Министерство образования и науки РФ

ФГБПОУ ВПО Тульский государственный университитет

КАФЕДРА АВТОМАТИКИ И ТЕЛЕМЕХАНИКИ

РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ (НУ) И СИСТЕМ НЕЛИНЕЙНЫХ УРАВНЕНИЙ

Лабораторная работа № 5

по курсу «Вычислительный практикум»

Вариант № 4

Выполнил: студент группы 220601 ________ Белым А.А.

(подпись)

Проверил: к. т. н., доцент ________ Карцева А.С.

(подпись)

Тула 2011

Цель работы

Цель работы заключается в том, чтобы изучить различные методы решения нелинейных уравнений и систем нелинейных уравнений и написать программу, реализующий один из таких методов.

Задание на работу

Решить нелинейное уравнение методом хорд:

$$\ln(\ln x) - e^{-x^2} = 0$$

Теоретическая справка

Пусть дана некоторая функция $f(x)$ и требуется найти все или некоторые значения x, для которых

$$f(x) = 0 \tag{1}$$

Значение x_*, при котором $f(x_*)$, называется корнем (или решением) уравнения (1).

Относительно функции $f(x)$ часто предполагается, что $f(x)$ дважды непрерывно дифференцируема в окрестности корня.

Корень уравнения (1) называется простым, если первая производная функции $f(x)$ в точке x_* не равна нулю, т. е. $f'(x_*) \neq 0$. Если же $f'(x_*) = 0$, то корень x_* называется кратным корнем.

Геометрически корень уравнения (1) есть точка пересечения графика функции $f(x)$ с осью абсцисс.

Большинство методов решения уравнения (1) ориентировано на отыскание простых корней уравнения (1).

В процессе приближенного отыскания корней уравнения (1) обычно выделяют два этапа: локализация (или отделение) корня и уточнение корня.

Локализация корня заключается в определении отрезка $[a, b]$, содержащего один и только один корень. Не существует универсального алгоритма локализации корня. В некоторых случаях отрезок локализации может быть найден из физических соображений. Иногда удобно бывает локализовать корень с помощью построения графика или таблицы значений функции $f(x)$. На наличие корня на отрезке

$[a, b]$ указывает различие знаков функции на концах отрезка.

Если функция непрерывна на отрезке $[a, b]$ и принимает на его концах значения разных знаков, так, что $f(a)f(b) < 0$, то отрезок $[a, b]$ содержит, по крайней мере, один корень уравнения $f(x) = 0$.

Корень будет единственным, если производная $f'(x)$ существует на $[a, b]$ и сохраняет на нем свой знак.

На этапе уточнения корня вычисляют приближенное значение корня с заданной точностью $\varepsilon > 0$. Приближенное значение корня уточняют с помощью различных итерационных методов. Суть этих методов состоит в последовательном вычислении значений $x_0, x_1, ..., x_n$, которые являются приближениями к корню x_*.

Метод хорд является модификацией метода Ньютона. Метод Ньютона требует для своей реализации вычисления производной, что ограничивает его применение. Метод секущих лишен этого недостатка. Если производную заменить ее приближением:

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$$

, то вместо формулы метода Ньютона получим

$$x_{n+1} = x_n - \frac{(x_n - x_{n-1})f(x_n)}{f(x_n) - f(x_{n-1})} \tag{2}$$

Это означает, что касательные заменены секущими. Метод секущих является двушаговым методом, для вычисления приближения x_{n+1} необходимо вычислить два предыдущих приближения x_n и x_{n-1} , и, в частности, на первой итерации надо знать два начальных значения x_0 и x_1 .

Формула (2) является расчетной формулой метода секущих.

Очередное приближение x_{n+1} получается как точка пересечения с осью OX секущей, соединяющей точки графика функции $f(x)$ с координатами $(x_{n-1}, f(x_{n-1}))$ и $(x_n, f(x_n))$.

Пусть x_* – простой корень уравнения (1) , и в некоторой окрестности этого корня функция $f(x)$ дважды непрерывно дифференцируема, причем $f''(x) \neq 0$. Тогда найдется такая малая окрестность корня , что при произвольном выборе начальных приближений из этой окрестности итерационная последовательность, определенная по формуле (2) сходится.

Сравнение оценок показывает, что метод секущих сходится медленнее, чем метод Ньютона. Но в методе Ньютона на каждой итерации надо вычислять и функцию, и производную, а в методе секущих – только функцию. Поэтому при одинаковом объеме вычислений в методе секущих можно сделать примерно вдвое больше итераций и получить более высокую точность.

Так же, как и метод Ньютона, при неудачном выборе начальных приближений (вдали от корня) метод секущих может расходиться. Кроме того применение метода секущих осложняется из-за того, что в знаменатель расчетной формулы метода (2) входит разность значений функции. Вблизи корня эта разность мала, и метод теряет устойчивость.

Критерий окончания итераций метода секущих такой же, как и для метода Ньютона. При заданной точности вычисления нужно вести до тех пор, пока не будет выполнено неравенство

$$|x_n - x_{n-1}| < \varepsilon \ .$$

Схема алгоритма

На рисунке 1 представлена схема алгоритма решения нелинейного уравнения методом хорд.

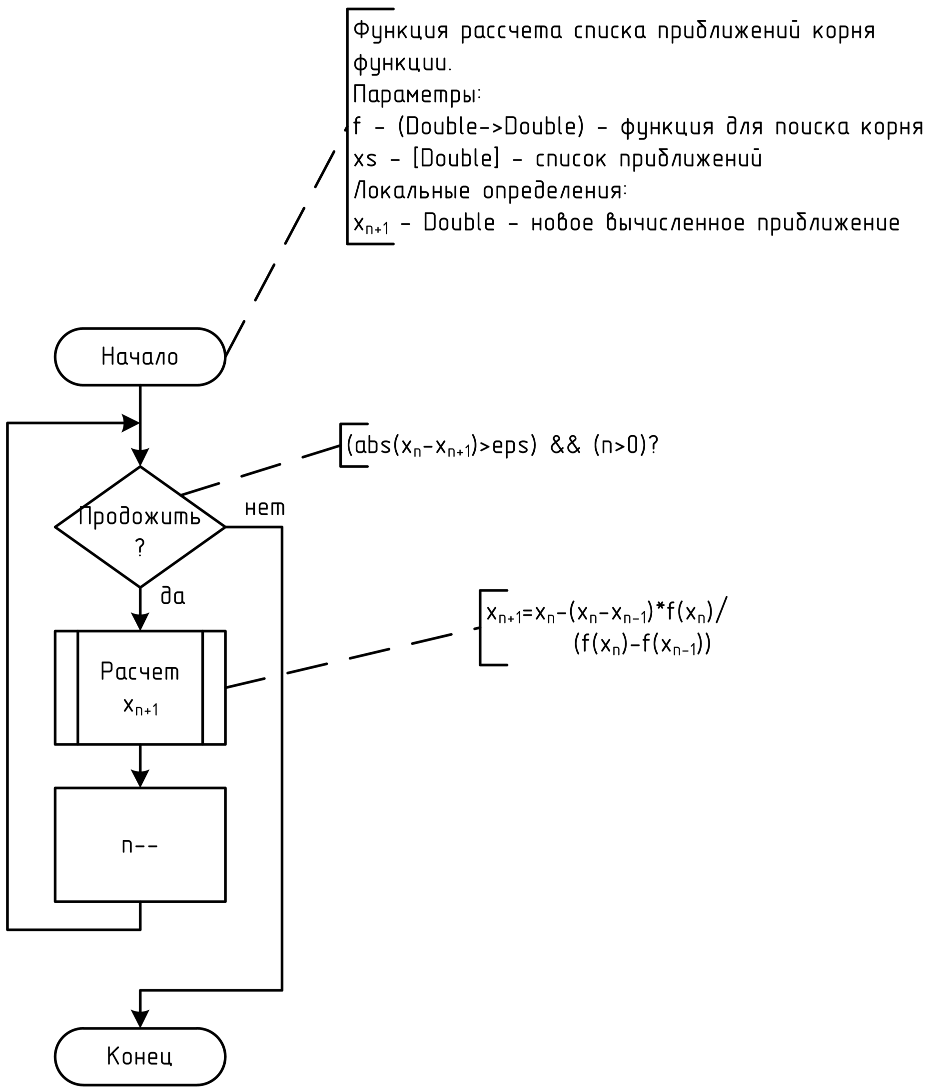

Рисунок 1 - Схема алгоритма решения нелинейного уравнения методом хорд

Инструкция пользователя

Программа позволяет решить нелинейное уравнение методом хорд.

Для работы введите параметры расчета (два начальных приближения корня, требуюмую точность и лимит итераций) в соотвествующие поля ввода на форме и нажмите кнопку "Посчитать". Программа выведет корень уравнения на форму.

Инструкция программиста

При разработке программы решения нелинейных уравнений были написаны следующие функции:

1. chordSolver - рассчитывает список приближений корня функции.

chordSolver::(Double->Double)->[Double]->[Double]

Параметры функции представлены в таблице 1 :

Таблица 1 - Параметры функции расчета списка приближений корня

имя	**тип**	**предназначение**
f	(Double->Double)	функция для поиска корня
xs	[Double]	список приближений

Локальные определения функции представлены в таблице 2 :

Таблица 2 - Локальные определения функции расчета списка приближений корня

имя	**тип**	**предназначение**
x	Double	новое вычисленное приближение

2. firstWhenEpsOrCount - возвращает первый элемент списка, для которого – выполнено условие точности либо некоторое количество итераций.

firstWhenEpsOrCount::(Double,Double)->[Double]->Double

Параметры функции представлены в таблице 3 :

Таблица 3 - Параметры функции выбора элемента списка

имя	тип	предназначение
xs	[Double]	список приближений
(eps,n)	(Double,Double)	значение точности и лимит итераций

3. chordSolve - расчет корня функции до выполнения условия точности либо до превышения лимита итераций.

chordSolve::(Double->Double)->(Double,Double)->(Double,Double)->Double

Параметры функции представлены в таблице 4 :

Таблица 4 - Параметры функции расчета корня

имя	тип	предназначение
f	(Double->Double)	функция для поиска корня
(x0,x1)	(Double,Double)	начальные приближения
(eps,maxCount)	(Double,Double)	значение точности и лимит итераций

Текст программы

Реализиция задачи решения нелинейных уравнений написана на языке Haskell 98 и состоит из двух частей.

Первая часть является вычислительным ядром программы, текст этой части приводится ниже:

```
module ChordSolve where
-- chordSolver - рассчитывает список приближений корня функции
-- Параметры:
-- f - (Double->Double) - функция для поиска корня
-- xs - [Double] - список приближений
-- Локальные определения:
-- x - Double - новое вычисленное приближение
chordSolver::(Double->Double)->[Double]->[Double]
chordSolver f xs=xs++zipWith (x) (chordSolver f xs) (tail (chordSolver f xs))
                where
                    x xn xn1=xn-(xn-xn1)*f(xn)/(f(xn)-f(xn1))
-- firstWhenEpsOrCount - возвращает первый элемент списка, для которого
-- выполнено условие точности либо некоторое количество итераций.
-- Параметры:
-- xs - [Double] - список приближений
-- (eps,n) - (Double,Double) - значение точности и лимит итераций
firstWhenEpsOrCount::(Double,Double)->[Double]->Double
firstWhenEpsOrCount (eps,n) (x:xs)|(abs(x-head xs)>eps) && (n>0) =
                                        firstWhenEpsOrCount (eps,(n-1)) (xs)
                                  |otherwise = head xs

-- chordSolve - расчет корня функции до выполнения условия точности
-- либо до превышения лимита итераций.
-- Параметры:
-- f - (Double->Double) - функция для поиска корня
-- (x0,x1) - (Double->Double) - начальные приближения
-- (eps,maxCount) - (Double,Double) - значение точности и лимит итераций
chordSolve::(Double->Double)->(Double,Double)->(Double,Double)->Double
chordSolve f (x0,x1) (eps,maxCount)=firstWhenEpsOrCount  (eps,maxCount)
                                          (chordSolver f [x0,x1])

f::Double->Double
f x=log(log x) - exp(-x^2)
```

Во втрой части содержится реализация графического интерфейса:

```
module Main where
import Prelude hiding (catch)
import Control.Exception
import Graphics.UI.Gtk
import Graphics.UI.Gtk.Builder
import ChordSolve


getValue:: Entry->IO (Double)
getValue entry=do
    y<-get entry entryText
    evaluate (read y)

onClickExit=mainQuit

getValues::[Entry]->IO [Double]
getValues xs=getValues' xs []

getValues'::[Entry]->String->IO [Double]
```

```haskell
getValues' [] ers |ers==[]= return []
                  |otherwise= error ers

getValues' (x:xs) ers=do
    y<-try(getValue x)::IO(Either SomeException Double)

    case y of
        Left e->do
            v<-entryGetText x
            vs<-getValues' xs (ers++"Неправильное значение: \"" ++ v++"\"!\n")
            return vs
        Right v->do
            vs<-getValues' xs ers
            return (v:vs)
main::IO ()
main = do
   let

   initGUI
   gtkbuilder<-builderNew
   builderAddFromFile gtkbuilder "gui.glade"
   let
        showFail::SomeException-> IO()
        showFail e=do
            dlg<-messageDialogNew Nothing [DialogModal] MessageError
                ButtonsClose ("Ошибки:\n"++ show e)
            _<-dialogRun dlg
            widgetDestroy dlg
        onClickRun=do
            entryA <- builderGetObject gtkbuilder castToEntry "entry1"
            entryB <- builderGetObject gtkbuilder castToEntry "entry2"
            entryEps <- builderGetObject gtkbuilder castToEntry "entry3"
            entryMaxN <- builderGetObject gtkbuilder castToEntry "entry4"
            labelResult <- builderGetObject gtkbuilder castToLabel "labelResult"
            [a,b,eps,maxN]<-getValues [entryA,entryB,entryEps,entryMaxN]
            if a==b then error "Границы равны!" else
                if eps<=0 then error "Точность меньше/равна 0!" else
                    if maxN<=0 then error "Неправильное число итераций!" else do
                        let
                            res=chordSolve f (a, b) (eps,maxN)
                        set labelResult [labelText:="Результат: "++ show res]
   window    <- builderGetObject gtkbuilder castToWindow "window1"
   buttonRun <- builderGetObject gtkbuilder castToButton "buttonRun"
   onClicked buttonRun (catch onClickRun showFail)
   buttonExit <- builderGetObject gtkbuilder castToButton "button2"
   onClicked buttonExit onClickExit
   onDestroy window mainQuit
   widgetShowAll window
   mainGUI
```

Тестовый пример

Ниже на рисунке 2 представлен пример работы программы при решении нелинейного уравнения $\ln(\ln x) - e^{-x^2} = 0$ методом хорд.

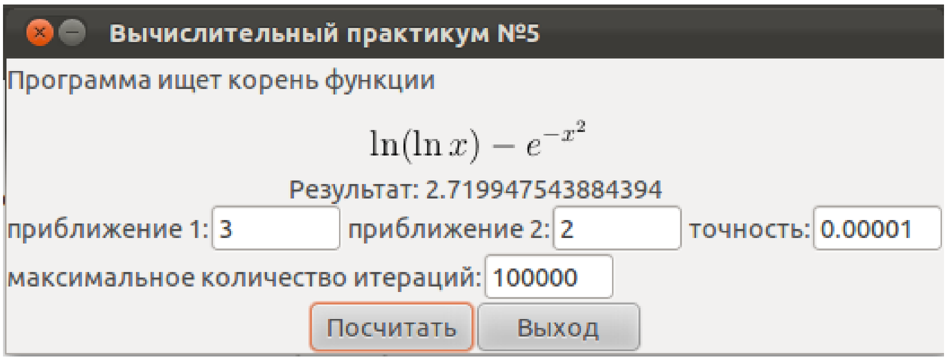

Рисунок 2 - Пример работы программы решения нелинейных уравнений

Вывод

В этой лабораторной работе я изучил различные методы решения нелинейных уравнений и систем нелинейных уравнений. Так как многие уравнения в прикладных задачах невозможно привести к виду линейного уравнения (или системы линейных уравнений), решение таких уравнений и систем представляет собой важную задачу в различных областях науки и техники.