**MINISTERUL EDUCAȚIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**RAPORT**

Lucrare de laborator nr. 1

la cursul ***„Metode numerice”***

**Varianta 13**

**A efectuat :**   **St. gr. CR-221FR Serba Cristina**

**A verificat: Seiciuc Veaceslav**

**Chișinău 2023**

# Sarcina lucrării:

1) Să se separe toate rădăcinile reale ale ecuaţiei f(x)=0 unde y=f(x) este o funcţie

reală de variabilă reală.

2) Să se determine o rădăcină reală a ecuaţiei date cu ajutorul metodei înjumătăţirii

intervalului cu o eroare mai mică decât e=10^(-2).

3) Să se precizeze rădăcina obţinută cu exactitatea e=10^(-6) utilizând

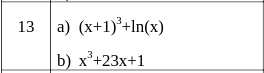
- metoda aproximaţiilor succesive

- metoda tangentelor (Newton)

- metoda secantelor.

4) Să se compare rezultatele luând în consideraţie numărul de iteraţii, evaluările

pentru funcţii şi derivată.



# Mersul lucrării:

Determinăm intervalul pe care se găsește soluția ecuației pentru fiecare caz utilizând șirul lui Rolle

|  |  |  |  |
| --- | --- | --- | --- |
| x | -2 | -1.47 | -1 |
| F(x) | -0.306853 +3.14159i | 0.281439 +3.14159i | 3.14159 i |

Ecuația are soluție pe intervalul [-2, -1.47]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x | -1 | -isqrt(23/3) | i | isqrt(23/3) | 1 |
| G(x) | -23 | 1-46/3i sqrt(23/3) | 1+22i | 1+46/3isqrt(23/3) | 25 |

Ecuația are soluție pe intervalul [-1, -isqrt(23/2)]

# Mersul programului:

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#define EROAREA 0.1

#define EXACTITATEA 0.00001

#define APROXIMARI 100

double f(double);

double fl(double);

double g(double);

double f1(double);

double g1(double);

double f2(double);

double g2(double x);

void input(float\* a, float\* b);

void metodaAproximatiilorSuccesive(float a, float b,

double (\* funct) (double));

void metodaNewton(float a, float b, double (\* funct) (double),

double (\* funct1) (double));

void metodaSecantelor(float a, float b, double (\* funct) (double));

int main()

{

int choice;

float a, b;

do

{

printf("Meniu: 1. Metoda aproximatiilor succesive\n2. Metoda tangentelor(Newton)\n3. Metoda secantelor\n0. Iesire din program\n");

scanf("%d", &choice);

switch (choice)

{

case 0:

{

exit(0);

}

case 1:

{

input(&a, &b);

printf("Cazul a:\n");

metodaAproximatiilorSuccesive(a, b, &f);

printf("Cazul b:\n");

metodaAproximatiilorSuccesive(a, b, &g);

break;

}

case 2:

{

input(&a, &b);

printf("Cazul a:\n");

metodaNewton(a, b, &f, &f1);

printf("Cazul b:\n");

metodaNewton(a, b, &g, &g1);

break;

}

case 3:

{

input(&a, &b);

printf("Cazul a:\n");

metodaSecantelor(a, b, &f);

printf("Cazul b:\n");

metodaSecantelor(a, b, &g);

break;

}

default:

{

printf("Alegere invalida. Incercati din nou\n");

break;

}

}

}

while (choice != 0);

return 0;

}

double f(double x)

{

return pow(x + 1, 3) + log(x);

}

double g(double x)

{

return pow(x, 3) + 23 \* x + 1;

}

double

f1(double x)

{

return 3 \* pow(x + 1, 2) + 1 / x;

}

double

g1(double x)

{

return 3 \* pow(x, 2) + 23;

}

double

f2(double x)

{

return -1 / pow(x, 2) + 6 \* x + 6;

}

double g2(double x)

{

return 6 \* x;

}

void input(float\* a, float\* b)

{

printf("Introduceti intervalul [a, b]\n");

scanf("%f%f", a, b);

}

void metodaAproximatiilorSuccesive(float a, float b, double (\* funct) (double x))

{

float x;

for (int i = 0; i < APROXIMARI; i++)

{

x = (a + b) / 2;

if ((\*funct)(x) == 0)

{

break;

}

else

{

if ((\*funct)(a) \* (\*funct)(x) > 0)

{

a = x;

}

else

{

b = x;

}

}

}

printf("Solutia ecuatiei este x = %f\n", x);

}

void

metodaNewton(float a, float b, double (\* funct) (double x),

double (\* funct1) (double))

{

float c, x;

int k = 0;

c = a - ((\*funct)(a) \* (b - a)) / ((\*funct)(b) - (\*funct)(a));

if ((\*funct)(c) \* (\*funct)(a) < 0)

{

x = a;

}

else

{

x = b;

}

for (int i = 0; i < APROXIMARI; i++)

{

x = x - (\*funct)(x) / (\*funct1)(x);

k++;

}

printf("Solutia ecuatiei este x = %f\n", x);

}

void

metodaSecantelor(float a, float b, double (\* funct) (double))

{

float c, e, x;

int k = 0;

c = a - ((\*funct)(a) \* (b - a)) / ((\*funct)(b) - (\*funct)(a));

if ((\*funct)(c) \* (\*funct)(a) < 0)

{

e = a;

x = b;

}

else

{

e = b;

x = a;

}

for (int i = 0; i < APROXIMARI; i++)

{

x = x - ((\*funct)(x) \* (e - x)) / ((\*funct)(e) - (\*funct)(x));

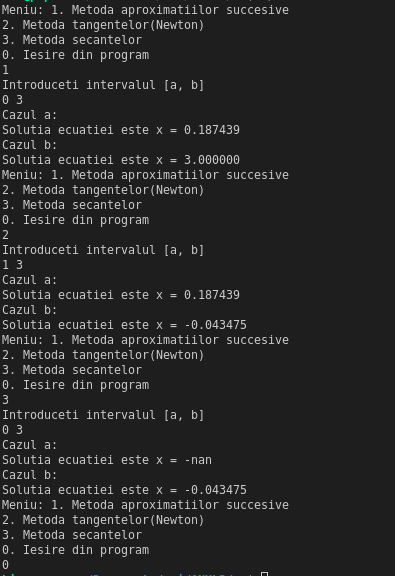
k++;

}

printf("Solutia ecuatiei este x = %f\n", x);

}

# Rezultatele obtinute:



# Concluzii:

Pentru ecuații complexe, cum ar fi cele de gradul al treilea sau mai mare, metodele numerice precum Newton sau bisectia sunt adesea preferate. Aceste metode implică aproximarea iterativă a soluției și se bazează pe concepte matematice precum derivata sau evaluarea funcției pentru a ghida procesul către rădăcini. Separarea soluțiilor înseamnă identificarea intervalelor pe care soluțiile le pot ocupa. Acest lucru este util în metodele numerice, deoarece reduce spațiul de căutare și face procesul de găsire a soluțiilor mai eficient. Separarea soluțiilor poate optimiza aceste metode numerice, contribuind la eficiența procesului de găsire a rădăcinilor în contextul ecuațiilor algebrice complexe.