Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОССУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОННИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №6

По дисциплине «Методы численного анализа»

По теме «Интерполяция таблично заданных фенкций»

Выполнил:

студент гр. 653504

Куликов А.Д.

Проверил:

Пашук А. В.

Минск 2018

**Задание.**

Интерполировать таблично заданную функцию



**Решение.**

1. **Интерполяционный многочлен Лагранжа.**

Найдем многочлен Лагранжа по формуле

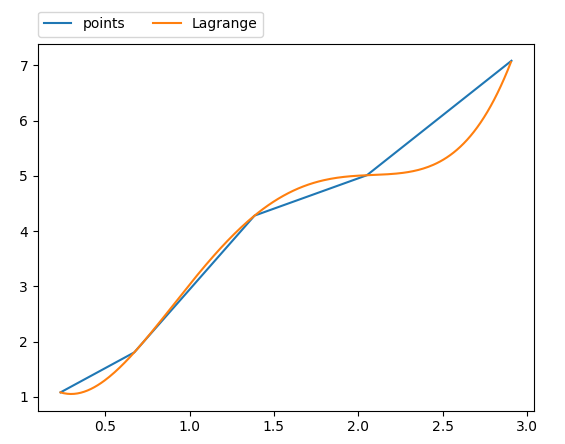
.

Найдем коэффициенты :

Подставив коэффициенты, получим

Отсюда, .

График интерполяционного многочлена Лагранжа:



1. **Таблицы конечных и разделенных разностей.**

Таблица конечных разностей:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Таблица разделенных разностей:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  | 1.654 |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

1. **Полином Ньютона**

Вычислим полином Ньютона, используя понятие конечной разности:

http://aco.ifmo.ru/el_books/numerical_methods/lectures/images/image159.png

Подставив значения, получим

Откуда . Как видим, значения многочленов Лагранжа и Ньютона совпадают.

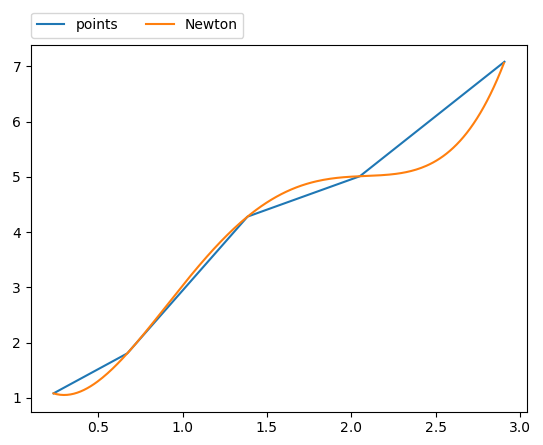


График многочлена

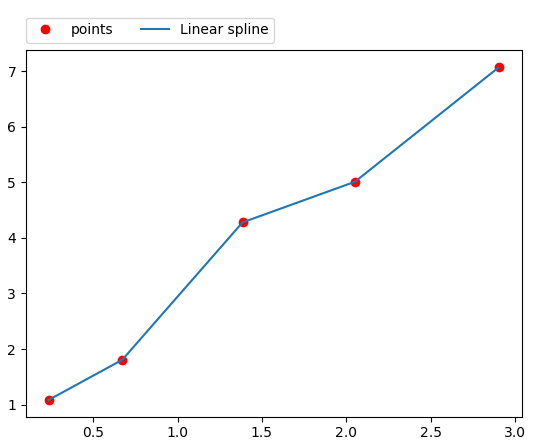
1. **Кусочно-линейный и кусочно-квадратичный сплайны.**

А) Вычислим коэффициенты линейных сплайнов, решив систему уравнений

В результате получим

Тогда полученная функция равна

График кусочно-линейного сплайна:

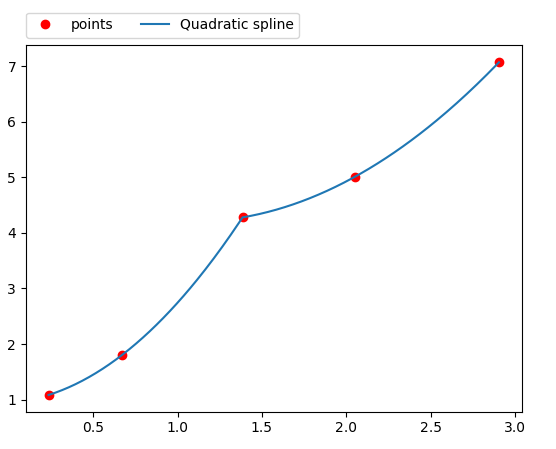


Б) Вычислим коэффициенты квадратичных сплайнов, решив систему уравнений

Решив систему, получим

В результате интерполирующая функция:

График кусочно-квадратичного сплайна:

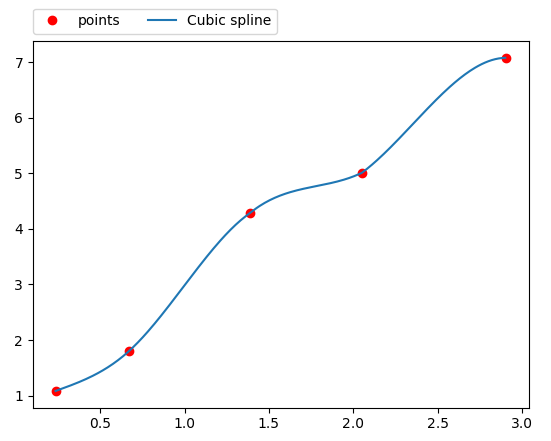


1. **Кубический интерполяционный сплайн**

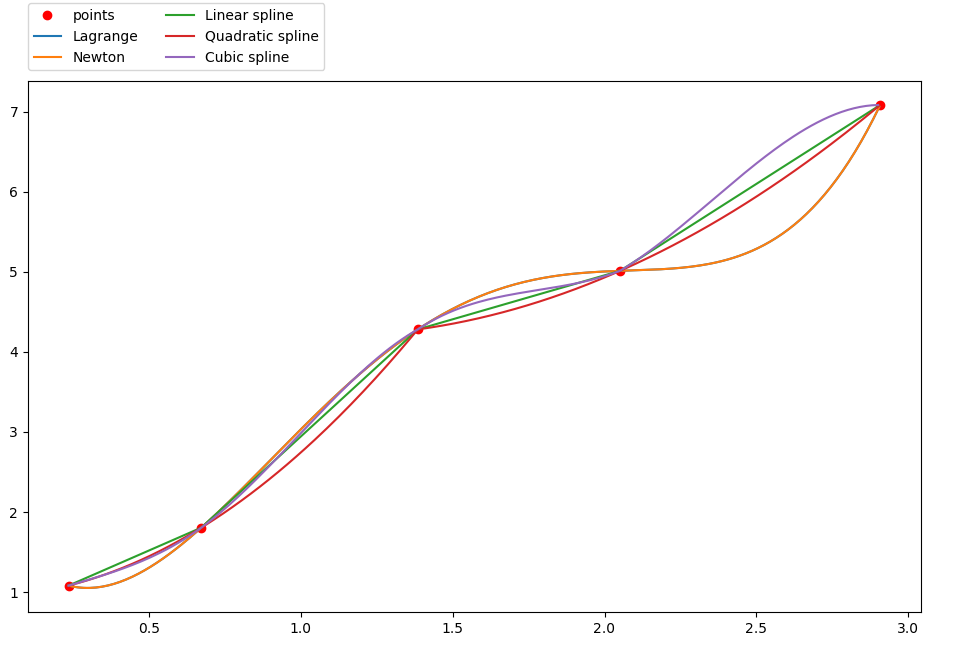
Вычислив коэффициенты методом прогонки, найдем все остальные коэффициенты.

В результате получим функцию:

График кубического сплайна:



1. **Графики интерполяций**



**Приложение**

import numpy as np

from matplotlib import pyplot as plt

def lagrange(x, y, x0):

n = len(x)

res = 0

for i in range(n):

b = 1

a = 1

for j in range(n):

if j != i:

a \*= x0 - x[j]

b \*= x[i] - x[j]

res += (a / b) \* y[i]

return res

def finite\_difference(x, y):

n = len(x)

M = np.zeros((n, n + 1))

M[:, 0] = y

for i in range(1, n):

for j in range(n - i):

M[j, i] = M[j + 1, i - 1] - M[j, i - 1]

return M

def divided\_difference(x, y):

n = len(x)

M = np.zeros((n, n + 1))

M[:, 0] = y

for i in range(1, n):

for j in range(n - i):

M[j, i] = (M[j + 1, i - 1] - M[j, i - 1]) / (x[j + i] - x[j])

return M

def newton(x, y, x0, m=None):

n = len(x)

if not m:

m = divided\_difference(x, y)

res = 0

coeffs = m[0, :]

for i in range(n):

a = 1

for j in range(i):

a \*= x0 - x[j]

res += coeffs[i] \* a

return res

def linear\_approx(x, y):

n = len(x)

a = np.zeros(n)

b = np.zeros(n)

for i in range(1, n):

a[i - 1] = (y[i - 1] - y[i]) / (x[i - 1] - x[i])

b[i - 1] = y[i] - a[i - 1] \* x[i]

return a, b

def quadratic\_approx(x, y):

n = len(x)

a = np.zeros((n - 2, 3))

for i in range(1, n - 1):

a[i - 1, 2] = (y[i + 1] - y[i - 1]) / ((x[i + 1] - x[i - 1]) \* (x[i + 1] - x[i])) - \

(y[i] - y[i - 1]) / ((x[i] - x[i - 1]) \* (x[i + 1] - x[i]))

a[i - 1, 1] = (y[i] - y[i - 1]) / (x[i] - x[i - 1]) - a[i - 1, 2] \* (x[i] + x[i - 1])

a[i - 1, 0] = y[i - 1] - a[i - 1, 1] \* x[i - 1] - a[i - 1, 2] \* x[i - 1] \*\* 2

return a

def build\_3diag(x):

n = len(x)

A = np.zeros(n)

B = np.zeros(n)

C = np.zeros(n)

for i in range(1, n):

A[i] = x[i] - x[i - 1]

for i in range(n - 1):

B[i] = x[i + 1] - x[i]

for i in range(n):

C[i] = 2 \* (x[i + 1] - x[i])

def cube\_splines(x, y):

n = len(x)

h = np.zeros(n)

l = np.zeros(n)

delta = np.zeros(n)

lam = np.zeros(n)

for i in range(n - 1):

h[i] = x[i + 1] - x[i]

l[i] = (y[i + 1] - y[i]) / h[i]

delta[0] = - 1/2 \* h[1] / (h[0] + h[1])

lam[0] = 3/2 \* (l[1] - l[0]) / (h[1] + h[0])

for i in range(2, n):

delta[i - 1] = - h[i] / (2 \* h[i - 1] + 2 \* h[i] + h[i - 1] \* delta[i - 2])

for i in range(2, n):

lam[i - 1] = (2 \* l[i] - 3 \* l[i - 1] - h[i - 1] \* lam[i - 2]) / (2 \* h[i - 1] + 2 \* h[i] + h[i - 1] \* delta[i - 2])

c = np.zeros(n)

a = np.zeros(n)

a[:-1] = y[1:]

b = np.zeros(n)

d = np.zeros(n)

for i in range(n - 1, 0, -1):

c[i - 1] = delta[i - 1] \* c[i] + lam[i - 1]

for i in range(1, n - 1):

b[i] = l[i] + 2/3 \* c[i] \* h[i] + 1/3 \* h[i] \* c[i - 1]

d[i] = (c[i] - c[i - 1]) / (3 \* h[i])

b[0] = l[0] + 2/3 \* c[0] \* h[0]

d[0] = c[0] / (3 \* h[0])

return a, b, c, d

def get\_line\_y(x, k, b):

return k \* x + b

def get\_parabola\_y(x, a, b, c):

return a \* x \*\* 2 + b \* x + c

def get\_cube\_y(x0, a, b, c, d, xk):

return a + b \* (x0 - xk) + c \* (x0 - xk)\*\*2 + d \* (x0 - xk)\*\*3

def main():

x = np.array([0.235, 0.672, 1.385, 2.051, 2.908])

y = np.array([1.082, 1.805, 4.280, 5.011, 7.082])

plt.plot(x, y, 'ro', label='points')

xpts = np.arange(np.min(x), np.max(x), 0.01)

ypts\_lagrange = np.array([lagrange(x, y, t) for t in xpts])

plt.plot(xpts, ypts\_lagrange, label='Lagrange')

ypts\_newton = np.array([newton(x, y, t) for t in xpts])

plt.plot(xpts, ypts\_newton, label='Newton')

print("L4(x1+x2) = " + str(lagrange(x, y, x[1] + x[2])))

print("N4(x1+x2) = " + str(newton(x, y, x[1] + x[2])))

a, b = linear\_approx(x, y)

ypts\_linear = []

xpts\_linear = []

for i in range(1, len(x)):

cur\_xpts = np.arange(x[i - 1], x[i], 0.01)

xpts\_linear += list(cur\_xpts)

ypts\_linear += [get\_line\_y(t, a[i - 1], b[i - 1]) for t in cur\_xpts]

plt.plot(xpts\_linear, ypts\_linear, label='Linear spline')

a = quadratic\_approx(x, y)

ypts\_quadratic = []

xpts\_quadratic = []

for i in range(1, len(x) - 1, 2):

cur\_xpts = np.arange(x[i - 1], x[i + 1], 0.01)

xpts\_quadratic += list(cur\_xpts)

ypts\_quadratic += [get\_parabola\_y(t, a[i - 1, 2], a[i - 1, 1], a[i - 1, 0]) for t in cur\_xpts]

if i < len(x) - 3:

cur\_xpts = np.arange(x[-2], x[-1], 0.01)

xpts\_quadratic += list(cur\_xpts)

ypts\_quadratic += [get\_parabola\_y(t, a[-1, 2], a[-1, 1], a[-1, 0]) for t in cur\_xpts]

plt.plot(xpts\_quadratic, ypts\_quadratic, label='Quadratic spline')

a, b, c, d = cube\_splines(x, y)

xpts\_cubic = []

ypts\_cubic = []

for i in range(len(x) - 1):

cur\_xpts = np.arange(x[i], x[i + 1], 0.01)

xpts\_cubic += list(cur\_xpts)

ypts\_cubic += [get\_cube\_y(t, a[i], b[i], c[i], d[i], x[i + 1]) for t in cur\_xpts]

plt.plot(xpts\_cubic, ypts\_cubic, label='Cubic spline')

plt.legend(bbox\_to\_anchor=(0., 1.02, 1., .102), loc=3,

ncol=2, borderaxespad=0.)

plt.show()

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

main()