Eduardo Brunaldi dos Santos — 8642515,
Jorge Ashkar Ferreira Simondi — 8517081,
Victor Luiz da Silva Mariano Pereira — 8602444

Trabalho 1
Métodos Iterativos para Sistemas Lineares

Brasil

2018

Eduardo Brunaldi dos Santos — 8642515,
Jorge Ashkar Ferreira Simondi — 8517081,
Victor Luiz da Silva Mariano Pereira — 8602444

Trabalho 1
Métodos Iterativos para Sistemas Lineares

Universidade de São Paulo – USP
Instituto de Ciências Matemáticas e de Computação – ICMC
Cálculo Numérico – SME0104

Professor Murilo Francisco Tomé

Brasil
2018

Conteúdo

Introdução

Métodos iterativos, em cálculo numérico, são utilizados para calcular de forma progressiva a solução de um sistema de equações lineares. Assim podemos achar a solução mais correta possível, conforme nossas escolhas de precisão ou de número máximo de iterações.

No caso desse trabalho, temos como objetivo implementar o método iterativo de Gauss-Seidel, este baseado no método de Jacobi, os quais se diferenciam somente na utilização dos valores já obtidos na procura dos próximos.

O método de Jacobi utiliza sempre os valores da etapa anterior, já o de Gauss-Seidel, utiliza os valores mais atualizados possíveis.

1 Método Iterativo de Gauss-Seidel

O método de Gauss-Seidel é um método iterativo baseado no método de Jacobi desenvolvido para resolver sistemas de equações lineares. Seu nome foi originado nos matemáticos Carl Friedrich Gauss e Philipp Ludwig von Seidel.

Diferente do seu precursor, o método de Gauss-Seidel se aproveita de resultados de iterações passadas para acelerar a convergência para a resposta do sistema linear.

Podemos fazer um comparativo visual entre as duas funções de iteração, primeiramente a de Jacobi:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1 j \neq i}^{n} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, n$$

Já o método de Gauss-Seidel possui a seguinte fórmula de iteração:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, n$$

Assim podemos ver que por se utilizar das iterações anteriores, onde o método proposto para esse trabalho converge mais rápido que o de Jacobi.

Esses dois métodos são indicados para aplicações onde a matriz A é esparsa, ou seja, quando possui uma grande quantidade de elementos que valem zero. Para a utilização do método, temos que aplicar o critério de Sassenfeld e, claro, a matriz A tem que ser não-singular.

2 Resultados

Apesar da precisão ϵ ser especificada em alguns casos, em outros não, o programa que nós fizemos sempre vai imprimir o resultado encontrado com 16 casas decimais. Outro ponto a se observar, é que na apresentação dos resultados mostraremos os componentes do vetor x obtidos, fizemos isso por questão de estética e ter que colocar um vetor grande com 50 a 100 elementos.

2.1 Teste 1

Nesse primeiro teste, foi solicitado que executássemos nosso programa com os seguintes dados:

- $n = 50$
- Regra de formação de A é denotada por:

$$\begin{cases} a_{i,i} = 4, & i = 1, 2, \ldots, n; \\ a_{i,i+1} = -1, & i = 1, 2, \ldots, n-1; \\ a_{i+1,i} = -1, & i = 1, 2, \ldots, n-1; \\ a_{i,i+3} = -1, & i = 1, 2, \ldots, n-3; \\ a_{i+3,i} = -1, & i = 1, 2, \ldots, n-3; \\ a_{i,j} = 0, & \text{no restante.} \end{cases}$$

- $b_i = \sum_{j=1}^{n} a_{ij}, \quad i = 1, 2, \ldots, n$
- A tolerância de erro e a quantidade de iterações máxima não foram especificadas, então deixamos como padrão $\epsilon = 10^{-5}$ e $itmax = 10^7$.

Com 429 iterações, conseguimos o seguinte vetor x como resultado:

$$\begin{aligned}
x_1 &= 0.9999337056119755 & x_7 &= 0.9997426561581359 & x_{13} &= 0.9995850658671707 \\
x_2 &= 0.9999022580617377 & x_8 &= 0.9997127717351563 & x_{14} &= 0.9995641285500906 \\
x_3 &= 0.9998767564132660 & x_9 &= 0.9996847662462976 & x_{15} &= 0.9995449525304450 \\
x_4 &= 0.9998373110649970 & x_{10} &= 0.9996574884419451 & x_{16} &= 0.9995275349651083 \\
x_5 &= 0.9998043818757397 & x_{11} &= 0.9996317401867282 & x_{17} &= 0.9995119448425527 \\
x_6 &= 0.9997745311874137 & x_{12} &= 0.9996076670815898 & x_{18} &= 0.9994982360680523
\end{aligned}$$

$x_{19} = 0.9994864324485824$ $x_{20} = 0.9994765653157160$ $x_{21} = 0.9994686592137438$ $x_{22} = 0.9994627261862783$ $x_{23} = 0.9994587736375708$ $x_{24} = 0.9994568022326934$ $x_{25} = 0.9994568043937440$ $x_{26} = 0.9994587659437409$ $x_{27} = 0.9994626656965930$ $x_{28} = 0.9994684755790514$ $x_{29} = 0.9994761612548965$

$x_{30} = 0.9994856808399818$ $x_{31} = 0.9994969866366955$ $x_{32} = 0.9995100271367293$ $x_{33} = 0.9995247400284760$ $x_{34} = 0.9995410581501044$ $x_{35} = 0.9995589220334093$ $x_{36} = 0.9995782457760340$ $x_{37} = 0.9995989330477007$ $x_{38} = 0.9996209526818133$ $x_{39} = 0.9996441831497855$ $x_{40} = 0.9996684275186901$

$x_{41} = 0.9996938378919464$ $x_{42} = 0.9997202487630036$ $x_{43} = 0.9997469483563225$ $x_{44} = 0.9997749237180661$ $x_{45} = 0.9998042064202979$ $x_{46} = 0.9998313692296440$ $x_{47} = 0.9998608077401657$ $x_{48} = 0.9998952663788790$ $x_{49} = 0.9999175540360030$ $x_{50} = 0.9999445904440422$

2.2 Teste 2

No segundo teste, foi solicitado que executássemos nosso programa com os seguintes dados:

- $n = 100$
- Regra de formação de A é denotada por:

$$\begin{cases} a_{i,i} = 4, & i = 1, 2, \ldots, n; \\ a_{i,i+1} = -1, & i = 1, 2, \ldots, n-1; \\ a_{i+1,i} = -1, & i = 1, 2, \ldots, n-1; \\ a_{i,i+3} = -1, & i = 1, 2, \ldots, n-3; \\ a_{i+3,i} = -1, & i = 1, 2, \ldots, n-3; \\ a_{i,j} = 0, & \text{no restante.} \end{cases}$$

- $b_i = \sum_{j=1}^{n} a_{ij}, \quad i = 1, 2, \ldots, n$
- A tolerância de erro e a quantidade de iterações máxima não foram especificadas, então deixamos como padrão $\epsilon = 10^{-5}$ e $itmax = 10^7$.

Com 1357 iterações, conseguimos o seguinte vetor x como resultado:

$x_1 = 0.9998738542396985$ $x_2 = 0.9998129435623880$ $x_3 = 0.9997626042371170$

$x_4 = 0.9996848479263462$
$x_5 = 0.9996182432345047$
$x_6 = 0.9995562138527778$
$x_7 = 0.9994889115229971$
$x_8 = 0.9994237903157294$
$x_9 = 0.9993605364284126$
$x_{10} = 0.9992968430669336$
$x_{11} = 0.9992341148124684$
$x_{12} = 0.9991725355247364$
$x_{13} = 0.9991116311073441$
$x_{14} = 0.9990516865321048$
$x_{15} = 0.9989928222278777$
$x_{16} = 0.9989349787055476$
$x_{17} = 0.9988782439532933$
$x_{18} = 0.9988226902835039$
$x_{19} = 0.9987683448745465$
$x_{20} = 0.9987152612888901$
$x_{21} = 0.9986634928002946$
$x_{22} = 0.9986130812852338$
$x_{23} = 0.9985640720869986$
$x_{24} = 0.9985165101388453$
$x_{25} = 0.9984704367415823$
$x_{26} = 0.9984258924917748$
$x_{27} = 0.9983829168228631$
$x_{28} = 0.9983415471628708$
$x_{29} = 0.9983018194640830$
$x_{30} = 0.9982637681626284$
$x_{31} = 0.9982274259594643$
$x_{32} = 0.9981928238934358$
$x_{33} = 0.9981599913289189$
$x_{34} = 0.9981289558893802$
$x_{35} = 0.9980997434513928$
$x_{36} = 0.9980723781293944$
$x_{37} = 0.9980468822474435$
$x_{38} = 0.9980232763240508$
$x_{39} = 0.9980015790579967$
$x_{40} = 0.9979818073123535$
$x_{41} = 0.9979639761023460$
$x_{42} = 0.9979480985851344$
$x_{43} = 0.9979341860505684$
$x_{44} = 0.9979222479140781$
$x_{45} = 0.9979122917114375$
$x_{46} = 0.9979043230951400$
$x_{47} = 0.9978983458326130$
$x_{48} = 0.9978943618062428$
$x_{49} = 0.9978923710151436$
$x_{50} = 0.9978923715787057$
$x_{51} = 0.9978943597419185$
$x_{52} = 0.9978983298824433$
$x_{53} = 0.9979042745194355$
$x_{54} = 0.9979121843241067$
$x_{55} = 0.9979220481320004$
$x_{56} = 0.9979338529569851$
$x_{57} = 0.9979475840069450$
$x_{58} = 0.9979632247011112$
$x_{59} = 0.9979807566890773$
$x_{60} = 0.9980001598714995$
$x_{61} = 0.9980214124222428$
$x_{62} = 0.9980444908122079$
$x_{63} = 0.9980693698350556$
$x_{64} = 0.9980960226336919$
$x_{65} = 0.9981244207284375$
$x_{66} = 0.9981545340486365$
$x_{67} = 0.9981863309622862$
$x_{68} = 0.9982197783064963$
$x_{69} = 0.9982548414300648$
$x_{70} = 0.9982914842231527$
$x_{71} = 0.9983296691384520$
$x_{72} = 0.9983693572692299$
$x_{73} = 0.9984105083744410$
$x_{74} = 0.9984530808251019$
$x_{75} = 0.9984970318264450$
$x_{76} = 0.9985423174643067$
$x_{77} = 0.9985888922234198$
$x_{78} = 0.9986367098011544$
$x_{79} = 0.9986857235278032$
$x_{80} = 0.9987358836422807$
$x_{81} = 0.9987871403698153$
$x_{82} = 0.9988394474634000$
$x_{83} = 0.9988927483413111$
$x_{84} = 0.9989469867000615$
$x_{85} = 0.9990021309945048$
$x_{86} = 0.9990581078225971$
$x_{87} = 0.9991148304313648$
$x_{88} = 0.9991723487039626$
$x_{89} = 0.9992305448790375$
$x_{90} = 0.9992891527897000$
$x_{91} = 0.9993486065323530$
$x_{92} = 0.9994087268533902$
$x_{93} = 0.9994682235509935$
$x_{94} = 0.9995292193590008$
$x_{95} = 0.9995919759926195$
$x_{96} = 0.9996496775214918$
$x_{97} = 0.9997113652847045$
$x_{98} = 0.9997829628126369$
$x_{99} = 0.9998293169924293$
$x_{100} = 0.9998851705692834$

2.3 Teste 3

Já no terceiro teste solicitado, diferente dos anteriores, foram especificados os valores de ϵ e de $itmax$, ficando da seguinte maneira:

- $n = 100$
- Regra de formação de A é denotada por:

$$\begin{cases} a_{i,i} = 4, & i = 1, 2, \ldots, n; \\ a_{i,i+1} = -1, & i = 1, 2, \ldots, n-1; \\ a_{i+1,i} = -1, & i = 1, 2, \ldots, n-1; \\ a_{i,i+3} = -1, & i = 1, 2, \ldots, n-3; \\ a_{i+3,i} = -1, & i = 1, 2, \ldots, n-3; \\ a_{i,j} = 0, & \text{no restante.} \end{cases}$$

- $b_i = 1.0/i, \quad i = 1, 2, \ldots, n$
- $\epsilon = 10^{-10}$
- $itmax = 10^7$

Com 4035 iterações, conseguimos o seguinte vetor x como resultado:

$$\begin{array}{lll}
x_1 = 0.9187448165907161 & x_{15} = 3.0827575717446547 & x_{29} = 3.8140332375616510 \\
x_2 = 1.1045965699783248 & x_{16} = 3.1682320874339032 & x_{30} = 3.8352664160698222 \\
x_3 = 1.2325874938164970 & x_{17} = 3.2474823074568488 & x_{31} = 3.8531706946495210 \\
x_4 = 1.5703826964083346 & x_{18} = 3.3207350197423259 & x_{32} = 3.8678532579607074 \\
x_5 = 1.7670539695354222 & x_{19} = 3.3885251582887751 & x_{33} = 3.8794144237793418 \\
x_6 = 1.9220373755819542 & x_{20} = 3.4510802518712860 & x_{34} = 3.8879486080854190 \\
x_7 = 2.1131445057330069 & x_{21} = 3.5086194860451511 & x_{35} = 3.8935446760295326 \\
x_8 = 2.2711992362214035 & x_{22} = 3.5614216952868021 & x_{36} = 3.8962863828783725 \\
x_9 = 2.4086968666307383 & x_{23} = 3.6096940558257294 & x_{37} = 3.8962528733599743 \\
x_{10} = 2.5461015719238098 & x_{24} = 3.6536219297971451 & x_{38} = 3.8935190268071943 \\
x_{11} = 2.6709016030530332 & x_{25} = 3.6933935356735150 & x_{39} = 3.8881557807540906 \\
x_{12} = 2.7843381717571969 & x_{26} = 3.7291740856229907 & x_{40} = 3.8802304488311668 \\
x_{13} = 2.8916633123568407 & x_{27} = 3.7611139751266089 & x_{41} = 3.8698069878026507 \\
x_{14} = 2.9910583414847695 & x_{28} = 3.7893564321402836 & x_{42} = 3.8569462390514479
\end{array}$$

$x_{43} = 3.8417061536023001$	$x_{63} = 3.1075299149135004$	$x_{83} = 1.7275545421499002$
$x_{44} = 3.8241419928140214$	$x_{64} = 3.0521558236288166$	$x_{84} = 1.6448165338014529$
$x_{45} = 3.8043065104340237$	$x_{65} = 2.9952196902850702$	$x_{85} = 1.5608636785986605$
$x_{46} = 3.7822501199568167$	$x_{66} = 2.9367455313930225$	$x_{86} = 1.4757416018946408$
$x_{47} = 3.7580210468883337$	$x_{67} = 2.8767566370039185$	$x_{87} = 1.3895174593800130$
$x_{48} = 3.7316654678906484$	$x_{68} = 2.8152756070057247$	$x_{88} = 1.3020337388756733$
$x_{49} = 3.7032276386672660$	$x_{69} = 2.7523243683867061$	$x_{89} = 1.2134028205516427$
$x_{50} = 3.6727500113467956$	$x_{70} = 2.6879242130190810$	$x_{90} = 1.1239837101476174$
$x_{51} = 3.6402733424392654$	$x_{71} = 2.6220958501560915$	$x_{91} = 1.0329873606760617$
$x_{52} = 3.6058367924089371$	$x_{72} = 2.5548593686640197$	$x_{92} = 0.9406162762941549$
$x_{53} = 3.5694780175913818$	$x_{73} = 2.4862342823983469$	$x_{93} = 0.8489160889277005$
$x_{54} = 3.5312332551719267$	$x_{74} = 2.4162397050438427$	$x_{94} = 0.7543267064481112$
$x_{55} = 3.4911374019048185$	$x_{75} = 2.3448940849725689$	$x_{95} = 0.6562892698483521$
$x_{56} = 3.4492240871063841$	$x_{76} = 2.2722152128718184$	$x_{96} = 0.5659849746877800$
$x_{57} = 3.4055257404294229$	$x_{77} = 2.1982210892729214$	$x_{97} = 0.4684758085009985$
$x_{58} = 3.3600736549446078$	$x_{78} = 2.1229287201109216$	$x_{98} = 0.3537028062007704$
$x_{59} = 3.3128980458356888$	$x_{79} = 2.0463535002357254$	$x_{99} = 0.2798420648291652$
$x_{60} = 3.2640281050369084$	$x_{80} = 1.9685137062025785$	$x_{100} = 0.1895794683325409$
$x_{61} = 3.2134920524875769$	$x_{81} = 1.8894256932616957$	
$x_{62} = 3.1613171838998899$	$x_{82} = 1.8090981340243647$	

Conclusão

Apêndices

APÊNDICE A – Códigos Fonte

A.1 Programa 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++)
```

```c
            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;
}
```

A.2 Biblioteca auxiliar

A.2.1 *Header* da biblioteca (gauss_seidel.h)

```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
 */

#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);
```

```c

#endif
```

A.2.2 Implementação da biblioteca (gauss_seidel.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 <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;

```

```c
    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){
            // Se quiser imprimir o número de iterações necessários para
            // chegar na solução encontrada, só descomentar o próximo
            // comando 'printf()'. Se caso não imprimir o número de
            // iterações mesmo descomentando o comando, quer dizer que
            // o it chegou ao itmax.

            // printf("Numero de iteracoes: %d\n", it);

            free(x_ant);
            return x;
        }

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

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

A.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;
}

/**
 * Função que gera o vetor b de acordo com a letra c) do enunciado do
 * trabalho
```

```c
 * @param  n Tamanho do vetor
 * @return   Vetor b com os valores solicitados
 */
long double *gera_vetor_b_c(int n){
    int i;
    long double *b;

    b = malloc(sizeof(int)* n);

    for(i = 0; i < n; i++)
        b[i] = 1.0/((i+1)*1.0);

    return b;
}

/**
 * Função para imprimir um vetor de inteiros, com um elemento por linha
 * @param v Vetor a ser impresso
 * @param n Dimensão do vetor
 */
void imprime_vetor(int *v, int n){
    int i;

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

/**
 * Função para imprimir uma matriz para usar como input de outro programa
 * @param A Matriz a ser impressa
 * @param n dimensão da matriz
 */
void imprime_matriz(int **A, int n){
    int i;
    int j;

    for (i = 0; i < n; i++)
        imprime_vetor(A[i], n);
}

/**
 * Função para imprimir um vetor para usar como input de outro programa
 * no caso o vetor é da forma do item c) do trabalho
 * @param b vetor a ser impresso
 * @param n dimensão do vetor
 */
void imprime_vetor_c(long double *b, int n){
```

```c
    int i;

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

int main(int argc, char *argv[]){
    // Variáveis
    int *x;
    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;
    }

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

    return 0;
}
```