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 № 2 по дисциплине «Вычислительные алгоритмы»

Тема: Многомерная интерполяция на регулярной сетке

Студент: Романов А. В.

Группа: ИУ7-43Б

Оценка (баллы) _____

Преподаватель: Градов В. М.

Москва.
2020 г.

Цель работы:

Изучить метод нахождения значения ФНП в заданной точке.

Задание:

Найти значение функции двух переменных в заданной точке.

Входные данные:

1. Таблица z = f(x, y)
2. Координаты точки по оси x и оси y.
3. Степень полинома по x и y.

Выходные данные:

Значение функции в заданной точке.

Анализ алгоритма:

1. Задаем полином по каждому направлению.

2. Для каждого ряда проводим интерполяцию

Можно построить двумерный полином Ньютона:

$$P_n(x,y) = \sum_{i=0}^{n} \sum_{j=0}^{n-i} z(x_0, x_1, \ldots, x_i; y_0, y_1, \ldots, y_j) \prod_{p=0}^{i-1} (x - x_p) \prod_{q=0}^{j-1} (y - y_q)$$

Разделенная разность:

$$z(x_0, x_1; y_0) = \frac{z(x_0, y_0) - z(x_1, y_0)}{x_0 - x_1}$$

Код программы:
Файл Main.hs:

```
import Interpolation
import Parse
import System.IO

main :: IO ()
main = do
  handle <- openFile "table.csv" ReadMode
  content <- hGetContents handle
  let table = parseTable $ lines content
  mapM_ print table
  hClose handle

  putStrLn "Введите x, y:"
  point <- fmap words getLine
  putStrLn "Введите Xn, Yn:"
  degrees <- fmap words getLine

  putStr "Результат вычислений: "
  print $ interpolation2 table (tuple point toDouble) (tuple degrees toInt)
```

Файл Interpolation.hs:

```
module Interpolation (
   interpolation2
) where

import Data.List
import Data.Maybe

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

type Point = (Double, Double)
type PolynomDegrees = (Int, Int)

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

interpolation2 :: ValueTable -> Point ->
PolynomDegrees -> Double
interpolation2 table pt n = result
   where
       xBorder = tail $ head table
       xColPairs = map (\x -> zip xBorder $ tail x) $
tail table
       xApprox = map (\x -> takeApproximation x (fst
pt) $ fst n + 1) xColPairs

       yBorder = map head $ tail table
       yApprox = reverse $ takeApproximation (zip
yBorder yBorder) (snd pt) $ snd n + 1
       xyApprox = map (\x -> (fst x, xApprox !!
(round $ fst x - 1))) yApprox

       finApprox = map (\x -> (fst x,
newtonPolynomial (snd x) (fst pt) (length
finApprox))) xyApprox
       result = newtonPolynomial finApprox (snd pt)
(length finApprox)
```