas, es decir, Matula nos dice

$$\gamma^{q-1} > \beta^p$$

es suficiente. En términos de base 2 y base 10, necesitamos al menos 16 decimales para representar 53 bits. Lo sorprendente es que un decimal más es suficiente para garantizar que las conversiones de ida y vuelta sean exactas. No es obvio a priori que cualquier número finito de decimales adicionales sea siempre suficiente, pero de hecho solo uno más es suficiente.

A continuación, se muestra un ejemplo para mostrar que el decimal adicional es necesario. Suponga que $p = 5$. Hay más números de 2 dígitos que números de 5 bits, pero si solo usamos dos dígitos, la conversión de base de ida y vuelta no siempre será exacta. Por ejemplo, el número $17/16$ escrito en binario es 1.0001_{dos} y tiene cinco bits significativos. El equivalente decimal es 1.0625_{diez} , que redondeado a dos dígitos significativos es 1.1_{diez} . Pero el número binario más cercano a 1.1_{diez} con 5 bits significativos es $1.0010_{dos} = 1.125_{diez}$. En resumen, redondeando al más cercano da

$$1.0001_{dos} - > 1.1_{diez} - > 1.0010_{dos}$$

y así no volvemos al punto de partida.

Error de Representación se refiere al hecho de que la mayoría de las fracciones decimales no pueden representarse exactamente como fracciones binarias (en base 2). Esta es la razón principal de por qué muchos lenguajes de programación (Python, Perl, C, C++, Java, Fortran, y tantos otros) frecuentemente no mostrarán el número decimal exacto que esperas.

¿Por qué es eso? $1/10$ no es representable exactamente como una fracción binaria. Casi todas las máquinas de hoy en día usan aritmética de punto flotante: IEEE-754, y casi todas las plataformas mapean los flotantes al «doble precisión» de IEEE-754. Estos «dobles» tienen 53 bits de precisión, por lo tanto en la entrada la computadora intenta convertir 0.1 a la fracción más cercana que puede de la forma $J/(2^N)$ donde J es un entero que contiene exactamente 53 bits. Reescribiendo

$$1/10 \approx J/2^N$$

lo que significa que el valor exacto almacenado en la computadora es igual al valor decimal

0.1000000000000000055511151231257827021181583404541015625.

en lugar de mostrar el valor decimal completo, muchos lenguajes, redondean el resultado a 17 dígitos significativos.

Error de Redondeo Comprimir infinitos números reales en un número finito de bits requiere una representación aproximada. Aunque hay un número infinito de enteros, en la mayoría de los programas el resultado de los cálculos de números enteros se puede almacenar en 32 bits o 64 bits. Por el contrario, dado cualquier número fijo de bits, la mayoría de los cálculos con números reales producirán cantidades que no se pueden representar exactamente usando tantos bits. Por lo tanto, el resultado de un cálculo de punto flotante a menudo debe redondearse para volver a ajustarse a su representación finita. Este error de redondeo es el rasgo característico del cálculo de punto flotante.

Dado que la mayoría de los cálculos de punto flotante tienen errores de redondeo de todos modos, ¿importa si las operaciones aritméticas básicas introducen un poco más de error de redondeo de lo necesario? Esa pregunta es un tema principal a lo largo de esta sección. La sección Dígitos de Guarda analiza los dígitos de protección, un medio para reducir el error al restar dos números cercanos. IBM consideró que los dígitos de guarda eran lo suficientemente importantes que en 1968 añadió un dígito de guarda al formato de doble precisión en la arquitectura System / 360 (la precisión simple ya tenía un dígito de guarda) y modernizó todas las máquinas existentes en el campo.

El estándar IEEE va más allá de sólo requerir el uso de un dígito de protección. Proporciona un algoritmo para la suma, resta, multiplicación, división y raíz cuadrada y requiere que las implementaciones produzcan el mismo resultado que ese algoritmo. Por lo tanto, cuando un programa se mueve de una máquina a otra, los resultados de las operaciones básicas serán los mismos en todos los bits si ambas máquinas admiten el estándar IEEE. Esto simplifica enormemente la portabilidad de programas. Otros usos de esta especificación precisa se dan en operaciones exactamente redondeadas.

Error Relativo y Ulp Dado que el error de redondeo es inherente al cálculo de punto flotante, es importante tener una forma de medir este error. Considere el formato de punto flotante con $\beta = 10$ y $p = 3$, que se utilizará

en esta sección. Si el resultado de un cálculo de punto flotante es 3.12×10^{-2} , y la respuesta cuando se calcula con precisión infinita es 0.0314, está claro que tiene un error de 2 unidades en el último lugar. De manera similar, si el número real 0.0314159 se representa como 3.14×10^{-2} , entonces tiene un error de 0.159 unidades en el último lugar.

En general, si el número de punto flotante $dd..d \times \beta^e$ se usa para representar z , entonces tiene un error de $|d.d\dots d - (z/\beta^e)|^{\beta^{p-1}}$ unidades en el último lugar. El término ulps se utilizará como abreviatura de (units in the last place) "unidades en último lugar". Si el resultado de un cálculo es el número de punto flotante más cercano al resultado correcto, aún podría tener un error de hasta 0.5 ulp.

Otra forma de medir la diferencia entre un número de punto flotante y el número real al que se aproxima es el error relativo, que es simplemente la diferencia entre los dos números divididos por el número real. Por ejemplo, el error relativo cometido al aproximar 3.14159 por 3.14×10^0 es $0.00159/3.141590 \approx 0005$.

Para calcular el error relativo que corresponde a .5 ulp, observe que cuando un número real es aproximado por el número de punto flotante

más cercano posible $\overbrace{d.dd\dots dd}^p \times \beta^e$, el error puede ser tan grande como $\overbrace{0.00\dots 00\beta'}^p \times \beta^e$, donde β' es el dígito $\beta/2$, hay p unidades en el significado del número de punto flotante y p unidades de 0 en el significado del error. Este error es $((\beta/2)\beta^{-p}) \times \beta^e$. Dado que los números de la forma $d.dd\dots dd \times \beta^e$ tienen todos el mismo error absoluto, pero tienen valores que oscilan entre β^e y $\beta \times \beta^e$, el error relativo varía entre $((\beta/2)\beta^{-p}) \times \beta^e/\beta^e$ y $((\beta/2)\beta^{-p}) \times \beta^e/\beta^{e+1}$. Eso es,

$$\frac{1}{2}\beta^{-p} \leq \frac{1}{2}ulp \leq \frac{\beta}{2}\beta^{-p} \quad (2.4)$$

En particular, el error relativo correspondiente a 0.5 ulp puede variar en un factor de β . Este factor se llama bamboleo. Estableciendo $\epsilon = (\beta/2)\beta^{-p}$ en el mayor de los límites en Ec. (2.4) anterior, podemos decir que cuando un número real se redondea al número de punto flotante más cercano, el error relativo siempre está limitado por ϵ , que se conoce como épsilon de la máquina.

En el ejemplo anterior, el error relativo fue $0.00159/3.14159 \approx 0005$. Para evitar números tan pequeños, el error relativo normalmente se escribe como

factor multiplicado ϵ , que en este caso es $\epsilon = (\beta/2)\beta^{-p} = 5(10)^{-3} = 0.005$. Por lo tanto, el error relativo se expresaría como $(0.00159/3.14159)/0.005\epsilon \approx 0.1\epsilon$.

Para ilustrar la diferencia entre ulps y error relativo, considere el número real $x = 12.35$. Se aproxima por $\tilde{x} = 1.24 \times 10^1$. El error es 0.5 ulps, el error relativo es 0.8ϵ . A continuación, considere el cálculo $8\tilde{x}$. El valor exacto es $8x = 98.8$, mientras que el valor calculado es $8\tilde{x} = 9.92 \times 10^1$. El error ahora es 4.0 ulps, pero el error relativo sigue siendo 0.8ϵ . El error medido en ulps es 8 veces mayor, aunque el error relativo es el mismo.

En general, cuando la base es β , un error relativo fijo expresado en ulps puede oscilar en un factor de hasta β . Y a la inversa, como muestra la Ec. (2.4) anterior, un error fijo de 0.5 ulps da como resultado un error relativo que puede oscilar por β .

La forma más natural de medir el error de redondeo es en ulps. Por ejemplo, el redondeo al número de punto flotante más cercano corresponde a un error menor o igual a 0.5 ulp. Sin embargo, al analizar el error de redondeo causado por varias fórmulas, el error relativo es una mejor medida. Dado que se puede sobrestimar el efecto de redondear al número de punto flotante más cercano por el factor de oscilación de β , las estimaciones de error de las fórmulas serán más estrictas en máquinas con una pequeña β .

Cuando sólo interesa el orden de magnitud del error de redondeo, ulps y ϵ pueden usarse indistintamente, ya que difieren como máximo en un factor de β . Por ejemplo, cuando un número de punto flotante tiene un error de n ulps, eso significa que el número de dígitos contaminados es $\log_\beta n$. Si el error relativo en un cálculo es $n\epsilon$, entonces

$$\text{los dígitos contaminados} \approx \log_\beta n. \quad (2.5)$$

Dígito de Guarda Un método para calcular la diferencia entre dos números de punto flotante es calcular la diferencia exactamente y luego redondearla al número de punto flotante más cercano. Esto es muy caro si los operandos difieren mucho en tamaño. Suponiendo que $p = 3$, $2.15 \times 10^{12} - 1.25 \times 10^{-5}$ se calcularía como

$$x = 2.15 \times 10^{12}$$

$$y = 0.000000000000000000125 \times 10^{12}$$

$$x - y = 2.149999999999999999875 \times 10^{12}$$

que se redondea a 2.15×10^{12} . En lugar de utilizar todos estos dígitos, el Hardware de punto flotante normalmente funciona con un número fijo

de dígitos. Suponga que el número de dígitos que se mantiene es p , y que cuando el operando más pequeño se desplaza hacia la derecha, los dígitos simplemente se descartan (en contraposición al redondeo). Entonces $2.15 \times 10^{12} - 1.25 \times 10^{-5}$ se convierte en

$$\begin{aligned}x &= 2.15 \times 10^{12} \\y &= 0.00 \times 10^{12} \\x - y &= 2.15 \times 10^{12}\end{aligned}$$

La respuesta es exactamente la misma que si la diferencia se hubiera calculado exactamente y luego se hubiera redondeado. Tome otro ejemplo: $10.1 - 9.93$. Esto se convierte en

$$\begin{aligned}x &= 1.01 \times 10^1 \\y &= 0.99 \times 10^1 \\x - y &= 0.02 \times 10^1\end{aligned}$$

La respuesta correcta es 0.17, por lo que la diferencia calculada está desviada en 30 ulps y es incorrecta en todos los dígitos. ¿Qué tan grave puede ser el error?.

Teorema 1 *Usando un formato de punto flotante con parámetros β y p , y calculando las diferencias usando p dígitos, el error relativo del resultado puede ser tan grande como $\beta - 1$.*

Cuando $\beta = 2$, el error relativo puede ser tan grande como el resultado, y cuando $\beta = 10$, puede ser 9 veces mayor. O para decirlo de otra manera, cuando $\beta = 2$, la Ec. (2.5) muestra que el número de dígitos contaminados es $\log_2(1/\epsilon) = \log_2(2^p) = p$. Es decir, ¡todos los dígitos p del resultado son incorrectos!. Suponga que se agrega un dígito adicional para protegerse contra esta situación (un dígito de guardia). Es decir, el número más pequeño se trunca a $p+1$ dígitos, y luego el resultado de la resta se redondea a p dígitos. Con un dígito de guardia, el ejemplo anterior se convierte en

$$\begin{aligned}x &= 1.010 \times 10^1 \\y &= 0.993 \times 10^1 \\x - y &= 0.017 \times 10^1\end{aligned}$$

y la respuesta es exacta. Con un solo dígito de guardia, el error relativo del resultado puede ser mayor que ϵ , como en $110 - 8.59$.

$$\begin{aligned}x &= 1.10 \times 10^2 \\y &= 0.085 \times 10^2 \\x - y &= 1.015 \times 10^2\end{aligned}$$

Esto se redondea a 102, en comparación con la respuesta correcta de 101.41, para un error relativo de 0.006, que es mayor que $\epsilon = 0.005$. En general, el error relativo del resultado puede ser solo un poco mayor que ϵ . De forma más precisa:

Teorema 2 *Si x y y son números de punto flotante en un formato con parámetros β y p , y si la resta se realiza con $p+1$ dígitos (es decir, un dígito de guarda), entonces el error de redondeo relativo en el resultado es menor que 2ϵ .*

Cancelación La última sección se puede resumir diciendo que sin un dígito de guarda, el error relativo cometido al restar dos cantidades cercanas puede ser muy grande. En otras palabras, la evaluación de cualquier expresión que contenga una resta (o una suma de cantidades con signos opuestos) podría resultar en un error relativo tan grande que todos los dígitos carecen de significado (Teorema (1)). Al restar cantidades cercanas, los dígitos más significativos de los operandos coinciden y se cancelan entre sí. Hay dos tipos de cancelación: catastrófica y benigna.

La cancelación catastrófica ocurre cuando los operandos están sujetos a errores de redondeo. Por ejemplo, en la fórmula cuadrática, aparece la expresión $b^2 - 4ac$. Las cantidades b^2 y $4ac$ están sujetas a errores de redondeo ya que son el resultado de multiplicaciones de punto flotante. Suponga que están redondeados al número de punto flotante más cercano y, por lo tanto, tienen una precisión de 0.5 ulp. Cuando se restan, la cancelación puede hacer que muchos de los dígitos precisos desaparezcan, dejando principalmente dígitos contaminados por errores de redondeo. Por tanto, la diferencia puede tener un error de muchos ulps. Por ejemplo, considere $b = 3.34$, $a = 1.22$ y $c = 2.28$. El valor exacto de $b^2 - 4ac$ es 0.0292. Pero b^2 se redondea a 11.2 y $4ac$ se redondea a 11.1, por lo que la respuesta final es 0.1, que es un error de 70 ulps, aunque $11.2 - 11.1$ es exactamente igual a 0.1. La resta no introdujo ningún error, sino que expuso el error introducido en las multiplicaciones anteriores.

La cancelación benigna ocurre al restar cantidades exactamente conocidas. Si x e y no tienen error de redondeo, entonces, según el Teorema (2), si la resta se realiza con un dígito de guarda, la diferencia $x - y$ tiene un error relativo muy pequeño (menos de 2ϵ).

Una fórmula que presenta una cancelación catastrófica a veces se puede reorganizar para eliminar el problema. Considere nuevamente la fórmula

cuadrática

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \text{ y } x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \quad (2.6)$$

Cuando $b^2 \gg 4ac$, entonces $b^2 - 4ac$ no implica una cancelación y

$$\sqrt{b^2 - 4ac} \approx |b|.$$

Pero la otra suma (resta) en una de las fórmulas tendrá una cancelación catastrófica. Para evitar esto, multiplique el numerador y denominador de x_1 por $-b - \sqrt{b^2 - 4ac}$ (y de manera similar para x_2) para obtener

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}} \text{ y } x_2 = \frac{-2c}{b - \sqrt{b^2 - 4ac}} \quad (2.7)$$

Si $b^2 \gg ac$ y $b > 0$, entonces calcular x_1 usando la Ec. (2.6) implicará una cancelación. Por lo tanto, use la Ec. (2.7) para calcular x_1 y (2.6) para x_2 . Por otro lado, si $b < 0$, use (2.6) para calcular x_1 y (2.7) para x_2 .

La expresión $x^2 - y^2$ es otra fórmula que presenta una cancelación catastrófica. Es más exacto evaluarlo como

$$(x - y)(x + y)$$

A diferencia de la fórmula cuadrática, esta forma mejorada todavía tiene una resta, pero es una cancelación benigna de cantidades sin error de redondeo, no catastrófica. Según el Teorema (2), el error relativo en $x - y$ es como máximo 2ϵ . Lo mismo ocurre con $x + y$. Multiplicar dos cantidades con un pequeño error relativo da como resultado un producto con un pequeño error relativo.

Errores de Redondeo y de Aritmética La aritmética que realiza una computadora es distinta de la aritmética de nuestros cursos de álgebra o cálculo. En nuestro mundo matemático tradicional consideramos la existencia de números con una cantidad infinita de cifras, en la computadora cada número representable tienen sólo un número finito, fijo de cifras (véase 1.1), los cuales en la mayoría de los casos es satisfactoria y se aprueba sin más, aunque a veces esta discrepancia puede generar problemas.

Un ejemplo de este hecho lo tenemos en el cálculo de raíces de:

$$ax^2 + bx + c = 0$$

cuando $a \neq 0$, donde las raíces se calculan comúnmente con el algoritmo Ec. (2.6) o de forma alternativa con el algoritmo que se obtiene mediante la racionalización del numerador Ec. (2.7).

Otro algoritmo que podemos implementar es el método de Newton-Raphson¹⁶ para buscar raíces, que en su forma iterativa está dado por

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

en el cual se usa x_0 como una primera aproximación a la raíz buscada y x_n es la aproximación a la raíz después de n iteraciones (sí se converge a ella), donde $f(x) = ax^2 + bx + c$ y $f'(x_i) = 2ax + b$.

Salida del Cálculo de Raíces Para resolver el problema, usamos por ejemplo el siguiente código en Python usando programación procedimental:

```
import math
def f(x, a, b, c):
    """ Evalua la Funcion cuadratica """
    return (x * x * a + x * b + c);
def df(x, a, b):
    """ Evalua la derivada de la funcion cuadratica """
    return (2.0 * x * a + b);
def evalua(x, a, b, c):
    """ Evalua el valor X en la función cuadratica """
    print('Raiz (%1.16f), evaluacion raiz: %1.16e' % (x, f(x, a, b, c)))
def metodoNewtonRapson(x, ni, a, b, c):
    """ Metodo Newton-Raphson x = x - f(x)/f'(x) """
    for i in range(ni):
        x = x - (f(x, a, b, c) / df(x, a, b))
    return x
def raices(A, B, C):
    """ Calculo de raices """
```

¹⁶También podemos usar otros métodos, como el de Newton Raphson Modificado para acelerar la convergencia

$$x_{i+1} = x_i - \frac{f(x_i)f'(x_i)}{[f'(x_i)]^2 - f(x_i)f''(x_i)}$$

que involucra la función $f(x)$, la primera derivada $f'(x)$ y a la segunda derivada $f''(x)$.

```

if A == 0.0:
    print("No es una ecuacion cuadratica")
    exit(1)
# Calculo del discriminante
d = B * B - 4.0 * A * C
# Raices reales
if d >= 0.0:
    print('\nPolinomio (%f) X^2 + (%f )X + (%f) = 0\n' % (A, B, C))
    print('\nChicharronera 1')
    X1 = (-B + math.sqrt(d)) / (2.0 * A)
    X2 = (-B - math.sqrt(d)) / (2.0 * A)
    evalua(X1, A, B, C)
    evalua(X2, A, B, C)
    print('\nChicharronera 2')
    X1 = (-2.0 * C) / (B + math.sqrt(d))
    X2 = (-2.0 * C) / (B - math.sqrt(d))
    evalua(X1, A, B, C)
    evalua(X2, A, B, C)
    # Metodo Newton-Raphson
    print("\n\nMetodo Newton-Raphson")
    x = X1 - 1.0;
    print("\nValor inicial aproximado de X1 = %1.16f" % x)
    x = metodoNewtonRapson(x, 6, A, B, C)
    evalua(x, A, B, C);
    x = X2 - 1.0;
    print("\nValor inicial aproximado de X2 = %1.16f" % x);
    x = metodoNewtonRapson(x, 6, A, B, C)
    evalua(x, A, B, C)
    print("\n\n")
else:
    # Raices complejas
    print("Raices Complejas ...")
if __name__ == '__main__':
    raices(1.0, 4.0, 1.0)

```

y generan la siguiente salida:

Polinomio (1.000000) X² + (4.000000)X + (1.000000) = 0

Chicharronera 1

Raiz (-0.2679491924311228), evaluacion raiz: -4.4408920985006262e-16

Raiz (-3.7320508075688772), evaluacion raiz: 0.0000000000000000e+00

Chicharronera 2

Raiz (-0.2679491924311227), evaluacion raiz: 0.0000000000000000e+00

Raiz (-3.7320508075688759), evaluacion raiz: -5.3290705182007514e-15

Metodo Newton-Raphson

Valor inicial aproximado de X1 = -1.2679491924311228

Raiz (-0.2679491924311227), evaluacion raiz: 0.0000000000000000e+00

Valor inicial aproximado de X2 = -4.7320508075688759

Raiz (-3.7320508075688772), evaluacion raiz: 0.0000000000000000e+00

En esta salida se muestra la raíz calculada y su evaluación en la ecuación cuadrática, la cual debería de ser cero al ser una raíz, pero esto no ocurre en general por los errores de redondeo. Además, nótese el impacto de seleccionar el algoritmo numérico adecuado a los objetivos que persigamos en la solución del problema planteado.

En cuanto a la implementación computacional, el paradigma de programación seleccionado depende la complejidad del algoritmo a implementar y si necesitamos reusar el código generado o no. Otras implementaciones computacionales se pueden consultar en las ligas, en las cuales se usan distintos lenguajes ([C](#), [C++](#), [Java](#) y [Python](#)) y diferentes paradigmas de programación ([secuencial](#), [procedimental](#) y [orientada a objetos](#)).

Si lo que necesitamos implementar computacionalmente es una fórmula o conjunto de ellas que generen un código de decenas de líneas, la implementación secuencial es suficiente, si es menor a una centena de líneas puede ser mejor opción la implementación procedimental y si el proyecto es grande o complejo, seguramente se optará por la programación orientada a objetos o formulaciones híbridas de las anteriores.

En última instancia, lo que se persigue en la programación es generar un código: correcto, claro, eficiente, de fácil uso y mantenimiento, que sea flexible, reusable y en su caso portable.

2.2 Trabajando con Punto Flotante

Hay varias trampas en las que incluso los programadores muy experimentados caen cuando escriben código que depende de la aritmética de punto flotante. En esta sección explicamos algunas cosas a tener en cuenta al trabajar con números de punto flotante, es decir, tipos de datos: float (32 bits), double (64 bits) o long double (80 bits).

Problemas de Precisión Como ya vimos en las secciones precedentes, los números de punto flotante se representan en el Hardware de la computadora en fracciones en base 2 (binario). Por ejemplo, la fracción decimal

$$0.125$$

tiene el valor $1/10 + 2/100 + 5/1000$, y de la misma manera la fracción binaria

$$0.001$$

tiene el valor $0/2 + 0/4 + 1/8$. Estas dos fracciones tienen valores idénticos, la única diferencia real es que la primera está escrita en notación fraccional en base 10 y la segunda en base 2.

Desafortunadamente, la mayoría de las fracciones decimales no pueden representarse exactamente como fracciones binarias. Como consecuencia, en general los números de punto flotante decimal que usamos en la computadora son sólo aproximados por los números de punto flotante binario que realmente se guardan en la máquina.

El problema es más fácil de entender primero en base 10. Consideremos la fracción $1/3$. Podemos aproximarla como una fracción de base 10 como: 0.3 o, mejor: 0.33 o, mejor: 0.333 y así. No importa cuántos dígitos desees escribir, el resultado nunca será exactamente $1/3$, pero será una aproximación cada vez mejor de $1/3$.

De la misma manera, no importa cuántos dígitos en base 2 quieras usar, el valor decimal 0.1 no puede representarse exactamente como una fracción en base 2. En base 2, $1/10$ es la siguiente fracción que se repite infinitamente

$$0.0001100110011001100110011001100110011001100110011001100110011...$$

si nos detenemos en cualquier número finito de bits, y tendremos una aproximación. En la mayoría de las máquinas hoy en día, los double se aproximan

usando una fracción binaria con el numerador usando los primeros 53 bits con el bit más significativo y el denominador como una potencia de dos. En el caso de $1/10$, la fracción binaria es

$$3602879701896397/2^{55}$$

que está cerca pero no es exactamente el valor verdadero de $1/10$.

La mayoría de los usuarios no son conscientes de esta aproximación por la forma en que se muestran los valores. Varios lenguajes de programación como C, C++, Java y Python solamente muestra una aproximación decimal al valor verdadero decimal de la aproximación binaria almacenada por la máquina. En la mayoría de las máquinas, si fuéramos a imprimir el verdadero valor decimal de la aproximación binaria almacenada para 0.1, debería mostrar

0.1000000000000000055511151231257827021181583404541015625

esos son más dígitos que lo que la mayoría de la gente encuentra útil, por lo que los lenguajes de programación mantiene manejable la cantidad de dígitos al mostrar en su lugar un valor redondeado

$$1/10$$

como

$$0.1$$

sólo hay que recordar que, a pesar de que el valor mostrado resulta ser exactamente $1/10$, el valor almacenado realmente es la fracción binaria más cercana posible. Esto queda de manifiesto cuando hacemos

$$0.1 + 0.1 + 0.1 \text{ ó } 0.1 * 3 \text{ ó } 0.1 + 0.2^{17}$$

obtendremos

$$0.30000000000000004$$

¹⁷Para el caso de $0.1 + 0.2$ podemos hacer un programa que nos de 0.3, usando:

```
double x = 0.1;
double y = 0.1;
double z = (x*10.0 + y*10.0) / 10.0
```

que es distinto al 0.3 que esperábamos.

Es de hacer notar que hay varios números decimales que comparten la misma fracción binaria más aproximada. Por ejemplo, los números

0.1, 0.100000000000000001 y

0.1000000000000000055511151231257827021181583404541015625

son todos aproximados por $3602879701896397/2^{55}$.

Notemos que esta es la verdadera naturaleza del punto flotante binario: no es un error del lenguaje de programación y tampoco es un error en tu código. Verás lo mismo en todos los lenguajes que soportan la aritmética de punto flotante de tu Hardware (a pesar de que en algunos lenguajes por omisión no muestren la diferencia, o no lo hagan en todos los modos de salida). Para una salida más elegante, quizás quieras usar el formateo de cadenas de texto para generar un número limitado de dígitos significativos.

Ejemplo, el número decimal 9.2 se puede expresar exactamente como una relación de dos enteros decimales $92/10$, los cuales se pueden expresar exactamente en binario como $0b1011100/0b1010$. Sin embargo, la misma proporción almacenada como un número de punto flotante nunca es exactamente igual a 9.2:

usando 32 bits obtenemos: 9.199999809265136

usando 64 bits obtenemos: 9.19999999999999928945

ya que se guarda como la fracción: $5179139571476070/2^{49}$.

Otro ejemplo, si tomamos el caso del número 0.02 y vemos su representación en el lenguaje de programación Python, tenemos que:

```
import decimal
print(decimal.Decimal(0.02))
```

y el resultado es:

0.0200000000000000004163336342344337026588618755340576171875

Además, tenemos la no representabilidad de π (y $\pi/2$), esto significa que un intento de cálculo de $\tan(\pi/2)$ no producirá un resultado de infinito, ni

quiera se desbordará en los formatos habituales de punto flotante. Simplemente no es posible que el Hardware de punto flotante estándar intente calcular $\tan(\pi/2)$, porque $\pi/2$ no se puede representar exactamente. Este cálculo en C:

```
double pi = 3.1415926535897932384626433832795;
tan (pi / 2.0);
```

dará un resultado de $1.633123935319537e + 16$. En precisión simple usando tanf, el resultado será -22877332.0 .

De la misma manera, un intento de cálculo de $\sin(\pi)$ no arrojará cero. El resultado será $1.2246467991473532e - 16$ en precisión doble.

Si bien la suma y la multiplicación de punto flotante son conmutativas ($a + b = b + a$ y $a \times b = b \times a$), no son necesariamente asociativas. Es decir, $(a + b) + c$ no es necesariamente igual a $a + (b + c)$. Usando aritmética decimal significativa de 7 dígitos:

$$a = 1234.567, b = 45.67834, c = 0.0004$$

$(a + b) + c :$

$$\begin{array}{ll} 1234.567 + 45.67834 & 1280.24534 \text{ se redondea a } 1280.245 \\ 1280.245 + 0.0004 & 1280.2454 \text{ se redondea a } 1280.245 \end{array}$$

$a + (b + c) :$

$$\begin{array}{ll} 45.67834 + 0.0004 & 45.67874 \\ 1234.567 + 45.6787 & 1280.24574 \text{ se redondea a } 1280.246 \end{array}$$

Tampoco son necesariamente distributivos. Es decir $(a + b) \times c$ puede no ser lo mismo que $a \times c + b \times c$:

$$\begin{array}{l} 1234.567 * 3.333333 = 4115.223, 1.234567 * 3.333333 = 4.115223, \\ 4115.223 + 4.115223 = 4119.338 \end{array}$$

pero

$$1234.567 + 1.234567 = 1235.802, 1235.802 * 3.333333 = 4119.340$$

Algunos Trucos Supongamos que necesitamos hacer el siguiente cálculo:

$$4.56 * 100$$

la respuesta es:

$$455.99999999999994$$

que no es la esperada. ¿Qué podemos hacer para obtener lo que esperamos?. Por ejemplo, con Python, podemos usar:

```
from sympy import *  
print(nsimplify(4.56 * 100, tolerance=1e-1))
```

que nos dará el 456 esperado.

SymPy es un paquete matemático simbólico para Python, y `nsimplify` toma un número de punto flotante y trata de simplificarlo como una fracción con un denominador pequeño, la raíz cuadrada de un entero pequeño, una expresión que involucra constantes famosas, etc.

Por ejemplo, supongamos que algún cálculo arrojó 4.242640687119286 y sospechamos que hay algo especial en ese número. Así es como puede probar de dónde vino:

```
from sympy import *  
print(nsimplify(4.242640687119286))
```

que nos arrojará:

$$3 * \sqrt{2}$$

Tal vez hagamos un cálculo numérico y encontremos una expresión simple para el resultado y eso sugiera una solución analítica. Creo que una aplicación más común de `nsimplify` podría ser ayudarte a recordar fórmulas medio olvidadas. Por ejemplo, quizás estés oxidado con tus identidades trigonométricas, pero recuerdas que $\cos(\pi/6)$ es algo especial.

```
from sympy import *  
print(nsimplify(cos(pi/6)))
```

que nos entregará:

$$\sqrt{3}/2$$

O, para tomar un ejemplo más avanzado, supongamos que recordamos vagamente que la función gamma toma valores reconocibles en valores semienteros, pero no recordamos exactamente cómo. Tal vez algo relacionado con π o e . Puede sugerir que `nsimplify` incluya expresiones con π y e en su búsqueda.

```
from sympy import *
print(nsimplify(gamma(3.5), constants=[pi, E]))
```

obteniendo:

$$15 * \sqrt{\pi}/8$$

También podemos darle a `nsimplify` una tolerancia, pidiéndole que encuentre una representación simple dentro de una vecindad del número. Por ejemplo, aquí hay una manera de encontrar aproximaciones a π .

```
from sympy import *
print(nsimplify(pi, tolerance=1e-5))
```

obteniendo:

$$355/113$$

con una tolerancia más amplia, devolverá una aproximación más simple.

```
from sympy import *
print(nsimplify(pi, tolerance=1e-2))
```

obteniendo:

$$22/7$$

finalmente, aquí hay una aproximación de mayor precisión a π que no es exactamente simple:

```
from sympy import *
print(nsimplify(pi, tolerance=1e-7))
```

obteniendo:

$$\exp(141/895 + \sqrt{780631})/895$$

Seno de un Googol ¿Cómo se evalúa el seno de un número grande en aritmética de punto flotante? ¿Qué significa el resultado?.

Seno de un billón Comencemos por encontrar el seno de un billón (10^{12}) usando aritmética de punto flotante. Hay un par de maneras de pensar en esto. El número de punto flotante $t = 1.0e12$ solo se puede representar con 15 o 16 cifras decimales significativas (para ser precisos, 53 bits¹⁸), por lo que podría considerarlo como un representante del intervalo de números que tienen todos la misma representación de punto flotante. Cualquier resultado que sea seno de un número en ese intervalo debe considerarse correcto.

Otra forma de ver esto es decir que t se puede representar exactamente -su representación binaria requiere 42 bits y tenemos 53 bits de precisión significativa disponibles-, por lo que esperamos $\sin(t)$ devolver el seno de exactamente un billón, con precisión total.

Resulta que la aritmética del IEEE hace lo último, calculando $\sin(1e12)$ correctamente hasta 16 dígitos.

Aquí está el resultado en Python.

```
sin(1.0e12)
-0.6112387023768895
```

y verificado por Mathematica calculando el valor con 20 decimales

```
In:= N[Sin[10^12], 20]
out= -0.61123870237688949819
```

Reducción de alcance tenga en cuenta que el resultado anterior no es el que obtendría si primero tomara el resto al dividir por 2π y luego tomará el seno.

```
sin(1.0e12 % (2*pi))
-0,6112078499756778
```

¹⁸Un doble IEEE 754 tiene 52 bits significativos, pero estos bits pueden representar 53 bits de precisión porque el primer bit de la parte fraccionaria es siempre 1, por lo que no es necesario representarlo.

Esto tiene sentido. El resultado de dividir un billón por la representación de punto flotante de 2π , 159154943091.89536, es correcto con precisión de punto flotante total. Pero la mayor parte de esa precisión está en la parte entera. La parte fraccionaria solo tiene cinco dígitos de precisión, por lo que debemos esperar que el resultado anterior sea correcto hasta un máximo de cinco dígitos. De hecho, tiene una precisión de cuatro dígitos.

Cuando su procesador calcula, $\sin(1e12)$ no toma ingenuamente el resto por 2π y luego calcula el seno. Si así fuera, obtendríamos cuatro cifras significativas en lugar de 16.

Empezamos diciendo que había dos maneras de ver el problema y, según la primera, devolver sólo cuatro cifras significativas sería bastante razonable. Si pensamos en el valor t como una cantidad medida, medida con la precisión de la aritmética de punto flotante (aunque casi nada se puede medir con tanta precisión), entonces todo lo que deberíamos esperar sería cuatro cifras significativas. Pero las personas que diseñaron la implementación de la función seno en su procesador hicieron más de lo que se esperaba que hicieran, encontrando el seno de un billón con total precisión. Lo hacen utilizando un algoritmo de reducción de rango que conserva mucha más precisión que hacer una división ingenuamente.

¿Seno de un Googol? ¿Qué pasa si intentamos tomar el seno de un número ridículamente grande como un Googol (10^{100})? Esto no funcionará porque un Googol no se puede representar exactamente como un número de punto flotante (es decir, como un doble IEEE 754). No es demasiado grande; Los números de punto flotante pueden ser tan grandes como alrededor de 10^{308} . El problema es que un número tan grande no se puede representar con total precisión. Pero podemos representar 2^{333} exactamente, y un Googol está entre 2^{332} y 2^{333} . Y sorprendentemente, la función sinusoidal de Python (o más bien la función sinusoidal integrada en el procesador AMD de mi computadora) devuelve el resultado correcto con total precisión.

```
sin(2**333)
0.9731320373846353
```

¿Cómo sabemos que esto es correcto? Lo verifiqué en Mathematica:

```
In:= sin[2.0^333]
Out= 0.9731320373846353
```


¿Cómo sabemos que Mathematica tiene razón? Podemos hacer una reducción de rango ingenua usando precisión extendida, digamos 200 decimales.

```
In:= p = N[Pi, 200]
In:= theta = x - IntegerPart[x/ (2 p)] 2 p
Out= 1.8031286178334699559384196689...
```

y verificar que el seno del valor reducido del rango sea correcto.

```
In:= Sin[theta]
Out= 0.97313203738463526...
```

Interpretación Aritmética de Intervalos debido a que los números de punto flotante tienen 53 bits de precisión, todos los números reales entre $2^{56} - 2^2$ y $2^{56} + 2^2$ se representan como el número de punto flotante 2^{56} . Este es un rango de ancho 8, más ancho que 2π , por lo que los senos de los números en este rango cubren los posibles valores de seno, es decir, $[-1, 1]$. Entonces, si piensa en un número de punto flotante como una muestra del conjunto de todos los números reales con la misma representación binaria, cada valor posible de seno es un valor de retorno válido para números mayores que 2^{56} . Pero si piensa que un número de punto flotante se representa exactamente a sí mismo, entonces tiene sentido preguntar por el seno de números como 2^{333} y mayores, hasta los límites de la representación de punto flotante¹⁹.

No Pruebes por la Igualdad Cuando se usa Punto Flotante no es recomendable escribir código como el siguiente²⁰:

```
double x;
double y;
...
if (x == y) {...}
```

¹⁹El mayor exponente de un doble IEEE es 1023, y el mayor significado es $2 - 2^{-53}$ (es decir, todos los unos), por lo que el mayor valor posible de un doble es $(2^{53} - 1)2^{1024-53}$ y de hecho la expresión de Python `sin((2**53 - 1)*2**(1024-53))` devuelve el valor correcto con ¹⁶ cifras significativas.

²⁰Sin pérdida de generalidad usamos algún lenguaje de programación en particular para mostrar los ejemplos, pero esto pasa en todos los lenguajes que usan operaciones de Punto Flotante.

La mayoría de las operaciones de punto flotante implican al menos una pequeña pérdida de precisión y, por lo tanto, incluso si dos números son iguales para todos los fines prácticos, es posible que no sean exactamente iguales hasta el último bit, por lo que es probable que la prueba de igualdad falle. Por ejemplo:

```
double x = 10;
double y = sqrt(x);
y *= y;
if (x == y)
    cout << "La raíz cuadrada es exacta\n";
else
    cout << x-y << "\n";
```

el código imprime: $-1.778636e-015$, aunque en teoría, elevar al cuadrado debería deshacer una raíz cuadrada, la operación de ida y vuelta es ligeramente inexacta. En la mayoría de los casos, la prueba de igualdad anterior debe escribirse de la siguiente manera:

```
double tolerancia = ...
if (fabs(x - y) < tolerancia) {...}
```

Aquí la tolerancia es un umbral que define lo que está "lo suficientemente cerca" para la igualdad. Esto plantea la pregunta de qué tan cerca está lo suficientemente cerca. Esto no puede responderse en abstracto; tienes que saber algo sobre tu problema particular para saber qué tan cerca está lo suficientemente cerca en tu contexto.

Por ejemplo: ¿hay alguna garantía de que la raíz cuadrada de un cuadrado perfecto sea devuelta exactamente?, por ejemplo si hago $\text{sqrt}(81.0) == 9.0$, por lo visto anteriormente, la respuesta es no, pero podríamos cambiar la pregunta por $9.0 * 9.0 == 81.0$, esto funcionará siempre que el cuadrado esté dentro de los límites de la magnitud del punto flotante.

Por otro lado, es posible que las expectativas de las matemáticas no se cumplan en el campo del cálculo de punto flotante. Por ejemplo, se sabe que

$$(x + y)(x - y) = x^2 + y^2$$

y esta otra

$$\sin^2 \theta + \cos^2 \theta = 1$$

sin embargo, no se puede confiar en estos hechos cuando las cantidades involucradas son el resultado de un cálculo de punto flotante. Además, el error de redondeo puede afectar la convergencia y precisión de los procedimientos numéricos iterativos.

Aún más y sin pérdida de generalidad, si comparamos usando el lenguaje de programación Python:

```
print(0.1 + 0.2 == 0.3)
print(0.2 + 0.2 + 0.2 == 0.6)
print(1.2 + 2.4 + 3.6 == 7.2)
print(0.1 + 0.2 <= 0.3)
```

en todos los casos dará un resultado de falso, además:

```
print(10.4 + 20.8 > 31.2)
print(0.8 - 0.1 > 0.7)
```

el resultado será verdadero. Por ello es siempre conveniente ver con que números trabajamos usando algo como:

```
print(format(0.1, ".17g"))
print(format(0.2, ".17g"))
print(format(0.3, ".17g"))
```

así cuando sumemos $0.1 + 0.2$, podemos ver el verdadero resultado:

```
print(print(format(0.1 + 0.2, ".17g")))
```

Teniendo esto en cuenta podemos comparar usando:

```
import math
print(math.isclose(0.1 + 0.2, 0.3))
```

nos dará la respuesta esperada. Podemos ajustar la tolerancia relativa usando:

```
import math
print(math.isclose(0.1 + 0.2, 0.3, rel_tol = 1e-20))
```

que en este caso nos dirá que es falso pues no son iguales en los 20 primeros dígitos solicitados.

Para las comparaciones $\geq$ o $\leq$, por ejemplo para $a + b \leq c$ se debe usar:

```
a, b, c = 0.1, 0.2, 0.3
print(math.isclose(a + b, c) or (a + c < c))
```

¿Cómo Compara Python los Flotantes y los Enteros? veamos los siguientes ejemplos:

```
>>> 9007199254740992 == 9007199254740992.0
True
>>> 9007199254740993 == 9007199254740993.0
False
>>> 9007199254740994 == 9007199254740994.0
True
```

¿Qué fue lo que salió mal?, para desmenuzar lo que pasó, veamos primero:

Representación IEEE-754 de 9007199254740992.0 comencemos con el valor del primer escenario: 9007199254740992.0, resulta que este número es en realidad es 2^{53} . Eso significa que en la representación binaria sería 1 seguido de 53 ceros:

100000000000000000000000000000000000000000000000000000000000.

la forma normalizada será $1.0 * 2^{53}$, que se obtendrá desplazando los bits 53 lugares a la derecha, lo que nos dará el exponente 53.

Además, recordemos que la mantisa tiene solo 52 bits de ancho y estamos desplazando nuestro valor 53 bits hacia la derecha, lo que significa que se perderá el bit menos significativo (LSB) de nuestro valor original. Sin embargo, como todos los bits aquí son 0, no hay ningún daño y aún podemos representar 9007199254740992.0 con precisión. Resumamos los componentes de 9007199254740992.0:

Bit de signo: 0

Exponente: $53 + 1023$ (bias) = 1076 (10000110100 en binario)

[illegible]

Representación IEEE-754:

0100001101000000000000000000000000000000000000000000000000000000

Representación IEEE-754 de 9007199254740993.0 el valor del segundo escenario de prueba es 9007199254740993.0 el que es uno mayor que el valor anterior. Su representación binaria se puede obtener simplemente sumando 1 a la representación binaria del valor anterior:

[illegible]

nuevamente, para convertir esto a la forma normalizada, tendremos que desplazarlo a la derecha 53 bits, lo que nos dará el exponente 53.

Sin embargo, la mantisa tiene sólo 52 bits de ancho. Cuando desplazamos nuestro valor a la derecha en 53 bits, el bit menos significativo (LSB) con el valor 1 se perderá, lo que dará como resultado que la mantisa sea todo ceros. Esto conduce a los siguientes componentes:

Bit de signo: 0

Exponente: $53 + 1023$ (bias) = 1076 (10000110100 en binario)

[illegible]

Representación IEEE-754:

01000011010000000000000000000000000000000000000000000000000000000000

Observe que la representación IEEE-754 de 9007199254740993.0 es la misma que la de 9007199254740992.0. Esto significa que en su representación en memoria del número 9007199254740993.0 en realidad se representa como 9007199254740992.0.

Esto explica por qué Python da el resultado para

$$9007199254740993 == 9007199254740993.0$$

False, porque lo ve como una comparación entre

9007199254740993 y 9007199254740992.0.

Representación IEEE-754 de 9007199254740994.0 El número en el tercer y último escenario de prueba es 9007199254740994.0, que es 1 más que el valor anterior. Su representación binaria se puede obtener sumando 1 a la representación binaria de 9007199254740993.0, dándonos:

[illegible]

Observe que esta vez, el segundo bit menos significativo (LSB) tiene el valor 1. Por lo tanto, cuando desplazamos este número a la derecha 53 bits, se convierte en el LSB de la mantisa (que tiene 52 bits de ancho). Los componentes se ven así:

[illegible]

9007199254740994 == 9007199254740994.0

El estándar IEEE-754 y la aritmética de punto flotante son inherentemente complejos y comparar números de punto flotante no es sencillo. Lenguajes como C y Java implementan la promoción de tipos implícita, convirtiendo números enteros en dobles y comparándolos poco a poco. Sin embargo, esto aún puede dar lugar a resultados inesperados debido a la pérdida de precisión.

Ya sea que esté utilizando C, Java, Python u otro lenguaje, es recomendable utilizar funciones de biblioteca para comparar valores de punto flotante en lugar de realizar comparaciones directas para evitar posibles errores.

Cuando Cómputo de Alto Rendimiento no es Alto Rendimiento

A todo el mundo le importa que los códigos se ejecuten rápidamente en sus computadoras. Las mejoras de Hardware de las últimas décadas lo han hecho posible. Pero, ¿qué tan bien estamos aprovechando las aceleraciones del Hardware?

Considere estos dos ejemplos de código C++. Supongamos aquí $n = 10000000$.

```
void sub(int* a, int* b) {
    for (int i=0; i<n; ++i)
        a[i] = i + 1;
    for (int i=0; i<n; ++i)
        b[i] = a[i];
}

void sub(int* a, int* b) {
    for (int i=0; i<n; ++i) {
        const int j = i + 1;
        a[i] = j;
        b[i] = j;
    }
}
```

¿Cuál corre más rápido? Ambos son simples y dan resultados idénticos (suponiendo que no haya alias). Sin embargo, en las arquitecturas modernas, dependiendo de la configuración de compilación, una generalmente se ejecutará significativamente más rápido que la otra.

En particular, se esperaría que el fragmento 2 se ejecutara más rápido que el fragmento 1. En el fragmento 1, los elementos de la matriz "a", que es demasiado grande para almacenarse en caché, deben recuperarse de la memoria después de escribirse, pero esto no es necesario para Fragmento 2. La tendencia durante más de dos décadas ha sido que la velocidad de computación de los sistemas recién entregados crezca mucho más rápido que la velocidad de la memoria, y la disparidad es extrema hoy en día. El rendimiento de estos núcleos depende casi por completo de la velocidad del ancho de banda de la memoria. Así, el fragmento 2, una versión de bucle fusionado del fragmento 1, mejora la velocidad al reducir el acceso a la memoria principal.

Es poco probable que bibliotecas como C++ STL ayuden, ya que esta operación es demasiado especializada para esperar que una biblioteca la admita (especialmente la versión de bucle fusionado). Además, el compilador no puede fusionar de forma segura los bucles automáticamente sin instrucciones específicas de que los punteros no tengan alias, y aun así no se garantiza que lo haga.

Afortunadamente, los lenguajes informáticos de alto nivel desde la década de 1950 han elevado el nivel de abstracción de la programación para todos nosotros. Naturalmente, a muchos de nosotros nos gustaría simplemente implementar la lógica empresarial requerida en nuestros códigos y dejar que el compilador y el Hardware hagan el resto. Pero, lamentablemente, no siempre se puede simplemente colocar el código en una computadora y esperar que se ejecute rápidamente. Cada vez más, a medida que el Hardware se vuelve más complejo, prestar atención a la arquitectura subyacente es fundamental para obtener un alto rendimiento.

Preocúpate más por la Suma y la Resta que por la Multiplicación y la División Los errores relativos en la multiplicación y división son siempre pequeños. La suma y la resta, por otro lado, pueden resultar en una pérdida completa de precisión. Realmente el problema es la resta; la suma sólo puede ser un problema cuando los dos números que se agregan tienen signos opuestos, por lo que puedes pensar en eso como una resta. Aún así, el código podría escribirse con un "+" que sea realmente una resta.

La resta es un problema cuando los dos números que se restan son casi iguales. Cuanto más casi iguales sean los números, mayor será el potencial de pérdida de precisión. Específicamente, si dos números están de acuerdo con n bits, se pueden perder n bits de precisión en la resta. Esto puede ser más fácil de ver en el extremo: si dos números no son iguales en teoría pero son iguales en su representación de máquina, su diferencia se calculará como cero, 100% de pérdida de precisión.

Aquí hay un ejemplo donde tal pérdida de precisión surge a menudo. La derivada de una función f en un punto x se define como el límite de

$$(f(x+h) - f(x))/h$$

cuando h llega a cero. Entonces, un enfoque natural para calcular la derivada de una función sería evaluar

$$(f(x+h) - f(x))/h$$

para alguna h pequeña. En teoría, cuanto menor es h , mejor se aproxima esta fracción a la derivada. En la práctica, la precisión mejora por un tiempo, pero más allá de cierto punto, valores más pequeños de h resultan en peores aproximaciones a la derivada. A medida que h disminuye, el error de aproximación se reduce pero el error numérico aumenta. Esto se debe a que la resta

$$f(x+h) - f(x)$$

se vuelve problemática. Si toma h lo suficientemente pequeño (después de todo, en teoría, más pequeño es mejor), entonces $f(x+h)$ será igual a $f(x)$ a la precisión de la máquina. Esto significa que todas las derivadas se calcularán como cero, sin importar la función, si solo toma h lo suficientemente pequeño. Aquí hay un ejemplo que calcula la derivada de $\sin(x)$ en $x = 1$.

```
cout << std::setprecision(15);
for (int i = 1; i < 20; ++i)
{
    double h = pow(10.0, -i);
    cout << (sin(1.0+h) - sin(1.0))/h << "\n";
}
cout << "El verdadero resultado es: " << cos(1.0) << "\n";
```

Aquí está la salida del código anterior. Para que la salida sea más fácil de entender, los dígitos después del primer dígito incorrecto se han reemplazado por puntos.

```
0.4.....
0.53.....
0.53.....
0.5402.....
0.5402.....
0.540301.....
0.5403022.....
0.540302302...
0.54030235....
0.5403022.....
0.540301.....
0.54034.....
0.53.....
```

0.544.....

0.55.....

0

0

0

0

El verdadero resultado es: 0.54030230586814

La precisión²¹ mejora a medida que h se hace más pequeña hasta que $h = 10^{-8}$. Pasado ese punto, la precisión decae debido a la pérdida de precisión en la resta. Cuando $h = 10^{-16}$ o menor, la salida es exactamente cero porque $\sin(1.0 + h)$ es igual a $\sin(1.0)$ a la precisión de la máquina. (De hecho, $1 + h$ equivale a 1 a la precisión de la máquina. Más sobre eso a continuación).

¿Qué haces cuando tu problema requiere resta y va a causar una pérdida de precisión? A veces la pérdida de precisión no es un problema; los doubles comienzan con mucha precisión de sobra. Cuando la precisión es importante, a menudo es posible usar algún truco para cambiar el problema de modo que no requiera resta o no requiera la misma resta con la que comenzaste.

Los Números de Punto Flotante Tienen Rangos Finitos Todos saben que los números de punto flotante tienen rangos finitos, pero esta limitación puede aparecer de manera inesperada. Por ejemplo, puede encontrar sorprendente la salida de las siguientes líneas de código

```
float f = 16777216;
cout << f << " " << f+1 << "\n";
```

Este código imprime el valor 16777216 dos veces. ¿Que pasó? De acuerdo con la especificación IEEE para aritmética de punto flotante, un tipo flotante tiene 32 bits de ancho. Veinticuatro de estos bits están dedicados al significado (lo que solía llamarse la mantisa) y el resto al exponente. El número 16777216 es 2^{24} y, por lo tanto, a la variable flotante f no le queda precisión para representar $f + 1$. Ocurriría un fenómeno similar para 2^{53} si f fuera

²¹Los resultados anteriores se calcularon con Visual C++ 2008. Cuando se compiló con gcc 4.2.3 en Linux, los resultados fueron los mismos, excepto los últimos cuatro números. Donde VC++ produjo ceros, gcc produjo números negativos: -0.017 ..., - 0.17 ..., -1.7 ... y 17

del tipo `double` porque un doble de 64 bits dedica 53 bits al significado. El siguiente código imprime 0 en lugar de 1.

```
x = 9007199254740992; // 2^53
cout << ((x+1) - x) << "\n";
```

También podemos quedarnos sin precisión al agregar números pequeños a números de tamaño moderado. Por ejemplo, el siguiente código imprime "¡Lo siento!" porque `DBL_EPSILON` (definido en `float.h`) es el número positivo más pequeño ϵ tal que $1 + \epsilon \neq 1$ cuando se usan tipos dobles.

```
x = 1.0;
y = x + 0.5*DBL_EPSILON;
if (x == y)
    cout << "¡Lo siento!\n";
```

De manera similar, la constante `FLT_EPSILON` es el número positivo más pequeño ϵ tal que $1 + \epsilon$ no es 1 cuando se usan tipos flotantes.

¿Por qué los Flotantes IEEE Tienen Dos Ceros: +0 y -0? Aquí hay un detalle extraño de la aritmética de punto flotante IEEE: las computadoras tienen dos versiones de 0: cero positivo y cero negativo. La mayoría de las veces, la distinción entre +0 y -0 no importa, pero de vez en cuando las versiones firmadas del cero son útiles.

Si una cantidad positiva llega a cero, se convierte en +0. Y si una cantidad negativa llega a cero, se convierte en -0. Podría pensar en +0 (respectivamente, -0) como el patrón de bits para un número positivo (negativo) demasiado pequeño para representarlo.

El estándar de punto flotante IEEE dice que $1/+0$ debería ser *+infinito* y $1/-0$ debería ser *-infinito*. Esto tiene sentido si interpreta $+/-0$ como el fantasma de un número que se desbordó dejando solo su signo. El recíproco de un número positivo (negativo) demasiado pequeño para representarlo es un número positivo (negativo) demasiado grande para representarlo.

Para demostrar esto, ejecute el siguiente código C:

```
int main()
{
    double x = 1e-200;
```

```
double y = 1e-200 * x;
printf("Reciproco de +0: %g\n", 1/y);
y = -1e-200*x;
printf("Reciproco de -0: %g\n", 1/y);
}
```

En Linux con gcc, la salida es:

```
Reciproco de +0: inf
Reciproco de -0: -inf
```

Sin embargo, hay algo acerca de los ceros firmados y las excepciones que no tiene sentido. El informe acertadamente denominado "Lo que todo informático debería saber sobre la aritmética de punto flotante" tiene lo siguiente que decir sobre los ceros con signo.

En aritmética IEEE, es natural definir $\log 0 = -\infty$ y $\log x$ como un *NaN* cuando $x < 0$. Suponga que x representa un pequeño número negativo que se ha desbordado a cero. Gracias al cero con signo, x será negativo, por lo que $\log$ puede devolver un *NaN*. Sin embargo, si no hubiera un cero con signo, la función logarítmica no podría distinguir un número negativo subdesbordado de 0 y, por lo tanto, tendría que devolver $-\infty$.

Esto implica que $\log(-0)$ debe ser *NaN* y $\log(+0)$ debe ser $-\infty$. Eso tiene sentido, pero eso no es lo que sucede en la práctica. La función de $\log$ devuelve $-\infty$ para $+0$ y -0 .

Ejecuté el siguiente código en C:

```
int main()
{
    double x = 1e-200;
    double y = 1e-200 * x;
    printf("Log de +0: %g\n", log(y));
    y = -1e-200*x;
    printf("Log de -0: %g\n", log(y));
}
```

En Linux, el código imprime:

```
Log de +0: -inf
Log de -0: -inf
```

Use Logaritmos para Evitar Desbordamiento y Subdesbordamiento

Las limitaciones de los números de punto flotante descritos en la sección anterior provienen de tener un número limitado de bits en el significado. El desbordamiento y el subdesbordamiento resultan de tener también un número finito de bits en el exponente. Algunos números son demasiado grandes o demasiado pequeños para almacenarlos en un número de punto flotante.

Muchos problemas parecen requerir calcular un número de tamaño moderado como la razón de dos números enormes. El resultado final puede ser representable como un número de punto flotante aunque los resultados intermedios no lo sean. En este caso, los logaritmos proporcionan una salida. Si desea calcular M/N para grandes números M y N , calcule $\log(M) - \log(N)$ y aplique $\exp()$ al resultado. Por ejemplo, las probabilidades a menudo implican proporciones de factoriales, y los factoriales se vuelven astronómicamente grandes rápidamente. Para $N > 170$, $N!$ es mayor que DBL_MAX, el número más grande que puede representarse por un doble (sin precisión extendida). Pero es posible evaluar expresiones como $200!/(190!10!)$ Sin desbordamiento de la siguiente manera:

```
x = exp( logFactorial(200) - logFactorial(190) - logFactorial(10) );
```

Una función logFactorial simple pero ineficiente podría escribirse de la siguiente manera:

```
double logFactorial(int n)
{
    double sum = 0.0;
    for (int i = 2; i <= n; ++i) sum += log((double)i);
    return sum;
}
```

Un mejor enfoque sería utilizar una función de registro gamma si hay una disponible. Consulte Cómo calcular las probabilidades binomiales para obtener más información.

Las Operaciones Numéricas no Siempre Devuelven Números Debido a que los números de punto flotante tienen sus limitaciones, a veces las operaciones de punto flotante devuelven "infinito" como una forma de decir "el resultado es más grande de lo que puedo manejar". Por ejemplo, el siguiente código imprime 1. # INF en Windows e inf en Linux.

```
x = DBL_MAX;
cout << 2*x << "\n";
```

A veces, la barrera para devolver un resultado significativo tiene que ver con la lógica en lugar de la precisión finita. Los tipos de datos de punto flotante representan números reales (a diferencia de los números complejos) y no hay un número real cuyo cuadrado sea -1 . Eso significa que no hay un número significativo que devolver si el código solicita $\text{sqrt}(-2)$, incluso con una precisión infinita. En este caso, las operaciones de punto flotante devuelven *NaN*. Estos son valores de punto flotante que representan códigos de error en lugar de números. Los valores *NaN* se muestran como 1. *#IND* en Windows y *NAN* en Linux.

Una vez que una cadena de operaciones encuentra un *NaN*, todo es un *NaN* de ahí en adelante. Por ejemplo, suponga que tiene un código que equivale a algo como lo siguiente:

```
if (x - x == 0)
    // hacer algo
```

¿Qué podría impedir que se ejecute el código que sigue a la instrucción *if*? Si *x* es un *NaN*, entonces también lo es $x - x$ y los *NaN* no equivalen a nada. De hecho, los *NaN* ni siquiera se igualan. Eso significa que la expresión $x == x$ se puede usar para probar si *x* es un número (posiblemente infinito). Para obtener más información sobre infinitos y *NaN*, consulte las excepciones de punto flotante IEEE en C++.

Trabajando con Factoriales Los cálculos de probabilidad a menudo implican tomar la razón de números muy grandes para producir un número de tamaño moderado. El resultado final puede caber dentro de un doble con espacio de sobra, pero los resultados intermedios se desbordarían. Por ejemplo, suponga que necesita calcular el número de formas de seleccionar 10 objetos de un conjunto de 200. ¡Esto es $200!/(190!10!)$, Aproximadamente $2.2e16$. Pero $200!$ y $190!$ desbordaría el rango de un doble.

Hay dos formas de solucionar este problema. Ambos usan la siguiente regla: Use trucos algebraicos para evitar el desbordamiento.

El primer truco es reconocer que

$$200! = 200 * 199 * 198 * \dots * 191 * 190!$$

y así

$$200!/(190!10!) = 200 * 199 * 198 * \dots * 191/10!$$

esto ciertamente funciona, pero está limitado a factoriales.

Una técnica más general es usar logaritmos para evitar el desbordamiento: tome el logaritmo de la expresión que desea evaluar y luego exponga el resultado. En este ejemplo

$$\log(200!/(190!10!)) = \text{Log}(200!) - \log(190!) - \log(10!)$$

si tiene un código que calcula el logaritmo de los factoriales directamente sin calcular primero los factoriales, puede usarlo para encontrar el logaritmo del resultado que desea y luego aplicar la función *exp*.

Calcular Inverso del Factorial Dado un número positivo x , ¿cómo se puede encontrar un número n tal que $n! = x$, o tal que $n! \approx x$?. El Software matemático tiende a trabajar con la función gamma en lugar de factoriales porque $\Gamma(x)$ extiende $x!$ en números reales, con la relación $\Gamma(x + 1) = x!$. El lado izquierdo a menudo se toma como una definición del lado derecho cuando x no es un número entero positivo.

No sólo preferimos trabajar con la función gamma, sino que es más fácil trabajar con el logaritmo de la función gamma para evitar el desbordamiento.

El siguiente código Python encuentra el inverso del logaritmo de la función gamma utilizando el método de bisección. Este método requiere un límite superior e inferior. Si solo pasamos valores positivos, 0 sirve como límite inferior. Podemos encontrar un límite superior probando potencias de 2 hasta obtener algo lo suficientemente grande.

```
from scipy.special import gammaln
from scipy.optimize import bisect
import math as mt
def inverse_log_gamma(logarg):
    assert(logarg > 0)
    a = 1
    b = 2
    while b < logarg:
        b = b*2
    return bisect(lambda x: gammaln(x) - logarg, a, b)
def inverse_factorial(logarg):
```

```
g = inverse_log_gamma(logarg)
return round(g)-1
```

Tenga en cuenta que la función `inverse_factorial` toma el logaritmo del valor cuyo inverso del factorial desea calcular. Por ejemplo:

```
print(inverse_factorial(mt.log(mt.factorial(42))))
regresa 42.
```

Trabajar en la escala logarítmica nos permite trabajar con números mucho mayores. El factorial de 171 es mayor que el mayor número de doble precisión de IEEE, por lo que `inverse_factorial` no podría devolver ningún valor mayor que 171 si pasáramos x en lugar de $\log x$. Pero al tomar $\log x$ como argumento, podemos calcular factoriales inversos de números tan grandes que el factorial se desborda.

Por ejemplo, supongamos que queremos encontrar el valor de m tal que $m!$ es el factorial más cercano a $\sqrt{(2024!)}$. Podemos usar el código.

```
print(inverse_factorial(gammaln(2025)/2))
```

para encontrar $m = 1112$ aunque $1112!$ es del orden de 10^{2906} , mucho mayor que el mayor número de punto flotante representable, que es del orden de 10^{308} .

Cuando $n! = x$ no tiene una solución exacta, existe alguna opción sobre qué valor de n devolver como inverso del factorial de x . El código anterior minimiza la distancia desde $n!$ a x en una escala logarítmica. Es posible que desee modificar el código para minimizar la distancia en una escala lineal o encontrar la n más grande con $n! < x$, o la n más pequeña con $n! > x$ dependiendo de su aplicación.

Trabajando con Exponenciales y Logaritmos Supongamos que desea evaluar la función²²

$$u(z) = \frac{e^z - 1 - z}{z^2}$$

para valores pequeños de z , digamos $z = 10^{-8}$.

El código Python

²²Este ejemplo proviene de Lloyd N. Trefethen and J. A. C. Weideman. The Exponentially Convergent Trapezoid Rule. SIAM Review. Vol. 56, No. 3. pp. 385-458.


```
import numpy as np
def f(z): return (np.exp(z) - 1 - z)/z**2
print(f(1e-8))
```

imprime:

-0.607747099184471.

Ahora suponga que sospecha dificultades numéricas y calcula su resultado con 50 decimales usando *bc* -l²³ usando:

```
scale=50
z=10^-8
(e(z)-1-z)/z^2
```

imprime:

.500000000166666667083333334166666660000000000000000

Esto sugiere que el cálculo original era completamente erróneo. ¿Qué está sucediendo?

Para z pequeña,

$$e^z \approx 1 + z$$

y así perdemos precisión al evaluar directamente el numerador en la definición de u (en nuestro ejemplo, perdimos toda precisión).

El teorema del valor medio del análisis complejo dice que el valor de una función analítica en un punto es igual al promedio continuo de los valores sobre un círculo centrado en ese punto. Si aproximamos este promedio tomando el promedio de 16 valores en un círculo de radio 1 alrededor de nuestro punto, obtenemos una precisión total. El código Python

```
def g(z):
    N = 16
    ws = z + np.exp(2j*np.pi*np.arange(N)/N)
    return sum(f(ws))/N
print(g(1e-8))
```

²³El comando *bc* (Basic Calculator) de la línea de comandos de Linux/Unix, es una calculadora con grandes posibilidades de uso.

imprime:

```
0.50000000016666668 + 8.673617379884035e-19j
```

que se aparta del resultado calculado con *bc* en el decimosexto decimal. En un nivel alto, evitamos dificultades numéricas al promediar puntos alejados de la región difícil.

Logit inverso A continuación, veamos el cálculo $f(x) = e^x / (1 + e^x)$. (Los estadísticos llaman a esto la función "logit inverso" porque es el inverso de la función que ellos llaman la función "logit".) El enfoque más directo sería calcular $\exp(x) / (1 + \exp(x))$. Veamos dónde se puede romper eso.

```
double x = 1000;
double t = exp(x);
double y = t / (1.0 + t);
```

Imprimir y da -1. # *IND*. Esto se debe a que el cálculo de *t* se desbordó, produciendo un número mayor que el que cabía en un doble. Pero podemos calcular el valor de *y* fácilmente. Si *t* es tan grande que no podemos almacenarlo, entonces $1 + t$ es esencialmente lo mismo que *t* y la relación es muy cercana a 1. Esto sugiere que descubramos qué valores de *x* son tan grandes que $f(x)$ será igual a 1 a la precisión de la máquina, luego solo devuelva 1 para esos valores de *x* para evitar la posibilidad de desbordamiento. Esta es nuestra siguiente regla: No calcules un resultado que puedas predecir con precisión.

El archivo de encabezado `float.h` tiene un `DBL_EPSILON` constante, que es la precisión doble más pequeña que podemos agregar a 1 sin recuperar 1. Un poco de álgebra muestra que si *x* es más grande que $-\log(\text{DBL\_EPSILON})$, entonces $f(x)$ será igual a 1 a la precisión de la máquina. Así que aquí hay un fragmento de código para calcular $f(x)$ para valores grandes de *x*:

```
const double x_max = -log(DBL_EPSILON);
if (x > x_max) return 1.0;
```

El código provisto en esta sección calcula siete funciones que aparecen en las estadísticas. Cada uno evita problemas de desbordamiento, subdesbordamiento o pérdida de precisión que podrían ocurrir para grandes argumentos negativos, grandes argumentos positivos o argumentos cercanos a cero:

- LogOnePlusX calcúlese como $\log(1 + x)$ como en ejemplo antes visto
- ExpMinusOne calcúlese como $e^x - 1$
- Logit calcúlese como $\log(x/(1 - x))$
- LogitInverse calcúlese como $e^x/(1 + e^x)$ como se discutió en el ejemplo último
- LogLogitInverse calcúlese como $\log(e^x/(1 + e^x))$
- LogitInverseDifference calcúlese como $\text{LogitInverse}(x) - \text{LogitInverse}(y)$
- LogOnePlusExpX calcúlese como $\log(1 + \exp(x))$
- ComplementaryLogLog calcúlese como $\log(-\log(1 - x))$
- ComplementaryLogLogInverse calcúlese como $1.0 - \exp(-\exp(x))$

Las soluciones presentadas aquí parecen innecesarias o incluso incorrectas al principio. Si este tipo de código no está bien comentado, alguien lo verá y lo "simplificará" incorrectamente. Estarán orgullosos de todo el desorden innecesario que eliminaron. Y si no prueban valores extremos, su nuevo código parecerá funcionar correctamente. Las respuestas incorrectas y los *NaN* solo aparecerán más tarde.

Log (1 + x) Ahora veamos el ejemplo del cálculo de $\log(x+1)$. Considere el siguiente código:

```
double x = 1e-16;  
double y = log(1 + x)/x;
```

En este código $y = 0$, aunque el valor correcto sea igual a 1 para la precisión de la máquina.

¿Qué salió mal? los números de doble precisión tienen una precisión de aproximadamente 15 decimales, por lo que $1 + x$ equivale a 1 para la precisión de la máquina. El registro de 1 es cero, por lo que y se establece en cero. Pero para valores pequeños de x , $\log(1 + x)$ es aproximadamente x , por lo que $\log(1 + x)/x$ es aproximadamente 1. Eso significa que el código anterior para calcular $\log(1 + x)/x$ devuelve un resultado con 100% de error relativo. Si x no es tan pequeño que $1 + x$ es igual a 1 en la máquina, aún podemos tener problemas. Si x es moderadamente pequeño, los bits en x no se pierden totalmente al calcular $1 + x$, pero algunos sí. Cuanto más se acerca x a 0, más bits se pierden. Podemos usar la siguiente regla: Utilice aproximaciones analíticas para evitar la pérdida de precisión.

La forma favorita de aproximación de los analistas numéricos es la serie de potencia. ¡La serie de potencia para

$$\log(1 + x) = x + x^2/2! + x^3/3! + \dots$$

Para valores pequeños de x , simplemente devolver x para $\log(1 + x)$ es una mejora. Esto funcionaría bien para los valores más pequeños de x , pero para algunos valores no tan pequeños, esto no será lo suficientemente preciso, pero tampoco lo hará directamente el cálculo del $\log(1 + x)$.

La Precisión Arbitraria no es una Panacea Tener acceso a aritmética de precisión arbitraria no necesariamente hace que desaparezcan las dificultades de cálculo numérico. Aún necesitas saber lo que estás haciendo. Antes de ampliar la precisión, hay que saber hasta dónde ampliarla. Si no lo amplía lo suficiente, sus respuestas no serán lo suficientemente precisas. Si lo extiendes demasiado, desperdicias recursos. Tal vez esté bien desperdiciar recursos, por lo que extiende la precisión más de lo necesario. ¿Pero hasta qué punto es más de lo necesario?

Como ejemplo, considere el problema de encontrar valores de n tales que $\tan(n) > n$, uno de los valores es:

$$k = 1428599129020608582548671.$$

Verifiquemos que $\tan(k) > k$. Usaremos *bc* porque admite precisión arbitraria. Nuestra k es un número de 25 dígitos, así que digamos a *bc* que queremos trabajar con 30 decimales para tener un poco de espacio²⁴. *bc* no

²⁴*bc* automáticamente le otorga un poco más de espacio del que solicita, pero le pediremos aún más

tiene una función tangente, pero sí la tiene $s()$ para seno y $c()$ para coseno, por lo que calcularemos la tangente como seno sobre coseno.

```
$ bc -l
escala = 30
k = 1428599129020608582548671
s(k)/c(k) - k
```

Esperamos que esto devuelva un valor positivo, pero en su lugar devuelve

```
-1428599362980017942210629.31...
```

Entonces, ¿está mal la hipótesis? ¿O es *bc* ignorar nuestra solicitud? Resulta que ninguna de las dos cosas es cierta.

No se puede calcular directamente la tangente de un número grande. Utiliza la reducción de rango para reducirlo al problema de calcular la tangente de un ángulo pequeño donde funcionarán sus algoritmos. Dado que la tangente tiene un período π , reducimos $k \bmod \pi$ calculando $k - [k/\pi] \pi$. Es decir, restamos tantos múltiplos de π como podamos hasta obtener un número entre 0 y π . Volviendo a *bc*, calculamos

```
pi = 4*a(1)
k/pi
```

esto regresa

```
454737226160812402842656.500000507033221370444866152761
```

y entonces calculamos la tangente de

```
t = 0,500000507033221370444866152761*pi
= 1,570797919686740002588270178941
```

Como t es ligeramente mayor que $\pi/2$, la tangente será negativa. No podemos tener $\tan(t)$ mayor que k porque $\tan(t)$ ni siquiera es mayor que 0. Entonces, ¿dónde se estropearon las cosas?.

El cálculo de π tuvo una precisión de 30 cifras significativas y el cálculo de k/π tuvo una precisión de 30 cifras significativas, dado nuestro valor de π . Hasta ahora ha funcionado según lo prometido.

El valor calculado de k/π tiene una precisión de 29 cifras significativas, 23 antes del decimal y 6 después. Entonces, cuando tomamos la parte fraccionaria, solo tenemos seis cifras significativas y eso no es suficiente. Ahí es donde las cosas van mal. Obtenemos un valor $[k/\pi]$ que es mayor que 0.5 en el séptimo decimal, mientras que el valor exacto es menor que 0.5 en el vigésimo quinto decimal. Necesitábamos $25 - 6 = 19$ cifras más significativas.

Ésta es la principal dificultad del cálculo en punto flotante: restar números casi iguales pierde precisión. Si dos números coinciden en m decimales, puedes esperar perder m cifras significativas en la resta. El error en la resta será pequeño en relación con los datos de entrada, pero no pequeño en relación con el resultado.

Observe que calculamos k/π hasta 29 cifras significativas y, dado ese resultado, calculamos la parte fraccionaria exactamente. Calculamos con precisión $[k/\pi]\pi$, pero perdimos precisión cuando calculamos y restamos ese valor de k .

Nuestro error al calcular $k - [k/\pi]\pi$ fue pequeño en relación con k , pero no en relación con el resultado final. Nuestra k es del orden de 10^{25} y el error en nuestra resta es del orden de 10^{-7} , pero el resultado es del orden de 1. No hay ningún error en *bc*. Llevó a cabo todos los cálculos con la precisión anunciada, pero no tuvo suficiente precisión de trabajo para producir el resultado que necesitábamos.

2.3 Aritmética de Baja Precisión

La popularidad de la aritmética de baja precisión para el cómputo de alto rendimiento se ha disparado desde el lanzamiento en 2017 de la GPU Nvidia Volta. Los núcleos tensores de media precisión de Volta ofrecieron una enorme ganancia de rendimiento 16 veces mayor que la doble precisión para operaciones clave. Y el rendimiento del Hardware está mejorando aún más: el FP16 con núcleo tensor Nvidia H100 es 58 veces más rápido que el FP64 estándar.

Esta sorprendente aceleración ciertamente llama la atención. Sin embargo, en el cálculo científico, la aritmética de baja precisión suele considerarse insegura para los códigos de modelado y simulación. De hecho, a veces se puede aprovechar una precisión más baja, comúnmente en una configuración de "precisión mixta" en la que sólo partes del cálculo se realizan con baja precisión. Sin embargo, en general, cualquier precisión menor que el doble se considera inadecuada para modelar fenómenos físicos complejos

con fidelidad.

En respuesta, los desarrolladores han creado herramientas para medir la seguridad de la aritmética de precisión reducida en códigos de aplicación. Algunas herramientas pueden incluso identificar qué variables o matrices se pueden reducir de forma segura a una precisión menor sin perder precisión en el resultado final. Sin embargo, el uso de estas herramientas a ciegas, sin el respaldo de algún tipo de proceso de razonamiento, puede resultar peligroso.

Un ejemplo ilustrará esto:

El método del gradiente conjugado para la resolución y optimización de sistemas lineales y el método de Lanczos, estrechamente relacionado, para la resolución de problemas de valores propios mostraron una gran promesa tras su invención a principios de los años cincuenta. Sin embargo, se consideraban inseguros debido a errores de redondeo catastróficos en la aritmética de punto flotante, que son aún más pronunciados a medida que se reduce la precisión del punto flotante.

No obstante, Chris Paige demostró en su trabajo pionero en la década de 1970 que el error de redondeo, aunque sustancial, no excluye la utilidad de los métodos cuando se utilizan correctamente. El método del gradiente conjugado se ha convertido en un pilar del cómputo científico.

Teniendo en cuenta que ninguna herramienta podría llegar a este hallazgo sin un cuidadoso análisis matemático de los métodos. Una herramienta detectaría inexactitudes en el cálculo pero no podría certificar que estos errores no puedan perjudicar el resultado final.

Algunos podrían proponer en cambio un enfoque puramente basado en datos: simplemente pruebe con baja precisión en algunos casos de prueba; si funciona, utilice baja precisión en producción. Sin embargo, este enfoque está lleno de peligros: es posible que los casos de prueba no capturen todas las situaciones que podrían encontrarse en producción.

Por ejemplo, uno podría probar un código de aerodinámica sólo en regímenes de flujo suaves, pero las series de producción pueden encontrar flujos complejos con gradientes pronunciados, que la aritmética de baja precisión no puede modelar correctamente. Los artículos académicos que prueban métodos y herramientas de baja precisión deben evaluarse rigurosamente en escenarios desafiantes del mundo real como este.

Lamentablemente, los equipos de ciencia computacional frecuentemente no tienen tiempo para evaluar sus códigos para un uso potencial de aritmética de menor precisión. Las herramientas ciertamente podrían ayudar. Además, las bibliotecas que encapsulan métodos de precisión mixta pueden ofrecer beneficios a muchos usuarios. Una gran historia de éxito son los solucionadores lineales densos de precisión mixta, basados en el sólido trabajo teórico de Nick Highnam y sus colegas, que han llegado a bibliotecas como: Lu, Hao; Matheson, Michael; Wang, Feiyi; Joubert, Wayne; Ellis, Austin; Oles, Vladyslav. "OpenMxP-OpenSource Mixed Precision Computing,".

Entonces la respuesta final es "depende". Cada nuevo caso debe examinarse cuidadosamente y tomar una decisión basada en alguna combinación de análisis y pruebas.

Sí bien, NVIDIA lidera el mercado de las GPU para inteligencia artificial (IA) con una cuota de mercado aproximada del 80%, pero no es en absoluto la única empresa que tiene en su portfolio chips de vanguardia para IA. La compañía californiana Cerebras posee, de hecho, los procesadores para este escenario de uso más complejos que existen. Su chip WSE-2, por ejemplo, aglutina nada menos que 2.6 billones de transistores contabilizados en la escala numérica larga y 850,000 núcleos optimizados para IA.

Cerebras entrega a sus clientes estos procesadores integrados en una plataforma para IA conocida como CS-2, y precisamente uno de ellos es la compañía de Emiratos Árabes G42. Esta última está construyendo seis superordenadores para IA capaces de superar la barrera de la exaescala que aglutinan una gran cantidad de sistemas CS-2. Y según la CIA algunas de estas máquinas irán a parar a las grandes tecnológicas chinas. No obstante, esto no es todo. Y es que Cerebras dió a conocer en 2024 un procesador para IA aún más potente que su WSE-2.

El procesador WSE-3 Cerebras ya tiene listo su procesador WSE-3 (Wafer Scale Engine 3), un producto que, como podemos intuir, está llamado a suceder al también ambicioso WSE-2.

Ambos procesadores se fabrican a partir de una oblea completa de silicio, lo que permite a Cerebras integrar muchos más bloques funcionales y núcleos en la lógica que una GPU convencional como las que fabrican NVIDIA, AMD o Huawei.

Y es que aglutina 4 billones de transistores, tiene una superficie de 46,225 mm², integra nada menos que 900,000 núcleos optimizados para IA y tiene

una potencia de cálculo, según Cerebras, de 125 petaflops.

Según Cerebras su procesador WSE-3 es el doble de potente que el WSE-2. De hecho, de acuerdo con las especificaciones que ha publicado rinde como 62 GPU H100 de NVIDIA trabajando al unísono, y no debemos pasar por alto que este procesador de la compañía liderada por Jensen Huang es el más potente que tiene hasta que se produzca el lanzamiento de la GPU H200.

Sea como sea Cerebras entrega sus procesadores WSE-3 integrados en un superordenador conocido como CS-3 que es capaz de entrenar grandes modelos de IA con hasta 24 billones de parámetros. El mapa de memoria externa de este superordenador oscila entre 1.5 TB y 1.2 PB, un espacio de almacenamiento descomunal que permite almacenar modelos de lenguaje masivos en un único espacio lógico.

Según informan, el chip WSE-3 optimizado para la IA es capaz de entrenar hasta 24,000 millones de parámetros, lo que también equivaldría a un rendimiento máximo de IA de 125 petaflops.

3 Entornos de Desarrollo y Herramientas de Programación

En los últimos años los lenguajes de programación han ido evolucionado en el desarrollo de sistemas o Software, con el objetivo principal de facilitar al usuario las actividades que realiza día con día; por tal motivo, como programador, es importante conocer los conceptos básicos de programación, los tipos de lenguajes que se utilizan para el desarrollo y su funcionamiento para la interpretación de algoritmos, así como para dar solución a los problemas que pudieran presentarse.

En términos generales, un lenguaje de programación es una herramienta que permite desarrollar Software o programas para computadora. Los lenguajes de programación son empleados para diseñar e implementar programas encargados de definir y administrar el comportamiento de los dispositivos físicos y lógicos de una computadora. Lo anterior se logra mediante la creación e implementación de algoritmos de precisión que se utilizan como una forma de comunicación humana con la computadora.

Los 5 lenguajes de programación más populares en la actualidad son Java, C, Python, C++, y #C según el índice de TIOBE²⁵ actualizado en enero de 2021, por otro lado en la lista de la IEEE a septiembre del 2021 los 10 lenguajes de programación más populares son Python, Java, C, C++, y JavaScript, C#, R, Go, HTML, Swift.

A grandes rasgos, un lenguaje de programación se conforma de una serie de símbolos y reglas de sintaxis y semántica que definen la estructura principal del lenguaje y le dan un significado a sus elementos y expresiones.

Programación es el proceso de análisis, diseño, implementación, prueba y depuración de un algoritmo, a partir de un lenguaje que compila y genera un código fuente ejecutado en la computadora.

La función principal de los lenguajes de programación es escribir programas que permiten la comunicación usuario-máquina. Unos programas especiales (compiladores o intérpretes) convierten las instrucciones escritas en código fuente, en instrucciones escritas en lenguaje máquina.

²⁵Pero si vemos los cinco lenguajes más populares a septiembre de 2020 según el índice TIOBE. La lista consta de lenguajes tan conocidos como C, Java, Python, C++ y C#. Salvo Python, los otros cuatro ya estaban en el TOP 5 allá por 2015 y por 2010. Es más, si comparamos el índice de 2020 y 2019, el único cambio es entre C y Java, que pasan de ser segundo y primero a primero y segundo. El resto permanece igual.

Los intérpretes leen la instrucción línea por línea y obtienen el código máquina correspondiente.

En cuanto a los compiladores, traducen los símbolos de un lenguaje de programación a su equivalencia escrita en lenguaje máquina (proceso conocido como compilación). Por último, se obtiene un programa ejecutable.

Para programar, es necesario como mínimo contar con un editor de texto -como *vi*, *nano* o *micro*- y acceso al compilador o intérprete del lenguaje que nos interese. En Linux se tiene una gran variedad de lenguajes y herramientas de desarrollo -Linux fue hecho por programadores para programadores- que se pueden instalar. Pero, también están los entornos de desarrollo integrado o entorno de desarrollo interactivo -en inglés Integrated Development Environment (IDE)-, estas son aplicaciones informáticas que proporcionan servicios integrales para facilitarle al programador el desarrollo de Software.

Normalmente, un IDE consiste de un editor de código fuente, herramientas de construcción automáticas y un depurador. La mayoría de los IDE tienen auto-completado inteligente de código (IntelliSense). Algunos IDE contienen un compilador, un intérprete, o ambos, tales como NetBeans y Eclipse. El límite entre un IDE y otras partes del entorno de desarrollo de Software más amplio no está bien definido. Muchas veces, a los efectos de simplificar la construcción de la interfaz gráfica de usuario (GUI, por sus siglas en inglés) se integran un sistema controlador de versión y varias herramientas. Muchos IDE modernos también cuentan con un navegador de clases, un buscador de objetos y un diagrama de jerarquía de clases, para su uso con el desarrollo de Software orientado a objetos.

Los IDE están diseñados para maximizar la productividad del programador proporcionando componentes muy unidos con interfaces de usuario similares. Los IDE presentan un único programa en el que se lleva a cabo todo el desarrollo. Generalmente, este programa suele ofrecer muchas características para la creación, modificación, compilación, implementación y depuración de Software. Esto contrasta con el desarrollo de Software utilizando herramientas no relacionadas, como *vi*, GNU Compiler Collection (*gcc*) o *make*.

Uno de los propósitos de los IDE es reducir la configuración necesaria para reconstruir múltiples utilidades de desarrollo, en vez de proveer el mismo conjunto de servicios como una unidad cohesiva. Reduciendo ese tiempo de ajustes, se puede incrementar la productividad de desarrollo, en casos donde aprender a usar un IDE es más rápido que integrar manualmente todas las herramientas por separado.

Una mejor integración de todos los procesos de desarrollo hace posible mejorar la productividad en general, que únicamente ayudando con los ajustes de configuración. Por ejemplo, el código puede ser continuamente armado, mientras es editado, previendo retroalimentación instantánea, como cuando hay errores de sintaxis. Esto puede ayudar a aprender un nuevo lenguaje de programación de una manera más rápida, así como sus librerías asociadas.

Algunos IDE están dedicados específicamente a un lenguaje de programación, permitiendo que las características sean lo más cercanas al paradigma de programación de dicho lenguaje. Por otro lado, existen muchos IDE de múltiples lenguajes tales como *Eclipse*, *ActiveState Komodo*, *IntelliJ IDEA*, *MyEclipse*, *Oracle JDeveloper*, *NetBeans*, *Codenvy* y *Microsoft Visual Studio*. Por otro lado *Xcode*, *Xojo* y *Delphi* están dedicados a un lenguaje cerrado o a un tipo de lenguajes de programación.

Los IDE ofrecen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como *C++*, *Python*, *Java*, *C#*, *Delphi*, *Visual Basic*, etc. En algunos lenguajes, un IDE puede funcionar como un sistema en tiempo de ejecución, en donde se permite utilizar el lenguaje de programación en forma interactiva, sin necesidad de trabajo orientado a archivos de texto. Es deseable que un IDE cuente con las siguientes características:

- Multiplataforma
- Soporte para diversos lenguajes de programación
- Integración con Sistemas de Control de Versiones
- Reconocimiento de Sintaxis
- Extensiones y Componentes para el IDE
- Integración con Framework populares
- Depurador
- Importar y Exportar proyectos
- Múltiples idiomas
- Manual de Usuarios y Ayuda

- Componentes
- Editor de texto
- Compilador.
- Intérprete
- Herramientas de automatización
- Depurador
- Posibilidad de ofrecer un sistema de control de versiones
- Factibilidad para ayudar en la construcción de interfaces gráficas de usuarios

Algunos de los más usados son: *Eclipse*, *Aptana*, *NetBeans*, *Sublime Text*, *Geany*, *Visual Studio*, *Brackets*, *Monodevelop*, *Komodo*, *Anjuta*, *CodeLite*, *Code::Blocks*, *PyDev*, *Eric*, *PyCharm*, *PTK*, *Spyder*, *Bluefish*, *Glade*, *Kdevelop*, *Emacs*, *QtCreator*, *Android SDK*, *WxFormBuilder*, etc.

3.1 Java

Java (véase [12]) es un lenguaje de programación de propósito general, concurrente, orientado a objetos, que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones «escriban el programa una vez y lo ejecuten en cualquier dispositivo (Write Once, Run Anywhere» o WORA)», lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para ejecutarse en otra.

El lenguaje de programación Java fue originalmente desarrollado por James Gosling, de Sun Microsystems (constituida en 1982 y posteriormente adquirida el 27 de enero de 2010 por la compañía Oracle), y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son compiladas a bytecode (clase Java), que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente.

La compañía Sun desarrolló la implementación de referencia original para los compiladores de Java, máquinas virtuales y librerías de clases en 1991, y las publicó por primera vez en 1995. A partir de mayo de 2007, en cumplimiento de las especificaciones del Proceso de la Comunidad Java, Sun volvió a licenciar la mayoría de sus tecnologías de Java bajo la Licencia Pública General de GNU (véase [20]). Otros también han desarrollado implementaciones alternas a estas tecnologías de Sun, tales como el Compilador de Java de GNU y el GNU Classpath.

Orientado a Objetos La primera característica, orientado a objetos (OO), se refiere a un método de programación y al diseño del lenguaje. Aunque hay muchas interpretaciones para OO, una primera idea es diseñar el Software de forma que los distintos tipos de datos que usen, estén unidos a sus operaciones. Así, los datos y el código (funciones o métodos) se combinan en entidades llamadas objetos. Un objeto puede verse como un paquete que contiene el "comportamiento" (el código) y el "estado" (datos). El principio es separar aquello que cambia de las cosas que permanecen inalterables. Frecuentemente, cambiar una estructura de datos implica un cambio en el código que opera sobre los mismos, o viceversa. Esta separación en objetos coherentes e independientes ofrece una base más estable para el diseño de un sistema Software. El objetivo es hacer que grandes proyectos sean fáciles de gestionar y manejar, mejorando como consecuencia su calidad y reduciendo el número de proyectos fallidos.

Otra de las grandes promesas de la programación orientada a objetos es la creación de entidades más genéricas (objetos) que permitan la reutilización del Software entre proyectos, una de las premisas fundamentales de la Ingeniería del Software.

La reutilización del Software ha experimentado resultados dispares, encontrando dos dificultades principales: el diseño de objetos realmente genéricos es pobremente comprendido y falta una metodología para la amplia comunicación de oportunidades de reutilización. Algunas comunidades de código abierto quieren ayudar en este problema dando medios a los desarrolladores para diseminar la información sobre el uso y versatilidad de objetos reutilizables y bibliotecas de objetos.

Independencia de la Plataforma La segunda característica, la independencia de la plataforma, significa que programas escritos en el lenguaje Java

pueden ejecutarse igualmente en cualquier tipo de Hardware. Este es el significado de ser capaz de escribir un programa una vez y que pueda ejecutarse en cualquier dispositivo, tal como reza el axioma de Java, "write once, run anywhere".

Para ello, se compila el código fuente escrito en lenguaje Java, para generar un código conocido como "bytecode" (específicamente Java bytecode), instrucciones máquina simplificadas específicas de la plataforma Java. Esta pieza está "a medio camino" entre el código fuente y el código máquina que entiende el dispositivo destino. El bytecode es ejecutado entonces en la máquina virtual (JVM), un programa escrito en código nativo de la plataforma destino (que es el que entiende su Hardware), que interpreta y ejecuta el código. Además, se suministran bibliotecas adicionales para acceder a las características de cada dispositivo (como los gráficos, ejecución mediante hebras o threads, la interfaz de red) de forma unificada. Se debe tener presente que, aunque hay una etapa explícita de compilación, el bytecode generado es interpretado o convertido a instrucciones máquina del código nativo por «compilación al vuelo JIT (Just In Time)».

Hay implementaciones del compilador de Java que convierten el código fuente directamente en código objeto nativo, como GCJ. Esto elimina la etapa intermedia donde se genera el bytecode, pero la salida de este tipo de compiladores sólo puede ejecutarse en un tipo de arquitectura.

La licencia sobre Java de Sun insiste en que todas las implementaciones sean "compatibles". Esto dio lugar a una disputa legal entre Microsoft y Sun, cuando este último alegó que la implementación de Microsoft no daba soporte a las interfaces RMI y JNI además de haber añadido características "dependientes" de su plataforma. Sun demandó a Microsoft y ganó por daños y perjuicios (unos 20 millones de dólares), así como una orden judicial forzando el acatamiento de la licencia de Sun. Como respuesta, Microsoft no ofrece Java con su versión de sistema operativo, y en recientes versiones de Windows, su navegador Internet Explorer no admite la ejecución de applets sin un «conector (Plugin)» aparte. Sin embargo, Sun y otras fuentes ofrecen versiones gratuitas para distintas versiones de Windows.

Las primeras implementaciones del lenguaje usaban una máquina virtual interpretada para conseguir la portabilidad. Sin embargo, el resultado eran programas que se ejecutaban comparativamente más lentos que aquellos escritos en C o C++. Esto hizo que Java se ganase una reputación de lento en rendimiento. Las implementaciones recientes de la JVM dan lugar a programas que se ejecutan considerablemente más rápido que las versiones antiguas,

empleando diversas técnicas, aunque sigue siendo mucho más lentos que otros lenguajes.

La primera de estas técnicas es simplemente compilar directamente en código nativo como hacen los compiladores tradicionales, eliminando la etapa del bytecode. Esto da lugar a un gran rendimiento en la ejecución, pero tapa el camino a la portabilidad. Otra técnica, conocida como «compilación al vuelo JIT (Just In Time)», convierte el bytecode a código nativo cuando se ejecuta la aplicación. Otras máquinas virtuales más sofisticadas usan una "recompilación dinámica" en la que la VM es capaz de analizar el comportamiento del programa en ejecución y recompila y optimiza las partes críticas. La recompilación dinámica puede lograr mayor grado de optimización que la compilación tradicional (o estática), ya que puede basar su trabajo en el conocimiento que de primera mano tiene sobre el entorno de ejecución y el conjunto de clases cargadas en memoria. La compilación JIT y la recompilación dinámica permiten a los programas Java aprovechar la velocidad de ejecución del código nativo sin por ello perder la ventaja de la portabilidad en ambos.

La portabilidad es técnicamente difícil de lograr, y el éxito de Java en ese campo ha sido dispar. Aunque es de hecho posible escribir programas para la plataforma Java que actúen de forma correcta en múltiples plataformas de distinta arquitectura, el gran número de estas con pequeños errores o inconsistencias llevan a que a veces se parodie el eslogan de Sun, "Write once, run anywhere" como "Write once, debug everywhere" (o "Escríbelo una vez, ejecútalo en cualquier parte" por "Escríbelo una vez, depúralo en todas partes").

El concepto de independencia de la plataforma de Java cuenta, sin embargo, con un gran éxito en las aplicaciones en el entorno del servidor, como los Servicios Web, los Servlets, los Java Beans, así como en sistemas empujados basados en OSGi, usando entornos Java empujados.

El Recolector de Basura En Java el problema de las fugas de memoria se evita en gran medida gracias a la «recolección automática de basura (o automatic garbage collector)». El programador determina cuándo se crean los objetos y el entorno en «tiempo de ejecución de Java (Java runtime)» es el responsable de gestionar el ciclo de vida de los objetos. El programa, u otros objetos, pueden tener localizado un objeto mediante una referencia a este. Cuando no quedan referencias a un objeto, el recolector de basura de Java

borra el objeto, liberando así la memoria que ocupaba previniendo posibles fugas (ejemplo: un objeto creado y únicamente usado dentro de un método sólo tiene entidad dentro de este; al salir del método el objeto es eliminado). Aun así, es posible que se produzcan fugas de memoria si el código almacena referencias a objetos que ya no son necesarios; es decir, pueden aún ocurrir, pero en un nivel conceptual superior. En definitiva, el recolector de basura de Java permite una fácil creación y eliminación de objetos y mayor seguridad.

Instalación de Java e IDEs Existen diversas versiones de Java para Linux, la más usada es JDK de Oracle pero también está una versión abierta llamada OpenJDK, para instalar por ejemplo OpenJDK 17 en Debian GNU/Linux es necesario hacer:

```
# apt install default-jdk
```

o

```
# apt install openjdk-17-jre openjdk-17-jdk openjdk-17-doc
```

si se desea instalar solo el Run-Time JRE, para ello usamos:

```
# apt install default-jre
```

o

```
# apt install openjdk-17-jre
```

y si hay más de una versión instalada, podemos actualizar la versión por omisión de Java:

```
# update-java-alternatives -s java-1.17.0-openjdk-amd64
```

para conocer la versión instalada usamos:

```
$ java -version
```

Además, se pueden instalar diversas herramientas e IDEs para facilitar la programación en Java, para ello usar:

```
# apt install scite jedit kate gedit nedit emacs medit \
kscope geany geany-plugins editra qtcreator anjuta \
anjuta-extras codelite codelite-plugins tea vim-gtk \
mousepad eric neovim neovim-qt medit kwrite katepart
# apt install eclipse eclipse-cdt eclipse-pydev netbeans \
bluefish bluefish-plugins codeblocks codeblocks-contrib
# apt install fte fte-console fte-terminal nano joe vim \
vim-python-jedi vim-tlib vim-latexsuite vim-nox micro \
neovim kakoune vim-athena jed
# apt install kdiff3 meld diffuse dirdiff kompare numdiff \
colordiff dwdiff wdiff xxdiff tkdiff ndiff ccdiff xxdiff
# apt install alioth astyle c2html java2html code2html \
c2html autodia txt2html html2text
# apt install git git-all gitk gitg git-cola git-gui qgit tig \
vim-fugitive git-extras
# apt install mercurial
# apt install subversion rapidsvn
# apt install cvs tkcvs
```

Además, es posible instalar varios editores especializados de las páginas oficiales de sus proyectos:

```
https://netbeans.apache.org/download/index.html
https://www.eclipse.org/downloads/
http://brackets.io/
https://www.jetbrains.com/idea/download/#section=Linux
https://www.oracle.com/tools/downloads/Jdeveloper-12c-downloads.html
http://www.drjava.org/
https://www.jgrasp.org/
https://www.bluej.org/
http://www.jcreator.com/index.htm
https://codenvy.com/
https://atom.io/
https://www.sublimetext.com/
```

Compilar y Ejecutar Para compilar el archivo ejemplo.java usamos *javac* en línea de comandos mediante:

```
$ javac ejemplo.java
```

y lo ejecutamos con:

```
$ java ejemplo
```

Monitoreo de Aplicaciones Java Java Development Kit (JDK) provee binarios, herramientas y compiladores para el desarrollo de aplicaciones en Java. Además incluye una poderosa herramienta de monitoreo que es *jconsole*. Podemos ejecutarla en una terminal de Linux usando:

```
$ jconsole
```

al lanzar la aplicación, nos preguntará si deseamos hacer el monitoreo local o de algún equipo conectado en red. Si seleccionamos local, entonces podemos ver el consumo de las aplicaciones que corren en la máquina virtual de Java en tiempo real de CPU, memoria, Threads, Clases, etc.

Además podemos usar el comando *jps* (Java Virtual Machine Process Status) que nos permite conocer cada proceso que corre en la máquina virtual de Java. El uso básico es:

```
$ jps
```

pero podemos pedirle más información usando:

```
$ jps -v
```

esta última nos proporciona el indicador de proceso y el nombre de la clase o archivo *Jar* que se detecte en cada instancia.

Crear y Ejecutar Archivos .jar Un archivo *.jar* (Java ARchive) es un formato de archivo independiente de la plataforma que se utiliza para agregar muchos archivos de clase Java, metadatos y recursos asociados, como texto, imágenes, etc., en un solo archivo para su distribución.

Permite que los tiempos de ejecución de Java implementen de manera eficiente una aplicación completa en un archivo de almacenamiento y brinda muchos beneficios, como seguridad, sus elementos pueden comprimirse, acortar los tiempos de descarga, permite el sellado y control de versiones de paquetes, admite la portabilidad. También es compatible con el empaquetado para extensiones.

Para crear y ejecutar archivos .jar necesitamos hacer lo siguiente:

1. Primero comencemos escribiendo una clase Java simple con un método principal para una aplicación llamada MiApp, con fines de demostración.

```
$ nano MiApp.java
```

Copie y pegue el siguiente código en el archivo MiApp.java.

```
public class MiApp {  
    public static void main(String[] args){  
        System.out.println("Solo ejecuta MiApp");  
    }  
}
```

Grabe el archivo y cierre este.

- 2 A continuación, necesitamos compilar y empaquetar la clase en un archivo JAR usando las utilidades javac y jar como se muestra:

```
$ javac -d . MiApp.java  
$ ls  
$ jar cvf MiApp.jar MiApp.class  
$ ls
```

- 3 Una vez creado MiApp.jar, ahora podemos ejecutar el archivo usando el comando java como se muestra:

```
$ java -jar MiApp.jar  
no main manifest attribute, in MiApp.jar
```

De la salida del comando anterior, encontramos un error. La JVM (Java Virtual Machine) no pudo encontrar nuestro atributo de manifiesto principal, por lo que no pudo ubicar la clase principal que contiene el método principal (`public static void main (String [] args)`).

El archivo JAR debe tener un manifiesto que contenga una línea con el formato `Main-Class: classname` que defina la clase con el método principal que sirve como punto de partida de nuestra aplicación.

- 4 Para corregir el error anterior, necesitaremos actualizar el archivo JAR para incluir un atributo de manifiesto junto con nuestro código. Creemos un archivo `MANIFEST.MF`:

```
$ nano MANIFEST.MF
```

Copie y pegue la siguiente línea en el archivo `MANIFEST.MF`:

```
Main-Class: MiApp
```

Guarde el archivo y agreguemos el archivo `MANIFEST.MF` a nuestro `MiApp.jar` usando el siguiente comando:

```
$ jar cvmf MANIFEST.MF MiApp.jar MiApp.class
```

- 5 Finalmente, cuando ejecutamos el archivo JAR nuevamente, debería producir el resultado esperado como se muestra en la salida:

```
$ java -jar MiApp.jar  
Solo ejecuta MiApp
```

Para obtener más información, debemos consultar las páginas de manual de los comandos `java`, `javac` y `jar`.

```
$ man java  
$ man javac  
$ man jar
```

3.2 Python

Python (véase [13]) es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License, que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores (véase apéndice 10.2).

Python fue creado a finales de los ochenta por Guido van Rossum en el Centro para las Matemáticas y la Informática (CWI, Centrum Wiskunde & Informatica), en los Países Bajos, como un sucesor del lenguaje de programación ABC, capaz de manejar excepciones e interactuar con el sistema operativo Amoeba. Van Rossum es el principal autor de Python, y su continuo rol central en decidir la dirección de Python es reconocido, refiriéndose a él como Benevolente Dictador Vitalicio (en inglés: Benevolent Dictator for Life, BDFL).

Características y paradigmas Python es un lenguaje de programación multiparadigma. Esto significa que más que forzar a los programadores a adoptar un estilo particular de programación, permite varios estilos: programación orientada a objetos, programación imperativa y programación funcional. Otros paradigmas están soportados mediante el uso de extensiones.

Python usa tipado dinámico y conteo de referencias para la administración de memoria.

Una característica importante de Python es la resolución dinámica de nombres; es decir, lo que enlaza un método y un nombre de variable durante la ejecución del programa (también llamado enlace dinámico de métodos).

Otro objetivo del diseño del lenguaje es la facilidad de extensión. Se pueden escribir nuevos módulos fácilmente en *C* o *C++*. Python puede incluirse en aplicaciones que necesitan una interfaz programable.

Aunque la programación en Python podría considerarse en algunas situaciones hostil a la programación funcional tradicional del Lisp, existen bastantes analogías entre Python y los lenguajes minimalistas de la familia Lisp como puede ser Scheme.

El intérprete de Python estándar incluye un modo interactivo en el cual

se escriben las instrucciones en una especie de intérprete de comandos: las expresiones pueden ser introducidas una a una, pudiendo verse el resultado de su evaluación inmediatamente, lo que da la posibilidad de probar porciones de código en el modo interactivo antes de integrarlo como parte de un programa. Esto resulta útil tanto para las personas que se están familiarizando con el lenguaje como para los programadores más avanzados.

Existen otros programas, tales como IDLE, bpython o IPython, que añaden funcionalidades extra al modo interactivo, como el autocompletado de código y el coloreado de la sintaxis del lenguaje.

Elementos del lenguaje Python fue diseñado para ser leído con facilidad. Una de sus características es el uso de palabras donde otros lenguajes utilizarían símbolos. Por ejemplo, los operadores lógicos: `!`, `||` y `&&`, en Python se escriben; `not`, `or` y `and`, respectivamente. Curiosamente el lenguaje Pascal es junto con COBOL uno de los lenguajes con muy clara sintaxis y ambos son de la década de los 70. La idea del código claro y legible no es algo nuevo.

El contenido de los bloques de código (bucles, funciones, clases, etc.) es delimitado mediante espacios o tabuladores, conocidos como indentación, antes de cada línea de órdenes pertenecientes al bloque. Python se diferencia así de otros lenguajes de programación que mantienen como costumbre declarar los bloques mediante un conjunto de caracteres, normalmente entre llaves `{}`. Se pueden utilizar tanto espacios como tabuladores para indentar el código, pero se recomienda no mezclarlos.

Debido al significado sintáctico de la indentación, cada instrucción debe estar contenida en una sola línea. No obstante, si por legibilidad se quiere dividir la instrucción en varias líneas, añadiendo una barra invertida: `\` al final de una línea, se indica que la instrucción continúa en la siguiente.

Variables Las variables se definen de forma dinámica, lo que significa que no se tiene que especificar cuál es su tipo de antemano y puede tomar distintos valores en otro momento, incluso de un tipo diferente al que tenía previamente. Se usa el símbolo `=` para asignar valores.

Módulos Existen muchas propiedades que se pueden agregar al lenguaje importando módulos, que son "minicódigos" (la mayoría escritos también en Python) que proveen de ciertas funciones y clases para realizar determinadas tareas. Un ejemplo es el módulo: Tkinter, que permite crear interfaces grá-

ficas basadas en la biblioteca Tk. Otro ejemplo es el módulo: `os`, que provee acceso a muchas funciones del sistema operativo. Los módulos se agregan a los códigos escribiendo la palabra reservada *import* seguida del nombre del módulo que queramos usar.

Python tiene una gran biblioteca estándar, usada para una diversidad de tareas. Esto viene de la filosofía "pilas incluidas" ("batteries included") en referencia a los módulos de Python²⁶. Los módulos de la biblioteca estándar pueden mejorarse por módulos personalizados escritos tanto en *C* como en Python. Debido a la gran variedad de herramientas incluidas en la biblioteca estándar, combinada con la habilidad de usar lenguajes de bajo nivel como *C* y *C++*, los cuales son capaces de interactuar con otras bibliotecas, Python es un lenguaje que combina su clara sintaxis con el inmenso poder de lenguajes menos elegantes.

Algunos Módulos para Python

TensorFlow Models sirve para el aprendizaje automático y aprendizaje profundo. TensorFlow Models es el repositorio de fuente abierta para encontrar muchas bibliotecas y modelos relacionados con el aprendizaje profundo.

Keras es una API de redes neuronales de alto nivel, escrita en Python y es capaz de ejecutarse sobre TensorFlow, CNTK o Theano. Fue desarrollado con un enfoque para permitir la experimentación rápida.

Frasco es un framework ligero de aplicaciones Web WSGI. está diseñado para que el inicio sea rápido y fácil, con la capacidad de escalar hasta aplicaciones complejas. Comenzó como un simple envoltorio alrededor de Werkzeug y Jinja y se ha convertido en uno de los frameworks de aplicación Web Python más populares.

Scikit-learn es un módulo de Python para el aprendizaje automático construido sobre SciPy y distribuido bajo la licencia BSD.

²⁶Una lista de módulos disponibles en Python está en su página oficial.

Para la versión 2 en: <https://docs.python.org/2/py-modindex.html>

Para la versión 3 en: <https://docs.python.org/3/py-modindex.html>

Zulip es una poderosa aplicación de chat grupal de código abierto que combina la inmediatez del chat en tiempo real con los beneficios de productividad de las conversaciones enhebradas. Zulip es utilizado por proyectos de código abierto, compañías de Fortune 500, cuerpos de grandes estándares y otros que necesitan un sistema de chat en tiempo real que les permita a los usuarios procesar fácilmente cientos o miles de mensajes al día. Con más de 300 colaboradores que fusionan más de 500 commits por mes, Zulip es también el proyecto de chat grupal de código abierto más grande y de más rápido crecimiento.

Django es un framework Web Python de alto nivel que fomenta un desarrollo rápido y un diseño limpio y pragmático de desarrollo Web de código abierto, escrito en Python, que respeta el patrón de diseño conocido como modelo-vista-template. Fue desarrollado en origen para gestionar varias páginas orientadas a noticias de la World Company de Lawrence, Kansas y fue liberada al público bajo una licencia BSD en julio del 2005.

Rebound es una herramienta de línea de comandos que obtiene instantáneamente los resultados de desbordamiento de pila cuando se produce un error de compilación.

Google Images Download Este es un programa de línea de comando de Python para buscar palabras clave / frases clave en Google Imágenes y opcionalmente descargar imágenes a su computadora. También puede invocar este script desde otro archivo Python.

YouTube-dl es usado para descargar videos de: youtube.com u otras plataformas de video.

System Design Primer este repositorio es una colección organizada de recursos para ayudar a aprender a construir sistemas a escala.

Mask R-CNN es para detección y segmentación de objetos. Esta es una implementación de Mask R-CNN en Python 3, Keras y TensorFlow. El modelo genera cuadros de delimitación y máscaras de segmentación para cada instancia de un objeto en la imagen. Se basa en Feature Pyramid Network (FPN) y ResNet101 backbone.

Face Recognition es usado para reconocer y manipular caras desde Python o desde la línea de comandos con la biblioteca de reconocimiento facial más simple del mundo. Esto también proporciona una herramienta de línea de comandos: `face_recognition` simple que permite hacer reconocimiento de rostros en una carpeta de imágenes desde la línea de comandos.

Snallygaster Herramienta para buscar archivos secretos en servidores HTTP.

Ansible es un sistema de automatización de TI radicalmente simple. Maneja la administración de configuraciones, la implementación de aplicaciones, el aprovisionamiento en la nube, la ejecución de tareas ad-hoc y la orquestación multinodo, incluida la trivialización de cosas como actualizaciones continuas de tiempo de inactividad cero con balanceadores de carga.

Detectron es el sistema de software de Facebook AI Research que implementa algoritmos de detección de objetos de última generación, incluyendo Mask R-CNN, está escrito en Python y funciona con el marco de aprendizaje profundo Caffe2.

Asciinema registrador de sesión de terminal y el mejor compañero de `asciinema.org`.

HTTPIe es un cliente HTTP de línea de comando. Su objetivo es hacer que la interacción de la CLI con los servicios Web sea lo más amigable posible para los humanos. Proporciona un comando `http` simple que permite el envío de solicitudes HTTP arbitrarias utilizando una sintaxis simple y natural, y muestra una salida coloreada. HTTPIe se puede usar para probar, depurar y, en general, interactuar con servidores HTTP.

You-Get es una pequeña utilidad de línea de comandos para descargar contenidos multimedia (videos, audios, imágenes) desde la Web, en caso de que no haya otra forma práctica de hacerlo.

Sentry es un servicio que ayuda a controlar y corregir fallas en tiempo real. El servidor está en Python, pero contiene una API completa para enviar eventos desde cualquier lenguaje, en cualquier aplicación.

Tornado es un framework Web de Python y una biblioteca de red asíncrona, desarrollada originalmente en FriendFeed. Mediante el uso de E/S de red sin bloqueo, Tornado puede escalar a decenas de miles de conexiones abiertas, lo hace ideal para largos sondeos, WebSockets y otras aplicaciones que requieren una conexión de larga duración para cada usuario.

Magenta es un proyecto de investigación que explora el papel del aprendizaje automático en el proceso de creación de arte y música. Principalmente, esto implica desarrollar nuevos algoritmos de aprendizaje profundo y aprendizaje de refuerzo para generar canciones, imágenes, dibujos y otros materiales. Pero también es una exploración en la construcción de herramientas e interfaces inteligentes que permiten a artistas y músicos ampliar sus procesos utilizando estos modelos.

ZeroNet crea sitios Web descentralizados utilizando Bitcoin Crypto y la red BitTorrent.

Gym OpenAI Gym es un conjunto de herramientas para desarrollar y comparar algoritmos de aprendizaje de refuerzo. Esta es la biblioteca de código abierto de Gym, que le da acceso a un conjunto estandarizado de entornos.

Pandas es un paquete de Python que proporciona estructuras de datos rápidas, flexibles y expresivas diseñadas para que trabajar con datos "relacionales" o "etiquetados" sea fácil e intuitivo. Su objetivo es ser el componente fundamental de alto nivel para hacer un análisis práctico y real de datos en Python. Además, tiene el objetivo más amplio de convertirse en la herramienta de análisis / manipulación de datos de código abierto más potente y flexible disponible en cualquier lenguaje.

Luigi es un paquete de Python que te ayuda a construir tuberías complejas de trabajos por lotes. Maneja la resolución de dependencia, la administración del flujo de trabajo, la visualización, el manejo de fallas, la integración de línea de comando y mucho más.

SpaCy (by Explosion AI) es una biblioteca para el procesamiento avanzado del lenguaje natural en Python y Cython, está basado en las últimas investigaciones y fue diseñado desde el primer día para ser utilizado en productos reales. SpaCy viene con modelos estadísticos precompilados y vectores de palabras, y actualmente admite tokenización para más de 20 lenguajes. Cuenta con el analizador sintáctico más rápido del mundo, modelos de redes neuronales convolucionales para etiquetado, análisis y reconocimiento de una entidad nombrada y fácil integración de aprendizaje profundo.

Theano es una biblioteca de Python que permite definir, optimizar y evaluar expresiones matemáticas que involucran matrices multidimensionales de manera eficiente. Puede usar GPU y realizar una diferenciación simbólica eficiente.

TFlearn es una biblioteca de aprendizaje profundo modular y transparente construida sobre Tensorflow. Fue diseñada para proporcionar una API de nivel superior a TensorFlow con el fin de facilitar y agilizar la experimentación, sin dejar de ser totalmente transparente y compatible con ella.

Kivy es un framework Python de código abierto y plataforma para el desarrollo de aplicaciones que hacen uso de interfaces de usuario innovadoras y multitáctiles. El objetivo es permitir un diseño de interacción rápido y fácil y un prototipado rápido a la vez que hace que su código sea reutilizable.

Mailpile es un cliente de correo electrónico moderno y rápido con características de cifrado y privacidad fáciles de usar. El desarrollo de Mailpile esta financiado por una gran comunidad de patrocinadores y todo el código relacionado con el proyecto es y será lanzado bajo una licencia de Software Libre aprobada por OSI.

Matplotlib es una biblioteca de trazado 2D de Python que produce figuras con calidad de publicación en una variedad de formatos impresos y entornos interactivos en todas las plataformas. Matplotlib se puede utilizar en scripts Python, el shell Python e IPython, así como en servidores de aplicaciones Web y varios toolkits de interfaz gráfica de usuario.

YAPF (by Google) toma el código y lo reformatea con el mejor formato que cumpla con la guía de estilo, incluso si el código original no viola la guía de estilo.

Cookiecutter una utilidad de línea de comandos que crea proyectos desde cookiecutters (plantillas de proyecto), por ejemplo creando un proyecto de paquete Python a partir de una plantilla de proyecto de paquete Python.

HTTP Prompt es un cliente HTTP interactivo de línea de comandos con autocompletado y resaltado de sintaxis, basado en `prompt_toolkit` y `HTTPie`.

Speedtest-cli interfaz de línea de comandos para probar el ancho de banda de Internet con `speedtest.net`: <http://www.speedtest.net/>

Pattern es un módulo de minería Web para Python. Tiene herramientas para Minería de datos, Procesamiento de lenguaje natural, Aprendizaje automático y Análisis de red.

Goocy (Beta) convierte (casi) cualquier programa de consola Python 2 o 3 en una aplicación GUI con una línea.

Wagtail CMS es un sistema de gestión de contenido creado en Django. Se centra en la experiencia del usuario y ofrece un control preciso para diseñadores y desarrolladores.

Bottle es un micro-Framework WSGI rápido, simple y liviano para Python. Se distribuye como un módulo de archivo único y no tiene dependencias distintas de la biblioteca estándar de Python.

Prophet (by Facebook) es un procedimiento para pronosticar datos de series temporales. Se basa en un modelo aditivo en el que las tendencias no lineales se ajustan a la estacionalidad anual y semanal, más las vacaciones. Funciona mejor con datos de periodicidad diaria con al menos un año de datos históricos. Prophet es robusto para datos faltantes, cambios en la tendencia y grandes valores atípicos.

Falcon es un marco Web de Python confiable y de alto rendimiento para construir Backend de aplicaciones a gran escala y microservicios. Fomenta el estilo arquitectónico REST e intenta hacer lo mínimo posible sin dejar de ser altamente efectivo.

Mopidy es un servidor de música extensible escrito en Python. Mopidy reproduce música desde el disco local, Spotify, SoundCloud, Google Play Music y más. Edita la lista de reproducción desde cualquier teléfono, tableta o computadora usando una gama de clientes MPD y Web.

Hug tiene como objetivo hacer que el desarrollar APIs impulsadas por Python sea lo más simple posible, pero no más simple. Como resultado, simplifica drásticamente el desarrollo de la API de Python.

SymPy es una biblioteca de Python para matemática simbólica.

Visdom es una herramienta flexible para crear, organizar y compartir visualizaciones de datos vivos y enriquecidos. Admite Torch y Numpy.

Pygame es una biblioteca de plataforma cruzada diseñada para facilitar la escritura de software multimedia, como juegos en Python.

Requests es una biblioteca de Python que le permite enviar solicitudes HTTP / 1.1, agregar encabezados, datos de formularios, archivos multiparte y parámetros con simples diccionarios de Python. También le permite acceder a los datos de respuesta de la misma manera.

Statsmodels es un paquete de Python que proporciona un complemento para Scipy para cálculos estadísticos que incluyen estadística descriptiva y estimación e inferencia para modelos estadísticos.

Scrapy es ampliamente utilizada en la biblioteca de raspado Web de Python. Se usa para crear programas de rastreo. Inicialmente, fue diseñado para raspar, como su nombre indica, pero ahora se usa para muchos propósitos, incluida la extracción de datos, las pruebas automatizadas, etc. Scrapy es de código abierto.

PyTorch es una biblioteca de código abierto, básicamente es un reemplazo de la biblioteca Numpy y está equipada con funcionalidades de nivel superior para construir redes neuronales profundas. Se puede usar otro lenguaje como Scipy, Cython y Numpy, que ayudan a extender PyTorch cuando sea necesario. Muchas organizaciones, incluyendo Facebook, Twitter, Nvidia, Uber y otras organizaciones usan Pytorch para la creación rápida de prototipos en investigación y para entrenar modelos de aprendizaje profundo.

Requests es una de las famosas bibliotecas de Python que tiene licencia bajo Apache2 y esta escrita en Python. Esta biblioteca ayuda a los humanos a interactuar con los lenguajes. Con la biblioteca de solicitudes, no es necesario que agregue consultas, cadenas manualmente a las URL ni codificar los datos POST. Se puede enviar solicitudes HTTP al servidor mediante la biblioteca de solicitudes y se puede agregar datos de formularios, contenido como encabezado, archivos en varias partes, etc.

PyFlux es una biblioteca de Python que se usa para predecir y analizar series temporales, está desarrollado por Ross Taylor, esta biblioteca tiene muchas opciones para la interfaz y contiene muchas clases nuevas de tipos de modelos. Pyflux permite a los usuarios implementar muchos modelos modernos de series de tiempo como GARCH y predecir la naturaleza de cómo reaccionará en el futuro.

Zappa es uno de los mejores paquetes de Python creados por Miserlou, es tan fácil de construir e implementar aplicaciones sin servidor en API Gateway y Amazon Web Services Lambda. Dado que AWS maneja la escala horizontal de forma automática, por lo que no habrá tiempo de espera de solicitud. Con Zappa, puede actualizar su código en una sola línea con Zappa.

Arrow es una famosa biblioteca de Python amigable para los humanos que ofrece funciones sensatas como crear, formatear, manipular y convertir fechas, horas y marcas de tiempo. Es compatible con Python 2 y 3 y es una alternativa de fecha y hora, ofrece funciones completas con una interfaz más agradable.

Pendulum es un paquete de Python que se utiliza para manipular fechas y horas, el código seguirá funcionando si se reemplazan todos los elementos de `DateTime`. Con `Pendulum`, se puede analizar `DateTime` y mostrar la fecha y hora con la zona horaria. Básicamente, `Pendulum` es una versión mejorada de la biblioteca `Arrow` y tiene todos los métodos útiles como redondear, truncar, convertir, analizar, formatear y aritmética.

Theano es una biblioteca de aprendizaje profundo de Python, que se utiliza para optimizar, definir y evaluar ecuaciones numéricas matemáticas y matrices multidimensionales, está desarrollado por el grupo de aprendizaje automático, por lo que, básicamente, `Theano` es un compilador de expresión matemática y proporciona una estrecha integración con `Numpy` y proporciona una optimización rápida y estable.

IPython esta es una de las herramientas de Python más útiles, ya que proporciona una rica arquitectura para el usuario. Esta herramienta permite escribir y ejecutar el código Python en el navegador. `IPython` funciona en varios sistemas operativos, incluidos `Windows`, `Mac OS X`, `Linux` y la mayoría de los sistemas operativos `Unix`. `IPython` brinda todas las características que obtendrá en el intérprete básico con algunas características adicionales como números, más funciones, funciones de ayuda, edición avanzada, etc.

Imbalanced-learn en un mundo ideal, tendríamos conjuntos de datos perfectamente equilibrados y todos entrenaríamos modelos y seríamos felices. Desafortunadamente, el mundo real no es así, y ciertas tareas favorecen datos muy desequilibrados. Por ejemplo, al predecir el fraude en las transacciones de tarjetas de crédito, es de esperar que la gran mayoría de las transacciones (+ 99.9%) sean realmente legítimas. El entrenamiento ingenuo de algoritmos de `ML` conducirá a un rendimiento deprimente, por lo que se necesita cuidado adicional al trabajar con estos tipos de conjuntos de datos. Afortunadamente, este es un problema de investigación estudiado y existe una variedad de técnicas. `Imbalanced-learn` es un paquete de Python que ofrece implementaciones de algunas de esas técnicas, para hacer la vida mucho más fácil. Es compatible con `Scikit-learn` y es parte de los proyectos `Scikit-learn-contrib`.

Caffe2 el marco original de Caffe ha sido ampliamente utilizado durante años, y es conocido por su rendimiento incomparable y base de código probado en batalla. Sin embargo, las tendencias recientes en DL hicieron que el marco se estancara en algunas direcciones. Caffe2 es el intento de llevar Caffe al mundo moderno. Admite formación distribuida, implementación (incluso en plataformas móviles), las CPU más nuevas y Hardware compatible con CUDA. Si bien PyTorch puede ser mejor para la investigación, Caffe2 es adecuado para despliegues a gran escala como se ve en Facebook.

Dash es una biblioteca de código abierto para crear aplicaciones Web, especialmente aquellas que hacen un buen uso de la visualización de datos, en Python puro. esta construido sobre Flask, Plotly.js y React, y proporciona abstracciones que te liberan de tener que aprender esos Frameworks y permitirte ser productivo rápidamente. Las aplicaciones se representan en el navegador y responderán para que se puedan usar en dispositivos móviles. No se requiere JavaScript.

Fire es una biblioteca de código abierto que puede generar automáticamente una CLI para cualquier proyecto de Python. La clave aquí es automática: ¡casi no es necesario escribir ningún código o docstrings para construir una CLI!. Para hacer el trabajo, solo se tiene que llamar a un método Fire y pasarlo como se quiera para convertirlo en una CLI: una función, un objeto, una clase, un diccionario, o incluso no pasar ningún tipo de argumento (lo que convertirá todo el código en una CLI).

Flashtext es una biblioteca para búsqueda y reemplazo de palabras en un documento. La belleza de FlashText es que el tiempo de ejecución es el mismo sin importar cuántos términos de búsqueda se tenga, en contraste con la expresión regular en la que el tiempo de ejecución aumentará casi linealmente con el número de términos.

Pipenv con Pipenv, se especifican todas las dependencias en un Pipfile, que normalmente se genera mediante el uso de comandos para agregar, eliminar o actualizar dependencias. La herramienta puede generar un archivo Pipfile.lock, lo que permite que las compilaciones sean deterministas, ayudando a evitar esos errores difíciles de detectar debido a una dependencia poco clara que ni siquiera se cree que es necesaria.

Luminoth las imágenes están en todas partes hoy en día y comprender su contenido puede ser crítico para varias aplicaciones. Afortunadamente, las técnicas de procesamiento de imágenes han avanzado mucho, impulsadas por los avances en DL. Luminoth es un kit de herramientas de código abierto Python para visión artificial, construido con TensorFlow y Sonnet. Actualmente, viene de fábrica y es compatible con la detección de objetos en forma de un modelo llamado Faster R-CNN.

Instalación de Python e IDEs Para instalar Python 3 en Debian GNU/Linux es necesario hacer:

```
# apt install ipython3 python3 idle3 python3-pip \
python3-matplotlib python3-rpy2 python3-numpy spyder3 \
python3-scipy bpython3 python3-pandas python-sklearn \
python-sklearn-docspe python-wxgtk3.0 jython xonsh \
python3-mpi4py python3-pyqt5 python3-pyqtgraph mypy \
python-wxgtk3.0-dev python3-numba pyflakes3
# apt install flake8 black
```

Para instalar Jupyter (entorno de trabajo orientado a científicos que soporta los lenguajes R y Python):

```
# apt install jupyter-console jupyter-notebook
# pip3 install jupyter
# pip3 install matplotlib
# pip3 install ipywidgets
# jupyter nbextension enable --py --sys-prefix \
widgetsnbextension
```

y podemos instalar PYREPOT usando:

```
# pip install pyreport
```

además podemos instalar editores especializados en Python usando:

```
# apt install eric pyzo pyzo-doc thonny
```

otras opciones se pueden descargar de:

<https://www.jetbrains.com/pycharm/>
<http://www.pydev.org/>
<https://wingware.com/>

También, se pueden instalar diversas herramientas e IDEs para facilitar la programación en Python, para ello usar:

```
# apt install scite jedit kate gedit nedit emacs medit \
kscope geany geany-plugins editra qtcreator anjuta \
anjuta-extras codelite codelite-plugins tea vim-gtk \
mousepad eric neovim neovim-qt medit kwrite katepart
# apt install fte fte-console fte-terminal nano joe vim \
vim-python-jedi vim-tlib vim-latexsuite vim-nox micro \
neovim micro kakoune vim-athena jed
# apt install kdiff3 meld diffuse dirdiff kompare numdiff \
colordiff dwdiff wdiff xxdiff tkdiff ndiff ccdiff xxdiff
# apt install alioth alioth-c2html java2html code2html \
c2html autodia txt2html html2text moreutils
# apt install git git-all gitk gitg git-cola git-gui qgit tig \
vim-fugitive git-extras
# apt install mercurial
# apt install subversion rapidsvn
# apt install cvs tkcvs
```

Compilar y Ejecutar Para compilar el archivo ejemplo.py en *python3* en línea de comandos:

- para compilarlo y ejecutarlo en *python3* en línea de comandos usamos:

```
$ python3 ejemplo.py
```

- en caso de necesitar depurar de forma interactiva un código, podemos usar el módulo *pdb*, por ejemplo:

```
$ pdb ejemplo.py
```

- en caso de necesitar hacer una compilación estática podemos usar:

```
$ pyflakes ejemplo.py
```

Codificar Según Guía de Estilo PEP8 Python cuenta con herramientas que permiten al programador conocer en que discrepa su código de la guía de estilo de Python PEP8 y en caso de que lo requiera, se puede usar estas para que reformatee nuestro código siguiendo al PEP8.

- en caso querer revisar las discrepancias de nuestro código con respecto a la guía oficial de estilo PEP8 de Python usamos:

```
$ flake8 ejemplo.py
```

- en caso de querer conocer qué cambios se sugiere hacer al código según la guía oficial de estilo PEP8 de Python usamos:

```
$ black --check ejemplo.py  
$ black --diff ejemplo.py
```

- en caso de querer reformatear el código según la guía oficial de estilo PEP8 de Python usamos:

```
$ black ejemplo.py  
$ black *.py
```

Anaconda Por otro lado existe **Anaconda**, una Suite de código abierto que abarca una serie de aplicaciones, librerías y conceptos diseñados para el desarrollo de la Ciencia de datos con Python. En líneas generales Anaconda Distribution es una distribución de Python que funciona como un gestor de entorno, un gestor de paquetes y que posee una colección de más de 720 paquetes de código abierto. Anaconda Distribution se agrupa en 4 sectores o soluciones tecnológicas; *Anaconda Navigator*, *Anaconda Project*, *las librerías de Ciencia de Datos y Conda*. Todas estas se instalan de manera automática y en un procedimiento muy sencillo.

Para más información ver: <https://www.anaconda.com/>.

También está **SageMath**, una Suite de código abierto bajo la licencia GPL de Software matemático como: *NumPy*, *SciPy*, *matplotlib*, *Sympy*, *Maxima*, *GAP*, *FLINT*, *R*, entre otros. Además combina acceso a una poderosa combinación del lenguaje basada en Python o directamente vía interfaces o *Wrappers*. La misión del proyecto es crear una alternativa de Software libre a *Magma*, *Maple*, *Mathematica* y *Matlab*.

Para más información ver: <http://www.sagemath.org/>.

Instalación de Aplicaciones Usando Pip Pip es un sistema de administración de paquetes que se utiliza para instalar y administrar paquetes de Software escritos en Python. Pip se usa principalmente para instalar paquetes disponibles en Python Package Index (PyPI, página del proyecto: <https://pypi.org>). Los desarrolladores también pueden usar Pip para instalar módulos y paquetes desarrollados localmente.

Para instalar Pip en Python 2 hacemos:

```
# apt install python-pip
```

y para instalar alguna aplicación para todos los usuarios, por ejemplo *ratarmount*, usamos:

```
# pip2 install ratarmount
```

y para instalar alguna aplicación para el usuario, por ejemplo *ratarmount*, usamos:

```
$ pip2 install --user ratarmount
```

Para instalar Pip en Python 3 hacemos:

```
# apt install python3-venv python3-pip
```

y para instalar alguna aplicación para todos los usuarios, por ejemplo *ratarmount*, usamos:

```
# pip3 install ratarmount
```

y para instalar alguna aplicación para el usuario, por ejemplo *ratarmount*, usamos:

```
$ pip3 install --user ratarmount
```

En caso de instalación para el usuario, para usar la aplicación debemos agregar al PATH:

```
export PATH="$PATH:/home/$USER/.local/bin"
```

Sin pérdida de generalidad (usando pip2 o pip3), podemos ver los detalles de algún paquete, usando:

```
# pip3 show nombre
```

Podemos instalar algún paquete, usando:

```
# pip3 install nombre
```

Podemos actualizar algún paquete, usando:

```
# pip3 install --upgrade nombre
```

Podemos desinstalar algún paquete, usando:

```
# pip3 uninstall nombre
```

Podemos listar los paquetes instalados, usando:

```
# pip3 list nombre
```

Podemos buscar algún paquete, usando:

```
# pip3 search nombre
```

Podemos listar los paquetes desactualizados, usando:

```
# pip3 list --outdated
```

3.3 Julia

es un lenguaje de programación homoicónico, multiplataforma y multiparadigma (véase [26]) -soporta programación orientada a objetos, por procedimientos, funcional y meta además de multietapas- de tipado dinámico de alto nivel y alto desempeño para la computación genérica, técnica y científica, con una sintaxis similar a la de otros entornos de computación similares, con licencia MIT (véase apéndice 10.2).

Dispone de un compilador avanzado (JIT), mecanismos para la ejecución en paralelo y distribuida, además de una extensa biblioteca de funciones matemáticas. La biblioteca, desarrollada fundamentalmente en Julia, también contiene código desarrollado en C o Fortran, para el álgebra lineal, generación de números aleatorios, procesamiento de señales, y procesamiento

de cadenas. Adicionalmente, la comunidad de desarrolladores de Julia contribuye con la creación y distribución de paquetes externos a través del gestor de paquetes integrado de Julia a un paso acelerado. IJulia es el resultado de la colaboración entre las comunidades de IPython y Julia, provee de una poderosa interfaz gráfica basada en el navegador para Julia.

Algunas características básicas son:

- El despacho múltiple: permite definir el comportamiento de las funciones a través de diversas combinaciones de tipos de argumentos
- Sistema de tipado dinámico: tipos para la documentación, la optimización y el despacho de funciones
- Buen desempeño, acercándose al de lenguajes estáticamente compilados como C
- Gestor de paquetes integrado
- Macros tipo Lisp y otras herramientas para la meta-programación
- Llamar funciones de Python: mediante el paquete PyCall
- Llamar funciones de C directamente: sin necesidad de usar envoltorios u APIs especiales
- Poderosas características de línea de comandos para gestionar otros procesos
- Diseñado para la computación paralela y distribuida
- Corutinas: hilos ligeros
- Los tipos definidos por el usuario son tan rápidos y compactos como los tipos estándar integrados.
- Generación automática de código eficiente y especializado para diferentes tipos de argumentos
- Conversiones y promociones para tipos numéricos y de otros tipos, elegantes y extensibles
- Soporte eficiente para Unicode, incluyendo UTF-8 pero sin limitarse solo a este

- Es de uso general, pero tiene todo lo necesario para hacer ciencia de datos, aprendizaje automático, ecuaciones diferenciales, resolvedores de sistemas lineales, etc.

Julia incluye una terminal interactiva, llamada REPL en donde se puede visualizar automáticamente los resultados de la ejecución del programa o segmento de código. Tanto el compilador justo a tiempo (JIT) basado en LLVM así como el diseño del lenguaje le permiten a Julia acercarse e incluso igualar a menudo el desempeño de C. Julia no le impone al usuario ningún estilo de paralelismo en particular. En vez de esto, le provee con bloques de construcción clave para la computación distribuida, logrando hacer lo suficientemente flexible el soporte de varios estilos de paralelismo y permitiendo que los usuarios añadan más.

Instalación de Julia e IDEs Existen diversas versiones de Julia para Linux, para instalar en Debian GNU/Linux es necesario hacer:

```
# apt install julia julia-doc
```

Además, se pueden instalar diversas herramientas e IDEs para facilitar la programación en Julia, para ello usar:

```
# apt install scite jedit kate gedit nedit emacs medit \
kscope geany geany-plugins editra qtcreator anjuta \
anjuta-extras codelite codelite-plugins tea vim-gtk \
mousepad eric neovim neovim-qt medit kwrite katepart
# apt install eclipse eclipse-cdt eclipse-pydev netbeans \
bluefish bluefish-plugins codeblocks codeblocks-contrib
# apt install fte fte-console fte-terminal nano joe vim \
vim-python-jedi vim-tlib vim-latexsuite vim-nox micro \
neovim micro kakoune vim-athena jed
# apt install kdiff3 meld diffuse dirdiff kompare numdiff \
colordiff dwdiff wdiff xxdiff tkdiff ndiff ccdiff xxdiff
# apt install alioth astyle c2html java2html code2html \
c2html autodia txt2html html2text
# apt install git git-all gitk gitg git-cola git-gui qgit tig \
vim-fugitive git-extras
```



```
# apt install mercurial
# apt install subversion rapidsvn
# apt install cvs tkcvs
```

3.4 C y C++

C (véase [14]) es un lenguaje de programación originalmente desarrollado por Dennis Ritchie entre 1969 y 1972 en los Laboratorios Bell, como evolución del lenguaje anterior B, a su vez basado en BCPL. Es un lenguaje orientado a la implementación de Sistemas operativos, concretamente Unix, Linux y el Kernel de Linux. *C* es apreciado por la eficiencia del código que produce y es el lenguaje de programación más popular para crear Software de sistemas, aunque también se utiliza para crear aplicaciones.

Se trata de un lenguaje de tipos de datos estáticos, débilmente tipificado, de medio nivel, ya que dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel. Los compiladores suelen ofrecer extensiones al lenguaje que posibilitan mezclar código en ensamblador con código *C* o acceder directamente a memoria o dispositivos periféricos.

Filosofía Uno de los objetivos de diseño del lenguaje *C* es que sólo sean necesarias unas pocas instrucciones en lenguaje máquina para traducir cada elemento del lenguaje, sin que haga falta un soporte intenso en tiempo de ejecución. Es muy posible escribir *C* a bajo nivel de abstracción; de hecho, *C* se usó como intermediario entre diferentes lenguajes.

En parte, a causa de ser relativamente de bajo nivel y tener un modesto conjunto de características, se pueden desarrollar compiladores de *C* fácilmente. En consecuencia, el lenguaje *C* está disponible en un amplio abanico de plataformas (más que cualquier otro lenguaje). Además, a pesar de su naturaleza de bajo nivel, el lenguaje se desarrolló para incentivar la programación independiente de la máquina. Un programa escrito cumpliendo los estándares e intentando que sea portátil puede compilarse en muchos computadores.

C se desarrolló originalmente (conjuntamente con el sistema operativo Unix, con el que ha estado asociado mucho tiempo) por programadores para programadores. Sin embargo, ha alcanzado una popularidad enorme, y se ha usado en contextos muy alejados de la programación de Software de sistemas, para la que se diseñó originalmente.

Propiedades Núcleo del lenguaje simple, con funcionalidades añadidas importantes, como funciones matemáticas y de manejo de archivos, proporcionadas por bibliotecas. Es un lenguaje muy flexible que permite programar con múltiples estilos. Uno de los más empleados es el estructurado "no llevado al extremo", entre sus principales propiedades destacan:

- Un sistema de tipos que impide operaciones sin sentido
- Usa un lenguaje de preprocesado, el preprocesador de *C*, para tareas como definir macros e incluir múltiples archivos de código fuente
- Acceso a memoria de bajo nivel mediante el uso de punteros
- Interrupciones al procesador con uniones
- Un conjunto reducido de palabras clave
- Por defecto, el paso de parámetros a una función se realiza por valor. El paso por referencia se consigue pasando explícitamente a las funciones las direcciones de memoria de dichos parámetros
- Punteros a funciones y variables estáticas, que permiten una forma rudimentaria de encapsulado y polimorfismo
- Tipos de datos agregados (struct) que permiten que datos relacionados (como un empleado, que tiene un id, un nombre y un salario) se combinen y se manipulen como un todo (en una única variable "empleado")

Carencias Aunque la lista de las características útiles de las que carece *C* es larga, éstos factores han sido importantes para su aceptación, porque escribir rápidamente nuevos compiladores para nuevas plataformas, mantiene lo que realmente hace el programa bajo el control directo del programador, y permite implementar la solución más natural para cada plataforma. Esta es la causa de que a menudo *C* sea más eficiente que otros lenguajes. Típicamente, sólo la programación cuidadosa en lenguaje ensamblador produce un código más rápido, pues da control total sobre la máquina, aunque los avances en los compiladores de *C* y la complejidad creciente de los microprocesadores modernos han reducido gradualmente esta diferencia, Algunas carencias son:

- Recolección de basura nativa, sin embargo se encuentran a tal efecto bibliotecas como la "*libgc*" desarrollada por Sun Microsystems, o el Recolector de basura de Boehm
- Soporte para programación orientada a objetos, aunque la implementación original de *C++* fue un preprocesador que traducía código fuente de *C++* a *C*. Véase también la librería *GObject*
- Funciones anidadas, aunque *GCC* tiene esta característica como extensión
- Soporte nativo para programación multihilo. Disponible usando librerías como *libpthread*

Ventajas estas se pueden resumir en:

- Lenguaje muy eficiente puesto que es posible utilizar sus características de bajo nivel para realizar implementaciones óptimas
- A pesar de su bajo nivel es el lenguaje más portado en existencia, habiendo compiladores para casi todos los sistemas de cómputo conocidos
- Proporciona facilidades para realizar programas modulares y/o utilizar código o bibliotecas existentes

Inconvenientes El mayor problema que presenta el lenguaje *C* frente a los lenguajes de tipo de dato dinámico es la gran diferencia en velocidad de desarrollo: es más lento programar en *C*, sobre todo para el principiante. La razón estriba en que el compilador de *C* se limita a traducir código sin apenas añadir nada. La gestión de la memoria es un ejemplo clásico: en *C* el programador ha de reservar y liberar la memoria explícitamente. En otros lenguajes (como *BASIC*, *MATLAB* o *C#*) la memoria es gestionada de forma transparente para el programador. Esto alivia la carga de trabajo humano y en muchas ocasiones evita errores, aunque también supone mayor carga de trabajo para el procesador.

El mantenimiento en algunos casos puede ser más difícil y costoso que con ciertos lenguajes de más alto nivel. El código en *C* se presta a sentencias cortas y enrevesadas de difícil interpretación.

Cabe destacar el contexto y época en la que fue desarrollado *C*. En aquellos tiempos existían muy pocos programadores, los cuales, a su vez, eran prácticamente todos expertos en el área. De esta manera, se asumía que los programadores eran conscientes de sus trabajos y capaces de manejar perfectamente el lenguaje. Por esta razón es muy importante que los recién iniciados adopten buenas prácticas a la hora de escribir en *C* y manejar la memoria, como por ejemplo un uso intensivo de indentación y conocer a fondo todo lo que implica el manejo de punteros y direcciones de memoria.

Aplicabilidad Hecho principalmente para la fluidez de programación en sistemas UNIX. Se usa también para el desarrollo de otros sistemas operativos como Windows o GNU/Linux. Igualmente para aplicaciones de escritorio como GIMP, cuyo principal lenguaje de programación es *C*.

De la misma forma, es muy usado en aplicaciones científicas (para experimentos informáticos, físicos, químicos, matemáticos, entre otros, conocidos como modelos y simuladores), industriales (industria robótica, cibernética, sistemas de información y base de datos para la industria petrolera y petroquímica). Predominan también todo lo que se refiere a simulación de máquinas de manufactura, simulaciones de vuelo (es la más delicada, ya que se tienen que usar demasiados recursos tanto de Hardware como de Software para desarrollar aplicaciones que permitan simular el vuelo real de una aeronave). Se aplica por tanto, en diversas áreas desconocidas por gran parte de los usuarios noveles.

Los equipos de cómputo de finales de los 90 son varios órdenes de magnitud más potentes que las máquinas en que *C* se desarrolló originalmente. Programas escritos en lenguajes de tipo dinámico y fácil codificación (Ruby, Python, Perl, etc.) que antaño hubieran resultado demasiado lentos, son lo bastante rápidos como para desplazar en uso a *C*. Aun así, se puede seguir encontrando código *C* en grandes desarrollos de animaciones, modelados y escenas en 3D en películas y otras aplicaciones multimedia.

Actualmente, los grandes proyectos de Software se dividen en partes, dentro de un equipo de desarrollo. Aquellas partes que son más "burocráticas" o "de gestión" con los recursos del sistema, se suelen realizar en lenguajes de tipo dinámico o de guión (script), mientras que aquellas partes "críticas", por su necesidad de rapidez de ejecución, se realizan en un lenguaje de tipo compilado, como *C* o *C++*. Si después de hacer la división, las partes críticas no superan un cierto porcentaje del total (aproximadamente el 10%)

entonces todo el desarrollo se realiza con lenguajes dinámicos. Si la parte crítica no llega a cumplir las expectativas del proyecto, se comparan las alternativas de una inversión en nuevo Hardware frente a invertir en el coste de un programador para que reescriba dicha parte crítica.

Ya que muchos programas han sido escritos en el lenguaje *C* existe una gran variedad de bibliotecas disponibles. Muchas bibliotecas son escritas en *C* debido a que *C* genera código objeto rápido; los programadores luego generan interfaces a la biblioteca para que las rutinas puedan ser utilizadas desde lenguajes de mayor nivel, tales como Java, Perl y Python.

C++ (véase [15]) es un lenguaje de programación diseñado a mediados de 1979 por Bjarne Stroustrup. La intención de su creación fue extender al lenguaje de programación *C* mecanismos que permiten la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, *C++* es un lenguaje híbrido.

Posteriormente se añadieron facilidades de programación genérica, que se sumaron a los paradigmas de programación estructurada y programación orientada a objetos. Por esto se suele decir que el *C++* es un lenguaje de programación multiparadigma.

Actualmente existe un estándar, denominado ISO *C++*, al que se han adherido la mayoría de los fabricantes de compiladores más modernos. Existen también algunos intérpretes, tales como ROOT.

Una particularidad de *C++* es la posibilidad de redefinir los operadores, y de poder crear nuevos tipos que se comporten como tipos fundamentales.

El nombre "*C++*" fue propuesto por Rick Mascitti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. Antes se había usado el nombre "*C* con clases". En *C++*, la expresión "*C++*" significa "incremento de *C*" y se refiere a que *C++* es una extensión de *C*.

El concepto de clase Los objetos en *C++* son abstraídos mediante una clase. Según el paradigma de la programación orientada a objetos un objeto consta de:

- Identidad, que lo diferencia de otros objetos (Nombre que llevará la clase a la que pertenece dicho objeto).
- Métodos o funciones miembro

- Atributos o variables miembro

Diferencias de tipos respecto a C En C++, cualquier tipo de datos que sea declarado completo (fully qualified, en inglés) se convierte en un tipo de datos único. Las condiciones para que un tipo de datos T sea declarado completo son a grandes rasgos las siguientes:

- Es posible al momento de compilación conocer el espacio asociado al tipo de datos (es decir, el compilador debe conocer el resultado de `sizeof(T)`)
- T Tiene al menos un constructor, y un destructor, bien declarados
- Si T es un tipo compuesto, o es una clase derivada, o es la especificación de una plantilla, o cualquier combinación de las anteriores, entonces las dos condiciones establecidas previamente deben aplicar para cada tipo de dato constituyente
- En general, esto significa que cualquier tipo de datos definido haciendo uso de las cabeceras completas, es un tipo de datos completo
- En particular, y a diferencia de lo que ocurría en C, los tipos definidos por medio de `struct` o `enum` son tipos completos. Como tales, ahora son sujetos a sobrecarga, conversiones implícitas, etcétera

Los tipos enumerados, entonces, ya no son simplemente alias para tipos enteros, sino que son tipos de datos únicos en C++. El tipo de datos `bool`, igualmente, también pasa a ser un tipo de datos único, mientras que en C funcionaba en algunos casos como un alias para alguna clase de dato de tipo entero.

Compiladores Uno de los compiladores libres de C++ es el de GNU, el compilador G++ (parte del proyecto GCC, que engloba varios compiladores para distintos lenguajes). Otros compiladores comunes son Intel C++ Compiler, el compilador de Xcode, el compilador de Borland C++, el compilador de CodeWarrior C++, el compilador g++ de Cygwin, el compilador g++ de MinGW, el compilador de Visual C++, Carbide.c++, entre otros.

Instalación de C y C++ e IDEs Existen diversas versiones de C y C++ para Linux, para instalarlos en Debian GNU/Linux es necesario hacer:

```
# apt install build-essential manpages-dev glibc-doc \
glibc-doc-reference gcc-doc-base gcc-doc splint \
c++-annotations-pdf g++ jam ohcount ctags \
cppcheck cccc autoconf automake make cmake scons
```

Además, se pueden instalar diversas herramientas e IDEs para facilitar la programación en C y C++, para ello usar:

```
# apt install scite jedit kate gedit nedit emacs medit \
kscope geany geany-plugins editra qtcreator anjuta \
anjuta-extras codelite codelite-plugins tea vim-gtk \
mousepad eric neovim neovim-qt medit kwrite katepart
# apt install eclipse eclipse-cdt eclipse-pydev netbeans \
bluefish bluefish-plugins codeblocks codeblocks-contrib
# apt install fte fte-console fte-terminal nano joe vim \
vim-python-jedi vim-tlib vim-latexsuite vim-nox micro \
neovim micro kakoune vim-athena jed
# apt install kdiff3 meld diffuse dirdiff kompare numdiff \
colordiff dwdiff wdiff xxdiff tkdiff ndiff ccdiff xxdiff
# apt install alioth astyle c2html java2html code2html \
c2html autodia txt2html html2text indent
# apt install git git-all gitk gitg git-cola git-gui qgit tig \
vim-fugitive git-extras
# apt install mercurial
# apt install subversion rapidsvn
# apt install cvs tkcvs
```

Compilar y Ejecutar Para compilar el archivo ejemplo.c en C en línea de comandos usamos:

```
$ gcc ejemplo.c -o ejemplo
```

y lo ejecutamos con:

```
$ ./ejemplo
```

Para compilar el archivo ejemplo.cpp en *C++* en línea de comandos usamos:

```
$ g++ ejemplo.cpp -o ejemplo
```

y lo ejecutamos con:

```
$ ./ejemplo
```

3.5 Fortran

Fortran (véase [?]) contracción del inglés The IBM Mathematical Formula Translating System, es un lenguaje de programación de alto nivel de propósito general, procedimental e imperativo, que está especialmente adaptado al cálculo numérico y a la computación científica. Desarrollado originalmente por IBM en 1957 para el equipo IBM 704, y usado para aplicaciones científicas y de ingeniería, el Fortran vino a dominar esta área de la programación desde el principio y ha estado en uso continuo por más de medio siglo en áreas de cómputo intensivo tales como la predicción numérica del tiempo, análisis de elementos finitos, dinámica de fluidos computacional, física computacional y química computacional. Es uno de los lenguajes más populares en el área de la computación de alto rendimiento y es el lenguaje usado para programas que evalúan el desempeño (benchmark) y el ranking de los supercomputadores más rápidos del mundo.

El Fortran abarca un linaje de versiones, cada una de las cuales evolucionó para añadir extensiones al lenguaje mientras que usualmente retenía compatibilidad con las versiones previas. Versiones sucesivas han añadido soporte para procesamiento de datos basados en caracteres (Fortran 77), programación de arreglos, programación modular y programación orientada a objetos (Fortran 90/95), y programación genérica (Fortran 2003/2008).

Ventajas e inconvenientes de su sintaxis como fue una primera tentativa de creación de un lenguaje de programación de alto nivel, tiene una sintaxis considerada arcaica por muchos programadores que aprenden lenguajes más modernos. Es difícil escribir un bucle "for", y errores en la escritura de un solo carácter pueden llevar a errores durante el tiempo de ejecución en vez de errores de compilación, en el caso de que no se usen las construcciones más frecuentes. Algunas de las primeras versiones no poseían

facilidades que son consideradas muy útiles, tal como la asignación dinámica de memoria.

Se debe tener en cuenta que la sintaxis de Fortran fue orientada para el uso en trabajos numéricos y científicos. Muchas de sus deficiencias han sido abordadas en revisiones recientes del lenguaje. Por ejemplo, en 1990 se presentó un tercer estándar ANSI conocido como FORTRAN 90, que contenía muchas nuevas características y permitía una programación más estructurada. Una serie de cambios menores se presentaron en 1995 conocido como FORTRAN 95 posee comandos mucho más breves para efectuar operaciones matemáticas con matrices y dispone de tipos y soporta OpenMP. Esto no solo mejora mucho la lectura del programa sino que además aporta información útil al compilador. Actualmente se tienen estándares para las versiones 2003 y 2008 y se continúa trabajando en mejoras al lenguaje.

Por estas razones Fortran prácticamente no se usa fuera de los campos científicos y del análisis numérico, pero permanece como el lenguaje preferido para desarrollar aplicaciones de computación numérica de alto rendimiento.

Características

Tipos de datos soportados:

- Numéricos (enteros, reales, complejos y de doble precisión).
- Boleanos (logical).
- Arreglos.
- Cadenas de caracteres.
- Archivos.

FORTRAN 90/95 ya es estructurado, y no requiere sentencias GOTO. Sólo admite dos ámbitos para las variables: local y global.

Variables y constantes

- Fortran no es sensible a mayúsculas y minúsculas. Los nombres de variables tienen de 6 a 31 caracteres máximo y deben comenzar por una letra. Los blancos son significativos.

- Declaración explícita de variables.
- Enteras (I-N), el resto reales. (se modifica con IMPLICIT).
- Punteros: en los primeros FORTRAN no hay punteros y todas las variables se almacenan en memoria estática. En FORTRAN 90 se declaran INTEGER, POINTER::P.
- Para memoria dinámica ALLOCATE y DEALLOCATE

Tipos de datos

- Arrays, pueden tener hasta 7 dimensiones y se guardan por columnas.
- REAL M(20),N(-5:5)
- DIMENSION I(20,20) (tipo por nomenclatura implícita)
- Cadenas de caracteres, el primer carácter es el 1, el operador // permite concatenar cadenas.
- CHARACTER S*10, T*25
- Almacenamiento de datos. Se usa COMMON para datos compartidos y EQUIVALENCE cuando almacenamos una variable con dos posibles tipos en la misma posición de memoria (como union en C). Se usa DATA para inicializar datos estáticos.
- DATA X/1.0/,Y/3.1416/,K/20/
- Tipos definidos por el usuario, con TYPE <nombre>... END TYPE <nombre>

Control de secuencia el conjunto de estructuras de control es limitado:

- Expresiones, prioridad de operadores
- Enunciados

- Asignación, cuando se hace entre cadenas hay ajuste de tamaño con blancos o truncamiento.
- Condicional. Permite IF ELSE IF... Para selección múltiple
SELECT CASE CASE.....CASE DEFAULT.... END SELECT
- Iteración. DO....END DO
- Nulo, se usa solo para la etiqueta. CONTINUE.
- Control de subprogramas. CALL invoca al subprograma y RETURN devuelve un valor al programa llamante.
- Construcciones propensas a error: GOTO.

Entrada y salida

- Tipos de archivos:
 - Secuenciales
 - De acceso directo
- Comandos: READ, WRITE, PRINT, OPEN , CLASE, INQUIRE (propiedades o estado del archivo) REWIND y ENDFILE (para ubicar el puntero del fichero).
- Para el tratamiento de excepciones en las sentencias READ/WRITE se puede introducir la posición de la rutina de dicho tratamiento (ERR=90).

Subprogramas

- Hay tres tipos de subprogramas:
 - Function, devuelven un solo valor de tipo numérico, lógico o cadena de caracteres.
 - Subroutine, devuelve valores a través de variables no locales COMMON.
 - Función de enunciado, permite calcular una sola expresión aritmética o lógica.
- $FN(X,Y)=\sin(X)^2-\cos(Y)^2$
- Gestión de almacenamiento.
 - Las variables son locales o globales (COMMON)
 - Recursividad: RECURSIVE FUNCTION FACTORIAL(X)
 - Parámetros de subprograma. Paso por referencia.

Abstracción y encapsulación. Evaluación del lenguaje

- La abstracción es posible mediante los subprogramas y el uso de variables COMMON, aunque su uso es propenso a errores.
- FORTRAN sigue siendo utilizado en el ámbito científico y es muy eficiente realizando cálculos.
 - La estructura del programa suele ser difícil de entender.
 - En FORTRAN 90/95 se incluye la recursividad y la memoria dinámica.
 - Las etiquetas de las sentencias ya no son necesarias, ni el GOTO, pues se ha transformado en un lenguaje estructurado.
 - El aspecto de los programas sigue siendo de procesamiento por lotes

Instalación de Fortran e IDEs Existen diversas versiones de Fortran para Linux, para instalarlos en Debian GNU/Linux es necesario hacer:

```
# apt install gfortran gfortran-doc fortran77-compiler \
fortran95-compiler fortran-compiler cfortran ftc fort77
```

Además, se pueden instalar diversas herramientas e IDEs para facilitar la programación en Fortran, para ello usar:

```
# apt install scite jedit kate gedit nedit emacs medit \
kscope geany geany-plugins editra qtcreator anjuta \
anjuta-extras codelite codelite-plugins tea vim-gtk \
mousepad eric neovim neovim-qt medit kwrite katepart
# apt install fte fte-console fte-terminal nano joe vim \
vim-python-jedi vim-tlib vim-latexsuite vim-nox micro \
neovim micro kakoune vim-athena jed
# apt install kdiff3 meld diffuse dirdiff kompare numdiff \
colordiff dwdiff wdiff xxdiff tkdiff ndiff ccdiff xxdiff
# apt install alioth astyle c2html java2html code2html \
c2html autodia txt2html html2text
# apt install git git-all gitk gitg git-cola git-gui qgit tig \
vim-fugitive git-extras
```

```
# apt install mercurial
# apt install subversion rapidsvn
# apt install cvs tkcvs
```

Compilar y Ejecutar Para compilar el archivo ejemplo.f77 en *Fortran 77* en línea de comandos usamos:

```
$ g77 ejemplo.f77 -o ejemplo
```

y lo ejecutamos con:

```
$ ./ejemplo
```

Para compilar el archivo ejemplo.f90 en *Fortran 90/95/2003* en línea de comandos usamos:

```
$ gfortran ejemplo.f90 -o ejemplo
```

y lo ejecutamos con:

```
$ ./ejemplo
```

3.6 R

El paquete R (véase [34]) es un lenguaje y entorno de programación para análisis estadístico y gráfico. Se trata de un proyecto de Software libre, resultado de la implementación GNU del premiado lenguaje S. SPSS, R y S-Plus —versión comercial de S— son, probablemente, los tres lenguajes más utilizados en investigación por la comunidad estadística, siendo además muy populares en el campo de la investigación biomédica, la bioinformática y las matemáticas financieras. A esto contribuye la posibilidad de cargar diferentes bibliotecas o paquetes con finalidades específicas de cálculo o gráfico.

Además, R puede integrarse con distintas bases de datos y existen bibliotecas que facilitan su utilización desde lenguajes de programación interpretados como Perl y Python. R soporta hacer interfase con lenguajes de programación como C, C++ y Fortran.

Otra de las características de R es su capacidad gráfica, que permite generar gráficos con alta calidad. R posee su propio formato para la documentación basado en LaTeX (véase [23]). R también puede usarse como

herramienta de cálculo numérico, campo en el que puede ser tan eficaz como otras herramientas específicas tales como FreeMat, GNU Octave y su equivalente comercial, MATLAB. Se ha desarrollado una interfaz RWeka para interactuar con Weka (véase [49]) que permite leer y escribir ficheros en el formato arff y enriquecer R con los algoritmos de minería de datos de dicha plataforma.

Los ambientes de desarrollo integrado para R existen como proyectos externos, como pueden ser editores —que sólo soportan la sintaxis—, los IDEs (Integrate Development Environments) y los GUI (Graphical User Interfaces) —permiten editar, ejecutar y depurar código desarrollado para R—. Hay más de 20 proyectos activos, tres de los más conocidos son Tinn-R (véase [50]), RStudio (véase [51]) y Rkward de KDE.

Instalación de R Existen diversos paquetes para *R* en Debian GNU/Linux, para instalarlos es necesario hacer:

```
# apt install r-base
# apt install 'r-cran*'
# apt install 'r-bioc'
```

Editores nativo en Debian GNU/Linux

```
# apt install rkward rkward-data
```

y podemos generar gráficos de datos usando:

```
# apt install rlplot
```

para conocer todos los paquetes asociados, usar:

```
$ apt search ^r-cran
$ apt search ^r-bioc
```

Paquetes internos

```
R2WinBugs mcmcplots devtools Rattle e1071 FactiMineR \
Ellipse xlsx hsaur shiny
```

3.7 Herramientas de Programación

En Linux existe una gran variedad de herramientas para programación, ya que este sistema operativo fue hecho por programadores y para programadores, por ello entre las miles de herramientas, tenemos algunas que son ampliamente usadas, entre las que destacan:

Editores de Terminal

- Diakonos
- Jet
- Joe
- LE
- Mined
- Nano
- Nice Editor (NE)
- Pico
- Setedit
- Vim
- Fte

Editores Sencillos con Interfaz Gráfica

- Gedit
- SciTE
- JEdit
- NEdit
- MEdit

- KScope
- Editra
- Kate
- KWrite
- Leafpad
- Mousepad
- Anjuta
- TEA
- Pluma
- GVim
- Emacs

Editores Avanzados con Interfaz Gráfica

- Atom
- Bluefish
- BlueGriffon
- Brackets
- Geany
- Glade
- Google Web Designer
- KompoZer
- Light Table
- Notepadqq
- Scribes
- Sublime Text

Entornos de Programación Integrado (IDEs)

- Aptana
- Arduino IDE
- Android Studio
- CodeLite
- Code::Blocks
- Eclipse
- Gambas
- JetBrains Suite
- NetBeans
- Ninja-IDE
- Python IDLE
- PyDev
- Postman
- Qt Creator
- Simply Fortran
- Visual Studio Code
- Wing Python IDE
- Spyder
- PyCharm
- Jupyter
- Eric

Kit de Desarrollo de Software

- .Net Core SDK
- Android SDK
- Java JDK

Comparadores de texto y fuentes

- KDiff3
- Meld
- Diffuse
- DirDiff
- kompare
- Numdiff
- colordiff
- wdiff
- xxdiff
- tkdiff
- Ndiff

Otras Herramientas

- Alleyoop
- C2HTML
- Java2HTML
- Code2HTML
- c2html

- AutoDia
- txt2html
- html2text

Programas para control de versiones que permite desarrollo colaborativo de Software:

- Git <https://git-scm.com/>
- Mercurial <https://www.mercurial-scm.org/>
- Subversion <https://subversion.apache.org/>
- Perforce
- Bazaar
- CVS
- LibreSource
- Monotone
- SmartGit
- GitKraken
- Git Cola

Generadores automaticos de documentación que generan salida en PDF, HTML y XML para lenguajes como C++ y Java:

- Doxygen <http://www.doxygen.org/>
- JavaDoc

Formateador de código fuente para C, C++, Java y C#

- Astyle <http://astyle.sourceforge.net>

Lenguaje Unificado de Modelado (Unified Modeling Language)²⁷ forja un lenguaje de modelado visual común semántica y sintácticamente rico para la arquitectura, el diseño y la implementación de sistemas de Software complejos tanto en estructura como en comportamiento:

- UML <https://www.uml.org/>

Depuradores de programas

- ddd <https://www.gnu.org/Software/ddd/>
- gdb <https://www.gnu.org/Software/gdb/>
- kdbg <http://www.kdbg.org/>

Programas para rastrear errores en la manipulación de memoria y punteros desbordados

- Valgrind <http://valgrind.org/>
- DUMA <http://duma.sourceforge.net/>

Programas para hacer análisis de rendimiento²⁸

- gprof <https://sourceware.org/binutils/docs/gprof/>
- Callgrind <http://valgrind.org/docs/manual/cl-manual.html>
- kCachegrind <http://kCachegrind.sourceforge.net/html/Home.html>
- time <https://www.cyberciti.biz/faq/unix-Linux-time-command-examples-usage-syntax/>

²⁷Otras opciones son: UML Diagram Generation, Code Generation, Document Generation and Reporting, Scaling, Database Schema Generation, Entity Relationship Diagrams, Data Flow Diagrams, StarUML BOUML, EclipseUML, UML Modeller, Papyrus, Nclass, PlantUML, UMLet, NetBeansIDE, Open ModelSphere, gModeler, RISE, Oracle jdeveloper, Oracle SQL Developer, Dia, Kivio, ArgoUML, Xfig, etc.

²⁸Otras opciones son: Splint, cppcheck, Rough Auditing Tool for Security, C y C++ Code Counter, CppNcss, Gnocchi, CUnit, CppUnit, OProfile, Intel VTune, Nemiver, Mudflap, etc.

En este apartado, solo tocaremos las más usadas, pero abunda la documentación de estas y otras importantes herramientas en línea de comandos. Iniciaremos por las de compilar²⁹ y depurar³⁰ programas compilables en C, C++, Fortran, entre otros.

3.7.1 ¿Qué es eso de ASCII, ISO-8859-1 y UTF-8?

Los tres estándares representan el esfuerzo informático por brindar un sistema de codificación que permita representar los caracteres que se usan en todos los idiomas. El primer esfuerzo lo hizo *ASCII* y fue para el idioma inglés (128 caracteres), luego ante su insuficiencia para representar otros caracteres como los latinos por ejemplo, nace *ISO-8859-1* (también llamado *LATIN-1* ó *ASCII* extendido) pero debido a que no podía representar caracteres de otros idiomas aparece el estándar Unicode (del cual es parte *UTF-8*).

Un buen detalle a saber es que mientras *ISO-8859-1* usa un byte para representar un carácter, no pasa lo mismo con *UTF-8* que puede usar hasta 4 bytes ya que es de longitud variable. Esto hace que una base de datos en *UTF-8* sea un poco más grande que una en *ISO-8859-1*. Esto sucede porque -por ejemplo- mientras *ISO-8859-1* usa un byte para representar la letra ñ, *UTF-8* usa dos bytes. Hay un tema más y es que muchas veces cuando vamos a migrar información nos encontramos con caracteres *ISO-8859-1* (los correspondientes a los números 147, 148, 149, 150, 151 y 133) que no pueden verse en un editor *UNIX/LINUX* pero si en un navegador *HTML*.

Unicode es un set de caracteres universal, es decir, un estándar en el que se definen todos los caracteres necesarios para la escritura de la mayoría de los idiomas hablados en la actualidad que se usan en la computadora. Su objetivo es ser, y, en gran medida, ya lo ha logrado, un superconjunto de

²⁹Un compilador es un programa informático que traduce un programa que ha sido escrito en un lenguaje de programación a un lenguaje común, usualmente lenguaje de máquina, aunque también puede ser traducido a un código intermedio (bytecode) o a texto y que reúne diversos elementos o fragmentos en una misma unidad, este proceso de traducción se conoce como compilación.

³⁰Un depurador (en inglés, debugger), es un programa usado para probar y depurar (eliminar) los errores del programa "objetivo". El código a ser examinado puede alternativamente estar corriendo en un simulador de conjunto de instrucciones (ISS), una técnica que permite gran potencia en su capacidad de detenerse cuando son encontradas condiciones específicas pero será típicamente más lento que ejecutando el código directamente en el apropiado (o el mismo) procesador.

todos los sets de caracteres que se hayan codificado. El texto que aparece en la computadora o en la Web se compone de caracteres. Los caracteres representan letras del abecedario, signos de puntuación y otros símbolos.

En el pasado, distintas organizaciones han recopilado diferentes sets de caracteres y han creado codificaciones específicas para ellos. Un set puede abarcar tan sólo los idiomas de Europa occidental con base en el latín (sin incluir países de la Unión Europea como Bulgaria o Grecia), otro set puede contemplar un idioma específico del Lejano Oriente (como el japonés), y otros pueden ser parte de distintos sets diseñados especialmente para representar otro idioma de algún lugar del mundo.

Lamentablemente, no es posible garantizar que su aplicación particular pueda soportar todas las codificaciones, ni que una determinada codificación pueda soportar todos sus requerimientos para la representación de un cierto idioma. Además, generalmente resulta imposible combinar distintas codificaciones en la misma página Web o en una base de datos, por lo que siempre es muy difícil soportar páginas plurilingües si se aplican enfoques "antiguos" cuando se trata de tareas de codificación.

El Consorcio Unicode proporciona un único y extenso set de caracteres que pretende incluir todos los caracteres necesarios para cualquier sistema de escritura del mundo, incluyendo sistemas ancestrales (como el cuneiforme, el gótico y los jeroglíficos egipcios). Hoy resulta fundamental para la arquitectura de la Web y de los sistemas operativos, y las principales aplicaciones y navegadores Web incluyen soporte para este elemento. En el Estándar Unicode también se describen las propiedades y algoritmos necesarios para trabajar con caracteres. Este enfoque facilita mucho el trabajo con sistemas o páginas plurilingües y responde mucho mejor a las necesidades del usuario que la mayoría de los sistemas de codificación tradicionales.

Sets de caracteres, sets de caracteres codificados y codificaciones
un set de caracteres o repertorio comprende el grupo de caracteres que se utilizarían para una finalidad específica, ya sea los necesarios para el soporte de los idiomas de Europa Occidental en la computadora.

Un set de caracteres codificados es un grupo de caracteres en el que se ha asignado un número exclusivo a cada carácter. Las unidades de un set de caracteres codificados se conocen como puntos de código. El valor de un punto de código representa la ubicación de un carácter en el set de caracteres codificados. Por ejemplo, el punto de código para la letra á en el set de

caracteres codificados Unicode es *225* en notación decimal, o *E1* en notación hexadecimal.

La codificación de caracteres refleja la manera en la que el set de caracteres codificados se convierte a bytes para su procesamiento en la computadora. Además, en Unicode existen distintas formas de codificar el mismo carácter. Por ejemplo, la letra *á* se puede representar mediante dos bytes en una codificación y con cuatro bytes, en otra. Los formatos de codificación que se pueden usar con Unicode se denominan *UTF-8*, *UTF-16* y *UTF-32*.

Por todo lo anterior, al programar es necesario tener en cuenta la codificación usada por el editor o IDE que se use para ello y que no todos los editores soportan las mismas codificaciones³¹, además puede haber problemas de portabilidad en los archivos entre distintos sistemas operativos. En el código fuente (las instrucciones del programa) no se suele usar caracteres distintos al *ASCII*, pero en las cadenas de visualización o en la documentación es común el uso de caracteres acentuados, es aquí donde hay que tomar una decisión sobre el usar o no dichos caracteres, siempre y cuando el compilador los soporte.

Si siempre se usa el mismo editor y la misma plataforma de desarrollo, no hay razón para no usar caracteres extendidos como los acentos. Pero si se usarán múltiples sistemas operativos y no hay garantía de usar editores que soporten dichos caracteres, entonces existe la posibilidad de perder dichos caracteres o bien pueden generar errores al compilar los archivos por no ser soportados. Por ello una opción para evitar problemas es sólo usar caracteres *ASCII* o tener el cuidado de usar editores que no pierdan dichos caracteres.

En Linux, para verificar la codificación de un archivo se utiliza el comando *file -i* o *-mime*, este comando permite mostrar en pantalla el tipo de archivo y su codificación, usando:

```
$ file -i Car.java
```

El comando *iconv* es utilizado para realizar esta tarea de convertir el código de un texto a otro. La lógica para aplicar correctamente el commando *iconv* es la siguiente:

```
$ iconv options -f from-encoding -t to-encoding inputfile(s) -o  
outputfile
```

³¹Dado que los archivos fuente se intercambian entre usuarios y es común el uso de diferentes sistemas operativos, la conversiones de los caracteres entre diferentes formatos puede ser causa de problemas de codificación, perdiéndose dichos caracteres en la conversión.

así, *-f* o *-from-code* significa la entrada de la codificación, y *-t* o *-to-encoding* especifica la salida de la misma. Para enumerar todos los juegos de caracteres podemos usar *-l* o *-list*:

```
$ iconv -l
```

Con todo esto en mente podemos proceder a explicar la codificación de *UTF-8* a *ASCII*. Primero hay que comenzar con conocer las codificaciones de los caracteres en el archivo y luego poder ver el contenido del mismo. Así, se podrán convertir todos los archivos a la codificación *ASCII*. Todo después de haber utilizado el comando *iconv*, para poder verificar lo que contiene la salida del archivo. Para eso hay que hacer lo siguiente:

```
$ file -i input.file
$ cat input.file
$ iconv -f ISO-8859-1 -t UTF-8//TRANSLIT input.file -o
out.file
$ cat out.file
$ file -i out.file
```

Cabe destacar que, si el comando *//IGNORE* se añade a *to-encoding*, los caracteres no pueden ser convertidos y un error se mostrará luego de la conversión. También, si el comando *//TRANSLIT* es añadido a *to-encoding* como en el ejemplo dado (*ASCII//TRANSLIT*), los caracteres convertidos son transliterados, si es posible, como necesarios.

Esto implicaría que en este evento los caracteres no pueden ser representados como se desea, aunque pueden haber aproximaciones del mismo, inclusive dos. Por lo que, si un carácter no puede ser transliterado, no será reconocido como un objetivo para reemplazo y se mostrará la marca (?) en la salida del archivo.

Algunas veces es necesario convertir el archivo de *UTF-8* a *ASCII* y lo hacemos mediante:

```
$ iconv -f UTF-8 -t ISO-8859-1 prog.c -o progMod.c
```

o mediante:

```
$ iconv -f UTF-8 -t ASCII//TRANSLIT prog.c -o progMod.c
```


3.7.2 Uso de Espacios o Tabuladores en Fuentes

En muchos lenguajes de programación la indentación es una forma opcional de hacer legible el código para el programador, en otros (como Python) es imprescindible (ya que de ella dependerá su estructura).

¿Qué es la indentación? En un lenguaje informático, la indentación es lo que a la sangría al lenguaje humano escrito (a nivel formal). Así como para el lenguaje formal, cuando uno redacta una carta, debe respetar ciertas sangrías, los lenguajes informáticos, pueden requerir una indentación.

No todos los lenguajes de programación necesitan una indentación, aunque sí se estila implementarla, a fin de otorgar mayor legibilidad al código fuente. Una indentación depende del lenguaje y estilo de codificación usado, por ejemplo en C, C++ y Java se acostumbra usar tres espacios en blanco. En el caso de Python se suele usar 4 espacios en blanco o un tabulador, que indicará que las instrucciones indentadas forman parte de una misma estructura de control.

Los programadores siempre han debatido entre el uso de espacios y tabulaciones para estructurar su código. Los espacios y las tabulaciones son utilizados por los programadores para estructurar el código de una forma determinada. La primera línea de código (sin espacio o tabulación) inicia un “bloque” de contenido. Si las sucesivas líneas de código forman parte de ese mismo bloque (encerrado entre corchetes) o forman nuevos subbloques, estas se van desplazando hacia la derecha para indicar esa subordinación. En caso de formar un bloque completamente nuevo, se mantiene en la misma posición que la línea inmediatamente anterior.

A nivel funcional, la diferencia entre el uso de espacios o tabulaciones es nula. Cuando el código pasa por el compilador antes de ser ejecutado, la máquina interpreta de igual forma ambos formatos. No obstante, sí existen diferencias técnicas que marcan la diferencia entre el uso de tabulaciones y espacios:

- **Precisión.** Una tabulación no es más que un conjunto de espacios agrupados. Por norma general, este conjunto suele ser de 8 caracteres, pero puede variar. ¿Qué quiere decir esto? Que cuando un mismo fichero de código se abre en dos máquinas diferentes, la apariencia del código puede ser diferente. En cambio, el uso de espacios no conlleva este problema: un espacio siempre ocupa el mismo “espacio” —valga la

redundancia— y asegura que el código se visualiza de la misma forma en todas las máquinas.

- Comodidad. En el caso de las tabulaciones, basta con pulsar la tecla de tabulación una única vez para estructurar correctamente el código. En el caso de los espacios, es necesario pulsar varias veces la misma tecla para lograr la estructura deseada.
- Almacenamiento. El uso de tabulaciones también reduce el tamaño del fichero final, mientras que el uso de espacios lo aumenta. Lo mismo sucedería con el uso de espacios en lugar de saltos de línea.

Entonces, ¿cuál es la más correcta? La realidad es que todo depende de las preferencias personales. Si necesitas optimizar el tamaño de los ficheros al máximo, el uso de espacios se convierte en un sacrilegio. Si, en cambio, tu código debe lucir exactamente igual en múltiples máquinas, el uso de espacios puede ser más conveniente para lograr esa homogeneidad.

Por suerte, existen múltiples editores en la actualidad que trabajan y facilitan la transición entre ambos sistemas. Asimismo, los equipos de desarrollo de software establecen en sus guidelines el uso de espacios o tabulaciones. De esta forma, se evitan conflictos entre los programadores de un mismo proyecto y se alcanza esa homogeneidad tan deseada.

El comando *expand* y *unexpand* (que vienen instalados en los paquetes *GNU Core*) permite convertir tabuladores en espacios y viceversa, según nuestras necesidades o gustos. Estos comandos sacan el resultado de stdin o de los archivos nombrados en la línea de comando. Utilizando la opción *-t* se pueden establecer una o más posiciones de tabulador.

Para ver si se usan espacios o tabuladores en un archivo fuente podemos usar el comando *cat* con la opción *-T* que nos mostrará los caracteres tabulador como $\backslash$ T, ejemplo:

```
$ cat -T archivo
```

Para convertir los espacios en tabuladores (un tabulador igual a 8 espacios) usamos:

```
$ unexpand progEsp.c
```

o redireccionando la salida usando:

```
$ unexpand progEsp.c > progTab.c
```

Para convertir los tabuladores en espacios (1 tabulador por ejemplo 4 caracteres) usamos:

```
$ expand -t 4 progTab.c
```

o redireccionando la salida usando:

```
$ expand -t 4 progTab.c > progEsp.c
```

También es posible buscar todos los archivos (digamos *.cpp) y cambiar los tabuladores por 4 espacios (para ello usamos el comando *sponge* que está contenido en el paquete *moreutils*), mediante:

```
$ find . -name '*.cpp' -type f -exec bash -c \
'expand -t 4 "$0" | sponge "$0"' {} \;
```

3.7.3 Comparar Contenido de Fuentes

Cuando se programa es común generar distintas versiones de un mismo archivo, en GNU/Linux se tiene varias herramientas para comparar y combinar cambios. En la línea de comandos el comando *diff* permite ver los cambios entre dos versiones de un archivo y el comando *merge* sirve para combinar cambios. Por otro lado *sdiff* nos permite ver las diferencias entre dos archivos y de forma interactiva combinar cambios.

Pese a que son poderosos estos comandos, en forma gráfica se puede obtener todo su potencial. Algunas de estas opciones son:

```
# apt install kdiff3 meld diffuse dirdiff kompare wdiff \
numdiff colordiff xxdiff tkdiff ndiff
```

Estos permiten comparar dos o tres versiones de un archivo simultáneamente, y hacerlo con el contenido de una o más carpetas. Cada uno tiene la capacidad de mostrar los cambios y si se requiere hacer la combinación de ellos.

meld nos muestra gráficamente las diferencias entre dos archivos o también, entre todos los archivos de dos directorios utilizando distintos colores, y nos permite editar estos archivos desde el propio programa, actualizando dinámicamente las diferencias. El programa incluye filtros y distintas ayudas para hacer la edición más sencilla, como flechas al lado de los cambios para aplicar cambio en cualquiera de los archivos con un simple clic. Este programa se puede utilizar como un sencillo cliente de control de cambios para Git, CVS, Subversion, etc.

kdiff3 nos muestra gráficamente las diferencias entre tres archivos utilizando distintos colores, y nos permite editar estos archivos desde el propio programa, actualizando dinámicamente las diferencias. El programa incluye filtros y distintas ayudas para hacer la edición más sencilla, como flechas al lado de los cambios para aplicar cambio en cualquiera de los archivos con un simple clic.

3.7.4 Astyle

Para dar uniformidad a la codificación de los programas fuente, se puede usar un formateador automático de código, *Astyle* soporta una gran variedad de lenguajes y opciones, para instalar en Debian GNU/Linux usar:

```
# apt install astyle
```

para formatear los archivos de C, C++, C# usar:

```
$ astyle -s3 -p -style=allman -lineend=Linux *.?pp
```

para Java, una opción es

```
$ astyle -s3 -p -style=java -lineend=Linux *.java
```

Algunos estilos disponibles son:

```
style=allman style=java style=kr style=stroustrup  
style=whitesmith style=vtk style=ratliff style=gnu  
style=Linux style=horstmann style=1tbs style=google  
style=mozilla style=pico style=lisp
```

más opciones en:

- <http://astyle.sourceforge.net/astyle.html>
- https://en.wikipedia.org/wiki/Programming_style
- https://en.wikipedia.org/wiki/Indent_style

3.7.5 Compilación y la Optimización del Ejecutable

Al programar es necesario revisar nuestro código por un compilador y los errores son inherentes al proceso de programación. Los errores de programación responden a diferentes tipos y pueden clasificarse dependiendo de la fase en que se presenten. Algunos tipos de errores son más difíciles de detectar y reparar que otros, veamos entonces:

- Errores de sintaxis
- Advertencias
- Errores de enlazado
- Errores de ejecución
- Errores de diseño

Errores de sintaxis son errores en el código fuente. Pueden deberse a palabras reservadas mal escritas, expresiones erróneas o incompletas, variables que no han sido declaradas, etc. Los errores de sintaxis se detectan en la fase de compilación. El compilador, además de generar el código objeto, nos dará una lista de errores de sintaxis. De hecho nos dará sólo una cosa o la otra, ya que si hay errores no es posible generar un código objeto.

Advertencias además de errores, el compilador puede dar también advertencias (Warnings). Las advertencias son errores, pero no lo suficientemente graves como para impedir la generación del código objeto. No obstante, es importante corregir estos errores la mayoría de las veces, ya que ante un aviso el compilador tiene que tomar decisiones, y estas no tienen porqué coincidir con lo que nosotros pretendemos hacer, ya se basan en las directivas que los creadores del compilador decidieron durante la creación del compilador. Por lo tanto, en ocasiones, ignorar las advertencias puede ocasionar que nuestro programa arroje resultados inesperados o erróneos.

Errores de enlazado el programa enlazador también puede encontrar errores. Normalmente se refieren a funciones que no están definidas en ninguno de los ficheros objetos ni en las bibliotecas. Puede que hayamos olvidado incluir alguna biblioteca, o algún fichero objeto, o puede que hayamos olvidado definir alguna función o variable, o lo hayamos hecho mal.

Errores de ejecución incluso después de obtener un fichero ejecutable, es posible que se produzcan errores durante la ejecución del código. En el caso de los errores de ejecución normalmente no obtendremos mensajes específicos de error o incluso puede que no obtengamos ningún error, sino que simplemente el programa terminará inesperadamente. Estos errores son más difíciles de detectar y corregir (pues se trata de la lógica como tal de nuestra aplicación). Existen herramientas auxiliares para buscar estos errores, son los llamados depuradores (Debuggers). Estos programas permiten detener la ejecución de nuestros programas, inspeccionar variables y ejecutar nuestro programa paso a paso (instrucción a instrucción). Esto resulta útil para detectar excepciones, errores sutiles, y fallos que se presentan dependiendo de circunstancias distintas. Generalmente los errores en tiempo de ejecución se dan por situaciones no consideradas en la aplicación, por ejemplo, que el usuario ingrese una letra en vez de un número y ésto no es controlado.

Errores de diseño finalmente los errores más difíciles de corregir y prevenir. Si nos hemos equivocado al diseñar nuestro algoritmo, no habrá ningún programa que nos pueda ayudar a corregirlos, pues es imposible que un programa pueda determinar qué es lo que tratamos de conseguir o un programa que realice aplicaciones cualquiera por nosotros. Contra estos errores sólo cabe practicar y pensar, realizar pruebas de escritorio, hacerle seguimiento y depuración a la aplicación hasta dar con el problema (una mala asignación, un valor inesperado, olvidar actualizar una variable, etc.), también es útil buscar un poco de ayuda de libros o en sitios y foros especializados.

Compilación y la Optimización del Ejecutable Para usar muchas de estas herramientas (en línea de comandos), primero debemos conocer cómo compilar fuentes³², sin pérdida de generalidad trabajaremos en C++ solici-

³²Compilador para C es gcc, para C++ es g++, para Fortran es f77 o f95, etc.

tando que el archivo ejecutable³³ tenga el nombre *ejemp*:

```
$ g++ *.cpp -o ejemp
```

Para ejecutar el programa ya compilado:

```
$ ./ejemp
```

Para compilar y ver todos los avisos usar:

```
$ g++ -pedantic -Wall -Wextra -O *.cpp
```

o de forma alternativa:

```
$ g++ -Weffc++ *.cpp
```

Por otro lado, también podemos hacer una revisión estática del código, por ejemplo en C++ usamos:

```
$ cppcheck --enable=all *.?pp
```

mostrará los avisos de análisis estático del código indicado.

Para conocer el tiempo de ejecución³⁴ de un programa, podemos usar el comando básico *time*, mediante:

```
$ time ejecutable
```

que entregará información del tipo:

```
$ time ls  
  
real    0m0.004s  
user    0m0.001s  
sys     0m0.004s
```

³³Un archivo ejecutable es tradicionalmente un archivo binario con instrucciones en código de máquina cuyo contenido se interpreta por el ordenador como un programa. Además, suele contener llamadas a funciones específicas de un sistema operativo.

³⁴El tiempo total de ejecución de un programa (tiempo real) es la suma del tiempo de ejecución del programa del usuario (tiempo de usuario) más el tiempo de ejecución del sistema necesario para soportar la ejecución (tiempo de sistema).

Pero podemos instalar una versión optimizada de este comando que proporciona información adicional, para ello instalar:

```
# apt install time
```

y su ejecución mediante:

```
$ /usr/bin/time ejecutable
```

por ejemplo para el comando *ls*, entrega una salida del tipo:

```
$ /usr/bin/time -v ls
Command being timed: "ls"
User time (seconds): 0.00
System time (seconds): 0.00
Percent of CPU this job got: 66%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.00
Average shared text size (kBytes): 0
Average unshared data size (kBytes): 0
Average stack size (kBytes): 0
Average total size (kBytes): 0
Maximum resident set size (kBytes): 2360
Average resident set size (kBytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 110
Voluntary context switches: 1
Involuntary context switches: 1
Swaps: 0
File system inputs: 0
File system outputs: 0
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (Bytes): 4096
Exit status: 0
```


Además, podemos compilar usando diversas optimizaciones³⁵ disponibles en todos los compiladores GNU de Linux, estas opciones de optimización están ordenadas, desde no optimizar, a la mejor optimización posible, estas son: -O0, O1, -O2, -O3, -O3 -march=native, -O3 -march=native -fito, -Ofast -march=native.

Para compilar y solicitar la optimización usamos:

```
$ g++ -O1 *.cpp
```

y para ejecutar el programa usamos:

```
$ ./a.out
```

El resultado de las optimizaciones dependen del programa y se puede ver que el rendimiento (tiempo de ejecución) mejora en varios órdenes de magnitud.

Por ejemplo en los siguientes test³⁶ se obtienen estos rendimientos: Crypto++ v8.2:

```
O0 (95), -O2 (660.46), -O3 (712.01), -O3 -march=native (751.56),  
-O3 -march=native -fito (699.80), -Ofast -march=native (751.01)
```

LeelaChessZero:

```
O0 (18,300), -O2 (157,289), -O3 (142,198), -O3 -march=native  
(136,608), -O3 -march=native -fito (163,773), -Ofast -march=native  
(157,629)
```

Himeno Benchmark v3.0:

```
O0 (597.53), -O2 (4,150.11), -O3 (4,015.86), -O3 -march=native  
(4,771.42), -O3 -march=native -fito (4,774.03), -Ofast -march=native  
(5,065.07)
```

³⁵La optimización de código es el conjunto de fases de un compilador que transforma un fragmento de código en otro fragmento con un comportamiento equivalente y se ejecuta de forma más eficiente, es decir, usando menos recursos de cálculo como memoria o tiempo de ejecución.

³⁶https://www.phoronix.com/scan.php?page=news_item&px=GCC-10.1-Compiler-Optimizations

C-Ray v1.1:

O0 (113.58), -O2 (69.70), -O3 (38.00), -O3 -march=native (30.46),
-O3 -march=native -fito (30.24), -Ofast -march=native (27.13)

Geometric Mean Of All Test Results:

O0 (222.36), -O2 (681.88), -O3 (709.76), -O3 -march=native (735.14),
-O3 -march=native -fito (755.97), -Ofast -march=native (758.30)

3.7.6 Análisis de Rendimiento y Depuración

El comando *gprof* produce un perfil de ejecución de programas en C, C++, Pascal, FORTRAN77, etc. El efecto de las rutinas llamadas se incorpora en el perfil de cada llamador. Los datos del perfil se toman del fichero de perfil de grafos de llamada ('gmon.out' por omisión) que es creado por programas que se han compilado con la opción -pg de cc(1), pc(1), y f77(1). La opción -pg también enlaza al programa versiones de las rutinas de biblioteca que están compiladas para la perfilación. *Gprof* lee el fichero objeto dado (el predeterminado es 'a.out') y establece la relación entre su tabla de símbolos y el perfil de grafo de llamadas de 'gmon.out'. Si se especifica más de un fichero de perfil, la salida de *gprof* muestra la suma de la información de perfilado en los ficheros de perfil dados.

Gprof calcula la cantidad de tiempo empleado en cada rutina. Después, estos tiempos se propagan a lo largo de los vértices del grafo de llamadas. Se descubren los ciclos, y se hace que las llamadas dentro de un ciclo compartan el tiempo del ciclo. El primer listado muestra las funciones clasificadas de acuerdo al tiempo que representan incluyendo el tiempo de sus descendientes en su grafo de llamadas. Debajo de cada entrada de función se muestran sus hijos (directos) del grafo de llamadas, y cómo sus tiempos se propagan a esta función. Un despliegue similar sobre la función muestra cómo el tiempo de esta función y el de sus descendientes se propagan a sus padres (directos) del grafo de llamadas.

También se muestran los ciclos, con una entrada para el ciclo completo y un listado de los miembros del ciclo y sus contribuciones al tiempo y número de llamadas del ciclo. En segundo lugar, se da un perfil plano, similar al producido por *prof*. Este listado de los tiempos de ejecución totales, los números de llamadas, el tiempo en milisegundos que la llamada empleó en la propia rutina, y el tiempo en ms que la llamada empleó en la propia rutina

pero incluyendo sus descendientes. Finalmente, se proporciona un índice a los nombres de función.

Para obtener el análisis de rendimiento, hacemos:

```
$ g++ -g -pg -O0 *.cpp
$ ./a.out
$ gprof -c -z a.out gmon.out > sal.txt
```

el archivo *sal.txt* contiene el análisis de rendimiento detallado. Un ejemplo de esta salida es:

```
Flat profile:
Each sample counts as 0.01 seconds.
% cumulative self self total
time seconds seconds calls s/call s/call name
23.25  0.60  0.60  40656734  0.00  0.00 retorna(int, int)
14.85  0.98  0.38  27627674  0.00  0.00 retoaNumColu(int, int)
12.89  1.31  0.33  91126931  0.00  0.00 Vector::retorna(int)
10.94  1.59  0.28           31  0.01  0.03 ResJacobi::resuelve()
...
```

que permite conocer en que parte del código se consume más tiempo de ejecución.

Depuración con ddd un depurador (en inglés: Debugger) es un programa usado para probar y depurar (eliminar) los errores de otros programas (el programa "objetivo"). El código a ser examinado puede alternativamente estar corriendo en un simulador de conjunto de instrucciones (ISS), una técnica que permite gran potencia en su capacidad de detenerse cuando son encontradas condiciones específicas pero será típicamente algo más lento que ejecutando el código directamente en el apropiado (o el mismo) procesador. Algunos depuradores ofrecen dos modos de operación - la simulación parcial o completa, para limitar este impacto.

Si es un depurador de nivel de fuente o depurador simbólico, comúnmente ahora visto en entornos de desarrollo integrados, cuando el programa "se estrella" o alcanza una condición predefinida, la depuración típicamente muestra la posición en el código original. Si es un depurador de bajo nivel o un depurador de lenguaje de máquina, muestra la línea en el fuente desensamblado (a menos que también tenga acceso en línea al código fuente

original y pueda exhibir la sección apropiada del código del ensamblador o del compilador). Un "estrellamiento" sucede cuando el programa no puede continuar normalmente debido a un error de programación. Por ejemplo, el programa pudo haber intentado usar una instrucción no disponible en la versión actual del CPU o haber intentado tener acceso a memoria protegida o no disponible.

Típicamente, los depuradores también ofrecen funciones más sofisticadas tales como correr un programa paso a paso (un paso o animación del programa), parar el programa (Breaking), es decir, pausar el programa para examinar el estado actual en cierto evento o instrucción especificada por medio de un Breakpoint, y el seguimiento de valores de algunas variables. Algunos depuradores tienen la capacidad de modificar el estado del programa mientras que está corriendo, en vez de simplemente observarlo. También es posible continuar la ejecución en una posición diferente en el programa pasando un estrellamiento o error lógico.

La importancia de un buen depurador no puede ser exagerada. De hecho, la existencia y la calidad de tal herramienta para un lenguaje y una plataforma dadas a menudo puede ser el factor de decisión en su uso, incluso si otro lenguaje/plataforma es más adecuado para la tarea.

Para hacer depuración del código mediante el depurador gráfico *ddd* usar:

```
$ g++ -g -O0 *.cpp
$ ddd ./a.out
```

Puede usarse también los depuradores *xxgdb*, *gdb*, *kdbg*, *neovimer*, *lldb* cada uno tiene sus pros y contras, depende del usuario cual es el más adecuado para usar.

Depuración con valgrind es un conjunto de herramientas libres que ayuda en la depuración de problemas de memoria y rendimiento de programas.

La herramienta más usada es *Memcheck*. *Memcheck* introduce código de instrumentación en el programa a depurar, lo que le permite realizar un seguimiento del uso de la memoria y detectar los siguientes problemas:

- Uso de memoria no inicializada.
- Lectura/escritura de memoria que ha sido previamente liberada.

- Lectura/escritura fuera de los límites de bloques de memoria dinámica.
- Fugas de memoria.
- Otros.

El precio a pagar es una notable pérdida de rendimiento; los programas se ejecutan entre cinco y veinte veces más lento al usar Valgrind, y su consumo de memoria es mucho mayor. Por ello normalmente no siempre se ejecuta un programa en desarrollo usando Valgrind, sino que se usa en situaciones concretas cuando se está buscando un error determinado se trata de verificar que no haya errores ocultos como los que Memcheck puede detectar.

Valgrind incluye además otras herramientas:

- Addrcheck, versión ligera de Memcheck que se ejecuta más rápido y requiere menos memoria pero que detecta menos tipos de errores.
- Massif, mide el rendimiento del montículo (heap).
- Helgrind, herramienta de detección de condiciones de carrera (race conditions) en código multihilo.
- Cachegrind, mide el rendimiento de la Caché durante la ejecución, de acuerdo a sus características (capacidad, tamaño del bloque de datos, grado de asociatividad, etc.).

Para el rastreo de problemas con la manipulación de memoria y punteros desbordados usamos:

```
$ g++ -g -O0 *.cpp
$ valgrind --tool=memcheck --leak-check=yes --show-reachable=yes
./a.out
```

o analizar la salida usando *kCachegrind*:

```
$ valgrind --tool=callgrind ./a.out
$ kCachegrind profile.callgrind
```

Existen varios paquetes de modo gráfico para *valgrind*, uno de ellos es *allegoop* y se usa:

```
$ alleyoop ./a.out -v --arg1=foo
```

otro es *kCachegrind*, podemos ver más opciones en:

- <http://valgrind.org/>
- <http://alleyoop.sourceforge.net/usage.html>
- <http://kCachegrind.sourceforge.net/html/Home.html>

3.7.7 Mejora del Rendimiento en Python

Python es un lenguaje interpretado, pero es posible compilar el código para generar Byte Code para el intérprete (no aumenta la velocidad de ejecución). Si se necesita crear un archivo *.pyc* para un módulo que no se importa, se puede usar los módulos *py_compile* y *compile_all* desde el intérprete de Python.

El módulo *py_compile* puede compilar manualmente cualquier módulo. Una forma de usar la función *py_compile.compile* en ese módulo de forma interactiva es:

```
>>> import py_compile
>>> py_compile.compile('modulo.py')
```

esto escribirá el archivo *modulo.pyc*.

En la línea de comando de Linux es posible compilar todos los archivos en un directorio utilizando el módulo *compileall*, para ello usar:

```
$ python -m compileall *.py
```

y ejecutar mediante:

```
$ python modulo.pyc
```

También es posible hacer ligera optimización del código generado mediante:

```
$ python -O -m compileall *.py
```

esto generará código Bytecode con extensión *.pyo*, y ejecutar mediante:

```
$ python modulo.pyo
```

Python es un lenguaje razonablemente rápido, pero no es tan rápido como los programas compilados de C o Fortran. Eso es porque se interpreta *CPython*, la implementación estándar. Para ser más precisos, su código de Python se compila en un código de *Bytes* que luego se interpreta. Eso es bueno para aprender, ya que puede ejecutar el código en el *REPL* de Python y ver los resultados de inmediato en lugar de tener que compilar y ejecutar. Pero como los programas de Python no son tan rápidos, los desarrolladores han creado varios compiladores de Python³⁷ a lo largo de los años, incluidos³⁸ Numba, Nuitka, PyPy, Cython, cx_FreezeIron, Pythran, Jython entre otros.

Comparando Compiladores de Python Alguien ya ha hecho el trabajo de crear un punto de referencia de Python. Opté por *PyStone*, una traducción de un programa en C de Guido van Rossum, el creador de Python (el programa en C era en sí mismo una traducción de un programa *Ada*). Encontré una versión convertida por el desarrollador Christopher Arndt en *Github* que era capaz de probar Python 3. Para dar un sentido de perspectiva, aquí está el rendimiento de *CPython* (es decir, Python estándar) con *Pystone*:

```
Python 2.7.15Rc1 2: 272.647 pystones / second.  
Python 3.6.5: 175,817
```

Como puede ver, hay una gran diferencia entre Python 2 y 3 (cuanto más Pystones por segundo, mejor). En los siguientes desgloses, todos los compiladores de Python se compararon con Python 3.

Nuitka Aunque puede seguir las instrucciones en la página de descarga, lo siguiente en Debian GNU/Linux funcionó bien para mí:

```
$ apt install nuitka
```

³⁷El rendimiento rápido no es la única razón para compilar; Posiblemente la mayor desventaja de los lenguajes de *Scripting* como Python es que se proporciona de manera implícita su código fuente a los usuarios finales.

³⁸Si está interesado en los compiladores de Python en general, tenga en cuenta que hay mucho debate y controversia sobre los "mejores" compiladores y la rapidez general del lenguaje.

adicionalmente Nuitka también puede usar otro compilador de C (además del *gcc*), así que descargué *clang*. Puedes instalarlo con esto:

```
$ apt install clang
```

De forma predeterminada, *Nuitka* usa *gcc*, pero un parámetro te permite usar el *clang*, así que lo probé con ambos. El compilador *clang* es parte de la familia *llvm*, y está pensado como un reemplazo moderno para *gcc*. Compilar *pystone.py* con *gcc* fue tan simple como esto (primera línea), o con *clang* (segunda línea), y con la optimización del tiempo de enlace para *gcc* (tercera línea):

```
$ nuitka pystone.py
$ nuitka pystone.py -clang
$ nuitka pystone.py -lto
```

Después de compilar, lo que tomó aproximadamente 10 segundos, ejecuté el *pystone.exe* desde la terminal con:

```
$ ./pystone.exe 500000
```

Hice 500,000 pases:

Tamaño Ejecución pystones / seg.

1. 223.176 Kb 597,000
2. 195,424 Kb 610,000
3. 194.2 kb 600,000

Estos fueron los promedios de más de 5 corridas. Había cerrado tantos procesos como pude, pero tomo los tiempos con un poco de variación porque había un +/- 5% en los valores de tiempo.

PyPy Guido van Rossum dijo una vez: "Si quieres que tu código se ejecute más rápido, probablemente debes usar PyPy". Para instalarlo en Debian GNU/Linux usar:

```
$ apt install pypy
```


Entonces lo corrí así:

```
$ pypy pystone.py
```

El resultado fue una asombrosa cantidad de 1,776,001 pystones por segundo, casi tres veces más rápido que *Nuitka*.

PyPy usa un compilador justo a tiempo y hace algunas cosas muy inteligentes para alcanzar su velocidad. De acuerdo con los puntos de referencia reportados, es 7.6 veces más rápido que el *CPython* en promedio. Puedo creer eso fácilmente. La única (leve) desventaja es que siempre está un poco por detrás de las versiones de Python (es decir, hasta 2.7.13 (no 2.7.15) y 3.5.3 (no 3.6.5)). Producir un exe requiere un poco de trabajo. Tienes que escribir tu Python en un subconjunto llamado *RPython*.

Cython no es solo un compilador para Python; es para un superconjunto de Python que admite la interoperabilidad con C / C++. CPython está escrito en C, por lo que es un lenguaje que generalmente se combina bien con Python.

Configurar las cosas con *Cython* es un poco complicado. No es como *Nuitka*, que acaba de salir de la caja. Primero, debes comenzar con un archivo de Python con una extensión .pyx; ejecuta *Cython* para crear un archivo pystone.c a partir de eso:

```
$ cython pystone.pyx --embed
```

No omita el parámetro *-embed*. Se agrega en *main* y eso es necesario. A continuación, compila pystone.c con esta hermosa línea:

```
$ gcc $(python3-config --includes) pystone.c -lpython3.6m -o  
pystone.exe
```

Si recibe algún error, como "no se puede encontrar la versión -lpython", podría ser el resultado de su versión de Python. Para ver qué versión está instalada, ejecute este comando:

```
$ pkg-config --cflags python3
```

Después de todo eso, *Cython* solo dio 228,527 pystones / sec. Sin embargo, *Cython* necesita que hagas un poco de trabajo especificando los tipos de variables. Python es un lenguaje dinámico, por lo que no se especifican los tipos; *Cython* utiliza la compilación estática y el uso de variables de tipo C le permite producir un código mucho mejor optimizado. (La documentación es bastante extensa y requiere lectura).

Tamaño Ejecución pystones / seg.

1. 219.552 Kb 228.527

cx_freeze es un conjunto de Scripts y módulos para "congelar" Scripts de Python en ejecutables, y se pueden encontrar en *Github*.

Lo instalé y creé una carpeta congelada para administrar cosas en:

```
$ pip3 install cx_Freeze --upgrade
```

Un problema que encontré con el Script de instalación fue un error que falta "lz". Necesitas tener instalado zlib; ejecuta esto para instalarlo:

```
$ apt install zlib1g-dev
```

Después de eso, el comando *cx_Freeze* tomó el Script *pystone.py* y creó una carpeta dist que contenía una carpeta *lib*, un archivo *lib* de 5MB y el archivo de aplicación pystone:

```
$ cxfreeze pystone.py --target-dir dist
```

Tamaño Ejecución pystones / seg.

1. 10,216 Kb 174,822

No es el rendimiento más rápido, porque es la misma velocidad que *CPython*. (La congelación de Python implica enviar su aplicación en un solo archivo (o carpeta) con los elementos Python necesarios, en lugar de compilar; significa que el destino no requiere Python).

Numba Este es un compilador "justo a tiempo" para Python que optimiza el código que se usa en algoritmos numéricos como son en las matrices, bucles y funciones de NumPy (también da soporte a *Threading*, vectorización *SIMD* y aceleración por *GPUs*: *Nvidia CUDA*, *AMD ROC*). La forma más común de usar Numba es a través de su colección de decoradores que se pueden aplicar a sus funciones para indicar a Numba que las compile usando el estándar *LLVM*. Cuando se realiza una llamada a una función decorada de Numba, se compila en el código de máquina "justo a tiempo" para su ejecución y todo o parte de su código puede ejecutarse posteriormente a la velocidad de código de máquina nativo. Numba también trabaja bien con Jupiter notebook para computación interactiva y con ejecución distribuida como *Dask* y *Spark*.

Se puede instalar en Debian GNU/Linux mediante:

```
$ apt install python3-numba
```

y se puede descargar mediante *CONDA* paquete de *Anaconda*, usando:

```
$ conda install numba
```

o mediante *PIP* usando:

```
$ pip install numba
```

Dando mejores resultados en la ejecución de múltiples pruebas que *PyPy*, pero no en todos los casos. Por ello, la recomendación es evaluar el rendimiento mediante pruebas en cada caso particular.

Conclusión Una buena opción es *PyPy* por el rendimiento obtenido en código general (y dependiendo del código en cuestión Numba puede ser mejor que *PyPy* en aplicaciones de cómputo científico), la compilación fue muy rápida y produjo los resultados en menos de un segundo después de presionar la tecla *RETURN*. Si requieres un ejecutable, sin embargo, te recomiendo Nuitka; fue una compilación sin complicaciones y se ejecuta más rápido que CPython. Experimenta con estos compiladores de Python y vea cuál funciona mejor para tus necesidades particulares.

3.7.8 Git

Git es un programa de control de versiones que sirve para la gestión de los diversos cambios que se realizan sobre los elementos de algún proyecto de Software y sus respectivos programas fuente o configuración del mismo guardando instantáneas (Snapshots) del código en un estado determinado, que viene dado por el autor y la fecha. Fue diseñado por Linus Torvalds y es usado para controlar los cambios de diversos proyectos como los fuentes del Kernel de Linux que tiene decenas de millones de líneas de código (en la versión 4.12 cuenta con 24,170,860 líneas de código repartidos en 59,806 archivos) y es trabajado por miles de programadores alrededor del mundo.

¿Qué es control de versiones? se define como control de versiones a la gestión de los diversos cambios (Commit es un conjunto de cambios guardados en el repositorio de Git y tiene un identificador SHA1 único) que se realizan sobre los elementos de algún producto o una configuración del mismo, es decir, es lo que se hace al momento de estar desarrollando un Software o una página Web. Exactamente es eso que haces cuando subes y actualizas tu código en la nube, o le añades alguna parte o simplemente editas cosas que no funcionan como deberían o al menos no como tú esperarías.

¿A que le llamamos sistema de control de versiones? son todas las herramientas que nos permiten hacer todas esas modificaciones antes mencionadas en nuestro código y hacen que sea más fácil la administración de las distintas versiones de cada producto desarrollado; es decir *Git*.

Antes de seguir avanzando, conviene definir algunos términos comunes que encontraremos más adelante:

- Un repositorio es una carpeta que contiene los archivos y subdirectorios de un proyecto. Puede ser público o privado, dependiendo de quién deba tener acceso.
- Una rama es una ruta de desarrollo separada en el mismo repositorio. En un mismo proyecto suelen utilizarse ramas separadas para trabajar en nuevas características sin interferir con la versión de producción. Una vez que el código es revisado y probado, un administrador del repositorio puede fusionar los cambios en la rama principal.

- Un commit es una instantánea de un repositorio en un momento dado. Permite a los programadores incluir comentarios y pedir la opinión de otras personas. Usando el Hash que lo identifica puedes volver a un estado anterior del proyecto si es necesario. Antes de que los archivos y directorios puedan ser comiteados, necesitamos decirle a Git que los rastree. Normalmente nos referimos a este paso simplemente como agregar los archivos al área de Staging.
- Un Pull Request (o simplemente PR) es un método para informar a otros desarrolladores y discutir los cambios recientes antes de incorporarlos a la ruta de desarrollo principal.
- Un Fork es un proyecto independiente que se basa en un repositorio determinado. A diferencia de las ramas, no es local a este último. Sin embargo, también puede fusionarse con él a través de un Pull Request adecuado.
- Se puede utilizar un archivo .gitignore para indicar qué contenido local no queremos incluir en el repositorio. Esto nos ayuda a evitar enviar archivos temporales o exponer información sensible en una solución basada en la Web.

Git fue creado pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente, es decir *Git* nos proporciona las herramientas para desarrollar un trabajo en equipo de manera inteligente y rápida y por trabajo nos referimos a algún Software o página que implique código el cual necesitemos hacerlo con un grupo de personas.

Algunas de las características más importantes de *Git* son:

- Rapidez en la gestión de ramas (Branches es como una línea de tiempo a partir de los Commits, siempre hay siempre como mínimo una rama principal predefinida llamada Master), debido a que *Git* nos dice que un cambio será fusionado mucho más frecuentemente de lo que se escribe originalmente.
- Gestión distribuida: Los cambios se importan como ramas adicionales y pueden ser fusionados de la misma manera como se hace en la rama local.

- Gestión eficiente de proyectos grandes.
- Realmacenamiento periódico en paquetes.

Con esto en mente, vamos a hacer una introducción a Git desde cero.

Instalación de Git para instalar *Git* completo en el servidor o en la máquina de trabajo, usamos:

```
# apt install git-all
```

Para instalar lo básico de *Git*, si no está instalado:

```
# apt install git
```

otras opciones para trabajar con *Git* son:

```
# apt install git git-all gitk gitg git-cola git-gui qgit tig lighttpd  
vim-fugitive
```

También existen otros proyectos parecidos a Git, algunos de ellos son:

```
# apt install mercurial  
# apt install subversion rapidsvn  
# apt install cvs
```

Configuración de Git si se quiere especificar la identidad del que controla el repositorio local en el equipo, debemos usar (por omisión toma la información de la cuenta del usuario y máquina):

```
$ git config --global user.name "Antonio Carrillo"  
$ git config --global user.email antoniocarrillo@ciencias.unam.mx
```

si se desea configurar el editor de texto a usar por *Git*, usamos (por omisión es *vim*):

```
$ git config --global Core.editor scite
```

si se desea configurar la herramienta de control de diferencias, usamos (por omisión *vimdiff*):

```
$ git config --global merge.tool meld
```

para comprobar podemos usar:

```
git config --global -list
```

el cual creará un archivo de texto llamado `.gitconfig`, y podemos ver su contenido, usando:

```
cat ~/.gitconfig
```

Clonar un repositorio Git podemos iniciar clonando algún repositorio remoto de la red, para ello debemos conocer su dirección y usar:

```
$ git clone <dirección>
```

esto permitirá bajar todo el contenido del proyecto a clonar.

Crear un repositorio Git local si lo que requiero es un control personal sin necesidad de compartir los archivos con ningún otro usuario, puedo usar *Git* de forma local en cualquier directorio mediante:

```
$ git init
```

Si se desea agregar la identidad del que controla el repositorio en este directorio, se debe usar:

```
$ git config user.name "Antonio Carrillo"  
$ git config user.email antoniocarrillo@ciencias.unam.mx
```

Ahora para agregar los archivos (todos los de este directorio), usar:

```
$ git add .
```

así podemos hacer la confirmación de los cambios, mediante:

```
$ git commit -m "Primer lanzamiento"
```

ahora cada que lo requiera al hacer modificaciones, puedo checar los cambios:

```
$ git status
```

o en forma gráfica con *gitk*, mediante:

```
$ gitk
```

para actualizar los cambios, usar:

```
$ git commit -a -m 'Actualizacion'
```

en caso necesario, podemos mover uno o más archivos de una trayectoria a otra, usando:

```
$ git mv archivo(s) /trayectoria
```

o podemos borrar uno o más archivos, usando:

```
$ git rm archivo(s)
```

además podemos editar el archivo *.gitignore*, para indicar los archivos a ignorar usando líneas independientes para cada tipo.

Podemos ver el histórico de las operaciones que hemos hecho, usando:

```
$ git log
```

y podemos comprobar el estado del repositorio, usando:

```
$ git status
```

La otra alternativa es preparar un directorio para el repositorio ya sea en el servidor o de forma local, mediante:


```
$ mkdir example.git  
$ cd example.git
```

Para inicializar el repositorio:

```
$ git --bare init
```

Es buena opción limitar el acceso a la cuenta vía *ssh*, por ello es mejor cambiar en */etc/passwd*, la línea del usuario predeterminada:

```
tlahuiz:x:1005:1005:Tlahuizcalpan,,,:/home/tlahuiz:/bin/bash
```

a esta otra:

```
tlahuiz:x:1005:1005:Tlahuizcalpan,,,:/home/tlahuiz:/usr/bin/git-  
Shell
```

En la máquina de trabajo o en el servidor en cualquier carpeta se genera la estructura del repositorio en un directorio temporal de trabajo para el repositorio:

```
$ mkdir tmp  
$ cd tmp  
$ git init
```

Para generar la estructura de trabajo para el repositorio y los archivos necesarios:

```
$ mkdir branches release trunk  
$ mkdir ...
```

Para adicionar todos y cada uno de los archivos y carpetas:

```
$ git add .
```

Para subir los cambios:

```
$ git commit -m "Texto"
```

Después debemos mandarlo al servidor:

```
$ git remote add origin ssh://usr@máquina/~ /trayectoria
```

o mandarlo a un directorio local:

```
$ git remote add origin ~/trayectoria
$ git push origin +master:refs/heads/master
```

Para usar el repositorio en cualquier otra máquina hay que bajar el repositorio por primera vez del servidor:

```
$ git clone ssh://usr@máquina/~ /trayectoria
```

o de una carpeta local:

```
$ git clone ~/trayectoria
```

Ahora, podemos configurar algunos datos usados en el control de cambios:

```
$ git config --global usr.name "Antonio Carrillo"
$ git config --global usr.email antoniocarrillo@ciencias.unam.mx
```

cuando se requiera actualizar del repositorio los cambios:

```
$ git pull
```

para subir los cambios al repositorio:

```
$ git commit -a -m "mensaje"
$ git push
```

Comando usados para el trabajo cotidiano en *Git* para ver el estado de los archivos locales, usamos:

```
$ git status
```

para generar una nueva rama y trabajar en ella:

```
$ git branch MiIdea  
$ git checkout MiIdea
```

o en un solo paso:

```
$ git checkout -b MiIdea
```

Para unificar las ramas generadas en el punto anterior:

```
$ git checkout master  
$ git merge MiIdea
```

Para borrar una rama:

```
$ git branch -d MiIdea
```

Para listar ramas:

```
$ git branch
```

Para listar ramas fusionadas:

```
$ git branch --merged
```

Para listar ramas sin fusionar:

```
$ branch --no-merged
```

Para ver los cambios en el repositorio:

```
$ git log
```

o verlos en forma acortada:

```
$ git log --pretty=oneline
```

Para recuperar un archivo de una actualización anterior:

```
$ git show a30ab2ca64d81876c939e16e9dac57c8db6fb103:ruta/al/archivo  
> ruta/al/archivo.bak
```

Para volver a una versión anterior:

```
$ git reset --hard 56f8fb550282f8dfaa75cd204d22413fa6081a11:
```

para regresar a la versión presente (cuidado con subir cambios en ramas anteriores):

```
$ git pull
```

Si en algún momento borramos algo o realizamos cambios en nuestra máquina y necesitamos regresar los archivos como estaban en nuestra última actualización, podemos usar:

```
$ git reset --hard HEAD
```

este trabaja con la información de nuestra copia local y no necesita conexión de red para la restitución. Eventualmente es necesario optimizar la copia local de los archivos en *Git*, para ello podemos usar:

```
$ git gc
```

Visualizador gráfico para *Git*:

```
# apt install gitk
```

Git es un proyecto pujante, amplio y bien documentado, ejemplos y documentación puede ser consultada en:

- <https://git-scm.com/book/es/v1>
- <http://git-scm.com/documentation>
- <https://coderwall.com/p/kucyaw/protect-secret-data-in-git-repo>

Git en Google Drive:

- <http://www.iexplain.org/using-git-with-google-drive-a-tutorial/>
- <https://techstreams.github.io/2016/09/07/google-drive-as-simple-git-Host/>

Un Resumen de los Comando de Git más Usados

CONFIGURACION

Configurar Nombre que salen en los commits

```
git config --global user.name "antonio"
```

Configurar Email

```
git config --global user.email antonio@gmail.com
```

Marco de colores para los comando

```
git config --global color.ui true
```

INICIANDO REPOSITORIO

Iniciamos GIT en la carpeta donde está el proyecto

```
git init
```

Clonamos el repositorio de github o bitbucket

```
git clone <url>
```

Añadimos todos los archivos para el commit

```
git add .
```

Hacemos el primer commit

```
git commit -m "Texto que identifique por que se hizo el com-  
mit"
```

Subimos al repositorio

```
git push origin master
```

GIT CLONE

Clonamos el repositorio de github, gitlab o bitbucket

```
git clone <url>
```

Clonamos el repositorio de github, gitlab o bitbucket

```
git clone <url> git-demo
```

GIT ADD

Añadimos todos los archivos para el commit

```
git add .
```

Añadimos el archivo para el commit

```
git add <archivo>
```

Añadimos todos los archivos para el commit omitiendo los nuevos

```
git add --all
```

Añadimos todos los archivos con la extensión especificada

```
git add *.txt
```

Añadimos todos los archivos dentro de un directorio y de una extensión específica

```
git add docs/*.txt
```

Añadimos todos los archivos dentro de un directorios

```
git add docs/
```

GIT COMMIT

Cargar en el HEAD los cambios realizados

```
git commit -m "Texto que identifique por que se hizo el commit"
```

Agregar y Cargar en el HEAD los cambios realizados

```
git commit -a -m "Texto que identifique por que se hizo el  
commit"
```

De haber conflictos los muestra

```
git commit -a
```

Agregar al último commit, este no se muestra como un nuevo commit en los logs. Se puede especificar un nuevo mensaje

```
git commit --amend -m "Texto que identifique por que se hizo  
el commit"
```

GIT PUSH

Subimos al repositorio

```
git push <origien> <branch>
```

Subimos un tag

```
git push --tags
```

GIT LOG

Muestra los logs de los commits

```
git log
```

Muestras los cambios en los commits

```
git log --oneline --stat
```

Muestra gráficos de los commits

```
git log --oneline --graph
```

GIT DIFF

Muestra los cambios realizados a un archivo

```
git diff
git diff --staged
```

GIT HEAD

Saca un archivo del commit

```
git reset HEAD <archivo>
```

Devuelve el último commit que se hizo y pone los cambios en staging

```
git reset --soft HEAD^
```

Devuelve el último commit y todos los cambios

```
git reset --hard HEAD^
```

Devuelve los 2 últimos commit y todos los cambios

```
git reset --hard HEAD^^
```

Rollback merge/commit

```
git log
git reset --hard <commit_sha>
```

GIT REMOTE

Agregar repositorio remoto

```
git remote add origin <url>
```

Cambiar de remote

```
git remote set-url origin <url>
```

Remover repositorio

```
git remote rm <name/origin>
```

Muestra lista repositorios


```
git remote -v
```

Muestra los branches remotos

```
git remote show origin
```

Limpiar todos los branches eliminados

```
git remote prune origin
```

GIT BRANCH

Crea un branch

```
git branch <nameBranch>
```

Lista los branches

```
git branch
```

Comando -d elimina el branch y lo une al master

```
git branch -d <nameBranch>
```

Elimina sin preguntar

```
git branch -D <nameBranch>
```

GIT TAG

Muestra una lista de todos los tags

```
git tag
```

Crea un nuevo tags

```
git tag -a <version> - m "esta es la versión x"
```

GIT REBASE

Los rebase se usan cuando trabajamos con branches esto hace que los branches se pongan al día con el master sin afectar al mismo

Une el branch actual con el master, esto no se puede ver como un merge

```
git rebase
```

Cuando se produce un conflicto no das las siguientes opciones:

cuando resolvemos los conflictos *-continue* continua la secuencia del rebase donde se pauso

```
git rebase -continue
```

Omite el conflicto y sigue su camino

```
git rebase -skip
```

Devuelve todo al principio del rebase

```
git rebase -abort
```

Para hacer un rebase a un branch en específico

```
git rebase <nameBranch>
```

OTROS COMANDOS

Lista un estado actual del repositorio con lista de archivos modificados o agregados

```
git status
```

Quita del HEAD un archivo y le pone el estado de no trabajado

```
git checkout - <file>
```

Crea un branch en base a uno Online

```
git checkout -b newlocalbranchname origin/branch-name
```

Busca los cambios nuevos y actualiza el repositorio

`git pull origin <nameBranch>`

Cambiar de branch

`git checkout <nameBranch/tagname>`

Une el branch actual con el especificado

`git merge <nameBranch>`

Verifica cambios en el repositorio Online con el local

`git fetch`

Borrar un archivo del repositorio

`git rm <archivo>`

Fork

Descargar remote de un fork

`git remote add upstream <url>`

Merge con master de un fork

`git fetch upstream`

`git merge upstream/master`

Git-crypt El paquete *git-crypt* es una solución que usa *GPG* por debajo de *Git* que permite cifrado y descifrado transparente de archivos en un repositorio *git*.

Los archivos que se requieran proteger serán cifrados al hacer commit y descifrados al hacer *checkout* y permite compartir libremente un repositorio que contenga contenido tanto público como privado. De esta forma, permite trabajar de manera transparente con el contenido descifrado, de forma que desarrolladores que no tengan la clave secreta podrán clonar y hacer commit en un repositorio con archivos cifrados.

Esto te permite almacenar tu material secreto (como pueden ser claves) en el mismo repositorio que tu código sin tener que bloquearlo. Solo un usuario autorizado puede dar permisos a otros usuarios.

Para instalar el paquete *git-crypt* usamos:

```
# apt install git-crypt
```

Ya instalado debemos preparar el repositorio git, para crear la llave, entonces usar:

```
$ git-crypt keygen ~/crypt-key
```

Ahora podemos crear el repositorio:

```
$ cd repo  
$ git-crypt init
```

Especifica que carpetas/archivos deben ser cifrados, como git-filters:

```
$ cat .gitattributes
```

```
keys filter=git-crypt diff=git-crypt
```

crear la lista de los archivos a cifrar

```
$ vi .gitattributes
```

Indicamos que se cifren, por ejemplo, los archivos *.html*, *.org*, directorio:secretdir/**secreto y archivo, con cualquier extensión o palabra que le preceda.

```
*.html filter=git-crypt diff=git-crypt  
*.org filter=git-crypt diff=git-crypt  
directorio_secreto/** filter=git-crypt diff=git-crypt  
*archivo* filter=git-crypt diff=git-crypt
```

ahora cada vez que hagamos un commit, los archivos *.html* y *.org*, subirán cifrados.

Ya podemos usar la llave para cifrar los archivos indicados por *.gitattributes* mediante:

```
$ git-crypt unlock ~/crypt-key
```

y agregar los archivos que deseamos cifrar, usando *git add*, revisando el estado de los archivos cifrados mediante:

```
$ git-crypt status -f
```

y podemos hacer los commits necesarios.

Al clonar el repositorio, los archivos cifrados se mostrarán como tal, hasta hacer en el repositorio:

```
$ git-crypt unlock ~/crypt-key
```

mostrando los archivos descifrados a partir de ese momento

Si se desea respaldar el repositorio en un solo archivo se puede usar:

```
$ git bundle create /tmp/Respaldo --all
```

y para restaurar usar algo como:

```
$ git clone /tmp/Respaldo newFolder
```

También podemos añadir usuarios autorizados (identificados por su clave *GPG*), mediante:

```
$ git-crypt add-gpg-user USER_ID
```

Flujos de trabajo comunes

- En la máquina del desarrollador: Crea el vault, añádate como usuario fiable. Pide las claves públicas a los miembros de tu equipo y añádelas al vault.
- En el entorno de Integración Continua (CI): Añade una clave *GPG* común para los ejecutores jenkins/CI. Autorízala en el repositorio.

Seguridad

- Git-crypt usa *GPG* internamente, así que el nivel de seguridad debería ser el dado por *GPG*, a excepción de posibles errores en el propio programa *git-crypt*.
- Git-crypt es más seguro que otros sistemas git de cifrado transparente, *git-crypt* cifra archivos usando *AES-256* en modo *CTR* con un synthetic IV derivado del *SHA-1 HMAC* del archivo. Este modo de operar proporciona seguridad semántica ante *CPAs* (chosen-plain attacks) determinísticos. Esto significa que pese a que el cifrado es determinística (lo cual es requerido para que git pueda distinguir cuando un archivo ha cambiado y cuando no), no filtra información más allá de mostrar si dos archivos son idénticos o no.

Limitaciones y Trucos

- Cualquier usuario no autorizado puede ver que estamos usando *git-crypt* basándose en la evidencia dejada en el archivo `.gitattributes`.
- Git-crypt no cifra nombres de archivo, mensajes de *commit*, *symlink targets*, *gitlinks*, u otros metadatos.
- Git-crypt se apoya en git filters, los cuales no fueron diseñados con el cifrado en mente. Así pues, *git-crypt* no es la mejor herramienta para cifrar la mayoría o totalidad de los archivos de un repositorio. Donde *git-crypt* destaca es en aquellos casos en que la mayoría del repositorio es público pero unos pocos archivos deben ser cifrados (por ejemplo, claves privadas o archivos con credenciales API). Para cifrar un repositorio entero, es mejor considerar usar un sistema como **git-remote-gcrypt**.
- Git-crypt no esconde cuando un archivo cambia o no, cuanto ocupa o el hecho de que dos archivos sean idénticos.
- Los archivos cifrados con *git-crypt* no se pueden comprimir. Incluso el más pequeño de los cambios en un archivo cifrado requiere que git archive el archivo modificado en su totalidad y no solo un delta.
- A pesar de que *git-crypt* protege el contenido de los archivos individuales con *SHA-1 HMAC*, *git-crypt* no puede ser usado de forma segura a menos que el repositorio entero esté protegido contra la alteración de

datos (un atacante que pueda mutar tu repositorio podrá alterar tu archivo *.gitattributes* para deshabilitar el cifrado). Si fuera necesario, usa características de *git* cómo signed tags en vez de contar únicamente con *git-crypt* para la integridad.

- El *diff* del *commit* varía cuando el vault está abierto vs cuando está cerrado. Cuando está abierto, los contenidos del archivo están en formato plano, es decir, descifrados. En consecuencia puedes ver el *diff*. Cuando el vault está cerrado, no se puede apreciar un *diff* efectivo ya que el texto cifrado cambia, pero el ojo humano no puede distinguir los contenidos.

Además de Git usado de forma local, existen diversos servicios en la nube³⁹ que permiten dar soporte a proyectos mediante *Git*, en los cuales es necesario crear una cuenta y subir los datos usando *Git*, algunos de estos servicios son:

3.7.9 GitLab vs GitHub

Aunque GitHub y GitLab tienen similitudes, incluso en el propio nombre que comienza por Git debido a que ambas se basa en la famosa herramienta de control de versiones escrita por Linus Torvalds, pero ni una ni otra son exactamente iguales. Por ello, el ganador de la batalla GitHub vs GitLab no está tan claro, tienen algunas diferencias que hacen que tengan sus ventajas y desventajas para los usuarios y desarrolladores que las suelen usar.

¿Qué es GitHub? es una plataforma de desarrollo colaborativo, también llamado forja. Es decir, una plataforma enfocada hacia la cooperación entre desarrolladores para la difusión y soporte de su software (aunque poco a poco se ha ido usando para otros proyectos más allá del Software).

Como su propio nombre indica, se apoya sobre el sistema de control de versiones *Git*. Así, se puede operar sobre el código fuente de los programas y llevar un desarrollo ordenado. Además, esta plataforma está escrita en Ruby on Rails.

³⁹Algunos de estos proyectos gratuitos son: Gitlab, Github, Bitbucket, Beanstalk, Launchpad, SourceForge, Phabricator, GitBucket, Gogs, Gitea, Apache Allura, entre otros.

Tiene una enorme cantidad de proyectos de código abierto almacenada en su plataforma y accesibles de forma pública. Tal es su valor que Microsoft optó por comprar esta plataforma en 2018, aportando una cifra de nada menos que de 7,500 millones de dólares.

Pese a las dudas sobre esa compra, la plataforma continuó operando como de costumbre, y continúa siendo una de las más usadas. En ella se alojan proyectos tan importantes como el propio Kernel Linux.

¿Qué es GitLab? es otra alternativa a GitHub, otro sitio de forja con un servicio Web y un sistema de control de versiones también basado en Git. Por supuesto, se ideó para el alojamiento de proyectos de código abierto y para facilitar la vida de los desarrolladores, pero existen algunas diferencias con el anterior.

Esta Web, además de la gestión de repositorios y control de versiones, también ofrece alojamiento para Wikis, y sistema de seguimiento de errores. Una completa suite para crear y gestionar los proyectos de todo tipo, ya que, al igual que GitHub, actualmente se alojan proyectos que van más allá del código fuente.

Fue escrito por unos desarrolladores ucranianos, Dmitry Zaporozhets y Valery Sizov, usando lenguaje de programación Ruby y algunas partes en Go. Después se mejoró su arquitectura con Go, Vue.js, y Ruby on Rails, como en el caso de GitHub.

A pesar de ser muy conocida y ser la gran alternativa a GitHub, no cuenta con tantos proyectos. Eso no quiere decir que la cantidad de código alojado sea muy grande, con organizaciones que confían en ella de la talla del CERN, la NASA, IBM, Sony, etc.

Personalmente, te diría que no existe un claro ganador en la batalla GitHub vs GitLab. No es tan sencillo elegir una plataforma que sea infinitamente superior a la otra, de hecho, cada una tiene sus puntos fuertes y sus puntos débiles. Y todo dependerá de lo que realmente busques para que te tengas que decantar por una u otra.

Diferencias GitHub vs GitLab A pesar de todas las similitudes, una de las claves a la hora de decantarse en la comparativa GitHub vs GitLab pueden ser las diferencias entre ambas:

Niveles de autenticación: GitLab puede establecer y modificar permisos a los diferentes colaboradores según su función. En el caso de GitHub, puede

decidir quién tiene derechos de lectura y escritura en un repositorio, pero es más limitado en ese sentido.

- Alojamiento: aunque ambas plataformas permiten alojar el contenido de los proyectos en las propias plataformas, en el caso de GitLab también te puede permitir autoalojar tus repositorios, lo que puede ser una ventaja en algunos casos. GitHub ha agregado esa característica también, pero solo con ciertos planes de pago. GitHub ofrece máximo 3 colaboradores gratis y hasta 500 MB por repositorio, en cambio en GitLab no hay límite al número de colaboradores y hasta 10 GB por repositorio.
- Importación y exportación: GitLab contiene información muy detallada de cómo poder importar proyectos para moverlos de una plataforma a otra, como GitHub, Bitbucket, o traerlos a GitLab. Además, a la hora de exportar, GitLab ofrece un trabajo muy sólido. En el caso de GitHub no se ofrece documentación detallada, aunque se puede usar GitHub Importer como herramienta, aunque puede ser algo más restrictivo cuando se trata de exportar.
- Comunidad: ambos tienen una buena comunidad tras de sí, aunque GitHub parece haber ganado la batalla en popularidad. Actualmente aglutina millones de desarrolladores. Por eso, será más sencillo encontrar ayuda al respecto.
- Versiones Enterprise: ambos las ofrecen si pagas la cuota, por lo que se podría pensar que la comparativa GitHub vs GitLab no tiene sentido en este punto, pero lo cierto es que GitLab ofrece unas prestaciones muy interesantes, y se ha hecho popular entre los equipos de desarrollo muy grandes.

Ventajas y Desventajas de GitLab una vez conocidas las diferencias y semejanzas entre GitHub vs GitLab, las ventajas y desventajas de estas plataformas te pueden ayudar a decidirte.

Ventajas

- Plan gratuito y sin limitaciones, aunque tiene planes de pago.
- Es de licencia de código abierto.

- Permite el autohospedaje en cualquier plan.
- Está muy bien integrado con Git.

Desventajas

- Su interfaz puede ser algo más lenta con respecto a la competencia.
- Existen algunos problemas habituales con los repositorios.

Ventajas y Desventajas de GitHub Por otro lado, GitHub también tiene sus pros y contras, entre los que destacan los siguientes:

Ventajas

- Servicio gratuito, aunque también tiene servicios de pago.
- Búsqueda muy rápida en la estructura de los repos.
- Amplia comunidad y fácil encontrar ayuda.
- Ofrece prácticas herramientas de cooperación y buena integración con Git.
- Fácil integración con otros servicios de terceros.
- Trabaja también con TFS, HG y SVN.

Desventajas

- No es absolutamente abierto.
- Tiene limitaciones de espacio, ya que no puedes exceder de 100MB en un solo archivo, mientras que los repositorios están limitados a 1GB en la versión gratis.

Como ves, no existe un claro ganador. La elección no es fácil y, como comenté, deberías vigilar muy bien las ventajas, desventajas y diferencias de cada uno para poder identificar cuál se adapta mejor a tus necesidades.

Personalmente te diría que si quieres disponer de un entorno totalmente abierto, mejor usa GitLab. En cambio, si prefieres más facilidades y usar el servicio Web con más presencia, entonces ve a por GitHub. Incluso incluiría un tercero en discordia y te diría que si buscas trabajar con servicios Atlassian deberías mirar del lado de Bitbucket.

Uso GitLab (<https://about.gitlab.com/>)

Para configurar:

```
git config --global user.name "Antonio Carrillo Ledesma"
git config --global user.email "antoniocarrillo@ciencias.unam.mx"
```

Para crear nuevo repositorio:

```
git clone https://gitlab.com/antoniocarrillo69/MDF.git
cd MDF
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

Para usar una carpeta existente:

```
cd existing_folder
git init
git remote add origin https://gitlab.com/antoniocarrillo69/MDF.git
git add .
git commit -m "Initial commit"
git push -u origin master
```

Para usar un repositorio existente:

```
cd existing_repo
git remote rename origin old-origin
git remote add origin https://gitlab.com/antoniocarrillo69/MDF.git
git push -u origin --all
git push -u origin --tags
```

Uso Github (<https://github.com/>)

Para configurar:

```
git config --global user.name "Antonio Carrillo Ledesma"
git config --global user.email "antoniocarrillo@ciencias.unam.mx"
```

Para configurar un nuevo repositorio:

```
$ touch README.md
$ git init
$ git add .
$ git commit -m "mi primer commit"
$ git remote add origin https://github.com/antoniocarrillo69/ejemploPruebas.git
$ git pull origin master
$ git push origin master
```

3.7.10 Otras opciones

Herramientas para convertir código fuentes en HTML, usando:

```
$ code2html Fuente Salida.html
```

o

```
$ c2html Fuente
```

Para java, usamos:

```
$ java2html Fuentes
```

También podemos convertir código fuente en PDF, usando:

```
$ nl test.cpp | a2ps -1 -l100 -otest.ps ; ps2pdf test.ps
```

el primer comando numera las líneas del fuente, el segundo comando generará del fuente numerado un .PS y el último comando convierte .PS a .PDF

Si se tiene que ejecutar múltiples programas que son independientes uno de otro se puede usar el programa *parallel* para correr N (número de cores del equipo) de ellos al mismo tiempo, por ejemplo si tenemos un archivo Bash con el nombre *mi-Bash* y cuyo contenido es:

```
./a.out 4 5 4 > a1.txt
./a.out 4 5 3 > a2.txt
./a.out 4 5 6 > a3.txt
./a.out 4 5 4 > a4.txt
./a.out 3 5 4 > a5.txt
./a.out 4 6 4 > a6.txt
```

entonces podemos ejecutarlo usando *parallel*, el programa usará el número máximo de cores disponibles:

```
$ parallel -v < mi-Bash
```

si solo se desea usar una determinada cantidad de cores (por ejemplo 3) entonces usamos:

```
$ parallel -v -j 3 < mi-Bash
```

3.8 Programando Desde la Nube

Existen diferentes servicios Web⁴⁰ que permiten editar, compilar y ejecutar código de diversos lenguajes y paquetes desde el **navegador**, esto en aras de que los estudiantes y profesores que cuenten con algún sistema de acceso a red y un navegador puedan programar en los más diversos lenguajes, IDEs y terminales sin hacer instalación alguna en su equipo de cómputo, tableta o teléfono celular.

Algunos ejemplos de estos servicios son:

- <https://www.jdoodle.com/>
- <https://repl.it/>
- <http://browxy.com>
- <https://jupyter.org/try>
- <https://tio.run/>
- <https://www.compilejava.net/>
- <http://codepad.org/>
- <https://code.hackerearth.com/>
- <https://www.remoteinterview.io/online-c-compiler>
- <https://ideone.com/>
- <https://hackide.herokuapp.com/>
- <https://www.codechef.com/ide>
- <http://cpp.sh/>

⁴⁰Cuando se trabaja desde la Web es recomendable usar el modo Privado o Incógnito para no guardar el historial de navegación, información introducida en los formularios y borrar al cerrar el navegador los datos de los sitios visitados. Pero recuerda que los sitios Web que visitamos sí guardan información de nuestra visita, nuestro proveedor de Internet también guarda constancia de nuestra visita y si descargamos algo, esto no se borra al igual que el historial de descargas, además de las marcas de páginas o favoritos se conservarán al cerrar el navegador.

- <https://codebunk.com/>
- <https://rextester.com/>
- <https://www.tutorialspoint.com/codingground.htm>
- <https://www.compileonline.com>
- <http://pythonfiddle.com/>
- <https://trinket.io/python>
- <https://www.pythonanywhere.com/try-ipython/>
- <https://www.rollapp.com/>
- <https://godbolt.org/>
- <https://www.codiva.io/>
- <https://paiza.io/en>
- <https://wandbox.org/>
- <http://coliru.stacked-crooked.com/>
- <http://quick-bench.com/>
- <https://cppinsights.io/>
- <https://ideone.com/>
- <http://cpp.sh/>
- <https://ide.geeksforgeeks.org/>
- <https://www.codechef.com/ide>
- <https://visualstudio.microsoft.com/services/visual-studio-online/>

Usando Editores Colaborativos

La escritura colaborativa es una escritura de códigos de programación en la Web hecha por más de una persona simultáneamente.

Algunos ejemplos de estos servicios son:

- <http://collabedit.com> (edita código, tiene chat, no compila)
- <https://gitduck.com/>
- <https://codeshare.io/>
- <https://www.tutorialspoint.com/codingground.htm>
- <http://ideone.com>
- <https://codebunk.com>
- <https://visualstudio.microsoft.com/services/visual-studio-online/>
- <https://ace.c9.io/build/kitchen-sink.html>
- <https://coderpad.io/>
- <https://peerpad.net/>
- <https://aws.amazon.com/cloud9/>
- <https://codeanywhere.com/>
- <https://stekpad.com/home/>

Algunas de las terminales soportados son:

CentOS, IPython, Lua, MemCached, Mongo DB, MySQL, Node.js, Numpy, Oracle, Octave, PowerShell, PHP, R Programming, Redis, Ruby, SciPy, SymPy, etc.

Algunos de los IDEs soportados son:

Ada (GNAT), Algol68, Angular JS, Assembly, AsciiDoc, AWK, Bash Shell, Befunge, Bootstrap, Brainf**k, C, CSS3, Chipmunk BASIC, Clojure, Cobol, CoffeeScript, ColdFusion, C99 Strict, C++, C++ 0x, C++ 11, C#, Dart, D Programming Language, Embedded C, Erlang, Elixir, Factor, Fantom, Falcon, Fortran-95, Forth, F#, Free Basic, Groovy, GO, Haxe, Haskell, HTML, ilasm, Intercal, Icon, Java, Java 8, Java MySQL, JavaScript, JSP, JQuery, Julia, Korn Shell (ksh), Latex, Lisp, LOLCODE, Lua, Matlab/Octave, Malbolge, Markdown, MathML, Mozart-Oz, Nimrod, Node.JS, Objective-C, OCaml, Pascal, PARI/GP, Pawn, Perl, Perl MySQL, PHP, PHP MySQL, WebView, Pike, Processing.js, p5.js, Prolog, Python-2, Python-3, Python MySQL, Jupyter Notebook, REXX, reStructure, Ruby, Rust, Scala, R Programming, Scheme, Smalltalk, SML/NJ, Simula, SQLite SQL, Tcl, TeX, Unlambda, VB.NET, Verilog, Whitespace, Ya Basic, etc.

Google Colaboratory Integrante de la G Suite for Education de Google permite a los usuarios que pertenezcan a esta Suite (como gran parte de los estudiantes de la UNAM) tener acceso desde el navegador para escribir y ejecutar código de Python (Jupyter), es posible elegir correr nuestro Notebook en una CPU, GPU o en una TPU de forma gratuita. Tiene algunas restricciones, como por ejemplo que una sesión dura 12 hrs, pasado ese tiempo se limpia nuestro ambiente y perdemos las variables y archivos que tengamos almacenados allí.

Es conveniente para principiantes que requieran experimentar con Machine Learning y Deep Learning pero sin recurrir en costos de procesamiento Cloud. Además el ambiente de trabajo ya viene con muchas librerías instaladas y listas para utilizar (como por ejemplo *Tensorflow*, *Scikit-learn*, *Pytorch*, *Keras* y *OpenCV*), ahorrándonos el trabajo de configurar nuestro ambiente de trabajo. Podemos importar nuestros archivos y datos desde *Google Drive*, *GitHub*, etc.

Más información sobre Google Colaboratory en:

<https://colab.research.google.com/notebooks/intro.ipynb>

4 Visualización Gráfica de Datos

La visualización gráfica de datos constituye una disciplina propia dentro del universo de las Ciencias e Ingenierías y en particular en la Ciencia de Datos. Esta práctica ha marcado hitos importantes a lo largo de la historia en la analítica de datos. Desde el gráfico más sencillo hasta el más complejo y refinado, todos ofrecen alto valor tanto al analista, durante su proceso de ciencia de datos, como al usuario final, al cual estamos comunicando una historia basada en datos.

¿Por qué es tan importante la visualización gráfica de los datos?

En la Ciencia de Datos existen muchos tipos diferentes de datos para analizar. Una forma de clasificación de los datos atiende al nivel de estructuración lógica que éstos tienen. Por ejemplo, los datos en formatos similares a hojas de cálculo -aquellos datos que se estructuran en forma de filas y columnas- son datos con una estructura bien definida -o datos estructurados-. Sin embargo, aquellos datos como los 140 caracteres de un Feed de X (Twitter) se consideran datos sin estructura -o desestructurados-.

En medio de estos dos extremos se encuentra toda una gama de grises, que va desde los ficheros delimitados por caracteres especiales (comas, puntos y comas, espacios, etc.) hasta las imágenes o los videos de Youtube. Es evidente que las imágenes y los vídeos solamente cobran sentido humano una vez representadas visualmente. De nada serviría (para un humano) que presentáramos una imagen como una matriz de números que representan una combinación de colores RGB (Red, Green, Blue). En el caso de los datos estructurados, su representación gráfica es necesaria en todas las etapas del proceso de análisis, desde la etapa exploratoria, hasta la presentación final de resultados.

La visualización gráfica de los datos tiene un papel fundamental en todos los estadios del análisis de datos. Existen múltiples aproximaciones sobre cómo realizar un proceso de análisis de datos de forma correcta y completa.

La visualización de datos está en el núcleo del proceso. Es una herramienta básica para el analista o científico de datos que -mediante un proceso iterativo- va transformando y componiendo un modelo lógico de los datos. Apoyándose en la visualización, el analista va descubriendo los secretos enterrados en los datos. La visualización permite de forma rápida:

- Descartar aquellos datos poco representativos o erróneos.

- Identificar aquellas variables que dependen unas de otras y por lo tanto contienen información redundante
- Realizar cortes a los datos para poder observarlos desde diferentes perspectivas.
- Finalmente, comprobar que aquellos modelos, tendencias, predicciones y agrupaciones que hemos aplicado sobre los datos, nos devuelven el resultado esperado.

Herramientas para el análisis visual de datos Tan importante es la visualización gráfica de los datos en todos los ámbitos de la ciencia, ingeniería, negocios, banca, medio ambiente, etc. que existen multitud de herramientas para diseñar, desarrollar y comunicar la visualización gráfica de los datos. Estas herramientas cubren un amplio espectro del público objetivo, desde desarrolladores de Software, hasta científicos de datos pasando por periodistas y profesionales de la comunicación.

Para desarrolladores de Software, existen cientos de librerías y paquetes de Software que contienen miles de tipos de visualizaciones. Los desarrolladores tan solo tienen que cargar estas librerías en sus respectivos entornos de trabajo de programación y parametrizar el tipo de gráfico que deseen generar. El desarrollador tan solo ha de indicar los datos de origen que desea representar, el tipo de gráfico (líneas, barras, etc.) y la parametrización de dicho gráfico (escalas, colores, etiquetas, etc.).

El científico de datos acostumbra a trabajar con un entorno de trabajo de análisis concreto que, normalmente, incluye todos los componentes, entre ellos su motor de análisis visual de los datos. Los entornos más populares, hoy en día, para la ciencia de datos son R y Python, y ambos incluyen librerías nativas para la analítica visual. Quizás la librería más popular y potente en R sea ggplot2, mientras que en Python, matplotlib y Plotly son de las más populares.

Para comunicadores profesionales o personal no técnico de las distintas áreas de negocio (Marketing, Recursos Humanos, Producción, etc.) que necesitan tomar decisiones basadas en datos, existen herramientas -que no son únicamente herramientas de Visual Analytics- con funcionalidades para generar representaciones gráficas de los datos. Herramientas modernas de Business Intelligence de autoservicio como MS Excel, MS Power BI, Qlik,

Tableau, etc. son estupendas herramientas para comunicar los datos sin necesidad de disponer de competencias en programación o codificación.

En definitiva, las herramientas de visualización permiten a todos estos profesionales acceder a los datos de una manera más ágil y sencilla. En un universo donde la cantidad de datos útiles a analizar no deja de crecer, cada vez son más necesarias este tipo de herramientas, que facilitan la obtención de valor procedente de los datos y, con ello, la toma de decisiones relativas al presente y al futuro de nuestro negocio o actividad. Se ponen a disposición los fuentes de los gráficos mostrados a continuación y otros más en la liga:

[Gráficos](#)

4.1 Python

El lenguaje Python permite realizar una amplia variedad de visualizaciones de datos en 2D, 3D y animaciones. La misma se realiza mediante la ayuda de módulos (como math o numpy) que están destinados a brindar herramientas de visualización.

En la lista que sigue se presentan algunos de los módulos más usados. El objetivo no es aprenderlos, sino saber que existen varias opciones y disminuir un poco la confusión que se puede producir al principio al buscar ayuda y bibliografía:

- Matplotlib
- Seaborn
- Plotly
- Bokeh
- Altair
- Pygal
- Pandas
- Plotnine

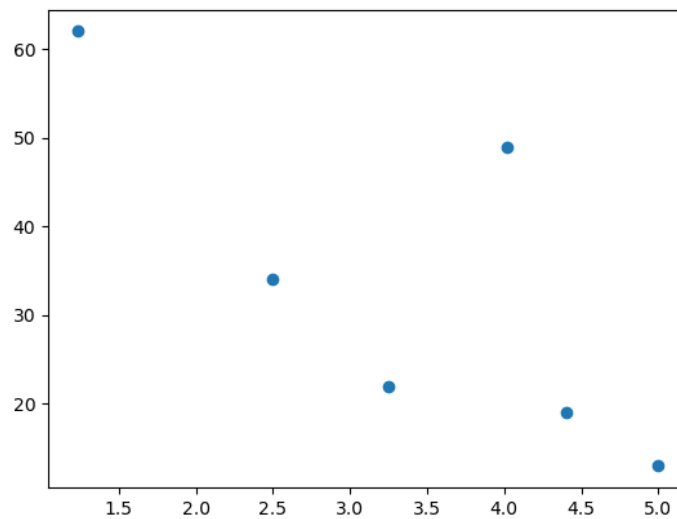
La elección de una de estas herramientas para hacer un gráfico depende del contexto, para qué se quiere hacer el gráfico, dónde se va a mostrar y a partir

de qué datos. Así, por ejemplo Plotly, Bokeh y Altair devuelven gráficos en HTML para ser mostrados en páginas Web, Pygal genera gráficos vectoriales y Pandas gráfica datos guardados en cierto tipo especial de estructura.

Por ejemplo:

```
import matplotlib.pyplot as plt
precio = [2.50, 1.23, 4.02, 3.25, 5.00, 4.40]
ventas_por_dia = [34, 62, 49, 22, 13, 19]
plt.scatter(precio, ventas_por_dia)
plt.savefig('Grafica.pdf')
plt.savefig('Grafica.png')
plt.show()
```

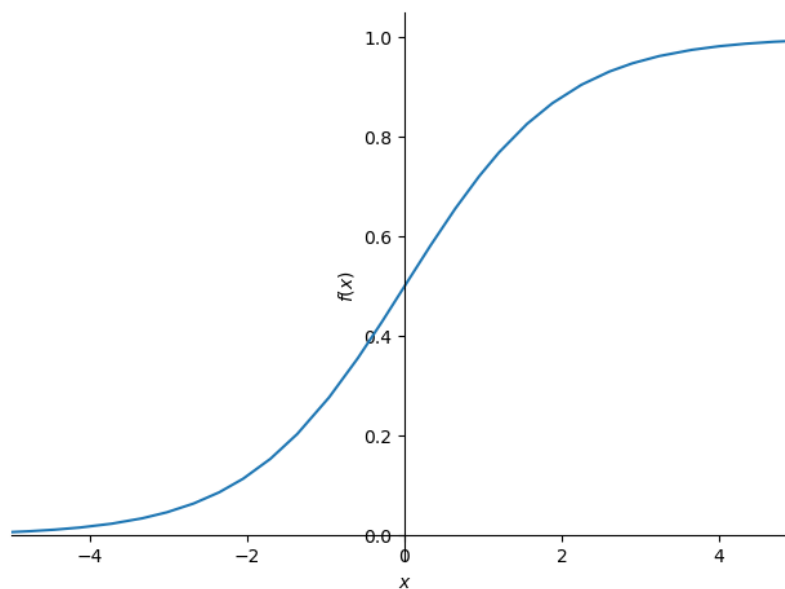
genera el siguiente gráfico (además lo graba en formato PDF y PNG):



Otro ejemplo:

```
import sympy
graf = sympy.plot("1/(1+exp(-x))", xlim=(-5,5));
graf.save('Grafica.pdf')
graf.save('Grafica.png')
```

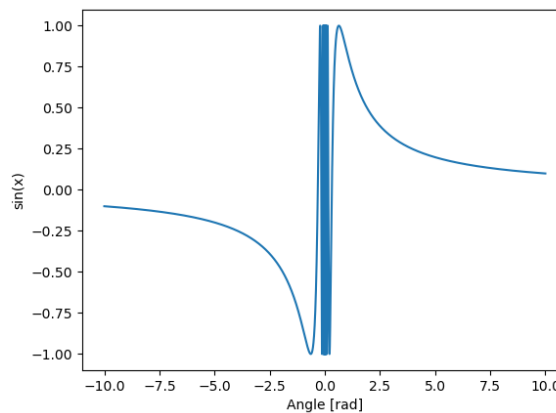
genera el siguiente gráfico (además lo graba en formato PDF y PNG):



Otro ejemplo:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-10, 10, 100000)
y = np.sin(1 / x)
# se graban los datos
np.savetxt('valores.txt',(x,y))
# se leen los datos
xx,yy = np.loadtxt('valores.txt')
# se grafica
plt.figure()
plt.plot(xx, yy)
plt.xlabel("Angle [rad]")
plt.ylabel("sin(x)")
plt.axis("tight")
plt.show()
```

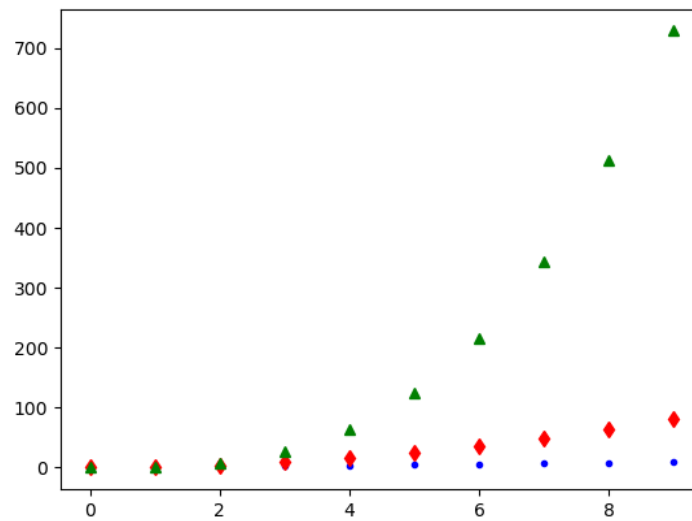
genera el siguiente gráfico (además de grabar y leer los valores de las variables x e y):



Otro ejemplo:

```
from pylab import *
import matplotlib.pyplot as plt
x = arange(10.) # array de floats, de 0.0 a 9.0
x2 = x**2       # definimos el array x2
x3 = x**3       # definimos el array x3
# dibujamos tres curvas en el mismo gráfico y figura
plot(x, x, 'b.', x, x2, 'rd', x, x3, 'g^')
show()
```

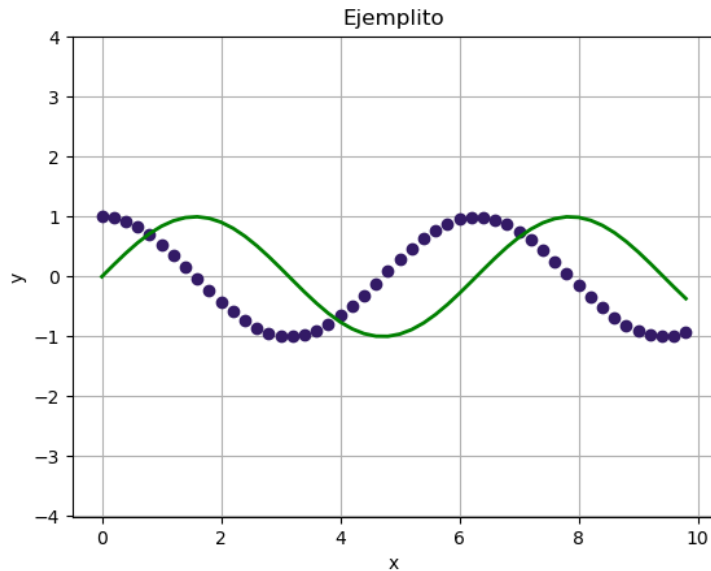
genera el siguiente gráfico:



Otro ejemplo:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(0,10,0.2)
y1 = np.cos(x)
y2 = np.sin(x)
plt.plot(x,y1,'o',linewidth=3,color=(0.2,0.1,0.4))
plt.plot(x,y2,'-',linewidth=2,color='g')
plt.grid()
plt.axis('equal')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Ejemplito')
plt.show()
```

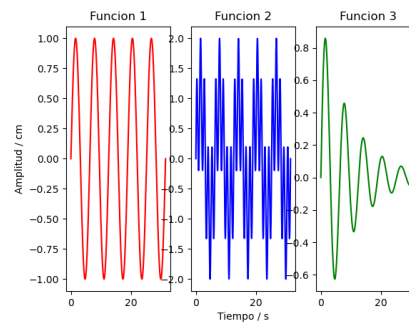
genera el siguiente gráfico:



Otro ejemplo:

```
from pylab import *
import matplotlib.pyplot as plt
x = linspace(0, 10*pi, 800) # array de valores a representar
def f1(x):
    return sin(x)
def f2(x):
    return sin(x) + sin(5.0*x)
def f3(x):
    return sin(x) * exp(-x/10.)
subplot(131)
p1, = plot(x,f1(x),'r-')
ylabel('Amplitud / cm')
title('Funcion 1')
subplot(132)
p2, = plot(x,f2(x),'b-')
xlabel('Tiempo / s')
title('Funcion 2')
subplot(133)
p3, = plot(x, f3(x),'g-')
title('Funcion 3')
show()
```

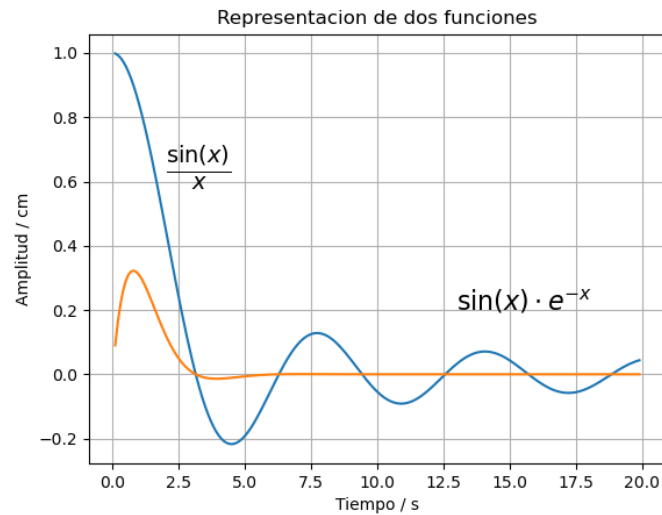
genera el siguiente gráfico:



Otro ejemplo:

```
from pylab import *
import matplotlib.pyplot as plt
t = arange(0.1, 20, 0.1)
y1 = sin(t)/t
y2 = sin(t)*exp(-t)
p1, p2 = plot(t, y1, t, y2)
texto1 = text(2, 0.6, r'$\frac{\sin(x)}{x}$', fontsize=20)
texto2 = text(13, 0.2, r'$\sin(x) \cdot e^{-x}$', fontsize=16)
grid()
title('Representacion de dos funciones')
xlabel('Tiempo / s')
ylabel('Amplitud / cm')
show()
```

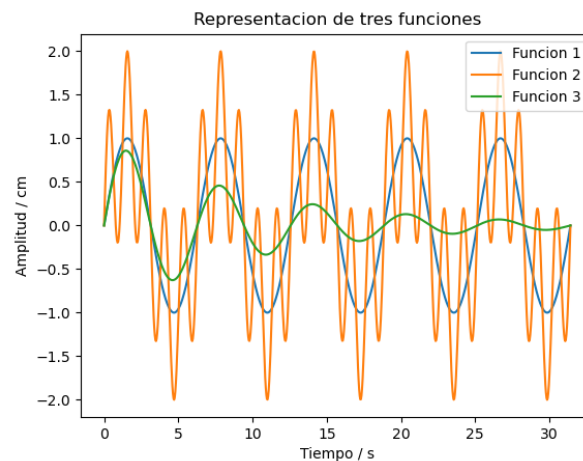
genera el siguiente gráfico:



Otro ejemplo:

```
from pylab import *
import matplotlib.pyplot as plt
def f1(x):
    return sin(x)
def f2(x):
    return sin(x) + sin(5.0*x)
def f3(x):
    return sin(x) * exp(-x/10.)
x = linspace(0, 10*pi, 800)
p1, p2, p3 = plot(x, f1(x), x, f2(x), x, f3(x))
legend(('Funcion 1', 'Funcion 2', 'Funcion 3'),
prop = {'size': 10}, loc='upper right')
xlabel('Tiempo / s')
ylabel('Amplitud / cm')
title('Representacion de tres funciones')
figure(figsize=(12, 5))
show()
```

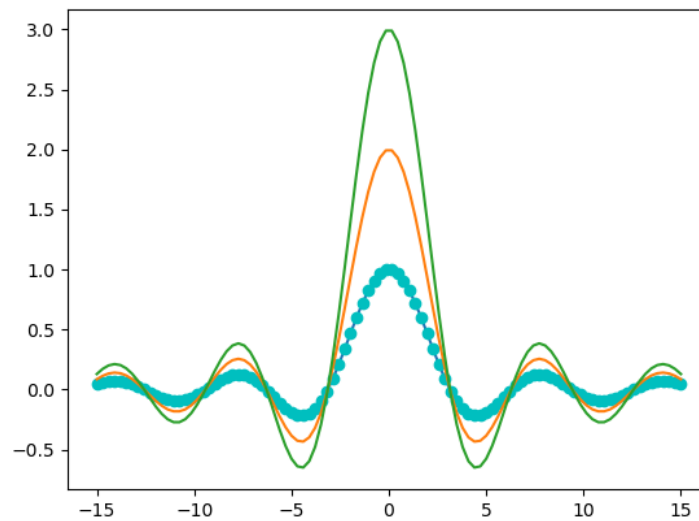
genera el siguiente gráfico:



Otro ejemplo:

```
import pylab
import numpy
x = numpy.linspace(-15, 15, 100)
y = numpy.sin(x) / x
pylab.plot(x, y)
pylab.plot(x, y, "co")
pylab.plot(x, 2 * y, x, 3 * y)
pylab.show()
```

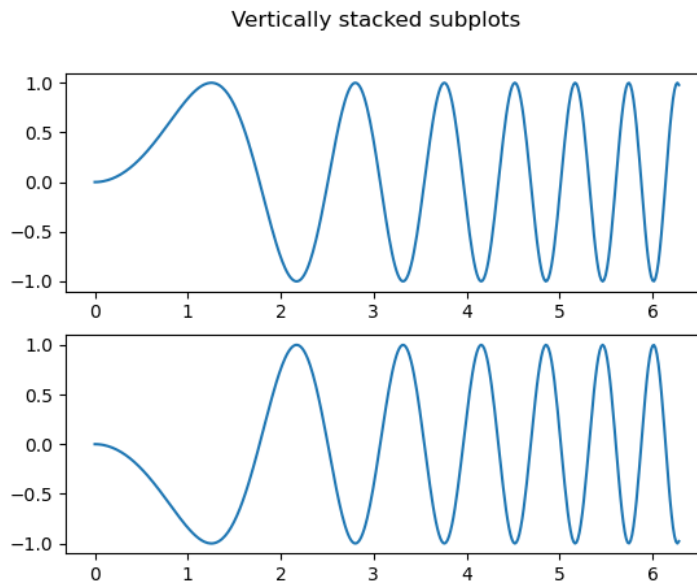
genera el siguiente gráfico:



Otro ejemplo:

```
import matplotlib.pyplot as plt
import numpy as np
def multiple_plots():
    # Some example data to display
    x = np.linspace(0, 2 * np.pi, 400)
    y = np.sin(x ** 2)
    fig, axs = plt.subplots(2)
    fig.suptitle('Vertically stacked subplots')
    axs[0].plot(x, y)
    axs[1].plot(x, -y)
    plt.show()
multiple_plots()
```

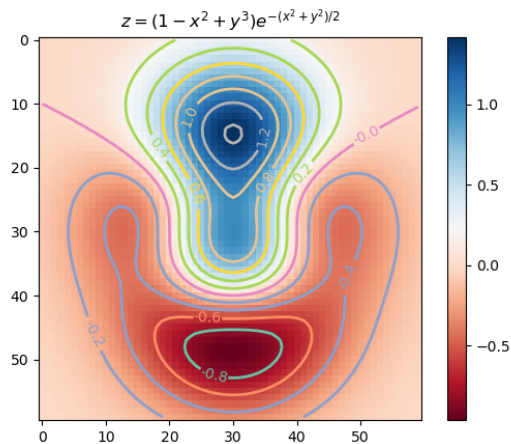
genera el siguiente gráfico:



Otro ejemplo:

```
from numpy import exp, arange
from pylab import meshgrid, cm, imshow, contour, clabel,
colorbar, axis, title, show
def z_func(x, y):
    return (1 - (x ** 2 + y ** 3)) * exp(-(x ** 2 + y ** 2) / 2)
x = arange(-3.0, 3.0, 0.1)
y = arange(-3.0, 3.0, 0.1)
X, Y = meshgrid(x, y)
Z = z_func(X, Y)
im = imshow(Z, cmap=cm.RdBu)
cset = contour(Z, arange(-1, 1.5, 0.2), linewidths=2, cmap=cm.Set2)
clabel(cset, inline=True, fmt="%1.1f", fontsize=10)
colorbar(im)
title("$z=(1-x^2+y^3) e^{-{(x^2+y^2)/2}}$")
show()
```

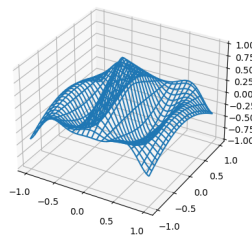
genera el siguiente gráfico:



Otro ejemplo:

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np
import time
def generate(X, Y, phi):
    R = 1 - np.sqrt(X ** 2 + Y ** 2)
    return np.cos(2 * np.pi * X + phi) * R
fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")
xs = np.linspace(-1, 1, 50)
ys = np.linspace(-1, 1, 50)
X, Y = np.meshgrid(xs, ys)
ax.set_zlim(-1, 1)
wframe = None
tstart = time.time()
for phi in np.linspace(0, 180.0 / np.pi, 100):
    if wframe:
        ax.collections.remove(wframe)
    Z = generate(X, Y, phi)
    wframe = ax.plot_wireframe(X, Y, Z, rstride=2, cstride=2)
    plt.pause(0.001)
```

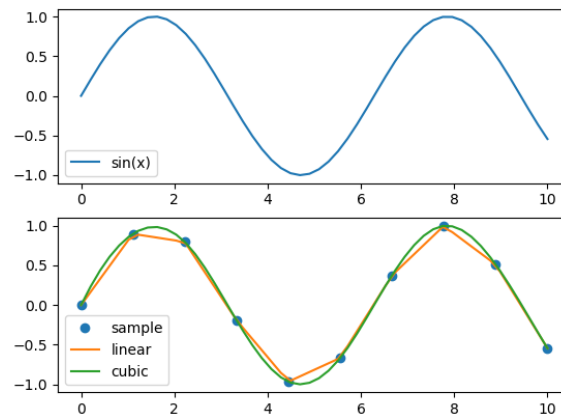
genera el siguiente animación:



Otro ejemplo:

```
import scipy.interpolate as sp
import numpy
import pylab
xx = numpy.linspace(0, 10, 50)
yy = numpy.sin(xx)
x = numpy.linspace(0, 10, 10)
y = numpy.sin(x)
fl = sp.interp1d(x, y, kind='linear')
fc = sp.interp1d(x, y, kind='cubic')
xnew = numpy.linspace(0, 10, 50)
pylab.subplot(211)
pylab.plot(xx, yy)
pylab.legend(['sin(x)'], loc='best')
pylab.subplot(212)
pylab.plot(x, y, 'o', xnew, fl(xnew), xnew, fc(xnew))
pylab.legend(['sample', 'linear', 'cubic'], loc='lower left')
pylab.show()
```

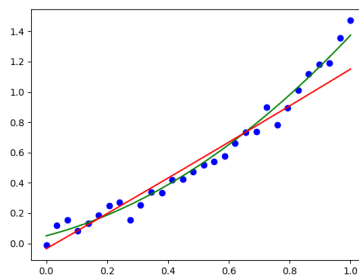
genera el siguiente gráfico:



Otro ejemplo:

```
from pylab import *
from numpy import *
from numpy.random import normal
from scipy.optimize import fmin
# parametric function, x independent variable, c parameters
fp = lambda c, x: c[0]+c[1]*x+c[2]*x*x
real_p = rand(3)
# error function to minimize
e = lambda p, x, y: (abs((fp(p,x)-y))).sum()
# generating data with noise
n = 30
x = linspace(0,1,n)
y = fp(real_p,x) + normal(0,0.05,n)
# fitting the data with fmin
p0 = rand(3) # initial parameter value
p = fmin(e, p0, args=(x,y))
print ('estimator parameters: ', p)
print ('real parameters: ', real_p)
xx = linspace(0,1,n*3)
plot(x,y,'bo', xx,fp(real_p,xx),'g', xx, fp(p,xx),'r')
show()
```

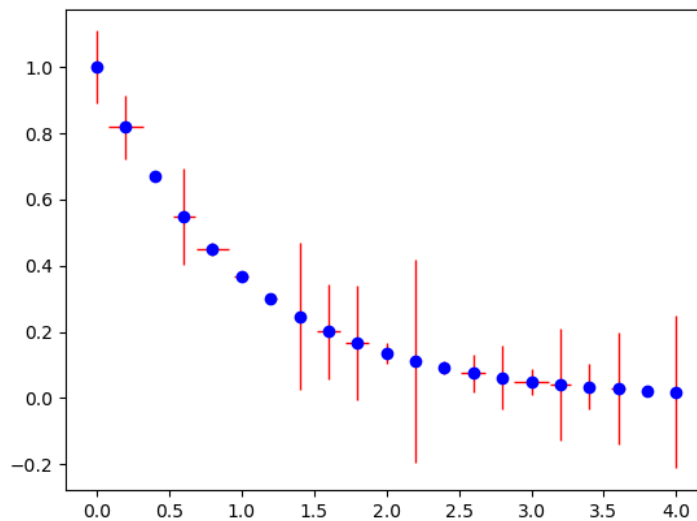
genera el siguiente gráfico:



Otro ejemplo:

```
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as npr
x=np.linspace(0,4,21)
y=np.exp(-x)
xe=np.abs(0.08*npr.randn(len(x)))
ye=np.abs(0.1*npr.randn(len(y)))
plt.errorbar(x,y,fmt='bo',lw=2,xerr=xe,yerr=ye,ecolor='r',elinewidth=1)
plt.show()
```

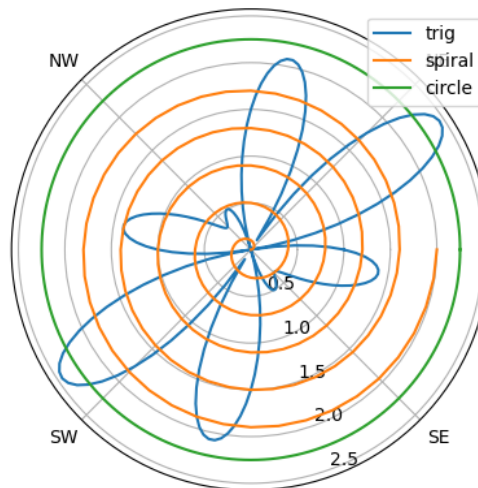
genera el siguiente gráfico:



Otro ejemplo:

```
import numpy as np
import matplotlib.pyplot as plt
theta=np.linspace(0,2*np.pi,201)
r1=np.abs(np.cos(5.0*theta) - 1.5*np.sin(3.0*theta))
r2=theta/np.pi
r3=2.25*np.ones_like(theta)
plt.polar(theta, r1,label='trig')
plt.polar(5*theta, r2,label='spiral')
plt.polar(theta, r3,label='circle')
plt.thetagrids(np.arange(45,360,90), ('NE','NW','SW','SE'))
plt.rgrids((0.5,1.0,1.5,2.0,2.5),angle=290)
plt.legend(loc='best')
plt.show()
```

genera el siguiente gráfico:

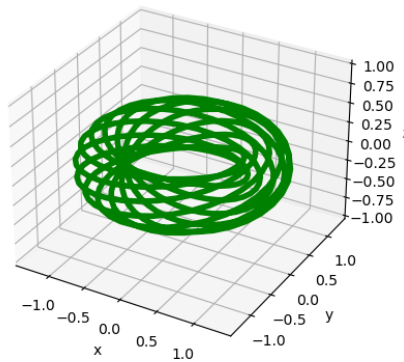


Otro ejemplo:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
theta=np.linspace(0,2*np.pi,401)
a, m, n = 0.3, 11, 9
x=(1+a*np.cos(n*theta))*np.cos(m*theta)
y=(1+a*np.cos(n*theta))*np.sin(m*theta)
z=a*np.sin(n*theta)
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.plot(x,y,z,'g',linewidth=4)
ax.set_zlim3d(-1.0,1.0)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.set_title('Una espiral como curva parametrica', weight='bold',size=16)
plt.show()
```

genera el siguiente gráfico:

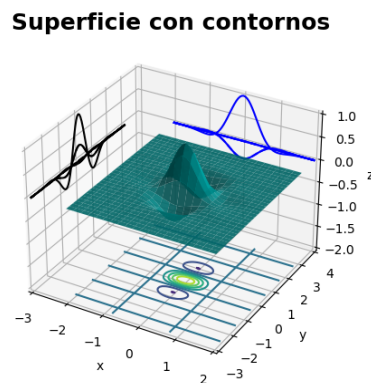
Una espiral como curva parametrica



Otro ejemplo:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
xx, yy=np.mgrid[-2:2:81j, -3:3:91j]
zz=np.exp(-2*xx**2-yy**2)*np.cos(2*xx)*np.cos(3*yy)
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.plot_surface(xx,yy,zz,rstride=4,cstride=3,color='c',alpha=0.9)
ax.contour3D(xx,yy,zz,zdir='x',offset=-3.0,colors='black')
ax.contour3D(xx,yy,zz,zdir='y',offset=4.0,colors='blue')
ax.contour3D(xx,yy,zz,zdir='z',offset=-2.0)
ax.set_xlim3d(-3.0,2.0)
ax.set_ylim3d(-3.0,4.0)
ax.set_zlim3d(-2.0,1.0)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.set_title('Superficie con contornos',weight='bold',size=18)
plt.show()
```

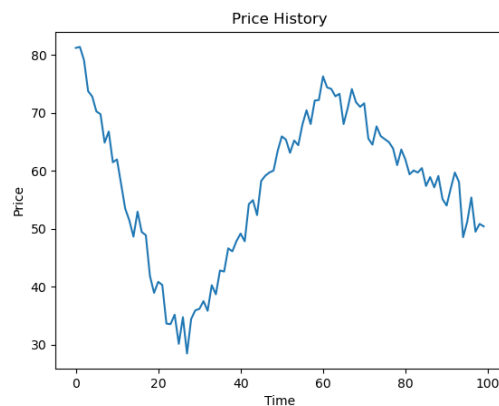
genera el siguiente gráfico:



Otro ejemplo:

```
import numpy as np
import matplotlib.pyplot as plt
prices = np.full(100, fill_value=np.nan)
prices[[0, 25, 60, -1]] = [80.0, 30.0, 75.0, 50.0]
x = np.arange(len(prices))
is_valid = ~np.isnan(prices)
prices = np.interp(x=x, xp=x[is_valid], fp=prices[is_valid])
prices += np.random.randn(len(prices)) * 2
mn = np.argmin(prices)
mx = mn + np.argmax(prices[mn:])
kwargs = {"markersize": 12, "linestyle": ""}
fig, ax = plt.subplots()
ax.plot(prices)
ax.set_title("Price History")
ax.set_xlabel("Time")
ax.set_ylabel("Price")
ax.plot(mn, prices[mn], color="green", **kwargs)
ax.plot(mx, prices[mx], color="red", **kwargs)
plt.show()
```

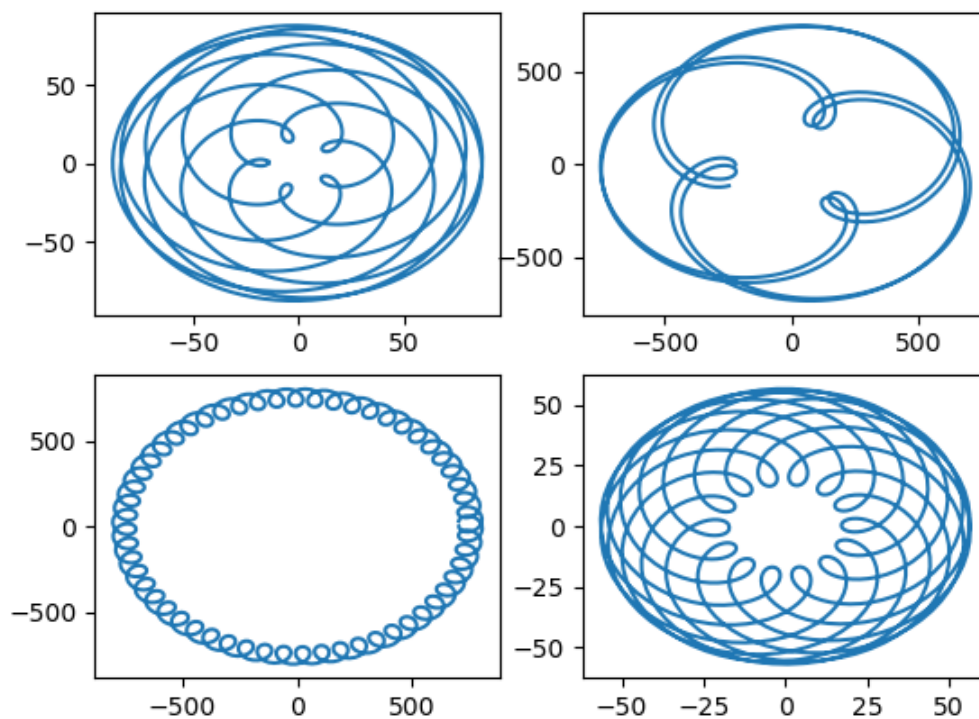
genera el siguiente gráfico:



Otro ejemplo:

```
import matplotlib.pyplot as plot
from numpy import linspace, sin, cos, pi
period = {
    'mercury' : 87.96926,
    'venus' : 224.7008,
    'earth' : 365.25636,
    'mars' : 686.97959,
    'ceres' : 1680.22107,
    'jupiter' : 4332.8201,
    'saturn' : 10755.699,
    'uranus' : 20687.153,
    'neptune' : 60190.03
}
dist = lambda T : T**(2/3) # Kepler
def plot_orbit(ax, planet1, planet2, periods=10):
    T1 = period[planet1]
    T2 = period[planet2]
    d1 = dist(T1)
    d2 = dist(T2)
    theta = linspace(0, 2*pi*periods, 1000)
    x = d1*cos(T2*theta/T1) - d2*cos(theta)
    y = d1*sin(T2*theta/T1) - d2*sin(theta)
    ax.plot(x, y)
fig=plot.figure()
ax = fig.add_subplot(2,2,1)
plot_orbit(ax, "venus", "earth", 8)
ax = fig.add_subplot(2,2,2)
plot_orbit(ax, "jupiter", "saturn", 4)
ax = fig.add_subplot(2,2,3)
plot_orbit(ax, "uranus", "earth", 57)
ax = fig.add_subplot(2,2,4)
plot_orbit(ax, "mercury", "venus", 9)
plot.show()
```

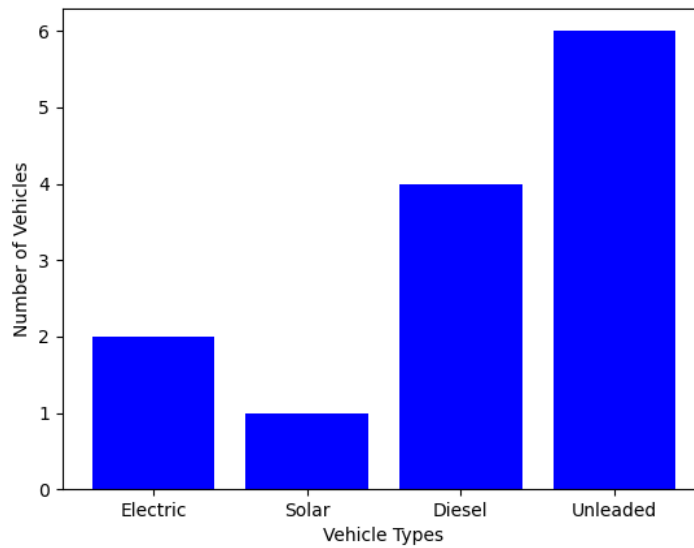
genera el siguiente gráfico:



Otro ejemplo:

```
import matplotlib.pyplot as plt
def bar_chart(numbers, labels, pos):
    plt.bar(pos, numbers, color="blue")
    plt.xticks(ticks=pos, labels=labels)
    plt.xlabel("Vehicle Types")
    plt.ylabel("Number of Vehicles")
    plt.show()
numbers = [2, 1, 4, 6]
labels = ["Electric", "Solar", "Diesel", "Unleaded"]
pos = list(range(4))
bar_chart(numbers, labels, pos)
```

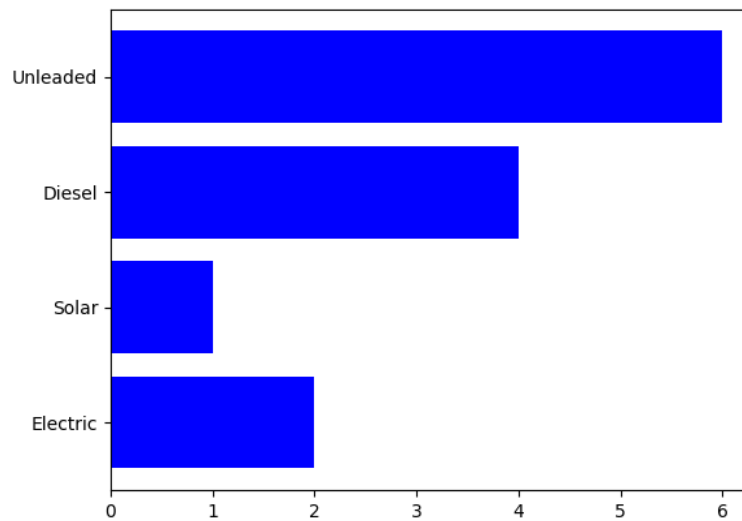
genera el siguiente gráfico:



Otro ejemplo:

```
import matplotlib.pyplot as plt
def bar_charth(numbers, labels, pos):
    plt.barh(pos, numbers, color="blue")
    plt.yticks(ticks=pos, labels=labels)
    plt.show()
numbers = [2, 1, 4, 6]
labels = ["Electric", "Solar", "Diesel", "Unleaded"]
pos = list(range(4))
bar_charth(numbers, labels, pos)
```

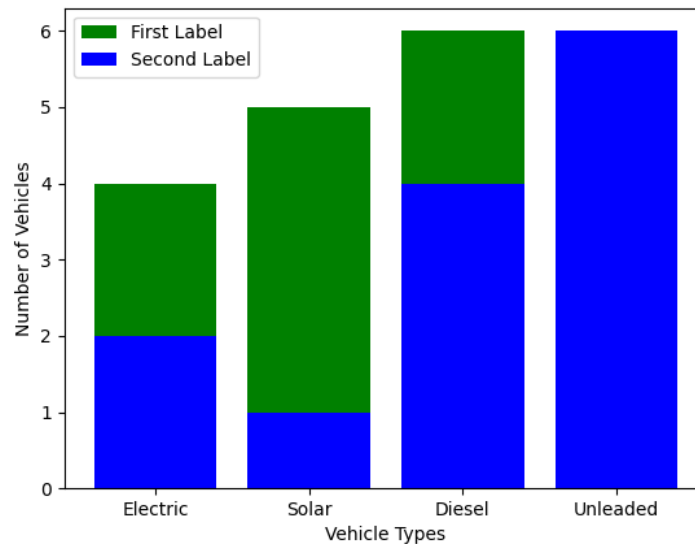
genera el siguiente gráfico:



Otro ejemplo:

```
import matplotlib.pyplot as plt
def bar_chart(numbers, labels, pos):
    plt.bar(pos, [4, 5, 6, 3], color="green")
    plt.bar(pos, numbers, color="blue")
    plt.xticks(ticks=pos, labels=labels)
    plt.xlabel("Vehicle Types")
    plt.ylabel("Number of Vehicles")
    plt.legend(["First Label", "Second Label"], loc="upper
left")
    plt.show()
numbers = [2, 1, 4, 6]
labels = ["Electric", "Solar", "Diesel", "Unleaded"]
pos = list(range(4))
bar_chart(numbers, labels, pos)
```

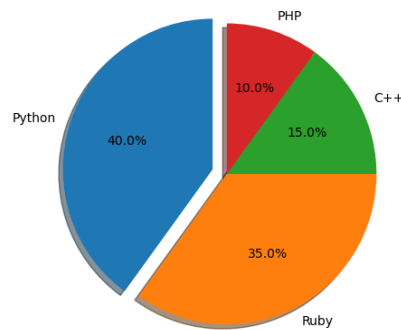
genera el siguiente gráfico:



Otro ejemplo:

```
import matplotlib.pyplot as plt
def pie_chart():
    numbers = [40, 35, 15, 10]
    labels = ["Python", "Ruby", "C++", "PHP"]
    # Explode the first slice (Python)
    explode = (0.1, 0, 0, 0)
    fig1, ax1 = plt.subplots()
    ax1.pie(
        numbers,
        explode=explode,
        labels=labels,
        shadow=True,
        startangle=90,
        autopct="%1.1f%%",
    )
    ax1.axis("equal")
    plt.show()
pie_chart()
```

genera el siguiente gráfico:



Otro ejemplo:

```
import folium
import pandas as pd
eco_footprints = pd.read_csv("footprint.csv")
max_eco_footprint = eco_footprints["Ecological footprint"].max()
political_countries_url = (
    "http://geojson.xyz/naturalearth-3.3.0/ne_50m_admin_0_countries.geojson"
)
m = folium.Map(location=(30, 10), zoom_start=3, tiles="cartodb
positron")
folium.Choropleth(
    geo_data=political_countries_url,
    data=eco_footprints,
    columns=("Country/region", "Ecological footprint"),
    key_on="feature.properties.name",
    bins=(0, 1, 1.5, 2, 3, 4, 5, 6, 7, 8, max_eco_footprint),
    fill_color="RdYlGn_r",
    fill_opacity=0.8,
    line_opacity=0.3,
    nan_fill_color="white",
    legend_name="Ecological footprint per capita",
    name="Countries by ecological footprint per capita",
).add_to(m)
folium.LayerControl().add_to(m)
m.save("M7.html")
```

genera el siguiente gráfico:

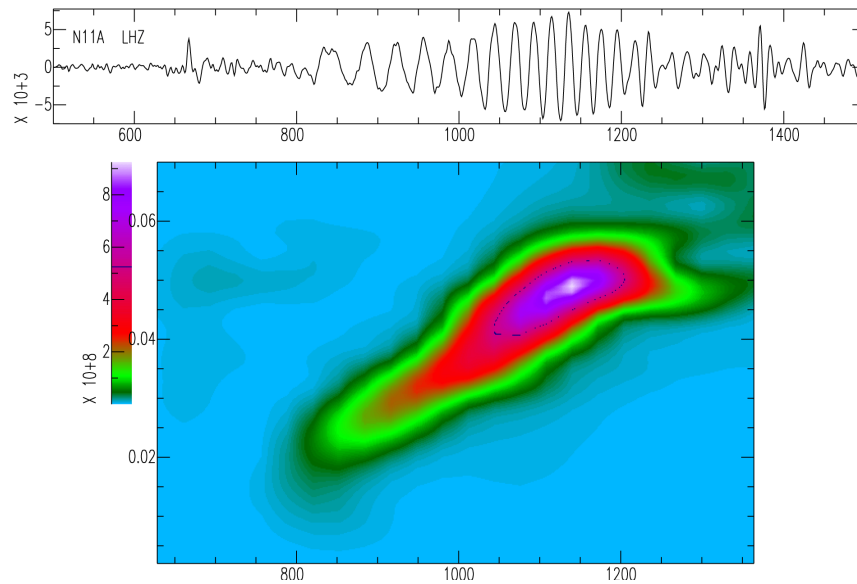
Existe una gran cantidad herramientas para generar gráficos 2D, 3D y videos que se pueden usar desde lenguajes de programación, en particular para Python hay una gran variedad, entre las utilidades destacan:

- <https://mpmath.org/>
- <https://plotly.com/python/>
- <https://github.com/szabolcsdombi/zengl>
- <https://towardsdatascience.com/interactive-animated-visualization-db91d1c858ad>
- <https://docs.entthought.com/mayavi/mayavi/>
- <https://docs.bokeh.org/en/latest/index.html>
- <https://matplotlib.org/>
- <https://pyopengl.sourceforge.net/>
- https://pandas.pydata.org/pandas-docs/stable/user_guide/visualization.html
- <https://pythonplot.com/>
- <http://www.pyqtgraph.org/>
- <http://seaborn.pydata.org/>
- <https://github.com/sepandhaghighi/samila>
- <https://github.com/paulcbogdan/NiChord>
- <https://pypi.org/project/Shapely/>
- <https://tech.marksblogg.com/pretty-maps-in-python.html>
- <https://www.manim.community/>
- <https://plotnine.readthedocs.io/en/stable/>
- <https://www.pythonguis.com/tutorials/pyqt6-plotting-pyqtgraph/>
- <https://pbpython.com/visualization-tools-1.html>

- <https://github.com/moshi4/pyCirclize>
- <https://mathspp.com/blog/til/xkcd-plots>
- <https://github.com/ResidentMario/geoplot>

4.2 SAC

SAC (Código de análisis sísmico) es un programa interactivo de propósito general diseñado para el estudio de señales secuenciales, especialmente datos de series temporales. Se ha puesto énfasis en las herramientas de análisis utilizadas por los sismólogos investigadores en el estudio detallado de eventos sísmicos. Las capacidades de análisis incluyen operaciones aritméticas generales, transformadas de Fourier, tres técnicas de estimación espectral, filtrado IIR y FIR, apilamiento de señales, diezmado, interpolación, correlación y selección de fase sísmica. SAC también contiene una amplia capacidad gráfica.

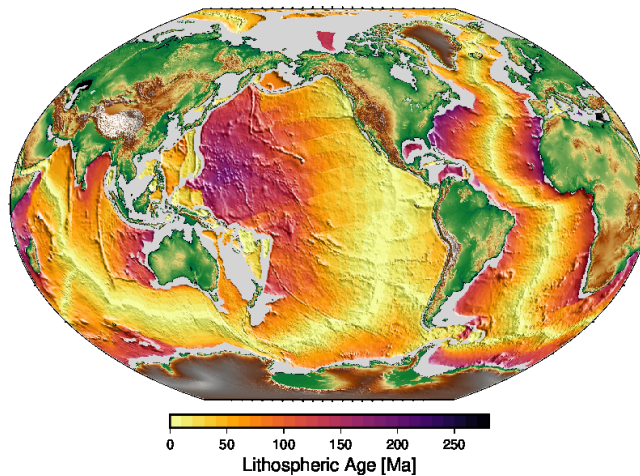


SAC fue desarrollado en el Laboratorio Nacional Lawrence Livermore y tiene derechos de autor de la Universidad de California. Actualmente es desarrollado y mantenido por un pequeño grupo de desarrolladores que trabajan en cooperación con IRIS. Las versiones binarias están disponibles para Intel Mac y Linux, pero SAC se puede construir a partir del código fuente para otros sistemas operativos de computadora. El código fuente está escrito en C.

4.3 Sistema de Información Geográfica

Un sistema de información geográfica (SIG) es un marco para recopilar, gestionar y analizar datos. Adaptado en la ciencia de la geografía, SIG integra muchos tipos de datos. Analiza la ubicación espacial y organiza capas de información en visualizaciones utilizando mapas y escenas en 3D. Con esta capacidad única, SIG revela conocimientos más profundos sobre los datos, como patrones, relaciones y situaciones, lo que ayuda a los usuarios a tomar decisiones más inteligentes.

Uno de los problemas más complejos de resolver cuando se trabaja con SIG, es la obtención de salidas cartográficas de suficiente calidad. No se trata de obtener mapas como los que producen los organismos cartográficos, sino tan sólo de disponer de un sistema lo suficientemente flexible como para crear un mapa que une los objetivos de representación y simplificación, así como los criterios estéticos, que se esperan de un mapa que va a publicarse, como tal o dentro de un trabajo científico.



Las salidas gráficas de un SIG se caracterizan por su inmediatez y por ser una representación para el usuario de los modelos de datos que utiliza el programa, asumiendo a este usuario los suficientes conocimientos de SIG, del área de estudio y de las variables representadas, como para que baste un representación cruda. Sin embargo un mapa es para uso de terceros a los que no debemos suponer ningún conocimiento a priori y que, por tanto, van a necesitar ayuda para interpretar el mapa (escalas, leyendas, etc.). Cons-

cientes de este problema, los desarrolladores de SIG han implementado una serie de herramientas de maquetación de mapas.

Los sistema de información geográfica debieran permitir:

- Crear, compartir y utilizar mapas inteligentes
- Compilar información geográfica
- Crear y administrar bases de datos geográficas
- Resolver problemas con el análisis espacial
- Crear aplicaciones basadas en mapas
- Dar a conocer y compartir información mediante la geografía y la visualización

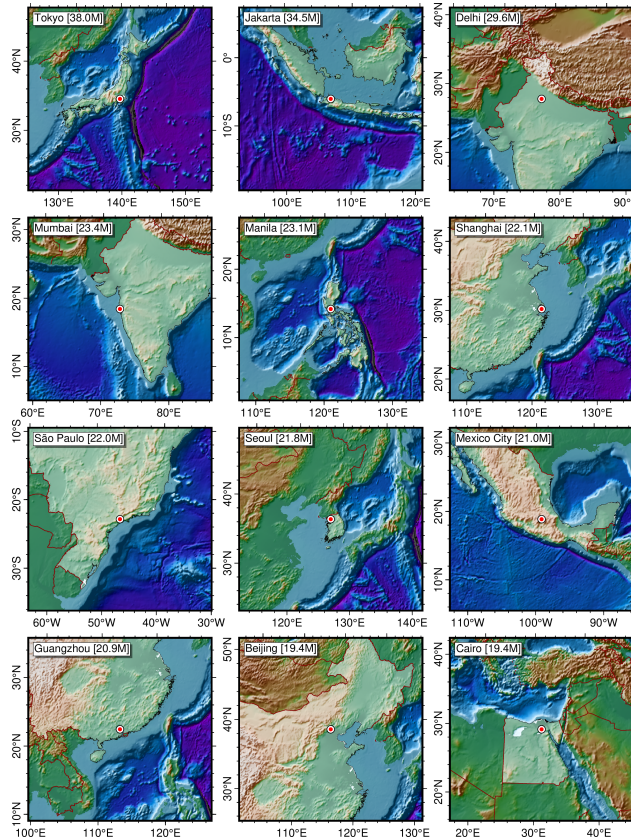
Existe una gran cantidad herramientas para que se pueden usar desde lenguajes de programación, entre las utilidades destacan:

- <https://geopy.readthedocs.io/en/stable/>

GMT **GMT** (Generic Mapping Tools) pone a disposición del usuario un conjunto de módulos orientados a la producción de cartografía a partir de datos codificados según los modelos lógicos (Raster, vectorial, lista de puntos) habituales en los SIG. Los datos raster se almacenan por defecto en formato propio (GMT/ netCDF) pero también se admiten ficheros binarios con datos en coma flotante, enteros, bytes o bits. Los datos vectoriales se almacenan como simples pares de coordenadas, no se incluye por tanto información topológica pero de hecho no tiene sentido incluirla si el resultado que espera obtenerse es un mapa para imprimir en papel.

El manejo de estos módulos en línea de comandos y la posibilidad de combinarlos (entre sí y con otras herramientas LINUX/UNIX) así como el elevado número de opciones de cada uno de ellos, convierte GMT en un entorno de maquetación de mapas extremadamente flexible. La contrapartida de esta flexibilidad es un lenguaje que puede llegar a ser bastante complejo y hermético.

Además de los módulos estrictamente destinados a la generación de mapas, GMT dispone de módulos de álgebra de mapas y de producción de



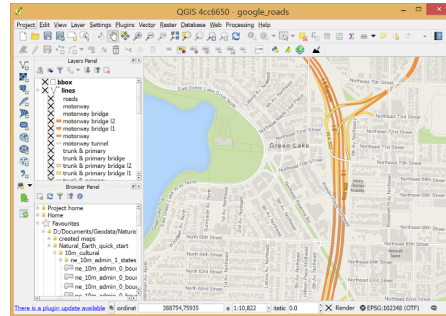
gráficos más generales (histogramas, etc.). GMT tiene capacidad de trabajar con lenguajes como: MATLAB/Octave, Julia o Python.

instalación en linux:

```
# apt install gmt gmt-gshhg-full
```

QGIS **QGIS** es un Sistema de Información Geográfica (SIG) de Código Abierto licenciado bajo GNU - General Public License . QGIS es un proyecto oficial de Open Source Geospatial Foundation (OSGeo). Corre sobre Linux, Unix, Mac OSX, Windows y Android y soporta numerosos formatos y funcionalidades de datos vector, datos ráster y bases de datos.

ArcGIS **ArcGIS** es un completo sistema que permite recopilar, organizar, administrar, analizar, compartir y distribuir información geográfica. Es una



plataforma para crear y utilizar sistemas de información geográfica (SIG), ArcGIS es utilizada por personas de todo el mundo para poner el conocimiento geográfico al servicio de los sectores del gobierno, la empresa, la ciencia, la educación y los medios. ArcGIS permite publicar la información geográfica para que esté accesible para cualquier usuario. El sistema está disponible en cualquier lugar a través de navegadores Web, dispositivos móviles como Smartphones y equipos de escritorio.

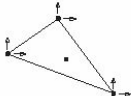
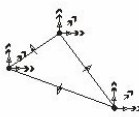
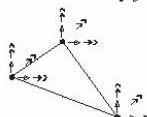
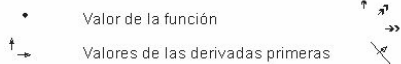


4.4 Generar Mallas con GMSH

GMSH es un generador para mallar tanto sus superficies como sus volúmenes para Elementos Finitos de código abierto con un motor integrado tipo CAD y un post-procesador. Su diseño tiene como fin proveer una herramienta rápida, ligera y amigable con un editor paramétrico y una gran capacidad de visualización avanzada. GMSH está organizado en cuatro módulos: geometría, malla, motor de cálculo, y post-proceso. La introducción de datos e instrucciones a cualquiera de los módulos puede hacerse ya sea de manera interactiva usando la interfaz gráfica, mediante archivos de texto ASCII empleando el

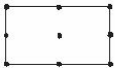
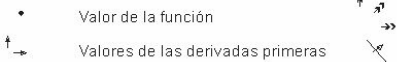
lenguaje propio de GMSH ó a través de la interfaz de programación de aplicaciones C, C++, Python, Julia, Fortran, etc.

Tipos de elementos finitos más comunes

Geometria Grados de libertad Σ	# gdl	Espacio de funciones	Continuidad del espacio MEF	Geometria Grados de libertad Σ	# gdl	Espacio de funciones	Continuidad del espacio MEF
	3	$P_1(K)$	C^0		6	$P_2(K)$	C^0
	10	$P_3(K)$	C^0		10	$P_3(K)$	C^0
	21	$P_5(K)$	C^1		18	$P_5(K)$	C^1
							
Valor de la función				Valores de las derivadas segundas			
Valores de las derivadas primeras				Valor de la derivada normal al lado			

Mallado. Se define como mallado la discretización de una línea, superficie o volumen en porciones de tamaño finito. Las porciones, además de tener un tamaño característico varias veces menor que el espacio discretizado, serán entidades geométricas elementales como los triángulos o cuadriláteros en dos dimensiones o los tetraedros o prismas en tres dimensiones. Esta discretización en estructuras más elementales es esencial para la resolución de ecuaciones en derivadas parciales en dominios arbitrarios por el método de volúmenes finitos (FVM) o el método de elementos finitos (FEM).

Tipos de elementos finitos más comunes (cont)

Geometría Grados de libertad Σ	# gdl	Espacio de funciones	Continuidad del espacio MEF	Geometría Grados de libertad Σ	# gdl	Espacio de funciones	Continuidad del espacio MEF
	4	$Q_1(K)$	C^0		9	$Q_2(K)$	C^0
	16	$Q_3(K)$	C^0		2	$P_1(K)$	C^0
	3	$P_2(K)$	C^0		4	$P_3(K)$	C^1
	4	$P_1(K)$	C^0		10	$P_2(K)$	C^0
							

El proceso de mallado más habitual es el de discretizar sucesivamente entidades de mayor dimensión como si de un problema de contorno se tratara. Por ejemplo, si queremos mallar un cubo primero definiremos los puntos de control en cada una de sus doce aristas. Luego discretizaremos en porciones elementales cada uno de sus seis lados y finalmente discretizaremos el volumen. Para entender mejor este proceso, y con anterioridad a entender la sintaxis de los archivos GMSH presentamos este ejemplo de mallado de un cubo con tetraedros. El primer paso es definir los puntos básicos de la geometría.

Mallas estructuradas Una malla estructurada es una malla que puede ser transformada, mediante una transformación biyectiva, a una cuadrícula. Esto significa que el problema podría ser resuelto en una malla uniforme y rectangular y devuelto a la geometría original sólo conociendo el jacobiano de la transformación porque este jacobiano se puede invertir.

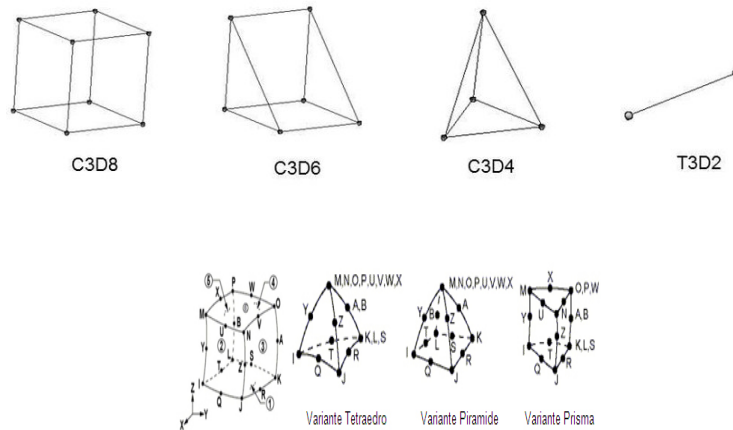


Fig. 3. Geometría del elemento finito sólido estructural

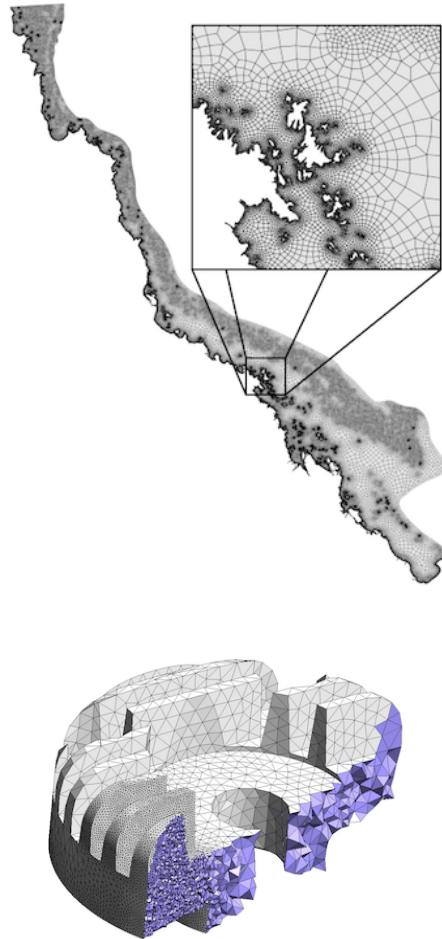
Mallas no estructuradas Las mallas no estructuradas parten de distribuciones de puntos que cumplen una serie de propiedades dadas, la mayoría de ellas relacionadas con una distribución lo más uniforme posible dentro del contorno. Son adecuadas para geometrías irregulares donde es muy difícil generar una malla estructurada.

GMSH utiliza un algoritmo de creación de malla basado en la triangulación Delaunay. Dado un contorno se demuestra que sólo existe una triangulación óptima en la que, para cada triángulo formado por tres puntos de la malla, no exista ningún punto dentro de la circunferencia que pasa por los tres puntos.

Si bien esta propiedad cierra el problema de la triangulación dados los puntos de la malla, en la mayoría de los casos sólo dispondremos del contorno. GMSH es capaz de encontrar una distribución no estructurada de puntos cuya triangulación resultante tiene suficiente calidad.

Discretización y calidad de malla. Una de las diferencias esenciales entre las mallas estructuradas y no estructuradas es el proceso de refinado. En algunos problemas la malla utilizada no produce suficiente precisión y es necesario rehacerla para aumentarla. En las mallas estructuradas el aumento de precisión no suele suponer un problema más allá de generar demasiados grados de libertad. Sin embargo en las mallas no estructuradas algunos algoritmos de refinado automático pueden funcionar mal.

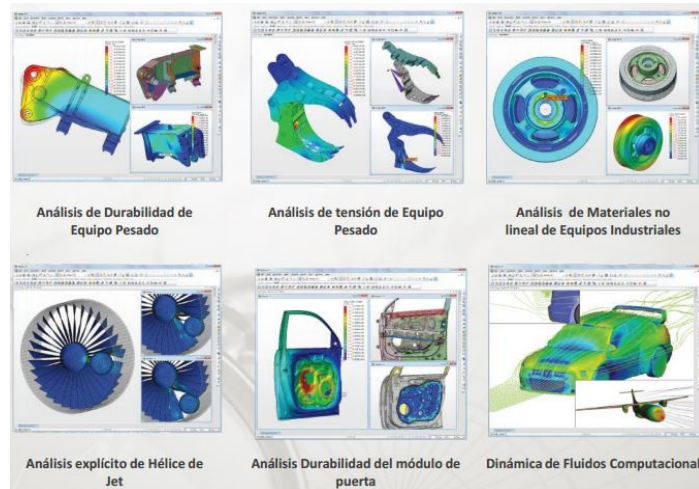
Uno de los algoritmos más utilizados es el de partir cada uno de los



triángulos o tetraedros en dos o más elementos. Este algoritmo recursivo puede refinar puntos arbitrarios en el espacio produciendo una pérdida de precisión al introducir gradientes espurios debidos a una mala definición de la geometría.

GMSH se puede usar en conjunto con FreeFem++ que es un software libre que sirve para la resolución numérica de problemas 2D y 3D utilizando el método de elementos finitos. ¿Qué problemas puedes resolver? De mecánica de fluidos, difusión de calor, elasticidad, electromagnetismo, etc. Todos aquellos que estén modelizados matemáticamente a través de una ecuación en derivadas parciales (EDP), por ejemplo:

<https://www.um.es/freefem/ff++/pmwiki.php?n=Main.Elasticidad3D>



Existe una gran cantidad herramientas para generar mallas en 2D, 3D que se pueden usar desde lenguajes de programación, entre las utilidades destacan:

- <https://www.salome-platform.org/>
- <https://www.paraview.org/>
- <https://www.meshlab.net/>
- <https://wias-berlin.de/software/index.jsp?id=TetGen&lang=1>
- <http://alice.loria.fr/software/geogram/doc/html/index.html>

5 Programación en Paquetes de Cálculo Numérico

El cálculo numérico permite diseñar e implementar algoritmos para que a través de números y reglas matemáticas se pueden simular procesos matemáticos complejos aplicados a la resolución de problemas de Ciencias e Ingenierías. De esta forma los paquetes de cálculo numérico proporcionan todo el andamiaje para llevar a cabo todos aquellos procedimientos matemáticos susceptibles de expresarse algorítmicamente que permitan su simulación o cálculo en procesos más sencillos empleando números.

Los paquetes de cálculo numérico, son programas matemáticos que ofrecen un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio con un amplio abanico de herramientas numéricas para la lectura, manipulación, análisis y graficación de datos. Entre sus prestaciones básicas se hallan:

- Manejo de números reales y complejos
- La manipulación de vectores y matrices tanto reales como complejas
- Manejo de funciones elementales y especiales
- Resolución de problemas de álgebra lineal
- Resolución de ecuaciones no lineales
- La representación de datos y funciones
- La implementación de algoritmos
- Integración de funciones
- Máximos y mínimos de funciones
- Manipulación de polinomios
- Integración de ecuaciones diferenciales
- Graficación de funciones en 2D y 3D
- La comunicación con programas en otros lenguajes de programación y con otros dispositivos de Hardware.

El programa comercial líder para el cálculo numérico es MATLAB (véase [24]) de la compañía MathWorks, este paquete salió a la venta en el año de 1984 con la versión 1.0 y casi año con año, ha generado nuevas versiones de su paquete y múltiples sistemas de licenciamiento.

La idea detrás de paquetes como MATLAB es la de emplear grupos de subrutinas escritas en principio en el lenguaje de programación FORTRAN como son las librerías LINPACK (véase [35]) y EISPACK (véase [36]) para manipular matrices y vectores y proporcionar un sencillo acceso a dicho Software y así, entre otras cosas, ser usado en resolución de problemas de álgebra lineal, análisis numérico, ecuaciones diferenciales parciales y ordinarias, sin necesidad de escribir programas en lenguajes de bajo nivel.

Estos paquetes, pueden disponer de herramientas adicionales que expanden sus prestaciones, como son el diseño y simulación de sistemas de control. Además, se pueden ampliar las capacidades base de dichos programas con las cajas de herramientas y con los paquetes de bloques. En algunos casos existen versiones para cómputo secuencial y paralelo -tanto en memoria compartida como distribuida, también para usar los múltiples Cores gráficos CUDA (GPUs) de las tarjetas NVIDIA (véase 8)-.

Los paquetes de cómputo para el Cálculo Numérico más usados actualmente son:

- MATLAB (véase [24]) «abreviatura de MATrix LABoratory (laboratorio de matrices)» es un Software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos Hardware. El paquete MATLAB dispone de dos herramientas adicionales que expanden sus prestaciones, a saber, una plataforma de simulación multidominio Simulink (véase [38]) y un editor de interfaces de usuario GUIDE. Además, se pueden ampliar las capacidades de MATLAB con su caja de herramientas (Tool Box) y las de Simulink con los paquetes de bloques.
- Octave (véase [28]) es un programa de cálculo numérico de licencia GNU (véase [20]), conocido por buscar una sintaxis similar a la de MATLAB, existiendo una gran compatibilidad en las funciones de cálculo numérico.

- FreeMat (véase [27]) es un programa de cálculo numérico de licencia GPL (véase [20]) que proporciona un ambiente libre para el rápido desarrollo de prototipos para la Ciencia y la Ingeniería, además del procesamiento de datos. Es similar a MATLAB y Octave, pero cuenta con una interfaz externa de código en los lenguajes de programación C, C++ y Fortran, incluso distribuye el desarrollo de algoritmos en paralelo con la interfaz de paso de mensajes llamada MPI.
- Scilab (véase [25]) es un programa de cálculo numérico de licencia CeCILL compatible con GPL (véase 10.2), desarrollado principalmente en Francia, que incluye su propia librería para gráficos. Es similar a MATLAB, pero no busca una compatibilidad total, como lo puede hacer FreeMat y Octave. Scilab tiene una herramienta para el diseño y simulación de sistemas de control Scicos (véase [37]) similar a Simulink (véase [38]) de MATLAB.
- Julia (véase [26]) es un lenguaje de programación homoicónico, multi-plataforma y multiparadigma de tipado dinámico de alto nivel y alto desempeño para la computación genérica, técnica y científica, con una sintaxis similar a la de otros entornos de computación similares, con licencia MIT (véase 10.2).

Dispone de un compilador avanzado (JIT), mecanismos para la ejecución en paralelo y distribuida, además de una extensa biblioteca de funciones matemáticas. La biblioteca, desarrollada fundamentalmente en Julia, también contiene código desarrollado en C o Fortran, para el álgebra lineal, generación de números aleatorios, procesamiento de señales, y procesamiento de cadenas. Adicionalmente, la comunidad de desarrolladores de Julia contribuye con la creación y distribución de paquetes externos a través del gestor de paquetes integrado de Julia a un paso acelerado. IJulia es el resultado de la colaboración entre las comunidades de IPython y Julia, provee de una poderosa interfaz gráfica basada en el navegador Web para Julia.

- Scipy (véase [29]) es una librería de herramientas numéricas para Python (véase [30]) con licencia Open Source (véase [20]). En su filosofía no ha tratado de imitar a ninguno de los paquetes anteriores y tiene detrás el respaldo de un auténtico lenguaje de programación orientado a objetos e interpretado, que también puede ser compilado para ganar velocidad

en la ejecución. Este hecho le confiere una gran potencia y la capacidad de beneficiarse de las mejoras del lenguaje base.

Existen otros paquetes que pueden ser usados en el cálculo numérico -estos poseen características que enriquecen las opciones clásicas de los paquetes de cálculo numérico-:

- R (véase [34])
- Maple (véase [46])
- Mathematica (véase [47])
- Maxima (véase [48])

Por otro lado existe **Anaconda**, una Suite de código abierto que abarca una serie de aplicaciones, librerías y conceptos diseñados para el desarrollo de la Ciencia de Datos con Python. En líneas generales Anaconda Distribution es una distribución de Python que funciona como un gestor de entorno, un gestor de paquetes y que posee una colección de más de 720 paquetes de código abierto. Anaconda Distribution se agrupa en 4 sectores o soluciones tecnológicas, Anaconda Navigator, Anaconda Project, las librerías de Ciencia de Datos y Conda. Todas estas se instalan de manera automática y en un procedimiento muy sencillo. Para más información ver: <https://www.anaconda.com/>.

También esta **SageMath**, una Suite de código abierto bajo la licencia GPL de Software matemático como: NumPy, SciPy, Matplotlib, Sympy, Maxima, GAP, FLINT, R, entre otros. Además combina acceso a una poderosa combinación del lenguaje basada en Python o directamente vía interfaces o Wrappers. La misión del proyecto es crear una alternativa de Software libre a Magma, Maple, Mathematica y Matlab. Para más información ver: <http://www.sagemath.org/>.

5.1 Cálculo Numérico con Octave

El paquete Octave (véase [28]), al mantener una sintaxis similar a la del paquete MATLAB (véase [24]) tiene cientos de características, tan variadas como el segmento de usuarios al que dicho Software está dirigido. Al ser un

paquete tan completo, es difícil que un usuario promedio use las características avanzadas de dicho paquete; esto repercute en que el usuario promedio pague un alto costo por el valor de las licencias de uso sin usar dichas características; y esto contrasta con un vertiginoso desarrollo de nuevas características, que permite a la compañía lanzar una o más versiones por año, cada una con múltiples opciones de licenciamiento, según las necesidades del segmento al que están dirigidas.

Las múltiples características y los miles de usuarios, han creado una comunidad robusta, la que ha logrado una gran cantidad de funciones optimizadas, que han permitido la publicación de decenas libros, cientos de artículos y miles de páginas Web en los cuales se muestra como resolver diversos problemas concomitantes en Ciencias e Ingenierías usando dicho paquete y la interacción con otros lenguajes como son C/C++, Fortran o Java.

Entre las principales aplicaciones de Octave/MATLAB incluyen la de métodos secuenciales y paralelos para resolver problemas de álgebra lineal con matrices (ralas, dispersas y densas), estadística, análisis de Fourier, optimización, integración numérica, resolución de ecuaciones diferenciales ordinarias y parciales, creación de gráficos y visualización de datos. Además de opciones para hacer interpolación y regresión de datos, cálculo de eigenvalores y valores singulares, etc.

Entornos de Programación Uno de los aspectos más agradables de Octave/MATLAB es su entorno de programación, que permite centralizar la información en un entorno de ventanas. El depurador está también incorporado en el editor.

5.2 Trabajando con Octave

GNU Octave es un lenguaje de alto nivel destinado para el cálculo numérico. Provee una interfaz sencilla, orientada a la línea de comandos (consola), que permite la resolución de problemas numéricos, lineales y no lineales, además permite la ejecución de scripts y puede ser usado como lenguaje orientado al procesamiento por lotes.

Octave posee una gran cantidad de herramientas que permiten resolver problemas de álgebra lineal, cálculo de raíces de ecuaciones no lineales, integración de funciones ordinarias, manipulación de polinomios, integración de ecuaciones diferenciales ordinarias y ecuaciones diferenciales algebraicas. Sus funciones también se pueden extender mediante funciones definidas por

el usuario escritas en el lenguaje propio de Octave o usando módulos dinámicamente cargados escritos en lenguajes como C, C++ y Fortran entre otros.

Cálculos básicos Vamos a empezar a trabajar con la tarea más básica que podemos darle a esta herramienta, vamos a utilizarla como una calculadora. A continuación, veamos un ejemplo sencillo de suma, multiplicación y división:

```
octave:1> 45 + 76 * 12.5 / 1.5
```

De igual manera también podemos calcular promedios de la siguiente forma:

```
octave:1> (12+15+20)/3
```

Para calcular potencias utilizamos el símbolo "^" o doble asterisco "**", ejemplo:

```
octave:1> 2^8
```

Octave, como toda calculadora cumple con cierta regla de precedencia para el uso de operadores, esta es: las expresiones se evaluarán de izquierda a derecha, la potencia tendrá el mayor orden de precedencia, seguido de la multiplicación y división, y con la suma y resta como los operadores con menor precedencia. No es lo mismo: " $4 + 5 / 4 - 3$ " que " $(4 + 5) / (4 - 3)$ ". En la primera expresión se evaluará primero la división entre 5 y 4 y luego se le sumará (restará) el 4 y el 3, en la segunda expresión primero se evaluarán las operaciones dentro de los paréntesis y luego se dividirá el resultado de estos valores.

Variables Las variables son identificadores que permiten almacenar datos, los cuales pueden cambiar durante la ejecución de un programa. Las variables nos permiten asignarle nombres a los valores para luego poder hacer referencia a estos. No hay límite para la longitud en el nombre de una variable, pero estos deben estar constituidos por una secuencia de letras, undersCores (símbolo de subrayado o guión bajo) o números y sólo pueden empezar con letras o undersCores.

Vectores y Matrices En el área de la computación un vector generalmente es definido como un arreglo, es decir, un conjunto de datos a los cuales se accede por medio de índices. Un vector es la forma más simple de una matriz, podemos decir que es una matriz de una dimensión. En Octave disponemos de una gran variedad de formas para definir vectores y matrices, usualmente lo hacemos encerrando los elementos dentro de corchetes, los elementos separados por espacios o comas (,) definen un vector fila, los elementos separados por el retorno de carro o por punto y coma (;) definen un nuevo vector fila en la matriz, ejemplo:

```
octave:1> a=[1,3,5]
octave:2> b=[1;2;3]
octave:3> e=[a ; b ; 2, 4 6]
```

Para definir la matriz identidad, por ejemplo de tamaño 3, hacemos:

```
octave:1> E=eye(3)
```

a partir de esta matriz, podemos definir un vector de ella, mediante:

```
octave:1> y=E(:,1)
```

ahora podemos hacer la multiplicación matriz vector, mediante:

```
v=E*v
```

Para definir una matriz de unos de tamaño 3, hacemos:

```
octave:1> A=ones(3)
```

podemos calcular la inversa de una matriz, para ello construyamos la matriz a invertir, mediante:

```
octave:1> A=ones(3)+eye(3)
octave:2> C=inv(A)
```

Por ejemplo para resolver un sistema lineal, hacemos:

```
octave:1> A=ones(3)+eye(3)
octave:2> b=A(:,3)
octave:3> x=inv(A)*b
```

o de forma alternativa, podemos hacer eliminación gaussiana si A es cuadrada y hacer:

```
octave:3> x=A\b
```

Podemos calcular la transpuesta a un vector mediante el símbolo $'$, por ejemplo:

```
octave:1> a=[0 0 1]'
```

Para sustituir los valores de una fila o columna por los de un vector, hacemos:

```
octave:1> A(3,:)=v
octave:2> A(:,2)=w
```

Para intercambiar, por ejemplo las filas 2 y 3 de A , de la siguiente forma:

```
octave:1> A([2 3],:)=A([3 2],:)
```

Algunas otras matrices de uso común son:

- $\text{diag}(v)$ genera una matriz diagonal con el vector v como diagonal
- $\text{toeplitz}(v)$ define una matriz simétrica de diagonal constante con v como primera fila y primera columna
- $\text{toeplitz}(w,v)$ define una matriz simétrica de diagonal constante con w como primera columna y v como primera fila
- $\text{ones}(n)$ genera una matriz de $n \times n$ con todos los valores iguales a uno
- $\text{zeros}(n)$ genera una matriz de $n \times n$ con todos los valores iguales a cero
- $\text{eye}(n)$ genera una matriz identidad de $n \times n$
- $\text{rand}(n)$ genera una matriz de $n \times n$ con elementos de valor aleatorio entre 0 y 1 (distribución uniforme)
- $\text{randn}(n)$ genera una matriz de $n \times n$ cuyos elementos siguen una distribución normal (media 0 y varianza 1)

- *ones(m,n)*, *zeros(m,n)*, *rand(m,n)* generan matrices de $m \times n$
- *ones(size(A))*, *zeros(size(A))*, *yes(size(A))* generan matrices de la misma forma que A
- *triu(A)* coloca ceros en todos los elementos por debajo de la diagonal (triangular superior)
- *tril(A)* coloca ceros en todos los elementos por encima de la diagonal (triangular inferior)
- *inv(A)* calcula la inversa de A si es invertible
- *pinv(A)* calcula la pseudoinversa de A
- *det(A)* calcula el determinante (si A es una matriz cuadrada)
- *rank(A)* es el rango (número de pivotes=dimensión del espacio de filas y del espacio de columnas)
- *trace(A)* es la traza, suma de autovalores
- *null(A)* es una matriz cuyas columnas $n - r$ forman una base ortogonal para el espacio nulo de A
- *orth(A)* es la matriz cuyas columnas r forman una base ortogonal para el espacio de columnas de A

Secuencias Una forma sencilla de producir una secuencia de números es utilizando la notación $n:m$ donde n es el número inicial y m el final, ejemplo:

```
octave:1> 1:10
```

también podemos usar la notación $p:q:r$ para crear una secuencia que inicia en p , finaliza en r con intervalos de q . En el siguiente caso almacenaremos en una variable b una secuencia partiendo de cero y finalizando en diez, con un intervalo de dos entre cada número, ejemplo:

```
octave:1> b=0:2:10
```

Funciones matemáticas Octave incluye una serie de funciones matemáticas y trigonométricas que nos ayudan a simplificar algunos cálculos, la lista completa está disponible en la ayuda del programa⁴¹. Las funciones pueden contener otras funciones como argumentos:

```
octave:1> sqrt( round( 25/3 ) + 1 )
```

en Octave, se asume que los argumentos de funciones trigonométricas están en radianes por lo que si queremos calcular los grados de alguna función trigonométrica debemos aplicar la siguiente fórmula:

```
octave:1> sin(90*pi/180)
```

Sistema de ecuaciones Para la resolución de sistemas de ecuaciones del tipo $Ax = b$ utilizamos la notación $a \backslash b$, por ejemplo, para calcular el siguiente sistema de ecuaciones de primer grado con dos incógnitas:

$$\begin{aligned} 6 \cdot x - 7 \cdot y &= 5 \\ 8 \cdot x - 9 \cdot y &= 7 \end{aligned}$$

para resolver, haremos lo siguiente: guardaremos los elementos x e y en una matriz a y la igualdad en un vector b , para finalmente ejecutar el comando $a \backslash b$, de la siguiente forma:

```
octave:1> a=[6, -7;8,-9]
octave:2> b=[5;7]
octave:3> a\b
```

Funciones definidas por el usuario En Octave podemos crear nuestras propias funciones, podemos escribirlas directamente desde la línea de comandos o ejecutarlas desde un archivo externo. Las funciones que creamos desde la línea de comandos deben cumplir con el siguiente formato:

```
function variable_salida = nombre_funcion (argumentos_entrada)
    cuerpo_funcion
endfunction
```

⁴¹Alguna de ellas son: sin, sinh, asin, asinh, cos, cosh, acos, acosh, tan, tanh, atan2, atanh, sec, sech, asec, asech, csc, csch, acsc, acsch, cot, coth, acot, acoth, exp, log, log10, pow2, sqrt, fix, floor, ceil, round, mod, rem, sig, etc.

en caso de devolver varias variables, estas deben estar encerradas entre corchetes "[]":

```
function [salida1,salida2] = nombre_funcion (argumentos_entrada)
    cuerpo_funcion
endfunction
```

Por ejemplo, crearemos una función que calcula $\text{seno}(x)$ en grados:

```
function s = sind(x)
    %SIND(X) Calcula seno(x) en grados
    s = sin(x*pi/180);
endfunction
```

y ejecutamos:

```
octave:|> sind(45)
```

Para que Octave ejecute un archivo o script, este debe tener extensión *.m* y debe encontrarse en el directorio desde donde estemos ejecutando Octave. También podemos agregar rutas adicionales con el comando *addpath('ruta')*, donde ruta es el camino al directorio que contiene los scripts. Para borrar la ruta usamos *rmpath('ruta')*.

Por ejemplo, el cálculo de las raíces de la ecuación cuadrática, podemos definir en el archivo *cuadratica.m*, el siguiente código:

Ejemplo en Octave/MatLab

```
% Calculo de raices de una ecuacion cuadratica
function [x1,x2] = raiz(a,b,c)
    fprintf('Polinomio(%1.6f) X^2 + (%1.6f) X + (%1.6f) =
0\n',a,b,c);
    x1 = 0;
    x2 = 0;
    if a == 0
        fprintf('\nNo es cuadratica\n')
        return
    end
    fprintf('\nChicharronera 1\n')
```

```

x1 = (-b+sqrt(b*b-4*a*c))/(2*a);
r = f(a, b, c, x1);
fprintf('Raiz(%1.16f), evaluacion raiz: (%e)\n',x1,r);
x2 = (-b-sqrt(b*b-4*a*c))/(2*a);
r = f(a, b, c, x2);
fprintf('Raiz(%1.16f), evaluacion raiz: (%e)\n',x2,r);
fprintf('\nChicharronera 2\n')
x1 = (-2*c)/(b+sqrt(b*b-4*a*c));
r = f(a, b, c, x1);
fprintf('Raiz(%1.16f), evaluacion raiz: (%e)\n',x1,r);
x2 = (-2*c)/(b-sqrt(b*b-4*a*c));
r = f(a, b, c, x2);
fprintf('Raiz(%1.16f), evaluacion raiz: (%e)\n',x2,r);
fprintf('\nMetodo Newton-Raphson\n')
x = x1 -1;
fprintf('Valor inicial aproximado de X1 = %1.16f\n',x);
for i = 1:10
    x = x - (f(a, b, c, x) / df(a, b, c, x));
end
r = f(a, b, c, x);
fprintf('Raiz (%1.16f), evaluacion raiz: (%e)\n',x,r);
x = x2 -1;
fprintf('Valor inicial aproximado de X2 = %1.16f\n',x);
for i = 1:10
    x = x - (f(a, b, c, x) / df(a, b, c, x));
end
r = f(a, b, c, x);
fprintf('Raiz (%1.16f), evaluacion raiz: (%e)\n',x,r);
end
% Funcion cuadratica
function r = f(a, b, c, x)
    r = x * x * a + x * b + c;
end
% Derivada de la funcion cuadratica
function r = df(a, b, c, x)
    r = 2.0 * x * a + b;
end

```

y podemos ejecutar mediante:

```
[x1,x2] = cuadratica(1,4,1)
```

para el polinomio $x^2 + 4x + 1 = 0$.

Gráficos 2D en Octave La forma más simple para producir gráficas bidimensionales en Octave se hace a través del comando `plot(x[,y][,fmt])`, donde x y opcionalmente y representan los vectores de coordenadas para cada punto, adicionalmente se le puede especificar una serie de formatos que cambiarán la apariencia de salida de la gráfica, esto incluye el estilo de línea, el color y otras características que explicaremos en los siguientes ejemplos. En este primer ejemplo crearemos un vector aleatorio de 50 elementos que luego graficaremos:

```
octave:1> x = (rand(50,1));
octave:2> plot(x)
```

acto seguido se abre una ventana mostrándonos la gráfica resultante.

Gráficos 3D en Octave Para producir gráficos en 3D disponemos de varias opciones, la más simple es usar el comando `plot3(x,y,z)` donde cada argumento es tomado para convertirse en los vértices del gráfico tridimensional. Si todos los argumentos son vectores de la misma longitud se dibujará una única línea continua. En caso de que todos los argumentos sean matrices cada una de las columnas de las matrices serán tratadas como líneas separadas. En caso de que sólo se le pasen dos argumentos en lugar de tres, `plot3(x,c)`, el segundo argumento 'c' debe ser un número complejo, así, las partes reales e imaginarias de este son usadas como las coordenadas y e z respectivamente. Si sólo se le pasa un argumento, `plot3(c)`, las partes reales e imaginarias de los argumentos son usados como los valores y y z , y se trazan frente a su índice.

El comando `plot3` también acepta los argumentos que permiten modificar el formato de presentación de la gráfica descritos en la sección de gráficas bidimensionales, ejemplo:

```
octave:1> t=0:pi/50:10*pi;
octave:2> plot3(sin(t),cos(t),t)
octave:3> grid on
octave:4> axis square
```


acto seguido se abre una ventana mostrándonos la gráfica resultante.

5.3 Desde la Nube

Existen diferentes servicios Web⁴² que permiten editar, compilar y ejecutar código de diversos lenguajes y paquetes desde el navegador, esto en aras de que los estudiantes y profesores que cuenten con algún sistema de acceso a red y un navegador puedan programar en los más diversos lenguajes, IDEs y terminales sin hacer instalación alguna en su equipo de cómputo, tableta o teléfono celular.

Algunos ejemplos de estos servicios son:

- Para Octave <https://octave-online.net/>
- Para Octave https://rextester.com/1/octave_online_compiler
- Para Octave http://www.compileonline.com/execute_matlab_online.php
- Para Octave http://www.tutorialspoint.com/octave_terminal_online.php
- Para Octave <https://www.jdoodle.com/execute-octave-matlab-online>
- Para SciLab <http://cloud.scilab.in/>
- Para SciLab <http://hotcalcul.com/on-line-calculator/scilab>
- Para Scipy <https://www.jdoodle.com>
- Para Scipy <https://try.jupyter.org/>
- Para Scipy <http://browxy.com>

⁴²Cuando se trabaja desde la Web es recomendable usar el modo Privado o Incógnito para no guardar el historial de navegación, información introducida en los formularios y borrar al cerrar el navegador los datos de los sitios visitados. Pero recuerda que los sitios Web que visitamos sí guardan información de nuestra visita, nuestro proveedor de Internet también guarda constancia de nuestra visita y si descargamos algo, esto no se borra al igual que el historial de descargas, además de las marcas de páginas o favoritos se conservarán al cerrar el navegador.

6 Programación en Paquetes de Cálculo Simbólico

En matemáticas y ciencias de la computación, el cálculo simbólico, también conocido como cálculo algebraico o álgebra computacional, es un área científica que se refiere al estudio y desarrollo de algoritmos y Software para la manipulación de expresiones matemáticas y otros objetos matemáticos. Aunque, hablando con propiedad, el álgebra computacional debe ser un subcampo de la computación científica, ellos son considerados generalmente como campos distintos, porque la computación científica se basa generalmente en el análisis numérico con números aproximados en punto flotante; mientras que, el álgebra computacional enfatiza el cálculo exacto con expresiones que contengan variables que no tienen cualquier valor dado y por lo tanto son manipulados como símbolos (de ahí se debe el nombre de cálculo simbólico).

Las aplicaciones de Software que realizan cálculos simbólicos son conocidos como sistemas de álgebra computacional, con el término sistema aludiendo a la complejidad de las principales aplicaciones que incluyen, al menos, un método para representar los datos matemáticos en una computadora, un lenguaje de programación de usuario (por lo general diferente del lenguaje usado para la ejecución), un administrador de memoria, una interfaz de usuario para la entrada/salida de expresiones matemáticas, un gran conjunto de subrutinas para realizar operaciones usuales, como la simplificación de expresiones, la regla de la cadena utilizando diferenciación, factorización de polinomios, integración indefinida, etc.

El álgebra computacional es ampliamente utilizado para experimentar en matemáticas y diseñar las fórmulas que se utilizan en los programas numéricos. También se usa para cálculos científicos completos, cuando los métodos puramente numéricos fallan, como en la criptografía asimétrica o para algunos problemas no lineales.

Los paquetes de cálculo simbólico, son programas matemáticos que ofrecen un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio con un amplio abanico de herramientas simbólicas y numéricas para la lectura, manipulación, análisis y graficación de datos. Entre sus prestaciones básicas se hallan:

- Bibliotecas de funciones matemáticas elementales y especiales
- Matrices y manipulación de datos

- Soporte para números complejos, precisión arbitraria, computación de intervalos aritméticos y simbólicos
- Datos en 2D y 3D, función y visualización geográfica y herramientas de animación
- Solucionadores para sistemas de ecuaciones, ecuaciones diofánticas, ecuaciones diferenciales ordinarias, parciales, diferenciales algebraicas, de retraso, diferenciales estocásticas y relaciones de recurrencia
- Herramientas numéricas y simbólicas para cálculo de variable continua o discreta
- Bibliotecas de Estadística multivariable, incluyendo ajuste, pruebas de hipótesis, y cálculos de probabilidad
- Soporte para datos censurados, datos temporales, series temporales y datos basados en unidades
- Cálculos y simulaciones en procesos aleatorios y colas
- Geometría computacional en 2D, 3D y dimensiones mayores
- Optimización restringida y no restringida, local y global
- Herramientas para problemas combinatorios
- Soporta el desarrollo de cálculos matemáticos de manera simbólica y numérica con precisión arbitraria
- Librerías para funciones matemáticas básicas y avanzadas
- Manejo de números complejos y sus diversas operaciones
- Aritmética, álgebra, operaciones para desarrollo de polinomios multivariados
- Límites, series y sucesiones
- Álgebra diferencial
- Herramientas para la manipulación de matrices incluyendo matrices dispersas

- Sistemas de solución para ecuaciones diferenciales en sus diferentes variedades (ODE, DAE, PDE, DDE)
- Herramientas simbólicas y numéricas para cálculo discreto y continuo, incluye integración definida e indefinida, diferenciación
- Optimización con restricciones y sin restricciones
- Herramientas para la resolución de problemas en el campo de la probabilidad
- Herramientas para el uso de series de tiempo
- Conexión a datos en línea, recopilados para aplicaciones financieras y económicas
- Herramientas para cálculos financieros, incluyendo: bonos, anualidades entre otros
- Cálculos y simulaciones para procesos aleatorios
- Herramientas para el desarrollo de sistemas lineales y no lineales
- Incluye matemáticas discretas

Los programas líderes para el cálculo simbólico son Mathemática y Maple, los costos de sus licencias respectivas son onerosos, pero para la UNAM se tienen licencias estudiantiles disponibles para profesores y estudiantes. Existen otros paquetes como Maxima que tiene licencia GPL y están disponibles para los usuarios en múltiples plataformas.

6.1 Cálculo Simbólico con Maxima

El sistema de álgebra computacional Maxima⁴³ es un motor de cálculo simbólico escrito en lenguaje Lisp publicado bajo licencia GNU GPL. Cuenta con un amplio conjunto de funciones para hacer manipulación simbólica de polinomios, matrices, funciones racionales, integración, derivación, manejo de gráficos en 2D y 3D, manejo de números de coma flotante muy grandes,

⁴³Véase <https://es.wikipedia.org/wiki/Maxima>

expansión en series de potencias y de Fourier, entre otras funcionalidades. Además tiene un depurador a nivel de fuente para el código de Maxima.

Maxima está basado en el sistema original de Macsyma desarrollado por MIT en los años 70. Es bastante fiable, tiene un buen recolector de basura, por lo que no desperdicia memoria. Viene con cientos de auto pruebas (test-suite). Maxima funciona en modo consola, sin embargo incluye las interfaces gráficas xMaxima y wxMaxima para facilitar su uso.

El editor de texto científico GNU TeXmacs también puede ser usado para facilitar una interfaz gráfica de usuario para Maxima. Otras opciones son, *Imaxima*, y el modo interactivo de *Emacs*. También puede hacer uso de la interfaz gráfica de *SageMath*, que facilita su integración con otras herramientas.

Como está escrito en *Common Lisp*, es fácilmente accesible para la programación, desde la capa inferior de Lisp puede llamarse a *Maxima*. Como la mayoría de sistemas algebraicos, *Maxima* se especializa en operaciones simbólicas. También ofrece capacidades numéricas especiales, como son los números enteros y racionales, los cuales pueden crecer en tamaño sólo limitado por la memoria de la máquina; y números reales en coma flotante, cuya precisión puede ser arbitrariamente larga (*bfloat*). Permite el manejo de expresiones simbólicas y numéricas, y además produce resultados con una alta precisión.

Para cálculos intensivos en reales de coma flotante, Maxima ofrece la posibilidad de generar código en otros lenguajes de programación, como Fortran, que quizá se ejecuten de manera más eficiente.

Maxima es un sistema de propósito general; como tal los cálculos especiales como la factorización de números grandes, la manipulación de polinomios extremadamente grandes, etc. son normalmente realizados de forma más eficiente y rápida en sistemas especializados.

- Números
- Listas, arreglos y matrices
- Transformaciones algebraicas
- Resolución de ecuaciones
- Límites, derivadas e integrales
- Conjuntos

- Vectores y campos
- Gráficos
- Ecuaciones diferenciales
- Probabilidades y análisis de datos
- Interpolación numérica
- Inecuaciones racionales
- Ecuaciones diferenciales ordinarias
- Sistemas de ecuaciones diferenciales ordinarias
- Series de potencias
- Transformada de Laplace
- Ecuaciones recurrentes
- Generación de expresiones en Tex
- Programación

6.2 Trabajando con Maxima

Maxima es un poderoso paquete que puede hacer múltiples operaciones por nosotros, a continuación haremos un viaje por sus capacidades, pero sólo es una muestra de ellas, para más información revise el manual y ejemplos en línea.

Operaciones Aritméticas Para iniciar, podemos usar a Maxima como una calculadora, por ejemplo $(34*6)/34$, entonces escribimos:

$(34*6)/34;$

y el paquete nos dará el resultado, las operaciones soportadas son $+$, $-$, $*$, $/$ y $^$. Además puede trabajar con números enteros tan grandes como se desee, por ejemplo para calcular el factorial de 200, hacemos:

200!;

Maxima devuelve los resultados de forma exacta sin aproximaciones decimales, por ejemplo para resolver la siguiente operación:

$$\left[\left(\frac{3^7}{4 + \frac{3}{5}} \right)^{-1} + \frac{7}{9} \right]^3$$

escribimos:

`((3^7/(4+3/5)^(-1)+7/9)^3;`

y obtenemos el resultado de forma simplificada. Si lo necesitamos podemos pedir el resultado en forma decimal; por ejemplo si queremos la expresión decimal del último resultado, escribimos:

`float(%);`

Maxima puede trabajar con precisión arbitraria. Para calcular el valor del cociente $\frac{e}{\pi}$ con cien cifras decimales, debemos especificar primero la precisión requerida asignándole a la variable *fpprec* el valor 100 y a continuación realizar el cálculo, solicitando la expresión decimal con una llamada a la función *bfloat*, de esta forma:

`fpprec:100$ bfloat(%e / %pi);`

aquí se usa el símbolo de \$ como delimitador entre instrucciones.

Factorización Maxima permite factorizar un número como el factorial de doscientos (200!) en sus factores primos, sólo hacemos:

`factor(200!);`

y nos entregará la lista de los factores primos. Si queremos saber el tiempo de ejecución que lleva hacer algún cálculo, hacemos:

`showtime:true$`

y especificamos la operación:

`factor(2^300-1);`

y desactivamos la información de tiempo de cálculo mediante:

`showtime:false$`

Números Complejos Maxima soporta números complejos, la unidad imaginaria $\sqrt{-1}$ se representa mediante el símbolo `%i` y soporta operaciones básicas con números complejos, por ejemplo definimos los números `z1` y `z2`:

```
z1:3+5*%i$ z2:1/2-4*%i$
```

ahora podemos hacer cualquier otra operación, por ejemplo sumarlas:

```
z1+z2;
```

Combinatoria Maxima también soporta combinatoria, por ejemplo el cálculo de permutaciones de 3 elementos, mediante:

```
factorial(3)
```

si queremos generarlas, hacemos uso de la función *permutations*, por ejemplo:

```
permutations([1,2,3]);
```

cuando se trabaja con factoriales, la función *minfactorial* puede ayudar a simplificar algunas expresiones, por ejemplo:

```
minfactorial(n!/(n+2)!);
```

Expresiones Algebraicas Una de las capacidades más destacables de Maxima es su habilidad para manipular expresiones algebraicas, por ejemplo a la variable q le asignaremos una expresión literal:

```
q: (x+3)^5-(x-a)^3+(x+b)^(-1)+(x-1/4)^(-5);
```

se observa que en principio Maxima no realiza ningún cálculo. La función *expand* se encarga de desarrollar las potencias y los productos algebraicos:

```
expand(q);
```


no obstante es posible que no nos interese desplegar toda la expresión, entre otras cosas para evitar una respuesta farragosa y difícil de interpretar; en tal caso podemos utilizar *expand* añadiéndole dos argumentos y operando de la siguiente manera:

```
expand(q,3,2);
```

con los argumentos adicionales indicamos que queremos la expansión de todas aquellas potencias con exponente positivo menor o igual a 3 y de las que teniendo el exponente negativo no excedan en valor absoluto de 2.

Dada una expresión con valores literales, podemos desear sustituir alguna letra por otra expresión; por ejemplo, si queremos hacer los cambios $a = 2$, $b = 2c$ en el último resultado obtenido:

```
%,a=2,b=2*c;
```

De forma más general, la función *subst* substituye subexpresiones enteras. En el siguiente ejemplo, introducimos una expresión algebraica y a continuación sustituimos todos los binomios $x + y$ por la letra k :

```
1/(x+y)-(y+x)/z+(x+y)^2;  
subst(k,x+y,%);
```

No obstante, el siguiente resultado nos sugiere que debemos ser precavidos con el uso de esta función, ya que Maxima no siempre interpretará como subexpresión aquella que para nosotros sí lo es:

```
subst(sqrt(k),x+y,(x+y)^2+(x+y));
```

la función *subst* realiza una simple sustitución sintáctica que, si bien es suficiente en la mayor parte de los casos, a veces se queda corta como en la situación anterior; la función *ratsubst* es capaz de utilizar información semántica y obtiene mejores resultados:

```
ratsubst(sqrt(k),x+y,(x+y)^2+(x+y));
```

la operación inversa de la expansión es la factorización. Expandamos y factoricemos sucesivamente un polinomio para comprobar los resultados:

```
expand((a-2)*(b+1)^2*(a+b)^5);
factor(%);
```

el máximo común divisor de un conjunto de polinomios se calcula con la función *gcd* y el mínimo común múltiplo con *lcm*:

```
p1: x^7-4*x^6-7*x^5+8*x^4+11*x^3-4*x^2-5*x;
p2: x^4-2*x^3-4*x^2+2*x+3;
gcd(p1,p2);
load(funcs)$
lcm(p1,p2);
```

es posible que deseemos disponer del *mcd* factorizado, por lo que hacemos:

```
factor(%);
```

la instrucción *solve* puede admitir como segundo argumento la incógnita que se pretende calcular, lo que resultará de utilidad cuando en la ecuación aparezcan constantes literales:

```
solve((2-a)/x-3=b*x+1/x,x);
```

Matrices Podemos también hacer uso de matrices, por ejemplo:

```
M : matrix([sqrt(3),2/5],[3,sin(2)]);
```

y resolverla:

```
linsolve__by__lu(M,M.M,'floatfield);
```

Límites Sin más preámbulos, veamos algunos ejemplos de cómo calcular límites con la asistencia de Maxima. En primer lugar vemos que es posible hacer que la variable se aproxime al infinito ($x \rightarrow \infty$), o que se aproxime al menos infinito ($x \rightarrow -\infty$):

```
limit(1/sqrt(x),x,inf);
```

Derivadas Maxima controla el cálculo de derivadas mediante la instrucción *diff*. A continuación se presentan algunos ejemplos sobre su uso:

```
diff(x^log(a*x),x);
```

la segunda derivada:

```
diff(x^log(a*x),x,2);
```

Integrales En Maxima la función que controla el cálculo de integrales es *integrate*, tanto para las definidas como indefinidas; empecemos por estas últimas:

```
integrate(cos(x)^3/sin(x)^4,x);
```

otro ejemplo sobre la integral definida es:

```
integrate(2*x/((x-1)*(x+2)),x,3,5);  
float(%);
```

6.3 Desde la Nube

Existen diferentes servicios Web⁴⁴ que permiten editar, compilar y ejecutar código de diversos lenguajes y paquetes desde el navegador, esto en aras de que los estudiantes y profesores que cuenten con algún sistema de acceso a red y un navegador puedan programar en los más diversos lenguajes, IDEs y terminales sin hacer instalación alguna en su equipo de cómputo, tableta o teléfono celular.

Algunos ejemplos de estos servicios son:

- <https://mathics.angusgriffith.com/> Clon de Mathematica
- <http://www.wolframalpha.com/>
- <http://www.quickmath.com/>
- <http://maxima-online.org>
- <http://maxima.cesga.es/>

⁴⁴Cuando se trabaja desde la Web es recomendable usar el modo Privado o Incógnito para no guardar el historial de navegación, información introducida en los formularios y borrar al cerrar el navegador los datos de los sitios visitados. Pero recuerda que los sitios Web que visitamos sí guardan información de nuestra visita, nuestro proveedor de Internet también guarda constancia de nuestra visita y si descargamos algo, esto no se borra al igual que el historial de descargas, además de las marcas de páginas o favoritos se conservarán al cerrar el navegador.

7 Programación en Paquetes Estadísticos

Los paquetes estadísticos, son programas matemáticos que ofrecen un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio con un amplio abanico de herramientas de lectura, manipulación, análisis y graficación de datos estadísticos. Entre sus prestaciones básicas destacan:

- Análisis de datos mediante operadores para cálculos sobre arreglos, matrices y/o Tablas
- Tablas Cruzadas
- Reordenamiento de Datos
- Análisis de la Varianza (ANOVA)
- Frecuencias
- Estadística Descriptiva
- Estadística Lineal
- Estadística no Lineal
- Estadística Bioestadística
- Pruebas Estadísticas Clásicas
- Análisis de Serie de Temporales
- Modelos de Regresión
- Clasificación
- Fiabilidad
- Categorías
- Clustering
- Validación de Datos
- Tendencias

- Gráficos y Diagramas

Actualmente, los paquetes estadísticos usados en las carreras de Actuaría, Ciencias de la Computación, Matemáticas, Matemáticas Aplicadas y los cursos de Matemáticas de las demás carreras de la Facultad (véase [2]) son:

- SPSS (véase [32])
- R (véase [34])
- SAS (véase [31])
- PSPP (véase [33])
- EViews (véase [40])
- Gretel (véase [41])
- Stata (véase [39])
- Statgraphics (véase [42])
- Statistica (véase [43])
- Systat (véase [44])
- Vensim (véase [45])
- Maple (véase [46])
- Mathematica (véase [47])
- MATLAB (véase [24])
- FreeMat (véase [27])
- Octave (véase [28])
- Maxima (véase [48])
- Scipy (véase [29])
- Anaconda (véase [52])

- SageMath (véase [53])

En el presente trabajo nos centraremos en R, pero el resto de los paquetes son muy usados ya sea por sus características, facilidad de uso o la accesibilidad del paquete para los estudiantes. Cabe aclarar, que los paquete SPSS y SAS tienen un alto costo monetario, en el caso de SAS existe una versión estudiantil gratuita y SPSS hay una versión de prueba.

7.1 Cálculo Estadístico con R

El paquete R (véase [34]) es un lenguaje y entorno de programación para análisis estadístico y gráfico. Se trata de un proyecto de Software libre, resultado de la implementación GNU del premiado lenguaje S. SPSS, R y S-Plus -versión comercial de S- son, probablemente, los tres lenguajes más utilizados en investigación por la comunidad estadística, siendo además muy populares en el campo de la investigación biomédica, la bioinformática y las matemáticas financieras. A esto contribuye la posibilidad de cargar diferentes bibliotecas o paquetes con finalidades específicas de cálculo o gráfico.

Además, R puede integrarse con distintas bases de datos y existen bibliotecas que facilitan su uso desde lenguajes de programación interpretados como Perl y Python. R soporta hacer interfase con lenguajes de programación como C, C++ y Fortran.

Otra de las características de R es su capacidad gráfica, que permite generar gráficos con alta calidad. R posee su propio formato para la documentación basado en LaTeX (véase [23]). R también puede usarse como herramienta de cálculo numérico, campo en el que puede ser tan eficaz como otras herramientas específicas tales como FreeMat, GNU Octave y su equivalente comercial, MATLAB. Se ha desarrollado una interfaz RWeka para interactuar con Weka (véase [49]) que permite leer y escribir ficheros en el formato *arff* y enriquecer R con los algoritmos de minería de datos de dicha plataforma.

Los ambientes de desarrollo integrado para R existen como proyectos externos, como pueden ser editores -que sólo soportan la sintaxis-, los IDEs (Integrate Development Environments) y los GUI (Graphical User Interfaces) -permiten editar, ejecutar y depurar código desarrollado para R-. Hay más de 20 proyectos activos, dos de los más conocidos son Tinn-R (véase [50]) y RStudio (véase [51]).

7.2 Trabajando con R

R es un poderoso paquete que puede hacer múltiples operaciones por nosotros, a continuación haremos un viaje por sus capacidades, pero sólo es una muestra de ellas, para más información revise el manual y ejemplos en línea. Veamos algunos aspectos básicos a tener en cuenta:

- R distingue mayúsculas y minúsculas
- Para asignar contenido a un objeto usamos `<-`. Por ejemplo, `x <- 10` asigna a `x` el valor `10`. En lugar de `<-` también podemos usar `=`
- Para ver el contenido de un objeto simplemente escribimos su nombre
- Para usar los comandos escribimos el nombre del comando seguido de sus argumentos entre paréntesis. Por ejemplo, `ls()` da una lista de los objetos en el área de trabajo. Como no usamos argumentos (diferentes a los que el comando tenga por defecto) no escribimos nada en el paréntesis
- Para obtener ayuda usamos el comando `help`. Por ejemplo, `help(mean)` para obtener ayuda sobre el comando `mean` que calcula la media

El comando `c` se usa para combinar varios objetos en uno solo. Por ejemplo:

```
x <- c(3, 5, 10)
```

crea el vector $x = (3, 5, 10)$, ahora lo visualizamos mediante:

```
x
```

para calcular, por ejemplo la media de x , hacemos:

```
mean(x)
```

Es habitual encontrar datos en la Web en formato de texto. Antes de poder trabajar con ellos hemos de importarlos a un formato con el que R pueda trabajar. Por ejemplo, en la comunidad de Madrid evalúa anualmente

a los alumnos de sexto de primaria de todos los colegios sobre varias materias. Los datos que vamos a utilizar corresponden a las notas medias obtenidas por los colegios en los años 2009 y 2010 (fuente: diario El País) junto con el tipo de colegio (concertado, privado o público).

Para ello, hay que bajar los datos y generar un fichero de R, el comando *read.table* lee un fichero de datos de texto y crea un fichero con el que R ya puede trabajar. En nuestro ejemplo vamos a usar los siguientes argumentos:

- El camino en el que se encuentra el fichero (entre comillas).
- El argumento *sep* que especifica la separación entre las distintas variables. En este caso, un espacio en blanco.
- El argumento *dec* que especifica cómo se separa en los números la parte entera y la parte decimal. En este caso, mediante una coma.
- El argumento *header* que especifica si la primera fila del fichero contiene (TRUE) o no (FALSE) los nombres de las variables. En este caso, la primera fila sí contiene los nombres de las variables.

```
notas <- read.table("http://www.uam.es/joser.berrendero/datos/notas.txt",  
  
sep = " ", dec = ",", header = TRUE)
```

Medidas descriptivas elementales, para ver los nombres de las variables del fichero *notas*, escribimos:

```
names(notas)
```

dentro de este fichero cada variable se identifica usando la sintaxis:

fichero\$variable

Por ejemplo, las medias de las notas de 2009 y 2010 se obtienen con los comandos:

```
mean(notas$nota09)
```

las desviaciones típicas de las notas de 2009 y 2010 se obtienen con los comandos:

```
sd(notas$nota09)
sd(notas$nota10)
```

el comando *summary* permite calcular algunas medidas descriptivas elementales de todas las variables del fichero simultáneamente:

```
summary(notas)
```

Ya con esta información, ahora podemos generar un histograma de las notas de 2009. La distribución es bastante simétrica aunque muestra una ligera asimetría a la izquierda.

```
hist(notas$nota09)
```

para comparar conjuntos de datos un diagrama de caja puede ser más útil que un histograma. Por ejemplo, para comparar las notas de 2009 con las de 2010:

```
boxplot(notas$nota09, notas$nota10)
```

vemos que en 2010 las notas tienden a ser más bajas. También podemos comparar las notas de 2010 para cada tipo de colegio. Aunque las diferencias no son muy grandes, los centros privados tienden a tener notas más altas que los concertados, y éstos más altas que los públicos.

```
boxplot(notas$nota10 ~ notas$tipo)
plot of chunk boxplot2
```

para estudiar si hay relación entre las notas de los dos años podemos representar la nube de puntos correspondiente:

```
plot(notas$nota09, notas$nota10)
```

se observa una relación positiva moderada. Numéricamente, esta relación se puede cuantificar mediante la covarianza y la correlación:

```
cov(notas$nota09, notas$nota10)
cor(notas$nota09, notas$nota10)
```

Además de analizar datos, R se puede usar para llevar a cabo todo tipo de representaciones gráficas y cálculos matemáticos. Por ejemplo, la función $\cos(x^2)$ se puede representar entre $[0, \frac{2}{\pi}]$ mediante:

```
curve(cos(x^2), 0, 2 * pi)
```

la integral de la función anterior entre $[0, \frac{2}{\pi}]$ se calcula con:

```
integrate(function(x) cos(x^2), 0, pi)
```

una derivada de la función se obtiene de la siguiente forma:

```
D(expression(cos(x^2)), "x")
```

para representar la función de densidad de una variable aleatoria normal de media $\mu = 3$ y desviación típica $\sigma = 1$ escribimos:

```
curve(dnorm(x, mean = 3, sd = 1), -1, 7)
```

hemos usado el comando *dnorm*, con los argumentos adecuados, para evaluar la función de densidad normal. Si lo que queremos es generar 1000 números aleatorios que sigan esta distribución y después representar el histograma correspondiente:

```
x <- rnorm(1000, mean = 3, sd = 1)
hist(x)
```

Podemos escribir también nuestras propias funciones. En este apartado vamos a ver un ejemplo muy simple. Se trata de escribir una función que simule los resultados de n tiradas de un dado. Por defecto n . El resultado es un vector que contiene las frecuencias de cada posible resultado. Abrimos el editor (Archivo, Abrir script) y escribimos:

```
Dado <- function(n = 50) {
  # Genera n lanzamientos de un dado y devuelve la tabla de frecuencias
  lanzamientos <- sample(1:6, n, rep = TRUE)
  # Selecciona con reemplazamiento n números en 1,2,...,6
  frecuencias <- table(lanzamientos) # Obtiene la tabla de frecuencias
  return(frecuencias)
}
```

una vez guardado el fichero que contiene la función podemos usar:

```
source(nombredelfichero)
```

para cargar la función en el área de trabajo y poder usarla o, alternatively, copiar y pegar el código anterior en la consola. Una vez hecho esto, ya podemos usar la función.

```
Dado(1000)
Dado()
```

7.2.1 R y Python

El paquete R es un lenguaje y entorno de programación para análisis estadístico y gráfico poderoso, pero es posible extender su uso al lenguaje de programación de Python, logrando una amalgama aún en proceso de desarrollo que permite tener lo mejor de ambos mundos con muy poco trabajo extra.

Para iniciar, se debe importar las funciones y demás que componen la parte del paquete R para poder usarse en Python, mediante:

```
from R_functions import *
```

y se está listo para usar R en Python, ejemplo:

```
lst=[20,12,16,32,27,65,44,45,22,18]
fivenum(lst)
```

obteniendo:

```
> array([12. , 18.5, 24.5, 41. , 65. ])
```

Un ejemplo más es calcular un diagrama de barras con unas cuantas líneas usando la función *dbinom*:

```
probs=[]
import matplotlib.pyplot as plt
for i in range(11):
    probs.append(dbinom(i,10,0.6))
plt.bar(range(11),height=probs)
plt.grid(True)
plt.show()
```

Por otro lado existe **Anaconda**, una Suite de código abierto que abarca una serie de aplicaciones, librerías y conceptos diseñados para el desarrollo de la Ciencia de datos con Python. En líneas generales Anaconda Distribution es una distribución de Python que funciona como un gestor de entorno, un gestor de paquetes y que posee una colección de más de 720 paquetes de código abierto. Anaconda Distribution se agrupa en 4 sectores o soluciones tecnológicas, Anaconda Navigator, Anaconda Project, Las librerías de Ciencia de datos y Conda. Todas estas se instalan de manera automática y en un procedimiento muy sencillo. Para más información ver: <https://www.anaconda.com/>.

También esta **SageMath**, una Suite de código abierto bajo la licencia GPL de Software matemático como: NumPy, SciPy, matplotlib, Sympy, Maxima, GAP, FLINT, R, entre otros. Además combina acceso a una poderosa combinación del lenguaje basada en Python o directamente vía interfaces o Wrappers. La misión del proyecto es crear una alternativa de Software libre a Magma, Maple, Mathematica y Matlab. Para más información ver: <http://www.sagemath.org/>.

7.3 Desde la Nube

Existen diferentes servicios Web⁴⁵ que permiten editar, compilar y ejecutar código de diversos lenguajes y paquetes desde el navegador, esto en aras de que los estudiantes y profesores que cuenten con algún sistema de acceso a red y un navegador puedan programar en los más diversos lenguajes, IDEs y terminales sin hacer instalación alguna en su equipo de cómputo, tableta o teléfono celular.

Algunos ejemplos de estos servicio son:

- Para R <https://nclab.com/free-portal/>
- Para R <https://cdn.datacamp.com/dcl-react-prod/example.html>

⁴⁵Cuando se trabaja desde la Web es recomendable usar el modo Privado o Incógnito para no guardar el historial de navegación, información introducida en los formularios y borrar al cerrar el navegador los datos de los sitios visitados. Pero recuerda que los sitios Web que visitamos sí guardan información de nuestra visita, nuestro proveedor de Internet también guarda constancia de nuestra visita y si descargamos algo, esto no se borra al igual que el historial de descargas, además de las marcas de páginas o favoritos se conservarán al cerrar el navegador.

8 El Cómputo en Paralelo

La computación paralela es el uso de múltiples recursos computacionales para resolver un problema. Se distingue de la computación secuencial en que varias operaciones pueden ocurrir simultáneamente. Los sistemas de cómputo con procesamiento en paralelo surgen de la necesidad de resolver problemas complejos en un tiempo razonable, utilizando las ventajas de memoria, velocidad de los procesadores, formas de interconexión de estos y distribución de la tarea, a los que en su conjunto denominamos arquitectura en paralelo. Entenderemos por una arquitectura en paralelo a un conjunto de procesadores interconectados capaces de cooperar en la solución de un problema.

Así, para resolver un problema en particular, se usa una arquitectura o combinación de múltiples arquitecturas (topologías), ya que cada una ofrece ventajas y desventajas que tienen que ser sopesadas antes de implementar la solución del problema en una arquitectura en particular. También es necesario conocer los problemas a los que se enfrenta un programador que desea que su programa corra en paralelo, como son: el partir eficientemente⁴⁶ un problema en múltiples subtareas y cómo distribuir eficazmente⁴⁷ estas según la arquitectura en particular con que se trabaje. La eficacia difiere de la eficiencia en el sentido de que la eficiencia hace referencia en la mejor utilización de los recursos computacionales (procesadores), en tanto que la eficacia hace referencia en la capacidad para alcanzar un objetivo, aunque en el proceso no se haya hecho el mejor uso de los recursos computacionales.

El cómputo de alto rendimiento (HPC por High-Performance Computing) está formado por un conjunto de computadoras unidas entre sí en forma de Cluster⁴⁸, para aumentar su potencia de trabajo y rendimiento. En 2019 la supercomputadora *SUMMIT* de IBM funcionaban a más de 148 peta-Flops⁴⁹ (cada uno de ellos equivale a la realización de más de 1000 billones

⁴⁶Es un indicador de la utilización efectiva de los diferentes recursos computacionales (principalmente del uso de los procesadores) en relación al algoritmo dado. Se entiende que la eficacia se da cuando se utilizan menos recursos para lograr un mismo objetivo. O al contrario, cuando se logran más objetivos con los mismos o menos recursos.

⁴⁷La podemos definir como el nivel de consecución de metas y objetivos. La eficacia hace referencia a nuestra capacidad de lograr lo que no proponemos.

⁴⁸Existe el Ranking de las 500 supercomputadoras más poderosas del mundo (esta se actualiza cada seis meses en junio y noviembre) y puede ser consultada en:

<https://top500.org>

⁴⁹“FLOPS” operaciones de punto flotante por segundo. Describe una velocidad de procesamiento teórica: para hacer posible esa velocidad es necesario enviar datos a los

de operaciones por segundo de las pruebas de rendimiento del HPC-AI), mientras que en junio de 2020 *FUGAKU* de Japón superaba los 415.5 peta-Flops y en noviembre del mismo año alcanzó los 2.0 exa-Flops, siendo este el primer equipo en alcanzar valores por arriba de un exa-Flops para cualquier precisión sobre cualquier tipo de Hardware, manteniéndose en el primer lugar del top 500 durante el 2021

El clásico uso del paralelismo, es el de diseño de programas eficientes en el ámbito científico. La simulación de problemas científicos es un área de gran importancia, los cuales requieren de una gran capacidad de procesamiento y de espacio de memoria, debido a las complejas operaciones que se deben realizar.

Otro uso clásico es el de las gráficas y vídeos de simulaciones generadas por computadora. La generación de fotogramas y videos requiere de una gran cantidad de cálculos matemáticos. Esto supone una tarea muy compleja para un solo procesador, luego es necesario que haya algún tipo de paralelismo, para distribuir la tarea para que esta sea realizada eficiente y eficazmente.

Tradicionalmente, los programas informáticos se han escrito para el cómputo en secuencial. Para resolver un problema, se construye un algoritmo y se implementa como un flujo en serie de instrucciones. Estas instrucciones se ejecutan en una unidad central de procesamiento en un ordenador. Sólo puede ejecutarse una instrucción a la vez y un tiempo después de que la instrucción ha terminado, se ejecuta la siguiente.

Actualmente, es una práctica común usar directivas de compilación en equipos paralelos sobre programas escritos de forma secuencial, con la esperanza que sean puestos por el compilador como programas paralelos. Esto en la gran mayoría de los casos genera códigos poco eficientes, pese a que corren en equipos paralelos y pueden usar toda la memoria compartida de dichos equipos, el algoritmo ejecutado continúa siendo secuencial en una gran parte del código.

La computación en paralelo, por el contrario, utiliza simultáneamente múltiples elementos de procesamiento para resolver un problema. Esto se logra mediante la división del problema en partes independientes de modo que cada elemento de procesamiento pueda ejecutar su parte del algoritmo de manera simultánea con los otros. Los elementos de procesamiento son di-

procesadores de forma continua. Por lo tanto, el procesamiento de los datos se debe tener en cuenta en el diseño del sistema. La memoria del sistema, junto con las interconexiones que unen los nodos de procesamiento entre sí, impactan en la rapidez con la que los datos llegan a los procesadores.

versos e incluyen recursos tales como una computadora con múltiples procesadores, varios ordenadores en red, Hardware especializado, o cualquier combinación de los anteriores.

Los programas informáticos paralelos son más difíciles de escribir que los secuenciales, porque la concurrencia introduce nuevos tipos de errores de Software, siendo las condiciones de sincronización las más comunes. La comunicación y sincronización entre diferentes subtareas son algunos de los mayores obstáculos para obtener un buen rendimiento del programa paralelo.

¿Cómo Paralelizo mi programa? El problema de todos los usuarios de Ciencias e Ingeniería que hacen uso de métodos numéricos y su implementación computacional es: ¿cómo paralelo mi programa?, esta pregunta no tiene una respuesta simple, depende de muchos factores, por ejemplo: en que lenguaje o paquete está desarrollado el programa, el algoritmo usado para solucionar el problema, a que equipos paralelo tengo acceso y un largo etc. Algunas respuestas ingenuas podrían ser:

- Si uso lenguajes de programación compilables, puedo conseguir un compilador que permita usar directivas de compilación en equipos de memoria compartida sobre programas escritos de forma secuencial, con la esperanza que sean puestos por el compilador como programas paralelos haciendo uso de hilos, OpenMP y optimización del código generado.
- Si uso paquetes como MatLab, Julia o Python, es posible conseguir una versión del paquete que implemente bibliotecas que usen CUDAs, OpenMP o MPI para muchos de los algoritmos más usados en la programación.
- Si mi problema cabe en un sólo equipo, entonces puedo usar dos o más cores para resolver mi problema usando memoria compartida (puedo usar OpenMP, trabajar con hilos o usando paquetes como MatLab o lenguajes como Python, Julia, Fortran, C y C++, etc.), pero sólo puedo escalar para usar el máximo número de cores de un equipo (que en la actualidad puede ser del orden de 128 cores, hasta Terabytes de RAM y con almacenamiento de algunas decenas de Terabytes).
- Si mi problema cabe en la GRAM de una GPU, entonces el posible usar una tarjeta gráfica (usando paquetes como MatLab o lenguajes como

Python, Julia, Fortran, C y C++, etc.), pero sólo puedo escalar a la capacidad de dichas tarjetas gráficas.

- Puedo usar un cluster que usa memoria distribuida-compartida en conjunto con tarjetas gráficas, este tipo de programación requiere una nueva expertes y generalmente implica el uso de paso de mensajes como MPI y rediseño de los algoritmos usados en nuestro programa.

Notemos primero, que no todos los algoritmos son paralelizables. En cualquier caso se tienen que ver los pros y contras de la paralelización para cada caso particular. Pero es importante destacar que existen una gran cantidad de bibliotecas y paquetes que ya paralelizan ciertos algoritmos que son ampliamente usados como la solución de sistemas lineales⁵⁰ y no lineales. Además, el tiempo de programación necesario para desarrollar una aplicación paralela eficiente y eficaz para la gran mayoría de los programadores puede ser de semanas o meses en el mejor de los casos. Por ello, es necesario hacer un balance entre las diferentes opciones de paralelización para no invertir un tiempo precioso que puede no justificar dicha inversión económica, de recursos computacionales y sobre todo de tiempo.

Antes de iniciar con los tópicos del cómputo en paralelo, es necesario conocer cómo está constituida nuestra herramienta de trabajo: la computadora. Los conocimientos básicos de arquitectura de las computadoras nos permite identificar los componentes de Hardware que limitarán nuestros esfuerzos por realizar una paralelización óptima de nuestro programa y en muchos casos descubriremos de la peor manera que los algoritmos usados en nuestro programa no son los más adecuados para paralelizar; y descubriremos con horror que en algunos casos los tiempos de ejecución no mejoran sin importar cuantos equipos adicionales usemos, en el peor de los casos, el tiempo de ejecución se incrementa al aumentar equipos en vez de disminuir.

⁵⁰ Algunas de las bibliotecas más usadas para resolver sistemas lineales usando matrices bandadas y dispersas son PETCs, HYPRE, ATLAS, LAPACK++, LAPACK, EISPACK, LINPACK, BLAS, entre muchas otras alternativas, tanto para implementaciones secuenciales como paralelas y más recientemente para hacer uso de los procesadores CUDA en las GPU de nVidia.

8.1 Computadoras Actuales

La computadora (también conocida como ordenador) actual es una máquina digital programable que ejecuta una serie de comandos para procesar los datos de entrada, obteniendo convenientemente información que posteriormente se envía a las unidades de salida. Una computadora está formada físicamente por numerosos circuitos integrados y varios componentes de apoyo, extensión y accesorios, que en conjunto pueden ejecutar tareas diversas con suma rapidez y bajo el control de un programa (Software).

La constituyen dos partes esenciales, el Hardware, que es su estructura física (circuitos electrónicos, cables, gabinete, teclado, etc.), y el Software, que es su parte intangible (programas, datos, información, documentación, etc.).

Con respecto al Hardware⁵¹, se encuentra compuesto por una serie de dispositivos, clasificados según la función que estos desempeñen. Dicha clasificación se compone de:

- Los dispositivos de entrada son todos aquellos que permiten la entrada de datos a una computadora. Estos dispositivos (periféricos), son los que permiten al usuario interactuar con la computadora. Ejemplos: teclado, Mouse (ratón), micrófono, Webcam, Scanner, etc.
- Los dispositivos de salida, son todos aquellos que permiten mostrar la información procesada por la computadora. Ejemplos: monitor, impresora, auriculares, altavoces, etc.
- Los dispositivos de comunicación son aquellos que permiten la comunicación entre dos o más computadoras. Ejemplos: Modem, Router, placa de red, Bluetooth, etc.

⁵¹En Debian GNU/Linux podemos instalar la aplicación *lshw* para conocer los distintos componentes de la computadora, mediante:

```
# apt install lshw
```

Así, para ver de forma resumida los dispositivos que componen la computadora usamos:

```
# lshw -short
```

Si necesitamos más detalle usamos:

```
# lshw
```

- Los dispositivos de almacenamiento, son todos aquellos que permiten almacenar datos en el ordenador. Ejemplos: disco duro, Pendrive, Diskette, CD y DVD, etc.
- Los dispositivos de cómputo, son aquellos encargados de realizar las operaciones de control necesarias, sobre el resto de los dispositivos la computadora.

La unidad central de procesamiento (Central Processing Unit CPU) se comunica a través de un conjunto de circuitos o conexiones llamada Bus de datos o canal de datos. El bus conecta la CPU a los dispositivos de almacenamiento, los dispositivos de entrada y los de salida.

En la actualidad, una computadora puede tener una CPU de 32 o de 64⁵² bits para describir anchura de registros, Bus de direcciones, Bus de datos o instrucciones. Es decir que la cantidad de bits nos permite diferenciar el tipo de CPU que tiene nuestro equipo.

La CPU, el sistema operativo, el Software en general y los Drivers se basan en una misma arquitectura, pero que tenga más o menos bits puede ser significativo en la experiencia del usuario.

⁵²¿Por qué se llama x86 a la de 32 bits y x64 a la de 64 bits?

Por pereza. El término adecuado para la arquitectura de 64 bits basada en las tecnologías desarrolladas por Intel y AMD desde 1976 es x86-64, que es como la llamó AMD cuando la desarrolló.

El término "x86" se acuñó porque Intel denominó 8086 a la primera CPU que utilizaba esta microarquitectura. Era una CPU de 16 bits con un conjunto de instrucciones similar al anterior Intel 8085 de 8 bits, por lo que Intel le dio un número más alto. A principios de los años 90, a los desarrollos posteriores de la 8086 se les asignaron números que terminaban en "86": 80186, 80286, 80386 y 80486. En la mayoría de los casos, se omitió la parte "80" y toda la familia de CPU se denominó "x86", por razones obvias. Con lo que habría sido el 80586, Intel eliminó los números porque no podían registrarlos como marca, así que nació la marca "Pentium", que primero significaba quinta generación, pero luego como una simple marca.

Cuando Intel pasó a los 32 bits con el conjunto de instrucciones x86 con el 386 en 1985, no cambiaron la marca de nada, pero cuando AMD extendió x86 a 64 bits, lo comercializaron como x86-64. Muchos desarrolladores de Software lo llamaron "AMD64" en su lugar, ya que AMD lo desarrolló. Intel no estaba dispuesto a hacer eso, así que lo llamaron "EMT64" cuando adoptaron las extensiones de 64 bits de AMD después del fracaso de Itanium. Al final, la mayoría de la gente simplemente lo llamó "x64", aunque esto no tiene ningún sentido.

En la actualidad, la mayoría de ordenadores cuentan con CPU de 64⁵³ bits. Ahora bien, todavía hay equipos algo antiguos que cuentan con la arquitectura de 32 bits, por lo que sigue coexistiendo en el mercado y el sector hace desarrollos también pensando en sus capacidades.

Desde el punto de vista funcional es una máquina que posee, al menos, una unidad central de procesamiento, unidad de memoria (Random Access Memory RAM, Read Only Memory ROM y Caché) y dispositivos de entrada/salida (periféricos). Los periféricos de entrada permiten el ingreso de datos, la CPU se encarga de su procesamiento (operaciones aritmético-lógicas) y los dispositivos de salida los comunican a los medios externos. Es así, que la computadora recibe datos, los procesa y emite la información resultante, la que luego puede ser interpretada, almacenada, transmitida a otra máquina o dispositivo o sencillamente impresa; todo ello a criterio de un operador o usuario y bajo el control de un programa de computación.

⁵³¿Alguna empresa ha fabricado una CPU de 128 bits?

Si alguien quiere fabricar una CPU de 128 bits, puede utilizar el diseño RISC-V, que admite núcleos de 32, 64 y 128 bits. No hay problema, solo se necesita un diseño adecuado y listo. Muchos dicen que el beneficio de una CPU (núcleo) de 64 bits es la enorme memoria direccionable, 64 bits en comparación con los 32 bits de los núcleos de 32 bits. ¡Pero eso no es cierto!. Todos los núcleos de 64 bits (excepto los muy especiales) tienen un bus de direcciones de 52 o 56 bits. Windows y Linux utilizan internamente un espacio de direccionamiento de a lo más 48 bits. No hay necesidad de memoria física de 64 bits porque hoy en día no es posible tenerla (teóricamente hasta 16 Exabytes).

Algo similar ocurre con los núcleos de 128 bits. Como la memoria no es importante, lo único que queda es la matemática de 128 bits. La pregunta es: ¿quién necesita la matemática de 128 bits? En 1978, el 8086 tenía un coprocesador matemático, el 8087, que admitía tipos de punto flotante de 80 bits. Hoy en día, es similar, pero el tipo máximo admitido sigue siendo de 64 bits que tiene aproximadamente 16 dígitos decimales de precisión con un rango del orden de 1.7×10^{-308} a 1.7×10^{308} , el número de 32 bits que tiene 14 dígitos decimales de precisión con un rango del orden de 3.4×10^{-38} a 3.4×10^{38} . Además, existe el número de 80 bits que tiene 18 dígitos decimales de precisión con un rango del orden de 3.4×10^{-4096} a 1.1×10^{4096} .

En algunos compiladores existen 128 bits de datos, pero no existe matemática de Hardware, sino que se implementa en Software. Se necesitan tipos tan grandes en muy pocas circunstancias.

Lo que importa hoy en día es la paralelización, el procesamiento de múltiples tipos en paralelo. Durante mucho tiempo, x86 ha tenido AVX512, que es un motor matemático de 512 bits (reemplazó al 8087), pero es capaz de trabajar con tipos de 8 a 64 bits en paralelo, por ejemplo, 8 operaciones matemáticas en paralelo utilizando FP64 (punto flotante de 64 bits).

CPU la Unidad Central de Proceso (Central Processing Unit CPU) es aquella parte del procesador que se encarga de ejecutar las diversas acciones que ordenemos al dispositivo que debe llevar a cabo. La CPU es el componente básico dentro de todo dispositivo inteligente, ya que prácticamente cualquier proceso que se ordene al sistema pasa por él. Con el paso del tiempo, además, su eficiencia y calidad ha alcanzado grandes cotas, aunque tecnologías al alza como las NPU han supuesto un mayor salto cualitativo que el de aumentar la potencia bruta de la CPU.

GPU, para el procesamiento gráfico la Unidad de Procesamiento Gráfico (Graphics Processing Unit GPU) es el apartado que se dedica a las acciones de mayor peso: las de componente gráfico. De este modo, acciones como la ejecución de videojuegos, o la edición y renderizado de vídeos, se llevan a cabo a través de la GPU del sistema. La calidad del teléfono, ordenador u otro dispositivo inteligente, suele ir supeditada a menudo a la calidad de su GPU, que dependerá de la banda de precio del aparato en cuestión. Eso sí, si los otros componentes no tienen la misma calidad, se puede producir el temido cuello de botella.

En los smartphones, la GPU ya va integrada en los procesadores, pero en los PC's, como AMD y NVIDIA se muestran como marcas especializadas en las GPUs. Cuentan con todo tipo de gamas y también poseen un amplio abanico de precios, un valor siempre directamente proporcional a la calidad y capacidad de sus gráficas, y a su vigencia en el mercado.

NPU, redes neuronales en tu dispositivo la Unidad de Procesamiento Neuronal (Neural Processing Unit NPU), a diferencia de la GPU, que cuenta con un funcionamiento paralelo al de la CPU, puede encargarse de funciones similares a las de la CPU, pero lo hace de un modo mucho más eficiente. Impulsada por Inteligencia Artificial, una NPU⁵⁴ es capaz de priorizar procesos para ejecutarlos de un modo exponencialmente más veloz y con un consumo mucho menor. En móviles, se usa especialmente para mejorar el procesado de fotografías, aunque participa en muchos otros procesos.

Además, esta arquitectura todavía tiene años de progreso por delante, a

⁵⁴Para muestra, la compañía Cerebras con su procesador WSE-3 aglutina 4 billones de transistores, tiene una superficie de 46,225 mm², integra nada menos que 900,000 núcleos optimizados para IA y es capaz de entrenar hasta 24,000 millones de parámetros, lo que también equivaldría a un rendimiento máximo de IA de 125 petaflops.

diferencia de la CPU y la GPU, cuyas mejoras ya son de carácter más leve y basadas en aumentar la potencia bruta. La tecnología NPU, en cambio, lleva menos tiempo entre nosotros y todavía tiene mucho margen de mejora para ofrecer un rendimiento cada vez más poderoso.

Desde la llegada de los procesadores de más de un núcleo a PC, como consecuencia de la imposibilidad de hacerlos escalar en potencia solo por velocidad de reloj, la forma de entender los diferentes chips cambió. Han pasado ya dos décadas de dicho cambio y ante la inminente salida de los Chips disgregados o por Chiplets al mercado de masas, no está de más recordar la organización más común en el mundo del Hardware en todo este tiempo.

¿Qué es un SoC? las siglas SoC significan System on a Chip y hace referencia a todo chip que tiene la mayoría de componentes integrados en una misma pieza de silicio sin llegar a ser un microcontrolador. Se trata de la pieza de Hardware más usada por el hecho de que a día de hoy todo procesador para PC, teléfono móvil, consola de videojuegos, televisor o incluso servidores, es un SoC y pese a las diferencias entre ellos, todos tienen una organización común.

En realidad, todas las CPU actuales son SoC, ya que se trata de "varias CPU" diseñadas para funcionar alrededor de un elemento de intercomunicación central. Este se encarga de interconectar los diferentes elementos entre sí y de darles acceso a interfaces externas.

Por ejemplo, con la memoria RAM (u otros), a la que está asociado el controlador de memoria que comparten todos sus elementos o a los periféricos, y a los cuales se puede acceder directamente con una serie de interfaces específicas, o a través de un Chipset externo encargado de gestionar los diferentes periféricos y componentes.

En PC, debido a que la comunicación con los periféricos se hace a través del uso de direcciones de memoria de la RAM principal, los componentes relacionados con esta se encuentran subordinados al controlador de memoria. Por lo que son una pieza más conectada a la parte central.

¿Qué es una APU y en qué se diferencia de un SoC? las siglas APU significan Accelerated Processor Unit y fue usada por AMD cuando sus CPU e iGPU (o GPU integrada, la veremos a continuación) no traían consigo ningún sistema de gestión de E/S (entrada y salida). Sin embargo, la cosa empezó a cambiar ya con la arquitectura "Carrizo" que fue el nombre

clave de los últimos SoC antes del lanzamiento de los AMD Ryzen, donde se incorporaron varias interfaces de periféricos directamente en la CPU.

A día de hoy todo es un SoC, lo que ocurre es que, en sobremesa, las torres tienen tanta conectividad y capacidad de expansión que se suele emplear un Chipset, y lo mismo ocurre en estaciones de trabajo y servidores, pero no en el resto. Y es que si hablamos de un Chip para un PC portátil, una consola o un móvil, entonces al no existir tantas interfaces para periféricos y otros componentes, entonces éstas se pueden integrar en un mismo chip.

En realidad, el término APU es más bien comercial de AMD y al día de hoy se utiliza como sinónimo de SoC, pero se puede resumir en que una APU carece de cualquier gestión de periféricos y requiere de un Chip externo para ello. Mientras que un SoC tiene las especificaciones mínimas en ese aspecto, pese a ser también ampliables. Para simplificar la idea, un SoC es una APU más completa.

¿iGPU qué es y cuáles son sus características concretas frente a una dGPU? las siglas iGPU corresponden a integrated GPU, y hace referencia a todo componente de este tipo que se encuentre integrado en una APU o un SoC. Por lo que se trata de procesadores gráficos de potencia limitada que se pueden ver lastrados en velocidad de reloj por el problema del ahogamiento termal (Thermal Throttling) que se produce cuando muchas partes comparten el mismo espacio físico.

Si bien, es posible llegar a ciertos niveles de rendimiento que son aceptables de cara a reproducir las escenas en 3D a tiempo real en los videojuegos, y otras tareas de carácter profesional donde se usa una tarjeta gráfica, estas se ven cuanto menos limitadas:

El hecho de tener que compartir espacio en el mismo Chip con los diferentes núcleos de la CPU produce que esta no pueda alcanzar la misma velocidad de reloj que se alcanzaría siendo un Chip aparte e independiente.

En PC, debido a que como la memoria RAM usa DDR o LPDDR, el ancho de banda es pequeño y hemos de partir del hecho de que el rendimiento de todo Chip gráfico, incluido una iGPU, depende del ancho de banda que se le puede otorgar con dicha memoria RAM.

Los SoC con iGPU actuales tienen un tamaño fijo, definido este por la cantidad de pines soportados por la interfaz con la placa base. Esto limita el tamaño, no solo del Chip, sino también de la gráfica integrada en el mismo.

iGPU en consolas de videojuegos al contrario de lo que ocurre con los PC, las consolas de videojuegos no tienen que seguir una serie de normas respecto a sus componentes. Para empezar, no ven el tamaño de sus chips limitados por un estándar de placa base, dado que son productos únicos y exclusivos. Esto les permite tener el tamaño que quieran, incluso más que una CPU convencional para PC, lo que les permite tener una iGPU en su SoC mucho más avanzada.

El otro punto es la memoria utilizada, ya que la de las tarjetas gráficas también se usan como memoria principal. Esta da el ancho de banda necesario para que la iGPU alcance cierto nivel de rendimiento, pero su latencia es mucho más alta que la RAM convencional de PC que está más optimizada en ese aspecto, por lo que el rendimiento de su CPU suele ser más bajo que su equivalente para ordenador.

¿Qué es una dGPU en un PC o portátil? las dGPU, o GPU dedicadas, no son otra cosa que las GPU de toda la vida, pero la particularidad es que también son SoC, ya que tiene varios núcleos, especializados en tareas gráficas, alrededor de una interfaz central y compartiendo todos ellos un mismo acceso a memoria.

Sin embargo, carecen de núcleos de CPU en su interior, de ahí a que no se les llame APU o SoC, por el hecho de que de existir estos elementos pasarían a ser una iGPU. Por lo tanto, su principal particularidad de las dGPU es que tienen su propia memoria RAM (SDRAM para ser concretos) la cual suele rodear estos Chips, ya sea en forma de tarjeta gráfica o soldados en la placa de los ordenadores portátiles. A esta la llamamos VRAM y es de uso exclusivo de la dGPU o GPU.

Los equipos de cómputo los podemos clasificar⁵⁵ por:

- Equipos móviles: estos equipos buscan un equilibrio entre su capacidad de cómputo versus el rendimiento energético de sus baterías -para operar el mayor tiempo posible sin recargarse- y su peso, entre estos equipos destacan las Laptops, Notebook, Netbook, Ultrabook, tabletas, teléfonos inteligentes, etc.

⁵⁵El ordenador del Apollo 11, el Block II, funcionaba a una velocidad de 2 MHz y tenía 2 KB de memoria RAM y 32 KB de memoria ROM. Para ponerlo en contexto, el chip de un cargador USB-C moderno es 563 veces más potente que la computadora que se usó en el Apollo 11, al menos en términos de potencia bruta.

- Equipos de escritorio: estos equipos al estar permanentemente conectados a la corriente eléctrica pueden tener un mayor número de componentes y disponen de una mejor capacidad de disipación de calor por lo que pueden contener una mayor cantidad de componentes, como discos, RAM o tarjetas de video y el tamaño, peso o consumo energético no es un inconveniente.
- Servidores: son equipos que suelen atender a múltiples usuarios simultáneamente y disponen de gran cantidad de Cores, RAM, disco y son interconectados por red de alta velocidad con otros servidores para atender las crecientes necesidades de los centros de datos los cuales deben estar permanentemente en operación. Generalmente los equipos son montados en Racks con otras decenas de equipos, por lo que su arquitectura se ve limitada a una moderada generación de calor por parte de sus componentes ya que su sistema de ventilación es por aire para todo el Rack.
- Estaciones de trabajo: son equipos individuales diseñados para atender cargas computacionales intensas, por lo que requieren Hardware más complejo y potente como puede ser múltiples tarjetas de video (con decenas de miles de Cores gráficos), discos (con cientos de Terabytes), gran cantidad de RAM (pueden llegar a superar el Terabyte) y sistema de refrigeración por aire o líquido, etc.
- Cómputo intensivo: son equipos interconectados por red de alta velocidad con procesadores y tarjetas gráficas dedicadas para el cálculo numérico que soportan cargas intensas por largos periodos de tiempo, los más comunes son los que forman parte de los Cluster que llegan a tener millones de cores.

FLOPS Una medida relativamente objetiva para analizar el rendimiento de un dispositivo suele ser medir sus operaciones de punto flotante por segundo o más conocidas como FLOPS⁵⁶. Hay que tener en cuenta que la medición de FLOPS es muy compleja porque las diferentes operaciones en punto flotante

⁵⁶El Cray-1 fue puesto marcha en 1975 y utilizaba una CPU a 80 MHz y llevaba integrada una unidad SIMD de 64 bits de precisión de punto flotante, lo cual fue un salto de gigante que permitió un salto de los 3 MFLOPS de potencia del CDC 6600 a los 160 MFLOPS en el Cray-1.

llevan diferentes cantidades de tiempo para ejecutarse. Y no todo el mundo utiliza las mismas operaciones para establecer los cálculos.

Por ejemplo, una división simple como $1/5$, toma significativamente menos tiempo que el cálculo del logaritmo de 5. Por eso, se estableció el algoritmo de Linpack como un estándar representativo con el que poder medir todos los sistemas bajo el mismo baremo de FLOPS-.

Es importante señalar que el algoritmo de Linpack utiliza el formato en punto flotante de doble precisión (64 bits⁵⁷ que tiene aproximadamente 16 dígitos decimales de precisión con un rango del orden de 1.7×10^{-308} a 1.7×10^{308}). Sin embargo, como veremos la mayoría de los valores que dan los fabricantes son con precisión simple (32 bits que tiene 14 dígitos decimales de precisión con un rango del orden de 3.4×10^{-38} a 3.4×10^{38}). Además, los valores que dan los fabricantes suelen ser teóricos y en la práctica suelen ser inferiores debido a otros factores limitantes como la frecuencia de reloj o la velocidad de las memorias ROM y RAM.

Por tanto, aunque todos hemos acabado midiendo el rendimiento en FLOPS, no es una medida absoluta de la potencia del CPU ni de una GPU. Por ejemplo para algunos dispositivos tenemos:

Móviles El SoC Snapdragon 821 que monta una GPU Adreno 530 tiene una potencia de 519.2 gigaFLOPS (0.52 TFLOPS), y los Chips Apple A9X del iPad Pro alcanzan los 345.6 gigaFLOPS (0.35 TFLOPS), todos ellos

⁵⁷¿Por qué no hay un procesador de 128 bits, si los hay de 64 y 32 bits?

En gran medida, esto se reduce a la cuestión de qué se entiende realmente por "procesador de 64 bits". Al inicio de las computadoras electrónicas, un "procesador de 8 bits" tenía registros de 8 bits, pero la mayoría tenía direccionamiento de 16 bits (y la Unidad de Aritmética y Lógica ALU a veces tenía 4 bits, por lo que cuando se añadían dos registros de 8 bits, la CPU a menudo lo hacía en dos pasos, añadiendo los 4 bits inferiores y luego añadiendo por separado los 4 bits superiores más el acarreo de los 4 inferiores).

Hoy en día, hemos invertido un poco esa situación: un procesador de "64 bits" (más o menos) tiene direccionamiento de 64 bits, pero tiene registros de 128 bits, 256 bits y, en algunos casos, incluso de 512 bits. Pero como los usos de números mayores a 64 bits son bastante raros, esos registros más grandes se usan normalmente para realizar operaciones en una cantidad de operandos más pequeños con una sola instrucción. Por lo tanto, en lugar de una ALU que tiene la mitad del tamaño de un registro y realiza una sola operación en múltiples ciclos de reloj, tenemos una ALU que tiene entre 2 y 4 veces el tamaño de un registro normal y realiza múltiples operaciones en un solo ciclo de reloj.

Dependiendo de cómo prefieras ver las cosas, podrías argumentar razonablemente que una CPU actual de "64 bits" es en realidad una CPU de 128 bits, 256 bits o incluso 512 bits.

medidos con precisión simple de 32-bits.

CPU

- Intel Xeon W-3245: 1.4 TFLOPS
- Intel Core i9-9900X: 1.2 TFLOPS
- AMD Ryzen 9 3950X: 1.1 TFLOPS

Los procesadores de gama media-alta rondan el medio TFLOPS:

- AMD Ryzen 7 3700X: 546.0 GFLOPS - 0.55 TFLOPS
- Intel Core i9-9900: 499.0 GFLOPS - 0.50 TFLOPS
- AMD Ryzen 5 3600X: 461.0 GFLOPS - 0.46 TFLOPS

Tarjetas Gráficas Ojo: La tabla está ordenada por los valores en precisión simple (32-bit) primer columna

GPU	FP32 TFLOPS	FP64 TFLOPS
TITAN V	13.8	6.9
Radeon RX Vega 64	12.7	0.8
GeForce GTX 1080 Ti	11.3	0.4
GeForce GTX 1080	8.9	0.3
Radeon R9 Fury X	8.6	0.5
Radeon HD 7990	7.8	1.9
GeForce GTX 1070	6.5	0.2
Radeon RX 480	5.8	0.4
GeForce GTX 690	5.6	0.2
Radeon R9 290X	5.6	0.7
GeForce GTX 780 Ti	5.3	0.2
Radeon HD 6990	5.1	1.3
GeForce GTX 980	4.9	0.15
Radeon RX 470	4.9	0.3
Radeon R9 290	4.8	0.6
GeForce GTX Titan	4.7	1.5
GeForce GTX 1060	4.4	0.14
Radeon HD 7970 GHz	4.3	1.1
GeForce GTX 780	4.1	0.17

Radeon R9 280X	4.0	1.0
Radeon R9 280	3.3	0.83
GeForce GTX 680	3.1	0.13
Radeon HD 7950	2.9	0.71

Como podemos ver, las tarjetas gráficas de Nvidia, normalmente, tienen una potencia muy alta en precisión simple, pero muy mala en precisión doble. La precisión simple es la que se usa en los juegos, pero la precisión doble es la que se utiliza en los cálculos complejos científicos y en el minado de muchas criptomonedas.

Consolas Todas ellas son en valores de precisión simple (32-bit)

- PlayStation 4: 1.3 TFLOPS
- Xbox One: 1.8 TFLOPS
- PlayStation 4 Pro: 4.2 TFLOPS
- Nintendo Switch: entre 0.4 y 0.5 TFLOPS
- PlayStation 5 promete una GPU con 10.28 TFLOPS
- La Xbox Series X promete una GPU de 12 TFLOPS

SuperCómputo Hace pocos se publicó la 64ª edición del ranking «Top 500» (noviembre del 2024) que clasifica los Clusters con mayor rendimiento del mundo. Esta lista se basa en la medida del rendimiento de los sistemas utilizando la prueba de referencia LINPACK, que calcula la velocidad a la que un sistema puede resolver un conjunto de ecuaciones lineales.

Y en esta nueva edición, los Clusters que ocuparon los primeros tres lugares son:

- El sistema El Capitan del Laboratorio Nacional Lawrence Livermore, en California, EE.UU., es el nuevo sistema número 1 del TOP500. El sistema HPE Cray EX255a ha obtenido 1.742 exaflop/s en el benchmark HPL. El Capitan tiene 11,039,616 núcleos y está basado en procesadores AMD EPYC(TM) de 4ª generación con 24 núcleos a 1.8 GHz y aceleradores AMD Instinct(TM) MI300A. Utiliza la red Cray Slingshot 11 para la transferencia de datos y alcanza una eficiencia energética de 60.3 Gigaflops/vatio.

- Frontier es ahora el sistema número 2 del TOP500. Este sistema HPE Cray EX fue el primer sistema estadounidense con un rendimiento superior a un exaflop/s. Está instalado en el Laboratorio Nacional Oak Ridge (ORNL) en Tennessee, EE.UU., donde se utiliza para el Departamento de Energía (DOE). Actualmente ha alcanzado 1.353 Exaflop/s utilizando 8,699,904 núcleos. La arquitectura HPE Cray EX combina CPUs AMD EPYC(TM) de tercera generación optimizadas para HPC e IA, con aceleradores AMD Instinct(TM) 250X y una interconexión Slingshot-11.
- Aurora es actualmente el número 3 con una puntuación HPL preliminar de 1.012 Exaflop/s. Está instalado en Argonne Leadership Computing Facility, Illinois, EE. UU., donde también se opera para el Departamento de Energía (DOE). Este nuevo sistema de Intel se basa en HPE Cray EX - Intel Exascale Compute Blades. Utiliza procesadores Intel Xeon CPU Max Series, aceleradores Intel Data Center GPU Max Series y una interconexión Slingshot-11.

Para poner en contexto los avances en este campo, en el año 2004 IBM era dueña y señora del mundo de la supercomputación, su espectacular BlueGene/L dominaba la lista TOP 500. Aquel monstruo contaba con 32,768 procesadores PowerPC 440 a 700 MHz y 16 TB de memoria. 20 años después una sola NVIDIA GeForce RTX 4090 con 24 GB de memoria GDDR6X es más potente que esa supercomputadora -lo es al menos en rendimiento bruto-, BlueGene/L contaba en ese momento con un rendimiento de 70.72 TFLOPS, pero la propia NVIDIA dejaba claro en el lanzamiento de sus RTX 4090 que estas tarjetas gráficas contaban con una potencia de 83 TFLOPS.

Es más, cuatro RTX 4090 con soporte FP8 logran también rivalizar con la supercomputadora más potente de 2009. Y eso sin apretarle las tuercas a las RTX 4090: en noviembre de 2022 es precisamente lo que hicieron en Wccftch y lograron que la RTX 4090 se convirtiera en la primera tarjeta gráfica del mundo en alcanzar los 100 TFLOPS.

Esa comparación es como decimos real en esa potencia de cálculo en bruto, pero también es cierto que en esa y otras supercomputadoras se tenían mecanismos especiales de comunicación entre procesadores o de transferencia de datos, algo para lo que las GPUs actuales, aún siendo sobresalientes, no están tan optimizadas.

¿Cómo Trabaja una Computadora? Todas las computadoras sean de uno o más procesadores ejecutan los programas realizando los siguientes pasos:

1. Se lee una instrucción
2. Se decodifica la instrucción
3. Se encuentra cualquier dato asociado que sea necesario para procesar la instrucción
4. Se procesa la instrucción
5. Se escriben los resultados

Esta serie de pasos, simple en apariencia, se complican debido a la jerarquía de memoria RAM, en la que se incluye la memoria Caché, la memoria principal y el almacenamiento no volátil como pueden ser los discos duros o de estado sólido (donde se almacenan las instrucciones y los datos del programa), que son más lentos que el procesador en sí mismo. Con mucha frecuencia, el paso (3) origina un retardo muy largo (en términos de ciclos del procesador) mientras los datos llegan en el bus de la computadora.

Durante muchos años, una de las metas principales del diseño microinformático ha sido la de ejecutar el mayor número posible de instrucciones en paralelo, aumentando así la velocidad efectiva de ejecución de un programa. No obstante, estas técnicas han podido implementarse en Chips semiconductores cada vez más pequeños a medida que la fabricación de estos fue progresando y avanzando, lo que ha abaratado notablemente su costo.

El procesador es el cerebro de un ordenador. No hay que olvidar otros componentes como la memoria, el almacenamiento o la tarjeta gráfica dedicada, desde luego, pero el procesador está un escalafón por encima en la jerarquía.

Piensa que, si cambiamos el procesador en dos equipos con la misma memoria, almacenamiento o tarjeta gráfica, el comportamiento puede variar notablemente. Sin embargo, para un mismo procesador, los cambios en el resto de componentes no impactan de forma tan directa en la experiencia de uso de un equipo.

¿Qué es una CPU? Antes de nada, vamos a definir exactamente lo que es una CPU o un procesador. Como bien indican sus siglas en inglés (Central Processing Unit) es la unidad de procesamiento -puede ser Intel, AMD, ARM, etc- encargada de interpretar las instrucciones de un Hardware haciendo uso de distintas operaciones aritméticas y matemáticas. Características principales de un procesador:

- Frecuencia de reloj. Este primer término hace referencia a la velocidad de reloj que hay dentro del propio procesador. Es un valor que se mide en Mhz o Ghz y es básicamente la cantidad de potencia que alberga la CPU. La mayoría de ellas cuentan con una frecuencia base -para tareas básicas- y otra turbo que se utiliza para procesos más exigentes -con un aumento en el consumo de energía y por ende un aumento en la temperatura del procesador, requiriendo sistemas de disipación de calor eficientes-.
- Consumo energético. Es normal que nos encontremos con CPU 's donde su consumo energético varía notablemente. Es un valor que se muestra en vatios (W) y como es obvio, aquellos procesadores de gama superior, serán más propensos a consumir más energía. Ante esto, es importante contar con un eficiente sistema de enfriamiento además de contar con una fuente de alimentación acorde a la potencia requerida por el procesador, la tarjeta gráfica y sus respectivos sistemas de enfriamiento.
- Número de núcleos. Con el avance de la tecnología, ya es posible encontrar tanto procesadores de Intel como de AMD que cuentan ya con decenas de núcleos. Estos cores son los encargados de llevar a cabo multitud de tareas de manera simultánea.
- Número de hilos. Si un procesador tiene Hyperthreading en el caso de Intel o SMT (Simultaneous Multi-Threading) en el caso de AMD, significa que cada uno de los núcleos es capaz de realizar dos tareas de manera simultánea, lo que se conoce como hilos de proceso. Por lo tanto, un procesador de cuatro núcleos físicos con Hyperthreading tendría ocho hilos de proceso, y sería capaz de ejecutar ocho órdenes al mismo tiempo -los hilos no tienen las mismas capacidades de un core real y en muchos casos su uso merma el rendimiento del CPU, pero los sistemas operativos los reconocen como si fueran cores reales-⁵⁸.

⁵⁸El AMD EPYC 9845 de 160 núcleos y 320 hilos a una frecuencia de 2,00 GHz basada

- **Memoria Caché.** A la hora de "recordar" cualquier tarea, el propio ordenador hace uso de la memoria RAM. Sin embargo no es eficiente este proceso y por tanto es necesario que utilice la memoria Caché de la CPU para paliar esta deficiencia. El Caché se caracteriza porque se llega a ella de forma más rápida y puede ser tipo L1, L2 y L3.
- **Zócalo.** Es el tipo de conector con pines o Socket al que se conecta la placa base. Por ejemplo, las últimas de Intel suelen tener el Socket LGA 1200, mientras que las de AMD con Ryzen son AM4.
- **Red.** Si bien la red es un recurso indispensable en un equipo de cómputo, en el caso de equipos paralelos la velocidad de la red es el mayor cuello de botella en cuanto a rendimiento, por ello es necesario usar redes de alto desempeño como las de InfiniBand con un alto costo económico pero de alto desempeño que pueden llegar al orden de cientos de Gigabytes por segundo.

Nuevos Procesadores la creciente demanda de dispositivos de cómputo ha generado una gran variedad de procesadores, los podemos clasificar como:

- Procesador compuesto por múltiples núcleos de alta eficiencia -con un consumo energético reducido- que sacrifican potencia de procesamiento en aras de extender la carga útil de las baterías de los dispositivos móviles.
- Procesador compuesto por múltiples núcleos de alto rendimiento que pueden estar al tope de su capacidad sin generar excesivo calor y son especialmente usados en servidores y en cómputo intensivo.
- Procesadores compuestos por múltiples núcleos de alto rendimiento que pueden ajustar su velocidad de reloj de manera dinámica para tratar cargas de trabajo pesadas por un cierto tiempo -pues generan gran cantidad de calor-, por lo que requieren un sistema eficiente de enfriamiento, son ideales para estaciones de trabajo.
- Procesadores compuestos por múltiples núcleos híbridos que en lugar de tener un único tipo de núcleo multipropósito, estos Chips cuentan con dos grupos de núcleos. El primero de ellos, compuesto por

en Zen 5c, este se acompaña de 640 MB de caché L3.

múltiples núcleos de alta eficiencia, se encarga de procesar las tareas más livianas o en segundo plano que deba realizar un procesador, todo ello, con un consumo energético menor. El otro grupo, compuesto por múltiples núcleos de alto rendimiento, sigue una dinámica opuesta, su consumo energético es superior, pero únicamente entran en funcionamiento cuando la tarea en cuestión requiere un extra de procesamiento.

Para gestionar esta división de núcleos híbridos, se ha integrado un "Thread Director", un elemento que se encarga de determinar qué núcleo procesa cada tarea. Las compañías, además, ha modificado cómo funciona la caché de sus procesadores:

- Cada núcleo de rendimiento tiene su propia caché L2.
- Cada cluster de núcleos de eficiencia tiene una "piscina" de memoria L2 común, de la que beben todos los núcleos que sean partícipes.
- Tanto los núcleos de rendimiento como los de eficiencia tienen acceso a una "piscina" de memoria L3 común para todos ellos.

Otros de los cambios que impactarán en el desempeño de las CPUs es el aumento de velocidad y una mayor cantidad de memoria Caché, compatibilidad con memorias DDR6 y con la interfaz PCIe 6.0.

PCIe PCI Express (Peripheral Component Interconnect Express), abreviado como PCIe, es una tecnología de conexión de Hardware utilizada para la comunicación de alta velocidad entre diferentes componentes de un equipo informático. Este estándar se ha convertido en la interfaz más habitual para la conexión de tarjetas de expansión, como tarjetas gráficas que sirven para correr juegos, tarjetas de sonido, tarjetas de red y dispositivos de almacenamiento de alta velocidad.

Una de las ventajas destacadas de PCI Express es su arquitectura de canales independientes, que permiten la transferencia simultánea de datos en ambos sentidos. Cada carril tiene una tasa de transferencia específica, medida en gigabits por segundo (Gbps), y la capacidad de un slot PCIe se expresa como el número de carriles que tiene. Esto se traduce en un ancho de banda total mayor, lo que facilita la conexión de dispositivos que

requieren altas tasas de transferencia, como las tarjetas gráficas modernas o los dispositivos de almacenamiento de última generación.

Los diferentes tipos de ranuras de PCI Express según su tamaño PCIe X1 carriles 1, pines 18, PCIe x4 carriles 4, pines 32, PCIe x8 carriles 8, pines 49, PCIe x16 Carriles 16, pines 82.

Adicionalmente, también es interesante fijarse en las diferentes versiones que se han ido lanzando desde que PCIe se lanzó al mercado:

- PCIe 1.0 ancho banda 8 GB/s, velocidad de transferencia 2.5 GT/s
- PCIe 2.0 ancho banda 16 GB/s, velocidad de transferencia 5 GT/s
- PCIe 3.0 ancho banda 32 GB/s, velocidad de transferencia 8 GT/s
- PCIe 4.0 ancho banda 64 GB/s, velocidad de transferencia 16 GT/s
- PCIe 5.0 ancho banda 128 GB/s, velocidad de transferencia 32 GT/s
- PCIe 6.0 ancho banda 256 GB/s, velocidad de transferencia 64 GT/s

Características Arquitectónicas los procesadores Intel x86 admiten un formato de precisión extendido de 80 bits con un significado de 64 bits, que es compatible con el especificado en el estándar IEEE. Cuando un compilador usa este formato con registros de 80 bits para acumular sumas y productos internos, está trabajando efectivamente con un redondeo unitario de 2^{-64} en vez de 2^{-53} para precisión doble, dando límites de error más pequeños en un factor de hasta $2^{11} = 2048$.

Algunos procesadores Intel y AMD tienen una operación fusionada de multiplicación y suma (FMA), que calcula una multiplicación y una suma combinadas $x + yz$ con un error de redondeo en lugar de dos. Esto da como resultado una reducción en los límites de error por un factor 2.

Las operaciones FMA de bloques de precisión mixta $D = C + AB$, con matrices A, B, C y D de tamaño fijo, están disponibles en las unidades de procesamiento tensorial de Google, las GPU NVIDIA y en la arquitectura ARMv8-A. Para entradas de precisión media, estos dispositivos pueden producir resultados de calidad de precisión simple, lo que puede proporcionar un aumento significativo en la precisión cuando los bloques FMA se encadenan para formar un producto matricial de dimensión arbitraria.

Meltdown y Spectre El tres de enero del 2018 se dio a conocer al público, que 6 meses antes se habían detectado dos distintos fallos en los procesadores de los equipos de cómputo, comunicaciones y redes de internet que usamos. Esto para dar tiempo a los desarrolladores de procesadores y de sistemas operativos de implementar estrategias para mitigar el problema. Estos son problemas de diseño de los procesadores de Intel, AMD, IBM POWER y ARM, esto significa que procesos con privilegios bajos -aquellos que lanzan las aplicaciones de usuarios convencionales- podían acceder a la memoria del Kernel del sistema operativo⁵⁹.

Un ataque que explota dicho problema permitiría a un Software malicioso espiar lo que están haciendo otros procesos y también espiar los datos que están en esa memoria en el equipo de cómputo (o dispositivo móvil) atacado. En máquinas y servidores multiusuario, un proceso en una máquina virtual podría indagar en los datos de los procesos de otros procesos en ese servidor compartido.

Ese primer problema, es en realidad solo parte del desastre. Los datos actuales provienen especialmente de un grupo de investigadores de seguridad formados por expertos del llamado Project Zero⁶⁰ de Google. Ellos han publicado los detalles de dos ataques (no son los únicos⁶¹) basados en estos fallos de diseño. Los nombres de esos ataques son Meltdown y Spectre. Y en un sitio Web dedicado a describir estas vulnerabilidades destacan que "aunque los programas normalmente no tienen permiso para leer datos de otros programas, un programa malicioso podría explotar Meltdown, Spectre

⁵⁹En GNU/Linux, el Kernel (si usamos una versión actualizada) nos indica las fallas del procesador a las que es vulnerable, usando:

```
$ cat /proc/cpuinfo
$ lscpu
```

⁶⁰<https://googleprojectzero.blogspot.com/>

⁶¹Entre las distintas vulnerabilidades detectadas y sus variantes resaltan: Meltdown (AC, DE, P, SM, SS, UD, GP, NM, RW, XD, BR, PK, BND), Spectre (PHT, BTB, RSB, STL, SSB, RSRE), PortSmash, Foreshadow, Spoiler, ZombieLoad (1 y 2), Kaiser, RIDL, Plundervolt, LVI, Take a Way, Collide+Probe, Load+Reload, LVI-LFB, MSD, CSME, RYZENFALL (1, 2, 3, 4), FALLOUT (1, 2, 3), CHIMERA (FW, HW), MASTERKEY (1, 2, 3), SWAPGS, ITLB_Multihit, SRBDS, L1TF, etc. Más información en:

<https://cve.mitre.org>
<https://meltdownattack.com/>

y apropiarse de secretos almacenados en la memoria de otros programas". Como revelan en su estudio, la diferencia fundamental entre ambos es que Meltdown permite acceder a la memoria del sistema, mientras que Spectre permite acceder a la memoria de otras aplicaciones para robar esos datos.

Ya que Meltdown y Spectre son problemas de diseño en los procesadores, no es posible encontrar solución por Hardware para los procesadores existentes y dado que constantemente aparecen nuevas formas de explotar dichos fallos, la única manera de mantener el equipo de cómputo, comunicaciones y redes de internet a salvo es mediante Software que debe implementar las soluciones en los sistemas operativos. En particular en el Kernel de Linux se trabaja en parchar en cada versión del Kernel todos los fallos reportados, por esto y por otra gama de fallos e inseguridades es necesario mantener siempre el sistema operativo y sus aplicaciones actualizadas.

Como se había comentado anteriormente, estos problemas de diseño afectan a todos los procesadores Intel, AMD, IBM POWER y ARM. Eso incluye básicamente a todos los procesadores que están funcionando al día de hoy⁶² en nuestros equipos, ya que estos procesadores llevan produciéndose desde 1995. Afecta a una amplia gama de sistemas.

En el momento de hacerse pública su existencia se incluían todos los dispositivos que no utilizasen una versión convenientemente parcheada de IOS, GNU/Linux, MacOS, Android, Windows y Android. Por lo tanto, muchos servidores y servicios en la nube se han visto impactados, así como potencialmente la mayoría de dispositivos inteligentes y sistemas embebidos que utilizan procesadores con arquitectura ARM (dispositivos móviles, televisores inteligentes y otros), incluyendo una amplia gama de equipo usado en redes. Se ha considerado que una solución basada únicamente en Software para estas fallas alenta los equipos de cómputo entre un 20 y un 40 por ciento dependiendo de la tarea que realizan y el procesador del equipo.

Memoria RAM La memoria RAM (Random Access Memory) o memoria de acceso aleatorio es un componente físico de nuestro ordenador, generalmente instalado sobre la misma placa base. La memoria RAM es extraíble y se puede ampliar mediante módulos de distintas capacidades.

La función de la memoria RAM es la de cargar los datos e instrucciones que se ejecutan en el procesador. Estas instrucciones y datos provienen del

⁶²Solo en el año 2021 se detectaron 16 vulnerabilidades en procesadores INTEL y 31 en los procesadores AMD.

sistema operativo, dispositivos de entrada y salida, de discos duros y todo lo que está instalado en el equipo.

En la memoria RAM se almacenan todos los datos e instrucciones de los programas que se están ejecutando, estas son enviadas desde las unidades de almacenamiento antes de su ejecución. De esta forma podremos tener disponibles todos los programas que ejecutamos. Se llama memoria de acceso aleatorio porque se puede leer y escribir en cualquiera de sus posiciones de memoria sin necesidad de respetar un orden secuencial para su acceso.

De forma general existen o han existido dos tipos de memorias RAM. Las de tipo asíncrono, que no cuentan con un reloj para poder sincronizarse con el procesador. Y las de tipo síncrono que son capaces de mantener la sincronización con el procesador para ganar en eficacia y eficiencia en el acceso y almacenamiento de información en ellas. Veamos cuales existen de cada tipo.

Memorias de Tipo Asíncrono o DRAM las primeras memorias DRAM (Dinamic RAM) o RAM dinámica eran de tipo asíncrono. Se denomina DRAM por su característica de almacenamiento de información de forma aleatoria y dinámica. Su estructura de transistor y condensador hace que para que un dato quede almacenado dentro una celda de memoria, será necesario alimentar el condensador de forma periódica.

Estas memorias dinámicas eran de tipo asíncrono, por lo que no existía un elemento capaz de sincronizar la frecuencia del procesador con la frecuencia de la propia memoria. Esto provocaba que existiera menor eficiencia en la comunicación entre estos dos elementos.

Memorias de Tipo Síncrono o SDRAM a diferencia de las anteriores esta memoria RAM dinámica cuenta con un reloj interno capaz de sincronizar esta con el procesador. De esta forma se mejoran notablemente los tiempos de acceso y la eficiencia de comunicación entre ambos elementos. Actualmente todas nuestras computadoras cuentan con este tipo de memorias operando en ellos. Las principales tipos de memoria son: DDR, DDR2, DDR3, DDR4 y la nueva DDR5. Donde las tasas de transferencia (GB/s) son: DDR (2.1 - 3.2), DDR2 (4.2 - 6.4), DDR3 (8.5 - 14.9), DDR4 (17 - 25.6) y DDR5 (38.4 - 51.2).

Aparte las características propias de cada una de las diferentes memorias DDR, la característica más importante es que, por ejemplo, en la memoria

DDR4 cuatro cores pueden acceder simultáneamente a ella y en la DDR5 serán cinco cores.

Caché L1, L2 y L3 La memoria Caché es otra de las especificaciones importantes de los procesadores, y sirve de manera esencial de la misma manera que la memoria RAM: como almacenamiento temporal de datos. No obstante, dado que la memoria Caché está en el procesador en sí, es mucho más rápida y el procesador puede acceder a ella de manera más eficiente, así que el tamaño de esta memoria puede tener un impacto bastante notable en el rendimiento, especialmente cuando se realizan tareas que demandan un uso intensivo del CPU como en el cómputo de alto desempeño o cómputo científico.

La Caché se divide en diferentes jerarquías de acceso:

- La Caché L1 es el primer sitio donde la CPU buscará información, pero también es la más pequeña y la más rápida, a veces para mayor eficiencia, la Caché L1 se subdivide en L1d (datos) y L1i (instrucciones), actualmente los procesadores modernos en cada core tiene su propio cache de datos e instrucciones.
- La Caché L2 suele ser más grande que la L1 pero es algo más lenta. Sin embargo, por norma general es la que mayor impacto tiene en el rendimiento, este también está incluido en cada core.
- La Caché L3 es mucho más grande que las anteriores, y generalmente se comparte entre todos los núcleos del procesador (a diferencia de las anteriores, que normalmente van ligadas a cada core). Este tercer nivel es en el que buscará el procesador la información tras no encontrarla en la L1 y L2, por lo que su tiempo de acceso es todavía mayor.

Para poner en contexto la relevancia de la memoria Caché, supongamos que el acceso a los datos de la memoria Caché L1 por el procesador es de dos ciclos de reloj, el acceso a la memoria Caché L2 es de 6 ciclos de reloj, el acceso a la memoria Caché L3 es de 12 ciclos y el acceso a la RAM es de 32 ciclos de reloj.

Además supongamos que la operación suma y resta necesitan de 2 ciclos de reloj para completar la operación una vez que cuente con los datos involucrados en dicha operación, que la multiplicación requiere 4 ciclos de reloj

para completar la operación, la división necesita 6 ciclos de reloj para completar la operación y estamos despreciando el tiempo necesario para poner los datos del Caché L1 a los registros del procesador para poder iniciar el cálculo, así también despreciamos el tiempo requerido para sacar el resultado de los registros del procesador al Caché L1.

Esto nos da una idea del número máximo teórico de operaciones básicas que un procesador puede realizar por segundo dependiendo de la velocidad de reloj de la CPU⁶³.

Si nosotros necesitamos hacer la multiplicación de una matriz $\underline{\underline{A}}$ es de tamaño $n \times n$ por un vector $\underline{u}$ de tamaño n y guardar el resultado en el vector $\underline{f}$ de tamaño n . Entonces algunos escenarios son posibles:

1. Si el código del programa cabe en el Caché L1 de instrucciones y la matriz $\underline{\underline{A}}$, los vectores $\underline{u}$ y $\underline{f}$ caben íntegramente en el Caché L1 de datos, entonces el procesador estará siendo utilizado de forma óptima al hacer los cálculos pues no tendrá tiempos muertos por espera de datos.
2. Si el código del programa cabe en el Caché L1 de instrucciones y los vectores $\underline{u}$ y $\underline{f}$ caben íntegramente en el Caché L1 de datos pero la matriz $\underline{\underline{A}}$ está dispersa entre los Cachés L1 y L2, entonces el procesador estará teniendo algunos tiempos muertos mientras carga la parte que necesita de la matriz del Caché L2 a L1 para hacer los cálculos y utilizado de forma óptima el procesador mientras no salga del Caché L1.
3. Si el código del programa cabe en el Caché L1 de instrucciones y los vectores $\underline{u}$ y $\underline{f}$ caben íntegramente en el Caché L1 de datos pero la matriz $\underline{\underline{A}}$ está dispersa entre los Cachés L1, L2 y L3, entonces el procesador estará teniendo muchos tiempos muertos mientras carga la parte que necesita de la matriz del Caché L3 y L2 a L1 para hacer los cálculos resultando en mediana eficiencia en el uso del procesador.
4. Si el código del programa cabe en el Caché L1 de instrucciones y los vectores $\underline{u}$ y $\underline{f}$ caben íntegramente en los Cachés L3, L2 y L1 pero

⁶³Por ejemplo en un procesador AMD Ryzen 9 3900X con 12 Cores (2 Threads por Core) por procesador emulando un total de 24 Cores, corre a una frecuencia base de 3,340 MHz, con una frecuencia mínima de 2,200 MHz y máxima de 4,917 Mhz, con Caché L1d de 384 KiB, L1i de 384 KiB, Caché L2 de 6 MiB y Caché L3 de 64 MiB.

los datos de la matriz $\underline{A}$ está dispersa entre la RAM y los Cachés L3, L2 y L1, entonces el procesador estará teniendo un exceso de tiempos muertos mientras carga la parte que necesita de la matriz de la RAM a los Cachés L3, L2 y L1 para hacer los cálculos resultando en una gran pérdida de eficiencia en el uso del procesador.

Además, debemos recordar que la computadora moderna nunca dedica el cien por ciento del CPU a un solo programa, ya que los equipos son multitarea⁶⁴ y multiusuario⁶⁵ por lo que la conmutación de procesos (que se realiza cada cierta cantidad de milisegundos) degrada aún más la eficiencia computacional de los procesos que demandan un uso intensivo de CPU⁶⁶.

Last Level Cache se le llama Last Level Cache siempre al último nivel de Caché de una CPU, existen dos tipos:

- Last Level Cache Estándar.
- Victim Cache.

⁶⁴Cuentan con la capacidad para ejecutar varios procesos simultáneamente en uno o más procesadores, para ello necesitan hacer uso de la conmutación de tareas, es decir, cada cierto tiempo detiene el programa que está corriendo y guardan sus datos, para poder cargar en memoria otro programa y sus respectivos datos y así reiniciar su ejecución por un período determinado de tiempo, una vez concluido su tiempo de ejecución se reinicia la conmutación de tareas con otro proceso.

⁶⁵Se refiere a todos aquellos sistemas operativos que permiten el empleo de sus procesamiento y servicios al mismo tiempo. Así, el sistema operativo cuenta con la capacidad de satisfacer las necesidades de varios usuarios al mismo tiempo, siendo capaz de gestionar y compartir sus recursos en función del número de usuarios que estén conectados a la vez.

⁶⁶Actualmente existen una gran cantidad de distribuciones de GNU/Linux que vienen muy optimizadas intentando conseguir la mejor desenvolvura de su arquitectura y configuraciones de serie. En el caso de la configuración por omisión de Debian GNU/Linux y Ubuntu, están pensadas para que sean lo más robusta posible y que se use en todas las circunstancias imaginables, por ello están optimizadas de forma muy conservadora para tener un equilibrio entre eficiencia y consumo de energía. Pero es posible agregar uno o más Kernels GNU/Linux generados por terceros que contenga las optimizaciones necesarias para hacer más eficiente y competitivo en cuestiones de gestión y ahorro de recursos del sistema.

Hay varias opciones del Kernel GNU/Linux optimizado (**Liquorix** viene optimizado para multimedia y Juegos, por otro lado **XanMod** tiene uno para propósito general, otro aplicaciones críticas en tiempo real y otro más para cálculos intensivos) de las últimas versiones estable del Kernel.

Una Victim Cache no actúa como la Caché de último nivel de una CPU, sino que en ese caso lo hace el penúltimo nivel y en la Victim Cache acaban los últimos datos descartados de la Caché y que han sido volcados en la RAM, los cuales son copiados en la Victim Cache para poder acceder a ellos más rápido.

Smart Cache la Smart Cache (o Caché) es esencialmente L3 pero optimizada por Intel para ser más eficiente a la hora de compartir la información en los núcleos de la CPU. A efectos prácticos, se comporta de igual manera que la Caché L3.

Disco Son dispositivos no volátiles (los hay del orden de hasta 32 TB y continuamente incrementan su capacidad⁶⁷), lo que significa que retienen datos incluso cuando no tienen energía. La información almacenada permanece segura e intacta a menos que el disco duro sea destruido o interferido. La información se almacena o se recupera de manera aleatoria en lugar de acceso secuencial. Esto implica que se puede acceder a los bloques de datos en cualquier momento sin necesidad de pasar por otros bloques de datos.

⁶⁷Durante años la tecnología más popular entre los fabricantes ha sido la PMR (Perpendicular Magnetic Recording), también conocida como CMR (Conventional Magnetic Recording). A esta tecnología luego se le sumó la variante SMR (Shingled Magnetic Recording), que lograba aumentar la densidad de grabación, pero lo hacía sacrificando velocidad de transferencia y fiabilidad de las operaciones.

Western Digital ha creado la tecnología ePMR (energy-assisted Perpendicular Magnetic Recording) que permite ofrecer mayores densidades de grabación y, según este fabricante, mejorar la fiabilidad de las escrituras y evitar así los sacrificios que había que hacer con SMR (en los últimos tiempos han aparecido unidades de 20 a 32 TB basadas en dicha tecnología).

Más interesante aún es el sistema de grabación MAMR (Microwave-Assisted Magnetic Recording) que hace uso de microondas para calentar el medio de almacenamiento y así lograr mejorar densidad de grabación y fiabilidad de lecturas y escrituras. Hace años ya prometían que gracias a esta tecnología contaríamos con unidades de 40 TB en 2025, pero parece que dicho logro aún tardará en llegar.

Esta otra opción es una alternativa a las microondas, pero en HAMR (Heat-Assisted Magnetic Recording) el proceso de calentar el medio de almacenamiento lo realiza un láser. Toshiba prometió lanzar unidades de más de 32 TB en 2024, mientras que Seagate también quería ofrecer esa capacidad de forma inminente para luego dar el salto a unidades de 40 TB e incluso a los 100 TB que plantean para 2030. Ahí es donde probablemente entre en acción la evolución de HAMR+, que tratará de exprimir aún más la densidad de grabación.

Actualmente, podemos agrupar los discos duros disponibles en cuatro tipos:

- Parallel Advanced Technology Attachment (PATA)
- Serial ATA (SATA)
- Interfaz de sistema de computadora pequeña (SCSI)
- Adjunto de tecnología avanzada paralela
- Unidades de estado sólido (SSD)

En promedio, las velocidades máximas de los discos actuales son:

- Disco SATA3 de 5,400 RPM, Lectura: 102 MB/s, Escritura: 96 MB/s
- Disco SATA3 de 7,200 RPM, Lectura: 272 MB/S, Escritura: 200 MB/s
- Disco SSD SATA, Lectura 550 MB/s, Escritura 520 MB/s
- Disco SSD NVMe, Lectura 6,600 MB/s, Escritura 5,500 MB/s
- Disco SSD PCI 5.0, Lectura 13,000 MB/s, Escritura 12,000 MB/s
- Unidad Flash USB⁶⁸ 2.0, 35 MB/s
- Unidad Flash USB 3.0 o 3.1 gen 1, 5 Gbit/s
- Unidad Flash USB 3.0 o 3.1 gen 2, 10 Gbit/s
- Unidad Flash USB 3.2 gen 2x2, 20 Gbit/s

⁶⁸Los colores en los puertos USB son: USB 1.X blanco (12 Mbps), USB 2.X Negro (480 Mbps), USB 3.0 Azul oscuro (5 Gbps), USB 3.1 Azul claro (10 Gbps), USB 3.2 Rojo (20 Gbps). En el caso del USB de color Amarillo, éste es un puerto de carga aún con el dispositivo apagado.

Disco de Estado Sólido SSD Estos son los últimos avances en tecnología de almacenamiento que tenemos en la industria de las computadoras. Son totalmente diferentes de las otras unidades en que no consisten en partes móviles. Tampoco almacenan datos utilizando magnetismo. En su lugar, hacen uso de la tecnología de memoria flash, circuitos integrados o dispositivos semiconductores para almacenar datos de forma permanente, al menos hasta que se borren. Estas son algunas de sus ventajas.

- Acceso a datos más rápido
- Menos susceptible a los golpes
- Menores tiempos de acceso y latencia
- Menos consumo de energía

Los SSD actuales están disponibles tanto en versiones SATA como en versiones M.2, U.2 y en formato de tarjeta PCI Express 4.0. Los tres últimos hacen uso del protocolo NVMe y la interfaz PCI Express 4.0 x4, lo que les permite superar los 6,600 MB/s de velocidades de lectura y escritura, frente a los 550 MB/s que suelen alcanzar como máximo las unidades SATA. La nueva versión PCI 5.0 ofrece un ancho de banda de 32 GT/s el doble de PCI 4.0, permitiendo discos SSD con 13,000 MS/s de velocidad de lectura secuencial y realizar hasta 2,500K operaciones por segundo de lectura aleatoria y tamaño máximo 15.36 TB a un precio exorbitante.

Tarjetas microSD Cuando compramos una tarjeta microSD para ampliar el almacenamiento de nuestro Smartphone, cámara, tableta o cualquier otro dispositivo electrónico la mayoría de la gente normalmente solo se fija en la capacidad de almacenamiento. Ahora bien, ¿qué significan todas esas etiquetas y nombres que llevan adscritas las microSD? ¿Cuál es la diferencia entre una microSDXC y una microSDHC? ¿Es mejor una UHS-I o una UHS-II? ¿Qué quiere decir que una tarjeta es A1 y V30? A continuación, intentamos aclarar toda esta nomenclatura.

Lo primero que tenemos que tener claro es que cuando analizamos una tarjeta micro SD existen multitud de factores que limitan su velocidad y capacidad de almacenamiento. Su rendimiento depende de factores como el tipo de tarjeta que estamos usando, su clase, y otros detalles como el tipo de bus o el número de operaciones que puede realizar por segundo.

Tipos de tarjetas microSD actualmente existen 4 generaciones distintas de tarjetas de memoria microSD. Cuanto más modernas sean, mayores velocidades y almacenamiento podrán ofrecer:

- Tarjetas micro SD (Secure Digital): Estas son las memorias de primera generación. Las desarrolló el fabricante SanDisk y fueron las primeras en utilizar el formato de 15 x 11 x 1 milímetros. Su capacidad máxima es de 32 GB.
- Tarjetas micro SDHC (Secure Digital High Capacity): Tarjetas de segunda generación. Cuentan con un bus de datos mejorado que permite alcanzar velocidades superiores, aunque su capacidad máxima sigue siendo de 32 GB.
- Tarjetas micro SDXC (Secure Digital Extended Capacity): Estas micro SD utilizan un sistema de archivos exFAT y su velocidad de transferencia puede llegar hasta los 312 MB/s. Su capacidad de almacenamiento puede llegar hasta los 2 TB y es el tipo de tarjeta más común utilizado a día de hoy.
- Tarjetas micro SDUC: Estas son las tarjetas de memoria más modernas y punteras. Utilizan el sistema de archivos exFAT y permiten almacenar entre 2 TB y 128 TB de datos.

Evidentemente con esto no es suficiente. Si queremos tener una idea aproximada de la velocidad de la micro SD tendremos que fijarnos en aspectos como la clase y tipo de bus que emplea.

La clase es una característica que nos indica la velocidad de transferencia de datos mínima de la tarjeta de memoria. Actualmente hay 4 tipos de clase diferentes:

- Clase 2: Velocidad mínima de 2 MB/s
- Clase 4: Velocidad mínima de 4 MB/s
- Clase 6: Velocidad mínima de 6 MB/s
- Clase 10: Velocidad mínima de 10MB/s

Hoy en día la mayoría de tarjetas micro SD son de clase 10, ya que son capaces de transferir más de 10 MB/s y superan esa cifra fácilmente.

El bus determina la velocidad de la interfaz de la tarjeta de memoria, y nos puede servir como indicativo para conocer la rapidez con la que se pueden leer y escribir los datos:

- Bus estándar: Su velocidad de transferencia alcanza hasta los 12.5 MB/s. Es el tipo de bus utilizado en tarjetas de clase 2, 4 y 6.
- Bus de alta velocidad (High Speed): Se utiliza en tarjetas de clase 10 y alcanza una velocidad de hasta 25 MB/s.
- Bus Ultra High Speed (UHS): Estos son los buses con la interfaz más rápida, y existen varios tipos:
 - UHS-I: Hay dos tipos de buses UHS-I. Por un lado, tenemos el UHS-I clase 1 (U1) que alcanza velocidades de 50 MB/s. Y luego tenemos el UHS-I clase 3 (U3) que llega hasta los 104 MB/s.
 - UHS-II: Alcanza velocidades de transferencia hasta 312 MB/s.
 - UHS-III: Velocidades de transferencia de datos que alcanzan hasta los 624 MB/s.
- SD-Express: Este es el tipo de bus más potente de todos, llegando hasta los 985 MB/s.

Como referencia, te interesará saber que actualmente la mayoría de tarjetas micro SD de gama media utilizan un bus UHS-I de clase 3 (U3) con velocidades de lectura de hasta 104 MB/s.

Otro factor importante es la velocidad de lectura y escritura aleatoria (IOPS) u operaciones por segundo que puede realizar una tarjeta. Este dato determina el rendimiento mínimo en la lectura y escritura aleatoria de la SD:

- Clase de rendimiento de aplicación A1: Las tarjetas A1 tienen una velocidad mínima de lectura aleatoria de 1,500 IOPS, y una velocidad mínima de escritura aleatoria de 500 IOPS.
- Clase de rendimiento de aplicación A2: Las tarjetas A2 ofrecen velocidades superiores, con 4,000 IOPS de lectura y 2,000 IOPS de escritura.

Normalmente con una tarjeta A1 es más que suficiente para tareas del día a día, aunque si necesitamos un rendimiento superior, por ejemplo, para ejecutar aplicaciones desde la SD o jugar a videojuegos, las tarjetas A2 ofrecen un mejor rendimiento.

La velocidad de escritura aleatoria (A1 y A2) es un dato relevante para los Smartphones y tabletas, pero si tenemos una cámara de grabación, una Action camera o un Dron, la característica en la que nos tenemos que fijar es en la Velocidad de escritura secuencial (sistema V) que utiliza (en inglés, Video Speed Class). O dicho de otra forma, en su velocidad de escritura secuencial.

Esta característica nos indica la cantidad de datos que se pueden grabar en la micro SD de forma constante sin bajar de una velocidad mínima. Esto resulta esencial cuando queremos grabar vídeos en alta y ultra-alta definición:

- V30 (Video Speed Class 30): Velocidad de escritura mínima de 30 MB/s
- V60 (Video Speed Class 60): Velocidad de escritura mínima de 60 MB/s
- V90 (Video Speed Class 90): Velocidad de escritura mínima de 90 MB/s

Por ejemplo, si vamos a grabar vídeo en resolución 4K directamente en la tarjeta micro SD, es necesario que la velocidad V sea lo máximo posible, especialmente si vamos a utilizar un amplio BitRate con bajos niveles de compresión, o calidades superiores como el 8K.

Cintas Magnéticas En el año 2010, se comunicó que todos los datos utilizados para el proyecto del satélite Nimbus se recuperaron de cintas que en ese momento tenían 46 años. A partir de dicho comunicado se extendió el uso de la cinta magnética para almacenamiento de datos en todo el mundo.

En un mundo altamente digital, la cinta magnética es una de las pocas tecnologías que utiliza señales analógicas para mover parte de los datos, en su esencia, la cinta se parece mucho a un HDD, utiliza materiales de base magnética, pero en este caso, la cinta es literalmente una base de material generalmente nailon que tiene un revestimiento magnético. Y en lugar de un disco giratorio, la cinta entra, está enhebrada. Puede parecer una cinta VHS, pero es mucho más robusta.

La cinta se mueve linealmente hacia la unidad de cinta y hacia el cartucho de cinta. Para escribir, el cabezal de la cinta toma señales electrónicas y crea un mini campo magnético que puede cambiar la polaridad del material de la película para formar un patrón de ceros y unos. Una vez que los datos se escriben en la cinta, no se pueden cambiar (pero se pueden borrar y reescribir).

La inmutabilidad y las capacidades de encriptación de la cinta, así como la simplicidad de crear un espacio para almacenarla en una bóveda hacen de la cinta un arma clave para asegurar que los datos sobrevivan frente al Ransomware. Uno de los grandes productores de cintas en la actualidad es IBM, los cuales argumentan que esta función hace que la cinta sea el medio ideal para almacenar datos de archivo a los que no es necesario acceder con frecuencia.

La cinta también puede servir como una copia de seguridad y de versiones fuera de línea de archivos importantes o confidenciales que son resistentes a los ataques cibernéticos. Los tipos de datos que permanecen en la cinta abarcan registros financieros, registros médicos, información de identificación personal y documentos que forman parte de una retención legal de múltiples gobiernos.

Una sola cinta mide aproximadamente 3 pulgadas por 3 pulgadas y 3/4 de pulgada de grosor. Es más pequeño que una unidad de disco duro (HDD), pero pesa alrededor de 0.6 kilogramos. Un cartucho puede almacenar 18 Terabytes de datos sin comprimir y 45 Terabytes comprimidos. IBM está trabajando para duplicar esta capacidad en la próxima generación de la tecnología. En cuanto a la velocidad de recuperación, se obtiene un flujo de datos de una unidad de cinta de 1,000 Megabits por segundo, comprimidos.

Una biblioteca de cintas puede variar en tamaño desde algo que puede poner en su escritorio hasta algo que es del tamaño de un refrigerador pequeño (alrededor de 8 pies cuadrados). La pequeña biblioteca del tamaño de un refrigerador tiene capacidad para 1584 cartuchos. IBM promociona que su biblioteca Diamondback será la biblioteca de cintas más densa del mercado. Podrá contener 69 Petabytes de información mientras ocupa menos de 8 pies cuadrados de espacio.

La cinta magnética supera al disco duro y al flash en cuanto a longevidad, costo financiero y costo de huella de carbono, pero pierde en velocidad de acceso. Las cintas no son recomendables para poner datos de producción en vivo o incluso copias de seguridad, pero son perfectas para cualquier información a la que se acceda con poca frecuencia y que deba conservarse durante

mucho tiempo, como registros médicos o datos de archivo.

Es conocido que muchas empresas han usado y seguirán usando cinta magnética en sus operaciones, entre las que destacan las hiperescalas (empresas que han crecido tanto que ofrecen sus propias infraestructuras o tienen datos masivos como resultado de su infraestructura) siempre necesitan muchas formas diferentes de tecnología para manejar la variedad de datos que ingresan a sus sistemas para alimentar una gama de servicios. Entre otras destacan: Bancos, Gobiernos, Milicia y organizaciones como CERN, así como corporaciones como Amazon, Google, Meta, Baidu, Alibaba y Tencent.

Tarjeta Gráfica La tarjeta gráfica o tarjeta de vídeo es un componente que viene integrado en la placa base de la computadora o se instala aparte para ampliar sus capacidades. Concretamente, esta tarjeta está dedicada al procesamiento de datos relacionados con el vídeo y las imágenes que se están reproduciendo en la computadora.

Procesador Gráfico GPU el corazón de la tarjeta gráfica es la GPU o Unidad de procesamiento gráfico, un circuito muy complejo que integra varios miles de millones de transistores diminutos y puede tener desde uno a miles de núcleos (ya es común encontrar computadoras personales con tarjeta de 10496 cores y 24 GB de GRAM) que tienen capacidad de procesamiento independiente. De la cantidad y capacidad de estos núcleos dependerá la potencia.

Así como los procesadores centrales de las CPU, están diseñados con pocos núcleos pero altas frecuencias de reloj, las GPU tienden al concepto opuesto, contando con grandes cantidades de núcleos con frecuencias de reloj relativamente bajas. Luego tienes la memoria gráfica de acceso aleatorio o GRAM, que son Chips de memoria que almacenan y transportan información entre sí. Esta memoria no es algo que vaya a determinar de forma importante el rendimiento máximo de una tarjeta gráfica, aunque si no es suficiente puede acabar lastrando y limitando la potencia de la CPU.

La idea de usar esa potencia para otros menesteres se denomina GPGPU (General Purpose Computation on Graphics Processing Units) o GPU Computing. En el momento en el que las tarjetas gráficas permiten que se programen funciones sobre su Hardware se empieza a hacer uso de GPGPU. Al principio era necesario utilizar los lenguajes enfocados a la visualización en pantalla (como OpenGL) para realizar otros cálculos no relacionados con los

gráficos. Esto implicaba el uso de funciones muy poco flexibles, originalmente diseñadas para otros fines, lo que hacía que la labor de programar para tal fin fuese realmente tediosa y complicada.

Para facilitar el empleo de las tarjetas gráficas para cualquier uso no vinculado con los gráficos, NVidia desarrolló toda una tecnología alrededor de la tarjeta, que permitía usar la misma para cualquier tarea: CUDA. ATI, la principal (y actualmente casi única) competidora, un poco más tarde haría lo propio lanzando su propia tecnología: Stream. En un principio las tarjetas gráficas solo trabajaban con aritmética de 32 bits, pero en la actualidad ya se cuenta con aritmética de 64 bit (para lograr esto, muchas tarjetas usan dos de sus cores de 32 bits para emular uno de 64 bits, reduciendo su número de cores útiles a la mitad).

Procesador Gráfico Integrado muchos procesadores CPU incorporan una o más GPU en su interior, llamada gráfica integrada (iGPU Integrated Graphics Processing Units o APU Accelerated Processing Unit). Generalmente es muy poco potente, pero lo suficiente para realizar tareas básicas como navegar por Internet, ver vídeos, e incluso para algunos juegos básicos, especialmente en las últimas generaciones puesto que cada vez son más potentes. No obstante, en las últimas generaciones de procesadores cada vez se están introduciendo gráficos integrados más potentes, y ya son capaces de manejar varios monitores, resoluciones 4K e incluso son capaces de mover algunos juegos a una tasa digna de FPS.

Tarjeta Gráfica por ejemplo, la tarjeta gráfica de AMD Instinct MI200 cuenta con más de 200,000 cores y 128 GB de HBM2, NVidia GEFORCE RTX 3090 proporciona 10,496 cores y 24 GB de GDDR6x, NVidia A100 cuenta con 80 GB de memoria HBM2 6192 cores y 432 núcleos tensor⁶⁹, mientras que la tarjeta NVidia Titan RTX proporciona 130 Tensor TFLOP de rendimiento, 576 núcleos tensores y 24 GB de memoria GDDR6.

Por otra parte, Intel ha desarrollado una aceleradora gráfica Arc Sound-M pensada para centro de datos (especialmente diseñada para juegos en la nube) que utiliza una GPU DG2 Xe-HPG que viene con una configuración de 512 unidades de ejecución lo que equivale a 4,096 Shaders, por ejemplo, esta

⁶⁹Un Tensor core (o núcleos Tensor) calculan la operación de una matriz 4x4 completa, la cual se calcula por reloj. Estos núcleos pueden multiplicar dos matrices FP16 4x4 y sumar la matriz FP32 al acumulador.

aceleradora puede manejar hasta 8 Streamings simultáneos de video 4K o más de 30 si el vídeo es en 1080p y cuenta con más de 60 funciones virtualizadas.

En agosto del 2021, se anunció la construcción de la supercomputadora Polaris, acelerado por 2240 GPU NVIDIA A100 Tensor Core, el sistema puede alcanzar casi 1.4 exaflops de rendimiento teórico de IA y aproximadamente 44 petaflops de rendimiento máximo de doble precisión. Polaris, que será construido por Hewlett Packard Enterprise, combinará simulación y aprendizaje automático al abordar cargas de trabajo informáticas de alto rendimiento de inteligencia artificial y con uso intensivo de datos, impulsadas por 560 nodos en total, cada uno con cuatro GPU NVIDIA A100.

Tipos de Redes Según el Medio Físico Si bien, nuestros dispositivos de cómputo pueden funcionar sin conexión de red, estos se ven inmediatamente limitados. La red nos permite conectarnos a Internet que es el camino por el cual nos conectamos con el mundo. Las formas de conectar nuestros equipos a Internet en un principio fue exclusivamente por red alámbrica, desde ya hace unos años a la fecha se dispone de conexión a red alámbrica e inalámbrica, pero actualmente nuestros dispositivos cuentan casi exclusivamente con conexión inalámbrica.

¿Qué es el ancho de banda? Se trata de la capacidad máxima y la cantidad de datos que se pueden transmitir a través de una conexión (de internet, por ejemplo), en un momento determinado. Algo que debemos tener claro es que el ancho de banda de red es fundamental para la calidad y velocidad de la conexión.

El ancho de banda se mide en *bit/s* o en sus múltiplos *k/bits* o *m/bits* por segundo. Y para la mayoría de los casos, debemos asegurarnos siempre de tener el mayor ancho de banda que nos sea posible, porque de esta manera podremos tener una mejor y más rápida transferencia de datos.

¿Qué es entonces la velocidad de transmisión? Este término se puede definir como la velocidad a la que se transmite la información. Cuando un usuario adquiere un paquete con una empresa prestadora de servicios de internet, recibe, por ejemplo, 10 mbps, 30 mbps, 100 mbps, etc. Y esto se refiere a la cantidad de datos que podemos descargar o subir a la red. Como recomendación, lo indicado es que para que la velocidad pueda existir será

necesario tener un ancho de banda igual o superior a la velocidad contratada en el paquete de servicio.

Normalmente vemos en los anuncios de todos los operadores, que estos ofrecen una cantidad cualquiera de megas de navegación; este valor numérico corresponde a la velocidad de descarga únicamente. Para encontrar la velocidad de subida, es necesario acceder a un test que nos revele cuál es el resultado y si lo que nos prometen, es verdad o no.

¿Qué es la latencia? Es el tiempo total que transcurre desde que enviamos una información, hasta que la misma llega a un receptor. Su valor de medición se hace en milisegundos, también se conoce como *Ping* y está presente en actividades que realiza cotidianamente como jugar en línea o hacer videollamadas.

¿Altera la latencia la velocidad de la conexión? La velocidad de conexión influye en la latencia una vez que pasamos de un rango predeterminado. Poniéndolo en un ejemplo, si tenemos una conexión a Internet de 1 Mbps y la comparamos con una conexión de 100 Mbps, dependiendo del tamaño del paquete se notará una mejora grande en la velocidad. Otros factores que importan a la hora de hablar de latencia son el estar conectado a internet por Wi-fi o un cable, si tiene servicio de fibra óptica o qué tanta distancia hay entre su ordenador y un Router, etc.

La latencia en la conexión también es la suma de otros retardos:

- De procesamiento: se define en el tiempo que tardan los Routers en examinar la cabecera y a su vez, la respuesta en determinar a dónde hay que enviar cualquier paquete haciendo una previa comprobación de sus tablas de enrutamiento.
- De cola: tiempo de espera del paquete para poder ser transmitido a través de un enlace físico. Cabe anotar que no podemos saber previamente si va a haber un retardo de cola o no, ya que este cambia en tiempo real.
- De transmisión: es el tiempo que tarda el paquete en arribar hasta el siguiente nodo o destino final.

- De propagación: es el tiempo que tarda un bit en propagarse desde un punto cualquiera de origen hasta llegar a uno de destino. Su velocidad depende del medio físico por el que se transporte.

El retardo total es la suma de todos los retardos anteriormente enunciados.

Comúnmente se asocia la latencia con la banda de ancha, pero existe una diferencia sustancial entre ambas. Si bien ambas afectan la velocidad de la conexión, la banda ancha permite que se pueda transmitir una gran cantidad de datos, mientras la latencia determina a qué velocidad se transmite esa cantidad de datos.

Si bien, tener red nos permite estar conectados, tenemos una gran limitación por las velocidades de conexión a las que tendremos acceso según el tipo de medio físico que usemos para conectarnos, así como el número de dispositivos con los que compartamos la conexión. Las redes inalámbricas parecen ser omnipresentes, pero las velocidades de interconexión dejan mucho que desear como veremos a continuación.

Redes alámbricas se comunica a través de cables de datos (generalmente basada en Ethernet). Los cables de datos, conocidos como cables de red de Ethernet o cables con hilos conductores (CAT5), conectan computadoras y otros dispositivos que forman las redes. Las redes alámbricas son mejores cuando se necesita mover grandes cantidades de datos a altas velocidades, como medios multimedia de calidad profesional.

Ventajas

- Costos relativamente bajos
- Ofrece el máximo rendimiento posible
- Mayor velocidad - cable de Ethernet estándar hasta 1 Gbps⁷⁰
- Mayor rendimiento de Voz sobre IP.
- Mejores estándares Ethernet en la industria.
- Mayor capacidad de ancho de banda por cables.

⁷⁰El término bps se refiere a transmitir Bits por segundo (se requieren 8 Bits para formar un Byte). Por eso el término de 1,000 Mbps es equivalente a 125 MB/s.

- Aplicaciones que utilizan un ancho de banda continuo.

Desventajas

- El costo de instalación siempre ha sido un problema común en este tipo de tecnología, ya que el estudio de instalación, canaletas, conectores, cables y otros suman costos muy elevados en algunas ocasiones.
- El acceso físico es uno de los problemas más comunes dentro de las redes alámbricas. Ya que para llegar a ciertos lugares, es muy complicado el paso de los cables a través de las paredes de concreto u otros obstáculos.
- Dificultad y expectativas de expansión es otro de los problemas más comunes, ya que cuando pensamos tener un número definido de nodos en una oficina, la mayoría del tiempo hay necesidades de construir uno nuevo y ya no tenemos espacio en los Switches instalados.

Hoy en día se puede hacer la siguiente clasificación de las redes de protocolo Ethernet para cable y fibra óptica⁷¹:

- Ethernet, que alcanza no más de 10 Mbps de velocidad
- Fast Ethernet, que puede trabajar con hasta 100 Mbps
- Gigabit Ethernet, alcanza hasta 1 Gbps (1000 Mbps aprox.)
- 2.5 y 5 Gigabit Ethernet hasta 2.5 Gbps si usamos cableado Cat 5a y nada menos que 5 Gbps con cableado Cat 6

⁷¹El récord mundial en mayo del 2022 de transmisión en fibra óptica es de 1.02 petabits por segundo enviados a través de 51.7 kilómetros (que podría transmitir hasta 10 millones de canales por segundo de vídeo a resolución 8K), que rompe al récord anterior de 319 terabits por segundo sobre una distancia de 1,800 millas (con el estimado de descarga de 80,000 películas simultáneamente en un segundo).

En 2023 los investigadores de Electronics and Computer Engineering de Aston University en U.K. alcanzaron una velocidad de 301 terabits por segundo (Tbps), equivalente a transferir 1,800 películas 4K a través de Internet en un segundo, utilizando cables de fibra óptica existentes. En comparación, la velocidad media de banda ancha fija en los EE. UU. es de 242,38 megabits por segundo (Mbps), según Speed Test.

Los resultados de la prueba, que se realizaron utilizando el tipo de cables de fibra ya tendidos en el suelo, lograron esta velocidad vertiginosa enviando luz infrarroja a través de hilos tubulares de vidrio, que es como funciona generalmente la banda ancha de fibra óptica. Pero aprovecharon una banda espectral que nunca se ha utilizado en sistemas comerciales, llamada "banda E", utilizando dispositivos nuevos hechos a medida.

- 10 Gigabit Ethernet, que puede alcanzar hasta los 10 GMbps
- 100GbE para alcanzar la Ethernet Terabit (125 Gbytes)

La categoría del cable o el tipo de fibra óptica determina la velocidad máxima soportada por cada tipo de cable, pero aunque haya cables con la misma velocidad hay otros factores que determinan su usabilidad, como el ancho de banda o la frecuencia. La frecuencia o ancho de banda determina la potencia de la red, a mayor frecuencia mayor ancho de banda y menor pérdida de datos. Este factor es importante si vamos a conectar varios equipos al mismo cable de red o vamos a hacer una gran tirada de cable, ya que cuanto más largo sea el cable de red más potencia perderá. Siempre tendrá más velocidad un cable corto que un cable largo, pero si el ancho de banda es amplio tardará más metros en perder potencia y velocidad.

Ethernet es el estándar que domina la gran mayoría de mercado, presente de manera casi exclusiva tanto en mercado doméstico, como en pequeña y mediana empresa representa también un porcentaje muy significativo en grandes centros de datos. Pero existen otros protocolos que se pueden situar en el mismo nivel de calidad que Gigabit Ethernet como InfiniBand.

InfiniBand es un bus serie bidireccional de comunicaciones de alta velocidad en las que las rápidas comunicaciones entre servidores son críticas para el rendimiento, llegando a ofrecer velocidades de hasta 2.0 Gbps netos en cada dirección del enlace en un nodo simple, 4 Gbps netos en un nodo doble y hasta 8 Gbps netos en un nodo quadruple. Estos nodos a su vez se pueden agrupar en grupos de 4 ó 12 enlaces llegando a velocidades de hasta 96 Gbps netos en un grupo de 12 nodos cuádruples. El factor de velocidad neta viene relacionado con que Infiniband de cada 10 bits que transmite 8 de ellos son datos, basándose en la codificación 8B/10B.

Recientemente se han implementado sistemas en los que ya no se utiliza esta codificación 8B/10B sino la 64B/66B que permite mejorar el porcentaje de datos útiles por trama enviada y que ha permitido los nodos FDR-10 (Fourteen Data Rate-10 a 10 Gbps), FDR (Fourteen Data Rate a 13.64 Gbps) y EDR (Enhanced Data Rate a 25 Gbps). Este último en un grupo de 12 nodos proporciona hasta 300 Gbps. Los últimos desarrollos de Gigabit Ethernet, proporcionan hasta 100 Gbps por puerto.

Estas enormes velocidades de conexión hacen que Infiniband sea una conexión con una muy importante presencia en superordenadores y clústers, por ejemplo del top 500 de superordenadores en 2020, 226 están conectados

internamente con Infiniband, 188 lo están con Gigabit Ethernet y el resto con Myrinet, Cray, Fat Tree u otras interconexiones a medida.

Una de las principales ventajas de Infiniband sobre Ethernet es su bajísima latencia, por ejemplo y basándonos en los datos del estudio de Qlogic "Introduction to Ethernet Latency, an explanation to Latency and Latency measurement", la latencia en 10 Gbps Ethernet se sitúa en 5 microsegundos mientras que la de Infiniband se sitúa por debajo de los 3 microsegundos.

Los sistemas de conmutadores inteligentes InfiniBand de NVIDIA Mellanox ofrecen el mayor rendimiento y densidad de puertos para computación de alto rendimiento (HPC), IA, Web 2.0, Big Data, nubes y centros de datos empresariales. La compatibilidad con configuraciones de 36 a 800 puertos a hasta 200 Gbps por puerto permite que los clústeres de cómputo y los centros de datos convergentes funcionen a cualquier escala, lo que reduce los costos operativos y la complejidad de la infraestructura.

Redes inalámbricas: es una red en la que dos o más terminales (ordenadores, tabletas, teléfonos inteligentes, etc.) se pueden comunicar sin la necesidad de una conexión por cable. Se basan en un enlace que utiliza ondas electromagnéticas (radio e infrarrojo) en lugar de cableado estándar. Permiten que los dispositivos remotos se conecten sin dificultad, ya se encuentren a unos metros de distancia como a varios kilómetros.

Asimismo, la instalación de estas redes no requiere de ningún cambio significativo en la infraestructura existente como pasa con las redes cableadas. Tampoco hay necesidad de agujerear las paredes para pasar cables ni de instalar porta cables o conectores. Esto ha hecho que el uso de esta tecnología se extienda con rapidez.

Tipos de redes inalámbricas

- LAN Inalámbrica: Red de área local inalámbrica. También puede ser una red de área metropolitana inalámbrica.
- GSM (Global System for Mobile Communications): la red GSM es utilizada mayormente por teléfonos celulares.
- D-AMPS (Digital Advanced Mobile Phone Service): está siendo reemplazada por el sistema GSM.
- Fixed Wireless Data: Es un tipo de red inalámbrica de datos que puede ser usada para conectar dos o más edificios juntos para extender o

compartir el ancho de banda de una red sin que exista cableado físico entre los edificios.

- Wi-Fi⁷²: es uno de los sistemas más utilizados para la creación de redes inalámbricas en computadoras, permitiendo acceso a recursos remotos como internet e impresoras. Utiliza ondas de radio.

Ventajas

- La instalación de redes inalámbricas suele ser más económica.
- Su instalación también es más sencilla.
- Permiten gran alcance; las redes hogareñas inalámbricas suelen tener hasta 100 metros desde la base transmisora.
- Permite la conexión de gran cantidad de dispositivos móviles. En las redes cableadas mientras más dispositivos haya, más complicado será el entramado de cables.

⁷²Notemos que una conexión de WiFi tradicional sin obstáculos con una intensidad de banda de 2.4 GHZ puede tener un alcance máximo de 46 metros y para la banda de 5.0 GHZ un alcance de 15 metros. Pero hay muchos objetos que interfieren con la señal del Router del WiFi y el sitio en el que colocamos nuestro dispositivo inalámbrico como computadora o teléfono inteligente:

- Superficies y/o objetos de metal o vidrio blindado
- Refrigeradores, lavadoras y radiadores
- Hornos de microondas, cámaras Web, monitores de bebés y teléfonos inalámbricos
- Paredes y muros
- Dispositivos arquitectónicos que funcionan como una jaula de Faraday (es un contenedor recubierto por materiales conductores de electricidad como mallas metálicas, papel aluminio, cajas o cestos de basura de acero que funcionan como un blindaje contra los efectos de un campo eléctrico proveniente del exterior).

En caso de que varios dispositivos se conecten a la red, se puede optar por colocar el Router en un punto medio, para que ninguna zona quede sin cobertura. Por otra parte, para una mejor conexión, también procura que el Router se encuentre en una zona elevada, pues esto mejorará el alcance de la señal inalámbrica. Una excelente opción para mejorar la calidad del internet inalámbrico cuando hay obstáculos es utilizar un repetidor de WiFi, este dispositivo es muy útil para amplificar la señal y llevarla a más sitios de nuestra red.

- Posibilidad de conectar nodos a grandes distancias sin cableado, en el caso de las redes inalámbricas corporativas.
- Permiten más libertad en el movimiento de los nodos conectados, algo que puede convertirse en un verdadero problema en las redes cableadas.
- Permite crear una red en áreas complicadas donde, por ejemplo, resulta dificultoso o muy caro conectar cables.

Desventajas

- Calidad de Servicio: La velocidad que posee la red inalámbrica no supera la cableada, ya que esta puede llegar a los 10 Mbps, frente a 100 Mbps que puede alcanzar la cableada. Hay que tomar en cuenta la tasa de error debida a las interferencias.
- Costo: En algunos casos, puede ser más barato cablear una casa/oficina que colocar un servicio de red inalámbrica.
- La señal inalámbrica puede verse afectada e incluso interrumpida por objetos, árboles, paredes, espejos, entre otros.

La velocidad máxima de transmisión inalámbrica de la tecnología 802.11b es de 11 Mbps. Pero la velocidad típica es solo la mitad: entre 1.5 y 5 Mbps dependiendo de si se transmiten muchos archivos pequeños o unos pocos archivos grandes. La velocidad máxima de la tecnología 802.11g es de 54 Mbps. Pero la velocidad típica de esta última tecnología es solo unas 3 veces más rápida que la de 802.11b: entre 5 y 15 Mbps. Resumiendo, las velocidades típicas de los diferentes tipos de red son:

Estándar	V.Máxima	V.Practica	Frecuencia	Ancho Banda	Alcance
802.11	2Mbit/s	1Mbit/s	2.4Ghz	22MHz	330 metros
802.11a(WiFi5)	54Mbit/s	22Mbit/s	5.4Ghz	20MHz	390 metros
802.11b	11Mbit/s	6Mbit/s	2.4Ghz	22MHz	460 metros
802.11g	54Mbit/s	22Mbit/s	2.4Ghz	20MHz	460 metros
802.11n	600Mbit/s	100Mbit/s	2.4Ghz y 5.4Ghz	20 y 40MHz	820 metros
802.11ac	6.93Gbps	100Mbit/s	5.4Ghz	80 o hasta 160MHz	Poco alcance, pero sin interferencias
802.11ad	7.13Gbit/s	Hasta 6Gbit/s	60Ghz	2MHz	300 metros
802.11ah	35.6Mbps	26.7Mbps	0.9Ghz	2MHz	1000 metros
802.11ax(WiFi6)	9.6Gbps	6.9Gbps	2.4Ghz y 5.4Ghz	20MHz	1000 metros

Como puedes ver, los principales factores que influyen en la calidad de una conexión WiFi, son la frecuencia, el ancho de banda y el alcance total. Considerando que, todo esto junto con la velocidad máxima y la velocidad práctica, se congregan en lo que es cada versión de este tipo de conexión a la red. Además, la conexión se degradará inexorablemente con la cantidad de dispositivos conectados y su consumo de datos.

Velocidad de los Proveedores cuando se contrata el servicio de internet, la velocidad de interconexión del mismo depende de cuánto sea el cobro, pero en la mayoría de los casos se tendrá una velocidad de descarga mayor a la velocidad de carga y nuestra red será una intranet (dirección de IP dinámica) compartida con otros miles de usuarios abonados al servicio de internet del proveedor. También es posible contratar una interconexión con velocidades homogéneas para carga y descarga dedicada, el costo del mismo se puede hasta triplicar con respecto a uno no homogéneo.

En caso de requerir una dirección de internet homologada o pública, el costo de contratar el servicio aumenta considerablemente, pero de esta forma nuestros equipos son visibles en el Internet (esto conlleva un aumento de riesgos al estar nuestros equipos más vulnerables a ataques informáticos).

El Tráfico Global en Internet El tráfico en Internet a nivel global sigue creciendo de manera vertiginosa. Sólo en 2024 aumentó un 17.2%, según los datos del informe de 2024 Cloudflare Radar Year in Review, publicado por la compañía por quinto año consecutivo, y en el que se repasa información sobre conectividad, seguridad, uso de dispositivos para acceso a la Red y frecuencia de los apagones, entre otras tendencias.

Según este informe, y para sorpresa de prácticamente nadie a estas alturas, los servicios de Internet más populares del mundo son Google, Facebook, Apple, TikTok y AWS. Chrome, con un 65.8% de los internautas como usuarios, es el navegador más utilizado en todo el mundo. WhatsApp sigue siendo la aplicación de mensajería más popular.

React, PHP y JQuery están entre las tecnologías para desarrollar webs más populares. HubSpot, Google y WordPress están entre los proveedores más populares de servicios y plataformas de soporte. Además, Go, ha superado a NodeJS como lenguaje más popular para hacer peticiones de API automatizadas.

Los rastreadores de IA son una gran fuente de tráfico, aunque despier-

tan cada vez más suspicacias por su actividad. Básicamente, se dedican a escanear la web para recopilar grandes cantidades de datos para entrenar modelos grandes de lenguaje. Preocupa bastante el hecho de que algunos recojan datos sin que tengan permiso para hacerlo, frente a los bots «buenos» y verificados, que suelen tener su origen en los motores de búsqueda y son transparentes sobre lo que son y lo que hacen. En este grupo entran GoogleBot, Qualys o BingBot.

Cloudflare, entre otras actividades, se dedica a rastrear el tráfico relacionado con la IA para determinar qué bots son los más agresivos, cuáles tienen el mayor volumen de peticiones y cuáles hacen rastreos de manera regular. Según los investigadores que han realizado el informe, el denominado facebookexternalhit es el que más tráfico de todos ha generado durante el año: un 27.16%. Le siguen Bytespider, de ByteDance, con un 23.35%; Amazonbot, con un 13.34%, ClaudeBot, de Anthropic, con un 8.06%; y GPTBot, con un 5.60%.

Al parecer, el tráfico generado por Bytespider, bot del propietario de TikTok, ha ido descendiendo de manera gradual durante 2024. En las semanas finales del año generaba entre un 80% y un 85% menos de tráfico que al principio. Mientras tanto el de ClaudeBot registró una subida muy pronunciada hacia mitad de 2024, y luego cayó otra vez de forma brusca.

La mayoría de las peticiones Web son todavía de HTTP2, que se lanzó en 2015, y cuenta aún con un 49.6% de las mismas. Un 29.9% todavía son de HTTP original, estandarizado en 1996, mientras que solo un 20.5% son de HTTP3, desplegado en 2022.

Cloudflare también registra, además del tráfico HTTP, las conexiones realizadas a través del protocolo de transmisión TCP, que asegura que hay una transferencia de datos fiable entre dispositivos de red. En 2024, un 20.7% de las conexiones TCP finalizaron de manera inesperada antes de que pudiesen terminar en un intercambio de datos útiles.

Estas anomalías en las conexiones TCP pueden deberse a varios motivos. Entre ellos a los ataques de denegación de servicio (DoS), al escaneado de redes, a las desconexiones de clientes, a la manipulación de las conexiones o a lo que Cloudflare ha llamado «comportamiento poco habitual del cliente».

La mayor parte de las interrupciones de una conexión TCP identificadas por Cloudflare se produjeron en 2024 después de que un servidor recibiese una petición de sincronización, pero antes de que recibiese un reconocimiento de la misma.

En cuanto a la seguridad, Cloudflare ha señalado que sólo un 4.3% de los

mensajes de correo electrónico enviados en 2024 eran maliciosos. En estos había sobre todo enlaces falsos (42.9%) o identidades falsas (35.1%). En el 70% de los casos se utilizaban los dos métodos.

Como datos curiosos en cuanto a seguridad, la compañía asegura que la vulnerabilidad Log4j todavía se usa como método de ataque, y se usa de manera mucho más activa que otras vulnerabilidades comunes. Además, prácticamente todos los mensajes de correo procesados por Cloudflare con dominios .bar, .rest y .uno eran o correos de Spam, o mensajes que, directamente, eran maliciosos.

En 2024, por otro lado, hubo 225 apagones de Internet de gran envergadura. La mayoría se dieron en África, Oriente Medio o India. Más de la mitad fueron apagones ordenados directamente por gobiernos, mientras que otros se debieron a otras causas.

Entre ellas, corte de cables, apagones eléctricos, problemas técnicos o de mantenimiento, episodios climáticos graves o ciberataques. Muchos duraron solo unas pocas horas, mientras que otros, como el experimentado en Bangladesh en julio, duró 10 días. Dentro de estos apagones está también el provocado por el incidente de CrowdStrike el pasado verano.

Otro de los aspectos tratados en el informe es la calidad de la conexión a Internet de los países, con base en su velocidad de subida, la de bajada y su latencia. España, con una velocidad de descarga media de 292.6 Mbps, y de 192.6 Mbps de subida, está a la cabeza en velocidad de conexión. Todos los países desarrollados presentan velocidades de descarga por encima de 200 Mbps de media.

Un 41.3% del tráfico de Internet a nivel mundial procede de dispositivos móviles, con el otro 58.7% producido en ordenadores portátiles y de sobremesa. Eso sí, en alrededor de un centenar de países del mundo, la mayoría del tráfico en 2024 procede de dispositivos móviles. Cuba y Siria tienen el mayor tráfico de dispositivos móviles, un 77%.

Otras áreas de alta demanda de tráfico de dispositivos de este tipo son Oriente Medio, África, Asia-Pacífico y América Central y del Sur. En este aspecto, las mediciones del tráfico son parecidas a las de 2023 y 2022.

Información de mi Computadora Si lo que deseamos es un listado detallado del Hardware de nuestro equipo de cómputo, entonces podemos usar cualquiera de estos comandos (que previamente deberemos instalar):

\$ lscpu

```
# lshw
# dmidecode
# hwinfo
```

8.2 ¿Que es Computación Paralela?

En el sentido más simple, la computación paralela es el uso simultáneo de múltiples recursos computacionales para resolver un problema computacional:

- Un problema se divide en partes discretas que se pueden resolver simultáneamente
- Cada parte se descompone en una serie de instrucciones
- Las instrucciones de cada parte se ejecutan simultáneamente en diferentes procesadores
- Se emplea un mecanismo global de control/coordinación

¿Por qué se hace programación paralela? El hecho de que la programación paralela sea un paradigma da cuenta de que existe una razón por la cual no ha dejado de ser necesaria o no ha sido totalmente automatizable, igualmente hay otras razones interesantes detrás para entender la existencia, actualidad y contemporaneidad de la programación paralela:

- Ley de Moore: Esta ley propuesta por Gordon E. Moore en 1965 dice resumidamente que el número de transistores en un Chip determinado se doblaría cada dos años. Esto quiere decir un aumento del rendimiento en los procesadores del alrededor del 50%, esto se traduce en escalar la velocidad de reloj de los procesadores, pero esta ley no es fidedigna desde el 2002 dónde solo ha habido un 20%, lo cuál sigue siendo un aumento considerable, sin embargo, no sería suficiente para que todos los avances en computación que se han logrado hasta el día y las necesidades de procesamiento en crecimiento exponencial puedan satisfacerse totalmente.
- Overclocking infinito: El Overclocking tiene un límite a pesar de que existiera una refrigeración perpetúa y adecuada del procesador. Esto

es debido a las corrientes parásitas que impiden una velocidad teóricamente infinita a la cual los circuitos pueden cambiar entre estados, o de hecho sus transistores.

- **Automatización del paralelismo:** Se dice en este paradigma que el éxito es inversamente proporcional al número de cores precisamente porque existen complejidades en el corazón del paralelismo que implican cosas que todavía no se pueden predecir ni con inteligencia artificial, se menciona cuales son las posibles estrategias para atacar un problema de forma paralela, esto da cuenta de que existe una forma prácticamente determinada de abordarlos pero no de automatizarlos, a pesar de que sí existan algunas partes que son automatizables en el proceso.
- **Solución en el Hardware:** Un diseño adecuado del Hardware permitiría que la paralelización siempre estuviera presente con respecto a los procesadores que se están usando de tal modo que alguno los problemas que son inherentes al paradigma pudieran evitarse. Esto ha resultado imposible hasta la fecha, de hecho, solo diseñar solamente algo tan efectivo y tradicional como se ha hecho en programación secuencial es algo que no existe hasta ahora. Existen algunas aproximaciones como OpenMP y programación por hilos de las que hablaremos más adelante.

Ventajas

- Resuelve problemas que no se podrían realizar en una sola CPU
- Resuelve problemas que no se pueden resolver en un tiempo razonable
- Permite ejecutar problemas de un orden y complejidad mayor
- Permite ejecutar código de manera más rápida (aceleración)
- Permite ejecutar en general más problemas
- Obtención de resultados en menos tiempo
- Permite la ejecución de varias instrucciones en simultáneo
- Permite dividir una tarea en partes independientes

Desventajas

- Mayor consumo de energía
- Mayor dificultad a la hora de escribir programas
- Dificultad para lograr una buena sincronización y comunicación entre las tareas
- Retardos ocasionados por comunicación entre tareas
- Número de componentes usados es directamente proporcional a los fallos potenciales
- Condiciones de carrera
 - Múltiples procesos se encuentran en condición de carrera si el resultado de los mismos depende del orden de su llegada
 - Si los procesos que están en condición de carrera no son correctamente sincronizados, puede producirse una corrupción de datos

Paralelismo de Grano Fino, Grano Grueso y Paralelismo Vergonzoso las aplicaciones a menudo se clasifican según la frecuencia con que sus subtareas se sincronizan o comunican entre sí. Una aplicación muestra un paralelismo de grano fino si sus subtareas deben comunicarse muchas veces por segundo, se considera paralelismo de grano grueso si no se comunican muchas veces por segundo, y es vergonzosamente paralelo si nunca o casi nunca se tienen que comunicar. Aplicaciones vergonzosamente paralelas son consideradas las más fáciles de paralelizar.

El tipo de problemas complejos que aborda el cómputo de alto rendimiento (HPC por High-Performance Computing) no se pueden resolver con una computadora de escritorio y tienen una escala determinada: toma mucho tiempo resolverlos, se necesita una gran cantidad de memoria (RAM), se tienen que realizar muchísimos experimentos parecidos y hay restricciones concretas de tiempo para encontrar resultados. La mayoría de estas temáticas están relacionadas con la innovación en la industria y su aplicación en áreas como energía, medio ambiente, biotecnología, medicina, ingeniería, etc.

A su vez, la evolución del procesador la cantidad de transistores sigue creciendo pero la velocidad de los núcleos individuales (core) casi no aumenta. Podríamos comprar un procesador de 128 núcleos pero el desafío de

aprovechar la potencia de un procesador es cómo distribuimos el tiempo de ejecución de una misma tarea computacional.

De este modo, cada tarea computacional se divide en dos partes: serial y paralelizable. Como todas las aplicaciones tienen una parte secuencial, el tiempo de cómputo de la aplicación paralela está acotado por esa sección serial (Ley de Ahmdahl). Esto implica dos cuestiones fundamentales:

1. Es necesario reducir lo más posible el tiempo de ejecución serial
2. En una tarea computacional existe un máximo de procesadores que conviene poner a trabajar en forma paralela, más allá de ese punto no se justifica seguir paralelizando la tarea.

¿Qué debe tener en cuenta un centro de HPC para aprovechar al máximo los recursos de cómputo? Cada centro de HPC tiene que configurar, optimizar aplicaciones y sistemas operativos. Allí el monitoreo es clave, con la escala de las infraestructuras también surgen aplicaciones con problemas de memoria (cuello de botella) y la necesidad de utilizar mejores redes (PARAMNet-3, Infiniband, GigE). Tener poder de cómputo no es barato y solamente con el Hardware no alcanza. El equipo de operaciones de HPC necesita un alto grado de conocimiento y de disciplina para encarar la tarea.

Con el uso del cómputo de alto desempeño es posible por ejemplo reducir el tiempo que nos lleva descubrir nuevos medicamentos de años a meses. Una computadora de alto rendimiento es capaz de resolver este y otros tipos de problemas científicos avanzados mediante simulaciones, modelos y análisis. Estos sistemas abren las puertas de la “Cuarta Revolución Industrial”, ya que ayudan a resolver muchas de las problemáticas más importantes del mundo. Los sistemas HPC ya se utilizan para las siguientes tareas:

- Descubrir nuevos componentes de drogas y probar los conocidos para combatir diferentes tipos de cáncer y otras enfermedades
- Simular dinámicas moleculares para crear nuevos materiales, como tejidos balísticos
- Pronosticar cambios climáticos considerables para mejorar la preparación de las comunidades afectadas

Las supercomputadoras representan lo último en sistemas HPC. La definición de supercomputadora depende de diferentes estándares que van cambiando a medida que las capacidades evolucionan. Un solo clúster de supercomputadoras puede incluir decenas de miles de procesadores, y los sistemas más caros y potentes del mundo pueden costar más de US\$ 100 millones de dólares.

Casos de Uso Emergentes a medida que las tecnologías mejoraron, la computación de alto rendimiento comenzó a implementarse en una gama de capacidades más amplia. Ahora contamos con más potencia de procesamiento y memoria que nunca para solucionar problemas más complejos.

- Aprendizaje automático: como subconjunto de la inteligencia artificial (IA), el aprendizaje automático (AA) se refiere a un sistema que tiene la capacidad de aprender de forma activa por sí mismo, a diferencia de recibir, de forma pasiva, instrucciones para ejecutar. Los sistemas HPC pueden utilizarse en aplicaciones de AA altamente avanzadas donde se analizan grandes cantidades de datos, por ejemplo, investigación del cáncer para detectar el melanoma en imágenes.
- Análisis de grandes conjuntos de datos: se recurre a la comparación rápida y a la correlación de grandes conjuntos de datos para complementar investigaciones y resolver problemas académicos, científicos, financieros, comerciales, gubernamentales, de salud y de seguridad cibernética. Este trabajo requiere un rendimiento masivo y capacidades de cómputo de una potencia enorme. Con un estimado de 50 petabytes de datos al año generados por las misiones, la NASA se apoya en la supercomputación para analizar observaciones y realizar simulaciones con grandes conjuntos de información.
- Modelado avanzado y simulación: al no tener que realizar un montaje físico en las primeras etapas del proceso, el modelado avanzado y la simulación permiten que las empresas ahorren tiempo, materiales y costos de contratación de personal para lanzar sus productos al mercado con mayor rapidez. El modelado y la simulación en HPC se aplican en el descubrimiento y la prueba de fármacos, diseños automotrices y aeroespaciales, pronóstico de sistemas climáticos o meteorológicos, y aplicaciones energéticas.

Cómo Funciona la Computación de Alto Rendimiento Existen dos métodos principales para procesar la información en HPC:

- Procesamiento en serie: es el que realizan las unidades de procesamiento central (CPU). Cada núcleo de CPU, por lo general, realiza solo una tarea a la vez. Las CPU son fundamentales para ejecutar diferentes funciones, como sistemas operativos y aplicaciones básicas (por ej., procesamiento de textos, productividad en la oficina).
- Procesamiento en paralelo: es el que se puede realizar mediante varias CPU o unidades de procesamiento de gráficos (GPU). Las GPU, diseñadas originalmente para gráficos independientes, son capaces de realizar diferentes operaciones aritméticas por medio de una matriz de datos (como píxeles de pantalla) de forma simultánea. La capacidad para trabajar en varios planos de datos al mismo tiempo hace que las GPU sean la elección natural para el procesamiento en paralelo en tareas de aplicaciones de aprendizaje automático (AA), como el reconocimiento de objetos en videos.

Para superar los límites de la supercomputación, se necesitan diferentes arquitecturas de sistemas. Para que el procesamiento en paralelo pueda llevarse a cabo, la mayoría de los sistemas HPC integran varios procesadores y módulos de memoria a través de interconexiones con un ancho de banda enorme. Ciertos sistemas HPC combinan varias CPU y GPU, lo que se conoce como computación heterogénea.

La potencia de procesamiento de las computadoras se mide en unidades llamadas “FLOPS”⁷³ (operaciones de punto flotante por segundo). A principios de 2019, la supercomputadora más potente que existe alcanzó los 143,5 peta-Flops (143×10^{15}). Este tipo de supercomputadora se llama equipo de petascale y puede realizar más de mil billones de FLOPS. Por su parte, una computadora de escritorio para juegos de alta gama es más de un millón de veces más lenta y llega apenas a los 200 giga-FLOPS (1×10^9).

Gracias a los avances tanto en procesamiento como rendimiento, en el año 2020 fuimos testigos de un nuevo salto en la era de la supercomputación: la

⁷³ “FLOPS” describe una velocidad de procesamiento teórica: para hacer posible esa velocidad es necesario enviar datos a los procesadores de forma continua. Por lo tanto, el procesamiento de los datos se debe tener en cuenta en el diseño del sistema. La memoria del sistema, junto con las interconexiones que unen los nodos de procesamiento entre sí, impactan en la rapidez con la que los datos llegan a los procesadores.

exaescala, que es casi 1000 veces más rápida que la petaescala. Esto significa que un sistema de exaescala realiza 10^{18} (o mil millones por mil millones) operaciones por segundo. Necesitaríamos 5 millones de computadoras de escritorio para alcanzar el nivel de rendimiento de procesamiento en supercomputación de 1 exa-Flops, suponiendo que cada computadora de escritorio es capaz de realizar 200 giga-FLOPS.

Arquitecturas de Software y Hardware En las dos siguientes secciones se explican en detalle las dos clasificaciones de computadoras más conocidas en la actualidad. La primera clasificación, es la clasificación clásica de Flynn en donde se tienen en cuenta sistemas con uno o varios procesadores, la segunda clasificación es moderna en la que sólo se tienen en cuenta los sistemas con más de un procesador.

El objetivo es presentar de una forma clara los tipos de clasificación que existen en la actualidad desde el punto de vista de distintos autores, así como cuáles son las ventajas e inconvenientes que cada uno ostenta, ya que es común que al resolver un problema particular se usen una o más arquitecturas de Hardware interconectadas generalmente por red.

8.3 Clasificación Clásica de Flynn

La clasificación clásica de arquitecturas de computadoras que hace alusión a sistemas con uno o varios procesadores fue realizada por Michael J. Flynn y la publicó por primera vez en 1966 y por segunda vez en 1970.

Esta taxonomía se basa en el flujo que siguen los datos dentro de la máquina y de las instrucciones sobre esos datos. Se define como flujo de instrucciones al conjunto de instrucciones secuenciales que son ejecutadas por un único procesador y como flujo de datos al flujo secuencial de datos requeridos por el flujo de instrucciones. Con estas consideraciones, Flynn clasifica los sistemas en cuatro categorías:

Single Instruction stream, Single Data stream (SISD) Los sistemas Monoprocesador de este tipo se caracterizan por tener un único flujo de instrucciones sobre un único flujo de datos, es decir, se ejecuta una instrucción detrás de otra. Este es el concepto de arquitectura serie de Von Neumann donde, en cualquier momento, sólo se ejecuta una única instrucción, un ejemplo de estos sistemas son las máquinas secuenciales convencionales.

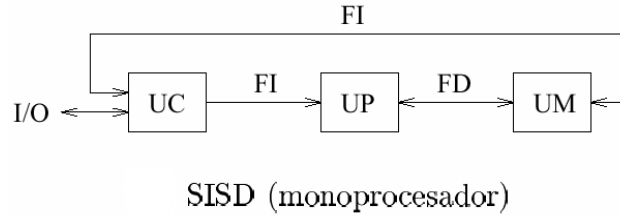


Figura 1: Ejemplo de máquina SISD

Single Instruction stream, Multiple Data stream (SIMD) Estos sistemas de procesador Matricial tienen un único flujo de instrucciones que operan sobre múltiples flujos de datos. Ejemplos de estos sistemas los tenemos en las máquinas vectoriales con Hardware escalar y vectorial.

El procesamiento es síncrono, la ejecución de las instrucciones sigue siendo secuencial como en el caso anterior, todos los elementos realizan una misma instrucción pero sobre una gran cantidad de datos. Por este motivo existirá concurrencia de operación, es decir, esta clasificación es el origen de la máquina paralela.

El funcionamiento de este tipo de sistemas es el siguiente. La unidad de control manda una misma instrucción a todas las unidades de proceso (ALUs). Las unidades de proceso operan sobre datos diferentes pero con la misma instrucción recibida.

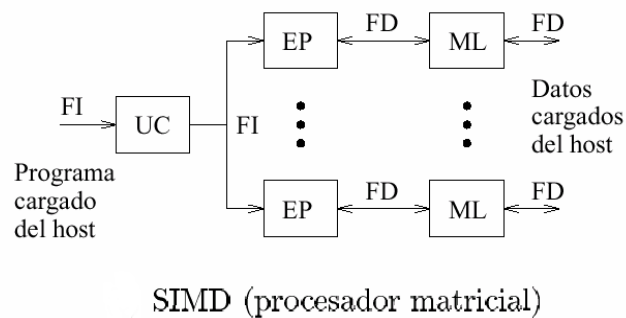


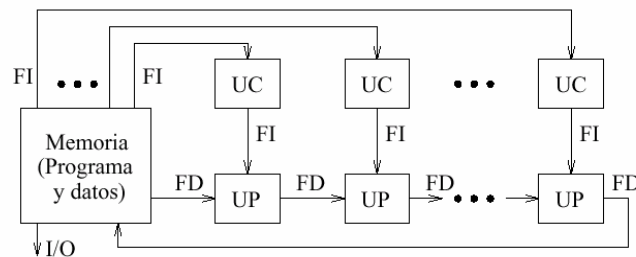
Figura 2: Ejemplo de máquina SIMD

Existen dos alternativas distintas que aparecen después de realizarse esta clasificación:

- Arquitectura Vectorial con segmentación, una CPU única particionada en unidades funcionales independientes trabajando sobre flujos de datos concretos.
- Arquitectura Matricial (matriz de procesadores), varias ALUs idénticas a las que el procesador da instrucciones, asigna una única instrucción pero trabajando sobre diferentes partes del programa.

Multiple Instruction stream, Single Data stream (MISD) Sistemas con múltiples instrucciones Array Sistólico que operan sobre un único flujo de datos. Este tipo de sistemas no ha tenido implementación hasta hace poco tiempo. Los sistemas MISD se contemplan de dos maneras distintas:

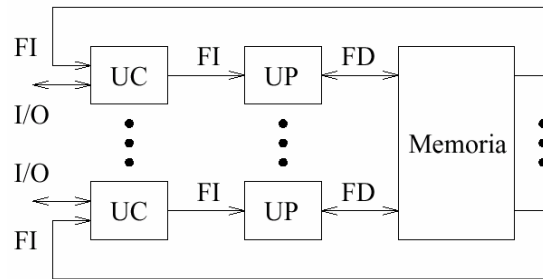
- Varias instrucciones operando simultáneamente sobre un único dato.
- Varias instrucciones operando sobre un dato que se va convirtiendo en un resultado que será la entrada para la siguiente etapa. Se trabaja de forma segmentada, todas las unidades de proceso pueden trabajar de forma concurrente.



MISD (array sistólico)

Figura 3: Ejemplo de máquina MISD

Multiple Instruction stream, Multiple Data stream (MIMD) Sistemas con un flujo de múltiples instrucciones Multiprocesador que operan sobre múltiples datos. Estos sistemas empezaron a utilizarse antes de la década de los 80s. Son sistemas con memoria compartida que permiten ejecutar varios procesos simultáneamente (sistema multiprocesador).



MIMD (multiprocesador)

Figura 4: Ejemplo de máquina MIMD

Cuando las unidades de proceso reciben datos de una memoria no compartida estos sistemas reciben el nombre de MULTIPLE SISD (MSISD). En arquitecturas con varias unidades de control (MISD Y MIMD), existe otro nivel superior con una unidad de control que se encarga de controlar todas las unidades de control del sistema -ejemplo de estos sistemas son las máquinas paralelas actuales-.

8.4 Categorías de Computadoras Paralelas

Clasificación moderna que hace alusión única y exclusivamente a los sistemas que tienen más de un procesador (i.e máquinas paralelas). Existen dos tipos de sistemas teniendo en cuenta su acoplamiento:

- Los sistemas fuertemente acoplados son aquellos en los que los procesadores dependen unos de otros.
- Los sistemas débilmente acoplados son aquellos en los que existe poca interacción entre los diferentes procesadores que forman el sistema.

Atendiendo a esta y a otras características, la clasificación moderna divide a los sistemas en dos tipos: Sistemas multiprocesador (fuertemente acoplados) y sistemas multicomputadoras (débilmente acoplados).

8.4.1 Equipo Paralelo de Memoria Compartida

Un multiprocesador puede verse como una computadora paralela compuesta por varios procesadores interconectados que comparten un mismo sistema de memoria.

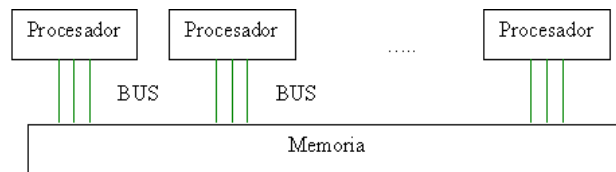


Figura 5: Arquitectura de una computadora paralela con memoria compartida

Los sistemas multiprocesadores son arquitecturas MIMD con memoria compartida. Tienen un único espacio de direcciones para todos los procesadores y los mecanismos de comunicación se basan en el paso de mensajes desde el punto de vista del programador.

Dado que los multiprocesadores comparten diferentes módulos de memoria, pueden acceder a un mismo módulo varios procesadores, a los multiprocesadores también se les llama sistemas de memoria compartida.

Para hacer uso de la memoria compartida por más de un procesador, se requiere hacer uso de técnicas de semáforos que mantienen la integridad de la memoria; esta arquitectura no puede crecer mucho en el número de procesadores interconectados por la saturación rápida del bus o del medio de interconexión.

Dependiendo de la forma en que los procesadores comparten la memoria, se clasifican en sistemas multiprocesador UMA, NUMA, COMA y Pipeline, que explicamos a continuación:

Uniform Memory Access (UMA) Sistema multiprocesador con acceso uniforme a memoria. La memoria física es uniformemente compartida por todos los procesadores, esto quiere decir que todos los procesadores tienen

el mismo tiempo de acceso a todas las palabras de la memoria. Cada procesador tiene su propia Caché privada y también se comparten los periféricos.

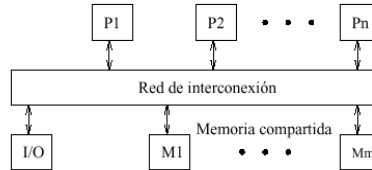


Figura 6: Acceso Uniforme a la memoria UMA

Los multiprocesadores son sistemas fuertemente acoplados (tightly-coupled), dado el alto grado de compartición de los recursos (Hardware o Software) y el alto nivel de interacción entre procesadores, lo que hace que un procesador depende de lo que hace otro. El sistema de interconexión debe ser rápido y puede ser de uno de los siguientes tipos: bus común, red Crossbar⁷⁴ y red Multietapa. Este modelo es conveniente para aplicaciones de propósito general y de tiempo compartido por varios usuarios, existen dos categorías de sistemas UMA.

- Sistema Simétrico: cuando todos los procesadores tienen el mismo tiempo de acceso a todos los componentes del sistema (incluidos los periféricos), reciben el nombre de sistemas multiprocesador simétrico. Los procesadores tienen el mismo dominio (prioridad) sobre los periféricos y cada procesador tiene la misma capacidad para procesar.
- Sistema Asimétrico: son sistemas con procesador principal y procesadores subordinados, en donde sólo el primero puede ejecutar aplicaciones y dónde el tiempo de acceso para diferentes procesadores no es el mismo. Los procesadores subordinados (Attached) ejecutan código usuario bajo la supervisión del principal, por lo tanto cuando una aplicación es ejecutada en un procesador principal dispondrá de una cierta prioridad.

⁷⁴Red reconfigurable que permite la conexión de cada entrada con cualquiera de las salidas, es decir, permite cualquier permutación.

Non Uniform Memory Access (NUMA) Un sistema multiprocesador NUMA es un sistema de memoria compartida donde el tiempo de acceso varía según donde se encuentre localizado el acceso.

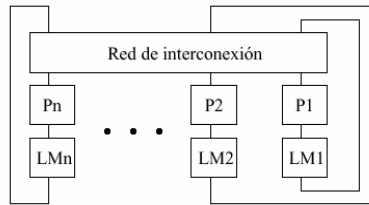


Figura 7: Acceso no uniforme a la memoria NUMA

El acceso a memoria, por tanto, no es uniforme para diferentes procesadores, existen memorias locales asociadas a cada procesador y estos pueden acceder a datos de su memoria local de una manera más rápida que a las memorias de otros procesadores, debido a que primero debe aceptarse dicho acceso por el procesador del que depende el módulo de memoria local.

Todas las memorias locales conforman la memoria global compartida y físicamente distribuida y accesible por todos los procesadores.

Cache Only Memory Access (COMA) Los sistemas COMA son un caso especial de los sistemas NUMA. Este tipo de sistemas no ha tenido mucha trascendencia, al igual que los sistemas SIMD.

Las memorias distribuidas son memorias Cachés, por este motivo es un sistema muy restringido en cuanto a la capacidad de memoria global. No hay jerarquía de memoria en cada módulo procesador. Todas las Cachés forman un mismo espacio global de direcciones. El acceso a las Cachés remotas se realiza a través de los directorios distribuidos de las Cachés.

Dependiendo de la red de interconexión utilizada, se pueden utilizar jerarquías en los directorios para ayudar a la localización de copias de bloques de Caché.

Procesador Vectorial Pipeline En la actualidad es común encontrar en un solo procesador los denominados Pipeline o Procesador Vectorial Pipeline del tipo MISD. En estos procesadores los vectores fluyen a través de las unidades aritméticas Pipeline.

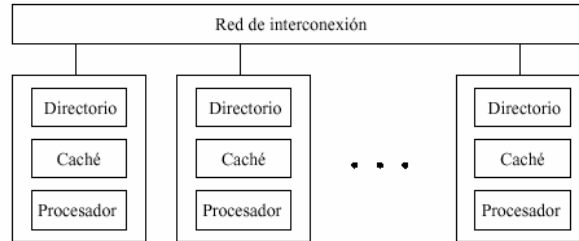


Figura 8: Ejemplo de COMA

Las unidades constan de una cascada de etapas de procesamiento compuestas de circuitos que efectúan operaciones aritméticas o lógicas sobre el flujo de datos que pasan a través de ellas, las etapas están separadas por registros de alta velocidad usados para guardar resultados intermedios. Así la información que fluye entre las etapas adyacentes está bajo el control de un reloj que se aplica a todos los registros simultáneamente.

8.4.2 Equipo Paralelo de Memoria Distribuida

Los sistemas multicomputadoras (los más comunes son los Clusters) se pueden ver como una computadora paralela en el cual cada procesador tiene su propia memoria local. En estos sistemas la memoria se encuentra distribuida y no compartida como en los sistemas multiprocesador. Los procesadores se comunican a través de paso de mensajes, ya que éstos sólo tienen acceso directo a su memoria local y no a las memorias del resto de los procesadores.

La transferencia de los datos se realiza a través de la red de interconexión que conecta un subconjunto de procesadores con otro subconjunto. La transferencia de unos procesadores a otros se realiza por múltiples transferencias entre procesadores conectados dependiendo del establecimiento de dicha red.

Dado que la memoria está distribuida entre los diferentes elementos de proceso, estos sistemas reciben el nombre de distribuidos. Por otra parte, estos sistemas son débilmente acoplados, ya que los módulos funcionan de forma casi independiente unos de otros. Este tipo de memoria distribuida es de acceso lento por ser peticiones a través de la red, pero es una forma muy efectiva de tener acceso a un gran volumen de memoria.

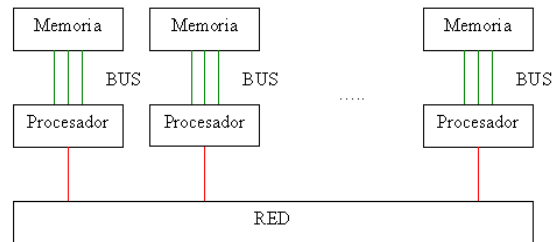


Figura 9: Arquitectura de una computadora paralela con memoria distribuida

8.4.3 Equipo Paralelo de Memoria Compartida-Distribuida

La tendencia actual en las máquinas paralelas es de aprovechar las facilidades de programación que ofrecen los ambientes de memoria compartida y la escalabilidad de los ambientes de memoria distribuida. En este modelo se conectan entre sí módulos de multiprocesadores, pero se mantiene la visión global de la memoria a pesar de que es distribuida.

Clusters El desarrollo de sistemas operativos y compiladores del dominio público (Linux y Software GNU), estándares para interfaz de paso de mensajes (Message Passing Interface MPI), conexión universal a periféricos (Peripheral Component Interconnect PCI), entre otros, han hecho posible tomar ventaja de los recursos económicos computacionales de producción masiva (procesadores, discos, redes).

La principal desventaja que presenta a los proveedores de multicomputadoras es que deben satisfacer una amplia gama de usuarios, es decir, deben ser generales. Esto aumenta los costos de diseños y producción de equipos, así como los costos de desarrollo de Software que va con ellos: sistema operativo, compiladores y aplicaciones. Todos estos costos deben ser añadidos cuando se hace una venta. Por supuesto alguien que sólo necesita procesadores y un mecanismo de pase de mensajes no debería pagar por todos estos añadidos que nunca usará. Estos usuarios son los que están impulsando el uso de Clusters principalmente de computadoras personales (PC), cuya arquitectura se muestra a continuación:

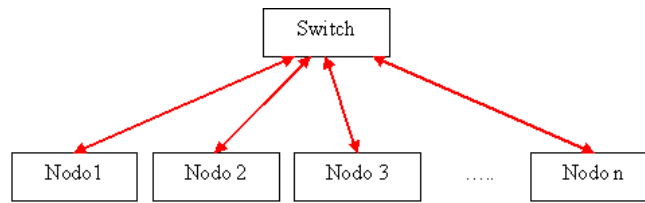


Figura 10: Arquitectura de un cluster

Los Cluster⁷⁵ se pueden clasificar en dos tipos según sus características físicas:

- Cluster homogéneo: si todos los procesadores y/o nodos participantes en el equipo paralelo son iguales en capacidad de cómputo -es permitido variar la cantidad de memoria o disco duro en cada procesador-
- Cluster heterogéneo: es aquel en que al menos uno de los procesadores y/o nodos participantes en el equipo paralelo son de distinta capacidad de cómputo

El Clustering no presenta dependencias a nivel de Hardware (no todos los equipos necesitan el mismo Hardware) ni a nivel de Software (no necesitan el mismo sistema operativo). Este tipo de sistemas disponen de una interfaz que permite dirigir el comportamiento de los Clusters. Dicha interfaz es la encargada de la interacción con usuarios y procesos, realizando la división de la carga entre los diversos servidores que compongan el Cluster.

Los Clusters pueden formarse de diversos equipos; los más comunes son los de computadoras personales, pero es creciente el uso de computadoras multiprocesador de más de un procesador de memoria compartida interconectados por red con los demás nodos del mismo tipo, incluso el uso de computadoras multiprocesador de procesadores vectoriales Pipeline. Los Clusters armados con la configuración anterior tienen grandes ventajas para procesamiento paralelo:

- La reciente explosión en redes implica que la mayoría de los componentes necesarios para construir un Cluster son vendidos

⁷⁵Existe el Ranking de las 500 supercomputadoras más poderosas del mundo (esta se actualiza cada seis meses en junio y noviembre) y puede ser consultada en:

<https://top500.org>

en altos volúmenes y por lo tanto son económicos. Ahorros adicionales se pueden obtener debido a que sólo se necesitará una tarjeta de vídeo, un monitor y un teclado por Cluster. El mercado de los multiprocesadores es más reducido y más costoso

- El reemplazar un componente defectuoso en un Cluster es relativamente trivial comparado con hacerlo en un multiprocesador, permitiendo una mayor disponibilidad de Clusters cuidadosamente diseñados

Desventajas del uso de Clusters de computadoras personales para procesamiento paralelo:

- Con raras excepciones, los equipos de redes generales producidos masivamente no están diseñados para procesamiento paralelo y típicamente su latencia es alta y los anchos de banda pequeños comparados con multiprocesadores. Dado que los Clusters explotan tecnología que sea económica, los enlaces en el sistema no son veloces implicando que la comunicación entre componentes debe pasar por un proceso de protocolos de negociación lentos, incrementando seriamente la latencia. En muchos y en el mejor de los casos (debido a costos) se recurre a una red tipo Fast Ethernet restringimiento la escalabilidad del Cluster
- Hay poco soporte de Software para manejar un Cluster como un sistema integrado
- Los procesadores no son tan eficientes como los procesadores usados en los multiprocesadores para manejar múltiples usuarios y/o procesos. Esto hace que el rendimiento de los Clusters se degrade con relativamente pocos usuarios y/o procesos
- Muchas aplicaciones importantes disponibles en multiprocesadores y optimizadas para ciertas arquitecturas, no lo están en Clusters

Sin lugar a duda los Clusters presentan una alternativa importante para varios problemas particulares, no sólo por su economía, si no también porque pueden ser diseñados y ajustados para ciertas aplicaciones. Las aplicaciones que pueden sacar provecho de Clusters son en donde el grado de comunicación entre procesos es de bajo a medio.

Tipos de Cluster Básicamente existen tres tipo de Clusters, cada uno de ellos ofrece ventajas y desventajas, el tipo más adecuado para el cómputo científico es del de alto-rendimiento, pero existen aplicaciones científicas que pueden usar más de un tipo al mismo tiempo.

- Alta disponibilidad (Fail-over o High-Availability): este tipo de Cluster está diseñado para mantener uno o varios servicios disponibles incluso a costa de rendimiento, ya que su función principal es que el servicio jamás tenga interrupciones como por ejemplo un servicio de bases de datos de transacciones bancarias
- Alto rendimiento (HPC o High Performance Computing): este tipo de Cluster esta diseñado para obtener el máximo rendimiento de la aplicación utilizada incluso a costa de la disponibilidad del sistema, es decir el Cluster puede sufrir caídas, este tipo de configuración está orientada a procesos que requieran mucha capacidad de cálculo.
- Balanceo de Carga (Load-Balancing): este tipo de Cluster está diseñado para balancear la carga de trabajo entre varios servidores, lo que permite tener, por ejemplo, un servicio de cálculo intensivo multiusuarios que detecte tiempos muertos del proceso de un usuario para ejecutar en dichos tiempos procesos de otros usuarios.

Grids Son cúmulos (grupo de Clusters) de arquitecturas en paralelo interconectados por red, los cuales distribuyen tareas entre los Clusters que lo forman, estos pueden ser homogéneos o heterogéneos en cuanto a los nodos componentes del cúmulo. Este tipo de arquitecturas trata de distribuir cargas de trabajo acorde a las características internas de cada Cluster y las necesidades propias de cada problema, esto se hace a dos niveles, una en la parte de programación en conjunto con el balance de cargas y otra en la parte de Hardware que tiene que ver con las características de cada arquitectura que conforman el cúmulo.

Balance de Carga A la hora de diseñar un sistema paralelo con compartición de recursos, es necesario considerar cómo balancear la carga de trabajo. Se entiende este concepto, como la técnica usada para dividir el trabajo a compartir entre varios procesos, equipos de cómputo, u otros recursos. Está

muy relacionada con los sistemas multiproceso, que trabajan o pueden trabajar con más de una unidad para llevar a cabo su funcionalidad.

Para evitar los cuellos de botella, el balance de la carga de trabajo se reparte de forma equitativa a través de un algoritmo que estudia las peticiones del sistema y las redirecciona a la mejor opción.

Balance de Carga por Hardware presenta las siguientes características:

- A partir de un algoritmo de planificación de procesos -Round Robin, LRU-, examina las peticiones entrantes y selecciona el más apropiado entre los distintos elementos del sistema
- La selección del siguiente libre del sistema está basada en el algoritmo de sustitución y es aleatoria
- La sesión debe de ser mantenida por el desarrollador
- Al ser un proceso de Hardware, es muy rápido

Balance de Carga por Software presenta las siguientes características:

- Examinan la solicitud para garantizar que se puede satisfacer la demanda de los usuarios
- Distintas peticiones del mismo usuario son servidas simultáneamente o secuencialmente según se definan
- Más lentos que los balanceadores de Hardware
- Normalmente son soluciones baratas

Escalabilidad Se entiende por escalabilidad a la capacidad de adaptación y respuesta de un sistema con respecto al rendimiento del mismo a medida que aumentan de forma significativa la carga computacional del mismo. Aunque parezca un concepto claro, la escalabilidad de un sistema es un aspecto complejo e importante del diseño de sistemas paralelos.

La escalabilidad está íntimamente ligada al diseño de sistemas paralelos, influye en el rendimiento de forma significativa. Si una aplicación está bien

diseñada, la escalabilidad no constituye un problema. Analizando la escalabilidad, se deduce de la implementación y del diseño general del sistema. No es atributo del sistema configurable.

La escalabilidad supone un factor crítico en el crecimiento de un sistema paralelo. Si un sistema tiene como objetivo crecer la carga computacional -en el número de usuarios o procesos- manteniendo su rendimiento actual, tiene que evaluar dos posibles opciones:

- Con un Hardware de mayor potencia o
- Con una mejor combinación de Hardware y Software

Se pueden distinguir dos tipos de escalabilidad, vertical y horizontal:

- El escalar verticalmente o escalar hacia arriba, significa el añadir más recursos a un solo nodo en particular dentro de un sistema, tal como el añadir memoria o un disco duro más rápido a una computadora.
- La escalabilidad horizontal, significa agregar más nodos a un sistema, tal como añadir una computadora nueva a un programa de aplicación para espejo.

Escalabilidad Vertical El escalar hacia arriba de un sistema viene a significar una migración de todo el sistema a un nuevo Hardware que es más potente y eficaz que el actual. Una vez se ha configurado el sistema futuro, se realizan una serie de validaciones y copias de seguridad y se pone en funcionamiento. Las aplicaciones que estén funcionando bajo la arquitectura Hardware antigua no sufren con la migración, el impacto en el código es mínimo.

Este modelo de escalabilidad vertical tiene un aspecto negativo. Al aumentar la potencia en base a ampliaciones de Hardware, llegará un momento que existirá algún tipo de limitación de Hardware. Además a medida que se invierte en Hardware de muy altas prestaciones, los costos se disparan tanto de forma temporal -ya que si se ha llegado al umbral máximo, hay componentes de Hardware que tardan mucho tiempo en ampliar su potencia de forma significativa- como económicos. Sin embargo a nivel estructural no supone ninguna modificación reseñable, lo que la convierte en una buena opción si los costos anteriores son asumibles.

Escalabilidad Horizontal La escalabilidad horizontal consiste en potenciar el rendimiento del sistema paralelo desde un aspecto de mejora global, a diferencia de aumentar la potencia de una única parte del mismo. Este tipo de escalabilidad se basa en la modularidad de su funcionalidad, por ello suele estar conformado por una agrupación de equipos que dan soporte a la funcionalidad completa. Normalmente, en una escalabilidad horizontal se añaden equipos para dar más potencia a la red de trabajo.

Con un entorno de este tipo, es lógico pensar que la potencia de procesamiento es directamente proporcional al número de equipos en la red. El total de la potencia de procesamiento es la suma de la velocidad física de cada equipo transferida por la partición de aplicaciones y datos extendida a través de los nodos.

Si se aplica un modelo de escalabilidad basado en la horizontalidad, no existen limitaciones de crecimiento a priori. Como principal defecto, este modelo de escalabilidad supone una gran modificación en el diseño, lo que conlleva a una gran trabajo de diseño y reimplantación. Si la lógica se ha concebido para un único servidor, es probable que se tenga que estructurar el modelo arquitectónico para soportar este modelo de escalabilidad.

El encargado de cómo realizar el modelo de partición de datos en los diferentes equipos es el desarrollador. Existen dependencias en el acceso a la aplicación. Es conveniente, realizar una análisis de actividad de los usuarios para ir ajustando el funcionamiento del sistema. Con este modelo de escalabilidad, se dispone de un sistema al que se pueden agregar recursos de manera casi infinita y adaptable al crecimiento de cargas de trabajo y nuevos usuarios.

La escalabilidad cuenta como factor crítico en el crecimiento de usuarios. Es mucho más sencillo diseñar un sistema con un número constante de usuarios -por muy alto que sea este- que diseñar un sistema con un número creciente y variable de usuarios. El crecimiento relativo de los números es mucho más importante que los números absolutos.

8.4.4 Cómputo Paralelo en Multihilos

En una computadora, sea secuencial o paralela, para aprovechar las capacidades crecientes del procesador, el sistema operativo divide su tiempo de procesamiento entre los distintos procesos, de forma tal que para poder ejecutar a un proceso, el Kernel les asigna a cada uno una prioridad y con ello una fracción del tiempo total de procesamiento, de forma tal que se pueda

atender a todos y cada uno de los procesos de manera eficiente.

En particular, en la programación en paralelo usando MPI, cada proceso -que eventualmente puede estar en distinto procesador- se lanza como una copia del programa con datos privados y un identificador del proceso único, de tal forma que cada proceso sólo puede compartir datos con otro proceso mediante paso de mensajes.

Esta forma de lanzar procesos por cada tarea que se desee hacer en paralelo es costosa, por llevar cada una de ellas toda una gama de subprocesos para poderle asignar recursos por parte del sistema operativo. Una forma más eficiente de hacerlo es que un proceso pueda generar bloques de subprocesos que puedan ser ejecutados como parte del proceso (como sub-tareas), así en el tiempo asignado se pueden atender a más de un subproceso de manera más eficiente, esto es conocido como programación multihilos.

Los hilos realizarán las distintas tareas necesarias en un proceso. Para hacer que los procesos funcionen de esta manera, se utilizan distintas técnicas que le indican al Kernel cuales son las partes del proceso que pueden ejecutarse simultáneamente y el procesador asignará una fracción de tiempo exclusivo al hilo del tiempo total asignado al proceso.

Los datos pertenecientes al proceso pasan a ser compartidos por los subprocesos lanzados en cada hilo y mediante una técnica de semáforos el Kernel mantiene la integridad de estos. Esta técnica de programación puede ser muy eficiente si no se abusa de este recurso, permitiendo un nivel más de paralelización en cada procesador. Esta forma de paralelización no es exclusiva de equipos multiprocesadores o multicomputadoras, ya que pueden ser implementados a nivel de sistema operativo.

8.4.5 Cómputo Paralelo en CUDA

Son las siglas de arquitectura unificada de dispositivos de cómputo (Compute Unified Device Architecture CUDA) que hace referencia a una plataforma de computación en paralelo (que incluye su propia RAM conocida como GRAM para hacer óptima la solicitud de dicho recurso por sus múltiples cores) incluyendo un compilador y un conjunto de herramientas de desarrollo que permiten a los programadores usar una variación del lenguaje de programación C -Por medio de Wrappers se puede usar MatLab, Python, Julia, Fortran y Java en vez de C/C++- para codificar algoritmos en las unidades de procesamiento de gráficos (Graphics Processing Unit GPU).

CUDA intenta explotar las ventajas de las GPU (de decenas a centenas de

cores⁷⁶) frente a las CPU (decenas de cores⁷⁷) de propósito general utilizando el paralelismo que ofrecen sus múltiples cores o núcleos, que permiten el lanzamiento de un altísimo número de hilos simultáneos. Si una aplicación está bien diseñada utilizando numerosos hilos que realizan tareas independientes usando la memoria compartida de la tarjeta (que es lo que hacen las GPU al procesar gráficos, su tarea natural) hará un uso eficiente de este dispositivo.

Por ejemplo, la tarjeta gráfica de NVidia GEFORCE RTX 3090 proporciona 10,496 cores y 24 GB de GDDR6x, mientras que la tarjeta NVidia Titan RTX proporciona 130 Tensor⁷⁸ TFLOP de rendimiento, 576 núcleos tensores y 24 GB de memoria GDDR6.

Ahora, miles de desarrolladores, científicos e investigadores están encontrando innumerables aplicaciones prácticas para esta tecnología en campos como el procesamiento de vídeo e imágenes, la biología y la química computacional, la simulación de la dinámica de fluidos, la reconstrucción de imágenes de Tomografía Axial Computarizada TAC, el análisis sísmico o el trazado de rayos, entre otros.

Procesamiento paralelo con CUDA Los sistemas informáticos están pasando de realizar el «procesamiento central» en la CPU a realizar «coprocesamiento» repartido entre la CPU y la GPU. Para posibilitar este nuevo paradigma computacional, NVIDIA ha inventado la arquitectura de cálculo paralelo CUDA, que ahora se incluye en las GPUs GeForce, ION Quadro y Tesla GPUs, lo cual representa una base instalada considerable para los desarrolladores de aplicaciones.

CUDA ha sido recibida con entusiasmo por la comunidad científica. Por

⁷⁶La GRAM de las tarjetas CUDA es relativamente pequeña (a lo más de algunas decenas de Gigabytes), pero está diseñada para que pueda ser compartida por sus cores de forma óptima. Pero si es necesario que múltiples cores de la GPU soliciten datos a la RAM del equipo de cómputo, pueden ocasionar degradación en la eficiencia de la GPU porque la RAM tiene una arquitectura DDR4 o DDR5 (lo que significa que solo puede atender peticiones de datos de sólo 4 o 5 cores simultáneamente).

⁷⁷Los actuales CPUs pueden tener decenas de cores y comparten hasta Terabytes de RAM. Pero la arquitectura actual de la RAM es de tipo DDR4 o DDR5, esto significa, que sólo hay 4 o 5 canales para satisfacer las solicitudes de petición de datos de/hacia los cores, si más cores necesita acceso a la RAM tendrán que esperar que los otros cores concluyan su uso, mermando la eficiencia de la CPU.

⁷⁸Un Tensor core (o núcleos Tensor) calculan la operación de una matriz 4x4 completa, la cual se calcula por reloj. Estos núcleos pueden multiplicar dos matrices FP16 4x4 y sumar la matriz FP32 al acumulador.

ejemplo, se está utilizando para acelerar AMBER, un simulador de dinámica molecular empleado por más de 60,000 investigadores del ámbito académico y farmacéutico de todo el mundo para acelerar el descubrimiento de nuevos medicamentos. En el mercado financiero, Numerix y CompatibL introdujeron soporte de CUDA para una nueva aplicación de cálculo de riesgo de contraparte y, como resultado, se ha multiplicado por 18 la velocidad de la aplicación. Cerca de 400 instituciones financieras utilizan Numerix en la actualidad.

Un buen indicador de la excelente acogida de CUDA es la rápida adopción de la GPU Tesla para aplicaciones de GPU Computing. En la actualidad hay más de 700 Clusters de GPUs instalados en compañías publicadas en Fortune 500 de todo el mundo, lo que incluye empresas como Schlumberger y Chevron en el sector energético o BNP Pariba en el sector bancario. Y en el 2021 se puso en funcionamiento el cluster Perlmutter que esta formado por 6,159 NVIDIA A100 Tensor Core GPUs para ciencias astrofísicas y atmosféricas.

Por otra parte, la reciente llegada de los últimos sistemas operativos de Microsoft y Apple está convirtiendo el GPU Computing en una tecnología de uso masivo. En estos nuevos sistemas, la GPU no actúa únicamente como procesador gráfico, sino como procesador paralelo de propósito general accesible para cualquier aplicación.

Plataforma de Cálculo Paralelo CUDA proporciona unas cuantas extensiones de MatLab, Fortran, Python, Julia, C, C++, etc. Que permiten implementar el paralelismo en el procesamiento de tareas y datos con diferentes niveles de granularidad. El programador puede expresar ese paralelismo mediante diferentes lenguajes de alto nivel como Fortran, Python, Julia, C y C++ o mediante estándares abiertos como las directivas de OpenACC -que es un estándar de programación para el cómputo en paralelo desarrollado por Cray, CAPS, Nvidia y PGI diseñado para simplificar la programación paralela de sistemas heterogéneos de CPU/GPU-. En la actualidad, la plataforma CUDA se utiliza en miles de aplicaciones aceleradas en la GPU y en miles de artículos de investigación publicados, en las áreas de:

- Bioinformática
- Cálculo financiero
- Dinámica de fluidos computacional (CFD)

- Ciencia de los datos, analítica y bases de datos
- Defensa e Inteligencia
- Procesamiento de imágenes y visión computarizadas
- EDA (diseño automatizado)
- Aprendizaje automático, Inteligencia artificial
- Ciencia de los materiales
- Medios audiovisuales y entretenimiento
- Imágenes médicas
- Dinámica molecular
- Análisis numérico
- Química cuántica
- Exploración sísmica
- Mecánica estructural computacional
- Visualización e interacción de proteínas
- Modelos meteorológicos y climáticos

Algunas bibliotecas aceleradas en la GPU:

- Thrust C++ Template
- cuBLAS
- cuSPARSE
- NPP
- cuFFT

Algunos lenguajes de programación:

- CUDA C/C++, Fortran, Python, Julia
- .NET, Java

Algunos compiladores disponibles:

- OpenACC, paraleliza automáticamente los bucles de código Fortran o C utilizando directivas
- Compilador de autoparalelización de C y Fortran (de PGI) para CUDA C
- Compilador de autoparalelización HMPP de CAPS para CUDA C basado en C y Fortran
- Fortran, Python, Julia, Java, C++ y C
- Compilador de Fortran para CUDA de PGI
- Traductor de Fortran a C para CUDA
- FLAGON: librería de Fortran 95 para cálculo en la GPU
- Interfaz (Wrapper) de Python para CUDA: PyCUDA
- Wrapper de Java
- jCUDA: Java para CUDA
- Vínculos para las librerías BLAS y FFT de CUDA
- JaCUDA
- Integración de .NET para CUDA
- Thrust: librería de plantillas de C++ para CUDA
- CuPP : infraestructura de C++ para CUDA
- Libra: capa de abstracción de C/C++ para CUDA
- F# para CUDA

- Librería ArrayFire para acelerar el desarrollo de aplicaciones de GPU Computing en C, C++, Fortran y Python

Soporte de MATLAB, Mathematica, R, LabView:

- MATLAB, Mathematica, R, LabView
- MathWorks: Librerías MATLAB procesadas con GPUs NVIDIA
- Plugin Jacket basado en CUDA para MATLAB
- GPULib: librería de funciones matemáticas con vínculos para IDL y MATLAB
- Programación con CUDA en Mathematica de Wolfram
- Plugin de Mathematica para CUDA
- Habilitación del GPU Computing en el entorno estadístico de R
- Librería CUDA para LabVIEW de National Instruments
- Formas de usar CUDA desde LabVIEW CVI

Además existen herramientas de productividad y Cluster:

- Soporte de Eclipse para CUDA
- CUDA Occupancy Calculator
- Administrador de Clusters de cálculo para GPUs Tesla
- PBS Works para GPUs Tesla
- Scyld de Penguin Computing

8.5 Métricas de Desempeño

Las métricas de desempeño del procesamiento de alguna tarea en paralelo es un factor importante para medir la eficiencia y consumo de recursos al resolver una tarea con un número determinado de procesadores y recursos relacionados de la interconexión de éstos.

Entre las métricas para medir desempeño en las cuales como premisa se mantiene fijo el tamaño del problema, destacan las siguientes:

- Factor de aceleración
- Eficiencia
- Fracción serial

Cada una de ellas mide algo en particular y sólo la combinación de estas dan un panorama general del desempeño del procesamiento en paralelo de un problema en particular en una arquitectura determinada al ser comparada con otras.

Factor de Aceleración (o Speed-Up) Se define como el cociente del tiempo que se tarda en completar el cómputo de la tarea usando un sólo procesador entre el tiempo que necesita para realizarlo en p procesadores trabajando en paralelo:

$$s = \frac{T(1)}{T(p)} \quad (8.1)$$

se asume que se usará el mejor algoritmo tanto para un solo procesador como para p procesadores.

Esta métrica en el caso ideal debería de aumentar de forma lineal al aumento del número de procesadores.

Eficiencia Se define como el cociente del tiempo que se tarda en completar el cómputo de la tarea usando un solo procesador entre el número de procesadores multiplicado por el tiempo que necesita para realizarlo en p procesadores trabajando en paralelo:

$$e = \frac{T(1)}{pT(p)} = \frac{s}{p}. \quad (8.2)$$

Este valor será cercano a la unidad cuando el Hardware se esté usando de manera eficiente, en caso contrario el Hardware será desaprovechado.

Fracción serial Se define como el cociente del tiempo que se tarda en completar el cómputo de la parte secuencial de una tarea entre el tiempo que se tarda en completar el cómputo de la tarea usando un solo procesador:

$$f = \frac{T_s}{T(1)} \quad (8.3)$$

pero usando la ley de Amdahl:

$$T(p) = T_s + \frac{T_p}{p}$$

y reescribiéndola en términos de factor de aceleración, obtenemos la forma operativa del cálculo de la fracción serial que adquiere la forma siguiente:

$$f = \frac{\frac{1}{s} - \frac{1}{p}}{1 - \frac{1}{p}}. \quad (8.4)$$

Esta métrica permite ver las inconsistencias en el balance de cargas, ya que su valor deberá de tender a cero en el caso ideal, por ello un incremento en el valor de f es un aviso de granularidad fina con la correspondiente sobrecarga en los procesos de comunicación.

Costo o Trabajo Se define el costo o trabajo de resolver un problema en paralelo como el producto del tiempo de cálculo en paralelo T_p por el número de procesadores usando p y se representa por:

$$C_p = p * T_p.$$

Aceleración y Eficiencia Relativa Cuando se trabaja en más de un equipo paralelo -supongamos con p y p' procesadores con $p \geq p'$ - es común comparar el desempeño entre dichos equipos. Para ello se define la aceleración relativa como:

$$S_p^{p'} = \frac{T_p}{T_{p'}}$$

para $p \geq p'$, en la cual se espera que:

$$S_p^{p'} \simeq \frac{p}{p'}$$

y eficiencia relativa como:

$$E_p^{p'} = \frac{p'}{p} S_p^{p'} = \frac{p'}{p} \frac{T_p}{T_{p'}}.$$

Análisis de Rendimiento Usando Métricas Suponiendo un Cluster con 17 Cores o núcleos⁷⁹, se muestra una ejemplificación de las métricas para un problema de ejemplo:

Cores	Tiempo	Aceleración	Eficiencia	Frac. Ser.
1	5295			
3	2538	2.08	0.69	0.218
5	1391	3.80	0.76	0.078
9	804	6.58	0.73	0.045
17	441	12.00	0.70	0.025

Nótese que en todos los casos la fracción serial disminuye sustancialmente con el aumento del número de procesadores, pero la aceleración está por debajo del valor esperado.

Suponiendo un Cluster A con 100 Cores y un Cluster B con 128 Cores, se muestra una ejemplificación de las métricas para un problema de ejemplo en ambos Clusters con los siguientes resultados:

Cores	Cluster A	Cluster B
16	9158 seg	ND
32	5178 seg	5937 seg
64	3647 seg	4326 seg
100	2661 seg	
128		2818 seg

Como se muestra en la tabla, en todos los casos el Cluster A usando como máximo 100 Cores obtiene un tiempo de cálculo inferior al que requiere Cluster B usando a lo más los 128 Cores.

Haciendo uso de las métricas de aceleración y eficiencia relativa⁸⁰ se tiene que para el Cluster B:

$$S_{128}^{32} = 5937/2818 = 2.10$$

⁷⁹A cada procesador encapsulado se le llama Core o núcleo, logrando que la comunicación entre ellos se realiza de una forma más rápida a través de un bus interno integrado en la propia pastilla de silicio sin tener que recurrir por tanto al bus del sistema mucho más lento. Al contrario del caso de la tecnología HyperThreading que el sistema operativo los reconoce como un core en el equipo, pero estos cores virtuales distan mucho del desempeño de uno real.

⁸⁰Aceleración relativa es $S_p^{p'} = \frac{T_p}{T_{p'}}$ para $p \geq p'$, en la cual se espera que $S_p^{p'} \simeq \frac{p}{p'}$ y eficiencia relativa es $E_p^{p'} = \frac{p'}{p} S_p^{p'} = \frac{p'}{p} \frac{T_p}{T_{p'}}$.

donde lo esperado sería:

$$S_{128}^{32} = 32/128 = 4.00,$$

para el caso de la eficiencia:

$$E_{128}^{32} = (32/128) * (5937/2818) = 0.52.$$

En el caso del Cluster A se tiene que:

$$S_{100}^{16} = 9158/2661 = 3.44$$

dónde lo esperado sería:

$$S_{100}^{16} = 16/100 = 6.35,$$

para el caso de la eficiencia:

$$E_{100}^{16} = (16/100) * (9158/2661) = 0.55.$$

Haciendo uso del mismo número de Cores base para el Cluster A que para Cluster B, se tiene que:

$$S_{100}^{32} = 5178/2661 = 1.94$$

dónde lo esperado sería:

$$S_{100}^{16} = 32/100 = 3.12,$$

para el caso de la eficiencia:

$$E_{100}^{16} = (32/100) * (5178/2661) = 0.62.$$

De todo lo anterior, se desprende que el Cluster A obtiene valores de una aceleración y eficiencias relativas ligeramente mejores que el Cluster B, pero esto no se refleja en la disminución de casi 6% del tiempo de ejecución y del uso de 28 Cores menos.

Además, el costo computacional:

$$C_p = P * T_p,$$

que para el caso del Cluster B es:

$$C_{128} = 360,704$$

y en Cluster A es:

$$C_{100} = 266,100$$

que representa una disminución de 27%; además de un factor muy importante, el Cluster A tuvo un costo monetario mucho menor con respecto del Cluster B.

8.6 Programación de Cómputo de Alto Rendimiento

Hay muchas aplicaciones a las herramientas computacionales, pero nos interesan aquellas que permitan resolver problemas concomitantes en Ciencia e Ingeniería. Muchas de estas aplicaciones caen en lo que comúnmente se llama cómputo científico. La computación científica es el campo de estudio relacionado con la construcción de modelos matemáticos, técnicas numéricas para resolver problemas científicos y de ingeniería; y su respectiva implementación computacional. La solución de estos problemas genera un alto consumo de memoria, espacio de almacenamiento o tiempo de cómputo; por ello nos interesa trabajar en computadoras que nos puedan satisfacer estas demandas.

La computación de alto rendimiento (véase 8) -High performance Computing o HPC en inglés- es la agregación de potencia de cálculo para resolver problemas complejos en Ciencia e Ingeniería o Gestión. Para lograr este objetivo, la computación de alto rendimiento se apoya en tecnologías computacionales como los Clusters, las supercomputadoras o la computación paralela. La mayoría de las ideas actuales de la computación distribuida se han basado en la computación de alto rendimiento.

La computación paralela o de alto rendimiento es una forma de cómputo en la que muchas instrucciones se ejecutan simultáneamente, operando sobre el principio de que problemas grandes, a menudo se pueden dividir en unos más pequeños, que luego son resueltos simultáneamente (en paralelo). Hay varias formas diferentes de computación paralela: paralelismo a nivel de bits, paralelismo a nivel de instrucción, paralelismo de datos y paralelismo de tareas.

El paralelismo se ha empleado durante muchos años, sobre todo en la computación de altas prestaciones, pero el interés en ella ha crecido últimamente debido a las limitaciones físicas que impiden el aumento de la frecuencia de los actuales procesadores comerciales. Como el consumo de energía –y por consiguiente la generación de calor– de las computadoras constituye una preocupación en los últimos años, la computación en paralelo se ha convertido en el paradigma dominante en la arquitectura de computadoras, principalmente en forma de procesadores multinúcleo.

Las computadoras paralelas pueden clasificarse según el nivel de paralelismo que admite su Hardware:

- Equipos con procesadores multinúcleo

- Equipos con multiprocesador (múltiples procesadores multinúcleo en una sola máquina)
- Procesadores masivamente paralelos
- Equipos con una o más tarjetas CUDA (Compute Unified Device Architecture)
- Cluster de equipos multinúcleo, multiprocesador y/o CUDAS
- Cúmulos de Clusters (Grids)

Muchas veces, para acelerar tareas específicas, se utilizan arquitecturas especializadas de computación en paralelo junto a procesadores tradicionales.

Existen múltiples vertientes en el cómputo en paralelo, algunas de ellas son:

- Cómputo en memoria compartida usando OpenMP
- Cómputo en memoria distribuida usando MPI

Como es de esperarse, los programas informáticos paralelos son más complejos y difíciles de escribir que los secuenciales, porque la concurrencia introduce nuevos tipos de errores de Software -por ello existe una creciente gama de herramientas que coadyuvan a mejorar la escritura, depuración y desempeño de los programas en paralelo-, pero la comunicación y sincronización entre diferentes subtareas son algunos de los mayores obstáculos para obtener un buen rendimiento del programa paralelo.

Además, el tiempo de programación necesario para desarrollar una aplicación paralela eficiente y eficaz para la gran mayoría de los programadores puede ser de semanas o meses en el mejor de los casos. Por ello, es necesario hacer un balance entre las diferentes opciones de paralelización para no invertir un tiempo precioso que puede no justificar dicha inversión económica, de recursos computacionales y sobre todo de tiempo.

Actualmente, en muchos centros de cómputo es una práctica común usar directivas de compilación en equipos paralelos sobre programas escritos de forma secuencial, con la esperanza que sean puestos por el compilador como programas paralelos. Esto en la gran mayoría de los casos genera códigos poco eficientes, pese a que corren en equipos paralelos y pueden usar toda

la memoria compartida de dichos equipos, el algoritmo ejecutado continúa siendo secuencial en una gran parte del código.

Si la arquitectura paralela donde se implemente el programa es UMA de acceso simétrico, los datos serán accesados a una velocidad de memoria constante. En caso contrario, al acceder a un conjunto de datos es común que una parte de estos sean locales a un procesador (con un acceso del orden de nanosegundos), pero el resto de los datos deberán ser accesados mediante red (con acceso del orden de milisegundos), siendo esto muy costoso en tiempo de procesamiento.

Por lo anterior, si se cuenta con computadoras con memoria compartida o que tengan interconexión por bus, salvo en casos particulares no será posible explotar éstas características eficientemente. Pero en la medida en que se adecuen los programas para usar bibliotecas y compiladores acordes a las características del equipo disponible -algunos de ellos sólo existen de manera comercial- la eficiencia aumentará de manera importante.

8.6.1 Programando con OpenMP para Memoria Compartida

OpenMP es una interfaz de programación de aplicaciones (API) para la programación multiproceso de memoria compartida en múltiples plataformas. Permite añadir concurrencia a los programas escritos en C, C++, Julia, Python y Fortran sobre la base del modelo de ejecución Fork-join. Está disponible en muchas arquitecturas, incluidas las plataformas de Unix, Linux y de Microsoft Windows. Se compone de un conjunto de directivas de compilador, rutinas de biblioteca, y variables de entorno que influyen en el comportamiento en tiempo de ejecución.

Definido conjuntamente por proveedores de Hardware y de Software, OpenMP es un modelo de programación portable y escalable que proporciona a los programadores una interfaz simple y flexible para el desarrollo de aplicaciones paralelas, para plataformas que van desde las computadoras de escritorio hasta supercomputadoras. Una aplicación construida con un modelo de programación paralela híbrido se puede ejecutar en un Cluster de computadoras utilizando OpenMP y MPI, o a través de las extensiones de OpenMP para los sistemas de memoria distribuida.

OpenMP se basa en el modelo *Fork-join*, paradigma que proviene de los sistemas Unix, donde una tarea muy pesada se divide en K hilos (Fork) con menor peso, para luego «recolectar» sus resultados al final y unirlos en un solo resultado (Join).

Cuando se incluye una directiva de compilador OpenMP esto implica que se incluye una sincronización obligatoria en todo el bloque. Es decir, el bloque de código se marcará como paralelo y se lanzarán hilos según las características que nos dé la directiva, y al final de ella habrá una barrera para la sincronización de los diferentes hilos (salvo que implícitamente se indique lo contrario con la directiva `Nowait`). Este tipo de ejecución se denomina *Fork-join*.

OpenMP también soporta el modelo de paralelismo de tareas. El equipo de hilos del bloque de código paralelo ejecuta las tareas especificadas dentro de dicho bloque. Las tareas permiten un paralelismo asíncrono. Desde la versión 4.0 lanzada en 2013 admite la especificación de dependencias entre tareas, relegando a la biblioteca de tiempo de ejecución de OpenMP el trabajo de planificar las tareas y ponerlas en ejecución. Los hilos de ejecución irán ejecutando las tareas a medida que estas estén disponibles (sus dependencias ya estén satisfechas). El uso de tareas da lugar a sincronización con una granularidad más fina. El uso de barreras no es estrictamente necesario, de manera que los hilos pueden continuar ejecutando tareas disponibles sin necesidad de esperar a que todo el equipo de hilos acabe un bloque paralelo. El uso de tareas con dependencias crea un grafo, pudiéndose aplicar propiedades de grafos a la hora de escoger tareas para su ejecución.

Salvo el uso de implementaciones de Hardware de la biblioteca de tiempo de ejecución OpenMP (p.ej. en una matriz de puertas programables FPGAs), los sobrecostes de las tareas es mayor, este sobrecoste ha de ser amortizado mediante el potencial paralelismo adicional que las tareas exponen.

Estructura del Programa en C++ Ejemplo de cálculo de Pi usando OpenMP:

```
#include <stdio.h>
// Indica si se carga lo referente a OpenMP
#ifdef _OPENMP
#include <omp.h>
int threads=omp_get_num_threads();
#else
int threads=0;
#endif
#define STEPCOUNTER 1000000000
int main (void)
```

```
{
    long i;
    double pi=0;
    printf("threads %d", threads);
    #pragma omp parallel for reduction(+:pi)
    for (i=0; i < STEPCOUNTER; i++)
    {
        pi += 1.0/(i*4.0 +1.0);
        pi -= 1.0/(i*4.0 +3.0);
    }
    pi = pi*4.0;
    printf("PI = %2.16lf ",pi);
    return 0;
}
```

El compilador de OpenMP es el mismo que para los lenguajes C, C++ o Fortran respectivamente (véase [3.7](#)). Por ello, para usarlo en C++ en línea de comandos, instalamos el compilador g++, mediante:

```
# apt install g++
```

Así, para compilar con g++^{[81](#)}, sin usar OpenMP, usamos:

```
$ g++ pi.cpp -o pi
```

Ejecutar midiendo el tiempo^{[82](#)}:

```
$ time ./pi
```

⁸¹Compilar fuentes en C++ solicitando que el ejecutable tenga el nombre *ejemp*:

```
$ g++ *.cpp -o ejemp
```

en este caso no se usa ninguna directiva para optimizar el ejecutable generado. Para compilar usando diversas optimizaciones (O1, -O2 o -O3) usar por ejemplo:

```
$ g++ -O1 *.cpp
```

⁸²Los sistemas operativos tipo Linux/Unix cuentan con un comando básico que mide el tiempo de ejecución de cualquier comando, usando:

```
$ time ls
```


Ahora, usando el compilar para OpenMP usamos:

```
$ g++ -o pi -fopenmp pi.cpp
```

Indicar el número de hilos, por ejemplo 2:

```
$ export OMP_NUM_THREADS=2
```

Ejecutar:

```
$ time ./pi
```

8.6.2 Programando con MPI para Memoria Distribuida

Para poder intercomunicar dos o más Cores en una o en múltiples computadoras se usa la «interfaz de paso de mensajes (Message Passing Interface MPI)» (véase [67], [68], [71] y [69]), una biblioteca de comunicación para procesamiento en paralelo que puede ser usada desde lenguajes de programación como: C, C++, Pytho, Julia, Fortran. MPI ha sido desarrollado como un estándar para el paso de mensajes y operaciones relacionadas.

Este enfoque es adoptado por usuarios e implementadores de bibliotecas, en el cual se proveen a los programas de procesamiento en paralelo de portabilidad y herramientas necesarias para desarrollar aplicaciones que puedan usar el cómputo paralelo de alto desempeño.

El modelo de paso de mensajes posibilita a un conjunto de procesos que tienen solo memoria local, la comunicación con otros procesos (usando Bus o red) mediante el envío y recepción de mensajes. Por definición el paso de mensajes posibilita transferir datos de la memoria local de un proceso a la memoria local de cualquier otro proceso que lo requiera.

En el modelo de paso de mensajes para equipos paralelos, los procesos se ejecutan en paralelo, teniendo direcciones de memoria separada para cada proceso, la comunicación ocurre cuando una porción de la dirección de memoria de un proceso es copiada mediante el envío de un mensaje dentro de otro proceso en la memoria local mediante la recepción del mismo.

pero podemos instalar el paquete que además de medir el tiempo de ejecución nos diga que recursos se usaron el la ejecución del comando indicado, para instalarlo usamos:

```
# apt install time
```

el uso es el mismo del comando interno time.

Las operaciones de envío y recepción de mensajes es cooperativa y ocurre sólo cuando el primer proceso ejecuta una operación de envío y el segundo proceso ejecuta una operación de recepción, los argumentos base de estas funciones son:

- Para el que envía, la dirección de los datos a transmitir y el proceso destino al cual los datos se enviarán.
- Para el que recibe, debe tener la dirección de memoria donde se pondrán los datos recibidos, junto con la dirección del proceso que los envió.

Es decir:

Send(dir, lg, td, dest, etiq, com)

$\{dir, lg, td\}$ describe cuántas ocurrencias lg de elementos del tipo de dato td se transmitirán empezando en la dirección de memoria dir .

$\{des, etiq, com\}$ describe el identificador $etiq$ de destino des asociado con la comunicación com .

Recv(dir, mlq, td, fuent, etiq, com, st)

$\{dir, lg, td\}$ describe cuántas ocurrencias lg de elementos del tipo de dato td se transmitirán empezando en la dirección de memoria dir .

$\{fuent, etiq, com, est\}$ describe el identificador $etiq$ de la fuente $fuent$ asociado con la comunicación com y el estado est .

El conjunto básico de directivas (en nuestro caso sólo se usan estas) en C++ de MPI son (véase [67] y [68]):

MPI::Init	Inicializa al MPI
MPI::COMM_WORLD.Get_size	Busca el número de procesos existentes
MPI::COMM_WORLD.Get_rank	Busca el identificador del proceso
MPI::COMM_WORLD.Send	Envía un mensaje
MPI::COMM_WORLD.Recv	Recibe un mensaje
MPI::Finalize	Termina al MPI

Estructura del Programa en C++ Ejemplo de Hola_Mundo en MPI:

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[])
{
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf("Hola! Soy el %d de %d\n", rank, size);
    MPI_Finalize();
    return 0;
}
```

Otro ejemplo, para realizar una suma en MPI:

```
#include <iostream>
#include <iomanip>
#include <mpi.h>
using namespace std;
int main(int argc, char ** argv){
    int mynode, totalnodes;
    int sum = 0, startval, endval, accum;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &totalnodes);
    MPI_Comm_rank(MPI_COMM_WORLD, &mynode);
    startval = 1000*mynode/totalnodes+1;
    endval = 1000*(mynode+1)/totalnodes;
    for(int i=startval; i<=endval; i=i+1) sum = sum + i;
    if(mynode!=0)
        MPI_Send(&sum, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
    else
        for(int j=1; j<totalnodes; j=j+1){
            MPI_Recv(&accum, 1, MPI_INT, j, 1,
                MPI_COMM_WORLD, &status);
            sum = sum + accum;
        }
}
```

```
}  
if(mynode == 0)  
cout << "The sum from 1 to 1000 is: " << sum << endl;  
MPI_Finalize();  
}
```

Existe una gran variedad de compiladores de MPI en línea de comandos, algunos disponibles en Debian GNU/Linux son instalados mediante:

```
# apt install lam-runtime libmpich-dev mpi-default-dev mpich\  
mpi-default-bin openmpi-bin valgrind-mpi xmpi
```

Para compilar y ejecutar es posible usar alguna de estas opciones:

```
mpic++, mpic++.openmpi, mpiexec.mpic, mpif90.openmpi,  
mpirun.lam, mpicc, mpicxx, mpiexec.openmpi, mpifort, mpirun.mpic,  
mpiCC, mpicxx.mpic, mpif77, mpifort.mpic, mpirun.openmpi,  
mpicc.mpic, mpicxx.openmpi, mpif77.mpic, mpifort.openmpi,  
mpitask, mpicc.openmpi, mpiexec, mpif77.openmpi, mpimsg, mpi-  
vars, mpiCC.openmpi, mpiexec.hydra, mpif90, mpipython, mpichver-  
sion, mpipython, mpiexec.lam, mpif90.mpic, mpirun
```

Por ejemplo, para compilar *ejemp.cpp* en *mpic++* solicitando que el ejecutable tenga el nombre *ejemp*, usamos:

```
$ mpic++ ejemp.cpp -o ejemp
```

en este caso no se uso ninguna opción de optimización en tiempo de compilación (véase 3.7), se puede hacer uso de ellas (-O1, -O2 o -O3), mediante:

```
$ mpic++ -O3 ejemp.cpp -o ejemp
```

para ejecutar el programa ya compilado en 4 procesos y medir el tiempo de ejecución (véase 3.7), usamos:

```
$ time mpirun -np 4 ejemp
```

También podemos compilar *ejemp.c* en *mpicc* solicitando que el ejecutable tenga el nombre *ejemp*:

```
$ mpicc ejemp.cpp -o ejemp
```

en este caso no se uso ninguna opción de optimización en tiempo de compilación, se puede hacer uso de ellas (-O1, -O2 o -O3), mediante:

```
$ mpicc -O3 ejemp.c -o ejemp
```

para ejecutar el programa ya compilado en 4 procesos, usamos:

```
$ mpirun -np 4 ejemp
```

Un último ejemplo, en el caso de usar *mpiCC.mpic* y *lamboot*, entoces es necesario compilar usando:

```
$ mpiCC.mpic -O2 ejemp.cpp -o ejemp -lm
```

iniciar ambiente de ejecución paralelo:

```
$ lamboot -v
```

correr usando 8 procesos por ejemplo:

```
$ mpirun.mpic -np 8 ejemp
```

correr usando 5 procesos segun lista *machines.lolb* por ejemplo:

```
$ mpirun.mpic -machinefile machines.lolb -np 5 ejemp
```

terminar ambiente de ejecución paralelo:

```
$ lamhalt -v
```

Observación 1 *Para que en la ejecución de MPI no pida la clave de usuario:*

```
$ ssh-keygen -t rsa
```

En cada pregunta responder con ENTER, para después copiar usando:

```
$ ssh-copy-id usuario@servidor
```

Ojo: Si continúa pidiendo clave, es que esta instalado rsh o lsh.

8.6.3 Esquema de Paralelización Principal-Subordinados

El esquema de paralelización Principal-Subordinados -también conocido como Maestro-Esclavo-, permite sincronizar por parte del nodo principal las tareas que se realizan en paralelo usando varios nodos subordinados, éste modelo puede ser explotado de manera eficiente si existe poca comunicación entre el principal y el subordinado -entre los subordinados no debe de existir comunicación- y los tiempos consumidos en realizar las tareas asignadas son mayores que los períodos involucrados en las comunicaciones para la asignación de dichas tareas. De esta manera se garantiza que la mayoría de los procesadores estarán trabajando de manera continua y existirán pocos tiempos muertos.

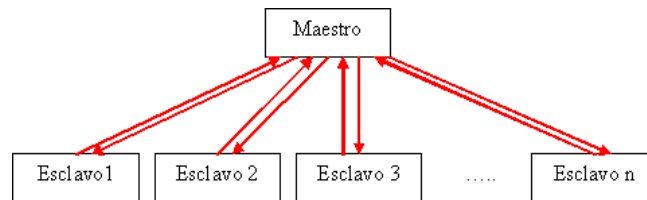


Figura 11: Esquema del Maestro-Esclavo

Donde, tomando en cuenta la implementación en estrella del Cluster, el modelo de paralelismo de MPI (véase 8.6.2) y las necesidades propias de comunicación del programa, el nodo principal tendrá comunicación sólo con cada nodo subordinado y no existirá comunicación entre los nodos subordinados, esto reducirá las comunicaciones y optimizará el paso de mensajes, algunos ejemplos de este esquema se pueden consultar en:

Herramientas

<http://132.248.181.216/Herramientas/>

Un factor limitante en este esquema es que el nodo principal deberá de atender todas las peticiones hechas por cada uno de los nodos subordinados, esto toma especial relevancia cuando todos o casi todos los nodos subordinados compiten por ser atendidos por el nodo principal.

Se recomienda implementar este esquema en un Cluster heterogéneo en donde el nodo principal sea más poderoso computacionalmente que los nodos subordinados. Si a este esquema se le agrega una red de alta velocidad y de

baja latencia, se le permitirá operar al Cluster en las mejores condiciones posibles, pero este esquema se verá degradado al aumentar el número de nodos subordinados inexorablemente.

Pero hay que ser cuidadosos en cuanto al número de nodos subordinados que se usan en la implementación en tiempo de ejecución versus el rendimiento general del sistema al aumentar estos, algunas observaciones posibles son:

- El esquema Principal-Subordinados programado usando MPI lanza P procesos (uno para el nodo principal y $P - 1$ para los nodos subordinados), estos en principio corren en un solo procesador pero pueden ser lanzados en múltiples procesadores usando una directiva de ejecución, de esta manera es posible que en una sola máquina se programe, depure y sea puesto a punto el código usando mallas pequeñas (del orden de cientos de nodos) y cuando este listo puede mandarse a producción en un Cluster.
- El esquema Principal-Subordinados no es eficiente si sólo se usan dos procesadores (uno para el nodo principal y otro para el nodo subordinado), ya que el nodo principal en general no realiza los cálculos pesados y su principal función será la de distribuir tareas; los cálculos serán delegados al nodo subordinado.

Estructura del Programa Principal-Subordinados La estructura del programa se realizó para que el nodo principal mande trabajos de manera síncrona a los nodos subordinados. Cuando los nodos subordinados terminan la tarea asignada, avisan al nodo principal para que se les asigne otra tarea (estas tareas son acordes a la etapa correspondiente del método de descomposición de dominio ejecutándose en un instante dado). En la medida de lo posible se trata de mandar paquetes de datos a cada nodo subordinado y que estos regresen también paquetes al nodo principal, a manera de reducir las comunicaciones al mínimo y tratar de mantener siempre ocupados a los nodos subordinados para evitar los tiempos muertos, logrando con ello una granularidad gruesa, ideal para trabajar con Clusters.

La estructura básica del programa bajo el esquema Principal-Subordinados codificada en C++ y usando MPI (véase 8.6.2) es:

```
main(int argc, char *argv[])
```

```
{
    MPI::Init(argc,argv);
    ME_id = MPI::COMM_WORLD.Get_rank();
    MP_np = MPI::COMM_WORLD.Get_size();
    if (ME_id == 0) {
        // Operaciones del Principal
    } else {
        // Operaciones del Subordinado con identificador ME_id
    }
    MPI::Finalize();
}
```

En este único programa se deberá de codificar todas las tareas necesarias para el nodo principal y cada uno de los nodos subordinados, así como las formas de intercomunicación entre ellos usando como distintivo de los distintos procesos a la variable *ME_id*. Para más detalles de esta forma de programación y otras funciones de MPI (véase 8.6.2, [67] y [68]).

El principal factor limitante para el esquema Principal-Subordinados es que se presupone contar con un nodo principal lo suficientemente poderoso para atender simultáneamente las tareas síncronas del método, ya que este distribuye tareas acorde al número de nodos subordinados, estas si son balanceadas, ocasionaran que muchos de los procesadores subordinados terminen aproximadamente al mismo tiempo y el nodo principal tendrá que atender múltiples comunicaciones simultáneamente, degradando su rendimiento al aumentar el número de nodos subordinados que atender.

Para los factores limitantes inherente al propio esquema Principal-Subordinados, es posible implementar algunas operaciones del nodo principal en paralelo, ya sea usando equipos multiprocesador o en más de un nodo distinto a los nodos subordinados.

8.6.4 Opciones de Paralelización Híbridas

En la actualidad, casi todos los equipos de cómputo usados en estaciones de trabajo y Clusters cuentan con dos o más Cores, en ellos siempre es posible usar MPI para intercambiar mensajes entre procesos corriendo en

el mismo equipo de cómputo, pero no es un proceso tan eficiente como se puede querer. En estas arquitecturas llamadas de memoria compartida es mejor usar OpenMP o cualquiera de sus variantes para trabajar en paralelo. Por otro lado es común contar con las cada vez más omnipresentes tarjetas NVIDIA, y con los cada vez más numerosos Cores CUDA -que una sola tarjeta NVIDIA TESLA puede tener del orden de miles de ellos- y que en un futuro serán cada vez más numerosos.

Para lograr obtener la mayor eficiencia posible de estos tres niveles de paralelización, se están implementando procesos híbridos (véase [72] y [70]), en donde la intercomunicación de equipos con memoria compartida se realiza mediante MPI y la intercomunicación entre Cores que comparten la misma memoria se realiza con OpenMP, además las operaciones matriciales, vectoriales, etc. se le encargan a los numerosos Cores CUDA de las tarjetas NVIDIA.

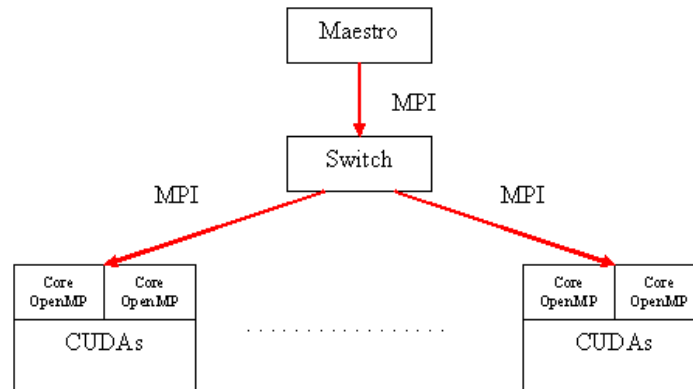


Figura 12: Paralelización Híbrida

Se han desarrollado múltiples programas que sus algoritmos resuelven ecuaciones diferenciales parciales concomitantes en Ciencias e Ingenierías que hacen uso de esta forma integradora de paralelismo. Para ello, la interconexión de equipos de memoria compartida se realizaría mediante MPI y en cada equipo de memoria compartida se manipularían uno o más subdominios mediante OpenMP -ya que cada subdominio es independiente de los demás- y la manipulación de matrices y operaciones entre matrices y vectores que requiere cada subdominio se realizarían en las tarjetas NVIDIA mediante los numerosos Cores CUDA sin salir a la RAM de la computadora.

Permitiendo así, tener una creciente eficiencia de paralelización que optimizan en gran medida los recursos computacionales, ya que todas las matrices y vectores se generarían en la GRAM de la tarjeta NVIDIA. De forma tal que sea reutilizable y que pueda usarse en problemas en los que el número de grados de libertad sea grande, permitiendo hacer uso de equipos de cómputo cada vez más asequibles y de menor costo, pero con una creciente eficiencia computacional que compiten con los grandes equipos de cómputo de alto desempeño.

8.7 Programando Desde la Nube

Existen diferentes servicios Web⁸³ que permiten editar, compilar y ejecutar código de diversos lenguajes y paquetes desde el navegador, esto en aras de que los estudiantes y profesores que cuenten con algún sistema de acceso a red y un navegador puedan programar en los más diversos lenguajes, IDEs y terminales sin hacer instalación alguna en su equipo de cómputo, tableta o teléfono celular.

Google Colaboratory Integrante de la G Suite for Education de Google permite a los usuarios que pertenezcan a esta Suite (como gran parte de los estudiantes de la UNAM) tener acceso desde el navegador para escribir y ejecutar código de Python (Jupyter), es posible elegir correr nuestro Notebook en una CPU, GPU o en una TPU de forma gratuita. Tiene algunas restricciones, como por ejemplo que una sesión dura 12 hrs, pasado ese tiempo se limpia nuestro ambiente y perdemos las variables y archivos que tengamos almacenados allí.

Es conveniente para principiantes que requieran experimentar con Machine Learning y Deep Learning pero sin recurrir en costos de procesamiento Cloud. Además el ambiente de trabajo ya viene con muchas librerías instaladas y listas para utilizar (como por ejemplo *Tensorflow*, *Scikit-learn*, *Pytorch*, *Keras* y *OpenCV*), ahorrándonos el trabajo de configurar nuestro ambiente

⁸³Cuando se trabaja desde la Web es recomendable usar el modo Privado o Incógnito para no guardar el historial de navegación, información introducida en los formularios y borrar al cerrar el navegador los datos de los sitios visitados. Pero recuerda que los sitios Web que visitamos sí guardan información de nuestra visita, nuestro proveedor de Internet también guarda constancia de nuestra visita y si descargamos algo, esto no se borra al igual que el historial de descargas, además de las marcas de páginas o favoritos se conservarán al cerrar el navegador.

de trabajo. Podemos importar nuestros archivos y datos desde *Google Drive*, *GitHub*, etc.

Más información sobre Google Colaboratory en:

<https://colab.research.google.com/notebooks/intro.ipynb>

9 Ejemplitos

Hay muchas aplicaciones a las herramientas computacionales, pero nos interesan aquellas que permitan resolver problemas concomitantes en Ciencia e Ingeniería. Muchas de estas aplicaciones caen en lo que comúnmente se llama cómputo científico. La computación científica es el campo de estudio relacionado con la construcción de modelos matemáticos, técnicas numéricas para resolver problemas científicos y de ingeniería; y su respectiva implementación computacional.

Este campo es distinto a las ciencias de la computación y el procesamiento de información, también es diferente a la teoría y experimentación, que son las formas tradicionales de la ciencia y la ingeniería. El enfoque de la computación científica es para ganar entendimiento, principalmente a través del análisis de modelos matemáticos implementados en computadoras.

Los programas de aplicación de la computación científica a menudo modelan cambios en las condiciones del mundo real, tales como el tiempo atmosférico, el flujo de aire alrededor de un avión, el movimiento de las estrellas en una galaxia, el comportamiento de un dispositivo explosivo, entre otros. Estos programas deberían crear una 'malla lógica' en la memoria de la computadora, donde cada ítem corresponda a un área en el espacio y contenga información acerca del espacio relevante para el modelo. Por ejemplo, en modelos para el tiempo atmosférico, cada ítem podría ser un kilómetro cuadrado, con la altitud del suelo, dirección actual del viento, humedad ambiental, temperatura, presión, etc. El programa debería calcular el probable siguiente estado basado en el estado actual, simulado en medidas de tiempo, resolviendo ecuaciones que describen cómo operan los sistemas; y repetir el proceso para calcular el siguiente estado.

El término científico computacional es usado para describir a alguien experto en computación científica. Esta persona es generalmente un científico, un ingeniero o un matemático aplicado que aplica computación de alto rendimiento en diferentes formas para avanzar en el estado del arte de su respectiva disciplina de la física, química o ingeniería. Científicos computacionales han impactado cada vez más en otras áreas como la economía, biología y medicina. La computación científica es hoy en día considerada como el tercer modo de ciencia, complementando y añadiendo a la experimentación/observación y teoría.

La computación científica es más estudiada por medio de la matemática aplicada o programas de las ciencias de la computación, o dentro de un

estándar matemático, ciencias, o programas de ingeniería. En algunas instituciones una especialización en computación científica puede ser obtenida como un "Minor" dentro de otro programa (el cual puede estar en muchos niveles). Sin embargo, hay cada vez más muchas maestrías y programas de doctorado en computación científica. Algunas universidades también ofrecen doctorados en ciencias computacionales, ingeniería computacional, ciencias computacionales e ingeniería o computación científica, también hay programas como el de física computacional, química computacional, entre otros.

Científicos e Ingenieros desarrollan Software y aplicaciones informáticas para modelar sistemas que estan siendo estudiados, para correr estos programas con diferentes conjuntos de entradas. Por lo general, estos modelos requieren una gran cantidad de cálculos (usualmente de punto flotante) y en general para problemas de interés, a menudo son ejecutados en supercomputadoras o plataformas de computación distribuida.

Simulaciones Numéricas Las simulaciones numéricas tienen diferentes objetivos dependiendo de la naturaleza de la tarea a ser simulada:

- Reconstruir y comprender los eventos conocidos (e.g. terremotos, maremotos y otros desastres naturales)
- Predecir el futuro o situaciones no observadas (e.g. tiempo atmosférico, comportamiento de partículas subatómicas)

Por Medio de Modelos Apropriados y Análisis de Datos se Busca

- Sintonizar apropiadamente los modelos o resolver ecuaciones para reflejar ciertas observaciones, sujetas a las restricciones del modelo (e.g. exploración geofísica de petróleo, lingüística computacional).
- Usar teoría de grafos para modelar redes, especialmente las conexiones individuales, organizaciones y sitios Web.

Optimización Se busca optimizar escenarios conocidos (e.g. técnicas y procesos de fabricación, interfaces de ingeniería).

Los algoritmos y métodos matemáticos usados en computación científica son variados. Los métodos comúnmente aplicados son:

- Análisis numérico
- Aplicación de series de Taylor como series convergentes y asintóticas
- Cálculo de derivadas mediante diferenciación automática
- Cálculo de derivadas mediante diferencias finitas, elemento finito, volumen finito
- Métodos aproximados de diferencias de alto orden mediante series de Taylor y la extrapolación de Richardson
- Métodos de integración sobre una malla uniforme: regla del rectángulo, regla del trapecio, regla del punto medio, regla de Simpson
- Método para resolver ecuaciones diferenciales ordinarias
- Método de Montecarlo
- Álgebra lineal numérica
- Cálculo de los factores LU vía eliminación Gaussiana
- Factorización de Cholesky
- Transformada de Fourier discreta y sus aplicaciones
- Método de Newton
- Métodos de time-stepping para sistemas dinámicos

Los lenguajes de programación comúnmente usados para los aspectos más matemáticos de las aplicaciones de la computación científica incluyen a Fortran, MATLAB, Scilab, GNU Octave, COMSOL Multiphysics y PDL. Los aspectos más computacionales son tratados a menudo con C y Fortran, aunque existen otros desarrollos importantes en lenguajes ahora poco conocidos como PL/1. La implementación computacional puede ser implementada usando uno o más tipos de programación como son: programación secuencial, estructurada, funcional y orientada a objetos; tanto en ejecución secuencial, paralela o distribuida.

9.1 Precisión Arbitraria

Serie de Fibonacci supongamos que nos piden sumar los primeros 100 términos de la serie de Fibonacci (un problema en principio trivial para una computadora como las actuales), sin pérdida de generalidad haremos el programa en Java (pero el mismo error aparece en los lenguajes C, C++, Fortran, entre otros). Si la declaración de datos usamos ingenuamente digamos el tipo *int*, aquí el programa:

```
public class FibonacciInt {
    public static void main(String args[]) {
        int i, n = 100;
        int r = 1, a = 0, b = 1, t = 0;
        System.out.println("val 1 ==> " + 0);
        System.out.println("val 2 ==> " + 1);
        for (i = 3; i <= n; i++) {
            t = a + b;
            System.out.println("val " + i + " ==> " + t);
            a = b;
            b = t;
            r += t;
        }
        System.out.println("Suma de la serie " + r);
    }
}
```

Al ejecutar el programa, observamos que:

```
val 47 ==> 1836311903
val 48 ==> -1323752223
```

Tenemos un error por desbordamiento. Si ahora cambiamos la línea:

```
int r = 1, a = 0, b = 1, t = 0;
```

por:

```
long r = 1, a = 0, b = 1, t = 0;
```

Al ejecutar el programa, observamos que:

```
val 93 ==> 7540113804746346429
val 94 ==> -6246583658587674878
```

Otra vez tenemos un error por desbordamiento. Si ahora hacemos el cambio por *double*, al ejecutar el programa, observamos que:

```
val 79 ==> 8.944394323791464E15
val 80 ==> 1.447233402467622E16
val 81 ==> 2.3416728348467684E16
```

existe discrepancia con los valores reales a partir del término 80:

```
val 79 ==> 8944394323791464
val 80 ==> 14472334024676221
val 81 ==> 23416728348467685
```

Ya utilice todos los tipos de datos que me da el lenguaje de programación, entonces ¿cómo soluciono el problema?.

Una opción es cambiar de estrategia al tratar de hacer el cálculo y ahora de lugar de usar tipos de datos primitivos, usaremos arreglos para almacenar los diferentes dígitos involucrados en los respectivos cálculos del problema, aquí el programa:

```
public class Fibonacci {
    public static void main(String args[]) {
        int i, n = 100;
        int TAM = 25; // Tamaño maximo del arreglo
        // Arreglos de trabajo
        int []A = new int[TAM];
        int []B = new int[TAM];
        int []C = new int[TAM];
        // Inicializacion de los valores de la serie
        A[0] = 0;
        B[0] = 1;
        System.out.println("\n\nSerie de fibonacci con aritmet-
ica de precision arbitraria, para n = " + n);
        for (int k = 3; k <= n; k++) {
            // Pongo en cero el resultante
            for (i = 0; i < TAM; i++) C[i] = 0;
```



```

        // Sumo A + B y guardo el resultado en C
        for (i = 0; i < TAM; i++) {
            C[i] += A[i] + B[i];
            if (C[i] >= 10) {
                C[i] -= 10;
                C[i + 1] += 1;
            }
        }
        // Visualizo
        System.out.print("val " + k + " ==> ");
        for (i = TAM - 1; i >= 0 ; i--) System.out.print(C[i]);
        System.out.println("");
        // Intercambio valores para continuar con el siguiente
termino de la serie
        for (i = 0; i < TAM; i++) {
            A[i] = B[i];
            B[i] = C[i];
        }
    }
}

```

De esta forma ya puedo calcular correctamente todos los términos de la serie, por ejemplo los últimos términos de la serie son:

```

val 97 ==> 0000051680708854858323072
val 98 ==> 0000083621143489848422977
val 99 ==> 0000135301852344706746049
val 100 ==> 0000218922995834555169026

```

que coinciden con los calculados:

```

val 97 ==> 51680708854858323072
val 98 ==> 83621143489848422977
val 99 ==> 135301852344706746049
val 100 ==> 218922995834555169026

```

Pero aún puedo hacer otra cosa, cambiar de lenguaje de programación, por ejemplo a Python. Usando el mismo programa:

```
# Suma los primeros n terminos de la serie de Fibonacci
def Fibonacci(n):
    """Suma los primeros n terminos de la serie de Fibonacci"""
    r = 1
    a = 0
    b = 1
    t = 0
    print("val 1 ==> 0")
    print("val 2 ==> 1")
    for i in range(3,n+1):
        t = a + b
        print("val " + str(i) + " ==> " + str(t))
        a = b
        b = t
        r += t
    print("Suma de la serie " + str(r))
# Serie de Fibonacci
if __name__ == '__main__':
    Fibonacci(300)
```

Ahora no sólo puedo calcular los primeros 100 términos, por ejemplo me permite obtener sin problema los primeros 300 términos y su suma:

```
val 298 ==> 52461916524905785334311649958648296484733611329035169538240802
val 299 ==> 84885164052257330097714121751630835360966663883732297726369399
val 300 ==> 137347080577163115432025771710279131845700275212767467264610201
Suma de la serie 359579325206583560961765665172189099052367214309267232255589800
```

Factorial de un Número si ahora intentamos calcular el factorial de algunos números, por ejemplo el de :10, 20, ...,100. Podemos utilizar un programa como:

```
#include<stdio.h>
int main() {
    int i, num = 41;
    int fi=1;
    for (i = 1; i <= num; i++) {
        fi *= i;
        if (i % 10 == 0) printf("\n%d! %d",i, fi);
    }
```

```
    }  
    long fl=1;  
    for (i = 1; i <= num; i++) {  
        fl *= (long) i;  
        if (i % 10 == 0) printf("\n%d!  %d",i, fl);  
    }  
    float ff=1.0;  
    for (i = 1; i <= num; i++) {  
        ff *= (float) i;  
        if (i % 10 == 0) printf("\n%d!  %f",i, ff);  
    }  
    double fd=1.0;  
    for (i = 1; i <= num; i++) {  
        fd *= (double) i;  
        if (i % 10 == 0) printf("\n%d!  %f",i, fd);  
    }  
    long double fld=1.0;  
    for (i = 1; i <= num; i++) {  
        fld *= (long double) i;  
        if (i % 10 == 0) printf("\n%d!  %LF",i, fld);  
    }  
}
```

Para probar todos los tipos de datos. En esta ocasión los resultados son:

int:

```
10! 3628800  
20! -2102132736
```

long

```
10! 3628800  
20! -2102132736
```

float

```
10! 3628800.000000  
20! 2432902023163674624.000000
```

double

```
10! 3628800.000000
20! 2432902008176640000.000000
30 265252859812191032188804700045312.000000
```

long double

```
10! 3628800.000000
20! 2432902008176640000.000000
30! 265252859812191058647452510846976.000000
```

Si comparamos con los valores reales (están más abajo en esta sección), vemos que todos están erróneos en el último valor calculado que se muestra.

Para resolver el problema (en lenguajes como C, C++, Java o Fortran), podemos hacer un programa análogo al usado para la serie de Fibonacci en el que no usamos tipos de datos primitivos y usamos arreglos para almacenar los diferentes dígitos involucrados en los respectivos cálculos del factorial, en donde ahora las multiplicaciones deberán ser hechas usando digamos sumas sucesivas, el programa sería algo más complejo, pero resolverá el problema.

Por otro lado, si usamos otro lenguaje de programación, por ejemplo Python, podemos usar este código:

```
N = 100
f = 1
for i in range(1, N + 1):
    f *= i
    if i % 10 == 0:
        print(i, f)
```

y obtendremos estos resultados:

```
10! 3628800
20! 2432902008176640000
30! 265252859812191058636308480000000
40! 8159152832478977343456112695961158942720000000000
```

50! 30414093201713378043612608166064768844377641568960512000000000000
60! 83209871127413901442763411832233643807541726063612459524492776964
096000000000000000
70! 11978571669969891796072783721689098736458938142546425857555362864
628009582789845319680000000000000000
80! 71569457046263802294811533723186532165584657342365752577109445058
227039255480148842668944867280814080000000000000000000
90! 14857159644817614973095227336208257378855699612846887669422168637
04985393094065876545992131370884059645617234469978112000000000000000
000000
100! 9332621544394415268169923885626670049071596826438162146859296389
52175999932299156089414639761565182862536979208272237582511852109168
64000000000000000000000000000000

los cuales son correctos en todos sus dígitos⁸⁴.

⁸⁴¿Cuántos dígitos tiene el factorial 100? 158
¿Cuántos ceros hay al final del factorial de 100? 24

9.2 Sistemas Dinámicos Discretos

Un sistema dinámico es un sistema cuyo estado evoluciona con el tiempo. Los sistemas físicos en situación no estacionaria son ejemplos de sistemas dinámicos, pero también existen modelos económicos, matemáticos y de otros tipos que son sistemas abstractos que son, además, sistemas dinámicos. El comportamiento en dicho estado se puede caracterizar determinando los límites del sistema, los elementos y sus relaciones; de esta forma se pueden elaborar modelos que buscan representar la estructura del mismo sistema.

Al definir los límites del sistema se hace, en primer lugar, una selección de aquellos componentes que contribuyan a generar los modos de comportamiento, y luego se determina el espacio donde se llevará a cabo el estudio, omitiendo toda clase de aspectos irrelevantes.

Tipos de Sistemas Dinámicos Los sistemas dinámicos se dividen en sistemas discretos en el tiempo y continuos en el tiempo. Un sistema dinámico se dice discreto si el tiempo se mide en pequeños lapsos; estos son modelados como relaciones recursivas, tal como la ecuación Logística:

$$x_{n+1} = rx_n(1 - x_n)$$

donde n denota los pasos discretos del tiempo y x es la variable que cambia con éste.

Si el tiempo es medido en forma continua, el sistema dinámico continuo resultante es expresado como una ecuación diferencial ordinaria; por ejemplo:

$$\frac{dx}{dt} = rx(1 - x)$$

donde x es la variable que cambia con el tiempo t . La variable cambiante x es normalmente un número real, aunque también puede ser un vector en $\mathbb{R}^k$.

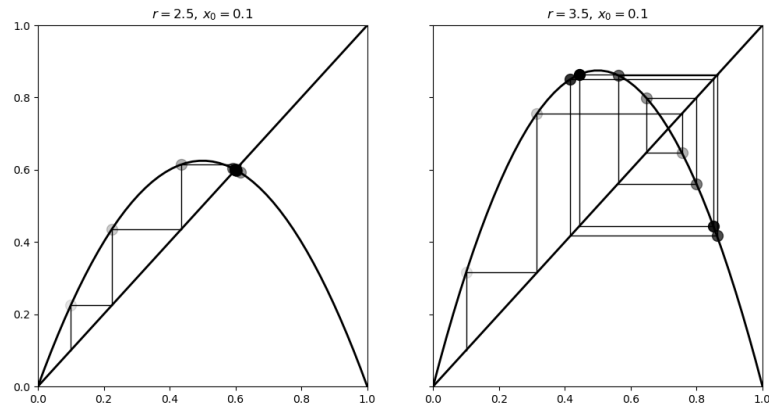
La ecuación logística discreta Continuando con el ejemplo del crecimiento de la población dada por la ecuación Logística, en este caso el crecimiento de la población se expresa como la fracción de la población máxima admisible existente, en el paso del período n al período $n + 1$, dado por la diferencia $x_{n+1} - x_n$, resultando esta proporcional a la población existente al principio del período temporal y a la diferencia entre el valor de saturación y dicha población.

En la ecuación $x_{n+1} = rx_n(1 - x_n)$ podemos observar que, dada una condición inicial x_0 , queda determinada una sucesión de valores $x_0, x_1, x_2, \dots, x_n$. Pero en el caso discreto -a diferencia del modelo continuo en el que se puede hallar la población como función explícita del tiempo t -, es posible obtener x como función explícita de n , esto significa que no se puede resolver analíticamente la ecuación en diferencias, de modo que si queremos obtener el valor de x_n tendremos que calcularlo uno a uno.

A diferencia de la ecuación logística continua, en la ecuación logística discreta los procesos estudiados se observan en instantes puntuales distintos y los datos experimentales obtenidos de esta manera forman entonces un conjunto discreto y ordenado de valores. Para el modelo que tomamos para este trabajo son la cantidad de población tras el primer ciclo, el siguiente número tras el segundo ciclo, luego el tercer número, y así sucesivamente.

Simulemos algunas iteraciones de este sistema con dos valores diferentes de r usando el código:

```
def plot_system(r, x0, n, ax=None):
    t = np.linspace(0, 1)
    ax.plot(t, logistic(r, t), 'k', lw=2)
    ax.plot([0, 1], [0, 1], 'k', lw=2)
    x = x0
    for i in range(n):
        y = logistic(r, x)
        ax.plot([x, x], [x, y], 'k', lw=1)
        ax.plot([x, y], [y, y], 'k', lw=1)
        ax.plot([x], [y], 'ok', ms=10,
                alpha=(i + 1) / n)
        x = y
    ax.set_xlim(0, 1)
    ax.set_ylim(0, 1)
    ax.set_title(f"$r={r:.1f}$, \, $x_0={x0:.1f}$")
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6), sharey=True)
plot_system(2.5, .1, 10, ax=ax1)
plot_system(3.5, .1, 10, ax=ax2)
plt.show()
```



A continuación generamos el diagrama de bifurcación de la ecuación logística discreta, este diagrama, conocido también como diagrama de Feigenbaum, al ejecutarlo podremos observar cómo el comportamiento del sistema pasa de ser ordenado a ser caótico:

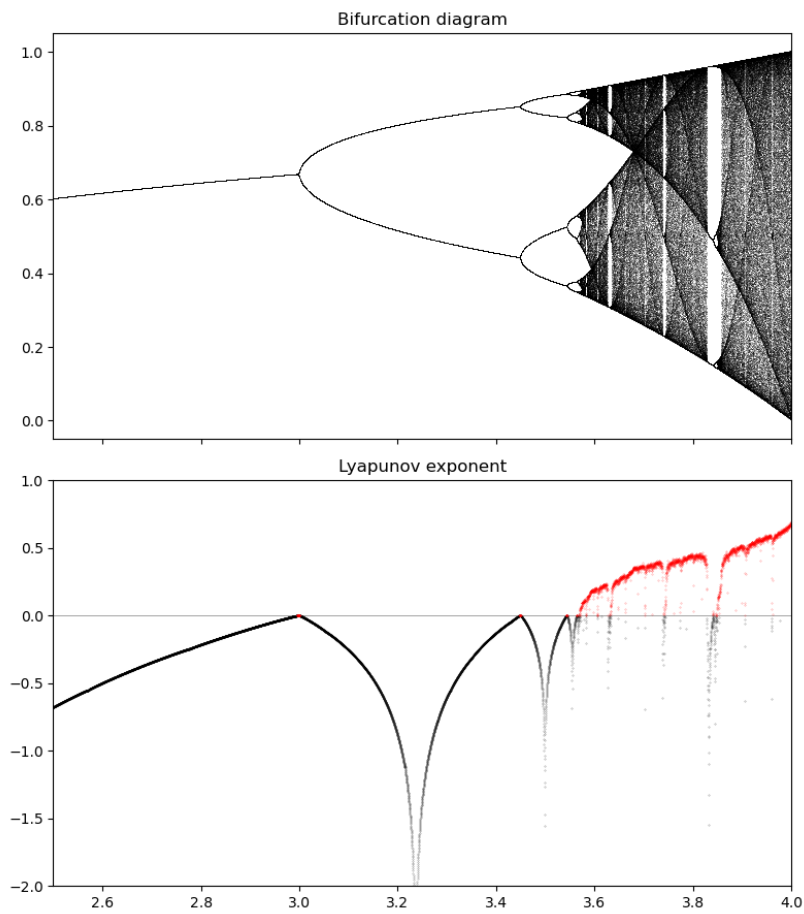
```
n = 10000
r = np.linspace(2.5, 4.0, n)
iterations = 1000
last = 100
x = 1e-5 * np.ones(n)
lyapunov = np.zeros(n)
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(8, 9), sharex=True)
for i in range(iterations):
    x = logistic(r, x)
    lyapunov += np.log(abs(r - 2 * r * x))
    if i >= (iterations - last):
        ax1.plot(r, x, 'k', alpha=.25)
ax1.set_xlim(2.5, 4)
ax1.set_title("Bifurcation diagram")
ax2.axhline(0, color='k', lw=.5, alpha=.5)
ax2.plot(r[lyapunov < 0],
        lyapunov[lyapunov < 0] / iterations, 'k',
        alpha=.5, ms=.5)
ax2.plot(r[lyapunov >= 0],
```



```

    lyapunov[lyapunov >= 0] / iterations,
    'r', alpha=.5, ms=.5)
ax2.set_xlim(2.5, 4)
ax2.set_ylim(-2, 1)
ax2.set_title("Lyapunov exponent")
plt.tight_layout()
plt.show()

```



También calculamos una aproximación del exponente de Lyapunov para

cada valor de r . El exponente de Lyapunov se define por:

$$\lambda(r) = \lim_{n \rightarrow \infty} \frac{1}{n} \sum_{i=0}^{n-1} \log \left| \frac{df_r}{dx} (x_i^r) \right|$$

El diagrama de bifurcación pone de manifiesto la existencia de un punto fijo para $r < 3$, luego dos y cuatro equilibrios, y un comportamiento caótico cuando r pertenece a ciertas áreas del espacio de parámetros.

Observamos una propiedad importante del exponente de Lyapunov: es positivo cuando el sistema es caótico (en rojo aquí).

9.3 Fractales

Un fractal es un objeto geométrico cuya estructura básica, fragmentada o aparentemente irregular, se repite a diferentes escalas. El término fue propuesto por el matemático Benoît Mandelbrot en 1975 y deriva del latín *fractus*, que significa quebrado o fracturado. Muchas estructuras naturales son de tipo fractal. La propiedad matemática clave de un objeto genuinamente fractal es que su dimensión métrica fractal es un número racional no entero.

Los ejemplos clásicos los primeros ejemplos de fractales surgen a finales del siglo XIX: en 1872 apareció la función de Weierstrass, cuyo grafo hoy en día se considera fractal, como ejemplo de función continua pero no diferenciable en ningún punto.

Curva de Koch en 1904, Helge von Koch definió una curva con propiedades similares a la de Weierstrass: el copo de nieve de Koch, también llamado estrella de Koch, es una curva cerrada continua pero no diferenciable en ningún punto descrita por el matemático sueco Helge von Koch en 1904 en un artículo titulado "Acerca de una curva continua que no posee tangentes y obtenida por los métodos de la geometría elemental".

En lenguaje actual, diríamos que es una curva fractal. Su construcción más simple se realiza mediante un proceso iterativo que se inicia partiendo en tres un segmento de recta e insertando dos más en el tercero medio a manera de un triángulo equilátero, el proceso se repite infinitas veces. La curva de Koch es un caso particular de curva de De Rham.

Veamos el proceso que lleva a sustituir cada lado por la llamada curva de Koch: Se toma un segmento, se divide en tres partes iguales, se reemplaza la

parte central por dos partes de igual longitud haciendo un ángulo de 60 grados. Luego, con los cuatro segmentos, se procede de la misma manera, lo que da lugar a 16 segmentos más pequeños en la segunda iteración. Y así sucesivamente. La figura representa las seis primeras etapas de la construcción. La última curva es una buena aproximación de la curva final.

Longitud Si se considera de nuevo la primera figura, notamos que para pasar de una línea a la siguiente se remplace tres segmentos por cuatro de igual longitud, o sea que la longitud total es multiplicada por $4/3$. Después de n pasos iterativos en la construcción recursiva la longitud de la curva es $3 \cdot (4/3)^n$, el límite de la sucesión geométrica anterior de razón $4/3$ es obviamente infinito, lo que significa que la figura final tiene una longitud infinita (lo que Mandelbrot denomina infinito interno). Esto está relacionado con el hecho de que la curva frontera del copo de Koch no es rectificable y tiene una dimensión fractal $d > 1$.

Propiedades fractales Autosimilitud exacta del Copo de nieve de Koch a todas las escalas.

La característica anterior, típica de muchas curvas fractales, añadida al hecho que la curva da la impresión de tener cierto espesor a causa de sus constantes cambios de dirección, sugiere que esta figura, en algún sentido, no es unidimensional. Para ello usaremos una generalización del concepto de dimensión: la dimensión fractal de Hausdorff.

La dimensión de Hausdorff tiene que estar entre 1, la de una recta, y 2, la del plano. Para hallarla veamos la última curva: Si agrandamos (mediante una homotecia) tres veces la sección $A'B'$ obtenemos exactamente la sección AB . En la curva final, obtendríamos la sección $A'C$, es decir cuatro veces la sección inicial.

Se sabe que una homotecia de razón tres multiplica las longitudes por 3, las superficies por $3^2 = 9$, los volúmenes por $3^3 = 27$, y más generalmente, el "volumen" de objeto de dimensión d por 3^d . Entonces tenemos $3^d = 4$ para el copo de Koch, lo que da:

$$d = \frac{\ln 4}{\ln 3} \approx 1.26186...$$

La dimensión de homotecia anterior coincide en este caso con la dimensión fractal de Hausdorff. La configuración opuesta-complementaria de un copo

de nieve de Koch o copo de nieve fractal suele ser denominada anticopo de nieve.

A continuación mostramos el código en Python para generar la curva del Copo de nieve de Koch:

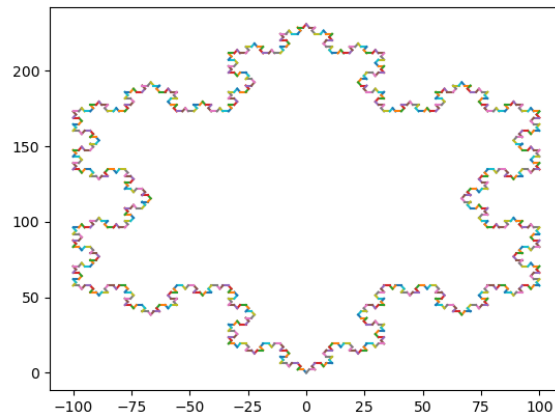
```
from math import sin, cos, pi
import matplotlib.pyplot as plt
import matplotlib

def kochSnowFlake(l, n):
    x_vertex1 = 0
    y_vertex1 = 0
    x_vertex2 = l * cos ( 2 * pi/3)
    y_vertex2 = l * sin(2 * pi/3)
    x_vertex3 = l * cos (pi/3)
    y_vertex3 = l * sin(pi/3)
    kochCurve(x_vertex1, y_vertex1, x_vertex2, y_vertex2, n)
    kochCurve(x_vertex2, y_vertex2, x_vertex3, y_vertex3, n)
    kochCurve(x_vertex3, y_vertex3, x_vertex1, y_vertex1, n)
    return

def kochCurve(xi, yi, xf, yf, n):
    if n==0:
        plt.plot([xi,xf],[yi,yf])
    elif n>0:
        x1 = xi + (xf - xi) / 3.0
        y1 = yi + (yf - yi) / 3.0
        x3 = xf - (xf - xi) / 3.0
        y3 = yf - (yf - yi) / 3.0
        x2 = (x1 + x3) * cos (pi/3) - ( y3 - y1) * sin (pi/3)
        y2 = (y1 + y3) * cos (pi/3) + ( x3 - x1) * sin (pi/3)
        kochCurve(xi,yi,x1,y1,n-1)
```

```
    kochCurve(x1,y1,x2,y2,n-1)
    kochCurve(x2,y2,x3,y3,n-1)
    kochCurve(x3,y3,xf,yf,n-1)

    return
plt.figure("Koch snowflack")
kochSnowFlake(200,4)
plt.show()
```



El conjunto de Mandelbrot La familia de conjuntos de Julia surgen como resultado de la aplicación reiterada de funciones Holomorfas -estas se definen sobre un subconjunto abierto del plano complejo, además son complejo-diferenciables en cada punto-

$$z \rightarrow f(z) \rightarrow f(f(z)) \rightarrow \dots$$

asociadas a la reiteración de funciones de la forma

$$f_c(z) = z^2 + c$$

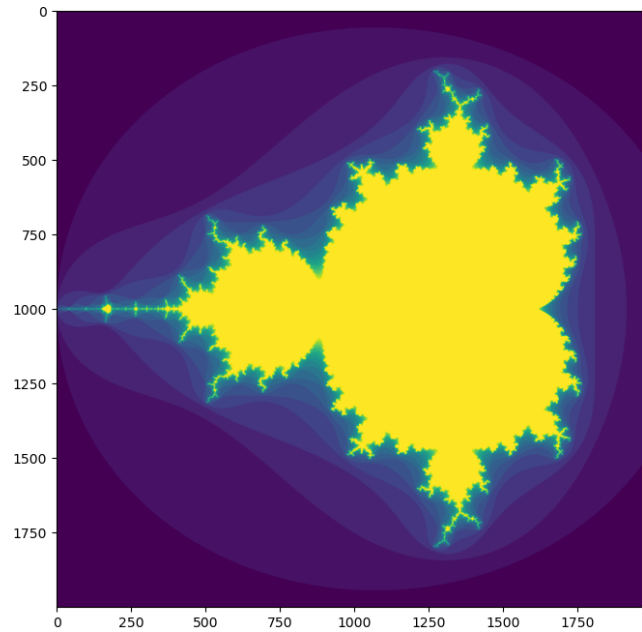
presenta conjuntos de una variedad sorprendente.

Dicha familia tendrá especial relevancia al quedar parametrizada en un mapa de fractales, popularizado en los años 1980, llamado conjunto de Mandelbrot. Este conjunto M representa un mapa en que cada pixel, correspondiente a un valor del parámetro c . Todos los puntos del plano complejo $C = (C_x, iC_y)$ son iterados por adición a la función correspondiente. El fractal derivado de la función $f_c(z) = z^2 + c$ se denomina conjunto de Mandelbrot.

A continuación mostramos el código en Python para generar el conjunto de Mandelbrot:

```
import matplotlib.pyplot as plt
import numpy as np
def mandelbrot(h, w, maxit=20):
    y, x = np.ogrid[-1.4 : 1.4 : h * 1j, -2 : 0.8 : w * 1j]
    c = x + y * 1j
    z = c
    divtime = maxit + np.zeros(z.shape, dtype=int)
    for i in range(maxit):
        z = z ** 2 + c
        diverge = z * np.conj(z) > 2 ** 2
        div_now = diverge & (divtime == maxit)
        divtime[div_now] = i
        z[diverge] = 2
    return divtime
plt.figure(figsize=(8, 8))
plt.imshow(mandelbrot(2000, 2000))
plt.show()
```

Generando:

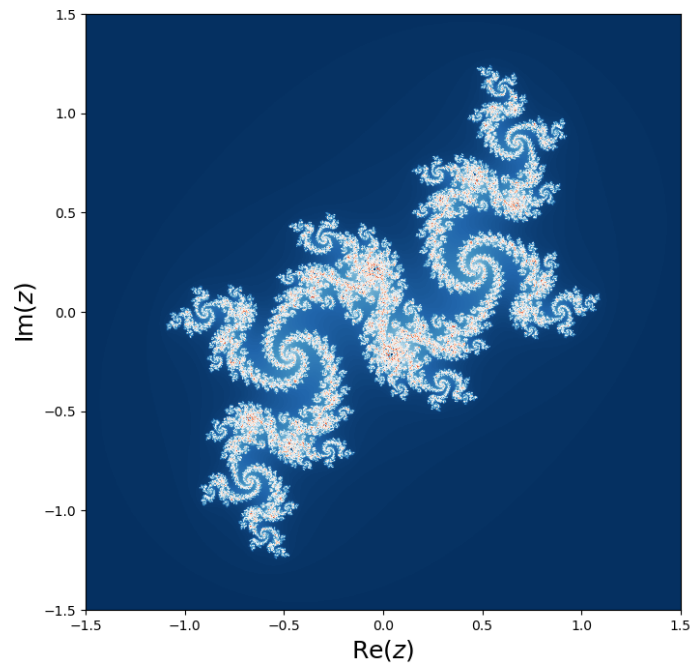


y tomando un punto en el plano complejo $(-0.05 + 0.68i)$ generamos el conjunto de Julia usando el código:

```
import numpy as np
import numba
def py_julia_fractal(z_re, z_im, j):
    for m in range(len(z_re)):
        for n in range(len(z_im)):
            z = z_re[m] + 1j * z_im[n]
            for t in range(256):
                z = z ** 2 - 0.05 + 0.68j
                if np.abs(z) > 2.0:
                    j[m, n] = t
                    break
jit_julia_fractal = numba.jit(nopython=True)(py_julia_fractal)
N = 1024
```

```
j = np.zeros((N, N), np.int64)
z_real = np.linspace(-1.5, 1.5, N)
z_imag = np.linspace(-1.5, 1.5, N)
jit_julia_fractal(z_real, z_imag, j)
fig, ax = plt.subplots(figsize=(8, 8))
ax.imshow(j, cmap=plt.cm.RdBu_r, extent=[-1.5, 1.5, -1.5,
1.5])
ax.set_xlabel(" $\mathrm{Re}(z)$ ", fontsize=18)
ax.set_ylabel(" $\mathrm{Im}(z)$ ", fontsize=18)
plt.show()
```

Generando:



Algo más de Fractales Existen una diversidad de sitios con imágenes y vídeos de fractales, por ejemplo en:

<https://www.youtube.com/watch?v=pCpLWbHVNhk&app=desktop#menu>

donde se muestra un acercamiento continuo dentro del fractal de Mandelbrot con una duración total de 1:10:59.

9.4 Derivadas

La derivación numérica es una técnica de análisis numérico para calcular una aproximación a la derivada de una función en un punto utilizando los valores y propiedades de la misma. Para ello usamos la expansión en Series de Taylor.

Expansión en Series de Taylor Sea $f(x)$ una función definida en (a, b) que tiene hasta la k -ésima derivada, entonces la expansión de $f(x)$ usando series de Taylor alrededor del punto x_i contenido en el intervalo (a, b) será

$$f(x) = f(x_i) + \frac{(x - x_i)}{1!} \left. \frac{df}{dx} \right|_{x_i} + \frac{(x - x_i)^2}{2!} \left. \frac{d^2f}{dx^2} \right|_{x_i} + \dots + \frac{(x - x_i)^k}{k!} \left. \frac{d^kf}{dx^k} \right|_{\varepsilon} \quad (9.1)$$

donde $\varepsilon = x_i + \theta(x - x_i)$ y $0 < \theta < 1$.

Aproximación de la Primera Derivada Existen distintas formas de generar la aproximación a la primera derivada, nos interesa una que nos dé la mejor precisión posible con el menor esfuerzo computacional.

Diferencias Progresivas Considerando la Ec.(9.1) con $k = 2$ y $x = x_i + \Delta x$, tenemos

$$f(x_i + \Delta x) = f(x_i) + \Delta x \left. \frac{df}{dx} \right|_{x_i} + \frac{\Delta x^2}{2!} \left. \frac{d^2f}{dx^2} \right|_{\varepsilon_p} \quad (9.2)$$

de esta ecuación obtenemos la siguiente expresión para la aproximación de la primera derivada

$$\left. \frac{df}{dx} \right|_{x_i} = \frac{f(x_i + \Delta x) - f(x_i)}{\Delta x} - \frac{\Delta x}{2!} \left. \frac{d^2f}{dx^2} \right|_{\varepsilon_p} \quad (9.3)$$

en este caso la aproximación de $f'(x)$ mediante diferencias progresivas es de primer orden, es decir $O(\Delta x)$. Siendo $O_p(\Delta x)$ el error local de truncamiento, definido como

$$O_p(\Delta x) = -\frac{\Delta x}{2!} \left. \frac{d^2 f}{dx^2} \right|_{\varepsilon_p}. \quad (9.4)$$

Es común escribir la expresión anterior como:

$$\left. \frac{df}{dx} \right|_{x_i} = \frac{f(x_i + \Delta x) - f(x_i)}{\Delta x} - O_p(\Delta x) \quad (9.5)$$

o

$$f'(x_i) = \frac{f_{i+1} - f_i}{\Delta x} \quad (9.6)$$

para simplificar la notación.

Diferencias Regresivas Considerando la Ec.(9.1) con $k = 2$ y $x = x_i - \Delta x$, tenemos

$$f(x_i - \Delta x) = f(x_i) - \Delta x \left. \frac{df}{dx} \right|_{x_i} + \frac{\Delta x^2}{2!} \left. \frac{d^2 f}{dx^2} \right|_{\varepsilon_r} \quad (9.7)$$

de esta ecuación obtenemos la siguiente expresión para la aproximación de la primera derivada

$$\left. \frac{df}{dx} \right|_{x_i} = \frac{f(x_i) - f(x_i - \Delta x)}{\Delta x} - \frac{\Delta x}{2!} \left. \frac{d^2 f}{dx^2} \right|_{\varepsilon_r} \quad (9.8)$$

en este caso la aproximación de $f'(x)$ mediante diferencias regresivas es de primer orden, es decir $O(\Delta x)$. Siendo $O_r(\Delta x)$ el error local de truncamiento, definido como

$$O_r(\Delta x) = \frac{\Delta x}{2!} \left. \frac{d^2 f}{dx^2} \right|_{\varepsilon_r}. \quad (9.9)$$

Es común escribir la expresión anterior como

$$\left. \frac{df}{dx} \right|_{x_i} = \frac{f(x_i) - f(x_i - \Delta x)}{\Delta x} + O_r(\Delta x) \quad (9.10)$$

o

$$f'(x_i) = \frac{f_i - f_{i-1}}{\Delta x} \quad (9.11)$$

para simplificar la notación.

Diferencias Centradas Considerando la Ec.(9.1) con $k = 3$ y escribiendo $f(x)$ en $x = x_i + \Delta x$ y $x = x_i - \Delta x$, tenemos

$$f(x_i + \Delta x) = f(x_i) + \Delta x \left. \frac{df}{dx} \right|_{x_i} + \frac{\Delta x^2}{2!} \left. \frac{d^2 f}{dx^2} \right|_{x_i} + \frac{\Delta x^3}{3!} \left. \frac{d^3 f}{dx^3} \right|_{\varepsilon_p} \quad (9.12)$$

y

$$f(x_i - \Delta x) = f(x_i) - \Delta x \left. \frac{df}{dx} \right|_{x_i} + \frac{\Delta x^2}{2!} \left. \frac{d^2 f}{dx^2} \right|_{x_i} - \frac{\Delta x^3}{3!} \left. \frac{d^3 f}{dx^3} \right|_{\varepsilon_r} \quad (9.13)$$

restando la Ec.(9.12) de la Ec.(9.13), se tiene

$$f(x_i + \Delta x) - f(x_i - \Delta x) = 2\Delta x \left. \frac{df}{dx} \right|_{x_i} + \frac{\Delta x^3}{3!} \left[\left. \frac{d^3 f}{dx^3} \right|_{\varepsilon_p} + \left. \frac{d^3 f}{dx^3} \right|_{\varepsilon_r} \right] \quad (9.14)$$

esta última expresión lleva a la siguiente aproximación de la primera derivada mediante diferencias centradas

$$\left. \frac{df}{dx} \right|_{x_i} = \frac{f(x_i + \Delta x) - f(x_i - \Delta x)}{2\Delta x} + O_c(\Delta x^2) \quad (9.15)$$

con un error local de truncamiento de segundo orden $O_c(\Delta x^2)$, es decir

$$O_c(\Delta x^2) = \frac{\Delta x^2}{3!} \left[\left. \frac{d^3 f}{dx^3} \right|_{\varepsilon_p} + \left. \frac{d^3 f}{dx^3} \right|_{\varepsilon_r} \right] \quad (9.16)$$

comparado el error local de truncamiento de la aproximación anterior $O_c(\Delta x^2)$, con los obtenidos previamente para diferencias progresivas $O_p(\Delta x)$ Ec.(9.5) y regresivas $O_r(\Delta x)$ Ec.(9.10), se tiene que

$$\lim_{\Delta x \rightarrow 0} O_c(\Delta x^2) < \lim_{\Delta x \rightarrow 0} O_p(\Delta x). \quad (9.17)$$

Es común encontrar expresada la derivada⁸⁵

$$\left. \frac{df}{dx} \right|_{x_i} = \frac{f(x_i + \Delta x) - f(x_i - \Delta x)}{2\Delta x} \quad (9.18)$$

como

$$f'(x_i) = \frac{f_{i+1} - f_{i-1}}{2\Delta x} \quad (9.19)$$

para simplificar la notación.

⁸⁵En el caso de que la derivada sea usada en una malla no homogénea, es necesario incluir en la derivada a que Δx se refiere, por ejemplo en cada punto i , tenemos la Δx_{i-} (por la izquierda) y la Δx_{i+} (por la derecha), i.e. $\left. \frac{df}{dx} \right|_{x_i} = \frac{f(x_i + \Delta x_{i-}) - f(x_i - \Delta x_{i+})}{(\Delta x_{i-}) + (\Delta x_{i+})}$.

Derivadas Usando Más Puntos Utilizando el valor de la función en más puntos se construyen fórmulas más precisas para las derivadas⁸⁶, algunos ejemplos son

$$\begin{aligned}
 f'(x_i) &= \frac{-3f_i + 4f_{i+1} - f_{i+2}}{2\Delta x} + O(\Delta x^2) \\
 f'(x_i) &= \frac{3f_i - 4f_{i-1} + f_{i-2}}{2\Delta x} + O(\Delta x^2) \\
 f'(x_i) &= \frac{2f_{i+1} + 3f_i - 6f_{i-1} + f_{i-2}}{6\Delta x} + O(\Delta x^3) \\
 f'(x_i) &= \frac{f_{i-2} - 8f_{i-1} + 8f_{i+1} - f_{i+2}}{12\Delta x} + O(\Delta x^4) \\
 f'(x_i) &= \frac{-25f_i + 48f_{i+1} - 36f_{i+2} + 16f_{i+3} - 3f_{i+4}}{12\Delta x} + O(\Delta x^4)
 \end{aligned} \tag{9.20}$$

Derivadas de Ordenes Mayores De forma análoga se construyen aproximaciones en diferencias finitas de orden mayor, aquí desarrollaremos la forma de calcular la derivada de orden dos en diferencias centradas.

Derivada de Orden Dos Partiendo del desarrollo de Taylor

$$f(x_i + \Delta x) = f(x_i) + \Delta x f'(x_i) + \frac{\Delta x^2}{2!} f''(x_i) + \frac{\Delta x^3}{3!} f'''(x_i) + \frac{\Delta x^4}{4!} f^{(4)}(\xi_p) \tag{9.21}$$

y

$$f(x_i - \Delta x) = f(x_i) - \Delta x f'(x_i) + \frac{\Delta x^2}{2!} f''(x_i) - \frac{\Delta x^3}{3!} f'''(x_i) + \frac{\Delta x^4}{4!} f^{(4)}(\xi_r) \tag{9.22}$$

eliminando las primeras derivadas, sumando las ecuaciones anteriores y despejando se encuentra que

$$f''(x_i) = \frac{f(x_i - \Delta x) - 2f(x_i) + f(x_i + \Delta x)}{\Delta x^2} - \frac{\Delta x^2}{12} f^{(4)}(\xi_c) \tag{9.23}$$

⁸⁶ Al usar estas derivadas en el método de diferencias finitas mostrado en la sección (9.9) las matrices generadas no serán tridiagonales.

así, la aproximación a la segunda derivada usando diferencias centradas con un error de truncamiento $O_c(\Delta x^2)$ es⁸⁷

$$f''(x_i) = \frac{f(x_i - \Delta x) - 2f(x_i) + f(x_i + \Delta x)}{\Delta x^2} \quad (9.24)$$

Es común escribir la expresión anterior como

$$f''(x_i) = \frac{f_{i-1} - 2f_i + f_{i+1}}{\Delta x^2}$$

para simplificar la notación.

Derivadas Usando Más Puntos Utilizando el valor de la función en más puntos se construyen fórmulas más precisas para las derivadas, algunos ejemplos son

$$\begin{aligned} f''(x_i) &= \frac{f_{i+2} - 2f_{i+1} + f_i}{\Delta x^2} + O(\Delta x) \\ f''(x_i) &= \frac{-f_{i+3} + 4f_{i+2} - 5f_{i+1} + 2f_i}{\Delta x^2} + O(\Delta x^2) \\ f''(x_i) &= \frac{-f_{i+2} + 16f_{i+1} - 30f_i + 16f_{i-1} - f_{i-2}}{12\Delta x^2} + O(\Delta x^4) \end{aligned} \quad (9.25)$$

Derivadas de Orden Tres y Cuatro De forma análoga se construyen derivadas de ordenes mayores utilizando el valor de la función en más puntos, algunos ejemplos para terceras derivadas son

$$\begin{aligned} f'''(x_i) &= \frac{f_{i+3} - 3f_{i+2} + 3f_{i+1} - f_i}{\Delta x^3} + O(\Delta x) \\ f'''(x_i) &= \frac{f_{i+2} - 2f_{i+1} + 2f_{i-1} - f_{i-2}}{2\Delta x^3} + O(\Delta x^2) \\ f'''(x_i) &= \frac{f_{i-3} - 8f_{i-2} + 13f_{i-1} - 13f_{i+1} + 8f_{i+2} - f_{i+3}}{8\Delta x^3} + O(\Delta x^4) \end{aligned} \quad (9.26)$$

⁸⁷En el caso de que la derivada sea usada en una malla no homogénea, es necesario incluir en la derivada a que Δx se refiere, por ejemplo en cada punto i , tenemos la Δx_{i-} (por la izquierda) y la Δx_{i+} (por la derecha), i.e. $f''(x_i) = \frac{f(x_i - \Delta x_{i-}) - 2f(x_i) + f(x_i + \Delta x_{i+})}{(\Delta x_{i-})(\Delta x_{i+})}$

Algunos ejemplos para cuartas derivadas son

$$\begin{aligned}f''''(x_i) &= \frac{f_{i+4} - 4f_{i+3} + 6f_{i+2} - 4f_{i+1} + f_i}{\Delta x^4} + O(\Delta x) \\f''''(x_i) &= \frac{f_{i+2} - 4f_{i+1} + 6f_i - 4f_{i-1} + f_{i-2}}{\Delta x^4} + O(\Delta x^2) \\f''''(x_i) &= \frac{-f_{i+3} + 12f_{i+2} - 39f_{i+1} + 56f_i - 39f_{i-1} + 12f_{i-2} - f_{i-3}}{6\Delta x^4} + O(\Delta x^4)\end{aligned}\tag{9.27}$$

Análisis de Convergencia En el siguiente ejemplo escrito en Octave se muestra la comparación de errores de truncamiento para los esquemas de aproximación de diferencias finitas progresiva, regresiva y centrada. Además gráfica el error y el orden estimado de convergencia, para la función *Seno* y su derivada exacta (*Coseno*) en el punto 1:

Ejemplo 1 % $u(x) = \sin(x)$ en $x=1$. Derivada exacta $u'(1) = \cos(1)$

```
clear; close all
h=1.0;
for k = 1:33,
    a(k,1) = h;
    a(k,2) = (sin(1 + h) - sin(1)) / h - cos(1);
    a(k,3) = (sin(1) - sin(1 - h)) / h - cos(1);
    a(k,4) = (sin(1 + h) - sin(1 - h)) / (2 * h) - cos(1);
    h = h / 2;
end

format short e
a % Visualiza el resultado

a = abs(a); % Toma el valor absoluto
h1 = a(:,1); % Extrae la primer columna que es la de h
e1 = a(:,2); e2 = a(:,3); e3 = a(:,4);

loglog(h1, e1, h1, e2, h1, e3)
axis('square')
axis([1e-6 1e1 1e-6 1e1])
gtext('Esperada de adelante y atras =1')
gtext('Esperada de central = 2')
```

Generado la siguiente salida (la primera columna es h , las demás son los errores para los esquemas de aproximación de diferencias finitas progresiva, regresiva y centrada respectivamente):

```

1.0000e+00 -4.7248e-01 3.0117e-01 -8.5654e-02
5.0000e-01 -2.2825e-01 1.8379e-01 -2.2233e-02
2.5000e-01 -1.1025e-01 9.9027e-02 -5.6106e-03
1.2500e-01 -5.3929e-02 5.1118e-02 -1.4059e-03
6.2500e-02 -2.6639e-02 2.5936e-02 -3.5169e-04
3.1250e-02 -1.3235e-02 1.3059e-02 -8.7936e-05
1.5625e-02 -6.5958e-03 6.5519e-03 -2.1985e-05
7.8125e-03 -3.2925e-03 3.2815e-03 -5.4962e-06
3.9062e-03 -1.6449e-03 1.6421e-03 -1.3741e-06
1.9531e-03 -8.2209e-04 8.2141e-04 -3.4351e-07
9.7656e-04 -4.1096e-04 4.1079e-04 -8.5879e-08
4.8828e-04 -2.0546e-04 2.0542e-04 -2.1470e-08
2.4414e-04 -1.0272e-04 1.0271e-04 -5.3673e-09
1.2207e-04 -5.1361e-05 5.1358e-05 -1.3417e-09
6.1035e-05 -2.5680e-05 2.5679e-05 -3.3579e-10
3.0518e-05 -1.2840e-05 1.2840e-05 -8.3860e-11
1.5259e-05 -6.4199e-06 6.4199e-06 -2.2014e-11
7.6294e-06 -3.2100e-06 3.2100e-06 -1.8607e-13
3.8147e-06 -1.6050e-06 1.6050e-06 7.0899e-12
1.9073e-06 -8.0249e-07 8.0247e-07 -7.4620e-12
9.5367e-07 -4.0120e-07 4.0125e-07 2.1642e-11
4.7684e-07 -2.0062e-07 2.0055e-07 -3.6566e-11
2.3842e-07 -1.0050e-07 1.0020e-07 -1.5298e-10
1.1921e-07 -4.9746e-08 4.9906e-08 7.9849e-11
5.9605e-08 -2.5532e-08 2.4760e-08 -3.8581e-10
2.9802e-08 -1.0630e-08 1.1721e-08 5.4551e-10
1.4901e-08 -6.9051e-09 7.9961e-09 5.4551e-10
7.4506e-09 5.4551e-10 5.4551e-10 5.4551e-10
3.7253e-09 5.4551e-10 5.4551e-10 5.4551e-10
1.8626e-09 -2.9257e-08 -2.9257e-08 -2.9257e-08
9.3132e-10 -2.9257e-08 -2.9257e-08 -2.9257e-08
4.6566e-10 -2.9257e-08 -2.9257e-08 -2.9257e-08
2.3283e-10 -2.9257e-08 -2.9257e-08 -2.9257e-08

```

Cómo se nota en la salida anterior, la convergencia mejora conforme h se hace pequeña, hasta llegar a cierto valor en que por errores de truncamiento, el error en los esquemas de aproximación en la derivada comienza a incrementarse. Esto nos da una idea de los valores de h que podemos usar para los esquemas de aproximación de diferencias finitas progresiva, regresiva y centrada respectivamente.

9.5 Integrales

En análisis numérico, la integración numérica constituye una amplia gama de algoritmos para calcular el valor numérico de una integral definida y, por extensión, el término se usa a veces para describir algoritmos numéricos para resolver ecuaciones diferenciales. El término cuadratura numérica (a menudo abreviado a cuadratura) es más o menos sinónimo de integración numérica, especialmente si se aplica a integrales de una dimensión a pesar de que para el caso de dos o más dimensiones (integral múltiple) también se utiliza.

El problema básico considerado por la integración numérica es calcular una solución aproximada a la integral definida

$$\int_a^b f(x)dx$$

Este problema también puede ser enunciado como un problema de valor inicial para una ecuación diferencial ordinaria, como sigue

$$y'(x) = f(x), \quad y(a) = 0$$

Encontrar $y(b)$ es equivalente a calcular la integral. Los métodos desarrollados para ecuaciones diferenciales ordinarias, como el método de Runge-Kutta, pueden ser aplicados al problema reformulado.

Razones para la integración numérica Hay varias razones para llevar a cabo la integración numérica. La principal puede ser la imposibilidad de realizar la integración de forma analítica. Es decir, integrales que requerirían de un gran conocimiento y manejo de matemática avanzada pueden ser resueltas de una manera más sencilla mediante métodos numéricos. Incluso existen funciones integrables pero cuya primitiva no puede ser calculada, siendo la integración numérica de vital importancia. La solución analítica

de una integral nos arrojaría una solución exacta, mientras que la solución numérica nos daría una solución aproximada. El error de la aproximación, que depende del método que se utilice y de qué tan fino sea, puede llegar a ser tan pequeño que es posible obtener un resultado idéntico a la solución analítica en las primeras cifras decimales.

Cuadratura de Gauss En análisis numérico un método de cuadratura es una aproximación de una integral definida de una función. Una cuadratura de Gauss n , es una cuadratura que selecciona los puntos de la evaluación de manera óptima y no en una forma igualmente espaciada, construida para dar el resultado de un polinomio de grado $2n - 1$ o menos, elegibles para los puntos x_i y los coeficientes w_i para $i = 1, \dots, n$. El dominio de tal cuadratura por regla es de $[-1, 1]$ dada por:

$$\int_{-1}^1 f(x)dx \approx \sum_{i=1}^n w_i f(x_i)$$

la cuadratura dará resultados precisos solo si $f(x)$ es aproximado por un polinomio dentro del rango $[-1, 1]$. Si la función puede ser escrita como $f(x) = W(x)g(x)$, donde $g(x)$ es un polinomio aproximado y $W(x)$ es conocido.

$$\int_{-1}^1 f(x)dx = \int_{-1}^1 W(x)g(x)dx \approx \sum_{i=1}^n w_i g(x_i)$$

Formula para calcular w_i también conocida como el método de Gauss-Legendre, los coeficientes están dados por

$$w_i = \frac{2}{(1 - x_i^2) [P'_n(x_i)]^2}$$

donde P_n son los polinomios de Legendre en el intervalo $[-1, 1]$.

Polinomios de Legendre las funciones de Legendre son las soluciones de las ecuaciones diferenciales de Legendre:

$$\frac{d}{dx} \left[(1 - x^2) \frac{d}{dx} P_n(x) \right] + n(n + 1) P_n(x) = 0$$

la ecuación diferencial de Legendre puede resolverse usando el método de series de potencias. En general la serie de potencias obtenida converge cuando $|x| < 1$ y en el caso particular cuando n sea un entero no negativo las soluciones forman una familia de polinomios ortogonales llamados polinomios de Legendre. Cada polinomio de Legendre $P_n(x)$ es un polinomio de grado n . Este se puede ser expresado usando la fórmula:

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} [(x^2 - 1)^n]$$

desarrollando esta fórmula, se obtiene la siguiente expresión para los polinomios de Legendre

$$P_n(x) = \frac{1}{2^n} \sum_{k=0}^n \binom{n}{k}^2 (x+1)^{n-k} (x-1)^k$$

esta última expresión es útil en caso de querer elaborar un programa que grafique los polinomios de Legendre.

Los primeros polinomios de Legendre se muestran en la siguiente tabla:

n	$P_n(x)$
0	1
1	x
2	$\frac{1}{2}(3x^2 - 1)$
3	$\frac{1}{2}(5x^3 - 3x)$
4	$\frac{1}{8}(35x^4 - 30x^2 + 3)$

Así, la lista de coeficientes de w_i y puntos x_i para $n = 1, \dots, 5$ esta dada en la siguiente tabla:

Número de puntos	Puntos x_i	Pesos w_i
1	0	2
2	$\pm\sqrt{\frac{1}{3}}$	1
3	0 $\pm\sqrt{\frac{3}{5}}$	$\frac{8}{9}$ $\frac{5}{9}$
4	$\pm\sqrt{\left(3 - 2\left(\sqrt{\frac{6}{5}}\right)\right)/7}$ $\pm\sqrt{\left(3 + 2\left(\sqrt{\frac{6}{5}}\right)\right)/7}$	$\frac{18+\sqrt{30}}{36}$ $\frac{18-\sqrt{30}}{36}$
5	0 $\pm\sqrt{5 - 2\left(\sqrt{\frac{10}{7}}\right)}$ $\pm\sqrt{5 + 2\left(\sqrt{\frac{10}{7}}\right)}$	$\frac{128}{225}$ $\frac{322+13\sqrt{70}}{900}$ $\frac{322-13\sqrt{70}}{900}$

Cambio de Intervalos Los cambios de intervalos van de $[-1, 1]$ después de aplicar la cuadratura de Gauss:

$$\int_a^b f(x)dx = \frac{b-a}{2} \int_{-1}^1 f\left(\frac{b-a}{2}x + \frac{a+b}{2}\right) dx$$

Después de aplicar la cuadratura la aproximación es:

$$\int_a^b f(x)dx \approx \frac{b-a}{2} \sum_{i=1}^n w_i f\left(\frac{b-a}{2}x_i + \frac{a+b}{2}\right)$$

Ejemplo de Uso La integral

$$\int_1^5 (x^3 + 2x^2)dx = 238.66667$$

si se resuelve por cuadratura usando $n = 2$, entonces tenemos

$$\frac{b-a}{2} \int_{-1}^1 f\left(\frac{b-a}{2}x + \frac{a+b}{2}\right) dx = \frac{5-1}{2} \int_{-1}^1 f\left(\frac{5-1}{2}x + \frac{5+1}{2}\right) dx =$$

$$2 \int_{-1}^1 f(2x+3) dx \approx 2 \sum_{i=1}^2 w_i f(2x_i+3) =$$

$$2(w_1 f(2x_1+3) + w_2 f(2x_2+3)) = 238.6667$$

En el caso del lenguaje de programación Python disponemos de esta opción usando:

```
from scipy import integrate
import numpy as np

# integral de x ^2 en 0 a 4
x2 = lambda x: x**2
print(integrate.quad(x2, 0, 4))
# solucion analitica
print(4**3 / 3.)

# integral de e ^-x en 0 a inf
invexp = lambda x: np.exp(-x)
print(integrate.quad(invexp, 0, np.inf))
```

Integrando en Dos o Más Dimensiones es posible hacer integraciones en dos o más dimensiones y sin pérdida de generalidad, para dos dimensiones tenemos:

Los cambios de intervalos van de $[-1, 1] \times [-1, 1]$ después de aplicar la cuadratura de Gauss:

$$\int_a^b \int_c^d f(x, y) dx dy = \frac{b-a}{2} \frac{d-c}{2} \int_{-1}^1 \int_{-1}^1 f\left(\frac{b-a}{2}x + \frac{a+b}{2}, \frac{d-c}{2}x + \frac{c+d}{2}\right) dx dy$$

Después de aplicar la cuadratura la aproximación es:

$$\int_a^b \int_c^d f(x, y) dx dy \approx \frac{b-a}{2} \frac{d-c}{2} \sum_{i=1}^n \sum_{j=1}^n w_i w_j f\left(\frac{b-a}{2}x_i + \frac{a+b}{2}, \frac{d-c}{2}y_j + \frac{c+d}{2}\right)$$

De esta forma se puede extender la integración por cuadratura Gaussiana para cualquier dimensión que se necesite, esto queda plasmado en el siguiente programa en C:

```
#include<stdio.h>
#include<math.h>
// Arreglo para las Xi
static double PUNTOS[][10] = {
    {0.577350269189625764509148780502,-0.577350269189625764509148780502,
    0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0},
    {0.774596669241483377035853079956,0.0,-0.774596669241483377035853079956,
    0.0,0.0,0.0,0.0,0.0,0.0},
    {0.861136311594052575223946488893,0.339981043584856264802665759103,
    -0.339981043584856264802665759103,-0.861136311594052575223946488893,
    0.0,0.0,0.0,0.0,0.0,0.0},
    {0.906179845938663992797626878299,0.538469310105683091036314420700,0.0,
    -0.538469310105683091036314420700,-0.906179845938663992797626878299,
    0.0,0.0,0.0,0.0,0.0},
    {0.932469514203152027812301554494,0.661209386466264513661399595020,
    0.238619186083196908630501721681,-0.238619186083196908630501721681
    ,-0.661209386466264513661399595020,-0.932469514203152027812301554494,
    0.0,0.0,0.0,0.0},
    {0.949107912342758524526189684048,0.741531185599394439863864773281,
    0.405845151377397166906606412077,0.0,-0.405845151377397166906606412077,
    -0.741531185599394439863864773281,-0.949107912342758524526189684048,
    0.0,0.0,0.0},
    {0.960289856497536231683560868569,0.796666477413626739591553936476,
    0.525532409916328985817739049189,0.183434642495649804939476142360,
    -0.183434642495649804939476142360,-0.525532409916328985817739049189,
    -0.796666477413626739591553936476,-0.960289856497536231683560868569,
    0.0,0.0},
    {0.968160239507626089835576202904,0.836031107326635794299429788070,
    0.613371432700590397308702039341,0.324253423403808929038538014643,0.0,
    -0.324253423403808929038538014643,-0.613371432700590397308702039341,
    -0.836031107326635794299429788070,-0.968160239507626089835576202904,0.0},
    {0.973906528517171720077964012084,0.865063366688984510732096688423,
    0.679409568299024406234327365115,0.433395394129247190799265943166,
    0.148874338981631210884626001130,-0.148874338981631210884626001130,
    -0.433395394129247190799265943166,-0.679409568299024406234327365115,
    -0.865063366688984510732096688423,-0.973906528517171720077964012084}
};
// Arreglo para los pesos
```

antonio carrillo@ciencias.unam.mx 384 Antonio Carrillo Ledesma

```

int j, i;
double x = 0.0;

// Integral en una dimension f(x)= x*x - 3x + 7
for (k = 0; k < P_I; k++)
{
x += PESOS[P_I-2][k] * (PUNTOS[P_I-2][k] *
PUNTOS[P_I-2][k] + 3.0 * PUNTOS[P_I-2][k] + 7.0);
}
printf("El resultado es %2.18lf, debe de ser 14.666667\n",x);

// Integral en una dimension f(x)= sin(x)
x = 0.0;
for (k = 0; k < P_I; k++)
{
x += PESOS[P_I-2][k] * (sinl(PUNTOS[P_I-2][k] ));
}
printf("El resultado es %2.18lf, debe de ser 0.0\n",x);

// Integral en dos dimensiones f(x,y)= (x*x*x - 1)(y - 1)(y - 1)
x = 0.0;
for (i = 0; i < P_I; i++)
{
for (j = 0; j < P_I; j++)
{
x += PESOS[P_I-2][i]*PESOS[P_I-2][j]* ( PUNTOS[P_I-2][j]*
PUNTOS[P_I-2][j]*PUNTOS[P_I-2][j]-1.0)*((PUNTOS[P_I-2][i]-1.0)
*(PUNTOS[P_I-2][i]-1.0)) );
}
}
printf("El resultado es %2.18lf, debe de ser -5.333333\n",x);

// Integral en tres dimensiones f(x,y,z)=(x*x)(y*y - 1)(z*z*z*z - 2)
x = 0.0;
for (i = 0; i < P_I; i++)
{
for (j = 0; j < P_I; j++)
{

```

```

for (k = 0; k < P_I; k++)
{
  x += PESOS[P_I-2][i]*PESOS[P_I-2][j]*PESOS[P_I-2][k]*
  ((PUNTOS[P_I-2][k]*PUNTOS[P_I-2][k])*(PUNTOS[P_I-2][i]*
  PUNTOS[P_I-2][j]-1.0)*(PUNTOS[P_I-2][i]*PUNTOS[P_I-2][i]*
  PUNTOS[P_I-2][i]*PUNTOS[P_I-2][i]-2.0) );
}
}
}

printf("El resultado es %2.18lf, debe de ser 3.2\n",x);
getchar();
}

```

En el caso del lenguaje de programación Python para dos dimensiones disponemos de esta opción usando:

```

import numpy as np
from scipy import integrate
# integra  $xy^2 dx dy$ , en x de 0 a 2 y y en 0 a 1
f = lambda y, x: x*y**2
print(integrate.dblquad(f, 0, 2, 0, 1))

# integra 1 dx dy, en x en 0 a  $\pi/4$  y y en  $\sin(x)$  a  $y \cos(x)$ 
f = lambda y, x: 1
print(integrate.dblquad(f, 0, np.pi/4, np.sin, np.cos))

# integra  $axy dx dy$  para  $a=1$  y 3, en x de 0 a 1 y y en x a  $2-x$ 
f = lambda y, x, a: a*x*y
print(integrate.dblquad(f, 0, 1, lambda x: x, lambda x: 2-x,
args=(1,)))
print(integrate.dblquad(f, 0, 1, lambda x: x, lambda x: 2-x,
args=(3,)))

# integra  $e^{-(x^2+y^2)} dx dy dz$ , en  $-\infty$  a  $\infty$ 
f = lambda x, y: np.exp(-(x ** 2 + y ** 2))
print(integrate.dblquad(f, -np.inf, np.inf, -np.inf, np.inf))

```


En el caso del lenguaje de programación Python para tres dimensiones disponemos de esta opción usando:

```
import numpy as np
from scipy import integrate
# integral xyz dxdydz, en x de 1 a 2, y de 2 a 3 y z de 0 a 1
f = lambda z, y, x: x*y*z
print(integrate.tplquad(f, 1, 2, 2, 3, 0, 1))

# integral xyz dxdydz, en x de 0 a 1, y de 0 a 1-2x y z de 0
a 1-x-2y
f = lambda z, y, x: x*y*z
print(integrate.tplquad(f, 0, 1, 0, lambda x: 1-2*x, 0, lambda
x, y: 1-x-2*y))

# integral axyz dxdydz con a = 1 y 3, en x de 0 a 1, y de 0
a 1 y z de 0 a 1
f = lambda z, y, x, a: a*x*y*z
print(integrate.tplquad(f, 0, 1, 0, 1, 0, 1, args=(1,)))
print(integrate.tplquad(f, 0, 1, 0, 1, 0, 1, args=(3,)))

# integra e-(x2+y2+z2) dxdydz, en -inf a inf
f = lambda x, y, z: np.exp(-(x ** 2 + y ** 2 + z ** 2))
print(integrate.tplquad(f, -np.inf, np.inf, -np.inf, np.inf, -np.inf,
np.inf))
```

9.6 Matrices y Vectores

Una parte fundamental de la implementación computacional de los métodos numéricos de resolución de sistemas algebraicos, es utilizar una forma óptima de almacenar, recuperar y operar las matrices, tal que, facilite los cálculos que involucra la resolución de grandes sistemas de ecuaciones lineales cuya implementación puede ser secuencial o paralela (véase [60]).

El sistema lineal puede ser expresado en la forma matricial $\underline{A}u = \underline{f}$, donde la matriz $\underline{A}$ -que puede ser real o virtual- es de tamaño $n \times n$ con banda b , pero el número total de datos almacenados en ella es a los más $n * b$ números de doble precisión, en el caso de ser simétrica la matriz, el número de datos almacenados es menor a $(n * b)/2$. Además si el problema

que la originó es de coeficientes constantes el número de valores almacenados se reduce drásticamente a sólo el tamaño de la banda b .

En el caso de que el método para la resolución del sistema lineal a usar sea del tipo Factorización LU o Cholesky, la estructura de la matriz cambia, ampliándose el tamaño de la banda de b a $2 * b + 1$ en la factorización, en el caso de usar métodos iterativos tipo CGM o GMRES la matriz se mantiene intacta con una banda b .

Para la resolución del sistema lineal virtual asociada a los métodos de descomposición de dominio, la operación básica que se realiza de manera reiterada, es la multiplicación de una matriz por un vector $\underline{v} = \underline{Cu}$, la cual es necesario realizar de la forma más eficiente posible.

Un factor determinante en la implementación computacional, para que esta resulte eficiente, es la forma de almacenar, recuperar y realizar las operaciones que involucren matrices y vectores, de tal forma que la multiplicación se realice en la menor cantidad de operaciones y que los valores necesarios para realizar dichas operaciones queden en la medida de lo posible contiguos para ser almacenados en el Cache⁸⁸ del procesador.

Almacenamiento en la Memoria RAM La memoria RAM (Random Access Memory) o memoria de acceso aleatorio es un componente físico de nuestro ordenador, generalmente instalado sobre la misma placa base. La memoria RAM es extraíble y se puede ampliar mediante módulos de distintas capacidades.

La función de la memoria RAM es la de cargar los datos e instrucciones que se ejecutan en el procesador. Estas instrucciones y datos provienen del sistema operativo, dispositivos de entrada y salida, de discos duros y todo lo que está instalado en el equipo.

En la memoria RAM se almacenan todos los datos e instrucciones de los programas que se están ejecutando, estas son enviadas desde las unidades

⁸⁸Nótese que la velocidad de acceso a la memoria principal (RAM) es relativamente lenta con respecto al Cache, este generalmente está dividido en subCaches L1 —de menor tamaño y el más rápido—, L2 y hasta L3 —el más lento y de mayor tamaño— los cuales son de tamaño muy reducido con respecto a la RAM.

Por ello, cada vez que las unidades funcionales de la Unidad de Aritmética y Lógica requieren un conjunto de datos para implementar una determinada operación en los registros, solicitan los datos primeramente a los Caches, estos consumen diversa cantidad de ciclos de reloj para entregar el dato si lo tienen —pero siempre el tiempo es menor que solicitarle el dato a la memoria principal—; en caso de no tenerlo, se solicitan a la RAM para ser cargados a los caches y poder implementar la operación solicitada.

de almacenamiento antes de su ejecución. De esta forma podremos tener disponibles todos los programas que ejecutamos. Se llama memoria de acceso aleatorio porque se puede leer y escribir en cualquiera de sus posiciones de memoria sin necesidad de respetar un orden secuencial para su acceso.

La RAM dinámica cuenta con un reloj interno capaz de sincronizar esta con el procesador. De esta forma se mejoran notablemente los tiempos de acceso y la eficiencia de comunicación entre ambos elementos. Actualmente todas nuestras computadoras cuentan con este tipo de memorias operando en ellos. Las principales tipos de memoria son: DDR, DDR2, DDR3, DDR4 y la nueva DDR5. Donde las tasas de transferencia (GB/s) son: DDR (2.1 - 3.2), DDR2 (4.2 - 6.4), DDR3 (8.5 - 14.9), DDR4 (17 - 25.6) y DDR5 (38.4 - 51.2). La característica más importante es que, por ejemplo, en la memoria DDR4 cuatro cores pueden acceder simultáneamente a ella y en la DDR5 serán cinco cores.

Caché L1, L2 y L3 La memoria Caché es otra de las especificaciones importantes de los procesadores, y sirve de manera esencial de la misma manera que la memoria RAM: como almacenamiento temporal de datos. No obstante, dado que la memoria Caché está en el procesador en sí, es mucho más rápida y el procesador puede acceder a ella de manera más eficiente, así que el tamaño de esta memoria puede tener un impacto bastante notable en el rendimiento, especialmente cuando se realizan tareas que demandan un uso intensivo del CPU como en el cómputo de alto desempeño o cómputo científico.

La Caché se divide en diferentes jerarquías de acceso:

- La Caché L1 es el primer sitio donde la CPU buscará información, pero también es la más pequeña y la más rápida, a veces para mayor eficiencia, la Caché L1 se subdivide en L1d (datos) y L1i (instrucciones), actualmente los procesadores modernos en cada core tiene su propio cache de datos e instrucciones.
- La Caché L2 suele ser más grande que la L1 pero es algo más lenta. Sin embargo, por norma general es la que mayor impacto tiene en el rendimiento, este también está incluido en cada core.
- La Caché L3 es mucho más grande que las anteriores, y generalmente se comparte entre todos los núcleos del procesador (a diferencia de las

anteriores, que normalmente van ligadas a cada core). Este tercer nivel es en el que buscará el procesador la información tras no encontrarla en la L1 y L2, por lo que su tiempo de acceso es todavía mayor.

Para poner en contexto la relevancia de la memoria Caché, supongamos que el acceso a los datos de la memoria Caché L1 por el procesador es de dos ciclos de reloj, el acceso a la memoria Caché L2 es de 6 ciclos de reloj, el acceso a la memoria Caché L3 es de 12 ciclos y el acceso a la RAM es de 32 ciclos de reloj.

Además supongamos que la operación suma y resta necesitan de 2 ciclos de reloj para completar la operación una vez que cuente con los datos involucrados en dicha operación, que la multiplicación requiere 4 ciclos de reloj para completar la operación, la división necesita 6 ciclos de reloj para completar la operación y estamos despreciando el tiempo necesario para poner los datos del Caché L1 a los registros del procesador para poder iniciar el cálculo, así también despreciamos el tiempo requerido para sacar el resultado de los registros del procesador al Caché L1.

Esto nos da una idea del número máximo teórico de operaciones básicas que un procesador puede realizar por segundo dependiendo de la velocidad de reloj de la CPU⁸⁹.

Si nosotros necesitamos hacer la multiplicación de una matriz $\underline{\underline{A}}$ es de tamaño $n \times n$ por un vector $\underline{u}$ de tamaño n y guardar el resultado en el vector $\underline{f}$ de tamaño n . Entonces algunos escenarios son posibles:

1. Si el código del programa cabe en el Caché L1 de instrucciones y la matriz $\underline{\underline{A}}$, los vectores $\underline{u}$ y $\underline{f}$ caben íntegramente en el Caché L1 de datos, entonces el procesador estará siendo utilizado de forma óptima al hacer los cálculos pues no tendrá tiempos muertos por espera de datos.
2. Si el código del programa cabe en el Caché L1 de instrucciones y los vectores $\underline{u}$ y $\underline{f}$ caben íntegramente en el Caché L1 de datos pero la matriz $\underline{\underline{A}}$ está dispersa entre los Cachés L1 y L2, entonces el procesador estará teniendo algunos tiempos muertos mientras carga la parte

⁸⁹Por ejemplo en un procesador AMD Ryzen 9 3900X con 12 Cores (2 Threads por Core) por procesador emulando un total de 24 Cores, corre a una frecuencia base de 3,340 MHz, con una frecuencia mínima de 2,200 MHz y máxima de 4,917 Mhz, con Caché L1d de 384 KiB, L1i de 384 KiB, Caché L2 de 6 MiB y Caché L3 de 64 MiB.

que necesita de la matriz del Caché L2 a L1 para hacer los cálculos y utilizado de forma óptima el procesador mientras no salga del Caché L1.

3. Si el código del programa cabe en el Caché L1 de instrucciones y los vectores $\underline{u}$ y $\underline{f}$ caben íntegramente en el Caché L1 de datos pero la matriz $\underline{A}$ está dispersa entre los Cachés L1, L2 y L3, entonces el procesador estará teniendo muchos tiempos muertos mientras carga la parte que necesita de la matriz del Caché L3 y L2 a L1 para hacer los cálculos resultando en mediana eficiencia en el uso del procesador.
4. Si el código del programa cabe en el Caché L1 de instrucciones y los vectores $\underline{u}$ y $\underline{f}$ caben íntegramente en los Cachés L3, L2 y L1 pero los datos de la matriz $\underline{A}$ está dispersa entre la RAM y los Cachés L3, L2 y L1, entonces el procesador estará teniendo un exceso de tiempos muertos mientras carga la parte que necesita de la matriz de la RAM a los Cachés L3, L2 y L1 para hacer los cálculos resultando en una gran pérdida de eficiencia en el uso del procesador.

Además, debemos recordar que la computadora moderna nunca dedica el cien por ciento del CPU a un solo programa, ya que los equipos son multitarea⁹⁰ y multiusuario⁹¹ por lo que la conmutación de procesos (que se realiza cada cierta cantidad de milisegundos) degrada aún más la eficiencia computacional de los procesos que demandan un uso intensivo de CPU⁹².

⁹⁰Cuentan con la capacidad para ejecutar varios procesos simultáneamente en uno o más procesadores, para ello necesitan hacer uso de la conmutación de tareas, es decir, cada cierto tiempo detiene el programa que está corriendo y guardan sus datos, para poder cargar en memoria otro programa y sus respectivos datos y así reiniciar su ejecución por un período determinado de tiempo, una vez concluido su tiempo de ejecución se reinicia la conmutación de tareas con otro proceso.

⁹¹Se refiere a todos aquellos sistemas operativos que permiten el empleo de sus procesamiento y servicios al mismo tiempo. Así, el sistema operativo cuenta con la capacidad de satisfacer las necesidades de varios usuarios al mismo tiempo, siendo capaz de gestionar y compartir sus recursos en función del número de usuarios que estén conectados a la vez.

⁹²Actualmente existen una gran cantidad de distribuciones de GNU/Linux que vienen muy optimizadas intentando conseguir la mejor desenvolvura de su arquitectura y configuraciones de serie. En el caso de la configuración por omisión de Debian GNU/Linux y Ubuntu, están pensadas para que sean lo más robusta posible y que se use en todas las circunstancias imaginables, por ello están optimizadas de forma muy conservadora

Dado que la multiplicación de una matriz $\underline{\underline{C}}$ por un vector $\underline{u}$, dejando el resultado en $\underline{v}$ se realiza mediante el algoritmo.

```

for (i=0; i<ren; i++)
{
    s = 0.0;
    for (j=0; j < col; j++)
    {
        s += C[i][j]*u[j];
    }
    v[i] = s;
}

```

Para lograr una eficiente implementación del algoritmo anterior, es necesario que el gran volumen de datos desplazados de la memoria al Cache y viceversa sea mínimo. Por ello, los datos se deben agrupar para que la operación más usada -en este caso multiplicación matriz por vector- se realice con la menor solicitud de datos a la memoria principal, si los datos usados -renglón de la matriz- se ponen contiguos minimizará los accesos a la memoria principal, pues es más probable que estos estarán contiguos en el Cache al momento de realizar la multiplicación.

Por ejemplo, en el caso de matrices bandadas de tamaño de banda b , el algoritmo anterior se simplifica a:.

```

for (i=0; i<ren; i++)
{
    s= 0.0;
    for (k=0; k < ban; k++)
    {
        if ((Ind[k] + i) >= 0 && (Ind[k]+i) < ren)
            s += Dat[i][k]*u[Ind[k]+i];
    }
}

```

para tener un equilibrio entre eficiencia y consumo de energía. Pero es posible agregar uno o más Kernels GNU/Linux generados por terceros que contenga las optimizaciones necesarias para hacer más eficiente y competitivo en cuestiones de gestión y ahorro de recursos del sistema.

Hay varias opciones del Kernel GNU/Linux optimizado ([Liquorix](#) viene optimizado para multimedia y Juegos, por otro lado [XanMod](#) tiene uno para propósito general, otro aplicaciones críticas en tiempo real y otro más para cálculos intensivos) de las últimas versiones estable del Kernel.

```

    }
    v[i]=s;
}
    
```

Si, la solicitud de memoria para $Dat[i]$ se hace de tal forma que los datos del renglón estén continuos -son b números de punto flotante-, esto minimizará los accesos a la memoria principal en cada una de las operaciones involucradas en el producto, como se explica en las siguientes secciones.

Matrices Bandadas En el caso de las matrices bandadas de banda b - sin pérdida de generalidad y para propósitos de ejemplificación se supone pentadiagonal- típicamente tiene la siguiente forma:

$$\underline{\underline{A}} = \begin{bmatrix} a_1 & b_1 & & & c_1 & & & & \\ & d_2 & a_2 & b_2 & & c_2 & & & \\ & & d_3 & a_3 & b_3 & & c_3 & & \\ & & & d_4 & a_4 & b_4 & & c_4 & \\ e_5 & & & & d_5 & a_5 & b_5 & & c_5 \\ & e_6 & & & & d_6 & a_6 & b_6 & \\ & & e_7 & & & & d_7 & a_7 & b_7 \\ & & & e_8 & & & & d_8 & a_8 & b_8 \\ & & & & e_9 & & & & d_9 & a_9 \end{bmatrix} \quad (9.28)$$

la cual puede ser almacenada usando el algoritmo (véase [60]) Compressed Diagonal Storage (CDS), optimizado para ser usado en C++, para su almacenamiento y posterior recuperación. Para este ejemplo en particular, se hará uso de un vector de índices

$$\underline{\underline{Ind}} = [-5, -1, 0, +1, +5] \quad (9.29)$$

y los datos serán almacenados usando la estructura

$$\underline{\underline{Dat}} = \begin{bmatrix} 0 & 0 & a_1 & b_1 & c_1 \\ 0 & d_2 & a_2 & b_2 & c_2 \\ 0 & d_3 & a_3 & b_3 & c_3 \\ 0 & d_4 & a_4 & b_4 & c_4 \\ e_5 & d_5 & a_5 & b_5 & c_5 \\ e_6 & d_6 & a_6 & b_6 & 0 \\ e_7 & d_7 & a_7 & b_7 & 0 \\ e_8 & d_8 & a_8 & b_8 & 0 \\ e_9 & d_9 & a_9 & 0 & 0 \end{bmatrix} \quad (9.30)$$

de tal forma que la matriz $\underline{\underline{A}}$ puede ser reconstruida de forma eficiente. Para obtener el valor $A_{i,j}$, calculo $ind = j - i$, si el valor ind esta en la lista de índices $\underline{Ind}$ -supóngase en la columna k -, entonces $A_{i,j} = Dat_{ik}$, en otro caso $A_{i,j} = 0$.

Casos Particulares de la Matriz Bandada $\underline{\underline{A}}$ Básicamente dos casos particulares surgen en el tratamiento de ecuaciones diferenciales parciales: El primer caso es cuando el operador diferencial parcial es simétrico y el otro, en el que los coeficientes del operador sean constantes.

Para el primer caso, al ser la matriz simétrica, sólo es necesario almacenar la parte con índices mayores o iguales a cero, de tal forma que se buscara el índice que satisfaga $ind = |j - i|$, reduciendo el tamaño de la banda a $b/2$ en la matriz $\underline{\underline{A}}$.

Para el segundo caso, al tener coeficientes constantes el operador diferencial, los valores de los renglones dentro de cada columna de la matriz son iguales, y sólo es necesario almacenarlos una sola vez, reduciendo drásticamente el tamaño de la matriz de datos.

Implementación de Matrices Bandadas Orientada a Objetos en C++ Una forma de implementar la matriz bandada $\underline{\underline{A}}$ de banda b en C++ es mediante:

```
// Creacion
double **Dat;
Dat = new double*[ren];
for (i = 0; i < ren; i++) Dat[i] = new double[b];
int *Ind;
Ind = new int[b];
// Inicializacion
for (i = 0; i < ren; i++)
    for (j = 0; j < b; j++) Dat[i][j] = 0.0;
for (i = 0; i < b; i++) Ind[i] = 0;
```

Matrices Dispersas Las matrices dispersas de a lo más b valores distintos por renglón -sin pérdida de generalidad y para propósitos de ejemplificación se supone $b = 3$ - que surgen en métodos de descomposición de dominio para

almacenar algunas matrices, típicamente tienen la siguiente forma

$$\underline{\underline{A}} = \begin{bmatrix} a_1 & & & b_1 & & c_1 \\ & a_2 & & b_2 & & c_2 \\ & & & a_3 & & b_3 & c_3 \\ a_4 & & b_4 & & & & \\ & a_5 & & b_5 & & & c_5 \\ a_6 & b_6 & c_6 & & & & \\ & & & & a_7 & b_7 & c_7 \\ & & & a_8 & & b_8 & c_8 \\ & & & & & a_9 & b_9 \end{bmatrix} \quad (9.31)$$

la cual puede ser almacenada usando el algoritmo (véase [60]) Jagged Diagonal Storage (JDC), optimizado para ser usado en C++. Para este ejemplo en particular, se hará uso de una matriz de índices

$$\underline{\underline{Ind}} = \begin{bmatrix} 1 & 6 & 9 \\ 2 & 5 & 8 \\ 5 & 8 & 9 \\ 1 & 4 & 0 \\ 3 & 6 & 9 \\ 1 & 2 & 3 \\ 7 & 8 & 9 \\ 4 & 7 & 8 \\ 7 & 8 & 0 \end{bmatrix} \quad (9.32)$$

y los datos serán almacenados usando la estructura

$$\underline{\underline{Dat}} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \\ a_4 & b_4 & 0 \\ a_5 & b_5 & c_5 \\ a_6 & b_6 & c_6 \\ a_7 & b_7 & c_7 \\ a_8 & b_8 & c_8 \\ a_9 & b_9 & 0 \end{bmatrix} \quad (9.33)$$

de tal forma que la matriz $\underline{\underline{A}}$ puede ser reconstruida de forma eficiente. Para obtener el valor $A_{i,j}$, busco el valor j en la lista de índices $\underline{\underline{Ind}}$ dentro del renglón i , si lo encuentro en la posición k , entonces $A_{i,j} = \underline{\underline{Dat}}_{ik}$, en otro caso $A_{i,j} = 0$.

Casos Particulares de la Matriz Dispersa $\underline{\underline{A}}$ Si la matriz $\underline{\underline{A}}$, que al ser almacenada, se observa que existen a lo más r diferentes renglones con valores distintos de los n con que cuenta la matriz y si $r \ll n$, entonces es posible sólo guardar los r renglones distintos y llevar un arreglo que contenga la referencia al renglón almacenado.

Implementación de Matrices Dispersas Orientada a Objetos en C++ Una forma de implementar la matriz bandada $\underline{\underline{A}}$ de banda b en C++ es mediante:

```
// Creacion
double **Dat;
Dat = new double*[ren];
for (i = 0; i < ren; i++) Dat[i] = new double[b];
int **Ind;
Ind = new int*[ren];
for (i = 0; i < ren; i++) Ind[i] = new int[b];
// Inicializacion
for (i = 0; i < ren; i++)
    for (j = 0; j < b; j++) Dat[i][j] = 0.0, Ind[i][j] = -1;
```

Multiplicación Matriz-Vector Los métodos de descomposición de dominio requieren por un lado la resolución de al menos un sistema lineal y por el otro lado requieren realizar la operación de multiplicación de matriz por vector, i.e. $\underline{\underline{Cu}}$ de la forma más eficiente posible, por ello los datos se almacenan de tal forma que la multiplicación se realice en la menor cantidad de operaciones.

Dado que la multiplicación de una matriz $\underline{\underline{C}}$ por un vector $\underline{u}$, dejando el resultado en $\underline{v}$ se realiza mediante el algoritmo:

```
for (i=0; i<ren; i++)
{
    s = 0.0;
    for (j=0; j < col; j++)
    {
        s += C[i][j]*u[j];
    }
    v[i] = s;
```

```
}
```

En el caso de matrices bandadas, se simplifica a:

```
for (i=0; i<ren; i++)
{
    s= 0.0;
    for (k=0; k < ban; k++)
    {
        if ((Ind[k] + i) >= 0 && (Ind[k]+i) < ren)
            s += Dat[i][k]*u[Ind[k]+i];
    }
    v[i]=s;
}
```

De forma similar, en el caso de matrices dispersas, se simplifica a:

```
for (i=0; i<ren; i++)
{
    s = 0.0, k = 0
    while (Ind[i][k] != -1)
    {
        s += Dat[i][k]*u[Ind[i][k]];
        k++;
        if (k >= b) break;
    }
    v[i] = s;
}
```

De esta forma, al tomar en cuenta la operación de multiplicación de una matriz por un vector, donde el renglón de la matriz involucrado en la multiplicación queda generalmente en una región contigua del Cache, se hace óptima la operación de multiplicación de matriz por vector.

Ejemplo 2 *Supongamos que la factura del teléfono e internet del mes pasado ascendía a un total de \$390 por un consumo de 80 minutos de telefonía y 50 GB de internet mientras que la de este mes asciende a \$250 por un consumo de 55 minutos de telefonía y 30 GB de internet ¿Cual es el costo de cada minuto de llamadas y de cada GB de internet?*

El sistema de ecuaciones para resolver el problema es:

$$80x + 50y = 390$$

$$55x + 30y = 250$$

resuelve el sistema de ecuaciones y graficar:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import solve
# Valores
A00, A01 = 80, 50
A10, A11 = 55, 30
b0, b1 = 390, 250
#Declaración de matriz y vector
A = np.array(
    [
        [A00, A01],
        [A10, A11]
    ]
)
print(A)
b = np.array([b0, b1]).reshape((2, 1))
print(b)
x=solve(A, b)
print('La solucion al sistema es x=', x[0], 'y=', x[1])
plt.plot(x[0],x[1],marker="o")
# Función lineal.
def f1(x):
    return ((-A00/A01)*(x) + (b0/A01))
def f2(x):
    return ((-A10/A11)*(x) + (b1/A11))
# Valores del eje X que toma el gráfico.
x = range(0, 5)
# Graficar ambas funciones.
plt.plot(x, [f1(i) for i in x])
plt.plot(x, [f2(i) for i in x])
plt.show()
```

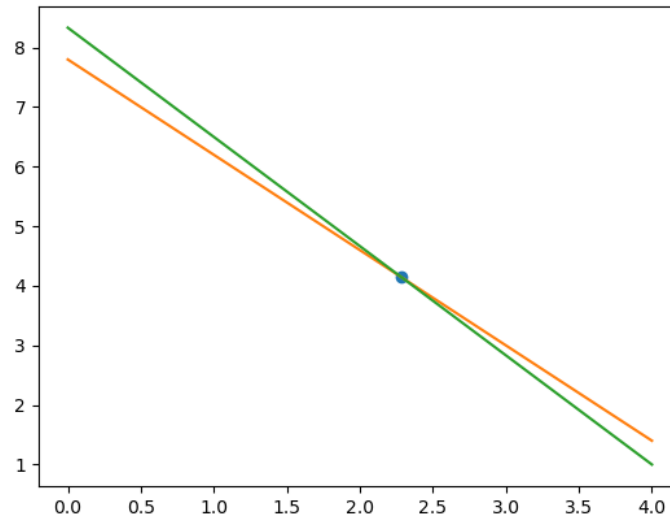


Figura 13: La solución del sistema lineal es: $x = 2.28571429$ e $y = 4.14285713$

Cálculo SVD y Pseudoinversa Dada la descomposición singular de una matriz $\underline{\underline{A}}$,

$$\underline{\underline{A}} = \underline{\underline{U}}\underline{\underline{\Sigma}}\underline{\underline{V}}^*$$

la pseudoinversa de Moore-Penrose viene dada por

$$\underline{\underline{A}}^+ = \underline{\underline{V}}\underline{\underline{\Sigma}}^+\underline{\underline{U}}^*$$

Descomposición de Valores Singulares (SVD) la descomposición en valores singulares de una matriz es una especie de cambio de coordenadas que simplifica la matriz, una generalización de la diagonalización.

Diagonalización de matrices si una matriz cuadrada $\underline{\underline{A}}$ es diagonalizable, entonces existe una matriz $\underline{\underline{P}}$ tal que

$$\underline{\underline{A}} = \underline{\underline{P}}\underline{\underline{D}}\underline{\underline{P}}^{-1}$$

donde la matriz $\underline{\underline{D}}$ es diagonal. Podrías pensar en $\underline{\underline{P}}$ como un cambio de coordenadas que hace que la acción de $\underline{\underline{A}}$ sea lo más simple posible. Los elementos en la diagonal de $\underline{\underline{D}}$ son los valores propios de $\underline{\underline{A}}$ y las columnas de $\underline{\underline{P}}$ son los vectores propios correspondientes.

Lamentablemente no todas las matrices se pueden diagonalizar. La descomposición de valores singulares es una forma de hacer algo así como la diagonalización de cualquier matriz, incluso las que no son cuadradas.

Generalización a SVD la descomposición de valores singulares generaliza la diagonalización. La matriz $\underline{\underline{\Sigma}}$ en SVD es análoga a $\underline{\underline{D}}$ en diagonalización. $\underline{\underline{\Sigma}}$ es diagonal, aunque puede que no sea cuadrada. Las matrices a cada lado de $\underline{\underline{\Sigma}}$ son análogas a la matriz $\underline{\underline{P}}$ en diagonalización, aunque ahora hay dos matrices diferentes y no son necesariamente inversas entre sí. Las matrices $\underline{\underline{U}}$ y $\underline{\underline{V}}$ son cuadradas, pero no necesariamente de la misma dimensión.

Los elementos a lo largo de la diagonal de $\underline{\underline{\Sigma}}$ no son necesariamente valores propios sino valores singulares, que son una generalización de los valores propios. De manera similar, las columnas de $\underline{\underline{U}}$ y $\underline{\underline{V}}$ no son necesariamente vectores propios sino vectores singulares izquierdos y vectores singulares derechos, respectivamente.

El superíndice de estrella indica transposición conjugada. Si una matriz tiene todos los componentes reales, entonces la transpuesta conjugada es solo la transpuesta. Pero si la matriz tiene entradas complejas, se toma el conjugado y se transpone cada entrada.

Las matrices $\underline{\underline{U}}$ y $\underline{\underline{V}}$ son unitarias. Una matriz $\underline{\underline{M}}$ es unitaria si su inversa es su transpuesta conjugada, es decir,

$$\underline{\underline{M}}^* \underline{\underline{M}} = \underline{\underline{M}} \underline{\underline{M}}^* = \underline{\underline{I}}.$$

Pseudoinversa y SVD la pseudoinversa (Moore-Penrose) de una matriz generaliza la noción de inversa, algo así como la forma en que SVD generalizó la diagonalización. No todas las matrices tienen una inversa, pero todas las matrices tienen una pseudoinversa, incluso las matrices no cuadradas.

Calcular la pseudoinversa a partir del SVD es sencillo. Si

$$\underline{\underline{A}} = \underline{\underline{U}} \underline{\underline{\Sigma}} \underline{\underline{V}}^*$$

entonces

$$\underline{\underline{A}}^+ = \underline{\underline{V}} \underline{\underline{\Sigma}}^+ \underline{\underline{U}}^*$$

donde $\underline{\underline{\Sigma}}^+$ se forma a partir de $\underline{\underline{\Sigma}}$ tomando el recíproco de todos los elementos distintos de cero, dejando todos los ceros solos y haciendo que la matriz tenga la forma correcta: si $\underline{\underline{\Sigma}}$ es una matriz de m por n , entonces $\underline{\underline{\Sigma}}^+$ debe ser una matriz de n por m .

Calcular SVD en Python Sea

$$\underline{\underline{A}} = \begin{bmatrix} 2 & -1 & 0 \\ 4 & 3 & -2 \end{bmatrix}$$

la descomposición en valores singulares de $\underline{\underline{A}}$ es

$$\begin{bmatrix} \frac{1}{\sqrt{26}} & -\frac{5}{\sqrt{26}} \\ \frac{5}{\sqrt{26}} & \frac{1}{\sqrt{26}} \end{bmatrix} \begin{bmatrix} \sqrt{30} & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix} \begin{bmatrix} \frac{11}{\sqrt{195}} & \frac{7}{\sqrt{195}} & -\sqrt{\frac{5}{39}} \\ -\frac{3}{\sqrt{26}} & 2\sqrt{\frac{2}{13}} & -\frac{1}{\sqrt{26}} \\ \frac{1}{\sqrt{30}} & \sqrt{\frac{2}{15}} & \sqrt{\frac{5}{6}} \end{bmatrix}$$

A continuación calculamos la descomposición del valor singular en Python (NumPy).

```
import numpy as np
A = np.matrix([[2, -1, 0],[4,3,-2]])
print(A)

[[ 2 -1 0]
 [ 4 3 -2]]

u, s, vt = np.linalg.svd(A, full_matrices=True)
print(u)
print(s)
print(vt)

[[ 0.19611614  0.98058068]
 [ 0.98058068 -0.19611614]]
[5.47722558 2. ]
[[ 0.78772636  0.50128041 -0.35805744]
 [ 0.58834841 -0.78446454  0.19611614]
 [ 0.18257419  0.36514837  0.91287093]]
```

Tengamos en cuenta que *np.linalg.svd* devuelve la transpuesta de $\underline{\underline{V}}$, no la $\underline{\underline{V}}$ en la definición de descomposición de valores singulares.

Además, el objeto *s* no es la matriz diagonal $\underline{\underline{\Sigma}}$ sino un vector que contiene sólo los elementos diagonales, es decir, sólo los valores singulares. Esto puede ahorrar mucho espacio si la matriz es grande. El método *NumPy svd* tiene otras opciones relacionadas con la eficiencia en las que no entraré aquí.

Podemos verificar que el SVD es correcto convirtiendo *s* nuevamente en una matriz

```
ss = np.matrix([s[0], 0, 0], [0, s[1], 0])
print(ss)
```

```
[[5.47722558 0. 0. ]
 [0. 2. 0. ]]
```

y multiplicando los componentes

```
a = u*ss*vt
print(a)
```

```
[[ 2.00000000e+00 -1.00000000e+00 -4.05114794e-16]
 [ 4.00000000e+00 3.00000000e+00 -2.00000000e+00]]
```

Esto devuelve la matriz $\underline{\underline{A}}$, con precisión de punto flotante. Dado que Python realiza cálculos de punto flotante, no cálculos simbólicos, el cero en $\underline{\underline{A}}$ se convierte en $-4.05114794e - 16$.

Calcular Pseudoinverso en Python el pseudoinverso se puede calcular en NumPy con *np.linalg.pinv*

```
si = np.linalg.pinv(a)
print(si)
```

```
[[ 0.31666667 0.08333333]
 [-0.36666667 0.16666667]
 [ 0.08333333 -0.08333333]]
```


esto devuelve el resultado hasta precisión de punto flotante, para ello hacemos:

```
print(a*si)

[[ 1.00000000e+00 1.54033727e-16]
 [-9.25185854e-18 1.00000000e+00]]
```

9.7 Resolución de Sistemas Lineales

Diez Sorpresas del Álgebra Lineal Numérica aquí hay diez cosas sobre el álgebra lineal numérica que pueden resultarle sorprendentes si no está familiarizado con el campo:

- El álgebra lineal numérica aplica matemáticas muy avanzadas para resolver problemas que se pueden plantear con matemáticas de secundaria.
- Las aplicaciones prácticas a menudo requieren resolver enormes sistemas de ecuaciones, millones o incluso miles de millones de variables.
- El corazón de Google es un enorme problema de álgebra lineal. PageRank es esencialmente un problema de valores propios.
- La eficiencia de la resolución de sistemas de ecuaciones muy grandes se ha beneficiado al menos tanto de los avances en los algoritmos como de la ley de Moore.
- Muchos problemas prácticos (optimización, ecuaciones diferenciales, procesamiento de señales, etc.) se reducen a resolver sistemas lineales, incluso cuando los problemas originales no son lineales. El Software de elementos finitos, diferencias finitas y volumen finito por ejemplo, dedica casi todo su tiempo a resolver ecuaciones lineales.
- A veces, un sistema de un millón de ecuaciones se puede resolver en una PC común y corriente en menos de un milisegundo, dependiendo de la estructura de las ecuaciones.
- Los métodos iterativos, métodos que en teoría requieren un número infinito de pasos para resolver un problema, suelen ser más rápidos y precisos que los métodos directos, métodos que en teoría producen una respuesta exacta en un número finito de pasos.
- Hay muchos teoremas que limitan el error en soluciones producidas en computadoras reales. Es decir, los teoremas no sólo limitan el error de cálculos hipotéticos realizados en aritmética exacta, sino que limitan el error de la aritmética realizada en aritmética de punto flotante en Hardware de computadora.

- Casi nunca es necesario calcular la inversa de una matriz.
- Existe un Software notablemente maduro para álgebra lineal numérica. Personas brillantes han trabajado en este Software durante muchos años.

Los modelos matemáticos de muchos sistemas en Ciencia e Ingeniería y en particular una gran cantidad de sistemas continuos geofísicos requieren el procesamiento de sistemas algebraicos de gran escala. En este trabajo se muestra como proceder para transformar un problema de ecuaciones diferenciales parciales en un sistema algebraico virtual de ecuaciones lineales; y así, poder hallar la solución a dicho problema al resolver el sistema lineal asociado al esquema DVS. La solución de este sistema virtual, involucra la solución acoplada de muchos sistemas lineales locales -uno por cada subdominio-, cada uno de estos sistemas lineales puede ser expresado en la siguiente forma matricial

$$\underline{\underline{A}}u = \underline{f} \quad (9.34)$$

donde la matriz $\underline{\underline{A}}$ es de tamaño $n \times n$ y generalmente bandada, cuyo tamaño de banda es b .

Los métodos de resolución del sistema algebraico de ecuaciones $\underline{\underline{A}}u = \underline{f}$ se clasifican en dos grandes grupos (véase [61]): los métodos directos y los métodos iterativos. En los métodos directos la solución $\underline{u}$ se obtiene en un número fijo de pasos y sólo están sujetos a los errores de redondeo. En los métodos iterativos, se realizan iteraciones para aproximarse a la solución $\underline{u}$ aprovechando las características propias de la matriz $\underline{\underline{A}}$, tratando de usar un menor número de pasos que en un método directo (véase [57], [58], [59] y [61]).

Por lo general, es conveniente usar librerías⁹³ para implementar de forma eficiente a los vectores, matrices -bandadas y dispersas- y resolver los sistemas lineales locales asociados a los métodos de resolución de ecuaciones diferenciales parciales.

⁹³Algunas de las librerías más usadas para resolver sistemas lineales usando matrices bandadas y dispersas son PETCs, HYPRE, ATLAS, LAPACK++, LAPACK, EISPACK, LINPACK, BLAS, entre muchas otras alternativas, tanto para implementaciones secuenciales como paralelas y más recientemente para hacer uso de los procesadores CUDA en las GPU de nVidia.

Métodos Directos En los métodos directos (véase [57] y [60]), la solución $\underline{u}$ se obtiene en un número fijo de pasos y sólo están sujetos a errores de redondeo. Entre los métodos más importantes se puede considerar: Factorización LU -para matrices simétricas y no simétricas- y Factorización Cholesky -para matrices simétricas-. En todos los casos la matriz original $\underline{A}$ es modificada y en caso de usar la Factorización LU el tamaño de la banda b crece a $2b + 1$ si la factorización se realiza en la misma matriz.

Factorización LU Sea $\underline{U}$ una matriz triangular superior obtenida de $\underline{A}$ por eliminación bandada. Entonces $\underline{U} = \underline{L}^{-1}\underline{A}$, donde $\underline{L}$ es una matriz triangular inferior con unos en la diagonal. Las entradas de $\underline{L}^{-1}$ pueden obtenerse de los coeficientes $\underline{L}_{ij}$ y pueden ser almacenados estrictamente en las entradas de la diagonal inferior de $\underline{A}$ ya que estas ya fueron eliminadas. Esto proporciona una Factorización $\underline{LU}$ de $\underline{A}$ en la misma matriz $\underline{A}$ ahorrando espacio de memoria, donde el ancho de banda cambia de b a $2b + 1$.

En el algoritmo de Factorización LU, se toma como datos de entrada del sistema $\underline{Au} = \underline{f}$, a la matriz $\underline{A}$, la cual será factorizada en la misma matriz, esta contendrá a las matrices $\underline{L}$ y $\underline{U}$ producto de la factorización, quedando el método numérico esquemáticamente como:

$$\begin{aligned}
 &A_{ii} = 1, \text{ para } i = 1, \dots, N \\
 &\text{Para } J = 1, 2, \dots, N \{ \\
 &\quad \text{Para } i = 1, 2, \dots, j \\
 &\quad \quad A_{ij} = A_{ij} - \sum_{k=1}^{i-1} A_{ik}A_{kj} \\
 &\quad \text{Para } i = j + 1, j + 2, \dots, N \\
 &\quad \quad A_{ij} = \frac{1}{A_{jj}} \left(A_{ij} - \sum_{k=1}^{j-1} A_{ik}A_{kj} \right) \\
 &\quad \}
 \end{aligned} \tag{9.35}$$

El problema original $\underline{Au} = \underline{f}$ se escribe como $\underline{LUu} = \underline{f}$, donde la búsqueda de la solución $\underline{u}$ se reduce a la solución sucesiva de los sistemas lineales triangulares

$$\underline{Ly} = \underline{f} \quad \text{y} \quad \underline{Uu} = \underline{y}. \tag{9.36}$$

i.e.

$$\underline{L}y = \underline{f} \Leftrightarrow \begin{cases} y_1 = f_1/l_{11} \\ y_i = \frac{1}{A_{ii}} \left(f_i - \sum_{j=1}^{i-1} A_{ij}y_j \right) \text{ para toda } i = 1, \dots, n \end{cases} \quad (9.37)$$

y

$$\underline{U}u = \underline{y} \Leftrightarrow \begin{cases} x_n = y_n/u_{nn} \\ x_i = \frac{1}{A_{ii}} \left(y_i - \sum_{j=i+1}^n A_{ij}x_j \right) \text{ para toda } i = n-1, \dots, 1 \end{cases} \quad (9.38)$$

La descomposición $\underline{LU}$ requiere $N^3/3$ operaciones aritméticas para la matriz llena, pero sólo Nb^2 operaciones aritméticas para la matriz con un ancho de banda de b siendo esto más económico computacionalmente.

Factorización Cholesky Cuando la matriz es simétrica y definida positiva, se obtiene la descomposición $\underline{LU}$ de la matriz $\underline{A} = \underline{LDU} = \underline{LDL}^T$ donde $\underline{D} = \text{diag}(\underline{U})$ es la diagonal con entradas positivas.

En el algoritmo de Factorización Cholesky, se toma como datos de entrada del sistema $\underline{Au} = \underline{f}$, a la matriz $\underline{A}$, la cual será factorizada en la misma matriz y contendrá a las matrices $\underline{L}$ y $\underline{L}^T$ producto de la factorización, quedando el método numérico esquemáticamente como:

$$\begin{aligned} &\text{para } i = 1, 2, \dots, n \text{ y } j = i + 1, \dots, n \\ &A_{ii} = \sqrt{\left(A_{ii} - \sum_{k=1}^{i-1} A_{ik}^2 \right)} \\ &A_{ji} = \left(A_{ji} - \sum_{k=1}^{i-1} A_{jk}A_{ik} \right) / A_{ii} \end{aligned} \quad (9.39)$$

El problema original $\underline{Au} = \underline{f}$ se escribe como $\underline{LL}^T \underline{u} = \underline{b}$, donde la búsqueda de la solución $\underline{u}$ se reduce a la solución sucesiva de los sistemas lineales triangulares

$$\underline{L}y = \underline{f} \quad \text{y} \quad \underline{L}^T \underline{u} = \underline{y} \quad (9.40)$$

usando la formulación equivalente dada por las Ec.(9.37) y (9.38) para la descomposición LU.

La mayor ventaja de esta descomposición es que, en el caso en que es aplicable, el costo de cómputo es sustancialmente reducido, ya que requiere de $N^3/6$ multiplicaciones y $N^3/6$ sumas.

Métodos Iterativos En los métodos iterativos, se realizan iteraciones para aproximarse a la solución $\underline{u}$ aprovechando las características propias de la matriz $\underline{\underline{A}}$, tratando de usar un menor número de pasos que en un método directo (véase [57] y [60]).

En los métodos iterativos tales como Jacobi, Gauss-Seidel y de Relajación Sucesiva (SOR) en el cual se resuelve el sistema lineal

$$\underline{\underline{A}}\underline{u} = \underline{f} \quad (9.41)$$

comienza con una aproximación inicial $\underline{u}^0$ a la solución $\underline{u}$ y genera una sucesión de vectores $\{\underline{u}^k\}_{k=1}^{\infty}$ que converge a $\underline{u}$. Los métodos iterativos traen consigo un proceso que convierte el sistema $\underline{\underline{A}}\underline{u} = \underline{f}$ en otro equivalente mediante la iteración de punto fijo de la forma $\underline{u} = \underline{T}\underline{u} + \underline{c}$ para alguna matriz fija $\underline{T}$ y un vector $\underline{c}$. Luego de seleccionar el vector inicial $\underline{u}^0$ la sucesión de los vectores de la solución aproximada se genera calculando

$$\underline{u}^k = \underline{T}\underline{u}^{k-1} + \underline{c} \quad \forall k = 1, 2, 3, \dots \quad (9.42)$$

La convergencia a la solución la garantiza el siguiente teorema (véase [61]).

Teorema 3 Si $\|\underline{T}\| < 1$, entonces el sistema lineal $\underline{u} = \underline{T}\underline{u} + \underline{c}$ tiene una solución única $\underline{u}^*$ y las iteraciones $\underline{u}^k$ definidas por la fórmula $\underline{u}^k = \underline{T}\underline{u}^{k-1} + \underline{c} \quad \forall k = 1, 2, 3, \dots$ convergen hacia la solución exacta $\underline{u}^*$ para cualquier aproximación inicial $\underline{u}^0$.

Nótese que, mientras menor sea la norma de la matriz $\underline{T}$, más rápida es la convergencia, en el caso cuando $\|\underline{T}\|$ es menor que uno, pero cercano a uno, la convergencia es lenta y el número de iteraciones necesario para disminuir el error depende significativamente del error inicial. En este caso, es deseable proponer al vector inicial $\underline{u}^0$ de forma tal que sea mínimo el error inicial. Sin embargo, la elección de dicho vector no tiene importancia si la $\|\underline{T}\|$ es pequeña, ya que la convergencia es rápida.

Como es conocido, la velocidad de convergencia de los métodos iterativos dependen de las propiedades espectrales de la matriz de coeficientes del sistema de ecuaciones, cuando el operador diferencial $\mathcal{L}$ de la ecuación del problema a resolver es auto adjunto se obtiene una matriz simétrica y positiva definida y el número de condicionamiento de la matriz $\underline{\underline{A}}$, es por definición

$$\text{cond}(\underline{\underline{A}}) = \frac{\lambda_{\max}}{\lambda_{\min}} \geq 1 \quad (9.43)$$

donde $\lambda_{\max}$ y $\lambda_{\min}$ es el máximo y mínimo de los eigen valores de la matriz $\underline{\underline{A}}$. Si el número de condicionamiento es cercano a 1 los métodos numéricos al solucionar el problema convergerá en pocas iteraciones, en caso contrario se requerirán muchas iteraciones.

Frecuentemente al usar el método de Elemento Finito, Diferencias Finitas, entre otros, se tiene una velocidad de convergencia de $O\left(\frac{1}{h^2}\right)$ y en el caso de métodos de descomposición de dominio sin preconditionar se tiene una velocidad de convergencia de $O\left(\frac{1}{h}\right)$, donde h es la máxima distancia de separación entre nodos continuos de la partición, es decir, que poseen una pobre velocidad de convergencia cuando $h \rightarrow 0$ (véase [56], [57] y [61]).

Los métodos Jacobi, Gauss-Seidel y de Relajación Sucesiva (SOR) son usualmente menos eficientes que los métodos discutidos en el resto de esta sección basados en el espacio de Krylov (véase [66] y [60]). Estos métodos minimizan, en la k -ésima iteración alguna medida de error sobre el espacio afín $\underline{x}_0 + \mathcal{K}_k$, donde $\underline{x}_0$ es la iteración inicial y $\mathcal{K}_k$ es el k -ésimo subespacio de Krylov

$$\mathcal{K}_k = \text{Generado} \{ \underline{r}_0, \underline{A}\underline{r}_0, \dots, \underline{A}^{k-1}\underline{r}_0 \} \text{ para } k \geq 1. \quad (9.44)$$

El residual es $\underline{r} = \underline{b} - \underline{A}\underline{x}$, tal $\{\underline{r}_k\}_{k \geq 0}$ denota la sucesión de residuales

$$\underline{r}_k = \underline{b} - \underline{A}\underline{x}_k. \quad (9.45)$$

Entre los métodos más usados definidos en el espacio de Krylov para el tipo de problemas tratados en el presente trabajo se puede considerar: Método de Gradiente Conjugado -para matrices simétricas- y GMRES -para matrices no simétricas-.

Método de Gradiente Conjugado Si la matriz generada por la discretización es simétrica $\underline{\underline{A}} = \underline{\underline{A}}^T$ y definida positiva $-\underline{u}^T \underline{\underline{A}} \underline{u} > 0$ para todo

$\underline{u} \neq 0$ -, entonces es aplicable el método de Gradiente Conjugado -Conjugate Gradient Method (CGM)-. La idea básica en que descansa el método del Gradiente Conjugado consiste en construir una base de vectores ortogonales en el espacio de Krylov $\mathcal{K}_n(\underline{A}, \underline{v}^n)$ y utilizarla para realizar la búsqueda de la solución en la forma lo más eficiente posible.

Tal forma de proceder generalmente no sería aconsejable porque la construcción de una base ortogonal utilizando el procedimiento de Gram-Schmidt requiere, al seleccionar cada nuevo elemento de la base, asegurar su ortogonalidad con respecto a cada uno de los vectores construidos previamente. La gran ventaja del método de Gradiente Conjugado radica en que cuando se utiliza este procedimiento, basta con asegurar la ortogonalidad de un nuevo miembro con respecto al último que se ha construido, para que automáticamente esta condición se cumpla con respecto a todos los anteriores.

En el algoritmo de Gradiente Conjugado, se toma a la matriz $\underline{A}$ como simétrica y positiva definida, y como datos de entrada del sistema

$$\underline{A}\underline{u} = \underline{f} \quad (9.46)$$

el vector de búsqueda inicial $\underline{u}^0$ y se calcula $\underline{r}^0 = \underline{f} - \underline{A}\underline{u}^0$, $\underline{p}^0 = \underline{r}^0$, quedando el método numérico esquemáticamente como:

$$\begin{aligned} \alpha^n &= \frac{\langle \underline{p}^n, \underline{p}^n \rangle}{\langle \underline{p}^n, \underline{A}\underline{p}^n \rangle} \\ \underline{u}^{n+1} &= \underline{u}^n + \alpha^n \underline{p}^n \\ \underline{r}^{n+1} &= \underline{r}^n - \alpha^n \underline{A}\underline{p}^n \\ \text{Prueba de convergencia} & \\ \beta^n &= \frac{\langle \underline{r}^{n+1}, \underline{r}^{n+1} \rangle}{\langle \underline{r}^n, \underline{r}^n \rangle} \\ \underline{p}^{n+1} &= \underline{r}^{n+1} + \beta^n \underline{p}^n \\ n &= n + 1 \end{aligned} \quad (9.47)$$

donde $\langle \cdot, \cdot \rangle = (\cdot, \cdot)$ será el producto interior adecuado al sistema lineal en particular, la solución aproximada será $\underline{u}^{n+1}$ y el vector residual será $\underline{r}^{n+1}$.

En la implementación numérica y computacional del método es necesario realizar la menor cantidad de operaciones posibles por iteración, en particular en $\underline{A}\underline{p}^n$, una manera de hacerlo queda esquemáticamente como:

Dado el vector de búsqueda inicial $\underline{u}$, calcula $\underline{r} = \underline{f} - \underline{A}\underline{u}$, $\underline{p} = \underline{r}$ y $\mu = \underline{r} \cdot \underline{r}$.

Para $n = 1, 2, \dots$, Mientras $(\mu < \varepsilon)$ {
 $\underline{v} = \underline{A}\underline{p}$

$$\begin{aligned}
 \alpha &= \frac{\underline{\mu}}{\underline{p} \cdot \underline{v}} \\
 \underline{u} &= \underline{u} + \alpha \underline{p} \\
 \underline{r} &= \underline{r} - \alpha \underline{v} \\
 \underline{\mu}' &= \underline{r} \cdot \underline{r} \\
 \beta &= \frac{\underline{\mu}'}{\underline{\mu}} \\
 \underline{p} &= \underline{r} + \beta \underline{p} \\
 \underline{\mu} &= \underline{\mu}' \\
 &\}
 \end{aligned}$$

la solución aproximada será $\underline{u}$ y el vector residual será $\underline{r}$.

Si se denota con $\{\lambda_i, V_i\}_{i=1}^N$ las eigen-soluciones de $\underline{A}$, i.e. $\underline{A}V_i = \lambda_i V_i$, $i = 0, 1, 2, \dots, N$. Ya que la matriz $\underline{A}$ es simétrica, los eigen-valores son reales y se pueden ordenar $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_N$. Se define el número de condición por $Cond(\underline{A}) = \lambda_N/\lambda_1$ y la norma de la energía asociada a $\underline{A}$ por $\|\underline{u}\|_{\underline{A}}^2 = \underline{u} \cdot \underline{A}u$ entonces

$$\|\underline{u} - \underline{u}^k\|_{\underline{A}} \leq \|\underline{u} - \underline{u}^0\|_{\underline{A}} \left[\frac{1 - \sqrt{Cond(\underline{A})}}{1 + \sqrt{Cond(\underline{A})}} \right]^{2k}. \quad (9.48)$$

El siguiente teorema da idea del espectro de convergencia del sistema $\underline{A}u = \underline{b}$ para el método de Gradiente Conjugado.

Teorema 4 Sea $\kappa = cond(\underline{A}) = \frac{\lambda_{\max}}{\lambda_{\min}} \geq 1$, entonces el método de Gradiente Conjugado satisface la $\underline{A}$ -norma del error dado por

$$\frac{\|e^n\|}{\|e^0\|} \leq \frac{2}{\left[\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1} \right)^n + \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \right)^n \right]} \leq 2 \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \right)^n \quad (9.49)$$

donde $\underline{e}^m = \underline{u} - \underline{u}^m$ del sistema $\underline{A}u = \underline{b}$.

Nótese que para κ grande se tiene que

$$\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \simeq 1 - \frac{2}{\sqrt{\kappa}} \quad (9.50)$$

tal que

$$\|\underline{e}^n\|_{\underline{A}} \simeq \|\underline{e}^0\|_{\underline{A}} \exp \left(-2 \frac{n}{\sqrt{\kappa}} \right) \quad (9.51)$$

de lo anterior se puede esperar un espectro de convergencia del orden de $O(\sqrt{\kappa})$ iteraciones (véase [61] y [66]).

Gradiente Conjugado Precondicionado Cuando la matriz $\underline{\underline{A}}$ es simétrica y definida positiva se puede escribir como

$$\lambda_1 \leq \frac{\underline{\underline{uA}} \cdot \underline{\underline{u}}}{\underline{\underline{u}} \cdot \underline{\underline{u}}} \leq \lambda_n \quad (9.52)$$

y tomando la matriz $\underline{\underline{C}}^{-1}$ como un preconditionador de $\underline{\underline{A}}$ con la condición de que

$$\lambda_1 \leq \frac{\underline{\underline{uC}}^{-1} \underline{\underline{A}} \cdot \underline{\underline{u}}}{\underline{\underline{u}} \cdot \underline{\underline{u}}} \leq \lambda_n \quad (9.53)$$

entonces la Ec. (9.46) se puede escribir como

$$\underline{\underline{C}}^{-1} \underline{\underline{A}} \underline{\underline{u}} = \underline{\underline{C}}^{-1} \underline{\underline{b}} \quad (9.54)$$

donde $\underline{\underline{C}}^{-1} \underline{\underline{A}}$ es también simétrica y definida positiva en el producto interior $\langle \underline{\underline{u}}, \underline{\underline{v}} \rangle = \underline{\underline{u}} \cdot \underline{\underline{Cv}}$, porque

$$\begin{aligned} \langle \underline{\underline{u}}, \underline{\underline{C}}^{-1} \underline{\underline{A}} \underline{\underline{v}} \rangle &= \underline{\underline{u}} \cdot \underline{\underline{C}} (\underline{\underline{C}}^{-1} \underline{\underline{A}} \underline{\underline{v}}) \\ &= \underline{\underline{u}} \cdot \underline{\underline{A}} \underline{\underline{v}} \end{aligned} \quad (9.55)$$

que por hipótesis es simétrica y definida positiva en ese producto interior.

La elección del producto interior $\langle \cdot, \cdot \rangle$ quedará definido como

$$\langle \underline{\underline{u}}, \underline{\underline{v}} \rangle = \underline{\underline{u}} \cdot \underline{\underline{C}}^{-1} \underline{\underline{A}} \underline{\underline{v}} \quad (9.56)$$

por ello las Ecs. (9.47[1]) y (9.47[3]), se convierten en

$$\alpha^{k+1} = \frac{\underline{\underline{r}}^k \cdot \underline{\underline{r}}^k}{\underline{\underline{p}}^{k+1} \cdot \underline{\underline{C}}^{-1} \underline{\underline{p}}^{k+1}} \quad (9.57)$$

y

$$\beta^{k+1} = \frac{\underline{\underline{p}}^k \cdot \underline{\underline{C}}^{-1} \underline{\underline{r}}^k}{\underline{\underline{p}}^k \cdot \underline{\underline{Ap}}^k} \quad (9.58)$$

generando el método de Gradiente Conjugado preconditionado con preconditionador $\underline{\underline{C}}^{-1}$. Es necesario hacer notar que los métodos Gradiente Conjugado y Gradiente Conjugado Precondicionado sólo difieren en la elección del producto interior.

Para el método de Gradiente Conjugado Precondicionado, los datos de entrada son un vector de búsqueda inicial $\underline{u}^0$ y el preconditionador $\underline{\underline{C}}^{-1}$. Calculándose $\underline{r}^0 = \underline{b} - \underline{\underline{A}}\underline{u}^0$, $\underline{p} = \underline{\underline{C}}^{-1}\underline{r}^0$, quedando el método esquemáticamente como:

$$\begin{aligned}\beta^{k+1} &= \frac{\underline{p}^k \cdot \underline{\underline{C}}^{-1}\underline{r}^k}{\underline{p}^k \cdot \underline{\underline{A}}\underline{p}^k} \\ \underline{p}^{k+1} &= \underline{r}^k - \beta^{k+1}\underline{p}^k \\ \alpha^{k+1} &= \frac{\underline{r}^k \cdot \underline{r}^k}{\underline{p}^{k+1} \cdot \underline{\underline{C}}^{-1}\underline{p}^{k+1}} \\ \underline{u}^{k+1} &= \underline{u}^k + \alpha^{k+1}\underline{p}^{k+1} \\ \underline{r}^{k+1} &= \underline{\underline{C}}^{-1}\underline{r}^k - \alpha^{k+1}\underline{\underline{A}}\underline{p}^{k+1}.\end{aligned}\tag{9.59}$$

Algoritmo Computacional del Método Dado el sistema $\underline{\underline{A}}\underline{u} = \underline{b}$, con la matriz $\underline{\underline{A}}$ simétrica y definida positiva de dimensión $n \times n$. La entrada al método será una elección de $\underline{u}^0$ como condición inicial, $\varepsilon > 0$ como la tolerancia del método, N como el número máximo de iteraciones y la matriz de preconditionamiento $\underline{\underline{C}}^{-1}$ de dimensión $n \times n$, el algoritmo del método de Gradiente Conjugado Precondicionado queda como:

$$\begin{aligned}\underline{r} &= \underline{b} - \underline{\underline{A}}\underline{u} \\ \underline{w} &= \underline{\underline{C}}^{-1}\underline{r} \\ \underline{v} &= (\underline{\underline{C}}^{-1})^T \underline{w} \\ \alpha &= \frac{\underline{r} \cdot \underline{r}}{\underline{v} \cdot \underline{v}} \\ k &= 1\end{aligned}$$

Mientras que $k \leq N$

Si $\|\underline{v}\|_\infty < \varepsilon$ Salir

$$\underline{x} = \underline{\underline{A}}\underline{v}$$

$$t = \frac{\alpha}{\sum_{j=1}^n v_j x_j}$$

$$\underline{u} = \underline{u} + t\underline{v}$$

$$\underline{r} = \underline{r} - t\underline{x}$$

$$\underline{w} = \underline{\underline{C}}^{-1}\underline{r}$$

$$\beta = \frac{\underline{r} \cdot \underline{r}}{\underline{w} \cdot \underline{w}}$$

Si $\|\underline{r}\|_\infty < \varepsilon$ Salir

$$s = \frac{\beta}{\alpha}$$

$$\underline{v} = (\underline{C}^{-1})^T \underline{w} + s\underline{v}$$

$$\alpha = \beta$$

$$k = k + 1$$

la salida del método será la solución aproximada $\underline{u} = (u_1, \dots, u_n)$ y el residual $\underline{r} = (r_1, \dots, r_n)$.

En el caso del método sin preconditionamiento, $\underline{C}^{-1}$ es la matriz identidad, que para propósitos de optimización sólo es necesario hacer la asignación de vectores correspondiente en lugar del producto de la matriz por el vector. En el caso de que la matriz $\underline{A}$ no sea simétrica, el método de Gradiente Conjugado puede extenderse para soportarlas, para más información sobre pruebas de convergencia, resultados numéricos entre los distintos métodos de solución del sistema algebraico $\underline{A}\underline{u} = \underline{b}$ generada por la discretización de un problema elíptico y como extender estos para matrices no simétricas ver [62] y [63].

Teorema 5 Sean $\underline{A}, \underline{B}$ y $\underline{C}$ tres matrices simétricas y positivas definidas entonces

$$\kappa(\underline{C}^{-1}\underline{A}) \leq \kappa(\underline{C}^{-1}\underline{B}) \kappa(\underline{B}^{-1}\underline{A}).$$

Método Residual Mínimo Generalizado Si la matriz generada por la discretización es no simétrica, entonces una opción, es el método Residual Mínimo Generalizado -Generalized Minimum Residual Method (GMRES)-, este representa una formulación iterativa común satisfaciendo una condición de optimización. La idea básica detrás del método se basa en construir una base ortonormal

$$\{\underline{v}^1, \underline{v}^2, \dots, \underline{v}^n\} \quad (9.60)$$

para el espacio de Krylov $\mathcal{K}_n(\underline{A}, \underline{v}^n)$. Para hacer $\underline{v}^{n+1}$ ortogonal a $\mathcal{K}_n(\underline{A}, \underline{v}^n)$, es necesario usar todos los vectores previamente construidos $\{\underline{v}^{n+1j}\}_{j=1}^n$ -en la práctica sólo se guardan algunos vectores anteriores- en los cálculos. Y el algoritmo se basa en una modificación del método de Gram-Schmidt para la generación de una base ortonormal. Sea $\underline{V}_n = [\underline{v}^1, \underline{v}^2, \dots, \underline{v}^n]$ la cual denota

la matriz conteniendo $\underline{v}^j$ en la j -ésima columna, para $j = 1, 2, \dots, n$, y sea $\underline{\underline{H}}_n = [h_{i,j}]$, $1 \leq i, j \leq n$, donde las entradas de $\underline{\underline{H}}_n$ no especificadas en el algoritmo son cero. Entonces, $\underline{\underline{H}}_n$ es una matriz superior de Hessenberg. i.e. $h_{ij} = 0$ para $j < i - 1$, y

$$\begin{aligned}\underline{\underline{AV}}_n &= \underline{\underline{V}}_n \underline{\underline{H}}_n + h_{n+1,n} [0, \dots, 0, \underline{v}^{n+1}] \\ \underline{\underline{H}}_n &= \underline{\underline{H}}_n^T \underline{\underline{AV}}_n.\end{aligned}\quad (9.61)$$

En el algoritmo del método Residual Mínimo Generalizado, la matriz $\underline{\underline{A}}$ es tomada como no simétrica, y como datos de entrada del sistema

$$\underline{\underline{Au}} = \underline{f} \quad (9.62)$$

el vector de búsqueda inicial $\underline{u}^0$ y se calcula $\underline{r}^0 = \underline{f} - \underline{\underline{A}}\underline{u}^0$, $\beta^0 = \|\underline{r}^0\|$, $\underline{v}^1 = \underline{r}^0/\beta^0$, quedando el método esquemáticamente como:

$$\begin{aligned}&\text{Para } n = 1, 2, \dots, \text{Mientras } \beta^n < \tau\beta^0 \{ \\&\quad \underline{w}_0^{n+1} = \underline{\underline{Av}}^n \\&\quad \text{Para } l = 1 \text{ hasta } n \{ \\&\quad \quad h_{l,n} = \langle \underline{w}_l^{n+1}, \underline{v}^l \rangle \\&\quad \quad \underline{w}_{l+1}^{n+1} = \underline{w}_l^{n+1} - h_{l,n} \underline{v}^l \\&\quad \quad \} \\&\quad h_{n+1,n} = \|\underline{w}_{n+1}^{n+1}\| \\&\quad \underline{v}^{n+1} = \underline{w}_{n+1}^{n+1}/h_{n+1,n} \\&\quad \text{Calcular } \underline{y}^n \text{ tal que } \beta^n = \|\beta^0 \underline{e}_1 - \hat{\underline{\underline{H}}}_n \underline{y}^n\| \text{ es mínima} \\&\quad \} \end{aligned}\quad (9.63)$$

donde $\hat{\underline{\underline{H}}}_n = [h_{ij}]_{1 \leq i \leq n+1, 1 \leq j \leq n}$, la solución aproximada será $\underline{u}^n = \underline{u}^0 + \underline{\underline{V}}_n \underline{y}^n$, y el vector residual será

$$\underline{r}^n = \underline{r}^0 - \underline{\underline{AV}}_n \underline{y}^n = \underline{\underline{V}}_{n+1} \left(\beta^0 \underline{e}_1 - \hat{\underline{\underline{H}}}_n \underline{y}^n \right). \quad (9.64)$$

Teorema 6 Sea $\underline{u}^k$ la iteración generada después de k iteraciones de GM-RES, con residual $\underline{r}^k$. Si la matriz $\underline{\underline{A}}$ es diagonalizable, i.e. $\underline{\underline{A}} = \underline{\underline{V}}\underline{\underline{\Lambda}}\underline{\underline{V}}^{-1}$ donde $\underline{\underline{\Lambda}}$ es una matriz diagonal de eigen-valores de $\underline{\underline{A}}$, y $\underline{\underline{V}}$ es la matriz cuyas columnas son los eigen-vectores, entonces

$$\frac{\|\underline{r}^k\|}{\|\underline{r}^0\|} \leq \kappa(\underline{\underline{V}}) \min_{p_\kappa \in \Pi_\kappa, p_\kappa(0)=1} \max_{\lambda_j} |p_\kappa(\lambda_j)| \quad (9.65)$$

donde $\kappa(\underline{\underline{V}}) = \frac{\|\underline{\underline{V}}\|}{\|\underline{\underline{V}}^{-1}\|}$ es el número de condicionamiento de $\underline{\underline{V}}$.

Algunos Ejemplos Sea el sistema lineal $\underline{A}\underline{u} = \underline{f}$ dado por

$$\begin{bmatrix} 4 & 3 & 0 \\ 3 & 4 & 1 \\ 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} 24 \\ 20 \\ -24 \end{bmatrix}$$

cuya solución es $\begin{bmatrix} 3 \\ 4 \\ -5 \end{bmatrix}$, si usamos métodos directos obtenemos los siguientes resultados:

- Solución usando el método Factorización LU:

$$\begin{bmatrix} +3.0000000000e+00 \\ +4.0000000000e+00 \\ -5.0000000000e+00 \end{bmatrix}$$

- Solución usando el método Tridiagonal:

$$\begin{bmatrix} +3.0000000000e+00 \\ +4.0000000000e+00 \\ -5.0000000000e+00 \end{bmatrix}$$

- Solución usando el método Choleski:

$$\begin{bmatrix} +3.0000000000e+00 \\ +4.0000000000e+00 \\ -5.0000000000e+00 \end{bmatrix}$$

- Solución usando el método de la Inversa:

$$\begin{bmatrix} +3.0000000000e+00 \\ +4.0000000000e+00 \\ -5.0000000000e+00 \end{bmatrix}$$

Si ahora usamos métodos iterativos (con tolerancia $1e-6$) obtenemos:

- Solución usando el método Jacobi, iteraciones necesarias para resolver el sistema lineal: 59

$$\begin{bmatrix} +3.0000036111e+00 \\ +4.0000042130e+00 \\ -5.0000012037e+00 \end{bmatrix}$$

- Solución usando el método Gauss-Seidel, iteraciones necesarias para resolver el sistema lineal: 25

$$\begin{bmatrix} +3.0000151461e + 00 \\ +3.9999873782e + 00 \\ -5.0000031554e + 00 \end{bmatrix}$$

- Solución usando el método CGM, iteraciones necesarias para resolver el sistema lineal: 3

$$\begin{bmatrix} +3.0000000000e + 00 \\ +4.0000000000e + 00 \\ -5.0000000000e + 00 \end{bmatrix}$$

Esto nos muestra que para ciertos sistemas lineales (en apariencia inofensivo) algunos métodos iterativos pueden requerir una gran cantidad de iteraciones para converger y en algunos casos, esto ni siquiera esta garantizado, es por ello que debemos elegir el método más adecuado para el tipo de sistema lineal con el que se trabaje:

- En los métodos directos la solución $\underline{u}$ se obtiene en un número fijo de pasos y sólo están sujetos a los errores de redondeo.
- En los métodos iterativos, se realizan iteraciones para aproximarse a la solución $\underline{u}$ aprovechando las características propias de la matriz $\underline{A}$, tratando de usar un menor número de pasos que en un método directo.

Por lo general, es conveniente usar bibliotecas⁹⁴ para implementar de forma eficiente a los vectores, matrices -bandadas, dispersas o ralas- y resolver el sistemas lineal.

Los métodos iterativos rara vez se usan para resolver sistemas lineales de dimensión pequeña (el concepto de dimensión pequeña es muy relativo), ya que el tiempo necesario para conseguir una exactitud satisfactoria rebasa el que requieren los métodos directos. Sin embargo, en el caso de sistemas grandes con un alto porcentaje de elementos cero, son eficientes tanto en

⁹⁴Algunas de las bibliotecas más usadas para resolver sistemas lineales usando matrices bandadas y dispersas son PETCs, HYPRE, ATLAS, LAPACK++, LAPACK, EISPACK, LINPACK, BLAS, entre muchas otras alternativas, tanto para implementaciones secuenciales como paralelas y más recientemente para hacer uso de los procesadores CUDA en las GPU de nVidia.

el almacenamiento en la computadora como en el tiempo que se invierte en su solución. Por ésta razón al resolver éstos sistemas algebraicos de ecuaciones es preferible aplicar métodos iterativos tal como Gradiente Conjugado -Conjugate Gradient Method (CGM)- o Residual Mínimo Generalizado -Generalized Minimum Residual Method (GMRES) según sea el tipo de matriz generada.

Usando Python En Python es común usar *numpy* para definir un matrices y vectores además de sus operaciones relacionadas, para ello podemos usar:

```
import numpy as np
A = np.matrix([[4, 3, 0], [3, 4, 1], [0, -1, 4]])
b = np.matrix([[24], [20], [-24]])
```

Antes de resolver un sistema lineal, es conveniente validar que una matriz tenga inversa. Para que tenga inversa, la matriz no tiene que tener vectores linealmente independientes, es decir, no debe haber filas o columnas que puedan ser escritas como la combinación de otras filas o columnas. Para ello podemos usar:

```
import numpy as np
A = np.array(
[[0,1,0,0],
[0,0,1,0],
[0,1,1,0],
[1,0,0,1]]
)
lambdas, V = np.linalg.eig(A.T)
print(A[lambdas == 0, :])
```

visualizando las entradas que son linealmente independientes y cuales no:

```
[[0 1 1 0]]
```

En Python hay diversas formas de resolver un sistema lineal, por ejemplo, si usamos el cálculo de la inversa A^{-1} :


```
import numpy as np
A = np.matrix([[4, 3, 0], [3, 4, 1], [0, -1, 4]])
b = np.matrix([[24], [20], [-24]])
if np.linalg.det(A) == 0:
    x = None
    print("No se puede resolver")
else:
    x = (A**-1)*b
    print(x)
    print(np.dot(A, x))
    print((np.dot(A, x) == b).all())
```

que nos entregara la siguiente salida:

```
[[3.]
 [4.]
 [-5.]]
[[24.]
 [20.]
 [-24]]
True
```

El primer vector es la solución del sistema lineal, la comprobación y la revisión de la igualdad entrada a entrada entre la solución y el vector b ⁹⁵.

Otro ejemplo usando *linalg.solve*:

```
import numpy as np
A = np.array([[4, 3, 0], [3, 4, 1], [0, -1, 4]])
b = np.array([[24], [20], [-24]])
x = np.linalg.solve(A, b)
print(x)
```

Otro ejemplo usando *linalg.inv*:

⁹⁵En este caso las soluciones son idénticas, pero en general pueden diferir en algunos dígitos por la pérdida de precisión, por lo que este ejemplo es solo es ilustrativo.

```
import numpy as np
A = np.array([[4, 3, 0], [3, 4, 1], [0, -1, 4]])
b = np.array([[24], [20], [-24]])
x = np.linalg.inv(A).dot(b)
print(x)
```

Otro ejemplo usando *linalg.pinv*:

```
import numpy as np
A = np.array([[4, 3, 0], [3, 4, 1], [0, -1, 4]])
b = np.array([[24], [20], [-24]])
x = np.linalg.pinv(A).dot(b)
print(x)
```

Otro ejemplo usando *linalg.qr*:

```
import numpy as np
A = np.array([[4, 3, 0], [3, 4, 1], [0, -1, 4]])
b = np.array([[24], [20], [-24]])
Q, R = np.linalg.qr(A)
y = np.dot(Q.T, b)
x = np.linalg.solve(R, y)
print(x)
```

Otro ejemplo usando *sympy*:

```
from sympy import symbols, Matrix, linsolve
x, y, z = symbols('x, y, z')
A = Matrix([[4, 3, 0], [3, 4, 1], [0, -1, 4]])
b = Matrix([[24], [20], [-24]])
xx = linsolve((A,b),[x, y, z])
print(xx)
```

Usando C++ GMM++⁹⁶ es una librería para C++ que permite definir diversos tipos de matrices y vectores además operaciones básicas de álgebra lineal. La facilidad de uso y la gran cantidad de opciones hacen que GMM++

⁹⁶GMM++ [<http://getfem.org/gmm.html>]

sea una buena opción para trabajar con operaciones elementales de álgebra lineal.

Se instala en Debian Linux y/o Ubuntu como:

```
# apt install libgmm++-dev
```

Para compilar el ejemplo usar:

```
$ g++ ejemplito.cpp
```

Para ejecutar usar:

```
$ ./a.out
```

Ejemplo 3 *Un sencillo ejemplo de manejo de matrices y vectores en C++:*

```
#include <gmm/gmm.h>
#include <math.h>
int main(void)
{
    int N = 100;
    // Matriz densa
    gmm::dense_matrix<double> AA(N, N);
    // Matriz dispersa
    gmm::row_matrix< gmm::rsvector<double> > A(N, N);
    // Vectores
    std::vector<double> x(N), b(N);
    int i;
    double P = -2 ;
    double Q = 1;
    double R = 1;
    A(0, 0) = P; // Primer renglon de la matriz A y vector b
    A(0, 1) = Q;
    b[0] = P;
    // Renglones intermedios de la matriz A y vector b
    for(i = 1; i < N - 1; i++)
    {
        A(i, i - 1) = R;
        A(i, i) = P;
        A(i, i + 1) = Q;
        b[i] = P);
    }
    A(N - 1, N - 2) = R; // Renglon final de la matriz A y vector b
    A(N - 1, N - 1) = P;
```

```
b[N - 1] = P;  
// Copia la matriz dispersa a la densa para usarla en LU  
gmm::copy(A, AA);  
// Visualiza la matriz y el vector  
std::cout << "Matriz A" << AA << gmm::endl;  
std::cout << "Vector b" << b << gmm::endl;  
return 0;  
}
```

Precondicionadores Una vía que permite mejorar la eficiencia de los métodos iterativos consiste en transformar al sistema de ecuaciones en otro equivalente, en el sentido de que posea la misma solución del sistema original pero que a su vez tenga mejores condiciones espectrales. Esta transformación se conoce como preconditionamiento y consiste en aplicar al sistema de ecuaciones una matriz conocida como preconditionador encargada de realizar el mejoramiento del número de condicionamiento.

Una amplia clase de preconditionadores han sido propuestos basados en las características algebraicas de la matriz del sistema de ecuaciones, mientras que por otro lado también existen preconditionadores desarrollados a partir de las características propias del problema que lo origina, un estudio más completo puede encontrarse en [65] y [64].

¿Qué es un Precondicionador? De una manera formal podemos decir que un preconditionador consiste en construir una matriz $\underline{\underline{C}}$, la cuál es una aproximación en algún sentido de la matriz $\underline{\underline{A}}$ del sistema $\underline{\underline{A}}u = \underline{\underline{b}}$, de manera tal que si multiplicamos ambos miembros del sistema de ecuaciones original por $\underline{\underline{C}}^{-1}$ obtenemos el siguiente sistema

$$\underline{\underline{C}}^{-1}\underline{\underline{A}}u = \underline{\underline{C}}^{-1}\underline{\underline{b}} \quad (9.66)$$

donde el número de condicionamiento de la matriz del sistema transformado $\underline{\underline{C}}^{-1}\underline{\underline{A}}$ debe ser menor que el del sistema original, es decir

$$\text{Cond}(\underline{\underline{C}}^{-1}\underline{\underline{A}}) < \text{Cond}(\underline{\underline{A}}), \quad (9.67)$$

dicho de otra forma un preconditionador es una inversa aproximada de la matriz original

$$\underline{\underline{C}}^{-1} \simeq \underline{\underline{A}}^{-1} \quad (9.68)$$

que en el caso ideal $\underline{\underline{C}}^{-1} = \underline{\underline{A}}^{-1}$ el sistema convergería en una sola iteración, pero el coste computacional del cálculo de $\underline{\underline{A}}^{-1}$ equivaldría a resolver el sistema por un método directo. Se sugiere que $\underline{\underline{C}}$ sea una matriz lo más próxima a $\underline{\underline{A}}$ sin que su determinación suponga un coste computacional elevado.

Dependiendo de la forma de plantear el producto de $\underline{\underline{C}}^{-1}$ por la matriz del sistema obtendremos distintas formas de preconditionamiento, estas son:

$\underline{\underline{C}}^{-1}\underline{\underline{A}}u = \underline{\underline{C}}^{-1}\underline{\underline{b}}$	Precondicionamiento por la izquierda
$\underline{\underline{A}}\underline{\underline{C}}^{-1}\underline{\underline{C}}u = \underline{\underline{b}}$	Precondicionamiento por la derecha
$\underline{\underline{C}}_1^{-1}\underline{\underline{A}}\underline{\underline{C}}_2^{-1}\underline{\underline{C}}_2u = \underline{\underline{C}}_1^{-1}\underline{\underline{b}}$	Precondicionamiento por ambos lados si $\underline{\underline{C}}$ puede factorizarse como $\underline{\underline{C}} = \underline{\underline{C}}_1\underline{\underline{C}}_2$.

El uso de un preconditionador en un método iterativo provoca que se incurra en un costo de cómputo extra debido a que inicialmente se construye y luego se debe aplicar en cada iteración. Teniéndose que encontrar un balance entre el costo de construcción y aplicación del preconditionador versus la ganancia en velocidad en convergencia del método.

Ciertos preconditionadores necesitan poca o ninguna fase de construcción, mientras que otros pueden requerir de un trabajo substancial en esta etapa. Por otra parte la mayoría de los preconditionadores requieren en su aplicación un monto de trabajo proporcional al número de variables; esto implica que se multiplica el trabajo por iteración en un factor constante.

De manera resumida un buen preconditionador debe reunir las siguientes características:

- i) Al aplicar un preconditionador $\underline{\underline{C}}$ al sistema original de ecuaciones $\underline{\underline{A}}u = \underline{\underline{b}}$, se debe reducir el número de iteraciones necesarias para que la solución aproximada tenga la convergencia a la solución exacta con una exactitud ε prefijada.
- ii) La matriz $\underline{\underline{C}}$ debe ser fácil de calcular, es decir, el costo computacional de la construcción del preconditionador debe ser pequeño comparado con el costo total de resolver el sistema de ecuaciones $\underline{\underline{A}}u = \underline{\underline{b}}$.
- iii) El sistema $\underline{\underline{C}}z = \underline{\underline{r}}$ debe ser fácil de resolver. Esto debe interpretarse de dos maneras:
 - a) El monto de operaciones por iteración debido a la aplicación del preconditionador $\underline{\underline{C}}$ debe ser pequeño o del mismo orden que

las que se requerirían sin preconditionamiento. Esto es importante si se trabaja en máquinas secuenciales.

b) El tiempo requerido por iteración debido a la aplicación del preconditionador debe ser pequeño.

En computadoras paralelas es importante que la aplicación del preconditionador sea paralelizable, lo cual eleva su eficiencia, pero debe de existir un balance entre la eficacia de un preconditionador en el sentido clásico y su eficiencia en paralelo ya que la mayoría de los preconditionadores tradicionales tienen un componente secuencial grande.

Clasificación de los Precondicionadores En general se pueden clasificar en dos grandes grupos según su manera de construcción: los algebraicos o a posteriori y los a priori o directamente relacionados con el problema continuo que lo origina.

Precondicionador a Posteriori Los preconditionadores algebraicos o a posteriori son los más generales, ya que sólo dependen de la estructura algebraica de la matriz $\underline{A}$, esto quiere decir que no tienen en cuenta los detalles del proceso usado para construir el sistema de ecuaciones lineales $\underline{A}u = \underline{b}$. Entre estos podemos citar los métodos de preconditionamiento del tipo Jacobi, SSOR, factorización incompleta, inversa aproximada, diagonal óptimo y polinomial.

Precondicionador Jacobi El método preconditionador Jacobi es el preconditionador más simple que existe y consiste en tomar en calidad de preconditionador a los elementos de la diagonal de $\underline{A}$

$$C_{ij} = \begin{cases} A_{ij} & \text{si } i = j \\ 0 & \text{si } i \neq j. \end{cases} \quad (9.69)$$

Debido a que las operaciones de división son usualmente más costosas en tiempo de cómputo, en la práctica se almacenan los recíprocos de la diagonal de $\underline{A}$.

Ventajas: No necesita trabajo para su construcción y puede mejorar la convergencia.

Desventajas: En problemas con número de condicionamiento muy grande, no es notoria la mejoría en el número de iteraciones.

Precondicionador SSOR Si la matriz original es simétrica, se puede descomponer como en el método de sobrerrelajamiento sucesivo simétrico (SSOR) de la siguiente manera

$$\underline{\underline{A}} = \underline{\underline{D}} + \underline{\underline{L}} + \underline{\underline{L}}^T \quad (9.70)$$

donde $\underline{\underline{D}}$ es la matriz de la diagonal principal y $\underline{\underline{L}}$ es la matriz triangular inferior.

La matriz en el método SSOR se define como

$$\underline{\underline{C}}(\omega) = \frac{1}{2-\omega} \left(\frac{1}{\omega} \underline{\underline{D}} + \underline{\underline{L}} \right) \left(\frac{1}{\omega} \underline{\underline{D}} \right)^{-1} \left(\frac{1}{\omega} \underline{\underline{D}} + \underline{\underline{L}} \right)^T \quad (9.71)$$

en la práctica la información espectral necesaria para hallar el valor óptimo de ω es demasiado costoso para ser calculado.

Ventajas: No necesita trabajo para su construcción, puede mejorar la convergencia significativamente.

Desventajas: Su paralelización depende fuertemente del ordenamiento de las variables.

Precondicionador de Factorización Incompleta Existe una amplia clase de preconditionadores basados en factorizaciones incompletas. La idea consiste en que durante el proceso de factorización se ignoran ciertos elementos diferentes de cero correspondientes a posiciones de la matriz original que son nulos. La matriz preconditionadora se expresa como $\underline{\underline{C}} = \underline{\underline{L}}\underline{\underline{U}}$, donde $\underline{\underline{L}}$ es la matriz triangular inferior y $\underline{\underline{U}}$ la superior. La eficacia del método depende de cuán buena sea la aproximación de $\underline{\underline{C}}^{-1}$ con respecto a $\underline{\underline{A}}^{-1}$.

El tipo más común de factorización incompleta se basa en seleccionar un subconjunto S de las posiciones de los elementos de la matriz y durante el proceso de factorización considerar a cualquier posición fuera de éste igual a cero. Usualmente se toma como S al conjunto de todas las posiciones (i, j) para las que $A_{ij} \neq 0$. Este tipo de factorización es conocido como factorización incompleta LU de nivel cero, ILU(0).

El proceso de factorización incompleta puede ser descrito formalmente como sigue:

Para cada k , si $i, j > k$:

$$S_{ij} = \begin{cases} A_{ij} - A_{ij}A_{ij}^{-1}A_{kj} & \text{Si } (i, j) \in S \\ A_{ij} & \text{Si } (i, j) \notin S. \end{cases} \quad (9.72)$$

Una variante de la idea básica de las factorizaciones incompletas lo constituye la factorización incompleta modificada que consiste en que si el producto

$$A_{ij} - A_{ij}A_{ij}^{-1}A_{kj} \neq 0 \quad (9.73)$$

y el llenado no está permitido en la posición (i, j) , en lugar de simplemente descartarlo, esta cantidad se le sustrae al elemento de la diagonal A_{ij} . Matemáticamente esto corresponde a forzar a la matriz preconditionadora a tener la misma suma por filas que la matriz original. Esta variante resulta de interés puesto que se ha probado que para ciertos casos la aplicación de la factorización incompleta modificada combinada con pequeñas perturbaciones hace que el número de condicionamiento espectral del sistema preconditionado sea de un orden inferior.

Ventaja: Puede mejorar el condicionamiento y la convergencia significativamente.

Desventaja: El proceso de factorización es costoso y difícil de paralelizar en general.

Precondicionador de Inversa Aproximada El uso del preconditionador de inversas aproximada se ha convertido en una buena alternativa para los preconditionadores implícitos debido a su naturaleza paralelizable. Aquí se construye una matriz inversa aproximada usando el producto escalar de Frobenius.

Sea $\mathcal{S} \subset C_n$, el subespacio de las matrices $\underline{\underline{C}}$ donde se busca una inversa aproximada explícita con un patrón de dispersión desconocido. La formulación del problema está dada como:

Encontrar $\underline{\underline{C}}_0 \in \mathcal{S}$ tal que

$$\underline{\underline{C}}_0 = \arg \min_{\underline{\underline{C}} \in \mathcal{S}} \|\underline{\underline{AC}} - \underline{\underline{I}}\|. \quad (9.74)$$

Además, esta matriz inicial $\underline{\underline{C}}_0$ puede ser una inversa aproximada de $\underline{\underline{A}}$ en un sentido estricto, es decir,

$$\|\underline{\underline{AC}}_0 - \underline{\underline{I}}\| = \varepsilon < 1. \quad (9.75)$$

Existen dos razones para esto, primero, la ecuación (9.75) permite asegurar que $\underline{\underline{C}}_0$ no es singular (lema de Banach), y segundo, esta será la base para construir un algoritmo explícito para mejorar $\underline{\underline{C}}_0$ y resolver la ecuación $\underline{\underline{A}}u = \underline{\underline{b}}$.

La construcción de $\underline{\underline{C}}_0$ se realiza en paralelo, independizando el cálculo de cada columna. El algoritmo permite comenzar desde cualquier entrada de la columna k , se acepta comúnmente el uso de la diagonal como primera aproximación. Sea r_k el residuo correspondiente a la columna k -ésima, es decir

$$r_k = \underline{\underline{A}}C_k - e_k \quad (9.76)$$

y sea $\mathcal{I}_k$ el conjunto de índices de las entradas no nulas en r_k , es decir, $\mathcal{I}_k = \{i = \{1, 2, \dots, n\} \mid r_{ik} \neq 0\}$. Si $\mathcal{L}_k = \{l = \{1, 2, \dots, n\} \mid C_{lk} \neq 0\}$, entonces la nueva entrada se busca en el conjunto $\mathcal{J}_k = \{j \in \mathcal{L}_k^c \mid A_{ij} \neq 0, \forall i \in \mathcal{I}_k\}$. En realidad las únicas entradas consideradas en $\underline{\underline{C}}_k$ son aquellas que afectan las entradas no nulas de r_k . En lo que sigue, asumimos que $\mathcal{L}_k \cup \{j\} = \{i_1^k, i_2^k, \dots, i_{p_k}^k\}$ es no vacío, siendo p_k el número actual de entradas no nulas de $\underline{\underline{C}}_k$ y que $i_{p_k}^k = j$, para todo $j \in \mathcal{J}_k$. Para cada j , calculamos

$$\|\underline{\underline{A}}C_k - e_k\|_2^2 = 1 - \sum_{l=1}^{p_k} \frac{\left[\det(\underline{\underline{D}}_l^k)\right]^2}{\det(\underline{\underline{G}}_{l-2}^k) \det(\underline{\underline{G}}_l^k)} \quad (9.77)$$

donde, para todo k , $\det(\underline{\underline{G}}_0^k) = 1$ y $\underline{\underline{G}}_l^k$ es la matriz de Gram de las columnas $i_1^k, i_2^k, \dots, i_{p_k}^k$ de la matriz $\underline{\underline{A}}$ con respecto al producto escalar Euclidiano; $\underline{\underline{D}}_l^k$ es la matriz que resulta de remplazar la última fila de la matriz $\underline{\underline{G}}_l^k$ por $a_{ki_1^k}, a_{ki_2^k}, \dots, a_{ki_{p_k}^k}$, con $1 \leq l \leq p_k$. Se selecciona el índice j_k que minimiza el valor de $\|\underline{\underline{A}}C_k - e_k\|_2$.

Esta estrategia define el nuevo índice seleccionado j_k atendiendo solamente al conjunto $\mathcal{L}_k$, lo que nos lleva a un nuevo óptimo donde se actualizan todas las entradas correspondientes a los índices de $\mathcal{L}_k$. Esto mejora el criterio de (9.74) donde el nuevo índice se selecciona manteniendo las entradas correspondientes a los índices de $\mathcal{L}_k$. Así $\underline{\underline{C}}_k$ se busca en el conjunto

$$\mathcal{S}_k = \{\underline{\underline{C}}_k \in \mathbb{R}^n \mid C_{ik} = 0, \forall i \in \mathcal{L}_k \cup \{j_k\}\},$$

$$\underline{\underline{m}}_k = \sum_{l=1}^{p_k} \frac{\det(\underline{\underline{D}}_l^k)}{\det(\underline{\underline{G}}_{l-2}^k) \det(\underline{\underline{G}}_l^k)} \tilde{m}_l \quad (9.78)$$

donde $\tilde{\underline{C}}_l$ es el vector con entradas no nulas i_h^k ($1 \leq h \leq l$). Cada una de ellas se obtiene evaluando el determinante correspondiente que resulta de remplazar la última fila del $\det(\underline{\underline{G}}_l^k)$ por e_h^t , con $1 \leq l \leq p_k$.

Evidentemente, los cálculos de $\|\underline{\underline{A}}\underline{C}_k - \underline{e}_k\|_2^2$ y de $\underline{C}_k$ pueden actualizarse añadiendo la contribución de la última entrada $j \in \mathcal{J}_k$ a la suma previa de 1 a $p_k - 1$. En la práctica, $\det(\underline{\underline{G}}_l^k)$ se calcula usando la descomposición de Cholesky puesto que $\underline{\underline{G}}_l^k$ es una matriz simétrica y definida positiva. Esto sólo involucra la factorización de la última fila y columna si aprovechamos la descomposición de $\underline{\underline{G}}_{l-1}^k$. Por otra parte, $\det(\underline{\underline{D}}_l^k) / \det(\underline{\underline{G}}_l^k)$ es el valor de la última incógnita del sistema $\underline{\underline{G}}_l^k \underline{d}_l = \left(a_{ki_1^k}, a_{ki_2^k}, \dots, a_{ki_l^k}\right)^T$ necesitándose solamente una sustitución por descenso. Finalmente, para obtener $\tilde{\underline{C}}_l$ debe resolverse el sistema $\underline{\underline{G}}_l^k \underline{v}_l = \underline{e}_l$, con $\tilde{\underline{C}}_{i_l^k} = \underline{v}_l$, ($1 \leq h \leq l$).

Ventaja: Puede mejorar el condicionamiento y la convergencia significativamente y es fácilmente paralelizable.

Desventaja: El proceso construcción es algo laborioso.

Precondicionador a Priori Los preconditionadores a priori son más particulares y dependen para su construcción del conocimiento del proceso de discretización de la ecuación diferencial parcial, dicho de otro modo dependen más del proceso de construcción de la matriz $\underline{\underline{A}}$ que de la estructura de la misma.

Estos preconditionadores usualmente requieren de más trabajo que los del tipo algebraico discutidos anteriormente, sin embargo permiten el desarrollo de métodos de solución especializados más rápidos que los primeros.

Veremos algunos de los métodos más usados relacionados con la solución de ecuaciones diferenciales parciales en general y luego nos concentraremos en el caso de los métodos relacionados directamente con descomposición de dominio.

En estos casos el preconditionador $\underline{\underline{C}}$ no necesariamente toma la forma simple de una matriz, sino que debe ser visto como un operador en general. De aquí que $\underline{\underline{C}}$ podría representar al operador correspondiente a una versión simplificada del problema con valores en la frontera que deseamos resolver.

Por ejemplo se podría emplear en calidad de preconditionador al operador original del problema con coeficientes variables tomado con coeficientes

constantes. En el caso del operador de Laplace se podría tomar como preconditionador a su discretización en diferencias finitas centrales.

Por lo general estos métodos alcanzan una mayor eficiencia y una convergencia óptima, es decir, para ese problema en particular el preconditionador encontrado será el mejor preconditionador existente, llegando a disminuir el número de iteraciones hasta en un orden de magnitud. Donde muchos de ellos pueden ser paralelizados de forma efectiva.

El Uso de la Parte Simétrica como Precondicionador La aplicación del método del Gradiente Conjugado en sistemas no auto-adjuntos requiere del almacenamiento de los vectores previamente calculados. Si se usa como preconditionador la parte simétrica

$$(\underline{\underline{A}} + \underline{\underline{A}}^T)/2 \quad (9.79)$$

de la matriz de coeficientes $\underline{\underline{A}}$, entonces no se requiere de éste almacenamiento extra en algunos casos, resolver el sistema de la parte simétrica de la matriz $\underline{\underline{A}}$ puede resultar más complicado que resolver el sistema completo.

El Uso de Métodos Directos Rápidos como Precondicionadores En muchas aplicaciones la matriz de coeficientes $\underline{\underline{A}}$ es simétrica y positivo definida, debido a que proviene de un operador diferencial auto-adjunto y acotado. Esto implica que se cumple la siguiente relación para cualquier matriz $\underline{\underline{B}}$ obtenida de una ecuación diferencial similar

$$c_1 \leq \frac{\underline{\underline{x}}^T \underline{\underline{A}} \underline{\underline{x}}}{\underline{\underline{x}}^T \underline{\underline{B}} \underline{\underline{x}}} \leq c_2 \quad \forall \underline{\underline{x}} \quad (9.80)$$

donde c_1 y c_2 no dependen del tamaño de la matriz. La importancia de esta propiedad es que del uso de $\underline{\underline{B}}$ como preconditionador resulta un método iterativo cuyo número de iteraciones no depende del tamaño de la matriz.

La elección más común para construir el preconditionador $\underline{\underline{B}}$ es a partir de la ecuación diferencial parcial separable. El sistema resultante con la matriz $\underline{\underline{B}}$ puede ser resuelto usando uno de los métodos directos de solución rápida, como pueden ser por ejemplo los basados en la transformada rápida de Fourier.

Como una ilustración simple del presente caso obtenemos que cualquier operador elíptico puede ser preconditionado con el operador de Poisson.

Implementación en C++ GMM++⁹⁷ es una librería para C++ que permite definir diversos tipos de matrices y vectores además operaciones básicas de algebra lineal. La facilidad de uso y la gran cantidad de opciones hacen que GMM++ sea una buena opción para trabajar con operaciones elementales de algebra lineal.

Se instala en Debian Linux y/o Ubuntu como:

```
# apt install libgmm++-dev
```

Para compilar el ejemplo usar:

```
$ g++ ejemplito.cpp
```

Para ejecutar usar:

```
$ ./a.out
```

Ejemplo 4 *Sea*

$$-u''(x) + u(x) = 0, \quad 0 \leq x \leq 1, \quad u(0) = 0, \quad u(1) = 1$$

entonces el programa queda implementado como:

```
#include <gmm/gmm.h>
#include <math.h>
const double pi = 3.141592653589793;
double LD(double x) // Lado derecho
{
    return ( -pi * pi * cos(pi * x));
}
double SA(double x) // Solucion analitica
{
    return (cos(pi * x));
}
int main(void)
{
    int M=11; // Particion
    int N=M-2; // Nodos interiores
    double a=0; // Inicio dominio
    double c=1; // Fin dominio
    double h=(c-a)/(M-1); // Incremento en la malla
    double Y0=1.0; // Condicion inicial en el inicio del dominio
    double Y1=-1.0; // Condicion inicial en el fin del dominio
```

⁹⁷GMM++ [<http://getfem.org/gmm.html>]

```

// Matriz densa
gmm::dense_matrix<double> AA(N, N);
// Matriz dispersa
gmm::row_matrix< gmm::rsvector<double> > A(N, N);
// Vectores
std::vector<double> x(N), b(N);
int i;
double P = -2 / (h * h);
double Q = 1 / (h * h);
double R = 1 / (h * h);
A(0, 0) = P; // Primer renglon de la matriz A y vector b
A(0, 1) = Q;
b[0] = LD(a + h) - (Y0 / (h * h));
// Renglones intermedios de la matriz A y vector b
for(i = 1; i < N - 1; i++)
{
    A(i, i - 1) = R;
    A(i, i) = P;
    A(i, i + 1) = Q;
    b[i] = LD(a + (i + 1) * h);
}
A(N - 1, N - 2) = R; // Renglon final de la matriz A y vector b
A(N - 1, N - 1) = P;
b[N - 1] = LD(a + (i + 1) * h) - (Y1 / (h * h));
// Copia la matriz dispersa a la densa para usarla en LU
gmm::copy(A, AA);
// Visualiza la matriz y el vector
std::cout << "Matriz A" << AA << gmm::endl;
std::cout << "Vector b" << b << gmm::endl;
// LU para matrices densas
gmm::lu_solve(AA, x, b);
std::cout << "LU" << x << gmm::endl;
gmm::identity_matrix PS; // Optional scalar product for cg
gmm::identity_matrix PR; // Optional preconditioner
gmm::iteration iter(10E-6); // Iteration object with the max residu
size_t restart = 50; // restart parameter for GMRES
// Conjugate gradient
gmm::cg(A, x, b, PS, PR, iter);

```

```

std::cout << "CGM"<< x << std::endl;
// BICGSTAB BiConjugate Gradient Stabilized
gmm::bicgstab(A, x, b, PR, iter);
std::cout << "BICGSTAB"<< x << std::endl;
// GMRES generalized minimum residual
gmm::gmres(A, x, b, PR, restart, iter);
std::cout << "GMRES"<< x << std::endl;
// Quasi-Minimal Residual method
gmm::qmr(A, x, b, PR, iter);
std::cout << "Quasi-Minimal"<< x << std::endl;
// Visualiza la solucion numerica
std::cout << "Solucion Numerica"<< std::endl;
std::cout << a << " " << Y0 << gmm::endl;
for(i = 0; i < N; i++)
{
std::cout << (i + 1)*h << " " << x[i] << gmm::endl;
}
std::cout << c << " " << Y1 << gmm::endl;
// Visualiza la solucion analitica
std::cout << "Solucion Analitica"<< std::endl;
std::cout << a << " " << SA(a) << gmm::endl;
for(i = 0; i < N; i++)
{
std::cout << (i + 1)*h << " " << SA((a + (i + 1)*h)) << gmm::endl;
}
std::cout << c << " " << SA(c) << gmm::endl;
// Visualiza el error en valor absoluto en cada nodo
std::cout << "Error en el calculo"<< std::endl;
std::cout << a << " " << abs(Y0 - SA(a)) << gmm::endl;
for(i = 0; i < N; i++)
{
std::cout << (i + 1)*h << " " << abs(x[i] - SA((a + (i + 1)*h))) <<
gmm::endl;
}
std::cout << c << " " << abs(Y1 - SA(c)) << gmm::endl;
return 0;
}

```

9.8 Ecuaciones Diferenciales Ordinarias

En matemáticas, una ecuación diferencial ordinaria (comúnmente abreviada "EDO") es la ecuación diferencial que relaciona una función desconocida de una variable independiente con sus derivadas. Es decir, una sola variable independiente (a diferencia de las ecuaciones diferenciales parciales que involucran derivadas parciales de varias variables), y una o más de sus derivadas respecto de tal variable.

Son recursos en física, la ingeniería, la economía, la meteorología, la biología, la química y en aplicaciones como las de modelado en ciencias, se las estudia en diversas áreas (como geometría, mecánica y astronomía) y perspectivas.

Matemáticamente, es de conveniente interés, la obtención de una familia de funciones que verifican una ecuación y establecen la solución general. Sólo las ecuaciones diferenciales más sencillas admiten soluciones dadas por fórmulas explícitas (como las lineales asociadas a una teoría desarrollada prácticamente por completo). No obstante, pueden determinarse algunas propiedades de las soluciones de una ecuación diferencial sin requerirse su formulación exacta, clave para resolver la mayoría de las ecuaciones diferenciales no lineales de sumo interés en numerosos casos. Casos carentes de una fórmula autocontenida para su solución que se suple con la aproximada numéricamente con el auxilio crucial de las computadoras.

La matemática pura centra el foco formal en la solución, su existencia y si es o no única. La aplicada controla la validez de los métodos para la solución numéricamente aproximada y el rigor de las justificaciones con que se los sustenta.

La teoría de los sistemas dinámicos prioriza el análisis cualitativo de sistemas descritos por ecuaciones diferenciales mientras se han venido sumando numerosos métodos numéricos para determinar soluciones con un grado dado de precisión.

En ingeniería, ciencias naturales y sociales hay muchos problemas de interés que, cuando se plantean, exigen la determinación de una función la cual debe verificar una ecuación que involucra derivadas de la función desconocida. Dichas ecuaciones se denominan ecuaciones diferenciales. Tal vez el ejemplo más conocido es la ley de Newton.

Efecto Mariposa la famosa frase se refiere a la idea de que las alas de una mariposa⁹⁸ podrían crear pequeños cambios en la atmósfera que, en última instancia, podrían alterar la trayectoria de un tornado o retrasar, acelerar o incluso impedir la aparición de un tornado en otro lugar.

El efecto mariposa es un fenómeno descrito en sistemas caóticos con dependencia sensitiva a las condiciones iniciales por el cual, cualquier pequeña variación en las condiciones iniciales en un sistema determinista no lineal, acabará dando lugar a una diferencia mayor en estados posteriores.

Algo de Historia Lorenz trabajaba en 1960 en la predicción del tiempo meteorológico con la ayuda de ordenadores y, al repetir unos cálculos introduciendo valores anteriormente obtenidos, observó cambios drásticos en los resultados del tiempo meteorológico previsto a largo plazo tras efectuar un levísimo redondeo, (la impresora, para ahorrar espacio recogía sólo tres cifras decimales del valor de una determinada magnitud, [0.506], que él introdujo como valor inicial para continuar los cálculos -considerando que el error era insignificante-, en lugar de introducir el valor más preciso almacenado en la memoria del ordenador, [0.506127]). Esta es su propia descripción:

En un momento dado, decidí repetir algunos de los cálculos con el fin de examinar con mayor detalle lo que estaba ocurriendo. Detuve el ordenador, tecleé una línea de números que había salido por la impresora un rato antes y lo puse en marcha otra vez. Me fui al vestíbulo a tomarme una taza de café y regresé al cabo de una hora, tiempo durante el cual el ordenador había simulado unos dos meses de tiempo meteorológico. Los números que salían por la impresora no tenían nada que ver con los anteriores.

Inmediatamente pensé que se había estropeado alguna válvula o que el ordenador tenía alguna otra avería, cosa nada infrecuente, pero antes de llamar a los técnicos decidí comprobar dónde se encontraba la dificultad, sabiendo que de esa forma podría acelerar la reparación. En lugar de una interrupción brusca, me encontré

⁹⁸En el ejemplo particular propuesto por Edward N. Lorenz, por el efecto mariposa, si se parte de dos mundos o situaciones globales casi idénticos, pero en uno de ellos hay una mariposa aleteando y en el otro no, a largo plazo, el mundo con la mariposa y el mundo sin la mariposa acabarán siendo muy diferentes. En uno de ellos puede producirse a gran distancia un tornado y en el otro no suceder nada en absoluto.

con que los nuevos valores repetían los anteriores en un principio, pero que enseguida empezaban a diferir, en una, en varias unidades, en la última cifra decimal, luego en la anterior y luego en la anterior. La verdad es que las diferencias se duplicaban en tamaño más o menos constantemente cada cuatro días, hasta que cualquier parecido con las cifras originales desaparecía en algún momento del segundo mes.

Con eso me bastó para comprender lo que ocurría: los números que yo había tecleado no eran los números originales exactos sino los valores redondeados que había dado a la impresora en un principio. Los errores redondeados iniciales eran los culpables: se iban amplificando constantemente hasta dominar la solución. Dicho con terminología de hoy: se trataba del caos.

Aunque el "efecto mariposa" se explica a menudo como sinónimo de dependencia sensible de las condiciones iniciales del tipo descrito por Lorenz en su trabajo de 1963 (y observado previamente por Poincaré), la metáfora de la mariposa se aplicó originalmente al trabajo que publicó en 1969 que llevó la idea un paso más allá. Lorenz propuso un modelo matemático de cómo los pequeños movimientos de la atmósfera afectan a sistemas mayores. Descubrió que los sistemas de ese modelo sólo podían predecirse hasta un punto específico en el futuro y que, más allá de ese punto, la reducción del error en las condiciones iniciales no aumentaba la predictibilidad (siempre que el error no fuera cero). Esto demostraba que un sistema determinista podía ser "indistinguible desde el punto de vista observacional" de uno no determinista en términos de predictibilidad.

La Teoría del Caos es la rama de la matemática, la física y otras ciencias (biología, meteorología, economía, entre ellas) que trata ciertos tipos de sistemas complejos y sistemas dinámicos no lineales muy sensibles a las variaciones en las condiciones iniciales.

Pequeñas variaciones en dichas condiciones iniciales pueden implicar grandes diferencias en el comportamiento futuro, imposibilitando la predicción a largo plazo. Esto sucede aunque estos sistemas son en rigor deterministas, es decir, su comportamiento puede ser completamente determinado conociendo sus condiciones iniciales, esto lo veremos en el ejemplo del atractor de Lorenz.

La teoría del caos se refiere a sistemas deterministas cuyo comportamiento puede, en principio, predecirse. Los sistemas caóticos son predecibles durante

un tiempo y luego "parecen" volverse aleatorios. La cantidad de tiempo durante el cual se puede predecir eficazmente el comportamiento de un sistema caótico depende de tres cosas: cuánta incertidumbre se puede tolerar en la predicción, con qué precisión se puede medir su estado actual y una escala de tiempo que depende de la dinámica del sistema, llamada tiempo de Lyapunov. Cuando no se pueden hacer predicciones significativas, el sistema parece aleatorio.

La teoría del caos es un método de análisis cualitativo y cuantitativo para investigar el comportamiento de sistemas dinámicos que no pueden explicarse ni predecirse mediante relaciones de datos individuales, sino que deben explicarse y predecirse mediante relaciones de datos continuas y completas.

Los sistemas dinámicos se pueden clasificar

- Estables, cuando dos soluciones con condiciones iniciales suficientemente cercanas siguen siendo cercanas a lo largo del tiempo. Así, un sistema estable tiende a lo largo del tiempo a un punto, u órbita, según su dimensión (atractor o sumidero).
- Inestables, cuando dos soluciones con condiciones iniciales diferentes acaban divergiendo por pequeñas que sean las diferencias entre las condiciones iniciales. Así un sistema inestable «escapa» de los atractores.
- Caóticos, cuando el sistema no es inestable y si bien dos soluciones se mantienen a una distancia «finita» cercana a un atractor del sistema dinámico, las soluciones se mueven en torno al atractor de manera irregular y pasado el tiempo ambas soluciones no son cercanas, si bien suelen ser cualitativamente similares. De esa manera, el sistema permanece confinado en una zona de su espacio de estados, pero sin tender a un atractor fijo.

Una de las principales características tanto de los sistemas inestables como los caóticos es que tienen una gran dependencia de las condiciones iniciales (esto diferencia a ambos tipos de los sistemas estables). De un sistema del que se conocen sus ecuaciones de evolución temporal características, y con unas condiciones iniciales fijas, se puede conocer exactamente su evolución en el tiempo. Pero en el caso de los sistemas caóticos, una mínima diferencia en esas condiciones hace que el sistema evolucione de manera totalmente distinta.

Caos Determinista comprende una serie de fenómenos encontrados en la teoría de sistemas dinámicos, la teoría de ecuaciones diferenciales y la mecánica clásica. En términos generales el caos determinista da lugar a trayectorias asociadas a la evolución temporal de forma muy irregular y aparentemente azarosa que sin embargo son totalmente deterministas, a diferencia del azar genuino. La irregularidad de las trayectorias está asociada a la imposibilidad práctica de predecir la evolución futura del sistema, aunque esta evolución sea totalmente determinista.

Definición de caos y atractores No hay una definición universal sobre el caos, pero hay tres ingredientes en los que todos los científicos están de acuerdo:

- **Movimiento oscilante.** Las trayectorias no se ajustan a un punto fijo, órbita periódica u órbita cuasiperiódica cuando el tiempo tiende a infinito.
- **Determinismo.** El sistema no es azaroso sino determinista. El comportamiento irregular, en dimensión finita, surge de la no linealidad. Por eso se define como determinista.
- **Sensibilidad a las condiciones.** Las trayectorias que comienzan cerca, con el tiempo se separan exponencialmente. Es decir, condiciones iniciales muy similares acaban dando lugar a comportamientos diferentes pasado un tiempo suficientemente largo.

Los sistemas caóticos típicamente se caracterizan por ser modelizables mediante un sistema dinámico que posee un atractor. Para definir propiamente un atractor hay que recurrir a tecnicismos, y es difícil dar una idea intuitiva sin ellos. En una primera aproximación puede decirse que un atractor es un conjunto en el que todas las trayectorias cercanas convergen. Los puntos fijos y círculos límite son un ejemplo de ello. Al igual que en la definición del caos, hay 3 ingredientes universales:

- Cualquier trayectoria que esté en un atractor, estará en él para $t \rightarrow \infty$.
- Atraen un conjunto de condiciones iniciales. El conjunto lo componen las condiciones iniciales de su trayectoria que acabe en el atractor cuando $t \rightarrow \infty$.

- No existen condiciones iniciales que satisfagan las dos reglas anteriores.

Dentro de los atractores se define como atractor extraño o caótico al atractor que exhibe dependencia sensible con las condiciones iniciales.

Atractor de Lorenz es la representación de un sistema de ecuaciones diferenciales autónomo (no depende explícitamente del tiempo), un sistema dinámico, que en principio su autor propuso para tratar de comprender los fenómenos meteorológicos. Se encontró con que a pesar de que los valores generados nunca se repiten y de que las condiciones iniciales pueden hacer variar completamente los valores generados (de ahí su nombre de atractor extraño o caótico, y también que los que predicen el tiempo se equivoquen tanto), el atractor toma una forma única y parece conservar cierto orden, sus infinitas trayectorias nunca se cortan (pues eso implicaría que entraríamos en un ciclo periódico en un sistema autónomo), y también podemos comprobar cómo este objeto presenta la autosimilitud característica de los fractales. Esta extrema sensibilidad a las condiciones iniciales fue lo que originó la frase del llamado *Efecto Mariposa*: "...el aleteo de una mariposa en Australia sería suficiente para provocar un tornado en Texas...". Conviene señalar que las condiciones necesarias para que exista caos en un sistema de ecuaciones diferenciales autónomo son: deben haber al menos tres ecuaciones diferenciales y al menos tres variables y al menos alguna no linealidad.

El modelo atmosférico que utilizó Lorenz consiste en una atmósfera bidimensional rectangular, cuyo extremo inferior está a una temperatura mayor que el superior. De esta manera el aire caliente subirá y el aire frío bajará creándose corrientes que harían un intercambio de calor por convección.

Las ecuaciones que describen este proceso son

$$\begin{aligned}\frac{dx}{dt} &= s(y - x) \\ \frac{dy}{dt} &= rx - y - xz \\ \frac{dz}{dt} &= xy - bz\end{aligned}$$

en donde las variables, que únicamente dependen del tiempo son:

- "x" representa el flujo convectivo
- "y" es la distribución de temperatura horizontal

- "z" es la distribución de temperatura vertical

Además tenemos de los tres parámetros que intervienen en las ecuaciones:

- "s" cociente entre la viscosidad y la conductividad térmica (número de Prandtl)
- "r" la diferencia de temperatura entre la capa inferior y superior (número de Rayleigh)
- "b" el cociente entre la altura y el ancho de nuestro rectángulo

Lorenz observó dos cosas fundamentales que ocurrían en su ecuación:

- Cualquier diferencia en las condiciones iniciales antes de los cálculos, incluso infinitesimal, cambiaba de forma drástica los resultados. Tan solo se podía predecir el sistema por cortos períodos. Llevando eso a la meteorología, suponía lo que se llamó efecto mariposa, hipersensibilidad a las condiciones iniciales.
- A pesar de lo anterior, la impredecibilidad del sistema, lejos de ser un comportamiento al azar, tenía una curiosa tendencia a evolucionar dentro de una zona muy concreta del espacio de fases, situando una especie de pseudocentro de gravedad de los comportamientos posibles.

Las ecuaciones de Lorenz fueron propuestas como un modelo muy simplificado de la convección en forma de anillos que parece ocurrir a veces en la atmósfera terrestre. Por ello, las tres magnitudes a las que Lorenz se refiere en su sistema son,

- $x \rightarrow$ Razón de rotación del anillo.
- $y \rightarrow$ Gradiente de temperatura.
- $z \rightarrow$ Desviación de la temperatura respecto a su valor de equilibrio.

Lorenz descubrió que su sistema contenía una dinámica extremadamente errática. Las soluciones oscilaban irregularmente sin llegar a repetirse, aunque lo hacían en una región acotada del espacio de fases. Vio que las trayectorias rondaban siempre alrededor de lo que ahora se define como atractor extraño.

A continuación detallamos un programa en Python que usa aritmética de 64 bits, que genera el gráfico del atractor de Lorenz:

```

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
def lorenz(x, y, z, s=10, r=28, b=2.667):
    x_dot = s * (y - x)
    y_dot = r * x - y - x * z
    z_dot = x * y - b * z
    return x_dot, y_dot, z_dot
dt = 0.01
stepCnt = 10000
xs = np.empty((stepCnt + 1,))
ys = np.empty((stepCnt + 1,))
zs = np.empty((stepCnt + 1,))
xs[0], ys[0], zs[0] = (0.0, 1.0, 1.0)
for i in range(stepCnt):
    x_dot, y_dot, z_dot = lorenz(xs[i], ys[i], zs[i])
    xs[i + 1] = xs[i] + (x_dot * dt)
    ys[i + 1] = ys[i] + (y_dot * dt)
    zs[i + 1] = zs[i] + (z_dot * dt)
ax = plt.figure().add_subplot(projection="3d")
ax.plot(xs, ys, zs, lw=0.5)
ax.set_xlabel("X Axis")
ax.set_ylabel("Y Axis")
ax.set_zlabel("Z Axis")
ax.set_title("Lorenz Attractor")
print(xs[stepCnt], ys[stepCnt], zs[stepCnt])
plt.show()

```

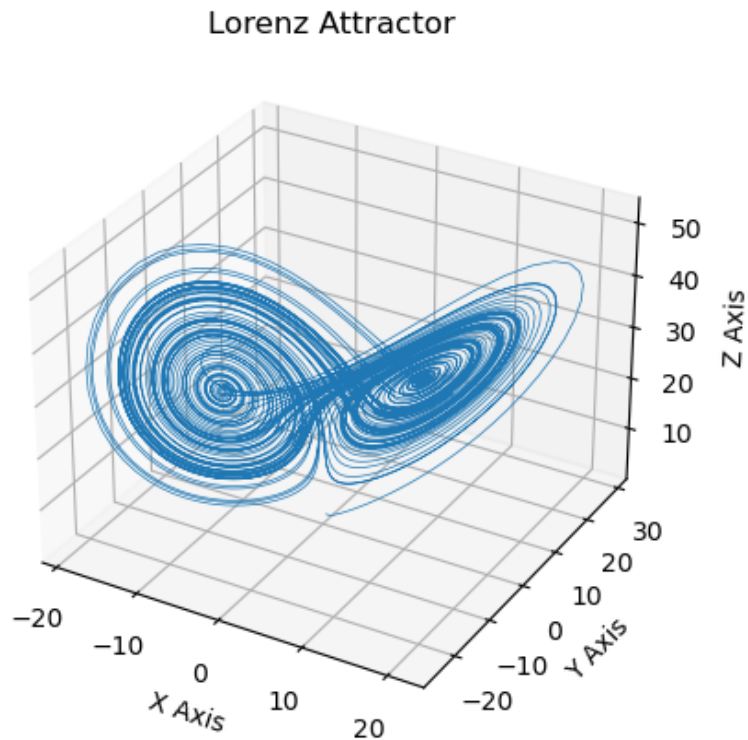
Al ejecutar el programa, se muestra el último valor calculado de la órbita usando:

```
print(xs[stepCnt], ys[stepCnt], zs[stepCnt])
```

que es:

```
-7.7348138109756315 -6.946435787579794 27.143075372419894
```

y generando la siguiente gráfica:



Ahora, tomaremos la línea:

$$xs[0], ys[0], zs[0] = (0.0, 1.0, 1)$$

y la cambiamos por las nuevas condiciones iniciales:

$$xs[0], ys[0], zs[0] = (0.0, 1.0, 1.05)$$

Al ejecutar nuevamente el programa, se muestra el último valor calculado de la órbita:

$$9.709961879594102 \ 14.526677276746598 \ 21.237584691842496$$

Los cuales son notablemente diferentes entre sí, lo que ilustra el efecto mariposa (es un fenómeno descrito en sistemas caóticos con dependencia sensitiva a las condiciones iniciales) por el cual, cualquier pequeña variación en las condiciones iniciales en un sistema determinista no lineal, acabará dando lugar a una diferencia mayor en estados posteriores.

La sensibilidad a las condiciones iniciales la podemos ver con la evolución de dos órbitas de condiciones iniciales próximas, por ejemplo:

```
[[ -14.8748934 -14.78390265 -14.99996569]
 [ -14.90930023 -14.95597323 -14.97229842]]
```

para ello, podemos usar el siguiente programa en Python:

```
import numpy as np
from scipy import integrate
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import cnames
from matplotlib import animation
def lorentz_deriv(xi, t0, sigma=10.0, beta=8.0 / 3, rho=28.0):
    [x, y, z] = xi
    return [sigma * (y - x), x * (rho - z) - y, x * y - beta * z]
np.random.seed(1)
N_trajectories = 2
x0 = -15 + 0.3 * np.random.random((N_trajectories, 3))
print(x0)
t = np.linspace(0, 50, 10000)
x_t = np.asarray([integrate.odeint(lorentz_deriv, x0i, t) for
x0i in x0])
fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1], projection="3d")
ax.axis("off")
colors = plt.cm.jet(np.linspace(0, 1, N_trajectories))
lines = sum([ax.plot([], [], [], "-", c=c) for c in colors], [])
pts = sum([ax.plot([], [], [], "o", c=c) for c in colors], [])
ax.set_xlim((-25, 25))
ax.set_ylim((-35, 35))
```

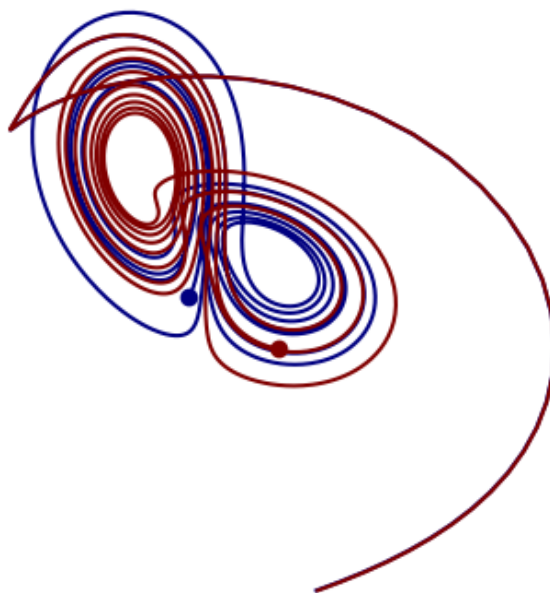


```

ax.set_zlim((5, 55))
ax.view_init(30, 0)
def init():
    for line, pt in zip(lines, pts):
        line.set_data([], [])
        line.set_3d_properties([])
        pt.set_data([], [])
        pt.set_3d_properties([])
    return lines + pts
def animate(i):
    i = (2 * i) % x_t.shape[1]
    for line, pt, xi in zip(lines, pts, x_t):
        x, y, z = xi[:i].T
        line.set_data(x, y)
        line.set_3d_properties(z)
        pt.set_data(x[-1:], y[-1:])
        pt.set_3d_properties(z[-1:])
    ax.view_init(30, 0.3 * i)
    fig.canvas.draw()
    return lines + pts
anim = animation.FuncAnimation(
    fig, animate, init_func=init, frames=1000, interval=30,
    blit=True
)
anim.save("lorentz_attractor.mp4", fps=15, extra_args=["-vcodec", "libx264"])
plt.show()

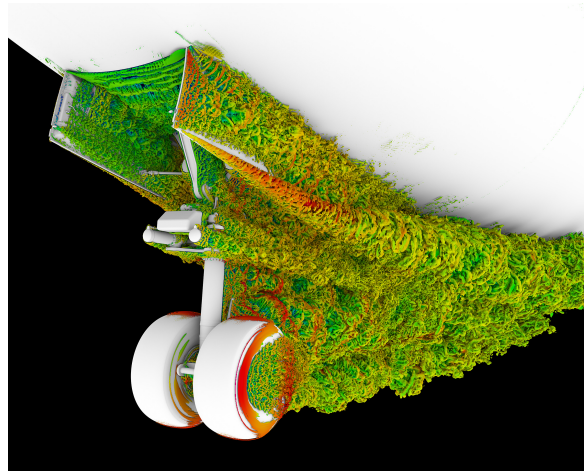
```

Obteniendo la siguiente animación:



9.9 Ecuaciones Diferenciales Parciales

En la búsqueda de una descripción cualitativa de un determinado fenómeno, por lo general se plantea un sistema de ecuaciones diferenciales ordinarias o parciales, validas para cierto dominio y donde se imponen sobre este, una serie de condiciones en la frontera y en su caso de condiciones iniciales. Después de esto, el modelo matemático se considera completo, y es aquí donde la implementación computacional entra a ayudar en la solución del problema, ya que sólo es posible resolver de forma exacta problemas simples y en fronteras geométricas triviales con los métodos matemáticos que disponemos.



En esta sección consideraremos como implementar la solución computacional del Método de Diferencias Finitas, este método es de carácter general que permite la resolución aproximada de ecuaciones diferenciales en derivadas parciales definidas en un dominio finito. Es de una gran sencillez conceptual y constituye un procedimiento muy adecuado para la resolución de una ecuación en una, dos o tres dimensiones.

El método consiste en una aproximación de las derivadas parciales por expresiones algebraicas con los valores de la variable dependiente en un número finito de puntos seleccionados en el dominio⁹⁹. Como resultado de la apro-

⁹⁹La técnica fundamental para los cálculos numéricos en diferencias finitas se basa en aproximar $f(x)$ mediante un polinomio cerca de $x = x_0$. Una de las aproximaciones clásicas es mediante la serie de Taylor, la cual también nos permite, aparte de la aproximación de las derivadas, el cálculo del error en la aproximación mediante su fórmula del residuo.

ximación, la ecuación diferencial parcial que describe el problema es remplazada por un número finito de ecuaciones algebraicas, en términos de los valores de la variable dependiente en los puntos seleccionados. El valor de los puntos seleccionados se convierten en las incógnitas. La solución del sistema de ecuaciones algebraico permite obtener la solución aproximada en cada punto seleccionado de la malla.

Método de Diferencias Finitas en Una Dimensión Sin pérdida de generalidad, consideremos la ecuación diferencial parcial

$$(p(x)u'(x))' + q(x)u'(x) - r(x)u(x) = f(x) \quad (9.81)$$

$$\text{en } a \leq x \leq b \quad \text{donde: } u(a) = u_\alpha \text{ y } u(b) = u_\beta$$

con condiciones de frontera Dirichlet, Neumann o Robin. Para usar el procedimiento general de solución numérica mediante el método de diferencias finitas, debemos de:

1. Generar una malla del dominio, i.e. una malla es un conjunto finito de puntos en los cuales buscaremos la solución aproximada a la ecuación diferencial parcial.
2. Sustituir las derivadas correspondientes con alguna de las formulas de diferencias finitas centradas (véase 9.4 y 9.4), en cada punto donde la solución es desconocida para obtener un sistema lineal algebraico de ecuaciones $\underline{\underline{A}}u = \underline{f}$.
3. Resolver el sistema lineal algebraico de ecuaciones $\underline{\underline{A}}u = \underline{f}$ (véase 9.7), y así obtener la solución aproximada en cada punto de la malla.

Problema con Condiciones de Frontera Dirichlet Consideremos el caso particular de la Ec.(9.81) definido por la ecuación

$$u''(x) = f(x), \quad 0 \leq x \leq 1, \quad u(0) = u_\alpha, \quad u(1) = u_\beta \quad (9.82)$$

con condiciones de frontera Dirichlet. Para usar el procedimiento general de solución numérica mediante el método de diferencias finitas, debemos de:

1. Generar una malla homogénea del dominio¹⁰⁰

$$x_i = ih, \quad i = 0, 1, \dots, n, \quad h = \frac{1}{n} = \Delta x \quad (9.83)$$

2. Sustituir la derivada con la Ec.(9.24) en cada punto donde la solución es desconocida para obtener un sistema algebraico de ecuaciones. Así, en cada punto x_i de la malla aproximamos la ecuación diferencial por¹⁰¹

$$u''(x_i) \approx \frac{u(x_i - h) - 2u(x_i) + u(x_i + h)}{h^2} \quad (9.84)$$

o en su forma simplificada

$$u''(x_i) \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} \quad (9.85)$$

definiendo la solución aproximada de $u(x)$ en x_i como u_i , entonces se genera el siguiente sistema lineal de ecuaciones

$$\begin{aligned} \frac{u_0 - 2u_1 + u_2}{h^2} &= f(x_1) \\ \frac{u_1 - 2u_2 + u_3}{h^2} &= f(x_2) \\ &\vdots \\ \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} &= f(x_i) \\ &\vdots \\ \frac{u_{n-3} - 2u_{n-2} + u_{n-1}}{h^2} &= f(x_{n-2}) \\ \frac{u_{n-2} - 2u_{n-1} + u_\beta}{h^2} &= f(x_{n-1}). \end{aligned} \quad (9.86)$$

¹⁰⁰En el caso de que la malla no sea homogénea, es necesario incluir en la derivada a que h se refiere, por ejemplo en cada punto i , tenemos la h_{i-} (por la izquierda) y la h_{i+} (por la derecha), i.e. $u''(x_i) \approx \frac{u(x_i - h_{i-}) - 2u(x_i) + u(x_i + h_{i+})}{(h_{i-})(h_{i+})}$.

¹⁰¹Notemos que en cada punto de la malla, la aproximación por diferencias finitas supone la solución de tres puntos de la malla x_{i-1} , x_i y x_{i+1} . El conjunto de estos tres puntos de la malla es comúnmente llamado el estencil de diferencias finitas en una dimensión.

Este sistema de ecuaciones se puede escribir como la matriz $\underline{\underline{A}}$ y los vectores $\underline{u}$ y $\underline{f}$ de la forma

$$\begin{bmatrix} -\frac{2}{h^2} & \frac{1}{h^2} & & & \\ \frac{1}{h^2} & -\frac{2}{h^2} & \frac{1}{h^2} & & \\ & \frac{1}{h^2} & -\frac{2}{h^2} & \frac{1}{h^2} & \\ & & \ddots & \ddots & \ddots \\ & & & \frac{1}{h^2} & -\frac{2}{h^2} & \frac{1}{h^2} \\ & & & & \frac{1}{h^2} & -\frac{2}{h^2} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{bmatrix} = \begin{bmatrix} f(x_1) - \frac{u_\alpha}{h^2} \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_{n-2}) \\ f(x_{n-1}) - \frac{u_\beta}{h^2} \end{bmatrix}.$$

en este caso, podemos factorizar $1/h^2$ del sistema lineal $\underline{\underline{A}}\underline{u} = \underline{f}$, quedando como

$$\frac{1}{h^2} \begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & 1 & -2 & 1 & \\ & & \ddots & \ddots & \ddots \\ & & & 1 & -2 & 1 \\ & & & & 1 & -2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{bmatrix} = \begin{bmatrix} f(x_1) - \frac{u_\alpha}{h^2} \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_{n-2}) \\ f(x_{n-1}) - \frac{u_\beta}{h^2} \end{bmatrix}.$$

esta última forma de expresar el sistema lineal algebraico asociado es preferible para evitar problemas numéricos al momento de resolver el sistema lineal por métodos iterativos (véase 9.7) principalmente cuando $h \rightarrow 0$.

3. Resolver el sistema lineal algebraico de ecuaciones $\underline{\underline{A}}\underline{u} = \underline{f}$ (véase 9.7), obtenemos la solución aproximada en cada punto interior de la malla. La solución completa al problema la obtenemos al formar el vector

$$\begin{bmatrix} u_\alpha & u_1 & u_2 & u_3 & \cdots & u_{n-2} & u_{n-1} & u_\beta \end{bmatrix}.$$

Uno de los paquetes más conocidos y usados para el cálculo numérico es MatLab¹⁰² el cual tiene un alto costo monetario para los usuarios. Por

¹⁰²MatLab es un programa comercial para el cálculo numérico el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería [<https://www.mathworks.com>].

ello, una opción es usar alternativas desarrolladas usando licencia de código abierto, un par de estos paquetes son: GNU OCTAVE¹⁰³ y SCILAB¹⁰⁴.

Octave corre gran parte del código desarrollado para MatLab sin requerir cambio alguno, en cuanto a SCILAB es requerido hacer algunos ajustes en la codificación y ejecución. En los siguientes ejemplos¹⁰⁵ se mostrará como implementar la solución computacional.

Ejemplo 5 Sea

$$-u''(x) = -\pi^2 \cos(\pi x), \quad xi \leq x \leq xf, \quad u(xi) = vi, \quad u(xf) = vf$$

El programa queda implementado en OCTAVE (MatLab) como:

```
function [A,b,x] = fdm1d(xi,xf,vi,vf,n)
    N=n-2; % Nodos interiores
    h=(xf-xi)/(n-1); % Incremento en la malla
    %A = sparse(N,N); % Matriz SPARSE
    A=zeros(N,N); % Matriz A
    b=zeros(N,1); % Vector b
    R=1/(h^2);
    P=-2/(h^2);
    Q=1/(h^2);
    % Primer renglon de la matriz A y vector b
    A(1,1)=P;
    A(1,2)=Q;
```

¹⁰³GNU OCTAVE es un programa open source para el cálculo numérico el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería [<http://www.gnu.org/software/octave>].

¹⁰⁴SCILAB es un programa open source para el cálculo numérico el cual provee un poderoso ambiente de cálculo para aplicaciones Científicas y de Ingeniería [<http://www.scilab.org>].

¹⁰⁵Los ejemplos que se muestran en el presente texto se pueden descargar de la página Web:

MDF

o desde GitHub (<https://github.com/antoniocarrillo69/MDF>) mediante

```
git clone git://github.com/antoniocarrillo69/MDF.git
```

```

b(1)=LadoDerecho(xi)-vi*R;
% Renglones intermedios de la matriz A y vector b
for i=2:N-1
    A(i,i-1)=R;
    A(i,i)=P;
    A(i,i+1)=Q;
    b(i)=LadoDerecho(xi+h*(i-1));
end
% Renglon final de la matriz A y vector b
A(N,N-1)=R;
A(N,N)=P;
b(N)=LadoDerecho(xi+h*N)-vf*Q;
% Resuelve el sistema lineal Ax=b
x=inv(A)*b;
% Prepara la graficacion
xx=zeros(n,1);
zz=zeros(n,1);
for i=1:n
    xx(i)=xi+h*(i-1);
    zz(i)=SolucionAnalitica(xx(i));
end
yy=zeros(n,1);
yy(1)=vi; % Condicion inicial
for i=1:N
    yy(i+1)=x(i);
end
yy(n)=vf; % Condicion inicial
% Grafica la solucion de la Ecuacion Diferencial Parcial en 1D
plot(xx,[yy,zz]);
endfunction
function y=LadoDerecho(x)
    y=-pi*pi*cos(pi*x);
endfunction
function y=SolucionAnalitica(x)
    y=cos(pi*x);
endfunction

```


Si el programa lo grabamos en el directorio de trabajo con el nombre **fdm1d.m**, entonces se puede ejecutar en la consola de OCTAVE (MatLab) mediante

$$[A, b, x] = \text{fdm1d}(-1, 2, -1, 1, 30);$$

donde es necesario indicar el inicio (-1) y fin (2) del dominio, el valor de la condición de frontera de inicio (-1) y fin (1) , además de el número de nodos (30) en la partición. La ejecución genera la gráfica de la solución calculada por el método de diferencias finitas y la solución analítica en los mismos puntos de la malla, además devuelve la matriz¹⁰⁶ y los vectores A, b, x generados por la función.

FreeFem++ es un lenguaje de programación y Software enfocado en resolver ecuaciones diferenciales parciales usando el método de elementos finitos. FreeFem++ está escrito en C++ y desarrollado y mantenido por la Universidad Pierre y Marie Curie y el Laboratorio Jacques-Louis Lions. Corre en GNU/Linux, Solaris, OS X y MS Windows. FreeFem++ es Software libre (GPL) y hay un IDE llamado FreeFem++-cs

FreeFem++ es un software libre que sirve para la resolución numérica de problemas 2D y 3D utilizando el método de elementos finitos. ¿Qué problemas puedes resolver? De mecánica de fluidos, difusión de calor, elasticidad, electromagnetismo, etc. Todos aquellos que estén modelizados matemáticamente a través de una ecuación en derivadas parciales (EDP), por ejemplo: <https://www.um.es/freefem/ff++/pmwiki.php?n=Main.Elasticidad3D>

Su desarrollo empezó en 1987 por Olivier Pironneau y se llamó MacFem. A partir de 1992. A partir de 2008, la versión homónima tiene una reescritura del núcleo de cálculo de los elementos finitos, lo que permite la inclusión de los cuestiones multidimensionales. Ejemplo de uso:

```
load "msh3"
// Parameters
int nn = 20; // Mesh quality
// Mesh
```

¹⁰⁶En Octave (MatLab) se define a una matriz mediante $A = \text{zeros}(N,N)$, este tipo de matriz no es adecuada para nuestros fines (véase capítulo 9.7). Para ahorrar espacio y acelerar los cálculos numéricos que se requieren para resolver el sistema lineal asociado usamos un tipo de matriz que no guarda valores innecesarios (ceros), esto se hace mediante la declaración de la matriz como $A = \text{sparse}(N,N)$.

```

int[int] labs = [1, 2, 2, 1, 1, 2]; // Label numbering
mesh3 Th = cube(nn, nn, nn, label=labs);
// Remove the ]0.5,1[^3 domain of the cube
Th = trunc(Th, (x < 0.5) | (y < 0.5) | (z < 0.5), label=1);
// Fespace
fespace Vh(Th, P1);
Vh u, v;
// Macro
macro Grad(u) [dx(u), dy(u), dz(u)] //
// Define the weak form and solve
solve Poisson(u, v, solver=CG)
    = int3d(Th)(
        Grad(u)' * Grad(v)
    )
    -int3d(Th)(
        1 * v
    )
    + on(1, u=0)
    ;
// Plot
plot(u, nbiso=15);

```

Algunos Problemas de Advección-Difusión en 2D En el caso de los problemas de Advección-Difusión al usar métodos que discretizan el dominio usando una malla homogénea, nos interesa encontrar una malla —lo más gruesa posible— en la cual el problema sea soluble sin obtener un error considerable al usar valores de viscosidad pequeños que normalmente generan inestabilidad numérica en los métodos de discretización, las cuales siempre se eliminan al refinar adecuadamente la malla (véase [?] y [?]).

Para el primer ejemplo, la ecuación utilizada es:

$$-\nu \Delta u + \underline{b} \cdot \nabla u = 0 \quad (9.87)$$

en $(x, y) \in [0, 1] \times [0, 1]$, donde

$$u(x, y) = \begin{cases} 0, & (x, y) \in \xi_1 \\ 1, & (x, y) \in \xi_2 \end{cases} \quad (9.88)$$

y $\underline{b} = (1, 3)$, como se muestra en la figura:

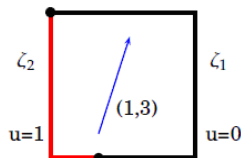


Figura 14: Dominio del problema

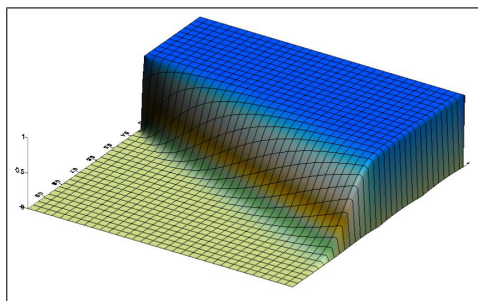


Figura 15: Solución del problema para $\nu = 0.01$

En este caso, la discretización se realizó mediante el método de Diferencias Finitas centradas y en la estabilización se usa el método de Difusión

Artificial. Los ejemplos se resolvieron mediante el método de GMRES con una tolerancia de 10^{-6} , en una malla global de 512×512 (261,121 grados de libertad), para distintos valores de la viscosidad ν . Los resultados obtenidos para las distintas descomposiciones de dominio usando el método BDDC versus los algoritmos DVS, se muestran en la siguiente tabla:

Partición	ν	BDDC	PRIMAL#1	PRIMAL#2	DUAL#1	DUAL#2
8×8 y 64×64	0.01	12	12	11	11	11
8×8 y 64×64	0.001	9	8	8	8	7
8×8 y 64×64	0.0001	9	7	7	7	7
8×8 y 64×64	0.00001	9	7	7	7	7
16×16 y 32×32	0.01	20	19	17	17	18
16×16 y 32×32	0.001	17	14	13	14	13
16×16 y 32×32	0.0001	15	13	13	13	13
16×16 y 32×32	0.00001	16	13	13	13	13
32×32 y 16×16	0.01	33	33	29	29	31
32×32 y 16×16	0.001	30	26	25	25	25
32×32 y 16×16	0.0001	28	25	25	25	25
32×32 y 16×16	0.00001	29	25	25	25	26
64×64 y 8×8	0.01	52	53	53	52	59
64×64 y 8×8	0.001	53	46	46	46	47
64×64 y 8×8	0.0001	53	45	45	47	47
64×64 y 8×8	0.00001	54	45	45	47	48

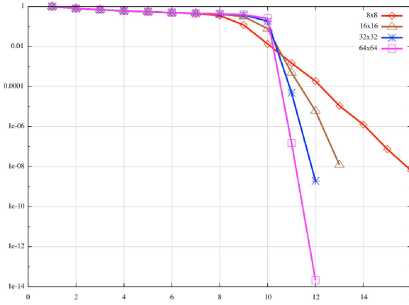


Figura 16: Residual relativo para la malla local de 16×16 , en este caso $\underline{b} = (1, 3)$ y $\nu = 0.00001$ que corresponde a un valor de $Pe = 3.16e + 5$.

Además se muestra el residual relativo de decaimiento para la malla gruesa 16×16 y varias mallas finas en las cuales se ve que la mejor convergencia se obtiene cuando la malla fina se incrementa y la convergencia es lenta cuando el subdominio tiene una pequeña cantidad de grados de libertad.

Para el segundo ejemplo, la ecuación a trabajar es

$$-\nu \Delta u + \underline{b} \cdot \nabla u + cu = 0 \quad (9.89)$$

en $(x, y) \in [-1, 1] \times [0 - 1, 1]$, donde

$$\begin{aligned} u(x, y) &= 1 \begin{cases} y = -1, & 0 < x \leq 1 \\ y = 1, & 0 < x \leq 1 \\ x = 1, & -1 \leq y \leq 1 \end{cases} \\ u(x, y) &= 0, \quad \text{en cualquier otro caso} \end{aligned} \quad (9.90)$$

el coeficiente advectivo esta dado por $\underline{b} = (y, -x)$, el valor de $c = 10^{-4}$.

En este caso, la discretización se realizo mediante el método de Diferencias Finitas centradas y en la estabilización se usa el método de Difusión Artificial. Los ejemplos se resolvieron mediante el método de GMRES con una tolerancia de 10^{-6} , en una malla global de 32×32 (961 grados de libertad), para distintos valores de la viscosidad ν , cuya solución es mostrada en la gráfica:

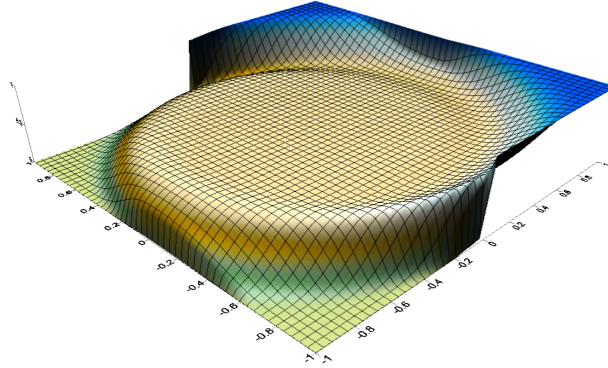


Figura 17: Solución del problema para $\nu = 0.01$

Los resultados obtenidos para las distintas descomposiciones de dominio usando el método FETI versus los algoritmos DVS, se muestran en la siguiente tabla:

Partición	ν	FETI-DP	PRIMAL#1	DUAL#1	PRIMAL#2	DUAL#2
4×4 y 8×8	1	11	9	8	8	8
4×4 y 8×8	0.01	12	11	8	10	9
4×4 y 8×8	0.001	23	20	16	20	16
4×4 y 8×8	0.0001	45	24	19	24	18
4×4 y 8×8	0.00001	69	24	19	24	18
8×8 y 4×4	1	10	9	8	8	8
8×8 y 4×4	0.01	11	16	9	10	13
8×8 y 4×4	0.001	27	24	21	24	22
8×8 y 4×4	0.0001	68	32	25	30	26
8×8 y 4×4	0.00001	111	33	24	29	27
16×16 y 2×2	1	9	8	6	6	6
16×16 y 2×2	0.01	16	26	8	9	21
16×16 y 2×2	0.001	63	47	23	28	41
16×16 y 2×2	0.0001	176	48	29	34	42
16×16 y 2×2	0.00001	200	48	30	34	42

Para el tercer ejemplo, la ecuación a trabajar es

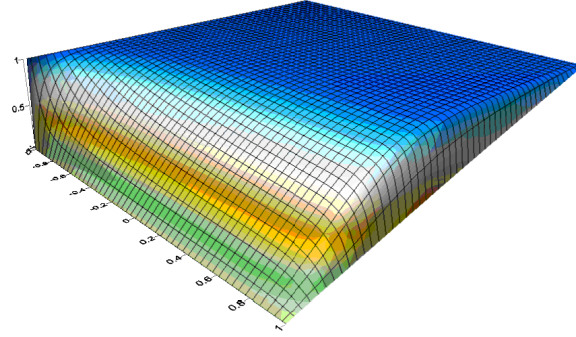
$$-\nu \Delta u + \underline{b} \cdot \nabla u + cu = 0 \quad (9.91)$$

en $(x, y) \in [-1, 1] \times [-1, 1]$, donde

$$\begin{aligned} u(x, y) &= 1 \begin{cases} x = -1, & -1 < y \leq 1 \\ y = 1, & -1 \leq x \leq 1 \end{cases} \\ u(x, y) &= 0, \quad y = -1, -1 \leq x \leq 1 \\ u(x, y) &= \frac{1+y}{2}, \quad x = 1, -1 \leq y \leq 1 \end{aligned} \quad (9.92)$$

el coeficiente advectivo esta dado por $\underline{b} = (\frac{1+y}{2}, 0)$, el valor de $c = 10^{-4}$.

En este caso, la discretización se realizo mediante el método de Diferencias Finitas centradas y en la estabilización se usa el método de Difusión Artificial, Los ejemplos se resolvieron mediante el método de GMRES con una tolerancia de 10^{-6} en la norma infinita en una malla global de 32×32 (961 grados de libertad), para distintos valores de la viscosidad ν , cuya solución es mostrada en la gráfica:

Figura 18: Solución del problema para $\nu = 0.01$

Los resultados obtenidos para las distintas descomposiciones de dominio usando el método FETI-DP versus los algoritmos DVS, se muestran en la siguiente tabla:

Partición	ν	FETI-DP	PRIMAL#1	DUAL#1	PRIMAL#2	DUAL#2
4×4 y 8×8	1	13	10	10	8	8
4×4 y 8×8	0.01	13	11	10	8	7
4×4 y 8×8	0.001	9	8	9	6	6
4×4 y 8×8	0.0001	10	10	10	4	4
4×4 y 8×8	0.00001	11	9	10	3	4
4×4 y 8×8	0.000001	11	9	9	2	3
8×8 y 4×4	1	44	9	9	8	8
8×8 y 4×4	0.01	34	15	14	10	10
8×8 y 4×4	0.001	16	15	15	10	10
8×8 y 4×4	0.0001	16	27	28	9	9
8×8 y 4×4	0.00001	16	32	32	8	8
8×8 y 4×4	0.000001	16	25	25	6	5
16×16 y 2×2	1	159	8	8	6	5
16×16 y 2×2	0.01	98	22	21	9	8
16×16 y 2×2	0.001	38	37	37	18	18
16×16 y 2×2	0.0001	33	48	48	23	22
16×16 y 2×2	0.00001	46	42	41	20	20
16×16 y 2×2	0.000001	51	37	36	15	15

Algo más de Ecuaciones Diferenciales Parciales Existen una gran cantidad herramientas para para resolver ecuaciones diferenciales parciales que se pueden usar, entre ellas destacan:

- <https://freefem.org/>
- <https://www.opengeosys.org/>
- <https://fenicsproject.org/>
- <https://www.openfoam.com/>
- <https://www.dealii.org/>
- <https://www.dune-project.org/>
- <https://www.aivc.org/resource/domus-20-whole-building-hygrothermal-simulation-program>
- <https://www.ctcms.nist.gov/fipy/>
- <http://onelab.info/>
- <https://www.csc.fi/web/elmer>
- <https://github.com/KratosMultiphysics/Kratos>
- <https://www.gomafem.com/>
- <http://www.juliafem.org/>
- <https://getfem.org/>
- <http://mofem.eng.gla.ac.uk/mofem/html/>
- <http://mech.fsv.cvut.cz/~sifel/>
- <https://dedalus-project.org/>
- <http://www.oofem.org/doku.php?id=en:oofem>
- <http://www.hpfem.org/hermes/>
- <https://cimec.org.ar/foswiki/Main/Cimec/PETScFEM>

- <https://pypi.org/project/HydPy/>
- <http://sfepy.org/doc-devel/index.html>
- <https://www.firedrakeproject.org/>
- <https://github.com/romeric/florence>
- https://gfs.sourceforge.net/wiki/index.php/Main_Page
- <https://www.comsol.com/>
- <https://www.featool.com/>
- <https://visualpde.com/>

10 Apéndice A: Software Libre y Propietario

Con el constante aumento de la comercialización de equipos de cómputo y/o comunicación (teléfonos inteligentes, tabletas, computadoras portátiles y de escritorio, etc.) y su relativo bajo costo, estos equipos se han convertido en objetos omnipresentes en nuestra vida diaria, ya que estos permiten realizar un creciente número de actividades cotidianas de miles de millones de usuarios.

Dichos equipos de cómputo y/o comunicación por sí solos tienen poca utilidad, pero su uso en conjunción con el Software adecuado forman un dúo que nos ha permitido tener los avances de los que actualmente disfrutamos. El Software -sistema operativo y los programas de aplicaciones- son los que realmente generan las soluciones al interactuar uno o más paquetes informáticos con los datos del usuario. También, es común que al comprar un equipo de cómputo y/o comunicación, en el costo total, se integre el del sistema operativo, aplicaciones ofimáticas y de antivirus, sean estos usados por el usuario o no y en la mayoría de los casos no es posible solicitar que no sean incluidos en el costo del equipo.

Por otro lado, el Software comercial suele quedar obsoleto muy rápido, ya que constantemente se le agregan nuevas funcionalidades al mismo y estas en general son vendidas como versiones independientes de la adquirida originalmente. Esto obliga al usuario -si quiere hacer uso de ellas- a comprar las nuevas versiones del Software para satisfacer sus crecientes necesidades informáticas y la obsolescencia programada.

Por lo anterior y dada la creciente complejidad de los paquetes de cómputo y el alto costo de desarrollo de aplicaciones innovadoras, en muchos casos, el costo total del Software que comúnmente los usuarios instalan -y que no necesariamente usan las capacidades avanzadas del programa, por las cuales el Software tiene un alto costo comercial- en sus equipos, suele ser más caro que el propio equipo en el que se ejecutan.

Hoy en día los usuarios disponemos de dos grandes opciones para adquirir el Software necesario para que nuestros equipos funcionen, a saber:

- Por un lado, podemos emplear programas comerciales (Software propietario), de los cuales no somos dueños del Software, sólo concesionarios al adquirir una licencia de uso del Software y nos proporcionan un instalable del programa adquirido. La licencia respectiva es en la gran mayoría de los casos muy restrictiva, ya que restringe su uso a un solo

equipo y/o usuario simultáneamente.

- Por otro lado, existe el Software libre¹⁰⁷, desarrollado por usuarios y para usuarios que, entre otras cosas, comparten los códigos fuente, el programa ejecutable y dan libertades para estudiar, adaptar y redistribuir a quien así lo requiera el programa y todos sus derivados.

Sobre la Obsolescencia Programada Es un conjunto de estrategias deliberadas destinadas a asegurarse que la versión actual de un determinado producto quedará desfasada o inservible en un plazo de tiempo predeterminado. De esta manera, los fabricantes se aseguran que los consumidores se verán obligados a reemplazarlo aunque funcione adecuadamente.

La obsolescencia puede lograrse mediante la introducción de un modelo con características superiores o diseñando intencionadamente un producto para que deje de funcionar correctamente en un plazo determinado. En cualquiera de los dos casos, se espera que los consumidores opten por el nuevo producto de la misma marca. Muchas veces la obsolescencia no es sobre el propio producto sino aplicando restricciones al producto de un competidor con la ayuda de una tercera empresa.

Tipos de Obsolescencia Programada Podemos dividir la obsolescencia programada en 4 tipos:

1- Establecimiento artificial del plazo de duración: Los productos se fabrican con piezas cuya duración tienen una vida útil limitada cuando, si se usaran otras de calidad superior ese plazo se extendería.

2- Actualizaciones de Software: Los desarrolladores de Software sacan nuevas versiones de sus aplicaciones que en un momento determinado dejan de ser compatibles con dispositivos antiguos. En muchos casos se ha podido comprobar que esa incompatibilidad es absolutamente artificial ya que al «engañar» al Software este funcionaba sin problemas.

¹⁰⁷A veces también se han usado términos como FOSS y FLOSS. Ambas cosas son similares, ya que FOSS (Free and Open Source Software) traducido como "Software de código abierto" y FLOSS (Free/Libre and Open Source Software) "Software libre y de código abierto". Según quienes adoptan estos términos, lo hacen por tener una imparcialidad entre la carga filosófica del Software libre y el aspecto técnico y/o las ventajas que brinda este modelo de desarrollo. Richard Stallman nos invita a no usarlas y no se trata de un ad hómitem. Stallman y el proyecto GNU nos aconsejan que hablemos siempre de Software libre y aquí no cabe imparcialidad.

3- Obsolescencia percibida: Esta es una táctica psicológica, se trata de convencer al consumidor mediante publicidad y el uso de influenciadores de que el producto que se tiene actualmente está viejo y que se necesita uno nuevo. Como por ejemplo: ¿cuantos megapíxeles necesitas en tu teléfono para sacar una buena foto de tu mascota?

4- Trabas a la reparación: En el caso de los teléfonos por ejemplo, lo de impedir sacar la batería (con la excusa de hacer los teléfonos más delgados) es una forma de obligar a los consumidores a recurrir a los servicios oficiales y a disuadirlos de reemplazarlas por sustitutos más económicos. Otras tácticas son la utilización de piezas no estándar o que necesitan herramientas específicas para la reparación. Muchas veces se suele restringir el acceso a estas piezas o hacer una reducida producción de las mismas para aumentar artificialmente el costo.

Ejemplos de Obsolescencia Programada

- iPhone cada vez más lentos: La Justicia francesa comprobó que actualizaciones de Software hacían cada vez más lento el rendimiento de los modelos más viejos. La empresa le echó la culpa a las baterías, pero pagó una compensación de decenas de millones de dólares. Además rebajó los precios de sus baterías de repuesto para que los teléfonos fueran más rápidos con el nuevo Software y se comprometió a hacer más en el futuro para garantizar que los teléfonos no volvieran a ser más lentos. Con la salida de un nuevo modelo de teléfono cada año, seguro que hay algo de obsolescencia planificada en alguna parte.
- Impresoras: Esto es algo que todos conocemos. Muchas veces nos encontramos con impresoras a precio rebajado, pero al momento de tener que comprar un cartucho de tinta nos encontramos con que este tiene un precio igual o superior a comprar una nueva. Además, se ponen restricciones a la recarga o al uso de cartuchos alternativos. Hubo denuncias de que algunos modelos dejaban de funcionar a partir de cierta cantidad de páginas impresas o cierto tiempo desde la primera impresión.
- Certificados de seguridad: Por ejemplo, el pasado 30 de septiembre de 2021 caduco otro certificado de autenticación (CA de DST Root CA X3 de Let's Encrypt) que ayudaba a validar la conexión en internet a

los dispositivos que no fueron actualizados a otro certificado más actual -en la mayoría de los casos por no ser del interés económico de sus creadores-. Esto ocasionó que millones de dispositivos (teléfonos inteligentes, Smart TV, tabletas, computadoras portátiles y de escritorio, etc.) con algunos años de ser creados y perfectamente funcionales dejarán de conectarse a internet de un día para otro, forzando a sus dueños a desechar el dispositivo por carecer del servicio de internet en las aplicaciones instaladas.

- Cambio de la versión del sistema operativo: En el caso del sistema operativo Windows 10 a 11, la solicitud de requisitos mínimos de Hardware es para muchos equipos excesivo, ya que se estima que dejará fuera en su actualización a casi todos los equipos con más de 4 años de antigüedad por no contar por ejemplo con el Chip TPM 2.0 o GPU compatible con DirectX 12, siendo perfectamente funcionales con la versión actual del sistema operativo. Si bien Windows 10 seguirá con soporte hasta 2025, los usuarios que deseen tener las nuevas características del sistema operativo tendrán que cambiar de equipo.

10.1 Software Propietario

No existe consenso sobre el término a utilizar para referirse al opuesto del Software libre. La expresión «Software propietario (Proprietary Software)» (véase [19]), en la lengua anglosajona, "Proprietary" significa «poseído o controlado privadamente (Privately Owned and Controlled)», que destaca la manutención de la reserva de derechos sobre el uso, modificación o redistribución del Software. Inicialmente utilizado, pero con el inconveniente de que la acepción proviene de una traducción literal del inglés, no correspondiendo su uso como adjetivo en el español, de manera que puede ser considerado como un barbarismo.

El término "propietario" en español resultaría inadecuado, pues significa que «tiene derecho de propiedad sobre una cosa», por lo que no podría calificarse de "propietario" al Software, porque éste no tiene propiedad sobre nada (es decir, no es dueño de nada) y además, no podría serlo (porque es una cosa y no una persona). Así mismo, la expresión "Software propietario" podría ser interpretada como: "Software sujeto a propiedad" (derechos o titularidad) y su opuesto, el Software libre, también está sujeto al derecho de autor. Otra interpretación es que contrariamente al uso popular del término, se puede

afirmar que "todo Software es propietario", por lo que la forma correcta de referirse al Software con restricciones de uso, estudio, copia o mejora es la de Software privativo, según esta interpretación el término "propietario" podría aplicarse tanto para Software libre como Software privativo, ya que la diferencia entre uno y otro está en que el dueño del Software privativo lo licencia como propiedad privada y el de Software libre como propiedad social.

Con la intención de corregir el defecto de la expresión "Software propietario" aparece el llamado "Software con propietario", sin embargo se argumenta contra el término "con propietario" y justamente su similitud con Proprietary en inglés, que sólo haría referencia a un aspecto del Software que no es libre, manteniendo una de las principales críticas a éste (de "Software sujeto a derechos" o "propiedad"). Adicionalmente, si "propietario" se refiere al titular de los derechos de autor -y está claro que no se puede referir al usuario, en tanto éste es simplemente un cesionario-, no resuelve la contradicción: todo el Software libre tiene también titulares de derechos de autor.

La expresión Software no libre (en inglés Non-Free Software) es usado por la FSF para agrupar todo el Software que no es libre, es decir, incluye al llamado en inglés "Semi-Free Software" (Software semilibre) y al "Proprietary Software". Asimismo, es frecuentemente utilizado para referirse al Software que no cumple con las Directrices de Software libre de Debian GNU/Linux, las cuales siguen la misma idea básica de libertad en el Software, propugnada por la FSF y sobre las cuales está basada la definición de código abierto de la Open Source Initiative.

Adicionalmente el Software de código cerrado nace como antónimo de Software de código abierto y por lo tanto se centra más en el aspecto de ausencia de acceso al código que en los derechos sobre el mismo, éste se refiere sólo a la ausencia de una sola libertad por lo que su uso debe enfocarse sólo a este tipo de Software y aunque siempre signifique que es un Software que no es libre, no tiene que ser Software de código cerrado.

La expresión Software privado es usada por la relación entre los conceptos de tener y ser privado. Este término sería inadecuado debido a que, en una de sus acepciones, la palabra "privado" se entiende como antónimo de "público", es decir, que «no es de propiedad pública o estatal, sino que pertenece a particulares», provocando que esta categoría se interpretará como no referente al Estado, lo que produciría la exclusión del Software no libre generado por el aparato estatal. Además, el "Software público" se asocia generalmente con Software de dominio público.

10.2 Software Libre

La definición de Software libre (véase [21], [4], [17], [18], [16] y [20]) estipula los criterios que se tienen que cumplir para que un programa sea considerado libre. De vez en cuando se modifica esta definición para clarificarla o para resolver problemas sobre cuestiones delicadas. «Software libre» significa que el Software respeta la libertad de los usuarios y la comunidad. En términos generales, los usuarios tienen la libertad de copiar, distribuir, estudiar, modificar y mejorar el Software. Con estas libertades, los usuarios -tanto individualmente como en forma colectiva- controlan el programa y lo que hace.

Cuando los usuarios no controlan el programa, el programa controla a los usuarios. Los programadores controlan el programa y a través del programa, controlan a los usuarios. Un programa que no es libre, llamado «privativo o propietario», es considerado por muchos como un instrumento de poder injusto.

El Software libre es la denominación del Software que respeta la libertad de todos los usuarios que adquirieron el producto y por tanto, una vez obtenido el mismo puede ser usado, copiado, estudiado, modificado y redistribuido libremente de varias formas. Según la Free Software Foundation (véase [21]), el Software libre se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir y estudiar el mismo, e incluso modificar el Software y distribuirlo modificado.

Un programa es Software libre si los usuarios tienen las cuatro libertades esenciales:

0. La libertad de usar el programa, con cualquier propósito.
1. La libertad de estudiar cómo funciona el programa y modificarlo, adaptándolo a tus necesidades.
2. La libertad de distribuir copias del programa, con lo cual puedes ayudar a tu prójimo.
3. La libertad de mejorar el programa y hacer públicas esas mejoras a los demás, de modo que toda la comunidad se beneficie.

Un programa es Software libre si los usuarios tienen todas esas libertades. Por tanto, el usuario debe ser libre de redistribuir copias, tanto con o sin modificaciones, ya sea gratuitamente o cobrando una tarifa por la distribución,

a cualquiera en cualquier parte. El ser libre de hacer estas cosas significa, entre otras cosas, que no tiene que pedir ni pagar el permiso.

También debe tener la libertad de hacer modificaciones y usarlas en privado para su propio trabajo o pasatiempo, sin siquiera mencionar que existen. Si publica sus cambios, no debe estar obligado a notificarlo a nadie en particular, ni de ninguna manera.

La libertad de ejecutar el programa significa que cualquier tipo de persona u organización es libre de usarlo en cualquier tipo de sistema de computación, para cualquier tipo de trabajo y finalidad, sin que exista obligación alguna de comunicarlo al programador ni a ninguna otra entidad específica. En esta libertad, lo que importa es el propósito de los usuarios, no el de los programadores. El usuario es libre de ejecutar el programa para alcanzar sus propósitos y si lo distribuye a otra persona, también esa persona será libre de ejecutarlo para lo que necesite; nadie tiene derecho a imponer sus propios objetivos.

La libertad de redistribuir copias debe incluir las formas binarias o ejecutables del programa, así como el código fuente, tanto para las versiones modificadas como para las que no lo estén. Distribuir programas en forma de ejecutables es necesario para que los sistemas operativos libres se puedan instalar fácilmente. Resulta aceptable si no existe un modo de producir un formato binario o ejecutable para un programa específico, dado que algunos lenguajes no incorporan esa característica, pero debe tener la libertad de redistribuir dichos formatos si encontrara o programara una forma de hacerlo.

Para que se de la libertad que se menciona en los puntos 1 y 3 de realizar cambios y publicar las versiones modificadas tenga sentido, el usuario debe tener acceso al código fuente del programa. Por consiguiente, el acceso al código fuente es una condición necesaria para el Software libre. El «código fuente» compilado no es código fuente real y no cuenta como código fuente.

La libertad 1 incluye la libertad de usar su versión modificada en lugar de la original. Si el programa se entrega con un producto diseñado para ejecutar versiones modificadas de terceros, pero rechaza ejecutar las suyas, una práctica conocida como «tivoización» o «arranque seguro» [«Lockdown»] la libertad 1 se convierte más en una ficción teórica que en una libertad práctica, esto no es suficiente, en otras palabras, estos binarios no son Software libre, incluso si se compilaron desde un código fuente que es libre.

Una manera importante de modificar el programa es agregándole subrutinas y módulos libres ya disponibles. Si la licencia del programa especifica que no se pueden añadir módulos que ya existen y que están bajo una licencia

apropiada, por ejemplo si requiere que usted sea el titular de los derechos de autor del código que desea añadir, entonces se trata de una licencia demasiado restrictiva como para considerarla libre.

La libertad 3 incluye la libertad de publicar sus versiones modificadas como Software libre. Una licencia libre también puede permitir otras formas de publicarlas; en otras palabras, no tiene que ser una licencia de Copyleft. No obstante, una licencia que requiera que las versiones modificadas no sean libres, no se puede considerar libre.

«Software libre» no significa que «no es comercial». Un programa libre debe estar disponible para el uso comercial, la programación comercial y la distribución comercial. La programación comercial de Software libre ya no es inusual; el Software libre comercial es muy importante, ejemplo de ello es la empresa RedHat (ahora propiedad de IBM). Puede haber pagado dinero para obtener copias de Software libre, o puede haber obtenido copias sin costo. Pero sin tener en cuenta cómo obtuvo sus copias, siempre tiene la libertad de copiar y modificar el Software, incluso de vender copias.

El término Software no libre se emplea para referirse al Software distribuido bajo una licencia de Software más restrictiva que no garantiza estas cuatro libertades. Las leyes de la propiedad intelectual reservan la mayoría de los derechos de modificación, duplicación y redistribución para el dueño del Copyright; el Software dispuesto bajo una licencia de Software libre rescinde específicamente la mayoría de estos derechos reservados.

Los manuales de Software deben ser libres por las mismas razones que el Software debe ser libre y porque de hecho los manuales son parte del Software. También tiene sentido aplicar los mismos argumentos a otros tipos de obras de uso práctico, es decir, obras que incorporen conocimiento útil, tal como publicaciones educativas y de referencia. Wikipedia es el ejemplo más conocido.

La lista de proyectos de este tipo es realmente impresionante, algunos han conseguido un uso y alta calidad, por ejemplo el compilador GCC, el Kernel de Linux y el sistema operativo Debian GNU/Linux y Android. Mientras que otros proyectos han caído en el olvido, pero de la gran mayoría se tiene copia del código fuente que permitiría a quienes estén interesados en dicho proyecto poder reusarlo y en su caso ampliarlo.

La característica más importante que aparece típicamente en un proyecto de este tipo, es que un conjunto de personas separadas a gran distancia, sean capaces, a través de la Web, de los E-mail y de foros de aunar sus esfuerzos para crear, mejorar y distribuir un producto, de forma que todos

ellos se benefician unos de otros. Evidentemente, la mayor parte del peso recae en los desarrolladores, pero también es necesaria una difusión para que los usuarios documenten, encuentren errores, hagan foros de discusión, etc.

Si bien, el Software libre no es más seguro (en el sentido de invulnerable) que el propietario, la diferencia estriba en que el código fuente en el Software libre está disponible para todos y cualquiera puede aportar una solución, y por lo general al poco tiempo de detectarse una vulnerabilidad (a veces en cuestión de horas) se puede disponer de una solución para la misma. Además, al tener acceso al código fuente se puede detectar fácilmente si alguien introdujo código malicioso a una determinada aplicación.

¿Por qué se Interesan los Autores, Alumnos y Profesores Universitarios en el Software Libre? La ventaja principal es porque bajo el Software libre subyace la idea de compartir conocimiento y favorecer la existencia de nuevas ideas¹⁰⁸; y ¿qué es investigar y enseñar?, sino crear conocimiento y procurar que los alumnos aprendan e incluso vayan más allá de lo aprendido. Se comparte la idea, que el espíritu del Software libre es similar al que debería reinar en las instituciones educativas:

- Porque así no se condiciona a los estudiantes a usar siempre lo mismo.
- No se fomenta la piratería en los estudiantes y se evita pagar licencias que no son necesarias al existir alternativas gratuitas.
- Es mucho más seguro ya que el Software libre es público y se puede ver qué hace exactamente sin recelos.
- Se ofrece libertad de elección a los estudiantes y profesores al no limitarlos a usar una solución determinada, ampliando sus opciones y permitiendo un mayor aprendizaje.

Concretando estas ideas, profesores e investigadores necesitan herramientas para la investigación y docencia y estas deben tener una calidad mínima y ser fácilmente distribuibles entre los alumnos. En muchos casos las compañías desarrolladoras y distribuidoras de programas de cómputo no han

¹⁰⁸ ¿Por qué el Software creado con dinero de los impuestos no se publica como Software Libre?

¡El código pagado por los ciudadanos debería estar disponible para los ciudadanos y el mismo gobierno!

sabido ofrecer sus productos con la flexibilidad adecuada para las labores docentes o, en otros casos, los productos desarrollados no tienen la calidad esperada.

El Software libre es aún joven, pese a las decenas de miles de proyectos actuales -en los que se trabaja constantemente en mejorar la parte computacional de los algoritmos involucrados en el proyecto, haciendo y puliendo interfaces gráficas, generando ayuda en línea así como la documentación necesaria para que usuarios noveles y avanzados usen la mayor cantidad de opciones programadas- existen muchas otras necesidades profesionales y de investigación que requieren el desarrollo innovador de programas de cómputo para automatizarlas y hacerlas eficientes. Esto queda plasmado en las decenas de proyectos que a diario son registrados en las páginas especializadas en busca de difusión y apoyo para su proyecto.

En los últimos años, muchos proyectos han pasado de ser simples programas en línea de comandos a complejas aplicaciones multiplataforma -se ejecutan en distintos sistemas operativos como son Windows, Linux, Unix, Mac OS, Android- con ambientes gráficos multimedia que en muchos casos han superado a sus contrapartes comerciales -por ejemplo los navegadores Web-. Para muestra de este maravilloso avance, tomemos el proyecto del sistema operativo Android, que actualmente se ejecuta en millones de equipos -como celulares, tabletas, electrodomésticos, etc.- y en los cuales se pueden descargar miles de aplicaciones y está soportado por una gran cantidad de usuarios y empresas comerciales como Google, IBM y últimamente Microsoft -que años atrás era acérrima enemiga del Software libre-.

El Software libre ha logrado desplazar a muchos de sus competidores por sus múltiples bondades y bajo costo de desarrollo -es el caso de Windows Phone que fue reemplazado por Android de Google-, al reusar miles de aplicaciones ya existentes que usan Software libre y permitir desarrollar otro tanto de aplicaciones bajo una plataforma que se ejecuta en los más diversos procesadores. Además, el uso de Software libre y su posibilidad de ampliarlo y/o especializarlo según sea necesario, ha permitido crear de forma cada vez más rápida y confiable; para poner a disposición de un gran público programas de uso común, así como especializado que satisfagan las nuevas necesidades de los usuarios.

Software Libre en Ciencia y Educación Algunos puntos y reflexiones sobre porqué se considera que es interesante el Software libre en Ciencia y

Educación son:

- Accesible a todo el mundo aunque no sea rentable su desarrollo: El Software libre entre sus libertades permite que se pueda ejecutar por terceros, copiarlo, distribuirlo y estudiarlo/modificarlo. Eso hace que si en ciencia se usa Software libre el acceso a esos programas no suponga una barrera (se puede distribuir y ejecutar).
- Muchos de los desarrollos en el campo de la accesibilidad se realizan en universidades y son distribuidos como Software libre: Aunque no sea rentable muchas veces el desarrollo de herramientas de accesibilidad (no sea algo monetizable) en la universidad se consigue escapar a esa la lógica capitalista de solamente invertir en lo que pueda ofrecer beneficio económico.
- Transparencia: En ciencia es importante ver las costuras para comprobar si es verdad lo que se afirma, tener acceso al código fuente del Software empleado permite poder estudiarlo por si realiza algún cálculo mal.
- Propicia el espíritu crítico: Si no tienes acceso a las revistas o el acceso es privativo para los bolsillos de mucha gente no puedes comprobar la información. Se nos pide que seamos críticos con la información que se nos da del mundo científico pero no podemos entrenar el espíritu crítico sin acceso al conocimiento libre.
- Caramelos con droga en la puerta del colegio: Muchas empresas buscan introducir en los colegios, institutos y universidad su Software. Ofrecer un programa que permita trabajar y genere una dependencia quedando los datos muchas veces en las nubes (ordenadores de otras personas). Un ejemplo es Microsoft con Office 365. Dando cuentas gratuitas durante un tiempo para que se use su Software. Otro ejemplo podría ser Unity3D en vez de por ejemplo Godot.
- Especificaciones de protocolos abiertas VS cerradas: Gracias a que los protocolos TCP/IP, HTTP, POP, SNMP, DHCP, etc. son abiertos es posible construir herramientas por cualquier con conocimientos de programación. Con protocolos cerrados solamente quienes tuvieran acceso a las especificaciones podrían desarrollar y conocer cómo funcionan.

- Uso de estándares: Existiendo un estándar para documentos ofimáticos (procesador de textos, hoja de cálculo, presentaciones, etc.) algunas empresas como Microsoft se empeñan en ir con su propio formato y estándar en vez de sumarse a que sea más sencillo ir a una y que el usuario pueda optar por que herramientas usar para editar o trabajar con documentos ofimáticos.
- Software libre para la Ciencia ciudadana: Un ejemplo en el que es importante la colaboración ciudadana es el cambio climático y la defensa medioambiental del territorio. Desde imvec.tech usan herramientas de Software libre para medición y monitorización de contaminación.

Software Libre: Beneficios Más Allá de la Informática El uso de las tecnologías de código abierto supone cultivar el conocimiento y la puesta en valor de la libertad individual, lo personal y lo privado, todo ello sin menospreciar lo público y la construcción de una sociedad. El movimiento del Software libre, con Linux a la cabeza, ha capitaneado durante décadas planteamientos para un cambio en el modo de producción: el abaratamiento de costes empresariales para grandes y pequeños, el trabajo en línea, el desarrollo de Software y Hardware a pequeña escala, el replanteamiento del negocio informático, el sistema de normas éticas que rigen los grupos, la documentación abierta, etc.

Todo ello derivado de una simple idea: la libertad. Ha sido la clave que ha llevado a todo este movimiento hacia una autonomía y motivación que pocas veces se ve en otros sectores. El Open Source está lleno de alternativas con la libertad como pilar y consecuencia filosófica, de hecho muchos Forks (derivaciones) de proyectos aparecen cuando la disputa sobre la misma toma relevancia. Esta manera de hacer las cosas debería trasladarse directamente a la sociedad promoviendo esos valores para evitar caer en un mundo despótico que toma fuerza a pasos agigantados.

No estaría mal promover la comprensión de las licencias libres. Leer y entender una licencia GPL, MIT o BSD es infinitamente más sencillo y rápido que hacerlo con otras, siendo unas normas fáciles de cumplir porque encajan con un modelo ético y práctico comprensible por muchos.

Podríamos decir que GPL, BSD y MIT son las Constituciones que vertebran todo el movimiento del Software Libre, cosechando derechos de uso y logros como el ahorro o la legítima copia privada. Lo mismo podría ocurrir con las licencias Creative Commons para cierto contenido, un arma poderosa

en el sector divulgativo que está poco extendida por desconocimiento y el Status Quo de la propiedad intelectual.

La cooperación libre y voluntaria supuso un éxito hasta ahora pero está siendo amenazada por la dinámica actual de la Web, donde la centralización de las principales plataformas supone estar bajo el yugo de normas que rastrean con lupa y cambian sus términos con demasiada frecuencia. Ya ni digamos cuando eso se junta con el ansia de monetización: si quieres dinero más te vale no cabrear a los anunciantes o no tener un Strike por uso de contenido que podría ser reclamado.

Los claroscuros de este sistema hace que los creadores cada vez hagan menos de forma libre y altruista, y ello podría verse potenciado por las búsquedas con inteligencia artificial, lo que apunta a un empobrecimiento del contenido cultural fresco y renovador. Las polémicas con GitHub Copilot o ChatGPT sobre el entrenamiento de las IAs hace sobrevolar una vez más la cuestión del Copyright y el uso legítimo de los resultados brindados por éstas, lo que también condiciona la creación.

La libertad de expresión, de uso, de creación, de modificación ... esas cuestiones llevan irremediablemente a una libertad de pensamiento y acción, a una adaptación creativa que puede ser la motivación para romper moldes en todos los aspectos. El código abierto y sus licencias son, por tanto, un beneficio personal y social mucho más grande que el simple hecho de usar Linux o Software libre. El contenido despreocupado, que no prioriza el dinero y el posicionamiento/visualizaciones, se vuelve esencial para el inconformismo.

10.3 Seguridad del Software

Si bien, el Software Libre no es más seguro (en el sentido de invulnerable) que el propietario, la diferencia puede estribar en que el código fuente en el Software libre está disponible para todos y cualquiera puede aportar una solución y por lo general al poco tiempo de detectarse una vulnerabilidad (a veces en cuestión de horas) se puede disponer de una solución para la misma. Además, al tener acceso al código fuente se puede detectar si alguien introdujo código malicioso a una determinada aplicación.

Pero de todos es sabido, que los usuarios de Software de código abierto, como por ejemplo los que de manera habitual trabajan con equipos comandados por sistemas Linux, por regla general se sienten orgullosos de la seguridad que estos programas aportan con respecto a los sistemas cerrados propios de otras firmas, dígame Microsoft Windows o Mac de Apple.

¿Es Seguro el Software Libre? En primer lugar definiremos el concepto de "seguridad" como salvaguarda de las propiedades básicas de la información. Entre las características que debe cumplir para ser seguro, encontramos la integridad, es decir, que sólo los usuarios autorizados pueden crear y modificar los componentes del sistema, la confidencialidad, sólo estos usuarios pueden acceder a esos componentes, la disponibilidad, que todos los componentes estén a disposición de los usuarios siempre que lo deseen y el "no repudio", o lo que es lo mismo, la aceptación de un protocolo de comunicación entre el servidor y un cliente, por ejemplo, mediante certificados digitales.

Entre las diferencias de seguridad entre un Software Libre y el Software Propietario, podemos destacar:

- Seguridad en el Software Propietario: En el caso de tener "agujeros de seguridad", puede que no nos demos cuenta y que no podamos repararlos. Existe una dependencia del fabricante, retrasándose así cualquier reparación y la falsa creencia de que es más seguro por ser oscuro (la seguridad por oscuridad determina los fallos de seguridad no parcheados en cada producto).
- Seguridad en el Software Libre: Por su carácter público y su crecimiento progresivo, se van añadiendo funciones y se nos permite detectar más fácilmente los agujeros de seguridad para poder corregirlos. Los problemas tardan mucho menos en ser resueltos por el apoyo que tiene de los Hackers y una gran comunidad de desarrolladores y al ser un Software de código libre, cualquier empresa puede aportar soporte técnico.

Sin embargo esta es una pregunta sobre la que los expertos al día de hoy, tras muchos años de discusiones, siguen sin ponerse de acuerdo. ¿Es más seguro el Software de código abierto que los programas cerrados, o viceversa? Lo cierto es que, en términos generales, ambos bandos tienen sus razones con las que defender sus argumentos. Por un lado, los usuarios de las aplicaciones y sistemas de código abierto, defienden que, al estar el código fuente disponible a los ojos de todo el mundo, es mucho más fácil localizar posibles agujeros de seguridad y vulnerabilidades que pongan en peligro los datos de los usuarios.

Por otro lado, aquellos que consideran que los sistemas cerrados son más seguros en este sentido, afirman que al tener acceso tan solo los expertos al código fuente de sus aplicaciones, es más complicado que se produzcan

filtraciones o inserciones de Software malicioso en este tipo de sistemas. Hay que tener en cuenta que, por ejemplo, Google premia a las personas que descubren fallos de seguridad en su Software como Chrome, aunque no es el único gigante de la tecnología en utilizar estas tácticas.

De hecho muchas empresas están gastando miles de millones de dólares y/o euros en hacer que sus propuestas sean lo más seguras posible, argumentando que la seguridad de sus proyectos es una de sus prioridades, todo con el fin de intentar frenar que los atacantes vulneren sus sistemas. Por otro lado, otros aseguran que cuando el código fuente es público, más ojos están disponibles para detectar posibles vulnerabilidades o errores en dicho código, por lo que siempre será más rápido y sencillo poner soluciones con el fin de ganar en seguridad.

Sea como sea, en cualquiera de los dos casos, lo que ha quedado más que demostrado es que la seguridad no está garantizada en ningún momento, ya sean propuestas de código abierto, o no. Pero también es cierto que lo que se procura es que los riesgos de ser atacados se reduzcan en medida de lo posible. Los sistemas Linux son considerados desde hace mucho tiempo como un sistema operativo seguro, en buena parte debido a las ventajas que ofrece su diseño. Dado que su código está abierto, son muchas las personas que incorporan mejoras de las que el resto de usuarios de Linux se benefician, a diferencia de las propuestas de Windows o MacOS, donde estas correcciones generalmente se limitan a las que detectan Microsoft y Apple.

No obstante, en nuestra defensa del Software libre, diremos que su código abierto permite que los errores sean encontrados y solucionados con mayor rapidez, por lo que determinamos que es el Software más recomendable.

En general, puede afirmarse que el Software libre es más seguro, ya que debido a su carácter abierto y distribuido, un gran número de programadores y personas expertas pueden estar atentas al código fuente -especialmente en los grandes proyectos-, lo cual permite hacer auditorías con objeto de detectar errores y puertas traseras (Backdoor, en inglés) que pongan en riesgo nuestros datos.

Así, los grandes programas y proyectos de Software libre, con una extensa comunidad de desarrollo y usuarios que lo respalden, presentan niveles muy altos de seguridad, un alto grado de protección y una rápida respuesta a posibles vulnerabilidades.

10.4 Tipos de Licencias

Tanto la Open Source Initiative como la Free Software Foundation mantienen en sus páginas Web (véase [21], [4], y [20]) listados oficiales de las licencias de Software libre que aprueban.

Una licencia es aquella autorización formal con carácter contractual que un autor de un Software da a un interesado para ejercer "actos de explotación legales". Pueden existir tantas licencias como acuerdos concretos se den entre el autor y el licenciatario. Desde el punto de vista del Software libre, existen distintas variantes del concepto o grupos de licencias:

Licencias GPL Una de las más utilizadas es la Licencia Pública General de GNU (**GNU GPL**). El autor conserva los derechos de autoría (Copyright), y permite la redistribución y modificación bajo términos diseñados para asegurarse de que todas las versiones modificadas del Software permanecen bajo los términos más restrictivos de la propia GNU GPL. Esto hace que sea imposible crear un producto con partes no licenciadas GPL, el conjunto tiene que ser GPL.

En la práctica, esto hace que las licencias de Software libre se dividan en dos grandes grupos, aquellas que pueden ser mezcladas con código licenciado bajo GNU GPL (y que inevitablemente desaparecerán en el proceso, al ser el código resultante licenciado bajo GNU GPL) y las que no lo permiten al incluir mayores u otros requisitos que no contemplan ni admiten la GNU GPL y que por lo tanto no pueden ser enlazadas ni mezcladas con código gobernado por la licencia GNU GPL.

GPL Versión 1 la versión 1 de GNU GPL, fue presentada el 25 de febrero de 1989, impidió lo que eran las dos principales formas con las que los distribuidores de Software restringían las libertades definidas por el Software libre. El primer problema fue que los distribuidores publicaban únicamente los archivos binarios, funcionales y ejecutables, pero no entendibles o modificables por humanos. Para prevenir esto, la GPLv1 estableció que cualquier proveedor de Software libre además de distribuir el archivo binario debía liberar a su vez código fuente entendible y que pudiera ser modificado por el ser humano bajo la misma licencia (secciones 3a y 3b de la licencia).

El segundo problema era que los distribuidores podían añadir restricciones adicionales, añadiendo restricciones a la licencia o mediante la combinación del Software con otro que tuviera otras restricciones en su distribución. Si

esto se hacía, entonces la unión de los dos conjuntos de restricciones sería aplicada al trabajo combinado, entonces podrían añadirse restricciones inaceptables. Para prevenir esto, GPLv1 obligaba a que las versiones modificadas en su conjunto, tuvieran que ser distribuidas bajo los términos GPLv1 (secciones 2b y 4 de la licencia). Por lo tanto, el Software distribuido bajo GPLv1 puede ser combinado con Software bajo términos más permisivos y no con Software con licencias más restrictivas, lo que entraría en conflicto con el requisito de que todo Software tiene que ser distribuido bajo los términos de la GPLv1.

GPL Versión 2 según Richard Stallman, el mayor cambio en GPLv2 fue la cláusula "Liberty or Death" («libertad o muerte»). Esta sección dice que si alguien impone restricciones que le prohíben distribuir código GPL de tal forma que influya en las libertades de los usuarios (por ejemplo, si una ley impone que esa persona únicamente pueda distribuir el Software en binario), esa persona no puede distribuir Software GPL. La esperanza es que esto hará que sea menos tentador para las empresas el recurrir a las amenazas de patentes para exigir una remuneración de los desarrolladores de Software libre.

En 1991 se hizo evidente que una licencia menos restrictiva sería estratégicamente útil para la biblioteca C y para las bibliotecas de Software que esencialmente hacían el trabajo que llevaban a cabo otras bibliotecas comerciales ya existentes. Cuando la versión 2 de GPL fue liberada en junio de 1991, una segunda licencia Library General Public License fue introducida al mismo tiempo y numerada con la versión 2 para denotar que ambas son complementarias. Los números de versiones divergieron en 1999 cuando la versión 2.1 de LGPL fue liberada, esta fue renombrada como GNU Lesser General Public License para reflejar su lugar en esta filosofía.

GPL Versión 3 A finales de 2005, la Free Software Foundation (FSF) anunció estar trabajando en la versión 3 de la GPL (GPLv3). El 16 de enero de 2006, el primer borrador de GPLv3 fue publicado y se inició la consulta pública. La consulta pública se planeó originalmente para durar de nueve a quince meses, pero finalmente se extendió a dieciocho meses, durante los cuales se publicaron cuatro borradores. La GPLv3 oficial fue liberada por la FSF el 29 de junio de 2007.

Según Stallman los cambios más importantes se produjeron en el campo

de las patentes de Software, la compatibilidad de licencias de Software libre, la definición de código fuente y restricciones a las modificaciones de Hardware. Otros cambios están relacionados con la internacionalización, cómo son manejadas las violaciones de licencias y cómo los permisos adicionales pueden ser concedidos por el titular de los derechos de autor. También añade disposiciones para quitar al DRM su valor legal, por lo que es posible romper el DRM (Digital Rights Management) en el Software de GPL sin romper leyes como la DMCA (Digital Millennium Copyright Act).

GPLv2 vs GPL v3 GPLv3 contiene la intención básica de GPLv2 y es una licencia de código abierto con un Copyleft estricto. Sin embargo, el idioma del texto de la licencia fue fuertemente modificado y es mucho más completo en respuesta a los cambios técnicos, legales y al intercambio internacional de licencias.

La nueva versión de la licencia contiene una serie de cláusulas que abordan preguntas que no fueron o fueron cubiertas de manera insuficiente en la versión 2 de la GPL. Las nuevas regulaciones más importantes son las siguientes:

- GPLv3 contiene normas de compatibilidad que hacen que sea más fácil combinar el código GPL con el código que se publicó bajo diferentes licencias. Esto se refiere en particular al código bajo la licencia de Apache v. 2.0.
- Se insertaron normas sobre gestión de derechos digitales para evitar que el Software GPL se modifique a voluntad, ya que los usuarios recurrieron a las disposiciones legales para protegerse mediante medidas técnicas de protección (como la DMCA o la directiva sobre derechos de autor).
- La licencia GPLv3 contiene una licencia de patente explícita, según la cual las personas que licencian un programa bajo licencia GPL otorgan derechos de autor y patentes, en la medida en que esto sea necesario para utilizar el código que ellos otorgan. Por lo tanto, no se concede una licencia de patente completa. Además, la nueva cláusula de patente intenta proteger al usuario de las consecuencias de los acuerdos entre los titulares de patentes y los licenciatarios de la licencia pública general que solo benefician a algunos de los licenciatarios (correspondientes al

acuerdo Microsoft / Novell). Los licenciarios deben garantizar que todos los usuarios disfruten de tales ventajas (licencia de patente o liberación de reclamos) o que nadie puede beneficiarse de ellos.

- A diferencia de la GPLv2, la GPLv3 establece claramente que no es necesario divulgar el código fuente en un uso ASP (Application Service Provider) de los programas GPL, siempre que no se envíe una copia del Software al cliente. Si el efecto Copyleft debe extenderse al uso de ASP, debe aplicarse la Licencia pública general de Affero, versión 3 (AGPL) que solo difiere de la GPLv3 en esta consideración.

Licencias Estilo BSD Llamadas así porque se utilizan en gran cantidad de Software distribuido junto a los sistemas operativos BSD. El autor, bajo tales licencias, mantiene la protección de Copyright únicamente para la renuncia de garantía y para requerir la adecuada atribución de la autoría en trabajos derivados, pero permite la libre redistribución y modificación, incluso si dichos trabajos tienen propietario. Son muy permisivas, tanto que son fácilmente absorbidas al ser mezcladas con la licencia GNU GPL con quienes son compatibles. Puede argumentarse que esta licencia asegura "verdadero" Software libre, en el sentido que el usuario tiene libertad ilimitada con respecto al Software y que puede decidir incluso redistribuirlo como no libre.

Licencia Copyleft Hay que hacer constar que el titular de los derechos de autor (Copyright) de un Software bajo licencia Copyleft puede también realizar una versión modificada bajo su Copyright original y venderla bajo cualquier licencia que desee, además de distribuir la versión original como Software libre. Esta técnica ha sido usada como un modelo de negocio por una serie de empresas que realizan Software libre (por ejemplo MySQL); esta práctica no restringe ninguno de los derechos otorgados a los usuarios de la versión Copyleft.

Licencia estilo MIT Las licencias MIT son de las más permisivas, casi se consideran Software de dominio público. Lo único que requieren es incluir la licencia MIT para indicar que el Software incluye código con licencia MIT.

Licencia Apache License La licencia Apache trata de preservar los derechos de autor, incluir la licencia en el Software distribuido y una lista de

los cambios realizados. En modificaciones extensivas del Software original permite licenciar el Software bajo otra licencia sin incluir esas modificaciones en el código fuente.

Licencia Mozilla Public License MPL Esta licencia requiere que los archivos al ser distribuidos conserven la misma licencia original pero pueden ser usados junto con archivos con otra licencia, al contrario de la licencia GPL que requiere que todo el código usado junto con código GPL sea licenciado como código GPL. También en caso de hacer modificaciones extensivas permite distribuirlas bajo diferentes términos y sin incluir el código fuente en las modificaciones.

Licencia Código de Dominio Público Es un código que no está sujeto a derechos de autor que puede utilizarse sin restricciones.

Licencia Creative Commons Las licencias de Creative Commons son más utilizadas para cualquier creación digital que para el Software, entendiendo como creación digital desde fotos, artículos en blogs, música, vídeos, este trabajo, etc. Hay varios tipos de licencias de Creative Commons diferenciando entre permitir modificaciones a la obra original, solicitando crédito de la creación o permitiendo un uso comercial de la obra.

Licencias de Código Abierto Las licencias de código abierto son un intermedio entre las licencias privativas y las licencias de Software libre. Las licencias de código abierto permiten el acceso al código fuente pero no todas se consideran licencias de Software libre al no otorgar otros derechos que se requieren para considerar un Software como Software libre como el derecho al uso o con cualquier propósito, modificación y distribución.

Dado el éxito del Software libre como modelo de desarrollo de Software algunas empresas cuyo Software era privativo pueden decidir hacerlo de código abierto con la intención de suplir algunas carencias de Software privativo pero sin perder ciertos derechos que son la fuente de sus ingresos como la venta de licencias.

Las expresiones «Software libre» y «código abierto» se refieren casi al mismo conjunto de programas. No obstante, dicen cosas muy diferentes acerca de dichos programas, basándose en valores diferentes. El movimiento del Software libre defiende la libertad de los usuarios de ordenadores, en un

movimiento en pro de la libertad y la justicia. Por contra, la idea del código abierto valora principalmente las ventajas prácticas y no defiende principios. Esta es la razón por la que gran parte de la comunidad de Software libre está en desacuerdo con el movimiento del código abierto y nosotros no empleamos esta expresión en este texto.

Licencia Microsoft Public License La Microsoft Public License es una licencia de código abierto que permite la distribución del Software bajo la misma licencia y la modificación para un uso privado. Tiene restricciones en cuanto a las marcas registradas.

En caso de distribuir el Software de forma compilada o en forma de objeto binario no se exige proporcionar los derechos de acceso al código fuente del Software compilado o en forma de objeto binario. En este caso esta licencia no otorga más derechos de los que se reciben, pero si permite otorgar menos derechos al distribuir el Software (compilado o en forma de objeto binario).

Modelo de Desarrollo de Software Bazar y Catedral El tipo de licencia no determina qué Software es mejor o peor, si el privativo o el Software libre, la diferencia entre las licencias está en sus características éticas y legales. Aunque el modelo de desarrollo con una licencia de código abierto a la larga suele tener un mejor desarrollo y éxito que el Software privativo, más aún con un medio como internet que permite colaborar a cualquier persona independiente de donde esté ubicada en el mundo.

Comparación con el Software de Código Abierto Aunque en la práctica el Software de código abierto y el Software libre comparten muchas de sus licencias, la Free Software Foundation opina que el movimiento del Software de código abierto es filosóficamente diferente del movimiento del Software libre. Los defensores del término «código abierto (Open Source)», afirman que éste evita la ambigüedad del término en ese idioma que es «Free» en «Free Software».

Mucha gente reconoce el beneficio cualitativo del proceso de desarrollo de Software cuando los desarrolladores pueden usar, modificar y redistribuir el código fuente de un programa. El movimiento del Software libre hace especial énfasis en los aspectos morales o éticos del Software, viendo la excelencia técnica como un producto secundario de su estándar ético. El movimiento de código abierto ve la excelencia técnica como el objetivo prioritario, siendo

el compartir el código fuente un medio para dicho fin. Por dicho motivo, la FSF se distancia tanto del movimiento de código abierto como del término «código abierto (Open Source)».

Puesto que la «iniciativa de Software libre Open Source Initiative (OSI)» sólo aprueba las licencias que se ajustan a la «definición de código abierto (Open Source Definition)», la mayoría de la gente lo interpreta como un esquema de distribución, e intercambia libremente "código abierto" con "Software libre". Aún cuando existen importantes diferencias filosóficas entre ambos términos, especialmente en términos de las motivaciones para el desarrollo y el uso de tal Software, raramente suelen tener impacto en el proceso de colaboración.

Aunque el término "código abierto" elimina la ambigüedad de libertad frente a precio (en el caso del inglés), introduce una nueva: entre los programas que se ajustan a la definición de código abierto, que dan a los usuarios la libertad de mejorarlos y los programas que simplemente tienen el código fuente disponible, posiblemente con fuertes restricciones sobre el uso de dicho código fuente. Mucha gente cree que cualquier Software que tenga el código fuente disponible es de código abierto, puesto que lo pueden manipular, sin embargo, mucho de este Software no da a sus usuarios la libertad de distribuir sus modificaciones, restringe el uso comercial, o en general restringe los derechos de los usuarios.

10.4.1 Licencias Creative Commons

Las Licencias¹⁰⁹ **Creative Commons** (CC) de forma general no tienen una definición oficial, sin embargo, entre las muchas definiciones aceptadas están la de la UNESCO, la cual expresa la siguiente descripción:

Las Licencias Creative Commons (CC) son modelos de contratos que sirven para otorgar públicamente el derecho de utilizar una publicación protegida por los derechos de autor. Entre menos restricciones implique una licencia, mayores serán las posibilidades de utilizar y distribuir un contenido. Las Licencias CC permiten a cualquier usuario descargar, copiar, distribuir, traducir, reutilizar, adaptar y desarrollar su contenido sin costo alguno.

Sin embargo, en la Web oficial de la Organización Creative Commons se nos dice sobre las mismas lo siguiente:

¹⁰⁹Las licencias de Creative Commons son más utilizadas para cualquier creación digital que para el Software, entendiendo como creación digital desde fotos, artículos en blogs, música, vídeos, este trabajo, etc.

Las Licencias Creative Commons (CC) brindan a todos, desde creadores individuales hasta grandes instituciones, una forma estandarizada de otorgar permiso al público para usar su trabajo creativo bajo la ley de derechos de autor. Desde la perspectiva del reutilizador, la presencia de una licencia Creative Commons sobre una obra protegida por derechos de autor responde a la pregunta: ¿Qué puedo hacer con esta obra?.

Las «Licencias Creative Commons» que hoy en día pertenecen a la organización mundial Creative Commons¹¹⁰, y buscan regularizar y mantener, de forma equilibrada y satisfactoria, todo lo relacionado con el derecho de utilizar una publicación protegida por los derechos de autor a nivel mundial, han logrado un buen trabajo, sin duda alguna. Y seguramente en el tiempo, se irán adaptando a las nuevas realidades sociales y tecnológicas para poder seguir manteniendo de forma armónica las posibilidades de utilizar y distribuir cualquier contenido libre y abierto sobre la Internet, y más allá.

¿Cuáles son y cómo funcionan o para qué se usan? Las 7 distintas Licencias Creative Commons son las siguientes:

CC BY Esta licencia permite a los reutilizadores distribuir, remezclar, adaptar y desarrollar el material en cualquier medio o formato, siempre que se otorgue la atribución al creador. La licencia permite el uso comercial.

CC BY-SA Esta licencia permite a los reutilizadores distribuir, remezclar, adaptar y desarrollar el material en cualquier medio o formato, siempre que se otorgue la atribución al creador. La licencia permite el uso comercial. Si remezcla, adapta o construye sobre el material, debe licenciar el material modificado bajo términos idénticos.

CC BY-NC Esta licencia permite a los reutilizadores distribuir, remezclar, adaptar y desarrollar el material en cualquier medio o formato, únicamente con fines no comerciales y siempre que se otorgue la atribución al creador.

¹¹⁰Una organización mundial sin ánimo de lucro que permite compartir y reutilizar la creatividad y el conocimiento mediante el suministro de herramientas legales gratuitas. Y cuyas herramientas legales (licencias) ayudan a quienes quieren fomentar la reutilización de sus obras ofreciéndolas para su uso bajo términos generosos y estandarizados; a quienes quieren hacer usos creativos de las obras; y a quienes quieren beneficiarse de esta simbiosis.

CC BY-NC-SA Esta licencia permite a los reutilizadores distribuir, remezclar, adaptar y desarrollar el material en cualquier medio o formato, únicamente con fines no comerciales y siempre que se otorgue la atribución al creador. Si remezcla, adapta o construye sobre el material, debe licenciar el material modificado bajo términos idénticos.

CC BY-ND Esta licencia permite a los reutilizadores copiar y distribuir el material en cualquier medio o formato únicamente en forma no adaptada, y siempre y cuando se otorgue la atribución al creador. La licencia permite el uso comercial.

CC BY-NC-ND Esta licencia permite a los reutilizadores copiar y distribuir el material en cualquier medio o formato únicamente en forma no adaptada, únicamente con fines no comerciales y siempre que se otorgue la atribución al creador.

CC0 (CC Cero) Esta licencia es una herramienta de dedicación pública que permite a los creadores renunciar a sus derechos de autor y poner sus obras en el dominio público mundial. CC0 permite a los reutilizadores distribuir, remezclar, adaptar y desarrollar el material en cualquier medio o formato, sin condiciones.

10.4.2 Nuevas Licencias para Responder a Nuevas Necesidades

El mundo de la tecnología avanza mucho más rápido que las leyes y estas tienen que esforzarse para alcanzarlo. En el caso del Software libre y de código abierto, tanto la Free Software Foundation como la Open Source Initiative (los organismos encargados de regular las diferentes licencias) enfrentan periódicamente el problema de cómo mantener sus principios y al mismo tiempo evitar que alguien se aproveche indebidamente.

En el último tiempo, la Open Source Initiative le dio el sello de aprobación a otras nuevas licencias para propósitos específicos.

Nuevas Licencias de Código Abierto

- Cryptographic Autonomy License version 1.0 (CAL-1.0)

Fue creada en el 2019 por el equipo del proyecto de código abierto Holochain, esta licencia fue desarrollada para ser utilizada con aplicaciones criptográficas distribuidas. El inconveniente con las licencias tradicionales es que no obligaba a compartir los datos. Esto podría perjudicar el funcionamiento de toda la red. Por eso la CAL también incluye la obligación de proporcionar a terceros los permisos y materiales necesarios para utilizar y modificar el Software de forma independiente sin que ese tercero tenga una pérdida de datos o capacidad.

- Open Hardware Licence (OHL)

De la mano de la Organización Europea para la Investigación Nuclear (CERN) llegó esta licencia con tres variantes enfocadas en la posibilidad de compartir libremente tanto Hardware como Software.

Hay que hacer una aclaración. La OSI fue creada en principio pensando en el Software por lo que no tiene mecanismos para la aprobación de licencias de Hardware. Pero, como la propuesta del CERN se refiere a ambos rubros, esto posibilitó la aprobación:

- CERN-OHL-S es una licencia fuertemente recíproca: El que utilice un diseño bajo esta licencia deberá poner a disposición las fuentes de sus modificaciones y agregados bajo la misma licencia.
- CERN-OHL-W es una licencia débilmente recíproca: Sólo obliga a distribuir las fuentes de la parte del diseño que fue puesta originalmente bajo ella. No así los agregados y modificaciones.
- CERN-OHL-P es una licencia permisiva: Permite a la gente tomar un proyecto, relicenciarlo y utilizarlo sin ninguna obligación de distribuir las fuentes.

Hay que decir que la gente del CERN parece haber encontrado la solución a un problema que viene afectando a algunos proyectos de código abierto. Una gran empresa utiliza ese proyecto para comercializar servicios y no solo hace ningún aporte al proyecto original (ya sea con código o dando apoyo financiero) si no que también compite en el mismo mercado.

Post Open Zero-Cost en el año 2024 se desarrolla la licencia «Post Open Zero-Cost» con la cual busca abordar los desafíos surgidos en la interacción entre desarrolladores de código abierto y empresas comerciales, especialmente en lo que respecta a la compensación justa por el uso comercial del código.

La característica distintiva de la licencia «Post-Open» en comparación con las licencias abiertas existentes, como la GPL, es la introducción de un componente contractual que puede ser rescindido en caso de violación de los términos. Esta licencia ofrece dos tipos de acuerdos contractuales: gratuitos y de pago. El acuerdo de pago permite negociar derechos adicionales para la distribución comercial de productos o modificaciones sin requerir su divulgación pública.

Las situaciones que podrían llevar a la terminación del acuerdo contractual incluyen: violación de los términos de la licencia; reclamaciones por infracción de patentes; imposición de condiciones adicionales (como sanciones en contratos con clientes por divulgación de información sensible); cambios sujetos a leyes de control de exportaciones; ocultamiento de información sobre vulnerabilidades; y uso del código para entrenamiento de modelos de aprendizaje automático bajo términos no permitidos. Las relaciones contractuales no se rescinden de inmediato, sino que se notifica la infracción y se otorgan 60 días para corregirla antes de la rescisión efectiva del acuerdo.

Uno de los problemas que la nueva licencia busca abordar está relacionado con las limitaciones de la GPL, la cual se centra en otorgar derechos sin la capacidad de revocarlos, lo que permite a las empresas encontrar formas de eludir sus requisitos, especialmente en lo que respecta al acceso al código fuente. Estas lagunas son utilizadas para restringir la disponibilidad del código subyacente en productos comerciales mediante la imposición de términos contractuales adicionales con los usuarios finales.

Un claro ejemplo es el de RHEL, la cual los clientes firman un acuerdo con Red Hat que limita la redistribución del código fuente al imponer condiciones sobre la coincidencia de las copias instaladas y compradas de RHEL. Esto coloca a los usuarios en la disyuntiva entre su libertad para disponer del software y mantener su estatus de cliente de Red Hat. Aunque la GPL permite la distribución de parches que solucionan vulnerabilidades en el código de RHEL, esto podría interpretarse como una violación del acuerdo con Red Hat y podría resultar en la terminación de los servicios por parte de la empresa.

10.5 Implicaciones Económico-Políticas del Software Libre

Una vez que un producto de Software libre ha empezado a circular, rápidamente está disponible a un costo muy bajo. Al mismo tiempo, su utilidad no decrece. El Software, en general, podría ser considerado un bien de uso inagotable, tomando en cuenta que su costo marginal es pequeñísimo y que no es un bien sujeto a rivalidad -la posesión del bien por un agente económico no impide que otro lo posea-.

10.5.1 Software Libre y la Piratería

Puesto que el Software libre permite el libre uso, modificación y redistribución, a menudo encuentra un hogar entre usuarios para los cuales el coste del Software no libre es a veces prohibitivo, o como alternativa a la piratería. También es sencillo modificarlo localmente, lo que permite que sean posibles los esfuerzos de traducción a idiomas que no son necesariamente rentables comercialmente, además:

- Porque así no se condiciona a los usuarios a usar siempre lo mismo.
- Porque así no se fomenta la piratería en los usuarios al no pagar licencias.
- Porque así no se obliga a usar una solución concreta y se ofrece libertad de elección a los usuarios.
- Porque es mucho más seguro ya que el Software libre es público y se puede ver qué hace exactamente sin recelos.

La mayoría del Software libre se produce por equipos internacionales que cooperan a través de la libre asociación. Los equipos están típicamente compuestos por individuos con una amplia variedad de motivaciones y pueden provenir tanto del sector privado, del sector voluntario o del sector público. En los últimos años se ha visto un incremento notable de grandes corporativos (como IBM, Microsoft, Intel, Google, Samsung, Red Hat, etc.) que han dedicado una creciente cantidad de recursos humanos y computacionales para desarrollar Software libre, ya que esto apoya a sus propios negocios.

10.5.2 ¿Cuánto Cuesta el Software Libre?

En esta sección intentaremos dar una idea de cuál es el costo del desarrollo del Software Libre, por supuesto que no se tratará más que de una conjetura aproximada basada en las cifras proporcionadas por desarrolladores de Software comercial (al año 2020).

Gratis no Significa Gratuito Supongamos que todos los recursos humanos participantes en el desarrollo de un proyecto de Software libre lo hagan de forma voluntaria. De todas formas tenemos lo que los contables llaman «Costo de oportunidad» esto es, los ingresos que podrían haber generado esas personas si hubieran dedicado el tiempo y los conocimientos invertidos en el proyecto a uno en el que les pagaran. Así, el calcular el costo promedio por hora que cobra un programador, por la cantidad de horas invertidas al proyecto, nos da un razonable costo mínimo. Lo mismo puede hacerse con los voluntarios dedicados a la difusión en las redes. El costo de una campaña de marketing digital puede estimarse fácilmente.

Muchos proyectos de código abierto como una distribución Linux, son contruidos a partir de la integración de otros proyectos, por los que sus costos de desarrollo también deberían sumarse.

Por otra parte, necesitamos recursos físicos. Aún cuando los voluntarios trabajen desde su casa, siguen teniendo que comprar y mantener sus equipos, además de pagar la electricidad que los hace funcionar.

Bases para el Cálculo Hay muchos factores que determinan el costo de desarrollar una pieza de Software. En un extremo tenemos una aplicación simple que requiere muy poca interacción del usuario o procesamiento del lado del servidor. Tal es el caso de un cliente de escritorio para redes sociales. Por el otro sistemas operativos que deben operar en múltiples plataformas realizando múltiples tareas (por ejemplo Debían que aspira a ser el sistema operativo universal). Sin embargo, el costo de una aplicación simple puede elevarse debido a que tiene múltiples pantallas diferentes. Por ejemplo un juego desarrollado con HTML5 y Javascript.

Los dos aspectos claves son la cantidad de horas de trabajo necesarias y las tecnologías involucradas. Para una aplicación de escritorio como un procesador de textos con las prestaciones habituales, optimizado para un determinado escritorio Linux, se estima que se tendría que contar con al menos el equivalente a 42,000 euros en trabajo voluntario. Un gestor de contenidos

para comercio electrónico con seguimiento de pedidos e integración con las principales plataformas de pago implicaría desembolsar unos 210,000 euros o su equivalente en trabajo voluntario.

Tomando en cuenta que este cálculo incluye lo que costó el desarrollo de las bibliotecas y otros proyectos libres y de código abierto incluidos, pero no los gastos que efectivamente deben desembolsarse en efectivo como la compra de equipos, Software de seguridad y desarrollo y el pago de electricidad e internet.

Por otro lado, el proceso de medición de costes del Software es un factor realmente importante en el análisis de un proyecto. Hay distintos métodos de estimación de costes de desarrollo de Software (también conocido como métrica del Software). La gran mayoría de estos métodos se basan en la medición del número de Líneas de Código que contiene el desarrollo (se excluyen comentarios y líneas en blanco de los fuentes).

Desarrollo de Fedora 9 La Linux Foundation ha calculado que costaría desarrollar el código de la distribución Fedora 9 que fue puesta a disposición del público el 13 de mayo de 2008, en el informe citado "Estimating the Total Development Cost of a Linux Distribution" se calcula que Fedora 9 tiene un valor de 10.8 mil millones de dólares y que el coste únicamente del Kernel (2.6.25 con 8,396,250 líneas de código) tendría un valor de 1.4 mil millones de dólares.

Esta distribución tiene unas 205 millones de líneas de código, el proyecto debería ser desarrollado por 1000 - 5000 desarrolladores (el trabajo invertido por una única persona desarrollándolo se alargaría durante unos 60.000 años) y esa estimación no va muy desencaminada ya que en los 2 últimos años del desarrollo de esa versión contribuyeron unos 3,200 desarrolladores aunque el número de trabajadores en la historia de la distribución es mucho mayor.

¿Qué pasa con GNU/Linux? en el año 2015 (las estadísticas más actuales que conseguimos) la Linux Foundation analizó el costo de desarrollo del núcleo. Combinando el aporte de los recursos humanos (voluntarios y de pago) y los desembolsos necesarios, la cuenta sumó 476,767,860,000.13 euros.

Todos sabemos que el hecho de tener desarrolladores asalariados no garantiza necesariamente Software de calidad. Pero, tener desarrolladores que pueden dedicar toda su atención a un proyecto en lugar de hacerlo en sus horas libres si lo hace. Lamentablemente, por el momento el único modo de

lograr eso es obtener el apoyo de corporaciones (Intel, Google, IBM, AMD, Sun Microsystems, Dell, Lenovo, Asus, HP, SGI, Oracle, RedHat, etc.) que solo lo hacen con los proyectos que son de su interés como el Kernel de Linux, hay que notar que para el Kernel de Linux un porcentaje importante (más del 10 %) lo hacen programadores independientes.

Costes Recordemos que la segunda de las cuatro libertades de un programa para ser Software libre es:

- Libre redistribución

y esta puede ser a través de un pago o sin costo. Es por ello que existen distintas empresas, organizaciones y usuarios que pueden apoyar a los usuarios finales en el desarrollo y soporte de algún programa de Software libre o una distribución personalizada de Linux por un costo determinado.

Mucha gente, en especial ejecutivos de empresas, se acercan a Linux bajo la promesa de que es una solución de bajo costo -muchos piensan que incluso es gratis-. Pero la realidad es que detrás de Linux y los programas de Software libre (y aquí la traducción correcta de la palabra inglesa, Free es libre, no gratis) pueden llevar una serie de costos ocultos que deben ser considerados al momento de decidir si se implementa una solución propietaria o una libre.

Los costos ocultos aparecen cuando se intenta instalar y capacitar en el uso de algún Software y se necesita la ayuda de un informático, al que se tiene que pagar, o alguna empresa quiere personalizar la interfaz de un programa y necesita la ayuda de un programador, que también tienen un coste, por lo que finalmente el comentario suele ser "el Software libre no es barato".

El primer punto a considerar al evaluar ambas alternativas es el costo de la licencia. Los productos de Software libre no suelen tener un costo de licencia asociado, que sí existe en los programas propietarios. De hecho es allí en donde los fabricantes de Software recargan sus costos de investigación y desarrollo, de producción e incluso sus ganancias. En este primer punto el ganador claro es el Software libre y es lo que los adeptos de este esquema publicitan: "su compañía puede ahorrar miles de dólares al año usando Software libre".

El segundo punto a considerar es el costo de instalación, configuración y capacitación. Dependiendo de su complejidad, algunos productos comerciales no contemplan costos extra por este concepto y otros -como Windows, por ejemplo- son tan populares que se puede encontrar numerosas opciones

de instalación -a través de empresas o profesionales- donde escoger en el mercado. A veces en el Software libre la configuración puede implicar recompilar el producto con algunas opciones particulares, algo que sólo pueden realizar técnicos con un nivel adecuado de conocimientos y que puede que no sean tan fáciles de encontrar. Aquí por lo general la ventaja va hacia el Software comercial.

Una vez instalado el Software, toca realizar actualizaciones de mantenimiento. Si bien es cierto lo que algunos de los fanáticos del Software libre dicen, que nadie lo obliga a mejorar el Software con que cuenta, la realidad -especialmente en lo que a seguridad se refiere- obliga a las empresas a mantener su Software actualizado. Aún así, los costos de actualizar Software libre suelen ser significativamente más bajos que los de productos comerciales y suelen ser menos exigentes con el Hardware necesario para ejecutarlos. La mayoría de las veces la ventaja es para el Software libre, pero hay que evaluar ya que varía dependiendo de cada caso.

Por último hay algunas casas que desarrollan productos de Software libre -dan el código y permiten que cualquiera lo modifique o reutilice- pero fijan contratos con cargos mensuales o anuales para mantenimiento del mismo, algo que se parece mucho a un cobro por licencia, por lo que hay que estar seguro de conocer todas las condiciones cuando una empresa nos ofrece una solución propia y la califica como Software libre.

Además de estas consideraciones hay una que debe sumarse eventualmente a esta evaluación: el costo de migrar a otra solución. En Software libre suelen usarse estándares abiertos para almacenar los datos, lo que facilita las migraciones. En cambio muchas soluciones propietarias suelen tener formatos propietarios que pueden dejar "amarrados" los datos de la empresa a una aplicación específica.

Sólo después de evaluar estos aspectos del Software, que pueden tener implicaciones importantes en el presupuesto, es que un CIO (Chief Information Officer) puede decir si una solución de Software libre le conviene más a una empresa o no, algo que va más allá de que la aplicación sea gratis o no.

10.5.3 La Nube y el Código Abierto

Desde hace años se han creado nuevos desafíos para el código abierto que plantea la nube, un término que para el usuario promedio puede significar cosas diferentes, pero que para la empresa se resume en servicios. Y es que los beneficios económicos que genera el mero Software de código abierto no

son comparables a los que se obtienen cuando se ofrece ese mismo Software a través de servicios, más allá -pero incluyendo- del soporte.

Este hecho diferencial lleva tiempo provocando fricciones entre desarrolladores y proveedores y hay quien adelantó incluso el fin del modelo de desarrollo del código abierto tal y como lo conocemos. ¿Quién tiene la razón?, ¿es para tanto la situación?

En sus exposiciones, representantes de compañías y proyectos de código abierto muy populares en el ámbito empresarial, explican el supuesto perjuicio que les ocasiona el uso que los grandes proveedores de servicios en la nube hacen del Software que ellos desarrollan y cómo algunos han considerado y aplicado un enfoque más cerrado para sus productos con el fin de evitar lo que denominan como expolio. Hay declaraciones que merecen ser rescatadas para dotar de contexto a la discusión:

- El papel que juega el código abierto en la creación de oportunidades comerciales ha cambiado, durante muchos años les permitimos que las empresas de servicios tomaran lo que se ha desarrollado y ganasen dinero con ello.
- Empresas como Amazon Web Services, Azure de Microsoft, etc. Han ganado cientos de millones de dólares ofreciendo a sus clientes servicios basados en Software libre sin contribuir tanto a la comunidad de código abierto que construye y mantiene ese proyecto. Es imposible saber exactamente de cuánto dinero estamos hablando, pero es cierto que los proveedores de la nube se benefician del trabajo de los desarrolladores de código abierto que no emplean.
- Hay un mito ampliamente instalado en el mundo de código abierto que dice que los proyectos son impulsados por una comunidad de contribuyentes, pero en realidad, los desarrolladores pagados contribuyen con la mayor parte del código en la mayoría de los proyectos de código abierto modernos.

En resumen, todas estas voces se quejan de dos cosas: los grandes beneficios que obtienen los proveedores de servicios en la nube con su Software sin retribuirles en consecuencia, y la falta de colaboración manteniendo los productos con los que lucran. Sin embargo, no nos engañemos, el quid de la cuestión está principalmente en el dinero: la opinión generalizada de la

comunidad es que el Software de código abierto nunca fue pensado para que las empresas de servicios en la nube lo tomaran y lo vendieran.

Por otro lado, si es posible bifurcar un proyecto libre que se cierra, ¿no hubiese sido mejor colaborar con él antes y haber evitado el cierre? Si no se invierte y se mantiene con salud aquello que da beneficios, puede terminar por desaparecer. A medio camino entre el depredador y el parásito: así es como ven muchas desarrolladoras de código abierto a los proveedores de servicios en la nube.

Vale la pena retomar ahora la frase «el Software de código abierto nunca fue pensado para que las empresas de servicios en la nube lo tomaran y lo vendieran». ¿Para qué fue pensado el código abierto entonces? No hay ninguna licencia de código abierto o Software libre reconocida por la Free Software Foundation o la Open Source Initiative que prohíba hacer negocio con el Software. Lo que prohíben es la discriminación en la capacidad y alcance de su uso en función de la parte, se trate de un individuo o de la mayor multinacional imaginable.

¿Cuál es la solución a un embrollo de tamaño envergadura? Lo único claro es que no es una cuestión de blancos y negros y las consideraciones son demasiadas como para seguir ahondando: empresas que cotizan en bolsa quieren más dinero de otras compañías -que también cotizan en bolsa y por mucho más-, que además del Software ponen la infraestructura sobre la que distribuyen sus ofertas y que tienen la capacidad de clonar tu producto en un abrir y cerrar de ojos, no solo porque tienen el capital, sino porque tienen la experiencia necesaria tras contribuir técnica, pero también en muchos casos, económicamente, durante largo tiempo.

Pese a ello, esta situación está alterando el paradigma actual, en el que el modelo de desarrollo del código abierto se ha impuesto como impulsor de la innovación en el sector empresarial y ya hay quien habla de que nos acercamos al fin, o al principio del fin de la Era del Open Source, cuya preponderancia estaría sentenciada por la revolución de la nube, a la postre el mayor estímulo que haya tenido el código abierto hasta la fecha.

El futuro, pues, pasaría por el Shared Source Software, bajo el cual diferentes compañías con intereses alineados colaborarán en el desarrollo de proyectos concretos, pero limitando su explotación comercial a sí mismas. Todavía no estamos ahí, no obstante, y no parece tampoco que el relevo se vaya a dar en breve. De suceder, será muy llamativo: la muerte del código abierto por un éxito mal entendido.

10.5.4 El Código Abierto como Base de la Competitividad

En septiembre del 2021, se publicó un amplio y detallado informe llevado a cabo por el Fraunhofer ISI y por OpenForum Europe para la Comisión Europea, «The impact of open source software and hardware on technological independence, competitiveness and innovation in the EU economy», cuantifica la importancia económica del código abierto aplicado tanto al Software como al Hardware, su efecto en la contribución al producto interior bruto generado, la reducción en aspectos como el coste total de propiedad, dependencia del proveedor y autonomía digital; lanza una serie de recomendaciones específicas de políticas públicas destinadas a lograr un sector público digitalmente autónomo, una investigación y desarrollo abierto que fomente el crecimiento europeo y una industria más digitalizada y competitiva.

En las estimaciones del informe se apunta que las empresas europeas invirtieron alrededor de mil millones de euros en Software de código abierto en 2018, lo que resultó en un impacto en la economía europea de entre 65,000 y 95,000 millones de euros. El análisis estima una relación costo-beneficio superior a 1:4 y predice que un aumento del 10% de las contribuciones de a repositorios de código abierto sería susceptible de generar anualmente entre un 0.4% y un 0.6% adicional en el PIB, así como más de seiscientas nuevas empresas tecnológicas en la Unión Europea.

El análisis de las contribuciones a repositorios de Software de código abierto en la Unión Europea revela que el ecosistema tiene una naturaleza diferente frente al norteamericano, con un volumen de contribuciones que provienen sobre todo de empleados de compañías pequeñas o muy pequeñas, frente a un escenario en los Estados Unidos en el que predominan grandes compañías tecnológicas que se benefician en sus modelos de negocio de la gran cantidad y de la rápida mejora del Software disponible. En Europa, los contribuyentes individuales ascendieron a más de 260,000, lo que representa el 8% de los casi 3.1 millones de empleados de la UE en el sector del desarrollo de Software en 2018. En total, los más de 30 millones de desarrollos consolidados en repositorios en los estados miembros de la Unión Europea representan una inversión de personal equivalente a casi mil millones de euros, que han pasado a estar disponibles en el dominio público y que, por lo tanto, no tienen que ser desarrollados por otros actores.

Según el análisis, cuanto más pequeña es la empresa, mayor es la inversión relativa en Software de código abierto (las empresas con 50 empleados o menos asumieron casi la mitad de los desarrollos en la muestra de las com-

pañías más activas). Aunque más del 50% de los contribuyentes pertenecen a la industria tecnológica (el 8% del total de sus empleados participaron en estos desarrollos), también hubo participación significativa de empresas de consultoría, científicas, técnicas y, en menor medida, de distribuidores, minoristas y empresas del ámbito financiero.

¿Puede una filosofía de desarrollo como el código abierto, disponible para todo el mundo, llegar a convertirse en una fuente de ventajas diferenciales para el resto de los países, que se ha visto tradicionalmente muy superado en su relevancia en el entorno tecnológico por los gigantes tecnológicos de Estados Unidos o de China? El informe afirma que su uso puede llegar a incidir en gran medida en el desarrollo de una independencia tecnológica superior, de una mayor competitividad y de más innovación. Veremos si llegamos a ver en el resto del mundo políticas que incentiven el uso del código abierto como una variable estratégica clave para ello. La idea, capitalizar la tecnología de una forma más orientada al procomún y al desarrollo colaborativo, suena sin duda atractiva e interesante.

10.5.5 Software Libre en Empresas y Corporaciones

En esta sección exploraremos algunas de las claves por las cuales el Software libre está hoy en el punto de mira de todo tipo de empresas y grandes corporaciones (algunas de las cuales ayer eran sus acérrimos enemigos). Pero hay que empezar destacando que corporativos como Google, Amazon Web Services, Azure de Microsoft, Microsoft, IBM, entre otras, en los últimos años han ganado miles de millones de dólares ofreciendo a sus clientes servicios y/o productos basados en Software libre y con una queja recurrente por su pobre o nula contribución a la comunidad de código abierto que construyó y mantiene esos proyectos.

Si bien es imposible saber exactamente de cuánto dinero estamos hablando, pero es cierto que empresas y corporaciones se benefician diariamente del trabajo de los desarrolladores de código abierto que no emplean.

Grandes Equipos de Programadores GNU/Linux ha demostrado que los equipos de desarrolladores grandes, distribuidos, aunque desorganizados pueden crear Software viable.

Antes de la llegada de GNU/Linux, la mayoría del Software era desarrollado por pequeños equipos de programadores que trabajaban en estrecha coordinación entre sí. Ese era el enfoque recomendado por informáticos de hace

unos decenios, que advertían que añadir más programadores a un proyecto tendía a disminuir su eficiencia. Y estaban muy equivocados.

Desde el principio, el Kernel de Linux se desarrolló con un enfoque diferente, en el que programadores de todo el mundo, que en la mayoría de los casos no se conocían, escribieron e integraron el código de forma rápida y poco organizada. Gracias a la publicación temprana y frecuente, consiguieron que funcionara y hoy día es el Kernel más usado en informática en supercomputadoras y dispositivos móviles.

Pero actualmente hay un mito ampliamente instalado en el mundo de código abierto, que dice que los proyectos son impulsados por una comunidad de contribuyentes gratuitos, pero en realidad, los desarrolladores pagados contribuyen con la mayor parte del código en la mayoría de los proyectos de código abierto modernos de los cuales las corporaciones pueden sacar provecho. Claro ejemplo es el propio Kernel de Linux, en el cual una gran cantidad de desarrolladores actuales pertenecen o son subvencionados por empresas, fundaciones o corporaciones (actualmente cientos de ellas), como en el caso de Linus Torvalds que trabaja bajo los auspicios de la Fundación de Linux.

Reutilización de Software Parte de la razón por la que Linux se hizo muy popular entre los ingenieros de Software con relativa rapidez fue que Linux -y el Software libre en general- facilita la reutilización del código escrito por otras personas.

Hoy en día, la reutilización de Software de terceros es habitual, incluso entre los equipos de desarrollo cuyos productos no son de código libre. Es difícil imaginar la construcción de una aplicación hoy en día sin hacer uso de las bibliotecas de Software de origen, las API de terceros u otros recursos externos a su propio proyecto.

Es cierto que proyectos como GNU, que precedió al Kernel de Linux en siete años, promovían la reutilización de código antes de que apareciera el núcleo. Pero, podría decirse que Linux fue el proyecto que trajo las prácticas de codificación libre a la corriente que tanto parece interesar a ciertas grandes empresas, ayudando a crear el modelo de ingeniería de Software de componentes de Software modulares y reutilizables.

Gestión Actual del Código Fuente Linus Torvalds, que creó el núcleo de Linux cuando era estudiante en Helsinki, es el más famoso por ese trabajo.

Pero un hecho a menudo olvidado es que Torvalds es también el padre de Git, el masivamente popular gestor de código fuente libre.

Torvalds creó Git para ayudar a gestionar el código fuente de Linux. Si Linux no existiera, tampoco existiría Git. Tampoco existiría GitHub, ni GitLab, ni GitOps. Y, lo que es más importante, sin la idea de Software libre y colaborativo de Richard Stallman, tampoco existiría la cultura de intercambio y colaboración abierta que sostienen estas tecnologías.

Estrategias de Despliegue de Software "App Store" Apple puede atribuirse el mérito de haber lanzado la primera App Store, un lugar donde los desarrolladores pueden compartir aplicaciones y los usuarios pueden instalarlas fácilmente, utilizando un catálogo Online centralizado.

Pero al igual que con muchas cosas que ha hecho Apple, el concepto de App Store (que ahora es una estrategia de despliegue de Software apilado como servicio, especialmente pero no sólo en el ecosistema móvil) se parece mucho a lo que los desarrolladores de GNU/Linux estaban haciendo a través de los repositorios de Software mucho antes de que las tiendas de aplicaciones se convirtieran en algo común en el mundo del Software propietario, como también lo es la Tienda de Windows.

Los repositorios de Software en GNU/Linux hacen más o menos lo mismo que las tiendas de aplicaciones: Permiten a los usuarios seleccionar las aplicaciones que quieren de una lista centralizada y en línea, y luego instalarlas con unos pocos clics o bien órdenes de terminal (como el famoso *apt* de Debian).

Es cierto que empresas como Apple parecen tener el mérito de crear tiendas de aplicaciones muy fáciles de usar de hacer clic e ir, pero no es un invento de ellos. Y la historia del concepto de tienda de aplicaciones en general implica a más actores que sólo la comunidad de Apple. Aún así, creo que se podría argumentar con fuerza que, sin GNU/Linux y los repositorios de Software de GNU/Linux, las tiendas de aplicaciones tal y como las conocemos hoy no existirían.

Formatos Abiertos para Intercambio de Información Hay una gran variedad de tecnologías disponibles para producir y almacenar datos. Como son: hojas de cálculo, bases de datos, Software estadístico más específico y más. Esto genera una enorme diversidad de formatos, a veces esto es por decir lo menos caótico.

La ventaja de los archivos de formatos abiertos, es que permiten a los de-

sarrolladores producir varios paquetes de Software y servicios utilizando esos formatos. Esto entonces reduce al mínimo los obstáculos para la reutilización de la información que contienen.

El advenimiento del Software libre ha generado algunos de los formatos abiertos más usados para el intercambio de información, pero los entes generadores de información no siempre se adecuan a los niveles de apertura deseados, algunos de estos formatos abiertos son: XML, JSON, YAML, RDF, REBOL, PDF, CSV, ODF, OOXML, TXT, HTML, HDF.

Pero, incluso si la información se proporciona en formato electrónico, formato legible por máquina y en detalle, puede existir problemas relacionados con el formato del archivo en sí (principalmente el generado por los diversos sistemas operativos). Los formatos en los cuales la información es publicada -en otras palabras, la base digital en la cual la información es almacenada- puede ser "abierta" o "cerrada".

Un formato abierto es aquel donde las especificaciones del Software están disponibles para cualquier persona, de forma gratuita, así cualquiera puede usar dichas especificaciones en su propio Software sin ninguna limitación en su reutilización que fuere impuesta por derechos de propiedad intelectual.

Si el formato del archivo es "cerrado", esto puede ser debido a que el formato es propietario y sus especificaciones no están disponibles públicamente, o porque el formato es propietario y aunque las especificaciones se han hecho públicas, su reutilización es limitada. Si la información es liberada en un formato de archivo cerrado, esto puede causar grandes obstáculos para reutilizar la información codificada en él, forzando a aquellos que deseen usar la información a comprar Software innecesario.

El uso de formatos de archivo con propiedad, para el que la especificación no está disponible públicamente, puede crear dependencia de Software de terceros o de los titulares de licencias de los formatos de archivos. En el peor de los casos, esto puede significar que la información sólo se puede leer con cierto Software específico, que puede ser caro, o que puede quedar obsoleto.

La preferencia del término Gobierno de Datos Abiertos, es que la información se publicará en formatos de archivo abiertos, los cuales son de lectura mecánica y esto es una aportación más del Software libre.

Ciencia Abierta La ciencia abierta (Open Science) es el movimiento creciente para hacer que la ciencia sea abierta. La ciencia en sí misma se utilizó como un ejemplo principal de la eficacia del movimiento de código abierto, ci-

tando prácticas como la difusión abierta de información, métodos y revisión por pares de la literatura científica. Podría decirse que la ciencia abierta comenzó en el siglo XVII con el advenimiento de la revista científica y la práctica de repetir los experimentos presentados en los artículos académicos. Estas revistas se imprimían y distribuirían en todo el mundo, a menudo supervisadas por sociedades científicas como la Royal Society.

¿Qué impulsó la necesidad de un movimiento de ciencia abierta? La Royal Society tenía el famoso lema "Nullius in verba", traducido de forma aproximada como "no tome la palabra de nadie". Esto encarnaba un principio general en la ciencia de que todas las teorías están abiertas a ser cuestionadas y los resultados declarados deben ser repetibles. De hecho, es una práctica generalizada que fue realizada por la sociedad en esos primeros años. En los últimos años esta práctica no ha sido tan común, con más y más ciencia confiando en elementos cerrados, lo que en última instancia conduce a errores que son más difíciles de detectar sin un intercambio completo de información: datos, métodos y publicaciones.

El movimiento de ciencia abierta afirma en términos generales que la ciencia debe realizarse de manera abierta y reproducible donde todos los componentes de la investigación estén abiertos. Muchas revistas permanecen estancadas en un formato en el que se imprimían físicamente, a pesar de que en la actualidad se distribuyen en gran medida en línea. A menudo, todavía utilizan archivos PDF como una forma de "papel electrónico" con publicaciones fijas, procesos cerrados de revisión por pares y poco o ningún acceso a los datos. Este fue sin duda el modo más eficiente de difundir el conocimiento científico antes de los albores de internet, pero ahora un número cada vez mayor lo considera lejos de ser el óptimo.

La ciencia abierta encarna una serie de aspectos, en el núcleo esto incluye acceso abierto, datos abiertos, código abierto y estándares abiertos que ofrecen una diseminación sin restricciones del discurso científico. Estas cosas permiten una ciencia reproducible al brindar acceso completo a los componentes principales de la investigación científica. Hay una serie de componentes adicionales que también se están explorando, como la revisión por pares abierta, donde los revisores de publicaciones científicas publican revisiones abiertamente con su nombre adjunto y la ciencia de libreta abierta donde las libretas (tradicionalmente cerradas) se publican abiertamente en línea a medida que se realiza la investigación.

¿Por qué la ciencia abierta es tan importante en la era digital? También existe una creciente comprensión de que, dado que la investigación científica

depende cada vez más del código informático para simulaciones, cálculos, análisis, visualización y procesamiento de datos en general, es importante tener acceso a este código tal como tradicionalmente ha sido importante mostrar (y derivar) cualquier nueva técnica matemática introducida para el análisis. Hay revistas como PLOS ONE y F1000 que exploran el significado de las publicaciones, ya sea que se deben congelar en el tiempo o se pueden actualizar. Los repositorios de datos también están ganando importancia a medida que las agencias de financiación requieren la publicación y preservación de los datos generados por la investigación financiada.

En esencia, la ciencia abierta se trata de volver a esos valores fundamentales inculcados por algunos de los primeros científicos de que no debemos confiar en la palabra de nadie, que es esencial que todos los elementos pertinentes a un descubrimiento pretendido se publiquen para que los resultados puedan repetirse y validarse. El movimiento de la ciencia abierta varía en el grado en que lo requiere, pero están surgiendo patrones. Se están estableciendo recomendaciones sobre licencias, como CC0 para datos, CC-BY para publicaciones, licencias compatibles con OSI para código fuente y formatos abiertos para datos. En última instancia, se trata de empoderar a todos para que participen en la ciencia, con internet como vehículo principal para la amplia difusión de este conocimiento.

Este movimiento está cambiando la forma en que se hace la ciencia, está recibiendo el respaldo de muchas agencias de financiamiento, ya que requieren planes de gestión de datos, planes de distribución de código fuente y una mayor validación de los resultados a través del acceso abierto a estos resultados para todos. Esto también mejora la transferencia de conocimientos de la academia a la industria, ya que se brinda acceso completo en el momento de la publicación o después de un período de embargo. El movimiento de la ciencia abierta se limita en gran medida a la investigación que está financiada por las agencias de financiación nacionales de todo el mundo y exige que todos los que financian la investigación tengan acceso total e igualitario a ella.

Open Hardware El concepto de Software libre también se permeó al Hardware. El término Open Hardware u Open Source Hardware, se refiere al Hardware cuyo diseño se hace públicamente disponible para que cualquiera pueda estudiarlo, modificarlo y distribuirlo, además de poder producir y vender Hardware basado en ese diseño. Tanto el Hardware como el Software

que lo habilita, siguen la filosofía del Software libre. Hoy en día, el término "hágalo usted mismo" (DIY por sus siglas en inglés) se está popularizando en el Hardware gracias a proyectos como Arduino que es una fuente abierta de prototipos electrónicos, una plataforma basada en Hardware flexible y fácil de utilizar que nació en Italia en el año 2005.

El movimiento de Hardware abierto o libre, busca crear una gran librería accesible para todo el mundo, lo que ayudaría a las compañías a reducir en millones de dólares en trabajos de diseño redundantes. Ya que es más fácil tener una lluvia de ideas propuesta por miles o millones de personas, que por solo una compañía propietaria del Hardware, tratando así de que la gente interesada entienda cómo funciona un dispositivo electrónico, pueda fabricarlo, programarlo y poner en práctica esas ideas en alianza con las empresas fabricantes, además se reduciría considerablemente la obsolescencia programada y en consecuencia evitaríamos tanta basura electrónica que contamina el medio ambiente. Al hablar de Open Hardware hay que especificar de qué tipo de Hardware se está hablando, ya que está clasificado en dos tipos:

- Hardware estático. Se refiere al conjunto de elementos materiales de los sistemas electrónicos (tarjetas de circuito impreso, resistencias, capacitores, LEDs, sensores, etcétera).
- Hardware reconfigurable. Es aquél que es descrito mediante un HDL (Hardware Description Language). Se desarrolla de manera similar a como se hace Software. Los diseños son archivos de texto que contienen el código fuente.

Para tener Hardware reconfigurable debemos usar algún lenguaje de programación con licencia GPL (General Public License). La licencia GPL, al ser un documento que cede ciertos derechos al usuario, asume la forma de un contrato, por lo que usualmente se le denomina contrato de licencia o acuerdo de licencia. La Organización Europea para la investigación Nuclear (CERN) publicó el 8 de julio de 2011 la versión 1.1 de la Licencia de Hardware Abierto.

Existen programas para diseñar circuitos electrónicos y aprender de la electrónica como EDA (Electronic Design Automation) y GEDA (GPL Electronic Design Automation), son aplicaciones de Software libre que permiten poner en práctica las ideas basadas en electrónica.

Es posible realizar el ciclo completo de diseño de Hardware reconfigurable desde una máquina con GNU/Linux, realizándose la compilación, simulación,

síntesis y descarga en una FPGA (Field Programmable Gate Arrays). Para la compilación y simulación se puede usar GHDL (<https://ghdl.free.fr>) junto con GTKWave (<https://gtkwave.sourceforge.net>) y para la síntesis el entorno ISE de Xilinx. Este último es Software comercial pero existe una versión gratuita con algunas restricciones.

Sabemos que tanto en el caso del Software como el Hardware, libre no es lo mismo que gratis. Específicamente, en el caso del Hardware, como estamos hablando de componentes físicos que son fabricados, la adquisición de componentes electrónicos puede ser costosa. Aun así, es un campo que no solo es apasionante sino que también tiene mucho futuro y representa grandes oportunidades.

Entusiasmo de la Comunidad Por último, pero no menos importante, probablemente el mayor impacto duradero de GNU/Linux en el modelo de ingeniería de Software se reduce a lo que podría llamarse entusiasmo de la comunidad. Me refiero a la forma en que GNU/Linux en particular, y el Software libre en general, ha animado a los desarrolladores de todo tipo a considerar las contribuciones a la comunidad como uno de sus objetivos finales y esto ahora es notorio no solo en Software, sino en Hardware abierto, obras literarias, escritos técnicos (como este trabajo), imágenes, vídeo, música y un largo etc.

En un mundo de código libre en el que las contribuciones a los proyectos de código pueden ser aceleradores de carrera y el código de licencia libre se reutiliza ampliamente, los desarrolladores entienden que hay un valor real en la construcción de Software que puede beneficiar a tantos usuarios como sea posible.

Tal vez los desarrolladores valorarían a la comunidad en su conjunto si GNU/Linux y el código libre nunca hubieran aparecido. Pero me cuesta imaginar un mundo en el que corporaciones como Microsoft y Google trabajarán juntos en la construcción de Software para GNU/Linux si GNU/Linux no hubiera popularizado el concepto de proyectos de Software impulsados por la comunidad que nadie posee realmente, pero que todos pueden utilizar.

Si bien, es innegable que todo lo anterior puso en la mira de las empresas de todos los tamaños y de las grandes corporaciones el Software libre, la principal razón es el poder utilizar una gran cantidad de Software funcional, depurado y ampliamente usado para ofrecer servicios y/o productos basados en Software libre y así beneficiarse económicamente de ello.

Por otro lado, se ha visto a través de múltiples estudios, el impacto y la cuantificación de la importancia económica del código abierto aplicado tanto al Software como al Hardware, su efecto en la contribución al producto interior bruto generado, la reducción en aspectos como el coste total de propiedad, dependencia del proveedor y autonomía digital. Además de generar políticas públicas destinadas a lograr un sector público digitalmente autónomo, una investigación y desarrollo abierto que fomente el crecimiento de los países y una industria más digitalizada y competitiva.

Retomando la frase «el Software de código abierto nunca fue pensado para que las empresas de servicios lo tomaran y lo vendieran». ¿Para qué fue pensado el código abierto, entonces? No hay ninguna licencia de código abierto o Software libre reconocida por la Free Software Foundation o la Open Source Initiative que prohíba hacer negocio con el Software. Lo que prohíben es la discriminación en la capacidad y alcance de su uso en función de la parte, se trate de un individuo o de la mayor multinacional imaginable.

Pese a ello, esta situación está alterando el paradigma actual, en el que el modelo de desarrollo del código abierto se ha impuesto como impulsor de la innovación en el sector empresarial, a la postre el mayor estímulo que haya tenido el código abierto hasta la fecha.

¿Puede una filosofía de desarrollo como el código abierto, disponible para todo el mundo, llegar a convertirse en una fuente de ventajas diferenciales para el resto de los países, que se ha visto tradicionalmente muy superado en su relevancia en el entorno tecnológico por los gigantes tecnológicos de Estados Unidos o de China? Se afirma que su uso puede llegar a incidir en gran medida en el desarrollo de una independencia tecnológica superior, de una mayor competitividad y de más innovación. La idea, capitalizar la tecnología de una forma más orientada al procomún y al desarrollo colaborativo, suena sin duda atractiva e interesante pero no libre de inconvenientes para algunos sectores de desarrolladores de Software libre.

10.6 Código Abierto y las Organizaciones Internacionales

Aunque la Organización de las Naciones Unidas (ONU) ha hablado previamente bien del desarrollo del código abierto, varios eventos recientes muestran que la ONU está tomando medidas definitivas para presentar al mundo entero el camino del código abierto. En julio del 2021, el Consejo Económico y Social de la ONU (ECOSOC) adoptó un proyecto de resolución presentado por el representante de Pakistán titulado: Tecnologías de fuente abierta para

el desarrollo sostenible.

10.6.1 Las Naciones Unidas y el Código Abierto

El ECOSOC destacó la disponibilidad de tecnologías de código abierto que pueden contribuir a los Objetivos de Desarrollo Sostenible (ODS). El consejo invitó al Secretario General a "desarrollar propuestas específicas sobre formas de aprovechar mejor las tecnologías de código abierto para el desarrollo sostenible basadas en las aportaciones de los Estados Miembros interesados y otras partes interesadas".

El desarrollo de tecnología de código abierto puede ser una herramienta rápida y eficaz para la innovación. Aplicarlo a tecnologías apropiadas para ayudar a alcanzar los ODS es extremadamente prometedor. Las "tecnologías apropiadas" abarcan opciones y aplicaciones tecnológicas que son a pequeña escala, económicamente asequibles, descentralizadas, energéticamente eficientes, ambientalmente racionales y fácilmente utilizadas por las comunidades locales para satisfacer sus necesidades.

Existe un caso particularmente fuerte para las tecnologías apropiadas de código abierto OSAT (Outsourced Semiconductor Assembly and Test). OSAT podría ayudar a todos a salir de la pobreza y alcanzar un estado sostenible aprovechando el mismo tipo de desarrollo que hace que el Software de código abierto sea un éxito rotundo.

La Declaración Ministerial del Foro político de alto nivel sobre desarrollo sostenible también destacó la importancia de "tecnologías no patentadas que pueden contribuir a los Objetivos de Desarrollo Sostenible, a través de diversas fuentes de acceso abierto". Pidió "el desarrollo y la puesta en funcionamiento de una plataforma en línea en el marco del Mecanismo de facilitación de la tecnología para establecer un mapeo integral y servir como puerta de entrada a la información sobre iniciativas, mecanismos y programas de ciencia, tecnología e innovación existentes, dentro y fuera de las Naciones Unidas".

Es un pequeño paso, pero muy emocionante, porque las Naciones Unidas no se demoran una vez que ven formas de ayudar a sus Estados Miembros y a las personas que los integran. Ahora, el Departamento de Asuntos Económicos y Sociales de las Naciones Unidas (DESA) está trabajando para que esto suceda. DESA está utilizando una Nota sobre una base de datos centralizada propuesta de las Naciones Unidas de tecnologías apropiadas de código abierto publicada por la Conferencia de las Naciones Unidas sobre Comercio

y Desarrollo (UNCTAD) para hacerlo.

La Nota de la UNCTAD aboga por una base de datos centralizada de OSAT para acelerar el descubrimiento y la innovación en todos los sectores asociados con los ODS al tiempo que se minimizan los obstáculos legales o financieros. Esto es importante para la difusión del acervo mundial de conocimientos, especialmente en los países en desarrollo.

Actualmente, no existe un repositorio completo o una base de datos central de OSAT y Appropedia.org, quizás sea el mejor ejemplo. Sin embargo, la Nota de la UNCTAD dice: "Muchas organizaciones, organizaciones sin fines de lucro y empresas con fines de lucro están desarrollando OSAT y manteniendo bases de datos existentes a pequeña escala. Si bien hay muchos OSAT disponibles, se encuentran dispersos en varias bases de datos para tecnologías particulares. Mientras tanto, sigue existiendo una clara necesidad de aumentar la tasa de uso de OSAT.

Por lo tanto, existe una necesidad urgente de una base de datos de código abierto centralizada global (COSD) confiable. Al tener un alcance global, un repositorio de COSD proporcionaría una ventanilla única a la que todos pueden acceder para resolver los desafíos locales".

Concluye: "La ONU está bien posicionada para liderar el establecimiento de un COSD dado su papel bien establecido en la promoción de la tecnología de código abierto a través de varios foros y publicaciones intergubernamentales. En particular, 2030 Connect es una plataforma tecnológica en línea de la ONU que se desarrolló como parte del trabajo del Equipo de Trabajo Interinstitucional de la ONU. El COSD podría mejorarlo".

Con el liderazgo de la ONU, quizás no estemos demasiado lejos de cuándo, si tiene un problema local (sin importar en qué parte del mundo se encuentre), pueda descargar una solución de código abierto examinada y probada. Quizás, esta es la potencia de fuego que necesitamos para alcanzar los ambiciosos Objetivos de Desarrollo Sostenible.

10.6.2 La Comisión Europea se Compromete a Liberar Todo el Software que Pueda Beneficiar a la Sociedad

A finales del 2021, la Unión Europea (UE) y su órgano legislativo, la Comisión Europea siguen avanzando en su estrategia digital con el Software de código abierto como uno de los pilares fundamentales. En esta ocasión ha sido esta última la que anuncia novedades para con la distribución del Software desarrollado para cubrir necesidades internas de la organización.

De acuerdo a la información publicada, la Comisión Europea ha aprobado una nueva regulación que favorece el libre acceso al Software que producen siempre y cuando existan beneficios potenciales para «los ciudadanos, las empresas u otros servicios públicos», lo que de la teoría a la práctica bien puede abarcar todo lo que se desarrolle bajo su tejado.

Esta nueva disposición se apoya a su vez en un reciente estudio realizado también por la Comisión sobre el impacto del Software de código abierto en áreas como la independencia tecnológica, la competitividad y la innovación en la economía de la Unión Europea. El objetivo, hallar evidencias sólidas con las que conformar las políticas europeas de código abierto para los próximos años.

En términos económicos, de hecho, los cálculos son de lo más optimistas y apuntan un impacto económico contundente, de miles de millones de euros de ahorro al año -a modo de ejemplo, se estimó entre 65 y 95 mil millones de euros solo en 2018- y con un incremento mínimo en la apuesta, se podría dar un crecimiento del PIB de la UE de en torno a los 100,000 millones de euros.

Con semejante escenario, no es de extrañar que la misma Comisión Europea esté interesada en promover las soluciones de código abierto dentro y fuera de las instituciones y no solo se basan en el beneficio económico directo: son muchas otras las ventajas del modelo también recogidas en el informe, tal y como se ha mencionado: independencia, competitividad, innovación... y en el caso de las administraciones públicas, colaboración, reutilización y transparencia.

En palabras de Johannes Hahn, comisario de Presupuesto y Administración: «El código abierto ofrece grandes ventajas en un ámbito en el que la UE puede desempeñar un papel de liderazgo. Las nuevas normas aumentarán la transparencia y ayudarán a la Comisión, así como a los ciudadanos, las empresas y los servicios públicos de toda Europa, a beneficiarse del desarrollo de Software de código abierto. Poner en común los esfuerzos para mejorar el Software y la creación conjunta de nuevas funciones reduce los costes para la sociedad, ya que también nos beneficiamos de las mejoras realizadas por otros desarrolladores. Esto también puede mejorar la seguridad, ya que especialistas externos e independientes comprueban los fallos y las deficiencias de seguridad de los programas informáticos».

La comisaría de Innovación, Investigación, Cultura, Educación y Juventud, Mariya Gabriel, ha declarado: «La Comisión pretende, con su ejemplo, estar al frente de la transición digital en Europa. Con las nuevas normas, la

Comisión aportará un valor significativo a las empresas, también las emergentes, a los innovadores, a los ciudadanos y las administraciones públicas, poniendo a su disposición el código abierto de sus soluciones informáticas. Esta decisión también ayudará a estimular la innovación, gracias al código de la Comisión disponible públicamente».

Como muestra del Software desarrollado bajo el amparo de la Comisión Europea que va a ser liberado se incluyen proyectos como eSignature, «un conjunto de normas, herramientas y servicios gratuitos que ayudan a las administraciones públicas y a las empresas a acelerar la creación y verificación de firmas electrónicas jurídicamente válidas en todos los Estados miembros de la UE»; o LEOS (Legislation Editing Open Software), «el Software utilizado en toda la Comisión para elaborar textos jurídicos. LEOS, escrito originalmente para la Comisión, se está desarrollando en estrecha colaboración con Alemania, España y Grecia».

Esta nueva iniciativa de la Comisión Europea contempla asimismo la creación de un repositorio centralizado para facilitar el descubrimiento, el acceso y la reutilización del Software incluido, el cual se sumará a todos los proyectos realizados por las diferentes administraciones públicas comunitarias en base al mismo modelo de desarrollo. Y viene de lejos este impulso, aun cuando comienza a unificarse ahora.

Sin ir más lejos, hace años que la propia Comisión Europea puso en marcha el programa Interoperable Delivery of European eGovernment Services to Public Administrations, Businesses and Citizens que dio origen al observatorio JoinUp (<https://joinup.ec.europa.eu/>), en cuyas páginas se recogen casi 3,000 soluciones de Software abierto, 133 colecciones de recursos y cuantiosa información relacionada.

Más tarde, de 2014 a 2017 se inició la «primera fase» en la estrategia de código abierto de la Unión Europea, especialmente dentro de la propia Comisión, estableciendo determinados requisitos en materia de Software de código abierto; actualmente se está desarrollando la nueva «estrategia de código abierto 2020-2023», con la que la Comisión Europea pretende ampliar y afianzar los objetivos de la estrategia digital y la contribución al programa Europa Digital.

11 Bibliografía

Este texto es una recopilación de múltiples fuentes, mi aportación —si es que puedo llamarla así— es plasmarlo en este documento, en el que trato de dar coherencia a mi visión de los temas desarrollados.

En la realización de este texto se han revisado —en la mayoría de los casos indico la referencia, pero pude omitir varias de ellas, por lo cual pido una disculpa— múltiples páginas Web, artículos técnicos, libros, entre otros materiales bibliográficos, los más representativos y de libre acceso los pongo a su disposición en la siguiente liga:

Herramientas

<http://132.248.181.216/Herramientas/>

Referencias

- [1] Carreras en la Facultad de Ciencias, UNAM, <http://www.fciencias.unam.mx/licenciatura/resumen/>
- [2] Aulas y Talleres en el Tlahuizcalpan de las Carreras de Matemáticas de la Facultad de Ciencias, UNAM, <http://www.matematicas.unam.mx/tlahuiz/> 248
- [3] Proyectos de Software Sourceforge, <http://sourceforge.net/>
- [4] GNU Operating System, <http://www.gnu.org/> 465, 475
- [5] Google Code, <http://code.google.com>
- [6] <http://es.wikipedia.org/wiki/Linux>
- [7] The Linux Kernel Archives, <http://www.Kernel.org/>

- [8] Debian el Sistema Operativo Universal, <http://www.debian.org>
- [9] http://es.wikipedia.org/wiki/Microsoft_Windows
- [10] http://es.wikipedia.org/wiki/Mac_OS
- [11] <http://es.wikipedia.org/wiki/Android>
- [12] [https://es.wikipedia.org/wiki/Java_\(lenguaje_de_programaci%C3%B3n\)](https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)) 5, 79
- [13] <https://es.wikipedia.org/wiki/Python> 5, 88
- [14] [https://es.wikipedia.org/wiki/C_\(lenguaje_de_programaci%C3%B3n\)](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)) 5, 107
- [15] <https://es.wikipedia.org/wiki/C%2B%2B> 5, 111
- [16] <http://www.gnu.org/philosophy/free-sw.es.html> 465
- [17] http://es.wikipedia.org/wiki/Software_libre 465
- [18] <http://www.hispalinux.es/SoftwareLibre> 465
- [19] http://es.wikipedia.org/wiki/Software_propietario 463
- [20] Diferentes Tipos de Licencias para el Software, <http://www.gnu.org/licenses/license-list.html> 80, 223, 224, 465, 475
- [21] FSF, Free Software Foundation, <http://www.fsf.org/> 465, 475
- [22] GCC, the GNU Compiler Collection, <http://gcc.gnu.org/>
- [23] LaTeX, A Document Preparation System, <http://www.latex-project.org/> 119, 249
- [24] MATLAB, MathWorks, <http://www.mathworks.com/products/matlab/> 223, 225, 248
- [25] SCILAB, Scilab Open Source for Numerical Computation, <http://www.scilab.org/> 224
- [26] Julia, The Julia Programming Language, <https://julialang.org> 104, 224

- [27] FreeMat, FreeMat Open Source for rapid engineering and scientific prototyping and data processing, <http://freemat.sourceforge.net/> 224, 248
- [28] OCTAVE, GNU Octave, <https://www.gnu.org/software/octave/> 223, 225, 248
- [29] SCIPY Open Source Library of Scientific Tools, <http://www.scipy.org/> 224, 248
- [30] Python Programming Language, <http://www.python.org/> 224
- [31] SAS, Business Analytics and Business Intelligence Software, <http://www.sas.com/> 248
- [32] IBM SPSS Software, <http://www-01.ibm.com/Software/analytics/spss/> 248
- [33] GNU PSPP, <http://www.gnu.org/software/pspp/> 248
- [34] The R Project for Statistical Computing, <http://www.r-project.org/> 119, 225, 248, 249
- [35] LINPACK, <http://www.netlib.org/linpack/> 223
- [36] EISPACK, <http://www.netlib.org/eispack/> 223
- [37] Scicos Block Diagram Modeler/Simulator, <http://www.scicos.org/> 224
- [38] Simulink Simulation and Model-Based Design, <https://es.mathworks.com/products/simulink.html> 223, 224
- [39] MATLAB to Scilab conversion tips, http://help.scilab.org/docs/6.0.1/en_US/m2sci_doc.html 248
- [40] EViews, <http://www.eviews.com/home.html> 248
- [41] Gretl, <http://gretl.sourceforge.net/> 248
- [42] Statgraphics, <http://statgraphics.softonic.com/> 248
- [43] Statistica, <https://www.tibco.com/es/products/tibco-statistica> 248
- [44] Systat, <https://www.systatsoftware.com/> 248

- [45] Vensim, <http://vensim.com/vensim-Software/> 248
- [46] Maple, <http://www.maplesoft.com/> 225, 248
- [47] Mathematica, <http://www.wolfram.com/mathematica/> 225, 248
- [48] Maxima, <http://maxima.sourceforge.net> 225, 248
- [49] RWeKa, <https://cran.r-project.org/Web/packages/RWeKa/index.html> 120, 249
- [50] Tinn-R Edit code and run it in R, <http://nbcgib.uesc.br/lec/software/editores/tinn-r/en> 120, 249
- [51] RStudio Software, Education, and Services for the R community, <http://www.rstudio.com/> 120, 249
- [52] Suite de código abierto con más de 700 paquetes para el desarrollo de la ciencia de datos con Python, <https://www.anaconda.com/> 248
- [53] Suite de código abierto de Software matemático como: NumPy, SciPy, matplotlib, Sympy, Maxima, GAP, FLINT, R, entre otros, basada en Python o directamente vía interfaces o Wrappers, <http://www.sagemath.org/> 249
- [54] El economista, <http://eleconomista.com.mx/tecnociencia/2013/01/22/clausraran-negocios-mexico-uso-ilegal-Software>
- [55] PCworld, <http://www.pcworld.com.mx/UNAM-y-BSA-promueven-el-uso-de-software-legal/>
- [56] L. F. Pavarino, A. Toselli; *Recent Developments in Domain Decomposition Methods*. Springer, 2003. 409
- [57] M.B. Allen III, I. Herrera & G. F. Pinder; *Numerical Modeling in Science And Engineering*. John Wiley & Sons, Inc . 1988. 405, 406, 408, 409
- [58] R. L. Burden y J. D. Faires; *Análisis Numérico*. Math Learning, 7 ed. 2004. 405
- [59] S. Friedberg, A. Insel, and L. Spence; *Linear Algebra*, 4th Edition, Prentice Hall, Inc. 2003. 405

- [60] Y. Saad; *Iterative Methods for Sparse Linear Systems*. SIAM, 2 ed. 2000. 387, 393, 395, 406, 408, 409
- [61] Y. Skiba; *Métodos y Esquemas Numéricos, un Análisis Computacional*. UNAM, 2005. 405, 408, 409, 411
- [62] I. Herrera; *Un Análisis del Método de Gradiente Conjugado*. Comunicaciones Técnicas del Instituto de Geofísica, UNAM; Serie Investigación, No. 7, 1988. 414
- [63] G. Herrera; *Análisis de Alternativas al Método de Gradiente Conjugado para Matrices no Simétricas*. Tesis de Licenciatura, Facultad de Ciencias, UNAM, 1989. 414
- [64] M. Diaz, I. Herrera; *Desarrollo de Precondicionadores para los Procedimientos de Descomposición de Dominio*. Unidad Teórica C, Posgrado de Ciencias de la Tierra, 22 pags, 1997. 422
- [65] A. Toselli, O. Widlund; *Domain Decomposition Methods - Algorithms and Theory*. Springer, 2005. 422
- [66] C.T. Keller, *Iterative Methods for Linear and Nonlinear Equations*, Society for Industrial and Applied Mathematics, 1995. 409, 411
- [67] W. Gropp, E. Lusk, A. Skjelleem, *Using MPI, Portable Parallel Programming With the Message Passing Interface*. Scientific and Engineering Computation Series, 2ed, 1999. 339, 340, 346
- [68] I. Foster; *Designing and Building Parallel Programs*. Addison-Wesley Inc., Argonne National Laboratory, and the NSF, 2004. 339, 340, 346
- [69] Jorge L. Ortega-Arjona, *Patterns for Parallel Software Design*, Wiley series in Software Design Patterns, 2010. 339
- [70] S.J. Pennycook, S.D. Hammond, S.A. Jarvis and G.R. Mudalige, *Performance Analysis of a Hybrid MPI/CUDA Implementation of the NAS-LU Benchmark*. ACM SIGMETRICS Perform. Eval. Rev. 38 (4). ISSN 0163-5999, (2011). 347
- [71] XMPI, *A Run/Debug GUI for MPI*. 339
<http://www.lam-mpi.org/software/xmpi/>

- [72] VAMPIR, *Performance Optimization*. 347
<http://www.vampir.eu/>
- [73] QEMU, http://wiki.qemu.org/Main_Page
- [74] KVM, http://www.linux-kvm.org/page/Main_Page
- [75] Oracle VM VirtualBox, <https://www.virtualbox.org>
- [76] VMware, <https://www.vmware.com>
- [77] Virtual PC,
<https://www.microsoft.com/es-mx/download/details.aspx?id=3702>
- [78] Hyper-V,
[https://msdn.microsoft.com/es-es/library/mt16937\(v=ws.11\).aspx](https://msdn.microsoft.com/es-es/library/mt16937(v=ws.11).aspx)
- [79] Parallels, <https://www.parallels.com>
- [80] Máquinas Virtuales, http://es.wikipedia.org/wiki/Máquina_virtual
- [81] Algunos usos de máquinas Virtuales,
<http://www.configurarequipos.com/doc747.html>
- [82] <https://es.wikipedia.org/wiki/SQL>
- [83] <https://es.wikipedia.org/wiki/NoSQL>
- [84] Handbook of Floating-Point Arithmetic 2010th Edition, Jean-Michel Muller, Nicolas Brisebarre, Et alii, Birkhäuser, 2010.
- [85] What Every Computer Scientist Should Know About Floating-Point Arithmetic, David Goldberg, ACM Computing Surveys, Vol 23, No 1, March 1991
- [86] 754-2008 - IEEE Standard for Floating-Point Arithmetic. IEEE. 29 de agosto de 2008. ISBN 978-0-7381-5752-8. doi:10.1109/IEEESTD.2008.4610935. (NB. Superseded by IEEE Std 754-2019, a revision of IEEE 754-2008.)

- [87] Stochastic Rounding and Its Probabilistic Backward Error Analysis, Michael P. Connolly, Nicholas J. Higham , Methods and Algorithms for Scientific Computing, SIAM Journal on Scientific Computing Vol. 43, Iss. 1 (2021)
- [88] IEEE, <https://www.ieee.org/>
- [89] Intel, <https://www.intel.la>
- [90] AMD, <https://www.amd.com>
- [91] ARM, <https://www.arm.com/>
- [92] Cerebras, <https://www.cerebras.net/>



Declaro terminado este trabajo sufrido, ideado y llevado a cabo entre los años 2010 al 2025, aún y a pesar de impedimentos tales como: la mala suerte, la desventura, el infortunio, la incomprensión, la gripe, el Covid-19, las horas de frío y de calor, la tristeza, la desesperanza, el cansancio, el presente, el pasado y mi futuro, el que dirán, la vergüenza, mis propias incapacidades y limitaciones, mis aversiones, mis temores, mis dudas y en fin, todo aquello que pudiera ser tomado por mi, o por cualquiera, como obstáculo en este tiempo de mentiras, verdades, de incredulidad e ignorancia o negación de la existencia real y física de la mala fe.

Atentamente

Antonio Carrillo Ledesma

