Algoritmo para matrices tridiagonales

De Wikipedia, la enciclopedia libre

El **algoritmo para matrices tridiagonales** o **algoritmo de Thomas** (por Llewellyn Thomas) es un algoritmo del álgebra lineal numérica para resolver matrices tridiagonales de forma eficiente.

Una matriz tridiagonal se corresponde a un sistema de ecuaciones de la forma

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i,$$

donde $a_1 = 0$ y $c_n = 0$. lo que se puede representar matricialmente como

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}.$$

Para este tipo de sistemas se puede obtener con este algoritmo una solución con solo $O(n)$ operaciones en vez de las $O(n^3)$ que requiere la eliminación gaussiana. El algoritmo primero elimina las a_i y luego usa una sustitución para obtener la solución.

Este tipo de matrices suelen salir al plantear discretizaciones por métodos de diferencias finitas, volúmenes finitos o elementos finitos de problemas unidimensionales. Algunos de los problemas físicos que se plantean así son la Ecuación de Poisson, la ecuación de calor, la ecuación de onda o la interpolación por splines.

Índice

Método

El primer paso del método es modificar los coeficientes como sigue:

$$c'_i = \begin{cases} \dfrac{c_i}{b_i} & ; \quad i = 1 \\ \dfrac{c_i}{b_i - c'_{i-1} a_i} & ; \quad i = 2, 3, \ldots, n-1 \end{cases}$$

donde se marcan con superíndice ' los nuevos coeficientes.

De igual manera se opera:

$$d'_i = \begin{cases} \dfrac{d_i}{b_i} & ; \quad i = 1 \\ \dfrac{d_i - d'_{i-1} a_i}{b_i - c'_{i-1} a_i} & ; \quad i = 2, 3, \ldots, n. \end{cases}$$

a lo que se llama barrido hacia adelante. A continuación se obtiene la solución por sustitución hacia atrás:

$$x_n = d'_n$$

$$x_i = d'_i - c'_i x_{i+1} \qquad ; \, i = n-1, n-2, \ldots, 1.$$

Implementaciones

Implementación en C

La siguiente función en C resolverá el sistema (aunque sobreescribirá el vector de entrada c en el proceso). Se ha de notar que aquí los subíndices están basados en el cero, es decir $n = 0, 1, \ldots, N-1$ donde N es el número de ecuaciones:

```c
void solve_tridiagonal_in_place_destructive(float x[], const size_t N, const float a[], const float b[], float c[]) {
size_t n;

/*
solves Ax = v where A is a tridiagonal matrix consisting of vectors a, b, c
note that contents of input vector c will be modified, making this a one-time-use function
x[] - initially contains the input vector v, and returns the solution x. indexed from [0, ..., N - 1]
N - number of equations
a[] - subdiagonal (means it is the diagonal below the main diagonal) -- indexed from [1, ..., N - 1]
b[] - the main diagonal, indexed from [0, ..., N - 1]
c[] - superdiagonal (means it is the diagonal above the main diagonal) -- indexed from [0, ..., N - 2]
*/

c[0] = c[0] / b[0];
x[0] = x[0] / b[0];

/* loop from 1 to N - 1 inclusive */
for (n = 1; n < N; n++) {
float m = 1.0f / (b[n] - a[n] * c[n - 1]);
if(n < (N-1)) c[n] = c[n] * m;
x[n] = (x[n] - a[n] * x[n - 1]) * m;
}

/* loop from N - 2 to 0 inclusive */
for (n = N - 2; n-- > 0; ) {
x[n] = x[n] - c[n] * x[n + 1];
}
}
```

La siguiente variante preserva el sistema de ecuaciones para reutilizarlo en otras funciones. Se hacen llamadas a la biblioteca para reservar más especiao. Otras variantes usan un puntero a memoria disponible.

```c
void solve_tridiagonal_in_place_reusable(float x[], const size_t N, const float a[], const float b[], const float c[]) {
size_t n;

/* allocate scratch space */
float * const cprime = malloc(sizeof(float) * N);

cprime[0] = c[0] / b[0];
x[0] = x[0] / b[0];

/* loop from 1 to N - 1 inclusive */
for (n = 1; n < N; n++) {
float m = 1.0f / (b[n] - a[n] * cprime[n - 1]);
if(n < (N-1)) cprime[n] = c[n] * m;
x[n] = (x[n] - a[n] * x[n - 1]) * m;
}

/* loop from N - 2 to 0 inclusive */
for (n = N - 2; n-- > 0; )
x[n] = x[n] - cprime[n] * x[n + 1];

/* free scratch space */
```

```
free(cprime);
}
```

Implementación en Python

La siguiente implementación usa el lenguaje de programación Python.De nuevo, los subíndices son basados en el cero ($i = 0, 1, \ldots, n-1$ donde n es el número de incógnitas).

```
def TDMASolve(a, b, c, d):
n = len(d)#n in the numbers is rows

# Modify the first-row coefficients
c[0] /= b[0] #Division by zero risk.
d[0] /= b[0]

for i in xrange(1, nmax):
ptemp = b[i] - (a[i] * c[i-1])
c[i] /= ptemp
d[i] = (d[i] - a[i] * d[i-1])/ptemp

#Back Substitution
x = [0 for i in xrange(nmax)]
x[-1] = d[-1]
for i in range(-2,-nmax-1,-1):
x[i] = d[i] - c[i] * x[i+1]
    return x
```

Implementación en Matlab

En Matlab/Octave el algoritmo queda como sigue. Esta vez los vectores están basados en el uno, por lo que $i = 1, 2, \ldots, n$ con n que el número de incógnitas.

```
function x = TDMAsolver(a,b,c,d)
%a, b, c are the column vectors for the compressed tridiagonal matrix, d is the right vector
% N is the number of rows
N = length(d);

% Modify the first-row coefficients
c(1) = c(1) / b(1); % Division by zero risk.
d(1) = d(1) / b(1);

for n = 2:1:N
    temp = b(n) - a(n) * c(n - 1);
    if (n<N)
        c(n) = c(n) / temp;
    end
    d(n) = (d(n) - a(n) * d(n - 1)) / temp;
end

% Now back substitute.
x(N) = d(N);
for n = (N - 1):-1:1
    x(n) = d(n) - c(n) * x(n + 1);
end
end
```

Implementación en Fortran 90

Fortran usa también una nomenclatura basada en el uno, es decir $i = 1, 2, \ldots, n$ siendo n el número de incógnitas.

Algunas veces no es deseable que el programa sobreescriba los coeficientes (por ejemplo para resolver diversos sistemas que solo difieren en el término independiente), así que esta implementación mantiene dichos coeficientes.

```
      subroutine solve_tridiag(a,b,c,d,x,n)
      implicit none
!   a - sub-diagonal (means it is the diagonal below the main diagonal)
!   b - the main diagonal
!   c - sup-diagonal (means it is the diagonal above the main diagonal)
!   d - right part
!   x - the answer
!   n - number of equations
```

```
      integer,intent(in) :: n
      real(8),dimension(n),intent(in) :: a,b,c,d
      real(8),dimension(n),intent(out) :: x
      real(8),dimension(n) :: cp,dp
      real(8) :: m
      integer i

! initialize c-prime and d-prime
      cp(1) = c(1)/b(1)
      dp(1) = d(1)/b(1)
! solve for vectors c-prime and d-prime
       do i = 2,n
         m = b(i)-cp(i-1)*a(i)
         cp(i) = c(i)/m
         dp(i) = (d(i)-dp(i-1)*a(i))/m
       enddo
! initialize x
        x(n) = dp(n)
! solve for x from the vectors c-prime and d-prime
      do i = n-1, 1, -1
        x(i) = dp(i)-cp(i)*x(i+1)
      end do

   end subroutine solve_tridiag
```

Derivación

Algoritmo

Se puede obtener dicho algoritmo usando la eliminación gaussiana de forma genérica. Suponiendo como incógnitas $x_1, \ldots, x_n$ y con ecuaciones a resolver:

$$\begin{aligned} b_1 x_1 + c_1 x_2 &= d_1; & i &= 1 \\ a_i x_{i-1} + b_i x_i + c_i x_{i+1} &= d_i; & i &= 2, \ldots, n-1 \\ a_n x_{n-1} + b_n x_n &= d_n; & i &= n. \end{aligned}$$

Modificando la segunda ecuación a partir de la primera obtenemos:

$$(\text{ecuación 2}) \cdot b_1 - (\text{ecuación 1}) \cdot a_2$$

lo que da:

$$(a_2 x_1 + b_2 x_2 + c_2 x_3) b_1 - (b_1 x_1 + c_1 x_2) a_2 = d_2 b_1 - d_1 a_2$$

$$(b_2 b_1 - c_1 a_2) x_2 + c_2 b_1 x_3 = d_2 b_1 - d_1 a_2$$

y se ha eliminado x_1 de la segunda ecuación. Si repetimos usando la segunda ecuación en la tercera obtenemos:

$$(a_3 x_2 + b_3 x_3 + c_3 x_4)(b_2 b_1 - c_1 a_2) - ((b_2 b_1 - c_1 a_2) x_2 + c_2 b_1 x_3) a_3 = d_3 (b_2 b_1 - c_1 a_2) - (d_2 b_1 - d_1 a_2) a_3$$

$$(b_3 (b_2 b_1 - c_1 a_2) - c_2 b_1 a_3) x_3 + c_3 (b_2 b_1 - c_1 a_2) x_4 = d_3 (b_2 b_1 - c_1 a_2) - (d_2 b_1 - d_1 a_2) a_3.$$

Esta vez se ha eliminado x_2. El procedimiento se puede repetir hasta la fila enésima, dejando cada ecuación con solo dos incógnitas menos la última que solo tiene una incógnita. Entonces es inmediata la resolución de esta y desde ahí la de las anteriores.

Fórmulas para coeficientes

La determinación de los coeficientes en las ecuaciones generales es más complicada. Examinando el procedimiento, se pueden definir de forma recursiva:

$$\tilde{a}_i = 0$$

$$\begin{aligned} \tilde{b}_1 &= b_1 \\ \tilde{b}_i &= b_i \tilde{b}_{i-1} - \tilde{c}_{i-1} a_i \end{aligned}$$

$$\tilde{c}_1 = c_1$$
$$\tilde{c}_i = c_i \tilde{b}_{i-1}$$

$$\tilde{d}_1 = d_1$$
$$\tilde{d}_i = d_i \tilde{b}_{i-1} - \tilde{d}_{i-1} a_i.$$

Para acelerar la resolución $\tilde{b}_i$ puede ser dividido (si no hay problemas por división por cero) y los nuevos coeficientes, indicados con primas, serán:

$$a'_i = 0$$

$$b'_i = 1$$

$$c'_1 = \frac{c_1}{b_1}$$
$$c'_i = \frac{c_i}{b_i - c'_{i-1} a_i}$$

$$d'_1 = \frac{d_1}{b_1}$$
$$d'_i = \frac{d_i - d'_{i-1} a_i}{b_i - c'_{i-1} a_i}.$$

Lo que da el siguiente sistema con las mismas incógnitas y los coeficientes definidos en función de los originales:

$$x_i + c'_i x_{i+1} = d'_i \quad ; \quad i = 1, \ldots, n-1$$
$$x_n = d'_n \quad ; \quad i = n.$$

Donde la última ecuación solo afecta a una incógnita. Resolviéndola podemos resolver la anterior y así sucesivamente:

$$x_n = d'_n$$

$$x_i = d'_i - c'_i x_{i+1} \quad ; \, i = n-1, n-2, \ldots, 1.$$

Variantes

En algunas situaciones, particularmente con condiciones de contorno periódicas, se busca resolver una forma perturbada de la matriz tridiagonal:

$$\begin{aligned} a_1 x_n + b_1 x_1 + c_1 x_2 &= d_1, \\ a_i x_{i-1} + b_i x_i + c_i x_{i+1} &= d_i, \qquad i = 2, \ldots, n-1 \\ a_n x_{n-1} + b_n x_n + c_n x_1 &= d_n. \end{aligned}$$

Para este caso se usa la Fórmula Sherman-Morrison para evitar las operaciones adicionales de una eliminación gaussiana y poder seguir usando el algoritmo de Thomas. Este método requiere resolver una versión no cíclica del sistema para la entrada y un vector de corrección y combinar los resultados. Todo ello se puede hacer eficientemente de una vez dado que ambos problemas compartan el *barrido hacia delante* de la matriz triangular..

En otras situaciones el sistema de ecuaciones puede ser tridiagonal por bloques, con submatrices como elementos formando tres diagonales. Este tipo de problemas suelen darse cuando se quiere extender los métodos de discretización a dos dimensiones y puede verse también como una matriz pentadiagonal. Variantes de este algoritmo se usan para estas situaciones.

El libro *Numerical Mathematics* de Quarteroni, Sacco y Saleri, recoge una versión modificada para

multiplicaciones en vez de divisiones, lo que es útil en ciertas arquitecturas.

Referencias

- Conte, S.D., and deBoor, C. (1972). *Elementary Numerical Analysis*. McGraw-Hill, New York. ISBN 0070124469.
- Este artículo incluye texto del artículo Tridiagonal matrix algorithm - TDMA (Thomas algorithm) (http://www.cfd-online.com/Wiki/Tridiagonal_matrix_algorithm_-_TDMA_%28Thomas_algorithm%29) publicado con licencia GNU en CFD online wiki (http://www.cfd-online.com/Wiki/Main_Page)
- Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007). «Section 2.4» (http://apps.nrbook.com/empanel/index.html?pg=56). *Numerical Recipes: The Art of Scientific Computing* (3rd edición). New York: Cambridge University Press. ISBN 978-0-521-88068-8.

Enlaces externos

- Example with VBA code (http://web.archive.org/web/http://www.water.tkk.fi/wr/kurssit/Yhd-12.122/www_book/sgh_422b.htm)