**Documentación proyecto de Métodos Numéricos**

* El lenguaje de programación utilizado es C++

El proceso para el uso del programa es seleccionar el método del cual quieras hacer uso, ingresando los datos que se piden para poder proporcionar la respuesta correcta.

Dentro del código se crearon funciones preestablecidas, por lo tanto, el usuario no ingresa la función y lo único que evalúa son los limites, las cuales son: Trapezoidal, trapezoidal múltiple, Simpson 1/3, Simpson ¾, Bisección, Secante, Newton-Raphson.

Las funciones preestablecidas se encuentran en la parte de arriba.

**CODIGO:**

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

/\*

PROYECTO FINAL DE METODOS NUMERICOS

INTEGRANTES:

Manuel Eduardo Torres Magdaleno A01066869

Marielisa Madero Alvarez A01282353

Cristina Nohemí de León A01282017

Danya Morelos A00820080

\*/

#define f1(x)(1.0 \* pow(x,4) - 8.0 \* pow(x,3) - 35.0 \* pow(x , 2) + 450.0 \* x - 1001)//secante

#define fnr(x) (x \* exp(cos(x)) / 1.5 - 1) ///Raphson

#define f\_derivada(x) (exp(cos(x)) \* (1 - x \* sin(x)) / 1.5) ///Raphson

#define N 30

void tabula(double a, double b, int intervalos){

int puntos = intervalos + 1;

double ancho = (b - a) / intervalos;

cout << "\n\tx\t\tf(x) " << endl;

for (int i = 0; i < puntos; i++) {

cout << "\t" << a << "\t\t" << fnr(a) << endl;

a += ancho;

}

}

float f(float(x)){

return (3\*pow(x,3)+ 5\*pow(x,2)-(3\*x)-10);

}

float g(float(x)){

return (3\*pow(x,2)+2\*x-4);

}

float h(float(x)){

return (6\*x+4);

}

double funcion(double dx){ ///simpson

return 1.7 + 50.0 \* dx - 70.0 \* pow(dx ,2) + 65.0 \* pow(dx , 3) + 100.0 \* pow(dx , 4);

}

void Menu(){

cout << "\nSeleccione un metodo: " << endl;

cout << " 0.- Salir" << endl;

cout << " 1.- Trapezoidal Simple" << endl;

cout << " 2.- Trapezoidal Multiple" << endl;

cout << " 3.- Simpson 1/3" << endl;

cout << " 4.- Simpson 3/8" << endl;

cout << " 5.- Simpson Multiple" << endl;

cout << " 6.- Secante" << endl;

cout << " 7.- Newton-Raphson" << endl;

cout << " 8.- Biseccion" << endl;

cout << " 9.- Lagrange" << endl;

cout << " 10.- Eliminacion Gausseana" << endl;

cout << " 11.- Newton (diferencias divididas) " << endl;

cout << " 12.- Montante" << endl;

cout << " 13.- Ajuste de Curvas Potencial" << endl;

cout << " 14.- Ajuste de Curvas Exponencial" << endl;

cout << " 15.- Ajuste de Curvas Polinomial" << endl;

}

void llenaMatriz(int n1, float mat[N][N]){

for (int i = 1; i <= n1; i++){

for(int j = 1; j <= n1 + 1; j++){

cout << "a[" << i << "][" << j << "] = ";

cin >> mat[i][j];

}

}

}

void llenaMatrizAjuste(int n1, float n2, float n3, float n4, float n5, float n6, float mat[N][N]){

mat[1][1] = n1;

mat[1][2] = n2;

mat[1][3] = n3;

mat[2][1] = n4;

mat[2][2] = n5;

mat[2][3] = n6;

}

void llenaMatrizP(int n1, float n2, float n3, float n4, float n5, float n6, float n7, float n8, float mat[N][N]){

mat[1][1] = n1;

mat[1][2] = n2;

mat[1][3] = n3;

mat[1][4] = n6;

mat[2][1] = n2;

mat[2][2] = n3;

mat[2][3] = n4;

mat[2][4] = n7;

mat[3][1] = n3;

mat[3][2] = n4;

mat[3][3] = n5;

mat[3][4] = n8;

}

void muestramat(int n, float x[N]){

for (int i = 1; i <= n; i++)

cout << "x[" << i << "] = " << x[i] << endl;

}

void GaussJordan(int n, float mat[N][N], float x[N]){

float aux;

for(int iA = 1; iA <= n; iA++ ){

aux = mat[iA][iA];

for (int j = iA; j <= n + 1; j++ ){

mat[iA][j] = mat[iA][j] / aux;

}

for (int j = 1; j <= n; j++ ){

if(j != iA ){

aux = mat[j][iA];

for (int k = iA; k <= n + 1; k++){

mat[j][k] = (mat[j][k]) - ((mat[iA][k]) \* aux);

}

}

}

}

for (int i = 1; i <= n; i++ ){

x[i] = mat[i][n + 1];

}

}

void montante(){

int n, i, j,w;

cout << "Numero de ecuaciones? ";

cin >> n;

double A[n][n], B[n][n], mata[n], matb[n];

for (i= 0; i < n; i++){

cout << "Esccriba incognitas\n";

for(j = 0; j < n; j++){

cin >> A[i][j];

B[i][j]=0;

}

matb[i]=0;

}

cout<<"Escriba resultados" << endl;

for (i = 0; i < n; i++){

cin >> mata[i];

matb[i] = mata[i];

}

for(i = 0; i < n ; i++){

for (j = 0; j < n ; j++){

B[i][j]=A[i][j];

}

for (j = 0; j < n ; j++){

if (j == i){

}

else {

B[j][i]=0;

}

}

for (j = 0; j < n; j++){

B[j][j]=A[i][i];

}

for (j = 0; j < n; j++){

if (j == i){}

else{

matb[j] = (mata[j]\* A[i][i]) - (A[j][i] \* mata[i]);

if (i == 0){

}

else {

matb[j]= matb[j] / A[0][0];

}

}

for (w = i+1 ; w<n ; w++){

if (j == i){

}

else {

B[j][w]= (A[j][w]\* A[i][i]) - (A[j][i] \* A[i][w]);

if (i == 0){

}

else {

B[j][w]=B[j][w] / A[0][0];

}

}

}

}

for (j = 0; j < n ; j++){

for (w = 0; w<n; w++){

A[j][w] = B[j][w];

}

mata[j] = matb[j];

}

} ///Fin for

cout << "Los valores de x son : ";

for(i = 0; i<n ; i++){

cout<< "X" << i << " = " << matb[i]/B[i][i] <<endl;

}

}

void Trapezoidal(){

double suma = 0, f0, f1, a, b;

cout <<"Integracion por metodo Trapezoidal simple\n\n\n";

cout << "Funcion: cos(x) + 5X^3\n\n";

cout << "Limite inferior (a) ";

cin >> a;

cout << "Limite superior (b) ";

cin >> b;

f0 = cos(a) +(5 \* a\*a\*a);

f1 = cos(b) +(5 \* b\*b\*b);

cout << "\nResultado: " << (b-a)\*((f0 + f1)/2) << endl;

}

void TrapezoidalMultiple(){

int N1;

double h ,a ,b , suma = 0, F, xi, res;

cout <<"Integracion por metodo Trapezoidal multiple\n\n\n";

cout << "Funcion: cos(x) + 5X^3\n\n";

cout <<"Limite inferior (a) ";

cin >> a;

cout << "Limite superior (b): ";

cin >> b;

cout <<"Numero de iteraciones? ";

cin >> N1;

h = (b-a)/N1;

for(int i=0; i<=N1; i++){

xi = a + i\*h;

F = cos(xi) + (5 \* xi\*xi\*xi);

suma += F;

res = suma\*h;

}

cout<< "h = " << h << endl;

cout<<" Resultado de la integral : "<< res << endl;

}

void simpson(){

float h, a, b, FuncionS, suma = 0, f0 = 0, f1 = 0, f2 = 0;

cout << " Formula f(x)= 3x^3 + 5x^2 - 3x - 10 " << endl << endl;

cout << "Limite inferior (a) ";

cin >> a;

cout << "Limite superior (b) ";

cin >> b;

h = (b-a)/2.0;

while(true){

suma += h;

if (f0 == 0){

f0 = f(suma);

}

else

if (f1 == 0){

f1 = f(suma);

}

else

if (f2 == 0){

f2 = f(suma);

break;

}

}

cout << "\nResultado: " << (b-a) \* ((f0 + 4\*(f1) + f2)/6.0) << endl;

///Simp13 \_ 2\*h\* (f0\_4\*f1\_f2) / 6

}

void simpsonTresOctavos(){

float a, b, d, n, I = 0, J = 0, A, K = 0, E = 0;

cout << " Formula f(x)= 3x^3 + 5x^2 - 3x - 10 " << endl << endl;

cout << "Limite inferior a ";

cin >> a;

cout << "Limite superior b ";

cin >> b;

cout << "\nNumero de intervalos : ";

cin >> n;

cout << endl;

d = (b - a)/n;

for(int i = 1; i < n; i++) {

I += f(a + (i \* d));

}

for(int i = 3; i < n - 1; i++){

if((i % 3) == 0){

J += f(a + (i \* d));

}

}

A = 3 \*(d / 8) \* (f(a) + (3 \* I) - J + f(b));

cout << "Resultado de la integral: " << A << endl;

E = -(d \* d \* d \* d \* d \* 6 \* 3 / 80);

cout << "El error total es : " << E << endl;

}

void simpsonMultiple(){

double dArea, dS1 = 0, dS2 = 0, dx , dN, dA, dB, di , dH;

cout << "Funcion: 1.7 + 50x - 70x^2 + 65x^3 + 100x^4\n\n";

cout << "Ingrese el numero de Trapecios: ";

cin >> dN;

cout << "Limite inferior a: ";

cin >> dA;

cout << "Limite superior b ";

cin >> dB;

dx = dA;

dH = (dB - dA) / dN;

cout<< "h: " << dH << endl;

if(dN == 2) {

dx += dH;

dS1 += funcion(dx);

dArea = (dH / 3) \* (funcion(dA) + funcion(dB) + 4 \* dS1 + 2 \* dS2);

cout << "\nEl area es : " << dArea;

cout << endl;

}

else{

for(di = 1; di <= ((dN / 2 ) - 1); di++) {

dx += dH;

dS1 += funcion(dx);

dx += dH;

dS2 += funcion(dx);

}

dx += dH;

dS1 += funcion(dx);

dArea =((funcion(dA) + funcion(dB) + 4 \* dS1 + 2 \* dS2) \*(dH / 3));

cout << "\nLa respuesta es : " << dArea;

}

}

float Potencia(int n, float Num){

float res = 1;

for(int A=1; A<=n; A++){

res \*= Num;

}

return res;

}

void llenaArray(int iN, float iX[100]){

for(int i = 0; i < iN ; i++){

cin >> iX[i];

}

}

float SumatoriaLineal(int n, float iX[100]){

float iSum = 0;

for (int i = 0; i < n; i++){

iSum += iX[i];

}

return iSum;

}

float SumatoriaSquared(int n, float iX[100]){

float iSum = 0;

for (int i = 0; i < n; i++){

iSum += (iX[i]\*iX[i]);

}

return iSum;

}

float SumatoriaLn(int n, float iY[100]){

float iSum = 0, iSacarLog = 0, iLog = 0;

for (int i = 0; i < n; i++){

iSacarLog = iY[i];

iLog = logf (iSacarLog);

iSum += iLog;

}

return iSum;

}

float SumatoriaLn2(int n, float iY[100]){

float iSum = 0, iSacarLog = 0, iLog = 0;

for (int i = 0; i < n; i++){

iSacarLog = iY[i];

iLog = logf (iSacarLog);

iSum = iSum + (iLog\*iLog);

}

return iSum;

}

float SumatoriaXln(int n, float iY[100], float iX[100]){

float iSum = 0, iSacarLog = 0, x = 0;

for (int i = 0; i < n; i++){

iSacarLog = iY[i];

x = iX[i];

iSum = iSum + logf(iSacarLog)\*x;

}

return iSum;

}

float Sumatoria2logs(int n, float iY[100], float iX[100]){

float iSum = 0, iSacarLogx = 0, iSacarLogy = 0;

for (int i = 0; i < n; i++){

iSacarLogx = iX[i];

iSacarLogy = iY[i];

iSum = iSum + logf(iSacarLogx)\* logf(iSacarLogy);

}

return iSum;

}

float SumatoriaY(int n, float iY[100]){

float iSum = 0;

for (int i = 0; i < n; i++){

iSum = iSum + iY[i];

}

return iSum;

}

float SumatoriaTres(int n, float iX[100]){

float iSum = 0;

for (int i = 0; i < n; i++){

iSum = iSum + (iX[i]\*iX[i]\*iX[i]);

}

return iSum;

}

float SumatoriaCuatro(int n, float iX[100]){

float iSum = 0;

for (int i = 0; i < n; i++){

iSum = iSum + (iX[i]\*iX[i]\*iX[i]\*iX[i]);

}

return iSum;

}

float SumatoriaXY(int n, float iX[100], float iY[100]){

float iSum = 0;

for (int i = 0; i < n; i++){

iSum = iSum + (iX[i]\*iY[i]);

}

return iSum;

}

float SumatoriaX2Y(int n, float iX[100], float iY[100]){

float iSum = 0;

for (int i = 0; i < n; i++){

iSum = iSum + (iX[i]\*iX[i]\*iY[i]);

}

return iSum;

}

void PreparaSistema(int Ord, int Dat, float Sist[][102], float Val[][102]){

int A,B,C,Exp;

float suma,termino;

for(A=1;A<=Ord;A++) for(B=1;B<=Ord;B++){

suma=0;

Exp=A+B-2;

for(C=1;C<=Dat;C++) {

termino=Val[0][C];

suma=suma+Potencia(Exp,termino);

}

Sist[A][B]=suma;

}

for(A=1;A<=Ord;A++) {

suma=0;

Exp=A-1;

for(C=1;C<=Dat;C++)

{

termino=Val[0][C];

suma=suma+Val[1][C]\*Potencia(Exp,termino);

}

Sist[A][Ord+1]=suma;

}

}

void llenaMat2(int iN, float iX[100], char cNom[10]){

for(int i = 1; i <= iN ; i++){

cout << cNom << "[" << i << "] : ";

cin >> iX[i];

}

}

void escribirMatriz(int iN , int iM, float dA[50][50], char cNom[10]){

cout << "Matriz" << cNom << endl;

for (int i = 1; i <= iN; i++){

cout << endl;

for(int j = 1; j <= iM; j++){

cout << setw(10) << dA[i][j];

}

}

cout << endl;

}

void escpol(int iN, float dA[50], char cNom[10]){

cout << "Polinomio" << cNom << " = ";

for(int i = 0; i <= iN; i++){

cout << dA[i] << "X^" << i;

if (i<iN){

cout << "+ ";

}

}

cout << endl;

}

void Muestravec(int iN, float dX[50], char cNom[10]){

for(int iC = 1; iC <= iN ; iC++){

cout << endl;

cout << cNom << "[" << iC << "] : " << dX[iC];

}

cout << endl;

}

void DifDivNewton(int iN, float dX[100], float dY[100], float dA[100]){

float DD[50][50], dP[50][50], dVX[50][50], dB[50], P[50][50];

for(int i = 1; i <= iN ;i++){

DD[i][1] = i - 1;

DD[i][2] = dX[i];

DD[i][3] = dY[i];

}

for(int iR = 1; iR <= iN; iR++){

for(int i = 1; i <= iN - iR; i++){

DD[i][iR+3] = ((DD[i+1][2+iR])- (DD[i][2 + iR]))/((DD[i + iR][2]) -(DD[i][2]));

}

}

escribirMatriz(iN, iN+2,DD, "DD");

for (int i = 1; i <= iN; i++){

dB[i] = DD[1][i+2];

}

cout << "Valores de B" << endl;

Muestravec(iN, dB, "dB");

for(int i = 1; i <= iN ; i++){

for(int iR = 1; iR <= iN; iR++){

P[i][iR] = 0;

}

}

for(int i = 1; i <= iN ; i++){

P[i][1] = 1;

}

for(int i = 2; i <= iN; i++){

for(int iR = 2; iR <= i; iR++){

P[i][iR] = ((P[i -1][iR - 1])\* (-1 \* dX[i -1 ]))+ P[i - 1][iR];

}

}

escribirMatriz(iN, iN, P, "Polinomios");

for(int i = 1; i <= iN; i++){

for(int iR = 1; iR <= iN; iR++){

P[i][iR] = P[i][iR] \* dB[i];

}

}

escribirMatriz(iN, iN , P , "Polinomios");

for(int iR = 1; iR <= iN; iR++){

dA[iR -1] = 0;

for(int i = iR; i <= iN ; i++) {

dA[iR -1] = dA[iR - 1] + P[i][i+1 - iR];

}

}

escpol(iN - 1, dA, "P(x)" );

}

//BISECCION

double FunBi(double x){

return ((3.5 \*pow(x,3)) - ((10 \* pow(x,2))/2));

}

double biseccion(double a, double b, double tol, int maxlter){

double c;

int res = 0;

do {

c = (a+b)/2;

if (FunBi(a)\*FunBi(c) < 0){

b = c;

}

else{

a = c;

}

cout << res << "\t" << a << "\t" << b << "\t" << c << "\t" <<FunBi(c)<<endl;

res++;

}

while((fabs(FunBi(c)) >tol) && (res < maxlter));

return c;

}

//SECANTE

double secante(double dX0, double dX1, double des, int iIter){

double dX2, dea, dY0, dY1, dPend;

cout << "iteracion" << setw(10) << "xi-1" << setw(10) << "xi" << setw(10) << "xi+1" << setw(10) << "Error" << setw(10)<< "f(xi-1)"<< setw(10) << "f(xi)" << setw(10) << "pend" << endl;

for(int i = 1; i <= iIter; i++){

dY0 = f1(dX0);

dY1 = f1(dX1);

dPend = (dY1 - dY0)/(dX1 - dX0);

dX2 = dX1 - (dY1/dPend);

dea = fabs((dX2 - dX1)/(dX2));

cout << setw(5) << i << setw(15)<< dX0 << setw(10) << dX1 << setw(10)<< dX2 << setw(10)<< dea << setw(10) << dY0 << setw(10)<< dY1<< setw(10) << dPend << endl;

dX0 = dX1;

dX1 = dX2;

if(dea < des) {

cout << "El metodo converge a las " << i << " iteraciones" << endl;

return dX2;

break;

}

}

if(dea > des)

cout << "No se puede" << endl;

return 0;

}

//NEWTON - RAPHSON

void NewtonRaphson(){

double a, b, tolerancia, x0, x1, error;

int iteracion;

bool converge = true;

cout << setprecision(10);

cout << "\nCalculo de las raices de una funcion aplicando el metodo de Newton-Raphson\n";

cout << "\nIngrese intervalos a y b -> (a, b):\n\n";

cout << "a = ";

cin >> a;

cout << "b = ";

cin >> b;

tabula(a, b, 6); // Se tabulan los valores de f para INTERVALOS intervalos

cout << "\nEscoja el punto inicial adecuado: x0 = ";

cin >> x0; // Primera aproximacion

cout << "Tolerancia = ";

cin >> tolerancia;

cout << "\nAproximacion inicial:\n";

cout << "x0 = " << x0 << "\n";

cout << "f(x0) = " << fnr(x0) << "\n";

cout << "f'(x0) = " << f\_derivada(x0) << endl;

iteracion = 1;

do {

if (iteracion > 100){

converge = false;// Si se pasa la cantidad de iteraciones permitidas, se sale

break;

}

else{

x1 = x0 - fnr(x0) / f\_derivada(x0);// Siguiente aproximacion

error = fabs(x1 - x0);

// Se imprimen los valores de la siguiente aproximacion x1, f(x1), f\_derivada(x1), error

cout << "\nIteracion #" << iteracion << endl;

cout << "x" << iteracion << " = " << x1 << "\n";

cout << "f(x" << iteracion << ") = " << f(x1) << "\n";

cout << "f'(x" << iteracion << ") = " << f\_derivada(x1) << "\n";

cout << "error = " << error << endl;

if (error <= tolerancia){

converge = true;

break;

}

else{ // Si no se cumple el criterio, pasa a la siguiente iteracion

x0 = x1;

iteracion++;

}

}

}while (true);

if (converge){

cout << "\n\nPara una tolerancia de " << tolerancia << " la raiz de f es: " << x1 << endl;

}

else {

cout << "\n\nPasa la maxima cantidad de iteraciones permitidas" << endl;

}

}

double ProcesoInterpolacion(double X, int sizeDatos, double\* arrX, double\* arrayY) {

double fx = 0, L;

for(int i=0; i<sizeDatos ;i++){

double aproximado =arrayY[i];

for(int j=0; j<sizeDatos; j++){

if(i!=j){

aproximado=(aproximado\*(X-arrX[j]))/(arrX[i]-arrX[j]);

}

}

fx+= aproximado;

}

return fx;

}

void InterpolacionLagrange(){

int sizeDatos;

double X, f1x;

cout << "Cantidad de elementos: ";

cin >> sizeDatos;

cout << "Ingresa los "<< sizeDatos << " de X:" << endl;

double arrX[sizeDatos];

double arrY[sizeDatos];

for(int i=0; i<sizeDatos; i++){

cin >> arrX[i];

}

cout<<"Ingresa los " << sizeDatos <<" de Y:"<<endl;

for(int i=0; i<sizeDatos; i++){

cin >> arrY[i];

}

cout << "¿Que valor quieres evaluar en la función?"<<endl;

cin >> X;

f1x = ProcesoInterpolacion(X, sizeDatos, arrX, arrY);

cout << "f(" << X << ") = " << f1x;

}

int main(){

int opcion, numeroIteracciones, n1, iD, iNumE = 2;;

float mat[N][N], x[N];

double errorMenor, LimiteInferior, LimiteSuperior, Raiz, dX0, dX1;

do{

Menu();

cin >> opcion;

switch(opcion){

case 1:

Trapezoidal();

break;

case 2:

TrapezoidalMultiple();

break;

case 3:

simpson();

break;

case 4:

simpsonTresOctavos();

break;

case 5:

simpsonMultiple();

break;

case 6:{

int iteracionI;

double des, dX0, dX1, dRaiz;

cout << "Punto inicial 1 (xi-1)\n";

cin >> dX0;

cout << "Punto inicial 2 (xi)" << endl;

cin >> dX1;

cout << "Error " << endl;

cin >> des;

cout << "Numero de iteraciones" << endl;

cin >> iteracionI;

cout.precision(4);

dRaiz = secante(dX0,dX1, des, iteracionI);

cout << "La raiz es: " << setprecision(10) << dRaiz << endl << endl;

break;

}

case 7:{

NewtonRaphson();

break;

}

case 8:{

double limA, limB, tol, raiz;

int maxlter;

cout << "Metodo de Biseccion\n\n";

cout << "Limite Inferior: ";

cin >> limA;

cout << endl;

cout << "Limite superior: ";

cin >> limB;

cout << endl;

cout << "Error: ";

cin >> tol;

cout << endl;

cout << "Iteraccion: ";

cin >> maxlter;

cout << endl;

cout << "La raiz es: " << biseccion(limA, limB, tol,maxlter) << "%" << endl;

break;

}

case 9:

///Lagrange

InterpolacionLagrange();

break;

case 10:

cout << "Numero de ecuaciones: ";

cin >> n1;

cout << "Introduce los valores de la matriz: ";

llenaMatriz(n1, mat);

GaussJordan(n1, mat, x);

cout << "Solucion " << endl;

muestramat(n1, x);

break;

case 11:{

int iM, iN;

float dX[100], dY[100], dA[100];

cout << "Numero de puntos : " << endl;

cin >> iN;

cout << "Vector x" << endl;

llenaMat2(iN, dX, "X");

cout << "Vector y" << endl;

llenaMat2(iN, dY, "Y");

DifDivNewton(iN, dX, dY, dA);

break;

}

case 12:

montante();

break;

case 13:{

// Ajuste de Curvas Potencial

float sumLnx, sumLnx2, sumLnyy, sumlnxlny;

float eX[100], eY[100];

cout << "Numero de datos: " << endl;

cin >> iD;

cout << "Dame los datos de 'X': " << endl;

llenaArray(iD,eX);

cout << "Dame los datos de 'Y': " << endl;

llenaArray(iD,eY);

sumLnx = SumatoriaLn(iD, eX);

sumLnx2 = SumatoriaLn2(iD, eX);

sumLnyy = SumatoriaLn(iD, eY);

sumlnxlny = Sumatoria2logs(iD, eY, eX);

cout << sumLnx2 << " " << sumLnx << " " << sumlnxlny << " " << sumLnyy << endl;

llenaMatrizAjuste(iD, sumLnx, sumLnyy, sumLnx, sumLnx2, sumlnxlny, mat);

GaussJordan(iNumE, mat, x);

cout << "Solucion donde x1= a0 y x2 = a1" << endl;

muestramat(iNumE, x);

break;

}

case 14:{

// Ajuste de Curvas Exponencial

int iT, iNumEc = 2;

float sumX, sumX2, sumLny, sumXlny;

float aX[100], aY[100];

cout << "Numero de datos: " << endl;

cin >> iT;

cout << "Dame los datos de 'X': " << endl;

llenaArray(iT,aX);

cout << "Dame los datos de 'Y': " << endl;

llenaArray(iT,aY);

sumX = SumatoriaLineal(iT, aX);

sumX2 = SumatoriaSquared(iT, aX);

sumLny = SumatoriaLn(iT, aY);

sumXlny = SumatoriaXln(iT, aY, aX);

llenaMatrizAjuste(iT, sumX, sumLny, sumX, sumX2, sumXlny, mat);

GaussJordan(iNumEc, mat, x);

cout << "Solucion donde x1= a0 y x2 = a1" << endl;

muestramat(iNumEc, x);

break;

}

case 15:{

// Ajuste de Curvas Polinomial

int iP, iNumEcu = 3;

float sumX, sumX2, sumX3, sumX4, sumY, sumXY, sumX2Y;

float pX[100], pY[100];

cout << "Numero de datos: " << endl;

cin >> iP;

cout << "Dame los datos de 'X': " << endl;

llenaArray(iP, pX);

cout << "Dame los datos de 'Y': " << endl;

llenaArray(iP, pY);

sumX = SumatoriaLineal(iP, pX);

sumX2 = SumatoriaSquared (iP, pX);

sumX3 = SumatoriaTres(iP, pX);

sumX4 = SumatoriaCuatro(iP, pX);

sumY = SumatoriaY(iP, pY);

sumXY = SumatoriaXY(iP, pX, pY);

sumX2Y = SumatoriaX2Y(iP, pX, pY);

llenaMatrizP(iP, sumX, sumX2, sumX3, sumX4, sumY, sumXY, sumX2Y, mat);

GaussJordan(iNumEcu, mat, x);

cout << "Solucion donde x1= a0 y x2 = a1" << endl;

muestramat(iNumEcu, x);

break;

}

}

}

while (opcion != 0);

system("PAUSE");

}