Métodos iterativos de resolución de sistemas de ecuaciones lineales

Grupo de Dinámica de Flujos Ambientales
Universidad de Granada

Hidráulica Ambiental
master oficial

Índice

Lista de Tablas

Lista de Algoritmos

Iniciábamos el tema anterior con varios ejemplos de teoría de circuitos y difusión térmica. Volviendo a ese segundo ejemplo, la matriz resultante era

$$
\begin{array}{ccccccccc|c}
-4u_1 & +u_2 & & +u_4 & & & & & & = -50 \\
u_1 & -4u_2 & +u_3 & & +u_5 & & & & & = -50 \\
 & u_2 & -4u_3 & & & +u_6 & & & & = -150 \\
u_1 & & & -4u_4 & +u_5 & & +u_7 & & & = 0 \\
 & u_2 & & +u_4 & -4u_5 & +u_6 & & +u_8 & & = 0 \\
 & & u_3 & & +u_5 & -4u_6 & & & +u_9 & = -100 \\
 & & & u_4 & & & -4u_7 & +u_8 & & = -50 \\
 & & & & u_5 & & +u_7 & -4u_8 & & = -50 \\
 & & & & & u_6 & & +u_8 & -4u_9 & = -150
\end{array}
\qquad (0.1)
$$

A este tipo de matrices se las llama *dispersas* por el elevado número de entradas nulas que presenta. En estos casos, los métodos de resolución de sistemas lineales más eficientes suelen ser los llamados *métodos iterativos*, en contraposición con los métodos directos. Además, hay que decir que los métodos iterativos son mucho menos costosos computacionalmente que los directos, incluso para órdenes pequeños.
Comparativamente, el número de operaciones que tiene lugar en los métodos directos es muy elevado, dando lugar a errores por redondeo. Asimismo, necesitan mucha menos memoria de almacenamiento[1].

Las técnicas iterativas de resolución de sistemas lineales de orden $n \times n$ de la forma

$$Ax = b \qquad (0.2)$$

donde $x, b \in M_{n\times 1}(\mathbb{R})$, $A \in M_n(\mathbb{R})$ y $\det(A) \neq 0$, consisten en la construcción de una sucesión de vectores $\left\{x^{(0)},\ldots,x^{(k)},\ldots\right\}$ que converja a la solución del sistema. Esto es

$$\lim_{k\to\infty} x^{(k)} = x. \qquad (0.3)$$

Los métodos iterativos que vamos a considerar en este capítulo son aquellos que se construyen partiendo de un $x^{(0)}$ arbitrario y después generando los siguientes de la forma

$$x^{(k+1)} = F\left(x^{(k)}\right) \qquad (0.4)$$

siendo $F : \mathbb{R}^n \to \mathbb{R}^n$. En nuestro caso, consideraremos únicamente funciones F de tipo lineal, es decir $F(x) = Tx + c$, siendo $c \in \mathbb{R}^n$ y $T \in M_n(\mathbb{R})$. Por tanto, dado un vector inicial, la sucesión se genera calculando

$$x^{(k)} = Tx^{(k-1)} + c\,. \qquad (0.5)$$

[1] Los métodos directos de resolución de sistemas de ecuaciones lineales son, por lo general, computacionalmente costosos, especialmente cuando se incrementa el orden del sistema; sin mencionar los problemas por errores de redondeo. En estos casos (v.gr. en EDPs), métodos iterativos de resolución de sistemas de ecuaciones lineales suelen resultar más efectivos que los directos. Otra situación en la que aquéllos se muestran más eficientes que estos es cuando la matriz asociada al sistema presenta numerosas entradas nulas (véase ejemplos del Tema4).

Lo que viene a decir (0.5) es que x es un punto fijo del *mapa iterativo* $x = Tx + c$ cuya solución formal es $x = (I - T)^{-1} c$.

Ejemplo [Burden y Faires, 1985]

Comenzamos con un ejemplo. Sea el sistema lineal siguiente

$$\begin{aligned} 10x_1 \quad -x_2 \quad +2x_3 \quad &= 6 \\ -x_1 \quad +11x_2 \quad -x_3 \quad +3x_4 &= 25 \\ 2x_1 \quad -x_2 \quad 10x_3 \quad -x_4 &= -11 \\ \quad 3x_2 \quad -x_3 \quad +8x_4 &= 15\,. \end{aligned} \tag{0.6}$$

La idea es convertir este sistema de la forma (0.2) a la forma $x = Tx + c$. Para ello se despeja de cada ecuación i la componente x_i. En este ejemplo,

$$T = \begin{pmatrix} 0 & 1/10 & -1/5 & 0 \\ 1/11 & 0 & 1/11 & -3/11 \\ -1/5 & 1/10 & 0 & 1/10 \\ 0 & -3/8 & 1/8 & 0 \end{pmatrix} \tag{0.7}$$

y

$$c = \begin{pmatrix} 3/5 \\ 25/11 \\ -11/10 \\ 15/8 \end{pmatrix} \tag{0.8}$$

Suponiendo que el vector inicial es $x^{(0)} = (0,0,0,0)^T$ los siguientes se generan de la forma siguiente

$$\begin{pmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \\ x_4^{(1)} \end{pmatrix} = \begin{pmatrix} 0 & 1/10 & -1/5 & 0 \\ 1/11 & 0 & 1/11 & -3/11 \\ -1/5 & 1/10 & 0 & 1/10 \\ 0 & -3/8 & 1/8 & 0 \end{pmatrix} \begin{pmatrix} x_1^{(0)} \\ x_2^{(0)} \\ x_3^{(0)} \\ x_4^{(0)} \end{pmatrix} + \begin{pmatrix} 3/5 \\ 25/11 \\ -11/10 \\ 15/8 \end{pmatrix} \tag{0.9}$$

Mediante iteraciones sucesivas. Nótese que es importante elegir un vector inicial próximo a la solución. Eso hará que la convergencia sea más rápida. ¿Dónde detener el proceso iterativo? El criterio se establece en función del error relativo $e = \dfrac{\left\| x^{(k+1)} - x^{(k)} \right\|}{x^{(k+1)}}$ entre dos pasos iterativos, estos es, si, por ejemplo, $e \leq 10^{-3}$ se considera que el proceso ha convergido. En este caso la solución del sistema es $x^{(\infty)} = (1,2,-1,1)^T$ y se alcanza el criterio de convergencia en la iteración 10[2].

[2] Diremos que un método iterativo converge si converge para cualquier vector inicial.

Método iterativo de Jacobi

El método visto en el ejemplo anterior se denomina método de Jacobi. Como hemos visto, la idea es resolver cada ecuación para su respectiva componente, suponiendo que el coeficiente de la misma $a_{ii} \neq 0$. En tal caso,

$$x_i = \sum_{\substack{j=1 \\ j \neq i}}^{n} \left(\frac{a_{ij} x_j}{a_{ii}} \right) + \frac{b_i}{a_{ii}} \tag{0.10}$$

para $1 \leq i \leq n$. Los vectores en cada iteración se generan de la forma siguiente

$$x_i^{(k)} = \sum_{\substack{j=1 \\ j \neq i}}^{n} \left(\frac{-a_{ij} x_j^{(k-1)}}{a_{ii}} \right) + \frac{b_i}{a_{ii}} . \tag{0.11}$$

Para este método, la matriz A del sistema (0.2) suede dividirse en suma de matriz diagonal D, triangular superior U y triangular inferior L, esto es, $A = D - L - U$ [3]. De tal forma que de (0.2) se llega a

$$Dx = (L + U)x + b \tag{0.12}$$

o, puesto que hemos supuesto que $a_{ii} \neq 0$ para todo i, tenemos

$$x = D^{-1}(L + U)x + D^{-1}b . \tag{0.13}$$

Escrito de esta forma (ecuación (0.13)) es el método iterativo de Jacobi. No obstante, la expresión (0.13) suele dejarse para tratamientos teóricos, mientras que (0.11) es la que directamente se emplea en computación.

Algoritmo [Burden y Faires, 1985]: Método de Jacobi

Input $n, a_{ij}, b_i, X0_i, M, TOL$

Ouput $0, X0_i$

$k \leftarrow 1$

While $k \leq M$

$k \leftarrow 1$

For $i = 1$ **to** n

$$x_i \leftarrow \frac{\left(b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} X0_j \right)}{a_{ii}}$$

End

If $\|x - X0\| < TOL$ **Then**

Output k, x_i

Output *Solución determinada con éxito*

[3] El signo es arbitrario: sólo cambia el signo de los coeficientes de las matrices triangulares.

Exit
End

$X0 \leftarrow x$
End

Output *Número máximo de iteraciones superado*
Output *No se he encontrado una solución*
Exit

Los escalares M y TOL representan, respectivamente, el número máximo de iteraciones y la tolerancia admitida. Si está garantizada la convergencia a la solución real, se recomienda tomar como criterio que el error relativo sea menor o igual que TOL. En otro caso, añadir al criterio anterior un número máximo de iteraciones M.

Una pequeña modificación que mejora al algoritmo es determinar todas las divisiones antes de entrar en el proceso iterativo.Una forma de interpretar esto es que el sistema original (0.2) sea sustituido por $D^{-1}Ax = D^{-1}b$ donde $D = (a_{ii})$ es matriz diagonal.

```
% JACOBI ITERATIVE ALGORITHM 7.1
%
% To solve Ax = b given an initial approximation x(0).
%
% INPUT:   the number of equations and unknowns n; the entries
%          A(I,J), 1<=I, J<=n, of the matrix A; the entries
%          B(I), 1<=I<=n, of the inhomogeneous term b; the
%          entries XO(I), 1<=I<=n, of x(0); tolerance TOL;
%          maximum number of iterations N.
%
%  OUTPUT: the approximate solution X(1),...,X(n) or a message
%          that the number of iterations was exceeded.
 syms('AA', 'OK', 'NAME', 'INP', 'N', 'I', 'J', 'A', 'X1');
 syms('TOL', 'NN', 'K', 'ERR', 'S', 'X2', 'FLAG', 'OUP');
 TRUE = 1;
 FALSE = 0;
 fprintf(1,'This is the Jacobi Method for Linear Systems.\n');
 fprintf(1,'The array will be input from a text file in the order\n');
 fprintf(1,'A(1,1), A(1,2), ..., A(1,n+1), \n');
 fprintf(1,'A(2,1), A(2,2), ..., A(2,n+1), \n');
 fprintf(1,'..., A(n,1), A(n,2), ..., A(n,n+1)\n');
 fprintf(1,'Place as many entries as desired on each line, but
separate\n');
 fprintf(1,'entries with ');
 fprintf(1,'at least one blank.\n\n\n');
 fprintf(1,'The initial approximation should follow in same
format.\n');
 fprintf(1,'Has the input file been created? - enter Y or N.\n');
 AA = input(' ','s');
 OK = FALSE;
 if AA == 'Y' | AA == 'y'
 fprintf(1,'Input the file name in the form - drive:\\name.ext\n');
 fprintf(1,'for example:   A:\\DATA.DTA\n');
 NAME = input(' ','s');
 INP = fopen(NAME,'rt');
 OK = FALSE;
 while OK == FALSE
```

```
 fprintf(1,'Input the number of equations - an integer.\n');
 N = input(' ');
 if N > 0
 A = zeros(N,N+1);
 X1 = zeros(N);
 X2 = zeros(N);
 for I = 1 : N
 for J = 1 : N+1
 A(I,J) = fscanf(INP, '%f',1);
 end;
 end;
 for I = 1 : N
 X1(I) = fscanf(INP, '%f',1);
 end;
 OK = TRUE;
 fclose(INP);
 else
 fprintf(1,'The number must be a positive integer\n');
 end;
 end;
 OK = FALSE;
 while OK == FALSE
 fprintf(1,'Input the tolerance.\n');
 TOL = input(' ');
 if TOL > 0
 OK = TRUE;
 else
 fprintf(1,'Tolerance must be a positive.\n');
 end;
 end;
 OK = FALSE;
 while OK == FALSE
 fprintf(1,'Input maximum number of iterations.\n');
 NN = input(' ');
 if NN > 0
 OK = TRUE;
 else
 fprintf(1,'Number must be a positive integer.\n');
 end;
 end;
 else
 fprintf(1,'The program will end so the input file can be
created.\n');
 end;
 if OK == TRUE
% STEP 1
 K = 1;
 OK = FALSE;
% STEP 2
 while OK == FALSE & K <= NN
% err is used to test accuracy - it measures the infinity-norm
 ERR = 0;
% STEP 3
 for I = 1 : N
 S = 0;
 for J = 1 : N
 S = S-A(I,J)*X1(J);
 end;
 S = (S+A(I,N+1))/A(I,I);
 if abs(S) > ERR
 ERR = abs(S);
```

```
 end;
% use X2 for X
 X2(I) = X1(I)+S;
 end;
% STEP 4
 if ERR <= TOL
% process is complete
 OK = TRUE;
 else
% STEP 5
 K = K+1;
% STEP 6
 for I = 1 : N
 X1(I) = X2(I);
 end;
 end;
 end;
 if OK == FALSE
 fprintf(1,'Maximum Number of Iterations Exceeded.\n');
% STEP 7
% procedure completed unsuccessfully
 else
 fprintf(1,'Choice of output method:\n');
 fprintf(1,'1. Output to screen\n');
 fprintf(1,'2. Output to text file\n');
 fprintf(1,'Please enter 1 or 2.\n');
 FLAG = input(' ');
 if FLAG == 2
 fprintf(1,'Input the file name in the form - drive:\\name.ext\n');
 fprintf(1,'for example:   A:\\OUTPUT.DTA\n');
 NAME = input(' ','s');
 OUP = fopen(NAME,'wt');
 else
 OUP = 1;
 end;
 fprintf(OUP, 'JACOBI ITERATIVE METHOD FOR LINEAR SYSTEMS\n\n');
 fprintf(OUP, 'The solution vector is :\n');
 for I = 1 : N
 fprintf(OUP, ' %11.8f', X2(I));
 end;
 fprintf(OUP, '\nusing %d iterations\n', K);
 fprintf(OUP, 'with Tolerance  %.10e in infinity-norm\n', TOL);
 if OUP ~= 1
 fclose(OUP);
 fprintf(1,'Output file %s created successfully \n',NAME);
 end;
 end;
 end;
```

Algoritmo 1. Método de Jacobi.

La pregunta es cuándo converge el método de Jacobi. Para ello, presentamos un par de teoremas que darán las condiciones necesarias y suficientes de convergecia (además de asegurar que la matriz A es no singular).

Teorema

La condición necesaria y suficiente para que la sucesión $\left\{x^{(k)}\right\}_{k=0}^{\infty}$ asociada al mapa $x^{(k)} = Tx^{(k-1)} + c$ converja a $(I-T)^{-1}c \in \mathbb{R}^n$ es que el radio espectral de T sea menor que 1.

En otras palabras, y como corolario, la fórmula de iteración (0.5) da lugar a una sucesión convergente a la solución de (0.2), *para cualquier* vector inicial $x^{(0)} \in \mathbb{R}^n$.

Como corolario al teorema anterior cabe mostrar el resultado siguiente que servirá para dar condiciones suficientes un tanto más prácticas. Si $\|T\| < 1$ para cualquier norma matricial $\|\cdot\|$, la sucesión $\left\{x^{(k)}\right\}_{k=0}^{\infty}$ converge a la solución del sistema para todo $x^{(0)} \in \mathbb{R}^n$ y los errores de acotación están dados por

$$\left\|x - x^{(k)}\right\| \leq \|T\|^k \left\|x - x^{(0)}\right\|. \tag{0.14}$$

Nótese que la expresión (0.14) da una idea de lo rápido que puede converger un método, relacionándolo con el radio espectral. Se puede demostrar que

$$\left\|x - x^{(k)}\right\| \approx \rho(T)^k \left\|x - x^{(0)}\right\|, \tag{0.15}$$

por tanto, suponiendo que el error relativo $e \approx \rho(T)^k$ después de k iteraciones es $10^{-\eta}$, tendremos que $\rho(T)^k \leq 10^{-\eta}$. Esto es,

$$k \geq \frac{-\eta}{-\log \rho(T)}. \tag{0.16}$$

Según (0.16) es deseable escoger un método con un radio espectral mínimo, de tal forma que el número de iteraciones hasta alcanzar el error relativo prescrito sea mínimo.

Teorema

Si la matriz A del sistema (0.2) es diagonalmente dominante, la sucesión proporcionada por el método de Jacobi converge a la solución correcta para cualquier vector inicial $x^{(0)} \in \mathbb{R}^n$.

Este segundo teorema, a efectos computacionales, es más práctico que el primero. Comprobar que una matriz es diagonalmente dominante es mucho más sencillo que determinar el radio espectral.

Las demostraciones pueden encontrarse en [Burden y Faires, 1985; Kincaid y Cheney, 1990].

Método de Gauss – Seidel

Otra posible mejora del algoritmo es la siguiente. Admitiendo que la solución $x_i^{(k)}$ es una mejor aproximación que $x_i^{(k-1)}$ a la solución real, uno podría plantearse emplear soluciones de la ecuación i en la iteración k para determinar las soluciones $x_j^{(k)}$ con $j > i$. Esto es, en vez de emplear en la iteración k todas las soluciones de la iteración anterior $x_i^{(k-1)}$ $1 \leq i \leq n$, se emplearían los valores recientemente calculados en la iteración k (los $x_i^{(k)}$). En tal caso, la fórmula de recurrencia ya no sería (0.11) si no la siguiente

$$x_i^{(k)} = \frac{-\sum_{j=1}^{i-1}\left(a_{ij}x_j^{(k)}\right) - \sum_{j=i+1}^{n}\left(a_{ij}x_j^{(k-1)}\right) - b_i}{a_{ii}}. \qquad (0.17)$$

La expresión es análoga a la (0.11) excepto en los términos $\sum_{j=1}^{i-1}\left(a_{ij}x_j^{(k)}\right)$ que son los determinados en la presente iteración k y que sustituyen a los correspondientes a la iteración anterior $k-1$.

La fórmula de recurrencia (0.17) es la del método de Gauss – Seidel .

Para resolver $Ax = b$ dada una estimación inicial $x_i^{(0)}$ se debe introducir el número de ecuaciones y de incógnitas (n), esto es, el orden del sistema; las entradas de la matriz A, a_{ij} para $1 \leq i, j \leq n$; las entradas del término inhomogéneo b_i ; cada componente de $x_i^{(0)}$, $X0_i$; una tolerancia TOL y el máximo número de iteraciones M .

Algoritmo [Burden y Faires, 1985]: Método de Gauss – Seidel.

Input $n, a_{ij}, b_i, X0_i, M, TOL$

Ouput $0, X0_i$

$k \leftarrow 1$

While $k \leq M$

$k \leftarrow 1$

For $i = 1$ **to** n

$$x_i \leftarrow \frac{\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j - \sum_{j=i+1}^{n} a_{ij}X0_j\right)}{a_{ii}}$$

End

If $\|x - X0\| < TOL$ **Then**

Output k, x_i

Output *Solución determinada con éxito*

Exit

End

$X0 \leftarrow x$

End

Output *Número máximo de iteraciones superado*
Output *No se he encontrado una solución*
Exit

Los escalares M y TOL representan, respectivamente, el número máximo de iteraciones y la tolerancia admitida. Si está garantizada la convergencia a la solución real, se recomienda tomar como criterio que el error relativo sea menor o igual que TOL. En otro caso, añadir al criterio anterior un número máximo de iteraciones M.

Una pequeña modificación que mejora al algoritmo es determinar todas las divisiones antes de entrar en el proceso iterativo.Una forma de interpretar esto es que el sistema original (0.2) sea sustituido por $D^{-1}Ax = D^{-1}b$ donde $D = (a_{ii})$ es matriz diagonal.

Puesto que las variables con este método se calculan de forma secuencial, el método de Jacobi es más eficiente cuando se emplean códigos o computación en paralelo, puesto que la actualización de variables es simultánea.

```
% GAUSS-SEIDEL ITERATIVE TECHNIQUE ALGORITHM 7.2
%
% To solve Ax = b given an initial approximation x(0).
%
% INPUT:   the number of equations and unknowns n; the entries
%          A(I,J), 1<=I, J<=n, of the matrix A; the entries
%          B(I), 1<=I<=n, of the inhomogeneous term b; the
%          entries XO(I), 1<=I<=n, of x(0); tolerance TOL;
%          maximum number of iterations N.
%
%  OUTPUT: the approximate solution X(1),...,X(n) or a message
%          that the number of iterations was exceeded.
 syms('AA', 'OK', 'NAME', 'INP', 'N', 'I', 'J', 'A', 'X1');
 syms('TOL', 'NN', 'K', 'ERR', 'S', 'FLAG', 'OUP');
 TRUE = 1;
 FALSE = 0;
 fprintf(1,'This is the Gauss-Seidel Method for Linear Systems.\n');
 fprintf(1,'The array will be input from a text file in the order\n');
 fprintf(1,'A(1,1), A(1,2), ..., A(1,n+1), \n');
 fprintf(1,'A(2,1), A(2,2), ..., A(2,n+1), \n');
 fprintf(1,'..., A(n,1), A(n,2), ..., A(n,n+1)\n');
 fprintf(1,'Place as many entries as desired on each line, but
separate\n');
 fprintf(1,'entries with ');
 fprintf(1,'at least one blank.\n\n\n');
 fprintf(1,'The initial approximation should follow in same
format.\n');
 fprintf(1,'Has the input file been created? - enter Y or N.\n');
 AA = input(' ','s');
 OK = FALSE;
 if AA == 'Y' | AA == 'y'
 fprintf(1,'Input the file name in the form - drive:\\name.ext\n');
 fprintf(1,'for example:   A:\\DATA.DTA\n');
 NAME = input(' ','s');
 INP = fopen(NAME,'rt');
 OK = FALSE;
 while OK == FALSE
```

```
 fprintf(1,'Input the number of equations - an integer.\n');
 N = input(' ');
 if N > 0
 A = zeros(N,N+1);
 X1 = zeros(1,N);
 for I = 1 : N
 for J = 1 : N+1
 A(I,J) = fscanf(INP, '%f',1);
 end;
 end;
% Use X1 for X0
 for I = 1 : N
 X1(I) = fscanf(INP, '%f',1);
 end;
 OK = TRUE;
 fclose(INP);
 else
 fprintf(1,'The number must be a positive integer\n');
 end;
 end;
 OK = FALSE;
 while OK == FALSE
 fprintf(1,'Input the tolerance.\n');
 TOL = input(' ');
 if TOL > 0
 OK = TRUE;
 else
 fprintf(1,'Tolerance must be a positive.\n');
 end;
 end;
 OK = FALSE;
 while OK == FALSE
 fprintf(1,'Input maximum number of iterations.\n');
 NN = input(' ');
 if NN > 0
 OK = TRUE;
 else
 fprintf(1,'Number must be a positive integer.\n');
 end;
 end;
 else
 fprintf(1,'The program will end so the input file can be
created.\n');
 end;
 if OK == TRUE
% STEP 1
 K = 1;
 OK = FALSE;
% STEP 2
 while OK == FALSE & K <= NN
% ERR is used to test accuracy - it measures the infinity-norm
 ERR = 0;
% STEP 3
 for I = 1 : N
 S = 0;
 for J = 1 : N
 S = S-A(I,J)*X1(J);
 end;
 S = (S+A(I,N+1))/A(I,I);
 if abs(S) > ERR
 ERR  = abs(S);
```

```
 end;
 X1(I) = X1(I) + S;
 end;
% STEP 4
 if ERR <= TOL
 OK = TRUE;
% process is complete
 else
% STEP 5
 K = K+1;
% STEP 6 - is not used since only one vector is required
 end;
 end;
 if OK == FALSE
 fprintf(1,'Maximum Number of Iterations Exceeded.\n');
% STEP 7
% procedure completed unsuccessfully
 else
 fprintf(1,'Choice of output method:\n');
 fprintf(1,'1. Output to screen\n');
 fprintf(1,'2. Output to text file\n');
 fprintf(1,'Please enter 1 or 2.\n');
 FLAG = input(' ');
 if FLAG == 2
 fprintf(1,'Input the file name in the form - drive:\\name.ext\n');
 fprintf(1,'for example:   A:\\OUTPUT.DTA\n');
 NAME = input(' ','s');
 OUP = fopen(NAME,'wt');
 else
 OUP = 1;
 end;
 fprintf(OUP, 'GAUSS-SEIDEL METHOD FOR LINEAR SYSTEMS\n\n');
 fprintf(OUP, 'The solution vector is :\n');
 for I = 1 : N
 fprintf(OUP, ' %11.8f', X1(I));
 end;
 fprintf(OUP, '\nusing %d iterations\n', K);
 fprintf(OUP, 'with Tolerance  %.10e in infinity-norm\n', TOL);
 if OUP ~= 1
 fclose(OUP);
 fprintf(1,'Output file %s created successfully \n',NAME);
 end;
 end;
 end;
```

Algoritmo 2. Método de Gauss – Seidel.

Un teorema similar al anterior para el método de Jacobi garantiza que si la matriz asociada al sistema lineal de ecuaciones es diagonalmente dominante, el método de Gauss – Seidel converge para cualquier selección del vector inicial.

En otros casos la convergencia podría no estar garantizada. Incluso puede haber casos en los que el método de Jacobi converja, pero no el Gauss – Seidel, y viceversa.

Ejercicio

Demuéstrese que, aplicando el método iterativo de Gauss – Seidel al sistema siguiente

$$\begin{pmatrix} 2 & -1 & 0 \\ 1 & 6 & -2 \\ 4 & -3 & 8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ -4 \\ 5 \end{pmatrix} \tag{0.18}$$

Se obtiene la solución $(x_1, x_2, x_3) = (0.62, -0.76, 0.03)^T$ en la iteración $k = 13$ con un error relativo de $e \leq 10^{-6}$.

Teorema de Stein – Rossenberg

En general no se sabe cuál de las dos técnicas (Jacobi o Gauss – Seidel) es mejor; pero es casos especiales es conocida. Uno de ellos lo recogemos en el siguiente teorema

Si $a_{ij} \leq 0$ para cada $i \neq j$ y $a_{ii} > 0$ para $1 \leq i \leq n$ entonces se cumple uno y sólo uno de los siguientes puntos

- $0 < \rho(T_{GS}) < \rho(T_J) < 1$ ó
- $1 < \rho(T_J) < \rho(T_{GS})$ ó
- $0 = \rho(T_{GS}) = \rho(T_J)$ ó
- $\rho(T_{GS}) = \rho(T_J) = 1$.

Es claro a partir de lo anterior (primer punto), que cuando un método converge, el otro también converge, siendo el método de Gauss – Seidel más rápido que el de Jacobi. Pero ojo, bajo otras condiciones, puede que esto no sea así.

Métodos de relajación

Hay que recordar que la rapidez en la convergencia depende del radio de convergencia que, a su vez, depende del método empleado. Por tanto sería conveniente buscar el método con el menor radio de convergencia que resuelva un sistema dado para acelerar la convergencia.
El procedimiento emplea el concepto de vector residual, ya introducido en el tema anterior (o en los apéndices de matrices: véase condicionamiento de matrices).

El residual en los procedimientos iterativos está asociado a cada iteración, esto es

$$r_i^{(k)} = \left(r_{1i}^{(k)}, r_{2i}^{(k)}, \ldots, r_{ni}^{(k)} \right)^T. \tag{0.19}$$

Evidentemente, se espera de cualquier método que el residual converja a cero con $k \to \infty$. Para el método de Gauss – Seidel, la componente m -ésima del residual verifica

$$r_{mi}^{(k)} = b_m - \sum_{j=1}^{i-1} \left(a_{mj} x_j^{(k)} \right) - \sum_{j=i}^{n} \left(a_{mj} x_j^{(k-1)} \right) \tag{0.20}$$

para cada m. O escrito de otro modo, particularizando para la componente i [Burden y Faires, 1985]

$$a_{ii}x_i^{(k-1)} + r_{ii}^{(k)} = b_i - \sum_{j=1}^{i-1}\left(a_{ij}x_j^{(k)}\right) - \sum_{j=i+1}^{n}\left(a_{ij}x_j^{(k-1)}\right) \tag{0.21}$$

siendo

$$a_{ii}x_i^{(k-1)} + r_{ii}^{(k)} = a_{ii}x_i^{(k)} \tag{0.22}$$

o bien

$$x_i^{(k)} = x_i^{(k-1)} + \frac{r_{ii}^{(k)}}{a_{ii}}. \tag{0.23}$$

En contra de lo que cabría pensar en un principio con respecto a la expresión (0.23), reducir el residual $r_{ii}^{(k)}$ a cero, podría no acelerar el proceso de convergencia. Nótese que el residual está referido a la diferencia entre dos iteraciones consecutivas, pero no respecto de la solución. De hecho, residuales pequeños podrían ralentizar la convergencia. Por el contrario modificando el método de Gauss – Seidel a partir de (0.23) según

$$x_i^{(k)} = x_i^{(k-1)} + \omega\frac{r_{ii}^{(k)}}{a_{ii}} \tag{0.24}$$

con ciertos $0 < \omega$ conducirá a una mejora sustancial de la velocidad de convergencia. A estos métodos se los denomina ***métodos de relajación***. A los métodos de relajación con $0 < \omega < 1$ se los denomina de sub-relajación, mientras que a los que tienen $1 < \omega$ se los llama de sobre-relajación, los cuales pueden emplearse para acelerar la convergencia de sistemas convergentes por el método de Gauss – Seidel. Nos referiremos a estos últimos como *SRS* (Sobre-relajación sucesiva). Nótese que el método de Gauss – Seidel es también un método de relajación con $\omega = 1$.

En cuanto a la convergencia de los métodos de relajación, presentamos en siguiente teorema.

Teorema

Si A es simétrica y definida positiva, entonces el método de relajación converge si y sólo si $0 < \omega < 2$.

En particular vimos antes que para el método de Gauss – Seidel había convergencia para $\omega = 1$.

En los métodos de relajación se pasa del vector $x_i^{(k-1)}$ al $x_i^{(k)}$ en n etapas, a diferencia del método de Jacobi en el que no hay etapas intermedias. Asimismo en cada una de las etapas se emplea un vector distinto (véase (0.17) y Algoritmo 2) que se diferencia del anterior en una componente.

A partir de (0.21) y (0.24), y separando las sumas en matrices diagonal y triangulares superior e inferior, se obtiene (véase [Burden y Faires, 1985] para detalles)

$$x^{(k)} = (D - \omega L)^{-1}\left[(1-\omega)D + \omega U\right]x^{(k-1)} + \omega(D - \omega L)^{-1}b. \tag{0.25}$$

Ejemplo

Un ejemplo ilustrador propuesto en [Burden y Faires, 1985] , y que reproducimos aquí, muestra la eficiencia del método.
Se trata del sistema siguiente

$$\begin{pmatrix} 4 & 3 & 0 \\ 3 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 24 \\ 30 \\ -24 \end{pmatrix} \tag{0.26}$$

cuya solución es $(x_1, x_2, x_3)^T = (3,4,-5)^T$. Se inicia con el vector $x^{(0)} = (1,1,1)^T$ y se resuelve con el método de Gauss – Seidel y con el método SRS con $\omega = 1.25$. Se puede demostrar (hágase como ejercicio) que en la iteración $k = 7$ se obtienen los resultados mostrados en la Tabla 1.

	x_1	x_2	x_3
Gauss – Seidel	3.0134110	3.9888241	-5.0027940
SRS	3.0000490	4.0002586	-5.0003486

Tabla 1. Comparativa métodos de Gauss – Seidel y SRS.

Como puede observarse un $\omega = 1.25 > 1$ acelera sustancialmente la convergencia. De hecho, para llegar errores relativos de 10^{-7} se requieren 34 iteraciones para Gauss – Seidel y sólo 14 para SRS.

La pregunta obvia es cómo determinar el valor de ω óptimo. Para ello presentamos el siguiente teorema para matrices tridiagonales, como la del ejemplo (0.26).

Teorema

Si A es definida positiva y tridiagonal entonces

1. $0 < \rho(T_{GS}) = \rho(T_J)^2 < 1$
2. El ω óptimo para el método SRS es $\omega = 2/\left[1 + \sqrt{1 - \rho(T_J)^2}\right]$.

Y con esta elección de ω se verifica $\rho(T_{GS}) = \omega - 1$.

Así pues, para el ejemplo (0.26), que es una matriz tridiagonal, $T_J = D^{-1}(L + U)$ (véase (0.13)). Luego la matriz diagonal es

$$D = \begin{pmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \end{pmatrix}, \tag{0.27}$$

por tanto

$$D^{-1} = \begin{pmatrix} -1/4 & 0 & 0 \\ 0 & -1/4 & 0 \\ 0 & 0 & -1/4 \end{pmatrix}. \tag{0.28}$$

Así

$$D^{-1}(L+U)=\begin{pmatrix}-1/4 & 0 & 0\\ 0 & -1/4 & 0\\ 0 & 0 & -1/4\end{pmatrix}\begin{pmatrix}0 & -3 & 0\\ -3 & 0 & 1\\ 0 & 1 & 0\end{pmatrix} \tag{0.29}$$

y desarrollando el producto de ambas matrices

$$D^{-1}(L+U)=\begin{pmatrix}0 & -0.75 & 0\\ -0.75 & 0 & 0.25\\ 0 & 0.25 & 0\end{pmatrix}. \tag{0.30}$$

A continuación determinamos el radio espectral de la matriz $D^{-1}(L+U)$. Para ello, determinamos el polinomio característico

$$\det(T_J-\lambda I_3)=\begin{pmatrix}-\lambda & -0.75 & 0\\ -0.75 & -\lambda & 0.25\\ 0 & 0.25 & -\lambda\end{pmatrix}=-\lambda\left(\lambda^2-0.625\right). \tag{0.31}$$

De aquí se sigue que

$$\rho(T_J)=\sqrt{0.625} \tag{0.32}$$

y, de acuerdo con el teorema anterior,

$$\omega=2/\left[1+\sqrt{1-0.625}\right]\approx 1.24 \tag{0.33}$$

que no está muy alejado del que se ha empleado en el ejemplo (0.26).

El algoritmo para SRS es el siguiente

Algoritmo [Burden y Faires, 1985]: Método SRS.

Input $n, a_{ij}, b_i, X0_i, \omega, M, TOL$

Ouput $0, X0_i$

$k \leftarrow 1$

While $k \le M$

$k \leftarrow 1$

For $i=1$ **to** n

$$x_i \leftarrow (1-\omega)\,X0_i+\frac{\omega\left(b_i-\sum_{j=1}^{i-1}a_{ij}x_j-\sum_{j=i+1}^{n}a_{ij}X0_j\right)}{a_{ii}}$$

End

If $\|x-X0\|<TOL$ **Then**

Output k, x_i

Output *Solución determinada con éxito*

Exit

End

$X0 \leftarrow x$
End

Output *Número máximo de iteraciones superado*
Output *No se he encontrado una solución*
Exit

Para resolver $Ax = b$ dado el parámetro ω y una estimación inicial $x_i^{(0)}$ se debe introducir el número de ecuaciones y de incógnitas (n), esto es, el orden del sistema; las entradas de la matriz A, a_{ij} para $1 \leq i, j \leq n$; las entradas del término inhomogéneo b_i; cada componente de $x_i^{(0)}$, $X0_i$; una tolerancia TOL y el máximo número de iteraciones M.

```
% SOR ALGORITHM 7.3
%
% To solve Ax = b given the parameter w and an initial approximation
% x(0):
%
% INPUT:   the number of equations and unknowns n; the entries
%          A(I,J), 1<=I, J<=n, of the matrix A; the entries
%          B(I), 1<=I<=n, of the inhomogeneous term b; the
%          entries XO(I), 1<=I<=n, of x(0); tolerance TOL;
%          maximum number of iterations N; parameter w (omega).
%
% OUTPUT:  the approximate solution X(1),...,X(n) or a message
%          that the number of iterations was exceeded.
 syms('OK', 'AA', 'NAME', 'INP', 'N', 'I', 'J', 'A', 'X1');
 syms('TOL', 'NN', 'W', 'K', 'ERR', 'S', 'FLAG', 'OUP');
 TRUE = 1;
 FALSE = 0;
 fprintf(1,'This is the SOR Method for Linear Systems.\n');
 OK = FALSE;
 fprintf(1,'The array will be input from a text file in the
order:\n');
 fprintf(1,'A(1,1), A(1,2), ..., A(1,n+1) \n');
 fprintf(1,'A(2,1), A(2,2), ..., A(2,n+1),\n');
 fprintf(1,'..., A(n,1), A(n,2), ..., A(n,n+1)\n\n');
 fprintf(1,'Place as many entries as desired on each line, but \n');
 fprintf(1,'separate entries with at least one blank.\n');
 fprintf(1,'The initial approximation should follow in same
format.\n\n\n');
 fprintf(1,'Has the input file been created? - enter Y or N.\n');
 AA = input(' ','s');
 if AA == 'Y' | AA == 'y'
 fprintf(1,'Input the file name in the form - drive:\\name.ext\n');
 fprintf(1,'for example:   A:\\DATA.DTA\n');
 NAME = input(' ','s');
 INP = fopen(NAME,'rt');
 OK = FALSE;
 while OK == FALSE
 fprintf(1,'Input the number of equations - an integer.\n');
 N = input(' ');
 if N > 0
 A = zeros(N,N+1);
 X1 = zeros(1,N);
 for I = 1 : N
```

```
 for J = 1 : N+1
 A(I,J) = fscanf(INP, '%f',1);
 end;
 end;
 for I = 1 : N
 X1(I) = fscanf(INP, '%f',1);
 end;
% use X1 for X0
 OK = TRUE;
 fclose(INP);
 else
 fprintf(1,'The number must be a positive integer.\n');
 end;
 end;
 OK = FALSE;
 while OK == FALSE
 fprintf(1,'Input the tolerance.\n');
 TOL = input(' ');
 if TOL > 0
 OK = TRUE;
 else
 fprintf(1,'Tolerance must be a positive number.\n');
 end;
 end;
 OK = FALSE;
 while OK == FALSE
 fprintf(1,'Input maximum number of iterations.\n');
 NN = input(' ');
 if NN > 0
 OK = TRUE;
 else
 fprintf(1,'Number must be a positive integer.\n');
 end;
 end;
 fprintf(1,'Input parameter w (omega)\n');
 W = input(' ');
% use W for omega
 else
 fprintf(1,'The program will end so the input file can be
created.\n');
 end;
 if OK == TRUE
% STEP 1
 K = 1;
 OK = FALSE;
% STEP 2
 while OK == FALSE & K <= NN
% ERR is used to test accuracy - it measures the infinity norm
 ERR = 0;
% STEP 3
 for I = 1 : N
 S = 0;
 for J = 1 : N
 S = S-A(I,J)*X1(J);
 end;
 S = W*(S+A(I,N+1))/A(I,I);
 if abs(S) > ERR
 ERR = abs(S);
 end;
 X1(I) = X1(I)+S;
 end;
```

```
% STEP 4
 if ERR <= TOL
 OK = TRUE;
% process is complete
 else
% STEP 5
 K = K+1;
% STEP 6 - is not used since only one vector is required
 end;
 end;
 if OK == FALSE
 fprintf(1,'Maximum Number of Iterations Exceeded.\n');
% STEP 7
% procedure completed unsuccessfully
 else
 fprintf(1,'Choice of output method:\n');
 fprintf(1,'1. Output to screen\n');
 fprintf(1,'2. Output to text file\n');
 fprintf(1,'Please enter 1 or 2.\n');
 FLAG = input(' ');
 if FLAG == 2
 fprintf(1,'Input the file name in the form - drive:\\name.ext\n');
 fprintf(1,'for example:   A:\\OUTPUT.DTA\n');
 NAME = input(' ','s');
 OUP = fopen(NAME,'wt');
 else
 OUP = 1;
 end;
 fprintf(OUP, 'SOR ITERATIVE METHOD FOR LINEAR SYSTEMS\n\n');
 fprintf(OUP, 'The solution vector is :\n');
 for I = 1 : N
 fprintf(OUP, ' %11.8f', X1(I));
 end;
 fprintf(OUP, '\nusing %d iterations with\n', K);
 fprintf(OUP, 'Tolerance  %.10e in infinity-norm\n', TOL);
 fprintf(OUP, 'with Parameter %.10e\n', W);
 if OUP ~= 1
 fclose(OUP);
 fprintf(1,'Output file %s created successfully \n',NAME);
 end;
 end;
 end;
```

Algoritmo 3. Método de sobre-relajación sucesiva.

Nótese que cuando se hace $\omega = 1$ se recuperan el algoritmo de Gauss – Seidel (Algoritmo 2).

Antes de pasar a la siguiente sección vamos a resumir los métodos vistos hasta ahora en la siguiente tabla. Para ello supondremos que (como, por ejemplo, en (0.13)) la matriz A se puede descomponer según $A = D - L - U$, siendo D la matriz compuesta por los elementos de la diagonal principal de A ($diag(A)$) y L y U son las matrices formadas por los elementos (cambiados de signo) que componen la parte triangular inferior y triangular superior, respectivamente, de A.

Método del descenso más rápido

Otro método iterativo para resolver un sistema lineal de ecuaciones tipo $Ax = b$ con $x, b \in \mathbb{R}^n$, $A \in M_n(\mathbb{R})$, $\det(A) \neq 0$, y siendo A simétrica y definida positiva, es el *método del descenso más rápido* (*steepest descent*).

Recordamos que las hipótesis anteriores significan que $A = A^T$ (simétrica) y $x^T Ax > 0$ para $x \neq 0$ (definida positiva).

El método se basa en el resultado siguiente

Si A es una matriz simétrica y definida positiva, el problema de resolver el sistema $Ax = b$ es equivalente al problema de minimizar la forma cuadrática

$$q(x) = \langle x, Ax \rangle - 2\langle x, b \rangle \quad (\in \mathbb{R}) \tag{0.34}$$

No vamos a demostrar el resultado; simplemente usaremos un par de aspectos que se desarrollan en la demostración. En ella se puede comprobar que la forma $q(x + tv)$, con $v \in \mathbb{R}^n$ y $t \in \mathbb{R}$, es mínima para $t = t_{\min} = \langle v, b - Ax \rangle / \langle v, Av \rangle$, tomando como valor

$$q(x + t_{\min} v) = q(x) - \langle v, b - Ax \rangle^2 / \langle v, Av \rangle. \tag{0.35}$$

Este resultado indica que q disminuye al pasar de x a $x + t_{\min} v$, excepto cuando

$$\langle v, b - Ax \rangle^2 / \langle v, Av \rangle = 0 \tag{0.36}$$

esto es, cuando v sea ortogonal al residual ($\langle v, b - Ax \rangle = 0$).

Si x es la solución del sistema $Ax = b$ es claro que se cumple (0.36), siendo x un punto mínimo para (0.34).

El somero esbozo de la demostración nos sirve para diseñar el algoritmo, puesto que sugiere un método iterativo de resolución. La idea es minimizar $q(x)$ a lo largo de distintas trayectorias (dadas por v). A v también se le denomina vector de dirección de búsqueda. En la iteración k del algoritmo se tendrán $\left(x^{(0)}, x^{(1)}, \ldots, x^{(k)}\right)$ y se deberá elegir para cada una de las iteraciones un valor apropiado de $v^{(k)}$. La siguiente iteración se genera del modo siguiente

$$x^{(k)} = x^{(k-1)} + t^{(k-1)} v^{(k-1)} \tag{0.37}$$

siendo

$$t^{(k)} = \langle v^{(k)}, b - Ax^{(k)} \rangle / \langle v^{(k)}, Av^{(k)} \rangle. \tag{0.38}$$

Si $\left\| v^{(k-1)} \right\| = 1$ (es un vector dirección), $x^{(k)}$ mide exactamente la distancia recorrida de $x^{(k-1)}$ a $x^{(k)}$.

El método del descenso más rápido se define suponiendo que $v^{(k)}$ debe ser el gradiente negativo de q en $x^{(k)}$. Se hace esta elección porque la derivada direccional de q (que no es más que la derivada de (0.35)) en un punto según la dirección $v^{(k)}$ mide el cambio en q relativo al cambio de variable en la dirección de $v^{(k)}$. Asimismo se sabe que la derivada direccional en un punto es máxima según la dirección del gradiente en dicho punto [Molina, 1996]. Por lo tanto, la dirección en la que el valor de q en $x^{(k)}$ decrece más, es aquella dada por $-\nabla q(x^{(k)})$. El tal caso, el gradiente lleva la dirección del residuo $r^{(k)} = b - Ax^{(k)}$. Por tanto, el método del descenso más rápido viene dado, a partir de un $x^{(0)}$ inicial, por

$$v^{(k)} = b - Ax^{(k)}, \qquad x^{(k)} = x^{(k-1)} + \frac{\left\langle v^{(k-1)}, v^{(k-1)} \right\rangle}{\left\langle v^{(k-1)}, Av^{(k-1)} \right\rangle} v^{(k-1)}. \tag{0.39}$$

El algoritmo formal para el método del descenso más rápido es el siguiente:

Algoritmo [Kincaid y Cheney, 1990]: Steepest Descent.

Input $A, b_i, x^{(0)}, M$

Output $0, x^{(0)}$

For $k = 0$ **to** $k = M - 1$
  $v^{(k)} \leftarrow b - Ax^{(k)}$
  $t^{(k)} \leftarrow \left\langle v^{(k)}, v^{(k)} \right\rangle / \left\langle v^{(k)}, Av^{(k)} \right\rangle$
  $x^{(k+1)} \leftarrow x^{(k)} + t^{(k)} v^{(k)}$
  Output $k + 1, x^{(k+1)}$
End

Algoritmo 4. Algoritmo formal para el método del descenso más rápido.

Y para su implementación en un computador

Algoritmo [Kincaid y Cheney, 1990]: Steepest Descent.

Input A, b, x, M

Output $0, x$

For $k = 1$ **to** $k = M$
  $v \leftarrow b - Ax$
  $t \leftarrow \langle v, v \rangle / \langle v, Av \rangle$
  $x \leftarrow x + tv$
  Output k, x
End

Algoritmo 5. Método del descenso más rápido.

El método del descenso más rápido se emplea raras veces para resolver problemas unidimensionales como $Ax = b$. El método encuentra mucha mayor aplicación en problemas multidimensionales.

Método del gradiente conjugado

El método del gradiente conjugado es un tipo especial de *método de dirección conjugada*. Estos últimos son aquellos que pretenden minimizar la forma cuadrática (0.34) a lo largo de una sucesión de trayectorias.

En particular, el método del gradiente conjugado aplica a sistemas de ecuaciones lineales como $Ax = b$ en los cuales A es simétrica y definida positiva. En este método, las direcciones $v^{(k)}$, que forman un sistema A-ortogonal, se determinan una a una en cada iteración. Asimismo los residuales $r^{(k)} = b - Ax^{(k)}$, forman un sistema ortogonal (en el sentido clásico), esto es, $\left\langle r^{(j)}, r^{(k)} \right\rangle = 0$ si $j \neq k$.

Al formar las direcciones $v^{(k)}$ un sistema A-ortogonal, el teorema siguiente da las pautas para establecer el algoritmo

Teorema

Sea $\left(v^{(0)}, \ldots, v^{(n)}\right)$ un sistema A-ortogonal de vectores no nulos, para A es simétrica y definida positiva. Entonces dado un $x^{(0)}$ inicial y definiendo

$$x^{(k)} = x^{(k-1)} + \frac{\left\langle b - Ax^{(k-1)}, v^{(k)} \right\rangle}{\left\langle v^{(k)}, Av^{(k)} \right\rangle} v^{(k)}, \qquad (0.40)$$

para $1 \leq i \leq n$, se tiene que $Ax^{(n)} = b$.

El algoritmo formal (no para ser implementado en un computador) es el siguiente

Algoritmo [Kincaid y Cheney, 1990]: Método del gradiente conjugado.

Input $x^{(0)}, A, b, \varepsilon, M$
$r^{(0)} \leftarrow b - Ax^{(0)}$
$v^{(0)} \leftarrow r^{(0)}$

Output $0, x^{(0)}, r^{(0)}$

For $k = 0$ **to** $k = M - 1$
  If $v^{(k)} = 0$ **Then Stop**
  $t^{(k)} \leftarrow \left\langle r^{(k)}, r^{(k)} \right\rangle / \left\langle v^{(k)}, Av^{(k)} \right\rangle$
  $x^{(k+1)} = x^{(k)} + t^{(k)} v^{(k)}$
  $r^{(k+1)} = r^{(k)} - t^{(k)} Av^{(k)}$
  If $\left\| r^{(k+1)} \right\|_2^2 < \varepsilon$ **Then Stop**
  $s^{(k)} \leftarrow \left\langle r^{(k+1)}, r^{(k+1)} \right\rangle / \left\langle r^{(k)}, r^{(k)} \right\rangle$
  $v^{(k+1)} = r^{(k)} + s^{(k)} v^{(k)}$
  Output $k + 1, x^{(k+1)}, r^{(k+1)}$
End

Algoritmo 6. Algoritmo formal para el método del gradiente conjugado.

Nótese que en el algoritmo, si $v^{(k)} = 0$ entonces, teóricamente $x^{(k)}$ es solución del sistema lineal $Ax = b$. La computación requiere almacenar cuatro vectores $x^{(k)}$, $r^{(k)}$, $v^{(k)}$ y $Av^{(k)}$. El algoritmo para computación es el siguiente

Algoritmo [Kincaid y Cheney, 1990]: Método del gradiente conjugado.

Input $x_i^{(0)}, a_{ij}, b_i, \varepsilon, M, \delta$

$r \leftarrow b - Ax$
$v \leftarrow r$
$c \leftarrow \langle r, r \rangle$

Output $0, x^{(0)}, r^{(0)}$

For $k = 1$ **to** $k = M$
  If $\langle v, v \rangle^{1/2} < \delta$ **Then Stop**
  $z \leftarrow Av$
  $t \leftarrow c / \langle v, z \rangle$
  $x \leftarrow x + tv$
  $r \leftarrow r - tz$
  $d \leftarrow \langle r, r \rangle$
  If $d^2 < \varepsilon$ **Then Stop**
  $v \leftarrow r + (d / c) v$
  $c \leftarrow d$
  Output k, x, r
End

Algoritmo 7. Método del gradiente conjugado.

Inicialmente fue concebido como un método directo, se comprobó que la solución exacta no podía obtenerse en muchos casos en un número finito de pasos, así que acabó empleándose como método iterativo. Este método es más recomendable que uno de eliminación Gaussiana en el caso de tener matrices de órdenes elevados y dispersas. En teoría este método conduce a la solución del sistema en, como mucho, n pasos. De hecho, para problemas bien condicionados y sistemas muy grandes, la convergencia en menos iteraciones que el orden del sistema está garantizada.

Bibliografía

Burden, R.L. and Faires, J.D., 1985. Numerical Analysis. PWS-Kent Publishing Company, Boston, EE. UU.

Kincaid, D. and Cheney, W., 1990. Numerical Analysis. Brooks/Cole Publishing, Pacific Grove, California, EE. UU.

Molina, J.J.Q., 1996. Ecuaciones Diferenciales, Análisis Numérico y Métodos Matemáticos. Editorial Santa Rita, Granada.