MÉTODOS NUMÉRICOS PARA SOLUCIONAR SISTEMAS LINEARES

Evandro Pedro Alves de Mendonça[a], Marcelino José de Lima Andrade[a].

[a] *Núcleo de Tecnologia (NTI), Universidade Federal de Pernambuco (UFPE), Campus Acadêmico do Agreste (CAA), Rodovia BR-104, km 59, S/N, Nova Caruaru, CEP. 55.014-900, Caruaru-PE, Brasil, http://www.ufpe.br/caa*

Palavras Chave: sistemas lineares, matrizes, incógnitas, métodos numéricos, solução, métodos diretos, métodos iterativos.

Resumo. Na ciência e na engenharia, é muito comum ter que resolver sistemas lineares. Para isso, existem diversos métodos, que podem ser classificados como métodos diretos (que dão uma solução exata para o sistema) e métodos iterativos (que dão uma solução aproximada para o sistema). Há vantagens e desvantagens em cada um deles e elas serão abordadas no decorrer do trabalho. Quando os sistemas são muito grandes (como na grande maioria dos casos reais), é inviável resolvê-los manualmente. Portanto, é feito o uso de computadores para realizar esse trabalho através de algoritmos que implementam os métodos numéricos. Alguns algoritmos também serão desenvolvidos nesse trabalho e serão comparados a algoritmos já criados para o mesmo fim e são nativos do MATLAB®. Com esse trabalho, pretende-se discutir, comparar, exemplificar e analisar alguns métodos numéricos para solução de sistemas lineares.

1 INTRODUÇÃO

Em diversos problemas da ciência e da engenharia se faz necessário resolver um sistema ou conjunto de equações lineares. Problemas desse tipo podem ser resolvidos analiticamente de forma relativamente fácil, quando envolvem um pequeno número de equações. Porém, ao lidar com grandes sistemas, é necessária a utilização de métodos computacionais para a solução dessas equações. Para a resolução dos sistemas lineares, existem vários métodos que podem ser empregados. Uma boa escolha do método, levará à obtenção dos resultados de forma mais rápida e com um menor custo computacional.

Os métodos utilizados se dividem em dois grupos: iterativos e diretos. Nos métodos iterativos, dá-se um valor de estimativa inicial qualquer para cada variável que se deseja descobrir o valor e a cada iteração o valor das variáveis vai sendo modificado até se aproximar do valor real o bastante para que esteja dentro de um valor de tolerância pré-estabelecido. Dois métodos iterativos bastante conhecidos são o método de Jacobi e o método de Gauss-Seidel. Nos métodos diretos, são feitas operações elementares nas matrizes do sistema até que se obtenha um conjunto de equações que é resolvido facilmente. Dentre os métodos diretos, se destacam o método de Eliminação de Gauss, Gauss-Jordan e a decomposição LU.

2 EXERCÍCIOS PROPOSTOS

Segue, abaixo, a solução dos exercícios propostos sobre o tema.

2.1 1ª questão

O algoritmo do Anexo 1 foi usado para solucionar os sistemas lineares das letras A e B dessa questão. Mais detalhes podem ser vistos no Anexo 6. Os resultados estão listados a seguir.

2.1.1 x_1: resultado para sistema A; x_2: resultado para sistema B.

$$x_1 = \begin{bmatrix} 0{,}1768252974993456 \\ 0{,}01269269086768745 \\ -0{,}02065405013713113 \\ -1{,}182608695468154 \end{bmatrix} \quad x_2 = \begin{bmatrix} 0{,}09276104702948032 \\ -0{,}06299433961595419 \\ -0{,}03624582269162138 \\ 0{,}04670801937981649 \end{bmatrix}$$

2.2 2ª questão

O algoritmo do Anexo 2 foi usado para solucionar os sistemas lineares das letras A e B dessa questão. Mais detalhes podem ser vistos no Anexo 7. Os resultados estão listados no item 2.2.1.

Uma observação necessária é que, na questão 1, letra B, houve uma operação de pivoteamento parcial (ou pivotação). Isso acontece porque em um determinado momento da eliminação de Gauss um dos elementos da diagonal principal da matriz **A** é zerado, o que inviabiliza o método. Portanto, quando isso ocorre, é feita uma permutação entre as linhas da matriz **A** fazendo com que o elemento em questão deixe de ser nulo e o procedimento possa ser continuado. Obviamente, a mesma permutação é feita também no vetor resposta do sistema linear (vetor **b**). No caso da decomposição LU, também fazemos essa permutação. Para essa questão, a permutação foi feita antes de iniciar a decomposição e, para isso, foi definida uma matriz **P** de permutação de forma que o sistema linear tenha a seguinte forma:

$$PA\vec{x} = P\vec{b}$$

Onde:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ou seja, **P** é simplesmente a matriz identidade com a segunda e terceira linhas permutadas.

2.2.1 x_3: resultado para sistema A; x_4: resultado para sistema B.

$$\boldsymbol{x_3} = \begin{bmatrix} \mathbf{0,1768252974993456} \\ \mathbf{0,01269269086768745} \\ \mathbf{-0,02065405013713113} \\ \mathbf{-1,182608695468154} \end{bmatrix} \quad \boldsymbol{x_4} = \begin{bmatrix} \mathbf{0,09276104702948032} \\ \mathbf{-0,06299433961595419} \\ \mathbf{-0,03624582269162138} \\ \mathbf{0,04670801937981649} \end{bmatrix}$$

Podemos perceber que os resultados são idênticos aos resultados da questão 1, o que confirma a exatidão do processo.

```
>> respostas = [x1 x3 x2 x4]

respostas =

    1.768252974993456e-01     1.768252974993456e-01     9.276104702949123e-02     9.276104702949123e-02
    1.269269086768745e-02     1.269269086768745e-02    -6.299433961594411e-02    -6.299433961594411e-02
   -2.065405013713113e-02    -2.065405013713113e-02    -3.624582269162138e-02    -3.624582269162138e-02
   -1.182608695468154e+00    -1.182608695468154e+00     4.670801937981647e-02     4.670801937981647e-02
```

Figura 1: o vetor "respostas" contém as soluções dos sistemas A e B utilizando os métodos de Eliminação de Gauss (colunas 1 e 3, referentes a x1 e x2) e Decomposição LU (colunas 2 e 4, referentes a x3 e x4).

2.3 3ª questão

O algoritmo do Anexo 3 foi usado para solucionar o sistema linear dessa questão. Mais detalhes podem ser vistos no Anexo 8. O resultado é mostrado no item 2.3.1.

É importante salientar que, para que o método pudesse ser aplicado, é necessário fazer duas checagens a mais do que as que já são feitas nos métodos anteriores. A primeira delas, e mais simples, é se a matriz dos coeficientes é simétrica, isto é, a matriz é igual à sua transposta. A segunda, um pouco mais trabalhosa, é se a matriz dos coeficientes é definida positiva, ou seja, os determinantes de todos os seus menores principais e o da própria matriz é maior que zero. Sendo satisfeitas essas duas condições, o método da Decomposição de Cholesky é válido. Essas duas checagens são feitas automaticamente na própria função, na etapa de validação dos parâmetros, como pode ser visto no Anexo 3.

2.3.1 Resultado para o vetor de incógnitas

$$x = \begin{bmatrix} 0{,}1785714285714286 \\ 0{,}7142857142857143 \\ 0{,}25 \\ 0{,}3214285714285714 \end{bmatrix}$$

2.4 4ª questão

Essa questão usa algoritmos implementados que estão nos Anexos 4 e 5, e funções nativas do MATLAB®. O detalhamento dessa questão está no Anexo 9.

Primeiramente, os dois sistemas foram resolvidos por um método direto do MATLAB®, para que se tenha uma solução com a qual comparar os resultados obtidos. A solução é mostrada a seguir.

```
>> A1\b1

ans =

   1.447761194029851e+00
  -8.358208955223880e-01
  -4.477611940298504e-02

>> A2\b2

ans =

   9.000000000000001e-01
  -7.999999999999999e-01
   7.000000000000001e-01
```

Figura 2: solução dos dois sistemas pelo método da matriz inversa.

2.4.1 4.a)

Os resultados para esse item foram:

```
resposta_a =

   1.448215914351852e+00   1.447816349987140e+00   8.884175778371142e-01   8.996205292396300e-01
  -8.356592592592592e-01  -8.358173037765774e-01  -7.918180607515059e-01  -8.002846030702775e-01
  -4.508972222222218e-02  -4.479961848422502e-02   6.821278642529894e-01   7.002371692252313e-01
```

Figura 3: resultados do item 4.a). Da esquerda para a direita, temos: método de Jacobi no sistema A; método de Gauss-Seidel no sistema A; método de Jacobi no sistema B e; método de Gauss-Seidel no sistema B.

A matriz dos coeficientes do sistema A é diagonalmente dominante, isto é, os métodos de Jacobi e Gauss-Seidel convergirão para uma solução. Porém, a matriz dos coeficientes do sistema B não é, então existe a possibilidade de haver divergência nos métodos de Jacobi e/ou Gauss-Seidel para este sistema. Sabendo disso, vemos que o método de Jacobi realizou 9 iterações no sistema A e 147 no sistema B, enquanto que o método de Gauss-Seidel realizou 6 iterações no sistema A e 17 no sistema B. Fica claro, aqui, que o método de Gauss-Seidel tende a convergir mais rápido para a solução do que o método de Jacobi. Percebe-se também que, para a tolerância exigida (que foi grande), a solução se aproximou razoavelmente bem da solução obtida pelo método direto, sendo que o método de Gauss-Seidel se aproximou mais. Com uma tolerância menor, seria possível atingir mais casas decimais de exatidão.

2.4.2 4.b)

Os resultados para esse item foram:

```
resposta_b =

   1.440694444444445e+00   1.447803931970165e+00   8.916910565061291e-01   8.997397914786034e-01
  -8.388888888888887e-01  -8.357661447616599e-01  -7.961442289268697e-01  -8.001951563910474e-01
  -5.355555555555562e-02  -4.481511488340186e-02   6.909849127123907e-01   7.001626303258729e-01
```

Figura 4: resultados do item 4.b). Da esquerda para a direita, temos: método de Jacobi no sistema A; método de Gauss-Seidel no sistema A; método de Jacobi no sistema B e; método de Gauss-Seidel no sistema B.

Realizando apenas uma alteração do vetor de estimativa inicial, obtivemos resultados um pouco diferentes, mas ainda assim próximos do valor obtido pelo método direto. O que chama atenção nesse caso é que o método de Jacobi realizou apenas 5 iterações para atingir a tolerância no sistema A, enquanto que o método de Gauss-Seidel manteve as 6 iterações. Isso mostra que nem sempre o método de Gauss-Seidel converge mais rapidamente que o de Jacobi. Já no sistema B, temos 117 iterações do método de Jacobi contra apenas 16 do método de Gauss-Seidel. Temos, então, menos iterações realizadas em cada um dos métodos apenas com uma pequena mudança no vetor de estimativa inicial. Isso nos mostra que, escolhendo bem os parâmetros, podemos reduzir bastante o custo computacional para solucionar um sistema linear por métodos iterativos.

2.4.3 4.c)

Os resultados para esse item foram:

```
resposta_c =

   1.447761194029851e+00     9.000000000000001e-01     1.447761194029851e+00     9.000000000000004e-01
  -8.358208955223880e-01    -7.999999999999999e-01    -8.358208955223880e-01    -7.999999999999999e-01
  -4.477611940298504e-02     7.000000000000001e-01    -4.477611940298516e-02     7.000000000000002e-01
```

Figura 5: resultados do item 4.c). Da esquerda para a direita, temos: função *linsolve* no sistema A e no sistema B; função *bicg* no sistema A e no sistema B.

A função *linsolve* foi escolhida pois ela implementa a decomposição LU, que é um método abordado e implementado neste trabalho, portanto, sabemos exatamente como ele funciona. A função *bicg* foi escolhida porque, apesar de implementar um método que não é abordado ou implementado neste trabalho, os parâmetros fornecidos a esta função são parecidos com os parâmetros fornecidos às funções *jacobi* e *gauss_seidel*; além disso, esta é uma das funções iterativas para solução de sistemas lineares que não exige que a matriz seja simétrica e/ou definida positiva (como o método de Cholesky ou do Gradiente Conjugado Precondicionado).

Sendo assim, observamos o seguinte: a função *linsolve*, que implementa um método direto, obteve os mesmos resultados que o método da matriz inversa, o que já era esperado.

Na Figura 5, podemos observar que o método iterativo da função *bicg* convergiu com bastante exatidão para a solução. Só veio gerar diferença nas últimas duas casas decimais (porém, em dois casos, a resposta foi exatamente igual). Isso mostra que esta função/método é muito mais eficiente que as funções implementadas com os métodos de Jacobi e Gauss-Seidel, pois conseguiu um resultado muito mais acurado, com bem menos iterações (3 para cada sistema) e utilizando o mesmo valor de tolerância.

2.5 5ª questão

Essa questão usa algoritmos implementados que estão nos Anexos 1, 4 e 5. O detalhamento dessa questão está no Anexo 10 junto com os parâmetros para a resolução no MATLAB®.

2.5.1 5.a) Os resultados para esse item foram:

```
3.003000000000589e+03
3.002000000000589e+03
3.000000000000589e+03
3.000000000000589e+03
2.999000000000589e+03
```

Figura 6: Resultado utilizando a Eliminação de Gauss.

Mais informações sobre a matriz aumentada do sistema linear estão no anexo 10. Não foi realizada nenhuma operação de pivoteamento no método.

2.5.2 5.b) Os resultados para esse item foram:

```
5.b) Usando o Método de Jacobi:
A matriz dos coeficientes não é diagonalmente dominante,
ou seja, existe a possibilidade do método não convergir.
Função finalizada por atingir o número máximo de iterações.

Número total de iterações: 300

x2 =

     1.121721778222176e+02
     1.114129196261346e+02
     1.106168693875239e+02
     1.094129196261346e+02
     1.081721778222176e+02
```

Figura 7: Resultado utilizando o método de Jacobi.

A matriz dos coeficientes não é diagonalmente dominante, isto é, existe a possibilidade de haver divergência nos métodos de Jacobi e/ou Gauss-Seidel. Sabendo disso, vemos que o método de Jacobi realizou 300 iterações finalizou a execução pelo critério de parada de número máximo de iterações. O vetor de estimativa inicial utilizado foi o vetor nulo de 5 elementos. Além disso, percebe-se que os resultados para o sistema têm valores bastante diferentes com relação aos obtidos pelo método direto do item anterior. Caso aumentássemos o número máximo de iterações, o método tenderia à solução exata. Então conclui-se que o método converge, porém com taxa de convergência muito pequena.

2.5.3 5.c) Os resultados para esse item foram:

```
5.c) Usando o Método de Gauss-Seidel:
A matriz dos coeficientes não é diagonalmente dominante,
ou seja, existe a possibilidade do método não convergir.
Função finalizada por atingir o número máximo de iterações.

Número total de iterações: 300

x3 =

     2.180678112409260e+02
     2.177636963455634e+02
     2.171551187861704e+02
     2.164594075658669e+02
     2.154594075658669e+02
```

Figura 8: Resultado utilizando o método de Gauss-Seidel diante da exposição dos parâmetros.

Neste item, a solução obtida se aproximou muito mais da solução exata em comparação com o item anterior. Isso se deve ao fato de que o método de Gauss-Seidel tende a ter uma maior taxa de convergência do que o método de Jacobi. Para os mesmos parâmetros, obtivemos uma solução aproximada com erro menor (grande, mas ainda assim muito menor que do método de Jacobi).

2.6 6ª questão

Para essa questão, foi utilizada a operação de divisão à esquerda do MATLAB® (A\b) para solucionar os sistemas lineares. Detalhes no Anexo 11.

2.6.1 6.a)

A solução para os dois sistemas e o número de condicionamento das matrizes dos coeficientes de cada sistema estão na figura a seguir.

```
Resolvendo Sistema A por matriz inversa:

x1 =

    1.0000
    1.0000

Resolvendo Sistema B por matriz inversa:

x2 =

   -1.0000
    2.0000


respostas_6 =

    1.0000   -1.0000
    1.0000    2.0000

Condicionamento das matrizes A1 (c1) e A2 (c2):

c1 =

   5.0001e+04


c2 =

   4.9999e+04
```

Figura 9: solução para os sistemas lineares A e B e número de condicionamento das matrizes dos coeficientes de cada sistema.

Embora os sistemas lineares A e B sejam parecidos, suas soluções são completamente diferentes. Isso acontece quando a matriz dos coeficientes de um sistema linear é mal condicionada, o que implica que o sistema é muito sensível a erros de dados. Caso aconteça alguma mudança no sistema, por pequena que seja, os resultados obtidos serão muito diferentes do sistema original. O que nos fornece a informação do condicionamento das matrizes é o comando *cond*(Matriz) do MATLAB®. Um número de condicionamento próximo de 1 indica um bom condicionamento. Quanto maior for o número, pior é o condicionamento. Como vemos na figura acima, ambos os sistemas têm número de condicionamento na ordem de 10^4, o que indica que as matrizes estão muito mal condicionadas.

2.6.1 6.b)

Num sistema linear em que se deseja calcular a solução numericamente, como não conhecemos o valor real da solução, não podemos obter um valor de erro absoluto (ou relativo) diretamente como em $erro = x - \bar{x}$. Portanto, uma forma de identificar se a solução numérica é satisfatória é através do cálculo do resíduo, definido por:

$$\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\bar{x}$$

A interpretação dessa equação é que, quando a solução numérica é substituída no sistema linear e ela está próxima do valor real, o resíduo tende a zero. Porém, o resíduo não indica quão grande é o erro, apenas quão bem a solução numérica satisfaz o sistema linear, isto é, o quanto o vetor resposta gerado se aproxima do vetor resposta original. É possível que haja um sistema linear com resíduo pequeno e erro grande, isso depende da ordem de grandeza da matriz dos coeficientes.

```
>> residuo1 = b - A1*x1

residuo1 =

     0
     0

>> residuo2 = b - A2*x2

residuo2 =

   1.0e-15 *

         0
   -0.4441
```

Figura 10: cálculo de resíduos para sistemas lineares A e B.

Na figura acima, observamos que o resíduo do sistema A é nulo e o do sistema B é muito próximo de zero (na ordem de 10^{-15}). Dessa forma, vemos que as soluções encontradas são bastante satisfatórias para os sistemas lineares fornecidos.

Portanto, utilizando o cálculo do resíduo, conseguimos identificar se a solução encontrada é boa o suficiente, mas não necessariamente conseguimos melhorar a solução encontrada através do resíduo. Ele é só um parâmetro que serve como um guia para ações posteriores.

2.7 7ª questão

Nesta questão, foram usados, para o item 7.a), como método direto, a Eliminação de Gauss (Anexo 1), por encontrar a solução em menos iterações comparando com a decomposição LU e a decomposição de Cholesky (que não pode ser utilizado para esse sistema, pois a matriz dos coeficientes não é simétrica e definida positiva) e, além disso, já realiza o pivoteamento parcial automaticamente, caso necessário; e, como método aberto, o método de Gauss-Seidel (Anexo 5), por possuir maior taxa de convergência em comparação ao método de Jacobi. Para o item 7.b), como método direto, foi utilizado o comando *linsolve*, que implementa a Decomposição LU, pois foi utilizado na questão 4 e se mostrou um comando simples e eficaz; como método aberto, foi utilizado, também como na questão 4, o comando *bicg*, pelo alto grau de exatidão atingido com o método do Gradiente Biconjugado.

As respostas para todos os métodos estão listadas na figura a seguir.

```
Vetor com todas as respostas em ordem de execução:

respostas =

   8.749970902461146e+01                         NaN     8.749970902461172e+01     8.749970902459228e+01
  -5.575380417303086e+01                         NaN    -5.575380417303131e+01    -5.575380417312918e+01
   1.777411634849739e+01                         NaN     1.777411634849743e+01     1.777411634862331e+01
   3.204744365636587e+01                         NaN     3.204744365636594e+01     3.204744365641339e+01
   1.174397126243134e+01                         NaN     1.174397126243135e+01     1.174397126239938e+01
  -9.436514182219534e+01                         NaN    -9.436514182219554e+01    -9.436514182222106e+01
   9.184895568160265e+00                         NaN     9.184895568160274e+00     9.184895568178058e+00
  -8.106280536611729e+00                         NaN    -8.106280536611651e+00    -8.106280536570683e+00
   9.997509067177154e+01                         NaN     9.997509067177180e+01     9.997509067175038e+01
   5.253605363780216e+01                         NaN     5.253605363780227e+01     5.253605363780079e+01
```

Figura 11: soluções do sistema linear da 7ª questão encontradas pelos métodos: Eliminação de Gauss, Gauss-Seidel, Decomposição LU (*linsolve*) e Gradiente Biconjugado (*bicg*), respectivamente.

Vemos, portanto, que os métodos diretos nos dão soluções muito próximas uma da outra e que o método do Gradiente Biconjugado, como esperado, nos dá uma solução muito aproximada da solução dos métodos diretos. Já o método de Gauss-Seidel diverge. Isto se dá ao fato de que a matriz não é diagonalmente dominante e que o número de condicionamento da matriz ($cond(A) \approx 410{,}86$) é muito alto, isto é, a solução do sistema linear é muito sensível a erros de dados (qualquer pequena diferença nos dados, causa uma grande distorção nos resultados). Além disso, os zeros na diagonal principal fazem com que o método de Gauss-Seidel realizem divisões por zero, o que faz com que o método divirja já na primeira iteração.

Mais detalhes sobre essa questão podem ser vistos no Anexo 12.

3 CONCLUSÃO

Após a implementantação dos códigos e observação dos resultados obtidos, conclui-se que há uma variedade de métodos numéricos disponíveis para calcular a solução numérica de sistemas de equações lineares, em que pode se chegar ao mesmo resultado. Contudo, haverá sempre um método melhor que outro e situações variadas, pois isso depende das convergências de cada método e o condicionamento das matrizes para a escolha. É importante estar ciente de questões como tempo de processamento, número de iterações, convergência e precisão. Todos esses parâmetros podem ser ajustados conforme especificações dos projetos com os quais se está trabalhando para solucionar os problemas.

REFERÊNCIAS

Chapra, S. C., e Canale, R. P. *Métodos Numéricos para Engenharia*. 5ª edição. Porto Alegre: AMGH, 2011.

Gilat, A., e Subramaniam, V. *Métodos Numéricos para Engenheiros e Cientistas: uma introdução com aplicações usando o MATLAB*. Porto Alegre: Bookman, 2008.

ANEXO 1

```
function [ x ] = eliminacao_gauss(A,b)
%Função que calcula a solução de um sistema linear usando o método da
%Eliminação de Gauss.
%Parâmetros: eliminacao_gauss(A,b)
%A = matriz dos coeficientes (deve ser quadrada e det(A)~=0)
%b = vetor resposta ou vetor das constantes do sistema linear (deve ter o
%mesmo número de elementos que a dimensão da matriz A)

%validação dos parâmetros
if size(A,1)~=size(A,2) %testa se o número de linhas é igual ao de colunas.
    disp('Erro. A matriz dos coeficientes não é quadrada.');
    return;
elseif det(A)==0 %testa se o determinante de A é igual a 0.
    disp('Erro. O determinante da matriz dos coeficientes é nulo.');
    return;
end
dim = size(A,1);%determina a dimensão da matriz A e armazena em dim.
if size(b,1)~=1 && size(b,2)~=1%testa se é vetor linha ou coluna
    disp('Erro. O vetor resposta não é um vetor linha ou vetor coluna.');
    return;
elseif size(b,1)~=dim && size(b,2)~=dim %testa equivalência entre A e b.
    fprintf('Erro. O vetor resposta deve ter o mesmo número de elementos');
    fprintf(' que a dimensão da matriz dos coeficientes.\n');
    return;
end
if size(b,1)==1 %se b for um vetor linha,
    b = b';     %é transformado em vetor coluna.
end
%fim da validação dos parâmetros.
%#ok<*NASGU>
m = 0; %constante usada para transformação da matriz em Triangular Superior
temp = [];%vetor temporário auxiliar
temp1 = 0;%variável temporária auxiliar
cont = 0;%contador de pivotações

format short;
disp('Matriz aumentada do sistema linear inicial:');
disp([A b]);
```

Figura 12: algoritmo do Método da Eliminação de Gauss implementado em MATLAB® (parte 1).

```matlab
%Transformando matriz dos coeficientes em uma matriz triangular superior
%através de operações elementares em suas linhas
for i = 1:(dim-1) %for para linhas

    %O if a seguir evita que exista um pivô nulo
    if(A(i,i)==0)
        for k = (i+1):dim %esse for realiza a pivotação
            if(A(k,i)~=0)
                cont = cont + 1; %incrementa contador de pivotações
                %permutando linha da matriz dos coeficientes
                temp = A(i,:);
                A(i,:) = A(k,:);
                A(k,:) = temp;

                %permutando linha do vetor resposta
                temp1 = b(i);
                b(i) = b(k);
                b(k) = temp1;
                break;
            end
        end
    end

    for j = (i+1):dim %for para colunas
        m = A(j,i)/A(i,i);%const. usada p/ zerar elementos abaixo do pivô
        A(j,:) = A(j,:) - m*A(i,:);%transformação em Triangular Superior
        b(j,1) = b(j,1) - m*b(i,1);%Alteração no vetor das constantes
    end
end
disp('Matriz aumentada do sistema linear após primeira etapa:');
disp([A b]);
fprintf('Foram realizadas %d operações de pivotação.\n\n',cont);
clear temp temp1 m k; %apaga variáveis auxiliares
x = [];%declaração da variável de saída como um vetor vazio.

%Resolução do sistema linear por retrossubstituição
for i = dim:-1:1 %for para linhas
    soma = b(i);%constante que será somada com os valores já conhecidos
    for j = dim:-1:1 %for para colunas
        if(j~=i)
            soma = soma-(A(i,j)*x(j));%valores já descobertos sendo somados
        end
        if(j==i)
            x(i,1) = soma / A(i,j); %cálculo do valor da incógnita
        end
    end
end
disp('Solução para o vetor de incógnitas:');
format longe;
clear soma i j; %apaga variáveis auxiliares
end
```

Figura 13: algoritmo do Método da Eliminação de Gauss implementado em MATLAB® (parte 2).

ANEXO 2

```
function [ x ] = decomposicaolu(A,b)
%Função que calcula a solução de um sistema linear usando o método da
%Decomposição LU.
%Obs.: Esse algoritmo não faz pivoteamento parcial. Caso seja necessário,
%as permutações na matriz dos coeficientes devem ser feitas antes de usar
%essa função.
%Parâmetros: x = decomposicaolu(A,b)
%A = matriz dos coeficientes (deve ser quadrada e det(A)~=0)
%b = vetor resposta ou vetor das constantes do sistema linear (deve ter o
%mesmo número de elementos que a dimensão da matriz A)
%x = vetor de saída.

%validação dos parâmetros
if size(A,1)~=size(A,2) %testa se o número de linhas é igual ao de colunas.
    disp('Erro. A matriz dos coeficientes não é quadrada.');
    return;
elseif det(A)==0 %testa se o determinante de A é igual a 0.
    disp('Erro. O determinante da matriz dos coeficientes é nulo.');
    return;
end
dim = size(A,1);%determina a dimensão da matriz A e armazena em dim.
if size(b,1)~=1 && size(b,2)~=1%testa se é vetor linha ou coluna
    disp('Erro. O vetor resposta não é um vetor linha ou vetor coluna.');
    return;
elseif size(b,1)~=dim && size(b,2)~=dim %testa equivalência entre A e b.
    fprintf('Erro. O vetor resposta deve ter o mesmo número de elementos');
    fprintf(' que a dimensão da matriz dos coeficientes.\n');
    return;
end
if size(b,1)==1 %se b for um vetor linha,
    b = b';     %é transformado em vetor coluna.
end
%fim da validação dos parâmetros.

%transformação de A para LU
%#ok<*NASGU>
m = 0; %constante usada para transformação da matriz em Triangular Superior
temp = [];%vetor temporário auxiliar
temp1 = 0;%variável temporária auxiliar
U = A; %matriz triangular superior
L = zeros(dim); %matriz triangular inferior
    for i=1:dim %preenchendo a diagonal principal de L com 1
        L(i,i)=1;
    end
    format short;
    %Transformando a matriz U em uma matriz triangular superior
    %através de operações elementares em suas linhas
```

Figura 14: algoritmo do Método da Decomposição LU implementado em MATLAB® (parte 1).

```matlab
    for i = 1:(dim-1)
        %O if a seguir evita que exista um pivô nulo
        if(U(i,i)==0)
            disp('Erro. É necessário permutar as linhas da matriz dos');
            disp('coeficientes para prosseguir com o método. Esse');
            disp('algoritmo não realiza pivoteamento parcial.');
            x = 0;
            return;
        end
        for j = (i+1):dim
            m = U(j,i)/U(i,i);%const. usada p/ zerar elem. abaixo do pivô
            U(j,:) = U(j,:) - m*U(i,:);%transf. em Triangular Superior
            L(j,i) = m; %adiciona multiplicador à matriz L na posição (j,i)
        end
    end
    disp('A matriz A foi decomposta em LxU. Onde');
    disp('L = '); disp(L);
    disp('e U = '); disp(U);
    disp('Resolvendo inicialmente Ly = b, por substituições sucessivas,');
    disp('temos y =');
    format longe;
    for i=1:dim %preenchendo vetores y e x com zeros
        %#ok<*AGROW>
        y(i,1)=0;
        x(i,1)=0;
    end
    for i = 1:dim %for para linhas
        soma = b(i,1);%const. que será somada com os valores já conhecidos
        for j = 1:dim %for para colunas
            if(j~=i)
                soma = soma-(L(i,j)*y(j,1));
            end
            if(j==i)
                y(i,1) = soma / L(i,j); %cálculo do valor da incógnita
            end
        end
    end
    disp(y);
    disp('Resolvendo agora Ux = y, por retrossubstituição,');
    disp('temos a solução para o vetor de incógnitas:');
    for i = dim:-1:1 %for para linhas
        soma = y(i,1);%const. que será somada com os valores já conhecidos
        for j = dim:-1:1 %for para colunas
            if(j~=i)
                soma = soma-(U(i,j)*x(j,1));
            end
            if(j==i)
                x(i,1) = soma / U(i,j); %cálculo do valor da incógnita
            end
        end
    end
end
```

Figura 15: algoritmo do Método da Decomposição LU implementado em MATLAB® (parte 2).

ANEXO 3

```matlab
function [ x ] = cholesky(A,b)
%Função que calcula a solução de um sistema linear usando o método da
%Decomposição de Cholesky.
%Parâmetros: x = cholesky(A,b)
%A = matriz dos coeficientes (deve ser quadrada e det(A)>0)
%b = vetor resposta ou vetor das constantes do sistema linear (deve ter o
%mesmo número de elementos que a ordem da matriz A)
%x = vetor de saída.

%validação dos parâmetros
if size(A,1)~=size(A,2) %testa se o número de linhas é igual ao de colunas.
    disp('Erro. A matriz dos coeficientes não é quadrada.');
    x = 0; return;
elseif det(A)==0 %testa se o determinante de A é igual a 0.
    disp('Erro. O determinante da matriz dos coeficientes é nulo.');
    x = 0; return;
end
dim = size(A,1);%determina a dimensão da matriz A e armazena em dim.
if size(b,1)~=1 && size(b,2)~=1%testa se é vetor linha ou coluna
    disp('Erro. O vetor resposta não é um vetor linha ou vetor coluna.');
    x = 0; return;
elseif size(b,1)~=dim && size(b,2)~=dim %testa equivalência entre A e b.
    fprintf('Erro. O vetor resposta deve ter o mesmo número de elementos');
    fprintf(' que a dimensão da matriz dos coeficientes.\n');
    x = 0; return;
end
if size(b,1)==1 %se b for um vetor linha,
    b = b';     %é transformado em vetor coluna.
end
At = A'; %At recebe a transposta de A
for i=1:dim %linhas
    for j=1:dim %colunas
        if A(i,j)~=At(i,j) %compara elemento a elemento
            disp('Erro. A matriz dos coeficientes não é simétrica.');
            x = 0; return;
        end
    end
end
clear At;
AA = []; %#ok<*AGROW> %matriz auxiliar
for i = 1:dim %for para ordem da matriz auxiliar
    for j = 1:i %for para linhas da matriz auxiliar
        for k = 1:i %for para colunas da matriz auxiliar
            AA(j,k) = A(j,k); %preenche matriz auxiliar com dados de A
        end
    end
    if det(AA)<=0 %testa se é definida positiva
        disp('Erro. A matriz dos coeficientes não é definida positiva.');
        x = 0; return;
    end
end
clear AA;
%fim da validação dos parâmetros.

```

Figura 16: algoritmo do Método da Decomposição de Cholesky implementado em MATLAB® (parte 1).

```matlab
%transformação de A para GxGt
%#ok<*NASGU>
G = zeros(dim);%declaração da matriz G como matriz nula
format short;
%Transformando a matriz A em uma matriz triangular inferior G
%através do método da decomposição de Cholesky
soma = 0; %variável auxiliar
for j = 1:dim %for para colunas
    for i = j:dim %for para linhas a partir da diagonal principal
        if i==j %G(i,j) é um elemento da diagonal principal
            soma = A(i,i);
            for k = 1:i
                if i~=k
                    soma = soma - (G(i,k)^2);
                else
                    G(i,i) = sqrt(soma);
                end
            end
        else %G(i,j) não é um elemento da diagonal principal
            soma = A(i,j);
            for k = 1:j
                if k<j
                    soma = soma - G(i,k)*G(i-1,k);
                else
                    G(i,k) = soma/G(k,k);
                end
            end
        end
    end
end
Gt = G';
disp('A matriz A foi decomposta em GxGt. Onde');
disp('G = '); disp(G);
disp('e Gt = '); disp(Gt);
%disp('GxGt = '); disp(G*Gt);
disp('Resolvendo inicialmente Gy = b, por substituições sucessivas,');
disp('temos y =');
format longe;
for i=1:dim %preenchendo vetores y e x com zeros
    %#ok<*AGROW>
    y(i,1)=0;
    x(i,1)=0;
end
for i = 1:dim %for para linhas
    soma = b(i,1);%const. que será somada com os valores já conhecidos
    for j = 1:dim %for para colunas
        if(j~=i)
            soma = soma-(G(i,j)*y(j,1));
        end
        if(j==i)
            y(i,1) = soma / G(i,j); %cálculo do valor da incógnita
        end
    end
end
disp(y);
disp('Resolvendo agora Gtx = y, por retrossubstituição,');
disp('temos a solução para o vetor de incógnitas:');
for i = dim:-1:1 %for para linhas
    soma = y(i,1);%const. que será somada com os valores já conhecidos
    for j = dim:-1:1 %for para colunas
        if(j~=i)
            soma = soma-(Gt(i,j)*x(j,1));
        end
        if(j==i)
            x(i,1) = soma / Gt(i,j); %cálculo do valor da incógnita
        end
    end
end
end
```

Figura 17: algoritmo do Método da Decomposição de Cholesky implementado em MATLAB® (parte 2).

ANEXO 4

```matlab
function [ x ] = jacobi(A,b,x0,tol,imax)
%Função que calcula a solução de um sistema linear usando o método
%iterativo de Jacobi.
%Parâmetros: x = jacobi(A,b,x0,tol,imax)
%A = matriz dos coeficientes (deve ser quadrada e det(A)~=0)
%b = vetor resposta ou vetor das constantes do sistema linear (deve ter o
%mesmo número de elementos que a dimensão da matriz A)
%x0 = vetor de estimativa inicial
%tol = tolerância para convergência do método
%imax = número máximo de iterações

%validação dos parâmetros
if isnumeric(tol)==false || isnumeric(imax)==false || ...
    isnumeric(x0)==false || isnumeric(A)==false || isnumeric(b)==false
    disp('Erro. Algum dos parâmetros não é numérico.');
    x = 0; return;
end
if size(A,1)~=size(A,2) %testa se o número de linhas é igual ao de colunas.
    disp('Erro. A matriz dos coeficientes não é quadrada.');
    x = 0; return;
elseif det(A)==0 %testa se o determinante de A é igual a 0.
    disp('Erro. O determinante da matriz dos coeficientes é nulo.');
    x = 0; return;
end
dim = size(A,1);%determina a dimensão da matriz A e armazena em dim.
if (size(b,1)~=1 && size(b,2)~=1) || ...%testa se é vetor linha ou coluna
        (size(x0,1)~=1 && size(x0,2)~=1)
    disp('Erro. Vetor b ou vetor x0 não é um vetor linha ou coluna.');
    x = 0; return;
elseif (size(b,1)~=dim && size(b,2)~=dim)||... %testa equivalência entre A,
        (size(x0,1)~=dim && size(x0,2)~=dim)   %b e x0.
    fprintf('Erro. Vetor b ou vetor x0 não tem quantidade de elementos');
    fprintf(' compatível com a matriz dos coeficientes.\n');
    x = 0; return;
end
if size(b,1)==1 %se b for um vetor linha,
    b = b';     %é transformado em vetor coluna.
end
if size(x0,1)==1 %se x0 for um vetor linha,
    x0 = x0';    %é transformado em vetor coluna.
end
soma = 0; cont = 0; %variáveis auxiliares
for i = 1:dim %for para linhas
    for j = 1:dim %for para colunas
        if i~=j %soma apenas os que estão fora da diagonal principal
            soma = soma + abs(A(i,j));
        end
    end
    if(A(i,i) >= soma)  %se o elemento da diagonal principal é maior que a
        cont = cont + 1;%soma dos módulos dos outros elementos, então o
    else                %contador é incrementado
        break;
    end
    soma = 0; %restaura o valor de soma
end
if(cont ~= dim) %matriz dos coeficientes não é diagonalmente dominante.
    disp('A matriz dos coeficientes não é diagonalmente dominante,');
    disp('ou seja, existe a possibilidade do método não convergir.');
else
    disp('A matriz dos coeficientes é diagonalmente dominante e o');
    disp('método converge.');
end
%fim da validação dos parâmetros.
```

Figura 18: algoritmo do Método iterativo de Jacobi implementado em MATLAB® (parte 1).

```
    %processamento
    format longe;%formato longo com notação exponencial
    x = zeros(dim,1); %declarando vetor x como vetor nulo.
    for k=1:imax
        soma = b; %variável auxiliar
        %cálculo da estimativa atual
        for i = 1:dim %for para linhas
            for j = 1:dim %for para colunas
                if i~=j
                    soma(i,1) = soma(i,1) - A(i,j)*x0(j,1);
                end
            end
            x(i,1) = soma(i,1)/A(i,i); %cálculo de x da iteração k
        end
        erro = abs((max(x-x0))/max(x)); %cálculo de erro
        %quando o erro relativo da estimativa atual com a anterior é menor
        %que a tolerância, temos o resultado da raiz.
        if erro<tol
            break; %finaliza laço de repetição
        end
        x0 = x; %atualiza o valor da estimativa anterior
    end
    if k==imax %núemro máximo de iterações atingido
       disp('Função finalizada por atingir o número máximo de iterações.');
    end
    fprintf('\nNúmero total de iterações: %d\n',k);
end
```

Figura 19: algoritmo do Método iterativo de Jacobi implementado em MATLAB® (parte 2).

ANEXO 5

```
    %processamento
    format longe;%formato longo com notação exponencial
    x = zeros(dim,1); %declarando vetor x como vetor nulo.
    for k=1:imax
        soma = b; %variável auxiliar
        xaux = x0; %vetor auxiliar que recebe o valor de x0.
        %cálculo da estimativa atual
        for i = 1:dim %for para linhas
            for j = 1:dim %for para colunas
                if i~=j
                    soma(i,1) = soma(i,1) - A(i,j)*x0(j,1);
                end
            end
            x(i,1) = soma(i,1)/A(i,i); %cálculo de x da iteração k
            x0(i,1) = x(i,1); %atualiza estimativa anterior no índice i
        end
        erro = abs((max(x-xaux))/max(x)); %cálculo de erro
        %quando o erro relativo da estimativa atual com a anterior é menor
        %que a tolerância, temos o resultado da raiz.
        if erro<tol
            break; %finaliza laço de repetição
        end
    end
    if k==imax %núemro máximo de iterações atingido
       disp('Função finalizada por atingir o número máximo de iterações.');
    end
    fprintf('\nNúmero total de iterações: %d\n',k);
end
```

Figura 20: trecho do algoritmo do Método iterativo de Gauss-Seidel implementado em MATLAB®. Todo o restante do algoritmo é idêntico ao de Jacobi, mostrado no Anexo 4.

ANEXO 6

Detalhes da 1ª questão.

```
questao_1.m

%#ok<*NOPTS>
%letra A)
A1 = [1.19 2.11 -100 1;14.2 -0.122 12.2 -1;0 100 -99.9 1;15.3 0.11 -13.1 -1];
b1 = [1.12 3.44 2.15 4.16];
x1 = eliminacao_gauss(A1,b1)

%letra B)
A2 = [2.12 -2.12 51.3 100;0.333 -0.333 -12.2 19.7;6.19 8.20 -1.00 -2.01;-5.73 6.12 1 -1];
b2 = [pi sqrt(2) 0 -1];
x2 = eliminacao_gauss(A2,b2)
```

Figura 21: script em MATLAB® para resolução da primeira questão com chamada da função eliminacao_gauss e parâmetros fornecidos nas letras A e B.

```
>> questao_1
Matriz aumentada do sistema linear inicial:
    1.1900    2.1100 -100.0000    1.0000    1.1200
   14.2000   -0.1220   12.2000   -1.0000    3.4400
         0  100.0000  -99.9000    1.0000    2.1500
   15.3000    0.1100  -13.1000   -1.0000    4.1600

Matriz aumentada do sistema linear após primeira etapa:
   1.0e+03 *

    0.0012    0.0021   -0.1000    0.0010    0.0011
         0   -0.0253    1.2055   -0.0129   -0.0099
         0         0    4.6648   -0.0501   -0.0371
         0         0         0   -0.0002    0.0002

Foram realizadas 0 operações de pivotação.

Solução para o vetor de incógnitas:

x1 =

     1.768252974993456e-01
     1.269269086768745e-02
    -2.065405013713113e-02
    -1.182608695468154e+00
```

Figura 22: resposta da função eliminação_gauss para a letra A.

```
Matriz aumentada do sistema linear inicial:
    2.1200   -2.1200   51.3000  100.0000    3.1416
    0.3330   -0.3330  -12.2000   19.7000    1.4142
    6.1900    8.2000   -1.0000   -2.0100         0
   -5.7300    6.1200    1.0000   -1.0000   -1.0000

Matriz aumentada do sistema linear após primeira etapa:
    2.1200   -2.1200   51.3000  100.0000    3.1416
         0   14.3900 -150.7863 -293.9911   -9.1729
         0         0  -20.2580    3.9925    0.9207
         0         0         0  305.5795   14.2730

Foram realizadas 1 operações de pivotação.

Solução para o vetor de incógnitas:

x2 =

     9.276104702949123e-02
    -6.299433961594411e-02
    -3.624582269162138e-02
     4.670801937981647e-02
```

Figura 23: resposta da função eliminação_gauss para a letra B.

ANEXO 7

Detalhes da 2ª questão:

questao_2.m

```matlab
%#ok<*NOPTS>
%letra A)
A1 = [1.19 2.11 -100 1;14.2 -0.122 12.2 -1;0 100 -99.9 1;15.3 0.11 -13.1 -1];
b1 = [1.12 3.44 2.15 4.16];
x3 = decomposicaolu(A1,b1)

%letra B)
A2 = [2.12 -2.12 51.3 100;0.333 -0.333 -12.2 19.7;6.19 8.20 -1.00 -2.01;-5.73 6.12 1 -1];
P = [1 0 0 0;0 0 1 0;0 1 0 0;0 0 0 1];
A3 = P*A2;
b2 = [pi sqrt(2) 0 -1]';
b3 = P*b2;
x4 = decomposicaolu(A3,b3)
```

Figura 24: script em MATLAB® para resolução da segunda questão com chamada da função *decomposicaolu* e parâmetros fornecidos nas letras A e B. Na letra B, definimos uma matriz de permutação P que foi multiplicada à esquerda pela matriz A2 e pelo vetor resposta b2, seguindo o procedimento do pivoteamento parcial.

```
>> questao_2
A matriz A foi decomposta em LxU. Onde
L =
    1.0000         0         0         0
   11.9328    1.0000         0         0
         0   -3.9525    1.0000         0
   12.8571    1.0679   -0.0032    1.0000

e U =
   1.0e+03 *

    0.0012    0.0021   -0.1000    0.0010
         0   -0.0253    1.2055   -0.0129
         0         0    4.6648   -0.0501
         0         0         0   -0.0002

Resolvendo inicialmente Ly = b, por substituições sucessivas,
temos y =
     1.120000000000000e+00
    -9.924705882352944e+00
    -3.707785196089439e+01
     2.416401546850583e-01

Resolvendo agora Ux = y, por retrossubstituição,
temos a solução para o vetor de incógnitas:

x3 =

     1.768252974993456e-01
     1.269269086768745e-02
    -2.065405013713113e-02
    -1.182608695468154e+00
```

```
A matriz A foi decomposta em LxU. Onde
L =
    1.0000         0         0         0
    2.9198    1.0000         0         0
    0.1571         0    1.0000         0
   -2.7028    0.0271   -7.0956    1.0000

e U =
    2.1200   -2.1200   51.3000  100.0000
         0   14.3900 -150.7863 -293.9911
         0         0  -20.2580    3.9925
         0         0         0  305.5795

Resolvendo inicialmente Ly = b, por substituições sucessivas,
temos y =
     3.141592653589793e+00
    -9.172857795151330e+00
     9.207464144271513e-01
     1.427301478789979e+01

Resolvendo agora Ux = y, por retrossubstituição,
temos a solução para o vetor de incógnitas:

x4 =

     9.276104702949123e-02
    -6.299433961594411e-02
    -3.624582269162138e-02
     4.670801937981647e-02
```

Figura 25: resposta da função decomposicaolu para as letras A (à esquerda) e B (à direita).

ANEXO 8

Detalhes da 3ª questão:

```
questao_3.m
A = [4 1 1 1;1 3 0 -1;1 0 2 1;1 -1 1 4];
b = [2 2 1 1]';
x = cholesky(A,b) %#ok<*NOPTS>
```

Figura 26: script em MATLAB® para resolução da terceira questão com chamada da função *cholesky* e parâmetros fornecidos no enunciado da questão.

```
>> questao_3
A matriz A foi decomposta em GxGt. Onde
G =
    2.0000         0         0         0
    0.5000    1.6583         0         0
    0.5000   -0.1508    1.3143         0
    0.5000   -0.7538    0.4842    1.7168

e Gt =
    2.0000    0.5000    0.5000    0.5000
         0    1.6583   -0.1508   -0.7538
         0         0    1.3143    0.4842
         0         0         0    1.7168

Resolvendo inicialmente Gy = b, por substituições sucessivas,
temos y =
     1.000000000000000e+00
     9.045340337332909e-01
     4.842001247062522e-01
     5.518254055364692e-01

Resolvendo agora Gtx = y, por retrossubstituição,
temos a solução para o vetor de incógnitas:

x =

     1.785714285714286e-01
     7.142857142857143e-01
     2.500000000000000e-01
     3.214285714285714e-01
```

Figura 27: resposta da função cholesky para o sistema linear da terceira questão.

ANEXO 9

Detalhes da 4ª questão:

questao_4_a.m | questao_4_b.m | questao_4_c.m | +

```matlab
%#ok<*NOPTS>
clear, clc
%Sistema A
A1 = [4 1 -1;-1 3 1;2 2 5];
b1 = [5;-4;1];
%Sistema B
A2 = [1 0 -1;-0.5 1 -0.25;1 -0.5 1];
b2 = [0.2;-1.425;2];

%4.a)
x0 = zeros(3,1);
tol = 1e-3;
imax = 300;
disp('4.a) Métodos abertos Jacobi e Gauss-Seidel no Sistema A.');
disp('Usando Jacobi no sistema A:');
x1 = jacobi(A1,b1,x0,tol,imax)
disp('Usando Gauss-Seidel no sistema A:');
x2 = gauss_seidel(A1,b1,x0,tol,imax)
disp('4.a) Métodos abertos Jacobi e Gauss-Seidel no Sistema B.');
disp('Usando Jacobi no sistema B:');
x3 = jacobi(A2,b2,x0,tol,imax)
disp('Usando Gauss-Seidel no sistema B:');
x4 = gauss_seidel(A2,b2,x0,tol,imax)

resposta_a = [x1 x2 x3 x4]
```

Figura 28: script em MATLAB® para resolução do item 4.a) da 4ª questão com chamada das funções *jacobi* e *gauss_seidel* e parâmetros fornecidos no enunciado da questão.

questao_4_b.m | questao_4_c.m | +

```matlab
%#ok<*NOPTS>
clear, clc
%Sistema A
A1 = [4 1 -1;-1 3 1;2 2 5];
b1 = [5;-4;1];
%Sistema B
A2 = [1 0 -1;-0.5 1 -0.25;1 -0.5 1];
b2 = [0.2;-1.425;2];

%4.b)
tol = 1e-3;
imax = 300;
x0 = [1;0;1];
disp('4.b) Métodos abertos Jacobi e Gauss-Seidel no Sistema A.');
disp('Usando Jacobi no sistema A:');
x5 = jacobi(A1,b1,x0,tol,imax)
disp('Usando Gauss-Seidel no sistema A:');
x6 = gauss_seidel(A1,b1,x0,tol,imax)
disp('4.b) Métodos abertos Jacobi e Gauss-Seidel no Sistema B.');
disp('Usando Jacobi no sistema B:');
x7 = jacobi(A2,b2,x0,tol,imax)
disp('Usando Gauss-Seidel no sistema B:');
x8 = gauss_seidel(A2,b2,x0,tol,imax)

resposta_b = [x5 x6 x7 x8]
```

Figura 29: script em MATLAB® para resolução do item 4.b) da 4ª questão com chamada das funções jacobi e *gauss_seidel* e parâmetros fornecidos no enunciado da questão.

questao_4_c.m

```
%#ok<*NOPTS>
clear, clc
%Sistema A
A1 = [4 1 -1;-1 3 1;2 2 5];
b1 = [5;-4;1];
%Sistema B
A2 = [1 0 -1;-0.5 1 -0.25;1 -0.5 1];
b2 = [0.2;-1.425;2];
format longe;

%método direto
disp('4.c) Método direto: linsolve.');
disp('Usando linsolve no sistema A:');
x9 = linsolve(A1,b1)
disp('Usando linsolve no sistema B:');
x10 = linsolve(A2,b2)

%método aberto
tol = 1e-3;
imax = 300;
disp('4.c) Método aberto: Gradientes Biconjugados (bicg).');
disp('Usando bicg no sistema A:');
[x11,flag11,relres11,iter11] = bicg(A1,b1,tol,imax)
disp('Usando bicg no sistema B:');
[x12,flag12,relres12,iter12] = bicg(A2,b2,tol,imax)

resposta_c = [x9 x10 x11 x12]
```

Figura 30: script em MATLAB® para resolução do item 4.c) da 4ª questão com chamada das funções *linsolve* e *bicg* e parâmetros fornecidos no enunciado da questão.

```
4.a) Métodos abertos Jacobi e Gauss-Seidel no Sistema A.
Usando Jacobi no sistema A:
A matriz dos coeficientes é diagonalmente dominante e o
método converge.

Número total de iterações: 9

x1 =

     1.448215914351852e+00
    -8.356592592592592e-01
    -4.508972222222218e-02

Usando Gauss-Seidel no sistema A:
A matriz dos coeficientes é diagonalmente dominante e o
método converge.

Número total de iterações: 6

x2 =

     1.447816349987140e+00
    -8.358173037765774e-01
    -4.479961848422502e-02
```

Figura 31: saída do script da Figura 28 para o sistema A.

```
4.a) Métodos abertos Jacobi e Gauss-Seidel no Sistema B.
Usando Jacobi no sistema B:
A matriz dos coeficientes não é diagonalmente dominante,
ou seja, existe a possibilidade do método não convergir.

Número total de iterações: 147

x3 =

     8.884175778371142e-01
    -7.918180607515059e-01
     6.821278642529894e-01

Usando Gauss-Seidel no sistema B:
A matriz dos coeficientes não é diagonalmente dominante,
ou seja, existe a possibilidade do método não convergir.

Número total de iterações: 17

x4 =

     8.996205292396300e-01
    -8.002846030702775e-01
     7.002371692252313e-01
```

Figura 32: saída do script da Figura 28 para o sistema B.

```
4.b) Métodos abertos Jacobi e Gauss-Seidel no Sistema A.
Usando Jacobi no sistema A:
A matriz dos coeficientes é diagonalmente dominante e o
método converge.

Número total de iterações: 5

x5 =

     1.440694444444445e+00
    -8.388888888888887e-01
    -5.355555555555562e-02

Usando Gauss-Seidel no sistema A:
A matriz dos coeficientes é diagonalmente dominante e o
método converge.

Número total de iterações: 6

x6 =

     1.447803931970165e+00
    -8.357661447616599e-01
    -4.481511488340186e-02
```

Figura 33: saída do script da Figura 29 para o sistema A.

```
4.b) Métodos abertos Jacobi e Gauss-Seidel no Sistema B.
Usando Jacobi no sistema B:
A matriz dos coeficientes não é diagonalmente dominante,
ou seja, existe a possibilidade do método não convergir.

Número total de iterações: 117

x7 =

     8.916910565061291e-01
    -7.961442289268697e-01
     6.909849127123907e-01

Usando Gauss-Seidel no sistema B:
A matriz dos coeficientes não é diagonalmente dominante,
ou seja, existe a possibilidade do método não convergir.

Número total de iterações: 16

x8 =

     8.997397914786034e-01
    -8.001951563910474e-01
     7.001626303258729e-01
```

Figura 34: saída do script da Figura 29 para o sistema B.

```
4.c) Método direto: linsolve.
Usando linsolve no sistema A:

x9 =

     1.447761194029851e+00
    -8.358208955223880e-01
    -4.477611940298504e-02

Usando linsolve no sistema B:

x10 =

     9.000000000000001e-01
    -7.999999999999999e-01
     7.000000000000001e-01
```

Figura 35: saída do script da Figura 30 para os sistemas A e B, contemplando apenas a função *linsolve*.

```
4.c) Método aberto: Gradientes Biconjugados (bicg).
Usando bicg no sistema A:

x11 =

     1.447761194029851e+00
    -8.358208955223880e-01
    -4.477611940298516e-02


flag11 =

     0
```

```
Usando bicg no sistema B:

x12 =

     9.000000000000004e-01
    -7.999999999999999e-01
     7.000000000000002e-01


flag12 =

     0
```

Figura 36: saída do script da Figura 30 para os sistemas A e B, contemplando apenas a função *bicg*.

ANEXO 10

Detalhes da 5ª questão:

```
>> A = [1 -1 0 0 0;-1 2 -1 0 0;0 -1 2.001 -1 0;0 0 -1 2 -1;0 0 0 1 -1]

A =

    1.0000   -1.0000         0         0         0
   -1.0000    2.0000   -1.0000         0         0
         0   -1.0000    2.0010   -1.0000         0
         0         0   -1.0000    2.0000   -1.0000
         0         0         0    1.0000   -1.0000

>> b = [1 1 1 1 1]

b =

     1     1     1     1     1
```

Figura 37: Exposição dos parâmetros para resolução da questão 5.

```
>> eliminacao_gauss(A,b)
Matriz aumentada do sistema linear inicial:
    1.0000   -1.0000         0         0         0    1.0000
   -1.0000    2.0000   -1.0000         0         0    1.0000
         0   -1.0000    2.0010   -1.0000         0    1.0000
         0         0   -1.0000    2.0000   -1.0000    1.0000
         0         0         0    1.0000   -1.0000    1.0000

Matriz aumentada do sistema linear após primeira etapa:
    1.0000   -1.0000         0         0         0    1.0000
         0    1.0000   -1.0000         0         0    2.0000
         0         0    1.0010   -1.0000         0    3.0000
         0         0         0    1.0010   -1.0000    3.9970
         0         0         0         0   -0.0010   -2.9930

Foram realizadas 0 operações de pivotação.

Solução para o vetor de incógnitas:

ans =

     3.003000000000589e+03
     3.002000000000589e+03
     3.000000000000589e+03
     3.000000000000589e+03
     2.999000000000589e+03
```

Figura 38: Resolução da letra (a) da questão 5.

ANEXO 11

```
questao_6.m
%#ok<*NOPTS>
clear, clc
%Matriz dos coeficientes do sistema A
A1 = [1 2;1.0001 2];
%Matriz dos coeficientes do sistema B
A2 = [1 2;0.9999 2];
%vetor resposta (igual para ambos os sistemas)
b = [3;3.0001];

%resolvendo por eliminação de gauss
disp('Resolvendo Sistema A por matriz inversa:');
x1 = A1\b
disp('Resolvendo Sistema B por matriz inversa:');
x2 = A2\b
respostas_6 = [x1 x2]
disp('Condicionamento das matrizes A1 (c1) e A2 (c2):');
c1 = cond(A1), c2 = cond(A2)
```

Figura 39: script em MATLAB® para resolução da 6ª questão, item 6.a).

ANEXO 12

Detalhes da 7ª questão:

```
%#ok<*NOPTS>
clear, clc
%matriz dos coeficientes
A = [0,1,5,-7,23,-1,7,8,1,-5];
A = [A;17,0,-24,-75,100,-18,10,-8,9,-50];
A = [A;3,-2,15,0,-78,-90,-70,18,-75,1];
A = [A;5,5,-10,0,-72,-1,80,-3,10,-18];
A = [A;100,-4,-75,-8,0,83,-10,-75,3,-8];
A = [A;70,85,-4,-9,2,0,3,-17,-1,-21];
A = [A;1,15,100,-4,-23,13,0,7,-3,17];
A = [A;16,2,-7,89,-17,11,-73,0,-8,-23];
A = [A;51,47,-3,5,-10,18,-99,-18,0,12];
A = [A;1,1,1,1,1,1,1,1,1,0];
b = [10;-40;-17;43;-53;12;-60;100;0;100];%vetor resposta

%7.a) Método direto: Eliminação de Gauss
disp('Resolvendo sistema por Eliminação de Gauss:');
x1 = eliminacao_gauss(A,b)
%7.a) Método iterativo: Gauss-Seidel
disp('Resolvendo sistema por Gauss-Seidel:');
x0 = zeros(size(A,1),1);%vetor de estimativa inicial nulo
x2 = gauss_seidel(A,b,x0,1e-5,100)
%7.b) Método direto: linsolve (decomposição LU)
disp('Resolvendo sistema por Decomposição LU:');
x3 = linsolve(A,b)
%7.a) Método iterativo: bicg (Gradiente Biconjugado)
disp('Resolvendo sistema por Gradiente Biconjugado:');
[x4,flag4,relres4,iter4] = bicg(A,b,1e-5,100)

disp('Vetor com todas as respostas em ordem de execução:');
respostas = [x1,x2,x3,x4]
```

Figura 40: script em MATLAB® para resolução da 7ª questão com chamada das funções e parâmetros fornecidos no enunciado da questão.

Resolvendo sistema por Eliminação de Gauss:
Matriz aumentada do sistema linear inicial:

0	1	5	-7	23	-1	7	8	1	-5	10
17	0	-24	-75	100	-18	10	-8	9	-50	-40
3	-2	15	0	-78	-90	-70	18	-75	1	-17
5	5	-10	0	-72	-1	80	-3	10	-18	43
100	-4	-75	-8	0	83	-10	-75	3	-8	-53
70	85	-4	-9	2	0	3	-17	-1	-21	12
1	15	100	-4	-23	13	0	7	-3	17	-60
16	2	-7	89	-17	11	-73	0	-8	-23	100
51	47	-3	5	-10	18	-99	-18	0	12	0
1	1	1	1	1	1	1	1	1	0	100

Matriz aumentada do sistema linear após primeira etapa:
1.0e+03 *

0.0170	0	-0.0240	-0.0750	0.1000	-0.0180	0.0100	-0.0080	0.0090	-0.0500	-0.0400
0	0.0010	0.0050	-0.0070	0.0230	-0.0010	0.0070	0.0080	0.0010	-0.0050	0.0100
0	0	0.0292	-0.0008	-0.0496	-0.0888	-0.0578	0.0354	-0.0746	-0.0002	0.0101
0	0	0	0.0563	-0.2639	-0.0756	-0.0131	-0.0068	-0.0689	0.0215	0.0144
0	0	0	0	1.5587	0.9935	0.2246	-0.0511	0.6725	0.1109	0.0886
0	0	0	0	0	-0.4360	-1.2553	-0.1169	-0.4098	0.1819	-0.8556
0	0	0	0	0	0	-0.4692	-0.1650	-0.0297	0.1029	-0.5372
0	0	0	0	0	0	0	-0.0305	0.0394	-0.0796	0.0019
0	0	0	0	0	-0.0000	0.0000	0	0.0513	-0.0955	0.1115
0	0	0	0	0	0.0000	-0.0000	0	0	0.0018	0.0945

Foram realizadas 1 operações de pivotação.

Figura 41: saída do script para a chamada da função *eliminacao_gauss* (parte 1).

```
Solução para o vetor de incógnitas:

x =

     8.749970902461146e+01
    -5.575380417303086e+01
     1.777411634849739e+01
     3.204744365636587e+01
     1.174397126243134e+01
    -9.436514182219534e+01
     9.184895568160265e+00
    -8.106280536611729e+00
     9.997509067177154e+01
     5.253605363780216e+01
```

Figura 42: saída do script para a chamada da função *eliminacao_gauss* (parte 2).

```
Resolvendo sistema por Gauss-Seidel:
A matriz dos coeficientes não é diagonalmente dominante,
ou seja, existe a possibilidade do método não convergir.
Função finalizada por atingir o número máximo de iterações.

Número total de iterações: 100

x2 =

   NaN
   NaN
   NaN
   NaN
   NaN
   NaN
   NaN
   NaN
   NaN
   NaN

Resolvendo sistema por Decomposição LU:

x3 =

     8.749970902461172e+01
    -5.575380417303131e+01
     1.777411634849743e+01
     3.204744365636594e+01
     1.174397126243135e+01
    -9.436514182219554e+01
     9.184895568160274e+00
    -8.106280536611651e+00
     9.997509067177180e+01
     5.253605363780227e+01
```

```
Resolvendo sistema por Gradiente Biconjugado:

x4 =

     8.749970902459228e+01
    -5.575380417312918e+01
     1.777411634862331e+01
     3.204744365641339e+01
     1.174397126239938e+01
    -9.436514182222106e+01
     9.184895568178058e+00
    -8.106280536570683e+00
     9.997509067175038e+01
     5.253605363780079e+01


flag4 =

     0


relres4 =

     1.604492142254538e-10


iter4 =

    10
```

Figura 43: saída do script para as chamadas das funções *gauss_seidel linsolve e bicg*. O método de Gauss-Seidel diverge, o método da Decomposição LU funciona corretamente e o método do Gradiente Biconjugado converge em 10 iterações para a tolerância exigida.