Universidade Federal do Rio Grande
do Norte
Instituto Metrópole Digital
Bacharelado em Tecnologia da
Informação

Métodos para solucionar sistemas lineares

Raquel Lopes de Oliveira
Vinícius Campos Tinoco Ribeiro
Vitor Rodrigues Greati

Docente
Luiz Amorim Carlos
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA

Natal(RN), abril de 2017

Universidade Federal do Rio Grande
do Norte
Instituto Metrópole Digital
Bacharelado em Tecnologia da
Informação

Métodos para solucionar sistemas lineares

Raquel Lopes de Oliveira
Vinícius Campos Tinoco Ribeiro
Vitor Rodrigues Greati

Docente
Luiz Amorim Carlos
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA

Relatório apresentado à disciplina de Cálculo Numérico para Computação (DIM0404) para contabilização de nota parcial da primeira unidade no semestre 2017.1.

Natal(RN), abril de 2017

Sumário

Lista de Algoritmos

Introdução

Sistemas lineares são conjuntos de equações lineares, aparecendo em diversos problemas nos vários campos das ciências exatas e engenharias. Comumente, são representados na forma matricial $Ax = b$, onde

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix},$$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix},$$

$$b = \begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ \vdots \\ k_m \end{bmatrix}.$$

O objetivo, ao se resolver um sistema linear, é encontrar o vetor x que, transformado por A, produz b. Um sistema linear pode ser impossível, quando não tem solução (ou seja, b não pode ser produzido por uma combinação linear das colunas de A); possível e indeterminado, quando pos-

sui infinitas soluções; ou possível e determinado, quando possui uma única solução.

A eliminação Gaussiana é geralmente o método aplicado para encontrar a solução de um sistema linear. No contexto computacional, outros métodos - baseados ou não nessa eliminação - merecem atenção, e são eles o objeto deste trabalho.

Este relatório abarca quatro algoritmos implementados para a resolução de sistemas lineares possíveis determinados, elucidando os princípios que os guiam e apresentando as implementações e resultados quanto à precisão e ao esforço computacional exigidos por cada um.

Os primeiros métodos apresentados são chamados de diretos, consistindo, em essência, na fatoração da matriz A em matrizes triangulares e subsequente resolução de dois sistemas lineares de mais fácil resolução, alcançando o resultado do sistema original. Na seção seguinte, são apresentados dois métodos indiretos, os quais consistem na busca pela solução no espaço vetorial utilizando um operador que, dado que atenda a certas condições, sempre produz vetores mais próximos do resultado desejado, até que finaliza após atingir uma margem de erro predefinida.

Desenvolvimento

1 Métodos Diretos

1.1 LU com Pivotamento Parcial

O método LU consiste na fatoração da matriz $A_{n\times n}$ em duas matrizes, uma triangular inferior, L, e outra triangular superior, U, de forma que $A = LU$. Para tanto, são necessárias as chamadas **matrizes de transformação de Gauss**, que possuem a seguinte forma, para $1 \leq i \leq n$:

$$G_i = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & \frac{-u^{i-1}_{(i+1)(i)}}{u^{i-1}_{ii}} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \frac{-u^{i-1}_{mi}}{u^{i-1}_{ii}} & 0 & \dots & 1 \end{pmatrix} \tag{1}$$

onde u^i_{kj} é o elemento na linha k e coluna j da matriz U_{i-1}, sendo $U_i = G_i \cdot U_{i-1}$, considerando $U_0 = A$.

Calculadas as matrizes de transformação de Gauss, temos as seguintes formas de obter L e U:

$$U = G_{n-1} \cdot \ldots \cdot G_1 \cdot A$$

$$L = G_1^{-1} \cdot G_2^{-1} \cdot \ldots \cdot G_{n-1}^{-1}$$

Uma vez calculada a fatoração, temos, para o sistema $Ax = b$, que $LUx =$

b. Fazendo $Ux = y$, resolvemos o sistema $Ly = b$ facilmente, assim como o sistema $Ux = y$ logo depois, descobrindo x, a solução desejada.

Muitas vezes, ao se escolher o pivô em cada U_i, notamos que ele pode ser menor que outros elementos da coluna i, e, caso a diferença de magnitude seja considerável, os erros numéricos podem crescer exageradamente. Para evitar esse problema, faz-se com que o pivô seja sempre o maior possível na coluna abaixo da linha i, trocando-se a linha i de U_i com a linha contendo esse máximo valor da coluna. Este procedimento recebe o nome de **pivotamento parcial**.

1.1.1 Implementação - LU

Implementaram-se dois algoritmos para a fatoração LU e solução do sistema $Ax = b$, quando possível e determinado. O primeiro, visto em sala de aula, é baseado diretamente na definição do método por multiplicações matriciais diretas, já fornecendo o $L^{-1}b$ utilizado para resolver o sistema.

Algorithm 1.1: Fatoração LU, seguindo as definições de L e U.

Input: Matriz $A_{n\times n}$ e vetor b
Output: Matrizes U e L^{-1}, e vetor $L^{-1}b$
begin
  $U \leftarrow A$
  $L^{-1} \leftarrow I_{n\times n}$
  for i = *1* **to** n - *1* **do**
    $G_i \leftarrow I_{n\times n}$
    for j=i+*1* **to** n **do**
      $g_{ji} \leftarrow -u_{ji}/u_{ii}$
    end
    $L^{-1} \leftarrow G_i \times L^{-1}$
    $U \leftarrow G_i \times U$
    $b \leftarrow G_i \times b$
  end
  for $i = n$ **to** *1* **do**
    $x_i \leftarrow b_i$
    for j = i+*1* **to** n **do**
      $x_i \leftarrow x_i - u_{ij}x_j$
    end
    $x_i \leftarrow x_i/u_{ii}$
  end
end

O segundo segue o algoritmo disponível em [1], que produz L e U, além de uma matriz de permutação P derivada da pivotação. Este apenas fornece a fatoração, bastando, em seguida, resolver os sistemas para encontrar y e depois x.

Algorithm 1.2: Fatoração LU, seguindo o algoritmo no livro "Numerical Linear Algebra", de Trefethen.

Input: Matriz $A_{n\times n}$
Output: Matrizes U, L, P
begin
  $U \leftarrow A$
  $L \leftarrow I_{n\times n}$
  $P \leftarrow I_{n\times n}$
  for $k = 1$ **to** $n\text{-}1$ **do**
    $i \leftarrow argmax_{k\leq m\leq n}|u_{mk}|$
    $u_{k,k:n} \leftrightarrow u_{i,k:n}$
    $l_{k,1:k-1} \leftrightarrow l_{i,1:k-1}$
    $p_{k,:} \leftrightarrow p_{i,:}$
    for $j = k + 1$ **to** n **do**
      $l_{jk} \leftarrow u_{jk}/u_{kk}$
      $u_{j,k:n} \leftarrow u_{j,k:n} - l_{jk}u_{k,k:n}$
    end
  end
end

1.2 Fatoração de Cholesky

Quando a matriz A em $Ax = b$ é simétrica, ou seja, $A = A^T$, e é positiva definida, ou seja, apresenta todos os autovalores positivos não nulos, a fatoração Cholesky pode ser executada, produzindo uma matriz R tal que $A = RR^T$. Aproveitando-se a fatoração LU, descrita na seção anterior, é possível mostrar que

$$R = LD^{\frac{1}{2}},$$

onde D é diagonal de U, e $D^{\frac{1}{2}}$ é a matriz obtida de D aplicando-se a raiz quadrada aos elementos da diagonal.

Uma vez obtida R, observamos, do sistema $Ax = b$, que $RR^Tx = b$, e, fazendo $y = R^Tx$, podemos facilmente resolver $Ry = b$, e, em seguida, $R^Tx = y$, obtendo x, a solução do sistema.

1.2.1 Implementação - Cholesky

Assim como para a fatoração LU, implementaram-se dois algoritmos: um visto em sala, baseado essencialmente na fatoração LU, e outro presente em [2]. O primeiro computa L^{-1}, U e o vetor $L^{-1}b$ e utiliza uma relação derivada da forma de R para resolver diretamente o sistema linear.

Algorithm 1.3: Método de Cholesky, implementado a partir das definições.

Input: Matriz $A_{n\times n}$ simétrica positiva definida
Output: Matrizes U e L^{-1}, vetor $L^{-1}b$ e x
begin
  $U \leftarrow A$
  $L^{-1} \leftarrow I_{n\times n}$
  for i = *1* **to** n - *1* **do**
    $G_i \leftarrow I_{n\times n}$
    for j=i+*1* **to** n **do**
      $g_{ji} \leftarrow -u_{ji}/u_{ii}$
    end
    $L^{-1} \leftarrow G_i \times L^{-1}$
    $U \leftarrow G_i \times U$
    $b \leftarrow G_i \times b$
  end
  $D \leftarrow diagonal(U)$
  $x \leftarrow L^{-T}D^{-1}b$
end

O segundo algoritmo realiza a multiplicação das matrizes indiretamente, computando a matriz $R = L$ tal que $A = LL^T$. Uma vez fornecida, basta resolver os sistemas para y e x, como descrito na seção anterior.

Algorithm 1.4: Método de Cholesky, implementado a partir do algoritmo no livro "Numerical Analysis", de Burden.

Input: Matriz $A_{n\times n}$, simétrica positiva definida
Output: Matriz L, tal que $A = LL^T$
begin
  $l_{11} \leftarrow \sqrt{a_{11}}$
  for j = *2* **to** *n-1* **do**
    $l_{j1} \leftarrow a_{j1}/l_{11}$
  end
  for i = *2* **to** *n-1* **do**
    $l_{ii} \leftarrow \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}$
    for $j = i + 1$ **to** n **do**
      $l_{ji} \leftarrow \frac{a_{ji} - \sum_{k=1}^{i-1} l_{jk} l_{ik}}{l_{ii}}$
    end
  end
  $l_{nn} \leftarrow \sqrt{a_{nn} - \sum_{k=1}^{n-1} l_{nk}^2}$
end

2 Métodos Indiretos

Os métodos indiretos são capazes de aproximar a solução do sistema linear de tal forma que o número de suas operações, comparado com os métodos diretos, é menor, principalmente se a matriz referente ao sistema for esparsa, ou seja, possuir muitos elementos nulos. Como já sabemos, o sistema é da forma:

$$Ax = b$$

Se tomarmos o $A = B + C$, podemos rescrever assim:

$$\begin{aligned} Ax &= b \\ (B + C)x &= b \\ x &= B^{-1}b - B^{-1}Cx \end{aligned}$$

ou seja, iterativamente teremos:

$$x^{k+1} = B^{-1}b - B^{-1}Nx^k \tag{2}$$

Para que possamos de fato melhorar no processo de solução do sistema, precisamos que B seja mais simples que A.

De forma que fique mais simples para observar essa iteração nos métodos que serão descritos a seguir, iremos desmembrar A em três matrizes: matriz diagonal (D), matriz triangular inferior(I) e triangular superior(S). Ou seja, $A = D + I + S$.

2.1 Método de Jacobi

Para o método de Jacobi, usamos o processo iterativo descrito na equação 2, sendo o $B = D$ e $C = I + S$, de tal forma que:

$$x^{k+1} = B^{-1}b - B^{-1}Nx^k$$
$$x^{k+1} = D^{-1}b - D^{-1}(I + S)x^k$$

Ou seja, ele usa o operador T, visto em sala de aula, que (dado que obedece à restrição abaixo indicada) quando aplicado a um vetor x^e, produz outro, x^s, mais próximo da solução x, a qual é um ponto fixo de T. Tal operador é definido por:

$$Tx = -D^{-1}(A - D)x + D^{-1}b,$$

onde D é a matriz diagonal de A. Isso, entretanto, só é garantido quando

$$||D^{-1}(A - D)|| < 1,$$

onde $||A|| = \max\limits_{1\leq i\leq n} \sum\limits_{j=1}^{n} |A_{ij}|$.

Dessa forma, para um dado vetor inicial x^e e um $\varepsilon > 0$, o método consiste em aplicar T a x^e, obtendo $x^s = Tx^e$, checar se $||x^s - x^e|| < \varepsilon$ e, caso contrário, fazer $x^e = x^s$, e retomar os passos anteriores. Dessa maneira, obter-se-á um um vetor aproximado tanto quanto se queira do resultado do sistema linear dado.

2.1.1 Implementação - Jacobi

A implementação baseia-se no algoritmo [1] visto em sala de aula:

Algorithm 2.1: Método de Jacobi

Input: Matriz $A_{n\times n}$, vetor b, vetor x^e, ε
Output: Aproximação para x
checkConvergence();
begin
  repeat
    $x^{aux} \leftarrow x^e$
    $x^s \leftarrow Tx^e$
    $x^e \leftarrow x^s$
  until $||x^e - x^{aux}|| < \varepsilon$;
  return x^s
end

[1] Disponível em `http://migre.me/wnK4n` , acesso em 1 abril 2017

2.2 Método de Gauss-Seidel

O método de Gauss-Seidel baseia-se no mecanismo do de Jacobi, utilizando o mesmo operador T sob as mesmas restrições. A diferença está no modo como computa os vetores a cada iteração: ao invés de utilizar o vetor de entrada, x^e, original, utiliza as componentes já calculadas na mesma iteração. Dessa forma, pode-se reduzir o número de iterações do algoritmo até chegar a aproximação desejada.

Para o método de Gauss-Seidel, usamos o processo iterativo descrito na equação 2, sendo o $B = I + D$ e $C = S$, de tal forma que:

$$x^{k+1} = B^{-1}b - B^{-1}Nx^k$$
$$x^{k+1} = D^{-1}Ib - D^{-1}(S)x^k$$

Ou, de forma simples, podemos dizer que:

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

Há três formas (**suficientes**) de verificar de o método de Gauss-Seidel irá convergir:

- Se a matriz dos coeficiente for estritamente diagonal
- Se o critério de linhas for satisfeito
- Se o critério de Sassenfeld for satisfeito

Na implementação, usamos apenas a verificação do critério de linhas, ou seja, "o valor absoluto do termo diagonal na linha i é maior do que a soma dos valores absolutos de todos os outros termos na mesma linha":

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| < |a_{ii}|$$

É importante lembrar que se o critério de linhas for satisfeito, o sistema vai convergir, mas a inversa não é correta. Vimos também uma limitação em

relação em apenas essa verificação, pois há casos em que os vértices livres possam produzir equações em que o elemento da diagonal é igual a soma dos demais elementos da linha.

2.2.1 Implementação - Gauss-Seidel

A implementação baseia-se no algoritmo[2] visto em sala de aula:

Algorithm 2.2: Método de Gauss-Seidel

Input: Matriz $A_{n \times n}$, vetor b, vetor x^e, ε
Output: Aproximação para x
begin
  checkConvergence(A);
  repeat
    $x^{aux} \leftarrow x^e$
    for $i = 1$ **to** n **do**
      $x_i^s \leftarrow -\frac{\sum_{j=1, j \neq i}^{i} a_{ij} x_j^s + \sum_{j=i+1}^{n} a_{ij} x_j^e - b_i}{a_{ii}}$
    end
    $x^e \leftarrow x^s$
  until $||x^e - x^{aux}|| < \varepsilon$;
  return x^s
end

[2] Disponível em `http://migre.me/wnK4L` , acesso em 1 abril 2017

3 Testes e Resultados

3.1 Métodos diretos

Utilizaram-se matrizes de diversas dimensões para testar os métodos diretos. Em todas, o sistema linear foi tal que b correspondia à soma das colunas da matriz, fazendo com que se esperasse sempre a solução $x_i = 1, \forall i$. Chamamos de algoritmos "didáticos" aqueles vistos em sala, por se basearem diretamente na definição. Para o Cholesky "didático", algumas instâncias geraram erro, provavelmente por problemas de precisão durante as várias multiplicações de matrizes.

A pivotação parcial ficou clara a partir da matriz de permutação gerada e das matrizes fatoradas, as quais diferiram daquelas geradas pelos métodos quando retirada a pivotação.

A Tabela 1 resume os resultados, mostrando o tempo em segundos para cada execução:

Dimensão	LU	Cholesky	Didático LU	Didático Cholesky
5x5	0.015286	0.012809	0.050517	0.040998
10x10	0.038966	0.026361	0.464253	0.869828
20x20	0.134158	0.05492	16.0118	6.89225
30x30	0.360649	0.128106	32.9281	-
40x40	0.747341	0.251166	102.131	103.648
50x50	1.33657	0.432224	249.097	251.084
60x60	2.20472	0.690858	514.509	517.351
70x70	3.3237	1.023	946.277	-
80x80	4.82391	1.72569	1616.41	1696.81
90x90	6.73681	2.03473	2620.9	2599
100x100	9.25865	2.85575	3970.46	3950.56
150x150	29.2534	8.17796	20035.2	-
200x200	66.4677	18.548	69281.5	67986.9
250x250	127.024	34.8935	164388	-
300x300	217.056	59.2172	343029	369484
350x350	342.593	92.778	636412	630656
400x400	511.915	137.945	+15min	+15min
450x450	721.025	192.795	+15min	+15min

Tabela 1: Tempo de execução em milissegundos

Como se esperava, os resultados para o algoritmo de Cholesky superaram aqueles do método LU com pivotação parcial, mais evidentemente para matrizes de dimensão maior que 30. Isso, porém, apenas ocorreu para os métodos advindos dos livros, visto que o método de Cholesky visto em sala se aproveita do algoritmo da fatoração LU.

3.2 Métodos iterativos

Para os métodos iterativos, testaram-se cinco sistemas lineares, para $\varepsilon = 0.1$, utilizando diferentes vetores iniciais, aplicando tanto o método de Jacobi, quanto o de Gauss-Seidel. Os resultados foram expressos em tempo

de execução (segundos) e número de iterações, na Tabela 3.2 a seguir:

Matriz	Vetor inicial	Jacobi	Iterações	Gauss-Seidel	Iterações
1 (3x3)	0	0.070105	4	0.011437	3
2 (3x3)	0	0.037312	4	0.009241	3
3 (4x4)	0	0.08077	6	0.019188	5
4 (2x2)	0	0.029007	5	0.01726	3
5 (2x2)	0	0.041	4	0.012726	3
1 (3x3)	5	0.030223	5	0.005229	4
2 (3x3)	5	0.018607	6	0.003728	3
3 (4x4)	5	0.044471	7	0.007003	5
4 (2x2)	5	0.013156	6	0.003506	4
5 (2x2)	5	0.011672	6	0.003193	4
1 (3x3)	10	0.091356	6	0.017651	4
2 (3x3)	10	0.063236	7	0.014118	4
3 (4x4)	10	0.123886	9	0.027253	6
4 (2x2)	10	0.044007	7	0.012765	5
5 (2x2)	10	0.039837	7	0.011702	5
1 (3x3)	50	0.101314	7	0.019505	5
2 (3x3)	50	0.072548	9	0.016688	5
3 (4x4)	50	0.138285	11	0.034124	8
4 (2x2)	50	0.052632	9	0.012408	5
5 (2x2)	50	0.04776	9	0.013516	6
1 (3x3)	100	0.101298	8	0.022492	6
2 (3x3)	100	0.115226	9	0.017085	5
3 (4x4)	100	0.175004	13	0.034043	8
4 (2x2)	100	0.087302	10	0.014584	6
5 (2x2)	100	0.075164	10	0.013837	6
1 (3x3)	1000	0.116178	10	0.022372	6
2 (3x3)	1000	0.090088	12	0.020001	6
3 (4x4)	1000	0.177455	16	0.041847	10
4 (2x2)	1000	0.064203	12	0.016232	7
5 (2x2)	1000	0.059864	12	0.015376	7
1 (3x3)	100000	0.068154	14	0.013407	9
2 (3x3)	100000	0.052454	17	0.010834	8
3 (4x4)	100000	0.109593	25	0.026044	15
4 (2x2)	100000	0.0362	17	0.009444	10
5 (2x2)	100000	0.034481	17	0.009338	10
1 (3x3)	1000000	0.049761	16	0.010497	10
2 (3x3)	1000000	0.060622	19	0.00893	9
3 (4x4)	1000000	0.102968	30	0.021374	17
4 (2x2)	1000000	0.040368	20	0.007645	11
5 (2x2)	1000000	0.047839	20	0.006955	11

Tabela 2: Tempo de execução em milissegundos

Nota-se que o método de Gauss-Seidel apresentou melhores tempos de execução na maioria dos casos, chegando à solução com menos iterações. É, portanto, mais recomendado, por otimizar o processo de Jacobi.

Considerações finais

Neste trabalho, implementaram-se quatro algoritmos para a resolução de sistemas lineares: por fatoração LU, por fatoração Cholesky, Jacobi e Gauss-Seidel. O primeiro é interessante por não exigir características da matriz de coeficientes, fatorando-a sem restrições. Cholesky, por sua vez, exige uma matriz simétrica positiva definida, e é mais rápido para esses casos. Já o método de Jacobi produz vetores que se aproximam da solução desejada através de um operador, quando bem condicionado. Entretanto, produz resultados menos eficientes quando comparado ao de Gauss-Seidel, fundado nos princípios do de Jacobi, mas aproveitando atualizações de cada componente do vetor de saída, demandando menos iterações e esforços.

Os resultados obtidos com os testes confirmaram o que a teoria afirma sobre a eficiência dos métodos, a precisão numérica, e as condições para funcionarem. O contato com diferentes implementações possibilitou ao grupo perceber as vantagens e desvantagens de cada uma, constituindo um crescimento acadêmico satisfatório.

O código produzido será disponibilizado no GitHub, para que outras pessoas possam chegar às mesmas conclusões alcançadas e, até mesmo, possam contribuir com mais métodos e ideias. Isso significa que este trabalho produziu um repositório que permanecerá vivo e será mantido pelos autores e público externo, contribuindo para o crescimento continuado do grupo e dos demais interessados em métodos numéricos.

Referências Bibliográficas

[1] Trefethen, Lloyd N. and Bau, David, *Numerical Linear Algebra*, 1997

[2] R. L. Burden and J. D. Faires *Numerical Analysis*, 2010

[3] Sistemas Lineares Iterativos Jacobi Richardso 2008

[4] http://www.dca.ufrn.br/ nogueira/dca0304/material/

[5] https://www.math.tecnico.ulisboa.pt/ calves/cursos/SisLin-Iter.htm

[6] http://conteudo.icmc.usp.br/pessoas/andretta/ensino/aulas/sme0300-2-12/aula7-sassenfeld.pdf