#include <iostream>

#include <stdlib.h>

#include <stdio.h>

#include <math.h>

#define M\_PI acos(-1.0) // 3.14159265358979323846

using namespace std;

float NewtonCodes\_Fechada\_Trapézio\_Particao(float a, float b, int num\_particao){

float Ic = 0.0, h = (a + b)/num\_particao;

int i = 0;

float xi = a, xf = a + h;

while (i < num\_particao){

Ic += ((xf - xi) / 2 ) \* (f(xi) + f(xf));

xi = xf;

xf += h;

i ++;

}

return Ic;

}

float NewtonCodes\_Fechada\_Trapézio\_Tolerancia(float a, float b, float tolerancia){

float Ic = 0.0, Ia = 0.0, h;

int i = 0, num\_ partição = 1;

float xi = a, xf = a + h, erro = xi;

while (erro > tolerância){

i = 0;

h = (a + b) / num\_particao;

while (i < num\_particao){

Ic += ((xf - xi) / 2 ) \* (f(xi) + f(xf));

xi = xf;

xf += h;

i ++;

}

erro = Ic - Ia; // vê se é assim

Ia = Ic;

num\_particao += 1;

}

return Ic;

}

float NewtonCodes\_Fechada\_Grau2\_Particao(float a, float b, int num\_particao){

float Ic = 0.0, h = (a + b)/num\_particao, xi = a, xf = a + h, h\_aux = (xi + xf)/3;

int i = 0;

float x1 = xi + h\_aux, x2 = xi + 2\*h\_aux;

while (i < num\_particao){

Ic += (h / 3) \* (f(xi) + 4\*f(xm) + f(xf));

xi = xf;

xf += h;

x1 = h\_aux + xi;

x2 += 2\*h\_aux + xi;

i ++;

}

return Ic;

}

float NewtonCodes\_Fechada\_Grau2\_Tolerancia(float a, float b, float tolerancia){

float Ic = 0.0, h = a + b;

int i = 0;

float xi = a, xf = a + h, xm = (xi + xf) / 2, erro = xi;

while (erro > tolerância){

i = 0;

h = (a + b) / num\_particao;

while (i < num\_particao){

Ic += (h / 3) \* (f(xi) + 4\*f(xm) + f(xf));

xi = xf;

xf += h;

x1 = h\_aux + xi;

x2 += 2\*h\_aux + xi;

}

erro = Ic - Ia; // vê se é assim

Ia = Ic;

num\_particao += 1;

}

return Ic;

}

float NewtonCodes\_Fechada\_Grau3\_Particao(float a, float b, int num\_particoes){

float Ic = 0.0, h = (a + b)/num\_particao, xi = a, xf = a + h, h\_aux = (xi + xf)/4;

int i = 0;

float x1 = xi + h\_aux, x2 = xi + 2\*h\_aux, x3 = 3\*h\_aux + xi;

while (i < num\_particao){

Ic += (h / 3) \* (f(xi) + 4\*f(xm) + f(xf));

xi = xf;

xf += h;

x1 = h\_aux + xi;

x2 += 2\*h\_aux + xi;

x3 = 3\*h\_aux + xi;

i ++;

}

return Ic;

}

float NewtonCodes\_Fechada\_Grau2\_Tolerancia(float a, float b, float tolerancia){

float Ic = 0.0, h = a + b;

int i = 0;

float xi = a, xf = a + h, h\_aux = (xi + xf)/4, x1 = xi + h\_aux, x2 = xi + 2\*h\_aux, x3 = 3\*h\_aux + xi;

while (erro > tolerância){

i = 0;

h = (a + b) / num\_particao;

while (i < num\_particao){

Ic += (h / 3) \* (f(xi) + 4\*f(xm) + f(xf));

xi = xf;

xf += h;

x1 = h\_aux + xi;

x2 = 2\*h\_aux + xi;

x3 = 3\*h\_aux + xi;

}

erro = Ic - Ia; // vê se é assim

Ia = Ic;

num\_particao += 1;

h\_aux = (xi + xf) / 4;

}

return Ic;

}

float NewtonCodes\_Fechada\_Grau4\_Particao(){

float Ic = 0.0, h = (a + b)/num\_particao, xi = a, xf = a + h, h\_aux = (xi + xf)/5;

int i = 0;

float x1 = xi + h\_aux, x2 = xi + 2\*h\_aux, x3 = 3\*h\_aux + xi, x4 = 4\*h\_aux + xi;

while (i < num\_particao){

Ic += (h / 3) \* (f(xi) + 4\*f(xm) + f(xf));

xi = xf;

xf += h;

x1 = h\_aux + xi;

x2 += 2\*h\_aux + xi;

x3 = 3\*h\_aux + xi;

x4 = 4\*h\_aux + xi;

i ++;

}

return Ic;

}

float NewtonCodes\_Fechada\_Grau4\_Tolerancia(float a, float b, float tolerancia){

float Ic = 0.0, h = a + b;

int i = 0;

float xi = a, xf = a + h, h\_aux = (xi + xf)/5, x1 = xi + h\_aux, x2 = xi + 2\*h\_aux, x3 = 3\*h\_aux + xi, x4 = 4\*h\_aux + xi;

while (erro > tolerância){

i = 0;

h = (a + b) / num\_particao;

while (i < num\_particao){

Ic += (h / 3) \* (f(xi) + 4\*f(xm) + f(xf));

xi = xf;

xf += h;

x1 = h\_aux + xi;

x2 = 2\*h\_aux + xi;

x3 = 3\*h\_aux + xi;

x4 = 4\*h\_aux + xi;

}

erro = Ic - Ia; // vê se é assim

Ia = Ic;

num\_particao += 1;

h\_aux = (xi + xf) / 5;

}

return Ic;

}

float NewtonCodes\_Aberta\_Trapézio(){}

float NewtonCodes\_Aberta\_Grau2(){}

float NewtonCodes\_Aberta\_Grau3(){}

float NewtonCodes\_Aberta\_Grau4(){}

float GaussLegendre\_Grau2(){}

float GaussLegendre\_Grau3(){}

float GaussLegendre\_Grau4(){}

float GaussLegendre\_Grau5(){}

void menu(){

// vê se não é melhor modularizar isso aqui e deixar bonitinho

int opc\_formulaInt, opc\_funcao, opc\_xMin, opc\_xMax, opc\_particao, num\_particao;

float a, b, tolerancia;

cout << “\n###################################################################”;

cout << “\n###################################################################”;

cout << “\n###################################################################”;

cout << “\nDigite a fórmula desejada:”;

cout << “\n1 - sen(x)”;

cout << “\n2 - cos(x)”;

cout << “\n3 - tan(x)”;

cin >> opc\_funcao;

switch(opc\_funcao){

case 1:

break;

case 2:

break;

case 3:

break;

default:

cout << “\nOpção inválida! Tente novamente.”;

}

//clear

cout << “\n###################################################################”;

cout << “\n###################################################################”;

cout << “\n###################################################################”;

cout << “\nDigite a opção de partição desejada:”;

cout << “\n1 - Tolerância”;

cout << “\n2 - Número definido de partições”;

cin >> opc\_particao;

switch(opc\_particao){

case 1:

//clear

cout << “\n###################################################################”;

cout << “\n###################################################################”;

cout << “\n###################################################################”;

cout << “\nDigite o valor da tolerância:”;

cin >> tolerancia;

break;

case 2:

//clear

cout << “\n###################################################################”;

cout << “\n###################################################################”;

cout << “\n###################################################################”;

cout << “\nDigite o número de partições:”;

cin >> num\_particao;

break;

default:

cout << “\nOpção inválida! Tente novamente.”;

}

//clear

cout << “\n###################################################################”;

cout << “\n###################################################################”;

cout << “\n###################################################################”;

cout << “\nDigite o xMínimo:”;

cout << “\n1 - 0”;

cout << “\n2 - r/2”;

cout << “\n3 - r/4”;

cout << “\n4 - r/8”;

cout << “\n5 - Outro”;

cin >> opc\_xMin;

switch(opc\_xMin){

case 1:

a = 0.;

break;

case 2:

a = M\_PI/ 2;

break;

case 3:

a = M\_PI / 4;

break;

case 4:

a = M\_PI / 8;

break;

case 5:

cout << “\nDigite o valor:”;

cin >> a;

break;

default:

cout << “\nOpção inválida! Tente novamente.”;

}

//clear

cout << “\n###################################################################”;

cout << “\n###################################################################”;

cout << “\n###################################################################”;

cout << “\nDigite o x\_Máximo:”;

cout << “\n1 - 0”;

cout << “\n2 - r/2”;

cout << “\n3 - r/4”;

cout << “\n4 - r/8”;

cout << “\n5 - Outro”;

cin >> opc\_xMax;

switch(opc\_xMax){

case 1:

b = 0.;

break;

case 2:

b = M\_PI / 2;

break;

case 3:

b = M\_PI / 4;

break;

case 4:

b = M\_PI / 8;

break;

case 5:

cout << “\nDigite o valor:”;

cin >> b;

break;

default:

cout << “\nOpção inválida! Tente novamente.”;

}

//clear

cout << “\n###################################################################”;

cout << “\n###################################################################”;

cout << “\n###################################################################”;

cout << “\nInforme a fórmula de integração desejada:”;

cout << “\n1 - Newton-Codes grau 1 - 4 FÓRMULA FECHADA”;

cout << “\n2 - Newton-Codes grau 1 - 4 FÓRMULA ABERTA”;

cout << “\n3 - Gauss-Legendre 2 - 5”;

cin >> opc\_formulaInt;

switch(opc\_formulaInt){

case 1:

//clear

cout << “\n###################################################################”;

cout << “\n###################################################################”;

cout << “\n###################################################################”;

cout << “\nInforme o grau do polinômio”;

cin >> grau;

switch(grau){

case 1:

break;

case 2:

break;

case 3:

break;

case 4:

break;

default:

cout << “\nOpção inválida! Tente novamente.”;

break;

}

break;

case 2:

//clear

cout << “\n###################################################################”;

cout << “\n###################################################################”;

cout << “\n###################################################################”;

cout << “\nInforme o grau do polinômio”;

cin >> grau;

switch(grau){

case 1:

break;

case 2:

break;

case 3:

break;

case 4:

break;

default:

cout << “\nOpção inválida! Tente novamente.”;

break;

}

break;

case 3:

//clear

cout << “\n###################################################################”;

cout << “\n###################################################################”;

cout << “\n###################################################################”;

cout << “\nInforme o grau do polinômio”;

cin >> grau;

switch(grau){

case 2:

break;

case 3:

break;

case 4:

break;

case 5:

break;

default:

cout << “\nOpção inválida! Tente novamente.”;

break;

}

break;

default:

cout << “\nOpção inválida! Tente novamente.”;

}

}

void main(){

}

https://www.ufrgs.br/numerico/livro/in-quadratura\_de\_gauss-legendre.html