Министерство образования и науки РФ

ФГБПОУ ВПО Тульский государственный университитет

КАФЕДРА АВТОМАТИКИ И ТЕЛЕМЕХАНИКИ

**АЛГЕБРАИЧЕСКОЕ ИНТЕРПОЛИРОВАНИЕ**

Лабораторная работа № 2

по курсу «Вычислительный практикум»

Вариант № 4


Выполнил: студент группы 220601 ________ Белым А.А.

(подпись)

Проверил: к. ф.-м. н., доцент ________ Карцева А.С.

(подпись)


Тула 2011

## Цель работы

Цель работы заключается в том, чтобы изучить различные методы интерполяции и написать программу, реализующий один из таких методов.

## Задание на работу

По значениям функции f(x) построить полином Ньютона с разделенными разностями.

$$f(x) = x^4 + 3x - 1$$

## Теоретическая справка

При проведении эксперимента, при табулировании сложных функций результат получают в виде таблично-заданной функции.

Таблицы делятся на два вида: регулярные (с равноотстоящими узлами) и нерегулярные:

$$x_i = x_0 + i_h. \qquad 1$$

Величину $h$ называют шагом таблицы. У нерегулярных таблиц точки по оси абсцисс размещаются произвольно. Точки с координатами $(x_i, y_i), i = 0, 1, \ldots, n$ называют узлами интерполяции.

Интерполяцию функций понимают в двух значениях.

В узком значении под интерполяцией понимают отыскание величин таблично заданной функции, соответствующих промежуточным (межузловым) значениям аргумента, отсутствующим в таблице .

Под интерполяцией в широком смысле понимают отыскание аналитического вида функции $y = F(x)$, выбранной из определѐнного класса функций и точно проходящую через узлы интерполяции . При этом задача формулируется таким образом : на отрезке $[a, b]$ заданы $(n+1)$ точки $x_0, x_1, x_2, \ldots, x_n$ и значения некоторой функции $f(x)$ в этих точках :

$$f(x_0) = y_0, f(x_1) = y_1, \ldots, f(x_n) = y_n. \qquad (1)$$

Вид функции либо неизвестен вовсе, либо он неудобен для расчѐтов (сложная функция, которую необходимо при расчѐтах интегрировать, дифференцировать и т. п.) .

Требуется построить функцию $F(x)$ (интерполирующую функцию), принимающую в узлах интерполяции те же значения, что и исходная функция $f(x)$, т. е.

$$F(x_0) = y_0, F(x_1) = y_1, \ldots, F(x_n) = y_n. \tag{2}$$

В такой постановке задача имеет бесконечное множество решений, или совсем не имеет. Однако эта задача становится однозначной, если вместо произвольной функции искать полином степени не выше $n$, удовлетворяющий условиям (3):

$$F(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n \tag{3}$$

Такую задачу называют алгебраической или полиномиальной интерполяцией.

Возможность применения полиномов для интерполяции обосновывается теоремами Вейерштрасса. Применение полиномов для вычислений представляет определѐнные удобства (при интегрировании, дифференцировании, вычислении значений функции в силу свойств аддитивности членов полинома и простого вида каждого члена).

Сущность применения интерполяционных формул состоит в том, что функция $y = f(x)$, для которой известна лишь таблица значений, заменяется интерполяционным многочленом, который рассматривается как приближѐнное аналитическое выражение для функции $f(x)$. При этом, естественно, возникает вопрос о точности такого приближения и оценки погрешности, возникающей при замене $f(x)$ на $F(x)$ .

Для построения интерполирующего многочлена применяются многочисленные способы интерполяции. Рассмотрим построение интерполяционных многочленов Ньютона с разделнными разностями.

Интерполяционным многочленом Ньютона называется многочлен

$$P_n(x) = f(x_0) + f(x_0, x_1)(x - x_0) + f(x_0, x_1, x_2)(x - x_0)(x - x_1) + ...+$$
$$+ f(x_0, x_1, ..., x_n)(x - x_0)(x - x_1)...(x - x_{n-1}),$$

где $f(x_0, x_1), f(x_0, x_1, x_2), \ldots, f(x_0, x_1, ..., x_n)$ – разделенные разности, которые могут быть вычислены рекуррентно по формулам:

Разделенная разность нулевого порядка функции $f(x)$ – сама функция $f(x)$.

Разделенная разность n-го порядка определяется через разделенную разность $(n-1)$-го порядка по формуле:

$$f(x_0, x_1, ..., x_n) = \frac{f(x_1, x_2, ..., x_n) - f(x_0, x_1, ..., x_{n-1})}{x_n - x_0}$$

## Схема алгоритма

На рисунке 1 представлена схема алгоритма получения интерполяционного многочлена Ньютона с разделенными разностями.

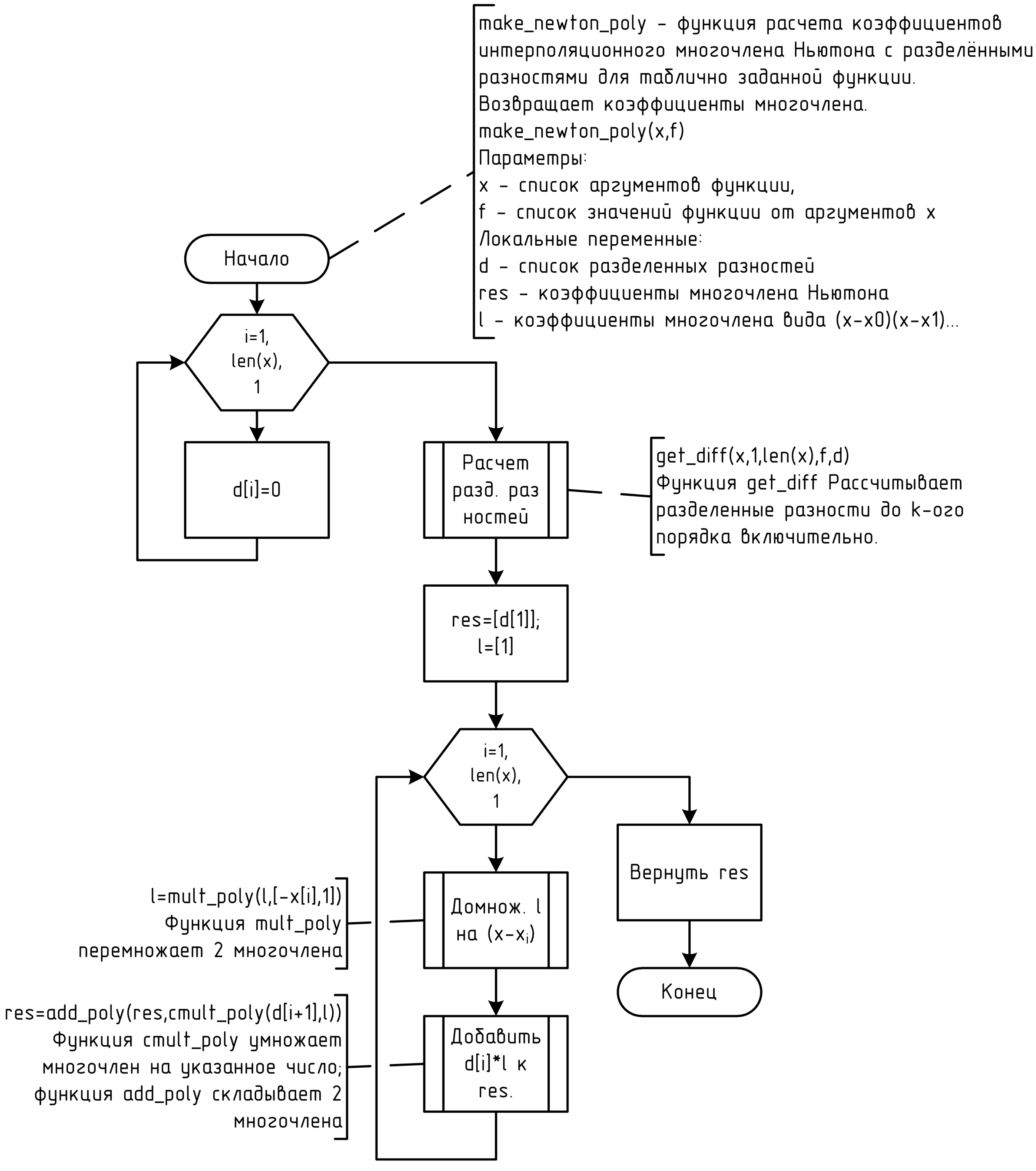


Рисунок 1 - Схема алгоритма получения интерполяционного многочлена Ньютона

На рисунке 2 представлена схема алгоритма получения разделенных разностей для интерполяционного многочлена Ньютона.


Рисунок 2 - Схема алгоритма получения разделенных разностей

На рисунке 3 представлена схема алгоритма перемножения многочленов.

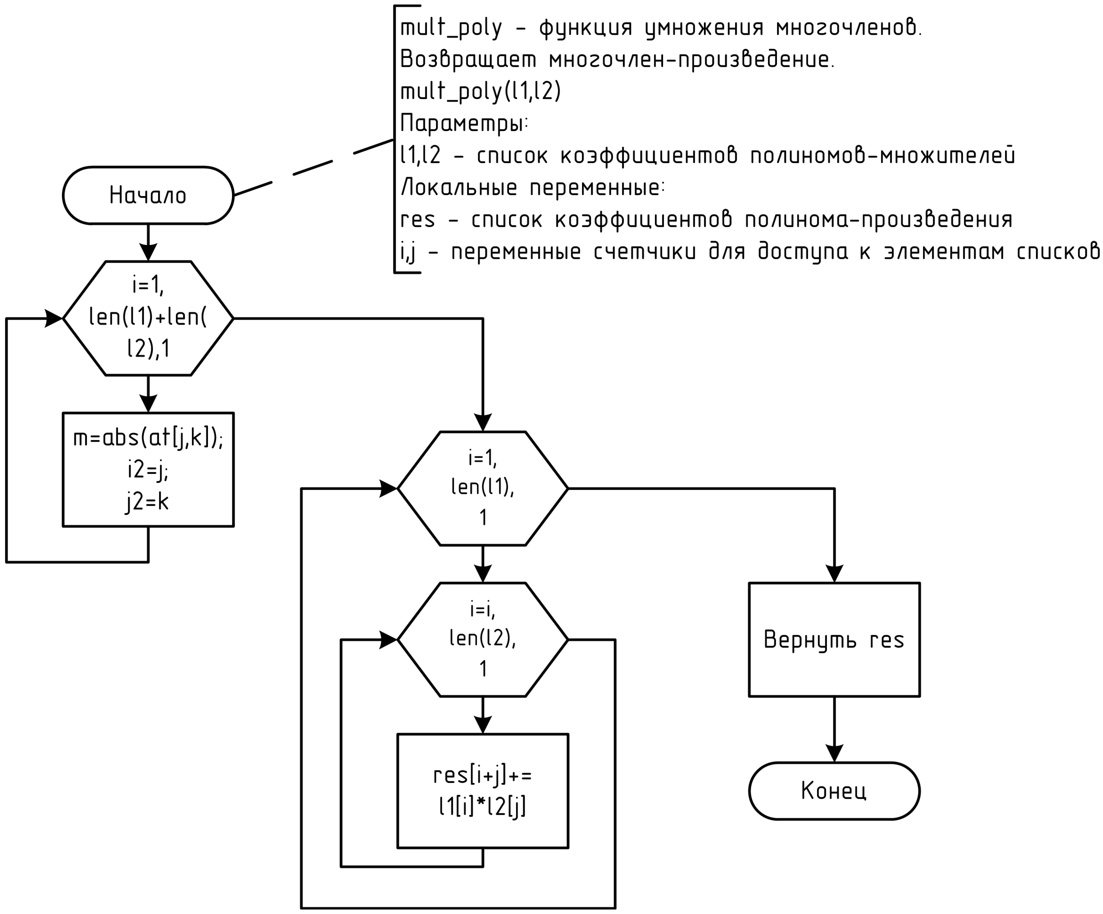


Рисунок 3 - Схема алгоритма перемножения многочленов

На рисунке 4 представлена схема алгоритма умножения многочлена на число.

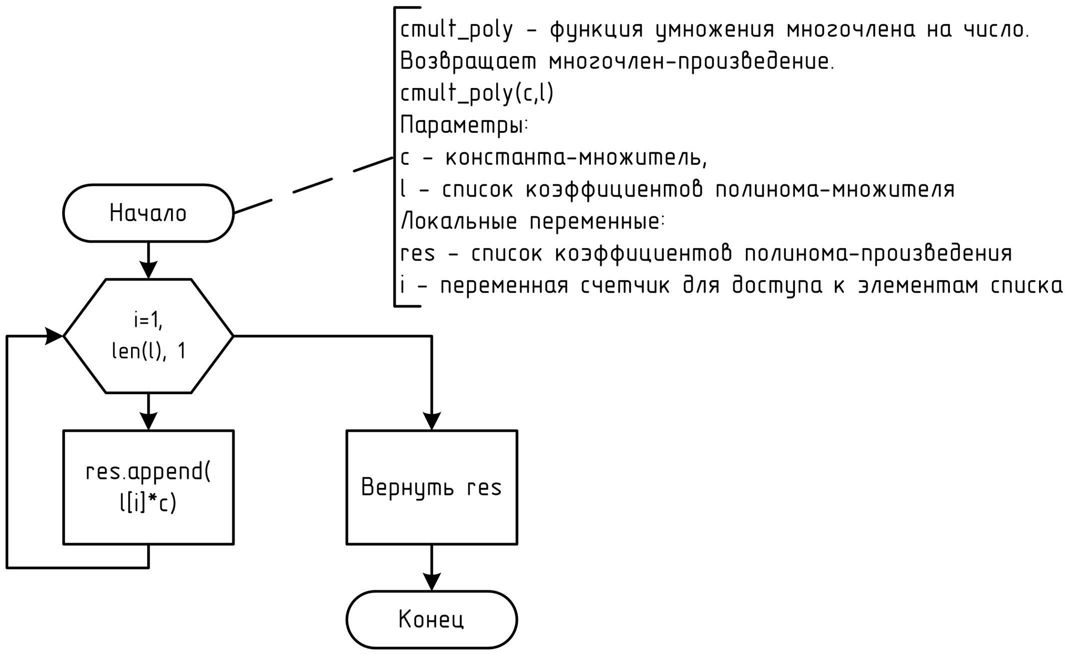


Рисунок 4 - Схема алгоритма умножения многочлена на число

На рисунке 5 представлена схема алгоритма сложения многочленов.

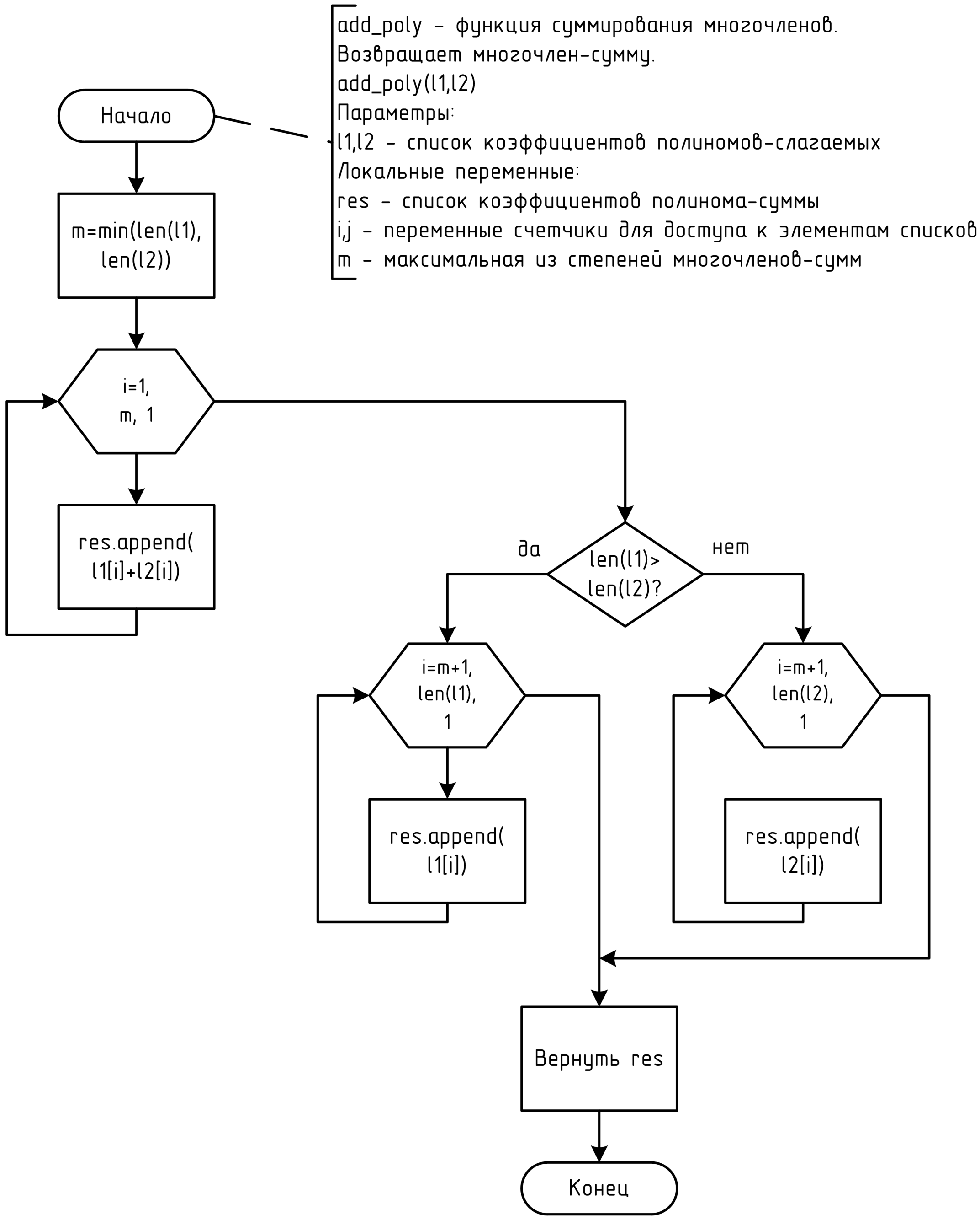


Рисунок 5 - Схема алгоритма сложения многочленов

## Инструкция пользователя

Программа позволяет построить интерполяционный многочлен Ньютона с разделенными разностями.

Программе необходимо передать списки аргументов и значений интерполируемой функции. Каждый список можно создать 2-мя способами. Список аргументов можно полностью ввести с клавиатуры, указав перед этим количество элементов списка, а можно сгенерировать значениями из некоторого диапазона - в данном случае передайте программе границы интервала и шаг изменения аргумента. Список значений можно также ввести с клавиатуры для соответствующих аргументов и в том же порядке, в котором вводились аргументы, а можно рассчитать значения для введенных уже аргументов от тестовой функции $f(x) = x^4 + 3x - 1$.

После завершения расчетов программа выведет на экран искомый интерполяционный многочлен.

## Инструкция программиста

При разработке программы построение интерполяционного многочлена Ньютона с разделенными разностями были написаны следующие процедуры и функции:

1. make_newton_poly - функция расчета коэффициентов интерполяционного многочлена Ньютона с разделёнными разностями для таблично заданной функции.

Возвращает коэффициенты многочлена.

make_newton_poly(x,f)

Параметры функции представлены в таблице 1 :

Таблица 1 - Параметры функции расчета коэффициентов интерполяционного многочлена

| **имя** | **тип** | **предназначение** |
|---|---|---|
| x | | список аргументов функции, |
| f | | список значений функции от аргументов x |

Локальные переменные функции представлены в таблице 2 :

Таблица 2 - Локальные переменные функции расчета коэффициентов интерполяционного многочлена

| имя | тип | предназначение |
|---|---|---|
| d | | список разделенных разностей |
| res | | коэффициенты многочлена Ньютона |
| l | | коэффициенты многочлена вида (x-x0)(x-x1)... |

2. get_diff - рекурсивная функция расчета разделенных разностей.

Возвращает список разделенных разностей.

get_diff(x,i,k,f,l)

Параметры  функции представлены в таблице 3 :

Таблица 3 - Параметры функции расчета разделенных разностей

| имя | тип | предназначение |
|---|---|---|
| x | | список аргументов функции, |
| i | | индекс разделенной разности |
| k | | степень разделенной разности |
| f | | список значений функции от аргументов x |
| l | | список разделенных разностей |

3. mult_poly - функция умножения многочленов.

Возвращает многочлен-произведение.

mult_poly(l1,l2)

Параметры  функции представлены в таблице 4 :

Таблица 4 - Параметры функции умножения многочленов

| имя | тип | предназначение |
|---|---|---|
| l1,l2 | | списки коэффициентов полиномов-множителей |

Локальные переменные  функции представлены в таблице 5 :

Таблица 5 - Локальные переменные функции умножения многочленов

| имя | тип | предназначение |
|---|---|---|
| res | | список коэффициентов полинома-произведения |
| i,j | | переменные счетчики для доступа к элементам списков |

4. cmult_poly - функция умножения многочлена на число.

Возвращает многочлен-произведение.

cmult_poly(c,l)

Параметры функции представлены в таблице 6 :

Таблица 6 - Параметры функции умножения многочлена на число

| имя | тип | предназначение |
|---|---|---|
| c | | константа-множитель, |
| l | | список коэффициентов полинома-множителя |

Локальные переменные функции представлены в таблице 7 :

Таблица 7 - Локальные переменные функции умножения многочлена на число

| имя | тип | предназначение |
|---|---|---|
| res | | список коэффициентов полинома-произведения |
| i | | переменная счетчик для доступа к элементам списка |

5. add_poly - функция суммирования многочленов.

Возвращает многочлен-сумму.

add_poly(l1,l2)

Параметры функции представлены в таблице 8 :

Таблица 8 - Параметры функции суммирования многочленов

| имя | тип | предназначение |
|---|---|---|
| l1,l2 | | список коэффициентов полиномов-слагаемых |

Локальные переменные функции представлены в таблице 9 :

Таблица 9 - Локальные переменные функции суммирования многочленов

| **имя** | **тип** | **предназначение** |
|---|---|---|
| res | | список коэффициентов полинома-суммы |
| i,j | | переменные счетчики для доступа к элементам списков |
| m | | максимальная из степеней многочленов-сумм |

## Текст программы

Реализиция задачи построения интерполяционного многочлена Ньютона с разделенными разностями написана на языке Python 3.2 и состоит из двух частей.

Первая часть, файл mat.py, содержит вычислительное ядро и консольный интерфейс. Исходный текст этого модуля приводится ниже.

```
def get_diff(x,i,k,f,l):
    '''
    get_diff — рекурсивная функция расчета разделенных разностей.
      Значение функции - некоторая разделенная разность.
      Возвращает через параметр список разделенных разностей.
    Параметры:
    x — список аргументов функции,
    i — индекс разделенной разности
    k — степень разделенной разности
    f — список значений функции от аргументов x
    l — список разделенных разностей
    '''
    if k==0:
        if i==0:
            l[0]=f[0]
            return l[0]
        return f[i]
    else:
        if i==0:
            l[k]=(get_diff(x,i+1,k—1,f,l)—
                  get_diff(x,i,k—1,f,l))/(x[i+k]—x[i])
            return l[k]
        return (get_diff(x,i+1,k—1,f,l)—
                get_diff(x,i,k—1,f,l))/(x[i+k]—x[i])


def mult_poly(l1,l2):
    '''
    mult_poly — функция умножения многочленов.
```

```
      Возвращает многочлен—произведение.
    Параметры:
    l1,l2 — список коэффициентов полиномов—множителей
    Локальные переменные:
    res — список коэффициентов полинома—произведения
    i,j — переменные счетчики для доступа к элементам списков
    '''
    res=[0 for i in range(0,len(l1)+len(l2)—1)]
    for i in range(0,len(l1)):
        for j in range(0,len(l2)):
            res[i+j]+=l1[i]*l2[j]
    return res

def add_poly(l1,l2):
    '''
    add_poly — функция суммирования многочленов.
      Возвращает многочлен—сумму.
    Параметры:
    l1,l2 — список коэффициентов полиномов—слагаемых
    Локальные переменные:
    res — список коэффициентов полинома—суммы
    i,j — переменные счетчики для доступа к элементам списков
    m — максимальная из степеней многочленов—сумм
    '''
    res=[]
    m=min(len(l1),len(l2))
    for i in range(0,m):
        res.append(l1[i]+l2[i])
    if len(l1)>len(l2):
        for i in range(m,len(l1)):
            res.append(l1[i])
    else:
```

```
        for i in range(m,len(l2)):
            res.append(l2[i])

    return res

def cmult_poly(c,l):
    '''
    cmult_poly — функция умножения многочлена на число.
      Возвращает многочлен—произведение.
    Параметры:
    c — константа—множитель,
    l — список коэффициентов полинома—множителя
    Локальные переменные:
    res — список коэффициентов полинома—произведения
    i — переменная счетчик для доступа к элементам списка
    '''
    res=[]
    for i in l:
        res.append(c*i)
    return res

def make_newton_poly(x,f):
    '''
    make_newton_poly — функция расчета коэффициентов
    интерполяционного многочлена
      Ньютона с разделёнными разностями для таблично заданной функции.
      Возвращает коэффициенты многочлена.
    Параметры:
    x — список аргументов функции,
    f — список значений функции от аргументов x
    Локальные переменные:
    d — список разделенных разностей
    res — коэффициенты многочлена Ньютона
```

```
    l — коэффициенты многочлена вида (x—x0)(x—x1)...
    '''
    d=[0 for i in x]
    get_diff(x,0,len(x)—1,f,d)
    res=[d[0]];l=[1]
    for i in range(0,len(x)—1):
        l=mult_poly(l,[—x[i],1])
        res=add_poly(res,cmult_poly(d[i+1],l))
    return res

def input_data():
    x=input_x()
    return x,input_f(x)

def func(x):
    return x**4+3*x—1

def input_x():
    x=[];
    answ=input('Хотите ввести аргументы функции вручную? y,[n]: ');
    if answ=='y':
        n=int(input('Введите количество аргументов: '))
        for i in range (0,n):
            x_i=input('Введите x[{}]: '.format(i))
            x.append(float(x_i))
    else:
        a=float(input('Введите минимальное значение: '))
        b=float(input('Введите максимальное значение: '))
        if b<a:
            raise ValueError
        step=float(input('Введите шаг изменения: '))
        while a+step<=b:
```

```
            x.append(a);
            a+=step
        x.append(b);
    return x


def input_f(x):
    f=[]
    answ=input('Хотите ввести значения функции вручную? y,[n]: ');
    if answ=='y':
        for i in range(0,len(x)):
            f_i=input('Введите f(x[{}]): '.format(i))
            f.append(float(f_i))
    else:
        for i in x:
            f.append(func(i))
    return f


def print_poly(poly):
    if len(poly)>2:
        for i in range(len(poly)-1,1,-1):
            if poly[i]>0:
                print('+{}x^{}'.format(poly[i],i),end='')
            elif poly[i]<0:
                print('{}x^{}'.format(poly[i],i),end='')
    if len(poly)>1:
        if poly[1]>0:
            print('+{}x'.format(poly[1]),end='')
        elif poly[1]<0:
            print('{}x'.format(poly[1]),end='')
    if poly[0]>0:
        print('+{}'.format(poly[0]),end='')
    elif poly[0]<0:
        print('{}'.format(poly[0]),end='')
```

```
    print()


if __name__=='__main__':
    try:
        print('Программа строит интерполяционный многочлен Ньютона'
              ' с конечными разностями.')
        x,f=input_data()
    except ValueError as e:
        print("Ошибка ввода данных!\nИнформация об ошибке:",e)
    else:
        poly=make_newton_poly(x,f)
        print_poly(poly)
        print('Программа завершена...');
```

Вторая часть - файл inter.py - является графическим интерфейсом для вычислительного ядра первого модуля. Далее представлен текст этого второго модуля.

```
# Eliminate the need to prefix everything with "tkinter."
from tkinter import *
from math import floor,cos,sin,sqrt
from mat import make_newton_poly,func
class polynomial:
    def __init__(self,coef):
        self.coef=coef
    def __call__(self,x):
        res=0;
        p_x=1
        for i in self.coef:
            res+=p_x*i
            p_x*=x
        return res
class plotlet:
    def __init__(self,f,color,width):
        self.f=f
```

```
        self.color=color
        self.width=width
class plotter:
    def __init__(self,canv=None,dx=1,zoom=1,x0=0,y0=0,delta=1,plotlets=None):
        self.canv=canv;
        self.dx=dx
        self.x0=x0
        self.y0=y0
        self.delta=delta
        self.zoom=zoom
        if plotlets:
            self.plotlets=plotlets
        else:
            self.plotlets=[]
        if canv:
            self.canv.bind('<Button-1>',self.__lock_mouse)
            self.canv.bind('<B1-Motion>',self.__move_handler)
            if plotlets:
                self.plot()
    def draw_orts(self):
        self.canv.create_line(3,self.canv["height"],3,1,width=2,arrow=LAST)
        self.canv.create_line(0,int(self.canv["height"]),self.canv["width"],int(self.can
        min_=floor(-self.x0/self.zoom/self.delta)+1
        max_=floor((int(self.canv["width"])-self.x0)/self.zoom/self.delta)
        while min_<=max_:
            self.canv.create_line(self.x0+min_*self.delta*self.zoom,
                                  self.canv["height"],
                                  self.x0+min_*self.delta*self.zoom,
                                  int(self.canv["height"])-5,width=2)
            canv.create_text(self.x0+min_*self.zoom*self.delta,int(self.canv["height"])-
                             anchor='s',text=str(min_*self.delta),
                             font="Verdana 7",justify=CENTER,fill='black')
            min_+=1
```

```
        min_=floor(-self.y0/self.zoom/self.delta)
        max_=floor((int(self.canv["height"])-self.y0)/self.zoom/self.delta)+1
        while max_>=min_:
            self.canv.create_line(3,
                                  int(self.canv["height"])-self.y0-max_*self.delta*self.
                                  8,
                                  int(self.canv["height"])-self.y0-max_*self.zoom*self.d
                                  width=2)
            canv.create_text(11,int(self.canv["height"])-self.y0-max_*self.delta*self.zo
                             anchor='w',text=str(max_*self.delta),
                             font="Verdana 7",justify=CENTER,fill='black')
            max_-=1
    def plot(self):
        self.canv.delete('all')
        maxx,maxy=int(self.canv["width"]),int(self.canv["height"])
        for plt in self.plotlets:
            if plt.f:
                x=-self.x0/self.zoom;y=plt.f(x)
                while x<maxx/self.zoom:
                    x2=x+self.dx;y2=plt.f(x2)
                    canv.create_line(self.x0+self.zoom*x, #x begin
                                     maxy-self.zoom*y-self.y0, #y begin
                                     self.x0+self.zoom*x2, #x end
                                     maxy-self.zoom*y2-self.y0, #y end
                                     width=plt.width,fill=plt.color)
                    x,y=x2,y2
        self.draw_orts()
    def __lock_mouse(self,event):
        self._lock_x,self._lock_y=event.x-self.x0,int(self.canv["height"])-event.y-self.
    def __move_handler(self,event):
        self.x0=event.x-self._lock_x
        self.y0=int(self.canv["height"])-event.y-self._lock_y
```

```
        self.plot()


def add_click(event):
    try:
        float(addl.get())
    except:
        pass
    else:
        lb.insert(END,addl.get())
def del_click(event):
    x=lb.curselection()
    while x:
        lb.delete(x[0])
        x=lb.curselection()
def run_click(event):
    global poly
    x=[float(i) for i in lb.get(0, END)]
    y=[func(i) for i in x]
    print(1)
    coeff=make_newton_poly(x,y)
    poly=polynomial(coeff)
    print(2)
    pl.plotlets.append(plotlet(poly,"red",3))
    pl.plot()
    print(3)
def zoom_move(event):
    pl.zoom=sclzoom.get()
    pl.plot()
root=Tk()
canv=Canvas(root,width=500,height=500,bg="white")


def det_move(event):
```

```
    try:
        pl.dx=1/scldet.get()
    except:
        pass
    else:
        pl.plot()

def delta_enter(event):
    try:
        pl.delta=float(ec.get())
    except:
        pass
    else:
        pl.plot()

def check(event):
    try:
        x=float(e1.get())
    except:
        exit()
    f1.set(str(func(x)));
    try:
        f2.set(str(poly(x)))
    except:
        pass
frm1=Frame(root,width=150,height=350)
delb=Button(frm1,text="Удалить")
delb.bind("<Button-1>",del_click)
lb = Listbox(frm1,selectmode=EXTENDED)
scr = Scrollbar(frm1,command=lb.yview)
lb.configure(yscrollcommand=scr.set)
```

```
addl=Entry(frm1,width=20)

addb=Button(frm1,text='+')
addb.bind("<Button-1>",add_click)
addb.bind("<Return>",add_click)
runb=Button(frm1,text="Посчитать")
runb.bind('<Button-1>',run_click)
frm1.grid(row=0,column=0)
delb.grid(row=0,column=0,columnspan=2)

lb.grid(row=1,column=0)
scr.grid(row=1,column=1,sticky=N+S)
addl.grid(row=3,column=0)
addb.grid(row=3,column=1)
runb.grid(row=4,column=0,columnspan=2)
canv.grid(row=0,column=1)

frm2=Frame(root,width=150,height=350)
lz=Label(frm2,text="Масштаб:")
sclzoom=Scale(frm2,from_=1,to=500,orient=HORIZONTAL)
sclzoom.set(30)
ld=Label(frm2,text="Детализация:")
scldet=Scale(frm2,from_=1,to=100,orient=HORIZONTAL)
scldet.set(10)
lc=Label(frm2,text="Цена деления:")
ec=Entry(frm2)
l1=Label(frm2,text="Проверка точки:")
e1=Entry(frm2,width=20)
f1,f2=StringVar(),StringVar()
l2=Label(frm2,text="Значение интерполируемой функции:")
e2=Entry(frm2,width=20,textvariable=f1)
l3=Label(frm2,text="Значение интерполирующей функции:")
e3=Entry(frm2,width=20,textvariable=f2)
```

```
b2=Button(frm2,text="Посчитать")


sclzoom.bind("<B1—Motion>",zoom_move)

scldet.bind("<B1—Motion>",det_move)

ec.bind("<Return>",delta_enter)

b2.bind("<Button—1>",check)

lz.grid(row=0,column=0)

sclzoom.grid(row=0,column=1)

ld.grid(row=1,column=0)

scldet.grid(row=1,column=1)

lc.grid(row=2,column=0)

ec.grid(row=2,column=1)

l1.grid(row=4,column=0)

e1.grid(row=5,column=0)

b2.grid(row=3,column=1,rowspan=4)

l2.grid(row=3,column=2)

e2.grid(row=4,column=2)

l3.grid(row=5,column=2)

e3.grid(row=6,column=2)

frm2.grid(row=1,column=0,columnspan=2)

mainfunc=plotlet(func,"blue",3)

pl=plotter(canv,0.1,30,plotlets=[mainfunc])

pl.plot()

root.mainloop()
```

## Тестовый пример

Ниже на рисунке 6 представлен пример работы программы при построении полинома Ньютона второй степени для функции $f(x) = x^4 + 3x - 1$.

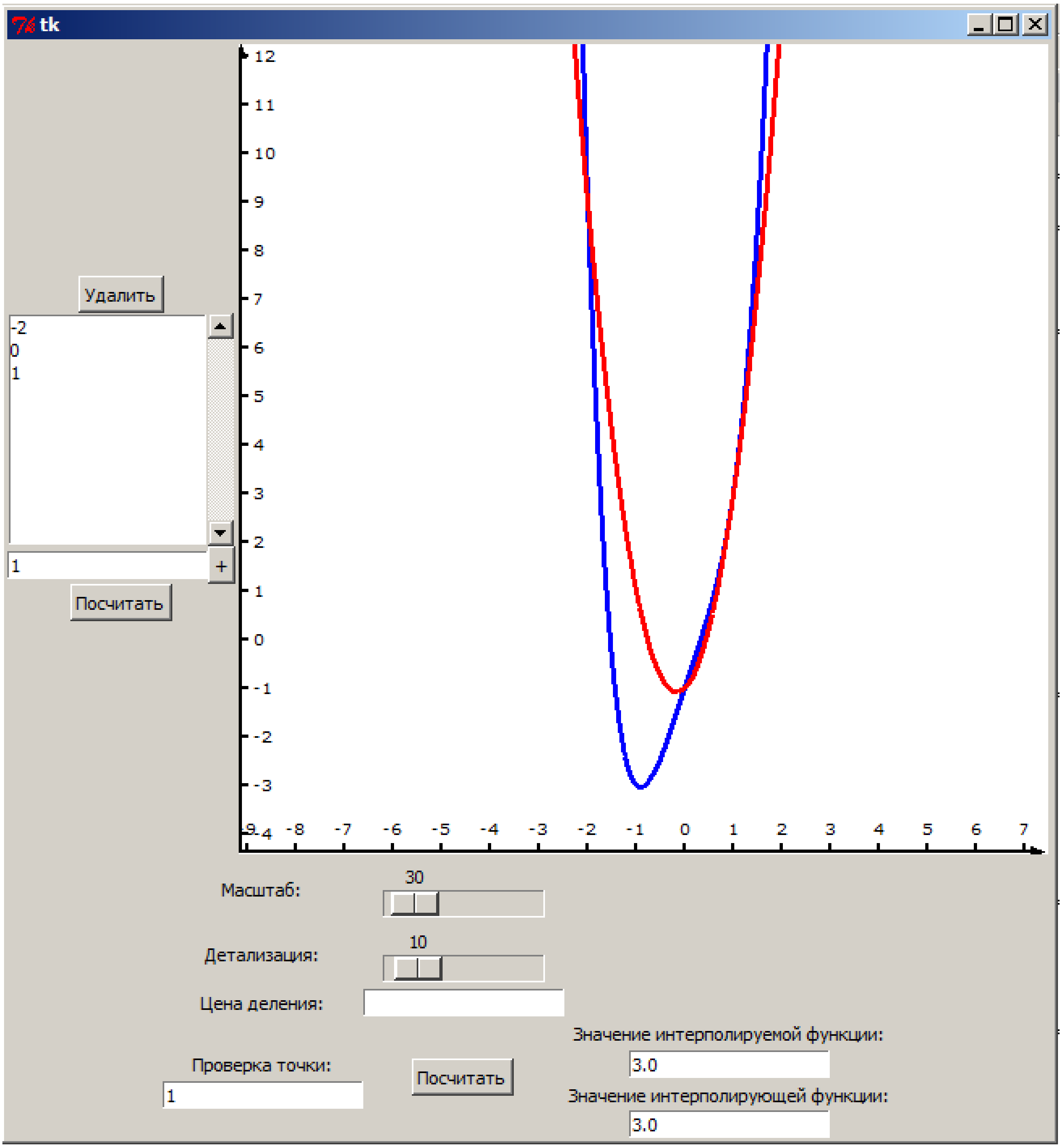


Рисунок 6 - Пример работы программы

# Вывод

В этой лабораторной работе я изучил различные методы интерполяции. Итерполяция необходима, когда из-за сложности исследуемой функции трудно провести её анализ, но допустимо взять другую, более простую функцию, которая проходит через некоторые точки исходной функции. Процесс получения такой более простой функции и называется интерполяцией. Интерполяция широко применяется в различных областях науки и техники.