Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

РЕШЕНИЕ УРАВНЕНИЙ С ОДНОЙ ПЕРЕМЕННОЙ

Отчет по лабораторной работе по дисциплине

численные методы

Обучающийся гр. 439-3
(группа)

_____ А. С. Мазовец
(подпись) *(И. О. Фамилия)*

« _____ » _____ 2021 г.
(дата)

Проверил:

ассистент кафедры АСУ
(должность, ученая степень, звание)

_____ А. Е. Косова
(подпись) *(И. О. Фамилия)*

_____ « _____ » _____ 2021 г.
(оценка) *(дата)*

Томск 2021

1 Введение

1.1 Цель

Реализовать ряд методов решения уравнений f (x) = 0, где x ∈ [a, b] – скалярный аргумент функции f.

1.2 Задачи

Необходимо реализовать ряд методов решения уравнений f (x) = 0, где x ∈ [a, b] – скалярный аргумент функции f. При этом предполагается, что отделение корней уже произведено, т.е. на отрезке [a, b] находится только одно решение уравнения ξ ∈ [a, b]. В этом случае выполняется условие f (a) f (b) ≤ 0. Решение должно быть найдено с абсолютной погрешностью по аргументу ε и/или абсолютной погрешностью по значению функции δ, т.е. |ξ – x * | < ε и/или | f (x *)| < δ, где ξ – точное решение уравнения f (x) = 0, а x * – приближенное.

1.3 Входные данные

1) n – номер метода;

2) f(x) – исследуемая функция в аналитическом виде;

3) a b – границы отрезка;

4) ε – требуемая точность решения.

1.4 Выходные данные

1) x* – решение уравнения;

2) f(x*) – значение функции в найденной точке x*;

3) ε* – погрешность полученного решения.

2 Алгоритм

2.1 Метод дихотомии

1) Найти c - середину отрезка [a, b]

2) Если f(a) * f(b) ≤ 0 то b = c, иначе a = c

3) Если (b-a) / 2 < значения ошибки, то завершить метод, c - решение уравнения

4) Если |f(c)| < значения ошибки, то завершить метод, c - решение уравнения

5) Перейти к шагу 1.

2.2 Метод хорд

1) Найти ck = a - (f(a) * (b - a) / (f(b) - f(a))).

2) Если f(a) * f(ck) ≤, то b = ck, иначе a = ck.

3) Если |ck-ck-1| < значения ошибки, то завершить метод, c - решение уравнения

4) Если |f(ck)| < значения ошибки, то завершить метод, c - решение уравнения

5) k = k + 1

6) Перейти к шагу 1.

2.3 Метод Ньютона

1) Если f(a)*f’’(x) < 0, то x = a, иначе x = b.

2) h = f(x) / f’(x).

3) x = x - h.

4) Если |x| < значения ошибки, то завершить метод, c - решение уравнения.

5) Если |f(x)| < значения ошибки, то завершить метод, c - решение уравнения.

6) Перейти к шагу 2.

3 Результат работы

3.1 Метод хорд

См. рисунок 3.1, рисунок 3.2, рисунок 3.3.

```
asmazovec@mobilehost insert ~/dev/sem4.ЧМ.lab1 > cat input
1
5-x^6
-1 4
1.0e-15
```

Рисунок 3.1 - Входной файл.

```
asmazovec@mobilehost insert ~/dev/sem4.ЧМ.lab1 > ./Main
Method    : 1
Function  : Oper2 Minus (Number 5.0) (Oper2 Power (Symbol "x") (Number 6.0))
Interval  : (-1.0, 4.0)
Accuracy  : 1.0E-15

Chords ::
x    : 1.30766048601182
f(x) : 0.00000000000017
Eps  : 1.0E-15
```

Рисунок 3.2 - Работа программы

```
asmazovec@mobilehost insert ~/dev/sem4.ЧМ.lab1 > cat output
Chords ::
x    : 1.30766048601182
f(x) : 0.00000000000017
Eps  : 1.0E-15
```

Рисунок 3.3 - Выходной файл

3.2 Метод Ньютона

См. рисунок 3.4, рисунок 3.5, рисунок 3.6.

```
asmazovec@mobilehost insert ~/dev/sem4.ЧМ.lab1 > cat input
2
5-x^6
-1 4
1.0e-15
```

Рисунок 3.4 - Входной файл

```
asmazovec@mobilehost insert ~/dev/sem4.ЧМ.lab1 > ./Main
Method    : 2
Function  : Oper2 Minus (Number 5.0) (Oper2 Power (Symbol "x") (Number 6.0))
Interval  : (-1.0, 4.0)
Accuracy  : 1.0E-15

Newton ::
x    : 1.30766048601183
f(x) : 0.00000000000000
Eps  : 1.0E-15
```

Рисунок 3.5 - Работа программы

```
asmazovec@mobilehost insert ~/dev/sem4.ЧМ.lab1 > cat output
Newton ::
x    : 1.30766048601183
f(x) : 0.00000000000000
Eps  : 1.0E-15
```

Рисунок 3.6 - Выходной файл

3.3 Метод дихотомии

См. рисунок 3.7, рисунок 3.8, рисунок 3.9.

```
asmazovec@mobilehost insert ~/dev/sem4.ЧМ.lab1 > cat output
Newton ::
x    : 1.30766048601183
f(x) : 0.00000000000000
Eps  : 1.0E-15
```

Рисунок 3.7 - Входной файл

```
asmazovec@mobilehost insert ~/dev/sem4.ЧМ.lab1 > ./Main
Method    : 0
Function  : Oper2 Minus (Number 5.0) (Oper2 Power (Symbol "x") (Number 6.0))
Interval  : (-1.0, 4.0)
Accuracy  : 1.0E-15

Dichotomy ::
x    : 1.30766048601183
f(x) : 0.00000000000000
Eps  : 1.0E-15
```

Рисунок 3.8 - Работа программы

```
asmazovec@mobilehost insert ~/dev/sem4.ЧМ.lab1 > cat output
Dichotomy ::
x    : 1.30766048601183
f(x) : 0.00000000000000
Eps  : 1.0E-15
```

Рисунок 3.9 - Выходной файл

3.4 Решение аналитически

Точное значение корня - см. рисунок 3.10.

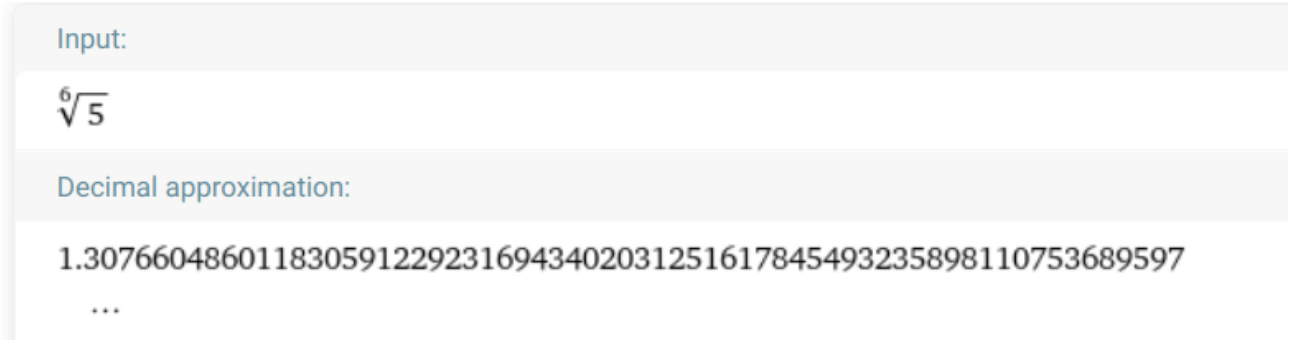

Рисунок 3.10 - Точное значение корня

4 Вывод

Были реализованы методы дихотомии, хорд, Ньютона для решения уравнений f (x) = 0, где x ∈ [a, b] – скалярный аргумент функции f.

5 Листинг программы

5.1 Модуль реализации методов

```
 1 | module Methods
 2 |     ( dichotomy
 3 |     , chords
 4 |     , newton
 5 |     ) where
 6 |
 7 | import MathParser
 8 | import Data.List ( iterate' )
 9 |
10 |
11 | dichotomy ::
12 |      Expr      -- ^ Expression
13 |   -> Double    -- ^ Left edge a
14 |   -> Double    -- ^ Right edge b
15 |   -> Double    -- ^ Epsilon
16 |   -> (Double, Double)    -- ^ (Result, Result Epsi)
17 | dichotomy expr a b epsi = (fst res,  abs $ (fst $ snd res) -
   (snd $ snd res))
18 |   where
19 |     res = head $ dropWhile fine iters
20 |
21 |     fine (m, (l, r)) = or $
22 |         [ (abs $ (r - l) / 2)  >= epsi -- error by X
23 |         , (abs $ eval' expr m) >= epsi -- error by Y
24 |         ]
25 |
26 |     next (m, (l, r))
27 |         | eval' expr l * eval' expr m <= 0 = (xi l m, (l, m))
28 |         | otherwise                        = (xi m r, (m, r))
29 |
30 |     iters = iterate' next (xi a b, (a, b))
31 |
32 |     xi l r = (l + r) / 2 -- i-тое приближение
33 |
34 |
35 | chords ::
36 |      Expr      -- ^ Expression
37 |   -> Double    -- ^ Left edge a
38 |   -> Double    -- ^ Right edge b
39 |   -> Double    -- ^ Epsilon
40 |   -> (Double, Double)    -- ^ Result
41 | chords expr a b epsi = (fst $ fst res, abs $ (fst $ fst res) -
   (fst $ snd res))
42 |   where
43 |     res = head $ dropWhile fine pairs
44 |
45 |     fine ((m0, _), (m1, _)) = or $
```

```
46 |         [ (abs $ (m1 - m0) / 2)                 >= epsi --
     error by X
47 |         , (abs $ eval' expr m1 - eval' expr m0) >= epsi --
     error by Y
48 |         ]
49 |
50 |     next (m1, (l1, r1))
51 |         | eval' expr l1 * eval' expr m1 <= 0 = (xi l1 m1, (l1,
     m1))
52 |         | otherwise                          = (xi m1 r1, (m1,
     r1))
53 |
54 |     iters = iterate' next (xi a b, (a, b))
55 |
56 |     pairs = zip iters (tail iters)
57 |
58 |     xi l r = l-(eval' expr l)/((eval' expr r)-(eval' expr
     l))*(r-l)
59 |
60 |
61 | newton ::
62 |      Expr      -- ^ Expression
63 |   -> Double    -- ^ Left edge a
64 |   -> Double    -- ^ Right edge b
65 |   -> Double    -- ^ Epsilon
66 |   -> (Double, Double)    -- ^ Result
67 | newton expr a b epsi = (fst res, abs $ (fst res) - (snd res))
68 |   where
69 |     res = head $ dropWhile fine pairs
70 |
71 |     fine (m0, m1) = or $
72 |         [ (abs $ (m1 - m0) / 2)                 > epsi --
     error by X
73 |         , (abs $ eval' expr m1 - eval' expr m0) > epsi --
     error by Y
74 |         ]
75 |
76 |     next m0 = m0-(eval' expr m0)/(deriv1 m0)
77 |
78 |     iters = iterate' next x0
79 |
80 |     x0
81 |         | eval' expr a * deriv2 a > 0 = a
82 |         | otherwise                   = b
83 |
84 |     pairs = zip iters (tail iters)
85 |
86 |     deriv1 x = (eval' expr (x+epsi)-eval' expr
     (x-epsi))/(2*epsi)
87 |     deriv2 x = (deriv1 (x+epsi)-deriv1 (x-epsi))/(4*epsi*epsi)
```

5.2 Модуль парсинга математических выражений

```
 1 | module MathParser
 2 |     ( Expr (..)
 3 |     , Func (..)
 4 |     , eval
 5 |     , eval'
 6 |     , parseExpression
 7 |     ) where
 8 |
 9 | import Text.Parsec
10 | import Text.Parsec.String
11 |
12 | data Expr
13 |     = Number Double
14 |     | Symbol String
15 |     | Oper1 Func Expr
16 |     | Oper2 Func Expr Expr
17 |     | Func1 Func Expr
18 |     | Func2 Func Expr Expr
19 |     deriving (Show, Eq)
20 |
21 | data Func
22 |     = UnaryPlus    -- ^ op      1 argument   '+'
23 |     | UnaryMinus   -- ^ op      1 argument   '-'
24 |     | Plus         -- ^ op      2 argument   '+'
25 |     | Minus        -- ^ op      2 argument   '-'
26 |     | Multip       -- ^ op      2 argument   '*'
27 |     | Divide       -- ^ op      2 argument   '/'
28 |     | Power        -- ^ op      2 argument   '^'
29 |     | LogBase      -- ^ func    2 argument   'logBase'
30 |     | Log          -- ^ func    1 argument   'log'
31 |     | Sin          -- ^ func    1 argument   'sin'
32 |     | Cos          -- ^ func    1 argument   'cos'
33 |     | Tan          -- ^ func    1 argument   'tan'
34 |     | Cot          -- ^ func    1 argument   'cot'
35 |     | Exp          -- ^ func    1 argument   'exp'
36 |     deriving (Show, Eq)
37 |
38 |
39 | {- Evaluating -}
40 |
41 | eval :: String -> Double -> Double
42 | eval s a = eval' (parseExpression s) a
43 |
44 | eval' :: Expr -> Double -> Double
45 | eval' (Number x) _             = x
46 | eval' (Symbol _) a             = a
47 | eval' (Oper1 UnaryPlus  x  ) a = eval' x a
48 | eval' (Oper1 UnaryMinus x  ) a = negate $ eval' x a
49 | eval' (Oper1 _          _  ) _ = error "undefined operator"
50 | eval' (Oper2 Plus       x y) a = eval' x a +  eval' y a
51 | eval' (Oper2 Minus      x y) a = eval' x a -  eval' y a
```

```
 52 | eval' (Oper2 Multip     x y) a = eval' x a *  eval' y a
 53 | eval' (Oper2 Divide     x y) a = eval' x a /  eval' y a
 54 | eval' (Oper2 Power      x y) a = eval' x a ** eval' y a
 55 | eval' (Oper2 _          _ _) _ = error "undefined operator"
 56 | eval' (Func1 Log        x  ) a = log $ eval' x a
 57 | eval' (Func1 Sin        x  ) a = sin $ eval' x a
 58 | eval' (Func1 Cos        x  ) a = cos $ eval' x a
 59 | eval' (Func1 Tan        x  ) a = tan $ eval' x a
 60 | eval' (Func1 Cot        x  ) a = (1/) . tan $ eval' x a
 61 | eval' (Func1 Exp        x  ) a = exp $ eval' x a
 62 | eval' (Func1 _          _  ) _ = error "undefined function"
 63 | eval' (Func2 LogBase    x y) a = logBase (eval' x a) (eval' y
    a)
 64 | eval' (Func2 _          _ _) _ = error "undefined function"
 65 |
 66 |
 67 | {- Parser -}
 68 |
 69 | parseExpression :: String -> Expr
 70 | parseExpression s =
 71 |     case parse (whitespace *> parseExpression' <* eof) "" s of
 72 |         Left e  -> error $ show e
 73 |         Right x -> x
 74 |
 75 | parseExpression' :: Parser Expr
 76 | parseExpression'
 77 |     =  priority2
 78 |    <|> priority1
 79 |    <|> unary
 80 |    <|> priority0
 81 |    <|> roundBrace parseExpression'
 82 |    <|> parseFunction1
 83 |    <|> parseFunction2
 84 |    <|> number
 85 |    <|> symbol
 86 |
 87 | unary :: Parser Expr
 88 | unary = try $ do
 89 |     operator <- choice
 90 |         [ UnaryPlus  <$ char '+'
 91 |         , UnaryMinus <$ char '-'
 92 |         ]
 93 |     whitespace
 94 |     expression <- parseExpression''
 95 |     return $ Oper1 operator expression
 96 |   where
 97 |     parseExpression''
 98 |         =  roundBrace parseExpression'
 99 |        <|> parseFunction1
100 |        <|> parseFunction2
101 |        <|> number
102 |        <|> symbol
103 |
```

```
104 | priority0 :: Parser Expr
105 | priority0 = try $ parseExpression'' `chainl1` operator
106 |   where
107 |     operator = choice
108 |         [ Oper2 Power   <$ char '^'
109 |         ] <* whitespace
110 |
111 |     parseExpression''
112 |         =  roundBrace parseExpression'
113 |        <|> parseFunction1
114 |        <|> parseFunction2
115 |        <|> number
116 |        <|> symbol
117 |
118 | priority1 :: Parser Expr
119 | priority1 = try $ parseExpression'' `chainl1` operator
120 |   where
121 |     operator = choice
122 |         [ Oper2 Multip  <$ char '*'
123 |         , Oper2 Divide  <$ char '/'
124 |         ] <* whitespace
125 |
126 |     parseExpression''
127 |         =  priority0
128 |        <|> roundBrace parseExpression'
129 |        <|> parseFunction1
130 |        <|> parseFunction2
131 |        <|> number
132 |        <|> symbol
133 |
134 | priority2 :: Parser Expr
135 | priority2 = try $ parseExpression'' `chainl1` operator
136 |   where
137 |     operator = choice
138 |         [ Oper2 Plus  <$ char '+'
139 |         , Oper2 Minus <$ char '-'
140 |         ] <* whitespace
141 |
142 |     parseExpression''
143 |         =  priority1
144 |        <|> priority0
145 |        <|> roundBrace parseExpression'
146 |        <|> parseFunction1
147 |        <|> parseFunction2
148 |        <|> number
149 |        <|> symbol
150 |
151 |
152 | parseFunction1 :: Parser Expr
153 | parseFunction1 = try $ do
154 |     function <- choice
155 |         [ Log <$ try (string "log")
156 |         , Sin <$ try (string "sin")
```

```
157 |         , Cos <$ try (string "cos")
158 |         , Tan <$ try (string "tan")
159 |         , Cot <$ try (string "cot")
160 |         , Exp <$ try (string "exp")
161 |         ]
162 |     whitespace
163 |     expression <- roundBrace parseExpression'
164 |     return $ Func1 function expression
165 |
166 | parseFunction2 :: Parser Expr
167 | parseFunction2 = try $ do
168 |     function <- choice
169 |         [ LogBase <$ try (string "logBase")
170 |         ]
171 |     whitespace
172 |     expression1 <- firstArg  parseExpression'
173 |     expression2 <- secondArg parseExpression'
174 |     return $ Func2 function expression1 expression2
175 |
176 | number :: Parser Expr
177 | number = do
178 |     number1 <- many1 digit
179 |     number2 <- option "" afterDot
180 |     whitespace
181 |     return $ Number (read $ number1 ++ number2)
182 |   where
183 |     afterDot = do
184 |         dot     <- char '.'
185 |         number' <- many1 digit
186 |         return $ dot:number'
187 |
188 | symbol :: Parser Expr
189 | symbol = do
190 |     x <- char 'x'
191 |     whitespace
192 |     return $ Symbol [x]
193 |
194 | -- | gives an expression which is a parse first argument [ '('
      <fst> ',' ]
195 | firstArg :: Parser a -> Parser a
196 | firstArg = between leftParen comma
197 |
198 | -- | gives an expression which is a second argument [ ','
      <snd> ')' ]
199 | secondArg :: Parser a -> Parser a
200 | secondArg = between comma rightParen
201 |
202 | -- | parse an expression which is an expression between
      roundBrace
203 | roundBrace :: Parser a -> Parser a
204 | roundBrace = between leftParen rightParen
205 |
206 | -- | gives a comma
```

```
207 | comma :: Parser Char
208 | comma = char ',' <* whitespace
209 |
210 | -- | gives a left paren
211 | leftParen :: Parser Char
212 | leftParen = char '(' <* whitespace
213 |
214 | -- | gives a right paren
215 | rightParen :: Parser Char
216 | rightParen = char ')' <* whitespace
217 |
218 | -- | skips a white space
219 | whitespace :: Parser ()
220 | whitespace = skipMany $ oneOf " \n\t"
```

5.3 Основной модуль

```
  1 | import System.IO
  2 | import Methods
  3 | import MathParser
  4 | import Text.Printf
  5 |
  6 | main :: IO ()
  7 | main = do
  8 |     let inp = "input"
  9 |     let out = "output"
 10 |
 11 |     hInp <- openFile inp ReadMode
 12 |
 13 |     method <- (read <$> hGetLine hInp) :: IO Int
 14 |     func   <- parseExpression <$> hGetLine hInp
 15 |     [a, b] <- map (read :: String -> Double)
 16 |         .  sequence [(!!0), (!!1)]
 17 |         .  words
 18 |        <$> hGetLine hInp
 19 |     epsi   <- (read <$> hGetLine hInp) :: IO Double
 20 |
 21 |     hClose hInp
 22 |
 23 |     printf "Method    : %d\n" method
 24 |     putStr "Function  : "; print func
 25 |     printf "Interval  : (%F, %F)\n" a b
 26 |     printf "Accuracy  : %E\n" epsi
 27 |     putStrLn ""
 28 |
 29 |     let d = dichotomy func a b epsi
 30 |     let c = chords    func a b epsi
 31 |     let n = newton    func a b epsi
 32 |     let outFormat m = m ++ " ::\n"
 33 |                     ++ "x    : %.*F\n"
 34 |                     ++ "f(x) : %.*F\n"
 35 |                     ++ "Eps  : %E\n"
 36 |                     ++ "Eps* : %.*F\n"
```

```
37 |     let epsiN = negate $ truncate $ logBase 10 epsi :: Int
38 |
39 |     hOut <- openFile out WriteMode
40 |
41 |     case method of
42 |         0 -> do
43 |             printf (outFormat "Dichotomy") epsiN (fst d) epsiN
     (eval' func (fst d)) epsi epsiN (snd d)
44 |             hPrintf hOut (outFormat "Dichotomy") epsiN (fst d)
     epsiN (eval' func (fst d)) epsi epsiN (snd d)
45 |             return ()
46 |         1 -> do
47 |             printf (outFormat "Chords") epsiN (fst c) epsiN
     (eval' func (fst c)) epsi epsiN (snd c)
48 |             hPrintf hOut (outFormat "Chords") epsiN (fst c)
     epsiN (eval' func (fst c)) epsi epsiN (snd c)
49 |             return ()
50 |         2 -> do
51 |             printf (outFormat "Newton") epsiN (fst n) epsiN
     (eval' func (fst n)) epsi epsiN (snd n)
52 |             hPrintf hOut (outFormat "Newton") epsiN (fst n)
     epsiN (eval' func (fst n)) epsi epsiN (snd n)
53 |             return ()
54 |         _ -> error "no method"
55 |
56 |     hClose hOut
57 |
58 |     return ()
```