Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 1

Тема: Алгоритм и программа построения интерполяционного полинома Ньютона

Студент: Романов А. В.

Группа: ИУ7-43Б

Оценка (баллы) _____

Преподаватель: Градов В. М.

Москва.
2020 г.

Цель работы:

Изучить метод нахождения значения функции в заданной точке с помощью интерполяционного полинома Ньютона.

Задание:

1. Найти Pn(x).

2. Найти корень табличной функции методом половинного деления.

3. Найти корень табличной функции методом обратной интерполяции.

Входные данные:

1. Таблица координат.

2. Координата точки по оси x.

3. Степень полинома.

Выходные данные:

1. Значение функции в точке x.

2. Корень функции, найденный с помощью двух методов.

Анализ алгоритма:

В алгоритме подсчитываются разделенные разности.

Они вычисляются по формуле (первая степень):

$$y(x_i, x_j) = \frac{y_i - y_j}{x_i - x_j}$$

Далее с помощью этих разделенных разностей подсчитывается полином Ньютона, имеющий формулу:

$$P_n(x) = y_0 + \sum_{k=0}^{n} (x - x_n) \dots (x - x_{k-1}) y(x_0, x_1, \dots, x_k)$$

При поиска корня обратной интерполяцией, столбцы меняются местами, а x задается равным 0.

Код программы:

Файл **Main.hs:**

```
import Interpolation
import System.IO
import System.Environment
import Data.List

main :: IO ()
main = do
   (x:n:findType) <- getArgs
   let table = initialConditions $ head findType

   putStr "Результат вычислений: "
   case head findType of
     "bisection" -> print $ bisection table (read n)
     _ -> print $ newtonPolynomial table (read x) (read
n)
```

Файл **Interpolation.hs:**

```
module Interpolation (
   initialConditions,
   newtonPolynomial,
   bisection
) where

import Data.List
import Data.Maybe
import Data.Sort

type TableXY = [(Double, Double)]
type Point = (Double, Double)
type Matrix = [[Double]]

epsilon :: Double
epsilon = 1e-4

f :: Double -> Double
f x = x * x

initialConditions :: String -> TableXY
initialConditions findType
   | findType == "back-intpol" = sortOn fst $ zip ys xs
   | otherwise = zip xs ys
   where xs = [1..20]
         ys = map f xs

slice :: TableXY -> Int -> Int -> TableXY
slice table n pos = take n $ drop pos table

takeApproximation :: TableXY -> Double -> Int ->
TableXY
takeApproximation table x0 n
   | (<=) x0 . fst $ head table = take n table
   | (>=) x0 . fst $ last table = reverse $ take n $ reverse
table
   | otherwise = left ++ right
      where indexL = fromJust $ findIndex (\x -> fst x >=
x0) table
            left = slice table (n `div` 2 ) (indexL - n `div` 2)
            indexR = fromJust $ findIndex (== last left) table
            right = slice table (n - length left) (indexR + 1)

createMatrix :: [Double] -> [Double] -> Int -> Matrix
createMatrix _ (_:[]) _ = []
createMatrix xs ys step =  divDiff xs ys step :
createMatrix xs (divDiff xs ys step) (step + 1)
   where divDiff _ (_:[]) _ = []
         divDiff xs ys step = (ys !! 1 - ys !! 0) / (xs !! (1 +
step) - xs !! 0) : divDiff (tail xs) (tail ys) step

newtonPolynomial :: TableXY -> Double -> Int -> Double
newtonPolynomial table x0 n = foldl (\x y -> x + fst y *
snd y) y0 $ pairs
  where approximation = unzip $ takeApproximation
table x0 (n + 1)
        matrix = createMatrix (fst approximation) (snd
approximation) 0
        y0 = head $ snd approximation
        xDifference = reverse $ init $ foldl (\x y -> (x0 - y) *
head x : x) [1] (fst approximation)
        pairs = zip (map head matrix) xDifference

bisection' :: Point -> Point -> TableXY -> Int -> Double
bisection' left right table n
  | fst right - fst left < epsilon = middle
  | approximation * snd right <= 0 = bisection' (middle,
approximation) right table n
  | otherwise = bisection' left (middle, approximation)
table n
   where
     middle = (fst left + fst right) / 2
     approximation = newtonPolynomial table middle n

bisection :: TableXY -> Int -> Double
bisection table = bisection' (head table) (last table)
table
```