Trabalho de Calculo Numérico

Nomes: Christian Franchin e Guilherme Locks

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

//OBS: PARA EVITAR BUGS QUE ESTAVAM SENDO CAUSADOS PELA ALOCAÇÃO DINÂMICA FOI NECESSÁRIO UTILIZAR
//UMA CAMADA EXTRA DE PONTEIRO PARA QUE HOUVESSE SEGURANÇA EM RELAÇÃO AOS RESULTADOS
void Aloca(int n, double ****funcoes, double ***jacobiana, double ***l, double ***u, double **xa,
double **xd);
void Desaloca(int n, double ****funcoes, double ***jacobiana, double ***l, double ***u, double **xa,
double **xd);
void LeFuncoes(int n, double ****funcoes, double ***jacobiana, double **xa);
void ImprimeFuncao(int n, double ****funcoes);
void imprimeMatriz(int n, double **jacobiana, double **l, double **u);
void aplicaFuncoes(int n, double ****funcoes, double **x, double **result);
void zerarTriangInf(int n, double ***p, double ***l, double ***jacobiana);
void calculaResultadoU(int n, double ***p, double **B, double **result);
void calculaResultadoL(int n, double ***l, double **B, double **result);
double maior(int n, double **x);

int main()
{
	int n, e, i, k;
	double ***funcoes;
	double **jacobiana, **l, **u;
	double *xa,*xd; //X atual e X diferença
	double erro, parada;

	printf("Quantos variáveis você deseja encontrar?\n");
	scanf("%d", &n);
	printf("Qual o erro desejado?\n10^");
	scanf("%d", &e);
	erro = pow(10, e);

	double *xaux  = calloc(n,sizeof(double));
	double *xaux2 = calloc(n,sizeof(double));

	Aloca(n, &funcoes, &jacobiana, &l, &u, &xa, &xd);

	LeFuncoes(n, &funcoes, &jacobiana, &xa);

	system("clear");

	zerarTriangInf(n, &u, &l, &jacobiana);

	ImprimeFuncao(n, &funcoes);
	imprimeMatriz(n, jacobiana, l, u);

	parada = erro + 4;
	k = 0;
	printf("============================================================\n");
	printf("===============MÉTODO DE NEWTON MODIFICADO===============\n");
	printf("============================================================\n");
	printf("___K___|");
	for(i = 0; i < n; i++)
		printf("__X%d__|", i);
	printf("___PARADA___\n");
	printf("  %2d   |", 0);
	for(i = 0; i < n; i++)
		printf("%2.4lf|", xa[i]);
	printf("   \n");

	while(parada > erro)
	{
		aplicaFuncoes(n, &funcoes, &xa, &xaux);

		calculaResultadoL(n, &l, &xaux,  &xaux2);
		calculaResultadoU(n, &u, &xaux2, &xd);

		for(i = 0; i < n; i++)
			xa[i] += xd[i];

		parada = fabs(maior(n, &xd));
		printf("  %2d   |", ++k);
		for(i = 0; i < n; i++)
			printf("%2.4lf|", xa[i]);
		printf("   %.5lf\n", parada);
	}

	printf("\nResultado final encontrado:\n");
```

```
	for(i = 0; i < n; i++)
	{
		printf("x%d = %2.4lf\n", i, xa[i]);
	}
	printf("Com erro de: %.5lf\n", parada);
	Desaloca(n, &funcoes, &jacobiana, &l, &u, &xa, &xd);
	free(xaux);
	free(xaux2);
}

void aplicaFuncoes(int n, double ****funcoes, double **x, double **result)
{
	int i, j;
	for(i = 0; i < n; i++)
		(*result)[i] = 0;
	for(i = 0; i < n; i++)
	{
		for(j = 0; j < n; j++)
		{
			(*result)[i] += (*funcoes)[i][j][0] * pow((*x)[j],(*funcoes)[i][j][1]);
		}
		(*result)[i] += (*funcoes)[i][n][0];
	}

}

void Aloca(int n, double ****funcoes, double ***jacobiana, double ***l, double ***u, double **xa,
double **xd)
{
	int i, j;
	*funcoes = (double ***) calloc(n, sizeof(double **));
	for(i = 0; i < n; i++)
	{
		(*funcoes)[i] = (double **) calloc(n+1, sizeof(double *));
		for(j = 0; j <= n; j++)
			(*funcoes)[i][j] = (double *) calloc(2, sizeof(double));
	}
	*jacobiana = (double**) malloc(n * sizeof(double*));
	*l = (double**) malloc(n * sizeof(double*));
	*u = (double**) malloc(n * sizeof(double*));
	for(i = 0; i < n; i++)
	{
		(*jacobiana)[i] = (double*) malloc(n * sizeof(double));
		(*l)[i] = (double*) malloc(n * sizeof(double));
		(*u)[i] = (double*) malloc(n * sizeof(double));
	}
	(*xa) = (double *) calloc(n,sizeof(double));
	(*xd) = (double *) calloc(n,sizeof(double));
}

void Desaloca(int n, double ****funcoes, double ***jacobiana, double ***l, double ***u, double **xa,
double **xd)
{
	int i, j;
	for(i = 0; i < n; i++)
	{
		for(j = 0; j <= n; j++)
			free((*funcoes)[i][j]);
		free((*funcoes)[i]);
	}
	free((*funcoes));
	for(i = 0; i < n; i++)
	{
		free((*jacobiana)[i]);
		free((*l)[i]);
		free((*u)[i]);
	}
	free((*jacobiana));
	free((*l));
	free((*u));
	free((*xa));
	free((*xd));
}

void LeFuncoes(int n, double ****funcoes, double ***jacobiana, double **xa)
{
	int i, j;
	for(i = 0; i < n; i++)
	{
		printf("Função: %2d\n", i+1);
```

```
                for(j = 0; j < n; j++)
                {
                        printf("a * x%2d^(?)\na = ", j+1);
                        scanf("%lf", (*funcoes)[i][j]);
                        printf("? = ");
                        scanf("%lf", (*funcoes)[i][j]+1);
                        putchar('\n');
                }
                printf("Caso exista uma constante, insira a mesma, caso contrário entre com 0: ");
                scanf("%lf", &(*funcoes)[i][n][0]);
                (*funcoes)[i][n][1] = 1;
                putchar('\n');
                putchar('\n');
        }

        for(i = 0; i < n; i++)
                for(j = 0; j < n; j++)
                        (*jacobiana)[i][j] = (*funcoes)[i][j][0];

        printf("Insira os valores de x0\n");
        for(i = 0; i < n; i++)
                scanf("%lf", &(*xa)[i]);

}

void ImprimeFuncao(int n, double ****funcoes)
{
        int i, j;
        printf("Funções:\n");
        for(i = 0; i < n; i++)
        {
                for(j = 0; j < n; j++)
                {
                        if((*funcoes)[i][j][1] == 1)
                                printf("%+.4lf*x%d          ", (*funcoes)[i][j][0], j+1);
                        else
                                printf("%+.4lf*x%d^(%.4lf) ", (*funcoes)[i][j][0], j+1, (*funcoes)[i]
[j][1]);
                }
                if((*funcoes)[i][n][0]==0)
                        printf("= %+.4lf\n", (*funcoes)[i][n][0]);
                else
                        printf("= %+.4lf\n", -(*funcoes)[i][n][0]);
        }
}

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

        for(a = 0; a < n; a++)
        {//Zera a matriz L e define sua diagonal principal como 1. E copia os valores para a matriz P
                for(b = 0; b < n; b++)
                {
                        if(a==b)
                        {
                                (*l)[a][b] = 1;
                        }
                        else
                                (*l)[a][b] = 0;
                        (*p)[a][b] = (*jacobiana)[a][b];
                }
        }

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

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

                        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 imprimeMatriz(int n, double **jacobiana, double **l, double **u)
{
        int i,j;

        printf("Jacobiana:\t\t\t\tL:\t\t\t\t\tU:\n");

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

        putchar('\n');
}

double maior(int n, double **x)
{
        int i;
        double maior = fabs((*x)[0]);
        for(i = 1; i < n; i++)
        {
                if( fabs((*x)[i]) > maior )
                        maior = fabs((*x)[i]);
        }
        return maior;
}

void calculaResultadoU(int n, double ***p, double **B, double **result)
{
        double sum;
        int a, b, count=(n-1);
        for(a = 0; a < n; a++)
                (*result)[a] = 0;
        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--;
        }
}

void calculaResultadoL(int n, double ***l, double **B, double **result)
{
        double sum;
        int a, b;
        for(a = 0; a < n; a++)
                (*result)[a] = 0;
        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];
        }
}
```

IO (TUDO QUE ESTIVER SEGUIDO DE '>' CORRESPONDE A UMA ENTRADA DO USUARIO)

```
Quantos variáveis você deseja encontrar?
>3
Qual o erro desejado?
10^ >-4
Função:  1
a * x 1^(?)
a = >1
? = >2

a * x 2^(?)
a = >1
? = >2

a * x 3^(?)
a = 1
? = 2

Caso exista uma constante, insira a mesma, caso contrário entre com 0: >-1


Função:  2
a * x 1^(?)
a = >2
? = >2

a * x 2^(?)
a = >1
? = >2

a * x 3^(?)
a = >-4
? = >1

Caso exista uma constante, insira a mesma, caso contrário entre com 0: >0


Função:  3
a * x 1^(?)
a = >3
? = >2

a * x 2^(?)
a = >-4
? = >1

a * x 3^(?)
a = >1
? = >2

Caso exista uma constante, insira a mesma, caso contrário entre com 0: >0


Insira os valores de x0
>0.5
>0.5
>0.5
```

Funções:
+1.0000*x1^(2.0000) +1.0000*x2^(2.0000) +1.0000*x3^(2.0000) = +1.0000
+2.0000*x1^(2.0000) +1.0000*x2^(2.0000) -4.0000*x3 = +0.0000
+3.0000*x1^(2.0000) -4.0000*x2 +1.0000*x3^(2.0000) = +0.0000

Jacobiana:
+1.000 +1.000 +1.000
+2.000 +1.000 -4.000
+3.000 -4.000 +1.000

L:
+1.000 +0.000 +0.000
+2.000 +1.000 +0.000
+3.000 +7.000 +1.000

U:
+1.000 +1.000 +1.000
+0.000 -1.000 -6.000
+0.000 +0.000 +40.000

==========
==========MÉTODO DE NEWTON MODIFICADO==========
==========

K	X0	X1	X2	PARADA
0	0.5000	0.5000	0.5000	
1	0.8750	0.5000	0.3750	0.37500
2	0.7266	0.4969	0.3703	0.14844
3	0.8153	0.4966	0.3700	0.08869
4	0.7672	0.4966	0.3699	0.04810
5	0.7952	0.4966	0.3699	0.02801
6	0.7794	0.4966	0.3699	0.01575
7	0.7885	0.4966	0.3699	0.00905
8	0.7833	0.4966	0.3699	0.00514
9	0.7863	0.4966	0.3699	0.00294
10	0.7846	0.4966	0.3699	0.00167
11	0.7855	0.4966	0.3699	0.00096
12	0.7850	0.4966	0.3699	0.00054
13	0.7853	0.4966	0.3699	0.00031
14	0.7851	0.4966	0.3699	0.00018
15	0.7852	0.4966	0.3699	0.00010
16	0.7852	0.4966	0.3699	0.00006

Resultado final encontrado:
x0 = 0.7852
x1 = 0.4966
x2 = 0.3699
Com erro de: 0.00006