|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |

**Институт информационных технологий (ИТ)**

**Кафедра прикладной математики (ПМ)**

**КУРСОВАЯ РАБОТА**

по дисциплине: Вычислительная математика

по профилю: Анализ данных

направления профессиональной подготовки: Прикладная математика (01.03.04), бакалавриат

Тема: «Первый интерполяционный полином Ньютона с равномерными и оптимальными узлами»

Студент: Блинков Денис Евгеньевич

Группа: ИМБО-01-18

Работа представлена к защите\_\_\_\_\_\_\_\_\_\_\_(дата)\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

(подпись и Ф. И. О. студента)

Руководитель: Чердынцев Виктор Викторович

Работа допущена к защите\_\_\_\_\_\_\_\_\_\_\_(дата)\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

(подпись и Ф. И. О. руководителя)

Оценка по итогам защиты: \_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /

(подпись, дата, Ф. И. О., должность, звание, уч. степень преподавателя)

М. МИРЭА. 2019 г.

УДК 004.4

Блинков Д.Е.Разработка компьютерной программы для вычисления значений первого интерполяционного полинома Ньютона. **/ Курсовая работа** по дисциплине «Вычислительная математика» профиля «Анализ данных» направления профессиональной подготовки бакалавриата 01.03.04. «Прикладная математика» (2ой семестр) / руководитель. В.В. Чердынцев / кафедра ПМ Института ИТ РТУ МИРЭА.

Целью работы является разработка компьютерного приложения для вычисления значений полинома Ньютона.

М. МИРЭА. Ин-т ИТ. Каф. ПМ. 2019 г. Блинков Д.Е.

СОДЕРЖАНИЕ

1. Введение в тему……………………………………………………...6
2. Первая интерполяционная формула Ньютона. Теория метода……7
3. Реализация приложения……………………………………………..8
4. Заключение………………………………………………………… 17

**1. Введение в тему.**

**Интерполяция, интерполирование** – способ нахождения промежуточных значений величины по имеющемуся дискретному набору известных величин.

Задача интерполяции возникает в следующей ситуации. Часто приходится оперировать с наборами значений (данные) полученных опытным путем. Как правило, иногда требуется на основании этих наборов значений построить функцию, которая бы приближала следующие получаемые значения с нужной точностью. Такая задача называется *аппроксимацией. Интерполяция* – разновидность аппроксимации, при котором получаемая кривая точно проходит через известные имеющиеся точки данных, называемых узлами.

Существует также близкая к интерполяции задача, которая заключается в аппроксимации какой-либо сложной функции более простой. Если некоторая функция слишком сложна для производительных вычислений, то можно попытаться вычислить её значение в нескольких точках, а по ним уже интерполировать более простую функцию. Разумеется, использование упрощенной интерполированной функции не даст такие же точные результаты, как исходная функция. Но в некоторых задачах достигнутый выигрыш в простоте и скорости вычислений может перевесить получаемую погрешность.

Существуют не мало разновидностей интерполяции многочленами, такие как:

* Линейная интерполяция
* Интерполяционная формула Ньютона
* Интерполяционный многочлен Лагранжа
* Сплайн-функция

и другие.

В нашем случае мы остановимся подробно на интерполяционной формуле Ньютона.

**2.Первая интерполяционная формула Ньютона. Теория метода.**

Осуществим вывод формулы.

Пусть для функции заданы значения для равностоящих значений независимой переменной: , где h- шаг интерполяции. Требуется подобрать полином степени не выше n, принимающий в точках значения

Будем искать полином в виде

Наша задача состоит в определении коэффициентов полинома . Полагая получим

Чтобы найти , составим первую конечную разность и получим и получим

и получим

Продолжая этот процесс, обнаружим

И в итоге получим первую **интерполяционную формулу Ньютона** через конечные разности:

Также в приложении использовались равномерные узлы, которые мы получали следующим образом:

Применение интерполяционной формулы Ньютона:

Как было сказано выше, типичные задачи – это

1. Замена сложной функции более простой
2. Получение значений функции в неизвестных точках

И другие.

**3.Реализация приложения**

Для демонстрации работы приложения была выбрана эталонная функция , которую мы приближали с помощью равномерных узлов и оптимальных узлов.

Для тестового примера мы выбрали промежуток [a, b] = [-5, 8].

N = 4

x = [-4.681867, -2.320604, 1.499999, 5.320604]

y = [-1381.461475, -179.441716, 16.999999, 1556.367187]

**Исходный код:**

Файл***newton\_poly\_with\_uniform\_nodes.py***

# creating first newton polynomial

from math import factorial

def calculate\_newton\_polynomial\_uniform\_nodes(x\_dots, y\_dots, inputed\_x):

if len(x\_dots) == 1:

step = x\_dots[0]

else:

step = x\_dots[1] - x\_dots[0]

terms\_coeffs = [y\_dots]

# creating lists[list] to coeffs of finite differences

for j in range(1, len(x\_dots)):

tmp\_list = []

for v in range(len(y\_dots) - j):

tmp\_list.append(1)

terms\_coeffs.append(tmp\_list)

# calculating finite differences

for current\_list\_pointer in range(1, len(terms\_coeffs)):

for i in range(len(terms\_coeffs[current\_list\_pointer - 1]) - 1):

current\_coeffs = terms\_coeffs[current\_list\_pointer - 1]

new\_delta\_y = current\_coeffs[i + 1] - current\_coeffs[i]

terms\_coeffs[current\_list\_pointer][i] = new\_delta\_y

polynomial\_sum = terms\_coeffs[0][0] # initial y0 value

# calculating y=f(x) at the x inputed

for i in range(1, len(terms\_coeffs)):

first\_part\_term = terms\_coeffs[i][0] / (factorial(i) \* (step \*\* i))

second\_part\_term = 1 # (x-x0)(x-x1)...(x-xi)

for j in range(i):

second\_part\_term \*= (inputed\_x - x\_dots[j])

result\_term = first\_part\_term \* second\_part\_term

polynomial\_sum += result\_term

return polynomial\_sum

Файл ***newton\_poly\_with\_optimal\_nodes.py***

def calculate\_newton\_polynomial\_optimal\_nodes(x\_dots, y\_dots, inputed\_x):

terms\_coeffs = []

for i in range(len(x\_dots)):

tmp\_list = []

for j in range(len(x\_dots)):

tmp\_list.append(None)

terms\_coeffs.append(tmp\_list)

for i in range(0, len(x\_dots)):

terms\_coeffs[i][0] = y\_dots[i]

for k in range(1, len(x\_dots)):

for i in range(0, len(x\_dots) - k):

terms\_coeffs[i][k] = (terms\_coeffs[i + 1][k - 1] - terms\_coeffs[i][k - 1]) / (x\_dots[i + k] - x\_dots[i])

polynomial\_sum = y\_dots[0]

for k in range(1, len(x\_dots)):

r = 1

for i in range(0, k - 1 + 1):

r = r \* (inputed\_x - x\_dots[i])

polynomial\_sum += terms\_coeffs[0][k] \* r

return polynomial\_sum

Файл ***tools.py***

import math

import numpy as np

from newton\_poly\_with\_optimal\_nodes import calculate\_newton\_polynomial\_optimal\_nodes

from newton\_poly\_with\_uniform\_nodes import calculate\_newton\_polynomial\_uniform\_nodes

def get\_x\_dots\_optimal(x0, xn, n):

"""

get optimal Chebyshev nodes

:param x0: lower border (a)

:param xn: upper border (b)

:param n: count

:return: list of x dots

"""

x\_dots = []

for i in range(0, n):

xi = ((xn + x0) / 2) - ((xn - x0) / 2) \* math.cos(math.pi \* (2 \* i + 1) / (2 \* n + 2))

x\_dots.append(xi)

return x\_dots

def get\_y\_dots\_optimal(count):

"""

gives list of ordinates of optimal Chebyshev nodes for specified x0 and xn (-5, 8)

:param count: count of terms

:return: list of ordinates

"""

if count == 4:

return [-1381.4614750443766,

-179.44171639342537,

16.99999999999998,

1556.3671878257146]

elif count == 8:

return [1.3008848966265942,

2.5389166448734786,

1.272654113572865,

-1.3347452240844833,

6.6875,

41.21983426316527,

105.73833675454303,

180.80483335512648]

def calculate\_polynomial(x\_dots, y\_dots, inputed\_x, calculating\_type):

"""

return ordinate value of specified x

:param x\_dots: specified x dots

:param y\_dots: specified y dots

:param inputed\_x: dot it needs to calculate in

:param calculating\_type: choosing algorithm

:return: P(inputed\_x)

"""

uniform\_nodes\_type = 0

optimal\_nodes\_type = 1

result = None

if calculating\_type == uniform\_nodes\_type:

result = calculate\_newton\_polynomial\_uniform\_nodes(x\_dots=x\_dots, y\_dots=y\_dots, inputed\_x=inputed\_x)

elif calculating\_type == optimal\_nodes\_type:

result = calculate\_newton\_polynomial\_optimal\_nodes(x\_dots=x\_dots, y\_dots=y\_dots, inputed\_x=inputed\_x)

else:

print('Ошибка')

return result

def get\_additional\_dots(x\_dots\_uniform, x\_dots\_optimal, y\_dots\_uniform, y\_dots\_optimal):

"""

agmenting x, y dots to draw smooth graph

:param x\_dots\_uniform: x dots to uniform nodes

:param x\_dots\_optimal: x dots to optimal nodes

:param y\_dots\_uniform: y dots to uniform nodes

:param y\_dots\_optimal: y dots to optimal nodes

:return: augmented 4 list of x, y dots (for uniform and optimal nodes)

"""

new\_y\_dots\_uniform\_nodes = []

new\_y\_dots\_optimal\_nodes = []

new\_x\_dots = list(np.linspace(-15, 15, 30))

for x\_dot in new\_x\_dots:

res\_uniform = calculate\_polynomial(x\_dots=x\_dots\_uniform, y\_dots=y\_dots\_uniform, inputed\_x=x\_dot,

calculating\_type=0)

res\_optimal = calculate\_polynomial(x\_dots=x\_dots\_optimal, y\_dots=y\_dots\_optimal, inputed\_x=x\_dot,

calculating\_type=1)

new\_y\_dots\_uniform\_nodes.append(res\_uniform)

new\_y\_dots\_optimal\_nodes.append(res\_optimal)

return new\_x\_dots, new\_y\_dots\_uniform\_nodes, new\_y\_dots\_optimal\_nodes

Файл ***main\_w\_gui.py***

from tkinter import \*

from tkinter import messagebox

from tkinter import ttk

from tkinter import Text

from tkinter import filedialog

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk

from matplotlib.figure import Figure

import numpy as np

import matplotlib

from tools import get\_x\_dots\_optimal, get\_y\_dots\_optimal, get\_additional\_dots, calculate\_polynomial

matplotlib.use('TkAgg')

window\_width = 1500

window\_height = 900

root = Tk()

root.title("Вычисление первого полинома Ньютона")

root.geometry(f'{window\_width}x{window\_height}')

tab\_control = ttk.Notebook(root)

reading\_console\_tab = ttk.Frame(tab\_control)

tab\_control.add(reading\_console\_tab, text='Считывание с консоли')

tab\_control.pack(expand=1, fill='both')

example\_tab = ttk.Frame(tab\_control)

tab\_control.add(example\_tab, text='Эталонный пример')

tab\_control.pack(expand=1, fill='both')

reading\_file\_tab = ttk.Frame(tab\_control)

tab\_control.add(reading\_file\_tab, text='Считывание с файла')

tab\_control.pack(expand=1, fill='both')

header\_label = Label(example\_tab, text="Сравнение оптимальных и равномерных узлов"

"\nдля функции 12x^3-8x^2-5x+2",

font=("Arial Bold", 13, 'bold'))

header\_label.grid(column=0, row=0)

label\_input\_iteration = Label(example\_tab, text="Выберите n", font=("Arial Bold", 11))

label\_input\_iteration.grid(column=0, row=1)

label\_borders = Label(example\_tab, text="Введите a и b для оптимальных узлов", font=("Arial Bold", 11))

label\_borders.grid(column=0, row=2)

label\_input\_set\_x\_ex = Label(example\_tab, text="Введите множество x через пробел", font=("Arial Bold", 11))

label\_input\_set\_x\_ex.grid(column=0, row=3)

label\_input\_set\_y\_ex = Label(example\_tab, text="Введите множество y через пробел", font=("Arial Bold", 11))

label\_input\_set\_y\_ex.grid(column=0, row=4)

label\_input\_set\_y\_ex = Label(example\_tab, text="Введите точку х, для вычисление P(x)",

font=("Arial Bold", 11))

label\_input\_set\_y\_ex.grid(column=0, row=5)

count\_n = StringVar(root)

count\_n.set('4')

count\_n\_switching\_menu = OptionMenu(example\_tab, count\_n, "4", "8")

count\_n\_switching\_menu.grid(column=1, row=1)

label\_borders = Label(example\_tab, text="Введите a и b для оптимальных узлов", font=("Arial Bold", 11))

input\_borders = Entry(example\_tab, width=20)

label\_borders.grid(column=0, row=2)

input\_borders.grid(column=1, row=2, pady=10)

label\_input\_set\_x\_ex = Label(example\_tab, text="Введите множество x через пробел", font=("Arial Bold", 11))

input\_set\_x\_ex = Entry(example\_tab, width=20)

label\_input\_set\_x\_ex.grid(column=0, row=3)

input\_set\_x\_ex.grid(column=1, row=3, pady=10)

label\_input\_set\_y\_ex = Label(example\_tab, text="Введите множество y через пробел", font=("Arial Bold", 11))

input\_set\_y\_ex = Entry(example\_tab, width=20)

input\_set\_y\_ex.grid(column=1, row=4, pady=10)

label\_input\_set\_y\_ex.grid(column=0, row=4)

label\_input\_set\_y\_ex = Label(example\_tab, text="Введите точку х, для вычисление P(x)", font=("Arial Bold", 11))

input\_point\_x\_ex = Entry(example\_tab, width=20)

label\_input\_set\_y\_ex.grid(column=0, row=5)

input\_point\_x\_ex.grid(column=1, row=5, pady=10)

fig = Figure(figsize=(3, 3), dpi=120)

canvas = FigureCanvasTkAgg(fig, master=root) # A tk.DrawingArea.

toolbar = NavigationToolbar2Tk(canvas, root)

canvas.get\_tk\_widget().pack(side=TOP, fill=BOTH, expand=1)

# //////////////

def create\_draw\_from\_gui(x\_dots, y\_dots, nodes\_type):

if nodes\_type == 0:

lbl = 'Равномерные узлы'

else:

lbl = 'Оптимальные узлы'

fig.clear()

ax = fig.add\_subplot(111)

ax.plot(x\_dots, y\_dots, label=lbl)

ax.legend()

canvas.draw\_idle()

def create\_draw\_file\_tab(x\_dots, y\_dots, nodes\_type):

if nodes\_type == 0:

lbl = 'Равномерные узлы'

else:

lbl = 'Оптимальные узлы'

fig.clear()

ax = fig.add\_subplot(111)

ax.plot(x\_dots, y\_dots, label=lbl)

ax.legend()

canvas.draw\_idle()

def calculate\_polynomial\_from\_gui\_interface():

"""

Reading data from inputs in the program

:return:

"""

calculating\_type = type\_of\_polynomial.get()

uniform\_nodes\_type = 0

optimal\_nodes\_type = 1

print('POLY TYPE', calculating\_type)

if calculating\_type == uniform\_nodes\_type:

x\_dots\_uniform = list(map(float, input\_set\_x.get().split()))

y\_dots\_uniform = list(map(float, input\_set\_y.get().split()))

inputed\_x = float(input\_point\_x.get())

new\_x\_dots = list(np.linspace(x\_dots\_uniform[0], x\_dots\_uniform[-1], 100))

new\_y\_dots = []

for x\_dot in new\_x\_dots:

res = calculate\_polynomial(x\_dots=x\_dots\_uniform, y\_dots=y\_dots\_uniform, inputed\_x=x\_dot,

calculating\_type=0)

new\_y\_dots.append(res)

result = calculate\_polynomial(x\_dots=x\_dots\_uniform, y\_dots=y\_dots\_uniform, inputed\_x=inputed\_x,

calculating\_type=0)

create\_draw\_from\_gui(new\_x\_dots, new\_y\_dots, 0)

if calculating\_type == uniform\_nodes\_type:

messagebox.showinfo('Результат', f'Значение полинома P(x) с равномерными узлами '

f'P({inputed\_x}) = {result}')

elif calculating\_type == optimal\_nodes\_type:

messagebox.showinfo('Результат', f'Значение полинома P(x) с оптимальными узлами '

f'P({inputed\_x}) = {result}')

else:

print('Ошибка')

elif calculating\_type == optimal\_nodes\_type:

a, b, count = input\_set\_x.get().split()

a, b, count = float(a), float(b), int(count)

y\_dots\_optimal = get\_y\_dots\_optimal(4) # list(map(float, input\_set\_y.get().split()))

inputed\_x = float(input\_point\_x.get())

x\_dots\_optimal = get\_x\_dots\_optimal(a, b, count)

new\_x\_dots = list(np.linspace(x\_dots\_optimal[0], x\_dots\_optimal[-1], 100))

new\_y\_dots = []

for x\_dot in new\_x\_dots:

res = calculate\_polynomial(x\_dots=x\_dots\_optimal, y\_dots=y\_dots\_optimal, inputed\_x=x\_dot,

calculating\_type=1)

new\_y\_dots.append(res)

result = calculate\_polynomial(x\_dots=x\_dots\_optimal, y\_dots=y\_dots\_optimal, inputed\_x=inputed\_x,

calculating\_type=1) # switch 1 to variable

create\_draw\_from\_gui(new\_x\_dots, new\_y\_dots, 1)

if calculating\_type == uniform\_nodes\_type:

messagebox.showinfo('Результат', f'Значение полинома P(x) с равномерными узлами '

f'P({inputed\_x}) = {result}')

elif calculating\_type == optimal\_nodes\_type:

messagebox.showinfo('Результат', f'Значение полинома P(x) с оптимальными узлами '

f'P({inputed\_x}) = {result}')

else:

print('Ошибка')

# /////

def pattern\_func(x):

return 12 \* (x \*\* 3) - 8 \* (x \*\* 2) - 5 \* x + 2

def create\_draw\_ex\_tab(x\_dots\_polynomial=None, y\_dots\_uniform=None, y\_dots\_optimal=None):

pattern\_func\_x\_dots = np.linspace(-10, 13, 200)

pattern\_func\_y\_dots = []

for x\_dot in pattern\_func\_x\_dots:

pattern\_func\_y\_dots.append(pattern\_func(x\_dot))

fig.clear()

ax = fig.add\_subplot(111)

if y\_dots\_uniform is not None:

ax.plot(x\_dots\_polynomial, y\_dots\_uniform, label='равномерные узлы')

if y\_dots\_optimal is not None:

ax.plot(x\_dots\_polynomial, y\_dots\_optimal, label='оптимальные узлы')

ax.plot(pattern\_func\_x\_dots, pattern\_func\_y\_dots, label='12x^3-8x^2-5x+2')

ax.legend()

canvas.draw\_idle()

create\_draw\_ex\_tab()

def calculate\_polynomial\_from\_example\_tab():

"""

Reading data from inputs in the program

:return:

"""

a, b = list(map(float, input\_borders.get().split()))

count = int(count\_n.get())

x\_dots\_uniform = list(map(float, input\_set\_x\_ex.get().split()))

y\_dots\_uniform = list(map(float, input\_set\_y\_ex.get().split()))

inputed\_x = float(input\_point\_x\_ex.get())

x\_dots\_optimal = get\_x\_dots\_optimal(a, b, count)

y\_dots\_optimal = get\_y\_dots\_optimal(count)

new\_x\_dots\_polynomial, new\_y\_dots\_uniform, new\_y\_dots\_optimal = get\_additional\_dots(x\_dots\_uniform=x\_dots\_uniform,

x\_dots\_optimal=x\_dots\_optimal,

y\_dots\_optimal=y\_dots\_optimal,

y\_dots\_uniform=y\_dots\_uniform)

uniform\_nodes = 0

optimal\_nodes = 1

result\_with\_optimal = calculate\_polynomial(x\_dots=x\_dots\_optimal, y\_dots=y\_dots\_optimal, inputed\_x=inputed\_x,

calculating\_type=optimal\_nodes)

result\_with\_uniform = calculate\_polynomial(x\_dots=x\_dots\_uniform, y\_dots=y\_dots\_uniform, inputed\_x=inputed\_x,

calculating\_type=uniform\_nodes)

create\_draw\_ex\_tab(new\_x\_dots\_polynomial, new\_y\_dots\_uniform, new\_y\_dots\_optimal)

messagebox.showinfo('Результат', f'Оптимальные узлы P({inputed\_x}) = {result\_with\_optimal}\n'

f'Равномерные узлы P({inputed\_x}) = {result\_with\_uniform}\n'

f'Значение самой функции: {pattern\_func(inputed\_x)}')

btn\_calculate\_polynomial\_ex\_tab = Button(example\_tab, text="Найти значение полинома",

command=calculate\_polynomial\_from\_example\_tab,

width=25)

btn\_calculate\_polynomial\_ex\_tab.grid(column=2, row=5, padx=15, pady=5)

# ///////

label\_input\_set\_x = Label(reading\_console\_tab, text="Введите множество x через пробел", font=("Arial Bold", 11))

input\_set\_x = Entry(reading\_console\_tab, width=20)

label\_input\_set\_x.grid(column=0, row=0)

input\_set\_x.grid(column=1, row=0, pady=10)

label\_input\_set\_y = Label(reading\_console\_tab, text="Введите множество y через пробел", font=("Arial Bold", 11))

input\_set\_y = Entry(reading\_console\_tab, width=20)

input\_set\_y.grid(column=1, row=3, pady=10)

label\_input\_set\_y.grid(column=0, row=3)

label\_input\_set\_y = Label(reading\_console\_tab, text="Введите точку х, для вычисление P(x)", font=("Arial Bold", 11))

input\_point\_x = Entry(reading\_console\_tab, width=20)

label\_input\_set\_y.grid(column=0, row=4)

input\_point\_x.grid(column=1, row=4, pady=10)

type\_of\_polynomial = IntVar()

type\_of\_polynomial.set(0)

radio\_btn\_uniform\_nodes = Radiobutton(reading\_console\_tab, text='Полином с равномерными узлами',

variable=type\_of\_polynomial, value=0)

radio\_btn\_optimal\_nodes = Radiobutton(reading\_console\_tab, text='Полином с оптимальными узлами',

variable=type\_of\_polynomial, value=1)

radio\_btn\_uniform\_nodes.grid(column=0, row=5)

radio\_btn\_optimal\_nodes.grid(column=1, row=5)

def create\_draw(x\_dots, y\_dots\_uniform, y\_dots\_optimal):

fig.clear()

ax = fig.add\_subplot(111)

ax.plot(x\_dots, y\_dots\_uniform, label='uniform')

ax.plot(x\_dots, y\_dots\_optimal, label='optimal')

print('Create\_draw\_2: y\_uni, y\_opti', y\_dots\_uniform, '\n', y\_dots\_optimal)

ax.legend()

canvas.draw\_idle()

btn\_calculate\_polynomial = Button(reading\_console\_tab, text="Найти значение полинома",

command=calculate\_polynomial\_from\_gui\_interface,

width=25)

btn\_calculate\_polynomial.grid(column=0, row=6, padx=15, pady=5)

# ==================================

radio\_btn\_uniform\_nodes = Radiobutton(reading\_file\_tab, text='Полином с равномерными узлами',

variable=type\_of\_polynomial, value=0)

radio\_btn\_optimal\_nodes = Radiobutton(reading\_file\_tab, text='Полином с оптимальными узлами',

variable=type\_of\_polynomial, value=1)

radio\_btn\_uniform\_nodes.grid(column=0, row=2)

radio\_btn\_optimal\_nodes.grid(column=1, row=2)

label\_for\_add\_member = Label(reading\_file\_tab, text='Считывание с файла', font=("Arial Bold", 12, "bold"))

label\_for\_add\_member.grid(column=0, row=1, pady=1)

label\_for\_add\_member = Label(reading\_file\_tab, text='Считанные с файла значения', font=("Arial Bold", 12))

label\_for\_add\_member.grid(column=0, row=5, pady=1)

description\_text = Text(reading\_file\_tab, height=4, width=30, font=("Arial Bold", 11))

description\_text.grid(column=0, row=6, padx=5)

global\_file\_path = None

def chose\_file():

file\_path = filedialog.askopenfilename()

global global\_file\_path

global\_file\_path = file\_path

with open(file\_path, mode='r', encoding='utf8') as f:

x\_dots = f.readline()

y\_dots = f.readline()

inputed\_x = f.readline()

description\_text.insert(END, f'x: {x\_dots}'

f'y: {y\_dots}'

f't: {inputed\_x}')

def calculate\_polynomial\_from\_file():

global global\_file\_path

with open(global\_file\_path, mode='r', encoding='utf8') as f:

x\_dots = f.readline().split()

y\_dots = f.readline().split()

inputed\_x = f.readline()

calculating\_type = type\_of\_polynomial.get() # 0 - uniform nodes, 1 - optimal nodes

uniform\_nodes\_type = 0

optimal\_nodes\_type = 1

x\_dots = list(map(float, x\_dots))

y\_dots = list(map(float, y\_dots))

inputed\_x = float(inputed\_x)

result = calculate\_polynomial(x\_dots=x\_dots, y\_dots=y\_dots, inputed\_x=inputed\_x, calculating\_type=calculating\_type)

new\_x\_dots = list(np.linspace(x\_dots[0], x\_dots[-1], 100))

new\_y\_dots = []

for x\_dot in new\_x\_dots:

res = calculate\_polynomial(x\_dots=x\_dots, y\_dots=y\_dots, inputed\_x=x\_dot,

calculating\_type=calculating\_type)

new\_y\_dots.append(res)

create\_draw\_file\_tab(new\_x\_dots, new\_y\_dots, calculating\_type)

if calculating\_type == uniform\_nodes\_type:

messagebox.showinfo('Результат', f'Значение полинома P(x) с равномерными узлами '

f'P({inputed\_x}) = {result}')

elif calculating\_type == optimal\_nodes\_type:

messagebox.showinfo('Результат', f'Значение полинома P(x) с оптимальными узлами '

f'P({inputed\_x}) = {result}')

else:

print('Ошибка')

btn\_add\_member = Button(reading\_file\_tab, text="Выбрать файл", command=chose\_file,

width=15)

btn\_add\_member.grid(column=0, row=3, pady=5)

btn\_add\_member = Button(reading\_file\_tab, text="Найти значение полинома", command=calculate\_polynomial\_from\_file,

width=30)

btn\_add\_member.grid(column=0, row=4, pady=10)

if \_\_name\_\_ == '\_\_main\_\_':

def setup():

input\_borders.insert(0, '-5 8')

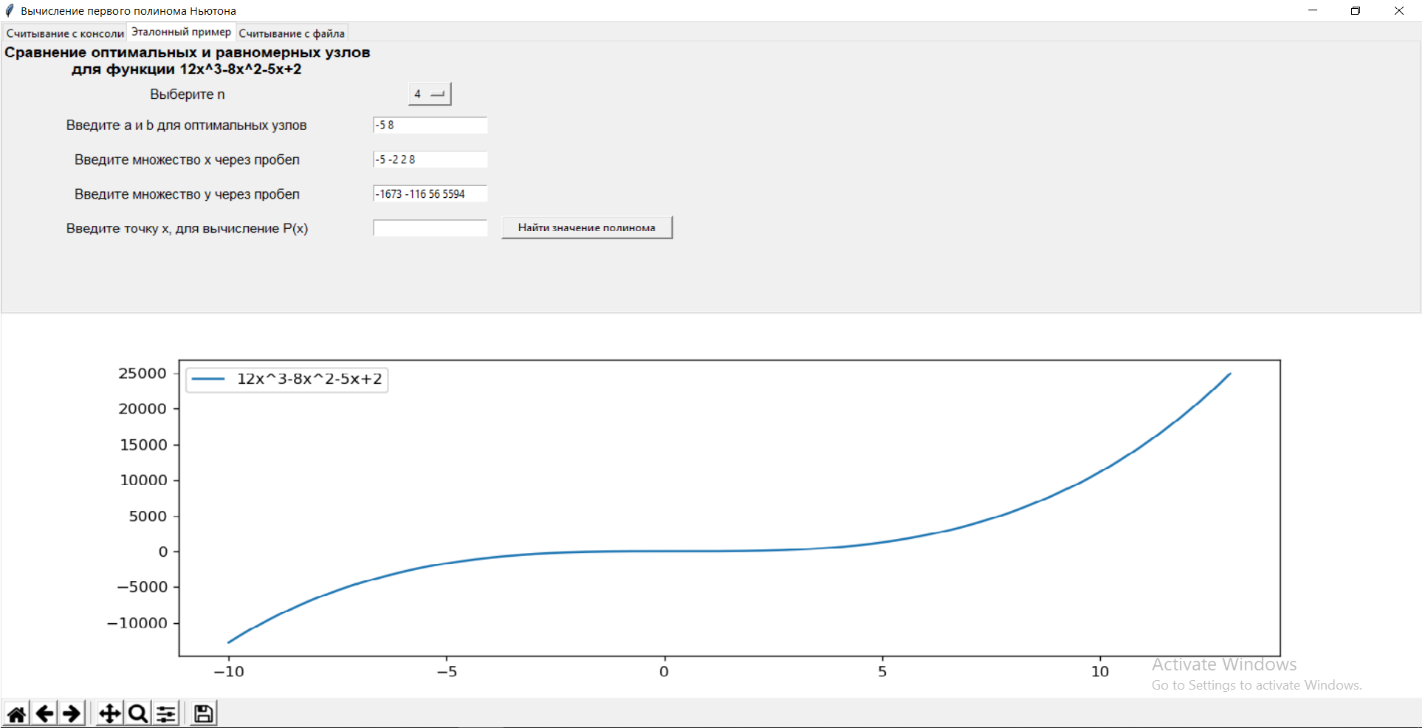
input\_set\_x\_ex.insert(0, '-5 -2 2 8')

input\_set\_y\_ex.insert(0, '-1673 -116 56 5594')

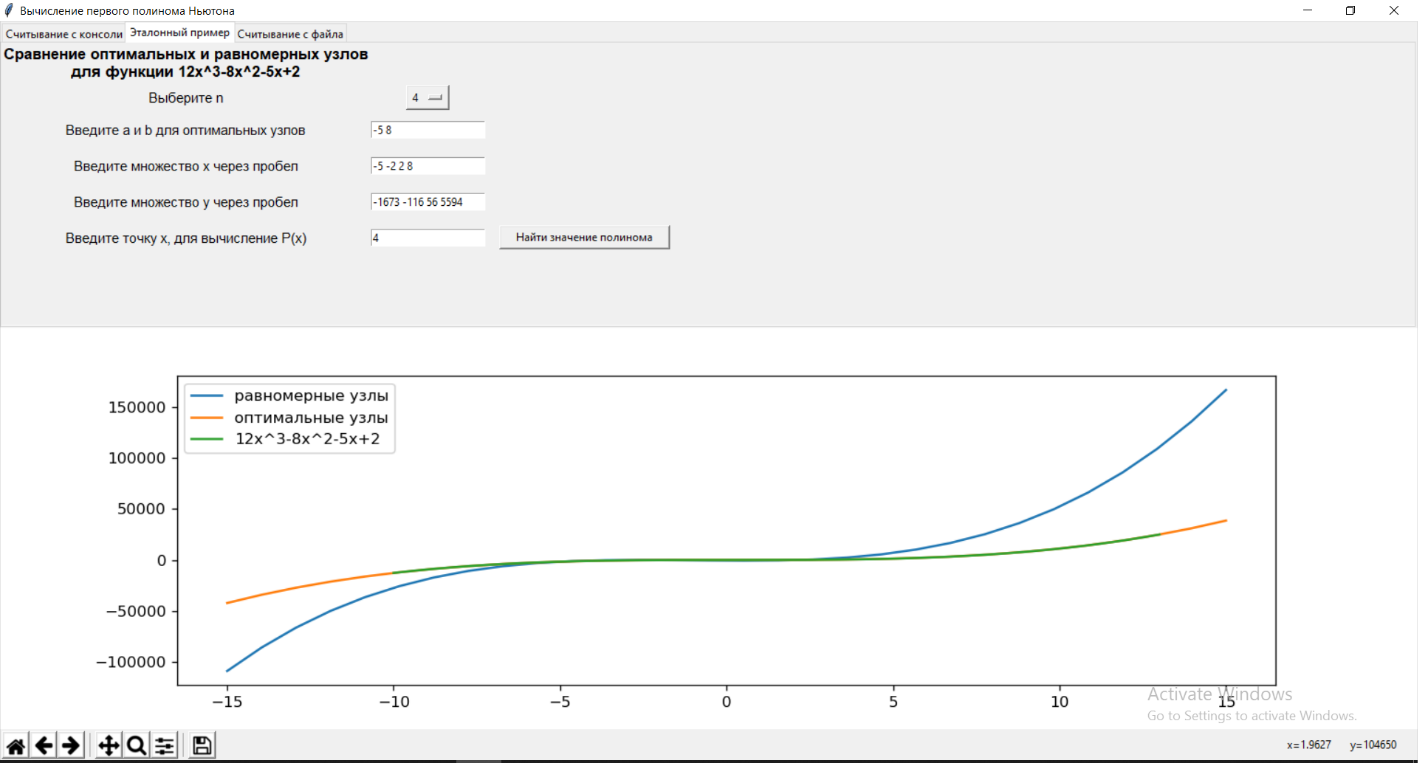
setup()

root.mainloop()

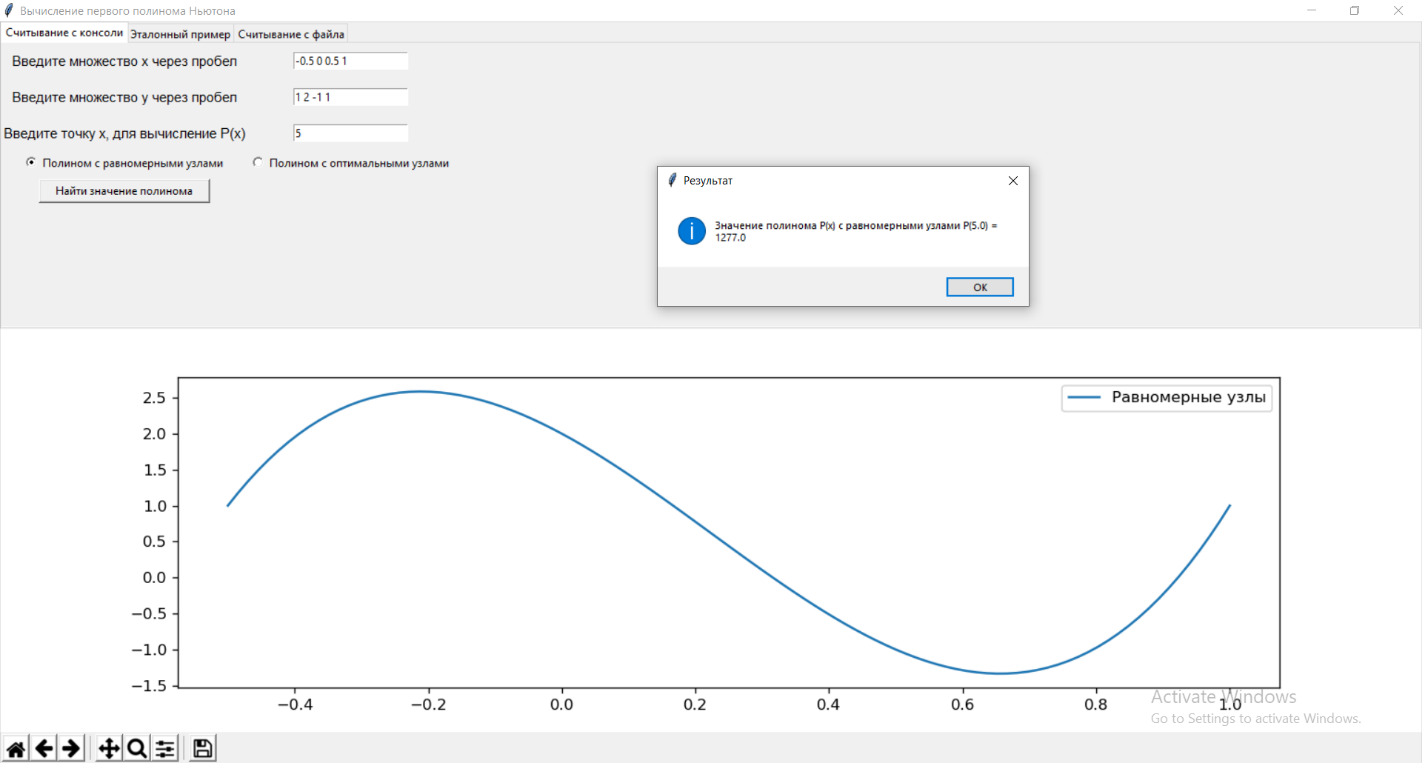
**Скриншоты работы приложения**



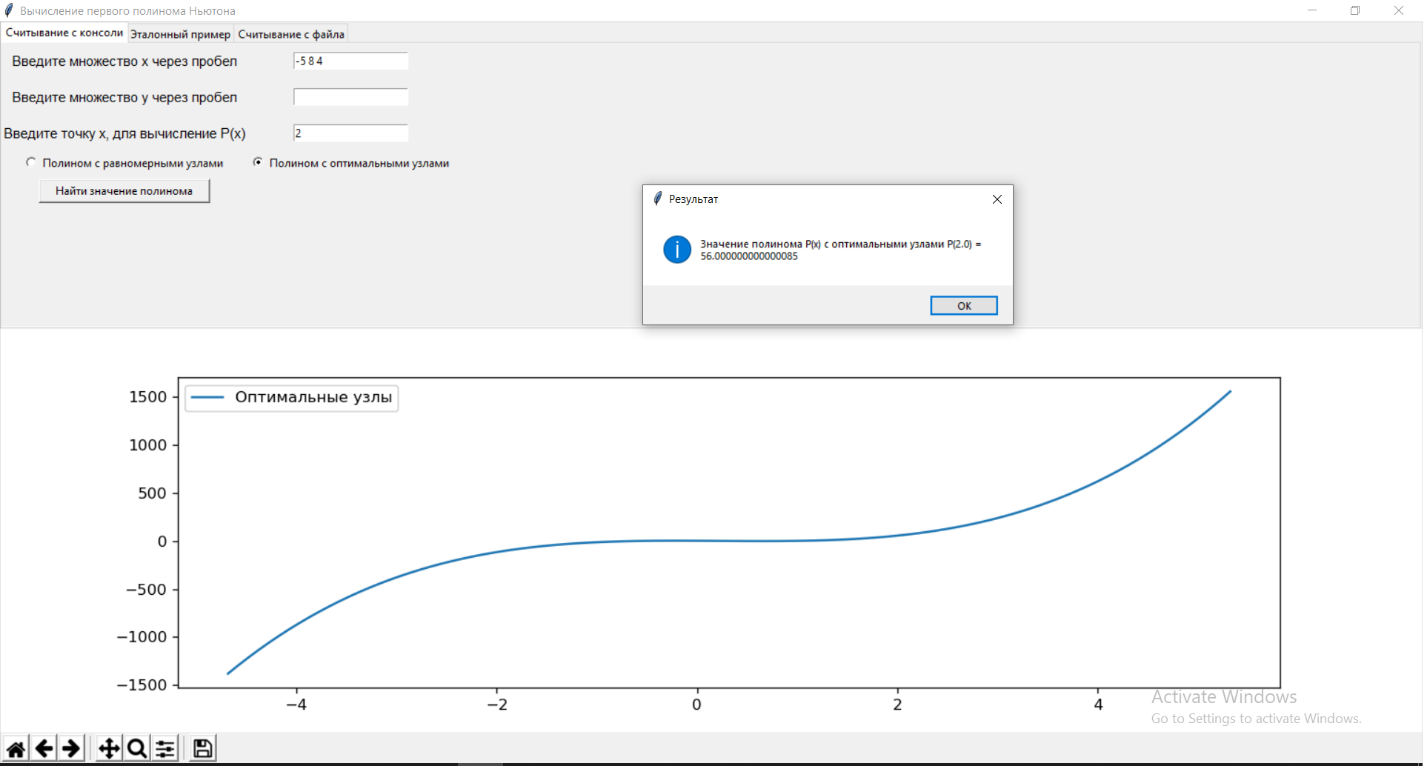
Скриншот 1



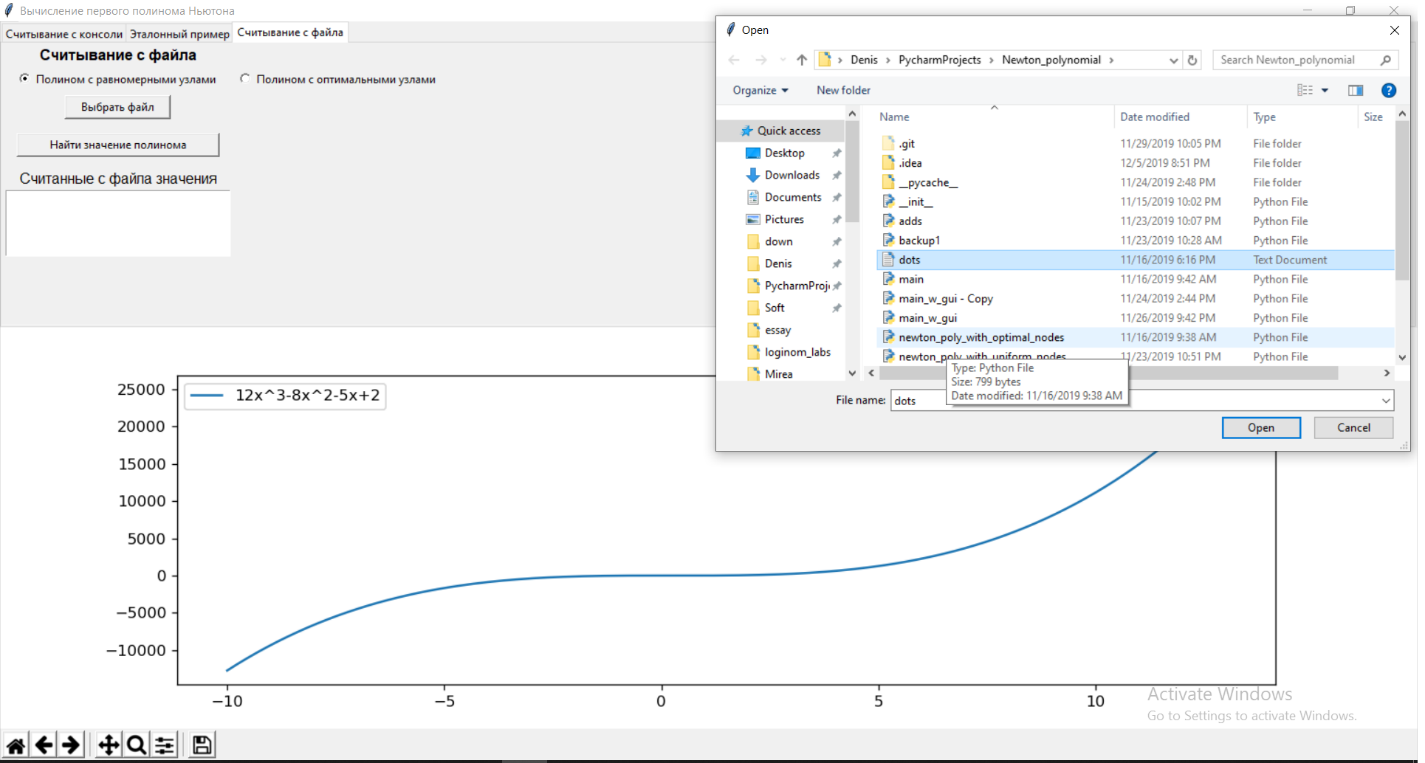
Скриншот 2



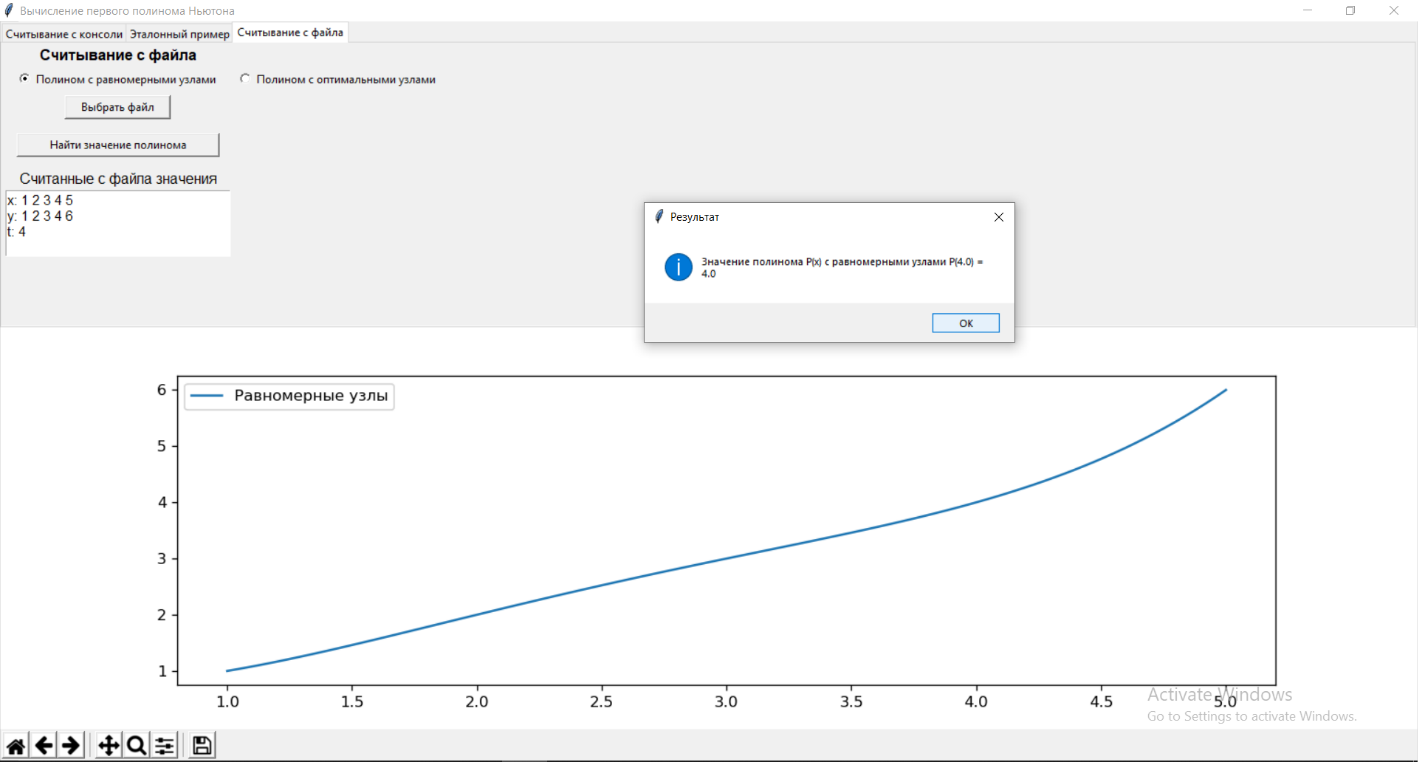
Скриншот 3



Скриншот 4



Скриншот 5



Скриншот 6

**4.Заключение**

Было разработано вычисляющее приложение для первого интерполяционного полинома Ньютона со следующим функционалом:

Мы можем осуществлять ввод вручную, явно указав координаты узлов по оси Х и по оси У, задав точку, в которой хотим вычислить значение полинома.

В случае если узлов много, то можно осуществлять ввод данных в программу посредством чтения файла. Мы указываем файл, с которого читаются данные. Данные должны быть записаны в файле в специальном формате –

* Первая строка чисел – множество X
* Вторая строка чисел в файле – множество Y
* Третья строка содержит число, в которой следует вычислить значение полинома.

Убедились на примере, что в случае с неравномерными узлами нам стоит пользоваться алгоритмом, который базируется на оптимальных узлах, иначе мы будем получать неверный ответ, если будем использовать алгоритм с равномерными узлами.

По итогу работы мы исследовали первый интерполяционный полином Ньютона на практике, написав приложение, которое осуществляет соответствующие вычисления с равномерными и оптимальными узлами. Реализовали различные способы работы программы.