

Grupo: Adriano Donizete Pila; Daniel Santini Rodrigues; Eduardo Yutaka Uwaide; Gustavo Guedes Alberto; Osmar Tonon.

Capítulo 3. Técnicas Iterativas Para Resolução de Sistemas Lineares

Uma técnica iterativa para resolver um sistema $Ax=b$ começa com uma aproximação inicial $x^{(0)}$ para a solução x e gerando uma sequência de vetores $\{x^{(k)}\}_{k=0}^{\infty}$ que converge para x . A técnica iterativa envolve um processo que converte o sistema $Ax=b$ para um sistema equivalente da forma $x = Tx + c$ para alguma matriz T e vetor c . Depois do vetor $x^{(0)}$ ser selecionado, a sequência de soluções (vetores) aproximada é gerada pela computação:

$$x^{(k)} = T x^{(k-1)} + c \quad \text{com } k=1,2,3,4,\dots$$

Técnicas iterativas são raramente usadas para resolver sistema lineares de pequena dimensão, desde que o tempo requerido para uma exatidão suficiente excede ao tempo requerido pôr técnicas diretas como o método de eliminação de Gauss. Para grandes sistemas com alta porcentagem de zero entrados, no entanto, estas técnicas são eficientes em termos de armazenamento e tempo de computação. Sistemas deste tipos freqüentemente surgem em análise de circuito e soluções numéricas de problemas de fronteiras e equações diferenciais parciais.

Ex:

$$E_1 : 10x_1 - x_2 + 2x_3 = 6$$

$$E_2 : -x_1 + 11x_2 - x_3 + 3x_4 = 25$$

$$E_3 : 2x_1 - x_2 + 10x_3 - x_4 = -11$$

$$E_4 : 3x_2 - x_3 + 8x_4 = 15$$

tem a solução $x = (1, 2, -1, 1)$. Para converter $Ax = b$ para a forma $x = Tx + c$ resolve a equação

E_i para x_i , para $i=1,2,3,4$ para obter:

$$\begin{aligned}
x_1 &= x_2/10 - x_3/5 + 3/5 \\
x_2 &= x_1/11 + x_3/11 - 3/11 x_4 + 25/11 \\
x_3 &= -x_1/5 + x_2/10 + x_4/10 + 11/10 \\
x_4 &= -3/8 x_2 + x_3/8 + 15/8
\end{aligned}$$

No exemplo acima a matriz T e o vetor c são:

$$T = \begin{bmatrix} 0 & 1/10 & -1/5 & 0 \\ 1/11 & 0 & 1/11 & -3/11 \\ -1/5 & 1/10 & 0 & 1/10 \\ 0 & -3/8 & 1/8 & 0 \end{bmatrix} \quad c = \begin{bmatrix} 3/5 \\ 25/11 \\ -11/10 \\ 15/8 \end{bmatrix}$$

Para uma aproximação inicial, fazendo $x^{(0)} = (0,0,0,0)$ e gerando $x^{(1)}$ por:

$$\begin{aligned}
x_1^{(1)} &= x_2^{(1)}/10 - x_3^{(1)}/5 + 3/5 = 0.6000 \\
x_2^{(1)} &= x_1^{(1)}/11 + x_3^{(1)}/11 - 3/11 x_4^{(1)} + 25/11 = 1.998 \\
x_3^{(1)} &= -x_1^{(1)}/5 + x_2^{(1)}/10 + x_4^{(1)}/10 + 11/10 = -0.9998 \\
x_4^{(1)} &= -3/8 x_2^{(1)} + x_3^{(1)}/8 + 15/8 = 0.9998
\end{aligned}$$

Fazendo k iterações, obtemos $x^{(k)} = (x_1^{(k)}, x_2^{(k)}, x_3^{(k)}, x_4^{(k)})$, que são gerados de maneira similar e apresentados

abaixo:

Tabela 3.1.1

K	0	1	2	3	4	5	6	7	8	9	10
$x_1^{(K)}$	0.0000	0.6000	1.0473	0.9326	1.0152	0.9890	1.0032	0.9981	1.0006	0.9997	1.0001
$x_2^{(K)}$	0.0000	2.2727	1.7159	2.0533	1.9537	2.0114	1.9922	2.0023	1.9987	2.0004	1.9998
$x_3^{(K)}$	0.0000	-1.1000	-0.8052	-1.0493	-0.9681	-1.0103	-0.9945	-1.0020	-0.9990	-1.0004	-0.9998
$x_4^{(K)}$	0.0000	1.8750	0.8852	1.1309	0.9739	1.0214	1.0036	1.0036	0.9989	1.0006	0.9998

A decisão para parar depois de 10 iterações é tomada baseada no critério:

$$\|x^{(10)} - x^{(9)}\|_{\infty} = 8 \times 10^{-4} < 10^{-3}$$

$$\|x^{(10)}\|_{\infty} = 1.99998$$

Em fato, $\|x^{(10)} - x\|_{\infty} = 0.0002$.

O método do exemplo anterior é chamado **método iterativo de Jacobi**. Ele consiste em resolver a i -ésima equação em $Ax = b$ por x_i , para obter:

$$(3.1.2) \quad x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1, j \neq i}^n a_{ij} x_j \right), \text{ para } i=1,2,3,\dots$$

e generalizando cada $x_i^{(k)}$ origina dos componentes $x^{(k-1)}$ para $k \geq 1$ por :

$$(3.1.3) \quad x_i^{(k)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1, j \neq i}^n a_{ij} x_j^{(k-1)} \right), \text{ para } i=1,2,3, \dots$$

O método está escrito na forma $x^{(k)} = T x^{(k-1)} + c$ para dividir A em diagonal e diagonal de partes. Para ver melhor isto, deixe D ser a matriz cuja diagonal a mesma que A , - L ser estritamente a triangular menor de A , e U ser estritamente a triangular maior de A . Com esta notação, A é dividida em:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \end{bmatrix} - \begin{bmatrix} 0 & 0 & \dots & 0 \\ -a_{21} & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \end{bmatrix} + \begin{bmatrix} 0 & -a_{12} & \dots & -a_{1n} \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \end{bmatrix}$$

Passo 1 : Iniciar $K = 1$

Passo 2 : Enquanto $(K \leq N)$ fazer passos 3 ao 6

Passo 3 : Para $i = 1, \dots, n$

$$- \sum_{j=1, j \neq i}^n (a_{ij} X_{O_j}) + b_i$$

$$X_i = \frac{\quad}{a_{ii}}$$

Passo 4 : Se $\|x - X\| < \text{TOL}$ então saída(x_1, \dots, x_n);

(Procedimento completado com sucesso)

Parar

Passo 5 : $K = K + 1$

Passo 6 : Para $i = 1, \dots, n$ fazer $X_{O_i} = x_i$

Passo 7 : Saída('Número máximo de iterações foi excedido');

(Procedimento completado sem sucesso)

Parar

Passo 3 do algoritmo requer que $a_{ij} \neq 0$ para cada $i = 1, \dots, n$. Se isso não ocorre, e o sistema não é homogêneo, uma reordenação das equações pode ser feita de tal forma que $a_{ij} \neq 0$. Para acelerar a convergência, as equações devem ser arranjadas de forma que a_{ij} seja tão grande quanto possível. Este tópico é discutido à frente em mais detalhes.

Outro possível critério de parada no passo 4 é iterar até

$$\frac{\|X^{(k)} - X^{(k-1)}\|}{\|X^{(k)}\|}$$

seja menor que alguma tolerância $\varepsilon > 0$. Para esse propósito, uma norma conveniente pode ser usada, a existência usual da norma l_∞ .

Uma melhora no algoritmo anterior é sugerido pôr uma análise da equação 7.3. Para computar $x_i^{(k)}$, os componentes de $x_i^{(k-1)}$ são usados. Desde que, para $i > 1$, $x_1^{(k)}$, ..., $x_{i-1}^{(k)}$ já tenha sido computado e são provavelmente a melhor aproximação para a solução atual x_1, \dots ,

x_{i-1} do que $x_1^{(k-1)}$, ..., $x_{i-1}^{(k-1)}$, isso parece razoável para computar $x_i^{(k)}$ usando estes mais recentes valores calculados; isto é

$$(3.2.1) \quad x_i = \frac{-\sum_{j=1}^{i-1} (a_{ij} x_j^{(k)}) - \sum_{j=i+1}^n (a_{ij} x_j^{(k-1)}) + b_i}{a_{ii}}$$

para cada $i = 1, \dots, n$ ao invés da equação 3.1.3. Isto é chamado de técnica iterativa de Gauss-Seidel e é ilustrada no exemplo a seguir.

Exemplo 2 : O sistema linear dado por

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

foi resolvido no exemplo 1 pelo método iterativo de Jacobi. Incorporando a equação 3.2.1 dentro do algoritmo 3.2 obtém-se a equação para ser usada a cada $k = 1, 2, \dots$

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

Fazendo $x^{(0)} = (0,0,0,0)^t$, nós generalizamos as iterações na tabela 3.2.2

k	0	1	2	3	4	5
$x_1(k)$	0.0000	0.6000	1.030	1.0065	1.0009	1.0001
$x_2(k)$	0.0000	2.3272	2.037	2.0036	2.0003	2.0000
$x_3(k)$	0.0000	-0.9873	-1.014	-1.0025	-1.0003	-1.0000
$x_4(k)$	0.0000	0.8789	0.9844	0.9983	0.9999	1.0000

Desde que

$$\frac{\|x^{(5)} - x^{(4)}\|_{\infty}}{\|x^{(5)}\|_{\infty}} = \frac{0.0008}{2.000} = 4.10^{-4},$$

$x^{(5)}$ é uma aproximação razoável para a solução. Note que o método de Jacobi no exemplo 1 necessitou duas vezes mais o número de iterações para o mesmo resultado.

Para escrever o método de Gauss-Seidel na forma matricial, multiplica-se ambos os lados da equação 7.5 por a_{ij} e junte todos os k -ésimos termos da iteração para obter

$$a_{i1}x_1^{(k)} + a_{i2}x_2^{(k)} + \dots + a_{ii}x_i^{(k)} = -a_{i,i+1}x_{i+1}^{(k-1)} - \dots - a_{in}x_n^{(k-1)} + b_i,$$

para cada $i = 1, 2, \dots, n$. Escrevendo todas as n equações obtemos

$$\begin{array}{rcl} a_{11}x_1^{(k)} & = & -a_{12}x_2^{(k-1)} - a_{13}x_3^{(k-1)} - \dots - a_{1n}x_n^{(k-1)} + b_1, \\ a_{21}x_1^{(k)} + a_{22}x_2^{(k)} & = & -a_{23}x_3^{(k-1)} - \dots - a_{2n}x_n^{(k-1)} + b_2, \\ \vdots & & \vdots \\ a_{n1}x_1^{(k)} + a_{n2}x_2^{(k)} + \dots + a_{nn}x_n^{(k)} & = & b_n, \end{array}$$

e disto seque que a forma matricial para o método de Gauss-Seidel é

$$(D - L)x^{(k)} = Ux^{(k-1)} + (D - L)^{-1}b$$

ou

$$(3.2.3) \quad x^{(k)} = (D - L)^{-1} Ux^{(k-1)} + (D - L)^{-1}b, \text{ para cada } k = 1, 2, \dots$$

3.2. Método Iterativo de Gauss-Seidel

Para solucionar $Ax = b$ tendo uma aproximação inicial $x^{(0)}$:

Entrada : o número de equações e incógnitas n ; as entradas a_{ij} , $1 \leq i, j \leq n$ da matriz A ;

as entradas x_0^i , $1 \leq i \leq n$ de $x_0 = x^{(0)}$; tolerância TOL ; número máximo de iterações N .

Saída : a solução aproximada x_1, \dots, x_n ou a mensagem que o número de iterações foi excedido.

Algoritmo

Passo 1 : Iniciar $K = 1$

Passo 2 : Enquanto ($K \leq N$) fazer passos 3 ao 6

Passo 3 : Para $i = 1, \dots, n$

$$x_i = \frac{- \sum_{j=1}^{i-1} a_{ij} x_j - \sum_{j=i+1}^n a_{ij} x_{oj} + b_i}{a_{ii}}$$

Passo 4 : Se $\|x - x_o\| < \text{TOL}$ então saída(x_1, \dots, x_n);

(Procedimento completado com sucesso)

Parar

Passo 5 : $K = K + 1$

Passo 6 : Para $i = 1, \dots, n$ fazer $x_{oi} = x_i$

Passo 7 : Saída('Número máximo de iterações foi excedido');

(Procedimento completado sem sucesso)

Parar

Os comentários seguintes sobre o algoritmo 3.2, atendendo a reordenação e critério de parada também aplicáveis para os algoritmo 3.3 de Gauss-Seidel.

Os resultados dos exemplos 1 e 2 aparecem para implicar que o método de Gauss-Seidel é superior ao método de Jacobi. Isto é generalizado, mas não sempre, verdadeiro. Existem sistemas lineares para os quais o método de Jacobi converge e o método de Gauss-Seidel não, e outros para os quais o método de Gauss-Seidel converge e o método de Jacobi não. (Ver Vaga[148], pág. 74)

O resultado dos exemplos 1 e 2 parece implicar que o método Gauss-Seidel é superior ao método de Jacobi. Isto é geralmente, mas nem sempre, verdade. Existem sistemas lineares, para o qual o método de Jacobi converge e o método de Gauss não, e outros para o qual o método de Gauss-Seidel converge e o método de Jacobi não.

Para estudar a convergência de técnicas genéricas de iteração, nós consideramos a fórmula.

$$x^{(k)} = Tx^{(k-1)} + c \text{ para cada } k = 1, 2, \dots$$

onde $x^{(0)}$ é arbitrário. Para a compreensão do lema 3.3.1, a definição abaixo é importante.

DEFINIÇÃO: O raio espectral $\rho(A)$ de uma matriz A é definido por:

$$\rho(A) = \max |y|, \text{ onde } y \text{ é um auto-valor de } A.$$

LEMA 3.3.1:

Se o raio espectral $\rho(T)$ satisfaz $\rho(T) < 1$, então $(1-T)^{-1}$ existe e

$$(1-T)^{-1} = 1 + T + T^2 + \dots$$

PROVA :

Para qualquer autovalor λ de T , $1-\lambda$ é um autovalor de $1-T$. Desde que $|\lambda| \leq \rho(T) < 1$, segue-se que nenhum autovalor de $1-T$ pode ser zero e, conseqüentemente, que $1-T$ não é singular.

Deixe $S_m = 1 + T + T^2 + \dots + T^m$. Então

$$(1-T)S_m = 1 - T^{m+1}$$

e desde que T é convergente, o resultado no final da seção 3.3 implica que

$$\lim_{m \rightarrow \infty} (1-T)S_m = \lim_{m \rightarrow \infty} (1 - T^{m+1}) = 1$$

Deste modo, $\lim_{m \rightarrow \infty} S_m = (1-T)^{-1}$.

TEOREMA 3.3.2

Para qualquer $x^{(0)} \in \mathbb{R}^n$, a seqüência $\{x^{(k)}\}_{k=0}^{\infty}$ definida pôr

$$(*) \quad x^{(k)} = Tx^{(k-1)} + c \text{ para cada } k \geq 1 \text{ e } c \neq 0 \text{ converge para a solução única de } x = Tx + c$$

se e somente se $\rho(T) < 1$

PROVA

$$\text{Da equação } (*) \quad x^{(k)} = Tx^{(k-1)} + c$$

$$= T(Tx^{(k-2)} + c) + c$$

$$\begin{aligned}
&= T^2 x^{(k-2)} + (T + 1)c \\
&= T^k x^{(0)} + (T^{k-1} + \dots + T + 1)c
\end{aligned}$$

Se $\rho(T) < 1$, então o resultado do final da última seção implica que $\lim_{k \rightarrow \infty} T^k x^{(0)} = 0$

Usando isto e o lema 3.3.1 dá

$$\lim_{k \rightarrow \infty} x^{(k)} = \lim_{k \rightarrow \infty} T^k x^{(0)} + \lim_{k \rightarrow \infty} \left(\sum_{j=0}^{k-1} T^j \right) c = 0 + (1-T)^{-1} c = (1-T)^{-1} c$$

Desde que $x = Tx + c$ implica que $(1-T)x = c$, a sequência $\{x^{(k)}\}$ converge para a única solução para a equação, o vetor $x = (1-T)^{-1}c$

Para provar, primeiro note que para qualquer $x^{(0)}$

$$x - x^{(k)} = T(x - x^{(k-1)}) = \dots = T^k(x - x^{(0)})$$

Assuma z sendo qualquer vetor de \mathbb{R}^n e define $x^{(0)} = x - z$.

Então

$$\lim_{k \rightarrow \infty} T^k z = \lim_{k \rightarrow \infty} (T^k(x - x^{(0)})) = \lim_{k \rightarrow \infty} (x - x^{(k)}) = x - x = 0$$

Pelo teorema 3.3.2 do final da seção anterior, isto é equivalente a ter $\rho(T) < 1$

COROLÁRIO 3.3.3

Se $\|T\| < 1$ para qualquer norma matriz natural, então a sequência $\{x^{(k)}\}_{k=0}^{\infty}$ na Eq. (7.7)

converge, para qualquer $x^{(0)} \in \mathbb{R}^n$, para um vetor $x \in \mathbb{R}^n$, e a seguir o limite do erro suportado:

i) $\|x - x^{(k)}\| \leq \|T\|^k \|x^{(0)} - x\|;$

ii) $\|x - x^{(k)}\| \leq (\|T\|^k / 1 - \|T\|) \|x^{(1)} - x^{(0)}\|.$

Para aplicar o resultado precedente para as técnicas iterativas de Jacobi ou Gauss-Seidel, nós precisamos escrever as matrizes de iteração para o método de Jacobi, T_j , dada na equação (3.1.4) e para o método de Gauss-Seidel, T_g , dada na Eq. (3.2.3) como :

$$T_j = D^{-1}(L + U) \text{ e } T_g = (D - L)^{-1}U$$

Se $\rho(T_j)$ or $\rho(T_g)$ for menor que 1, então a correspondente sequência $\{x^{(k)}\}_{k=0}^{\infty}$ poderá convergir para a solução x de $Ax = b$. Por exemplo, o esquema Jacobi for

$$x^{(k)} = D^{-1}(L + U)x^{(k-1)} + D^{-1}b$$

e se $\{x^{(k)}\}_{k=0}^{\infty}$ converge para x , então

$$x = D^{-1}(L + U)x + D^{-1}b$$

Isto implica que

$$Dx = (L+U)x + b \quad \text{e} \quad (D - L - U)x = b$$

Desde que $D - L - U = A$, x satisfaz $Ax = b$.

Nós podemos agora facilmente verificar as condições suficientes para convergência dos métodos de Jacobi e de Gauss-Seidel. Os comentários a seguir do algoritmo 3.2 com respeito a reordenação e critérios de parada também é aplicado para o algoritmo 3.3 de Gauss-Seidel.

Teorema 3.3.4 Se A é estritamente dominante, então para qualquer escolha de $x^{(0)}$, tanto o método de Jacobi como o de Gauss-Seidel dá a sequência $\{x^{(k)}\}_{k=0}^{\infty}$ que converge para a solução única de $Ax = b$.

- relacionamento da rapidez da convergência do raio espectral da matriz de iteração T pode ser vista do Corolário 3.3.3. Desde que as inequações satisfazem, para todo natural, a forma da matriz segue da afirmação do Teorema “se A é uma matriz $n \times n$, então:

i) $[\rho(A^t A)]^{1/2} = \|A\|_2$,

ii) $\rho(A) \leq \|A\|$ ”.

(3.3.5) $\|x^{(k)} - x\| \approx \rho(T)^k \|x^{(0)} - x\|$

Além disso, é desejável selecionar a técnica iterativa com mínimo $\rho(T) < 1$ para um sistema particular $Ax = b$.

Não há resultados gerais para afirmar qual das duas técnicas, Jacobi ou Gauss-Seidel, será melhor para um sistema linear arbitrário. Em casos especiais, entretanto, a resposta é conhecida, como é demonstrado no teorema a seguir. A demonstração deste resultado pode ser encontrada em Young [159], páginas 120-127.

Teorema 3.3.6 (Stein-Rosenberg)

Se $a_{ij} \leq 0$ para $i \neq j$ e $a_{ii} > 0$ para cada $i = 1, 2, \dots, n$, então uma e somente uma afirmação das seguintes é verdadeira:

- a. $0 \leq \rho(T_g) < \rho(T_j) < 1$,
- b. $1 < \rho(T_j) < \rho(T_g)$,
- c. $\rho(T_j) = \rho(T_g) = 0$,
- d. $\rho(T_j) = \rho(T_g) = 1$.

Para o caso especial descrito no Teorema 7.3.6, observamos que quando um método converge, então ambos convergem, e o método de Gauss-Seidel converge mais rapidamente que o de Jacobi.

Desde que a taxa de convergência de um procedimento depende do raio espectral da matriz associada com o método, um modo de selecionar um procedimento para acelerar a convergência é escolher um método cuja matriz associada tem mínimo raio espectral. Antes de descrever um procedimento para selecionar tal método, precisamos introduzir novos significados para medidas de valor que aproximam a solução de um sistema linear que difere da solução do sistema. O método utiliza o vetor descrito na definição seguinte.

Definição 3.3.7 Suponha que $x \in \mathbb{R}^n$ é uma aproximação para a solução do sistema linear definido por $Ax = b$. O vetor residual para x , com relação a este sistema, é definido por $r = b - Ax$.

Em procedimentos tais como os métodos de Jacobi ou de Gauss-Seidel, o vetor residual está associado com cada cálculo de um componente aproximado para o vetor solução. O objetivo do método é gerar uma sequência de aproximações que irão gerar os vetores residuais associados para convergir rapidamente a zero. Suponha que tomemos

$$\mathbf{r}_i^{(k)} = (r_{1i}^{(k)}, r_{2i}^{(k)}, \dots, r_{ni}^{(k)})^t$$

que denota o vetor residual para o método de Gauss-Seidel, correspondendo a solução aproximada do vetor solução $\mathbf{x}_i^{(k)}$ definido por

$$\mathbf{x}_i^{(k)} = (x_1^{(k)}, x_2^{(k)}, \dots, x_{i-1}^{(k)}, x_i^{(k-1)}, \dots, x_n^{(k-1)})^t$$

O m-ésimo componente de $\mathbf{r}_i^{(k)}$ é

$$(3.3.8) \quad r_{mi}^{(k)} = b_m - \sum_{j=1}^{i-1} a_{mj} x_j^{(k)} - \sum_{j=1}^n a_{mj} x_j^{(k-1)}$$

ou, equivalentemente,

$$r_{mi}^{(k)} = b_m - \sum_{j=1}^{i-1} a_{mj} x_j^{(k)} - \sum_{j=i+1}^n a_{mj} x_j^{(k-1)} - a_{mi} x_i^{(k-1)}$$

para cada $m = 1, 2, \dots, n$.

Em particular, o i-ésimo componente de $\mathbf{r}_i^{(k)}$ é

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

então

$$(3.3.9) \quad a_{ii} x_i^{(k-1)} + r_{ii}^{(k)} = b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^n a_{ij} x_j^{(k-1)}$$

Lembrando, entretanto, que no método de Gauss-Seidel, $\mathbf{x}_i^{(k)}$ é escolhido como

$$(3.3.10) \quad x_i^{(k)} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^n a_{ij} x_j^{(k-1)} \right],$$

então Equação (3.3.9) pode ser reescrita como

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

Conseqüentemente, o método de Gauss-Seidel pode ser caracterizado como escolhas de $x_i^{(k)}$ que satisfazem

$$(3.3.11) \quad x_i^{(k)} = x_i^{(k-1)} + \frac{r_{ii}^{(k)}}{a_{ii}}$$

Podemos observar outra conexão entre os vetores residuais e a técnica de Gauss-Seidel. Considere o vetor residual $r_{i+1}^{(k)}$ associado com o vetor $x_{i+1}^{(k)} = (x_1^{(k)}, \dots, x_i^{(k-1)}, x_{i+1}^{(k-1)}, \dots, x_n^{(k-1)})^t$, o i-ésimo componente de r é. Por (3.3.8), o i-ésimo componente de $r_{i+1}^{(k)}$ é

$$\begin{aligned} r_{i,i+1}^{(k)} &= b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^n a_{ij}x_j^{(k-1)} \\ &= b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^n a_{ij}x_j^{(k-1)} - a_{ii}x_i^{(k)} \end{aligned}$$

Equação (3.3.10) implica que $r_{i,i+1}^{(k)} = 0$. De certo modo, então, a técnica de Gauss-Seidel é também caracterizada pelo requerimento de que o i-ésimo componente de $r_{i+1}^{(k)}$ é zero.

Reduzindo uma coordenada do vetor residual para zero ,contudo, não é geralmente o caminho mais eficiente para reduzir a norma do vetor r_{i+1} . De fato, modificando o procedimento de GAUSS SEIDEL dado pela equação (3.3.11) para

$$(3.3.12) \quad x_i^{(k)} = x_i^{(k-1)} + w \cdot r_{ii}^{(k)} / a_{ii}$$

para escolhas certas de positivos w isto conduzirá significativamente mais rápida para a convergência.

Métodos envolvendo Eq. (3.3.12) são chamados de métodos de descanso. Para escolhas de $0 < w < 1$,os procedimentos são chamados de métodos de baixo descanso e podem ser usados para obter convergência de alguns sistemas que não são convergentes pelo método GAUSS SEIDEL . Para escolhas de $w > 1$, os procedimentos são chamados de métodos de alto descanso que são usados para acelerar a convergência de sistemas que são convergentes pela

técnica de GAUSS SEIDEL. Esses métodos são abreviados por SOR para SUCCESSIVE OVER-RELAXATION e são particularmente proveitosos para resolver sistemas lineares que ocorrem em soluções numéricas de certas equações diferenciais parciais .

Antes de ilustrar as desvantagens do método SOR nós notamos que usando Eq(3.3.9) ,a Eq (3.3.12) pode ser reformulada para propósitos de cálculo para

$$x_i^{(k)} = (1-w) x_i^{(k-1)}$$

Para determinar a forma da matriz do método de SOR nós rescrevemos isto como
então

$$(D-wL)x^{(k)} = [(1-w)D + wU]x^{(k-1)} + wb$$

ou

$$(3.3.13) \quad x^{(k)} = (D - wL)^{-1} [(1-w)D + wU]x^{(k-1)} + w(D-wL)^{-1}b$$

Exemplo 3

O sistema linear $Ax=b$ dado por

$$4x_1 + 3x_2 = 24$$

$$3x_1 + 4x_2 - x_3 = 30$$

$$-x_2 + 4x_3 = -24$$

tem uma solução $(3,4,-5)^t$. GAUSS SEIDEL e o método SOR com $w=1.25$ serão usado para resolver este sistema usando $x^{(0)}=(1,1,1)^t$ para ambos métodos. As equações para o método de GAUSS SEIDEL são

$$x_1^{(k)} = -0.75x_2^{(k-1)} + 6$$

$$x_2^{(k)} = -0.75x_1^{(k)} + 0.25x_3^{(k-1)} + 7.5$$

$$x_3^{(k)} = 0.25x_2^{(k)} - 6$$

e as equações para o método SOR com $w=1.25$ são

$$x_1^{(k)} = -0.25x_1^{(k-1)} - 0.9375x_2^{(k-1)} + 7.5$$

$$x_2^{(k)} = -0.9375x_1^{(k)} - 0.25x_2^{(k-1)} + 0.3125x_3^{(k-1)} + 9.375$$

$$x_1^{(k)} = 0.3125x_2^{(k)} - 0.25x_3^{(k-1)} - 7.5$$

As primeiras sete interações para cada método são listadas na tabela 3.3.14 e 3.3.15. Para as interações serem exatas para sete casas decimais o método GAUSS-SEIDEL requer 34 interações, ao passo que para o método SOR com $w=1.25$, 14 interações são necessárias.

Tabela 3.3.14 - Gauss-Seidel

k	0	1	2	3	4	5	6	7
x1	1	5.250000	3.1406250	3.0878906	3.0549316	3.0343323	3.0214577	3.0134110
x2	1	3.812500	3.8828125	3.9267578	3.9542236	3.9713898	3.9821186	3.988824
X3	1	-5.046875	-5.022969	-5.0183105	-5.0114441	-5.0071526	-5.0044703	-5.002794

Tabela 3.3.15 - Sor com $w=1.25$

k	0	1	2	3	4	5	6	7
x1	1	6.312500	2.6223145	3.1333027	2.9570512	3.0037211	2.9963276	3.00004
x2	1	3.5195313	3.9585266	4.0102646	4.0074838	4.0029250	4.0009262	4.00025
x3	1	-6.6501465	-4.6004238	-5.0966863	-4.9734897	-5.0057135	-4.9982822	-5.00034

A questão óbvia a perguntar é como o valor apropriado de w é escolhido. Apesar de nenhuma resposta completa para esta questão seja conhecida para um geral sistema linear $n \times n$, o seguinte resultado pode ser usado em certas situações.

TEOREMA 3.3.16 (KAHAN)

Se $a_{ii} \neq 0$ para cada $i=0,1,2,\dots,n$ então $\rho(T_w) \leq |w-1|$. Isto implica que $\rho(T_w) < 1$ somente se $0 < w < 2$ onde

$$T_w = (D - wL)^{-1} [(1-w)D + wU]$$

é a matriz iterada para o método SOR

TEOREMA 3.3.17(Ostrowsi-Reich)

Se A é uma matriz definida positiva e $0 < w < 2$ então o método SOR converge para alguma escolha de vetores de aproximações iniciais $x^{(0)}$.

TEOREMA 3.3.18

Se A é definida positiva e tridiagonal então $\rho(T_g) = [\rho(T_j)]^2 < 1$ e a ótima escolha de w para o método SOR é

$$w = 2/(1+\sqrt{1-\rho(T_g)}) = 2/(1+\sqrt{1-[\rho(T_j)]^2})$$

Com esta escolha de w $\rho(T_w) = w-1$

EXEMPLO 4

No exemplo 3 a matriz A era dada por

$$A = \begin{bmatrix} 4 & 3 & 0 \\ 3 & 4 & -1 \\ 0 & -1 & 4 \end{bmatrix}$$

Esta matriz é definida positiva e tridiagonal então Teorema 3.3.18 aplica

$$T^j = D^{-1} (L + U)$$

$$= \begin{bmatrix} 1/4 & 0 & 0 \\ 0 & 1/4 & 0 \\ 0 & 0 & 1/4 \end{bmatrix} * \begin{bmatrix} 0 & -3 & 0 \\ -3 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -0.75 & 0 \\ -0.75 & 0 & 0.25 \\ 0 & 0.25 & 0 \end{bmatrix}$$

nós temos

$$T_j - \lambda I = \begin{bmatrix} -\lambda & -0.75 & 0 \\ -0.75 & -\lambda & 0.25 \\ 0 & 0.25 & -\lambda \end{bmatrix}$$

então

$$\det(T_j - \lambda I) = -\lambda(\lambda^2 - 0.625)$$

Portanto $\rho(T_j) = \sqrt{0.625}$

e

$$w = 2/(1+\sqrt{1-\rho(T_g)}) = 2/(1+\sqrt{1-[\rho(T_j)]^2}) = 2/(1+\sqrt{1-0.625}) \approx 1.24$$

Isto explica a rápida convergência obtida no exemplo 3 usando $w=1.25$.

3.3. Programas, Algoritmos e Fluxograma

- Algoritmo SOR :

Para resolver $Ax=b$ dado o parâmetro w e a aproximação inicial $x^{(0)}$.

Entrada:

Número de equações e incógnita h, entradas a_{ij} $1 \leq i, j \leq n$ da matriz A, as entradas b_i , $1 \leq i \leq n$ de b, as entradas XO_i , $1 \leq i \leq n$ de $XO = x^{(0)}$, o parâmetro w, tolerância Tol, máximo número de interação N

Saída:

As aproximações das soluções x_1, \dots, x_n ou a mensagem que o número de interações foi excedido.

Passo 1 : $k=1$

Passo2 : enquanto ($k \leq N$) faça passos 3-6

Passo3 : para $i = 1, 2, \dots, n$

$$x_i = (1 - \omega)XO_i + \frac{\omega(-\sum_{j=1}^{i-1} a_{ij} x_j - \sum_{j=i+1}^n a_{ij} XO_j + b_i)}{a_{ii}}$$

Passo4: Se $\|x - XO\| < TOL$ então SAÍDA (x_1, \dots, x_n) FIM

Passo5: $k = k+1$

Passo6 : para $i = 1, 2, \dots, n$ $XO_i = x_i$.

Passo7 : SAÍDA (Número máximo de iterações excedido) FIM

- Programa Gauss-Seidel:

```
// Programa p/ solucao de sistemas lineares :
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <conio.h>
```

```
#include <iostream.h>
```

```

#include <math.h>

#define matrizmax 20

float matriz[matrizmax][matrizmax];

int numeq;

float xo[matrizmax],xj[matrizmax];

float tol;

int k=0;

int i,j,h,n;

float par1,par2,par3,par4;

struct soluc {

    float x[matrizmax];

    } solucao[matrizmax];

void lermatriz(void)

{

printf("Entre com o numero de equacoes : ");

scanf("%i",&numeq);

for(int q=0;q<numeq;q++)

{

```

```

for(int p=0;p<numeq+1;p++)

{printf("Entre com o %io. coeficiente da %ia. equacao : ",p+1,q+1);

scanf("%f",&matriz[q][p]);

}

}

}

void main(void)

{

clrscr();

lermatriz();

clrscr();

printf("Entre com a tolerfncia");

scanf("%f",&tol);

for (i=0;i<numeq;i++)

{

printf("Entre com o valor inicial para %ia equa#o : ",i+1);

scanf("%f",&xo[i]);

}

```

```

for(i=0;i<numeq;i++)

solucao[k].x[i]=xo[i];//atribui os valores iniciais para para solucao da equacao

printf("Entre com o valor de iteracoes que se deseja");

scanf("%i",&n);

k=1;

while (k<=n)

{

for (i=0;i<numeq;i++) //controla os valores de cada variavel na iteracao k

{ par1=par2=0;

for (int j=0;j<=i-1;j++) //calcula a primeira somatoria

par1+=matriz[i][j]*xj[j];

for ( j=i+1;j<=numeq;j++) //calcula a segunda somatoria

par2 += matriz[i][j]*xo[j];

solucao[k].x[i] = (-par1 - par2+ matriz[i][numeq])/matriz[i][i];

for (int y=0;y<j;y++)

xj[y]=solucao[k].x[y];

}

par2=0;

```

```

for

(j=0;j<numeq;j++)    //calcula

{
    par1=0;

par1 += solucao[k].x[j] - xo[j];

par2 += (par1)*(par1);

}

if ( (sqrt(par2)) < tol)

{

for (i=0;i<numeq;i++)

printf("\n%4.7f", solucao[k].x[i]);

exit(1);

}

for(i=0;i<numeq;i++)

xo[i] = solucao[k].x[i];

k++ ;

}

//for(i=0;i<numeq;i++)

//printf("\n%2.5f",xo[i]);

```

```
printf("\nNumero maximo de iteracoes excedidas");

printf("\n%i",k);

}
```

- Programa Jacobi :

```
// Programa para resolver sistemas lineares atraves da tecnica iterativa

//de Jacobi

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

#include<iostream.h>

#include<math.h>

#define matrizmax 20

float matriz[matrizmax][matrizmax];

float x0[matrizmax],tol,somatoria1,somatoria2,comp;

int numeq,k,n,i,j;

struct soluc

{

float x[matrizmax];
```

```

} solucao[matrizmax];

void lermatriz(void)

{

printf("Entre com o numero de equacoes : ");

scanf("%i",&numeq);

for(int q=0;q<numeq;q++)

{

for(int p=0;p<numeq+1;p++)

{

printf("Entre com o %io. coeficiente da %ia. equacao : ",p+1,q+1);

scanf("%f",&matriz[q][p]);

}

}

}

void main(void)

{

clrscr();

lermatriz();

```



```

printf("Entre com a tolerancia");

scanf("%f",&tol);

printf("Entre com o numero de interacoes");

scanf("%i",&n);

for (i=0;i<numeq;i++)

{

printf("Entre com o valor inicial para a %i variavel",i+1);

scanf("%f",&x0[i]);

}

int k=1;

clrscr();

while(k<=n)

{

for(i=0;i<numeq;i++)

{

somatical=0;

for(j=0;j<numeq;j++)

{

```

```

if(i!=j)

somatoria1+= -matriz[i][j]*x0[j];

}

solucao[k-1].x[i]=(somatoria1+ matriz[i][numeq])/matriz[i][i];

}

somatoria1=somatoria2=0;

for(j=0;j<numeq;j++)

{

somatoria1=pow((solucao[k-1].x[j]-x0[j]),2);

somatoria2=somatoria2+somatoria1;

}

comp=sqrt(somatoria2);

if(comp<tol)

{ for (j=0;j<numeq;j++)

printf("\n%4.4f",x0[j]);

printf("\nO numero de iteracoes foi %i",k);

exit(1);

}

```

```

}

for(j=0;j<numeq;j++)

x0[j]=solucao[k-1].x[j];

k+=1;

for (i=0;i<k;i++)

{

for (j=0;j<numeq;j++)

printf("\n%4.4f",solucao[i].x[j]);

printf("\n");

}

}

}

```

- Programa SOR :

```

// Programa p/ solucao de sistemas lineares usando m, todo SOR

//*****//

// COMENTARIOS SOBRE AS VARIAVEIS

//MATRIZ:array bidimensional usado para armazenar o sistema fornecido pelo

//usuario

```

```

//numeq:numero de equacoes que o sistema possui

//xo:armazena a solucao momentanea do sistema

//xj:usado para calcular xi,dado que xj contem a solucao parcial do sistema

//solucao:array de uma estrutura, que armazena todas as aproximacoes

//tol:precisao desejada para resolucao do sistema

//somatori1,somatoria2:variaveis auxiliares usadas para aproximar as solucoes

//omega:parametro usado para aumentar a velocidade de convergencia da solucao

//i,j,k:contadores

//*****

//

//Declara de bibliotecas e variaveis usadas no programa

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <iostream.h>

#include <math.h>

#define matrizmax 20

float matriz[matrizmax][matrizmax];

```

```

int numeq;

float xo[matrizmax],xj[matrizmax],somatoria1,somatoria2,omega,tol;

int k=0;

int i,j,n;

struct soluc {

float x[matrizmax];

} solucao[matrizmax];

//Procedimento para ler o sistema

void lermatriz(void)

{

printf("Entre com o numero de equacoes : ");

scanf("%i",&numeq);

for(int q=0;q<numeq;q++)

{

for(int p=0;p<numeq+1;p++)

{

printf("Entre com o %io. coeficiente da %ia. equacao : ",p+1,q+1);

scanf("%f",&matriz[q][p]);

```

```

}

}

}

//Programa principal

void main(void)

{

    clrscr();

    lermatriz();

    clrscr();

    printf("Entre com a tolerfncia");

    scanf("%f",&tol);

    for (i=0;i<numeq;i++)

    {

        printf("Entre com o valor inicial para %ia equa#o : ",i+1);

        scanf("%f",&xo[i]);

    }

    for(i=0;i<numeq;i++)

        solucao[k].x[i]=xo[i];//atribui os valores iniciais para para solucao da equacao

```

```

printf("Entre com o valor de iteracoes que se deseja");

scanf("%i",&n);

printf("Entre com o valor de omega");

scanf("%f",&omega);

k=1;

while (k<=n)

{

for (i=0;i<numeq;i++)

{

somatoria1=somatoria2=0;

for (int j=0;j<=i-1;j++)

somatoria1+=matriz[i][j]*xj[j];

for ( j=i+1;j<=numeq;j++)

somatoria2 += matriz[i][j]*xo[j];

solucao[k].x[i]      =((1-omega)*xo[i])+(omega*(-somatoria1      -      somatoria2+
matriz[i][numeq]))/matriz[i][i];

for (int y=0;y<j;y++)

xj[y]=solucao[k].x[y];

```

```

    }

    somatoria1=somatoria2=0;

    for(j=0;j<numeq;j++)

    {

        somatoria1=pow((solucao[k].x[j]-xo[j]),2);

        somatoria2=somatoria2+somatoria1;

    }

    if ( (sqrt(somatoria2)) < tol)

    {

        for (i=0;i<numeq;i++)

            printf("\n%4.7f", solucao[k].x[i]);

        exit(1);

    }

    for(i=0;i<numeq;i++)

        xo[i] = solucao[k].x[i];

    k++ ;

}

printf("\nNumero maximo de iteracoes excedidas");

```



```
printf("Tentativa fracassada, experimente novamente com maior no. de tentativas");

}
```

- Fluxograma genérico para os casos : Jacobi, Gauss-Seidel e SOR:

No fluxograma que seque, x_i assume os seguintes valores de acordo com os algoritmos :

Jacobi

$$x_i = \frac{\sum_{j=1, j \neq i}^n (a_{ij} X_{oj}) + b_i}{a_{ii}}$$

Gauss-Seidel

$$x_i = \frac{- \sum_{j=1}^{i-1} a_{ij} x_j - \sum_{j=i+1}^n a_{ij} X_{Oj} + b_i}{a_{ii}}$$

SOR

$$x_i = (1 - \omega)X_{Oj} + \frac{\omega(- \sum_{j=1}^{i-1} a_{ij} x_j - \sum_{j=i+1}^n a_{ij} X_{Oj} + b_i)}{a_{ii}}$$

