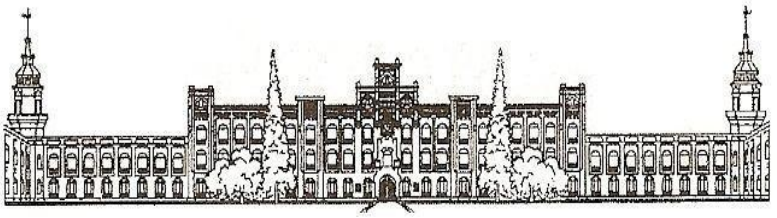

Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут» ім. Ігоря Сікорського

Лабораторна робота №2

з дисципліни «Алгоритми та методи обчислень»

«Розв'язання рівнянь з одним невідомим»

Виконав студент групи: КВ-33

Козлов Сергій

Київ 2025

Мета роботи

Дослідити методи наближеного розв’язання алгебраїчних та трансцендентних рівнянь.

Завдання для лабораторної роботи

Для заданого рівняння відповідно до варіанту (табл. 2.4) виконати відокремлення коренів та обчислити значення констант m_1 або q (залежно від методу уточнення коренів, що використовується). Розв‘язати задане рівняння з точністю $\varepsilon_i = \varepsilon_{(i-1)} / 10^{-3}$, $i = 1, 2, ..., 4$; $\varepsilon_0 = 0.01$. Результати подати у вигляді трьох таблиць виду:

Таблиця 2.1

Метод ітерації (I)

ε_i	Значення кореня	Оцінка точності кореня за методом I
10^{-2}	0.05	0.00867
...	...	...

Таблиця 2.2

Метод бісекції (Б) / метод дотичних (Д) / метод хорд (Х)

ε_i	Значення кореня	Оцінка точності кореня за методом Б/Д/Х
10^{-2}	0.05	0.00867
...	...	...

Для першого кореня рівняння побудувати порівняльну таблицю швидкості збігання методу ітерації та іншого методу, заданого за варіантом.

Таблиця 2.3

Порівняння швидкості збігання ітераційного та іншого (заданого) методу

ε_i	Кількість ітерацій за методом I	Кількість ітерацій за методом Б/Д/Х
10^{-2}	3	6
...	...	...

Завдання за варіантом

Варіант №6

6	$15(x+1)\sqrt{1+\cos(x/3)}-1.1\cos(x/5)-3=0$	I, X

Методи: Ітерації, Хорд

Підготовчі розрахунки

Побудуємо задану за варіантом функцію *f(x)* і знайдемо проміжок [a,b], на якому знаходиться один з коренів рівняння:

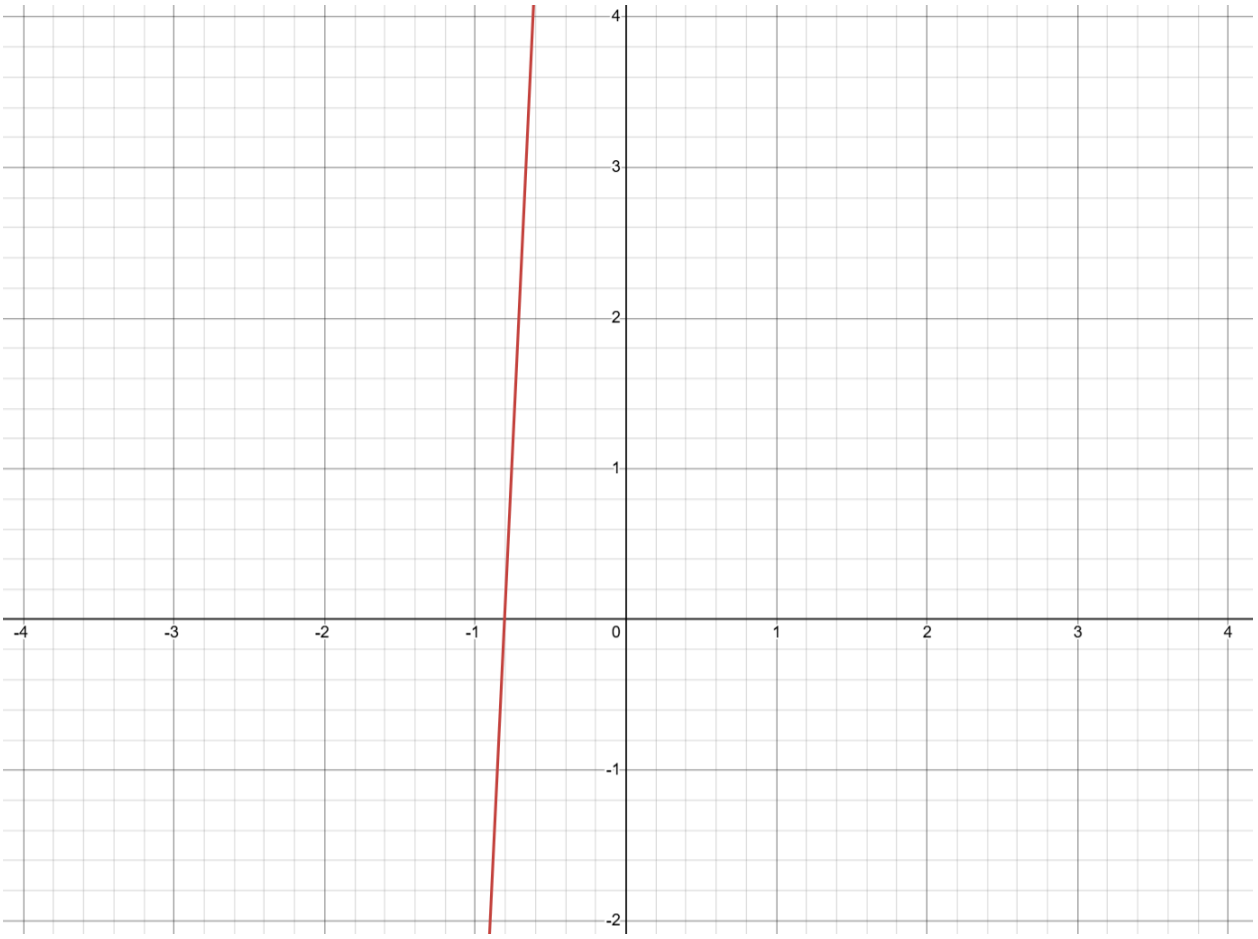

Рис. 1 - побудована функція *f(x)*

Як видно з рис. 1, корінь рівняння знаходиться на проміжку [-1, -0.5].

Одразу обчислимо значення *f(x)* в цих точках:

$f(-1) \approx -4.08$

$f(-0.5) \approx 6.48$

Обчислимо *f'(x)* для метода ітерацій:

$$f'(x) = (15(x+1)\sqrt{1+\cos\left(\frac{x}{3}\right)} - 1.1\cos\left(\frac{x}{5}\right) - 3)'$$

$$f'(x) = (15(x+1))'\sqrt{1+\cos\left(\frac{x}{3}\right)} + (15(x+1))\left(\sqrt{1+\cos\left(\frac{x}{3}\right)}\right)' - (1.1\cos\left(\frac{x}{5}\right))'$$

$$f'(x) = (15)\sqrt{1+\cos\left(\frac{x}{3}\right)} + (15(x+1))\left(\frac{1}{2}(1+\cos\left(\frac{x}{3}\right))^{-\frac{1}{2}} \cdot \left(-\frac{1}{3}\sin\left(\frac{x}{3}\right)\right)\right) + 1.1 \cdot \frac{1}{5}\sin\left(\frac{x}{5}\right)$$

$$f'(x) = 15\sqrt{1+\cos\left(\frac{x}{3}\right)} - \frac{15(x+1)\sin\left(\frac{x}{3}\right)}{6\sqrt{1+\cos\left(\frac{x}{3}\right)}} + \frac{1.1}{5}\sin\left(\frac{x}{5}\right)$$

$$f'(x) = 15\sqrt{1+\cos\left(\frac{x}{3}\right)} - \frac{5(x+1)\sin\left(\frac{x}{3}\right)}{2\sqrt{1+\cos\left(\frac{x}{3}\right)}} + \frac{1.1}{5}\sin\left(\frac{x}{5}\right)$$

Оскільки на інтервалі [-1, -0.5] *f(x)* > 0, можемо розрахувати m_1 та M_1:

$m_1 = f'(-1) \approx 20.87$

$M_1 = f'(-0.5) \approx 21.26$

тоді розрахуємо q і λ:

$q = 1 - m_1 / M_1 \approx 0.018$

$\lambda = 1 / M_1 \approx 0.047$

Обчислимо *f''(x)* для метода хорд:

$$f''(x) = (15\sqrt{1+\cos\left(\frac{x}{3}\right)} - \frac{5(x+1)\sin\left(\frac{x}{3}\right)}{2\sqrt{1+\cos\left(\frac{x}{3}\right)}} + \frac{1.1}{5}\sin\left(\frac{x}{5}\right))'$$

$$f''(x) = 15 \cdot (\sqrt{1+\cos(\frac{x}{3})})' - \frac{5}{2} \cdot (\frac{(x+1)(\sin(\frac{x}{3})}{\sqrt{1+\cos(\frac{x}{3})}})' + \frac{1.1}{5} \cdot (\sin(\frac{x}{5}))'$$

$$f''(x) = -\frac{5}{2} \cdot \frac{\sin(\frac{x}{3})}{\sqrt{1+\cos(\frac{x}{3})}} - \frac{5}{2} \cdot \frac{((x+1)(\sin(\frac{x}{3})))'(\sqrt{1+\cos(\frac{x}{3})}) - ((x+1)(\sin(\frac{x}{3}))(\sqrt{1+\cos(\frac{x}{3})})'}{1+\cos(\frac{x}{3})} + \frac{1.1}{5} \cdot \frac{\cos(\frac{x}{5})}{5}$$

$$f''(x) = -\frac{5}{2} \cdot \frac{\sin(\frac{x}{3})}{\sqrt{1+\cos(\frac{x}{3})}} - \frac{5}{2} \cdot \frac{((x+1)'(\sin(\frac{x}{3}) + (x+1)(\sin(\frac{x}{3})')(\sqrt{1+\cos(\frac{x}{3})}) - ((x+1)(\sin(\frac{x}{3}))(-\frac{\sin(\frac{x}{3})}{6\sqrt{1+\cos(\frac{x}{3})}})}{1+\cos(\frac{x}{3})} + \frac{1.1}{5} \cdot \frac{\cos(\frac{x}{5})}{5}$$

$$f''(x) = -\frac{5}{2} \cdot \frac{\sin(\frac{x}{3})}{\sqrt{1+\cos(\frac{x}{3})}} - \frac{5}{2} \cdot \frac{((\sin(\frac{x}{3}) + (x+1)(\frac{\cos(\frac{x}{3})}{3}))(\sqrt{1+\cos(\frac{x}{3})}) + ((x+1)(\sin(\frac{x}{3}))(\frac{\sin(\frac{x}{3})}{6\sqrt{1+\cos(\frac{x}{3})}})}{1+\cos(\frac{x}{3})} + \frac{1.1}{5} \cdot \frac{\cos(\frac{x}{5})}{5}$$

$$f''(x) = -\frac{5\sin(\frac{x}{3})}{2\sqrt{1+\cos(\frac{x}{3})}} - \frac{10(3\sin(\frac{x}{3}) + (x+1)\cos(\frac{x}{3}))(1+\cos(\frac{x}{3})) + 5(x+1)\sin(\frac{x}{3})^2}{12(1+\cos(\frac{x}{3}))^{\frac{3}{2}}} + \frac{1.1\cos(\frac{x}{5})}{25}$$

Тепер обчислимо значення 2ї похідної в точках [-1, -0.5]:

f''(-1) ≈ 1.22

f''(-0.5) ≈ 0.34

Для методу хорд необхідно обрати нерухому точку *c* таку, що *f(x)f''(x)* > 0.

В даному випадку приймемо *c* = -0.5.

Блок схеми алгоритмів

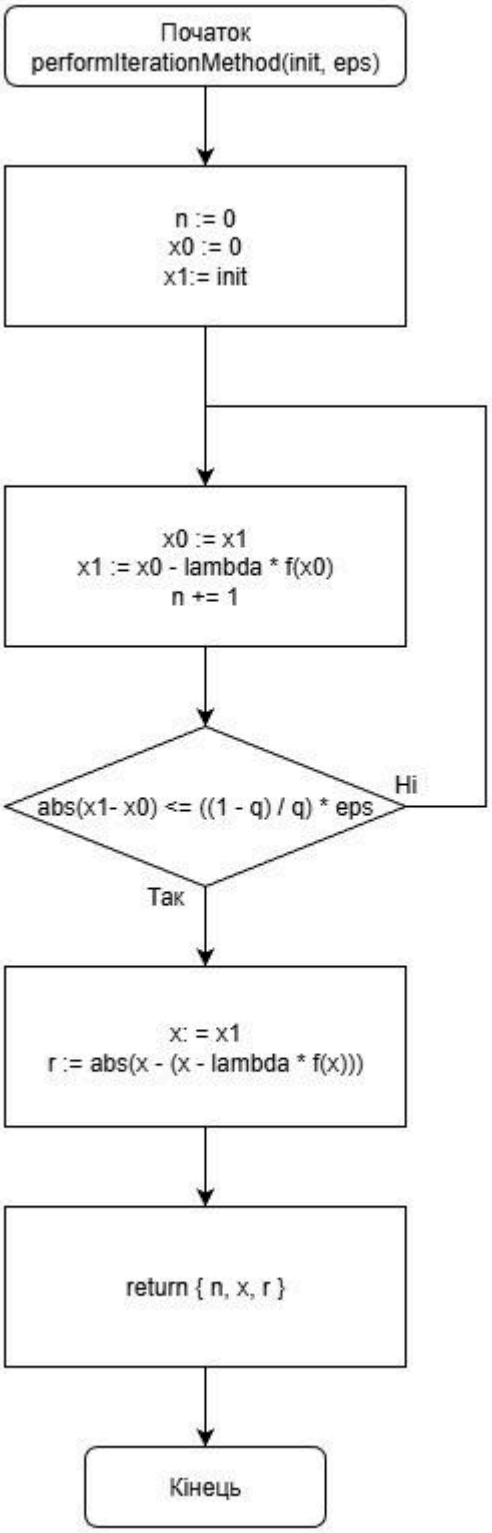

Рис. 2 - Блок-схема для методу ітерацій (I)

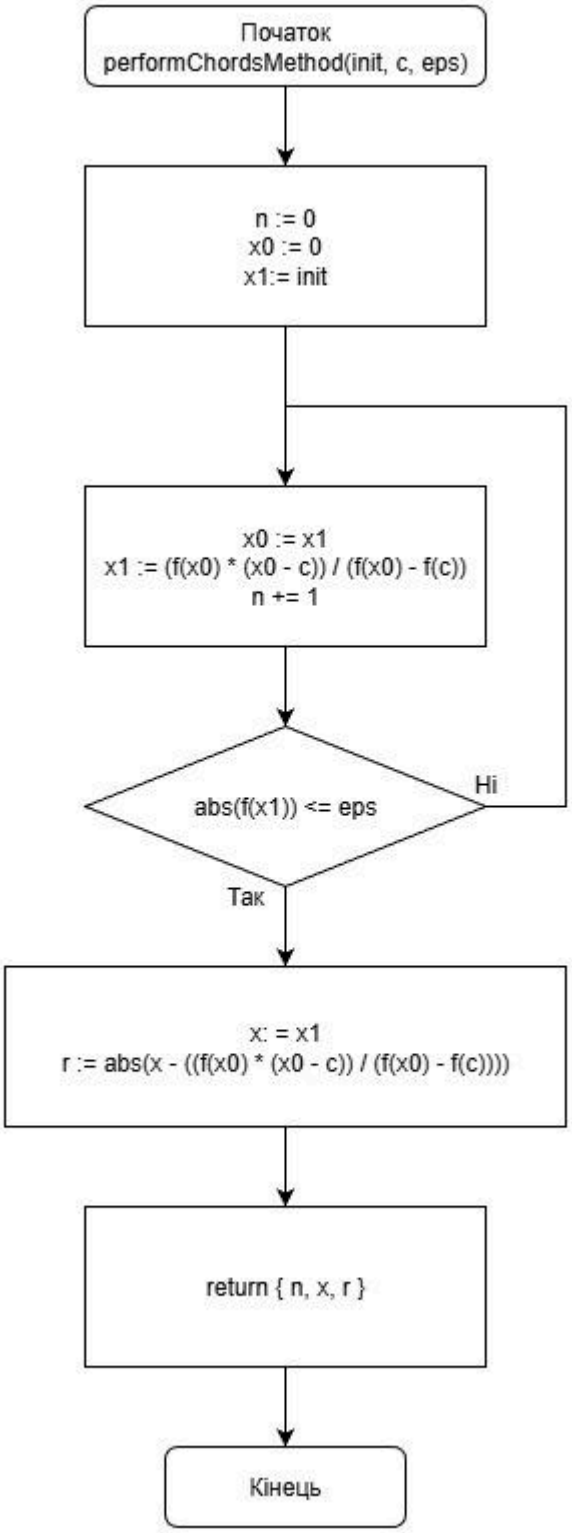

Рис. 3 - Блок-схема для методу хорд (X)

Результати виконання

ε	Значення кореня	Оцінка точності за методом I
10^-2	-0.8051909041	4.5127084353e-04
10^-5	-0.8056421749	4.1877151551e-06
10^-8	-0.8056464015	3.6135860970e-10
10^-11	-0.8056464019	3.3567593150e-12
10^-14	-0.8056464019	3.3306690739e-16

Табл. 1

ε	Значення кореня	Оцінка точності за методом X
10^-2	-0.8067879804	1.1358283141e-03
10^-5	-0.8056521521	5.7212257979e-06
10^-8	-0.8056464021	1.4487433475e-10
10^-11	-0.8056464019	7.2897243797e-13
10^-14	-0.8056464019	3.6637359813e-15

Табл. 2

ε	Кількість ітерацій за методом I	Кількість ітерацій за методом X
10^-2	1	1
10^-5	2	2
10^-8	4	4
10^-11	5	5
10^-14	7	6

Табл. 3

Висновки

Під час виконання лабораторної роботи були досліджені методи чисельного розв'язання рівнянь з одним невідомим – конкретно метод Ітерацій (послідовних наближень) і метод Хорд (пропорційного поділу відрізка).

Задана функція була побудована графічно і було визначено проміжок [*a*,*b*] на якому лежить перший корінь рівняння.

Були знайдені перша і друга похідні заданої функції і обчислені їх значення в точках *a* і *b*. Значення похідних в цих точках дозволили розрахувати значення m_1, M_1, q і λ і обрати нерухому точку *c* для подальшого уточнення значення коренів методами І та Х.

Для виконання уточнення за методами І та Х була розроблена програма мовою С. Цією програмою були апроксимовані корені заданого рівняння з точністю ε від 10^{-2} до 10^{-14}. Результати виконання програми представлені табл. 1-3.

Для заданого рівняння на обраному проміжку [*a*,*b*] розгляданні методи досягали бажаної точності за однакову кількість ітерацій. Лише для $\varepsilon = 10^{-14}$ зафіксована відмінність – 7 ітерацій для методу І проти 6 для метода Х. Дослідивши залежність між кількістю необхідних ітерацій і точністю, можна зробити висновок, що кожна ітерація збільшує точність на ~2 порядки.

Враховуючи, що методи збігаються з приблизно рівною швидкістю, доцільно обирати метод базуючись на складності його підготовчого етапу. Так, для методу ітерацій необхідно знати лише *f'(x)*, тоді як для метода хорд необхідна і *f''(x)*. Однак, для метода хорд достатньо знати *sign(f''(x))* для вибору фіксованої точки. В такому випадку, метод хорд доцільно обрати для рівнянь, для яких *sign(f''(x))* в точках [*a*,*b*] можна визначити аналітично, і обирати метод ітерацій в інших випадках.

Вихідний код програми

```c
// main.c
#include <math.h>
#include <stdlib.h>
#include "f.h"
#include "log.h"
#include "iteration.h"
#include "chords.h"

void testIterationMethod() {
  FILE* outputFile = fopen("../iteration_method_results.csv", "w");

  writeTableRowToConsole(1, "Iteration Method Results:");
  writeTableRowToConsole(3, "Epsilon", "Approximation", "Residual");

  writeTableRowToFile(outputFile, 3, "Epsilon", "Approximation", "Residual");

  double initialGuess = (A + B) / 2.0;

  for (int i = 0; i < 5; i++) {
    int power = (i * 3) + 2;
    double epsilon = pow(10.0, -power);

    IterationMethodResult result = performIterationMethod(initialGuess, epsilon);

    char* epsilonStr = exponentToString(10, -power);
    char* residualStr = doubleToString(result.residual, "%.10e");
    char* approximationStr = doubleToString(result.approximation, "%.10f");

    writeTableRowToConsole(3, epsilonStr, approximationStr, residualStr);

    writeTableRowToFile(outputFile, 3, epsilonStr, approximationStr, residualStr);

    free(epsilonStr);
    free(residualStr);
    free(approximationStr);
  }
}

void testChordsMethod() {
  FILE* outputFile = fopen("../chords_method_results.csv", "w");

  writeTableRowToConsole(1, "Chords Method Results:");
  writeTableRowToConsole(3, "Epsilon", "Approximation", "Residual");

  writeTableRowToFile(outputFile, 3, "Epsilon", "Approximation", "Residual");

  double fixedPoint = B;
  double initialGuess = A;

  for (int i = 0; i < 5; i++) {
    int power = (i * 3) + 2;
    double epsilon = pow(10.0, -power);

    ChordsMethodResult result = performChordsMethod(initialGuess, fixedPoint,
epsilon);

    char* epsilonStr = exponentToString(10, -power);
    char* residualStr = doubleToString(result.residual, "%.10e");
    char* approximationStr = doubleToString(result.approximation, "%.10f");

    writeTableRowToConsole(3, epsilonStr, approximationStr, residualStr);

    writeTableRowToFile(outputFile, 3, epsilonStr, approximationStr, residualStr);

    free(epsilonStr);
    free(residualStr);
    free(approximationStr);
  }
```

```
}

void testComparison() {
  FILE* outputFile = fopen("../comparison.csv", "w");

  writeTableRowToConsole(1, "Iteration vs Chords:");
  writeTableRowToConsole(3, "Epsilon", "Iterations (I)", "Iterations (C)");

  writeTableRowToFile(outputFile, 3, "Epsilon", "Iterations (I)", "Iterations (C)");

  double initialGuessIterations = (A + B) / 2.0;

  double fixedPointChords = B;
  double initialGuessChords = A;

  for (int i = 0; i < 5; i++) {
    int power = (i * 3) + 2;
    double epsilon = pow(10.0, -power);

    IterationMethodResult resultIterations =
performIterationMethod(initialGuessIterations, epsilon);
    ChordsMethodResult resultChords = performChordsMethod(initialGuessChords,
fixedPointChords, epsilon);

    char* epsilonStr = exponentToString(10, -power);
    char* iterationsIterations = intToString(resultIterations.iterations);
    char* chordsIterations = intToString(resultChords.iterations);

    writeTableRowToConsole(3, epsilonStr, iterationsIterations, chordsIterations);

    writeTableRowToFile(outputFile, 3, epsilonStr, iterationsIterations,
chordsIterations);

    free(epsilonStr);
    free(iterationsIterations);
    free(chordsIterations);
  }
}

int main() {
  testIterationMethod();

  writeTableRowToConsole(1, "\n");

  testChordsMethod();

  writeTableRowToConsole(1, "\n");

  testComparison();

  writeTableRowToConsole(1, "\n");

  return 0;
}
```

```c
// log.h
#ifndef LOG_H
#define LOG_H

#include <stdio.h>

extern const int MAX_STRING_SIZE;

char* exponentToString(int base, int exponent);
char* doubleToString(double value, char* format);
char* intToString(int value);
void writeTableRowToConsole(int columns, ...);
void writeTableRowToFile(FILE* file, int columns, ...);

#endif
```

```c
// log.c
#include "log.h"

#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>

const int MAX_STRING_SIZE = 256;

char* exponentToString(int base, int exponent) {
  char* string = (char*) malloc(MAX_STRING_SIZE * sizeof(char));

  snprintf(string, MAX_STRING_SIZE, "%d^%d", base, exponent);

  return string;
}

char* doubleToString(double value, char* format) {
  char* string = (char*) malloc(MAX_STRING_SIZE * sizeof(char));

  snprintf(string, MAX_STRING_SIZE, format, value);

  return string;
}

char* intToString(int value) {
  char* string = (char*) malloc(MAX_STRING_SIZE * sizeof(char));

  snprintf(string, MAX_STRING_SIZE, "%d", value);

  return string;
}

void writeTableRowToConsole(int columns, ...) {
  va_list args;
  va_start(args, columns);

  for (int i = 0; i < columns; i++) {
    const char* column_value = va_arg(args, const char*);

    printf("%-20s", column_value);
  }

  printf("\n");

  va_end(args);
}

void writeTableRowToFile(FILE* file, int columns, ...) {
  va_list args;
  va_start(args, columns);

  for (int i = 0; i < columns; i++) {
```

```
    const char* column_value = va_arg(args, const char*);

    fprintf(file, "%-20s", column_value);

    if (i < columns - 1) {
      fprintf(file, ",");
    }
  }

  fprintf(file, "\n");

  va_end(args);
}
```

```h
// f.h
#ifndef F_H
#define F_H

extern double A;
extern double B;
extern double m1;
extern double M1;
extern double LAMBDA;
extern double Q;

double f(double x);

#endif
```

```c
// f.c
#include "f.h"

#include <math.h>

double f(double x) {
  double a = 15.0 * (x + 1.0);
  double b = sqrt(1.0 + cos(x / 3.0));
  double c = 1.1 * cos(x / 5.0);
  double d = 3.0;

  return (a * b) - c - d;
}

double A = -1.0;
double B = -0.5;

double m1 = 20.87;
double M1 = 21.26;

double LAMBDA = 0.047;
double Q = 0.018;
```

```h
// iteration.h
#ifndef ITERATION_H
#define ITERATION_H

typedef struct {
  int iterations;
  double approximation;
  double residual;
} IterationMethodResult;

double approximateRootIterations(double x);
int hasConvergedIterations(double previousApproximation, double currentApproximation,
double epsilon);
IterationMethodResult performIterationMethod(double initialGuess, double epsilon);

#endif
```

```c
// iteration.c
#include "iteration.h"

#include <math.h>
#include "f.h"

double approximateRootIterations(double x) {
  return x - (LAMBDA * f(x));
}

int hasConvergedIterations(double previousApproximation, double currentApproximation,
double epsilon) {
  double error = fabs(currentApproximation - previousApproximation);
  double threshold = ((1 - Q) / Q) * epsilon;

  return error <= threshold;
}

IterationMethodResult performIterationMethod(double initialGuess, double epsilon) {
  int iterationCount = 0;
  double previousApproximation = 0.0;
  double currentApproximation = initialGuess;

  do {
    previousApproximation = currentApproximation;
    currentApproximation = approximateRootIterations(previousApproximation);
    iterationCount++;
  } while (!hasConvergedIterations(previousApproximation, currentApproximation,
epsilon));

  IterationMethodResult result;
  result.iterations = iterationCount;
  result.approximation = currentApproximation;
  result.residual = fabs(currentApproximation -
approximateRootIterations(currentApproximation));

  return result;
}
```

```c
// chords.h
#ifndef CHORDS_H
#define CHORDS_H

typedef struct {
  int iterations;
  double approximation;
  double residual;
} ChordsMethodResult;

double approximateRootChords(double x, double c);
int hasConvergedChords(double currentApproximation, double epsilon);
ChordsMethodResult performChordsMethod(double initialGuess, double fixedPoint, double
epsilon);

#endif
```

```c
// chords.c
#include "chords.h"

#include <math.h>
#include <stdio.h>
#include "f.h"

double approximateRootChords(double x, double c) {
  double nominator = f(x) * (x - c);
  double denominator = f(x) - f(c);

  return x - (nominator / denominator);
}

int hasConvergedChords(double currentApproximation, double epsilon) {
  double error = fabs(f(currentApproximation)) / m1;
  double threshold = epsilon;

  return error <= threshold;
}

ChordsMethodResult performChordsMethod(double initialGuess, double fixedPoint, double
epsilon) {
  int iterationCount = 0;
  double previousApproximation = 0.0;
  double currentApproximation = initialGuess;

  do {
    previousApproximation = currentApproximation;
    currentApproximation = approximateRootChords(previousApproximation, fixedPoint);
    iterationCount++;
  } while (!hasConvergedChords(currentApproximation, epsilon));

  ChordsMethodResult result;
  result.iterations = iterationCount;
  result.approximation = currentApproximation;
  result.residual = fabs(currentApproximation -
approximateRootChords(currentApproximation, fixedPoint));

  return result;
}
```