Projeto Final Técnicas Computacionais

UERJ - Mestrado em Modelagem Computacional

Lucas Magalhães Espinosa Ferreira

11/07/2021

Contents

Resumo

O objetivo do presente trabalho é a resolução de um problema numérico Linear utilizando o método de Gradientes Conjugados. O algoritmo foi todo construído em Linguagem C usando o erro máximo Absoluto como critério de aprovação do modelo que foi solucionado após 10 iterações.

Palavras-chave: Métodos Numéricos, Gradientes Conjugados, Linguagem C.

Introdução

O problema

Estamos interessados em resolver o seguinte sistema Linear:

$Ax = d \quad (1)$

Onde:

A : É uma matriz de coeficientes quadrada esparsa de Ordem $\mathbb{R}^{25x25}$

x : É a Matriz de variáveis de Ordem $\mathbb{R}^{25x1}$

d : É a Matriz resposta de Ordem $\mathbb{R}^{25x1}$

Para este caso, faremos uso do Método dos Gradientes Conjugados que é um método iterativo e que parte da mesma idéia do Método dos Gradientes (minimizar uma função quadrática). Embora o tempo para convergência seja um pouco mais demorado, ele se mostra mais eficiente do que os métodos diretos ou de Gauss quando temos uma matriz esparsa.

Como no Método dos Gradientes, trocaremos o problema tradicional $Ax = b$ para um problema equivalente aonde devemos encontrar um minimizador de $1/2x^TAx - b^Tx$, com nossa matriz de coeficientes A simétrica definida positiva.

Dada a função $f(x) = 1/2x^TAx - b^Tx$ (2), devemos levar em conta que:

$\nabla f(x) = Ax - b; \quad (3)$

$\nabla^2 f(x) = A; \quad (4)$

Encontrar a solução para $Ax - b$ é o mesmo de encontrar o ponto x que satisfaz $\nabla f(x) = Ax - b = 0$, ou seja, o minimizador da função f.

Falaremos agora de forma geral sobre como o algoritmo foi estruturado e como foi estipulado o critério de aceite.

Metodologia

Para resolução do problema, o algoritmo foi construido em linguagem C a partir de um método iterativo que a cada passo é calculado o erro máximo absoluto da diferença entre matriz de variáveis x^{k+1} nova e a matriz de variáveis do passo anterior x^k a partir da seguinte fórmula:

$max|x^{k+1} - x^k| \leq \epsilon \quad (5)$

Onde ϵ é o erro máximo de aceite que em nosso caso está na ordem de 10^{-2}.

Primeiramente o algoritmo deve ter como premissa os argumentos iniciais de x^0, r^0 e p^0.

Para a matriz de variáveis inicial x^0 foi proposto valores iguais ao da matriz resposta d.

Logo:

$$d, x^0 = \begin{bmatrix} 100 \\ 100 \\ \vdots \\ 80 \\ \vdots \\ 100 \\ 100 \end{bmatrix}_{25x1}$$

r^0 foi calculado da seguinte forma:

$r^0 = d - Ax^0$ (6), Sendo A nossa matriz de coeficientes $25x25$;

Por último temos o valor de p^0 sendo $p^0 = r^0$ (7);

Até atingirmos o critério de aceite em **(5)** o algoritmo irá seguir os seguintes passos para cada iteração k:

Pseudo Código

Primeiro vamos calcular g^K_{25x1};

Passo 1: $g^k = Ap^k$; (8)

Com g^k podemos calcular o escalar a^k a partir da seguinte fórmula:

Passo 2: $a^k = r^k.r^k/p^k.g^k$; (9)

Agora com a^k podemos calcular o próximo x^{k+1}:

Passo 3: $x^{k+1} = x^k + a^k p^k$; (10)

Passo 4: Com x^{k+1} e x^k podemos calcular o erro máximo absoluto EMA e compararmos com ϵ.

Passo 5 : Se $\epsilon >= EMA$:

O algoritmo para com a nossa melhor estimativa x^{k+1} que minimiza f

Se não:

Devemos calcular os novos valores de r e k para próxima iteração:

Passo 6: $r^{k+1} = r^k - a^k g^k$; (11)

Passo 7: $b^{k+1} = r^{k+1}.r^{k+1}/r^k.r^k$; (12)

Passo 8: $p^{k+1} = r^{k+1} + b^{k+1}p^k$; (13)

E assim o algoritmo vai para $k = 1, 2, 3, ...$ até satisfazer o critério de aceite.

Antes de irmos para os resultados nós definimos A como uma matriz quadrada de ordem 25. Para essa matriz de coeficientes temos os seguintes valores para A_{ij}:

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21	V22	V23	V24	V25
-7.53	-6.25	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
-6.25	-13.78	-6.25	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	-6.25	-13.78	-6.25	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	-6.25	-13.78	-6.25	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	-6.25	-7.53	0.00	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
-2.78	0.00	0.00	0.00	0.00	-10.31	-6.25	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	-2.78	0.00	0.00	0.00	-6.25	-16.56	-6.25	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	-2.78	0.00	0.00	0.00	-6.25	-16.56	-6.25	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	-2.78	0.00	0.00	0.00	-6.25	-16.56	-6.25	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	-6.25	-10.31	0.00	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	0.00	-10.31	-6.25	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	-6.25	-16.56	-6.25	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	-6.25	-16.56	-6.25	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	-6.25	-16.56	-6.25	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	-6.25	-10.31	0.00	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	0.00	-10.31	-6.25	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	-6.25	-16.56	-6.25	0.00	0.00	0.00	-2.78	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	-6.25	-16.56	-6.25	0.00	0.00	0.00	-2.78	0.00	0.00

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21	V22	V23	V24	V25
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	-6.25	-16.56	-6.25	0.00	0.00	0.00	-2.78	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	-6.25	-10.31	0.00	0.00	0.00	0.00	-2.78
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	0.00	-7.53	-6.25	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	-6.25	-13.78	-6.25	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	-6.25	-13.78	-6.25	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	-6.25	-13.78	-6.25
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-2.78	0.00	0.00	0.00	-6.25	-7.53

Tabela 1: Matriz 25 x 25 de coeficientes A.

Resultados

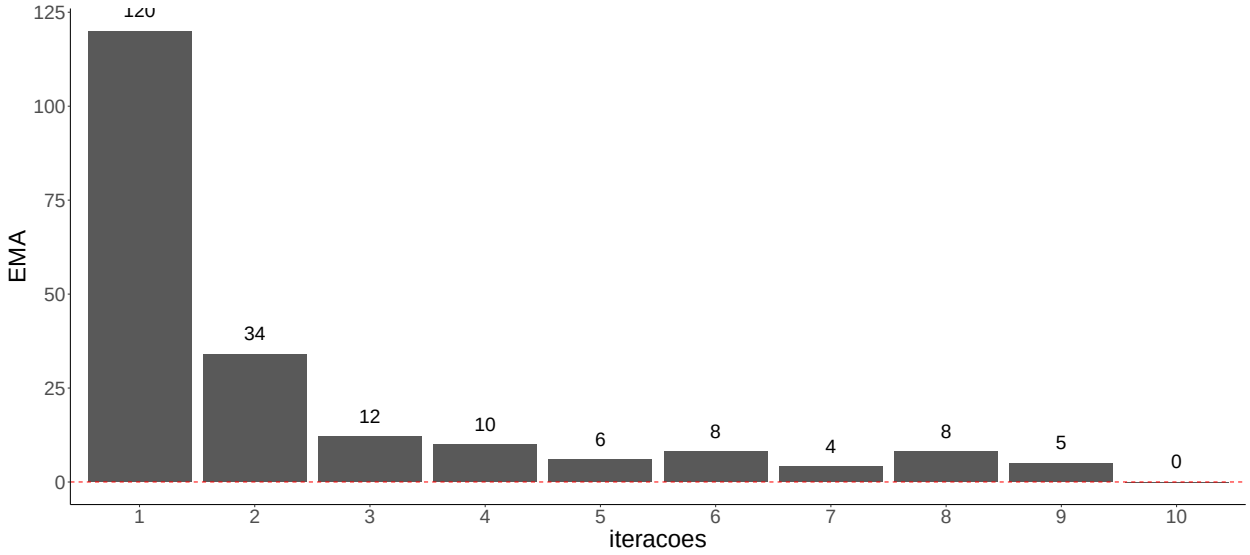

Figura 1: EMA por iteração.

Em cerca de 2000 microsegundos e 10 iterações o algoritmo conseguiu chegar com sucesso na minimização do EMA em 0, mostrando-se eficaz para o problema retratado. Para a matriz resposta x_{25x1}, tivemos os seguintes resultados:

$$x = \begin{bmatrix} -7.7905 \\ -2.6533 \\ -3.2880 \\ -2.6533 \\ -7.7905 \\ -8.9045 \\ 2.0874 \\ -7.7428 \\ 2.0874 \\ -8.9045 \\ 0.1499 \\ -8.3259 \\ 4.0533 \\ -8.3259 \\ 0.1499 \\ -8.9045 \\ 2.0874 \\ -7.7428 \\ 2.0874 \\ -8.9045 \\ -7.7905 \\ -2.6533 \\ -3.2880 \\ -2.6533 \\ -7.7905 \end{bmatrix}_{25x1}$$

Conclusão

O método se mostrou satisfatório para um sistema de ordem 25 mas poderíamos ter um desempenho pior para uma matriz esparsa maior, já que temos uma complexidade tempo-espaço de $O(n^2)$. Podemos concluir essa complexidade aproximada, já que os métodos principais são baseados em *for loops* aninhados (Principalmente 2 níveis). Para matrizes maiores digamos de ordem 1000 em diante, talvez esse método não seja o mais aconselhável.

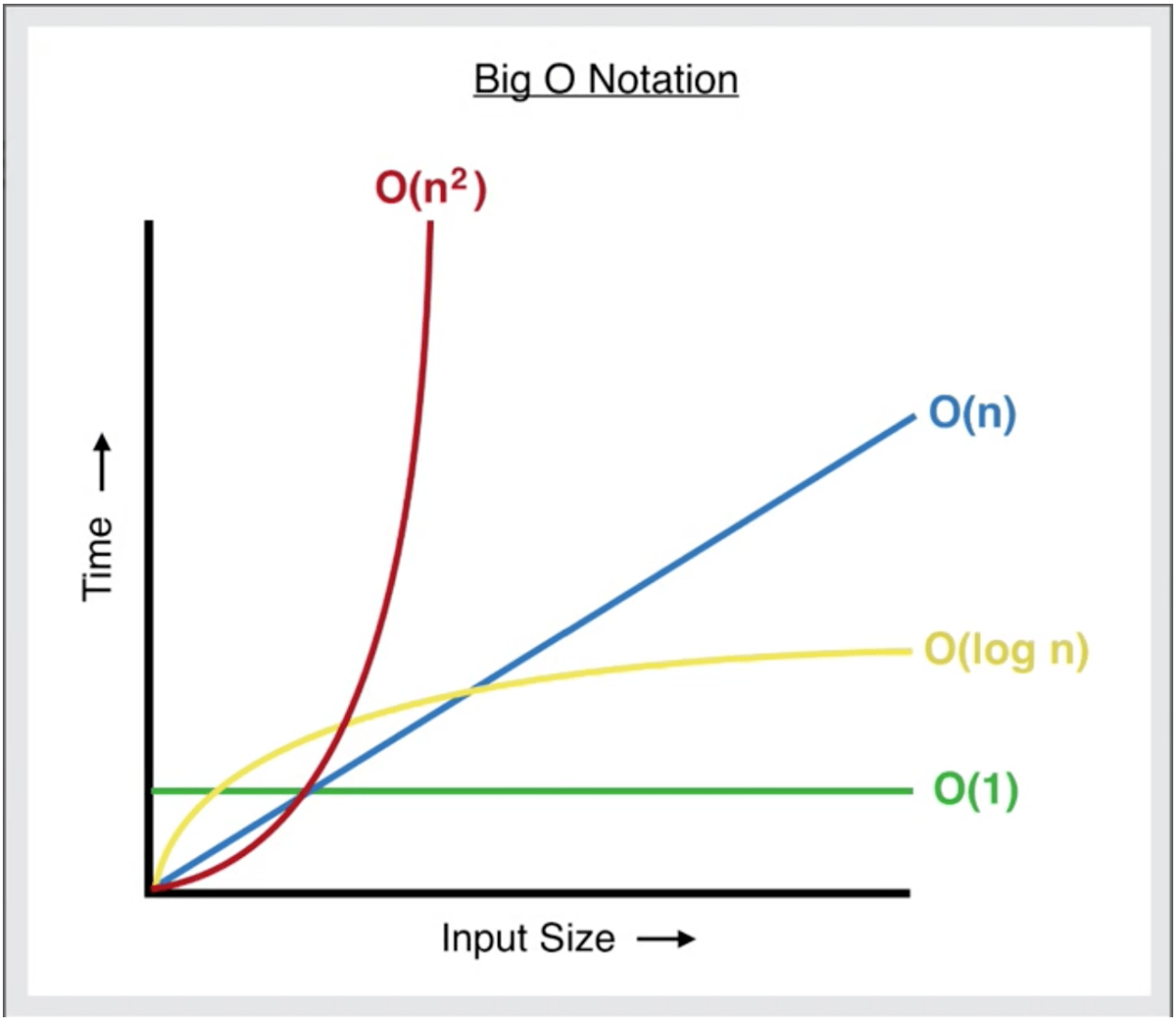

Figura 2: A famosa notação do "BIG O" comparando tamanho do input do algoritmo x Tempo de Processamento.

Bibliografia

- Shewchuk, J R.An Introduction to the Conjugate Gradient MethodWithout the Agonizing PainEdition 114. School of Computer ScienceCarnegie Mellon University Pittsburgh. PA, 1994.
- Franco, N B. Cálculo Numérico. Editora: Pearson / Prentice Hal. 2006.
- Herbert Schildt, C completo e total, Pearson/ Prentice Hal. 1997

Anexos

main.c

```
/*  Projeto Final
    Aluno: Lucas Magalhães Espinosa Ferreira
    Mestrado em Modelagem Computacional 2021/1     */

// inclusão das bibliotecas

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "functions.h"
// inclusao de variaveis globais
struct timeval stop, start;
// estrutura principal
int main() {

    read_matrix();
    // grava o time inicial
    gettimeofday(&start, NULL);
    create_init_var();
    run_model();
    //grava o time final
    gettimeofday(&stop, NULL);
    //printa a diferença em microsegundos
    printf("Levou %lu us\n", (stop.tv_sec - start.tv_sec) * 1000000 +
    stop.tv_usec - start.tv_usec);
    return 0;

}
```

functions.h

```
#ifndef __FUNCTIONS_H__
    #define __FUNCTIONS_H__
    #include <stdio.h>
    #include <stdlib.h>
    #include <math.h>

    //definiciao das variaveis globais
    // matriz de coeficientes A
    double matrix_a[25][25];
    // matriz para o x°
    double x_ini[25] = {100,100,100,100,100,100,100,100,100,100,100,100,
    80,100,100,100,100,100,100,100,100,100,100,100,100};
    //matriz d
    double ans_vector[25] = {100,100,100,100,100,100,100,100,100,100,100,100,
    80,100,100,100,100,100,100,100,100,100,100,100,100};
    //variaveis de iteração do método, x,r,p,g e epsilon (critério de aceite)
```

```
double x_next[25],r_ini[25],p_ini[25],g_ini[25],a_ini=0,epsilon=0.01;
//endereco do arquivo txt que contem a matriz
char matrix_a_address[50];
// outras variaveis usadas durante a iteracao
int i,j;
int iteration=1;

//metodo para ler a matrix do arquivo txt e salvar em matrix_a
//(o arquivo é o matrix_sparse.txt)
void read_matrix(){
    FILE *file;
    printf("Digite o caminho para o arquivo da matriz: \n");
    scanf("%s",&matrix_a_address);
    file = fopen(matrix_a_address, "r");
    if (file) {
        for(i=0;i<25;i++) {
            for(j=0;j<25;j++) {
                fscanf(file, "%lf,", &matrix_a[i][j]);
            }
        }
        fclose(file);
    }
}
//metodo para criar as variaveis iniciais r°,p°
void create_init_var(){
    float aux=0;
    float a_x[25];
    for (i=0;i<25;i++) {
        for (j=0;j<25;j++){
            aux += matrix_a[i][j] * x_ini[j];
        }
        a_x[i] = aux;
        aux = 0;
    }
    for (i=0;i<25;i++){
        r_ini[i] = ans_vector[i] - a_x[i];
        p_ini[i] = r_ini[i];
    }
}

// roda o metodo de gradientes conjugados
void run_model(){
    // variáveis auxiliares para calcular a tolerância
    float aux_error=0,aux_error2=0;
    //laço do-while principal que faz um check na tolerância
    do
    {
        // variáveis auxiliares para o calculo das variáveis
        float aux=0;
        float aux2=0;
        // for loop para calculo do g
        for (i=0;i<25;i++){
            for (j=0;j<25;j++){
```

```
            aux += matrix_a[i][j] * p_ini[j];
        }
        g_ini[i] = aux;
        aux = 0;
    }
    // for loop para o calculo do a
    for (i=0;i<25;i++){
        aux += r_ini[i] * r_ini[i];
        aux2 += p_ini[i] * g_ini[i];
    }
    a_ini = aux / aux2;
    // criacao do vetor auxiliar para calculo do proximo x
    float aux_vec[25]={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};
    for (i=0;i<25;i++){
        aux_vec[i] = a_ini * p_ini[i];
        x_next[i] = x_ini[i] + aux_vec[i];
    }
    // for loop para o calculo da tolerância
    for (i=0;i<25;i++){
        aux_error2 = abs(x_next[i] - x_ini[i]);
        if(i==0){
            aux_error = aux_error2;
            continue;
        }
        if(aux_error2 > aux_error) {
            aux_error = aux_error2;
        }
    }
    // for loop para o calculo do proximo r;
    float r_next[25]={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};
    for (i=0;i<25;i++){
        r_next[i] = r_ini[i] - (a_ini * g_ini[i]);

    }
    // for loop para o calculo do próximo b
    float b = 0;
    aux = 0, aux2 = 0;
    for (i=0;i<25;i++){
        aux += r_next[i] * r_next[i];
        aux2 += r_ini[i] * r_ini[i];
    }
    b = aux / aux2;
    //for loop para o calculo do proximo p
    float p_next[25]={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};
    for (i=0;i<25;i++){
    p_next[i] = r_next[i] + (b * p_ini[i]);
    }

    //reset das variaveis
    for (i=0;i<25;i++){
        x_ini[i] = x_next[i];
        r_ini[i] = r_next[i];
        p_ini[i] = p_next[i];
```

```
            }
            //print dos resultados obtidos
            printf("Resultado da %d iteração:\n",iteration);
            printf("Valor do Erro: %lf\n",aux_error);
            printf("Vetor de variáveis resposta:\n");
            printf("{");
            for(j=0;j<25;j++) {
                printf(" ");
                printf("%.4f",x_next[j]);
                printf(" ");
            }
            printf("}\n");
            iteration += 1;
        }while (aux_error > epsilon);
    }



#endif
```