1. OperacionesMatriz.h

Aquí se definen todas las operaciones de la Matriz que utilizaremos en nuestros métodos.

```
#ifndef OPERACIONESMATRIZ_H
#define OPERACIONESMATRIZ_H

#include <iostream>
#include <cmath>
#include <vector>

using namespace std;

typedef long double Num;
typedef vector<vector<Num>> Matriz;
typedef vector<Num> Lista;

void mostrarLista(Lista &A){
    for(int i = 0; i < A.size(); i++){
        cout<<A[i]<<" ";
    }
    cout<<endl<<endl;;
}

Matriz matrizAumentada(Matriz A, Lista B){
    Matriz res = A;
    for(int i = 0; i < B.size(); i++){
        res[i].push_back(B[i]);
    }
    return res;
}

void mostrarMatriz(Matriz &A){
    for(int i = 0; i < A.size(); i++){
        for(int j = 0; j < A[i].size(); j++){
            cout<<A[i][j]<<" ";
        }
        cout<<endl;
    }
    cout<<endl;
}

int buscarMayor(int ini, int fin, int j, Matriz &A){
    int mayor = abs(A[ini][j]);
    int index = ini;
    for(int i = ini +1; i <= fin; i++){
        if(mayor < abs(A[i][j])){
            mayor = abs(A[i][j]);
            index = i;
        }
    }
    return index;
}

Lista operator *(Matriz a, Lista b){
    int n = a.size();
    Lista res = b;
    for(int i = 0; i < n; i++){
        for(int k = 0; k < n; k++){
            Num sum = 0;
            for(int j = 0; j < n; j++){
                sum += a[i][j] * b[j];
            }
            res[i] = sum;
        }
    }
    return res;
}

Matriz operator *(Matriz a, Matriz b){
    int n = a.size();
    Matriz res = a;
    for(int i = 0; i < n; i++){
        for(int k = 0; k < n; k++){
            Num sum = 0;
            for(int j = 0; j < n; j++){
                sum += a[i][j] * b[j][k];
            }
            res[i][k] = sum;
        }
    }
    return res;
}

Matriz operator +(Matriz a, Matriz b){
    int n = a.size();
    Matriz res = a;
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
```

```
            res[i][j] = a[i][j] + b[i][j];
        }
    }
    return res;
}

Matriz zeros(int n){
    Matriz res(n);
    for(int i = 0; i < n; i++){
        res[i] = vector<Num>(n);
    }
    return res;
}

Matriz identidad(int n){
    Matriz res(n);
    for(int i = 0; i < n; i++){
        res[i] = vector<Num>(n);
    }
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            if(i == j) res[i][j] = 1;
        }
    }
    return res;
}

#endif
```

2. Sustituciones.h

Aquí se definen los dos tipos de sustituciones a usar: La sustitución regresiva y la sustitución progresiva.

```
#ifndef SUSTITUCIONES_H
#define SUSTITUCIONES_H

#include <iostream>
#include <cmath>
#include <algorithm>
#include <vector>
#include "OperacionesMatriz.h"

using namespace std;

Lista SustitucionRegresiva(Matriz A){ //Con Matriz Aumentada
    int j = 0;
    Lista res(A.size());
    for(int i = A.size() - 1; i >= 0; i--, j++){
        Num sum = 0;
        for(int k = 0; k < j; k++){
            sum += A[i][A[i].size() - 2 - k] * res[A[i].size() -2 -k];
        }
        res[A[i].size() -2-j] = (A[i][A[i].size()-1] - sum) / A[i][A[i].size()-2-j];

    }
    return res;
}

Lista SustitucionProgresiva(Matriz A){ //Con Matriz Aumentada
    int j = 0;
    Lista res(A.size());
    for(int i = 0; i < A.size(); i++, j++){
        Num sum = 0;
        for(int k = 0; k < j; k++){
            sum += A[i][k] * res[k];
        }
        res[j] = (A[i][A[i].size()-1] - sum) / A[i][j];
    }
    return res;
}


#endif
```

3. Gauss.cpp

Aquí está resuelto el método de Gauss con pivoteo.

```
#include <iostream>
#include "Sustituciones.h"
#include "OperacionesMatriz.h"

using namespace std;

Matriz Gauss(Matriz A, Lista &B){
    A = matrizAumentada(A,B);
    int pibot = 0;
    while(pibot != B.size() - 1){
        int mayor = buscarMayor(pibot,B.size()-1,pibot,A);
        if (mayor != pibot) swap(A[pibot],A[mayor]);
        for(int i = pibot+1; i < B.size(); i++){
            Num d = A[i][pibot]/A[pibot][pibot];
            for(int j = pibot; j < A[i].size(); j++){
                A[i][j] = A[i][j] - d*A[pibot][j];
            }
        }
        pibot++;
    }
    return A;
}

int main(){
    Matriz A = {{1,1,-1},{2,-1,1},{4,1,-2}};
    Lista B = {1,2,3};
    Matriz AA = Gauss(A,B);
    Lista res = SustitucionRegresiva(AA);
    for(Num n : res){
        cout<<n<<" ";
    }
    cout<<endl;
}
```

4. PLU.cpp

Aquí está resuelto el método de PLU con piboteo.

```
#include <iostream>
#include <tuple>
#include "Sustituciones.h"
#include "OperacionesMatriz.h"

using namespace std;

enum Tipes{M_P,M_L,M_U};

tuple<Matriz,Matriz,Matriz> PLU(Matriz A){
    Matriz P = identidad(A.size());
    Matriz L = zeros(A.size());
    Matriz U = A;
    int pibot = 0;
    while(pibot != U.size() - 1){
        int mayor = buscarMayor(pibot,U.size()-1,pibot,U);
        if (mayor != pibot){
            swap(U[pibot],U[mayor]);
            swap(P[pibot],P[mayor]);
            swap(L[pibot],L[mayor]);
        }
        for(int i = pibot+1; i < U.size(); i++){
            Num d = U[i][pibot]/U[pibot][pibot];
            L[i][pibot] = d;
            for(int j = pibot; j < U[i].size(); j++){
                U[i][j] = U[i][j] - d*U[pibot][j];
            }
        }
        pibot++;
    }
    L = L + identidad(L.size());
    return make_tuple(P,L,U);
}

int main(){
    Matriz A = {{1,1,-1},{2,-1,1},{4,1,-2}};
```

```
    Lista B = {1,2,3};
    auto t = PLU(A);
    Matriz P = get<M_P>(t);
    Matriz L = get<M_L>(t);
    Matriz U = get<M_U>(t);
    mostrarMatriz(P);
    mostrarMatriz(L);
    mostrarMatriz(U);
    Matriz Q = P * A;
    Matriz W = L * U;
    mostrarMatriz(Q);
    mostrarMatriz(W);
    Lista Y = SustitucionProgresiva(matrizAumentada(L,P*B));
    mostrarLista(Y);
    Lista res = SustitucionRegresiva(matrizAumentada(U,Y));
    mostrarLista(res);
}
```