Федеральное государственное автономное образовательное учреждение высшего образования

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
"ВЫСШАЯ ШКОЛА ЭКОНОМИКИ"

Московский институт электроники и математики имени А. Н. Тихонова
Программа "Прикладная математика"

Нигматуллин Роман Максимович

Лабораторная работа

Приближение функций

3 курс, группа БПМ203

Преподаватель:
Брандышев Петр Евгеньевич

Москва, 2021 г.

Содержание

1 Приближение функции многочленом при помощи МНК (6.1.16)

1.1 Формулировка задачи

Функция $y = f(x)$ задана таблицей значений $y_0, y_1, ..., y_n$ в точках $x_0, x_1, ..., x_n$. Используя метод наименьших квадратов, найти многочлен наилучшего среднеквадратичного приближения оптимальной степени $m = m^*$. За оптимальное решение принять ту степень, начиная с которой величина:

$$\sigma_m = \sqrt{\frac{1}{n-m}\sum_{k=0}^{n}(P_m(x_k) - y_k)^2}$$

стабилизируется или начинает возрастать.

1. Построить многочлены при помощи МНК, вычислить соответствующие значения σ.
2. Построить гистограмму значений, определить лучшее значение.
3. Построить точки из условия и линии значений многочленов на одном графике.

$$x = (-3.2, -2.66, -2.12, -1.58, -1.04, -0.5, 0.04, 0.58, 1.12, 1.66, 2.2)$$
$$y = (-0.173, -0.574, -1.811, -1.849, 0.123, 1.462, 2.399, 1.3, 1.703, -2.045, 2.817)$$

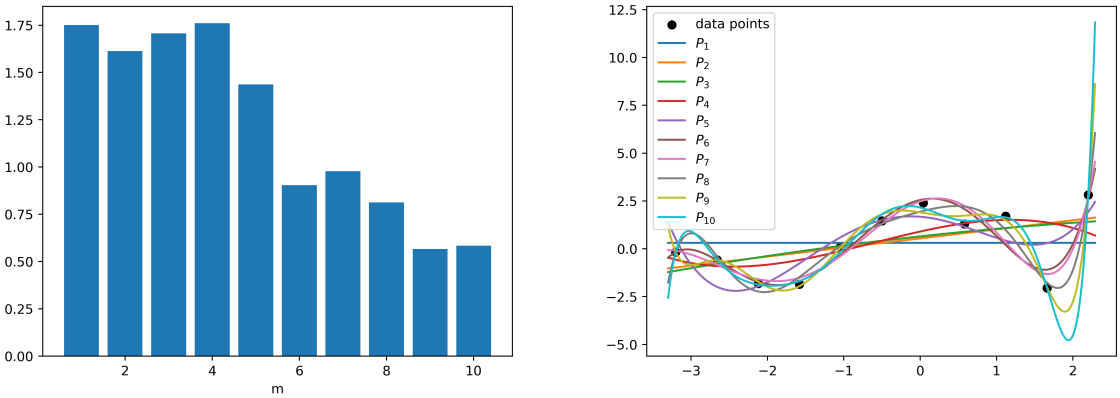
1.2 Код на Python

```
import numpy as np
import numpy.linalg
import matplotlib.pyplot as plt


def lstsq(X, Y, m):
    b = np.zeros(m)
    G = np.zeros((m, m))
    for j in range(m):
        b[j] = sum(y * x ** j for y, x in zip(Y, X))
        for k in range(m):
            G[j, k] = sum(x ** (k + j) for x in X)
    return np.linalg.solve(G, b)


def approximate(x, y, m):
    weights, sigma = {}, {}
    for m in range(1, m):
        X = np.stack([x ** k for k in range(m)]).T
        weight = lstsq(x, y, m)
        sigma[m] = np.sqrt((1 / (n - m)) * ((y - X.dot(weight)) ** 2).sum())
        weights[m] = weight
    plt.bar(sigma.keys(), sigma.values())
    plt.xlabel('m')
    plt.savefig('plots/histogram_lstsq.png', dpi=400)
    plt.close()
    print('Enter optimal M = ', end='')
    optimal = int(input())
    return weights, optimal


def plot(weights, ms, x, y):
    xs = np.arange(min(x) - 0.1, max(x) + 0.1, 0.01)
    plt.scatter(x, y, label='data points', color='black')
    for m in ms:
        w = weights[m]
        X = np.stack([xs ** k for k in range(m)]).T
        Y = X.dot(w)
        plt.plot(xs, Y, label=f'$P_{ {m} }$')
    plt.legend()
    plt.savefig('plots/least_squares.png', dpi=400)
    plt.close()


x = np.array([-3.2, -2.66, -2.12, -1.58,
              -1.04, -0.5, 0.04, 0.58,
              1.12, 1.66, 2.2])
y = np.array([-0.173, -0.574, -1.811, -1.849,
              0.123, 1.462, 2.399, 1.3,
```

```
                1.703, -2.045, 2.817])
n = len(x)

weights, optimal_m = approximate(x, y, 11)
plot(weights, range(1, 11), x, y)
```

1.3 Результат работы программы

По гистограмме видно, что после значения полинома 9 значение σ перестает падать и стабилизируется. Также видно, что итоговый многочлен такой степени лучше всего проходит рядом с точками.



(a) Гистограмма значений σ

(b) Итоговые многочлены

Рис. 1: МНК для подбора многочленов

2 Тратектория материальной точки (6.4.4)

2.1 Формулировка задачи

В таблице приведены результаты наблюдений за движением материальной точки в плоскости (x, y). Известно, что движение осуществляется по кривой, описываемой многочленом $y = kx^m + b$ с заданной степенью m.

1. Используя метод наименьших квадратов, определить коэффициенты k и b.
2. Определить значение $\hat{x}$ координаты x, соответствующее значению $\hat{y}$ координаты y, заданному по условию.

$$m = 3$$

$$\hat{y} = 5$$

$$x = (0.5, 0.75, 1, 1.25, 1.5, 1.75, 2, 2.25, 2.5)$$

$$y = (3.69, 3.9, 4.3, 4.97, 5.96, 7.35, 9.2, 11.57, 14.54)$$

2.2 Код на Python

```
import numpy as np
import matplotlib.pyplot as plt


m = 3
y_target = 5
x = np.array([0.5, 0.75, 1, 1.25, 1.5, 1.75, 2, 2.25, 2.5])
y = np.array([3.69, 3.9, 4.3, 4.97, 5.96, 7.35, 9.2, 11.57, 14.54])
X = np.stack([x ** k for k in [0, m]]).T
weights, _, _, _ = np.linalg.lstsq(X, y, rcond=None)
k, b = weights
print(f'f = {k:.5f} * x ^ {m} + {b:.5f}')

x_net = np.arange(0, 3, 0.01)
X_net = np.stack([x_net ** k for k in [0, m]]).T
y_net = X_net.dot(weights)
plt.plot(x_net, y_net, label='approximated $f$')
plt.scatter(x, y, label='data points')
plt.legend()
plt.savefig('plots/material_point.png', dpi=400)

x_target = ((y_target - b) / k) ** (1 / m)
print(f'x = {x_target}')
```

2.3 Результат работы программы

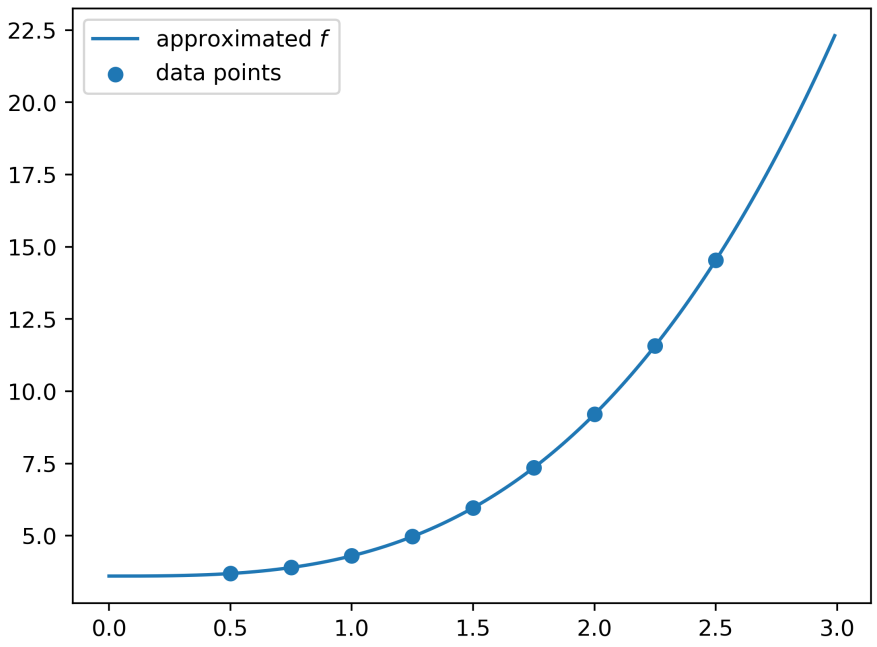


Рис. 2: График кривой

Итоговое значение $\hat{x}$ рассчитано и выведено программой:

```
f = 3.60135 * x ^ 3 + 0.69985
x = 1.0608963123076616
```

3 Кусочно-линейная интерполяция (6.7.8)

3.1 Формулировка задачи

Дана кусочно-гладкая функция $y = f(x)$. Сравнить качество приближения функции кусочно-линейной и глобальной интерполяциями.

1. Вычислить значения функции $y_i = f(x_i)$ в произвольных точках $x_i i = 0, 1, ..., k-1$ отрезка $[a, b]$, по которым будет осуществляться интерполяция функции.

2. Составить программу-функцию, вычисляющую значение интерполяционного многочлена 1-ой степени по точкам (x_i, y_i) и (x_{i+1}, y_{i+1}) в произвольной точке отрезка $[x_i, x_{i+1}]$. С её помощью вычислить приближенные значения функции $f(x)$ при кусочно-линейной интерполяции в $3k$ точках исходного отрезка $[a, b]$.

3. Вычислить приближенные значения функции $f(x)$ в тех же $3k$ точках отрезка при глобальной интерполяции, используя встроенную функцию. На одном чертеже построить графики интерполирующих функций, график исходной функции $f(x)$, а также отметить точки интерполяции.

4. Вычислить практическую величину погрешностей $\delta_j, j = 0, 1, ..., 3k-1$ приближения функции $f(x)$ в $3k$ точках для кусочно-линейной и глобальной интерполяций. На одном чертеже построить графики погрешностей. Сравнить качество приближения.

$$[a, b] = [0, 2]$$
$$f(x) = (x + 1) * |x^2 - 2|$$

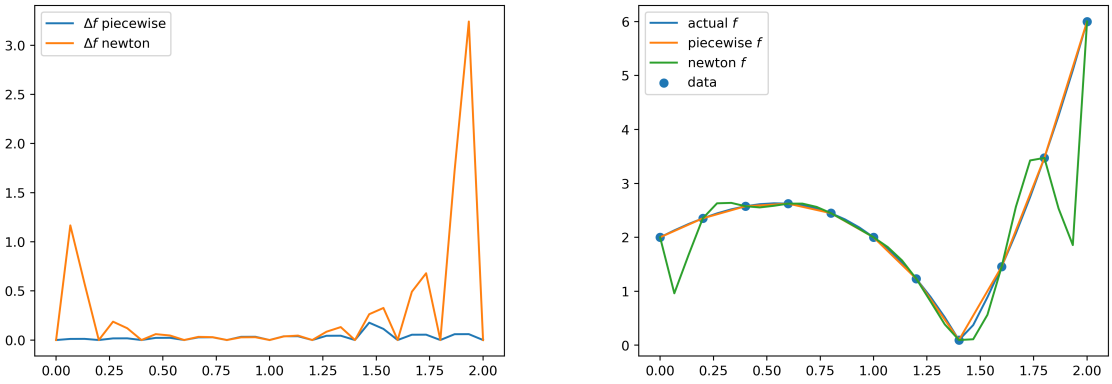
3.2 Код на Python

```
import numpy as np
import matplotlib.pyplot as plt


def linear_interpolate(x0, y0, x1, y1):
    return lambda x: (y0 * (x1 - x) + y1 * (x - x0)) / (x1 - x0)


def poly_newton_coefficient(x, y):
    m = len(x)
    x = np.copy(x)
    a = np.copy(y)
    for k in range(1, m):
        a[k:m] = (a[k:m] - a[k - 1])/(x[k:m] - x[k - 1])
    return a
```

```
def newton_polynomial(x_data, y_data, x):
    a = poly_newton_coefficient(x_data, y_data)
    n = len(x_data) - 1
    p = a[n]
    for k in range(1, n + 1):
        p = a[n - k] + (x - x_data[n - k]) * p
    return p


left, right = 0, 2
k = 10
step = (right - left) / k
x, wide_x = [], []
for i in range(1, k + 1):
    val = i * step
    x.append(val)
    wide_x.append(val - 2 * step / 3)
    wide_x.append(val - step / 3)
    wide_x.append(val)
x, wide_x = np.array([0.] + x), np.array([0.] + wide_x)

y = (x + 1) * np.abs(x ** 2 - 2)
wide_y = (wide_x + 1) * np.abs(wide_x ** 2 - 2)
linear_inter = []
newton_inter = np.array(newton_polynomial(x, y, wide_x))

j = 0
for i in range(k):
    f = linear_interpolate(x[i], y[i], x[i + 1], y[i + 1])
    while j < 3 * k + 1 and wide_x[j] <= x[i + 1]:
        linear_inter.append(f(wide_x[j]))
        j += 1

linear_inter = np.array(linear_inter)

plt.plot(wide_x, wide_y, label='actual $f$')
plt.plot(wide_x, linear_inter, label='piecewise $f$')
plt.plot(wide_x, newton_inter, label='newton $f$')
plt.scatter(x, y, label='data')
plt.legend()
plt.savefig('plots/piecewise_linear_interpolation.png', dpi=400)
plt.close()

plt.plot(wide_x, np.abs(linear_inter - wide_y), label='$\Delta f$ piecewise')
plt.plot(wide_x, np.abs(newton_inter - wide_y), label='$\Delta f$ newton')
plt.legend()
plt.savefig('plots/piecewise_linear_errors.png', dpi=400)
plt.close()
```

3.3 Результат работы программы

Заметно, что кусочно-линейная интерполяция гораздо меньше ошибается на крайних промежутках.



(a) График ошибок интерполяций

(b) Исходная $f(x)$ и интерполяции

Рис. 3: Сравнение глобальной и кусочно-линейной интерполяций

4 Интерполяция сплайнами (6.9.8)

4.1 Формулировка задачи

Дана функция $y = f(x)$. Приблизить $f(x)$ на отрезке $[a, b]$ методом глобальной интерполяции и указанным в индивидуальном варианте сплайном. На одном чертеже построить графики приближающей функции и функции $f(x)$. Сравнить качество приближения при разном количестве узлов интерполяции. Сплайн по варианту - квадратичный.

$$[a, b] = [1, 1.28]$$

$$f(x) = 12 * \sin(e^x)$$

4.2 Код на Python

```
import numpy as np
import scipy.interpolate as intp
import matplotlib.pyplot as plt

left, right = 1, 1.28
fig1, ax1 = plt.subplots(nrows=1, ncols=1)
fig2, ax2 = plt.subplots(nrows=3, ncols=2, figsize=(9, 9))
ks = [3, 4, 6, 8, 10, 12]
for i in range(6):
    k = ks[i]
    x = np.linspace(left, right, k, endpoint=True)
    wide_x = np.linspace(left, right, 3 * k, endpoint=True)
    y = 12 * np.sin(np.exp(x))
```

```
    wide_y = 12 * np.sin(np.exp(wide_x))

    poly = intp.interp1d(x, y, kind='quadratic')

    ax1.plot(wide_x, np.abs(wide_y - poly(wide_x)), label=f'{k}')
    ax2[i // 2][i % 2].plot(wide_x, wide_y, label='f(x)')
    ax2[i // 2][i % 2].plot(wide_x, poly(wide_x), label=f'{k} knots')
    ax2[i // 2][i % 2].legend()


fig1.legend()
fig1.suptitle('Error of splines by knot quantity')
fig1.savefig('plots/spline_by_knot_error.png', dpi=400)
plt.close(fig1)


fig2.suptitle('Splines by knot quantity')
fig2.tight_layout()
fig2.savefig('plots/spline_by_knot.png', dpi=400)
plt.close(fig2)
```

4.3 Результат работы программы

Заметно, что уже при шести узлах интерполяции уровень ошибки очень маленький и визуально не заметен на графике из-за размаха значений функции.

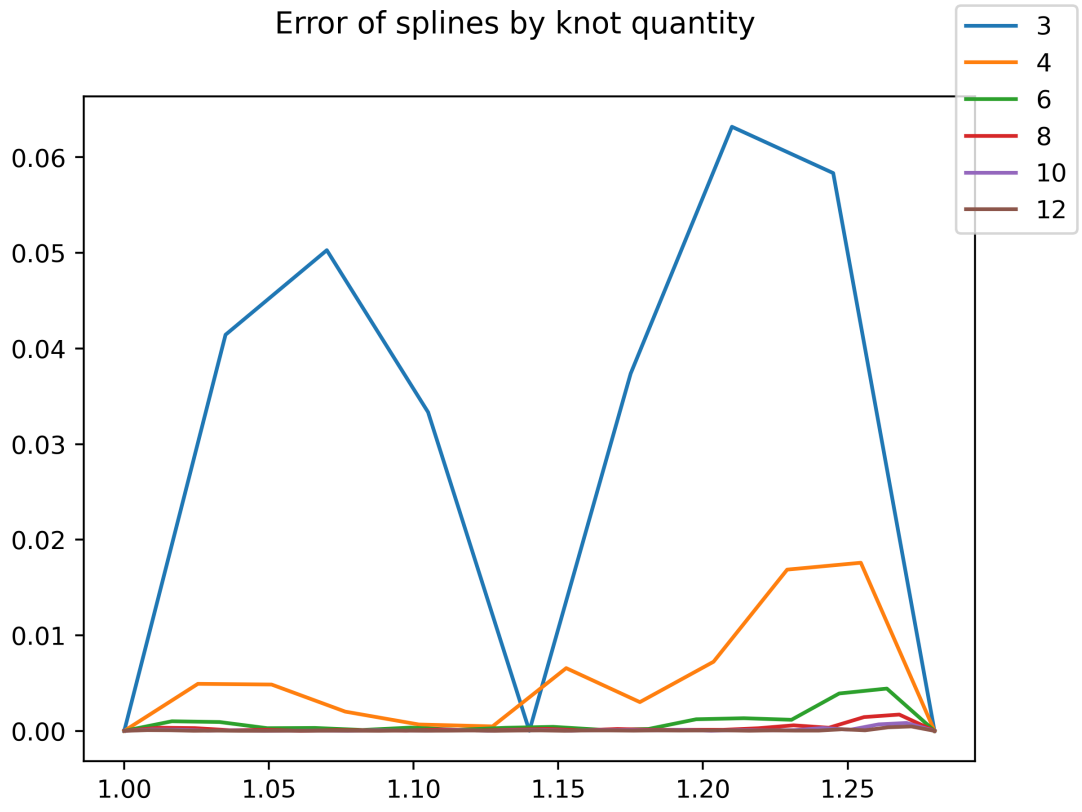


Рис. 4: График ошибок интерполяций

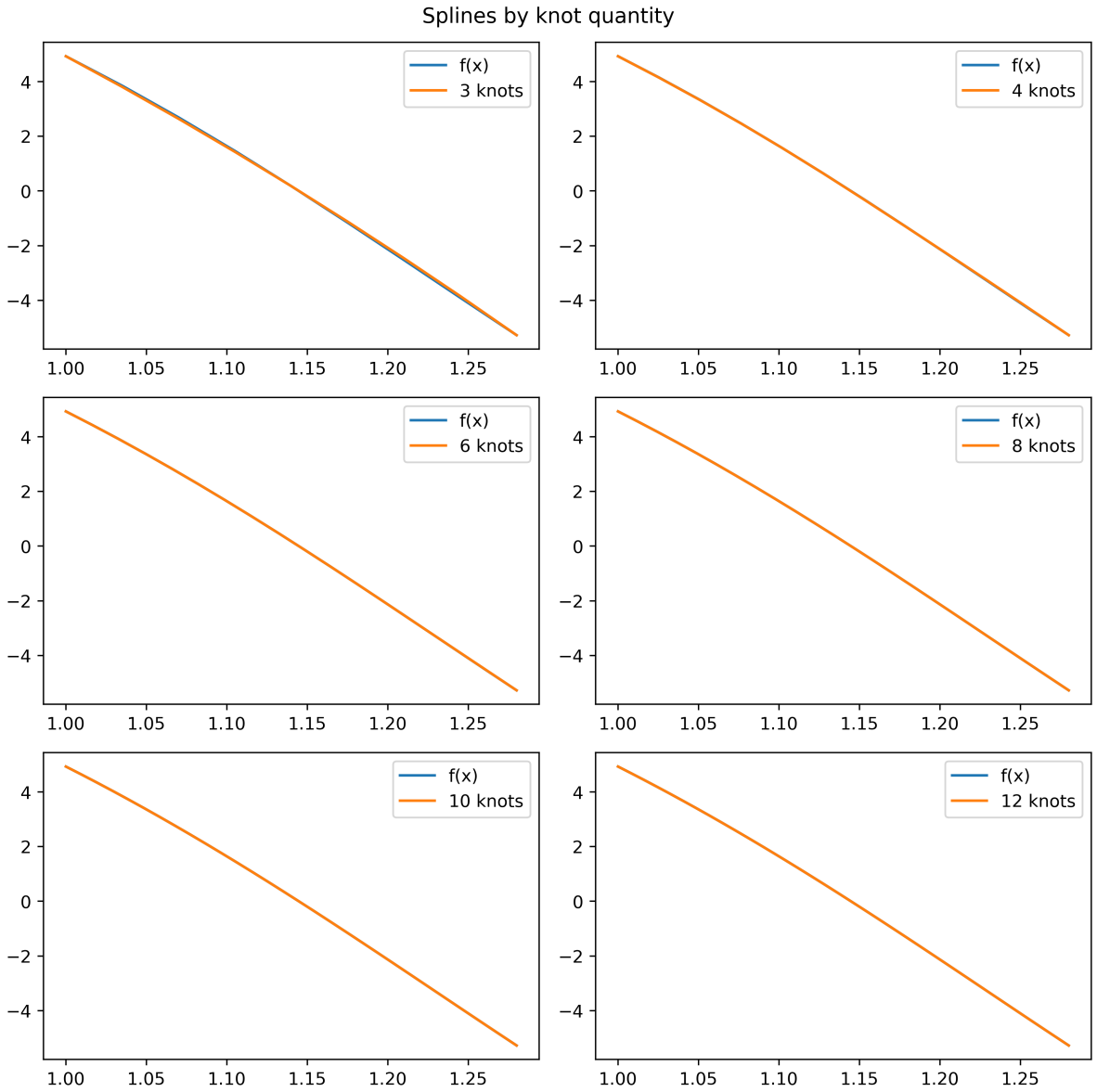


Рис. 5: Графики интерполяции сплайнами