**Московский авиационный институт**

**(Национальный исследовательский университет)**

**Факультет: «Информационные технологии и прикладная математика»**

**Кафедра: 806 «Вычислительная математика и программирование»**

**Дисциплина: «Численные методы»**

**Курсовая работа по теме:**

**«Интерполяция экспоненциальными сплайнами»**

Студент: Ваньков Д. А.

Группа: М8О-307Б-17

Преподаватель: Ревизников Д. Л.

Дата: 30.05.2020

Оценка:

Москва, 2020

**Постановка задачи.**

Реализовать алгоритм, производящий построение экспоненциального сплайна.

**Введение.**

Известно, что кусочно-кубическая полиномиальная сплайн-интерполяция или сглаживание часто дает нежелательные точки перегиба (или нежелательные экстремумы). Для этого был реализован метод сплайн-интерполяции, который позволяет избежать этих точек перегиба и содержит в качестве особого случая кубические сплайны.

**Алгоритм.**

Для заданных пар точек , где и необходимо найти параметров, а именно , где для функции . Функция имеет вид:

здесь - коэффициент натяжения для определенного участка кривой. Тогда, принимая во внимание, что , и , можно сделать вывод, что .

Это означает, что мы ищем интерполяционную функцию, состоящую из кусочных экспонент, которые плавно соединяются в узлах . Параметры пропорциональны квадратным корням напряжений, действующих на концах отрезка на участках кривой. Тогда функцию можно записать в следующем виде:

где

,

.

Пусть

тогда все коэффициенты примут следующий вид:

Полагая, что значения можно определить коэффициенты :

.

Пусть

,

тогда:

.

Для выполнения условия вторые производные функции должны принадлежать . Тогда результат будет выглядеть:

.

Сделав замену:

,

получаем:

Для заданных и получается трехдиагональная симметричная система линейных уравнений для . Так каки для , матрица коэффициентов определена строго по диагонали и является положительно определенной.

Следовательно, существует уникальное решение и метод исключения численно стабилен без поворота. Рассчитав можно вычислить коэффициенты и, таким образом, получая требуемые значения .

**Кубическая сплайн-интерполяция.**

Если взять случай при котором 0 для фиксированного k, получаются следующие ограничения:

Можно заметить, что при является полиномом третьей степени. Если считать при умножив k-е уравнение на , получится уравнения для согласования кубических полиномов.

Это означает, что для фиксированного k для получится кубический кусок в кривой , а для при получится обычная кусочно-кубическая сплайн-функция. То есть

сделав замену:

коэффициенты приобретают вид:

Теперь функцию можно записать как:

, где

Таким образом, был получен алгоритм для выполнения сплайн-интерполяции по смешанным: кубическим и экспоненциальным.

**Устранение нежелательных точек перегиба.**

После построения нужно модифицировать экспоненциальную сплайн-интерполяцию, показывая, что можно выбрать такие параметры натяжения , чтобы нежелательные точки перегиба, которые появлялись при через определенные интервалы, исчезали.

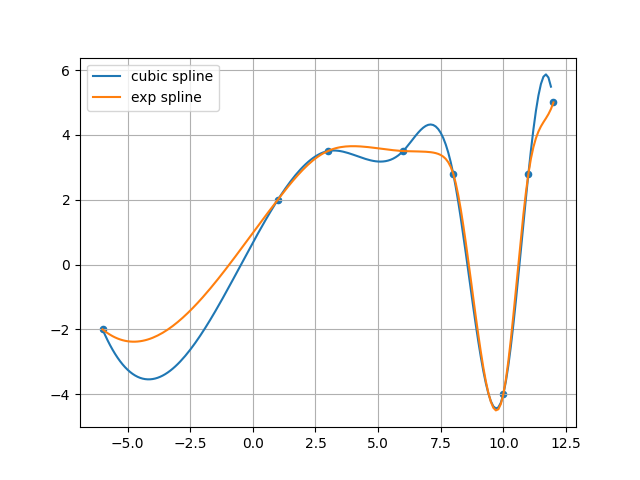
Если вторая производная данной функции не меняет знак в определенном интервале, а интерполирующая функция меняет, то такая точка перегиба будет нежелательной. Поскольку исходная функция задается только дискретными значениями в точке , используется приближение второй производной:

.

Осталось определить точку перегиба, существующую в интервале чтобы она была нежелательной, если .

**Примеры выполнения программы на тестовых функциях.**

**Тест 1.**



n: 8

x: [-6, 1, 3, 6, 8, 10, 11, 12]

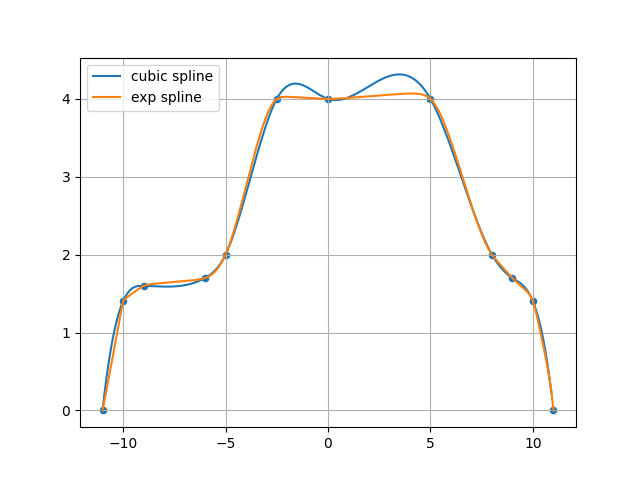
y: [-2, 2, 3.5, 3.5, 2.8, -4, 2.8, 5]

p: [0, 0, 1, 3.6, 0, 0, 0]

step: 0.1

Здесь и далее n — количество точек, списки x и y — координаты точек, p — список, определяющий натяжение между некоторыми двумя точками, step — шаг для интерполяции.

**Тест 2.**

****

n: 12

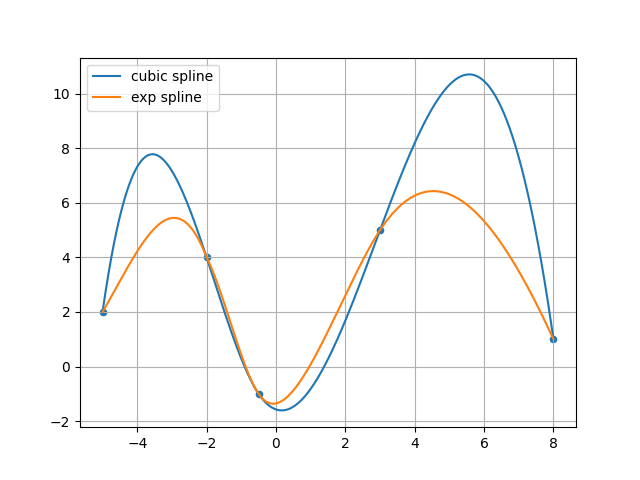
x: [-11, -10, -9, -6, -5, -2.5, 0, 5, 8, 9, 10, 11]

y: [0, 1.4, 1.6, 1.7, 2, 4, 4, 4, 2, 1.7, 1.4, 0]

p: [8.4, 8.4, 3.4, 0, 0, 5.6, 2.8, 0, 8, 7.6, 7.6]

step: 0.1

**Тест 3.**

****

n: 5

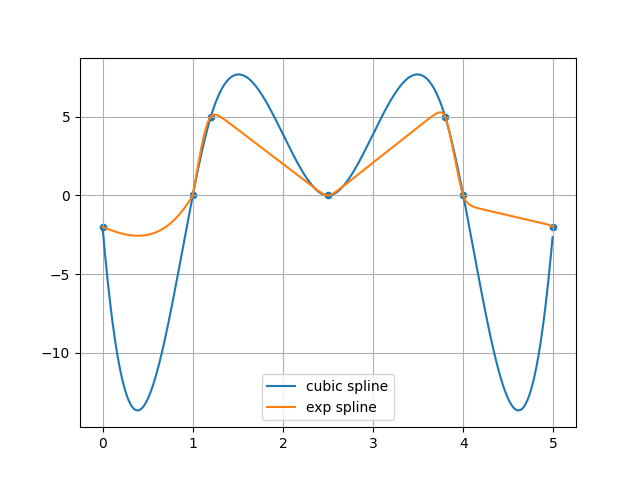
x: [-5, -2, -0.5, 3, 8]

y: [2, 4, -1, 5, 1]

p: [0, 1, 1, 0]

step: 0.01

**Тест 4.**

****

n: 7

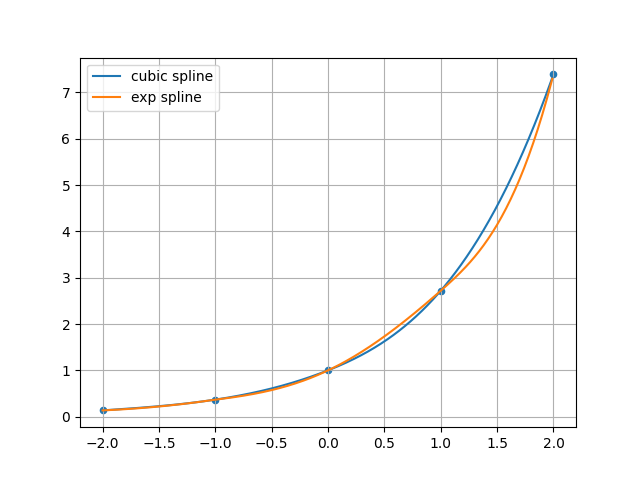
x: [0, 1, 1.2, 2.5, 3.8, 4, 5]

y: [-2, 0, 5, 0, 5, 0, -2]

p: [0, 5, 25, 25, 25, 25]

step: 0.01

**Тест 5.**

****

n: 5

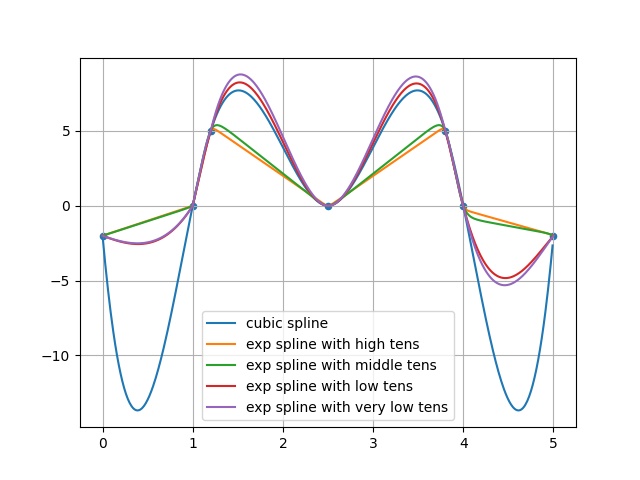
x: [-2, -1, 0, 1, 2]

y: [0.13534, 0.36788, 1.0, 2.7183, 7.3891]

p: None

step: 0.01

**Пример экспериментов с натяжением.**



P1 = [50, 50, 50, 50, 50, 50] (hight)

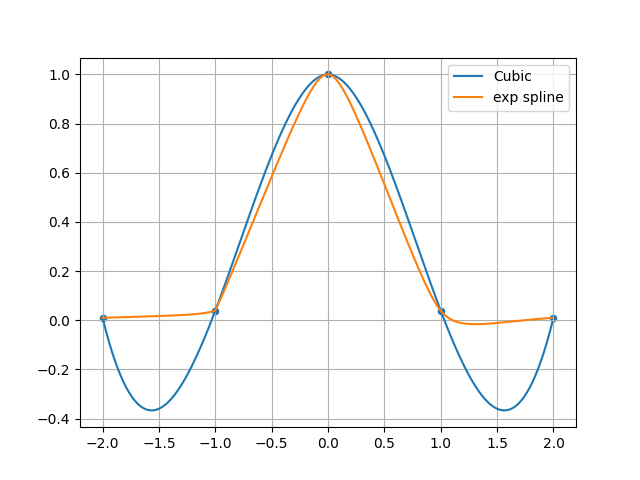
p2 = [20, 20, 20, 20, 20, 20] (middle)

p3 = [0, 0, 0, 0, 0, 0] (low)

p4 = [0.6, 0.6, 0.6, 0.6, 0.6, 0.6] (very low)

**Функция Рунге.**

Для функции Рунге построить кубический сплайн и экспоненциальный, где с шагом step = 1. Коэффициенты натяжения p = [8, 8, 8, 8].



**Листинг программы.**

import math

import numpy as np

import json

import argparse

import matplotlib.pyplot as plt

from scipy.interpolate import CubicSpline

class CoeffsException(Exception):

pass

class SplineException(Exception):

pass

def expCoeff(n, x, y, p):

"""Computation exponential spline coefficients"""

# Out coefficients

d = [0.0] \* (n + 1)

dq = [0.0] \* n

h = [0.0] \* n

hp = [0.0] \* n

ph = [0.0] \* n

# Computation elements of the tridiagonal system

index, n\_less = 0, n - 1

c, c1, c2, u, v, w = 0.0, 0.0, 0.0, y[0], 0.0, 0.0

if n < 3:

raise CoeffsException(f'The value n must be >= 3, from function \"expCoeff\"')

q = [0.0] \* (n + 1)

r = [0.0] \* (n + 1)

for i in range(1, n):

index = i - 1

h[index] = x[i] - x[index]

if h[index] < 0.0:

raise CoeffsException(f'The value h[i] = {h[index]} must be >= 0, from function \"expCoeff\"')

v = y[i]

hp[index] = abs(h[index] \* p[index])

if h[index] == 0.0:

d[i] = v

else:

d[i] = (v - u) / h[index]

u = v

if hp[index] > 0.5:

ph[index] = math.exp(-hp[index])

c = ph[index] \*\* 2

c1 = 1.0 - c

c2 = c1 / hp[index]

c1 \*= hp[index] / h[index]

q[i] = (1.0 - c2 + c) / c1

r[i] = (c2 - 2.0 \* ph[index]) / c1

else:

# using auxiliary function sinh

c = hp[index] \* hp[index]

ph[index] = sinh(c)

w = h[index] / (1.0 + c \* ph[index])

c \*= 0.25

c2 = 1.0 + c \* sinh(c)

q[i] = (0.5 \* c2 \*\* 2 - ph[index]) \* w

r[i] = ph[index] \* w

"""

solution of the tridiagonal system with

diagonal: q[i] + q[i+1] i = 1, n\_less

off-diagonal: r[i] i = 2, n\_less

right hand side: d[i+1] - d[i] i = 1, n\_less

second difference coefficient: dq[i] = d[i+1] - d[i] i = 1, n\_less

"""

d[0] = 0.0

u = 0.0

for i in range(n\_less):

q[i] = q[i] + q[i + 1] - u \* r[i]

dq[i] = d[i + 1] - d[i]

d[i] = dq[i] - u \* d[i - 1]

u = r[i + 1] / q[i]

d[n] = 0.0

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

d[i] = (d[i] - r[i + 1] \* d[i + 1]) / q[i]

return d, dq, h, hp, ph

def expSpline(n, x, y, p, d, h, hp, ph, oX, i):

"""Building exponential spline"""

if i > n - 1:

raise SplineException(f'Wrong parameters, from function \"expSpline\"')

index = i + 1

t = (oX - x[i]) / h[i]

t\_1 = 1.0 - t

if t > 1.0 or t < 0.0:

raise SplineException(f'Wrong parameters, from function \"expSpline\"')

if hp[i] > 0.5:

e = math.exp(-t \* hp[i])

e\_1 = math.exp(-t\_1 \* hp[i])

c = 1.0 - ph[i] \*\* 2

exp = y[index] \* t + y[i] \* t\_1 + (d[index] \* (e\_1 \* (1.0 - e \*\* 2) / c - t) +

d[i] \* (e \* (1.0 - e\_1 \*\* 2) / c - t\_1)) / (p[i] \*\* 2)

else:

e = t \* hp[i]

e\_1 = t\_1 \* hp[i]

c = h[i] \*\* 2 / (1.0 + hp[i] \*\* 2 \* ph[i])

exp = t \* (y[index] + d[index] \* c \* (t \*\* 2 \* sinh(e \*\* 2) - ph[i])) + \

t\_1 \* (y[i] + d[i] \* c \* (t\_1 \*\* 2 \* sinh(e\_1 \*\* 2) - ph[i]))

return exp

def sinh(a):

"""auxiliary function with best approximation for sinh"""

return ((0.27713991169e1 - 5 \* a + 0.19840927713e1 - 3) \* a +

0.83333336379e1 - 2) \* a + 0.16666666666

def get\_spline(x, y, n, step, p=None):

if not p:

p = [0.0] \* (n - 1)

d, dq, h, hp, ph = expCoeff(n, x, y, p)

X = []

Y = []

for i, point in enumerate(x[:-1]):

X.append(x[i])

Y.append(y[i])

for ox in np.arange(point + step, x[i + 1], step):

X.append(ox)

ex = expSpline(n, x, y, p, d, h, hp, ph, ox, i)

Y.append(ex)

return X, Y

def draw(x, y, dot\_x, dot\_y, labels, save\_file="plot.png"):

fig, ax = plt.subplots()

for i in range(len(x)):

ax.plot(x[i], y[i], label=f"{labels[i]}")

ax.scatter(dot\_x, dot\_y, s=20)

ax.legend(loc='best')

ax.grid()

if save\_file:

fig.savefig(save\_file)

print(f"Saved in {save\_file} successfully")

plt.show()

plt.close(fig)

def readData(filename, need\_args):

with open(filename, 'r') as json\_data:

data = json.load(json\_data)

output = []

for item in data:

dict\_ = {}

for arg in need\_args:

if arg not in item:

raise ValueError('No "{0}" in given data'.format(arg))

dict\_[arg] = item[arg]

output.append(dict\_)

return output

if \_\_name\_\_ == "\_\_main\_\_":

parser = argparse.ArgumentParser()

need\_args = ('n', 'x', 'y', 'p', 'step')

init\_tests = readData('tests.json', need\_args)

for idx, test in enumerate(init\_tests):

for k, v in test.items():

print(f'{k}: {v}')

X, Y, labels = [], [], []

cs = CubicSpline(np.array(test['x']), np.array(test['y']))

xs = np.arange(test['x'][0], test['x'][-1], test['step'])

x, y = get\_spline(test['x'], test['y'], test['n'], test['step'], test['p'])

X.append(xs)

Y.append(cs(xs))

labels.append('cubic spline')

X.append(x)

Y.append(y)

labels.append('exp spline')

draw(X, Y, test['x'], test['y'], labels, f'plot{idx}.png')

print()

print("Examples of various tension")

tens\_test = init\_tests[3]

X\_, Y\_, = [], []

cs = CubicSpline(np.array(tens\_test['x']), np.array(tens\_test['y']))

xs = np.arange(tens\_test['x'][0], tens\_test['x'][-1], tens\_test['step'])

p1 = [50, 50, 50, 50, 50, 50]

p2 = [20, 20, 20, 20, 20, 20]

p3 = [0, 0, 0, 0, 0, 0]

p4 = [0.6, 0.6, 0.6, 0.6, 0.6, 0.6]

x1, y1 = get\_spline(tens\_test['x'], tens\_test['y'], tens\_test['n'], tens\_test['step'], p1)

x2, y2 = get\_spline(tens\_test['x'], tens\_test['y'], tens\_test['n'], tens\_test['step'], p2)

x3, y3 = get\_spline(tens\_test['x'], tens\_test['y'], tens\_test['n'], tens\_test['step'], p3)

x4, y4 = get\_spline(tens\_test['x'], tens\_test['y'], tens\_test['n'], tens\_test['step'], p4)

tens\_labels = []

X\_.append(xs)

Y\_.append(cs(xs))

tens\_labels.append('cubic spline')

X\_.append(x1)

Y\_.append(y1)

tens\_labels.append('exp spline with high tens')

X\_.append(x2)

Y\_.append(y2)

tens\_labels.append('exp spline with middle tens')

X\_.append(x3)

Y\_.append(y3)

tens\_labels.append('exp spline with low tens')

X\_.append(x4)

Y\_.append(y4)

tens\_labels.append('exp spline with very low tens')

draw(X\_, Y\_, tens\_test['x'], tens\_test['y'], tens\_labels, f'plot\_tension.png')

def runge(x):

return [1 / (1 + 25 \* x\_i \*\* 2) for x\_i in x]

x\_runge = [-2, -1, 0, 1, 2]

y\_runge = runge(x\_runge)

cs = CubicSpline(np.array(x\_runge), np.array(y\_runge))

xs = np.arange(x\_runge[0], x\_runge[-1], 0.01)

labels\_runge = []

X\_ = []

Y\_ = []

X\_.append(xs)

Y\_.append(cs(xs))

labels\_runge.append('Cubic')

x\_new, y\_new = get\_spline(x\_runge, y\_runge, 5, 0.01, [8, 8, 8, 8])

X\_.append(x\_new)

Y\_.append(y\_new)

labels\_runge.append('exp spline')

draw(X\_, Y\_, x\_runge, y\_runge, labels\_runge, f'runge\_func.png')

**Вывод.**

Данная работа оказалась довольно интересной для выполнения. Хотя кубический сплайн является лучшим способом для интерполяции, нежели интерполяция полиномами (полиномиальная), но из-за возникающих нежелательных точек перегиба также не является совершенным инструментом. Экспоненциальный сплайн позволяет устранить данные точки перегиба с помощью параметра натяжения между двумя точками, что дает существенное преимущество в настройке поведения функции.

**Список литературы.**

[1] Статья по интерполяции экспоненциальными сплайнами -

[http://quantlabs.net/academy/download/free\_quant\_instituitional\_books\_/%5BComputing,%20Spath%5D%20Exponential%20Spline%20Interpolation.pdf](http://quantlabs.net/academy/download/free_quant_instituitional_books_/%5BComputing, Spath%5D Exponential Spline Interpolation.pdf)

[2] Статья на Wiki -

[https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%BB%D0%B0%D0%B9%D0%BD](https://ru.wikipedia.org/wiki/Сплайн)