UNIVERSIDADE FEDERAL FLUMINENSE

Fatoração LU com Pivoteamento Parcial

Alunos

Anna O. Monteiro
Caio Silva Couto
Patrick Andrade
Pedro Lameirão

PROFESSOR

Marco Antônio Monteiro Silva Ramos

DISCIPLINA

Métodos Numéricos

Dezembro de 2022

Objetivos

O presente documento tem por objetivo a elucidação de um código computacional em linguagem C++ para a resolução de sistemas lineares Ax = b por meio da Fatoração PA = LU com pivoteamento parcial.

Metodologia

Para o desenvolvimento do código foi utilizado o editor de código-fonte Visual Studio Code e conhecimentos de programação alinhados à Álgebra Linear para a edificação do programa.
Para a elaboração e formatação deste arquivo foi utilizado o sistema de preparação de documentos, LaTeX, e sua respectiva linguagem de programação, por meio do editor colaborativo Overleaf.

Sumário

1 Aplicação da Fatoração LU

A fatoração de matrizes leva em conta três importantes premissas, chamadas "operações elementares". São elas:

- Permutação duas equações;
- Multiplicação uma equação por uma constante não-nula;
- Adição ou subtração do múltiplo de uma equação à outra

A fatoração LU considera sistemas lineares no formato "Ax = b", onde "A" é a matriz principal, "x" representa a coluna "x1, x2, x3, (...), xn", e "b" é a coluna de resultados de cada equação do sistema.
Para iniciar o processo, deve-se fatorar a matriz A, dividindo-a em duas matrizes triangulares — L e U.
Em teoria, a Fatoração LU equivale ao método da Eliminação de Gauss, entretanto, neste método, os multiplicadores usados para a transformação da matriz inicial A em uma matriz triangular superior U são guardados.
A matriz L é, então, composta por zeros acima da diagonal principal (compostas por "1"), a U por zeros abaixo da diagonal principal, conforme o exemplo abaixo.

$$\begin{bmatrix} 2 & -1 & 4 & 0 \\ 4 & -1 & 5 & 1 \\ -2 & 2 & -2 & 3 \\ 0 & 3 & -9 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 5 \\ 9 \\ 1 \\ -2 \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{1}{2} & \frac{1}{2} & 1 & 0 \\ \frac{1}{2} & -\frac{1}{6} & 0 & 1 \end{bmatrix} \quad \text{e} \quad U = \begin{bmatrix} 4 & -1 & 5 & 1 \\ 0 & 3 & -9 & 4 \\ 0 & 0 & 5 & \frac{3}{2} \\ 0 & 0 & 0 & \frac{1}{6} \end{bmatrix}$$

Após a obtenção das matrizes L e U, por fim, são feitas operações as seguintes operações, respectivamente:

- $L \cdot y = b$
- $U \cdot x = y$

Assim, será possível a obtenção dos valores corretos para a coluna "y" e para a coluna "x", chegando-se ao fim do processo de fatoração LU.

2 Função main( )

Para iniciar é importante falar sobre a função "main( )". Esta é a premissa básica para que um programa seja executado, em outros termos, é o "entry point" do programa. Com essa função, é possível a execução das chamadas de outras funções dentro do programa.

3 Inclusão De Arquivos à Biblioteca Padrão C e ao Código Fonte

```
#include <iostream>
#include <cmath>
#include <tuple>
#include <fstream>
#include <vector>
#include <string>
```

#include

- iostream : Leitura e gravação a partir dos fluxos de saída e entrada, em específico para inputs e outputs no terminal do programa
- cmath : Funções para cálculo matemático
- tuple : Conter elementos que são inicializados como argumentos em ordem de acesso
- fstream : Leitura ou escrita em arquivos
- vector : organização com acesso aleatório rápido de elementos em sequência linear.
- string : representação de palavras, frases ou textos por meio de sequência de caracteres.

4 Leitura de Arquivo de texto para Retorno de Vetor Float

O bloco começa por declarar um leitor de texto do tipo ifstream com nomeado “vectorb” para a abertura do arquivo de texto.
Em seguida é declarada uma variável para o armazenamento do número atual “num” e uma outra variável para o armazenamento do vetor em si.

```
10 vector<float> read_b(){
11         ifstream vectorb;
12         vectorb.open("b_vector.txt");
13 
14         float num;
15         vector<float> b;
16 
17         while(vectorb.good()){
18                 vectorb >> num;
19                 b.push_back(num);
20         }
21 
22         vectorb.close();
23         return b;
24 }
```

- “vectob.good()” retornará verdadeiro (true) enquanto o arquivo de texto não chegar ao final.
- ”vectorb num” extrairá um número do arquivo de texto e colocará o valor como um float em “num”.
- “b.push_back(num)” posicionará esse valor no vetor.

Em seguida, o arquivo de texto é fechado e retorna o vetor lido.

5 “Vector Within a Vector”

Representa a criação de “vetores dentro de vetores”.
É, basicamente, a mesma ideia do tópico anterior, entretanto, agora, para uma matriz com “n” colunas ou “n” linhas (no caso para “n” linhas).
Nesta etapa, a função “read_A” recebe um inteiro obtido a partir do comando “size.vector_b()” e declarado em main.

```
26 vector<vector<float>> read_A(int n){
27         ifstream matrix_A;
28         matrix_A.open("A_matrix.txt");
29 
30         vector<vector<float>> A;
31         int i = 0;
32         float num;
33         vector<float> temp;
34 
35         while(matrix_A.good()){
36                 matrix_A >> num;
37                 temp.push_back(num);
38                 i++;
39                 if(i == n){
40                         A.push_back(temp);
41                         temp.clear();
42                         i = 0;
43                 }
44         }
45         matrix_A.close();
46 
47         return A;
48 }
```

Sendo assim, se os arquivos estiverem corretamente formatados, a função “size.vector_b()” retornará a ordem do sistema linear, considerando que o inteiro “n” representa a quantidade de termos independentes. Importante destacar que, após isso, o processo é semelhante ao “read_b”, pois o programa aloca, à variável “num”, um valor da esquerda para a direita e de cima para baixo — $[a_{11}, a_{12}, a_{13}, (...), a_{1n}]$

A variável “num” é alocada, a cada volta de “while” , em um vetor temporário, enquanto a variável “i” recebe o acréscimo de uma unidade (i + 1). Seguindo essa linha, quando “i” tiver passado por todos os elementos de uma linha da matriz em questão, isto é, quando “i” for equivalente à ordem do sistema, o comando “if” é agregado e, dentro dele, o vetor “temp” é alocado em “A”, para que este possa ser zerado e “i” possa assumir “i = 1”.

6 Vetor “x” em Arquivo de Texto

```
50 void write_x(vector<float> x){
51         ofstream vectorx("x_vector.txt");
52         int n = x.size();
53 
54         for(int i=0; i<n; i++){
55                 vectorx<<x[i];
56                 vectorx<<endl;
57         }
58         vectorx.close();
59 }
```

A presente função escreve o vetor “x” em um arquivo de texto.
O “ofstream” também é usado para iterar através de arquivos de texto como o “ifstream”, porém, enquanto o “ifstream” é usado para ler e interpretar arquivos de texto, o “ofstream” é feito apenas para escrevê-los.

7 Escolha do Pivô e Manipulação Algébrica

Neste bloco, é escolhida uma matriz ”U” e, dentro dela, é buscado um vetor a partir da “p-ésima” linha até a “n-ésima” linha.
A função ”fabs( )” retornará o valor absoluto. Sendo assim, começa no p-ésimo elemento da coluna ”p”. Dessa forma, vai sendo feita a iteração pela coluna ”p”, mudando apenas a linha.

```
61 int find_piv(vector<vector<float>> U, int n, int p){
62         float par = fabs(U[p][p]);
63         int line = p;
64 
65         for(int i=p; i<n; i++){
66                 if(fabs(U[i][p])>par){
67                         par=fabs(U[i][p]);
68                         line = i;
69                 }
70         }
71         return line;
72 }
```

Se o valor em módulo do pivô da linha ”i” da coluna ”p” for maior que o anterior, a troca de linhas é feita, fazendo com que essa linha seja igual a ”i”. Em outras palavras, a função deste bloco é procurar o maior número em módulo de uma coluna e retornar a linha onde esse número está.

8 Escalonando a Matriz

Esta função, em suma, resolve o sistema linear.
A partir da matriz "L", inicia-se um vetor "c" (vetor das constantes) com a premissa "c0 = b0", pois o primeiro valor inserido em "c" é b0.
A função "push_back" é justamente utilizada para inserir novos elementos ao final de um vetor ou enviar um elemento para o vetor anterior.

```
110 vector<float> solve_systemL(vector<vector<float>> L,
111 vector<float> b, int n){
112         vector<float> c;
113         c.push_back(b[0]);
114         float sum = 0;
115 
116         for(int i=1; i<n; i++){
117                 for(int j=0; j<i; j++){
118                         sum += L[i][j]*c[j];
119                 }
120                 c.push_back(b[i]-sum);
121                 sum = 0;
122         }
123         return c;
124 }
```

Algebricamente, a conta seria:

L[1][0]×c0+c1=b1
L[2][0]×c0+L[2][1]×c1+c2=b2
(...)

E assim até encontrar todos os valores abaixo da diagonal principal da matriz (código retorna o valor de "c").

9 Matrizes L e U

Uma tupla é retornada de duas matrizes e um vetor.
Para as duas matrizes são criados dois vetores: "temp1" e "temp2". Esses vetores armazenam temporariamente as linhas que serão trocadas.

```
74  tuple<vector<vector<float>>,vector<vector<float>>,
75  vector<float>> ch_line(vector<vector<float>> U,
76  vector<vector<float>> L,
77  vector<float> x, int n, int p, int line){
78          vector<float> temp1, temp2;
79          float temp3;
80
81          for(int i=0; i<n; i++){
82                  temp1.push_back(U[line][i]);
83                  U[line][i] = U[p][i];
84                  U[p][i] = temp1[i];
85          }
86          for(int i=0; i<p; i++){
87                  temp2.push_back(L[line][i]);
88                  L[line][i] = L[p][i];
89                  L[p][i] = temp2[i];
90          }
91          temp3 = x[line];
92          x[line] = x[p];
93          x[p] = temp3;
94
95          return {U, L, x};
96  }
```

A matriz U possui "n" colunas que são percorridas e os valores da linha "line" são colocados dentro do vetor "temp1", e os valores dessa linha são trocados com os valores da linha p dessa matriz.
Na matriz "L" é feito o mesmo processo, porém a diferença está em que essa matriz tem "p" colunas.
Para o vetor, o valor que está na "line-ésima" linha de "line" é trocado com o valor "p".
Ao final, a função retorna as novas matrizes

10 Pivoteamento

Esta função retorna uma tupla (matrizes U e L) e um vetor "x".

```
tuple<vector<vector<float>>,vector<vector<float>>,
vector<float>> piv(vector<vector<float>> U,
vector<vector<float>> L, vector<float> x , int n, int p){
        float pivo = U[p][p];

        for(int i=p+1; i<n; i++){
                float coef = U[i][p];
                for(int j=0; j<n; j++){
                        U[i][j]=U[i][j]-(coef/pivo)*U[p][j];
                        L[i][p] = coef/pivo;
                }
        }
        return {U, L, x};
}
```

A partir disso, é feito o pivoteamento — com o pivô definido como valor "p" da linha p da matriz U. É identificado o coeficiente (valor "i" da linha p) e este é alterado.
Então é feita a iteração de "j" até o último valor.

11 "x Inverso"

O seguinte bloco faz o mesmo do sistema de L, mas para U. Ao invés de definir um vetor "x", foi definido um vetor "x" inverso ("x_inv"), isto é, de trás para frente.

O último valor de "c" (n - 1) é dividido pelo último valor da última linha de U.
É feita a iteração de trás para frente, multiplicando pelo valor de "x" inverso. Ou seja, é a mesma ideia da sessão anterior, porém de forma inversa.Isso se dá dessa forma pois a matriz L é composta de zeros acima da diagonal principal e a matriz U é composta de zeros abaixo da diagonal principal.

```
125 vector<float> solve_systemU(vector<vector<float>> U,
126 vector<float> c, int n){
127         vector<float> x_inv, x;
128         x_inv.push_back(c[n-1]/(U[n-1][n-1]));
129         float sum = 0;
130 
131         for(int i=n-2; i>=0; i--){
132                 for(int j=n-1; j>i; j--){
133                         sum += U[i][j]*x_inv[n-(1+j)];
134                 }
135                 x_inv.push_back((c[i]-sum)/(U[i][i]));
136                 sum = 0;
137         }
138         for(int i=n-1; i>=0; i--){
139                 x.push_back(x_inv[i]);
140         }
141 
142         return x;
143 }
```

Em síntese, como a matriz U é resolvida no sentido de baixo para cima, a função “push_back” adiciona, primeiramente, o “n-ésimo” valor “x”, e, em seguida, o “n-ésimo” valor “x” menos uma unidade (“x-1”) e assim por diante.
Por essa razão, este vetor precisa ter os elementos ”pivoteados” para se encontrarem na mesma ordem de apresentação dos coeficientes na ”A_matrix.txt”.

12 Finalização do Problema

O código segue, então, realizando operações algebricas — assim como as citadas na sessão ”Fatoração LU” — para determinar os valores finais das matrizes L, U e b, assim como no bloco a seguir.

```
for(int p=0; p<n-1; p++){
        tuple<vector<vector<float>>, vector<vector<float>>,
vector<float>> temp;
        line = find_piv(U, n, p);
        temp = ch_line(U, L, b, n, p, line);
        U = get<0>(temp);
        L = get<1>(temp);
        b = get<2>(temp);
        temp = piv(U, L, b, n, p);
        U = get<0>(temp);
        L = get<1>(temp);
}
c = solve_systemL(L, b, n);
x = solve_systemU(U, c, n);

write_x(x);
return 0;
}
```

O código encerra, por fim, retornando o resuldado final, isto é os valores de ”x”.

13 Conclusão

A Fatoração LU é uma das formas mais eficientes para resolver sistemas lineares.
Algumas das principais razões para a utilização desse método é o fato de fornecer uma forma hábil para o cálculo de matrizes inversas, a qual é muito utilizada para rotacionar ou transpor um objeto no espaço 3D, por exemplo. Além disso, é um excelente meio de avaliação do condicionamento de sistemas. Dessa forma, aliada à programação, o método da fatoração LU se torna rápido e sistemático, que beneficia aqueles que precisam utilizá-lo, facilitando sua aplicação, tendo em vista que, para as máquinas, é um trabalho completamente trivial.

14 Código na íntegra

```
#include <iostream>
#include <cmath>
#include <tuple>
#include <fstream>
#include <vector>
#include <string>

using namespace std;

vector<float> read_b(){
        ifstream vectorb;
        vectorb.open("b_vector.txt");

        float num;
        vector<float> b;

        while(vectorb.good()){
                vectorb >> num;
                b.push_back(num);
        }

        vectorb.close();
        return b;
}

vector<vector<float>> read_A(int n){
        ifstream matrix_A;
        matrix_A.open("A_matrix.txt");

        vector<vector<float>> A;
        int i = 0;
        float num;
        vector<float> temp;

        while(matrix_A.good()){
                matrix_A >> num;
                temp.push_back(num);
                i++;
                if(i == n){
                        A.push_back(temp);
                        temp.clear();
                        i = 0;
                }
        }
        matrix_A.close();

        return A;
}

```

```
50  void write_x(vector<float> x){
51          ofstream vectorx("x_vector.txt");
52          int n = x.size();
53
54          for(int i=0; i<n; i++){
55                  vectorx<<x[i];
56                  vectorx<<endl;
57          }
58          vectorx.close();
59  }
60
61  int find_piv(vector<vector<float>> U, int n, int p){
62          float par = fabs(U[p][p]);
63          int line = p;
64
65          for(int i=p; i<n; i++){
66                  if(fabs(U[i][p])>par){
67                          par=fabs(U[i][p]);
68                          line = i;
69                  }
70          }
71          return line;
72  }
73
74  tuple<vector<vector<float>>,vector<vector<float>>, vector<float>>
75  ch_line(vector<vector<float>> U, vector<vector<float>> L, vector<float> x,
76  int n, int p, int line){
77          vector<float> temp1, temp2;
78          float temp3;
79
80          for(int i=0; i<n; i++){
81                  temp1.push_back(U[line][i]);
82                  U[line][i] = U[p][i];
83                  U[p][i] = temp1[i];
84          }
85          for(int i=0; i<p; i++){
86                  temp2.push_back(L[line][i]);
87                  L[line][i] = L[p][i];
88                  L[p][i] = temp2[i];
89          }
90          temp3 = x[line];
91          x[line] = x[p];
92          x[p] = temp3;
93
94          return {U, L, x};
95  }
96
97  tuple<vector<vector<float>>,vector<vector<float>>, vector<float>>
98  piv(vector<vector<float>> U, vector<vector<float>> L, vector<float> x,
99  int n, int p){
100         float pivo = U[p][p];
```

```
101
102         for(int i=p+1; i<n; i++){
103                 float coef = U[i][p];
104                 for(int j=0; j<n; j++){
105                         U[i][j]=U[i][j]-(coef/pivo)*U[p][j];
106                         L[i][p] = coef/pivo;
107                 }
108         }
109         return {U, L, x};
110 }
111
112 vector<float> solve_systemL(vector<vector<float>> L, vector<float> b, int n){
113         vector<float> c;
114         c.push_back(b[0]);
115         float sum = 0;
116
117         for(int i=1; i<n; i++){
118                 for(int j=0; j<i; j++){
119                         sum += L[i][j]*c[j];
120                 }
121                 c.push_back(b[i]-sum);
122                 sum = 0;
123         }
124         return c;
125 }
126
127 vector<float> solve_systemU(vector<vector<float>> U, vector<float> c, int n){
128         vector<float> x_inv, x;
129         x_inv.push_back(c[n-1]/(U[n-1][n-1]));
130         float sum = 0;
131
132         for(int i=n-2; i>=0; i--){
133                 for(int j=n-1; j>i; j--){
134                         sum += U[i][j]*x_inv[n-(1+j)];
135                 }
136                 x_inv.push_back((c[i]-sum)/(U[i][i]));
137                 sum = 0;
138         }
139         for(int i=n-1; i>=0; i--){
140                 x.push_back(x_inv[i]);
141         }
142
143         return x;
144 }
145
146 void print_matrix(vector<vector<float>> Matrix, int n){
147                 for(int i=0; i<n; i++){
148                         for(int j=0; j<n; j++){
149                                 cout<<Matrix[i][j]<<' ';
150                 }
151                 cout<<endl;
```

```
152         }
153 }
154
155 void print_vector(vector<float> b, int n){
156         for(int i=0; i<n; i++){
157                 cout<<b[i]<<' ';
158         }
159         cout<<endl;
160 }
161
162 int main(){
163         int line;
164         vector<float> b;
165         vector<vector<float>> A;
166
167         b = read_b();
168         int n = b.size();
169         A = read_A(n);
170
171         vector<float> c, x;
172         vector<vector<float>> U, L;
173
174         U = A;
175
176         for(int i=0; i<n; i++){
177                 vector<float> temp;
178                 for(int j=0; j<n; j++){
179                         if(i==j){
180                                 temp.push_back(1);
181                         }else{
182                                 temp.push_back(0);
183                         }
184                 }
185                 L.push_back(temp);
186         }
187
188         for(int p=0; p<n-1; p++){
189                 tuple<vector<vector<float>>, vector<vector<float>>,
190                 vector<float>> temp;
191                 line = find_piv(U, n, p);
192                 temp = ch_line(U, L, b, n, p, line);
193                 U = get<0>(temp);
194                 L = get<1>(temp);
195                 b = get<2>(temp);
196                 temp = piv(U, L, b, n, p);
197                 U = get<0>(temp);
198                 L = get<1>(temp);
199         }
200         c = solve_systemL(L, b, n);
201         x = solve_systemU(U, c, n);
202
```

```
203         write_x(x);
204         return 0;
205 }
```

O mesmo pode ser também encontrado em:
https://github.com/th3worst4/Trab_PALU