Álgebra Linear Computacional

Trabalho Prático 1

Antonny Victor da Silva, DRE: 120031917

16 de maio de 2023

1 Introdução

Este relatório consiste na apresentação da implementação do Trabalho Prático 1 da disciplina de Álgebra Linear Computacional (ALC), contendo seções explicativas sobre as tomadas de decisão, casos de teste e explicações de partes centrais do código que pode ser encontrado aqui mesmo nesse relatório ou no link abaixo.
Código-Fonte: `https://github.com/antonnyvictor18/Algebra_Linear`

2 Escolha da Linguagem de Programação

Como trata-se de um programa que precisará ter um bom desempenho de tempo e memória para fazer várias operações de alta complexidade, optei por fazer em C++. Essa linguagem de programação oferece mais controle sobre o gerenciamento de memória utilizada no meu sistema bem como operações rápidas em relação a muitas outras opções de linguagem.

3 Arquivo Utils

Visando uma melhor organização e um melhor entendimento do código, resolvi criar uma biblioteca de funções necessárias para fazer as operações com as matrizes e com os vetores, como: multiplicar,imprimir, decompor matrizes etc. Segue abaixo todo o código do Utils.h:

```
#ifndef UTILS_H
#define UTILS_H
#include <iostream>
#include <fstream>
#include <vector>
#include <iomanip>
#include <cmath>
using namespace std;

void identidade(vector<vector<double>> &P, int &n){
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j) {
                P[i][j] = 1.0;
            }
            else {
                P[i][j] = 0.0;
            }
        }
    }
}


void imprimirVetor(vector<double>&x){
    cout << "[";
```

```
    for (int i = 0; i < x.size(); i++) {
        cout << x[i] << ", ";
    }
    cout << "]" << endl;
}

void imprimirMatriz(vector<vector<double>>& matriz) {
    int n = matriz.size();
    int m = matriz[0].size();
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cout << matriz[i][j] << " ";
        }
        cout << "\n";
    }
}

void lerVetor(vector<double> &vetor, int &n, string &arquivo){
    ifstream fin(arquivo);
    char ch;
    string last_ch = " ";
    bool negativo = false;
    int contador= 0;

    while(fin.get(ch)){
        if (ch == ' '){
            if(last_ch == " "){
                continue;
            }
            else if (last_ch != " "){
                if (negativo){
                    last_ch = '-' + last_ch;
                }

                vetor[contador] = stod(last_ch);
                last_ch = ch;
                negativo = false;
                contador++;
                continue;
            }

        }

        else if (ch == '-'){
            negativo =true;
            continue;
        }
```

```
        else if (ch != ' '){
            if (last_ch == " "){
                last_ch = ch;
                continue;
            }

            else if (last_ch != " "){
                last_ch = last_ch + ch;
                continue;
            }
        }
    }
    vetor[contador] = stod(last_ch);
}

void lerMatriz(vector<vector<double>> &A, int &n, string &arquivo){
 ifstream fin(arquivo);
 char ch;
 string last_ch = " ";
 bool negativo = false;
 int contador = 0;
 int m = n*n;
 vector<double> vetor(m,0.0);

 while(fin.get(ch)){
    if (ch == ' '){
        if( last_ch == " "){
            continue;
        }
        else if (last_ch != " "){
            if (negativo){
                last_ch = '-' + last_ch + ".0";
            }


            vetor[contador] = stod(last_ch);
            last_ch = ch;
            negativo = false;
            contador++;
            continue;
        }

    }

    else if (ch == '-'){
        negativo =true;
        continue;
    }
```

```
    else if (ch != ' '){
        if (last_ch == " "){
            last_ch = ch;
            continue;
        }

        else if (last_ch != " "){
            last_ch = last_ch + ch;
            continue;
        }


    }

    }
    vetor[contador] = stod(last_ch);
    contador = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            A[i][j] = vetor[contador];
            contador++;
        }
    }
}


vector<double> prodMatVec(vector<vector<double>> &A, vector<double> &x) {
    int n = A.size();
    vector<double> y(n,0);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            y[i] += A[i][j] * x[j];
        }
    }
    return y;
}

void imprimeAutovaloreeAutovetores(vector<vector<double>>&A,
    vector<vector<double>>&V, int &n){
    for (int i = 0; i < n; i++) {
    cout << "Autovalor " << i+1 << ": " << A[i][i] << endl;
    cout << "Autovetor " << i+1 << ": ";
        for (int j = 0; j < n; j++) {
            cout << V[j][i] << " ";
        }
    cout << endl << endl;
    }
```

```
}



vector<vector<double>> decomposicaoCholesky(vector<vector<double>>& A,
    vector<vector<double>>& L, int n) {

    for (int i = 0; i < n; i++) {
        for (int j = 0; j <= i; j++) {
            double s = 0;
            for (int k = 0; k < j; k++) {
                s += L[i][k] * L[j][k];
            }
            if (i == j) {
                L[i][j] = sqrt(A[i][i] - s);
            } else {
                L[i][j] = (1.0 / L[j][j]) * (A[i][j] - s);
            }
        }
    }

    return L;
}


void decomposicaoLU(vector<vector<double>>& A, vector<vector<double>>& L,
    int n) {

    for (int k = 0; k < n; k++) {
        L[k][k] = 1.0;
        for (int i = k + 1; i < n; i++) {
            L[i][k] = A[i][k] / A[k][k];
            for (int j = k; j < n; j++) {
                A[i][j] = A[i][j] - L[i][k] * A[k][j];
            }
        }
    }

}

bool converge(vector<vector<double>>&A){
    int n = A.size();
    double soma_linha, soma_coluna;
    for (int i = 0; i < n; i++){
        soma_linha = 0.0;
        soma_coluna = 0.0;

        for(int j = 0; j < n; j++ ){
```

```
            if(j == i){
                continue;
            }

            soma_linha += abs(A[i][j]);
            soma_coluna += abs(A[j][i]);
        }

        if(soma_linha > abs(A[i][i]) or soma_coluna > abs(A[i][i]) ){
            return false;
        }
    }
    return true;
}

vector<double> jacobi(vector<vector<double>>& A, vector<double>& B, int
    &n, double &tol, int &maxIter) {
    if(!converge(A)){
        cerr << "Matriz no converge !!" << endl;
        exit(1);
    }

    int k = 0;
    vector<double> Xold(n, 1.0);
    vector<double> Xnew(n, 0.0);
    double numerador,denominador;
    double residuo = tol + 1.0;

    while (residuo > tol && k < maxIter) {
        for (int i = 0; i < n; i++) {
            double soma = 0.0;
            for (int j = 0; j < n; j++) {
                if (j != i) {
                    soma += A[i][j] * Xold[j];
                }
            }
            Xnew[i] = (B[i] - soma) / A[i][i];
        }

        residuo = 0;
        numerador = 0;
        denominador = 0.0;
        for (int i = 0; i < n; i++) {
            numerador += pow(Xnew[i] - Xold[i], 2);
            denominador += pow(Xnew[i],2);
        }
        residuo = sqrt(numerador)/sqrt(denominador);
```

```
        Xold = Xnew;
        k++;
    }
    if (k == maxIter) {
        cout << "O mtodo iterativo de Jacobi no convergiu em " << maxIter
            << " iteraes!" << endl;
    } else {
        cout << "O mtodo iterativo de Jacobi convergiu em " << k << "
            iteraes!" << endl;
    }
    return Xnew;
}


vector<double> gauss_seidel(vector<vector<double>> &A, vector<double> &B,
    int &n, double &tol) {
    int iter = 0;
    vector<double> Xold(n, 1.0);
    vector<double> Xnew(n, 0.0);
    double residuo = tol + 1;
    double numerador,denominador;

    while (residuo > tol) {
        numerador = 0.0;
        denominador = 0.0;
        for (int i = 0; i < n; i++) {
            double soma = 0;
            for (int j = 0; j <= i-1; j++) {
                soma += A[i][j] * Xnew[j];
            }

            for (int j = i + 1; j < n; j++){

                soma += A[i][j] * Xold[j];
            }


            Xnew[i] = (B[i] - soma)/A[i][i];
            numerador += pow(Xnew[i] - Xold[i], 2);
            denominador += pow(Xnew[i],2);

            }
            residuo = sqrt(numerador)/sqrt(denominador);
            Xold = Xnew;
            iter++;
        }

    cout << "O mtodo de Gauss-Seidel convergiu em " << iter << " iteraes."
```

```
        << endl;

    return Xold;
}
```

```
vector<double> resolverSistemaLU(vector<vector<double>>& A,
    vector<vector<double>>& L, vector<double>& b, int n) {
    // Encontra a  soluo  de Ly = b
    vector<double> y(n, 0.0);
    for (int i = 0; i < n; i++) {
        double soma = 0.0;
        for (int j = 0; j < i; j++) {
            soma += L[i][j] * y[j];
        }
        y[i] = b[i] - soma;
    }


    vector<double> x(n, 0.0);
    for (int i = n - 1; i >= 0; i--) {
        double soma = 0.0;
        for (int j = i + 1; j < n; j++) {
            soma += A[i][j] * x[j];
        }
        x[i] = (y[i] - soma) / A[i][i];
    }

    return x;
}

vector<double> resolverSistemaCholesky(vector<vector<double>>& A,
    vector<vector<double>>& L, vector<double>& B, int n) {
    vector<double> y(n, 0.0);
    vector<double> x(n, 0.0);

    for (int i = 0; i < n; i++) {
        double s = 0.0;
        for (int j = 0; j < i; j++) {
            s += L[i][j] * y[j];
        }
        y[i] = (1.0 / L[i][i]) * (B[i] - s);
```

```
    }


    for (int i = n - 1; i >= 0; i--) {
        double s = 0.0;
        for (int j = i + 1; j < n; j++) {
            s += L[j][i] * x[j];
        }
        x[i] = (1.0 / L[i][i]) * (y[i] - s);
    }

    return x;
}


double normaMaiorValor(vector<double> &y){
    double maior = 0;
    for (int i = 0; i < y.size(); i++){
        if (y[i] > maior){
            maior = y[i];
        }
    }
    return maior;
}


double maxElemento(vector<vector<double>>&A, int& p, int& q) {
    double max = 0.0;
    int n = A.size();
    for (int i = 0; i < n; i++) {
        for (int j = i+1; j < n; j++) {
            if (abs(A[i][j]) > max) {
                max = abs(A[i][j]);
                p = i;
                q = j;
            }
        }
    }
    return max;
}


double anguloRotacao(vector<vector<double>>&A, int &p, int &q) {
    if (A[p][p] == A[q][q]) {
        return M_PI/4.0;
    } else {
        double tau = 2.0*A[p][q]/(A[p][p]-A[q][q]);
        return atan(tau)/2.0;
```

```
    }
}

void transporMatriz(vector<vector<double>>& matriz,
    vector<vector<double>>& matrizTransposta, int &n) {

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            matrizTransposta[j][i] = matriz[i][j];
        }
    }
}

void prodMatMat(vector<vector<double>> &X, vector<vector<double>> &P, int
    &n){
    vector<vector<double>> multi(n, vector<double>(n));
    for(int i =0; i<n; i++){
        for(int j =0; j<n;j++){
            multi[i][j] = 0.0;
            for(int k = 0; k<n; k++){
                multi[i][j] += X[i][k] * P[k][j];
            }
        }
    }
    X = multi;
}

void prodMatMatMat(vector<vector<double>> &P_trans,
    vector<vector<double>> &A,vector<vector<double>> &P, int &n){
    vector<vector<double>> multi(n, vector<double>(n));
    for(int i =0; i<n; i++){
        for(int j =0; j<n;j++){
            multi[i][j] = 0.0;
            for(int k = 0; k<n; k++){
                multi[i][j] += P_trans[i][k] * A[k][j];
            }

        }
    }

    for(int i =0; i<n; i++){
        for(int j =0; j<n;j++){
            A[i][j] = 0.0;
            for(int k = 0; k<n; k++){
                A[i][j] += multi[i][k] * P[k][j];
            }

        }
```

```
    }
}



void rotacaoJacobi(vector<vector<double>>&A, vector<vector<double>>&P,
    vector<vector<double>>&P_trans, vector<vector<double>>&X, int &p, int
    &q, int &n) {
    double angulo = anguloRotacao(A,p,q);
    double c = cos(angulo);
    double s = sin(angulo);
    P[p][p] = c;
    P[q][q] = c;
    P[p][q] = -s;
    P[q][p] = s;
    transporMatriz(P,P_trans,n);
    prodMatMatMat(P_trans,A,P,n);
    prodMatMat(X,P,n);
}


vector<double> divVec(double &lambda, vector<double>&y) {
    int n = y.size();
    vector<double> x(n,0);
    for (int i = 0; i < y.size(); i++) {
        x[i] = y[i]/lambda;
    }
    return x;
}




#endif
```

4 Task 01

4.1 Descrição Geral

O objetivo era preparar um programa computacional para efetuar a solução de um sistema linear de equações AX = B onde o usuário pudesse escolher entre os métodos:

- Decomposição LU ($ICOD = 1$);
- Decomposição de Cholesky ($ICOD = 2$);

- Procedimento iterativo Jacobi ($ICOD = 3$) e
- Procedimento iterativo Gauss-Seidel ($ICOD = 4$).

4.2 Implementação

Segue a implementação do arquivo .cpp main da task 01:

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cmath>
#include "utils.h"
using namespace std;


int main() {
    int n, ICOD, maxIter;
    double tol;
    int sair = 1;
    string arquivo = "Matriz_A.dat";
    string id;
    n = 10;

    vector<vector<double>> A(n, vector<double>(n, 0.0));
    vector<vector<double>> L(n, vector<double>(n, 0.0));
    vector<double> b(n, 0.0);
    vector<double> x(n,0.0);


    cout << "Lendo a matriz A... " << endl;
    lerMatriz(A,n,arquivo);
    cout << "Matriz A Lida: " << endl;
    imprimirMatriz(A);


    while (sair){
        cout << "Escolha o mtodo de  resoluo \n";
        cout << " Decomposio  LU (ICOD =1), Decomposio  de Cholesky (ICOD
            =2), Procedimento iterativo Jacobi (ICOD =3) ou Procedimento
            iterativo Gauss-Seidel (ICOD =4)\n";
        cin >> ICOD;

        if (ICOD == 1){
            // Executa a  decomposio  LU
            decomposicaoLU(A, L, n);
            for (int i = 1; i<= 3; i++){
                id = to_string(i);
```

```
        cout << "Lendo o vetor B"+id+"...\n";
        arquivo = "Vetor_B_0"+id+".dat";
        lerVetor(b,n,arquivo);
        cout << "Vetor B_0"+id+" Lido : " << endl;
        imprimirVetor(b);

        x = resolverSistemaLU(A, L, b, n);
        cout << "A  soluo  do sistema  :\n";
        imprimirVetor(x);
    }

}


else if (ICOD == 2){
    decomposicaoCholesky(A,L,n);
    for (int i = 1; i<= 3; i++){
        id = to_string(i);
        cout << "Lendo o vetor B_0"+id+"...\n";
        arquivo = "Vetor_B_0"+id+".dat";
        lerVetor(b,n,arquivo);
        cout << "Vetor B_0"+id+" Lido:" << endl;
        imprimirVetor(b);


        x = resolverSistemaCholesky(A,L,b,n);
        cout << "A  soluo  do sistema  :\n";
        imprimirVetor(x);
    }
}

else if (ICOD == 3){
    cout << "Escolha uma tolerncia (Entre com um valor de ponto
       flutuante entre 0 e 1): ";
    cin >> tol;

    if(tol < 0){
        cerr << "Tolerncia  no  pode ser negativa" << endl;
        return EXIT_FAILURE;
    }
    cout << "Qual a quantidade mxima de  iteraes  desejada? ";
    cin >> maxIter;
    for (int i = 1; i<= 3; i++){
        id = to_string(i);
        cout << "Lendo o vetor B_0"+id+"...\n";
        arquivo = "Vetor_B_0"+id+".dat";
        lerVetor(b,n,arquivo);
        cout << "Vetor B_0"+id+" Lido:" << endl;
```

```
                imprimirVetor(b);


                x = jacobi(A,b,tol,maxIter);
                cout << "A  soluo  do sistema  :\n";
                imprimirVetor(x);
            }


        }
        else if (ICOD == 4){
            cout << "Escolha uma tolerncia (Entre com um valor de ponto
                flutuante entre 0 e 1): ";
            cin >> tol;
            if(tol < 0){
                cerr << "Tolerncia  no  pode ser negativa" << endl;
                return EXIT_FAILURE;
            }
            for (int i = 1; i<= 3; i++){
                id = to_string(i);
                cout << "Lendo o vetor B_0"+id+"...\n";
                arquivo = "Vetor_B_0"+id+".dat";
                lerVetor(b,n,arquivo);
                cout << "Vetor B_0"+id+" Lido:" << endl;
                imprimirVetor(b);

                x = gauss_seidel(A,b,tol);
                cout << "A  soluo  do sistema  :\n";
                imprimirVetor(x);
            }
        }
        else {
            cerr << "ICOD no definido !"<<endl;
            return EXIT_FAILURE;
        }
        cout << "Deseja escolher outro mtodo? Se sim, escola 1. Do
            contrrio, entre com qualquer digito. ";
        cin >> sair;
    }
    return 0;
}
```

5 Task 02

5.1 Descrição Geral

O objetivo era preparar um programa computacional para calcular os autovalores e autovetores (possíveis) de uma matriz A pelos métodos:

- Método da Potência ($ICOD = 1$);
- Método de Jacobi ($ICOD = 2$);

Além disto, quando for requisitado pelo usuário e a técnica de solução permitir (caso contrário deve ser emitido um "warning"), que também seja efetuado o cálculo o determinante de A.

5.2 Implementação

Segue a implementação do arquivo .cpp main da task 02:

```
#include <iostream>
#include <cmath>
#include <vector>
#include <fstream>
#include "utils.h"
using namespace std;


int main() {
    int ICOD, iter;
    int n = 10;
    double tol;
    string arquivo = "Matriz_A.dat";
    vector<vector<double>> A(n, vector<double>(n));

    cout << "Lendo a Matriz A:" << endl;
    lerMatriz(A,n,arquivo);
    cout << "Matriz A Lida:" << endl;
    imprimirMatriz(A);

    cout << "Escolha o mtodo de  resoluo  (Metodo da Potncia -> 1 ou Mtodo
        de Jacobi -> 2): ";
    cin >> ICOD;

    cout << "Escolha uma tolerncia (Entre com um valor de ponto flutuante
        entre 0 e 1): ";
    cin >> tol;

    if (ICOD == 1){
        iter = 0;
        vector<double> x(n, 1.0);
```

```
    vector<double> y(n,0.0);
    double lambda = 1;
    double lambda_ant;
    do {
        lambda_ant = lambda;
        y = prodMatVec(A, x);
        lambda = normaMaiorValor(y);
        x = divVec(lambda,y);
        iter++;
    } while (abs((lambda - lambda_ant)/lambda) > tol);

    cout << "Nmero de  iteraes : " << iter << endl;
    cout << "Autovalor: " << lambda << endl;
    cout << "Autovetor: ";
    imprimirVetor(x);
}

else if (ICOD == 2){
    iter = 0;
    int p, q;
    double max;
    vector<vector<double>> P(n, vector<double>(n));
    vector<vector<double>> P_trans(n, vector<double>(n));
    vector<vector<double>> X(n, vector<double>(n));
    identidade(P,n);
    identidade(X,n);
    max = maxElemento(A, p, q);
    while (max > tol) {

        cout << " iterao : " << iter << ", valor mximo fora da
            diagonal: "<< max <<endl;
        cout << "Matriz P: " <<endl;
        imprimirMatriz(P);

        cout << "Matriz A: " <<endl;
        imprimirMatriz(A);

        cout << "Matriz X: " << endl;
        imprimirMatriz(X);

        cout << "\n";
        identidade(P,n);
        rotacaoJacobi(A, P, P_trans, X, p, q, n);
        max = maxElemento(A, p, q);
        iter++;
    }
    cout << "Matriz com Autovalores: "<<endl;
    imprimirMatriz(A);
```

```
            cout << "Matriz com Autovetores: "<<endl;
            imprimirMatriz(X);

        }
        return 0;
    }
```

6 Conclusão

Foi uma experiência muito desafiadora, aprendi muito mais sobre os algoritmos apresentados em aula e também aprimorei meus conhecimentos sobre paradigmas da linguagem de programação.