ALC Prova 2: Trabalho de Implementação

Professor Érito - Algebra Linear Computacional

William Anderson & Lívia de Azevedo

08 de dezembro de 2014

Sumário

Lista de Figuras

Exercício 1

Para a primeira questão a solução atribuida 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_{mxn} pelo vetor b_n repetidos n *vezes:*

- *código de implementação :*

```c
#include <stdio.h>

int main(void)
{
    // variaveis de controle
    int m, n, num_flops = 0;

    // obtendo dimencao da maitriz
    printf("Entre com o numero de linhas\n");
    scanf("%d", &m);
    printf("entre com o numero de colunas\n");
    scanf("%d", &n);

    // criando os fatores a serem multiplicados
    float A[m][n], b[n], Result[m];

    printf("Entre com a matriz A\n");
    int i, j;
    for(i = 0; i < m ; i++)
    {
        for(j = 0; j < n ; j++)
        {
            scanf("%f", &A[i][j]);
        }
    }

    printf("Entre com o vetor b\n");
    for(i = 0; i < n ; i++)
    {
        scanf("%f", &b[i]);
    }


    // aplicando a multiplicacao de Result = Ab
    for(i = 0; i < m ; i++)
    {
        float somatorio = 0;
        for(j = 0; j < n ; j++)
        {
            somatorio += A[i][j]*b[j];// guardando o valor temporario
            num_flops += 2;// contando o numero de flops
        }
        Result[i] = somatorio;// ap s o calculo atribui o valor
    }
```

```

    // imprimindo o resultado
    for(i = 0; i < m ; i++)
    {
        printf("resultado de Ab:\n");
        printf("%f\n", Result[i]);
    }

    // imprimindo numero de flops
    printf("numero de flops:\n");
    printf("%d\n", num_flops);
}
```

Listing 1: Multiplicação A_{mxn} por b_n

Exercício 2

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

- código de implementação :

```
#include <stdio.h>

int main(void)
{
    // variaveis de controle
    int m, n, k, num_flops = 0;

    // obtendo dimencoes das matrizes
    printf("Entre com o valor de m\n");
    scanf("%d", &m);
    printf("Entre com o valor de n\n");
    scanf("%d", &n);
    printf("Entre com o valor de k\n");
    scanf("%d", &k);

    // matrizes operadoras
    float A[m][n], B[n][k], Result[m][k];

    // lendo A
    printf("Entre com a matriz A\n");
    int i, j;
    for(i = 0; i < m ; i++)
    {
        for(j = 0; j < n ; j++)
        {
            scanf("%f", &A[i][j]);
        }
    }
```

```

    printf("Entre com a matriz B\n");
    for(i = 0; i < n ; i++)
    {
        for(j = 0; j < k ; j++)
        {
            scanf("%f", &B[i][j]);
        }
    }

    // multiplicando Result = AB
    for(i = 0; i < m ; i++)
    {
        int l;
        for(l = 0; l < k ; l++)
        {
            float somatorio = 0;

            for(j = 0; j < n ; j++)
            {
                somatorio += A[i][j]*B[j][l];// guardando o valor temporario
                num_flops += 2;// contando o numero de flops
            }
            Result[i][l] = somatorio;// ap s o calculo atribui o valor
        }
    }

    // imprimindo o resultado
    for(i = 0; i < m ; i++)
    {
        for(j = 0; j < k ; j++)
        {
            printf("resultado de AB:\n");
            printf("%f\n", Result[i][j]);
        }
    }

    // imprimindo o numero de flops
    printf("numero de flops:\n");
    printf("%d\n", num_flops);
}
```

Listing 2: Multiplicao A_{mxn} por B_{nxk}

Exercício 3

Para a terceira questão a solução atribuida 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 funçã é implementada a fórmula do determinante de Laplace: $\sum_{i=1}^{n}(-1)^{i+j} \cdot a_{ij} \cdot Det2x2(subMatrizA_{ij})$.

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 variavel l controla em qual linha será depoistado 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 :

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

//funcao que calcula um determinante de uma matriz 2x2
float Det2x2(float A[2][2])
{
    return A[0][0]*A[1][1] - A[0][1]*A[1][0];
}

//funcao que calcula um determinante de uma matriz 3x3 por la place usando a 2x2
float Det3x3 (float B[3][3])
{
    float DetB = 0;
    int i;
    for(i=0;i<3;i++)
    {
        float subB[2][2];
        int l,c;
        for(l=0;l<2;l++)//forma a matriz 2x2 que resta dos isolamentos
        {
            for(c=0;c<3;c++)
            {
                if(c<i)
                    subB[l][c] = B[l+1][c];// deduzidas a partir de observacoes
                if(c>i)
                    subB[l][c-1] = B[l+1][c];// deduzidas a partir de observacoes
                if(c==i)
                    continue;
            }
        }
        //formula de do determinante de la place
        DetB += ( pow( (-1),((i+1)+1) ) * Det2x2(subB) * B[i][0]);
    }

    return DetB;
```

```
}

int main(void)
{
    //variaveis de controle
    int i, j;
    printf("Entre com uma matriz 2x2\n");
    float A[2][2];
    for(i=0;i<2;i++)
        for(j=0;j<2;j++)
            scanf("%f", &A[i][j]);*

    //imprimindo o primeiro resultado
    printf("Resultado do determinante 2x2 : %f\n", Det2x2(A));
    printf("Entre com uma matriz 3x3\n");
    float B[3][3];

    for(i=0;i<3;i++)
        for(j=0;j<3;j++)
            scanf("%f", &B[i][j]);

    //imprimindo o segundo resultado
    printf("Resultado do determinante 3x3 : %f\n", Det3x3(B));
}
```

Listing 3: Determinante de A_{2x2} e B_{3x3}

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 :

```
#include <stdio.h>

int main(void)
{
    //variaveis de controle
    int k, num_flops = 0;

    //lendo a dimencao do sistema
    printf("valor de k\n");
    scanf("%d", &k);

    //criando operadores
```

```c
    float A[k][k], b[k], x[k];

    //leitura da natriz de coeficientes
    printf("ler matriz A\n");
    int i, j;
    for(i = 0; i < k ; i++)
    {
        for(j = 0; j < k ; j++)
        {
            scanf("%f", &A[i][j]);
        }
    }

    //leitura do vetor b
    printf("ler vetor b\n");
    for(i = 0; i < k ; i++)
    {
        scanf("%f", &b[i]);
    }

    //seguindo da ultima linha ate a primeira
    for(i = k-1 ; i >= 0 ; i--)
    {
        //seguindo da ultima coluna ate a coluna de mesmo numero da linha
        for(j = k-1 ; j >= i ; j--)
        {
            if(j>i)
            {
                //utilizando o proprio b para guardar os resultados intermediarios
                b[i] -= A[i][j]*x[j];
                num_flops += 2;//contagem do numero de flops (subtracao e multiplicacao)
            }
            else
            {
                //enverrando a operacao da linha com a divisao
                b[i] /= A[i][j];
                num_flops += 1;//contagem do numero de flops (divisao)
            }
        }
        x[i] = b[i];//guardando o resultado em x
    }

    //imprimindo a solucao
    printf("valores de x\n");
    for(i = 0; i < k ; i++)
        printf("%f\n", x[i]);

    //imprimindo o numero de flops
    printf("numero de flops:\n");
    printf("%d\n", num_flops);
}
```

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 :

```
#include <stdio.h>

int main(void)
{
    //variaveis de controle
    int k, num_flops = 0;

    //lendo a dimencao do sistema
    printf("valor de k\n");
    scanf("%d", &k);

    //criando os operadores
    float A[k][k], b[k], x[k];

    //lendo a matriz de coeficientes
    printf("ler matriz A\n");
    int i, j;
    for(i = 0; i < k ; i++)
    {

        for(j = 0; j < k ; j++)
        {
            scanf("%f", &A[i][j]);
        }
    }
    //kendo o vetor b
    printf("ler vetor b\n");
    for(i = 0; i < k ; i++)
    {
        scanf("%f", &b[i]);
    }

    //indo da primeira linha da matriz ate a ultima
    for(i = 0 ; i < k ; i++)
```

```
    {
        //indo da primeira coluna ate o termo da diagonal
        for(j = 0 ; j <= i  ; j++)
        {
            if(j<i)
            {
                //usando o proprio b para guardar os valores intermediarios
                b[i] -= A[i][j]*x[j];
                num_flops += 2;//contagem do numero de flops (subtracao e multiplicacao)
            }
            else
            {
                b[i] /= A[i][j];//dividindo pelo termo da diagonal
                num_flops += 1;//contagem do numero de flops (divisao)
            }
        }
        //encerrando a operacao da linha guardando o valor em x
        x[i] = b[i];
    }

    //imprimindo a solucao do sistema
    printf("valores de x\n");

    for(i = 0; i < k ; i++)
        printf("%4f\n", x[i]);

    //imprimindo o numero de flops
    printf("numero de flops:\n");
    printf("%d\n", num_flops);
}
```

Listing 5: resolução de um sistema triangular superior

Exercício 6

Neste exercício, calcularemos o fator de Cholesky de uma matriz C_{mxn}. O programa pedirá ao usuário a inserção de uma matriz A_{mxn} 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:

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

Fórmula 3:

$$r_{ii} = +\sqrt{a_{ii} - \sum_{k=1}^{i1} r_{ki}^2}$$

Fórmula 4:

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

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 :

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

int main()
{
```

```
    int m,n,k,i,j;
    double somatorio = 0;

    printf("Qual a dimensao m da matriz?\n");
    scanf("%d",&m);
    printf("Qual a dimensao n da matriz?\n");
    scanf("%d",&n);

    printf("Qual a matriz?(de prefer ncia uma matriz invers vel)\n");

    double Matriz[m][n];

    for(i = 0;i < m;i++)
    {
        for(j = 0;j < n;j++)
        {
            scanf("%lf",&Matriz[i][j]);
        }
    }

    double MatrizTranposta[n][m];

    for(i = 0; i < n;i++)
    {
        for(j = 0;j < m;j++)
        {
            MatrizTranposta[i][j] = Matriz[j][i];
        }
    }

    double C[n][n];

    for(i = 0;i < n;i++)
    {
        for(j = 0;j < n;j++)
        {
            C[i][j] = 0;
        }
    }

    //Processo da multiplicacao de matrizes(A^T * A).

    for(i = 0;i < n;i++)
    {
        for(j = 0;j < n;j++)
        {
            for(k = 0;k < m;k++)
            {
                C[i][j] += MatrizTranposta[i][k] * Matriz[k][j];
            }
        }
    }

```

```c
    printf("Matriz C:\n");

    for(i = 0; i < n;i++)
    {
        for(j = 0; j < n;j++)
        {
            printf("%f  ",C[i][j]);
        }

        printf("\n");
    }

    double fatorCholesky[n][n];

    //Zerar os valores de memoria da variavel.("Inicializar" a variavel)
    for(i = 0; i < n;i++)
    {
        for(j = 0; j < n;j++)
        {
            fatorCholesky[i][j] = 0;
        }
    }

    //Passo 1: Determinar os primeiros elementos de cholesky (Formula 1):
    //Obs: funcao pow(x,n) = x elevado a n.

    fatorCholesky[0][0] = sqrt(C[0][0]);

    for(j = 1;j < n;j++)
    {
        fatorCholesky[0][j] = C[0][j] / fatorCholesky[0][0];
    }

    //Passo 2: Calcular os elementos r[i][i] de Cholesky (Formula 2):

    for(i = 1; i < n;i++)
    {
        for(k = 0;k < i;k++)
        {
            somatorio += pow(fatorCholesky[k][i],2);
        }

        fatorCholesky[i][i] = C[i][i] - somatorio;
        fatorCholesky[i][i] = sqrt(fatorCholesky[i][i]);
        somatorio = 0; //zerar para iniciar o somat rio da proxima parcela.

    //Com todos os elementos r[i][i] determinados,podemos calcular os r[i][j] (
        Formula 3):
        for(j = i + 1;j < n;j++)
        {
            for(k = 0; k < i;k++)
            {
                somatorio += fatorCholesky[k][i] * fatorCholesky[k][j];
```

```
            }

            fatorCholesky[i][j] = C[i][j] - somatorio;
            fatorCholesky[i][j] = fatorCholesky[i][j] / fatorCholesky[i][i];
            somatorio = 0; //zerar para iniciar o somatorio da proxima parcela.
        }

    }

    //Mostrar o fator de Cholesky da matriz C
    printf("Fator de Cholesky:\n");

    for(i = 0; i < n;i++)
    {
        for(j = 0; j < n;j++)
        {
            printf("%f  ",fatorCholesky[i][j]);
        }

        printf("\n");
    }

    //Passo para determinar um sistema Ax = b com a fatoracao de Cholesky

    printf("Digite a matriz B para resolver um sistema Ax = B com Cholesky:\n");
    double B[n];

    for(i = 0;i < n;i++)
    {
        scanf("%lf",&B[i]);
    }

    double x[n];
    double y[n];
    double transpostaCholesky[n][n];

    //Determinar a transposta de Cholesky(R^T)
    for(i = 0; i < n;i++)
    {
        for(j = 0;j < n;j++)
        {
            transpostaCholesky[i][j] = fatorCholesky[j][i];
        }
    }

    //Resolvendo Ax = B -------> R^T * Rx = B:

    //Primeira parte: R^T * y = B;
    //Mesma logica usada na questao 5 da prova.

     y[0] = B[0] / transpostaCholesky[0][0];

```

```c
    for(i = 0;i <= n - 1;i++)
    {
        for(j = 0; j <= i;j++)
        {
            if(j < i)
            {
                B[i] -= transpostaCholesky[i][j] * y[j];
            }
            else
            {
                B[i] = B[i] / transpostaCholesky[i][j];

            }
        }

        y[i] = B[i];
    }

    //Segunda parte: Resolver Rx = y;
    //Mesma logica usada na questao 4 da prova.

    x[n - 1] = y[n - 1] / fatorCholesky[n - 1][n - 1];

    for(i = n - 1;i >= 0;i--)
    {
        for(j = n - 1; j >= i;j--)
        {
            if(j > i)
            {
                y[i] -= fatorCholesky[i][j] * x[j];
            }
            else
            {
                y[i] = y[i] / fatorCholesky[i][j];

            }
        }

        x[i] = y[i];
    }

    //Resultados
    printf("Solu  o do sistema:\n");

    for(i = 0;i < n;i++)
    {
        printf("x_%d = %f\n",i,x[i]);
    }

    return 0;
}
```

Listing 6: Decomposição de Cholesky

Exercício 7

Nesta questão a solução atribuida 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 :

```c
#include <stdio.h>

int main()
{
    int n;
    printf("Qual a ordem da matriz?\n");
    scanf("%d", &n);

    float A[n][n];
    int i, j;

    printf("Qual a matriz?\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%f", &A[i][j]);
        }
    }

    int EhBanda = 1;
    int c;
    int EhPivo0 = 0;
    int PrimeiraDiagonal0;

    for(c=1;c<n;c++)
    {
        if(A[c][0] == 0.0 && A[0][c] == 0.0)
        {
            EhPivo0 = 1;
            if(A[PrimeiraDiagonal0][0] != 0)
                PrimeiraDiagonal0 = c;
        }
```

```
            else
            {
                EhPivo0 = 0;
                PrimeiraDiagonal0 = -1;
            }

            if(EhPivo0)
            {
                if(c == n-1)
                {
                    i = c;
                    j = 0;
                }
                else
                {
                    i = c+1;
                    j = 1;
                }
                for(;i<n;i++)
                {
                    if(A[i][j] != 0.0 || A[j][i] != 0.0)
                    {
                        EhPivo0 = 0;
                        PrimeiraDiagonal0 = -1;
                        break;
                    }
                    j++;
                }
                if(EhPivo0)
                {
                    continue;
                }
            }

        }

        if(EhPivo0)
            EhBanda = 1;
        else
            EhBanda = 0;

        if(EhBanda)
        {
            printf("e banda\n");
            if(PrimeiraDiagonal0==2)
                printf("tamb m e tridiagonal\n");
        }
        else
        {
            printf("nao e banda\n");
        }


```

```
}
```

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

Exercício 8

Neste exercício usaremos os passos da eliminação de Gauss com pivotiamento 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

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

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

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

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 :

```c
#include<stdio.h>

int main()
{
    int n;
    scanf("%d", &n);
    float A[n][n], L[n][n], U[n][n], b[n];

    int i, j, k;
```

```
    //lendo A e montando a estrutura basica de L e U
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%f", &A[i][j]);

            if(i==j)
                L[i][j] = 1;
            if(i<j)
                L[i][j] = 0;

            U[i][j] = A[i][j];
        }
    }
    float copiaB[n];
    printf("\n");
    //lendo o vetor B e criando uma c pia para a prova real (apenas test case)
    for(i=0;i<n;i++)
    {
        scanf("%f", &b[i]);
        copiaB[i] = b[i];
    }

    //construindo as matrizes L e U
    for(k=0;k<n-1;k++)
    {
        for(i=k+1;i<n;i++)
        {
            L[i][k] = U[i][k]/U[k][k];
            U[i][k] = 0;
            for(j=k+1;j<n;j++)
            {
                U[i][j] -= L[i][k]*U[k][j];
            }
        }
    }

    //printando L
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
            printf("%f ", L[i][j]);
        printf("\n");
    }

    printf("\n");

    //printando U
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
            printf("%f ", U[i][j]);
```

```
        printf("\n");
    }

    printf("\n");

    float x[n], y[n];
    //calculando L*y=b substituicao pra tras
    for(i= 0;i<n;i++)
    {
        for(j= 0 ;j<=i;j++)
        {
            if(j<i)
            {
                b[i] -= L[i][j]*y[j];
            }
            if(i==j)
            {
                b[i] /= L[i][j];
            }
        }
        y[i] = b[i];
    }

    //calculando Ux = y por substituicao pra tras
    for(i= n-1;i>=0;i--)
    {
        for(j= n-1 ;j>=i;j--)
        {
            if(j>i)
            {
                y[i] -= U[i][j]*y[j];
            }
            if(i==j)
            {
                y[i] /= U[i][j];
            }
        }
        x[i] = y[i];
    }

/*  //printando b
    for(i=0;i<n;i++)
        printf("%f\n", b[i]);*/

    printf("\no resultado de Ax=b   : \n");

    for(i=0;i<n;i++)
        printf("%f\n", x[i]);
    printf("\n");

    //teste de prova real que aplica x    matriz A original (apenas test case)
/*  for(i=0;i<n;i++)
    {
```

```
        printf("%f     ", copiaB[i]);
        float acumulador = 0;
        for(j=0;j<n;j++)
             acumulador += A[i][j]*x[j];
         printf("%f\n", acumulador);
    }*/



}
```

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 dimenção n usando a fórmula:

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

Função Norma2 :
Esta função calcula a norma 2 de um vetor de dimenção n usando a fórmula:

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

Função NormaInfinito :
Esta função calcula a norma infinito de um vetor de dimenção 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 "accos" (arcosseno) 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 matrz A_{rxc} 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_{rxc} utilizando o seguinte recurso:

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

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

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

função NormaInfinitoMatriz :
Esta função calcula a norma infinito de uma matriz A_{rxc} utilizando o seguinte recurso:

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

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

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 posssibilidade de escolher qual norma será aplicada (chamando as funções anteriores referente as matrizes) na fórmula:

considere "result[r][c]" uma natriz 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 :

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

//calcula a norma 1 de um vetor de tamanho n
float Norma1(float x[], int n)
{
    int i;
    float somatorio = 0;
    for(i=0;i<n;i++)
    {
        if(x[i] < 0)
            x[i] *= -1;
        somatorio += x[i];
    }

    return somatorio;
}

//calcula a norma 2 de um vetor de tamanho n
float Norma2(float x[], int n)
{
    int i;
    float somatorio = 0;

    for(i=0;i<n;i++)
    {
        somatorio += pow(x[i], 2);
    }

    return sqrt(somatorio);
}

//calcula a norma infinito de um vetor de tamanho n
float NormaInfinito(float x[], int n)
{
    float Maior = 0;
```

```
    int i;

    for(i=0;i<n;i++)
    {
        if(x[i]<0)
            x[i] *= -1;

        if(i==0)
            Maior = x[i];
        else
        {
            if(x[i] > Maior)
                Maior = x[i];
        }
    }

    return Maior;
}

//calcula a distancia entre dois vetores x e y de tamanho n permitindo escolher
    a norma
float DistanciaEntreDoisVetores(float x[], float y[], int n, char norma)
{
    float result[n];
    int i;
    for(i=0;i<n;i++)
        result[i] = x[i] - y[i];

    switch(norma)
    {
        case '1':
            return Norma1(result, n);
        case '2':
            return Norma2(result, n);
        case 'i':
            return NormaInfinito(result, n);
    }
}

//calcula o produto interno entre dois vetores x e y de tamanho n
float ProdutoInterno(float x[], float y[], int n)
{
    float somatorio =0;
    int i;
    for(i=0;i<n;i++)
    {
        somatorio += x[i]*y[i];
    }

    return somatorio;
}
//retorna o cos do angulo entre dois vetores
float CosDoAnguloEntreDoisVetores(float x[], float y[], int n)
```

```c
{
    return ProdutoInterno(x, y, n)/(Norma2(x,n)*Norma2(y,n));
}

//calcula a norma de fobenius (usando um ponteiro para referir a matriz Arxc devido a questoes da linguagem)
float NormaFrobenius(float* matrizA, int r, int c)
{
    int i, j;
    float somatorio = 0;

    float A[r][c];
    int k = 0;
    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
        {
            A[i][j] = *(matrizA + k); //equivale a fazer A[i][j] = matrizA[k]
            k++;
        }

    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
            somatorio += pow(A[i][j], 2);

    return sqrt(somatorio);
}

//calcula a norma 1 de uma matriz A (referenciada por matrizA) de tamanho rxc
float Norma1Matriz(float* matrizA, int r, int c)
{
    float result[r], somatorio;
    int i, j;

    float A[r][c];
    int k = 0;
    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
        {
            A[i][j] = *(matrizA + k);
            k++;
        }
    for(i=0;i<r;i++)
    {
        somatorio = 0; //iniciando com 0
        for(j=0;j<c;j++)
        {
            if(A[i][j]<0)
                A[i][j] *= -1; // modulo

            somatorio += A[i][j];
        }
        result[i] = somatorio; //vetor resultado tem a soma dos elementos em cada linha
```

```
    }

    return NormaInfinito(result, r);//e aqui usamos a norma infinito nesse vetor com as somas
}

//calcula a norma infinito da matriz A (referenciada por matrizA) de tamanho rxc
float NormaInfinitoMatriz(float* matrizA, int r, int c)
{
    float result[c], somatorio;
    int i, j;

    float A[r][c];
    int k = 0;
    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
        {
            A[i][j] = *(matrizA + k);
            k++;
        }

    for(j=0;j<c;j++)
    {
        somatorio = 0;//iniciando com 0
        for(i=0;i<r;i++)
        {
            if(A[i][j]<0)
                A[i][j] *= -1;//modulo

            somatorio += A[i][j]; //somatorio
        }
        result[j] = somatorio;// resultado tem todos os somatorios dos elementos de todas as colunas
    }
    return NormaInfinito(result, c);// e aqui usamos a norma infinito do vetor com as somas
}

//calculando a distancia entre duas matrizes A e B (permitindo escolher a norma)
float DistanciaEntreDuasMatrizes(float* matrizA, float* matrizB, int r, int c, char norma)
{
    float result[r][c];
    int i,j;

    float A[r][c], B[r][c];
    int k = 0;
    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
        {
            A[i][j] = *(matrizA + k);
```

```c
                B[i][j] = *(matrizB + k);
                k++;
            }

    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
            result[i][j] = A[i][j] - B[i][j];//subtracao das matrizes elemento a elemento

    switch(norma)
    {
        case '1':
            return Norma1Matriz(&result[0][0], r, c);//chamando a funcao para a norma
        case 'f':
            return NormaFrobenius( &result[0][0] , r, c);//chamando a funcao para a norma
        case 'i':
            return NormaInfinitoMatriz(&result[0][0] ,r,c);//chamando a funcao para a norma
    }
}

int main()
{
    //lendo tamanho do vetor
    int n;
    printf("Digite a dimencao do vetor x: ");
    scanf("%d", &n);

    float x[n];
    //lendo vetor x
    int i, j;
    printf("Digite o vetor x: ");
    for(i=0;i<n;i++)
        scanf("%f", &x[i]);

    //calculando as normas do vetor
    printf("norma 1 de x: %f\n", Norma1(x, n) );
    printf("norma 2 de x: %f\n", Norma2(x, n) );
    printf("norma infinito de x: %f\n", NormaInfinito(x, n) );
    printf("\n");

    //lendo o vetor y
    float y[n];
    printf("Digite um vetor y de dimencao %d", n);
    for(i=0;i<n;i++)
        scanf("%f", &y[i]);

    //calculando as distancias entre x e y variando as normas
    printf("distancia entre x e y com norma 1: %f\n", DistanciaEntreDoisVetores(x, y, n, '1') );
    printf("distancia entre x e y com norma 2: %f\n", DistanciaEntreDoisVetores
```

```c
        (x, y, n, '2') );
    printf("distancia entre x e y com norma infinito: %f\n",
        DistanciaEntreDoisVetores(x, y, n, 'i') );
    printf("\n");

    //calculando o produto interno e o angulo entre os vetores
    printf("produto interno entre dois vetores : %f\n", ProdutoInterno(x, y, n)
        );
    printf("angulo entre x e y : %f\n", acos(CosDoAnguloEntreDoisVetores(x, y,
        n)) );
    printf("\n");

    //repetindo todo o processo para as matrizes (exceto o angulo e o produto
        interno)
    int r,c;
    //forcando que a matriz seja quadrada
    printf("Digite a dimencao de A e B: ");
    scanf("%d", &r);
    c = r;
    float A[r][c], B[r][c];

    printf("Digite a matriz A: ");
    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
            scanf("%f", &A[i][j]);

    printf("Digite a matriz B: ");
    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
            scanf("%f", &B[i][j]);


    printf("norma 1 de A: %f\n", Norma1Matriz(&A[0][0], r, c) );
    printf("norma 1 de B: %f\n", Norma1Matriz(&B[0][0], r, c) );

    printf("\n");

    printf("norma frobenius de A: %f\n", NormaFrobenius(&A[0][0], r, c) );
    printf("norma frobenius de B: %f\n", NormaFrobenius(&B[0][0], r, c) );

    printf("\n");

    printf("norma infinito de A: %f\n", NormaInfinitoMatriz(&A[0][0], r, c) );
    printf("norma infinito de B: %f\n", NormaInfinitoMatriz(&B[0][0], r, c) );

    printf("\n\n");

    printf("distancia entre A e B com norma 1: %f\n",
        DistanciaEntreDuasMatrizes(&A[0][0], &B[0][0], r, c, '1') );
    printf("distancia entre A e B com norma de Forbenius: %f\n",
        DistanciaEntreDuasMatrizes(&A[0][0], &B[0][0], r, c, 'f') );
    printf("distancia entre A e B com norma infinito: %f\n",
        DistanciaEntreDuasMatrizes(&A[0][0], &B[0][0], r, c, 'i') );
```

```

}
```

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 cdigo 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:

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

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

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

Assim obtemos a norma da matriz inversa.
Feito os cálculos, mostraremos os resultados na tela e assim finalizamos nossos cálculos.

- Código de implementação :

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

double Norma1(double* enderecoMatriz, int ordemMatriz)
{
    int i,j,k = 0; //variaveis de iteracao
    double somatorio = 0; //variavel para realizar o somatorio
```

```
    double norma1 = 0; //valor da norma 1
    double Matriz[ordemMatriz][ordemMatriz];

    //Necessario para passarmos os elementos da matriz A
    for(i = 0; i < ordemMatriz;i++)
    {
        for(j = 0; j < ordemMatriz; j++)
        {
            Matriz[i][j] = *(enderecoMatriz + k);
            k++;
        }
    }

    //Logica da norma 1: Maior soma dos elementos em modulo de cada coluna.
    for(i = 0; i < ordemMatriz;i++)
    {
        for(j = 0; j < ordemMatriz; j++)
        {
            somatorio += fabs(Matriz[j][i]);
        }

        if(norma1 < somatorio)
        {
            norma1 = somatorio;
        }

        somatorio = 0; //zera-se para iniciar a soma de uma nova coluna.
    }

    return norma1;
};

double NormaInfinito(double* enderecoMatriz,int ordemMatriz)
{
    int i,j,k = 0;
    double somatorio = 0;
    double normaInfinito = 0;
    double Matriz[ordemMatriz][ordemMatriz];

    for(i = 0; i < ordemMatriz;i++)
    {
        for(j = 0; j < ordemMatriz; j++)
        {
            Matriz[i][j] = *(enderecoMatriz + k);
            k++;
        }
    }

    //Logica da norma infinito: Maior soma dos elementos em modulo de cada
        linha
    for(i = 0; i < ordemMatriz;i++)
    {
        for(j = 0; j < ordemMatriz; j++)
```

```
        {
            somatorio += fabs(Matriz[i][j]);
        }

        if(normaInfinito < somatorio)
        {
            normaInfinito = somatorio;
        }

        somatorio = 0;
    }

    return normaInfinito;
};


double NormaFrobenius(double* enderecoMatriz, int ordemMatriz)
{
    int i,j,k = 0;
    double somatorio = 0;
    double normaFrobenius = 0;
    double Matriz[ordemMatriz][ordemMatriz];

    for(i = 0; i < ordemMatriz;i++)
    {
        for(j = 0; j < ordemMatriz; j++)
        {
            Matriz[i][j] = *(enderecoMatriz + k);
            k++;
        }
    }

    //Logica da norma Frobenius: A raiz quadrada da soma dos elementos da
        matriz em modulo ao quadrado.
    //Obs: pow(x,n) = funcao que retorna x elevado n.
    for(i = 0; i < ordemMatriz;i++)
    {
        for(j = 0; j < ordemMatriz; j++)
        {
            somatorio += pow(fabs(Matriz[i][j]),2);
        }
    }

    normaFrobenius = sqrt(somatorio);

    return normaFrobenius;
};

int main()
{
    int i,j,n;
    double numeroCondicaoNorma1, numeroCondicaoNormaFrobenius,
        numeroCondicaoNormaInfinito;
```

```

    printf("Qual a ordem da matriz A?\n");
    scanf("%d",&n);

    double A[n][n];

    printf("Qual a matriz A?\n");

    for(i = 0;i < n;i++)
    {
        for(j = 0;j < n;j++)
        {
            scanf("%lf",&A[i][j]);
        }
    }

    printf("Qual o n mero condi  o na norma 1 de A(retirado do matlab)?\n");
    scanf("%lf",&numeroCondicaoNorma1);

    printf("Qual o n mero condi  o na norma infinito de A(retirado do matlab)?\n");
    scanf("%lf",&numeroCondicaoNormaInfinito);

    printf("Qual o n mero condi  o na norma de frobenius de A(retirado do matlab)?\n");
    scanf("%lf",&numeroCondicaoNormaFrobenius);

    //Calculo das normas da matriz inversa

    double matrizInversaNorma1, matrizInversaNormaFrobenius, matrizInversaNormaInfinito;
    double norma1A,normaFrobeniusA,normaInfinitoA;

    //Necessario para enviar a matriz A para as funcoes que calcularao as normas.
    /*Obs: Nao eh possivel realizar a passagem de matriz de ordem nao constante por passegem por valor,
    apenas por referencia, por isso estamos usando ponteiros para realizar a passagem por referencia.*/
    double* enderecoA;
    enderecoA = &A[0][0];

    //Calculando as normas da matriz A.
    norma1A = Norma1(enderecoA,n);
    normaFrobeniusA = NormaFrobenius(enderecoA,n);
    normaInfinitoA = NormaInfinito(enderecoA,n);

    //Calculando as normas da matriz inversa, utilizando a formula: ||A-  || = cond(A) / ||A||

    matrizInversaNorma1 = numeroCondicaoNorma1 / norma1A;
    matrizInversaNormaFrobenius = numeroCondicaoNormaFrobenius / normaFrobeniusA;
```

```
    matrizInversaNormaInfinito = numeroCondicaoNormaInfinito / normaInfinitoA;

    //Resultados

    printf("Matriz inversa:\n");

    printf("Norma 1 = %f\n",matrizInversaNorma1);
    printf("Norma de Frobenius = %f\n",matrizInversaNormaFrobenius);
    printf("Norma infinito = %f\n",matrizInversaNormaInfinito);

    return 0;
}
```

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:

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

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 questo 1 da prova. Ratificando, o resultado da multiplicação será um vetor.

Finalizando, calcularemos a norma de 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 :

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

//Funcoes que calculam as normas de vetor ou matriz.

double Norma1Vetor(double* vetor,int tamanhoVetor)
{
    double norma1 = 0; // resultado da norma 1
    int i = 0; //variavel de iteracao

    //Logica da norma 1 de vetor: resultado da soma de todos os elementos em modulo do vetor
    for(i = 0;i < tamanhoVetor;i++)
    {
        norma1 += fabs(vetor[i]);
    }

    return norma1;
};

double NormaInfinitoVetor(double* vetor,int tamanhoVetor)
{
    double normaInfinito = 0;
    int i = 0;

    //Logica da norma infinito: Maior elemento em modulo do vetor.
    for(i = 0;i < tamanhoVetor;i++)
    {
        if(fabs(vetor[i]) > normaInfinito)
        {
            normaInfinito = fabs(vetor[i]);
        }
    }

    return normaInfinito;
};

//Mesma funcao usada na questao 10
double Norma1Matriz(double* enderecoMatriz,int ordemMatriz)
{
    int i,j,k = 0;
    double somatorio = 0;
    double norma1 = 0;
    double Matriz[ordemMatriz][ordemMatriz];

    //Necessario para a funcao receber todos os elementos da matriz passada como parametro.
    for(i = 0; i < ordemMatriz;i++)
    {
        for(j = 0; j < ordemMatriz; j++)
        {
            Matriz[i][j] = *(enderecoMatriz + k);
            k++;
```

```
        }
    }

    for(i = 0; i < ordemMatriz;i++)
    {
        for(j = 0; j < ordemMatriz; j++)
        {
            somatorio += fabs(Matriz[j][i]);
        }

        if(norma1 < somatorio)
        {
            norma1 = somatorio;
        }

        somatorio = 0; //necessario para o somatorio ser reutilizado.
    }

    return norma1;
};

//Mesma funcao usada na questao 10
double NormaInfinitoMatriz(double* enderecoMatriz,int ordemMatriz)
{
    int i,j,k = 0;
    double somatorio = 0;
    double normaInfinito = 0;
    double Matriz[ordemMatriz][ordemMatriz];

    for(i = 0; i < ordemMatriz;i++)
    {
        for(j = 0; j < ordemMatriz; j++)
        {
            Matriz[i][j] = *(enderecoMatriz + k);
            k++;
        }
    }

    for(i = 0; i < ordemMatriz;i++)
    {
        for(j = 0; j < ordemMatriz; j++)
        {
            somatorio += fabs(Matriz[i][j]);
        }

        if(normaInfinito < somatorio)
        {
            normaInfinito = somatorio;
        }

        somatorio = 0;
    }

```

```
    return normaInfinito;
};

int main()
{
    int i,j,n;
    double numeroCondicaoAproximadoNorma1,numeroCondicaoAproximadoNormaInfinito;

    printf("Qual a ordem da matriz?\n");
    scanf("%d",&n);

    double Matriz[n][n];

    printf("Qual a matriz?\n");

    for(i = 0;i < n;i++)
    {
        for(j = 0;j < n;j++)
        {
            scanf("%lf",&Matriz[i][j]);
        }
    }

    double MatrizInversa[n][n];

    printf("Entre com a matriz inversa obtida no Matlab\n");

    for(i = 0;i < n;i++)
    {
        for(j = 0;j < n;j++)
        {
            scanf("%lf",&MatrizInversa[i][j]);
        }
    }

    double w[n]; //vetor w qualquer

    printf("Entre com um vetor w qualquer,com exce  o do vetor nulo, para o c lculo do limitante inferior\n");

    for(i = 0;i < n;i++)
    {
        scanf("%lf",&w[i]);
    }

    //Multiplica  o da matriz inversa com o vetor w
    //Mesmo raciocinio retirado da questao numero 1.

    double MatrizResultadoInversaW[n]; //Resultado da multiplicacao entre a inversa e w.

    //Zerar todos os elementos para eliminar os numeros de memoria contidos na
```

```
        variavel.
    for(j = 0;j < n;j++)
    {
        MatrizResultadoInversaW[j] = 0;
    }

    for(i = 0;i < n;i++)
    {
        for(j = 0;j < n;j++)
        {
            MatrizResultadoInversaW[i] += MatrizInversa[i][j] * w[j];
        }
    }

    //Variaveis de endereco necessarias para serem enviadas para as funcoes para o calculo das normas.
    double* enderecoMatriz = &Matriz[0][0];
    double* enderecoMatrizInversaW = &MatrizResultadoInversaW[0];
    double* enderecoW = &w[0];

    //Calculando o limitante inferior do n mero condicao, de acordo com a formula dada 2.2.28 no livro.

    numeroCondicaoAproximadoNorma1 = Norma1Matriz(enderecoMatriz,n) * Norma1Vetor(enderecoMatrizInversaW,n) / Norma1Vetor(enderecoW,n);
    numeroCondicaoAproximadoNormaInfinito = NormaInfinitoMatriz(enderecoMatriz,n) * NormaInfinitoVetor(enderecoMatrizInversaW,n) / NormaInfinitoVetor(enderecoW,n);

    //Resultados

    printf("O n mero condi o   maior ou igual a %f na norma 1\n", numeroCondicaoAproximadoNorma1);
    printf("O n mero condi o   maior ou igual a %f na norma infinito\n", numeroCondicaoAproximadoNormaInfinito);

    return 0;
}
```

Listing 11: cálculo do limitante inferior de uma matriz A_{nxn}

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:

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

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

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

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 r̂ (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 x̂ do MATLAB. Obtido isso, efetuaremos. $b - A \cdot \hat{x}$.

Com o número condição em mãos e o valor de r̂, podemos calcular o nosso objetivo com o antepasso de calcular a norma 2 de r̂ 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 execício, 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 :

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

//Funcao da norma 2 de um vetor ja aplicada em exercicios anteriores.
```

```
double Norma2(double* vetor,int tamanhoVetor)
{
    double norma2 = 0;
    int i = 0;

    for(i = 0;i < tamanhoVetor;i++)
    {
        norma2 += pow(fabs(vetor[i]),2);
    }

    norma2 = sqrt(norma2);

    return norma2;
};

int main()
{
    int i,j,ordem;

    printf("Qual a ordem do sistema?\n");
    scanf("%d",&ordem);

    double A[ordem][ordem];

    printf("Qual a matriz do sistema?\n");
    for(i = 0; i < ordem;i++)
    {
        for(j = 0;j < ordem;j++)
        {
            scanf("%lf",&A[i][j]);
        }
    }

    double b[ordem];

    printf("Qual o vetor b do sistema?\n");
    for(i = 0;i < ordem;i++)
    {
        scanf("%lf",&b[i]);
    }

    double xhat[ordem];

    printf("Qual a solu  o aproximada obtida no matlab?\n");

    for(i = 0;i < ordem;i++)
    {
        scanf("%lf",&xhat[i]);
    }

    double numeroCondicaoNorma2;

    printf("Qual o numero condi  o da norma 2,obtido no matlab, da matriz do
```

```
        sistema?\n");
    scanf("%lf",&numeroCondicaoNorma2);

    //Enderecos da matriz e vetor para serem passados para a funcao.
    double* enderecoB = &b[0];
    double* enderecoA = &A[0][0];

    //Incio dos calculos

    double residuoHat[ordem];

    double vetorResultado[ordem];

    //Calculo do residuohat.

    //Calculando A * xhat

    //Mulplicacao de matriz com vetor

    for(j = 0;j < ordem;j++)
    {
        vetorResultado[j] = 0;
    }

    for(i = 0;i < ordem;i++)
    {
        for(j = 0;j < ordem;j++)
        {
            vetorResultado[i] += A[i][j] * xhat[j];
        }
    }

    //Calculo final do residuohat rhat = b - A*xhat

    for(i = 0;i < ordem;i++)
    {
        residuoHat[i] = b[i] - vetorResultado[i];
    }

    //Mesma ideia para o enderecoA e enderecoB.
    double* enderecoResiduohat = &residuoHat[0];

    //Calculo da proximidade de xhat da solucao real(a sua precisao de acordo
        com a solucao real)
    double precisaoXhatNorma2;

    //Aplicacao da formula.
    precisaoXhatNorma2 = (numeroCondicaoNorma2 * Norma2(enderecoResiduohat,
        ordem)) / Norma2(enderecoB,ordem);

    //Avaliacao do resultado.

    printf("Precisao na norma 2 da solu  o x aproximada: %f\n",
```

```
        precisaoXhatNorma2);

    //O usuario pode inserir um valor de precisao aceitavel para avaliar se a
        solucao aproximada eh boa ou nao para
    //ele.

    double nivelPrecisaoAceitavel;

    printf("Insira um valor de precis o , que ao seu ver,    bom para a
        solu   o aproximada:\n");
    scanf("%lf",&nivelPrecisaoAceitavel);

    //Analise das precisoes
    //Se o nivel de precisao encontrado for menor ou igual ao desejado, a
        solucao eh adequada.
    if(precisaoXhatNorma2 <= nivelPrecisaoAceitavel)
        printf("Usando a norma 2, a solu   o aproximada    uma boa
            aproxima   o para seu caso\n");
    else
        printf("Usando a norma 2, a solu   o aproximada    uma ruim
            aproxima   o para seu caso\n");

    return 0;
}
```

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 :

```
#include<stdio.h>

int main()
{
    int n;
    printf("Qual a ordem da matriz?\n");
    scanf("%d", &n);

    float A[n][n], At[n][n];
    int i, j;
    printf("Qual a matriz A?\n");
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
        {
            scanf("%f", &A[i][j]);
```

```
            At[j][i] = A[i][j];
        }

    int EhOrtogonal = 1;

    float Result[n][n];
    for(i = 0; i < n ; i++)
    {
        int l;
        for(l = 0; l < n ; l++)
        {
            float somatorio = 0;

            for(j = 0; j < n ; j++)
            {
                somatorio += A[i][j]*At[j][l];
            }
            Result[i][l] = somatorio;
        }
    }


    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            if(i==j)
            {
                if(Result[i][j]!=1)
                {
                    EhOrtogonal = 0;
                    break;
                }
            }
            else
            {
                if(Result[i][j]!=0)
                {
                    EhOrtogonal = 0;
                    break;
                }
            }
        }
        if(EhOrtogonal == 0)
            break;
    }

    if(EhOrtogonal)
        printf("   ortogonal\n");
    else
        printf("n o   ortogonal\n");

}
```

Listing 13: Avaliar se uma matriz é ortogonal

Exercício 14

Nesta questão a solução atribuida foi escolher os valores para os rotatores arbritariamente, 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:

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

A matriz R é gerada através da fórmula: $Q\dot{A}$(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 valida
E assim achando a solução resolvendo o sistema triangular inferior:

$$R \cdot x = y$$

- Código de implementação :

```
#include<stdio.h>

int main()
{
    //operadores de tamanho 2x2 (especificadona questao)
    float A[2][2], R[2][2];
    //lendo amatriz A
    int i, j, k;
    for(i=0;i<2;i++)
        for(j=0;j<2;j++)
            scanf("%f", &A[i][j]);
    //lendo o vetor b
    float b[2];
    for(i=0;i<2;i++)
        scanf("%f", &b[i]);

    //incializando o rotator com angulo de 45 graus
    float Q[2][2] = { {1, -1}, {1, 1} };
```

```c

    // fazendo o calculo da patriz R = Qt.A
    for(i=0;i<2;i++)
    {
        for(k=0;k<2;k++)
        {
            float somatorio = 0;
            for(j=0;j<2;j++)
            {
                // ler nota 1 ao fim do c digo
                somatorio += Q[j][i] * A[j][k];
            }
            R[i][k] = somatorio;
        }
    }

    float Qtb[2];//vetor que gardara a multiplicacao da matriz Qt pelo vetor b
    for(i=0;i<2;i++)
    {
        float somatorio = 0;
        for(j=0;j<2;j++)
            somatorio += Q[j][i]*b[j];//simulando transposta de novo

        Qtb[i] = somatorio;
    }

    float x[2];

    //triangular inferior adaptado (R.x = Qt.b)
    for(i=1;i>=0;i--)
    {
        float somatorio = Qtb[i];
        for(j=1;j>=i;j--)
        {
            if(i==j)
                somatorio /= R[i][j];
            if(j>i)
                somatorio -= R[i][j]*x[j];
        }
        x[i] = somatorio;
    }

    //imprimindo as matrizes
    printf("matriz Q: \n");
    for(i=0;i<2;i++)
    {
        for(j=0;j<2;j++)
            printf("%f ", Q[i][j]);
        printf("\n");
    }

    printf("\n");

```

```
    printf("matriz R: \n");
    for(i=0;i<2;i++)
    {
        for(j=0;j<2;j++)
            printf("%f ", R[i][j]);
        printf("\n");
    }

    printf("\n");

    //imprimindo a solucao
    printf("vetor solu  o x: \n");
    for(i=0;i<2;i++)
        printf("%f\n", x[i]);
}
//nota 1:
/* a formula seria R = Qt*A, mas inverter a posi  o de i e j no acesso a Q
    o suficiente para simular a transposi  o de Q */
```

Listing 14: Decomposição QR

Exercício 15

Letra a:

O execício 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:

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

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

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

O somatório do numerador de cada equao será representa 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 at β_n. Será calculado por vez no loop o valor de "$|a_{ji}| \cdot \beta_j$" e sendo adicionado no "somatorio". Quando finalizar, o valor de "beta[i]" será o "somatorio" dividido por 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:

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

A matriz B representa os valores dos coeficientes de cada x quando calculamos o valor de $x_1, x_2, x_3, \ldots, x_{n-1}, x_n$. 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 :

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

//Funcoes para calcular normas ja usadas anteriormente.
double Norma1(double* enderecoMatriz,int ordemMatriz)
{
    int i,j,k = 0;
    double somatorio = 0;
    double norma1 = 0;
    double Matriz[ordemMatriz][ordemMatriz];

    for(i = 0; i < ordemMatriz;i++)
    {
        for(j = 0; j < ordemMatriz; j++)
        {
            Matriz[i][j] = *(enderecoMatriz + k);
            k++;
        }
    }

    for(i = 0; i < ordemMatriz;i++)
    {
        for(j = 0; j < ordemMatriz; j++)
        {
            somatorio += fabs(Matriz[j][i]);
        }

        if(norma1 < somatorio)
        {
            norma1 = somatorio;
        }

        somatorio = 0;
    }

    return norma1;
};

double NormaInfinito(double* enderecoMatriz,int ordemMatriz)
{
    int i,j,k = 0;
    double somatorio = 0;
    double normaInfinito = 0;
    double Matriz[ordemMatriz][ordemMatriz];

    for(i = 0; i < ordemMatriz;i++)
    {
        for(j = 0; j < ordemMatriz; j++)
        {
            Matriz[i][j] = *(enderecoMatriz + k);
            k++;
        }
    }
```

```

    for(i = 0; i < ordemMatriz;i++)
    {
        for(j = 0; j < ordemMatriz; j++)
        {
            somatorio += fabs(Matriz[i][j]);
        }

        if(normaInfinito < somatorio)
        {
            normaInfinito = somatorio;
        }

        somatorio = 0;
    }

    return normaInfinito;
};


double NormaFrobenius(double* enderecoMatriz,int ordemMatriz)
{
    int i,j,k = 0;
    double somatorio = 0;
    double normaFrobenius = 0;
    double Matriz[ordemMatriz][ordemMatriz];

    for(i = 0; i < ordemMatriz;i++)
    {
        for(j = 0; j < ordemMatriz; j++)
        {
            Matriz[i][j] = *(enderecoMatriz + k);
            k++;
        }
    }

    for(i = 0; i < ordemMatriz;i++)
    {
        for(j = 0; j < ordemMatriz; j++)
        {
            somatorio += pow(fabs(Matriz[i][j]),2);
        }
    }

    normaFrobenius = sqrt(somatorio);

    return normaFrobenius;
};

int main()
{
    int i,j,n;

```

```c
    //Variaveis de validacao dos criterios. Caso seus valores sejam 0, o
        criterio nao eh satisfeito.
    //caso contrario(valor 1), os criterios sao satisfeitos.
    int criterioLinhas = 1, criterioColunas = 1, criterioNorma = 1,
        criterioSassenfeld = 1;

    double somatorio = 0;

    double somatorioLinhas = 0;
    double somatorioColunas = 0;

    printf("Qual a ordem do sistema Ax = b?\n");
    scanf("%d",&n);

    double A[n][n];
    printf("Qual a matriz A?\n");

    for(i = 0;i < n;i++)
    {
        for(j = 0;j < n;j++)
        {
            scanf("%lf",&A[i][j]);
        }
    }

    double b[n];
    printf("Qual o vetor b?\n");

    for(i = 0;i < n;i++)
    {
        scanf("%lf",&b[i]);
    }

    //Aplicar o criterio das linhas

    for(i = 0;i < n;i++)
    {
        for(j = 0;j < n;j++)
        {
            //nao somamos elementos da diagonal.
            if(i != j)
            {
                somatorioLinhas += fabs(A[i][j]);
            }
        }
        //Se o somatorio linha i for maior ou igual ao valor absoluto de aii,
        //o criterio das linhas nao eh satisfeito.
        if(somatorioLinhas >= fabs(A[i][i]))
        {
            criterioLinhas = 0;
            break;
        }

```

```
            somatorioLinhas = 0;
        }

        //Criterio das colunas

        for(i = 0;i < n;i++)
        {
            for(j = 0;j < n;j++)
            {
                //nao somamos elementos da diagonal.
                if(i != j)
                {
                    somatorioColunas += fabs(A[j][i]);
                }
            }

            //Se o somatorio coluna i for maior ou igual ao valor absoluto de aii,
            //o criterio das colunas nao eh satisfeito.
            if(somatorioColunas >= fabs(A[i][i]))
            {
                criterioColunas = 0;
                break;
            }

            somatorioColunas = 0;
        }

        //Criterio de Sassenfeld

        double beta[n]; //vetor que guarda os valores dos betas.
        double maxBeta = 0; //maior valor de beta.

        //Inicio da aplica  o da formula do criterio de Sassenfeld

        //Calculo do beta[0](primeiro beta)
        for(j = 0;j < n;j++)
        {
            //Nao se pode somar o elemento da diagonal ii no calculo do beta(i).
            if(j != 0)
                somatorio += fabs(A[0][j]);
        }

        beta[0] = somatorio / fabs(A[0][0]);
        somatorio = 0; //reinicio da variavel para o calculo do proximo beta.

        for(i = 1;i < n;i++)
        {
            //Serve para eliminar o problema dos calculos do somatorio com os betas.
            //Em outras palavras: "N o temos no calculo do beta(i) a opera  o aji * beta(i), temos aji. Para
            //eliminar este problema, atribuimos ao beta(i) o valor de 1, pois todo numero multiplicado por 1 eh
```

```
        //ele mesmo."
        beta[i] = 1;

        for(j = 0;j < n;j++)
        {
            //Nao se pode somar o elemento da diagonal ii no calculo do beta(i).
            if(i != j)
                somatorio += fabs(A[i][j]) * beta[j];
        }

        beta[i] = somatorio / fabs(A[i][i]);
        somatorio = 0;//reiniciando a variavel para o calculo do proximo beta.
    }

    //Comparacao para ver quem eh o maior valor de beta.
    for(i = 0;i < n;i++)
    {
        if(beta[i] > maxBeta)
            maxBeta = beta[i];
    }

    //Se o maior valor de beta for >= 1, o criterio de Sassenfeld nao eh cumprido.
    if(!(maxBeta < 1))
        criterioSassenfeld = 0;

    //Criterio da Norma

    double B[n][n];

    //Pelo raciocenio, o valor de x(k) quando calculamos o valor de x(k) em fun  o de x(k - n),...,x(k - 1),x(k),x(k + 1),...,
    //x(k + n) sera sempre zero, devido a definicao dos metodos iterativos.
    //Isso representa que os elementos da diagonal da matriz B sempre ser o zero, pois x = Bx + g.
    for(i = 0;i < n;i++)
    {
        B[i][i] = 0;
    }

    //Calculando so valores de B.
    //Esta parte representa o processo de isolamento da variavel x(k) em funcao das outras variaveis x.
    //A matriz B representa o valor dos coeficientes das variaveis x quando serao usadas no calculo da variavel x(k).
    for(i = 0;i < n;i++)
    {
        for(j = 0;j < n;j++)
        {
            if(i != j)
                B[i][j] = -1 * (A[i][j] / A[i][i]);
        }
```

```
    }
    //Endereco para a passagem para as funcoes.
    double* enderecoB = &B[0][0];

    // Validacao do criterio da norma: Para qualquer norma de B, temos ||B|| < 1. Se ocorre o contrario, o criterio nao
    //eh satisfeito.
    if((Norma1(enderecoB,n) >= 1) && (NormaInfinito(enderecoB,n) >= 1) && (NormaFrobenius(enderecoB,n) >= 1))
        criterioNorma = 0;

    //Resultados
    //Estaremos aqui comparando cada criterio com seu respectivo metodo iterativo.

    if(criterioLinhas == 1 || criterioColunas == 1)
        printf("O crit rio das linhas e colunas convergem.Ent o, o m todo de Jacobi e Gauss-Seidel convergem.\n");
    else
        printf("Os crit rios de linhas ou colunas n o convergem. N o podemos afirmar com certeza que o m todo de Jacobi e Gauss-Seidel convergem.\n");

    if(criterioSassenfeld == 1)
        printf("O crit rio de Sassenfeld   satisfeito. Ent o, o m todo de Gauss-Seidel converge.\n");
    else
        printf("O crit rio de Sassenfeld n o   satisfeito. N o podemos afirmar com certeza que o m todo de Gauss-Seidel converge.\n");

    if(criterioNorma == 1)
        printf("O crit rio da norma   satisfeito. Portanto o m todo iterativo com a matriz B converge.\n");
    else
        printf("O crit rio da norma n o   satisfeito.N o podemos afirmar que o m todo iterativo com a matriz B converge.\n");

    return 0;
}
```

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

Letra b :

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

$$x_1^{k+1} = \frac{b_1 - a_{12} \cdot x_2^k - a_{13} \cdot x_3^k - \ldots - a_{1n} \cdot x_n^k}{a_{11}}$$

$$x_2^{k+1} = \frac{b_2 a_{21} \cdot x_1^k a_{23} \cdot x_3^k - \ldots - a_{2n} \cdot x_n^k}{a_{22}}$$

$$\ldots$$

$$x_n^{k+1} = \frac{b_n - a_{n1} \cdot x_1^k - a_{n2} \cdot x_2^k - \ldots - a_{nn-1} \cdot x_{n-1}^k}{a_{nn}}$$

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, atribuirmos 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 no 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 :

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

int main()
{
    int i,j,h,n; //i,j e h -----> Variaveis de iteracao de loop.
                 //n -----> ordem da matriz A do sistema Ax = b.

    printf("Qual a ordem do sistema Ax = b?\n");
    scanf("%d",&n);

    double A[n][n];
    printf("Qual a matriz A?\n");

    for(i = 0;i < n;i++)
    {
        for(j = 0;j < n;j++)
        {
            scanf("%lf",&A[i][j]);
        }
    }

    double b[n];
```

```
    printf("Qual o vetor b?\n");

    for(i = 0;i < n;i++)
    {
        scanf("%lf",&b[i]);
    }

    double xK[n]; //x(k)
    double xKmais1[n]; //x (k + 1)
    double xK2[n]; //Servira para guardar os valores de x(k) antes que o mesmo
       receba o valor
                    //de x(k + 1) depois que a iteracao i for feita.
                    //Necessario para o caculo do criterio de parada.

    printf("Escolha a solu  o inicial para a itera  o de Jacobi:\n");

    for(i = 0;i < n;i++)
    {
        scanf("%lf",&xK[i]);
    }

    double p; //valor do criterio de parada.

    printf("Escolha um valor de crit rio de parada do algoritmo:\n");
    scanf("%lf",&p);

    //Implementacao do metodo de Jacobi

    int parada = 0; //Varavel que ira controlar o loop do algoritmo.
                    //Quando parada == 1, o loop "while" eh rompido.

    while(parada == 0)
    {
        //Calculo de cada valor de x(k + 1) na iteracao, considerando o
            isolamento de cada x em cada linha do sistema.
        for(i = 0;i < n;i++)
        {
            xKmais1[i] = b[i];

            for(j = 0;j < n;j++)
            {
                //Se j == i, estaremos usando o valor de xj para calcular xj,
                    sendo algo sem sentido.
                if(j != i)
                {
                    xKmais1[i] += -1 * (A[i][j] * xK[j]);
                }
            }

            xKmais1[i] = xKmais1[i] / A[i][i];

            //Quando chegarmos no calculo do ultimo x, iremos atualizar o valor
                 de x(k) para proxima iteracao.
```

```
            if(i == n - 1)
            {
                for(h = 0;h < n;h++)
                {
                    xK2[h] = xK[h]; //Uso de criterio de parada.
                    xK[h] = xKmais1[h]; //atualizacao de x(k)
                }
            }
        }

        for(h = 0;h < n;h++)
        {
            //se x(k) for diferente de 0, iremos verificar o criterio de parada.
            if(xK2[h] != 0)
            {
                //Se cumprirmos a condicao, a iteracao sera continuada.
                if((fabs(xKmais1[h] - xK2[h]) / fabs(xK2[h])) > p)
                {
                    break;
                }
            }

            //Se chegarmos no ultimo valor de x e o loop nao for rompido, entao podemos dizer
            //que o criterio de parada foi cumprido e sairemos do loop principal.
            if(h == n - 1 && xK2[h] != 0)
            {
                parada = 1;
            }
        }
    }

    //Resultado(solucao aproximada do sistema)

    printf("Solu  o aproximada:\n");

    for(i = 0;i < n;i++)
    {
        printf("x[%d] = %f\n",i,xKmais1[i]);
    }

    return 0;
}
```

Listing 16: resolução do sistema pelo método interativo de Jaccobi

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:

$$x_1^{k+1} = \frac{b_1 - a_{12} \cdot x_2^k - a_{13} \cdot x_3^k - \ldots - a_{1n} \cdot x_n^k}{a_{11}}$$

$$x_2^{k+1} = \frac{b_2 - a_{21} \cdot x_1^{k+1} - a_{23} \cdot x_3^k - \ldots - a_{2n} \cdot x_n^k}{a_{22}}$$

$$\ldots$$

$$x_n^{k+1} = \frac{b_n - a_{n1} \cdot x_1^{k+1} - a_{n2} \cdot x_2^{k+1} - \ldots - a_{nn-1} \cdot x_{n-1}^{k+1}}{a_{nn}}$$

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 diferenas: 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 "xKmais1[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 :

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

int main()
{
    int i,j,h,n;

    printf("Qual a ordem do sistema Ax = b?\n");
    scanf("%d",&n);

    double A[n][n];
    printf("Qual a matriz A?\n");

    for(i = 0;i < n;i++)
    {
        for(j = 0;j < n;j++)
        {
            scanf("%lf",&A[i][j]);
        }
    }

```

```c
    double b[n];
    printf("Qual o vetor b?\n");

    for(i = 0;i < n;i++)
    {
        scanf("%lf",&b[i]);
    }

    double xK[n]; //x(k)
    double xKmais1[n]; //x (k + 1)
    double xK2[n]; //Servira para guardar os valores de x(k) depois que a iteracao i for feita.
                    //Necessario para o caculo do criterio de parada.

    printf("Escolha a solu  o inicial para a itera  o de Gauss-Seidel:\n");

    for(i = 0;i < n;i++)
    {
        scanf("%lf",&xK[i]);
        xK2[i] = xK[i]; //para guardar o valor inicial de xK para realizar o teste de parada.
    }

    double p; //valor do criterio de parada

    printf("Escolha um valor de crit rio de parada do algoritmo:\n");
    scanf("%lf",&p);

    //Implementacao do metodo de Gauss-Seidel

    int parada = 0; //Varavel que ira controlar o loop do algoritmo.
                    //Quando parada == 1, o loop "while" eh rompido.

    while(parada == 0)
    {
        //Calculo de cada valor de x(k + 1) na iteracao, considerando o isolamento de cada x em cada linha do sistema.
        for(i = 0;i < n;i++)
        {
            xKmais1[i] = b[i];

            for(j = 0;j < n;j++)
            {
                //Se j == i, estaremos usando o valor de xj para calcular xj, sendo algo sem sentido.
                if(j != i)
                {
                    xKmais1[i] += -1 * (A[i][j] * xK[j]);
                }
            }

            xKmais1[i] = xKmais1[i] / A[i][i];
            xK2[i] = xK[i]; //guardar o valor da iteracao k para efeito de
```

```
                teste de parada.
            xK[i] = xKmais1[i]; //x(k) sera x(k + 1) para ser usado na mesma iteracao.Nesta linha nos diferenciamos do
                                //metodo de Jacobi, pois atualizamos o valor de x(k) na mesma iteracao.
        }

        for(h = 0;h < n;h++)
        {
            //se x(k) for diferente de 0, iremos verificar o criterio de parada.
            if(xK2[h] != 0)
            {
                //Se cumprirmos a condicao, a iteracao sera continuada.
                if((fabs(xKmais1[h] - xK2[h]) / fabs(xK2[h])) > p)
                {
                    break;
                }
            }

            //Se chegarmos no ultimo valor de x e o loop nao for rompido, entao podemos dizer
            //que o criterio de parada foi cumprido e sairemos do loop principal.
            if(h == n - 1 && xK2[h] != 0)
            {
                parada = 1;
            }
        }
    }

    //Resultado(solucao aproximada do sistema)

    printf("Solu  o aproximada:\n");

    for(i = 0;i < n;i++)
    {
        printf("x[%d] = %f\n",i,xKmais1[i]);
    }

    return 0;
}
```

Listing 17: resolução de um sistema pelo método interativo 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:

$$x_1^{k+1} = (1-\omega)\cdot x_1^k + \frac{\omega}{a_{11}}\cdot(b_1 - a_{12}\cdot x_2^k - a_{13}\cdot x_3^k - \ldots - a_{1n}\cdot x_n^k)$$

$$x_2^{k+1} = (1-\omega)\cdot x_2^k + \frac{\omega}{a_{22}}\cdot(b_2 - a_{21}\cdot x_1 k + 1 - a_{23}\cdot x_3^k - \ldots - a_{2n}\cdot x_n^k)$$

$$\vdots$$

$$x_n^{k+1} = (1-\omega)\cdot x_n^k + \frac{\omega}{a_{nn}}\cdot(b_n - a_{n1}\cdot x_1^{k+1} - a_{n2}\cdot x_2^{k+1} - \ldots - a_{n(n-1)}\cdot x_{n-1}^{k+1})$$

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 ômega 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 ômega. 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 ômega 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 ômega utilizado antes da iteração iniciar.

Agora podemos comear 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 w) · 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 :

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

/*Como gerar numeros aleatorios em ponto flutuante:
  Seja ]a;b[ o intervalo com o qual se deseja gerar o conjunto de numeros
      aletorios.
  Entao, o "aleatorio" gerado sera igual a:
                a + (b - a) * ((double)(rand())/RAND_MAX)
    Com isso, geraremos numeros de ponto flutuante aleatorios no intervalo ]a;b
        [.
*/

//Ira realizar o metodo SOR
```

```
void MetodoSOR(double*matrizA,double*matrizB,double criterioParada,int
    ordemSistema)
{
    int i,j,h,k = 0; //variaveis de iteracao
    double A[ordemSistema][ordemSistema];
    double b[ordemSistema];

    //Receber os elementos da matriz A
    for(i = 0; i < ordemSistema;i++)
    {
        for(j = 0; j < ordemSistema; j++)
        {
            A[i][j] = *(matrizA + k);
            k++;
        }
    }

    //Receber os elementos do vetor b
    for(i = 0; i < ordemSistema; i++)
    {
        b[i] = *(matrizB + i);
    }

    double xK[ordemSistema]; //x(k)
    double xKmais1[ordemSistema]; //x (k + 1)
    double xK2[ordemSistema]; //Servira para guardar os valores de x(k) depois
        que a iteracao i for feita.
                    //Necessario para o caculo do criterio de parada.

    printf("Escolha a solu  o inicial para a itera  o SOR:\n");

    for(i = 0;i < ordemSistema;i++)
    {
        scanf("%lf",&xK[i]);
        xK2[i] = xK[i]; //para guardar o valor inicial de xK para realizar o
            teste de parada.
    }

    //Implementa  o do metodo SOR.

    //Agora, iremos estimar o omega otimo. Como a matriz eh positiva definida,
        podemos afirmar
    //que o omega esta entre 0 < w < 2. Usaremos a funcao de gerador de numeros
         aleatorios do C
    //para obter um numero "aleatorio" entre 0 e 2.

    double w;

    srand((double)time(NULL));
    w = 2 * ((double)(rand())/RAND_MAX);

    printf("w que ser  usado: %f\n",w);

```

Exercício 16 continua na próxima página. . .

```

    //Com o omega, iniciaremos a iteracao do metodo SOR.

    int parada = 0; //Varavel que ira controlar o loop do algoritmo.
                    //Quando parada == 1, o loop "while" eh rompido.

    while(parada == 0)
    {
        //Calculo de cada valor de x(k + 1) na iteracao, considerando o isolamento de cada x em cada linha do sistema.
        for(i = 0;i < ordemSistema;i++)
        {
            xKmais1[i] = b[i];

            for(j = 0;j < ordemSistema;j++)
            {
                if(j != i)
                {
                    xKmais1[i] += -1 * (A[i][j] * xK[j]);
                }
            }

            xKmais1[i] = xKmais1[i] / A[i][i];

            //Nesta parte em exato, encontramos a diferenca do m todo SOR. O uso
            //de uma constante w para acelerar o processo. Afirmamos que o metodo SOR
            //seria apenas o metodo de Gauss-Seidel melhorado. O Gauss-Seidel eh o SOR
            //considerando w = 1.
            //---------------------------------------------------------------------------
            xKmais1[i] *= w; //multiplicando o w como diz a formula.
            xKmais1[i] += (1 - w) * xK[i]; // somando com (1 - w) * x(k) como diz a formula.
            //  ---------------------------------------------------------------------------

            xK2[i] = xK[i]; //guardar o valor da iteracao k para efeito de teste de parada.
            xK[i] = xKmais1[i]; //x(k) sera x(k + 1) para ser usado na mesma iteracao.Nesta linha encontramos
            //uma semalhanca com o Gauss-Seidel, pois atualizamos o valor de x(k) na mesma iteracao.
        }

        for(h = 0;h < ordemSistema;h++)
        {
         //se x(k) for diferente de 0, iremos verificar o criterio de parada.
            if(xK2[h] != 0)
            {
                //Se cumprirmos a condicao, a iteracao sera continuada.
```

```c
                if((fabs(xKmais1[h] - xK2[h]) / fabs(xK2[h])) > criterioParada)
                {
                    break;
                }
            }

            //Se chegarmos no ultimo valor de x e o loop nao for rompido, entao podemos dizer
            //que o criterio de parada foi cumprido e sairemos do loop principal.
            if(h == ordemSistema - 1 && xK2[h] != 0)
            {
                parada = 1;
            }
        }
    }

    //Resultado(solucao aproximada do sistema)

    printf("Solu  o aproximada:\n");

    for(i = 0;i < ordemSistema;i++)
    {
        printf("x[%d] = %f\n",i,xKmais1[i]);
    }
};

int main()
{
    int i,j,n;

    printf("Qual a ordem do sistema?\n");
    scanf("%d",&n);

    double A[n][n];
    printf("Qual a matriz A positiva definida?\n");

    for(i = 0;i < n;i++)
    {
        for(j = 0;j < n;j++)
        {
            scanf("%lf",&A[i][j]);
        }
    }

    double b[n];
    printf("Qual o vetor b?\n");

    for(i = 0;i < n;i++)
    {
        scanf("%lf",&b[i]);
    }

```

```
    double criterioParada;

    printf("Escolha um valor de crit rio de parada do algoritmo:\n");
    scanf("%lf",&criterioParada);

    //Endereco de A e b para enviar a fun  o que realizara o metodo SOR.
    double* enderecoA = &A[0][0];
    double* enderecoB = &b[0];

    //Funcao que executara o m todo SOR
    MetodoSOR(enderecoA,enderecoB,criterioParada,n);

    return 0;
}
```

Listing 18: resolução do sistema pelo método interativo SOR

Exercício 17

Este execício 17 é uma outra alternativa do mótodo 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 ômega(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_{3x3}, 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 :

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

int main()
{
    int i,j,h;

    double A[3][3];
    printf("Qual a matriz A de ordem 3,positiva definida e tridiagonal?\n");

    for(i = 0;i < 3;i++)
    {
        for(j = 0;j < 3;j++)
        {
            scanf("%lf",&A[i][j]);
        }
    }

    double b[3];
    printf("Qual o vetor b?\n");

    for(i = 0;i < 3;i++)
    {
        scanf("%lf",&b[i]);
    }

    double xK[3]; //x(k)
    double xKmais1[3]; //x (k + 1)
    double xK2[3]; //Servira para guardar os valores de x(k) depois que a iteracao i for feita.
                   //Necessario para o caculo do criterio de parada.

    printf("Escolha a solu  o inicial para a itera  o SOR:\n");

    for(i = 0;i < 3;i++)
    {
        scanf("%lf",&xK[i]);
        xK2[i] = xK[i]; //para guardar o valor inicial de xK para realizar o teste de parada.
    }
```

```

    double p;

    printf("Escolha um valor de crit rio de parada do algoritmo:\n");
    scanf("%lf",&p);

    //Implementacao do metodo SOR.

    //Aplicacao da formula generica.
    double pTj = 0;

    pTj += (A[2][1] * A[1][2]) / (A[2][2] * A[1][1]);
    pTj += (A[1][0] * A[0][1]) / (A[1][1] * A[0][0]);
    pTj = sqrt(pTj);

    //Agora, iremos calcular o omega otimo atraves de sua formula.

    double w = 0;

    w = 2 / (1 + sqrt(1 - (pTj * pTj)));

    printf("w = %f\n",w);

//
    --------------------------------------------------------------------------------------------------------

    //Com o omega, iniciaremos a iteracao do metodo SOR.

    int parada = 0;//Varavel que ira controlar o loop do algoritmo.
                   //Quando parada == 1, o loop "while"   rompido.

    while(parada == 0)
    {
        //Calculo de cada valor de x(k + 1) na iteracao, considerando o
            isolamento de cada x em cada linha do sistema.
        for(i = 0;i < 3;i++)
        {
            xKmais1[i] = b[i];

            for(j = 0;j < 3;j++)
            {
                if(j != i)
                {
                    xKmais1[i] += -1 * (A[i][j] * xK[j]);
                }
            }

            xKmais1[i] = xKmais1[i] / A[i][i];

            //Nesta parte em exato, encontramos a diferenca do metodo SOR. O
                uso
            //de uma constante w para acelerar o processo. Afirmamos que o
                metodo SOR
```

```
            //seria apenas o metodo de Gauss-Seidel melhorado. O Gauss-Seidel eh o SOR
            //considerando w = 1.
            //-----------------------------------------------------
            xKmais1[i] *= w;//multiplicando o w como diz a formula.
            xKmais1[i] += (1 - w) * xK[i];// somando com (1 - w) * x(k) como diz a formula.
            //-----------------------------------------------------


            xK2[i] = xK[i]; //guardar o valor da iteracao k para efeito de teste de parada.
            xK[i] = xKmais1[i]; //x(k) sera x(k + 1) para ser usado na mesma iteracao.Nesta linha encontramos
            //uma semalhanca com o Gauss-Seidel, pois atualizamos o valor de x(k) na mesma iteracao.
        }

        for(h = 0;h < 3;h++)
        {
             //se x(k) for diferente de 0, iremos verificar o criterio de parada.
            if(xK2[h] != 0)
            {
                //Se cumprirmos a condicao, a iteracao sera continuada.
                if((fabs(xKmais1[h] - xK2[h]) / fabs(xK2[h])) > p)
                {
                    break;
                }
            }

            //Se chegarmos no eltimo valor de x e o loop nao for rompido, entao podemos dizer
            //que o criterio de parada foi cumprido e sairemos do loop principal.
            if(h == 2 && xK2[h] != 0)
            {
                parada = 1;
            }
        }
    }
//------------------------------------------------------------------------------------------

    //Resultado(solucao aproximada do sistema)

    printf("Solu  o aproximada:\n");

    for(i = 0;i < 3;i++)
    {
        printf("x[%d] = %f\n",i,xKmais1[i]);
    }

```

```
    return 0;
}
```

Listing 19: resolução do sistema pelo método interativo 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 :

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

float Delta(float A[2][2])
{
    //formula geral de um delta da formula de baskara aplicado no polinomio
        caracteristico de uma matriz generica 2x2
    return pow(A[0][0]+A[1][1],2) - 4*(-1)*(A[1][0]*A[0][1] - A[0][0]*A[1][1]);
}

int main()
{
    float A[2][2];

    int i,j;
    for(i=0;i<2;i++)
        for(j=0;j<2;j++)
            scanf("%f", &A[i][j]);

    //vetor com as duas raizes
    float lambida [2];
    //calculando o delta da baskara
    float delta = Delta(A);
    if(delta>=0)//se maior que zero existe raiz real
    {
        //-b + raiz(dekta)/2a
```

```
        lambida[0] = ( (-(A[0][0]+A[1][1]) + sqrt( delta ) )/-2 );
        //-b - raiz(dekta)/2a
        lambida[1] = ( (-(A[0][0]+A[1][1]) - sqrt( delta ) )/-2 );

        printf("raiz 1: %f    raiz 2: %f\n", lambida[0], lambida[1]);
    }
    else
    {
        printf("delta negativo, n o h  raiz real");
    }

}
```

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 razes, através da fórmula de resoluo do 2^o grau(conhecida como fórmula de Bhaskara).

Observao: Calculamos a derivada de um polinmio(em especial a de 3^o 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 iterao do método de Newton-Raphson se d pela fórmula:

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

Sendo:
x_{k+1} = valor de x na iterao k + 1;
x_k = valor de x na iterao 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:
$\frac{|x_{k+1} - x(x)|}{|x(x)|}$ <=(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 razízes do 2^o 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 3^o grau:)

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

Onde: q(x) será nosso polinômio de 2^o *grau.*
Para encontrá-lo, basta dividir o polinômio de 3^o grau por $(x - x')$ (onde x' é uma das razízes do polinômio de 3^o grau), pelo algoritmo da diviso 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 nfimo, o resto será desprezado para que os cálculos sejam possíveis.

Com a equao do 2^o 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 :

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

int main()
{
    int i,j;

    double Matriz[3][3];
    double vetorLambda[3];

    printf("Qual a matriz 3x3?\n");

    for(i = 0;i < 3;i++)
    {
        for(j = 0;j < 3;j++)
```

```
            {
                scanf("%lf",&Matriz[i][j]);
            }
        }

        //Inicio do calculo do polinomio caracteristico. Sera dividido em passos:

        //Passo 1: Calcular o determinante da matriz 3x3.

        double vetorCoeficientesPolinimio[4];

        //Calculo do coeficiente 1
        vetorCoeficientesPolinimio[0] = -1;

        //Calculo do coeficiente 2
        vetorCoeficientesPolinimio[1] = Matriz[0][0] + Matriz[1][1] + Matriz[2][2];

        //Calculo do coeficiente 3

        vetorCoeficientesPolinimio[2] = Matriz[2][0] * Matriz[0][2];
        vetorCoeficientesPolinimio[2] += Matriz[2][1] * Matriz[1][2];
        vetorCoeficientesPolinimio[2] += Matriz[1][0] * Matriz[0][1];
        vetorCoeficientesPolinimio[2] -= Matriz[0][0] * Matriz[2][2];
        vetorCoeficientesPolinimio[2] -= Matriz[1][1] * Matriz[2][2];
        vetorCoeficientesPolinimio[2] -= Matriz[0][0] * Matriz[1][1];

        //Calculo do coeficiente 4

        vetorCoeficientesPolinimio[3] = Matriz[0][2] * Matriz[1][0] * Matriz[2][1];
        vetorCoeficientesPolinimio[3] += Matriz[0][1] * Matriz[1][2] * Matriz[2][0];
        vetorCoeficientesPolinimio[3] += Matriz[0][0] * Matriz[1][1] * Matriz[2][2];
        vetorCoeficientesPolinimio[3] -= Matriz[2][0] * Matriz[0][2] * Matriz[1][1];
        vetorCoeficientesPolinimio[3] -= Matriz[2][1] * Matriz[1][2] * Matriz[0][0];
        vetorCoeficientesPolinimio[3] -= Matriz[1][0] * Matriz[0][1] * Matriz[2][2];

        //Passo 2: Calcular uma raiz do polinomio.

        //Criterio de parada da iteracao
        double criterioParada;
        printf("Qual o crit rio de parada para o m todo de Newton-Raphson?\n");
        scanf("%lf",&criterioParada);

        //Valor inicial de x(k) na primeira iteracao
        double xK;
        printf("Qual o valor inicial de x para a itera  o?\n");
        scanf("%lf",&xK);

```

```
    int parada = 0;// variavel de controle de parada. Quando parada = 1, a iteracao sera desfeita.
    double xKmais1 = 0; // x(k + 1)

    double derivadaFxk = 0;
    double Fxk = 0;
    double xK2; //sera a memoria do valor antigo de xK. Servira para efeito de testes de parada da iteracao
                //de Newton-Raphson.

    do
    {
        //Calcular f(x(k)): O valor do polinomino caracteristico quando (lambda) = x(k).

        Fxk += vetorCoeficientesPolinimio[0] * pow(xK,3);
        Fxk += vetorCoeficientesPolinimio[1] * pow(xK,2);
        Fxk += vetorCoeficientesPolinimio[2] * xK;
        Fxk += vetorCoeficientesPolinimio[3];

        //Calcular f'(x(k)): Usando a formula da derivada encontrada.

        derivadaFxk += vetorCoeficientesPolinimio[0] * (3 * pow(xK,2));
        derivadaFxk += vetorCoeficientesPolinimio[1] * (2 * xK);
        derivadaFxk += vetorCoeficientesPolinimio[2];

        //Calcular x(k + 1)

        xKmais1 = xK - (Fxk / derivadaFxk); //iteracao de Newton-Raphson.

        xK2 = xK; //guardar o valor antigo de xK para ver se devemos terminar a iteracao.
        xK = xKmais1; //xK sera agora o x(k + 1) na proxima iteracao.

        Fxk = 0;            //Zerar as vari veis para iniciarmos a proxima iteracao.
        derivadaFxk = 0;

        //Teste de parada: Usando a formula  de criterio de parada ja citada.
        if(fabs(xKmais1 - xK2) /fabs(xK2) <= criterioParada)
            parada = 1;


    }while(parada == 0);

    vetorLambda[0] = xKmais1; //A primeira raiz recebe o valor de x(k + 1)



    //Passo 3: Calcular as duas raizes restantes.

    //Coeficientes da equacao do 2  grau.
    double a = -1;
```

```
    double b = vetorCoeficientesPolinimio[1] - vetorLambda[0];
    double c = (-1 * pow(vetorLambda[0],2)) + (vetorLambda[0] *
        vetorCoeficientesPolinimio[1]) + vetorCoeficientesPolinimio[2];

    //Calculo do delta
    double delta = pow(b,2) - (4 * a * c);

    //Calculo das raizes x1 e x2
    vetorLambda[1] = (-1 * b + sqrt(delta)) / 2 * a;
    vetorLambda[2] = (-1 * b - sqrt(delta)) / 2 * a;

//
    ---------------------------------------------------------------------------------

    //Resultados

    printf("Raizes do polin mio:\n");
    printf("x1 = %f\n",vetorLambda[0]);

    //Se o delta for menor que zero, a equacao do segundo grau nao tera raizes
        reias!

    if(delta >= 0)
    {
        printf("x2 = %f\n",vetorLambda[1]);
        printf("x3 = %f\n",vetorLambda[2]);
    }
    else
    {
        printf("N o h  raizes reais para x2 e x3!\n");
    }

    return 0;
}
```

Listing 21: raízes do polinômio característico de uma matrz A_{3x3}

Exercício 20

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

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

Nossa lógica da côlonia foi voltada particularmente a resoluçao de sistemas lineares.

A PSO disponibiliza para nós duas fórmulas que servião 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 manipulaçao 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 uma 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 :

```c
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <stdlib.h>

//Estrutura contendo os elementos da PSO de um morcego.
//Obs: Iremos trabalhar as informacoes do morcegos com ponteiro, ou seja, para podermos atribuir
//valores a eles, precisaremos dar-lhes os enderecos das matrizes onde estaro amarzenadas os vetores
//de posicao, velocidade e melhor posicao de cada morcego.
typedef struct{

    double* posicao;
    double* melhorPosicao;
    double* velocidade;
    double funcaoFitness;
    double melhorFuncaoFitness;

} Morcego;

//Calcuclar a funcao de fitness de um morcego.
double CalcularFuncaoFitness(Morcego morcego, double* enderecoA,double* enderecoB,int ordem)
{
    int i,j,k = 0,h = 0; //em geral trabalharemos com essas variaveis para fazer os loops e
                         //para varer cada valor dos vetores que o morcego contem
                         //(posicao,velocidade,melhor posicao).
    double somatorio = 0;
    double A[ordem][ordem];
    double b[ordem];

    for(i = 0; i < ordem;i++)
    {
        for(j = 0; j < ordem; j++)
        {
            A[i][j] = *(enderecoA + k);
```

```
                k++;
            }
        }

        for(i = 0; i < ordem; i++)
        {
            b[i] = *(enderecoB + i);
        }

        morcego.funcaoFitness = 0;

        for(i = 0;i < ordem;i++)
        {
            somatorio += b[i];

            for(j = 0;j < ordem;j++)
            {
                somatorio -= *(morcego.posicao + h) * A[i][j];
                h++;
            }

                h = 0; //variavel necessaria para varrer os vetores dos morcegos.
                       //se zera aqui para iniciar uma nova varredura.

                morcego.funcaoFitness += fabs(somatorio);
                somatorio = 0; //somar a diferenca da proxima equacao do sistema.
            }

        return morcego.funcaoFitness;
};

/*Como gerar numeros aleatorios em ponto flutuante:
  Seja ]a;b[ o intervalo com o qual se deseja gerar o conjunto de numeros aletorios.
  Ent o, o "aleatorio" gerado sera igual a:
              a + (b - a) * ((double)(rand())/RAND_MAX)
    Com isso, geraremos numeros de ponto flutuante aleatorios no intervalo ]a;b[.
*/

int main()
{
    int i,j,k = 0,h = 0; //variaveis de iteracao
    int ordem; //ordem do sistema.

    srand((double)time(NULL)); //gerador de semente, para gerarmos sequencias de
                               //numeros aleatorios diferentes a cada vez que o
                               //programa for iniciado.

    printf("Qual a ordem do sistema?\n");
    scanf("%d",&ordem);
```

```

    double A[ordem][ordem];

    printf("Qual a matriz A do sistema?\n");
    for(i = 0;i < ordem;i++)
    {
        for(j = 0;j < ordem;j++)
        {
            scanf("%lf",&A[i][j]);
        }
    }

    double b[ordem];

    printf("Qual o vetor b do sistema?\n");
    for(i = 0;i < ordem;i++)
    {
        scanf("%lf",&b[i]);
    }

    //Necessario para obtermos a matriz A e b nas funcoes que futuramente
    //iremos utilizar em nossos calculos.
    double* enderecoA = &A[0][0];
    double* enderecoB = &b[0];

    //Criando os "morcegos"
    //Obs: A funcao round() retorna o valor arredondado do parametro fornecido.
    //Neste caso, a quantidade de morcegos sera igual a 30% do numero de
        equacoes existentes.
    int quantidadeDeMorcegos = round(0.3 * ordem);

    //Usaremos no minimo 2 morcegos para a iteracao, caso os 30% do numero de
        equacoes
    //seja menor que 2.
    if(quantidadeDeMorcegos < 2)
        quantidadeDeMorcegos = 2;

        //Cada linha da matriz das caracteristicas de cada morcego
        //ira representar o numero do morcego; as colunas irao
        //representar os valores contidos no vetor daquele morcego.
        //Essas matrizes seriam como se fossem "Banco de Dados" para
        //amarzenar essas informacoes.

    Morcego morcegos[quantidadeDeMorcegos]; //vetor com todos os morcegos
        disponiveis.
    double posicoesMorcego[quantidadeDeMorcegos][ordem]; //vetor de posicoes dos
         morcegos, necessario para gerarmos
                                                        //aleatoriamente os
                                                            valores.
    double velocidadesMorcegos[quantidadeDeMorcegos][ordem]; //vetor de
        velocidades dos morcegos, necessario para gerarmos
                                                            //aleatoriamente os
                                                                valores.
```

```
    double melhorPosicaoMorcego[quantidadeDeMorcegos][ordem]; //Melhores posicoes de cada morcego.

    //double melhorPosicaoBando[ordem]; //melhor posicao do bando.

    double melhorFuncaoFitness = 10000000000; //foi iniciada com este valor alto pois sera logo substituida pela melhor
                                              //funcao de fitness entre os morcegos.
    double melhorPosicaoBandoTodasAsIteracoes[ordem]; //melhor posicao do bando dentre todas as iteracoes.

    //Criando os morcegos

   for(i = 0; i < quantidadeDeMorcegos;i++)
   {
        Morcego morcego;
        morcegos[i] = morcego;
   }

    //Criando os valores das posi es dos morcegos.
    //Aquela fun o ali dentro ir  gerar alet rios no intervalo de 0 a 20.

        srand((double)time(NULL));

        for(k = 0;k < quantidadeDeMorcegos;k++)
        {
            for(j = 0;j < ordem;j++)
            {
                posicoesMorcego[k][j] = 20 * ((double)(rand())/RAND_MAX);
            }

            morcegos[k].posicao = &posicoesMorcego[k][0];//envio do endereco da linha da matriz
                                                          //que contera as informacoes do morcego.

        }

    //Criando os valores de velocidade dos morcegos entre 0 e 3.

     for(k = 0;k < quantidadeDeMorcegos;k++)
    {
            for(j = 0;j < ordem;j++)
            {
                velocidadesMorcegos[k][j] = 3  * ((double)(rand())/RAND_MAX);
            }

            morcegos[k].velocidade = &velocidadesMorcegos[k][0];//envio do endereco da linha da matriz
                                                                 //que contera as
```

```
                                                            informacoes
                                                            do morcego.
    }

//Obtendo a fun  o de fitness inical para cada morcego.

   for(i = 0; i < quantidadeDeMorcegos;i++)
   {
        morcegos[i].funcaoFitness = CalcularFuncaoFitness(morcegos[i],enderecoA
            ,enderecoB,ordem);
        morcegos[i].melhorFuncaoFitness = morcegos[i].funcaoFitness;
   }

    //Obter a melhor posicao do bando inicialmente

    for(i = 0;i < quantidadeDeMorcegos;i++)
    {
        for(j = 0;j < quantidadeDeMorcegos;j++)
        {

            if(i != j) //Nao faria sentido comparar o morcego com ele mesmo.
            {
                if(morcegos[i].funcaoFitness < morcegos[j].funcaoFitness)
                {
                    melhorFuncaoFitness = morcegos[i].funcaoFitness;

                    for(h = 0;h < ordem;h++)
                    {
                        //melhorPosicaoBando[h] = *(morcegos[i].posicao + h);
                        melhorPosicaoBandoTodasAsIteracoes[h] = *(morcegos[i].
                            posicao + h);
                    }
                }
            }
        }
    }

    for(i = 0;i < quantidadeDeMorcegos;i++)
    {
        morcegos[i].melhorPosicao = &melhorPosicaoMorcego[i][0];
    }


    //Atualizar a melhor posicao de cada morcego inicialmente.

    for(i = 0;i < quantidadeDeMorcegos;i++)
    {
        for(j = 0;j < ordem;j++)
        {
            *(morcegos[i].melhorPosicao + j) = *(morcegos[i].posicao + j);
        }
    }

```

```c
    //Variaveis da formula da PSO
    double A1,A2,A3; //irao ser gerados aleatoriamente sempre.
    double c1 = 2,c2 = 2; //Considerados sempre iguais a 2, por definicao.
    double tempo = 1; //considerado sempre igual a 1, por definicao.

     double criterioParada;
     printf("Qual o crit rio de parada para o algoritmo?\n");
     scanf("%lf",&criterioParada);


//Comecaremos a nossa iteracao...(Consideramos aqui que a variacao do tempo sempre sera igual a 1)
//e os valores constantes de c1 e c2 serao iguais a 2, respectivamente.

     int parada = 0; //quando parada = 1, o loop sera rompido.

     while(parada == 0)
     {
         //Passo 1: Gerar aleatoriamente A1,A2,A3, entre 0 e 1.

         A1 = 1 * ((double)(rand())/RAND_MAX);
         A2 = 1 * ((double)(rand())/RAND_MAX);
         A3 = 1 * ((double)(rand())/RAND_MAX);

         //Passo 2: Atualizar o vetor de velocidade dos morcegos.

         //Aplicando a formula da velocidade do morcego na iteracao.
         for(i = 0; i < quantidadeDeMorcegos;i++)
         {
             for(k = 0; k < ordem;k++)
             {
                 //A1 * V na iteracao i do morcego q

                 *(morcegos[i].velocidade + k) = *(morcegos[i].velocidade + k) * A1;

                 //A2 * c1 * (melhor posicao do morcego q * posicao do morcego q na iteracao i) / tempo

                 *(morcegos[i].velocidade + k) += A2 * c1 *  *(morcegos[i].velocidade + k) + *(morcegos[i].velocidade + k) + ((*(morcegos[i].melhorPosicao + k) - *(morcegos[i].posicao + k)) / tempo);

                 //A3 * c2 * (melhor posicao do bando na iteracao i * posicao do morcego q na iteracao i) / tempo
                 //Obs: Usaremos nesse caso a melhor posicao do bando dentre todas ja feitas e nao a melhor de cada
                 //iteracao. Mudancas necessarias para o sucesso de convergencia do algoritmo.

                 *(morcegos[i].velocidade + k) += A3 * c2 * (melhorPosicaoBandoTodasAsIteracoes[k] - *(morcegos[i].
```

```
                    posicao + k) / tempo);

                //Realizar o controle da velocidade para a mesma nao aumentar
                    muito,
                //pois fugir  do contexto da realiadade.
                if(*(morcegos[i].velocidade + k) >= 2)
                {
                    *(morcegos[i].velocidade + k) = 3 * ((double)(rand())/
                        RAND_MAX);
                }
            }
        }

        //Passo 3: Atualizar as posicoes de cada morcego.

        //Aplicando a formula da posicaoo do morcego na iteracao.
        for(i = 0; i < quantidadeDeMorcegos;i++)
        {
            //S(q)(i + 1) = S(q)(i) + V(q)(i + 1) * tempo
            for(k = 0;k < ordem;k++)
            {
                *(morcegos[i].posicao + k) = *(morcegos[i].posicao + k) + *(
                    morcegos[i].velocidade + k) * tempo;
            }
        }

        //Passo 4: Calcular a fun  o de fitness de cada morcego.

        for(i = 0; i < quantidadeDeMorcegos;i++)
        {
            morcegos[i].funcaoFitness = CalcularFuncaoFitness(morcegos[i],
                enderecoA,enderecoB,ordem);
        }

        //Passo 5: Atualizar a melhor posicao do bando dentre todas as
            iteracoes

        for(i = 0;i < quantidadeDeMorcegos;i++)
        {
            if(morcegos[i].funcaoFitness < melhorFuncaoFitness)
            {
                melhorFuncaoFitness = morcegos[i].funcaoFitness;

                for(j = 0;j < ordem;j++)
                {
                    melhorPosicaoBandoTodasAsIteracoes[j] = *(morcegos[i].
                        posicao + j);
                }

            }
        }

        //printf("Melhor fun  o de fitness: %f\n",melhorFuncaoFitness);
```

```

        //Passo 6: Atualizar a melhor posicao de cada morcego

        for(i = 0;i < quantidadeDeMorcegos;i++)
        {
            if(morcegos[i].funcaoFitness < morcegos[i].melhorFuncaoFitness)
            {
                morcegos[i].melhorFuncaoFitness = morcegos[i].funcaoFitness;

                for(j = 0;j < ordem;j++)
                {
                    *(morcegos[i].melhorPosicao + j) = *(morcegos[i].posicao +
                        j);
                }
            }
        }

        //Passo 7: Verificar o criterio de parada.
        //Se a melhor funcao de fitness for menor que o criterio de parada,
        //encontramos uma solucao aproximada.
        if(melhorFuncaoFitness <= criterioParada)
        {
            parada = 1;
        }
    }

    //Resultados

    printf("Solu  o aproximada do sistema:\n");

    for(i = 0;i < ordem;i++)
    {
        printf("x%d = %f\n",i + 1,melhorPosicaoBandoTodasAsIteracoes[i]);
    }

    return 0;
}
```

Listing 22: Algoritmo colonia de morcegos

Exemplos Questão 1

- TERMINAL

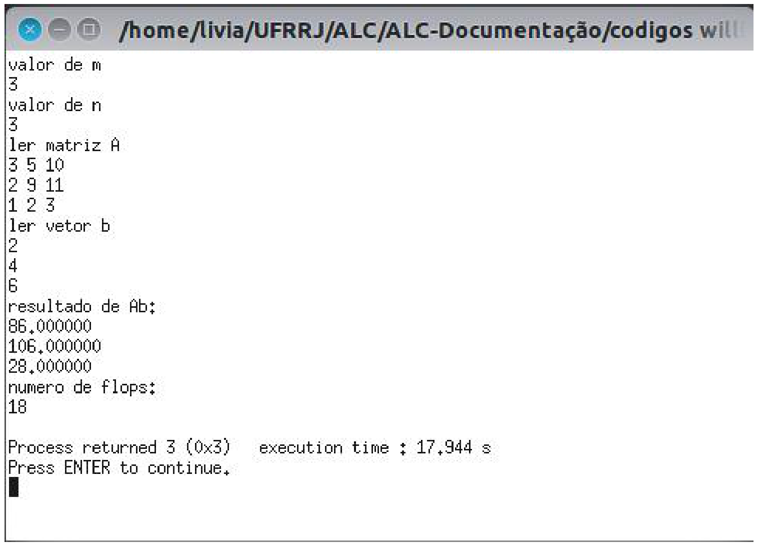


Figura 1: Exemplo.1 que mostra a execução do programa (quest01) no terminal

- MATLAB

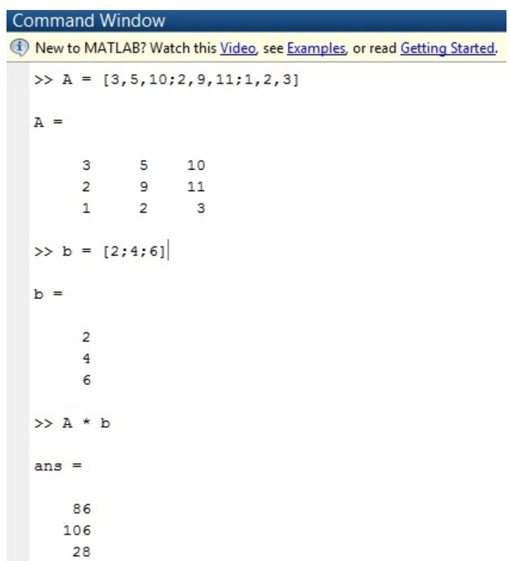


Figura 2: Exemplo que mostra o resultado do MATLAB (quest01) para o exemplo.1

Exemplos Questão 2

- TERMINAL

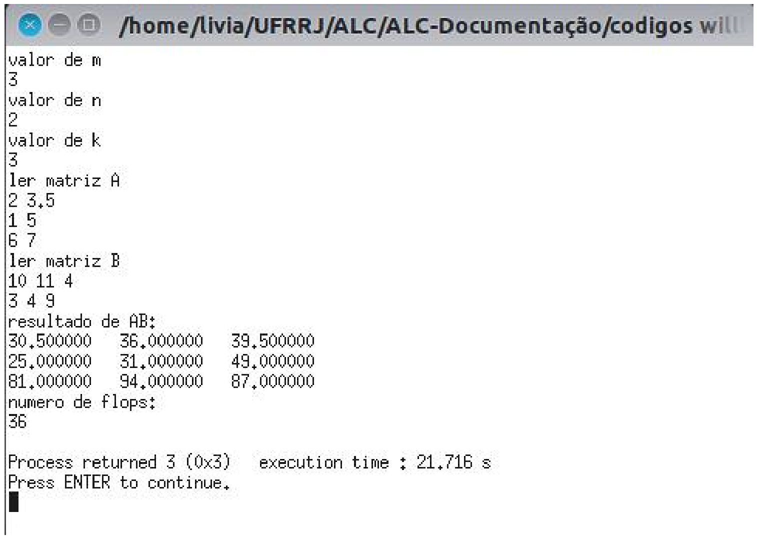


Figura 3: Exemplo.1 que mostra a execução do programa (quest02) no terminal

- MATLAB

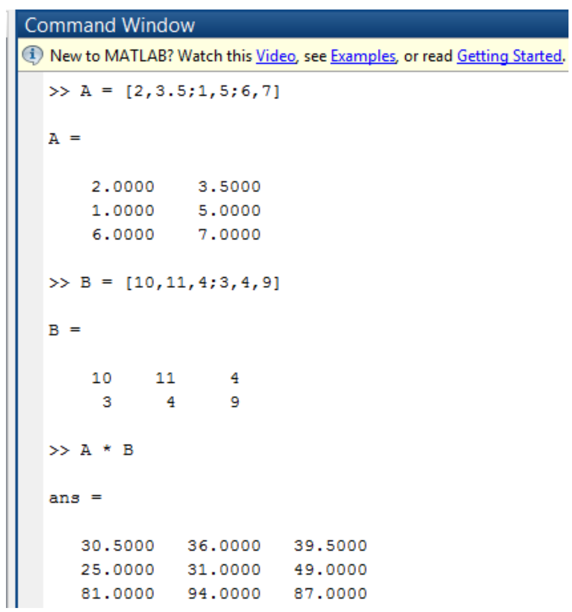


Figura 4: Exemplo que mostra o resultado do MATLAB (quest02) para o exemplo.1

Exemplos Questão 3

- TERMINAL

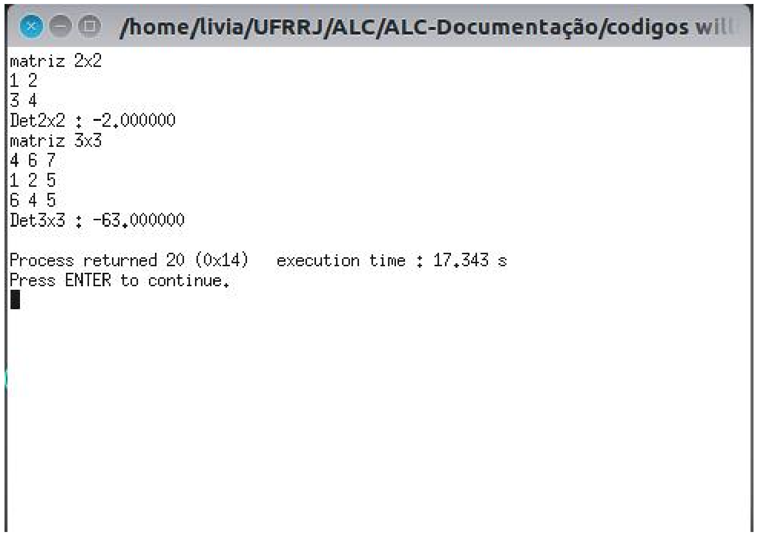


Figura 5: Exemplo.1 que mostra a execução do programa (quest03) no terminal

Exemplos Questão 4

- TERMINAL

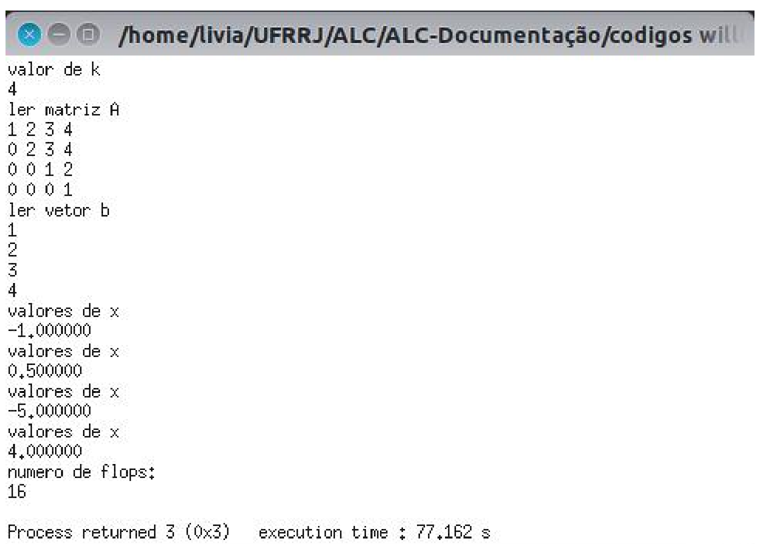
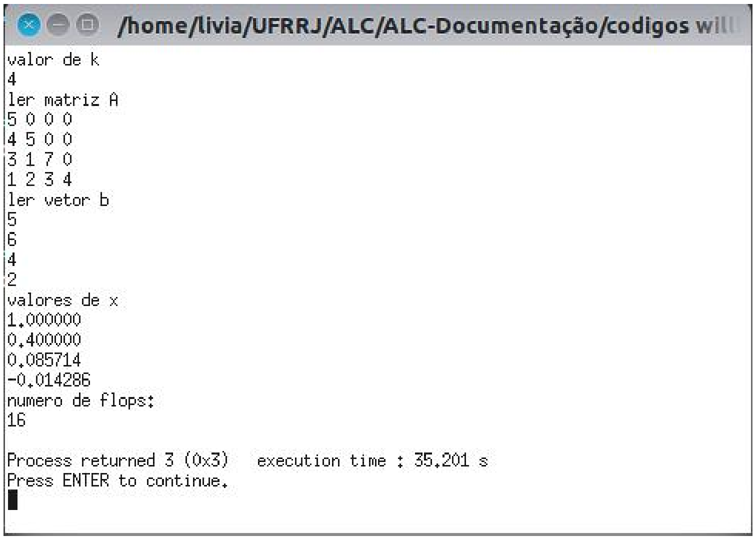


Figura 6: Exemplo.1 que mostra a execução do programa (quest04) no terminal

Exemplos Questão 5

- TERMINAL

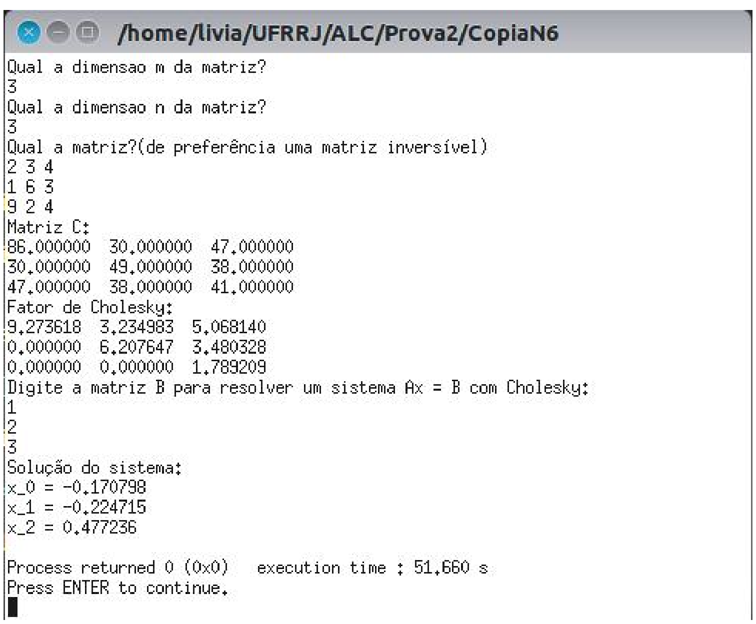


```
/home/livia/UFRRJ/ALC/ALC-Documentação/codigos will
valor de k
4
ler matriz A
5 0 0 0
4 5 0 0
3 1 7 0
1 2 3 4
ler vetor b
5
6
4
2
valores de x
1.000000
0.400000
0.085714
-0.014286
numero de flops:
16

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

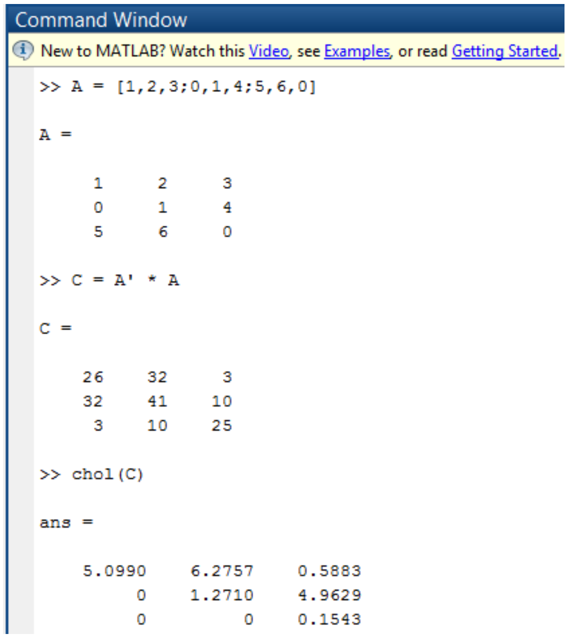


Figura 9: Exemplo que mostra o resultado do MATLAB (quest06) para o exemplo.1

Exemplos Questão 7

- TERMINAL

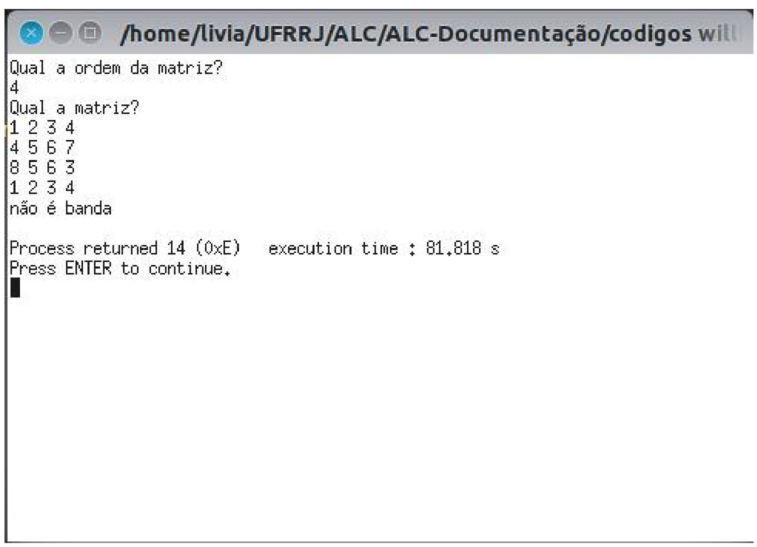


Figura 10: Exemplo.1 que mostra a execução do programa (quest07) no terminal

- TERMINAL-2

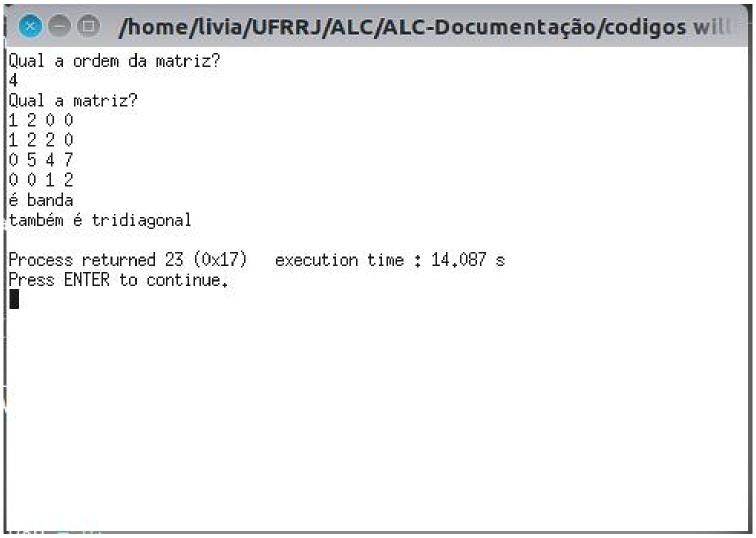


Figura 11: Exemplo.2 que mostra a execução do programa (quest07) no terminal

Exemplos Questão 8

- TERMINAL-1

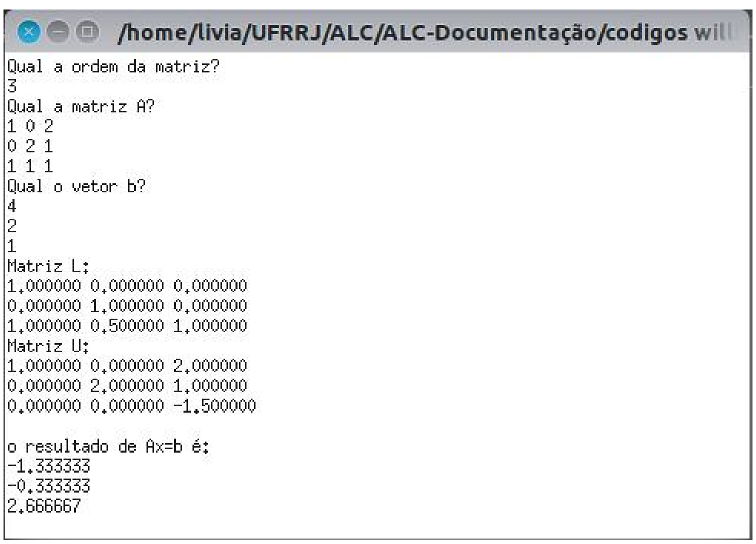
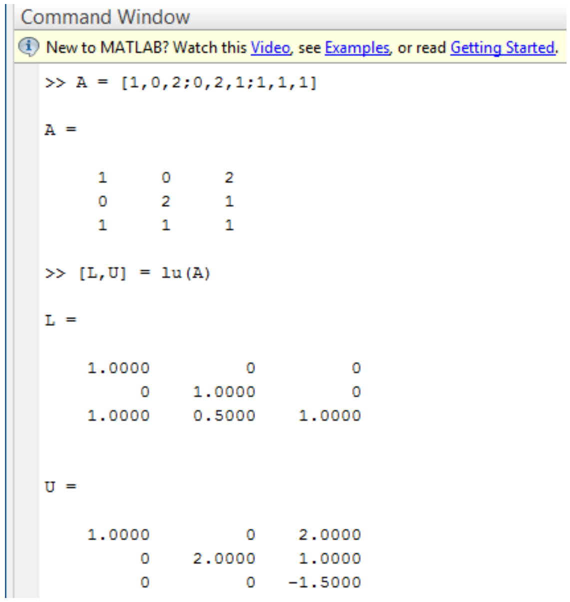


Figura 12: Exemplo.1 que mostra a execução do programa (quest08) no terminal

Exemplos Questão 8 continua na próxima página. . .

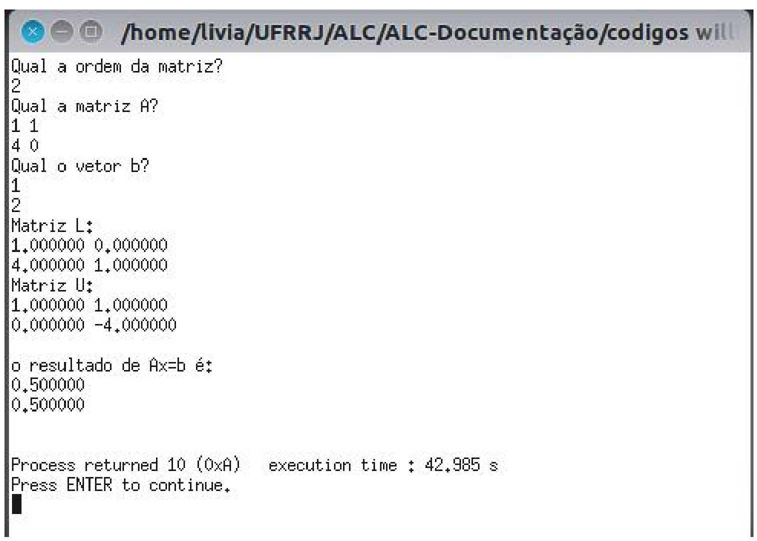
- MATLAB-1



```
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

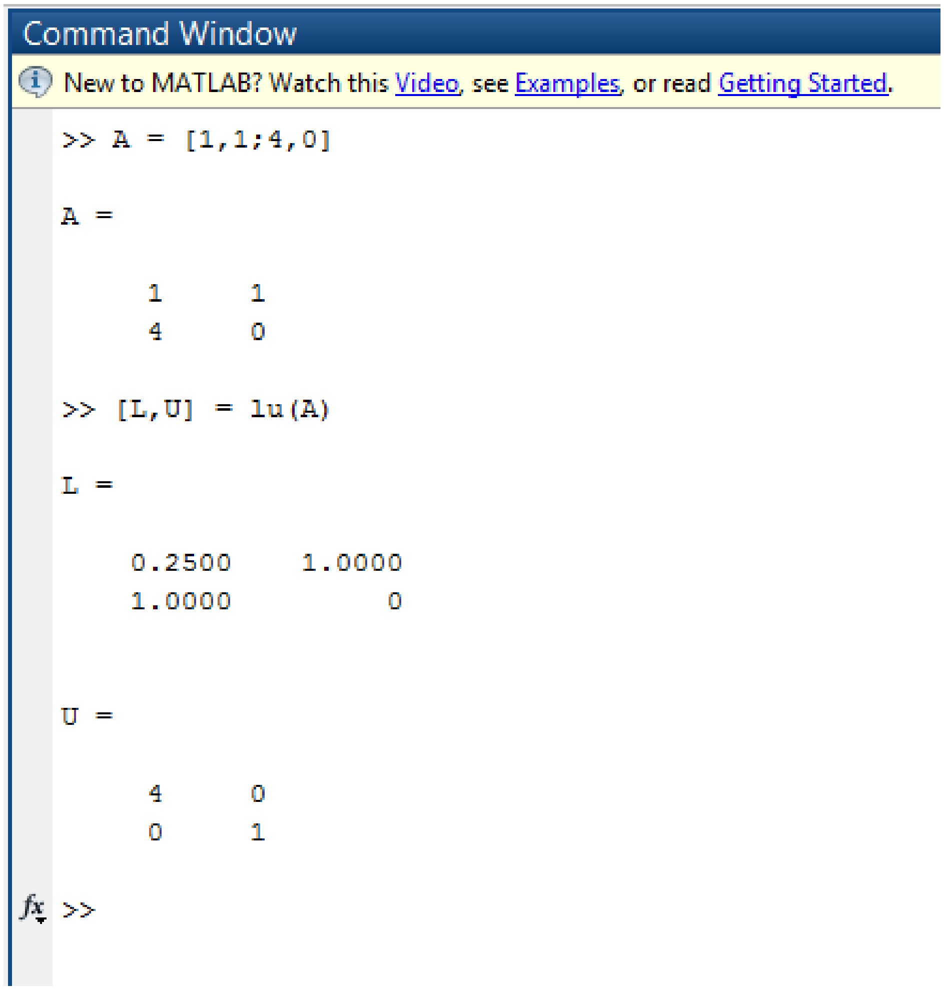


Figura 15: Exemplo que mostra o resultado do MATLAB (quest08) para o exemplo.2

Exemplos Questão 9

- TERMINAL

```
/home/livia/UFRRJ/ALC/ALC-Documentação/quest9
Digite a dimencao 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 dimencao 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 dimencao 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

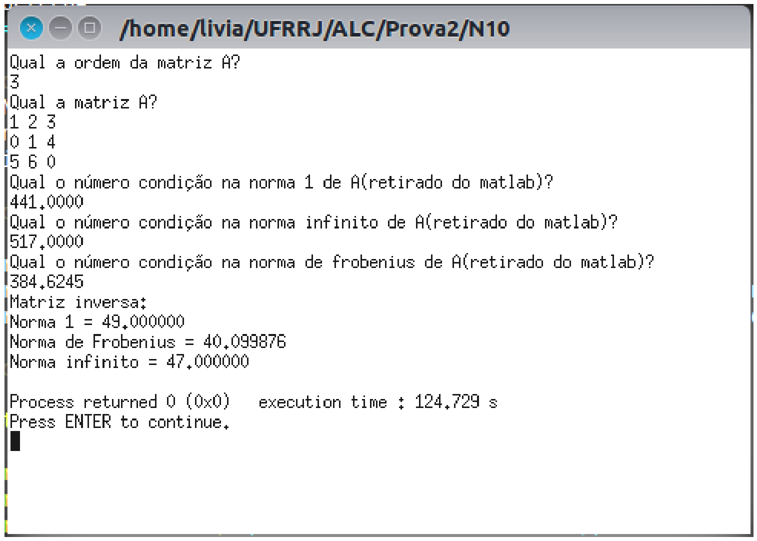


Figura 17: Exemplo.1 que mostra a execução do programa (quest10) no terminal

- MATLAB

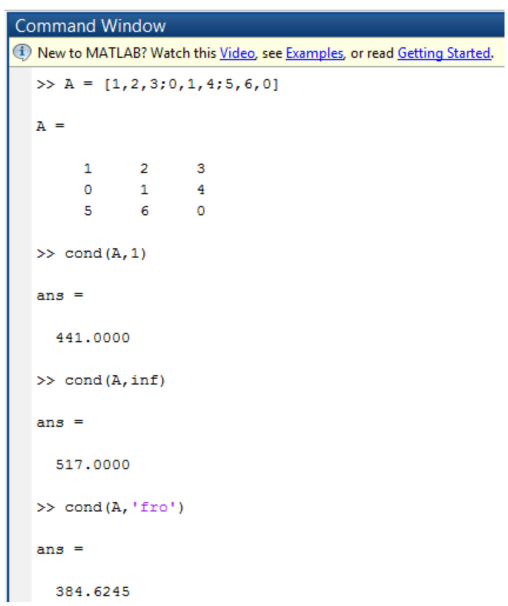


Figura 18: Exemplo que mostra o input do exemplo.1 (quest10) vindo do MATLAB

Exemplos Questão 11

- TERMINAL

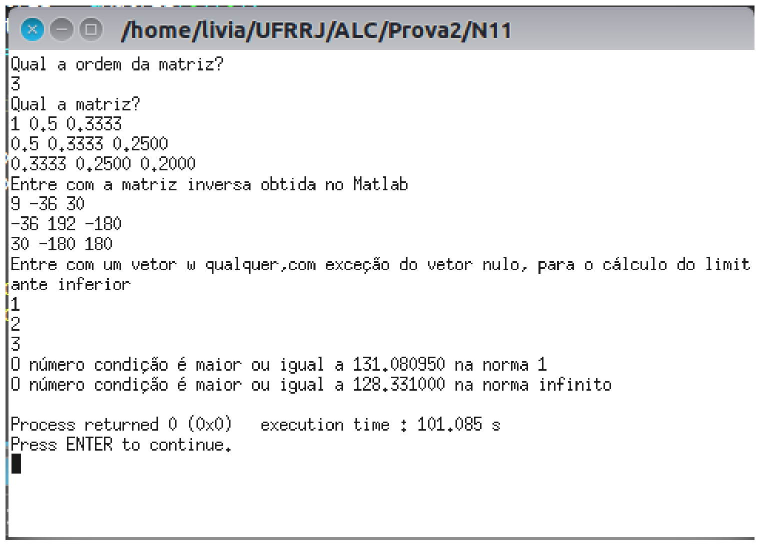


Figura 19: Exemplo.1 que mostra a execução do programa (quest11) no terminal

- MATLAB

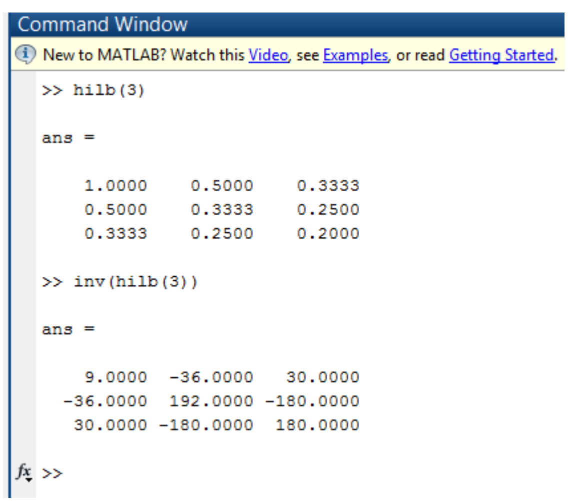
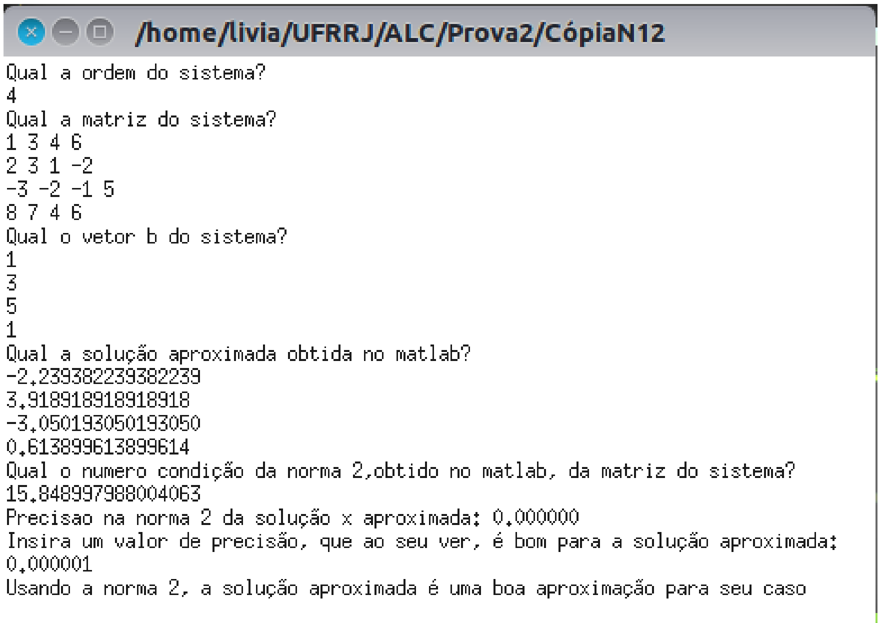


Figura 20: Exemplo que mostra o input do exemplo.1 (quest11) vindo do MATLAB

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

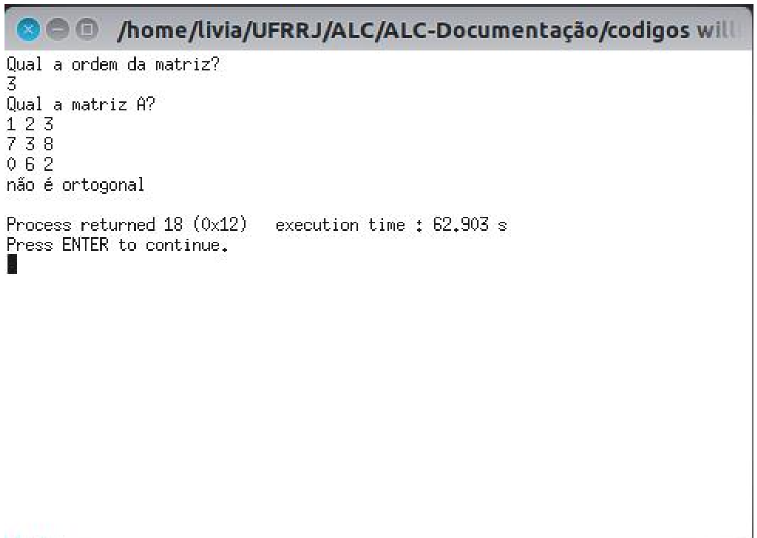


Figura 23: Exemplo.1 que mostra a execução do programa (quest13) no terminal

- TERMINAL-2

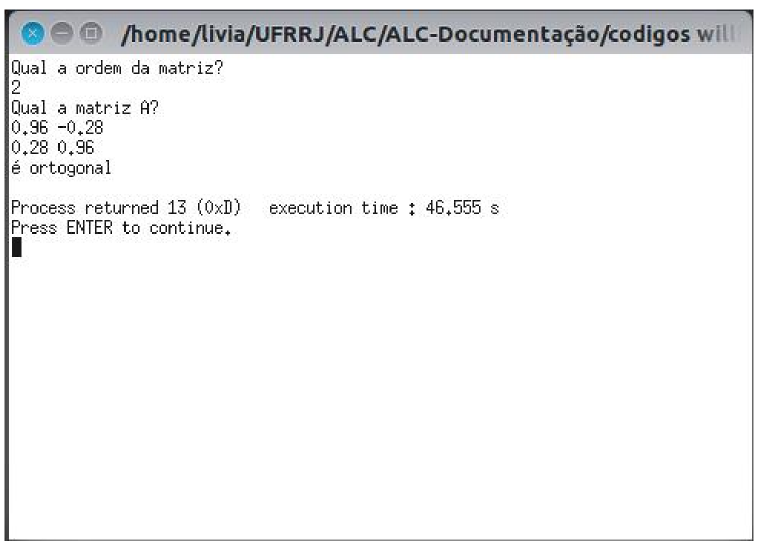
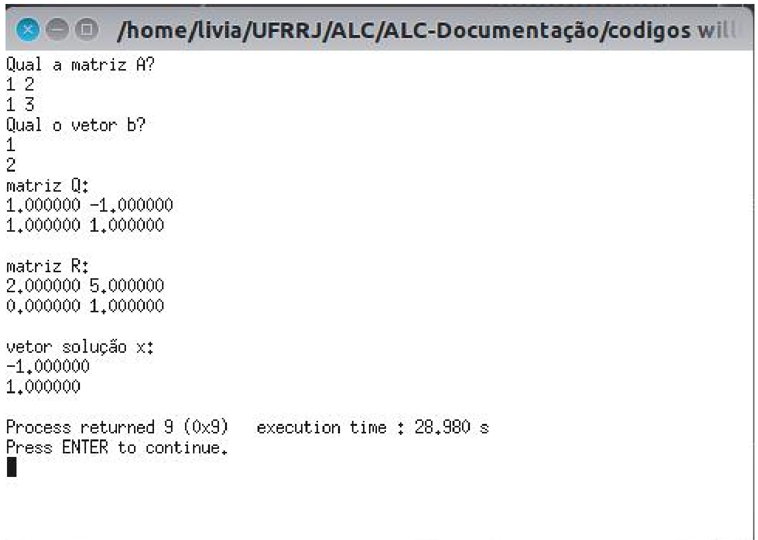


Figura 24: Exemplo.2 que mostra a execução do programa (quest13) no terminal

Exemplos Questão 14

- TERMINAL

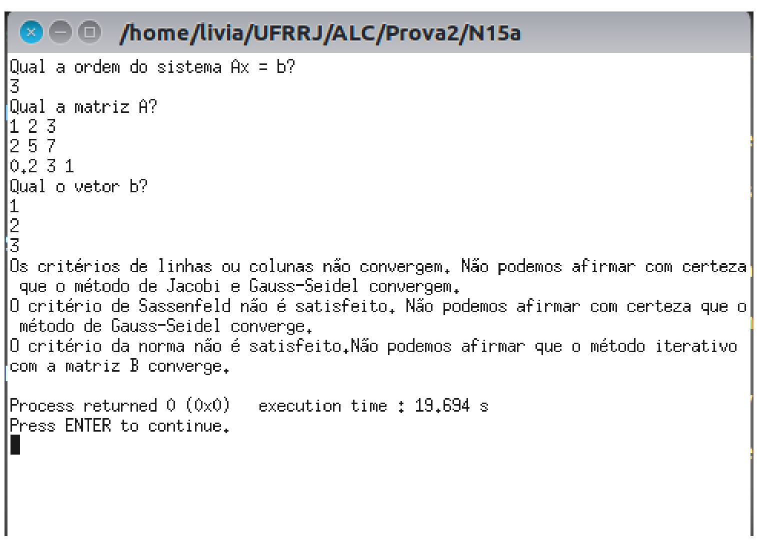


```
/home/livia/UFRRJ/ALC/ALC-Documentação/codigos will
Qual a matriz A?
1 2
1 3
Qual o vetor b?
1
2
matriz Q:
1.000000 -1.000000
1.000000 1.000000

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

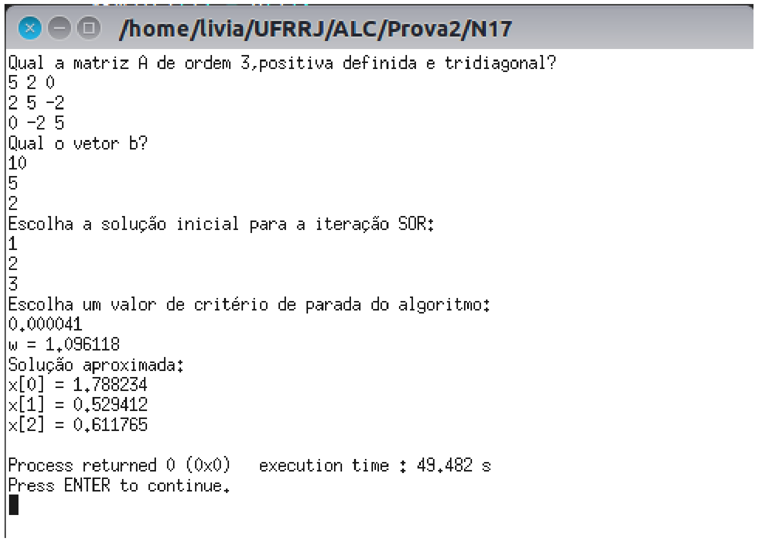


Figura 31: Exemplo.1 que mostra a execução do programa (quest17) no terminal

Exemplos Questão 18

- TERMINAL

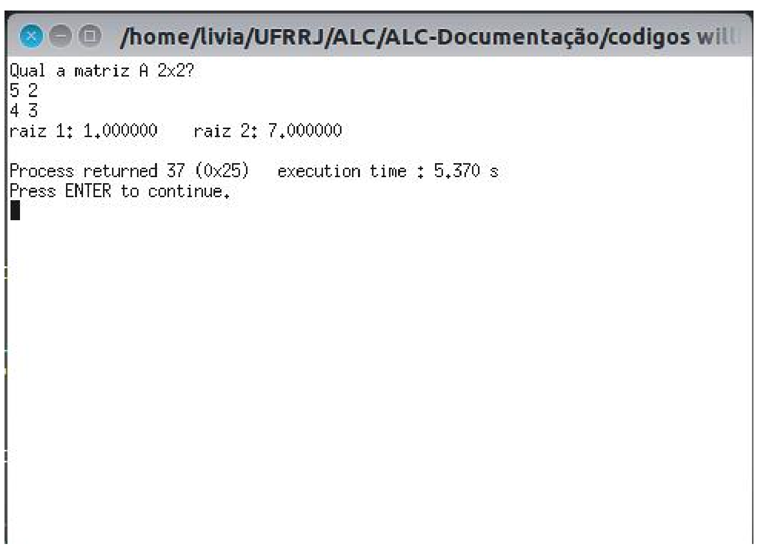
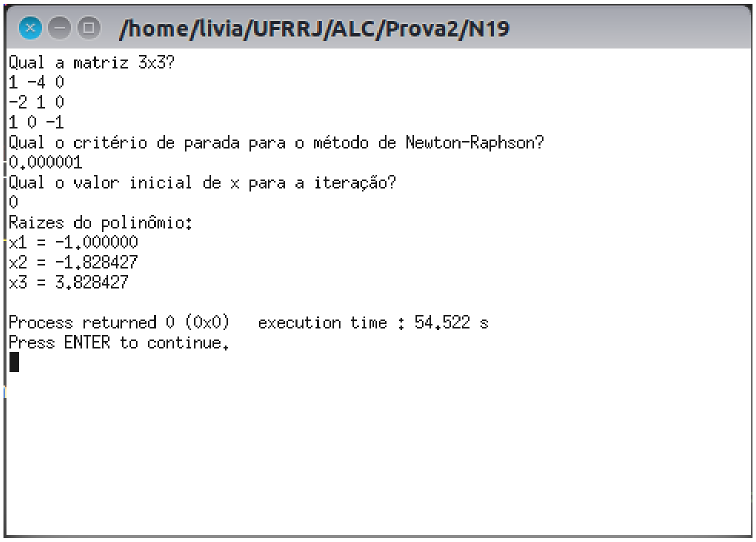


Figura 32: Exemplo.1 que mostra a execução do programa (quest18) no terminal

Exemplos Questão 19

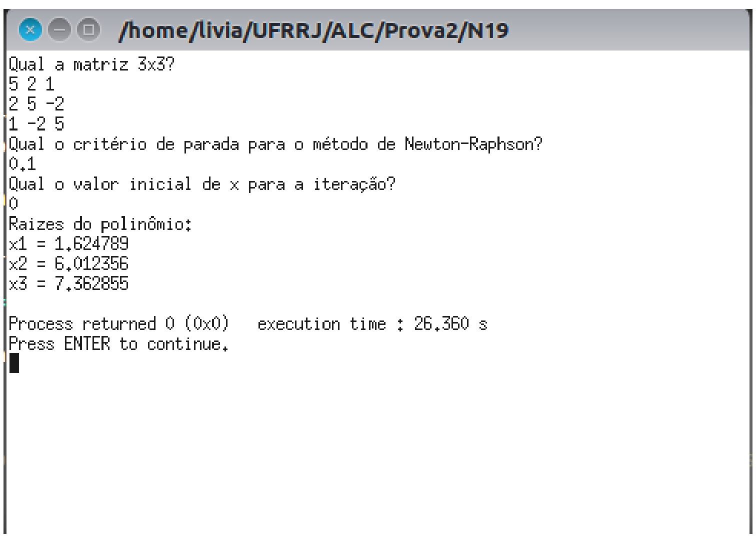
- TERMINAL



```
/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

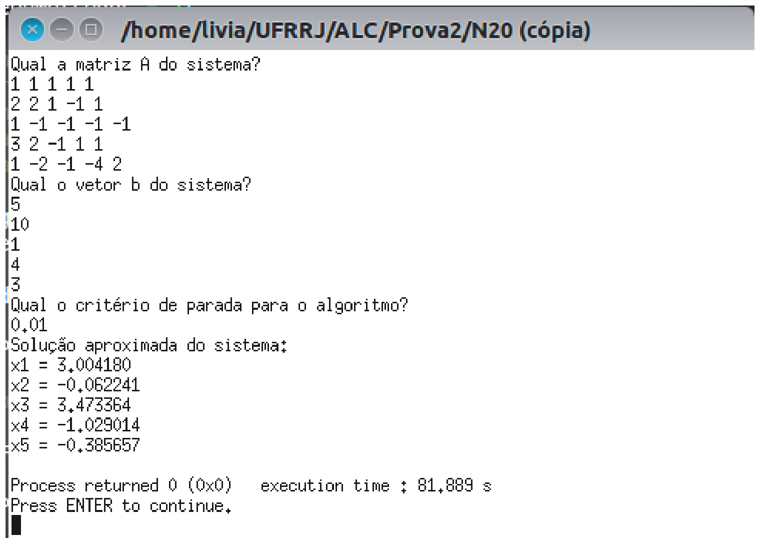


Figura 35: Exemplo.1 que mostra a execução do programa (quest20) no terminal