Trabalho 1
Métodos Iterativos para Sistemas Lineares

Eduardo Brunaldi dos Santos
8642515

Jorge Ashkar Ferreira Simondi
8517081

Victor Luiz da Silva Mariano Pereira
8602444

2018

Introdução

1 Método Iterativo de Gauss-Seidel

2 Códigos Fonte

2.1 Função principal (`main.c`)

```c
/**
 *    Trabalho 1 -  Métodos Iterativos para  Sitemas Lineares
 *
 *        Cálculo Numérico    SME-0104
 *        Prof.: Murilo Francisco Tomé
 *
 *        Eduardo Brunaldi dos Santos             8642515
 *        Jorge Ashkar Ferreira Simondi           8517081
 *        Victor Luiz da Silva Mariano Pereira    8602444
 */

#include <stdio.h>
#include <stdlib.h>
#include <gauss_seidel.h>

int main (int argc, char *argv[]){
    // Variáveis de entrada
    int n;              // Quantidade de equações do sistema linear
    int itmax;          // Número máximo de iterações
    long double **A;    // Matriz de sistemas lineares
    long double *b;     // Vetor resultado das equações
    long double *x;     // Vetor inicial
    long double e;      // Erro permitido, precisão

    // Iteradores
    int i;
    int j;

    // Dimensão da matriz (número de equações do sistema linear)
    scanf("%d", &n);

    // Alocação da matriz
    A = malloc(sizeof(long double *) * n);
    for (i = 0; i < n; i++)
        A[i] = malloc(sizeof(long double) * n);

    // Pegando valores de A
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%Lf", &(A[i][j]));

    // Alocação do vetor resultado
    b = malloc(sizeof(long double) * n);

    // Pegando os valores de b
    for(i = 0; i < n; i++)
        scanf("%Lf", &b[i]);

    // Alocação do vetor chute
    x = malloc(sizeof(long double) * n);

    // Pegando os valores do x(0), o vetor inicial
    for(i = 0; i < n; i++)
        scanf("%Lf", &x[i]);

    // Pegando o valor da precisão (erro permitido)
```

```
    scanf("%Lf", &e);

    // Pegando a quantidade máxima de iterações
    scanf("%d", &itmax);

    // Calcula um valor aproximado para a resposta
    // usando o método de Gauss-Seidel
    x = gauss_seidel(A, b, x, n, e, itmax);

    // Imprime a solução na tela
    imprime_vetor(x, n);

    // Liberando a memória
    free(x);
    free(b);
    for (i = 0; i < n; i++)
        free(A[i]);
    free(A);

    return 0;
}
```

2.2 Biblioteca auxiliar

2.2.1 *Header* (gauss_seidel.h)

```
/**
 *    Trabalho 1 -  Métodos Iterativos para  Sitemas Lineares
 *
 *        Cálculo Numérico    SME-0104
 *        Prof.: Murilo Francisco Tomé
 *
 *        Eduardo Brunaldi dos Santos             8642515
 *        Jorge Ashkar Ferreira Simondi           8517081
 *        Victor Luiz da Silva Mariano Pereira    8602444
 */

#ifndef GAUSS_SEIDEL_H
#define GAUSS_SEIDEL_H

/**
 * Função para imprimir de forma mais legível uma matriz quadrada
 * @param A Matriz a ser impressa
 * @param n dimensão da matriz
 */
void imprime_matriz(long double **A, int n);

/**
 * Função para imprimir um vetor de forma mais legível
 * @param v vetor a ser impresso
 * @param n tamanho do vetor
 */
void imprime_vetor(long double *v, int n);

/**
 * Função para retornar a norma infinita de um vetor obtido pela subtração
 * de dois vetores
 * @param  xk Vetor x(k+1)
```

```
 * @param  x  Vetor x(k)
 * @param  n  Dimensão dos vetores
 * @return    norma do vetor obtido pela subtração
 */
long double norma_infinita(long double *xk, long double *x, int n);

/**
 * Função para resolver o sistema linear usando o método de gauss-seidel
 * @param A     Matriz de funções do sistema linear
 * @param b     Resultados das equações do sistema linear
 * @param x     Vetor contendo os resultados iniciais
 * @param n     Dimensão do sistema linear
 * @param e     Precisão
 * @param itmax Número máximo de iterações
 */
long double *gauss_seidel(long double **A, long double *b, long double *x, int n, long double e, int itmax);

#endif
```

2.2.2 Implementação da biblioteca (gauss_seidel.c)

```
/**
 *    Trabalho 1 -  Métodos Iterativos para  Sitemas Lineares
 *
 *        Cálculo Numérico    SME-0104
 *        Prof.: Murilo Francisco Tomé
 *
 *        Eduardo Brunaldi dos Santos            8642515
 *        Jorge Ashkar Ferreira Simondi          8517081
 *        Victor Luiz da Silva Mariano Pereira   8602444
 */

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

/**
 * Função para imprimir de forma mais legível uma matriz quadrada
 * @param A Matriz a ser impressa
 * @param n dimensão da matriz
 */
void imprime_matriz(long double **A, int n){
    int i;
    int j;

    for (i = 0; i < n; i++){
        for (j = 0; j < n; j++)
            printf("%Lf\t", A[i][j]);
        printf("\n");
    }
}

/**
 * Função para imprimir um vetor de forma mais legível
 * @param v vetor a ser impresso
 * @param n tamanho do vetor
```

```
 */
void imprime_vetor(long double *v, int n){
    int i;

    for (i = 0; i < n; i++)
        printf("%.16Lf\n", v[i]);
}

/**
 * Função para retornar a norma infinita de um vetor obtido pela subtração
 * de dois vetores
 * @param  xk Vetor x(k+1)
 * @param  x  Vetor x(k)
 * @param  n  Dimensão dos vetores
 * @return    norma do vetor obtido pela subtração
 */
long double norma_infinita(long double *xk, long double *x, int n){
    int i;
    long double maximo = 0;

    for(i = 0; i < n; i++)
        if(fabs(xk[i] - x[i]) > maximo)
            maximo = fabs(xk[i] - x[i]);

    return maximo;
}

/**
 * Função para resolver o sistema linear usando o método de gauss-seidel
 * @param A     Matriz de funções do sistema linear
 * @param b     Resultados das equações do sistema linear
 * @param x     Vetor contendo os resultados iniciais
 * @param n     Dimensão do sistema linear
 * @param e     Precisão
 * @param itmax Número máximo de iterações
 */
long double *gauss_seidel(long double **A, long double *b, long double *x, int n, long
↪  double e, int itmax){
    // Variáveis auxiliares
    long double somaL;
    long double somaU;
    long double *x_ant;

    // Iteradores
    int i;
    int j;
    int it = 0;

    // Alocando espaço da memória para o vetor auxiliar
    x_ant = malloc(sizeof(long double) * n);

    do{
        // Para toda iteração, atualiza o vetor x anterior
        for(i = 0; i < n; i++)
            x_ant[i] = x[i];

        for(i = 0; i < n; i++){
            somaL = 0;
            somaU = 0;
            // Somatório da parte de baixo da matriz
```

```
            for(j = 0; j < i; j++)
                somaL += A[i][j] * x[j];
            // Somatório da parte de cima da matriz
            for(j = i + 1; j < n; j++)
                somaU += A[i][j] * x[j];
            // Aproximação do x
            x[i] = (b[i] - somaL - somaU )/A[i][i];
        }

        // Calcula a norma infinita e compara com a tolerância
        if(norma_infinita(x, x_ant, n) <= e){
            free(x_ant);
            return x;
        }

        it++;
    }while(it < itmax);

    // Libera memória
    free(x_ant);
    return x;
}
```

2.3 Programa auxiliar (gera_input.c)

```
/**
 *    Trabalho 1 -  Métodos Iterativos para  Sitemas Lineares
 *
 *        Cálculo Numérico    SME-0104
 *        Prof.: Murilo Francisco Tomé
 *
 *        Eduardo Brunaldi dos Santos             8642515
 *        Jorge Ashkar Ferreira Simondi           8517081
 *        Victor Luiz da Silva Mariano Pereira    8602444
 */

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

/**
 * Programa feito para gerar parte do input utilizado no trabalho
 * com ele é possível gerar a matriz A solicitada, o vetor b (que
 * dependendo da letra do enunciado é diferente), o vetor x com o
 * chute inicial 0, número máximo de iterações e a tolerância de erro.
 * O programa sempre imprimirá mensagem de como usar se caso não for
 * usado corretamente.
 * Se usado corretamente, ele sempre imprimirá na sequência:
 *            n
 *            A (um elemento por linha)
 *            b (um elemento por linha)
 *            x (um elemento por linha)
 *            e (precisão)
 *            itmax (número máximo de iterações)
 *
 * para facilitar, recomendo que use da seguinte forma:
 *
 *            ./gera_input n exercicio
 *
```

```
 * sendo que n é a dimensão da matriz, dos vetores x e do vetor b. Já a
 * variável exercicio pode assumiros seguintes valores:
 *                          0 -> para gerar exemplo para o item b) do trabalho
 *                        e 1 -> para gerar exemplo para o item c) do trabalho
 *
 */

/**
 * Função que gera uma matriz pentadiagonal como a descrita no enunciado
 * do trabalho
 * @param  n Dimensão da matriz
 * @return   Matriz alocada e com valores setados
 */
int **gera_matriz_A(int n){
    int i;
    int j;

    int **A;

    // Alocando memória
    A = malloc(sizeof(int *) * n);
    for(i = 0; i < n; i++)
        A[i] = malloc(sizeof(int) * n);

    for(i = 0; i < n; i++)
        for(j = 0; j < n; j++){
            // Caso da diagonal principal
            if(i == j)
                A[i][j] = 4;
            // Caso das diagonais especiais
            else if(j == i+1 || i == j+1 || j == i+3 || i == j+3)
                A[i][j] = -1;
            // Caso do resto
            else
                A[i][j] = 0;
        }

    return A;
}

/**
 * Função que gera o vetor b de acordo com a letra b) do enunciado do
 * trabalho
 * @param  A Matriz base para a criação do vetor b
 * @param  n Tamanho do vetor
 * @return   Vetor b com os valores solicitados
 */
int *gera_vetor_b_b(int **A, int n){
    int i;
    int j;
    int *b;

    b = calloc(sizeof(int), n);

    for(i = 0; i < n; i++)
        for(j = 0; j < n; j++)
            b[i] += A[i][j];

    return b;
}
```

```
94
95  /**
96   * Função que gera o vetor b de acordo com a letra c) do enunciado do
97   * trabalho
98   * @param  n Tamanho do vetor
99   * @return   Vetor b com os valores solicitados
100  */
101 long double *gera_vetor_b_c(int n){
102     int i;
103     long double *b;
104
105     b = malloc(sizeof(int)* n);
106
107     for(i = 0; i < n; i++)
108         b[i] = 1.0/((i+1)*1.0);
109
110     return b;
111 }
112
113 /**
114  * Função para imprimir um vetor de inteiros, com um elemento por linha
115  * @param v Vetor a ser impresso
116  * @param n Dimensão do vetor
117  */
118 void imprime_vetor(int *v, int n){
119     int i;
120
121     for(i = 0; i < n; i++)
122         printf("%d\n", v[i]);
123 }
124
125 /**
126  * Função para imprimir uma matriz para usar como input de outro programa
127  * @param A Matriz a ser impressa
128  * @param n dimensão da matriz
129  */
130 void imprime_matriz(int **A, int n){
131     int i;
132     int j;
133
134     for (i = 0; i < n; i++)
135         imprime_vetor(A[i], n);
136 }
137
138 /**
139  * Função para imprimir um vetor para usar como input de outro programa
140  * no caso o vetor é da forma do item c) do trabalho
141  * @param b vetor a ser impresso
142  * @param n dimensão do vetor
143  */
144 void imprime_vetor_c(long double *b, int n){
145     int i;
146
147     for(i = 0; i < n; i++)
148         printf("%.16Lf\n", b[i]);
149 }
150
151 int main(int argc, char *argv[]){
152     // Variáveis
153     int *x;
```

```c
    int *bb;
    int **A;
    long double *bc;
    int ex;
    int n;

    // Iteradores
    int i;

    if(argc != 3){
        printf("Usage: ./programa valor_de_n exercicio, sendo que o exercicio pode ter valores\n\t0 -> b\n\t1 -> c\n");
        return -1;
    }

    n = atol(argv[1]);
    ex = atol(argv[2]);

    if(ex > 1 && ex < 0 || n < 1){
        printf("Usage: ./programa valor_de_n exercicio, sendo que o exercicio pode ter valores\n\t0 -> b\n\t1 -> c\n");
        return -1;
    }

    A = gera_matriz_A(n);
    printf("%d\n", n);
    imprime_matriz(A, n);

    // Como o chute inicial é sempre o vetor nulo, podemos alocar com zeros
    x = calloc(sizeof(int), n);

    if(ex == 0){
        bb = gera_vetor_b_b(A, n);
        imprime_vetor(bb, n);

        imprime_vetor(x, n);

        printf("0.00001\n");
        printf("10000000\n");

        // Liberando memória
        free(bb);
    } else if(ex == 1){
        bc = gera_vetor_b_c(n);
        imprime_vetor_c(bc, n);

        imprime_vetor(x, n);

        printf("0.0000000001\n");
        printf("10000000\n");

        // Liberando memória
        free(bc);
    } else{
        // Liberando memória
        for(i = 0; i < n; i++)
            free(A[i]);
        free(A);
        free(x);
        return -1;
```

```
212     }
213
214     // Liberando memória
215     for(i = 0; i < n; i++)
216         free(A[i]);
217     free(A);
218     free(x);
219
220     return 0;
221 }
```

Considerações finais