Лабораторная работа №5

Тужилкина Н.Г., БПМ-18-2, вариант 23.

# Сплайны

Даны точка на плоскости .

Задан порядок следования данных точек вдоль будущей кривой. Далее упомянутый порядок будет определяться нумерацией точек.

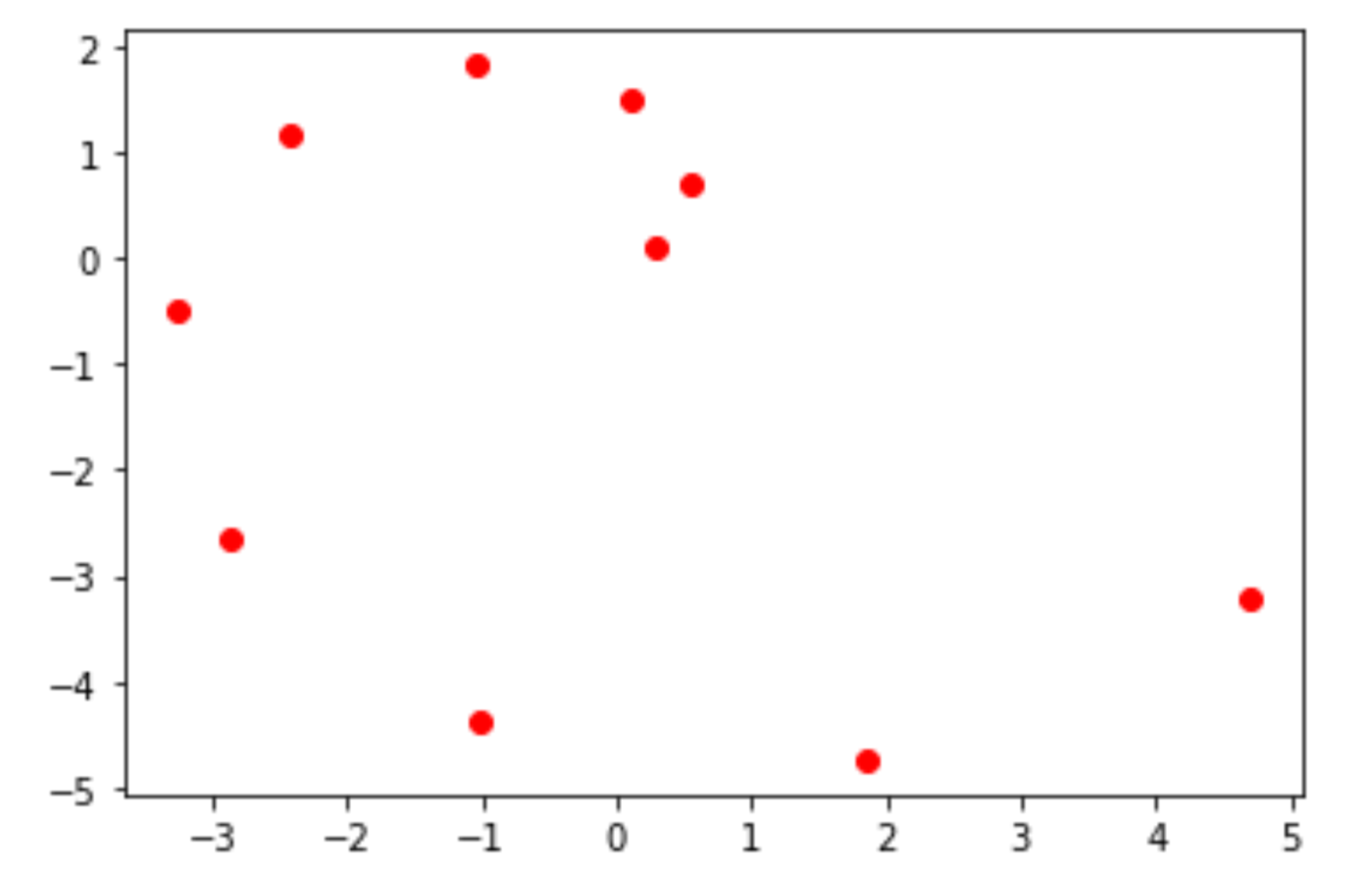
Кривая не предполагается графиком некоторой функции от переменной . Она будет строиться в параметрической форме:

## Цель работы:

1. Построить по заданным точкам аппроксимирующую кривую.
2. Построить традиционный сплайн по заданным точкам.

## Аппроксимирующая кривая

Заданы точки:



### Изложение метода

– заданная последовательность точек на плоскости. Определим вектор-функцию при всех по формуле

где функции определены при всех как

Функции являются сдвигами основной функции на величину – налево при отрицательных и направо при положительных .

Для функции

При *t* = 1, 2, …, *n* имеем

В правых частях равенств стоят двухмерные точки, которые складываются покоординатно.

Функция определена при всех . При , изменяющемся от 1 до , кривая проходит через точек, лежащих на серединах отрезков , , т. е. на серединах отрезков, концы которых являются исходными точками.

В этом смысле кривая является решением задачи аппроксимации исходной последовательности точек. Зависимость от является кусочно-квадратичной.

При построении сплайна слагаемые не зависят от исходных точек – они являются только числовыми коэффициентами при одних и тех же функциях.

### Текст программы

import matplotlib.pyplot as plt

import numpy as np

def F0(t):

  if t < -1:

    f = 0

  elif t < 0:

    f = ((t + 1) \*\* 2) / 2

  elif t < 1:

    f = -(t + 1) \*\* 2 + 3 \* (t + 1) - 3 / 2

  elif t < 2:

    f = ((t + 1) \*\* 2) / 2 - 3 \* (t + 1) + 9 / 2

  elif t >= 2:

    f = 0

  return f

def P(t):

  p = [0.0, 0.0]

  for i in range(len(points)):

    p = [x + y for x, y in zip(p, [F0(t - i) \* x for x in points[i]])]

  return p

points = [[0.285,0.088],[0.559,0.701],

            [0.112,1.490],[-1.044,1.815],

            [-2.422,1.172],[-3.252,-0.484],

            [-2.854,-2.638],[-1.008,-4.371],

            [1.845,-4.737],[4.687,-3.213]]

fig = plt.subplots()

t = np.arange(0.0, 9.0, 0.001)

f = [P(i) for i in t]

plt.plot([i[0] for i in points], [i[1] for i in points], 'ro')

plt.plot([i[0] for i in f], [i[1] for i in f])

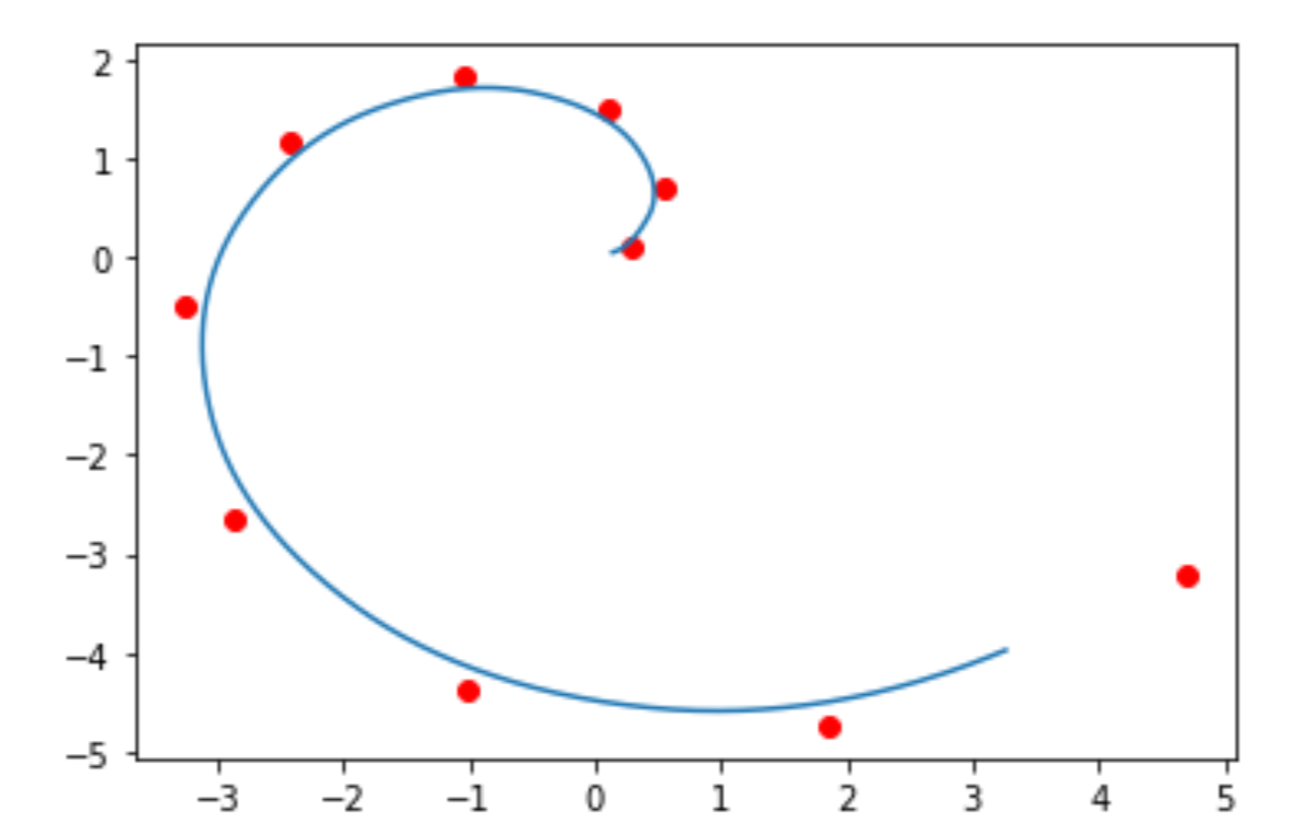
plt.show()

for i in range(1, 9):

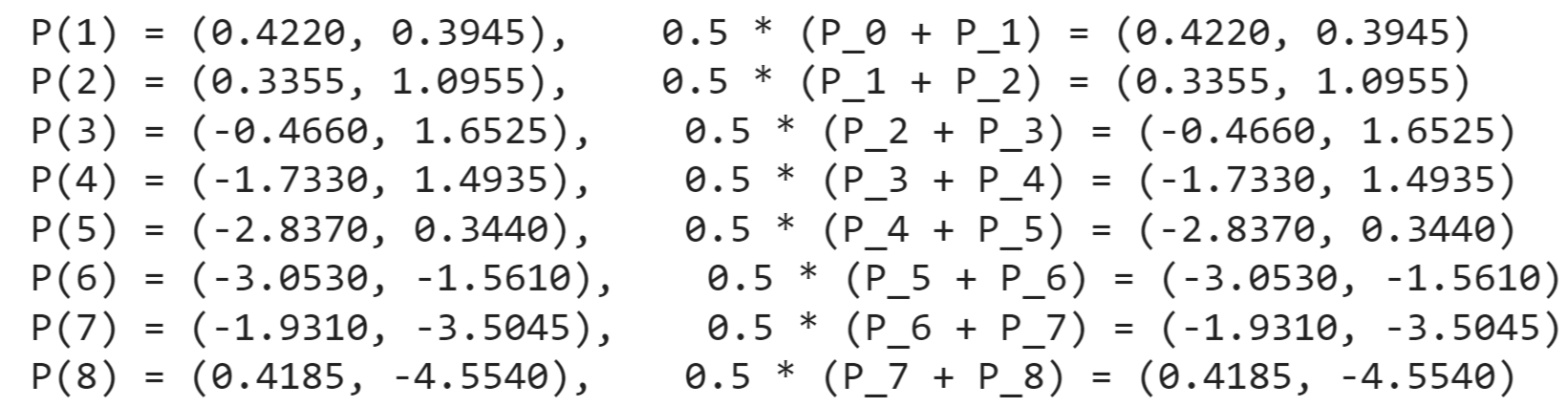
  temp = [0.5 \* (x + y) for x, y in zip(points[i-1], points[i])]

  print('P({0}) = ({1:.4f}, {2:.4f}),    0.5 \* (P\_{3} + P\_{4}) = ({5:.4f}, {6:.4f})'.format(i, P(i)[0], P(i)[1], i-1, i, temp[0], temp[1]))

### Представление результатов

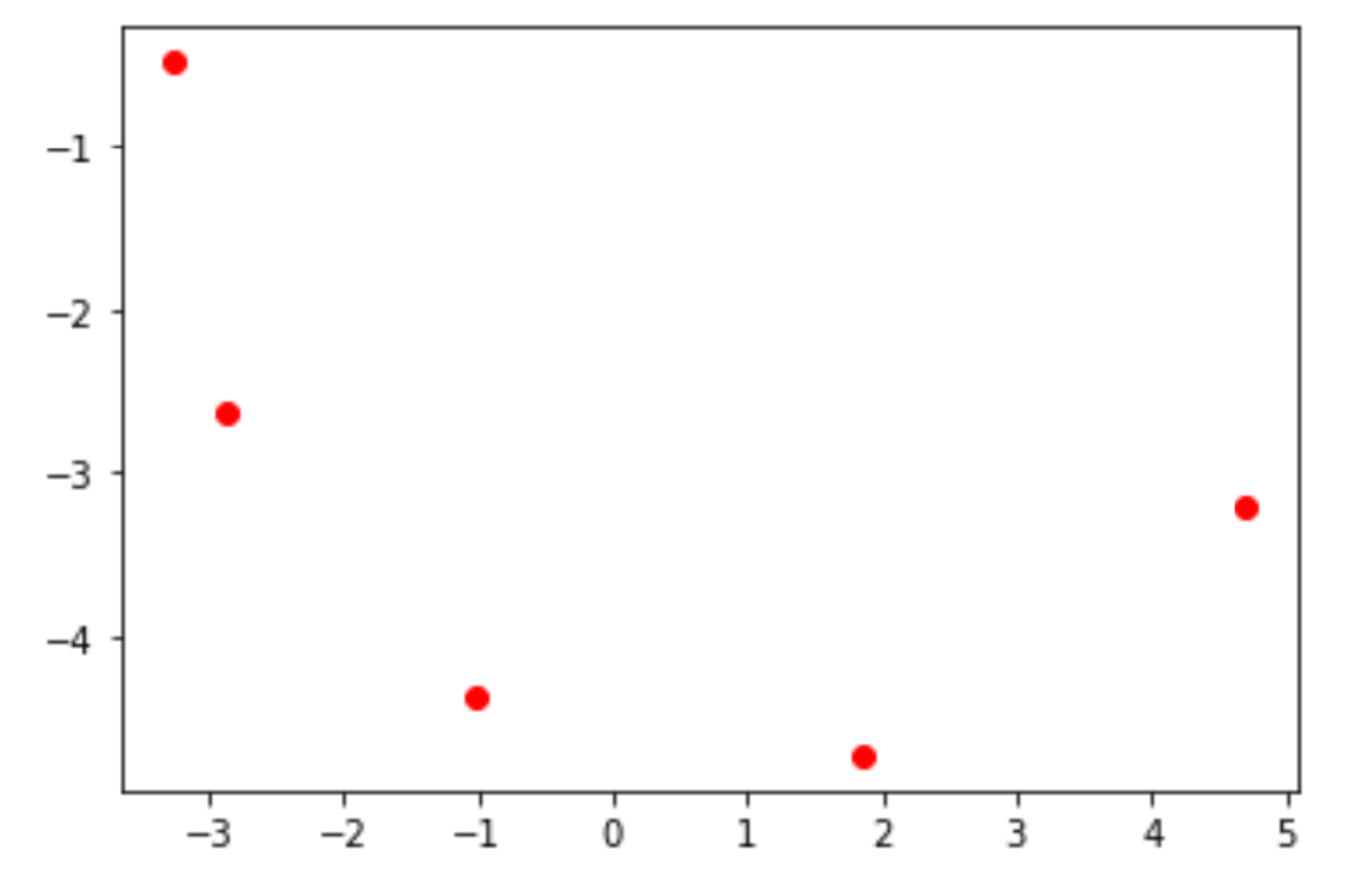


Чтобы убедиться, что функция вычислена правильно, была осуществлена проверка:



## Традиционные сплайны

Заданы точки:

**

### Изложение метода

Даны точка на плоскости: . Предполагается, что

Требуется найти функцию , такую, что

— интерполяционный квадратичный сплайн:

* определена на всём отрезке
* на каждом отрезке функция является квадратичной (т. е. многочленом 2-го порядка)
* на всём отрезке является непрерывной и непрерывно дифференцируемой функцией.

Обозначим многочлен, представляющий искомую функцию на отрезке , через

Условия для каждой из таких функций:

Условия непрерывной дифференцируемости означает совпадение производных в каждом из внутренних узлов :

Получившуюся систему уравнений можно дополнить уравнением

В каждой паре уравнений (1), (2) при фиксированном можно выразить линейно через . Для этого нужно решить систему двух линейных уравнений с двумя неизвестными , считая известным.

Подставляя эти выражения в уравнения (3), получаем систему уравнений, в каждое из которых входят только две переменные: и , .

Поскольку , то мы имеем систему уравнений c неизвестными .

Эта система является двухдиагональной, так как в -е уравнение входят только переменные и , а в 1-е уравнение – только переменная .

Из 1-го уравнения сразу находится единственная переменная , из 2-го – и т. д., вплоть до последнего значения.

Найдя все, с помощью уже найденных ранее выражений и через найдём все и , т. е. найдём искомую функцию .

### Текст программы

points2 = [[-3.252,-0.484],[-2.854,-2.638],[-1.008,-4.371],

           [1.845,-4.737],[4.687,-3.213]]

x = [i[0] for i in points2]

y = [i[1] for i in points2]

def calc\_c(i):

  left\_side = (2 \* x[i-1] \* x[i] \* (y[i] - c[i-1]) - x[i] \*\* 2 \* (y[i-1] - c[i-1]) - x[i-1] \*\* 2 \* (y[i] - c[i-1])) / (x[i-1] \* (x[i] - x[i-1]))

  right\_side = left\_side - (x[i] \*\* 2 \* y[i+1] - 2 \* x[i] \* x[i+1] \* y[i] + x[i+1] \*\* 2 \* y[i]) / (x[i+1] \* (x[i+1] - x[i]))

  result = right\_side \* x[i+1] / (x[i] - x[i+1])

  return result

def calc\_a(i):

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

def calc\_b(i):

  return (x[i+1] \*\* 2 \* (y[i] - c[i]) - x[i] \*\* 2 \* (y[i+1] - c[i])) / (x[i] \* x[i+1] \* (x[i+1] - x[i]))

c = [1]

a = [calc\_a(0)]

b = [calc\_b(0)]

for i in range(1, 4):

  c.append(calc\_c(i))

  a.append(calc\_a(i))

  b.append(calc\_b(i))

for i in range(0, 4):

  print('{0} < x < {1}: {2}\*x^2 + {3}\*x + {4}'.format(x[i], x[i+1], a[i], b[i], c[i]))

def S(n):

  if (n < x[1]):

    s = a[0] \* n \*\* 2 + b[0] \* n + c[0]

  elif (n < x[2]):

    s = a[1] \* n \*\* 2 + b[1] \* n + c[1]

  elif (n < x[3]):

    s = a[2] \* n \*\* 2 + b[2] \* n + c[2]

  else:

    s = a[3] \* n \*\* 2 + b[3] \* n + c[3]

  return s

t = np.linspace(x[0], x[4], 100)

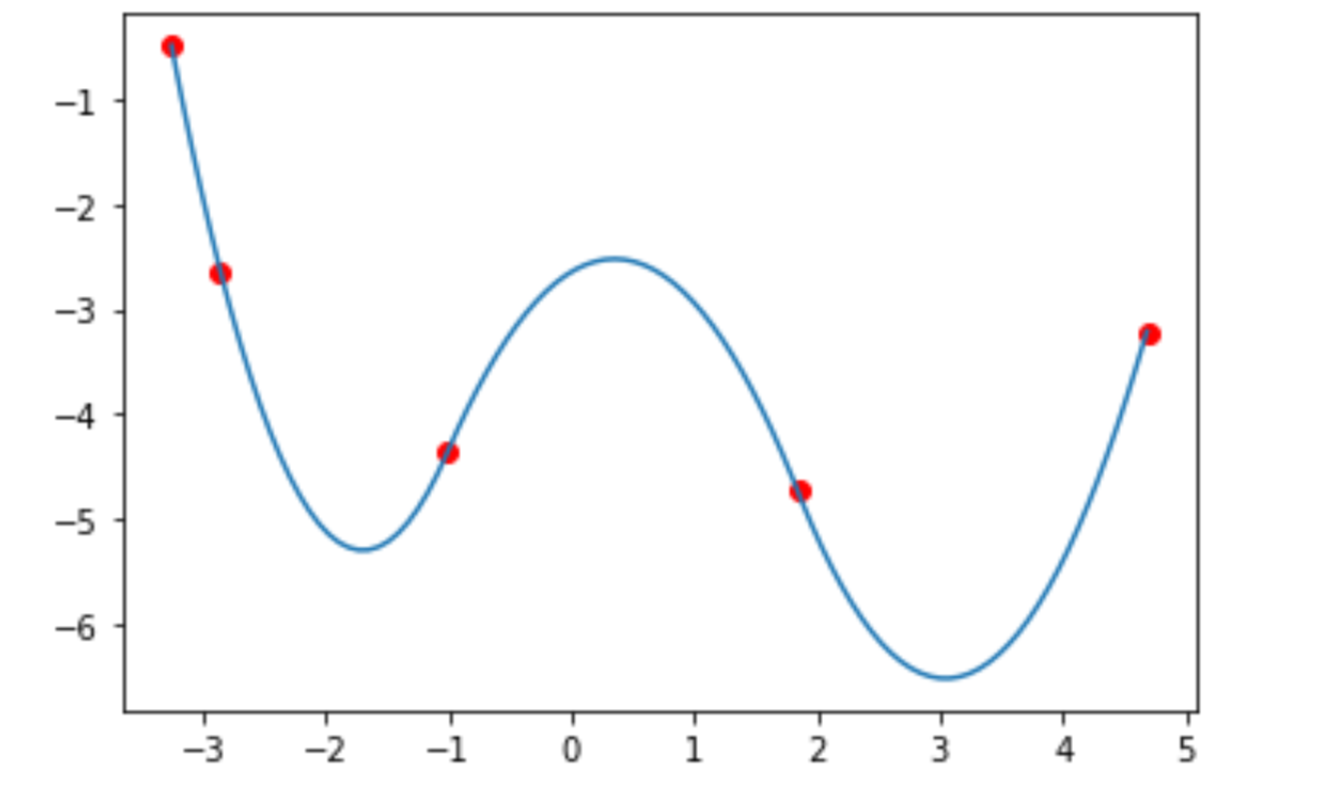
f = [S(i) for i in t]

plt.plot(x, y, 'ro')

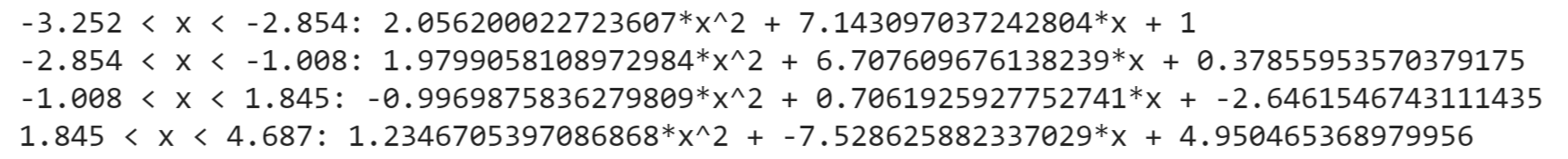
plt.plot(t, f)

plt.show()

### Представление результатов



Получившаяся функция:



## Вывод

B-сплайн не проходит точно через заданные точки, также существуют сложности с представлением функции в параметрической форме. Но кривая получается гораздо плавнее, чем традиционный сплайн:

