

ALC Prova 2: Trabalho de Implementação

Professor Érito - Algebra Linear Computacional

William Anderson & Livia de Azevedo

08 de dezembro de 2014

Sumário

Exercício 1	5
Exercício 2	6
Exercício 3	7
Exercício 4	9
Exercício 5	11
Exercício 6	12
Exercício 7	18
Exercício 8	20
Exercício 9	23
Exercício 10	31
Exercício 11	35
Exercício 12	39
Exercício 13	43
Exercício 14	45
Exercício 15	47
Exercício 16	60
Exercício 17	65
Exercício 18	69
Exercício 19	70
Exercício 20	75
Exemplos Questão 1	85
Exemplos Questão 2	86
Exemplos Questão 3	87
Exemplos Questão 4	87

Exemplos Questão 5	88
Exemplos Questão 6	88
Exemplos Questão 7	89
Exemplos Questão 8	90
Exemplos Questão 9	93
Exemplos Questão 10	94
Exemplos Questão 11	95
Exemplos Questão 12	96
Exemplos Questão 13	98
Exemplos Questão 14	99
Exemplos Questão 15	99
Exemplos Questão 16	101
Exemplos Questão 17	102
Exemplos Questão 18	102
Exemplos Questão 19	103
Exemplos Questão 20	104

Lista de Figuras

1	Exemplo.1 Questão 01 no terminal	85
2	Exemplo.1 Questão 01 no MATLAB	85
3	Exemplo.1 Questão 02 no terminal	86
4	Exemplo.1 Questão 02 no MATLAB	86
5	Exemplo.1 Questão 03 no terminal	87
6	Exemplo.1 Questão 04 no terminal	87
7	Exemplo.1 Questão 05 no terminal	88
8	Exemplo.1 Questão 06 no terminal	88
9	Exemplo.1 Questão 06 no MATLAB	89
10	Exemplo.1 Questão 07 no terminal	89
11	Exemplo.2 Questão 07 no terminal	90
12	Exemplo.1 Questão 08 no terminal	90
13	Exemplo.1 Questão 08 no MATLAB	91
14	Exemplo.2 Questão 08 no terminal	91
15	Exemplo.2 Questão 08 no MATLAB	92
16	Exemplo.1 Questão 09 no terminal	93
17	Exemplo.1 Questão 10 no terminal	94
18	Exemplo.1 Questão 10 no MATLAB	94
19	Exemplo.1 Questão 11 no MATLAB	95
20	Exemplo.1 Questão 11 no MATLAB	95
21	Exemplo.1 Questão 12 no terminal	96
22	Exemplo.1 Questão 12 no MATLAB	97
23	Exemplo.1 Questão 13 no terminal	98
24	Exemplo.2 Questão 13 no terminal	98
25	Exemplo.1 Questão 14 no terminal	99
26	Exemplo.1-a Questão 15 no terminal	99
27	Exemplo.1-b Questão 15 no terminal	100
28	Exemplo.1-c Questão 15 no terminal	100
29	Exemplo.1 Questão 16 no terminal	101
30	Exemplo.2 Questão 16 no terminal	101
31	Exemplo.1 Questão 17 no terminal	102
32	Exemplo.1 Questão 18 no terminal	102
33	Exemplo.1 Questão 19 no terminal	103
34	Exemplo.2 Questão 19 no terminal	103
35	Exemplo.1 Questão 20 no terminal	104

Exercício 1

Para a primeira questão a solução atribuída foi utilizar uma variável temporária que guardasse o valor durante as contas para fazer-se o somatório da multiplicação de uma linha da matriz $A_{m \times n}$ pelo vetor b_n repetidos n vezes:

- código de implementação :

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     // variaveis de controle
6     int m, n, num_flops = 0;
7
8     // obtendo dimensao da maitriz
9     printf("Entre com o numero de linhas\n");
10    scanf("%d", &m);
11    printf("entre com o numero de colunas\n");
12    scanf("%d", &n);
13
14    // criando os fatores a serem multiplicados
15    float A[m][n], b[n], Result[m];
16
17    printf("Entre com a matriz A\n");
18    int i, j;
19    for(i = 0; i < m ; i++)
20    {
21        for(j = 0; j < n ; j++)
22        {
23            scanf("%f", &A[i][j]);
24        }
25    }
26
27    printf("Entre com o vetor b\n");
28    for(i = 0; i < n ; i++)
29    {
30        scanf("%f", &b[i]);
31    }
32
33
34    // aplicando a multiplicacao de Result = Ab
35    for(i = 0; i < m ; i++)
36    {
37        float somatorio = 0;
38        for(j = 0; j < n ; j++)
39        {
40            somatorio += A[i][j]*b[j]; // guardando o valor temporario
41            num_flops += 2; // contando o numero de flops
42        }
43        Result[i] = somatorio; // ap s o calculo atribui o valor
44    }
```

```
45
46 // imprimindo o resultado
47 for(i = 0; i < m ; i++)
48 {
49     printf("resultado de Ab:\n");
50     printf("%f\n", Result[i]);
51 }
52
53 // imprimindo numero de flops
54 printf("numero de flops:\n");
55 printf("%d\n", num_flops);
56 }
```

Listing 1: Multiplicação $A_{m \times n}$ por b_n

Exercício 2

Para a segunda questão a solução atribuída foi utilizar a multiplicação padrão de matrizes sem pular nenhum step e contando o número de flops:

- código de implementação :

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     // variaveis de controle
6     int m, n, k, num_flops = 0;
7
8     // obtendo dimensoes das matrizes
9     printf("Entre com o valor de m\n");
10    scanf("%d", &m);
11    printf("Entre com o valor de n\n");
12    scanf("%d", &n);
13    printf("Entre com o valor de k\n");
14    scanf("%d", &k);
15
16    // matrizes operadoras
17    float A[m][n], B[n][k], Result[m][k];
18
19    // lendo A
20    printf("Entre com a matriz A\n");
21    int i, j;
22    for(i = 0; i < m ; i++)
23    {
24        for(j = 0; j < n ; j++)
25        {
26            scanf("%f", &A[i][j]);
27        }
28    }
```

```
29
30     printf("Entre com a matriz B\n");
31     for(i = 0; i < n ; i++)
32     {
33         for(j = 0; j < k ; j++)
34         {
35             scanf("%f", &B[i][j]);
36         }
37     }
38
39     // multiplicando Result = AB
40     for(i = 0; i < m ; i++)
41     {
42         int l;
43         for(l = 0; l < k ; l++)
44         {
45             float somatorio = 0;
46
47             for(j = 0; j < n ; j++)
48             {
49                 somatorio += A[i][j]*B[j][l]; // guardando o valor temporario
50                 num_flops += 2; // contando o numero de flops
51             }
52             Result[i][l] = somatorio; // ap s o calculo atribui o valor
53         }
54     }
55
56     // imprimindo o resultado
57     for(i = 0; i < m ; i++)
58     {
59         for(j = 0; j < k ; j++)
60         {
61             printf("resultado de AB:\n");
62             printf("%f\n", Result[i][j]);
63         }
64     }
65
66     // imprimindo o numero de flops
67     printf("numero de flops:\n");
68     printf("%d\n", num_flops);
69 }
```

Listing 2: Multiplicao $A_{m \times n}$ por $B_{n \times k}$

Exercício 3

Para a terceira questão a solução atribuída precisou de algumas adaptações e fórmulas deduzidas...

Det2x2 : Nesta função é implementada a fórmula da multiplicação cruzada invertendo o sinal da diagonal reversa

Det3x3 : Nesta função é implementada a fórmula do determinante de Laplace: $\sum_{i=1}^n (-1)^{i+j} \cdot a_{ij} \cdot \text{Det2x2}(\text{subMatriz}A_{ij})$.

Também foi deduzida a fórmula de criação “automática” das submatrizes. Considerando a linha isolada sempre sendo a primeira, as submatrizes sempre começam da linha $l+1$ (na qual a variável l controla em qual linha será depositado o valor) e com a coluna pega sempre sendo em função de c que é restringida pelo valor de i (caso sejam iguais, a coluna é ignorada, caso $c < i$ a coluna a receber o valor é dada pelo próprio valor de c , caso $c > i$ a coluna a ser depositado o valor é dada por $c - 1$)

Neste caso também é contado o número de flops

- código de implementação :

```

1  #include <stdio.h>
2  #include <math.h>
3
4  //funcao que calcula um determinante de uma matriz 2x2
5  float Det2x2(float A[2][2])
6  {
7      return A[0][0]*A[1][1] - A[0][1]*A[1][0];
8  }
9
10 //funcao que calcula um determinante de uma matriz 3x3 por la place usando a 2
    x2
11 float Det3x3 (float B[3][3])
12 {
13     float DetB = 0;
14     int i;
15     for (i=0; i<3; i++)
16     {
17         float subB[2][2];
18         int l, c;
19         for (l=0; l<2; l++) //forma a matriz 2x2 que resta dos isolamentos
20         {
21             for (c=0; c<3; c++)
22             {
23                 if (c<i)
24                     subB[l][c] = B[l+1][c]; // deduzidas a partir de observacoes
25                 if (c>i)
26                     subB[l][c-1] = B[l+1][c]; // deduzidas a partir de
27                                         observacoes
28                 if (c==i)
29                     continue;
30             }
31             //formula de do determinante de la place
32             DetB += ( pow( (-1), ((i+1)+1) ) * Det2x2(subB) * B[i][l] );
33         }
34     }
35     return DetB;

```



```
36 }
37
38 int main(void)
39 {
40     //variaveis de controle
41     int i, j;
42     printf("Entre com uma matriz 2x2\n");
43     float A[2][2];
44     for(i=0;i<2;i++)
45         for(j=0;j<2;j++)
46             scanf("%f", &A[i][j]);*
47
48     //imprimindo o primeiro resultado
49     printf("Resultado do determinante 2x2 : %f\n", Det2x2(A));
50     printf("Entre com uma matriz 3x3\n");
51     float B[3][3];
52
53     for(i=0;i<3;i++)
54         for(j=0;j<3;j++)
55             scanf("%f", &B[i][j]);
56
57     //imprimindo o segundo resultado
58     printf("Resultado do determinante 3x3 : %f\n", Det3x3(B));
59 }
```

Listing 3: Determinante de $A_{2 \times 2}$ e $B_{3 \times 3}$

Exercício 4

Nesta questão a solução atribuída foi, utilizando o próprio vetor b para guardar os valores intermediários, seguir da ultima linha da matriz até a primeira e subtraindo de b os coeficientes de todas as colunas maiores que o numero da linha atual de forma que na linha atual apenas necessitasse aplicar a divisão do próprio b da linha atual pelo coeficiente da diagonal na linha atual.

- código de implementação :

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     //variaveis de controle
6     int k, num_flops = 0;
7
8     //lendo a dimensao do sistema
9     printf("valor de k\n");
10    scanf("%d", &k);
11
12    //criando operadores
```

```
13 float A[k][k], b[k], x[k];
14
15 //leitura da matriz de coeficientes
16 printf("ler matriz A\n");
17 int i, j;
18 for(i = 0; i < k ; i++)
19 {
20     for(j = 0; j < k ; j++)
21     {
22         scanf("%f", &A[i][j]);
23     }
24 }
25
26 //leitura do vetor b
27 printf("ler vetor b\n");
28 for(i = 0; i < k ; i++)
29 {
30     scanf("%f", &b[i]);
31 }
32
33 //seguinte da ultima linha ate a primeira
34 for(i = k-1 ; i >= 0 ; i--)
35 {
36     //seguinte da ultima coluna ate a coluna de mesmo numero da linha
37     for(j = k-1 ; j >= i ; j--)
38     {
39         if(j>i)
40         {
41             //utilizando o proprio b para guardar os resultados
42             //intermediarios
43             b[i] -= A[i][j]*x[j];
44             num_flops += 2; //contagem do numero de flops (subtracao e
45                             //multiplicacao)
46         }
47         else
48         {
49             //enverrando a operacao da linha com a divisao
50             b[i] /= A[i][j];
51             num_flops += 1; //contagem do numero de flops (divisao)
52         }
53     }
54     x[i] = b[i]; //guardando o resultado em x
55 }
56
57 //imprimindo a solucao
58 printf("valores de x\n");
59 for(i = 0; i < k ; i++)
60     printf("%f\n", x[i]);
61
62 //imprimindo o numero de flops
63 printf("numero de flops:\n");
64 printf("%d\n", num_flops);
65 }
```

Listing 4: resolução de um sistema triangular inferior

Exercício 5

Este problema segue uma ideia extremamente semelhante a do exercício anterior, o código foi inteiramente copiado com duas pequenas alterações.

Uma delas foi a ordem dos loops, passando a andar da primeira linha até a última e das menores colunas da matriz até o elemento anterior ao termo da diagonal da linha atual. A segunda foi a condição que antes era $\forall j > i$ e passou a ser $\forall j < i$

- código de implementação :

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     //variaveis de controle
6     int k, num_flops = 0;
7
8     //lendo a dimensao do sistema
9     printf("valor de k\n");
10    scanf("%d", &k);
11
12    //criando os operadores
13    float A[k][k], b[k], x[k];
14
15    //lendo a matriz de coeficientes
16    printf("ler matriz A\n");
17    int i, j;
18    for(i = 0; i < k ; i++)
19    {
20
21        for(j = 0; j < k ; j++)
22        {
23            scanf("%f", &A[i][j]);
24        }
25    }
26    //kendo o vetor b
27    printf("ler vetor b\n");
28    for(i = 0; i < k ; i++)
29    {
30        scanf("%f", &b[i]);
31    }
32
33    //indo da primeira linha da matriz ate a ultima
34    for(i = 0 ; i < k ; i++)
```

```

35     {
36         //indo da primeira coluna ate o termo da diagonal
37         for(j = 0 ; j <= i ; j++)
38         {
39             if(j<i)
40             {
41                 //usando o proprio b para guardar os valores intermediarios
42                 b[i] -= A[i][j]*x[j];
43                 num_flops += 2;//contagem do numero de flops (subtracao e
44                             multiplicacao)
45             }
46             else
47             {
48                 b[i] /= A[i][j]; //dividindo pelo termo da diagonal
49                 num_flops += 1;//contagem do numero de flops (divisao)
50             }
51             //encerrando a operacao da linha guardando o valor em x
52             x[i] = b[i];
53         }
54
55         //imprimindo a solucao do sistema
56         printf("valores de x\n");
57
58         for(i = 0; i < k ; i++)
59             printf("%4f\n", x[i]);
60
61         //imprimindo o numero de flops
62         printf("numero de flops:\n");
63         printf("%d\n", num_flops);
64     }

```

Listing 5: resolução de um sistema triangular superior

Exercício 6

Neste exercício, calcularemos o fator de Cholesky de uma matriz $C_{m \times n}$. O programa pedirá ao usuário a inserção de uma matriz $A_{m \times n}$ inversível. A matriz C será:

$$C = A^T \cdot A$$

Pois o teorema afirma: "Se A é uma matriz inversível, o resultado de $A^T \cdot A$ será uma matriz positiva definida."

Se C será uma matriz positiva definida, podemos obter seu fator de Cholesky. Usaremos as seguintes fórmulas para calcular Cholesky:

Fórmula 1:

$$r_{11} = \sqrt{a_{11}}$$

Fórmula 2:

$$r_{1j} = \frac{a_{1j}}{r_{11}}, j = 2, \dots, n$$

Fórmula 3:

$$r_{ii} = + \sqrt{a_{ii} - \sum_{k=1}^{i-1} r_{ki}^2}$$

Fórmula 4:

$$r_{ij} = \frac{a_{ij} - \left(\sum_{k=1}^{i-1} (r_{ki} \cdot r_{kj}) \right)}{r_{ii}}, j = i + 1, \dots, n$$

Cada fórmula dessa será aplicada no código do programa. Primeiro calcularemos a primeira. Com todos os elementos da segunda encontrados, aplicaremos a terceira. Com todos os elementos da terceira obtidos, determinaremos os elementos da quarta. Seguiremos esse simples passo. Existe uma dependência entre as fórmulas e por isso é importante que sigamos esta ordem. Com o fim dos cálculos, teremos todos os elementos do fator de Cholesky determinados. Exibiremos na tela o fator de Cholesky encontrado.

Letra b :

Aqui utilizaremos a matriz C e seu fator de Cholesky para resolver o sistema $Ax = b$. O usuário irá inserir o vetor b e o retornaremos com a solução do sistema. O processo lógico é simples. Apenas faremos essas relações:

$$Ax = b \implies R^T \cdot R \cdot x = b$$

Considerando $R \cdot x = y$, teremos:

$$R^T \cdot y = b$$

Resolveremos o sistema em dois passos:

Passo 1 : Resolver $R^T \cdot y = b$ por substituição para trás e encontrar y. Utilizaremos o raciocínio do exercício 5.

Passo 2 : Resolver $R \cdot x = y$ por substituição para frente e encontrar x. Utilizaremos o raciocínio do exercício 4.

Encontrando x, teremos a nossa solução.

- código de implementação :

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main()
5 {
```

```
6      int m,n,k,i,j;
7      double somatorio = 0;
8
9      printf("Qual a dimensao m da matriz?\n");
10     scanf("%d",&m);
11     printf("Qual a dimensao n da matriz?\n");
12     scanf("%d",&n);
13
14     printf("Qual a matriz?(de prefer ncia uma matriz invers vel)\n");
15
16     double Matriz[m][n];
17
18     for(i = 0; i < m; i++)
19     {
20         for(j = 0; j < n; j++)
21         {
22             scanf("%lf",&Matriz[i][j]);
23         }
24     }
25
26     double MatrizTranposta[n][m];
27
28     for(i = 0; i < n; i++)
29     {
30         for(j = 0; j < m; j++)
31         {
32             MatrizTranposta[i][j] = Matriz[j][i];
33         }
34     }
35
36     double C[n][n];
37
38     for(i = 0; i < n; i++)
39     {
40         for(j = 0; j < n; j++)
41         {
42             C[i][j] = 0;
43         }
44     }
45
46     //Processo da multiplicacao de matrizes(A^T * A).
47
48     for(i = 0; i < n; i++)
49     {
50         for(j = 0; j < n; j++)
51         {
52             for(k = 0; k < m; k++)
53             {
54                 C[i][j] += MatrizTranposta[i][k] * Matriz[k][j];
55             }
56         }
57     }
58
```

```
59     printf("Matriz C:\n");
60
61     for(i = 0; i < n; i++)
62     {
63         for(j = 0; j < n; j++)
64         {
65             printf("%f ", C[i][j]);
66         }
67
68         printf("\n");
69     }
70
71     double fatorCholesky[n][n];
72
73     //Zerar os valores de memoria da variavel.("Inicializar" a variavel)
74     for(i = 0; i < n; i++)
75     {
76         for(j = 0; j < n; j++)
77         {
78             fatorCholesky[i][j] = 0;
79         }
80     }
81
82     //Passo 1: Determinar os primeiros elementos de cholesky (Formula 1):
83     //Obs: funcao pow(x,n) = x elevado a n.
84
85     fatorCholesky[0][0] = sqrt(C[0][0]);
86
87     for(j = 1; j < n; j++)
88     {
89         fatorCholesky[0][j] = C[0][j] / fatorCholesky[0][0];
90     }
91
92     //Passo 2: Calcular os elementos r[i][i] de Cholesky (Formula 2):
93
94     for(i = 1; i < n; i++)
95     {
96         for(k = 0; k < i; k++)
97         {
98             somatorio += pow(fatorCholesky[k][i], 2);
99         }
100
101         fatorCholesky[i][i] = C[i][i] - somatorio;
102         fatorCholesky[i][i] = sqrt(fatorCholesky[i][i]);
103         somatorio = 0; //zerar para iniciar o somat rio da proxima parcela.
104
105         //Com todos os elementos r[i][i] determinados, podemos calcular os r[i][j] (
106         //Formula 3):
107         for(j = i + 1; j < n; j++)
108         {
109             for(k = 0; k < i; k++)
110             {
111                 somatorio += fatorCholesky[k][i] * fatorCholesky[k][j];
```

```
111         }
112
113         fatorCholesky[i][j] = C[i][j] - somatorio;
114         fatorCholesky[i][j] = fatorCholesky[i][j] / fatorCholesky[i][i];
115         somatorio = 0; //zerar para iniciar o somatorio da proxima parcela.
116     }
117
118 }
119
120 //Mostrar o fator de Cholesky da matriz C
121 printf("Fator de Cholesky:\n");
122
123 for(i = 0; i < n; i++)
124 {
125     for(j = 0; j < n; j++)
126     {
127         printf("%f ", fatorCholesky[i][j]);
128     }
129     printf("\n");
130 }
131
132 //Passo para determinar um sistema Ax = b com a fatoracao de Cholesky
133
134 printf("Digite a matriz B para resolver um sistema Ax = B com Cholesky:\n")
135 ;
136 double B[n];
137
138 for(i = 0; i < n; i++)
139 {
140     scanf("%lf", &B[i]);
141 }
142
143 double x[n];
144 double y[n];
145 double transpostaCholesky[n][n];
146
147 //Determinar a transposta de Cholesky(R^T)
148 for(i = 0; i < n; i++)
149 {
150     for(j = 0; j < n; j++)
151     {
152         transpostaCholesky[i][j] = fatorCholesky[j][i];
153     }
154 }
155
156 //Resolvendo Ax = B —————> R^T * Rx = B:
157
158 //Primeira parte: R^T * y = B;
159 //Mesma logica usada na questao 5 da prova.
160
161 y[0] = B[0] / transpostaCholesky[0][0];
162
```



```
163     for (i = 0; i <= n - 1; i++)
164     {
165         for (j = 0; j <= i; j++)
166         {
167             if (j < i)
168             {
169                 B[i] -= transpostaCholesky[i][j] * y[j];
170             }
171             else
172             {
173                 B[i] = B[i] / transpostaCholesky[i][j];
174             }
175         }
176     }
177     y[i] = B[i];
178 }
179
180
181 //Segunda parte: Resolver Rx = y;
182 //Mesma logica usada na questao 4 da prova.
183
184 x[n - 1] = y[n - 1] / fatorCholesky[n - 1][n - 1];
185
186 for (i = n - 1; i >= 0; i--)
187 {
188     for (j = n - 1; j >= i; j--)
189     {
190         if (j > i)
191         {
192             y[i] -= fatorCholesky[i][j] * x[j];
193         }
194         else
195         {
196             y[i] = y[i] / fatorCholesky[i][j];
197         }
198     }
199 }
200
201 x[i] = y[i];
202 }
203
204 //Resultados
205 printf(" Solu o do sistema:\n");
206
207 for (i = 0; i < n; i++)
208 {
209     printf("x_%d = %f\n", i, x[i]);
210 }
211
212 return 0;
213 }
```

Listing 6: Decomposição de Cholesky

Exercício 7

Nesta questão a solução atribuída foi utilizar um controlador que checava os primeiros elementos de cada linha e de cada coluna simultaneamente, afim de ver se o primeiro elemento de cada diagonal era igual a zero.

Encontrando o primeiro, o programa guarda essa informação na variável "EhPivo0" e olha as diagonais referentes as diagonais que começam em 0,c e c,0. O programa segue em diante checando nesse padrão, caso encontre algum numero diferente de zero após entrar nesse estado ele volta ao estado anterior e continua olhando apenas os pivos das diagonais.

Se ao final o "EhPivo0" for igual a 1, quer dizer que a matriz é banda e com a variável "PrimeiraDiagonal0" é possível ver se é tridiagonal se o seu valor for 2, ou seja, a primeira diagonal zero é com o valor de $c = 2$, então é tridiagonal

- código de implementação :

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int n;
6     printf("Qual a ordem da matriz?\n");
7     scanf("%d", &n);
8
9     float A[n][n];
10    int i, j;
11
12    printf("Qual a matriz?\n");
13    for (i=0; i<n; i++)
14    {
15        for (j=0; j<n; j++)
16        {
17            scanf("%f", &A[i][j]);
18        }
19    }
20
21    int EhBanda = 1;
22    int c;
23    int EhPivo0 = 0;
24    int PrimeiraDiagonal0;
25
26    for (c=1; c<n; c++)
27    {
28        if (A[c][0] == 0.0 && A[0][c] == 0.0)
29        {
30            EhPivo0 = 1;
31            if (A[PrimeiraDiagonal0][0] != 0)
32                PrimeiraDiagonal0 = c;
33        }
```

```
34     else
35     {
36         EhPivo0 = 0;
37         PrimeiraDiagonal0 = -1;
38     }
39
40     if(EhPivo0)
41     {
42         if(c == n-1)
43         {
44             i = c;
45             j = 0;
46         }
47         else
48         {
49             i = c+1;
50             j = 1;
51         }
52         for (; i<n; i++)
53         {
54             if(A[i][j] != 0.0 || A[j][i] != 0.0)
55             {
56                 EhPivo0 = 0;
57                 PrimeiraDiagonal0 = -1;
58                 break;
59             }
60             j++;
61         }
62         if(EhPivo0)
63         {
64             continue;
65         }
66     }
67
68 }
69
70 if(EhPivo0)
71     EhBanda = 1;
72 else
73     EhBanda = 0;
74
75 if(EhBanda)
76 {
77     printf("e banda\n");
78     if(PrimeiraDiagonal0==2)
79         printf("tamb m e tridiagonal\n");
80 }
81 else
82 {
83     printf("nao e banda\n");
84 }
85
86
```

87 `}`

Listing 7: verifica se uma matriz é banda e se é tridiagonal

Exercício 8

Neste exercício usaremos os passos da eliminação de Gauss com pivotamento para encontrarmos as matrizes L (tringular inferior com toda a diagonal igual a 1) e U (triângular superior).

Basicamente a matriz L será composta pelos elementos encontrados por:

sendo k o controlador de colunas

$$m_{ik} = \frac{a_{ik}^k}{a_{kk}^k} \quad \forall i = k + 1, \dots, n$$

E a matriz U será a resultante da aplicação das seguintes fórmula :

$$a_{ij}^{k+1} = a_{ij}^k - m_{ik} \cdot a_{ij}^k \quad \forall i = k + 1, \dots, n; j = k + 1, \dots, n$$

A estrutura das matrizes serão:

$$A = LU \implies L = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ m_{21} & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ m_{n1} & \cdots & m_{nn-1} & 1 \end{bmatrix}, U = \begin{bmatrix} a_{11} & \cdots & \cdots & a_{1n} \\ 0 & a_{22} & \cdots & a_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{nn} \end{bmatrix}$$

Obtendo estes valores resolveremos o sistema em dois passos:

Passo 1 : Resolver $L \cdot y = b$ por substituição para frente e encontrar y. Utilizaremos o raciocínio do exercício 4.

Passo 2 : Resolver $U \cdot x = y$ por substituição para trás e encontrar x. Utilizaremos o raciocínio do exercício 5.

Encontrando x, teremos a nossa solução.

- código de implementação :

```
1 #include<stdio.h>
2
3 int main()
4 {
5     int n;
6     scanf("%d", &n);
7     float A[n][n], L[n][n], U[n][n], b[n];
8
9     int i, j, k;
```

```
10 //lendo A e montando a estrutura basica de L e U
11 for (i=0;i<n;i++)
12 {
13     for (j=0;j<n;j++)
14     {
15         scanf("%f", &A[i][j]);
16
17         if(i==j)
18             L[i][j] = 1;
19         if(i<j)
20             L[i][j] = 0;
21
22         U[i][j] = A[i][j];
23     }
24 }
25 float copiaB[n];
26 printf("\n");
27 //lendo o vetor B e criando uma copia para a prova real (apenas test case)
28 for (i=0;i<n;i++)
29 {
30     scanf("%f", &b[i]);
31     copiaB[i] = b[i];
32 }
33
34 //construindo as matrizes L e U
35 for (k=0;k<n-1;k++)
36 {
37     for (i=k+1;i<n;i++)
38     {
39         L[i][k] = U[i][k]/U[k][k];
40         U[i][k] = 0;
41         for (j=k+1;j<n;j++)
42         {
43             U[i][j] -= L[i][k]*U[k][j];
44         }
45     }
46 }
47
48 //printando L
49 for (i=0;i<n;i++)
50 {
51     for (j=0;j<n;j++)
52         printf("%f ", L[i][j]);
53     printf("\n");
54 }
55
56 printf("\n");
57
58 //printando U
59 for (i=0;i<n;i++)
60 {
61     for (j=0;j<n;j++)
62         printf("%f ", U[i][j]);
```

```
63     printf("\n");
64 }
65
66 printf("\n");
67
68 float x[n], y[n];
69 //calculando L*y=b substituicao pra tras
70 for(i= 0;i<n;i++)
71 {
72     for(j= 0 ;j<=i;j++)
73     {
74         if(j<i)
75         {
76             b[i] -= L[i][j]*y[j];
77         }
78         if(i==j)
79         {
80             b[i] /= L[i][j];
81         }
82     }
83     y[i] = b[i];
84 }
85
86 //calculando Ux = y por substituicao pra tras
87 for(i= n-1;i>=0;i--)
88 {
89     for(j= n-1 ;j>=i;j--)
90     {
91         if(j>i)
92         {
93             y[i] -= U[i][j]*y[j];
94         }
95         if(i==j)
96         {
97             y[i] /= U[i][j];
98         }
99     }
100     x[i] = y[i];
101 }
102
103 /* //printando b
104 for(i=0;i<n;i++)
105     printf("%f\n", b[i]);*/
106
107 printf("\no resultado de Ax=b : \n");
108
109 for(i=0;i<n;i++)
110     printf("%f\n", x[i]);
111 printf("\n");
112
113 //teste de prova real que aplica x matriz A original (apenas test case)
114 /* for(i=0;i<n;i++)
115 {
```

```
116         printf("%f", copiaB[i]);
117         float acumulador = 0;
118         for (j=0;j<n;j++)
119             acumulador += A[i][j]*x[j];
120         printf("%f\n", acumulador);
121     }*/
122
123
124
125 }
```

Listing 8: Decomposição LU

Exercício 9

Nesta questão foram necessários algumas "partes" para a sua resolução:

Função Norma1 :

Esta função calcula a norma 1 de um vetor de dimensão n usando a fórmula:

$$\sum_{i=0}^n |x_i|$$

Função Norma2 :

Esta função calcula a norma 2 de um vetor de dimensão n usando a fórmula:

$$\sqrt{x_0^2 + x_1^2 + \dots + x_{i-1}^2 + x_i^2}$$

Função NormaInfinito :

Esta função calcula a norma infinito de um vetor de dimensão n usando a fórmula:

$$\max |x_i|$$

Função DistanciaEntreDoisVetores :

Esta função calcula a distância entre o vetor x e o vetor y tendo a possibilidade de escolher qual norma será aplicada (chamando as funções anteriores) na fórmula:

$$\|x - y\|$$

função ProdutoInterno :

Esta função calcula o produto interno entre dois vetores simbolizado por $x \cdot y$ com a seguinte fórmula:

$$\sum_{i=0}^n x_i \cdot y_i$$

função CosDoAnguloEntreDoisVetores :

Esta função calcula o cosseno do ângulo entre os vetores x e y no qual será aplicado a função "acos" (arccosseno) da biblioteca "math.h" a qual retornará o valor do ângulo o qual possui este cosseno. A formula usada no cálculo do cosseno :

$$\cos \theta = \frac{x \cdot y}{\|x\|_2 \cdot \|y\|_2}$$

sempre utilizando a norma 2

Função NormaFrobenius :

Esta função calcula a norma de frobenius de uma matriz $A_{r \times c}$ usando a fórmula:

$$\sqrt{\sum_{i=0}^r \sum_{j=0}^c x_{ij}^2}$$

Função Norma1Matriz :

Esta função calcula a norma 1 de uma matriz $A_{r \times c}$ utilizando o seguinte recurso:

considere "result[r]" um vetor de dimensão "r" com o valor temporário da norma'

$$result[i] = \sum_{j=0}^c a_{ij}, \forall i = 0, 1, \dots, r-1, r$$

em seguida aplica-se a função NormaInfinito para obter o resultado

função NormaInfinitoMatriz :

Esta função calcula a norma infinito de uma matriz $A_{r \times c}$ utilizando o seguinte recurso:

considere "result[c]" um vetor de dimensão "c" com o valor temporário da norma'

$$result[j] = \sum_{i=0}^r a_{ij}, \forall j = 0, 1, \dots, c-1, c$$

em seguida aplica-se a função NormaInfinito para obter o resultado

função DistanciaEntreDuasMatrizes :

Esta função calcula a distância entre duas matrizes A_{rc} e B_{rc} tendo a possibilidade de escolher qual norma será aplicada (chamando as funções anteriores referente as matrizes) na fórmula:

considere "result[r][c]" uma matriz com a subtração de $A - B$

$$\|result\|$$

Com essas fórmulas é possível resolver as questões pedidas no exercício aplicando-as no programa principal

- Código de implementação :

```
1 #include <stdio.h>
2 #include <math.h>
3
4 //calcula a norma 1 de um vetor de tamanho n
5 float Normal(float x[], int n)
6 {
7     int i;
8     float somatorio = 0;
9     for(i=0;i<n;i++)
10     {
11         if(x[i] < 0)
12             x[i] *= -1;
13         somatorio += x[i];
14     }
15
16     return somatorio;
17 }
18
19 //calcula a norma 2 de um vetor de tamanho n
20 float Norma2(float x[], int n)
21 {
22     int i;
23     float somatorio = 0;
24
25     for(i=0;i<n;i++)
26     {
27         somatorio += pow(x[i], 2);
28     }
29
30     return sqrt(somatorio);
31 }
32
33 //calcula a norma infinito de um vetor de tamanho n
34 float NormaInfinito(float x[], int n)
35 {
36     float Maior = 0;
```

```
37     int i;
38
39     for (i=0; i<n; i++)
40     {
41         if (x[i]<0)
42             x[i] *= -1;
43
44         if (i==0)
45             Maior = x[i];
46         else
47         {
48             if (x[i] > Maior)
49                 Maior = x[i];
50         }
51     }
52
53     return Maior;
54 }
55
56 //calcula a distancia entre dois vetores x e y de tamanho n permitindo escolher
57 //a norma
58 float DistanciaEntreDoisVetores(float x[], float y[], int n, char norma)
59 {
60     float result[n];
61     int i;
62     for (i=0; i<n; i++)
63         result[i] = x[i] - y[i];
64
65     switch(norma)
66     {
67         case '1':
68             return Normal(result, n);
69         case '2':
70             return Norma2(result, n);
71         case 'i':
72             return NormaInfinito(result, n);
73     }
74 }
75 //calcula o produto interno entre dois vetores x e y de tamanho n
76 float ProdutoInterno(float x[], float y[], int n)
77 {
78     float somatorio =0;
79     int i;
80     for (i=0; i<n; i++)
81     {
82         somatorio += x[i]*y[i];
83     }
84
85     return somatorio;
86 }
87 //retorna o cos do angulo entre dois vetores
88 float CosDoAnguloEntreDoisVetores(float x[], float y[], int n)
```

```
89 {
90     return ProdutoInterno(x, y, n)/(Norma2(x,n)*Norma2(y,n));
91 }
92
93 //calcula a norma de fobenius (usando um ponteiro para referir a matriz Arxc
94 //devido a questoes da linguagem)
95 float NormaFrobenius(float* matrizA, int r, int c)
96 {
97     int i, j;
98     float somatorio = 0;
99
100     float A[r][c];
101     int k = 0;
102     for(i=0;i<r;i++)
103         for(j=0;j<c;j++)
104             {
105                 A[i][j] = *(matrizA + k); //equivale a fazer A[i][j] = matrizA[k]
106                 k++;
107             }
108
109     for(i=0;i<r;i++)
110         for(j=0;j<c;j++)
111             somatorio += pow(A[i][j], 2);
112
113     return sqrt(somatorio);
114 }
115
116 //calcula a norma 1 de uma matriz A (referenciada por matrizA) de tamanho rxc
117 float NormalMatriz(float* matrizA, int r, int c)
118 {
119     float result[r], somatorio;
120     int i, j;
121
122     float A[r][c];
123     int k = 0;
124     for(i=0;i<r;i++)
125         for(j=0;j<c;j++)
126             {
127                 A[i][j] = *(matrizA + k);
128                 k++;
129             }
130
131     for(i=0;i<r;i++)
132     {
133         somatorio = 0; //iniciando com 0
134         for(j=0;j<c;j++)
135             {
136                 if(A[i][j]<0)
137                     A[i][j] *= -1; // modulo
138
139                 somatorio += A[i][j];
140             }
141         result[i] = somatorio; //vetor resultado tem a soma dos elementos em
142                                 //cada linha
143     }
```

```
140     }
141
142     return NormaInfinito(result, r); //e aqui usamos a norma infinito nesse
        vetor com as somas
143 }
144
145 //calcula a norma infinito da matriz A (referenciada por matrizA) de tamanho
        rxc
146 float NormaInfinitoMatriz(float* matrizA, int r, int c)
147 {
148     float result[c], somatorio;
149     int i, j;
150
151     float A[r][c];
152     int k = 0;
153     for(i=0; i<r; i++)
154         for(j=0; j<c; j++)
155         {
156             A[i][j] = *(matrizA + k);
157             k++;
158         }
159
160     for(j=0; j<c; j++)
161     {
162         somatorio = 0; //iniciando com 0
163         for(i=0; i<r; i++)
164         {
165             if(A[i][j]<0)
166                 A[i][j] *= -1; //modulo
167
168             somatorio += A[i][j]; //somatorio
169         }
170         result[j] = somatorio; // resultado tem todos os somatorios dos
            elementos de todas as colunas
171     }
172     return NormaInfinito(result, c); // e aqui usamos a norma infinito do vetor
        com as somas
173 }
174
175 //calculando a distancia entre duas matrizes A e B (permitindo escolher a norma
        )
176 float DistanciaEntreDuasMatrizes(float* matrizA, float* matrizB, int r, int c,
        char norma)
177 {
178     float result[r][c];
179     int i, j;
180
181     float A[r][c], B[r][c];
182     int k = 0;
183     for(i=0; i<r; i++)
184         for(j=0; j<c; j++)
185         {
186             A[i][j] = *(matrizA + k);
```

```
187         B[i][j] = *(matrizB + k);
188         k++;
189     }
190
191     for (i=0; i<r; i++)
192         for (j=0; j<c; j++)
193             result[i][j] = A[i][j] - B[i][j]; //subtracao das matrizes elemento
194                                             a elemento
195
196     switch(norma)
197     {
198         case '1':
199             return NormalMatriz(&result[0][0], r, c); //chamando a funcao para a
200                                             norma
201         case 'f':
202             return NormaFrobenius(&result[0][0], r, c); //chamando a funcao
203                                             para a norma
204         case 'i':
205             return NormaInfinitoMatriz(&result[0][0], r, c); //chamando a funcao
206                                             para a norma
207     }
208 }
209
210 int main()
211 {
212     //lendo tamanho do vetor
213     int n;
214     printf("Digite a dimensao do vetor x: ");
215     scanf("%d", &n);
216
217     float x[n];
218     //lendo vetor x
219     int i, j;
220     printf("Digite o vetor x: ");
221     for (i=0; i<n; i++)
222         scanf("%f", &x[i]);
223
224     //calculando as normas do vetor
225     printf("norma 1 de x: %f\n", Normal(x, n));
226     printf("norma 2 de x: %f\n", Norma2(x, n));
227     printf("norma infinito de x: %f\n", NormaInfinito(x, n));
228     printf("\n");
229
230     //lendo o vetor y
231     float y[n];
232     printf("Digite um vetor y de dimensao %d", n);
233     for (i=0; i<n; i++)
234         scanf("%f", &y[i]);
235
236     //calculando as distancias entre x e y variando as normas
237     printf("distancia entre x e y com norma 1: %f\n", DistanciaEntreDoisVetores
238         (x, y, n, '1'));
239     printf("distancia entre x e y com norma 2: %f\n", DistanciaEntreDoisVetores
```

```

235     (x, y, n, '2') );
printf("distancia entre x e y com norma infinito: %f\n",
    DistanciaEntreDoisVetores(x, y, n, 'i') );
236 printf("\n");
237
238 //calculando o produto interno e o angulo entre os vetores
239 printf("produto interno entre dois vetores : %f\n", ProdutoInterno(x, y, n)
    );
240 printf("angulo entre x e y : %f\n", acos(CosDoAnguloEntreDoisVetores(x, y,
    n)) );
241 printf("\n");
242
243 //repetindo todo o processo para as matrizes (exceto o angulo e o produto
    interno)
244 int r, c;
245 //forçando que a matriz seja quadrada
246 printf("Digite a dimencao de A e B: ");
247 scanf("%d", &r);
248 c = r;
249 float A[r][c], B[r][c];
250
251 printf("Digite a matriz A: ");
252 for(i=0; i<r; i++)
253     for(j=0; j<c; j++)
254         scanf("%f", &A[i][j]);
255
256 printf("Digite a matriz B: ");
257 for(i=0; i<r; i++)
258     for(j=0; j<c; j++)
259         scanf("%f", &B[i][j]);
260
261
262 printf("norma 1 de A: %f\n", Norma1Matriz(&A[0][0], r, c) );
263 printf("norma 1 de B: %f\n", Norma1Matriz(&B[0][0], r, c) );
264
265 printf("\n");
266
267 printf("norma frobenius de A: %f\n", NormaFrobenius(&A[0][0], r, c) );
268 printf("norma frobenius de B: %f\n", NormaFrobenius(&B[0][0], r, c) );
269
270 printf("\n");
271
272 printf("norma infinito de A: %f\n", NormaInfinitoMatriz(&A[0][0], r, c) );
273 printf("norma infinito de B: %f\n", NormaInfinitoMatriz(&B[0][0], r, c) );
274
275 printf("\n\n");
276
277 printf("distancia entre A e B com norma 1: %f\n",
    DistanciaEntreDuasMatrizes(&A[0][0], &B[0][0], r, c, '1') );
278 printf("distancia entre A e B com norma de Forbenius: %f\n",
    DistanciaEntreDuasMatrizes(&A[0][0], &B[0][0], r, c, 'f') );
279 printf("distancia entre A e B com norma infinito: %f\n",
    DistanciaEntreDuasMatrizes(&A[0][0], &B[0][0], r, c, 'i') );

```

```
280
281 }
```

Listing 9: cálculo de normas e aplicação em vetores e matrizes

Exercício 10

Para o cálculo do número condição da matriz inversa nas normas 1, infinito e frobenius, iremos receber do matlab os números condições de cada norma (em relação a matriz A) e a própria matriz A.

Com essas informações, poderemos iniciar nossos cálculos...

Inicialmente pediremos qual será a ordem da matriz A que foi utilizada e em seguida os valores da matriz A utilizada no matlab para a obtenção dos números condições das normas. Em seguida, solicitará os respectivos números condições.

Antes de calcularmos a norma da matriz inversa em si, calcularemos os valores das normas da matriz A em cada norma solicitada (1, infinito e frobenius). No código da questão, foram inseridas funções que irão retornar estes valores. A lógica dessas funções já fora implementada em questões anteriores e apenas reaproveitamos o feito e a transformamos em funções para evitar reescrita de código e para deixar o código mais limpo.

Depois de calculada as normas na matriz A, finalmente poderemos obter o que desejamos. Para isso, aplicaremos a fórmula do número condição de A em qualquer norma:

$$\text{cond}(A) = \|A\| \cdot \|A^{-1}\|$$

Isolando $\|A^{-1}\|$, teremos:

$$\|A^{-1}\| = \frac{\text{cond}(A)}{\|A\|}$$

Assim obtemos a norma da matriz inversa.

Feito os cálculos, mostraremos os resultados na tela e assim finalizamos nossos cálculos.

- Código de implementação :

```
1 #include <stdio.h>
2 #include <math.h>
3
4 double Normal(double* enderecoMatriz, int ordemMatriz)
5 {
6     int i, j, k = 0; //variaveis de iteracao
7     double somatorio = 0; //variavel para realizar o somatorio
```

```
8      double normal = 0; //valor da norma 1
9      double Matriz[ordemMatriz][ordemMatriz];
10
11     //Necessario para passarmos os elementos da matriz A
12     for(i = 0; i < ordemMatriz; i++)
13     {
14         for(j = 0; j < ordemMatriz; j++)
15         {
16             Matriz[i][j] = *(enderecoMatriz + k);
17             k++;
18         }
19     }
20
21     //Logica da norma 1: Maior soma dos elementos em modulo de cada coluna.
22     for(i = 0; i < ordemMatriz; i++)
23     {
24         for(j = 0; j < ordemMatriz; j++)
25         {
26             somatorio += fabs(Matriz[j][i]);
27         }
28
29         if(normal < somatorio)
30         {
31             normal = somatorio;
32         }
33
34         somatorio = 0; //zera-se para iniciar a soma de uma nova coluna.
35     }
36
37     return normal;
38 };
39
40 double NormaInfinito(double* enderecoMatriz, int ordemMatriz)
41 {
42     int i, j, k = 0;
43     double somatorio = 0;
44     double normaInfinito = 0;
45     double Matriz[ordemMatriz][ordemMatriz];
46
47     for(i = 0; i < ordemMatriz; i++)
48     {
49         for(j = 0; j < ordemMatriz; j++)
50         {
51             Matriz[i][j] = *(enderecoMatriz + k);
52             k++;
53         }
54     }
55
56     //Logica da norma infinito: Maior soma dos elementos em modulo de cada
57     linha
58     for(i = 0; i < ordemMatriz; i++)
59     {
60         for(j = 0; j < ordemMatriz; j++)
```



```
60     {
61         somatorio += fabs(Matriz[i][j]);
62     }
63
64     if(normaInfinito < somatorio)
65     {
66         normaInfinito = somatorio;
67     }
68
69     somatorio = 0;
70 }
71
72 return normaInfinito;
73 };
74
75
76 double NormaFrobenius(double* enderecoMatriz, int ordemMatriz)
77 {
78     int i, j, k = 0;
79     double somatorio = 0;
80     double normaFrobenius = 0;
81     double Matriz[ordemMatriz][ordemMatriz];
82
83     for(i = 0; i < ordemMatriz; i++)
84     {
85         for(j = 0; j < ordemMatriz; j++)
86         {
87             Matriz[i][j] = *(enderecoMatriz + k);
88             k++;
89         }
90     }
91
92     //Logica da norma Frobenius: A raiz quadrada da soma dos elementos da
93     //matriz em modulo ao quadrado.
94     //Obs: pow(x,n) = funcao que retorna x elevado n.
95     for(i = 0; i < ordemMatriz; i++)
96     {
97         for(j = 0; j < ordemMatriz; j++)
98         {
99             somatorio += pow(fabs(Matriz[i][j]), 2);
100         }
101     }
102     normaFrobenius = sqrt(somatorio);
103
104     return normaFrobenius;
105 };
106
107 int main()
108 {
109     int i, j, n;
110     double numeroCondicaoNormal, numeroCondicaoNormaFrobenius,
        numeroCondicaoNormaInfinito;
```

```

111
112     printf("Qual a ordem da matriz A?\n");
113     scanf("%d",&n);
114
115     double A[n][n];
116
117     printf("Qual a matriz A?\n");
118
119     for(i = 0; i < n; i++)
120     {
121         for(j = 0; j < n; j++)
122         {
123             scanf("%lf",&A[i][j]);
124         }
125     }
126
127     printf("Qual o nmero condi o na norma 1 de A(retirado do matlab)?\n");
128     scanf("%lf",&numeroCondicaoNormal);
129
130     printf("Qual o nmero condi o na norma infinito de A(retirado do matlab
131             )?\n");
132     scanf("%lf",&numeroCondicaoNormaInfinito);
133
134     printf("Qual o nmero condi o na norma de frobenius de A(retirado do
135             matlab)?\n");
136     scanf("%lf",&numeroCondicaoNormaFrobenius);
137
138     //Calculo das normas da matriz inversa
139
140     double matrizInversaNormal, matrizInversaNormaFrobenius,
141           matrizInversaNormaInfinito;
142     double normalA, normaFrobeniusA, normaInfinitoA;
143
144     //Necessario para enviar a matriz A para as funcoes que calcularao as
145     //normas.
146     /*Obs: Nao eh possivel realizar a passagem de matriz de ordem nao constante
147     por passegem por valor,
148     apenas por referencia, por isso estamos usando ponteiros para realizar a
149     passagem por referencia.*/
150     double* enderecoA;
151     enderecoA = &A[0][0];
152
153     //Calculando as normas da matriz A.
154     normalA = Normal(enderecoA,n);
155     normaFrobeniusA = NormaFrobenius(enderecoA,n);
156     normaInfinitoA = NormaInfinito(enderecoA,n);
157
158     //Calculando as normas da matriz inversa, utilizando a formula: ||A- || =
159     //cond(A) / ||A||
160
161     matrizInversaNormal = numeroCondicaoNormal / normalA;
162     matrizInversaNormaFrobenius = numeroCondicaoNormaFrobenius /
163           normaFrobeniusA;

```

```
156     matrizInversaNormaInfinito = numeroCondicaoNormaInfinito / normaInfinitoA;
157
158     // Resultados
159
160     printf("Matriz inversa:\n");
161
162     printf("Norma 1 = %f\n", matrizInversaNormal);
163     printf("Norma de Frobenius = %f\n", matrizInversaNormaFrobenius);
164     printf("Norma infinito = %f\n", matrizInversaNormaInfinito);
165
166     return 0;
167 }
```

Listing 10: cálculo de $\|A^{-1}\|$ em função do número condição da matriz A

Exercício 11

Para o cálculo do limitante inferior do número condição, precisaremos da receber no programa a ordem da matriz, a matriz, sua inversa (retirada do matlab) e o valor de ω (um vetor qualquer diferente do vetor nulo).

O resultado final do programa irá retornar um valor aproximado do número condição. Uma estimativa, em exatidão; ou seja, o limitante inferior do número condição.

Para calcular o limitante inferior, segue-se a fórmula:

$$\text{cond}(A) \leq \frac{\|A\| \cdot \|A^{-1} \cdot \omega\|}{\|\omega\|}$$

A qual o programa irá usá-la para o cálculo.

Primeiramente, iremos pegar o vetor ω e multiplicá-lo com a matriz inversa, usando no algoritmo o mesmo raciocínio de multiplicação de um vetor com matriz da questão 1 da prova. Ratificando, o resultado da multiplicação será um vetor.

Finalizando, calcularemos a norma da matriz, a norma do vetor resultado da multiplicação e a norma do vetor ω . Para estes cálculos, utilizamos funções que calculem essas normas, como já vistas e aplicadas em exercícios anteriores da prova. Depois, aplicaremos a fórmula do limitante inferior e encontraremos o resultado.

Neste exercício, aplicamos os cálculos na norma 1 e na norma infinito, por maior facilidade de manipulação e de cálculo.

- Código de implementação :

```
1 #include <stdio.h>
2 #include <math.h>
3
4 //Funcoes que calculam as normas de vetor ou matriz.
5
6 double Norma1Vetor(double* vetor, int tamanhoVetor)
7 {
8     double normal = 0; // resultado da norma 1
9     int i = 0; //variavel de iteracao
10
11     //Logica da norma 1 de vetor: resultado da soma de todos os elementos em
12     // modulo do vetor
13     for(i = 0; i < tamanhoVetor; i++)
14     {
15         normal += fabs(vetor[i]);
16     }
17
18     return normal;
19 };
20
21 double NormaInfinitoVetor(double* vetor, int tamanhoVetor)
22 {
23     double normaInfinito = 0;
24     int i = 0;
25
26     //Logica da norma infinito: Maior elemento em modulo do vetor.
27     for(i = 0; i < tamanhoVetor; i++)
28     {
29         if(fabs(vetor[i]) > normaInfinito)
30         {
31             normaInfinito = fabs(vetor[i]);
32         }
33     }
34
35     return normaInfinito;
36 };
37
38 //Mesma funcao usada na questao 10
39 double NormaMatriz(double* enderecoMatriz, int ordemMatriz)
40 {
41     int i, j, k = 0;
42     double somatorio = 0;
43     double normal = 0;
44     double Matriz[ordemMatriz][ordemMatriz];
45
46     //Necessario para a funcao receber todos os elementos da matriz passada
47     // como parametro.
48     for(i = 0; i < ordemMatriz; i++)
49     {
50         for(j = 0; j < ordemMatriz; j++)
51         {
52             Matriz[i][j] = *(enderecoMatriz + k);
53             k++;
54         }
55     }
56 }
```

```
52     }
53 }
54
55 for(i = 0; i < ordemMatriz; i++)
56 {
57     for(j = 0; j < ordemMatriz; j++)
58     {
59         somatorio += fabs(Matriz[j][i]);
60     }
61
62     if(normal < somatorio)
63     {
64         normal = somatorio;
65     }
66
67     somatorio = 0; //necessario para o somatorio ser reutilizado.
68 }
69
70 return normal;
71 };
72
73 //Mesma funcao usada na questao 10
74 double NormaInfinitoMatriz(double* enderecoMatriz, int ordemMatriz)
75 {
76     int i, j, k = 0;
77     double somatorio = 0;
78     double normaInfinito = 0;
79     double Matriz[ordemMatriz][ordemMatriz];
80
81     for(i = 0; i < ordemMatriz; i++)
82     {
83         for(j = 0; j < ordemMatriz; j++)
84         {
85             Matriz[i][j] = *(enderecoMatriz + k);
86             k++;
87         }
88     }
89
90     for(i = 0; i < ordemMatriz; i++)
91     {
92         for(j = 0; j < ordemMatriz; j++)
93         {
94             somatorio += fabs(Matriz[i][j]);
95         }
96
97         if(normaInfinito < somatorio)
98         {
99             normaInfinito = somatorio;
100         }
101
102         somatorio = 0;
103     }
104 }
```

```
105     return normaInfinito;
106 };
107
108 int main()
109 {
110     int i,j,n;
111     double numeroCondicaoAproximadoNormal, numeroCondicaoAproximadoNormaInfinito;
112
113     printf("Qual a ordem da matriz?\n");
114     scanf("%d",&n);
115
116     double Matriz[n][n];
117
118     printf("Qual a matriz?\n");
119
120     for(i = 0; i < n; i++)
121     {
122         for(j = 0; j < n; j++)
123         {
124             scanf("%lf",&Matriz[i][j]);
125         }
126     }
127
128     double MatrizInversa[n][n];
129
130     printf("Entre com a matriz inversa obtida no Matlab\n");
131
132     for(i = 0; i < n; i++)
133     {
134         for(j = 0; j < n; j++)
135         {
136             scanf("%lf",&MatrizInversa[i][j]);
137         }
138     }
139
140     double w[n]; //vetor w qualquer
141
142     printf("Entre com um vetor w qualquer, com exceção do vetor nulo, para o cálculo do limitante inferior\n");
143
144     for(i = 0; i < n; i++)
145     {
146         scanf("%lf",&w[i]);
147     }
148
149     //Multiplica a matriz inversa com o vetor w
150     //Mesmo raciocínio retirado da questão número 1.
151
152     double MatrizResultadoInversaW[n]; //Resultado da multiplicação entre a
153         inversa e w.
154
155     //Zerar todos os elementos para eliminar os números de memória contidos na
```

```

155     variavel.
156     for (j = 0; j < n; j++)
157     {
158         MatrizResultadoInversaW[j] = 0;
159     }
160     for (i = 0; i < n; i++)
161     {
162         for (j = 0; j < n; j++)
163         {
164             MatrizResultadoInversaW[i] += MatrizInversa[i][j] * w[j];
165         }
166     }
167
168     //Variaveis de endereco necessarias para serem enviadas para as funcoes
169     //para o calculo das normas.
170     double* enderecoMatriz = &Matriz[0][0];
171     double* enderecoMatrizInversaW = &MatrizResultadoInversaW[0];
172     double* enderecoW = &w[0];
173
174     //Calculando o limitante inferior do n mero condicao, de acordo com a
175     //formula dada 2.2.28 no livro.
176
177     numeroCondicaoAproximadoNormal = NormalMatriz(enderecoMatriz, n) *
178     NormalVetor(enderecoMatrizInversaW, n) / NormalVetor(enderecoW, n);
179     numeroCondicaoAproximadoNormaInfinito = NormaInfinitoMatriz(enderecoMatriz,
180     n) * NormaInfinitoVetor(enderecoMatrizInversaW, n) / NormaInfinitoVetor(
181     enderecoW, n);
182
183     //Resultados
184
185     printf("O n mero condi o maior ou igual a %f na norma 1\n",
186     numeroCondicaoAproximadoNormal);
187     printf("O n mero condi o maior ou igual a %f na norma infinito\n",
188     numeroCondicaoAproximadoNormaInfinito);
189
190     return 0;
191 }

```

Listing 11: cálculo do limitante inferior de uma matriz $A_{n \times n}$

Exercício 12

Neste exercício, iremos pedir para que o usuário insira a ordem da matriz do sistema, a matriz do sistema, o vetor b do sistema, o valor de \hat{x} (solução aproximada) obtido do matlab e o número condição da matriz do sistema na norma 2, pois aqui trabalharemos com a norma 2.

Nosso objetivo será aplicar a fórmula do exemplo 2.4.2 seguinte:

$$\frac{\text{cond}(A, 2) \cdot \|\hat{r}\|_2}{\|b\|_2}$$

Sendo que esse resultado será um valor que comparará o nível da aproximação da solução, através do seguinte contexto:

$$\frac{\|x - \hat{x}\|_2}{\|x\|_2} \leq \frac{\text{cond}(A, 2) \cdot \|\hat{r}\|_2}{\|b\|_2}$$

Quanto menor o valor da equação a direita, maior será a proximidade da solução aproximada com a real.

Seguindo os passos do exemplo, primeiro devemos encontrar o valor de \hat{r} (resíduo chapéu), de forma que:

$$\hat{r} = b - A \cdot \hat{x}.$$

Seguindo isso, o programa antes calculará a multiplicação entre a matriz A do sistema e o \hat{x} do MATLAB. Obtido isso, efetuaremos. $b - A \cdot \hat{x}$.

Com o número condição em mãos e o valor de \hat{r} , podemos calcular o nosso objetivo com o antepasso de calcular a norma 2 de \hat{r} e b , usando as mesma função para essa finalidade no programa. Apenas aplicaremos nossa fórmula e obteremos o resultado. Exibiremos o resultado da conta.

Em adição, a este exercício, foi adicionado um critério de precisão aceitável que o usuário deseja. O programa pedirá esse valor e depois fará a comparação com o resultado obtido anteriormente. Se o desejo do usuário for menor ou igual ao valor encontrado, a solução aproximada será uma boa aproximação, caso contrário, a mesma não poderá ser aceita.

Observao: A aproximação do resultado do programa se resume até a 6 casas decimais. Caso o resultado seja zero, podemos dizer que o valor encontrado ultrapassou a representação de 6 decimais e por isso se arredonda para o valor zero.

Caso você deseja obter um valor diferente de zero, escreva no programa os valores com representações de at 6 casas decimais.

- Código de implementação :

```
1 #include <stdio.h>
2 #include <math.h>
3
4 //Funcao da norma 2 de um vetor ja aplicada em exercicios anteriores.
```



```
5 double Norma2(double* vetor,int tamanhoVetor)
6 {
7     double norma2 = 0;
8     int i = 0;
9
10    for(i = 0;i < tamanhoVetor;i++)
11    {
12        norma2 += pow(fabs(vetor[i]),2);
13    }
14
15    norma2 = sqrt(norma2);
16
17    return norma2;
18 };
19
20 int main()
21 {
22     int i,j,ordem;
23
24     printf("Qual a ordem do sistema?\n");
25     scanf("%d",&ordem);
26
27     double A[ordem][ordem];
28
29     printf("Qual a matriz do sistema?\n");
30     for(i = 0; i < ordem;i++)
31     {
32         for(j = 0;j < ordem;j++)
33         {
34             scanf("%lf",&A[i][j]);
35         }
36     }
37
38     double b[ordem];
39
40     printf("Qual o vetor b do sistema?\n");
41     for(i = 0;i < ordem;i++)
42     {
43         scanf("%lf",&b[i]);
44     }
45
46     double xhat[ordem];
47
48     printf("Qual a solu o aproximada obtida no matlab?\n");
49
50     for(i = 0;i < ordem;i++)
51     {
52         scanf("%lf",&xhat[i]);
53     }
54
55     double numeroCondicaoNorma2;
56
57     printf("Qual o numero condi o da norma 2,obtido no matlab, da matriz do
```

```
        sistema?\n");
58 scanf("%lf",&numeroCondicaoNorma2);
59
60 //Enderecos da matriz e vetor para serem passados para a funcao.
61 double* enderecoB = &b[0];
62 double* enderecoA = &A[0][0];
63
64 //Inicio dos calculos
65
66 double residuoHat[ordem];
67
68 double vetorResultado[ordem];
69
70 //Calculo do residuohat.
71
72 //Calculando A * xhat
73
74 //Multiplicacao de matriz com vetor
75
76 for(j = 0;j < ordem;j++)
77 {
78     vetorResultado[j] = 0;
79 }
80
81 for(i = 0;i < ordem;i++)
82 {
83     for(j = 0;j < ordem;j++)
84     {
85         vetorResultado[i] += A[i][j] * xhat[j];
86     }
87 }
88
89 //Calculo final do residuohat rhat = b - A*xhat
90
91 for(i = 0;i < ordem;i++)
92 {
93     residuoHat[i] = b[i] - vetorResultado[i];
94 }
95
96 //Mesma ideia para o enderecoA e enderecoB.
97 double* enderecoResiduohat = &residuoHat[0];
98
99 //Calculo da proximidade de xhat da solucao real(a sua precisao de acordo
    com a solucao real)
100 double precisaoXhatNorma2;
101
102 //Aplicacao da formula.
103 precisaoXhatNorma2 = (numeroCondicaoNorma2 * Norma2(enderecoResiduohat ,
    ordem)) / Norma2(enderecoB , ordem);
104
105 //Avaliacao do resultado.
106
107 printf("Precisao na norma 2 da solucao x aproximada: %f\n",
```

```

108         precisaoXhatNorma2);
109     //O usuario pode inserir um valor de precisao aceitavel para avaliar se a
110     solucao aproximada eh boa ou nao para
111     //ele.
112     double nivelPrecisaoAceitavel;
113
114     printf("Insira um valor de precisao, que ao seu ver, bom para a
115           solucao aproximada:\n");
116     scanf("%lf",&nivelPrecisaoAceitavel);
117
118     //Analise das precisoes
119     //Se o nivel de precisao encontrado for menor ou igual ao desejado, a
120     solucao eh adequada.
121     if(precisaoXhatNorma2 <= nivelPrecisaoAceitavel)
122         printf("Usando a norma 2, a solucao aproximada e uma boa
123               aproximacao para seu caso\n");
124     else
125         printf("Usando a norma 2, a solucao aproximada e uma ruim
126               aproximacao para seu caso\n");
127
128     return 0;
129 }

```

Listing 12: cálculo do resíduo da solução de um sistema

Exercício 13

Nesta questão a solução atribuída foi transpor a matriz A e multiplicar $A^T \cdot A$ comparando termo a termo com a matriz identidade. Se apenas um deles fossem diferente já seria possível dizer que a matriz A não é ortogonal

- Código de implementação :

```

1  #include<stdio.h>
2
3  int main()
4  {
5      int n;
6      printf("Qual a ordem da matriz?\n");
7      scanf("%d", &n);
8
9      float A[n][n], At[n][n];
10     int i, j;
11     printf("Qual a matriz A?\n");
12     for (i=0;i<n;i++)
13         for (j=0;j<n;j++)
14             {
15                 scanf("%f", &A[i][j]);

```

```
16         At[j][i] = A[i][j];
17     }
18
19     int EhOrtogonal = 1;
20
21     float Result[n][n];
22     for(i = 0; i < n ; i++)
23     {
24         int l;
25         for(l = 0; l < n ; l++)
26         {
27             float somatorio = 0;
28
29             for(j = 0; j < n ; j++)
30             {
31                 somatorio += A[i][j]*At[j][l];
32             }
33             Result[i][l] = somatorio;
34         }
35     }
36
37
38     for(i=0;i<n;i++)
39     {
40         for(j=0;j<n;j++)
41         {
42             if(i==j)
43             {
44                 if(Result[i][j]!=1)
45                 {
46                     EhOrtogonal = 0;
47                     break;
48                 }
49             }
50             else
51             {
52                 if(Result[i][j]!=0)
53                 {
54                     EhOrtogonal = 0;
55                     break;
56                 }
57             }
58         }
59         if(EhOrtogonal == 0)
60             break;
61     }
62
63     if(EhOrtogonal)
64         printf("    ortogonal\n");
65     else
66         printf(" n o    ortogonal\n");
67
68 }
```

Listing 13: Avaliar se uma matriz é ortogonal

Exercício 14

Nesta questão a solução atribuída foi escolher os valores para os rotadores arbitrariamente, de forma que o vetor rotator x_2 tivesse todos as suas posições iguais a 1, que para uma matriz 2x2 (especificada no exercício) gera a matriz Q ortogonal, que possuirá a seguinte forma:

$$Q = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \Rightarrow \frac{\sqrt{2}}{2} \cdot \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$$

A matriz R é gerada através da fórmula: $Q \cdot A$ (matriz a qual se quer fatorar).

Agora que a fatoração está completa, é possível achar a solução do sistema fazendo:

$$A = Q \cdot R \implies Q \cdot R \cdot x = b$$

chame $R \cdot x = y$, então para achar a solução, fizemos:

$$Q \cdot y = b \implies y = Q^T \cdot b$$

obs: devido Q ser uma matriz ortogonal a fórmula anterior é válida

E assim achando a solução resolvendo o sistema triangular inferior:

$$R \cdot x = y$$

- Código de implementação :

```

1 #include<stdio.h>
2
3 int main()
4 {
5     //operadores de tamanho 2x2 (especificadona questao)
6     float A[2][2], R[2][2];
7     //lendo amatriz A
8     int i, j, k;
9     for(i=0;i<2;i++)
10         for(j=0;j<2;j++)
11             scanf("%f", &A[i][j]);
12     //lendo o vetor b
13     float b[2];
14     for(i=0;i<2;i++)
15         scanf("%f", &b[i]);
16
17     //incializando o rotator com angulo de 45 graus
18     float Q[2][2] = { {1, -1}, {1, 1} };

```

```
19
20 // fazendo o calculo da patriz R = Qt.A
21 for (i=0;i<2;i++)
22 {
23     for (k=0;k<2;k++)
24     {
25         float somatorio = 0;
26         for (j=0;j<2;j++)
27         {
28             // ler nota 1 ao fim do c digo
29             somatorio += Q[j][i] * A[j][k];
30         }
31         R[i][k] = somatorio;
32     }
33 }
34
35 float Qtb[2]; //vetor que gardara a multiplicacao da matriz Qt pelo vetor b
36 for (i=0;i<2;i++)
37 {
38     float somatorio = 0;
39     for (j=0;j<2;j++)
40         somatorio += Q[j][i]*b[j]; //simulando transposta de novo
41
42     Qtb[i] = somatorio;
43 }
44
45 float x[2];
46
47 //triangular inferior adaptado (R.x = Qt.b)
48 for (i=1;i>=0;i--)
49 {
50     float somatorio = Qtb[i];
51     for (j=1;j>=i;j--)
52     {
53         if (i==j)
54             somatorio /= R[i][j];
55         if (j>i)
56             somatorio -= R[i][j]*x[j];
57     }
58     x[i] = somatorio;
59 }
60
61 //imprimindo as matrizes
62 printf("matriz Q: \n");
63 for (i=0;i<2;i++)
64 {
65     for (j=0;j<2;j++)
66         printf("%f ", Q[i][j]);
67     printf("\n");
68 }
69
70 printf("\n");
71
```

```

72     printf("matriz R: \n");
73     for (i=0;i<2;i++)
74     {
75         for (j=0;j<2;j++)
76             printf("%f ", R[i][j]);
77         printf("\n");
78     }
79
80     printf("\n");
81
82     //imprimindo a solucao
83     printf("vetor solu o x: \n");
84     for (i=0;i<2;i++)
85         printf("%f\n", x[i]);
86 }
87 //nota 1:
88 /* a formula seria R = Qt*A, mas inverter a posi o de i e j no acesso a Q
89     o suficiente para simular a transposi o de Q */

```

Listing 14: Decomposição QR

Exercício 15

Letra a:

O exercício pede para o programa verificar os seguintes critérios: Critério das linhas, Critério das colunas, Critério de Sassenfeld e o Critério da norma.

Para isso, o programa pedirá a ordem da matriz do sistema, a matriz do sistema e o vetor b do sistema.

A primeira parte do programa irá verificar o critério das linhas, definido como:

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

Usaremos uma variável "somatorioLinhas" para representar o somatório de cada linha da matriz do sistema. No loop adicionaremos o valor absoluto de cada elemento da linha, com exceção do elemento a_{ii} . Compara-se em um if se esse somatório é maior ou igual ao valor absoluto de a_{ii} , se verdadeiro, o critério é descumprido. Isso ocorre até o fim das análises de todas as linhas. Se nenhuma delas foge da regra, o critério será satisfeito.

Em segundo, o Critério das colunas é calculado. Usa-se o mesmo raciocínio que o critério das linhas, apenas com diferença no somatório das colunas e não das linhas.

Em terceiro, o Critério de Sassenfeld é calculado. Baseamos o algoritmo na fórmula:

$$\beta_j = \frac{|a_{j1}| \cdot \beta_1 + |a_{j2}| \cdot \beta_2 + \dots + |a_{jj-1}| \cdot \beta_{j-1} + |a_{jj+1}| + \dots + |a_{jn}|}{|a_{jj}|}$$

$$\beta_1 = \frac{|a_{12}| + |a_{13}| + \dots + |a_{1n}|}{|a_{11}|}$$

$$\beta = \max_{1 \leq j \leq n} \{\beta_j\}$$

O somatório do numerador de cada equação será representado pela variável "somatório" no código. Os valores de betas estarão contidos no vetor "beta[n]" e o maior valor de beta como "maxBeta". Primeiro teremos um loop separado para aplicar a primeira equação do β_1 . A partir do segundo, ser encontrado o valor de β_2 até β_n . Será calculado por vez no loop o valor de " $|a_{ji}| \cdot \beta_j$ " e sendo adicionado no "somatório". Quando finalizar, o valor de "beta[i]" será o "somatório" dividido pelo valor absoluto de a_{ii} . Mantém-se esse ciclo até calcular todos os betas. Depois iremos comparar cada valor de beta. O maior valor entre eles será o "maxBeta". Se o "maxBeta" ≥ 1 , o critério é descumprido, caso contrário, será satisfeito.

Em quarto e último, o Critério da norma é calculado. A existência de uma matriz B é necessária, de tal forma que:

$$x = B \cdot x + g$$

Tal que a matriz B cumpra esta forma:

$$B = \begin{bmatrix} 0 & \frac{-a_{12}}{a_{11}} & \frac{-a_{13}}{a_{11}} & \dots & \frac{-a_{1n}}{a_{11}} \\ \frac{-a_{21}}{a_{22}} & 0 & \frac{-a_{23}}{a_{22}} & \dots & \frac{-a_{2n}}{a_{22}} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \frac{-a_{(n-1)1}}{a_{(n-1)(n-1)}} & \dots & \ddots & \ddots & \frac{-a_{(n-1)n}}{a_{(n-1)(n-1)}} \\ \frac{-a_{n1}}{a_{nn}} & \frac{-a_{n2}}{a_{nn}} & \dots & \frac{-a_{nn-1}}{a_{nn}} & 0 \end{bmatrix}$$

A matriz B representa os valores dos coeficientes de cada x quando calculamos o valor de $x_1, x_2, x_3, \dots, x_{n-1}, x_n$. Com base nesta matriz, trabalharemos, calculando cada valor da matriz B, com exceção das diagonais, que serão sempre zero. Depois de determinada a matriz B, iremos calcular a norma 1, infinito e frobenius para verificar se algumas dessas normas são maiores ou iguais a 1, pois se $\|B\| \leq 1$ para qualquer norma, o critério é cumprido. Caso alguma norma seja igual ou maior que 1, o critério não é satisfeito.

Finalmente, iremos mostrar os resultados, verificando cada variável booleana de cada critério. Se o valor for igual a 1, o critério é cumprido; se for igual a zero, ele não é satisfeito.

- Código de implementação :


```
1 #include <stdio.h>
2 #include <math.h>
3
4 //Funcoes para calcular normas ja usadas anteriormente.
5 double Normal(double* enderecoMatriz, int ordemMatriz)
6 {
7     int i, j, k = 0;
8     double somatorio = 0;
9     double normal = 0;
10    double Matriz[ordemMatriz][ordemMatriz];
11
12    for(i = 0; i < ordemMatriz; i++)
13    {
14        for(j = 0; j < ordemMatriz; j++)
15        {
16            Matriz[i][j] = *(enderecoMatriz + k);
17            k++;
18        }
19    }
20
21    for(i = 0; i < ordemMatriz; i++)
22    {
23        for(j = 0; j < ordemMatriz; j++)
24        {
25            somatorio += fabs(Matriz[j][i]);
26        }
27
28        if(normal < somatorio)
29        {
30            normal = somatorio;
31        }
32
33        somatorio = 0;
34    }
35
36    return normal;
37 };
38
39 double NormaInfinito(double* enderecoMatriz, int ordemMatriz)
40 {
41     int i, j, k = 0;
42     double somatorio = 0;
43     double normaInfinito = 0;
44     double Matriz[ordemMatriz][ordemMatriz];
45
46     for(i = 0; i < ordemMatriz; i++)
47     {
48         for(j = 0; j < ordemMatriz; j++)
49         {
50             Matriz[i][j] = *(enderecoMatriz + k);
51             k++;
52         }
53     }
```

```
54
55     for(i = 0; i < ordemMatriz; i++)
56     {
57         for(j = 0; j < ordemMatriz; j++)
58         {
59             somatorio += fabs(Matriz[i][j]);
60         }
61
62         if(normaInfinito < somatorio)
63         {
64             normaInfinito = somatorio;
65         }
66
67         somatorio = 0;
68     }
69
70     return normaInfinito;
71 };
72
73
74 double NormaFrobenius(double* enderecoMatriz, int ordemMatriz)
75 {
76     int i, j, k = 0;
77     double somatorio = 0;
78     double normaFrobenius = 0;
79     double Matriz[ordemMatriz][ordemMatriz];
80
81     for(i = 0; i < ordemMatriz; i++)
82     {
83         for(j = 0; j < ordemMatriz; j++)
84         {
85             Matriz[i][j] = *(enderecoMatriz + k);
86             k++;
87         }
88     }
89
90     for(i = 0; i < ordemMatriz; i++)
91     {
92         for(j = 0; j < ordemMatriz; j++)
93         {
94             somatorio += pow(fabs(Matriz[i][j]), 2);
95         }
96     }
97
98     normaFrobenius = sqrt(somatorio);
99
100    return normaFrobenius;
101 };
102
103 int main()
104 {
105     int i, j, n;
106
```

```
107 //Variaveis de validacao dos criterios. Caso seus valores sejam 0, o
    criterio nao eh satisfeito.
108 //caso contrario(valor 1), os criterios sao satisfeitos.
109 int criterioLinhas = 1, criterioColunas = 1, criterioNorma = 1,
    criterioSassenfeld = 1;
110
111 double somatorio = 0;
112
113 double somatorioLinhas = 0;
114 double somatorioColunas = 0;
115
116 printf("Qual a ordem do sistema Ax = b?\n");
117 scanf("%d",&n);
118
119 double A[n][n];
120 printf("Qual a matriz A?\n");
121
122 for(i = 0; i < n; i++)
123 {
124     for(j = 0; j < n; j++)
125     {
126         scanf("%lf",&A[i][j]);
127     }
128 }
129
130 double b[n];
131 printf("Qual o vetor b?\n");
132
133 for(i = 0; i < n; i++)
134 {
135     scanf("%lf",&b[i]);
136 }
137
138 //Aplicar o criterio das linhas
139
140 for(i = 0; i < n; i++)
141 {
142     for(j = 0; j < n; j++)
143     {
144         //nao somamos elementos da diagonal.
145         if(i != j)
146         {
147             somatorioLinhas += fabs(A[i][j]);
148         }
149     }
150     //Se o somatorio linha i for maior ou igual ao valor absoluto de aii,
151     //o criterio das linhas nao eh satisfeito.
152     if(somatorioLinhas >= fabs(A[i][i]))
153     {
154         criterioLinhas = 0;
155         break;
156     }
157 }
```

```
158     somatorioLinhas = 0;
159 }
160
161 //Critério das colunas
162
163 for(i = 0; i < n; i++)
164 {
165     for(j = 0; j < n; j++)
166     {
167         //nao somamos elementos da diagonal.
168         if(i != j)
169         {
170             somatorioColunas += fabs(A[j][i]);
171         }
172     }
173
174     //Se o somatorio coluna i for maior ou igual ao valor absoluto de aii,
175     //o critério das colunas nao eh satisfeito.
176     if(somatorioColunas >= fabs(A[i][i]))
177     {
178         criterioColunas = 0;
179         break;
180     }
181
182     somatorioColunas = 0;
183 }
184
185 //Critério de Sassenfeld
186
187 double beta[n]; //vetor que guarda os valores dos betas.
188 double maxBeta = 0; //maior valor de beta.
189
190 //Inicio da aplica o da formula do critério de Sassenfeld
191
192 //Calculo do beta[0](primeiro beta)
193 for(j = 0; j < n; j++)
194 {
195     //Nao se pode somar o elemento da diagonal ii no calculo do beta(i).
196     if(j != 0)
197         somatorio += fabs(A[0][j]);
198 }
199
200 beta[0] = somatorio / fabs(A[0][0]);
201 somatorio = 0; //reinicio da variavel para o calculo do proximo beta.
202
203 for(i = 1; i < n; i++)
204 {
205     //Serve para eliminar o problema dos calculos do somatorio com os betas
206     .
207     //Em outras palavras: "N o temos no calculo do beta(i) a opera o
208     aji * beta(i), temos aji. Para
209     //eliminar este problema, atribuímos ao beta(i) o valor de 1, pois todo
210     numero multiplicado por 1 eh
```

```
208     //ele mesmo.”
209     beta[i] = 1;
210
211     for(j = 0; j < n; j++)
212     {
213         //Nao se pode somar o elemento da diagonal ii no calculo do beta(i)
214
215         if(i != j)
216             somatorio += fabs(A[i][j]) * beta[j];
217     }
218     beta[i] = somatorio / fabs(A[i][i]);
219     somatorio = 0; //reiniciando a variavel para o calculo do proximo beta.
220 }
221
222 //Comparacao para ver quem eh o maior valor de beta.
223 for(i = 0; i < n; i++)
224 {
225     if(beta[i] > maxBeta)
226         maxBeta = beta[i];
227 }
228
229 //Se o maior valor de beta for >= 1, o criterio de Sassenfeld nao eh
230 //cumprido.
231 if(!(maxBeta < 1))
232     criterioSassenfeld = 0;
233
234 //Critério da Norma
235
236 double B[n][n];
237
238 //Pelo raciocinio, o valor de x(k) quando calculamos o valor de x(k) em
239 //funcao de x(k - n), ..., x(k - 1), x(k), x(k + 1), ...,
240 //x(k + n) sera sempre zero, devido a definicao dos metodos iterativos.
241 //Isso representa que os elementos da diagonal da matriz B sempre ser o
242 //zero, pois  $x = Bx + g$ .
243 for(i = 0; i < n; i++)
244 {
245     B[i][i] = 0;
246 }
247
248 //Calculando so valores de B.
249 //Esta parte representa o processo de isolamento da variavel x(k) em funcao
250 //das outras variaveis x.
251 //A matriz B representa o valor dos coeficientes das variaveis x quando
252 //serao usadas no calculo da variavel x(k).
253 for(i = 0; i < n; i++)
254 {
255     for(j = 0; j < n; j++)
256     {
257         if(i != j)
258             B[i][j] = -1 * (A[i][j] / A[i][i]);
259     }
260 }
```

```

255     }
256     //Endereco para a passagem para as funcoes.
257     double* enderecoB = &B[0][0];
258
259     // Validacao do criterio da norma: Para qualquer norma de B, temos ||B|| <
260     // 1. Se ocorre o contrario, o criterio nao
261     //eh satisfeito.
262     if((Normal(enderecoB,n) >= 1) && (NormaInfinito(enderecoB,n) >= 1) && (
263         NormaFrobenius(enderecoB,n) >= 1))
264         criterioNorma = 0;
265
266     //Resultados
267     //Estaremos aqui comparando cada criterio com seu respectivo metodo
268     //iterativo.
269
270     if(criterioLinhas == 1 || criterioColunas == 1)
271         printf("O crit rio das linhas e colunas convergem.Ent o , o m todo de
272             Jacobi e Gauss-Seidel convergem.\n");
273     else
274         printf("Os crit rios de linhas ou colunas n o convergem. N o podemos
275             afirmar com certeza que o m todo de Jacobi e Gauss-Seidel
276             convergem.\n");
277
278     if(criterioSassenfeld == 1)
279         printf("O crit rio de Sassenfeld      satisfeito. Ent o , o m todo de
280             Gauss-Seidel converge.\n");
281     else
282         printf("O crit rio de Sassenfeld n o      satisfeito. N o podemos
283             afirmar com certeza que o m todo de Gauss-Seidel converge.\n");
284
285     if(criterioNorma == 1)
286         printf("O crit rio da norma      satisfeito. Portanto o m todo
287             iterativo com a matriz B converge.\n");
288     else
289         printf("O crit rio da norma n o      satisfeito.N o podemos afirmar
290             que o m todo iterativo com a matriz B converge.\n");
291
292     return 0;
293 }

```

Listing 15: avaliaçã dos critérios de convergência para os métodos iterativos

Letra b :

Neste exercício, iremos implementar o método de Jacobi. Nossa lógica baseia-se em:

$$\begin{aligned}
 x_1^{k+1} &= \frac{b_1 - a_{12} \cdot x_2^k - a_{13} \cdot x_3^k - \dots - a_{1n} \cdot x_n^k}{a_{11}} \\
 x_2^{k+1} &= \frac{b_2 - a_{21} \cdot x_1^k - a_{23} \cdot x_3^k - \dots - a_{2n} \cdot x_n^k}{a_{22}} \\
 &\dots \\
 x_n^{k+1} &= \frac{b_n - a_{n1} \cdot x_1^k - a_{n2} \cdot x_2^k - \dots - a_{nn-1} \cdot x_{n-1}^k}{a_{nn}}
 \end{aligned}$$

Pediremos ao usuário a matriz do sistema, o vetor b do sistema, a solução inicial do método e o critério de parada desejado.

O critério de parada utilizado aqui foi o seguinte:

$$\frac{|x_i^{k+1} - x_i^k|}{|x_i^k|} \leq p, \forall i = 1, \dots, n; |x_i^k| \neq 0.$$

Nosso loop do método de Jacobi resume-se no nosso while no código. Ali calculamos o valor de "xKmais1[i]" comeando atribuindo o valor de $b[i]$ na variável para depois somar com o valor negado da multiplicação entre os x com seus respectivos coeficientes e em final dividir tudo isto por a_{ii} .

Lembramos aqui que o método de Jacobi usa sempre os mesmo valores de x^k na mesma iteração, sem atualizá-la durante a iteração.

Quando chegamos na última variável x , atribuímos os valores do vetor "xK" no vetor "xK2" (efeito de critério de parada) e depois atualizamos o vetor "xK" com os valores de "xKmais1" para a próxima iteração. Partiremos para a verificação do critério de parada. Se for cumprido, o loop é desfeito; se não for, voltamos ao loop principal para a próxima iteração.

Por fim, exibiremos o último valor de "xKmais1", que será a solução aproximada obtida no método.

- Código de implementação :

```

1 #include <stdio.h>
2 #include <math.h>
3
4 int main()
5 {
6     int i,j,h,n; //i,j e h —————> Variaveis de iteracao de loop.
7                 //n —————> ordem da matriz A do sistema Ax = b.
8
9     printf("Qual a ordem do sistema Ax = b?\n");
10    scanf("%d",&n);
11
12    double A[n][n];
13    printf("Qual a matriz A?\n");
14
15    for(i = 0; i < n; i++)
16    {
17        for(j = 0; j < n; j++)
18        {
19            scanf("%lf",&A[i][j]);
20        }
21    }
22
23    double b[n];

```

```
24     printf("Qual o vetor b?\n");
25
26     for(i = 0; i < n; i++)
27     {
28         scanf("%lf",&b[i]);
29     }
30
31     double xK[n]; //x(k)
32     double xKmais1[n]; //x (k + 1)
33     double xK2[n]; //Servira para guardar os valores de x(k) antes que o mesmo
        receba o valor
34         //de x(k + 1) depois que a iteracao i for feita.
35         //Necessario para o caculo do criterio de parada.
36
37     printf("Escolha a solu o inicial para a itera o de Jacobi:\n");
38
39     for(i = 0; i < n; i++)
40     {
41         scanf("%lf",&xK[i]);
42     }
43
44     double p; //valor do criterio de parada.
45
46     printf("Escolha um valor de crit rio de parada do algoritmo:\n");
47     scanf("%lf",&p);
48
49     //Implementacao do metodo de Jacobi
50
51     int parada = 0; //Varavel que ira controlar o loop do algoritmo.
52         //Quando parada == 1, o loop "while" eh rompido.
53
54     while(parada == 0)
55     {
56         //Calculo de cada valor de x(k + 1) na iteracao , considerando o
            isolamento de cada x em cada linha do sistema.
57         for(i = 0; i < n; i++)
58         {
59             xKmais1[i] = b[i];
60
61             for(j = 0; j < n; j++)
62             {
63                 //Se j == i, estaremos usando o valor de xj para calcular xj,
                    sendo algo sem sentido.
64                 if(j != i)
65                 {
66                     xKmais1[i] += -1 * (A[i][j] * xK[j]);
67                 }
68             }
69
70             xKmais1[i] = xKmais1[i] / A[i][i];
71
72             //Quando chegarmos no calculo do ultimo x, iremos atualizar o valor
                de x(k) para proxima iteracao.
```



```

73         if(i == n - 1)
74         {
75             for(h = 0; h < n; h++)
76             {
77                 xK2[h] = xK[h]; //Uso de criterio de parada.
78                 xK[h] = xKmais1[h]; //atualizacao de x(k)
79             }
80         }
81     }
82
83     for(h = 0; h < n; h++)
84     {
85         //se x(k) for diferente de 0, iremos verificar o criterio de parada
86         .
87         if(xK2[h] != 0)
88         {
89             //Se cumprimos a condicao, a iteracao sera continuada.
90             if((fabs(xKmais1[h] - xK2[h]) / fabs(xK2[h])) > p)
91             {
92                 break;
93             }
94
95             //Se chegarmos no ultimo valor de x e o loop nao for rompido, entao
96             podemos dizer
97             //que o criterio de parada foi cumprido e sairemos do loop
98             principal.
99             if(h == n - 1 && xK2[h] != 0)
100             {
101                 parada = 1;
102             }
103         }
104
105         //Resultado(solucao aproximada do sistema)
106
107         printf(" Solu o aproximada:\n");
108
109         for(i = 0; i < n; i++)
110         {
111             printf("x[%d] = %f\n", i, xKmais1[i]);
112         }
113
114         return 0;
115     }

```

Listing 16: resolução do sistema pelo método iterativo de Jacobi

Letra c :

Neste exercício, implementaremos o método de Gauss-Seidel. Em suma, o algoritmo foi baseado no algoritmo da letra b deste mesmo exercício, com algumas modificações: Nossa lógica para esse

método irá baseia-se em:

$$\begin{aligned}
 x_1^{k+1} &= \frac{b_1 - a_{12} \cdot x_2^k - a_{13} \cdot x_3^k - \dots - a_{1n} \cdot x_n^k}{a_{11}} \\
 x_2^{k+1} &= \frac{b_2 - a_{21} \cdot x_1^{k+1} - a_{23} \cdot x_3^k - \dots - a_{2n} \cdot x_n^k}{a_{22}} \\
 &\dots \\
 x_n^{k+1} &= \frac{b_n - a_{n1} \cdot x_1^{k+1} - a_{n2} \cdot x_2^{k+1} - \dots - a_{nn-1} \cdot x_{n-1}^{k+1}}{a_{nn}}
 \end{aligned}$$

Teremos as mesmas condições de entrada e o mesmo critério de parada do exercício anterior (letra b).

Nosso loop do Gauss-Seidel resumira-se ao while novamente e seguirá os mesmos passos que o método de Jacobi, com algumas diferenças: Nosso "xK" irá ser atualizado durante o loop de uma iteração, ou seja, realizará a atribuição "xK[i] = xKmais1[i]" logo depois que calcular um "xK-mais1[i]", e não apenas no final, como foi no Jacobi. Assim conseguiremos respeitar e aplicar nossa lógica já apresentada.

Terminado a iteração, irá verificar-se o critério de parada. Em mesmo esquema do exercício anterior, se for cumprido, o loop while será desfeito, caso contrário, voltaremos a um novo loop (nova iteração).

Quando a parada for realizada, iremos exibir os resultados na tela, representando o último valor de "xKmais1" na última iteração e será a solução aproximada do sistema.

- Código de implementação :

```

1  #include <stdio.h>
2  #include <math.h>
3
4  int main()
5  {
6      int i,j,h,n;
7
8      printf("Qual a ordem do sistema Ax = b?\n");
9      scanf("%d",&n);
10
11     double A[n][n];
12     printf("Qual a matriz A?\n");
13
14     for(i = 0; i < n; i++)
15     {
16         for(j = 0; j < n; j++)
17         {
18             scanf("%lf",&A[i][j]);
19         }
20     }
21

```

```
22     double b[n];
23     printf("Qual o vetor b?\n");
24
25     for(i = 0; i < n; i++)
26     {
27         scanf("%lf", &b[i]);
28     }
29
30     double xK[n]; //x(k)
31     double xKmais1[n]; //x (k + 1)
32     double xK2[n]; //Servira para guardar os valores de x(k) depois que a
        iteracao i for feita.
33         //Necessario para o caculo do criterio de parada.
34
35     printf("Escolha a solu o inicial para a itera o de Gauss-Seidel:\n");
36
37     for(i = 0; i < n; i++)
38     {
39         scanf("%lf", &xK[i]);
40         xK2[i] = xK[i]; //para guardar o valor inicial de xK para realizar o
        teste de parada.
41     }
42
43     double p; //valor do criterio de parada
44
45     printf("Escolha um valor de crit rio de parada do algoritmo:\n");
46     scanf("%lf", &p);
47
48     //Implementacao do metodo de Gauss-Seidel
49
50     int parada = 0; //Varavel que ira controlar o loop do algoritmo.
        //Quando parada == 1, o loop "while" eh rompido.
51
52
53     while(parada == 0)
54     {
55         //Calculo de cada valor de x(k + 1) na iteracao , considerando o
        isolamento de cada x em cada linha do sistema.
56         for(i = 0; i < n; i++)
57         {
58             xKmais1[i] = b[i];
59
60             for(j = 0; j < n; j++)
61             {
62                 //Se j == i, estaremos usando o valor de xj para calcular xj,
        sendo algo sem sentido.
63                 if(j != i)
64                 {
65                     xKmais1[i] += -1 * (A[i][j] * xK[j]);
66                 }
67             }
68
69             xKmais1[i] = xKmais1[i] / A[i][i];
70             xK2[i] = xK[i]; //guardar o valor da iteracao k para efeito de
```

```
71         teste de parada.
72         xK[i] = xKmais1[i]; //x(k) sera x(k + 1) para ser usado na mesma
73         iteracao.Nesta linha nos diferenciamos do
74         //metodo de Jacobi, pois atualizamos o valor de
75         x(k) na mesma iteracao.
76     }
77     for(h = 0; h < n; h++)
78     {
79         //se x(k) for diferente de 0, iremos verificar o criterio de parada
80         .
81         if(xK2[h] != 0)
82         {
83             //Se cumprirmos a condicao, a iteracao sera continuada.
84             if((fabs(xKmais1[h] - xK2[h]) / fabs(xK2[h])) > p)
85             {
86                 break;
87             }
88         }
89         //Se chegarmos no ultimo valor de x e o loop nao for rompido, entao
90         podemos dizer
91         //que o criterio de parada foi cumprido e sairemos do loop
92         principal.
93         if(h == n - 1 && xK2[h] != 0)
94         {
95             parada = 1;
96         }
97     }
98     //Resultado(solucao aproximada do sistema)
99     printf(" Solu o aproximada:\n");
100     for(i = 0; i < n; i++)
101     {
102         printf("x[%d] = %f\n", i, xKmais1[i]);
103     }
104     return 0;
105 }
106 }
```

Listing 17: resolução de um sistema pelo método iterativo de Gauss-Seidel

Exercício 16

Neste exercício, implementaremos o método SOR, quando a matriz é positiva definida. A lógica do método SOR que usaremos no programa é:

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

Antes de descrever o passo a passo, deve-se ressaltar algo importante: o método SOR é praticamente o mesmo que o método de Gauss-Seidel. Por isso, nosso algoritmo ficará teoricamente idêntico ao do método Gauss-Seidel, feito no exercício 15 da prova. A diferença é que o Gauss-Seidel é o SOR aplicado sempre com o $\omega = 1$. O ω serve para acelerar a convergência para a solução do sistema. Por isso mesmo, repetiremos os passos de Gauss-Seidel. Citaremos apenas as distinções. Antecipando a iteração, necessitaremos o valor de ω . Aqui, de acordo com o teorema, quando a matriz da iteração é positiva definida, o valor ideal de ω está entre 0 e 2 e o SOR converge para qualquer valor inicial de x_i . Portanto, geraremos esse ω aleatoriamente, entre 0 e 2, com uma operação de gerador de números aleatórios em C, explicado com mais detalhes no código do programa. O usuário saberá qual será o ω utilizado antes da iteração iniciar.

Agora podemos começar nossos cálculos. Prosseguiremos como Gauss-Seidel, mas antes de atualizarmos o vetor "xK", teremos que realizar operações adicionais para calcular nosso "xKmais1": multiplicar o valor por ω ; somar o resultado com $(1 - \omega) \cdot xK[i]$. Somente agora poderemos atualizar o vetor "xK[i]" para o cálculo do novo "xKmais1[i]" e logo verificar nosso critério de parada (que é o mesmo que estamos usando até agora em todos os métodos iterativos feitos nesta prova). Se for cumprido, rompemos o loop principal.

Exibiremos os resultados, ou seja, a solução aproximada do sistema inserido.

- Código de implementação :

```

1  #include <stdio.h>
2  #include <math.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  /*Como gerar numeros aleatorios em ponto flutuante:
7   Seja ]a;b[ o intervalo com o qual se deseja gerar o conjunto de numeros
      aleatorios.
8   Entao, o "aleatorio" gerado sera igual a:
9           a + (b - a) * ((double)(rand())/RAND_MAX)
10   Com isso, geraremos numeros de ponto flutuante aleatorios no intervalo ]a;b
      [.
11 */
12
13 //Ira realizar o metodo SOR

```

```
14 void MetodoSOR(double*matrizA ,double*matrizB ,double criterioParada ,int
    ordemSistema)
15 {
16     int i,j,h,k = 0; //variaveis de iteracao
17     double A[ordemSistema][ordemSistema];
18     double b[ordemSistema];
19
20     //Receber os elementos da matriz A
21     for(i = 0; i < ordemSistema; i++)
22     {
23         for(j = 0; j < ordemSistema; j++)
24         {
25             A[i][j] = *(matrizA + k);
26             k++;
27         }
28     }
29
30     //Receber os elementos do vetor b
31     for(i = 0; i < ordemSistema; i++)
32     {
33         b[i] = *(matrizB + i);
34     }
35
36     double xK[ordemSistema]; //x(k)
37     double xKmais1[ordemSistema]; //x (k + 1)
38     double xK2[ordemSistema]; //Servira para guardar os valores de x(k) depois
        que a iteracao i for feita.
39         //Necessario para o caculo do criterio de parada.
40
41     printf("Escolha a solu o inicial para a itera o SOR:\n");
42
43     for(i = 0; i < ordemSistema; i++)
44     {
45         scanf("%lf",&xK[i]);
46         xK2[i] = xK[i]; //para guardar o valor inicial de xK para realizar o
            teste de parada.
47     }
48
49     //Implementa o do metodo SOR.
50
51     //Agora, iremos estimar o omega otimo. Como a matriz eh positiva definida ,
        podemos afirmar
52     //que o omega esta entre  $0 < w < 2$ . Usaremos a funcao de gerador de numeros
        aleatorios do C
53     //para obter um numero "aleatorio" entre 0 e 2.
54
55     double w;
56
57     srand((double)time(NULL));
58     w = 2 * ((double)(rand())/RANDMAX);
59
60     printf("w que ser usado: %f\n",w);
61
```

```

62
63 //Com o omega, iniciaremos a iteracao do metodo SOR.
64
65 int parada = 0; //Varavel que ira controlar o loop do algoritmo.
66 //Quando parada == 1, o loop "while" eh rompido.
67
68 while(parada == 0)
69 {
70     //Calculo de cada valor de x(k + 1) na iteracao , considerando o
71     //isolamento de cada x em cada linha do sistema.
72     for(i = 0; i < ordemSistema; i++)
73     {
74         xKmais1[i] = b[i];
75
76         for(j = 0; j < ordemSistema; j++)
77         {
78             if(j != i)
79             {
80                 xKmais1[i] += -1 * (A[i][j] * xK[j]);
81             }
82
83             xKmais1[i] = xKmais1[i] / A[i][i];
84
85             //Nesta parte em exato, encontramos a diferenca do m todo SOR. O
86             //uso
87             //de uma constante w para acelerar o processo. Afirmamos que o
88             //metodo SOR
89             //seria apenas o metodo de Gauss-Seidel melhorado. O Gauss-Seidel
90             //eh o SOR
91             //considerando w = 1.
92             //-----
93             xKmais1[i] *= w; //multiplicando o w como diz a formula.
94             xKmais1[i] += (1 - w) * xK[i]; // somando com (1 - w) * x(k) como
95             //diz a formula.
96             //
97             //-----
98
99             xK2[i] = xK[i]; //guardar o valor da iteracao k para efeito de
100             //teste de parada.
101             xK[i] = xKmais1[i]; //x(k) sera x(k + 1) para ser usado na mesma
102             //iteracao.Nesta linha encontramos
103             //uma semelhanca com o Gauss-Seidel, pois atualizamos o valor de x(
104             //k) na mesma iteracao.
105         }
106
107         for(h = 0; h < ordemSistema; h++)
108         {
109             //se x(k) for diferente de 0, iremos verificar o criterio de parada.
110             if(xK2[h] != 0)
111             {
112                 //Se cumprimos a condicao, a iteracao sera continuada.

```

```
105         if(( fabs(xKmais1[h] - xK2[h]) / fabs(xK2[h])) > criterioParada)
106         {
107             break;
108         }
109     }
110
111     //Se chegarmos no ultimo valor de x e o loop nao for rompido, entao
112     //podemos dizer
113     //que o criterio de parada foi cumprido e sairemos do loop
114     //principal.
115     if(h == ordemSistema - 1 && xK2[h] != 0)
116     {
117         parada = 1;
118     }
119 }
120 //Resultado(solucao aproximada do sistema)
121
122 printf(" Solu o aproximada:\n");
123
124 for(i = 0; i < ordemSistema; i++)
125 {
126     printf("x[%d] = %f\n", i, xKmais1[i]);
127 }
128 };
129
130 int main()
131 {
132     int i, j, n;
133
134     printf("Qual a ordem do sistema?\n");
135     scanf("%d", &n);
136
137     double A[n][n];
138     printf("Qual a matriz A positiva definida?\n");
139
140     for(i = 0; i < n; i++)
141     {
142         for(j = 0; j < n; j++)
143         {
144             scanf("%lf", &A[i][j]);
145         }
146     }
147
148     double b[n];
149     printf("Qual o vetor b?\n");
150
151     for(i = 0; i < n; i++)
152     {
153         scanf("%lf", &b[i]);
154     }
155 }
```



```

156     double criterioParada;
157
158     printf("Escolha um valor de crit rio de parada do algoritmo:\n");
159     scanf("%lf",&criterioParada);
160
161     //Endereco de A e b para enviar a fun o que realizara o metodo SOR.
162     double* enderecoA = &A[0][0];
163     double* enderecoB = &b[0];
164
165     //Funcao que executara o m todo SOR
166     MetodoSOR(enderecoA ,enderecoB , criterioParada ,n);
167
168     return 0;
169 }

```

Listing 18: resolução do sistema pelo método iterativo SOR

Exercício 17

Este exercício 17 é uma outra alternativa do método SOR quando fizemos a 16. Só que neste iremos trabalhar com uma matriz 3x3, tridiagonal e positiva definida. Usaremos o mesmo raciocínio do método SOR usado na questão 16, apenas com algumas diferenças.

Primeiro, devemos calcular o ω ótimo para o método, pois a matriz do sistema é tridiagonal e positiva definida, de acordo com o teorema do ω ótimo. Portanto, usaremos a fórmula dada para calcular o ω ótimo:

$$\omega = \frac{2}{1 + \sqrt{1 - (\rho(T_j))^2}}$$

E depois disso aplicaremos na iteração.

Antes, precisaremos calcular o $\rho(T_j)$ para depois calcular o ω .

Após a efetivação de todos aqueles passos para se achar o $\rho(T_j)$ da matriz genérica $A_{3 \times 3}$, tridiagonal e positiva definida:

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & a_{23} \\ 0 & a_{32} & a_{33} \end{bmatrix}$$

Conseguimos chegar a uma fórmula genérica em função dos elementos da matriz A. A fórmula consiste em:

$$\rho(T_j) = \sqrt{\frac{a_{32} \cdot a_{23}}{a_{33} \cdot a_{22}} + \frac{a_{21} \cdot a_{12}}{a_{22} \cdot a_{11}}}$$

Usaremos esta fórmula para encontrar $\rho(T_j)$ no algoritmo.

Calcularemos $\rho(T_j)$ e em seguida o ω ótimo.

Podemos agora iniciar nosso método SOR. Repetiremos os mesmos passos do exercício 16 com as mesmas condições para realizar a iteração.

Quando terminar, exibiremos a solução aproximada do sistema.

- Código de implementação :

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main()
5 {
6     int i,j,h;
7
8     double A[3][3];
9     printf("Qual a matriz A de ordem 3,positiva definida e tridiagonal?\n");
10
11     for(i = 0;i < 3;i++)
12     {
13         for(j = 0;j < 3;j++)
14         {
15             scanf("%lf",&A[i][j]);
16         }
17     }
18
19     double b[3];
20     printf("Qual o vetor b?\n");
21
22     for(i = 0;i < 3;i++)
23     {
24         scanf("%lf",&b[i]);
25     }
26
27     double xK[3]; //x(k)
28     double xKmais1[3]; //x (k + 1)
29     double xK2[3]; //Servira para guardar os valores de x(k) depois que a
30                     //iteracao i for feita.
31                     //Necessario para o caculo do criterio de parada.
32     printf("Escolha a solu o inicial para a itera o SOR:\n");
33
34     for(i = 0;i < 3;i++)
35     {
36         scanf("%lf",&xK[i]);
37         xK2[i] = xK[i]; //para guardar o valor inicial de xK para realizar o
38                         teste de parada.
39     }
```

```
39
40     double p;
41
42     printf("Escolha um valor de critério de parada do algoritmo:\n");
43     scanf("%lf",&p);
44
45     //Implementacao do metodo SOR.
46
47     //Aplicacao da formula generica.
48     double pTj = 0;
49
50     pTj += (A[2][1] * A[1][2]) / (A[2][2] * A[1][1]);
51     pTj += (A[1][0] * A[0][1]) / (A[1][1] * A[0][0]);
52     pTj = sqrt(pTj);
53
54     //Agora, iremos calcular o omega otimo atraves de sua formula.
55
56     double w = 0;
57
58     w = 2 / (1 + sqrt(1 - (pTj * pTj)));
59
60     printf("w = %f\n",w);
61
62     //

```

```
63     //Com o omega, iniciaremos a iteracao do metodo SOR.
64
65     int parada = 0; //Varavel que ira controlar o loop do algoritmo.
66                     //Quando parada == 1, o loop "while" rompido.
67
68     while(parada == 0)
69     {
70         //Calculo de cada valor de x(k + 1) na iteracao, considerando o
71         //isolamento de cada x em cada linha do sistema.
72         for(i = 0; i < 3; i++)
73         {
74             xKmais1[i] = b[i];
75
76             for(j = 0; j < 3; j++)
77             {
78                 if(j != i)
79                 {
80                     xKmais1[i] += -1 * (A[i][j] * xK[j]);
81                 }
82             }
83
84             xKmais1[i] = xKmais1[i] / A[i][i];
85
86             //Nesta parte em exato, encontramos a diferenca do metodo SOR. O
87             //uso
88             //de uma constante w para acelerar o processo. Afirmamos que o
89             //metodo SOR

```

```

87         //seria apenas o metodo de Gauss-Seidel melhorado. O Gauss-Seidel
88         eh o SOR
89         //considerando w = 1.
90         //_____
91         xKmais1[i] *= w; //multiplicando o w como diz a formula.
92         xKmais1[i] += (1 - w) * xK[i]; // somando com (1 - w) * x(k) como
93         diz a formula.
94         //_____
95
96         xK2[i] = xK[i]; //guardar o valor da iteracao k para efeito de
97         teste de parada.
98         xK[i] = xKmais1[i]; //x(k) sera x(k + 1) para ser usado na mesma
99         iteracao.Nesta linha encontramos
100        //uma semelhanca com o Gauss-Seidel, pois atualizamos o valor de x(
101        k) na mesma iteracao.
102    }
103
104    for(h = 0; h < 3; h++)
105    {
106        //se x(k) for diferente de 0, iremos verificar o criterio de
107        parada.
108        if(xK2[h] != 0)
109        {
110            //Se cumprirmos a condicao, a iteracao sera continuada.
111            if((fabs(xKmais1[h] - xK2[h]) / fabs(xK2[h])) > p)
112            {
113                break;
114            }
115        }
116
117        //Se chegarmos no eltimo valor de x e o loop nao for rompido, entao
118        podemos dizer
119        //que o criterio de parada foi cumprido e sairemos do loop
120        principal.
121        if(h == 2 && xK2[h] != 0)
122        {
123            parada = 1;
124        }
125    }
126
127    //
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
26
```

```

130     return 0;
131 }

```

Listing 19: resolução do sistema pelo método iterativo SOR com o ω ótimo

Exercício 18

Para a resolução desta questão foi utilizada a fórmula de baskara para achar as raízes do polinomio caracteristico. Isso foi possível utilizando o determinante de uma matriz 2x2 para encontrar um polinômio genérico para obter a, b e c

Sendo:

$$a = -1$$

$$b = a_{00} + a_{11}$$

$$c = a_{10} \cdot a_{01} - a_{00} \cdot a_{11}$$

Aplicando a formula de baskara com esses termos genericos é possível achar as raízes do polinômio caracteristico de qualquer matriz de ordem 2

- Código de implementação :

```

1  #include<stdio.h>
2  #include<math.h>
3
4  float Delta(float A[2][2])
5  {
6      //formula geral de um delta da formula de baskara aplicado no polinomio
        //caracteristico de uma matriz generica 2x2
7      return pow(A[0][0]+A[1][1], 2) - 4*(-1)*(A[1][0]*A[0][1] - A[0][0]*A[1][1]);
8  }
9
10 int main()
11 {
12     float A[2][2];
13
14     int i, j;
15     for (i=0; i<2; i++)
16         for (j=0; j<2; j++)
17             scanf("%f", &A[i][j]);
18
19     //vetor com as duas raizes
20     float lambida [2];
21     //calculando o delta da baskara
22     float delta = Delta(A);
23     if (delta >= 0) //se maior que zero existe raiz real
24     {
25         // -b + raiz(delta)/2a

```

```

26     lambida[0] = ( -(A[0][0]+A[1][1]) + sqrt( delta ) )/-2 );
27     // -b - raiz(delta)/2a
28     lambida[1] = ( -(A[0][0]+A[1][1]) - sqrt( delta ) )/-2 );
29
30     printf("raiz 1: %f      raiz 2: %f\n", lambida[0], lambida[1]);
31 }
32 else
33 {
34     printf("delta negativo, n o h raiz real");
35 }
36
37 }

```

Listing 20: encontrar as raízes do polinômio característico de uma matriz 2x2 qualquer

Exercício 19

Inicialmente, pedimos ao usuário para inserir a matriz 3x3 desejada para o cálculo do polinômio característico. Com isso iniciaremos nossos cálculos.

Dividiremos o algoritmo em trs passos:

Passo 1 : Calcular o determinante da matriz 3x3.

Para o cálculo, usaremos uma equação genérica do determinante de uma matriz 3x3:

$$A = \begin{bmatrix} a - \lambda & b & c \\ d & e - \lambda & f \\ g & h & i - \lambda \end{bmatrix}$$

na qual foi usado o método de Sarrus, para encontraremos :

$$P(\lambda) = -\lambda^3 + j \cdot \lambda^2 + k \cdot \lambda + l$$

Sendo:

$$j = a + e + i$$

$$k = cg + hf + db - ae - ai - ei$$

$$l = cdh + bfg + aeicgeahfbdi$$

Cada um desses coeficientes sero representados como "vetorCoeficientesPolinomio" no código:

```
vetorCoeficientesPolinomio[0] = -1;
```

```
vetorCoeficientesPolinomio[1] = j;
```

```
vetorCoeficientePolinomio[2] = k;  
vetorCoeficientePolinomio[3] = l;
```

Passo 2 : Calcular uma raiz do polinômio.

Usaremos o método iterativo de Newton-Raphson para encontrar uma raiz aproximada. Depois, usaremos essa raiz para calcular as outras duas raízes, através da fórmula de resolução do 2º grau (conhecida como fórmula de Bhaskara).

Observação: Calculamos a derivada de um polinômio (em especial a de 3º grau), da seguinte forma:

$$f'(x) = a \cdot (3x^2) + b \cdot (2x) + c$$

Onde :

```
a = vetorCoeficientePolinomio[0];  
b = vetorCoeficientePolinomio[1];  
c = vetorCoeficientePolinomio[2].
```

A iteração do método de Newton-Raphson se dá pela fórmula:

$$x_{k+1} = \frac{x_k - f(x_k)}{f'(x_k)}$$

Sendo:

```
 $x_{k+1}$  = valor de x na iteração k + 1;  
 $x_k$  = valor de x na iteração k;  
 $f(x_k)$  = valor de  $f(x)$ , quando  $x = x_k$ ;  
 $f'(x_k)$  = valor da derivada de  $f(x)$ .
```

Pediremos ao usuário a inserção de um critério de parada para a iteração de Newton-Raphson e o valor inicial de x_k para a realização da primeira iteração de Newton-Raphson. Logo em seguida, iniciaremos a iteração.

Adotamos o critério de parada através:

$$\frac{|x_{k+1} - x_k|}{|x_k|} \leq (\text{critério de parada fornecido pelo usuário})$$

Quando a condição for cumprida, a iteração será interrompida e a primeira raiz do polinômio será o valor de x_{k+1} atual.

Passo 3 : Calcular as duas raízes restantes.

Usaremos as equações de obtenção de raízes do 2º grau. Mas antes disso, precisamos reduzir o grau do polinômio de grau 3 para grau 2.

Pelo teorema de D’Lambert, podemos fatorar qualquer polinômio se tivermos suas raízes, da seguinte forma (no caso em um polinômio de 3º grau:)

$$f(x) = (x - x') \cdot q(x)$$

Onde: $q(x)$ será nosso polinômio de 2º grau.

Para encontrá-lo, basta dividir o polinômio de 3º grau por $(x - x')$ (onde x' é uma das raízes do polinômio de 3º grau), pelo algoritmo da divisão de Euclides.

Realizando a divisão, teremos o resultado genérico:

$$P(x) = -x^2 + (b - x') \cdot x + (-(x')^2 + x' \cdot b + c)$$

Sendo:

```
a = vetorCoeficientesPolinomio[0];  
b = vetorCoeficientesPolinomio[1];  
c = vetorCoeficientesPolinomio[2].
```

Observao: Sabemos que a raiz a ser utilizada não será a raiz exata, então obteremos no final um resto da divisão.

Como todas as raízes calculadas aqui serão aproximadas e o resto apresentará um valor nfmio, o resto será desprezado para que os cálculos sejam possíveis.

Com a equao do 2º grau encontrada, poderemos aplicar a fórmula de Bhaskara e encontrar as duas raízes restantes. Com todas as raízes determinadas, exibiremos o resultado na tela.

- Código de implementação :

```
1 #include <stdio.h>  
2 #include <math.h>  
3  
4 int main()  
5 {  
6     int i, j;  
7  
8     double Matriz[3][3];  
9     double vetorLambda[3];  
10  
11     printf("Qual a matriz 3x3?\n");  
12  
13     for(i = 0; i < 3; i++)  
14     {  
15         for(j = 0; j < 3; j++)
```



```

16         {
17             scanf("%lf",&Matriz[i][j]);
18         }
19     }
20
21     //Inicio do calculo do polinomio caracteristico. Sera dividido em passos:
22
23     //Passo 1: Calcular o determinante da matriz 3x3.
24
25     double vetorCoeficientesPolinimio[4];
26
27     //Calculo do coeficiente 1
28     vetorCoeficientesPolinimio[0] = -1;
29
30     //Calculo do coeficiente 2
31     vetorCoeficientesPolinimio[1] = Matriz[0][0] + Matriz[1][1] + Matriz[2][2];
32
33     //Calculo do coeficiente 3
34
35     vetorCoeficientesPolinimio[2] = Matriz[2][0] * Matriz[0][2];
36     vetorCoeficientesPolinimio[2] += Matriz[2][1] * Matriz[1][2];
37     vetorCoeficientesPolinimio[2] += Matriz[1][0] * Matriz[0][1];
38     vetorCoeficientesPolinimio[2] -= Matriz[0][0] * Matriz[2][2];
39     vetorCoeficientesPolinimio[2] -= Matriz[1][1] * Matriz[2][2];
40     vetorCoeficientesPolinimio[2] -= Matriz[0][0] * Matriz[1][1];
41
42     //Calculo do coeficiente 4
43
44     vetorCoeficientesPolinimio[3] = Matriz[0][2] * Matriz[1][0] * Matriz[2][1];
45     vetorCoeficientesPolinimio[3] += Matriz[0][1] * Matriz[1][2] * Matriz
46         [2][0];
47     vetorCoeficientesPolinimio[3] += Matriz[0][0] * Matriz[1][1] * Matriz
48         [2][2];
49     vetorCoeficientesPolinimio[3] -= Matriz[2][0] * Matriz[0][2] * Matriz
50         [1][1];
51     vetorCoeficientesPolinimio[3] -= Matriz[2][1] * Matriz[1][2] * Matriz
52         [0][0];
53     vetorCoeficientesPolinimio[3] -= Matriz[1][0] * Matriz[0][1] * Matriz
54         [2][2];
55
56     //Passo 2: Calcular uma raiz do polinomio.
57
58     //Critério de parada da iteracao
59     double criterioParada;
60     printf("Qual o critério de parada para o método de Newton-Raphson?\n");
61     scanf("%lf",&criterioParada);
62
63     //Valor inicial de x(k) na primeira iteracao
64     double xK;
65     printf("Qual o valor inicial de x para a iteração?\n");
66     scanf("%lf",&xK);

```

```

64     int parada = 0; // variavel de controle de parada. Quando parada = 1, a
        iteracao sera desfeita.
65     double xKmais1 = 0; // x(k + 1)
66
67     double derivadaFk = 0;
68     double Fk = 0;
69     double xK2; //sera a memoria do valor antigo de xK. Servira para efeito de
        testes de parada da iteracao
70         //de Newton-Raphson.
71
72     do
73     {
74         //Calcular f(x(k)): O valor do polinomio caracteristico quando (lambda
            ) = x(k).
75
76         Fk += vetorCoeficientesPolinimio[0] * pow(xK,3);
77         Fk += vetorCoeficientesPolinimio[1] * pow(xK,2);
78         Fk += vetorCoeficientesPolinimio[2] * xK;
79         Fk += vetorCoeficientesPolinimio[3];
80
81         //Calcular f'(x(k)): Usando a formula da derivada encontrada.
82
83         derivadaFk += vetorCoeficientesPolinimio[0] * (3 * pow(xK,2));
84         derivadaFk += vetorCoeficientesPolinimio[1] * (2 * xK);
85         derivadaFk += vetorCoeficientesPolinimio[2];
86
87         //Calcular x(k + 1)
88
89         xKmais1 = xK - (Fk / derivadaFk); //iteracao de Newton-Raphson.
90
91         xK2 = xK; //guardar o valor antigo de xK para ver se devemos terminar a
            iteracao.
92         xK = xKmais1; //xK sera agora o x(k + 1) na proxima iteracao.
93
94         Fk = 0; //Zerar as variaveis para iniciarmos a proxima
            iteracao.
95         derivadaFk = 0;
96
97         //Teste de parada: Usando a formula de criterio de parada ja citada.
98         if(fabs(xKmais1 - xK2) / fabs(xK2) <= criterioParada)
99             parada = 1;
100
101
102     }while(parada == 0);
103
104     vetorLambda[0] = xKmais1; //A primeira raiz recebe o valor de x(k + 1)
105
106
107
108     //Passo 3: Calcular as duas raizes restantes.
109
110     //Coeficientes da equacao do 2 grau.
111     double a = -1;

```

```
112     double b = vetorCoeficientesPolinimio[1] - vetorLambda[0];
113     double c = (-1 * pow(vetorLambda[0],2)) + (vetorLambda[0] *
        vetorCoeficientesPolinimio[1]) + vetorCoeficientesPolinimio[2];
114
115     //Calculo do delta
116     double delta = pow(b,2) - (4 * a * c);
117
118     //Calculo das raizes x1 e x2
119     vetorLambda[1] = (-1 * b + sqrt(delta)) / 2 * a;
120     vetorLambda[2] = (-1 * b - sqrt(delta)) / 2 * a;
121
122     //
123
124     //Resultados
125     printf("Raizes do polin mio:\n");
126     printf("x1 = %f\n",vetorLambda[0]);
127
128     //Se o delta for menor que zero , a equacao do segundo grau nao tera raizes
        reais!
129
130     if(delta >= 0)
131     {
132         printf("x2 = %f\n",vetorLambda[1]);
133         printf("x3 = %f\n",vetorLambda[2]);
134     }
135     else
136     {
137         printf("N o h raizes reais para x2 e x3!\n");
138     }
139
140     return 0;
141 }
```

Listing 21: raízes do polinômio característico de uma matr z $A_{3 \times 3}$

Exercício 20

Neste exercício foi proposto a implementação da PSO (*Particle Swarm Optimization*), que será vulgarmente chamada aqui nesta explicação como "colônia de morcegos".

Iremos iniciar uma explicação breve do que ocorre no código desta questão.

Nossa lógica da colônia foi voltada particularmente a resolução de sistemas lineares.

A PSO disponibiliza para nós duas fórmulas que serviam de base ao nosso algoritmo: uma que calcula a velocidade do morcego na próxima iteração e outra que calcula a posição do morcego na

próxima iteração. Serão suficientes para nossos cálculos.

Foi feito no código para auxiliar a manipulação uma *struct* (estrutura) contendo as informações que o morcego terá: posição, melhor posição, velocidade, função de fitness e a melhor função de fitness (necessária para a atualização da melhor posição do morcego). Além da estrutura, teremos uma função exclusiva que irá calcular a função de fitness do morcego a cada iteração.

Em andamento, iniciaremos o código principal. Pediremos ao usuário que insira a ordem do sistema, a matriz do sistema e o vetor solução. A quantidade de morcegos que será usada equivalerá a trinta por cento do total de equações (ordem) que o sistema tem; caso o valor desse resultado resulte em um valor menor que 2, usaremos 2 morcegos na busca da solução, pois não faz sentido colocarmos apenas um morcego (os cálculos da melhor posição do bando ficariam constantes e também perderíamos o real significado da PSO: trabalhar em grupo). Partiremos para a criação das variáveis do programa: o vetor contendo os morcegos, a matriz que contém as velocidades de cada morcego, a matriz que tem as melhores posições dos morcegos, a melhor função de fitness dentre todas as iterações e o vetor da melhor posição do bando dentre todas as iterações. Criaremos os morcegos, geraremos os números aleatoriamente para os vetores posição (valores entre 0 e 20) e velocidade (valores entre 0 e 3), calcularemos a função de fitness inicial para cada morcego e veremos a melhor posição do bando inicialmente.

Comçaremos agora a iteração do algoritmo, anteriormente criando as variáveis A_1, A_2, A_3, c_1, c_2 (ambos serão sempre iguais a 2), tempo (constante igual a 1) e o critério de parada que o usuário fornecerá. Os passos seguintes determinarão o que será feito em ordem em cada iteração:

Passo 1 : Gerar aleatoriamente A_1, A_2, A_3 , no intervalo entre 0 e 1;

Passo 2 : Atualizar o vetor de velocidade dos morcegos, usando a fórmula fornecida pela colônia de morcegos, com a diferença que a melhor posição do bando será em relação a todas as iterações já feitas, e não da iteração passada;

Passo 3 : Atualizar as posições de cada morcego;

Passo 4 : Calcular a função de fitness de cada morcego;

Passo 5 : Atualizar a melhor posicao do bando dentre todas as iterações;

Passo 6 : Atualizar a melhor posição de cada morcego; Passo 7: Verificar o critério de parada a qual será usada a melhor função fitness de todas as iterações.

Abriremos uma observação a respeito do critério de parada: quanto menor for este valor, maior será o tempo de execução do programa, mas não significa que o programa não funcionará, a convergência do algoritmo só será mais lenta, isto porque a ideia de um método iterativo é dar uma solução aproximada do sistema e, dependendo de quanto menor for o erro relativo aceitável, maior o tempo necessário de execução; ainda mais nesta proposta da colônia de morcegos o qual não há garantia de convergência do método.

Cumprindo o critério de parada, a solução do sistema será exibida.

- Código de implementação :

```
1 #include <stdio.h>
2 #include <math.h>
3 #include <time.h>
4 #include <stdlib.h>
5
6 //Estrutura contendo os elementos da PSO de um morcego.
7 //Obs: Iremos trabalhar as informacoes do morcegos com ponteiro , ou seja , para
   podermos atribuir
8 //valores a eles , precisaremos dar-lhes os enderecos das matrizes onde estar o
   armazenadas os vetores
9 //de posicao , velocidade e melhor posicao de cada morcego.
10 typedef struct{
11
12     double* posicao;
13     double* melhorPosicao;
14     double* velocidade;
15     double funcaoFitness;
16     double melhorFuncaoFitness;
17
18 } Morcego;
19
20 //Calcula a funcao de fitness de um morcego.
21 double CalcularFuncaoFitness(Morcego morcego, double* enderecoA, double*
   enderecoB, int ordem)
22 {
23     int i, j, k = 0, h = 0; //em geral trabalharemos com essas variaveis para
   fazer os loops e
24                               //para varer cada valor dos vetores que o morcego
   contem
25                               //(posicao , velocidade , melhor posicao).
26     double somatorio = 0;
27     double A[ordem][ordem];
28     double b[ordem];
29
30     for(i = 0; i < ordem; i++)
31     {
32         for(j = 0; j < ordem; j++)
33         {
34             A[i][j] = *(enderecoA + k);
```

```
35         k++;
36     }
37 }
38
39 for(i = 0; i < ordem; i++)
40 {
41     b[i] = *(enderecoB + i);
42 }
43
44 morcego.funcaoFitness = 0;
45
46 for(i = 0; i < ordem; i++)
47 {
48     somatorio += b[i];
49
50     for(j = 0; j < ordem; j++)
51     {
52         somatorio -= *(morcego.posicao + h) * A[i][j];
53         h++;
54     }
55
56     h = 0; //variavel necessaria para varrer os vetores dos morcegos.
57           //se zera aqui para iniciar uma nova varredura.
58
59     morcego.funcaoFitness += fabs(somatorio);
60     somatorio = 0; //somar a diferenca da proxima equacao do sistema.
61 }
62
63 return morcego.funcaoFitness;
64 };
65
66 /*Como gerar numeros aleatorios em ponto flutuante:
67 Seja ]a;b[ o intervalo com o qual se deseja gerar o conjunto de numeros
68 aleatorios.
69 Ent o , o "aleatorio" gerado sera igual a:
70     a + (b - a) * ((double)(rand())/RAND_MAX)
71 Com isso , geraremos numeros de ponto flutuante aleatorios no intervalo ]a;b[.
72 */
73
74 int main()
75 {
76     int i,j,k = 0,h = 0; //variaveis de iteracao
77     int ordem; //ordem do sistema.
78
79     srand((double)time(NULL)); //gerador de semente, para gerarmos sequencias
80                               //de
81                               //numeros aleatorios diferentes a cada vez que
82                               //o
83                               //programa for iniciado.
84
85     printf("Qual a ordem do sistema?\n");
86     scanf("%d",&ordem);
```

```
84
85     double A[ordem][ordem];
86
87     printf("Qual a matriz A do sistema?\n");
88     for(i = 0; i < ordem; i++)
89     {
90         for(j = 0; j < ordem; j++)
91         {
92             scanf("%lf", &A[i][j]);
93         }
94     }
95
96     double b[ordem];
97
98     printf("Qual o vetor b do sistema?\n");
99     for(i = 0; i < ordem; i++)
100    {
101        scanf("%lf", &b[i]);
102    }
103
104    //Necessario para obtermos a matriz A e b nas funcoes que futuramente
105    //iremos utilizar em nossos calculos.
106    double* enderecoA = &A[0][0];
107    double* enderecoB = &b[0];
108
109    //Criando os "morcegos"
110    //Obs: A funcao round() retorna o valor arredondado do parametro fornecido.
111    //Neste caso, a quantidade de morcegos sera igual a 30% do numero de
112    //equacoes existentes.
113    int quantidadeDeMorcegos = round(0.3 * ordem);
114
115    //Usaremos no minimo 2 morcegos para a iteracao, caso os 30% do numero de
116    //equacoes
117    //seja menor que 2.
118    if(quantidadeDeMorcegos < 2)
119        quantidadeDeMorcegos = 2;
120
121    //Cada linha da matriz das caracteristicas de cada morcego
122    //ira representar o numero do morcego; as colunas irao
123    //representar os valores contidos no vetor daquele morcego.
124    //Essas matrizes seriam como se fossem "Banco de Dados" para
125    //armazenar essas informacoes.
126
127    Morcego morcegos[quantidadeDeMorcegos]; //vetor com todos os morcegos
128    //disponiveis.
129    double posicoesMorcego[quantidadeDeMorcegos][ordem]; //vetor de posicoes dos
130    //morcegos, necessario para gerarmos
131    //aleatoriamente os
132    //valores.
133    double velocidadesMorcegos[quantidadeDeMorcegos][ordem]; //vetor de
134    //velocidades dos morcegos, necessario para gerarmos
135    //aleatoriamente os
136    //valores.
```

```
130     double melhorPosicaoMorcego[quantidadeDeMorcegos][ordem]; //Melhores
131         posicoes de cada morcego.
132
133     //double melhorPosicaoBando[ordem]; //melhor posicao do bando.
134
135     double melhorFuncaoFitness = 10000000000; //foi iniciada com este valor
136         alto pois sera logo substituida pela melhor
137         //funcao de fitness entre os
138         morcegos.
139
140     double melhorPosicaoBandoTodasAsIteracoes[ordem]; //melhor posicao do bando
141         dentre todas as iteracoes.
142
143     //Criando os morcegos
144
145     for(i = 0; i < quantidadeDeMorcegos; i++)
146     {
147         Morcego morcego;
148         morcegos[i] = morcego;
149     }
150
151     //Criando os valores das posições dos morcegos.
152     //Aquele funcao ali dentro ir gerar aleatorios no intervalo de 0 a 20.
153
154     srand((double)time(NULL));
155
156     for(k = 0; k < quantidadeDeMorcegos; k++)
157     {
158         for(j = 0; j < ordem; j++)
159         {
160             posicoesMorcego[k][j] = 20 * ((double)(rand())/RANDMAX);
161         }
162
163         morcegos[k].posicao = &posicoesMorcego[k][0]; //envio do endereco da
164             linha da matriz
165
166             //que contera as
167             informacoes do
168             morcego.
169     }
170
171     //Criando os valores de velocidade dos morcegos entre 0 e 3.
172
173     for(k = 0; k < quantidadeDeMorcegos; k++)
174     {
175         for(j = 0; j < ordem; j++)
176         {
177             velocidadesMorcegos[k][j] = 3 * ((double)(rand())/RANDMAX);
178         }
179
180         morcegos[k].velocidade = &velocidadesMorcegos[k][0]; //envio do
181             endereco da linha da matriz
182
183             //que contera
184             as
```



```
informacoes
do morcego.

174     }
175
176 //Obtendo a função de fitness inicial para cada morcego.
177
178 for(i = 0; i < quantidadeDeMorcegos;i++)
179 {
180     morcegos[i].funcaoFitness = CalcularFuncaoFitness(morcegos[i],enderecoA
181     ,enderecoB,ordem);
182     morcegos[i].melhorFuncaoFitness = morcegos[i].funcaoFitness;
183 }
184 //Obter a melhor posicao do bando inicialmente
185
186 for(i = 0;i < quantidadeDeMorcegos;i++)
187 {
188     for(j = 0;j < quantidadeDeMorcegos;j++)
189     {
190
191         if(i != j) //Nao faria sentido comparar o morcego com ele mesmo.
192         {
193             if(morcegos[i].funcaoFitness < morcegos[j].funcaoFitness)
194             {
195                 melhorFuncaoFitness = morcegos[i].funcaoFitness;
196
197                 for(h = 0;h < ordem;h++)
198                 {
199                     //melhorPosicaoBando[h] = *(morcegos[i].posicao + h);
200                     melhorPosicaoBandoTodasAsIteracoes[h] = *(morcegos[i].
201                         posicao + h);
202                 }
203             }
204         }
205     }
206
207 for(i = 0;i < quantidadeDeMorcegos;i++)
208 {
209     morcegos[i].melhorPosicao = &melhorPosicaoMorcego[i][0];
210 }
211
212 //Atualizar a melhor posicao de cada morcego inicialmente.
213
214 for(i = 0;i < quantidadeDeMorcegos;i++)
215 {
216     for(j = 0;j < ordem;j++)
217     {
218         *(morcegos[i].melhorPosicao + j) = *(morcegos[i].posicao + j);
219     }
220 }
221
222
```

```
223 //Variaveis da formula da PSO
224 double A1,A2,A3; //irao ser gerados aleatoriamente sempre.
225 double c1 = 2,c2 = 2; //Considerados sempre iguais a 2, por definicao.
226 double tempo = 1; //considerado sempre igual a 1, por definicao.
227
228 double criterioParada;
229 printf("Qual o crit rio de parada para o algoritmo?\n");
230 scanf("%lf",&criterioParada);
231
232
233 //Comecaremos a nossa iteracao...(Consideramos aqui que a variacao do tempo
    sempre sera igual a 1)
234 //e os valores constantes de c1 e c2 serao iguais a 2, respectivamente.
235
236 int parada = 0; //quando parada = 1, o loop sera rompido.
237
238 while(parada == 0)
239 {
240     //Passo 1: Gerar aleatoriamente A1,A2,A3, entre 0 e 1.
241
242     A1 = 1 * ((double)(rand())/RAND.MAX);
243     A2 = 1 * ((double)(rand())/RAND.MAX);
244     A3 = 1 * ((double)(rand())/RAND.MAX);
245
246     //Passo 2: Atualizar o vetor de velocidade dos morcegos.
247
248     //Aplicando a formula da velocidade do morcego na iteracao.
249     for(i = 0; i < quantidadeDeMorcegos;i++)
250     {
251         for(k = 0; k < ordem;k++)
252         {
253             //A1 * V na iteracao i do morcego q
254
255             *(morcegos[i].velocidade + k) = *(morcegos[i].velocidade + k) *
                A1;
256
257             //A2 * c1 * (melhor posicao do morcego q * posicao do morcego q
                na iteracao i) / tempo
258
259             *(morcegos[i].velocidade + k) += A2 * c1 * *(morcegos[i].
                velocidade + k) + *(morcegos[i].velocidade + k) + ((*
                morcegos[i].melhorPosicao + k) - *(morcegos[i].posicao + k))
                / tempo);
260
261             //A3 * c2 * (melhor posicao do bando na iteracao i * posicao do
                morcego q na iteracao i) / tempo
262             //Obs: Usaremos nesse caso a melhor posicao do bando dentre
                todas ja feitas e nao a melhor de cada
263             //iteracao. Mudancas necessarias para o sucesso de convergencia
                do algoritmo.
264
265             *(morcegos[i].velocidade + k) += A3 * c2 * (
                melhorPosicaoBandoTodasAsIteracoes[k] - *(morcegos[i].
```

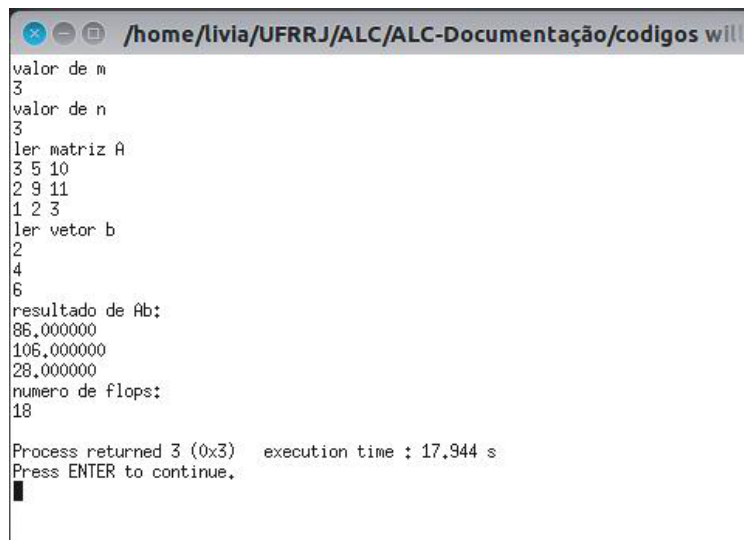
```
266         posicao + k) / tempo);
267
268         //Realizar o controle da velocidade para a mesma nao aumentar
269         //pois fugir do contexto da realidade.
270         if(*(morcegos[i].velocidade + k) >= 2)
271         {
272             *(morcegos[i].velocidade + k) = 3 * ((double)(rand())/
273             RANDMAX);
274         }
275     }
276
277     //Passo 3: Atualizar as posicoes de cada morcego.
278
279     //Aplicando a formula da posicao do morcego na iteracao.
280     for(i = 0; i < quantidadeDeMorcegos; i++)
281     {
282         //S(q)(i + 1) = S(q)(i) + V(q)(i + 1) * tempo
283         for(k = 0; k < ordem; k++)
284         {
285             *(morcegos[i].posicao + k) = *(morcegos[i].posicao + k) + *(
286             morcegos[i].velocidade + k) * tempo;
287         }
288     }
289
290     //Passo 4: Calcular a funcao de fitness de cada morcego.
291
292     for(i = 0; i < quantidadeDeMorcegos; i++)
293     {
294         morcegos[i].funcaoFitness = CalcularFuncaoFitness(morcegos[i],
295         enderecoA, enderecoB, ordem);
296     }
297
298     //Passo 5: Atualizar a melhor posicao do bando dentre todas as
299     iteracoes
300
301     for(i = 0; i < quantidadeDeMorcegos; i++)
302     {
303         if(morcegos[i].funcaoFitness < melhorFuncaoFitness)
304         {
305             melhorFuncaoFitness = morcegos[i].funcaoFitness;
306
307             for(j = 0; j < ordem; j++)
308             {
309                 melhorPosicaoBandoTodasAsIteracoes[j] = *(morcegos[i].
310                 posicao + j);
311             }
312         }
313     }
314
315     //printf("Melhor funcao de fitness: %f\n", melhorFuncaoFitness);
```

```
312
313 //Passo 6: Atualizar a melhor posicao de cada morcego
314
315 for(i = 0; i < quantidadeDeMorcegos; i++)
316 {
317     if(morcegos[i].funcaoFitness < morcegos[i].melhorFuncaoFitness)
318     {
319         morcegos[i].melhorFuncaoFitness = morcegos[i].funcaoFitness;
320
321         for(j = 0; j < ordem; j++)
322         {
323             *(morcegos[i].melhorPosicao + j) = *(morcegos[i].posicao +
324                 j);
325         }
326     }
327
328 //Passo 7: Verificar o criterio de parada.
329 //Se a melhor funcao de fitness for menor que o criterio de parada,
330 //encontramos uma solucao aproximada.
331 if(melhorFuncaoFitness <= criterioParada)
332 {
333     parada = 1;
334 }
335 }
336
337 //Resultados
338
339 printf(" Solu o aproximada do sistema:\n");
340
341 for(i = 0; i < ordem; i++)
342 {
343     printf("x%d = %f\n", i + 1, melhorPosicaoBandoTodasAsIteracoes[i]);
344 }
345
346 return 0;
347 }
```

Listing 22: Algoritmo colonia de morcegos

Exemplos Questão 1

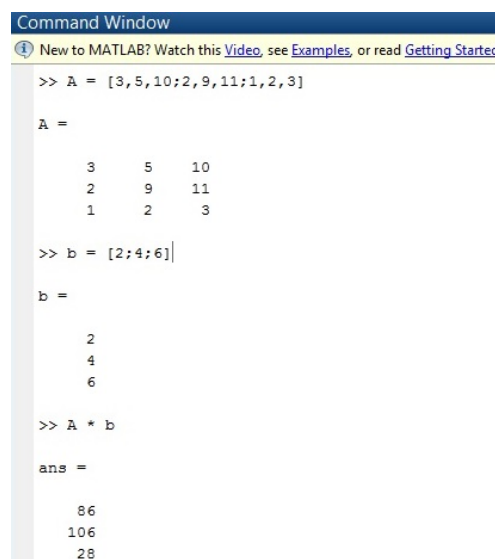
- TERMINAL



```
/home/livia/UFRJ/ALC/ALC-Documntação/codigos will
valor de m
3
valor de n
3
ler matriz A
3 5 10
2 9 11
1 2 3
ler vetor b
2
4
6
resultado de Ab:
86,000000
106,000000
28,000000
numero de flops:
18
Process returned 3 (0x3)   execution time : 17,944 s
Press ENTER to continue.
```

Figura 1: Exemplo.1 que mostra a execução do programa (quest01) no terminal

- MATLAB



```
Command Window
New to MATLAB? Watch this Video, see Examples, or read Getting Started.
>> A = [3,5,10;2,9,11;1,2,3]

A =

     3     5    10
     2     9    11
     1     2     3

>> b = [2;4;6]

b =

     2
     4
     6

>> A * b

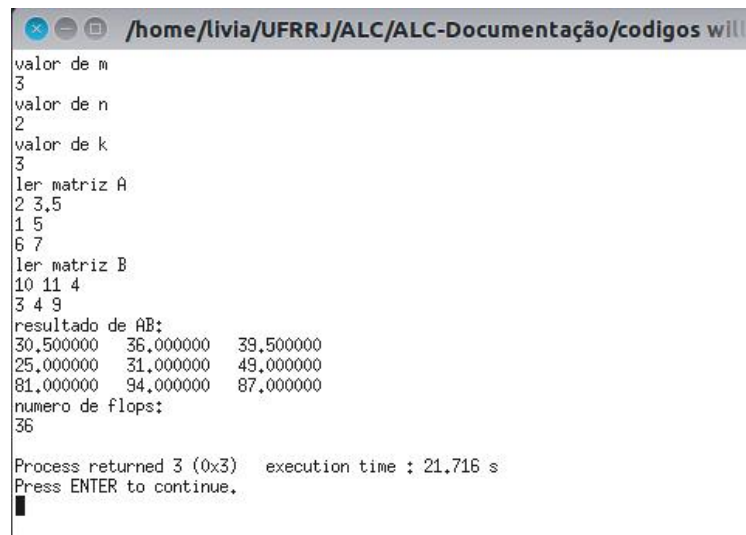
ans =

     86
    106
     28
```

Figura 2: Exemplo que mostra o resultado do MATLAB (quest01) para o exemplo.1

Exemplos Questão 2

- TERMINAL

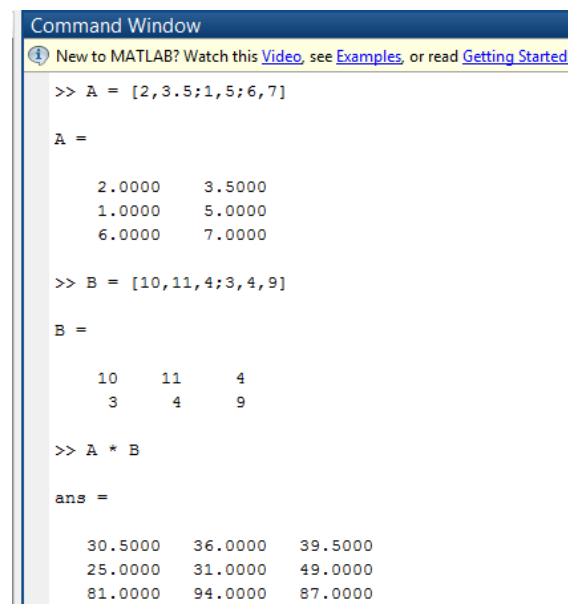


A terminal window with a title bar showing the path `/home/livia/UFRRJ/ALC/ALC-Documentação/codigos will`. The terminal displays the following text:

```
valor de m
3
valor de n
2
valor de k
3
ler matriz A
2 3.5
1 5
6 7
ler matriz B
10 11 4
3 4 9
resultado de AB:
30.500000 36.000000 39.500000
25.000000 31.000000 49.000000
81.000000 94.000000 87.000000
numero de flops:
36
Process returned 3 (0x3)   execution time : 21.716 s
Press ENTER to continue.
```

Figura 3: Exemplo.1 que mostra a execução do programa (quest02) no terminal

- MATLAB



A MATLAB Command Window screenshot. It shows the following commands and outputs:

```
>> A = [2,3.5;1,5;6,7]

A =

    2.0000    3.5000
    1.0000    5.0000
    6.0000    7.0000

>> B = [10,11,4;3,4,9]

B =

    10    11     4
     3     4     9

>> A * B

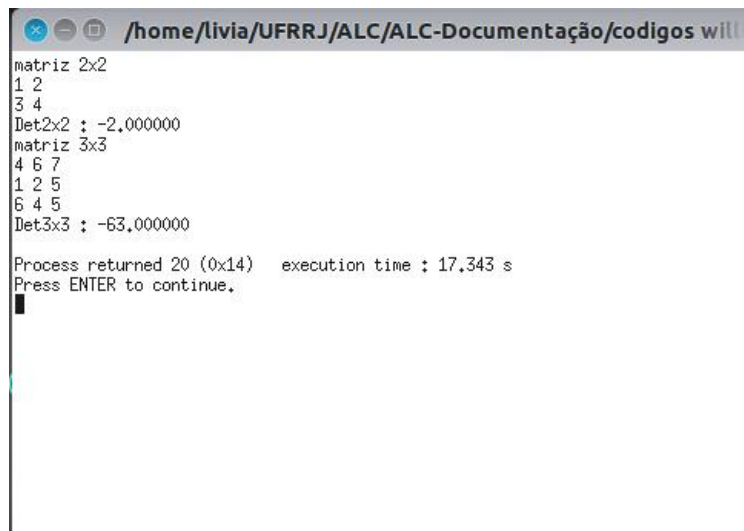
ans =

    30.5000    36.0000    39.5000
    25.0000    31.0000    49.0000
    81.0000    94.0000    87.0000
```

Figura 4: Exemplo que mostra o resultado do MATLAB (quest02) para o exemplo.1

Exemplos Questão 3

- TERMINAL



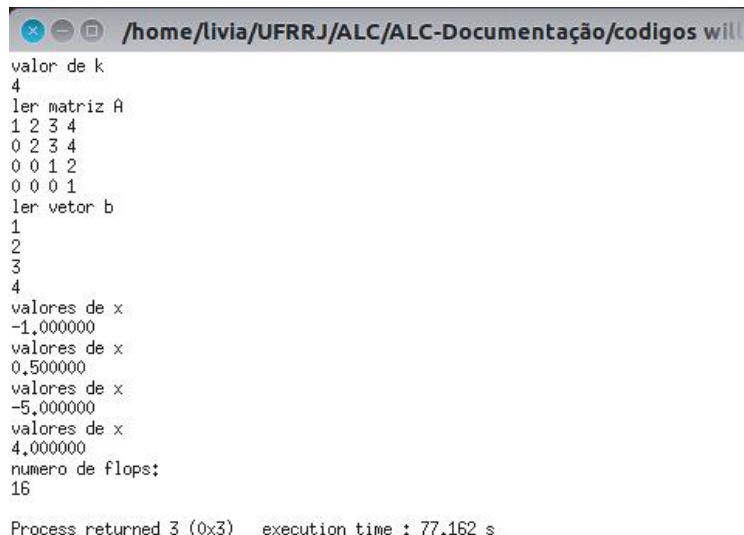
```
/home/livia/UFRRJ/ALC/ALC-Documentação/codigos will
matriz 2x2
1 2
3 4
Det2x2 : -2.000000
matriz 3x3
4 6 7
1 2 5
6 4 5
Det3x3 : -63.000000

Process returned 20 (0x14)   execution time : 17.343 s
Press ENTER to continue.
█
```

Figura 5: Exemplo.1 que mostra a execução do programa (quest03) no terminal

Exemplos Questão 4

- TERMINAL



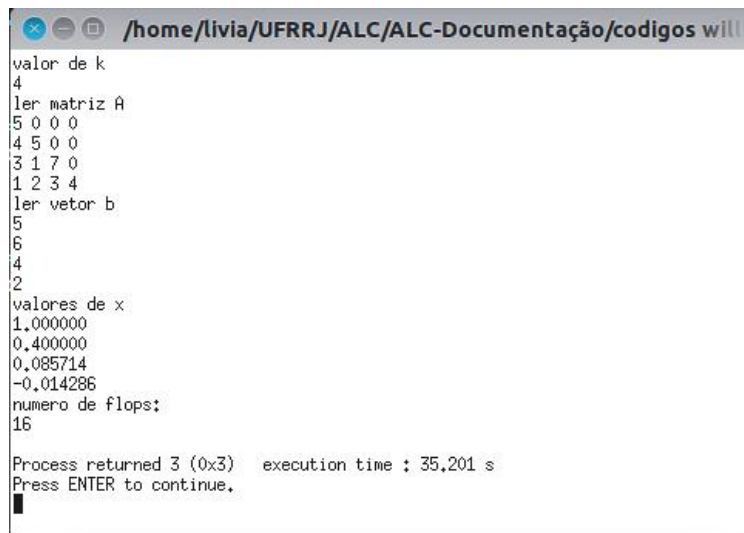
```
/home/livia/UFRRJ/ALC/ALC-Documentação/codigos will
valor de k
4
ler matriz A
1 2 3 4
0 2 3 4
0 0 1 2
0 0 0 1
ler vetor b
1
2
3
4
valores de x
-1.000000
valores de x
0.500000
valores de x
-5.000000
valores de x
4.000000
numero de flops:
16

Process returned 3 (0x3)   execution time : 77.162 s
```

Figura 6: Exemplo.1 que mostra a execução do programa (quest04) no terminal

Exemplos Questão 5

- TERMINAL



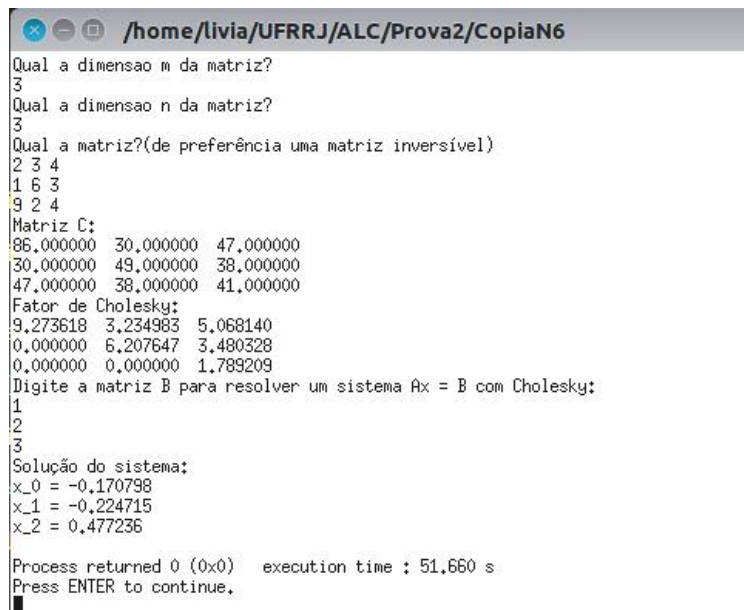
```
/home/livia/UFRRJ/ALC/ALC-Documantação/codigos will
valor de k
4
ler matriz A
5 0 0 0
4 5 0 0
3 1 7 0
1 2 3 4
ler vetor b
5
6
4
2
valores de x
1.000000
0.400000
0.085714
-0.014286
numero de flops:
16

Process returned 3 (0x3)   execution time : 35,201 s
Press ENTER to continue.
```

Figura 7: Exemplo.1 que mostra a execução do programa (quest05) no terminal

Exemplos Questão 6

- TERMINAL

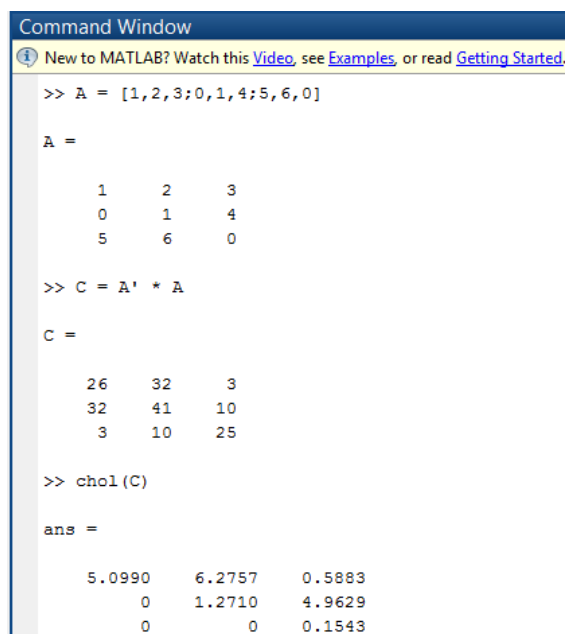


```
/home/livia/UFRRJ/ALC/Prova2/CopiaN6
Qual a dimensao m da matriz?
3
Qual a dimensao n da matriz?
3
Qual a matriz?(de preferência uma matriz inversível)
2 3 4
1 6 3
9 2 4
Matriz C:
86.000000 30.000000 47.000000
30.000000 49.000000 38.000000
47.000000 38.000000 41.000000
Fator de Cholesky:
9.273618 3.234983 5.068140
0.000000 6.207647 3.480328
0.000000 0.000000 1.789209
Digite a matriz B para resolver um sistema Ax = B com Cholesky:
1
2
3
Solução do sistema:
x_0 = -0.170798
x_1 = -0.224715
x_2 = 0.477236

Process returned 0 (0x0)   execution time : 51.660 s
Press ENTER to continue.
```

Figura 8: Exemplo.1 que mostra a execução do programa (quest06) no terminal

- MATLAB



```
Command Window
New to MATLAB? Watch this Video, see Examples, or read Getting Started.

>> A = [1,2,3;0,1,4;5,6,0]

A =

     1     2     3
     0     1     4
     5     6     0

>> C = A' * A

C =

    26    32     3
    32    41    10
     3    10    25

>> chol(C)

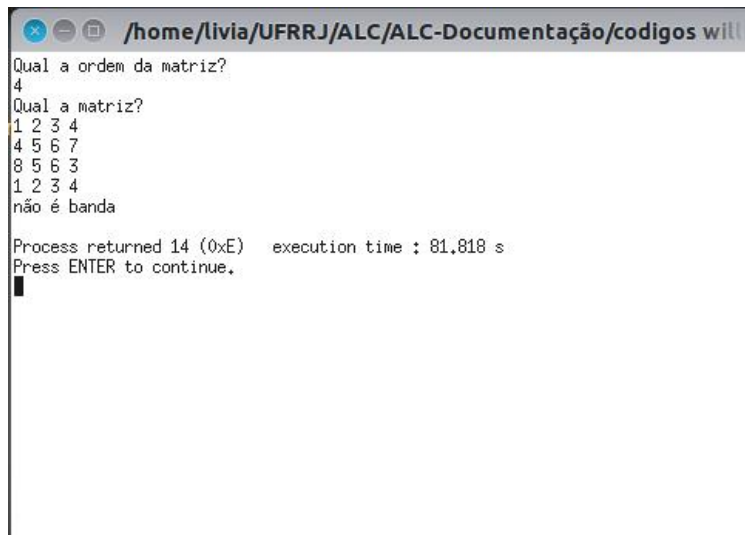
ans =

    5.0990    6.2757    0.5883
         0    1.2710    4.9629
         0         0    0.1543
```

Figura 9: Exemplo que mostra o resultado do MATLAB (quest06) para o exemplo.1

Exemplos Questão 7

- TERMINAL

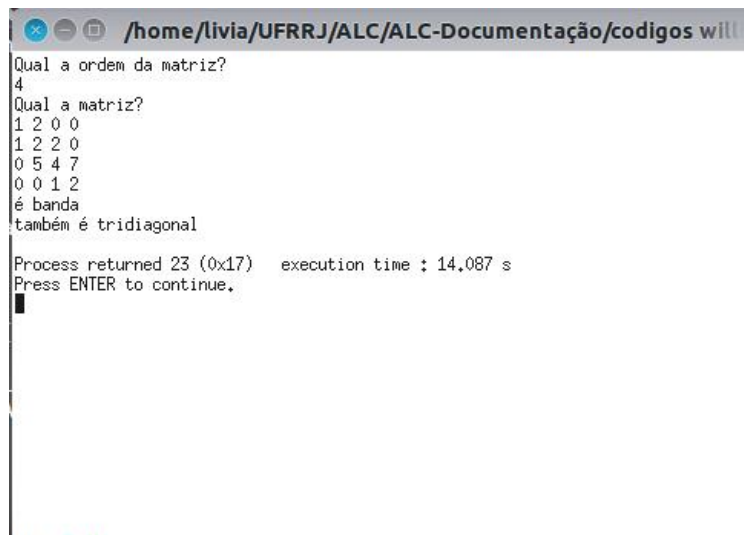


```
/home/livia/UFRRJ/ALC/ALC-Documentação/codigos will
Qual a ordem da matriz?
4
Qual a matriz?
1 2 3 4
4 5 6 7
8 5 6 3
1 2 3 4
não é banda

Process returned 14 (0xE)   execution time : 81.818 s
Press ENTER to continue.
```

Figura 10: Exemplo.1 que mostra a execução do programa (quest07) no terminal

- TERMINAL-2

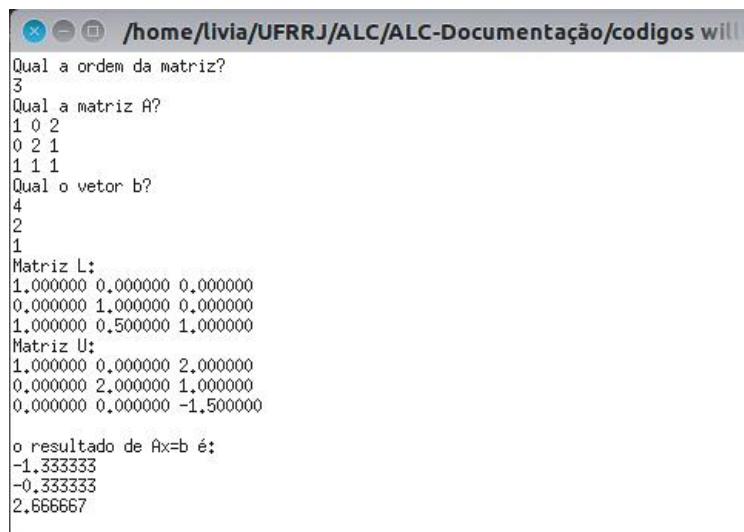


```
/home/livia/UFRRJ/ALC/ALC-Documantação/codigos will
Qual a ordem da matriz?
4
Qual a matriz?
1 2 0 0
1 2 2 0
0 5 4 7
0 0 1 2
é banda
também é tridiagonal
Process returned 23 (0x17)   execution time : 14.087 s
Press ENTER to continue.
```

Figura 11: Exemplo.2 que mostra a execução do programa (quest07) no terminal

Exemplos Questão 8

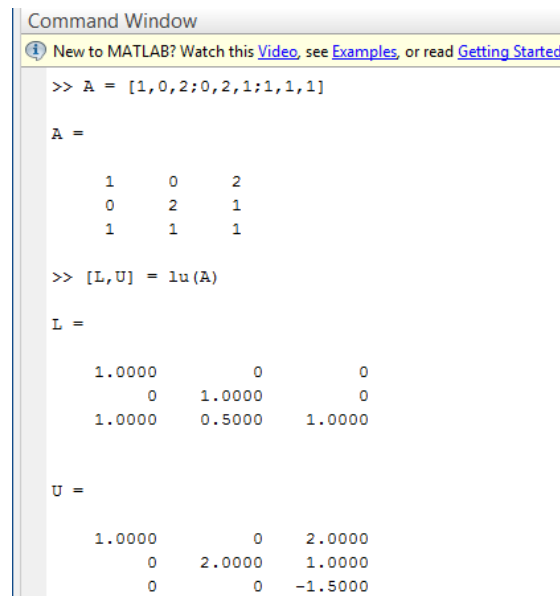
- TERMINAL-1



```
/home/livia/UFRRJ/ALC/ALC-Documantação/codigos will
Qual a ordem da matriz?
3
Qual a matriz A?
1 0 2
0 2 1
1 1 1
Qual o vetor b?
4
2
1
Matriz L:
1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
1.000000 0.500000 1.000000
Matriz U:
1.000000 0.000000 2.000000
0.000000 2.000000 1.000000
0.000000 0.000000 -1.500000
o resultado de Ax=b é:
-1.333333
-0.333333
2.666667
```

Figura 12: Exemplo.1 que mostra a execução do programa (quest08) no terminal

- MATLAB-1



```
Command Window
New to MATLAB? Watch this Video, see Examples, or read Getting Started.

>> A = [1,0,2;0,2,1;1,1,1]

A =

     1     0     2
     0     2     1
     1     1     1

>> [L,U] = lu(A)

L =

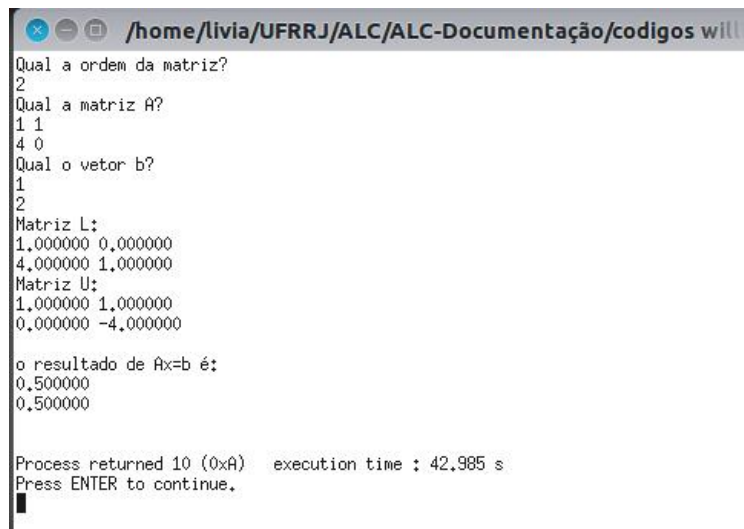
    1.0000         0         0
         0    1.0000         0
    1.0000    0.5000    1.0000

U =

    1.0000         0    2.0000
         0    2.0000    1.0000
         0         0   -1.5000
```

Figura 13: Exemplo que mostra o resultado do MATLAB (quest08) para o exemplo.1

- TERMINAL-2



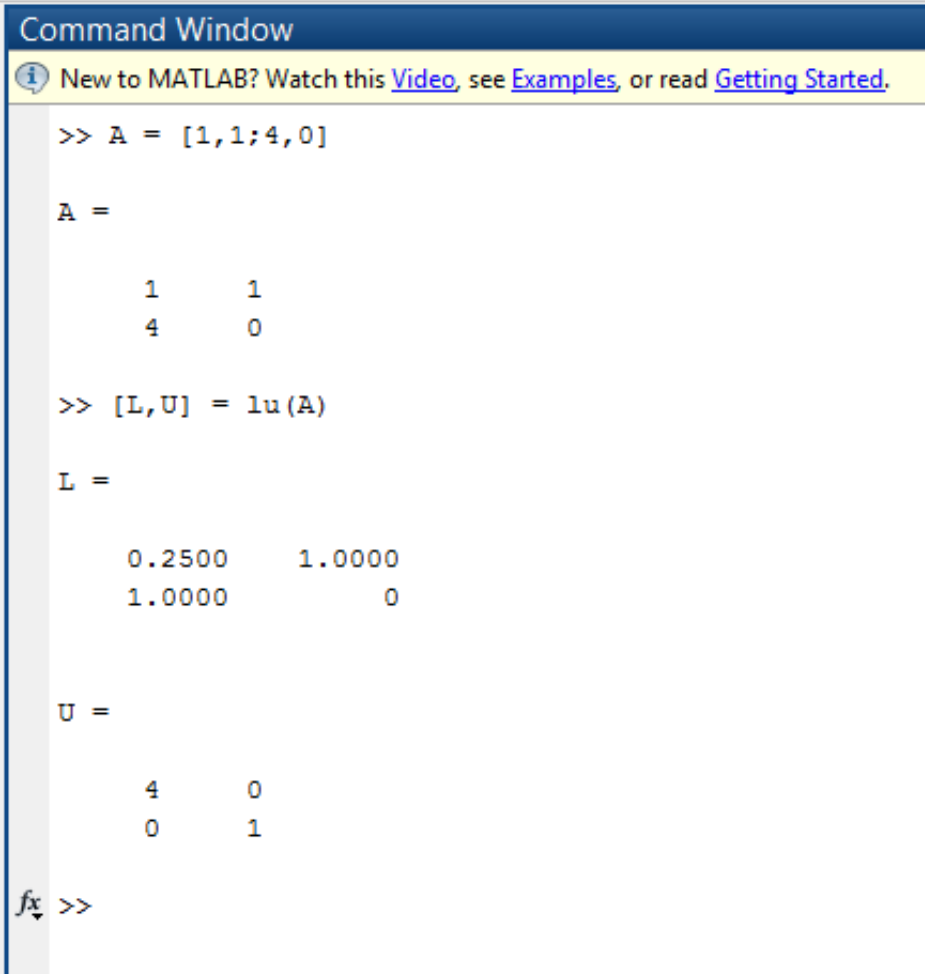
```
/home/livia/UFRRJ/ALC/ALC-Documentação/codigos will
Qual a ordem da matriz?
2
Qual a matriz A?
1 1
4 0
Qual o vetor b?
1
2
Matriz L:
1.000000 0.000000
4.000000 1.000000
Matriz U:
1.000000 1.000000
0.000000 -4.000000

o resultado de Ax=b é:
0.500000
0.500000

Process returned 10 (0xA)   execution time : 42.985 s
Press ENTER to continue.
```

Figura 14: Exemplo.2 que mostra a execução do programa (quest08) no terminal

- MATLAB-2



The image shows a MATLAB Command Window. At the top, there is a blue header bar with the text "Command Window". Below it, a yellow banner contains the text "New to MATLAB? Watch this [Video](#), see [Examples](#), or read [Getting Started](#)." The main area of the window displays the following MATLAB commands and their outputs:

```
>> A = [1,1;4,0]

A =

     1     1
     4     0

>> [L,U] = lu(A)

L =

    0.2500    1.0000
    1.0000         0

U =

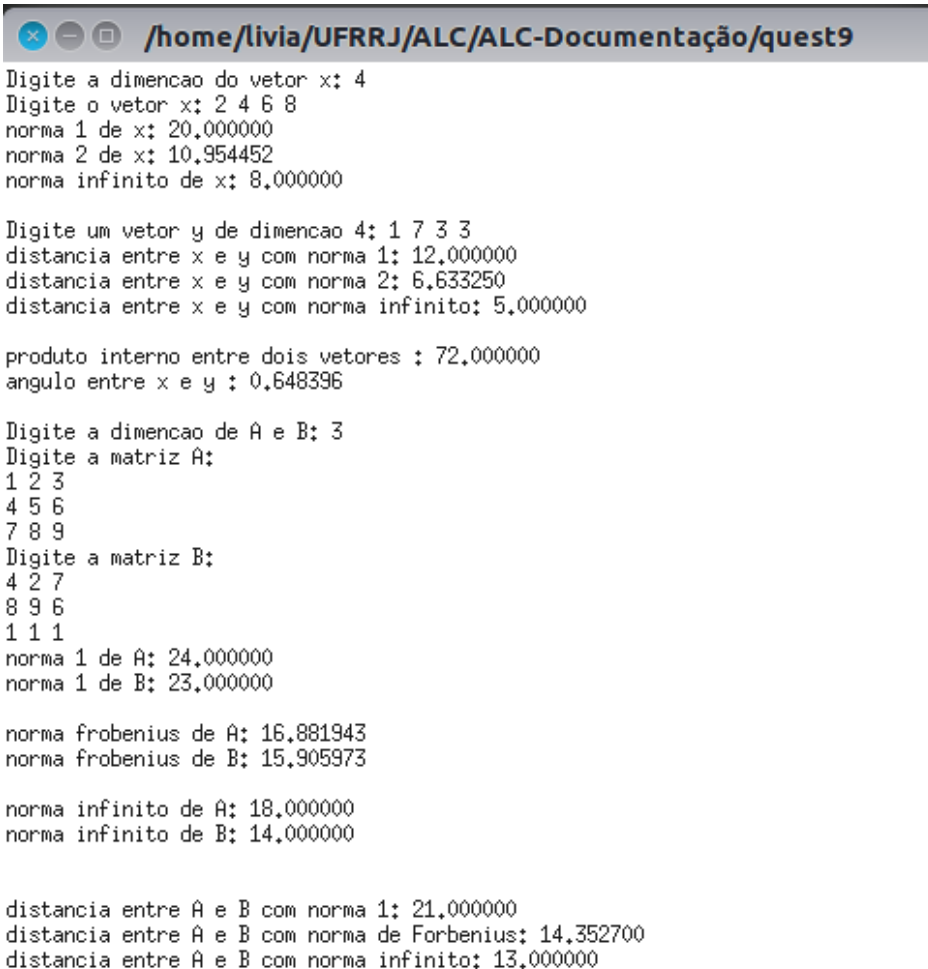
     4     0
     0     1

fx >>
```

Figura 15: Exemplo que mostra o resultado do MATLAB (quest08) para o exemplo.2

Exemplos Questão 9

- TERMINAL



```
/home/livia/UFRRJ/ALC/ALC-Documentação/quest9
Digite a dimensao do vetor x: 4
Digite o vetor x: 2 4 6 8
norma 1 de x: 20,000000
norma 2 de x: 10,954452
norma infinito de x: 8,000000

Digite um vetor y de dimensao 4: 1 7 3 3
distancia entre x e y com norma 1: 12,000000
distancia entre x e y com norma 2: 6,633250
distancia entre x e y com norma infinito: 5,000000

produto interno entre dois vetores : 72,000000
angulo entre x e y : 0,648396

Digite a dimensao de A e B: 3
Digite a matriz A:
1 2 3
4 5 6
7 8 9
Digite a matriz B:
4 2 7
8 9 6
1 1 1
norma 1 de A: 24,000000
norma 1 de B: 23,000000

norma frobenius de A: 16,881943
norma frobenius de B: 15,905973

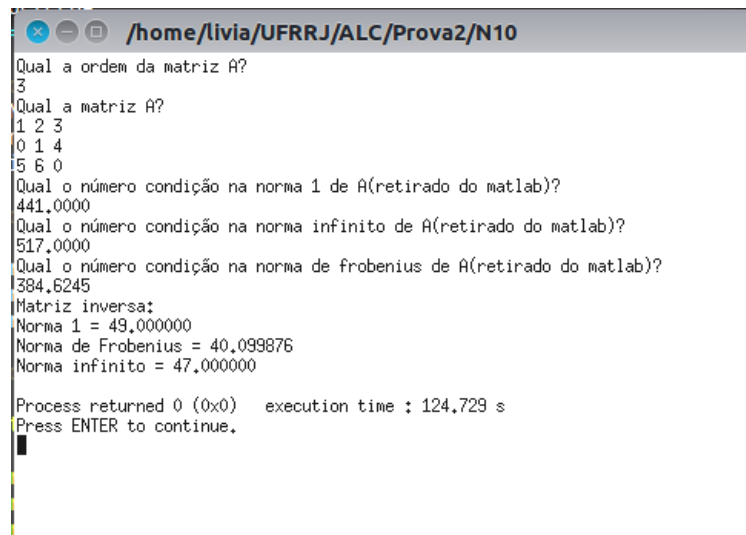
norma infinito de A: 18,000000
norma infinito de B: 14,000000

distancia entre A e B com norma 1: 21,000000
distancia entre A e B com norma de Forbenius: 14,352700
distancia entre A e B com norma infinito: 13,000000
```

Figura 16: Exemplo,1 que mostra a execução do programa (quest09) no terminal

Exemplos Questão 10

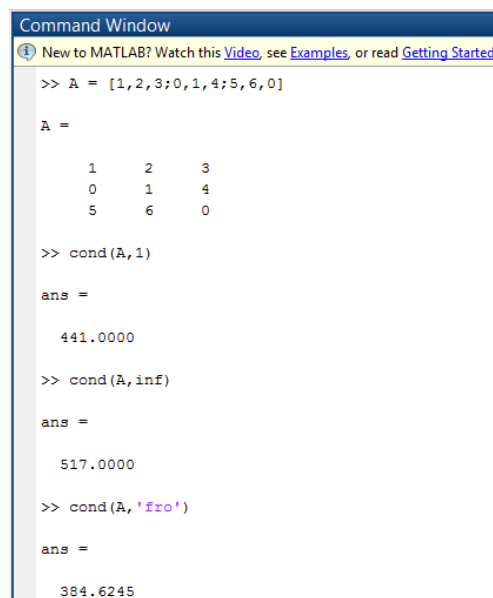
- TERMINAL



```
/home/livia/UFRRJ/ALC/Prova2/N10
Qual a ordem da matriz A?
3
Qual a matriz A?
1 2 3
0 1 4
5 6 0
Qual o número condição na norma 1 de A(retirado do matlab)?
441.0000
Qual o número condição na norma infinito de A(retirado do matlab)?
517.0000
Qual o número condição na norma de frobenius de A(retirado do matlab)?
384.6245
Matriz inversa:
Norma 1 = 49.000000
Norma de Frobenius = 40.099876
Norma infinito = 47.000000
Process returned 0 (0x0)   execution time : 124.729 s
Press ENTER to continue.
```

Figura 17: Exemplo.1 que mostra a execução do programa (quest10) no terminal

- MATLAB



```
Command Window
New to MATLAB? Watch this Video, see Examples, or read Getting Started.

>> A = [1, 2, 3; 0, 1, 4; 5, 6, 0]

A =

     1     2     3
     0     1     4
     5     6     0

>> cond(A, 1)

ans =

    441.0000

>> cond(A, inf)

ans =

    517.0000

>> cond(A, 'fro')

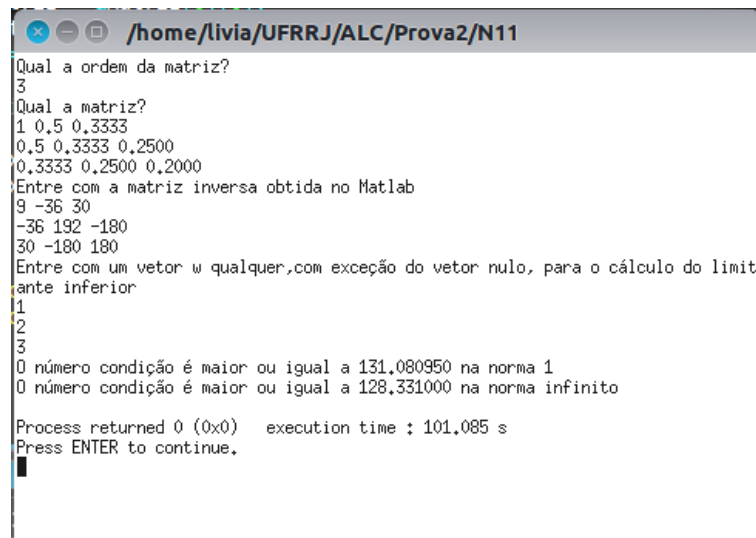
ans =

    384.6245
```

Figura 18: Exemplo que mostra o input do exemplo.1 (quest10) vindo do MATLAB

Exemplos Questão 11

- TERMINAL

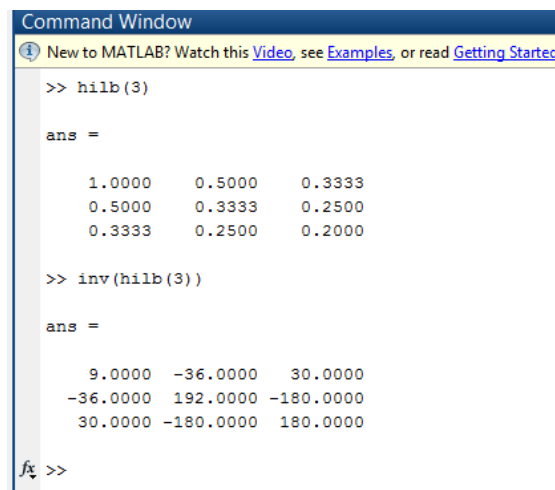


```
/home/livia/UFRRJ/ALC/Prova2/N11
Qual a ordem da matriz?
3
Qual a matriz?
1 0.5 0.3333
0.5 0.3333 0.2500
0.3333 0.2500 0.2000
Entre com a matriz inversa obtida no Matlab
9 -36 30
-36 192 -180
30 -180 180
Entre com um vetor w qualquer, com exceção do vetor nulo, para o cálculo do limit
ante inferior
1
2
3
0 número condição é maior ou igual a 131,080950 na norma 1
0 número condição é maior ou igual a 128,331000 na norma infinito

Process returned 0 (0x0)   execution time : 101,085 s
Press ENTER to continue.
```

Figura 19: Exemplo.1 que mostra a execução do programa (quest11) no terminal

- MATLAB



```
Command Window
New to MATLAB? Watch this Video, see Examples, or read Getting Started.

>> hilb(3)

ans =

    1.0000    0.5000    0.3333
    0.5000    0.3333    0.2500
    0.3333    0.2500    0.2000

>> inv(hilb(3))

ans =

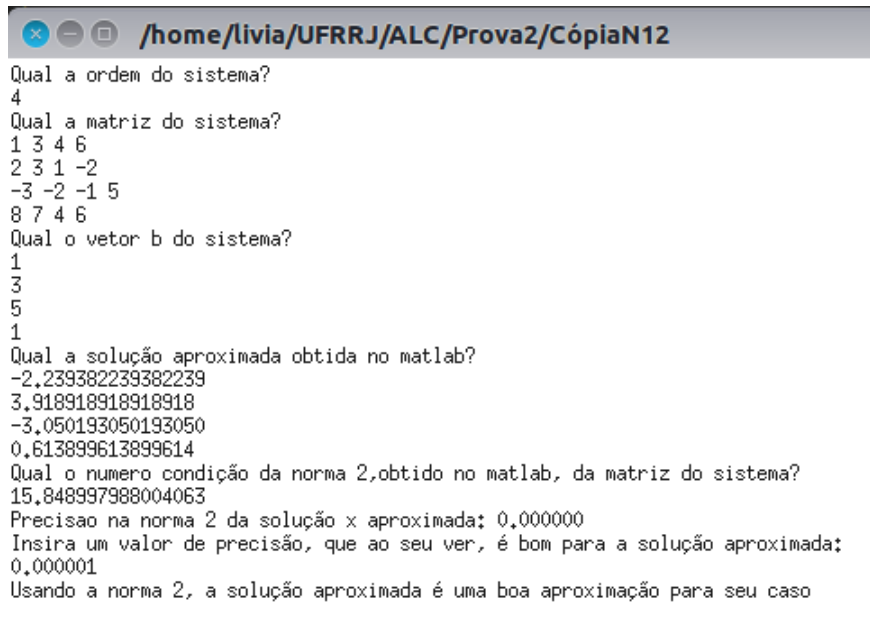
    9.0000   -36.0000    30.0000
   -36.0000   192.0000   -180.0000
    30.0000   -180.0000    180.0000

fx >>
```

Figura 20: Exemplo que mostra o input do exemplo.1 (quest11) vindo do MATLAB

Exemplos Questão 12

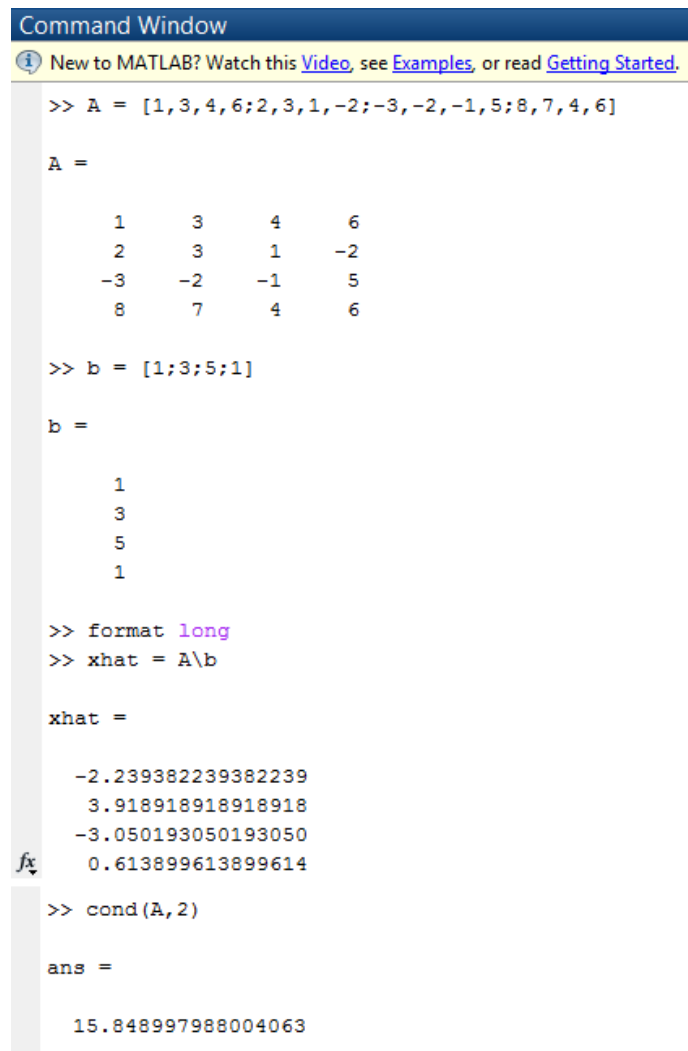
- TERMINAL



```
/home/livia/UFRRJ/ALC/Prova2/CópiaN12
Qual a ordem do sistema?
4
Qual a matriz do sistema?
1 3 4 6
2 3 1 -2
-3 -2 -1 5
8 7 4 6
Qual o vetor b do sistema?
1
3
5
1
Qual a solução aproximada obtida no matlab?
-2.239382239382239
3.918918918918918
-3.050193050193050
0.613899613899614
Qual o numero condição da norma 2, obtido no matlab, da matriz do sistema?
15.848997988004063
Precisao na norma 2 da solução x aproximada: 0.000000
Insira um valor de precisão, que ao seu ver, é bom para a solução aproximada:
0.000001
Usando a norma 2, a solução aproximada é uma boa aproximação para seu caso
```

Figura 21: Exemplo.1 que mostra a execução do programa (quest12) no terminal

- MATLAB



```
Command Window
New to MATLAB? Watch this Video, see Examples, or read Getting Started.

>> A = [1,3,4,6;2,3,1,-2;-3,-2,-1,5;8,7,4,6]

A =

     1     3     4     6
     2     3     1    -2
    -3    -2    -1     5
     8     7     4     6

>> b = [1;3;5;1]

b =

     1
     3
     5
     1

>> format long
>> xhat = A\b

xhat =

   -2.239382239382239
    3.918918918918918
   -3.050193050193050
    0.613899613899614

>> cond(A,2)

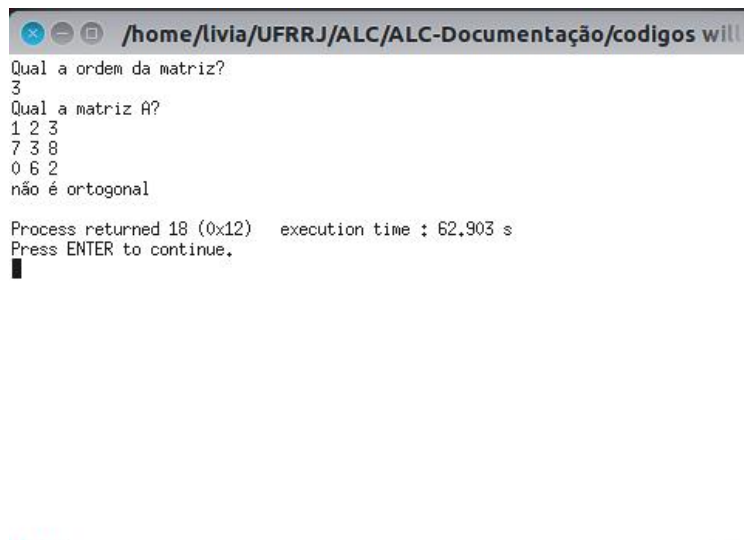
ans =

   15.848997988004063
```

Figura 22: Exemplo que mostra o input do exemplo.1 (quest12) vindo do MATLAB

Exemplos Questão 13

- TERMINAL-1

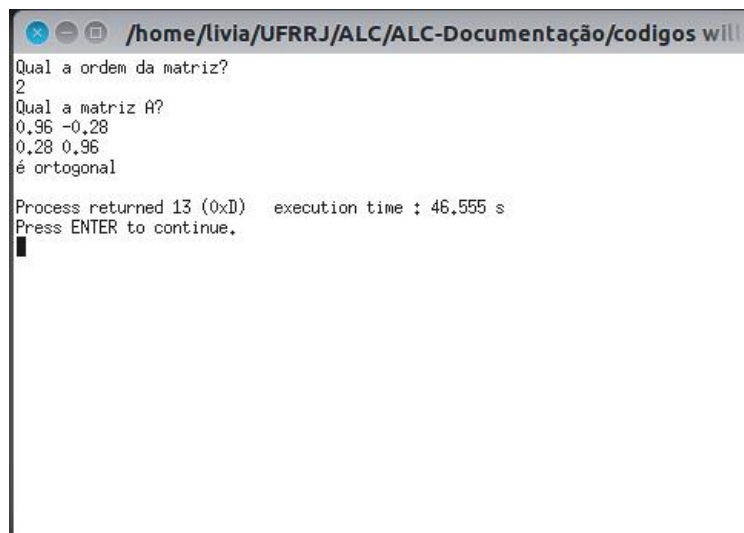


```
/home/livia/UFRRJ/ALC/ALC-Documentação/codigos will
Qual a ordem da matriz?
3
Qual a matriz A?
1 2 3
7 3 8
0 6 2
não é ortogonal

Process returned 18 (0x12)   execution time : 62,903 s
Press ENTER to continue.
```

Figura 23: Exemplo.1 que mostra a execução do programa (quest13) no terminal

- TERMINAL-2



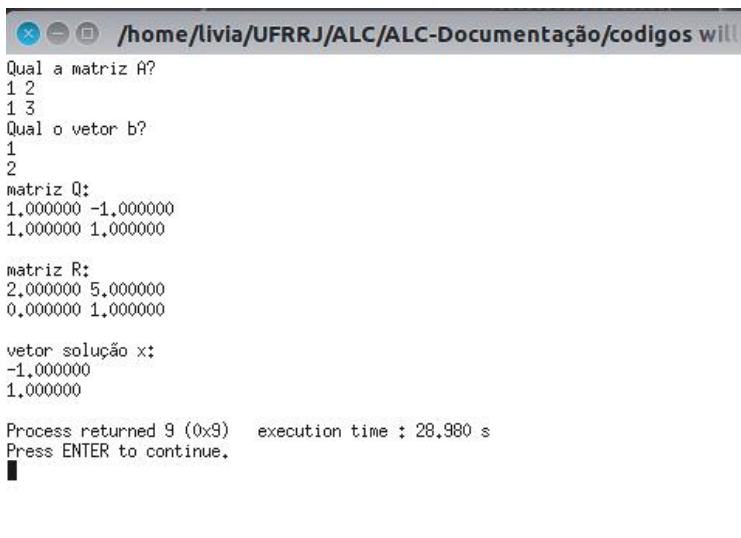
```
/home/livia/UFRRJ/ALC/ALC-Documentação/codigos will
Qual a ordem da matriz?
2
Qual a matriz A?
0,96 -0,28
0,28 0,96
é ortogonal

Process returned 13 (0xD)   execution time : 46,555 s
Press ENTER to continue.
```

Figura 24: Exemplo.2 que mostra a execução do programa (quest13) no terminal

Exemplos Questão 14

- TERMINAL



```
/home/livia/UFRRJ/ALC/ALC-Documentação/codigos will
Qual a matriz A?
1 2
1 3
Qual o vetor b?
1
2
matriz Q:
1.000000 -1.000000
1.000000 1.000000

matriz R:
2.000000 5.000000
0.000000 1.000000

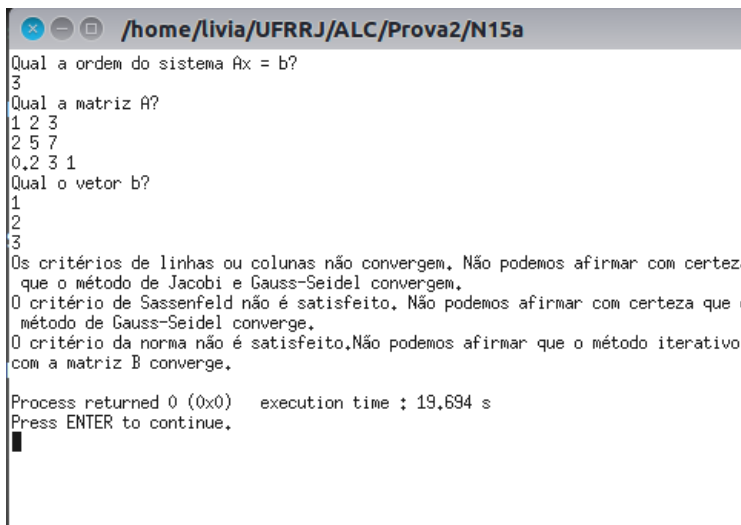
vetor solução x:
-1.000000
1.000000

Process returned 9 (0x9)   execution time : 28.980 s
Press ENTER to continue.
```

Figura 25: Exemplo.1 que mostra a execução do programa (quest14) no terminal

Exemplos Questão 15

- TERMINAL-1

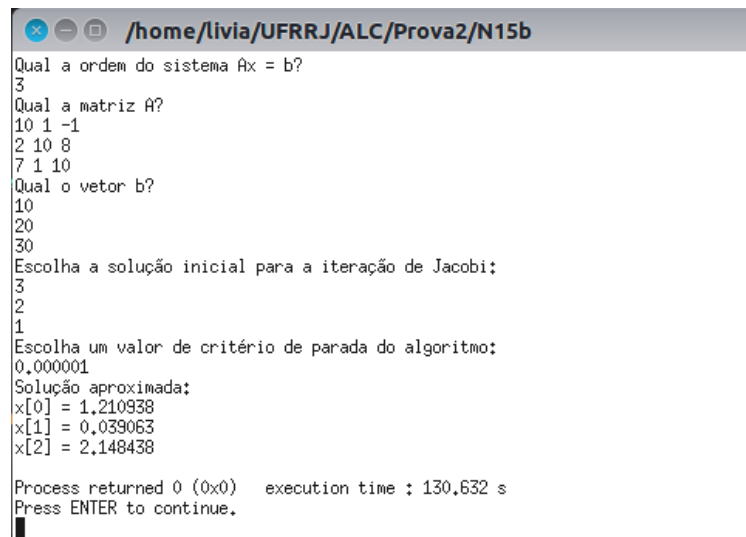


```
/home/livia/UFRRJ/ALC/Prova2/N15a
Qual a ordem do sistema Ax = b?
3
Qual a matriz A?
1 2 3
2 5 7
0.2 3 1
Qual o vetor b?
1
2
3
Os critérios de linhas ou colunas não convergem. Não podemos afirmar com certeza
que o método de Jacobi e Gauss-Seidel convergem.
O critério de Sassenfeld não é satisfeito. Não podemos afirmar com certeza que o
método de Gauss-Seidel converge.
O critério da norma não é satisfeito. Não podemos afirmar que o método iterativo
com a matriz B converge.

Process returned 0 (0x0)   execution time : 19.694 s
Press ENTER to continue.
```

Figura 26: Exemplo.1 letra-a que mostra a execução do programa (quest15) no terminal

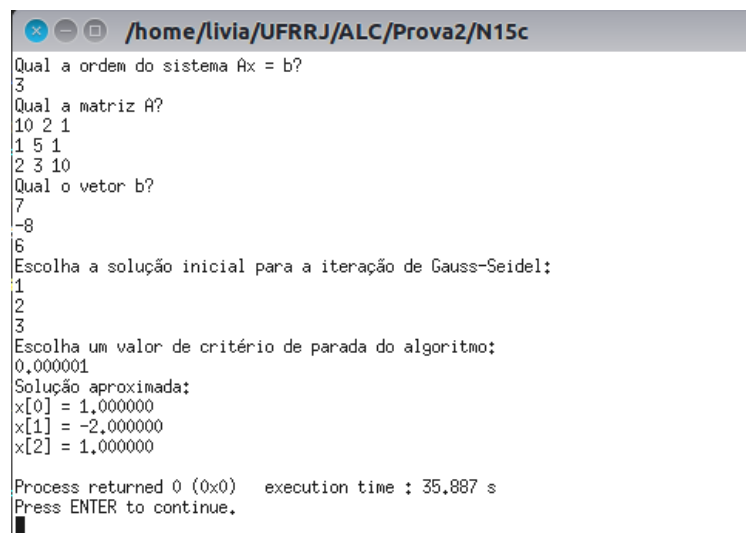
- TERMINAL-2



```
/home/livia/UFRRJ/ALC/Prova2/N15b
Qual a ordem do sistema Ax = b?
3
Qual a matriz A?
10 1 -1
2 10 8
7 1 10
Qual o vetor b?
10
20
30
Escolha a solução inicial para a iteração de Jacobi:
3
2
1
Escolha um valor de critério de parada do algoritmo:
0.000001
Solução aproximada:
x[0] = 1.210938
x[1] = 0.039063
x[2] = 2.148438
Process returned 0 (0x0)   execution time : 130.632 s
Press ENTER to continue.
```

Figura 27: Exemplo.1 letra-b que mostra a execução do programa (quest15) no terminal

- TERMINAL-3

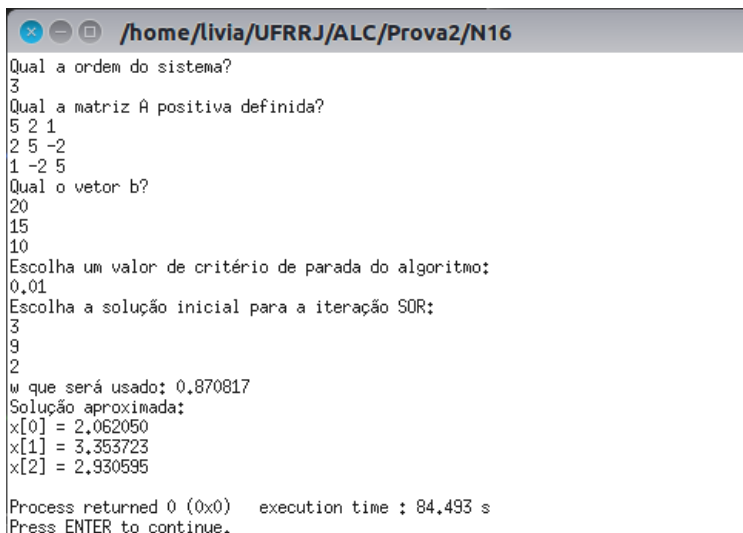


```
/home/livia/UFRRJ/ALC/Prova2/N15c
Qual a ordem do sistema Ax = b?
3
Qual a matriz A?
10 2 1
1 5 1
2 3 10
Qual o vetor b?
7
-8
6
Escolha a solução inicial para a iteração de Gauss-Seidel:
1
2
3
Escolha um valor de critério de parada do algoritmo:
0.000001
Solução aproximada:
x[0] = 1.000000
x[1] = -2.000000
x[2] = 1.000000
Process returned 0 (0x0)   execution time : 35.887 s
Press ENTER to continue.
```

Figura 28: Exemplo.1 letra-c que mostra a execução do programa (quest15) no terminal

Exemplos Questão 16

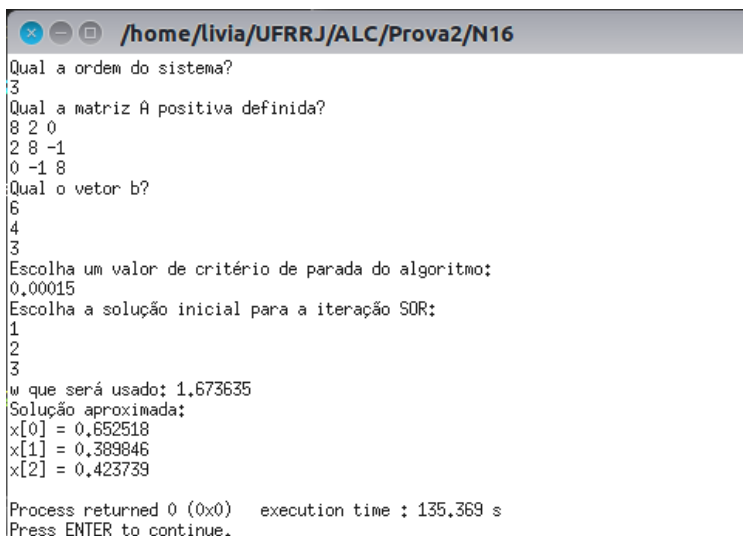
- TERMINAL



```
/home/livia/UFRRJ/ALC/Prova2/N16
Qual a ordem do sistema?
3
Qual a matriz A positiva definida?
5 2 1
2 5 -2
1 -2 5
Qual o vetor b?
20
15
10
Escolha um valor de critério de parada do algoritmo:
0.01
Escolha a solução inicial para a iteração SOR:
3
9
2
w que será usado: 0.870817
Solução aproximada:
x[0] = 2.062050
x[1] = 3.353723
x[2] = 2.930595
Process returned 0 (0x0)   execution time : 84.493 s
Press ENTER to continue.
```

Figura 29: Exemplo.1 que mostra a execução do programa (quest16) no terminal

- TERMINAL-2

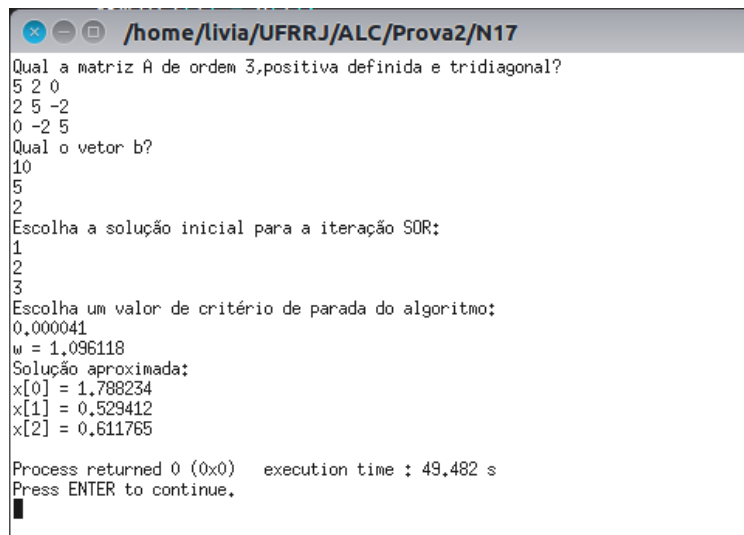


```
/home/livia/UFRRJ/ALC/Prova2/N16
Qual a ordem do sistema?
3
Qual a matriz A positiva definida?
8 2 0
2 8 -1
0 -1 8
Qual o vetor b?
6
4
3
Escolha um valor de critério de parada do algoritmo:
0.00015
Escolha a solução inicial para a iteração SOR:
1
2
3
w que será usado: 1.673635
Solução aproximada:
x[0] = 0.652518
x[1] = 0.389846
x[2] = 0.423739
Process returned 0 (0x0)   execution time : 135.369 s
Press ENTER to continue.
```

Figura 30: Exemplo.2 que mostra a execução do programa (quest16) no terminal

Exemplos Questão 17

- TERMINAL

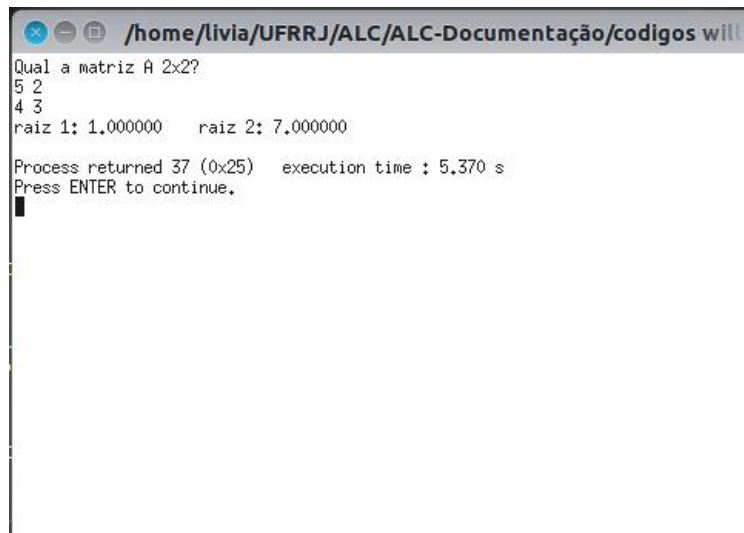


```
/home/livia/UFRJ/ALC/Prova2/N17
Qual a matriz A de ordem 3,positiva definida e tridiagonal?
5 2 0
2 5 -2
0 -2 5
Qual o vetor b?
10
5
2
Escolha a solução inicial para a iteração SOR:
1
2
3
Escolha um valor de critério de parada do algoritmo:
0.000041
w = 1.096118
Solução aproximada:
x[0] = 1.788234
x[1] = 0.529412
x[2] = 0.611765
Process returned 0 (0x0)   execution time : 49.482 s
Press ENTER to continue.
```

Figura 31: Exemplo.1 que mostra a execução do programa (quest17) no terminal

Exemplos Questão 18

- TERMINAL

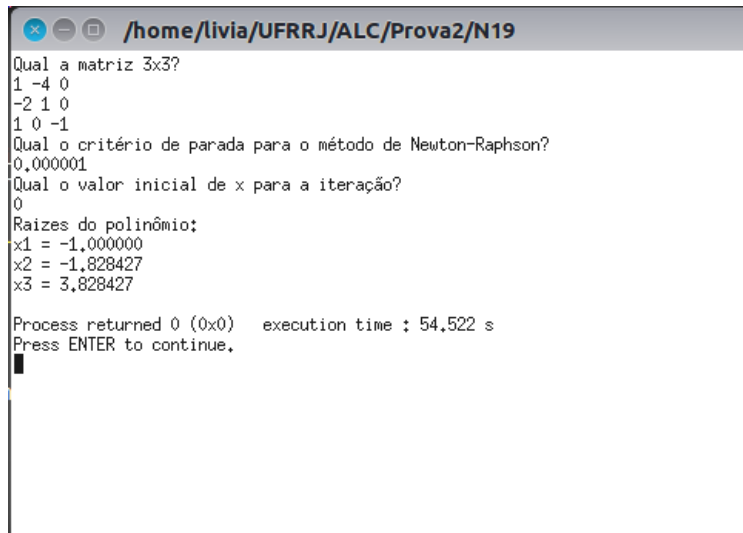


```
/home/livia/UFRJ/ALC/ALC-Documentação/codigos willu
Qual a matriz A 2x2?
5 2
4 3
raiz 1: 1.000000   raiz 2: 7.000000
Process returned 37 (0x25)   execution time : 5.370 s
Press ENTER to continue.
```

Figura 32: Exemplo.1 que mostra a execução do programa (quest18) no terminal

Exemplos Questão 19

- TERMINAL

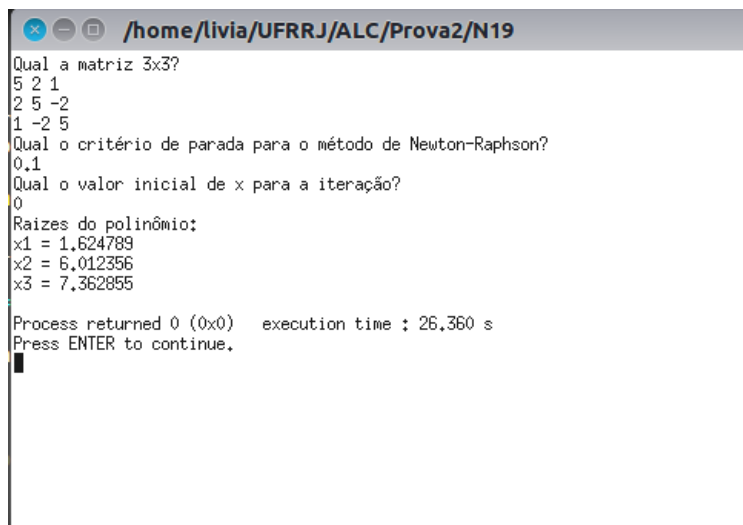


```
/home/livia/UFRRJ/ALC/Prova2/N19
Qual a matriz 3x3?
1 -4 0
-2 1 0
1 0 -1
Qual o critério de parada para o método de Newton-Raphson?
0.000001
Qual o valor inicial de x para a iteração?
0
Raizes do polinômio:
x1 = -1.000000
x2 = -1.828427
x3 = 3.828427

Process returned 0 (0x0)   execution time : 54.522 s
Press ENTER to continue.
```

Figura 33: Exemplo.1 que mostra a execução do programa (quest19) no terminal

- TERMINAL-2



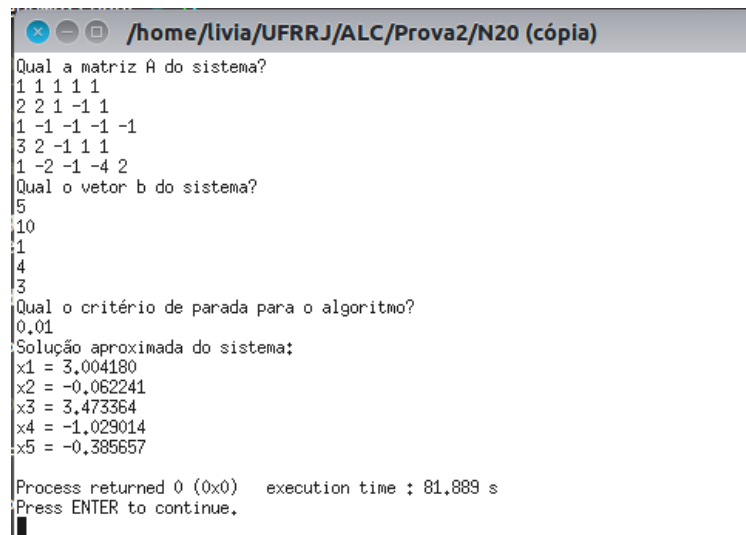
```
/home/livia/UFRRJ/ALC/Prova2/N19
Qual a matriz 3x3?
5 2 1
2 5 -2
1 -2 5
Qual o critério de parada para o método de Newton-Raphson?
0.1
Qual o valor inicial de x para a iteração?
0
Raizes do polinômio:
x1 = 1.624789
x2 = 6.012356
x3 = 7.362855

Process returned 0 (0x0)   execution time : 26.360 s
Press ENTER to continue.
```

Figura 34: Exemplo21 que mostra a execução do programa (quest19) no terminal

Exemplos Questão 20

- TERMINAL



```
/home/livia/UFRRJ/ALC/Prova2/N20 (cópia)
Qual a matriz A do sistema?
1 1 1 1 1
2 2 1 -1 1
1 -1 -1 -1 -1
3 2 -1 1 1
1 -2 -1 -4 2
Qual o vetor b do sistema?
5
10
1
4
3
Qual o critério de parada para o algoritmo?
0.01
Solução aproximada do sistema:
x1 = 3.004180
x2 = -0.062241
x3 = 3.473364
x4 = -1.029014
x5 = -0.385657

Process returned 0 (0x0)   execution time : 81.889 s
Press ENTER to continue.
```

Figura 35: Exemplo.1 que mostra a execução do programa (quest20) no terminal