Trabalho Prático 1

Filipe Santos Pacheco Prates - 116013311

Professor Luis Volnei Sagrilo

Introdução

Nesse relatório documento o trabalho computacional realizado para a matéria COC473 Álgebra Linear Computacional, como parte da primeira nota.

Listagem do código

https://github.com/FilipePrates/COC473-Algebra-Linear-Computacional

Tarefa 01:

```cpp
vector<double> X;
if (function == 1) {
    matriz LU = LUdecomp(A);
    X = solveLU(LU, B);
} else if (function == 2) {
    matriz L = cholesky(A);
    X = solveCholesky(L, B);
} else {
    cout << "Por favor respeite as opções disponíveis." << endl;
    return -1;
}
for (int i = 0; i < X.size(); i++) {
    cout << "x_" << i+1 << " = " << X[i] << endl;
}
```

Na primeira tarefa foi pedido um trabalho computacional que resolva sistemas lineares recebendo como input **uma Matriz A** (quadrada 10x10), e **um Vetor B** (1x10), utilizando as técnicas de Cholesky e decomposição LU.

Decomposição LU:

Na decomposição LU, através da técnica de eliminação de Gauss, recebemos uma matriz **A** e geramos LU, sendo L a “Lower”, matriz diagonal inferior com diagonal 1. E U, “Upper”, diagonal superior pela eliminação de Gauss.

```
matriz LUdecomp(matriz A) {
    int n = NUM_LINES;
    matriz LU = A;
    for (int j = 0; j < n-1; j++) {
        for (int i = j+1; i < n; i++) {
            LU.elements[i][j] = LU.elements[i][j] / LU.elements[j][j];
            for (int k = j+1; k < n; k++) {
                LU.elements[i][k] = LU.elements[i][k] - LU.elements[i][j] * LU.elements[j][k];
            }
        }
    }
    return LU;
}
```

Após LU = LUdecomp(A), resolvemos o sistema através da matriz LU com solve(LU,**B**)

```
vector<double> solveLU(matriz LU, vector<double> B) {
    int n = NUM_LINES;
    vector<double> X(n);
    vector<double> Y(n);
    //LY = B
    Y[0] = B[0];
    for (int i = 1; i < n; i++) {
        double sum = 0;
        for (int j = 0; j < i; j++) {
            sum += LU.elements[i][j] * Y[j];
        }
        Y[i] = B[i] - sum;
    }
    //UX = Y
    X[n-1] = Y[n-1] / LU.elements[n-1][n-1];
    for (int i = n-2; i >= 0; i--) {
        double sum = 0;
        for (int j = i+1; j < n; j++) {
            sum += LU.elements[i][j] * X[j];
        }
        X[i] = (Y[i] - sum) / LU.elements[i][i];
    }
    return X;
}
```

Cholesky:

Processo que matriz passa é parecido com LU, porém podemos utilizar do fato da Matriz A do exemplo ser **simétrica** para melhorar computacionalmente.

```
matriz cholesky(matriz A) {
    int n = NUM_LINES;
    matriz L = A;
    for (int j = 0; j < n; j++) {
        double sum = 0;
        for (int k = 0; k < j; k++) {
            sum += pow(L.elements[j][k], 2);
        }
        L.elements[j][j] = sqrt(L.elements[j][j] - sum);
        for (int i = j+1; i < n; i++) {
            sum = 0;
            for (int k = 0; k < j; k++) {
                sum += L.elements[i][k] * L.elements[j][k];
            }
            L.elements[i][j] = (L.elements[i][j] - sum) / L.elements[j][j];
        }
    }
    return L;
}
```

E novamente solucionamos o sistema usando as formulas do slide:

```
vector<double> solveCholesky(matriz L, vector<double> B) {
    int n = NUM_LINES;
    vector<double> X(n);
    vector<double> Y(n);
    //LY = B
    Y[0] = B[0] / L.elements[0][0];
    for (int i = 1; i < n; i++) {
        double sum = 0;
        for (int j = 0; j < i; j++) {
            sum += L.elements[i][j] * Y[j];
        }
        Y[i] = (B[i] - sum) / L.elements[i][i];
    }
    //L^TX = Y
    X[n-1] = Y[n-1] / L.elements[n-1][n-1];
    for (int i = n-2; i >= 0; i--) {
        double sum = 0;
        for (int j = i+1; j < n; j++) {
            sum += L.elements[j][i] * X[j];
        }
        X[i] = (Y[i] - sum) / L.elements[i][i];
    }
    return X;
}
```

Tarefa 02:

Nessa tarefa foi pedido para implementar computacionalmente um sistema que soluciona o sistema linear AX = B, através do método de **Jacobi** e **Gauss-Seidel**.

```
vector<double> X;
if (function == 1) {
    int maxIter;
    double tol;
    cout << "Max Iterações (1000): ";
    cin >> maxIter;
    cout << "Tolerância (0.0005): ";
    cin >> tol;
    X = jacobi(A, B, maxIter, tol);
} else if (function == 2) {
    int maxIter;
    double tol;
    cout << "Max Iterações (1000): ";
    cin >> maxIter;
    cout << "Tolerância (0.0005): ";
    cin >> tol;
    X = gaussSeidel(A, B, maxIter, tol);
} else {
    cout << "Por favor respeite as opções disponíveis." << endl;
    return -1;
}
for (int i = 0; i < X.size(); i++) {
    cout << "x_" << i+1 << " = " << X[i] << endl;
}
```

Jacobi:

É um método Iterativo, que **resolve AX=B sem precisar decompor ou inverter A**.

Começa com um "chute", x_0 , que então é atualizado segundo a regra de atualização de Jacobi para se obter um x_1, mais próximo do resultado correto. Assim sucessivamente até alguma condição de parada seja encontrada. É então necessário estabelecer uma tolerância para se obter o resultado e um número máximo de iterações, para para caso divirja.

```cpp
vector<double> jacobi(matriz A, vector<double> B, int maxIter, double tol) {
    vector<double> X(NUM_LINES, 0);
    vector<double> XX(NUM_LINES, 0);
    int iter = 0;
    double erro = 1;
    while (iter < maxIter && erro > tol) {
        XX = X;
        for (int i = 0; i < NUM_LINES; i++) {
            double soma = 0;
            for (int j = 0; j < NUM_COLS; j++) {
                if (j != i) {
                    soma += A.elements[i][j] * XX[j];
                }
            }
            X[i] = (B[i] - soma) / A.elements[i][i];
        }
        erro = relativeError(X, XX);
        iter++;
    }
    if (iter == maxIter) {
        cout << "Saiu por Max Iter" << endl;
    } else {
        cout << iter << " iterações" << endl;
    }
    return X;
}
```

Gauss-Seidel

Processo similar ao Jacobi (também é um método iterativo), porém com uma regra de atualização do vetor aproximado de solução X mais eficiente, levando em conta o “chute” x atual e o imediatamente anterior, em uma expressão recursiva.

```cpp
vector<double> gaussSeidel(matriz A, vector<double>
 B, int maxIter, double tol) {
    vector<double> X(NUM_LINES, 0);
    vector<double> XX(NUM_LINES, 0);
    int iter = 0;
    double erro = 1;
    while (iter < maxIter && erro > tol) {
        XX = X;
        for (int i = 0; i < NUM_LINES; i++) {
            double soma1 = 0;
            for (int j = 0; j < i; j++) {
                soma1 += A.elements[i][j] * X[j];
            }
            double soma2 = 0;
            for (int j = i + 1; j < NUM_COLS; j++) {
                soma2 += A.elements[i][j] * XX[j];
            }
            X[i] = (B[i] - soma1 - soma2) / A.elements[i][i];
        }
        erro = relativeError(X, XX);
        iter++;
    }
    if (iter == maxIter) {
        cout << "Saiu por Max Iter" << endl;
    } else {
        cout << iter << " iterações" << endl;
    }
    return X;
}
```

Exemplo resolvido pelo código

Tarefa 01: Exemplo Escolhido → **Matriz A.dat** e **Vetor_01.dat**, resolvido por decomposição de **Cholesky**

```
root@LAPTOP-CNT03N69:~/alglincomp# ./Main
Qual o Vetor B a ser usado? (1/2/3)
1
Digite o código do método desejado:
1 - Decomposição LU
2 - Decomposição de cholesky
2
Resultado :
x_1 = -4.16667
x_2 = -3.33333
x_3 = -10.6667
x_4 = -2.66667
x_5 = -13.5
x_6 = 2.61657e-15
x_7 = -10.6667
x_8 = 2.66667
x_9 = -4.16667
x_10 = 3.33333
Qual o Vetor B a ser usado? (1/2/3)
```

Tarefa 02: Exemplo Escolhido → **Matriz A.dat** e **Vetor_03.dat**, resolvido pelo algoritmo iterativo de **Jacobi.**

```
root@LAPTOP-CNT03N69:~/alglincomp# ./Main2
Qual o Vetor B a ser usado? (1/2/3)
3
Digite o código do método desejado:
1 - Jacobi
2 - Gauss-Seidel
1
Max Iterações (1000): 1000
Tolerância (0.0005): 0,00005
Saiu por Max Iter
x_1 = -12.5
x_2 = -10
x_3 = -32
x_4 = -8
x_5 = -40.5
x_6 = 1.06581e-15
x_7 = -32
x_8 = 8
x_9 = -12.5
x_10 = 10
root@LAPTOP-CNT03N69:~/alglincomp#
```

Enunciado:

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/cd6201e9-7578-44bb-a0a7-838284e2e66d/ALC473_P1_2023.pdf