Universidade Federal do Espírito Santo – Campus Goiabeiras
Graduação em Engenharia Elétrica
Algoritmos Numéricos

Trabalho Computacional

Alaf do Nascimento Santos

Igor Batista Vieira

Vitória, 06 de novembro de 2018

1. Introdução

Existem diferentes métodos para se resolver sistemas lineares do tipo Ax = b, dentre os métodos existentes destacam-se os métodos numéricos por facilitarem o desenvolvimento de soluções que, analiticamente, seriam impossíveis ou muito custosas. Para uso dos métodos numéricos, muitas vezes se faz necessário uma ferramenta computacional, e entender o funcionamento deste tipo de ferramenta é de grande importância para que sempre se busque usar o melhor e mais adequado método para um determinado tipo de problema. Neste trabalho computacional tem-se o objetivo de implementar dois métodos numéricos bastante conhecidos, o método da eliminação de Gauss (método direto) e o método de Gauss-Seidel (método iterativo), sendo que, a idéia principal é adequar a implementação dos mesmos para que se trabalhe com matrizes de entrada pentadiagonais, procurando-se sempre diminuir ao máximo a quantidade de operações realizadas para se chegar à solução.

2. Método Numérico 1

O método da eliminação de Gauss teve início com o pivoteamento, no qual observou-se que seria necessário somente comparar cada elemento da diagonal principal com os dois elementos abaixo dele, como pode ser visto no trecho de código abaixo:

```
for(int k = 0; k < (n-1); k++)
{
	int maior = fabs(A[k][k]), linhaMaior = k;
	for(int i = k+1; i < (k+3); i++)
	{
		if(fabs(A[i][k]) > maior)
		{
			maior = fabs(A[i][k]);
			linhaMaior = i;
		}
		if(k == n-2) break;//(*)
	}
	if(linhaMaior != k)
	{
		for(int j = 0; j < n; j++)
		{
			double auxiliar = A[k][j];
			A[k][j] = A[linhaMaior][j];
			A[linhaMaior][j] = auxiliar;
		}
		double bAuxiliar = b[k];
		b[k] = b[linhaMaior];
		b[linhaMaior] = bAuxiliar;
	}
	//escalonamento começa aqui
}
```

A observação feita anteriormente é válida na maioria dos casos, porém, quando trabalha-se com os dois últimos pivôs de A. tem-se que não existem mais dois elemento abaixo dos mesmos, sendo o que o último pivô não tem nada para se comparar, logo não é feita uma verificação para ele, em contrapartida para o penúltimo pivô é utilizado um if (*) que diz quando parar a verificação.

Sobre a etapa de escalonamento notou-se, experimentalmente, que para matrizes pentadiagonais, cada linha i só tem no máximo 2 multiplicadores m, tal que m é diferente de zero, tendo em vista isso, implementou-se um contador que para o escalamento para uma determinada linha i sempre que o processo já tenha sido feito duas vezes, novamente aqui foi utilizado também a verificação de posição da linha em que se trabalha tendo um if (**) responsável por parar o loop quando o mesmo fosse começar a acessar posições nulas. Veja abaixo:

```
for(int i = k+1; i < k+3; i++)
{
        if(contador >= 2)
                break;
        else
                contador++;
        double m = A[i][k]/A[k][k];
        numOp++;
        A[i][k] = 0;
        for(int j = k+1; j < k+2; j++)
        {
                A[i][j] -= m*A[k][j];
                numOp+=2;
        }
        b[i] -= m*b[k];
        numOp+=2;

        if(k == n-2) //(**)
                break;
}
```

Para tratar da retro-substituição envolvida no método de eliminação de Gauss, foi necessário observar que as posições não nulas de A estão onde o valor absoluto da diferença entre o índice da coluna e o índice da linha tratada é igual a 1, ou seja, um elemento ao lado direito e o outro ao lado esquerdo do pivô daquela linha. Para o caso da última linha, temos uma pequena diferença, pois o pivô nesse caso não tem um vizinho a direita, isso foi tratado com o if (***). Veja:

```
x[n-1]= b[n-1]/A[n-1][n-1];
numOp++;
for(int i = (n-2); i >= 0; i--)
{
        double soma = b[i];
        for(int j = i+1; j < n;j++)
        {
                if(i == 0 && j > 2) //(***)
                        break;
```

```
                else if(abs(i - j) != 1)
                        continue;
                soma -= A[i][j]*x[j];
                numOp+=2;
        }
        x[i]= soma/A[i][i];
        numOp++;
}
```

3. **Método Numérico 2**

A estratégia principal utilizada no método de Seidel para matrizes pentadiagonais está no trecho de código abaixo:

```
for(int i = 0; i < n; i++)
{
        x[i] = b2[i];
        for(int j = 0; j < n; j++)
        {
                if(j < (i-2))
                        continue;
                else if(j > (i+2))
                        break;
                else if(j > i)
                        x[i] -= A2[i][j]*x_anterior[j];
                else if(i > j)
                        x[i] -= A2[i][j]*x[j];
                if(i != j)
                        numOp+=2;
        }
        x[i] = x[i]/A2[i][i];
        numOp++;
}
```

O segredo do não acesso a posições nulas nesse código está nas verificações presentes nos primeiros if's (j < (i-2)) e j > (i+2), sendo j o contador de colunas e i o contador de linhas). Percebemos esse padrão ao implementarmos Gauss-Seidel para uma matriz comum e então analisá-lo verificando os valores de i e j nos quais eram acessados posições nulas em A e assim foi encontrado um padrão, que pode ser melhor visto observando-se a imagem abaixo:

$$\begin{bmatrix} d_1 & c_1 & f_1 & & & & & & \\ a_1 & d_2 & c_2 & f_2 & & & & & \\ e_1 & a_2 & d_3 & c_3 & f_3 & & & & \\ & e_2 & a_3 & d_4 & c_4 & f_4 & & & \\ & & & \ddots & \ddots & \ddots & \ddots & & \\ & & & e_{i-2} & a_{i-1} & d_i & c_i & f_i & \\ & & & & & \ddots & \ddots & \ddots & \ddots \\ & & & & & e_{n-4} & a_{n-3} & d_{n-2} & c_{n-2} \\ & & & & & & e_{n-3} & a_{n-2} & d_{n-1} \\ & & & & & & & e_{n-2} & a_{n-1} \end{bmatrix} \begin{bmatrix} f_{n-2} \\ c_{n-1} \\ d_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_i \\ \vdots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ \vdots \\ b_i \\ \vdots \\ b_{n-2} \\ b_{n-1} \\ b_n \end{bmatrix}$$

Percebe-se que, para uma determinada linha X, os únicos valores vizinhos à d_x não nulos estão localizados nas duas próximas colunas, a direita (col. X+1 e col. X+2), e no máximo, duas colunas atrás, a esquerda (col. X-1 e col. X-2, na medida do possível, pois para a linha 1, por exemplo, não existe vizinho a esquerda de d_1, mas o critério a direita se mantém).

Baseado nessas observações, sempre que a coluna acessada tiver um índice menor que o índice da linha a se trabalhar decrementado de duas unidades, pulou-se para a próxima coluna. E sempre que o índice da coluna acessada for maior que o índice da linha a se trabalhar incrementado de duas unidades, pulou-se para a próxima linha.

4. Resultados

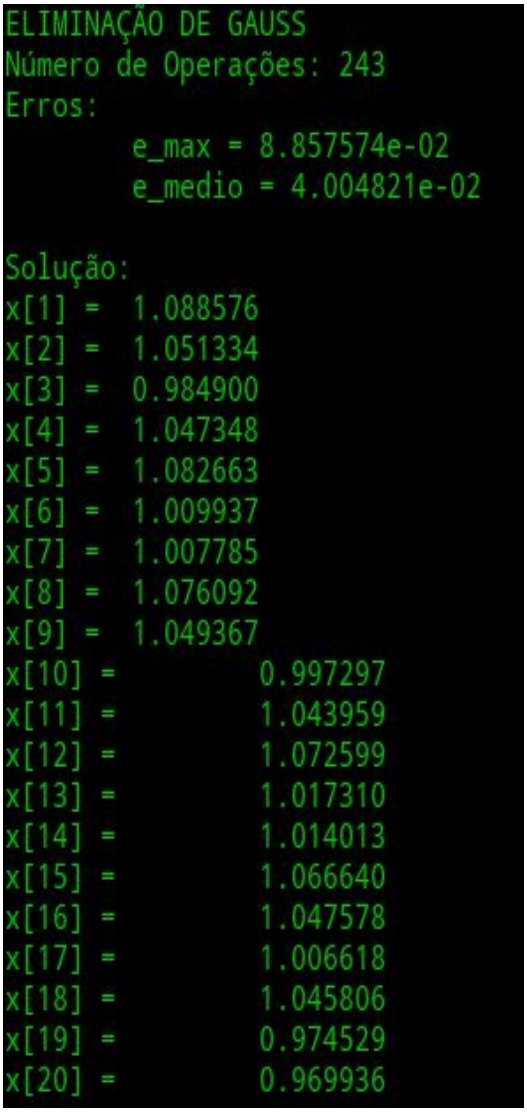
4.1 Apresentar os resultados para o problema linear descrito no item 1 da seção de execuções. Mostrar também a matriz A e o vetor b.

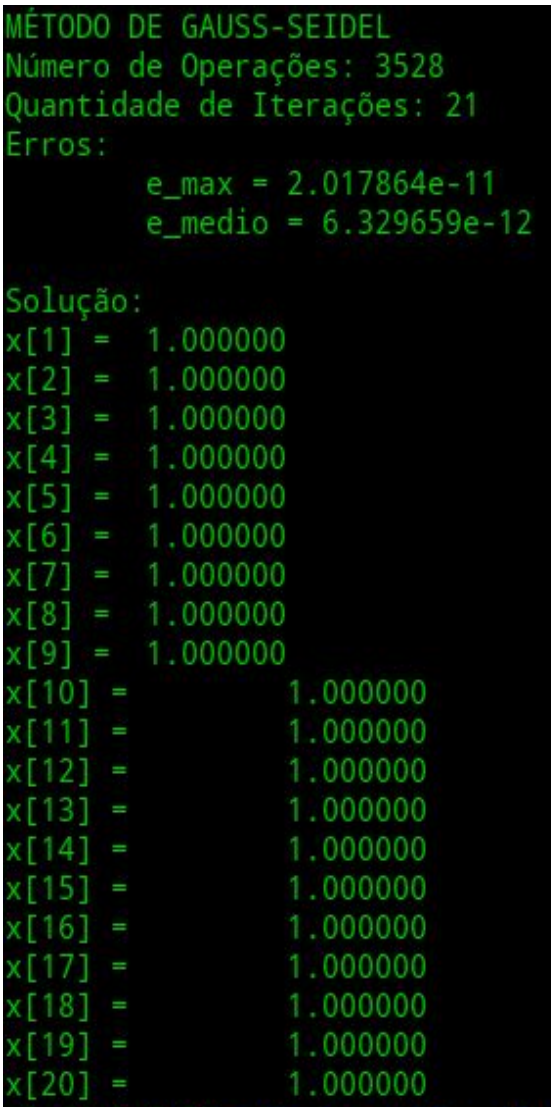
```
n: 20
dA: 2.0
dB 1.0
DP: 2.2
dC: 0.1
dD: 0.2


[A|b] =

  2.2   0.1   0.2   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0 | 2.5
  1.0   2.2   0.1   0.2   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0 | 3.5
  2.0   1.0   2.2   0.1   0.2   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0 | 5.5
  0.0   2.0   1.0   2.2   0.1   0.2   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0 | 5.5
  0.0   0.0   2.0   1.0   2.2   0.1   0.2   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0 | 5.5
  0.0   0.0   0.0   2.0   1.0   2.2   0.1   0.2   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0 | 5.5
  0.0   0.0   0.0   0.0   2.0   1.0   2.2   0.1   0.2   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0 | 5.5
  0.0   0.0   0.0   0.0   0.0   2.0   1.0   2.2   0.1   0.2   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0 | 5.5
  0.0   0.0   0.0   0.0   0.0   0.0   2.0   1.0   2.2   0.1   0.2   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0 | 5.5
  0.0   0.0   0.0   0.0   0.0   0.0   0.0   2.0   1.0   2.2   0.1   0.2   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0 | 5.5
  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   2.0   1.0   2.2   0.1   0.2   0.0   0.0   0.0   0.0   0.0   0.0   0.0 | 5.5
  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   2.0   1.0   2.2   0.1   0.2   0.0   0.0   0.0   0.0   0.0   0.0 | 5.5
  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   2.0   1.0   2.2   0.1   0.2   0.0   0.0   0.0   0.0   0.0 | 5.5
  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   2.0   1.0   2.2   0.1   0.2   0.0   0.0   0.0   0.0 | 5.5
  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   2.0   1.0   2.2   0.1   0.2   0.0   0.0   0.0 | 5.5
  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   2.0   1.0   2.2   0.1   0.2   0.0   0.0 | 5.5
  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   2.0   1.0   2.2   0.1   0.2   0.0 | 5.5
  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   2.0   1.0   2.2   0.1   0.2 | 5.5
  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   2.0   1.0   2.2   0.1 | 5.3
  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   2.0   1.0   2.2 | 5.2


ELIMINAÇÃO DE GAUSS
Número de Operações: 243
Erros:
        e_max = 8.857574e-02
        e_medio = 4.004821e-02
```





4.2 Apresentar os resultados (através de uma tabela) para os problemas lineares descritos no item 2 da seção de execuções. A tabela deve exibir, para cada dimensão (n): o esforço total (quantidade de operações), os valores eMax e eMedio, para cada método.

(dA = 2.0, dB = 1.0, DP = 2.2, dC = 0.1 e dD = 0.2)

Eliminação de Gauss			
ordem (n)	Esforço (operações)	e_{medio}	e_{max}
40	503	3.8960E-02	8.8576E-02
80	1023	3.8594E-02	8.8576E-02
160	2063	3.8156E-02	8.8576E-02
320	4143	3.7947E-02	8.8576E-02

640	8303	3.7841E-02	8.8576E-02
1280	16623	3.7789E-02	8.8576E-02
2560	33263	3.7762E-02	8.8576E-02
Gauss-Seidel			
ordem (n)	Esforço (operações)	e_{medio}	e_{max}
40	8700	6.5719E-12	4.4083E-11
80	22656	3.0793E-12	3.0767E-11
160	59976	4.0977E-12	7.7490E-11
320	174948	1.6261E-12	4.9685E-11
640	528816	1.9549E-12	8.0651E-11
1280	1703184	2.2959E-12	1.5062E-10
2560	5849112	1.5761E-12	1.6758E-10

4.3 Apresentar os resultados (através de uma tabela) para os problemas lineares descritos nos ítens 3 e 4 da seção de execuções. A tabela deve exibir, para cada dimensão (n): o esforço total (quantidade de operações), os valores eMax e eMedio, para cada método.

(dA = 1.0, dB = 1.0, DP = 4.5, dC = 1.0 e dD = 1.0)

Eliminação de Gauss			
ordem (n)	Esforço (operações)	e_{medio}	e_{max}
40	503	1.3128E-01	1.8789E-01
80	1023	1.3230E-01	1.8789E-01
160	2063	1.3282E-01	1.8789E-01
320	4143	1.3308E-01	1.8789E-01
640	8303	1.3320E-01	1.8789E-01
1280	16623	1.3327E-01	1.8789E-01
2560	33263	1.3330E-01	1.8789E-01

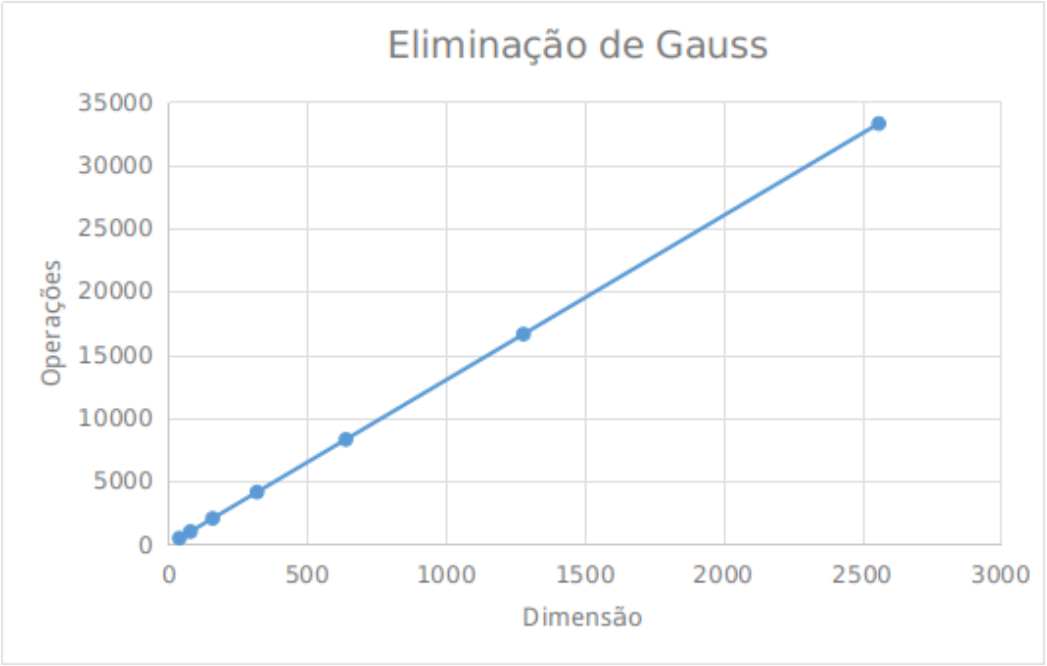
Gauss-Seidel			
ordem (n)	Esforço (operações)	e_{medio}	e_{max}
40	7656	9.1714E-12	3.2773E-11
80	15576	6.9615E-12	3.2773E-11
160	31416	5.9150E-12	3.2773E-11
320	63096	5.3918E-12	3.2773E-11
640	126456	5.1302E-12	3.2773E-11
1280	253176	4.9994E-12	3.2773E-11
2560	506616	4.9340E-12	3.2773E-11

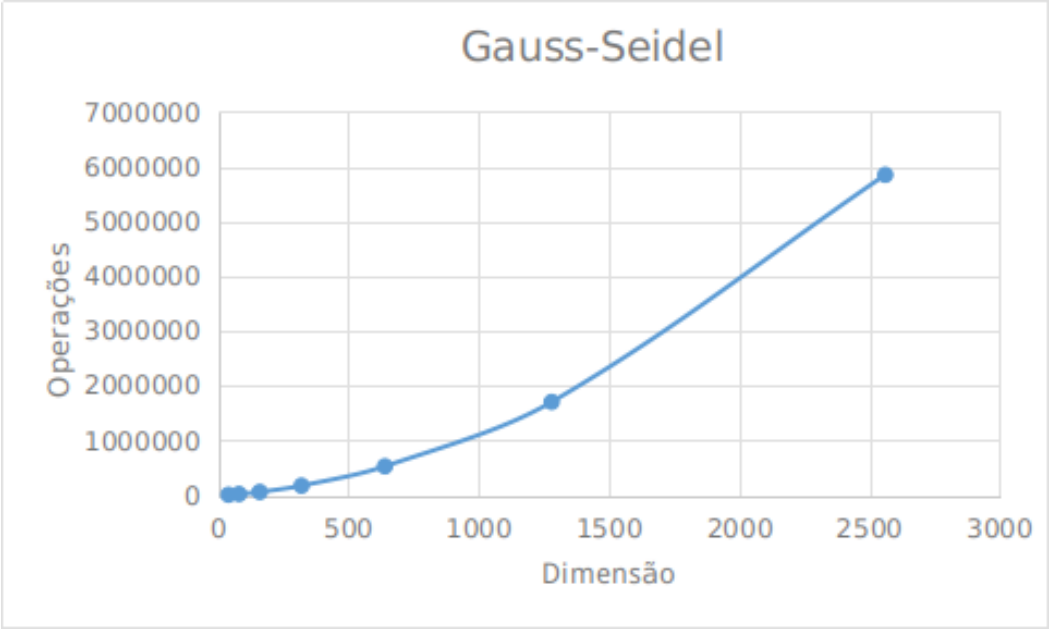
(dA = 1.0, dB = 2.0, DP = 3.1, dC = 2.0 e dD = 1.0)

Eliminação de Gauss			
ordem (n)	Esforço (operações)	e_{medio}	e_{max}
40	503	3.5770E+00	1.1762E+01
80	1023	3.5143E+00	1.1761E+01
160	2063	3.4831E+00	1.1761E+01
320	4143	3.4675E+00	1.1761E+01
640	8303	3.4597E+00	1.1761E+01
1280	16623	3.4558E+00	1.1761E+01
2560	33263	3.4538E+00	1.1761E+01
Gauss-Seidel			
ordem (n)	Esforço (operações)	e_{medio}	e_{max}
40	92220	6.8090E-10	1.5515E-09
80	188328	5.8678E-10	1.5413E-09
160	378420	5.5107E-10	1.5769E-09

320	760020	2.8778E-10	1.5783E-09
640	1523220	1.4389E-10	1.5783E-09
1280	3049620	7.1945E-11	1.5783E-09
2560	6102420	3.5973E-11	1.5783E-09

4.3 Para os problemas lineares descritos no item 2 da seção de execuções, traçar em um par de eixos cartesianos, para cada método, o esforço total (a quantidade de operações) versus a dimensão.





7. Código Fonte:

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

void main()
{
	int n = 0, numOp = 0;
	double dA, dB, DP, dC, dD, e_max = 0, e_medio = 0, tol = pow(10,-10);
	printf("n: "); scanf("%ld", &n);
	printf("dA: "); scanf("%lf", &dA);
	printf("dB ");  scanf("%lf", &dB);
	printf("DP: "); scanf("%lf", &DP);
	printf("dC: "); scanf("%lf", &dC);
	printf("dD: "); scanf("%lf", &dD);


	double  **A = calloc(n, sizeof(double*)); //aloca as linhas da matriz;
			for(int i = 0; i < n; i++)
				A[i] = (double*) calloc(n, sizeof(double)); //aloca as colunas da matriz
	double  **A2 = calloc(n, sizeof(double*)); //aloca as linhas da matriz;
			for(int i = 0; i < n; i++)
				A2[i] = (double*) calloc(n, sizeof(double)); //aloca as colunas da matriz

	double *b, *b2, *e, *x, *x_anterior;
	b = (double*) calloc(n, sizeof(double));
	b2 = (double*) calloc(n, sizeof(double));
	e = (double*) calloc(n, sizeof(double));
	x = (double*) calloc(n, sizeof(double));
	x_anterior = (double*) calloc(n, sizeof(double));

	/*CRIA A MATRIA "A"*/
	for(int i = 0; i < n; i++)
	{
		for(int j = 0; j < n; j++)
		{
			if((i-j) == 2) A[i][j] = dA;
			else if((i-j) == 1) A[i][j] = dB;
			else if((i-j) == 0) A[i][j] = DP;
			else if((i-j) == -1) A[i][j] = dC;
			else if((i-j) == -2) A[i][j] = dD;
		}
	}

	/*CRIA O VETOR "b"*/
	for(int i = 0; i < n; i++) b[i] = 0;
	for(int i = 0; i < n; i++)
		for(int j = 0; j < n; j++)
			if(fabs(i-j) <= 2) //considera a matriz como pentadiagonal
				b[i] += A[i][j];


	/*Faz uma cópia de A e b para usar no método de seidel*/
	for(int i = 0; i < n; i++)
```

```
	{
		b2[i] = b[i];
		for(int j = 0; j < n; j++)
			A2[i][j] = A[i][j];
	}

	/* IMPRIME O SISTEMA DADO*/
	printf("\n\n[A|b] = \n\n");
for(int i = 0; i < n; i++)
{
  for(int j = 0;j < n; j++) printf("  %.1f ", A[i][j]);
  printf(" | %.1lf\n", b[i]);
}

	printf("\n\nELIMINAÇÃO DE GAUSS\n");
	for(int k = 0; k < (n-1); k++)
	{
		int maior = fabs(A[k][k]), linhaMaior = k;
		/*PIVOTEAMENTO*/
		for(int i = k+1; i < (k+3); i++)
		{
			if(fabs(A[i][k]) > maior)
			{
				maior = fabs(A[i][k]);
				linhaMaior = i;
			}
			if(k == n-2)
				break;
		}
		if(linhaMaior != k)
		{
			for(int j = 0; j < n; j++)
			{
				double auxiliar = A[k][j];
				A[k][j] = A[linhaMaior][j];
				A[linhaMaior][j] = auxiliar;
			}
			double bAuxiliar = b[k];
			b[k] = b[linhaMaior];
			b[linhaMaior] = bAuxiliar;
		}//FIM PIVOTEAMENTO

		/*ESCALONAMENTO*/
		unsigned int contador = 0;
		for(int i = k+1; i < k+3; i++)
		{
		  if(contador >= 2)
			  break;
		  else
		    contador++;
			double m = A[i][k]/A[k][k];
			numOp++;
			A[i][k] = 0;
			for(int j = k+1; j < k+2; j++)
```

```
                        {
                                A[i][j] -= m*A[k][j];
                                numOp+=2;
                        }
                        b[i] -= m*b[k];
                        numOp+=2;

                        if(k == n-2)
                                break;
                }
                //FIM ESCALONAMENTO
        }

        /* RETRO-SUBSTITUIÇÃO*/
        x[n-1]= b[n-1]/A[n-1][n-1];
        numOp++;
  for(int i = (n-2); i >= 0; i--)
  {
   double soma = b[i];
   for(int j = i+1; j < n;j++)
   {
        if(i == 0 && j > 2)
                break;
        else if(abs(i - j) != 1)
                continue;
        soma -= A[i][j]*x[j];
        numOp+=2;
   }
   x[i]= soma/A[i][i];
   numOp++;
  }
/*CALCULO DO VETOR DE ERROS*/
  for(int i = 0; i < n; i++)
        e[i] = fabs(x[i] - 1);
  for(int i = 0; i < n; i++)
  {
        e_medio += e[i];
        if(e_max < e[i])
                e_max = e[i];
  } e_medio = e_medio/n;
        printf("Número de Operações: %d\n", numOp);
        printf("Erros:\n\te_max = %e\n\te_medio = %e\n\n", e_max, e_medio);
        printf("Solução:\n");
                for(int i = 0; i < n; i++)
                        printf("x[%d] = \t%lf\n", i+1, x[i]);
        numOp = 0; e_max = 0; e_medio = 0;

        printf("\n\nMÉTODO DE GAUSS-SEIDEL\n");
        /*Cria Vetor Chute inicial*/
        for(int i = 0; i < n; i++)
                x_anterior[i] = b2[i]/A2[i][i];

        unsigned int iteracao = 1;
        for(;;)
```

```
            {
        /*Seidel p/ Pentadiagonal*/
        for(int i = 0; i < n; i++)
        {
           x[i] = b2[i];
           for(int j = 0; j < n; j++)
           {
              if(j < (i-2))
                 continue;
              else if(j > (i+2))
                 break;
              else if(j > i)
                 x[i] -= A2[i][j]*x_anterior[j];
              else if(i > j)
                 x[i] -= A2[i][j]*x[j];
              if(i != j)
                 numOp+=2;
           }
           x[i] = x[i]/A2[i][i];
           numOp++;
        }
              /* VERIFICANDO CRITÉRIOS DE PARADA */
              double maiorBaixo = 0, maiorCima = 0, difRel = 0;
              for(int i = 0; i < n; i++)
              {
                        if(maiorBaixo < x[i])
                                  maiorBaixo = x[i];
                        if(maiorCima < fabs(x_anterior[i]-x[i]))
                                  maiorCima = fabs(x_anterior[i]-x[i]);
              }

                        difRel = maiorCima/maiorBaixo;
                        if(difRel < tol)
                                  break;
                        else iteracao++;
                        for(int i = 0; i < n; i++)
                                  x_anterior[i] = x[i];
              }
              /*CALCULO DO VETOR DE ERROS*/
     for(int i = 0; i < n; i++)
              e[i] = fabs(x[i] - 1);
     for(int i = 0; i < n; i++)
     {
              e_medio += e[i];
              if(e_max < e[i])
                        e_max = e[i];
     } e_medio = e_medio/n;
              printf("Número de Operações: %d\n", numOp);
              printf("Quantidade de Iterações: %d\n", iteracao);
              printf("Erros:\n\te_max = %e\n\te_medio = %e\n\n", e_max, e_medio);
              printf("Solução:\n");
                        for(int i = 0; i < n; i++)
                                  printf("x[%d] = \t%lf\n", i+1, x[i]);
}
```