1. Interpolacion de Lagrange

El Main1.cpp genera una funcion en c++ que es compilada luego por el Main2.cpp para dibujar la gráfica. Esto se hace para no calcular el polinomio cada vez que se tenga que calcular la f(x) de una x.

1.1. main1.cpp

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

using namespace std;

typedef long double Num;

void generarPolinomio(vector<Num> X, vector<Num> Y){
    ofstream archivo("funcion.h");
    archivo<<"#ifndef FUNCION_H"<<endl;
    archivo<<"#define FUNCION_H"<<endl;
    archivo<<"#include <iostream>"<<endl;
    archivo<<"#include <cmath>"<<endl;
    archivo<<"typedef long double Num;"<<endl<<endl;
    archivo<<"Num fun(Num x){"<<endl;
    string L = "";
    for(int i = 0; i < Y.size(); i++){
        L = L + to_string(Y[i]);
        L = L + "*((";
        for(int j = 0; j < X.size(); j++){
            if(i != j){
                L = L + "(x-" + to_string(X[j]) + ")*";
            }
        }
        L.pop_back();
        L = L + ")/(";
        for(int j = 0; j < X.size(); j++){
            if(i != j){
                L = L + "(" +  to_string(X[i]) + "-" + to_string(X[j]) + ")*";
            }
        }
        L.pop_back();
        L = L + "))+";
    }
    L.pop_back();
    archivo<<"return "<<L<<";"<<endl;
    archivo<<"}"<<endl<<endl;
    archivo<<"#endif";
    archivo.close();
    system("g++ -std=c++11 second.cpp -o run2");
    system("./run2");
}

int main(){
    vector<Num> X = {0,1,1.5,2.4,3,4};
    vector<Num> Y = {cos(0),cos(1),cos(1.5),cos(2.4),cos(3),cos(4)};
    generarPolinomio(X,Y);
}
```

1.2. main2.cpp

```
#include <iostream>
#include <fstream>
#include <vector>
#include "funcion.h"

using namespace std;

int main(){
    vector<Num> X = {0,1,1.5,2.4,3,4};
    vector<Num> Y = {cos(0),cos(1),cos(1.5),cos(2.4),cos(3),cos(4)};
    ofstream archivo("gra.m");
    archivo<<"#!/usr/bin/octave -qf"<<endl;
```

```
    string x = "x1 = [";
    for(int i = 0; i < X.size(); i++){
        x = x + to_string(X[i]) + ",";
    }
    x.pop_back();
    x = x + "];";
    archivo<<x<<endl;
    string y = "y1 =[";
    for(int i = 0; i < Y.size(); i++){
        y = y + to_string(Y[i]) + ",";
    }
    y.pop_back();
    y = y + "];";
    archivo<<y<<endl;
    archivo<<"plot(x1,y1,'r*')"<<endl;
    archivo<<"grid on"<<endl;
    archivo<<"hold on"<<endl;
    x = "x2 = [";
    y = "y2 = [";
    string x1 = "x3 = [";
    string y1 = "y3 = [";
    for(float i = -1; i < 6; i = i + 0.01){
        x1 = x1 + to_string(i) + ",";
        y1 = y1 + to_string(cos(i)) + ",";
        x = x + to_string(i) + ",";
        y = y + to_string(fun(i)) + ",";
    }
    x.pop_back();
    x = x + "];";
    y.pop_back();
    y = y + "];";
    x1.pop_back();
    x1 = x1 + "];";
    y1.pop_back();
    y1 = y1 + "];";
    archivo<<x<<endl;
    archivo<<y<<endl;
    archivo<<x1<<endl;
    archivo<<y1<<endl;
    archivo<<"plot(x2,y2)"<<endl;
    archivo<<"hold on"<<endl;
    archivo<<"plot(x3,y3,'g')"<<endl;
    archivo<<"pause()";
    archivo.close();
    system("chmod 755 gra.m");
    system("./gra.m");
}
```

2. Interpolación de Newton

Al igual que el de Lagrange, el Main1.cpp genera una función que es compilada en el Main2.cpp para dibujar el gráfico.

2.1. main1.cpp

```
#include <iostream>
#include <map>
#include <fstream>
#include "OperacionesMatriz.h"
#include "Punto.h"

using namespace std;

vector<Num> hallarAs(vector<Punto> &puntos){
    Matriz tabla = zeros(puntos.size(), puntos.size() + 1);
    for(int i = 0; i < puntos.size(); i++){
        tabla[i][0] = puntos[i].x;
        tabla[i][1] = puntos[i].y;
    }
    for(int i = 2; i < puntos.size() + 1; i++){
        for(int j = i - 1; j < puntos.size(); j++){
            tabla[j][i] = (tabla[j][i-1] - tabla[j-1][i-1]) / (tabla[j][0] - tabla[j-(i-1)][0]);
        }
    }
    mostrarMatriz(tabla);
    vector<Num> res;
    for(int i = 0; i < puntos.size(); i++){
```

```
        res.push_back(tabla[i][i+1]);
    }
    return res;
}

void Newton(vector<Punto> &puntos){
    auto As = hallarAs(puntos);
    ofstream archivo("funcion.h");
    archivo<<"#ifndef FUNCION_H"<<endl;
    archivo<<"#define FUNCION_H"<<endl;
    archivo<<"#include <iostream>"<<endl;
    archivo<<"#include <cmath>"<<endl;
    archivo<<"#include \"Punto.h\""<<endl<<endl;
    archivo<<"#include \"OperacionesMatriz.h\""<<endl;
    archivo<<"using namespace std;"<<endl<<endl;
    archivo<<"Num fun(Num x){"<<endl;
    archivo<<"return ";
    cout<<As.size()<<endl;
    for(int i = 0; i < As.size(); i++){
        if(i != 0) archivo<<"+";
        archivo<<to_string(As[i]);
        if(i != 0) archivo<<"*";
        string temp = " ";
        for(int j = 0; j < i; j++){
            temp += "(x-" + to_string(puntos[j].x) + ")*";
        }
        temp.pop_back();
        archivo<<temp;
    }
    cout<<"BB"<<endl;
    archivo<<";"<<endl;
    archivo<<"}"<<endl<<endl;
    archivo<<"vector<Punto> getPuntos(){"<<endl;
    archivo<<"vector<Punto> res;"<<endl;
    for(auto iter = puntos.begin(); iter != puntos.end(); ++iter){
        archivo<<"res.push_back(Punto("<<(*iter).x<<","<<(*iter).y<<"));"<<endl;
    }
    archivo<<"return res;"<<endl;
    archivo<<"}"<<endl;
    archivo<<"#endif";
    archivo.close();
    system("g++ -std=c++11 secondNewton.cpp -o runnew2");
    system("./runnew2");
}

int main(){
    vector<Punto> puntos;
    puntos.push_back(Punto(1,2));
    puntos.push_back(Punto(3,4));
    puntos.push_back(Punto(5,2));
    Newton(puntos);

}
```

2.2. main2.cpp

```
#include <iostream>
#include <fstream>
#include <algorithm>
#include "funcion.h"

using namespace std;

int main(){
    ofstream archivo("granew.m");
    archivo<<"#!/usr/bin/octave -qf"<<endl;
    auto puntos = getPuntos();
    string x = "x = [";
    string y = "y = [";
    cout<<"A"<<endl;
    for(auto iter = puntos.begin(); iter != puntos.end(); ++iter){
        x += to_string((*iter).x) + ",";
        y += to_string((*iter).y) + ",";
    }
    x.pop_back();
    y.pop_back();
    x += "];";
    y += "];";
    archivo<<x<<endl;
    archivo<<y<<endl;
    archivo<<"plot(x,y,'*')"<<endl;
    string x1 = "x = [";
    string y1 = "y = [";
    cout<<"A"<<endl;
    for(float i = -1; i <= 6; i+= 0.01){
        x1 += to_string(i) + ",";
```

```
        y1 += to_string(fun(i)) + ",";
    }
    x1.pop_back();
    y1.pop_back();
    x1 += "];";
    y1 += "];";
    archivo<<x1<<endl;
    archivo<<y1<<endl;
    cout<<"A"<<endl;
    archivo<<"hold on"<<endl;
    archivo<<"plot(x,y,'r')"<<endl;
    archivo<<"pause()"<<endl;
    archivo.close();
    system("chmod 755 granew.m");
    system("./granew.m");
}
```

3. Integración

```
#include <iostream>
#include <cmath>

using namespace std;

typedef long double Num;

Num f(Num x){
    return pow(x,3);
}

Num reglaTrapecio(Num a, Num b, int n, Num(*f)(Num)){
    Num h = (b-a)/n;
    Num suma = 0;
    for(int i = 1; i < n; i++){
        suma += f(a + i*h);
    }
    return suma * h + (h/2.0) * (f(a) + f(b));
}

Num reglaSimpson1_3(Num a, Num b, int n, Num(*f)(Num)){
    if(n %2 != 0) return -10000;
    Num h = (b-a)/n;
    Num suma1 = 0;
    Num suma2 = 0;
    for(int i = 1; i < n; i++){
        if(i %2 == 0) suma1 += 2 * (h/3) * f(a+i*h);
        else suma2 += 4 * (h/3) * f(a+i*h);
    }
    return suma1 + suma2 + (h/3) * (f(a) + f(b));
}

int main(){
    Num (*fun)(Num) = f;
    Num a;
    Num b;
    int n;
    int flag;
    cout<<"Regla Trapecio(1) - Regra Simpson 1/3 (2)->";
    cin>>flag;
    cout<<"Ingrese su a->";
    cin>>a;
    cout<<"Ingrese su b->";
    cin>>b;
    cout<<"Ingrese su n->";
    cin>>n;
    if(flag == 1) cout<<reglaTrapecio(a,b,n,f)<<endl;
    else if(flag == 2) cout<<reglaSimpson1_3(a,b,n,f)<<endl;
}
```