2° Trabalho de Calculo Numérico

Nomes: Christian Franchin dos Santos e Guilherme Locks Gregorio

GAUSS

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

double **alocarMatriz(int n);
double *alocarB(int n);
void lerMatriz(int n, double **p, double *B);
void imprimeMatriz(int n, double **p, double *B);
void pivotear(int n, double **p, double *B, int b, int a);
void zerarTriangInf(int n, double **p, double *B);
double *calculaResultado(int n, double **p, double *B);
void imprimeResultado(int n, double *r);
void desalocaMeB(int n, double **p, double *B);
void desalocaResultado(double *r);

void main()
{
	double **p, *B;

	double *resultado;
	int n;

	printf("Dimensão da matriz: ");
	scanf("%d", &n);

	p = alocarMatriz(n);
	B = alocarB(n);
	lerMatriz(n,p,B);

	imprimeMatriz(n,p,B);
	zerarTriangInf(n,p,B);
	imprimeMatriz(n,p,B);

	resultado = calculaResultado(n,p,B);
	imprimeResultado(n,resultado);

	desalocaMeB(n,p,B);
	desalocaResultado(resultado);
	return 0;
}

double **alocarMatriz(int n)
{
	int i,j;
	double **m = (double**) malloc(n * sizeof(double*)); // Aloca um vetor de ponteiros
	for (i = 0; i < n; i++)
		m[i] = (double*) malloc(n * sizeof(double));  // Aloca um vetor de valores double
	return m;
}


double *alocarB(int n)
{
	double *m = (double *) malloc(n * sizeof(double));
	return m;
}

void lerMatriz(int n, double **p, double *B)
{
	int i,j;
	printf("Valores da matriz M:\n");
	for(i = 0; i < n; i++)
	{
		for(j = 0; j < (n); j++)
		{
			scanf("%lf", &p[i][j]);
		}
	}
```

```
	printf("Valores da matriz B:\n");
	for(i = 0; i < n; i++)
	{
		scanf("%lf", &B[i]);
	}
}

void zerarTriangInf(int n, double **p, double *B)
{
	int a,b,c;
	double l;

	for(a = 0; a < (n-1); a++)
	{// Número de elementos de vezes a aplicar o algoritmo

		for(b = (a+1); b < n; b++)
		{
			pivotear(n,p,B,a,a);

			if(p[a][a] == 0)
			{
				printf("Divisão por 0 inesperada\n");
				break;
			}
			l = p[b][a] / p[a][a]; // Calcula o valor a ser aplicado as linhas
			for(c = a; c < n; c++)
			{// Aplica o valor calculado as linhas
				p[b][c] = p[b][c] - p[a][c]*l;
			}

			B[b] = B[b] - B[a]*l;//Aplica o valor calculado ao vetor B

		}
	}
}

void pivotear(int n, double **p, double *B, int b, int a)
{
	double maior;
	int i; //Contador para o for
	int j; //Posição do maior elemento
	double *q, aux;

	maior = fabs(p[b][a]);
	j = b;

	for(i = b; i < n; i++)
	{
		if(fabs(p[i][a]) > maior)
		{
			maior = fabs(p[i][a]);
			j = i;
		}
	}

	//Troca as linhas para evitar divisão por 0
	q = p[b];
	p[b] = p[j];
	p[j] = q;

	aux = B[b];
	B[b] = B[j];
	B[j] = aux;
}

double *calculaResultado(int n, double **p, double *B)
{
	double *result = malloc(n * sizeof(double));
	double sum;
	int a, b, count=(n-1);
```

```
	for(a = (n-1); a >= 0; a--)
	{
		sum = 0;

		for(b=count; b < n; b++)
		{
			sum += (p[a][b]*result[b]);
		}

		result[a] = (B[a]-sum)/p[a][a];
		count--;
	}

	return result;
}

void imprimeMatriz(int n, double **p, double *B)
{
	int i,j;

	for(i = 0; i < n; i++)
	{
		for(j = 0; j < n; j++)
		{
			printf("%+3.3lf\t", p[i][j]);
		}
		printf("|\t%+3.3lf\n", B[i]);
		putchar('\n');
	}

	putchar('\n');
}

void imprimeResultado(int n, double *r)
{
	int i;
	for(i = 0; i < n; i++)
	{
		printf("Resultado da variavel %2d: %+3.3lf\n", i+1, r[i]);
	}
}

void desalocaMeB(int n, double **p, double *B)
{
	int i,j;

	for(i = 0; i < n; i++)
	{
		free(p[i]);//Desaloca vetor de números
	}

	free(p);//Desaloca vetor de ponteiros

	free(B);//Desaloca matriz B
}

void desalocaResultado(double *r)
{
	free(r);
}
```

FATORAÇÃO LU

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

double **alocarMatriz(int n);
double *alocarB(int n);
void lerMatriz(int n, double **p, double *B);
void imprimeMatriz(int n, double **p);
void imprimeMatrizB(int n, double **p, double *B);
void pivotearB(int n, double **p, double *B, int b, int a);
void zerarTriangInf(int n, double **p, double **l, double *B);
double *calculaResultadoL(int n, double **p, double *B);
double *calculaResultadoU(int n, double **p, double *B);
void imprimeResultado(int n, double *r);
void desalocaMeB(int n, double **p, double **l, double *B);
void desalocaResultado(double *r);

void main()
{
	double **p;//Resultara em U
	double **l;//Resultara em L
	double *B;

	double *y;
	double *resultado;
	int n;

	printf("Dimensão da matriz: ");
	scanf("%d", &n);

	p = alocarMatriz(n);
	l = alocarMatriz(n);
	B = alocarB(n);

	lerMatriz(n,p,B);

	putchar('\n');
	printf("Matriz Mx = B\n");
	imprimeMatrizB(n,p,B);
	zerarTriangInf(n,p,l,B);

	printf("MATRIZ U:\n");
	imprimeMatriz(n,p);
	printf("MATRIZ L:\n");
	imprimeMatriz(n,l);

	imprimeMatrizB(n,l,B);
	y = calculaResultadoL(n,l,B);
	imprimeMatrizB(n,p,y);
	resultado = calculaResultadoU(n,p,y);
	imprimeResultado(n,resultado);

	desalocaMeB(n,p,l,B);
	desalocaResultado(y);
	desalocaResultado(resultado);
	return 0;
}

double **alocarMatriz(int n)
{
	int i,j;

	double **m = (double**) malloc(n * sizeof(double*)); // Aloca um vetor de ponteiros

	for (i = 0; i < n; i++)
		m[i] = (double*) malloc(n * sizeof(double));  // Aloca um vetor de valores double
```

```
	return m;
}

double *alocarB(int n)
{
	double *m = (double *) malloc(n * sizeof(double));
	return m;
}

void lerMatriz(int n, double **p, double *B)
{
	int i,j;

	printf("Valores da matriz M:\n");
	for(i = 0; i < n; i++)
	{
		for(j = 0; j < (n); j++)
		{
			scanf("%lf", &p[i][j]);
		}
	}

	printf("Valores da matriz B:\n");
	for(i = 0; i < n; i++)
	{
		scanf("%lf", &B[i]);
	}

}

void zerarTriangInf(int n, double **p, double **l, double *B)
{
	int a,b,c;
	double x;

	for(a = 0; a < n; a++)
	{//Zera a matriz L e define sua diagonal principal como 1
		for(b = 0; b < n; b++)
		{
			if(a==b)
			{
				l[a][b] = 1;
			}
			else
				l[a][b] = 0;
		}
	}

	for(a = 0; a < (n-1); a++)
	{// Número de elementos de vezes a aplicar o algoritmo

		for(b = (a+1); b < n; b++)
		{

			pivotearB(n,p,B,a,a);

			if(p[a][a] == 0)
			{
				printf("Divisão por 0 inesperada\n");
				break;
			}

			x = p[b][a] / p[a][a]; // Calcula o valor a ser aplicado as linhas

			l[b][a] = x;

			for(c = a; c < n; c++)
			{// Aplica o valor calculado as linhas
				p[b][c] = p[b][c] - p[a][c]*x;
			}
```

```
                }
        }
}

void pivotearB(int n, double **p, double *B, int b, int a)
{
        double maior;
        int i; //Contador para o for
        int j; //Posição do maior elemento
        double *q, aux;

        maior = fabs(p[b][a]);
        j = b;

        for(i = b; i < n; i++)
        {
                if(fabs(p[i][a]) > maior)
                {
                        maior = fabs(p[i][a]);
                        j = i;
                }
        }

        //Troca as linhas para evitar divisão por 0
        q = p[b];
        p[b] = p[j];
        p[j] = q;

        aux = B[b];
        B[b] = B[j];
        B[j] = aux;
}

double *calculaResultadoU(int n, double **p, double *B)
{
        double *result = malloc(n * sizeof(double));
        double sum;
        int a, b, count=(n-1);

        for(a = (n-1); a >= 0; a--)
        {
                sum = 0;

                pivotearB(n,p,B,a,a);

                for(b=count; b < n; b++)
                {
                        sum += (p[a][b]*result[b]);
                }

                result[a] = (B[a]-sum)/p[a][a];
                count--;
        }

        return result;
}

double *calculaResultadoL(int n, double **l, double *B)
{
        double *result = malloc(n * sizeof(double));
        double sum;
        int a, b, count=(n-1);

        for(a = 0; a < n; a++)
        {
                sum = 0;

                for(b=0; b < a; b++)
                {
```

```
                        sum += (l[a][b]*result[b]);
                }

                result[a] = (B[a]-sum)/l[a][a];
                count--;
        }

        return result;
}

void imprimeMatrizB(int n, double **p, double *B)
{
        int i,j;

        for(i = 0; i < n; i++)
        {
                for(j = 0; j < n; j++)
                {
                        printf("%+3.3lf\t", p[i][j]);
                }
                printf("|\t%+3.3lf", B[i]);
                putchar('\n');
        }

        putchar('\n');
}

void imprimeMatriz(int n, double **p)
{
        int i,j;

        for(i = 0; i < n; i++)
        {
                for(j = 0; j < n; j++)
                {
                        printf("%+3.3lf\t", p[i][j]);
                }
                putchar('\n');
        }

        putchar('\n');
}


void imprimeResultado(int n, double *r)
{
        int i;
        for(i = 0; i < n; i++)
        {
                printf("Resultado da variavel %2d: %+3.3lf\n", i+1, r[i]);
        }
}

void desalocaMeB(int n, double **p, double **l, double *B)
{
        int i,j;
        for(i = 0; i < n; i++)
        {
                free(p[i]);//Desaloca vetor de números
                free(l[i]);
        }
        free(p);//Desaloca vetor de ponteiros
        free(l);
        free(B);//Desaloca matriz B
}

void desalocaResultado(double *r)
{
        free(r);
}
```

FATORAÇÃO CHOLESKY

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

double **alocarMatriz(int n);
double *alocarB(int n);
void lerMatriz(int n, double **p, double *B);
void imprimeMatriz(int n, double **p);
void imprimeMatrizB(int n, double **p, double *B);
void pivotearB(int n, double **p, double *B, int b, int a);
void zerarTriangInf(int n, double **p, double **l, double *B);
void geraD(int n, double **p);
void geraDr(int n, double **p);
double  **matrizTransposta(int n, double **p);
double **multiplicaMatriz(int n, double **d, double **l);
double *calculaResultadoL(int n, double **p, double *B);
double *calculaResultadoU(int n, double **p, double *B);
void imprimeResultado(int n, double *r);
void desalocaM(int n, double **p);
void desalocaResultado(double *r);

void main()
{
	double **p;//Resultara em U e futuramente D e D~
	double **l;//Resultara em L
	double **G, **GT;//Matriz G e G Tranposta
	double *B;

	double *y;
	double *resultado;
	int n;

	printf("Dimensão da matriz: ");
	scanf("%d", &n);

	p = alocarMatriz(n);
	l = alocarMatriz(n);
	B = alocarB(n);

	lerMatriz(n,p,B);

	putchar('\n');
	printf("Matriz Mx = B\n");
	imprimeMatrizB(n,p,B);
	zerarTriangInf(n,p,l,B);

	printf("MATRIZ U:\n");
	imprimeMatriz(n,p);
	printf("MATRIZ L:\n");
	imprimeMatriz(n,l);

	geraD(n,p);

	printf("MATRIZ D:\n");
	imprimeMatriz(n,p);

	geraDr(n,p);
	printf("MATRIZ Dr:\n");
	imprimeMatriz(n,p);

	G = multiplicaMatriz(n,p,l);
	printf("MATRIZ G:\n");
	imprimeMatriz(n,G);

	GT = matrizTransposta(n,G);
	printf("MATRIZ G Tranposta:\n");
	imprimeMatriz(n,GT);
```

```
	y = calculaResultadoL(n,G,B);
	resultado = calculaResultadoU(n,GT,y);

	imprimeResultado(n,resultado);

	desalocaM(n,p);
	desalocaM(n,l);
	desalocaM(n,G);
	desalocaM(n,GT);
	desalocaResultado(B);
	desalocaResultado(y);
	desalocaResultado(resultado);
	return 0;
}

double **alocarMatriz(int n)
{
	int i,j;

	double **m = (double**) malloc(n * sizeof(double*)); // Aloca um vetor de ponteiros

	for (i = 0; i < n; i++)
		m[i] = (double*) malloc(n * sizeof(double));  // Aloca um vetor de valores double

	return m;
}

double *alocarB(int n)
{
	double *m = (double *) malloc(n * sizeof(double));
	return m;
}

void lerMatriz(int n, double **p, double *B)
{
	int i,j;

	printf("Valores da matriz M:\n");
	for(i = 0; i < n; i++)
	{
		for(j = 0; j < (n); j++)
		{
			scanf("%lf", &p[i][j]);
		}
	}

	printf("Valores da matriz B:\n");
	for(i = 0; i < n; i++)
	{
		scanf("%lf", &B[i]);
	}

}

void zerarTriangInf(int n, double **p, double **l, double *B)
{
	int a,b,c;
	double x;

	for(a = 0; a < n; a++)
	{//Zera a matriz L e define sua diagonal principal como 1
		for(b = 0; b < n; b++)
		{
			if(a==b)
			{
				l[a][b] = 1;
			}
			else
				l[a][b] = 0;
		}
```

```
	}

	for(a = 0; a < (n-1); a++)
	{// Número de elementos de vezes a aplicar o algoritmo

		for(b = (a+1); b < n; b++)
		{

			pivotearB(n,p,B,a,a);

			if(p[a][a] == 0)
			{
				printf("Divisão por 0 inesperada\n");
				break;
			}

			x = p[b][a] / p[a][a]; // Calcula o valor a ser aplicado as linhas

			l[b][a] = x;

			for(c = a; c < n; c++)
			{// Aplica o valor calculado as linhas
				p[b][c] = p[b][c] - p[a][c]*x;
			}

		}
	}
}

void pivotearB(int n, double **p, double *B, int b, int a)
{
	double maior;
	int i; //Contador para o for
	int j; //Posição do maior elemento
	double *q, aux;

	maior = fabs(p[b][a]);
	j = b;

	for(i = b; i < n; i++)
	{
		if(fabs(p[i][a]) > maior)
		{
			maior = fabs(p[i][a]);
			j = i;
		}
	}

	//Troca as linhas para evitar divisão por 0
	q = p[b];
	p[b] = p[j];
	p[j] = q;

	aux = B[b];
	B[b] = B[j];
	B[j] = aux;
}

void geraD(int n, double **p)
{
	int i, j;
	for(i = 0; i < n; i++)
	{
		for(j = 0; j < n; j++)
		{
			if(i!=j)
				p[i][j]=0;
		}
	}
}
```

```
void geraDr(int n, double **p)
{
	int i;
	for(i = 0; i < n; i++)
	{
		if(p[i][i] < 0)
		{
			printf("VALORES IMAGINARIOS NÃO SÃO ACEITOS\n");
			break;
		}
		p[i][i] = sqrt(p[i][i]);
	}
}

double **multiplicaMatriz(int n, double **d, double **l)
{
	int i, j;
	double sum;
	double **m = alocarMatriz(n);

	for(i = 0; i < n; i++)
	{//Matriz M recebe matriz l
		for(j = 0; j < n; j++)
		{
			m[i][j] = l[i][j];
		}
	}

	for(i = 0; i < n; i++)
	{//Multiplica cada coluna da matriz M pelos valores da matriz D
		for(j = 0; j < n; j++)
		{
			m[j][i] *= d[i][i];
		}
	}

	return m;
}

double  **matrizTransposta(int n, double **p)
{
	int i, j;

	double **m = alocarMatriz(n);

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

	return m;
}

double *calculaResultadoU(int n, double **p, double *B)
{
	double *result = malloc(n * sizeof(double));
	double sum;
	int a, b, count=(n-1);

	for(a = (n-1); a >= 0; a--)
	{
		sum = 0;

		pivotearB(n,p,B,a,a);

		for(b=count; b < n; b++)
```

```
		{
			sum += (p[a][b]*result[b]);
		}

		result[a] = (B[a]-sum)/p[a][a];
		count--;
	}

	return result;
}

double *calculaResultadoL(int n, double **l, double *B)
{
	double *result = malloc(n * sizeof(double));
	double sum;
	int a, b, count=(n-1);

	for(a = 0; a < n; a++)
	{
		sum = 0;

		for(b=0; b < a; b++)
		{
			sum += (l[a][b]*result[b]);
		}

		result[a] = (B[a]-sum)/l[a][a];
		count--;
	}

	return result;
}

void imprimeMatrizB(int n, double **p, double *B)
{
	int i,j;

	for(i = 0; i < n; i++)
	{
		for(j = 0; j < n; j++)
		{
			printf("%+3.3lf\t", p[i][j]);
		}
		printf("|\t%+3.3lf", B[i]);
		putchar('\n');
	}

	putchar('\n');
}

void imprimeMatriz(int n, double **p)
{
	int i,j;

	for(i = 0; i < n; i++)
	{
		for(j = 0; j < n; j++)
		{
			printf("%+3.3lf\t", p[i][j]);
		}
		putchar('\n');
	}

	putchar('\n');
}


void imprimeResultado(int n, double *r)
{
	int i;
```

```
	for(i = 0; i < n; i++)
	{
		printf("Resultado da variavel %2d: %+3.3lf\n", i+1, r[i]);
	}
}

void desalocaM(int n, double **p)
{
	int i;

	for(i = 0; i < n; i++)
	{
		free(p[i]);//Desaloca vetor de números
	}

	free(p);//Desaloca vetor de ponteiros
}

void desalocaResultado(double *r)
{
	free(r);
}
```

GAUSS-JACOBI

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

double **alocarMatriz(int n);
double *alocarV(int n);
void lerMatriz(int n, double **p, double *B);
void lerValoresIniciais(int n, double *k);
void Calcular(int n, double **m, double *b, double *kant, double *k);
void imprimeCabecalho(int n);
void imprimeIteracao(int n, double *k, double *kant, int i);
double maiorParada(int n, double *k, double *kant);
void imprimeMatriz(int n, double **p);
void imprimeMatrizB(int n, double **p, double *B);
void desalocaM(int n, double **p);
void desalocaV(double *r);

void main()
{
	double **m, *b, *kant, *k;
	int n;

	printf("Dimensão da matriz: ");
	scanf("%d", &n);

	m = alocarMatriz(n);
	b = alocarV(n);
	kant = alocarV(n);
	k = alocarV(n);

	lerMatriz(n,m,b);

	imprimeMatrizB(n,m,b);

	lerValoresIniciais(n,kant);

	Calcular(n,m,b,kant,k);

	desalocaM(n,m);
	desalocaV(b);
	desalocaV(kant);
	desalocaV(k);
}

double **alocarMatriz(int n)
{
	int i,j;

	double **m = (double**) malloc(n * sizeof(double*)); // Aloca um vetor de ponteiros

	for (i = 0; i < n; i++)
		m[i] = (double*) malloc(n * sizeof(double));  // Aloca um vetor de valores double

	return m;
}

double *alocarV(int n)
{
	double *m = (double *) malloc(n * sizeof(double));
	return m;
}

void lerMatriz(int n, double **p, double *B)
{
	int i,j;

	printf("Valores da matriz M:\n");
	for(i = 0; i < n; i++)
```

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

	printf("Valores da matriz B:\n");
	for(i = 0; i < n; i++)
	{
		scanf("%lf", &B[i]);
	}
}

void Calcular(int n, double **m, double *b, double *kant, double *k)
{
	int i, count = 0;
	int j;
	double e;
	double sum;

	printf("Critério de parada: ");
	scanf("%lf", &e);

	imprimeCabecalho(n);
	imprimeIteracao(n,kant,kant, count++);

	//Primeira iteração
	for(i = 0; i < n; i++)
	{
		sum = 0;
		for(j = 0; j < n; j++)
		{
			if(i!=j)
				sum += m[i][j]*kant[j];
		}
		k[i] = (b[i] - (sum)) / (m[i][i]);
	}

	imprimeIteracao(n,k,kant, count++);

	while(maiorParada(n,k,kant) >  e)
	{

		for(i = 0; i < n; i++)
		{//Atualiza os valores de kant
			kant[i] = k[i];
		}

		for(i = 0; i < n; i++)
		{//Calcula as iterações e guarda os resultados em k
			sum = 0;
			for(j = 0; j < n; j++)
			{
				if(i!=j)
				{
					sum += m[i][j]*kant[j];
				}
			}
			k[i] = (b[i] - (sum)) / (m[i][i]);
		}

		imprimeIteracao(n,k,kant,count++);

	}

}

void imprimeCabecalho(int n)
{
```

```
	int i;

	printf("MÉTODO DE GAUSS-JACOBI\n");

	printf("K\t|");

	for(i = 1; i <= n; i++)
	{
		printf("x%02d\t|", i);
	}

	printf(" max|k - kant|\n");
}

void imprimeIteracao(int n, double *k, double *kant, int i)
{
	int j;

	printf("%02d\t|", i);

	for(j = 0; j < n; j++)
	{
		printf("%2.3lf\t|", k[j]);
	}

	printf("   %2.4lf\n", maiorParada(n, k, kant));

}

double maiorParada(int n, double *k, double *kant)
{
	int i;
	double maior = fabs(k[0] - kant[0]);

	for(i = 0; i < n; i++)
	{
		if(fabs(k[i]-kant[i]) > maior)
			maior = fabs(k[i]-kant[i]);
	}

	return maior;
}

void imprimeMatrizB(int n, double **p, double *B)
{
	int i,j;

	for(i = 0; i < n; i++)
	{
		for(j = 0; j < n; j++)
		{
			printf("%+3.3lf\t", p[i][j]);
		}
		printf("|\t%+3.3lf", B[i]);
		putchar('\n');
	}

	putchar('\n');
}

void lerValoresIniciais(int n, double *k)
{
	int i;
	printf("Valores Iniciais:\n");
	for(i = 0; i < n; i++)
	{
		printf("x1: ");
		scanf("%lf", &k[i]);
	}
}
```

```
void desalocaM(int n, double **p)
{
	int i;

	for(i = 0; i < n; i++)
	{
		free(p[i]);//Desaloca vetor de números
	}

	free(p);//Desaloca vetor de ponteiros
}

void desalocaV(double *r)
{
	free(r);
}
```

GAUSS-SEIDEL

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

double **alocarMatriz(int n);
double *alocarV(int n);
void lerMatriz(int n, double **p, double *B);
void lerValoresIniciais(int n, double *k);
void Calcular(int n, double **m, double *b, double *kant, double *k);
void imprimeCabecalho(int n);
void imprimeIteracao(int n, double *k, double *kant, int i);
double maiorParada(int n, double *k, double *kant);
void imprimeMatriz(int n, double **p);
void imprimeMatrizB(int n, double **p, double *B);
void desalocaM(int n, double **p);
void desalocaV(double *r);

void main()
{
	double **m, *b, *kant, *k;
	int n;

	printf("Dimensão da matriz: ");
	scanf("%d", &n);

	m = alocarMatriz(n);
	b = alocarV(n);
	kant = alocarV(n);
	k = alocarV(n);

	lerMatriz(n,m,b);

	imprimeMatrizB(n,m,b);

	lerValoresIniciais(n,kant);

	Calcular(n,m,b,kant,k);

	desalocaM(n,m);
	desalocaV(b);
	desalocaV(kant);
	desalocaV(k);
}

double **alocarMatriz(int n)
{
	int i,j;

	double **m = (double**) malloc(n * sizeof(double*)); // Aloca um vetor de ponteiros

	for (i = 0; i < n; i++)
		m[i] = (double*) malloc(n * sizeof(double));  // Aloca um vetor de valores double

	return m;
}

double *alocarV(int n)
{
	double *m = (double *) malloc(n * sizeof(double));
	return m;
}

void lerMatriz(int n, double **p, double *B)
{
	int i,j;

	printf("Valores da matriz M:\n");
	for(i = 0; i < n; i++)
```

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

	printf("Valores da matriz B:\n");
	for(i = 0; i < n; i++)
	{
		scanf("%lf", &B[i]);
	}
}

void Calcular(int n, double **m, double *b, double *kant, double *k)
{
	int i, count = 0;
	int j;
	double e;
	double sum;

	printf("Critério de parada: ");
	scanf("%lf", &e);

	imprimeCabecalho(n);
	imprimeIteracao(n,kant,kant, count++);

	//Primeira iteração
	for(i = 0; i < n; i++)
	{
		sum = 0;
		for(j = 0; j < n; j++)
		{
			if(i!=j)
				sum += m[i][j]*k[j];
		}
		k[i] = (b[i] - (sum)) / (m[i][i]);
	}

	imprimeIteracao(n,k,kant, count++);

	while(maiorParada(n,k,kant) >  e)
	{

		for(i = 0; i < n; i++)
		{//Atualiza os valores de kant
			kant[i] = k[i];
		}

		for(i = 0; i < n; i++)
		{//Calcula as iterações e guarda os resultados em k
			sum = 0;
			for(j = 0; j < n; j++)
			{
				if(i!=j)
				{
					sum += m[i][j]*k[j];
				}
			}
			k[i] = (b[i] - (sum)) / (m[i][i]);
		}

		imprimeIteracao(n,k,kant,count++);

	}

}

void imprimeCabecalho(int n)
{
```

```
	int i;

	printf("MÉTODO DE GAUSS-SEIDEL\n");

	printf("K\t|");

	for(i = 1; i <= n; i++)
	{
		printf("x%02d\t|", i);
	}

	printf(" max|k - kant|\n");
}

void imprimeIteracao(int n, double *k, double *kant, int i)
{
	int j;

	printf("%02d\t|", i);

	for(j = 0; j < n; j++)
	{
		printf("%2.3lf\t|", k[j]);
	}

	printf("   %2.4lf\n", maiorParada(n, k, kant));

}

double maiorParada(int n, double *k, double *kant)
{
	int i;
	double maior = fabs(k[0] - kant[0]);

	for(i = 0; i < n; i++)
	{
		if(fabs(k[i]-kant[i]) > maior)
			maior = fabs(k[i]-kant[i]);
	}

	return maior;
}

void imprimeMatrizB(int n, double **p, double *B)
{
	int i,j;

	for(i = 0; i < n; i++)
	{
		for(j = 0; j < n; j++)
		{
			printf("%+3.3lf\t", p[i][j]);
		}
		printf("|\t%+3.3lf", B[i]);
		putchar('\n');
	}

	putchar('\n');
}

void lerValoresIniciais(int n, double *k)
{
	int i;
	printf("Valores Iniciais:\n");
	for(i = 0; i < n; i++)
	{
		printf("x1: ");
		scanf("%lf", &k[i]);
	}
}
```

```
void desalocaM(int n, double **p)
{
	int i;

	for(i = 0; i < n; i++)
	{
		free(p[i]);//Desaloca vetor de números
	}

	free(p);//Desaloca vetor de ponteiros
}

void desalocaV(double *r)
{
	free(r);
}
```

Saida obtida para Gauss:

+1.000 +1.000 +1.000 | +1.000

+2.000 -1.000 +3.000 | +0.000

-1.000 +1.000 -5.000 | +2.000

+2.000 -1.000 +3.000 | +0.000

+0.000 +1.500 -0.500 | +1.000

+0.000 +0.000 -3.333 | +1.667

Resultado da variavel 1: +1.000
Resultado da variavel 2: +0.500
Resultado da variavel 3: -0.500

Saida obtida para Fatoração LU:

Matriz Mx = B

+1.000 +1.000 +1.000 | +1.000
+2.000 -1.000 +3.000 | +0.000
-1.000 +1.000 -5.000 | +2.000

MATRIZ U:

+2.000 -1.000 +3.000
+0.000 +1.500 -0.500
+0.000 +0.000 -3.333

MATRIZ L:

+1.000 +0.000 +0.000
+0.500 +1.000 +0.000
-0.500 +0.333 +1.000

+1.000 +0.000 +0.000 | +0.000
+0.500 +1.000 +0.000 | +1.000
-0.500 +0.333 +1.000 | +2.000

+2.000 -1.000 +3.000 | +0.000
+0.000 +1.500 -0.500 | +1.000
+0.000 +0.000 -3.333 | +1.667

Resultado da variavel 1: +1.000
Resultado da variavel 2: +0.500
Resultado da variavel 3: -0.500

Saida obtida para Fatoração Cholesky:

Matriz Mx = B

+4.000	+2.000	-4.000 |	+0.000
+2.000	+10.000	+4.000|	+6.000
-4.000	+4.000	+9.000|	+5.000

MATRIZ U:

+4.000	+2.000	-4.000
+0.000	+9.000	+6.000
+0.000	+0.000	+1.000

MATRIZ L:

+1.000	+0.000	+0.000
+0.500	+1.000	+0.000
-1.000	+0.667	+1.000

MATRIZ D:

+4.000	+0.000	+0.000
+0.000	+9.000	+0.000
+0.000	+0.000	+1.000

MATRIZ Dr:

+2.000	+0.000	+0.000
+0.000	+3.000	+0.000
+0.000	+0.000	+1.000

MATRIZ G:

+2.000	+0.000	+0.000
+1.000	+3.000	+0.000
-2.000	+2.000	+1.000

MATRIZ G Tranposta:

+2.000	+1.000	-2.000
+0.000	+3.000	+2.000
+0.000	+0.000	+1.000

Resultado da variavel 1: +1.000
Resultado da variavel 2: +0.000
Resultado da variavel 3: +1.000

Saida obtida para o Método de Gauss-Jacobi:

```
+10.000        +2.000        +1.000|        +7.000
+1.000         +5.000        +1.000|        -8.000
+2.000         +3.000        +10.00|        +6.000
```

MÉTODO DE GAUSS-JACOBI

```
K       |x01    |x02    |x03    | max|k - kant|
00      |0.000  |0.000  |0.000  |   0.0000
01      |0.700  |-1.600 |0.600  |   1.6000
02      |0.960  |-1.860 |0.940  |   0.3400
03      |0.978  |-1.980 |0.966  |   0.1200
04      |0.999  |-1.989 |0.998  |   0.0324
05      |0.998  |-2.000 |0.997  |   0.0108
06      |1.000  |-1.999 |1.000  |   0.0035
07      |1.000  |-2.000 |1.000  |   0.0012
08      |1.000  |-2.000 |1.000  |   0.0004
```

Saida obtida para o Método de Gauss-Jacobi:

```
+10.000        +2.000        +1.000|        +7.000
+1.000         +5.000        +1.000|        -8.000
+2.000         +3.000        +10.00|        +6.000
```

MÉTODO DE GAUSS-SEIDEL

```
K       |x01    |x02    |x03    | max|k - kant|
00      |0.000  |0.000  |0.000  |   0.0000
01      |0.700  |-1.740 |0.982  |   1.7400
02      |0.950  |-1.986 |1.006  |   0.2498
03      |0.997  |-2.001 |1.001  |   0.0469
04      |1.000  |-2.000 |1.000  |   0.0033
05      |1.000  |-2.000 |1.000  |   0.0002
```