­­­Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет Программной Инженерии и Компьютерной Техники

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



Лабораторная работа №2 по дисциплине  
«Вычислительная математика»

"Численное решение нелинейных уравнений и систем"

Выполнил: Сафронов Егор Михайлович

Группа: P3213

Преподаватель: Малышева Татьяна Алексеевна

г. Санкт-Петербург

2022

Цель работы:

Научиться искать решение нелинейных уравнений в заданном интервале, научиться искать решения систем нелинейных уравнений при помощи численных методов, написать программу, которая будет совершать приближенные вычисления и находить решение.

Задание лабораторной работы:

1. № варианта определяется как номер в списке группы согласно ИСУ
2. Отделить корни заданного нелинейного уравнения графически
3. Уточнить *крайний правый корень* нелинейного уравнения методом половинного деления (или методом хорд, см. вариант задания) с точностью ε=10-2. Вычисления оформить в виде таблицы, удержать 3 знака после запятой
4. Уточнить *центральный корень* нелинейного уравнения методом простой итерации с точностью ε=10-2. Вычисления оформить в виде таблицы, удержать 3 знака после запятой
5. Пользователь выбирает уравнение, корень/корни которого требуется вычислить (3–5 функций, в том числе и трансцендентные), из тех, которые предлагает программа.
6. Предусмотреть ввод исходных данных (границы интервала/начальное приближение к корню и погрешность вычисления) из файла или с клавиатуры по выбору конечного пользователя.
7. Предусмотреть вывод результатов (найденный корень уравнения, значение функции в корне, число итераций) в файл или на экран по выбору конечного пользователя.

Описание метода половинного деления, расчетные формулы:

Метод половинного деления – итерационный метод, находящий решение путем последовательного деления интервала на половины. При очередной итерации вычисляются значения функции на границах интервала. После чего границами нового интервала выбираются: одна из старых границ и середина интервала, при чем таким образом, чтобы на новых границах интервала значения функции принимали разные знаки.  
Конечная формула метода половинного деления:   
Итерационный процесс продолжается пока не будет достигнута необходимая точность:

Реализация расчета

currentA = lowerBound  
currentB = upperBound  
currentX = (currentA + currentB) / 2  
  
currentFa = function(currentA)  
currentFb = function(currentB)  
currentFx = function(currentX)  
currentMod = math.Abs(currentA - currentB)

for precision < math.Abs(currentFx) && precision < currentMod {  
 if checkHalf() == 0 {  
 currentB = currentX  
 currentX = (currentA + currentB) / 2  
 currentFa = function(currentA)  
 currentFb = function(currentB)  
 currentFx = function(currentX)  
 currentMod = math.Abs(currentA - currentB)  
 } else if checkHalf() == 1 {  
 currentA = currentX  
 currentX = (currentA + currentB) / 2  
 currentFa = function(currentA)  
 currentFb = function(currentB)  
 currentFx = function(currentX)  
 currentMod = math.Abs(currentA - currentB)  
 } else {  
 os.Exit(1)  
 }  
}

Примеры работы программы:

Text

Description automatically generated with medium confidence ![Chart, line chart

Description automatically generated]()

Описание метода простых итераций, расчетные формулы:

Метод простой итерации – итерационный метод, находящий решение путем последовательного приближения к корню уравнения phi(x)=x полученного из f(x) = 0. При очередной итерации вычисляются значения phi(x) с использование x из предыдущего шага.   
Конечная формула метода просто итерации:   
Итерационный процесс продолжается пока не будет достигнута необходимая точность:

Реализация расчета

currentXk = x0  
currentFxk = function(currentXk) // k = 0  
xk = append(xk, myType{currentXk})  
fxk = append(fxk, myType{currentFxk})  
  
currentPhixk = currentXk + lambda\*currentFxk  
phixk = append(phixk, myType{currentPhixk})  
currentXk1 = currentPhixk  
xk1 = append(xk1, myType{currentXk1})  
  
currentModxk = math.Abs(currentXk - currentXk1)  
modxk = append(modxk, myType{currentXk})  
  
for precision < math.Abs(currentModxk) {  
 currentXk = currentXk1  
 currentFxk = function(currentXk)  
 currentPhixk = currentXk + lambda\*currentFxk  
 currentXk1 = currentPhixk  
 currentModxk = math.Abs(currentXk - currentXk1)  
  
 xk = append(xk, myType{currentXk})  
 fxk = append(fxk, myType{currentFxk})  
 phixk = append(phixk, myType{currentPhixk})  
 xk1 = append(xk1, myType{currentXk1})  
 modxk = append(modxk, myType{currentModxk})  
}

Примеры работы программы:

Text

Description automatically generatedChart, line chart

Description automatically generated

Описание метода Ньютона, расчетные формулы:

Метод Ньютона –метод, находящий решение системы уравнений путем последовательного приближения к некоторому приближению deltaX: такому что   
xi = ai + deltaXi. При очередной итерации вычисляются значения deltaX с использование x из предыдущего шага.   
Конечная формула метода просто итерации:   
Итерационный процесс продолжается пока не будет достигнута необходимая точность:

Реализация расчета

for true {  
 counter++  
  
 matrix = make([][]float64, lab1.Size)  
 for i := range matrix {  
 matrix[i] = make([]float64, lab1.Size+1)  
 }

lab1.Precision = precision  
 lab1.PrepareMatrixForCalculation(matrix)  
 lab1.SetResultMatrices()  
 lab1.EntryPoint()  
  
 secondX = startX + lab1.MatrixX2[0][0]  
 secondY = startY + lab1.MatrixX2[1][0]  
 if check() || counter > 20 {  
 break  
 } else {  
 startX = secondX  
 startY = secondY

}  
}

Примеры работы программы:

Text

Description automatically generatedDiagram

Description automatically generated

Выводы по работе:

Реализованы методы половинного деления и просто итерации для поиска корней нелинейного уравнения. А также реализован метод Ньютона для поиска корней системы нелинейным уравнений. Метод Ньютона мне показался достаточно чувствительным к начальному приближению, но это не делает его хуже какого-либо другого. Помимо этого, его сходимость сильно ухудшается по мере расширения системы. Для промежуточного поиска корней были использованы методы из первой лабораторной.