**Ministerul Educației, Culturii și Cercetării**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică și Microelectronică**

**Departamentul Ingineria Software și Automatică**

**Raport**

Lucrarea de laborator nr.1

Disciplina: Metode și modele de calcul

Tema:.

**Efectuat**: st.gr.TI-207 Rusu Cătălin.

**Verificat**: conf. univ. dr. Dohotaru Leonid

Chișinău 2021

**Scopul lucrării:**

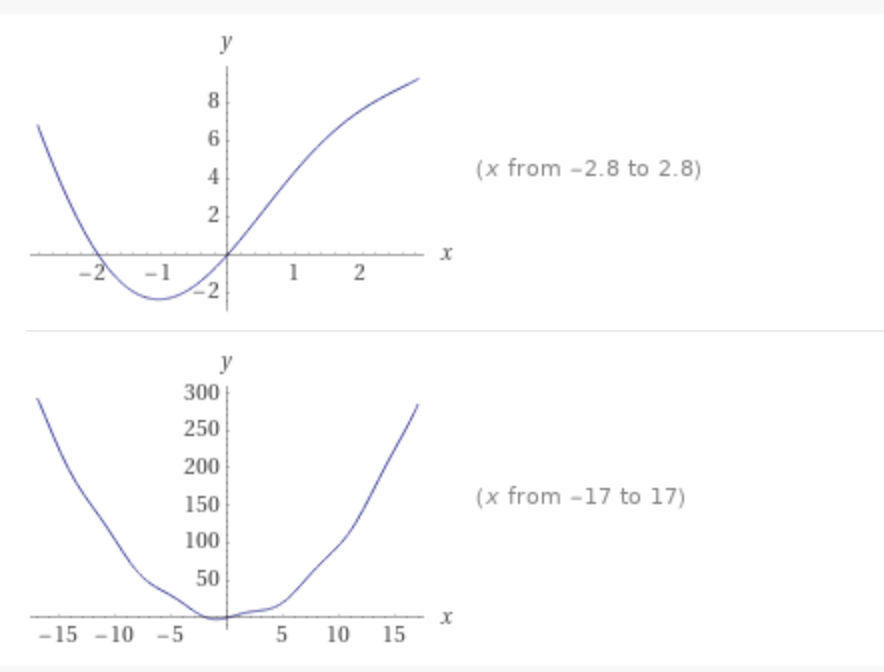
1. Rezolvarea numerică a ecuației algebrice și transcendente

**Obiectivele lucrări:**

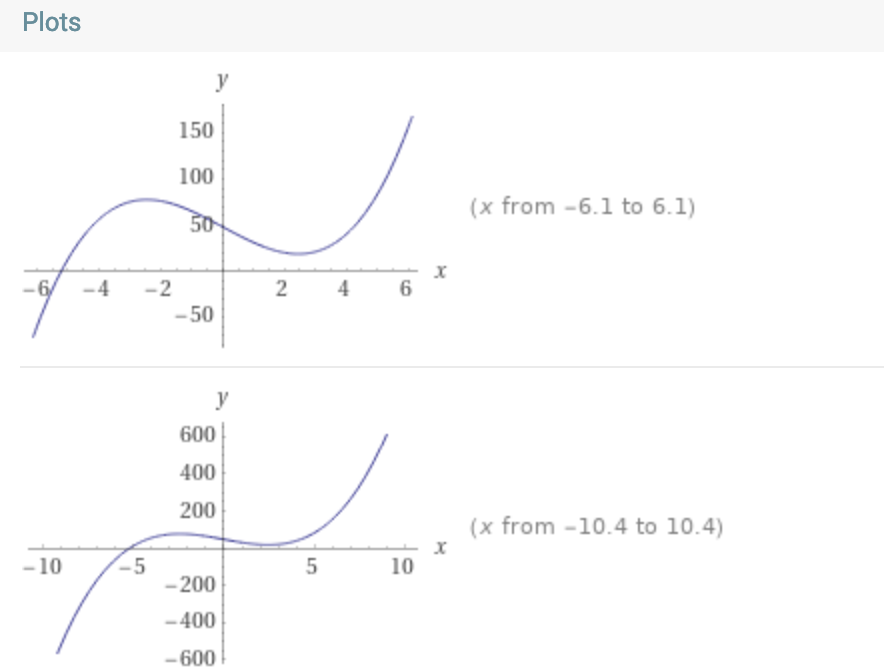
1. Să se separe toate rădăcinile ecuației f(x) = 0 unde y = f(x) de variabilă reală.
2. Să se determine o rădăcină reală a ecuației date cu ajutorul metodelor înjumătățiri intervalului cu o eroare mai mică decât ε = 10^-2.
3. Să se precizeze rădăcina obținută cu exactitatea ε = 10^-6 cu ajutorul metodelor:
4. Metoda aproximațiilor succesive.
5. Metoda secantelor.
6. Metoda tangentelor (Newton).
7. Să se compare rezultatele luând în considerație numărul de iterații pentru funcție și derivată.

**Problemă dată spre rezolvare:**

Varianta 17:



b)



**Calculul aproximativ al radacinilor**

Metoda injumatatirii intervalului

Consideram functia 𝑓 ( 𝑥 ) = 0, unde functia 𝑓 ( 𝑥 ) este continua pe intervalul [a,b], are o singura radacina reala si 𝑓 ( 𝑎 ) ∗ 𝑓 ( 𝑏 ) < 0.

b−a Se calculeaza c = a + a , daca c=0, atunci c este radacina cautata, in caz contrar :

daca 𝑓 ( 𝑎 ) ∗ 𝑓 ( 𝑏 ) < 0 => 𝑎 = 𝑎; 𝑏 = 𝑐 ;

daca 𝑓 ( 𝑎 ) ∗ 𝑓 ( 𝑏 ) > 0 => 𝑎 = 𝑐 ; 𝑏 = 𝑏 ;

Iteratiile se repeta pina cind este respectata conditia (b-a)<ε unde ε este eroarea

Metoda aproximatiilor succesive

Pentru a folosi Metoda aproximatiilor succesive, e necesar de pus ecuatia sub forma 𝑥 = 𝜑(𝑥). Pentru ca metoda sa convearga catre radacina e necesar de a se respecta conditia de convergenta :

Functia 𝜑(𝑥) e derivabila si derivata sa 𝜑′(𝑥) satisface inegalitatea |𝜑′ ( 𝑥 ) | ≤ 𝛼 < 1, oricare ar fi 𝑥 ∈ [ 𝑎, 𝑏 ] .

Metoda lui Newton(tangentelor)

Presupunem ecuatia 𝑓 ( 𝑥 ) = 0, care admite o singura solutie reala pe intervalul [a,b], iar derivatele 𝑓 ′ (𝑥) si 𝑓 ′′ ( 𝑥 ) pastreaza un semn constant pe intervalul [a,b].

Formula pentru metoda tangentelor este :

𝑓(𝑥 𝑘 ) 𝑘+1 = 𝑥 𝑘 –x

, 𝑘 = 0,1,2, …𝑓′(𝑥 𝑘 )

Pentru a asigura convergenta procesului iterational e necesar ca din valorile a si b sa fie aleasa in calitate de solutie initiala 𝑥 0 acea valoare pentru care are loc inegalitatea 𝑓 ( 𝑥 ) ∗ 𝑓 ′′ ( 𝑥 ) > 0

Metoda secantelor

Metoda secantelor poate fi dedusa din metoda lui Newton. Formula iteativa pentru calcularea aproximativa a solutiilor prin metoda secantelor este :

𝑥 𝑘 − 𝑥 𝑘−1 𝑥 𝑘+1 = 𝑥 𝑘 − 𝑓 ( 𝑥 𝑘 ) ∗ , 𝑘 = 0,1, … 𝑓 ( 𝑥 𝑘 ) − 𝑓(𝑥 𝑘−1 )

La fel ca pentru Metoda lui Newton in calitate de solutie initiala 𝑥 0 vom alege acea valoare pentru care are loc inegalitatea :

𝑓 ( 𝑥 ) ∗ 𝑓 ′′ ( 𝑥 ) > 0

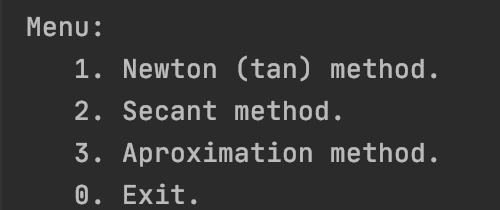
Finisarea procesului iterational are loc atunci cind aleasa in calitate de solutie initiala 𝑥 0 acea valoare pentru care are loc inegalitatea |𝑥 𝑘+1 − 𝑥 𝑘 | ≤ 𝜀, unde 𝜀 este eroarea.

**Listningul programului:**

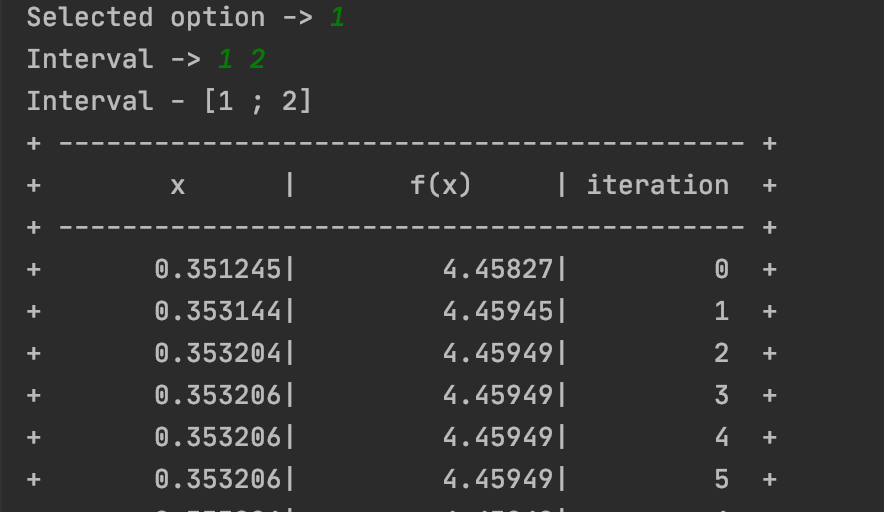
#include <iostream>  
#include <math.h>  
#include <iomanip>  
  
using namespace std;  
  
#define eps 0.00001  
  
void displayMenu();  
void methods(int selectedOption);  
  
void tangentMethod();  
double newtonFunc(double x);  
double newtonFunc1(double x);  
  
void secantMethod();  
double secantFunction(double x);  
  
void aproximationMethod();  
double aproximationFunction(double x);  
  
int iterationsNumber=200;  
int main(){  
 displayMenu();  
 return 0;  
}  
  
/\*  
 \* aici va fi meniu  
 \*/  
void displayMenu(){  
 int selectedOption;  
 cout << endl;  
 cout << "Menu:" << endl;  
 cout << " 1. Newton (tan) method." << endl;  
 cout << " 2. Secant method." << endl;  
 cout << " 3. Aproximation method." << endl;  
 cout << " 0. Exit." << endl;  
 cout << "Selected option -> ";  
 cin >> selectedOption;  
 methods(selectedOption);  
}  
  
  
void methods(int selectedOption){  
 switch (selectedOption){  
 case 1:  
 tangentMethod();  
 displayMenu();  
 break;  
 case 2:  
 secantMethod();  
 displayMenu();  
 break;  
 case 3:  
 aproximationMethod();  
 displayMenu();  
 break;  
 case 0:  
 exit(1);  
 default:  
 cout << "Error - Undefined option..." << endl;  
 displayMenu();  
 }  
}  
  
  
/\*  
 \* metoda aproximării  
 \*/  
void aproximationMethod(){  
 unsigned char i;double initialPoint, point;  
 cout << "Initial point -> ";  
 cin >> initialPoint;  
 cout << "+ ------------------------------------------- +" << endl;  
 cout << "+ x | f(x) | iteration +" << endl;  
 cout << "+ ------------------------------------------- +" << endl;  
  
 do{  
 i++;  
 point = initialPoint;  
 initialPoint = aproximationFunction(point);  
  
 cout.fill(' ');  
 cout << "+ " ;  
 cout << setw(11) << right << point << "|";  
 cout << setw(16) << right << initialPoint << "|";  
 if(i<10){  
 cout << setw(10) << right << (int)i << " +";  
 }else{  
 cout << setw(10) << right << (int)i << " +";  
 }  
 cout << endl;  
  
 }while((fabs(initialPoint - point ))>=eps);  
 cout << "+ ------------------------------------------- +" << endl;  
 cout << "Root -> " << point;  
}  
  
/\*  
 \* metoda secantei  
 \*/  
void secantMethod(){  
 unsigned char i;  
 double intervalStart, intervalEnd, temp;  
 cout << "Interval -> ";  
 cin >> intervalStart >> intervalEnd;  
 cout << "Interval - [" << intervalStart << " ; " << intervalEnd << "]" << endl;  
  
 cout << "+ ------------------------------------------- +" << endl;  
 cout << "+ x | f(x) | iteration +" << endl;  
 cout << "+ ------------------------------------------- +" << endl;  
  
 do{  
 if(intervalStart == intervalEnd){  
 cout << "Error -> interval start point and end point can' be same.";  
 }  
 temp = (intervalStart\*secantFunction(intervalEnd) - intervalEnd\*secantFunction(intervalStart)) / (secantFunction(intervalEnd) - secantFunction(intervalStart));  
 intervalStart = intervalEnd;  
 intervalEnd = temp;  
  
 cout.fill(' ');  
 cout << "+ " ;  
 cout << setw(11) << right << temp << "|";  
 cout << setw(16) << right << secantFunction(temp) << "|";  
 if(i<10){  
 cout << setw(10) << right << (int)i << " +";  
 }else{  
 cout << setw(10) << right << (int)i << " +";  
 }  
 cout << endl;  
  
 i++;  
  
 }while(fabs(secantFunction(temp)) > eps);  
  
 cout << "+ ------------------------------------------- +" << endl;  
 cout << "Root -> " << temp;  
}  
/\*  
 \* metoda tangentei.  
 \*/  
void tangentMethod(){  
 unsigned char i = 0;  
 double intervalStart, intervalEnd, y, y1, x;  
 cout << "Interval -> ";  
 cin >> intervalStart >> intervalEnd;  
 cout << "Interval - [" << intervalStart << " ; " << intervalEnd << "]" << endl;  
 x = intervalStart;  
 y = newtonFunc(x);  
 y1 = newtonFunc1(x);  
  
 cout << "+ ------------------------------------------- +" << endl;  
 cout << "+ x | f(x) | iteration +" << endl;  
 cout << "+ ------------------------------------------- +" << endl;  
  
 while ((i<=iterationsNumber) && ((y<-eps) || (y>eps))){  
 x = (x - y) / y1;  
 y = newtonFunc(x);  
 y1 = newtonFunc1(x);  
  
 cout.fill(' ');  
 cout << "+ " ;  
 cout << setw(11) << right << x << "|";  
 cout << setw(16) << right << y << "|";  
 if(i<10){  
 cout << setw(10) << right << (int)i << " +";  
 }else{  
 cout << setw(10) << right << (int)i << " +";  
 }  
 cout << endl;  
  
 i++;  
 }  
  
 cout << "+ ------------------------------------------- +" << endl;  
 if(i > iterationsNumber){  
 cout << "Problem can't be solved in maximum number of iterations.";  
 }else{  
 cout << "Root -> " << x;  
 }  
}  
/\*  
 \* Funcție de returnare pentru care trebuie să găsiți rădăcină.  
 \*/  
double func(double x){  
 return (pow(x,2) + 4 \* sin(x));  
}  
  
double newtonFunc(double x){  
 return 2 \*(x + 2\*(cos(x)));  
}  
  
double newtonFunc1(double x) {  
 return 3 \* pow(x, 2) - 12;  
}  
  
double aproximationFunction(double x){  
 return (pow(x,2) + 4 \* sin(x));  
}  
  
double secantFunction(double x){  
 return (pow(x,2) + 4 \* sin(x));  
}

**Pentru executarea funcției b înlocuim la “Funcía de returnare pentru care trebuie să găsiți rădăcina.”**

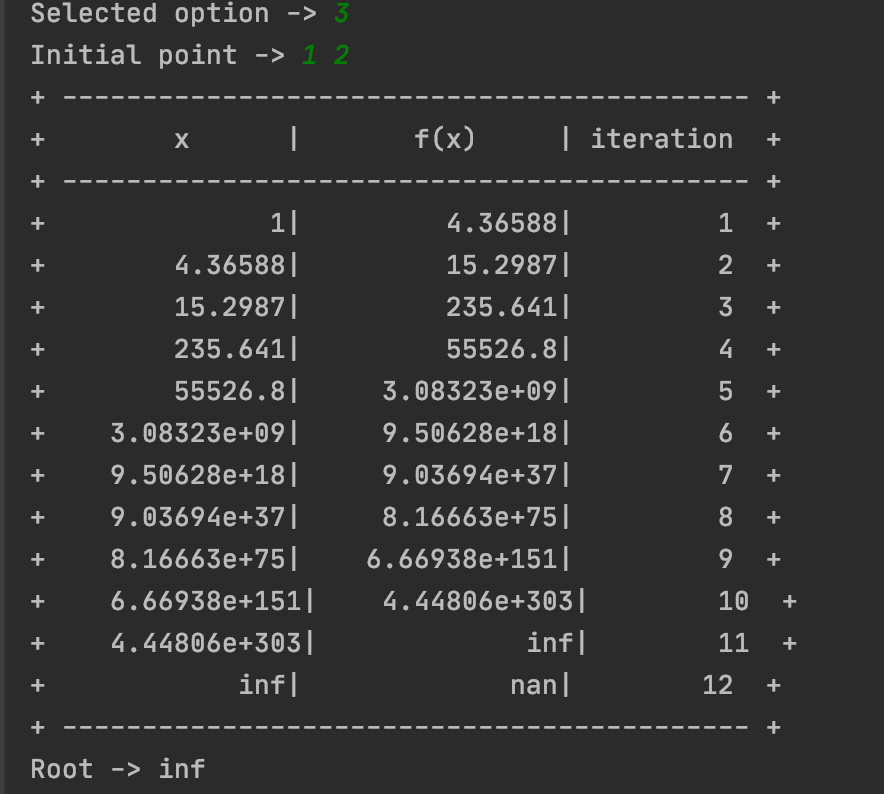
double func(double x){  
 return (pow(x,3) - 18(x)+48);  
}  
  
 double newtonFunc(double x){  
 return (pow(x,3) - 18(x)+48);  
 }  
  
 double newtonFunc1(double x) {  
 return sin(x) + 3\*pow(x+3,2);  
 }  
  
 double aproximationFunction(double x){  
 return (x(x(pow)x, 2) - 6);  
 }  
  
 double secantFunction(double x){  
 return (x(x(pow)x, 2) - 6);  
 }

****

**Figura.1.**Meniul programului.

****

**Figura.1.2.** Metoda Newton (tangentelor).



**Figura.1.3.** Metoda aproximațiilor.

**Concluzie:**

Elaborând această lucrare de laborator am folosit diferite metode de analiză numerică ce permite determinarea rădăcinilor ecuațiilor. Pe parcursul elaborării programului am folosit următoarele metode:

1. Metoda aproximațiilor succesive.
2. Metoda tangentelor (Newton).
3. Metoda secantelor.

Pe parcursul elaborării programului am înțeles că din toate 3 metode care au fost aplicate pentru rezolvarea funcțiilor pot spune că metoda de determinarea a soluțiilor cea mai eficientă este metoda aproximațiilor succesive, pentru că se execută într-un număr foarte mic de pași.