Университет ИТМО МФ КТиУ, Ф ПИиКТ

**Лабораторная работа №4**

**Дисциплина «Вычислительная математика»**

**Аппроксимация функции методом наименьших квадратов**

**Выполнил** Галлямов Камиль Рустемович

**Преподаватель:**

Машина Екатерина Алексеевна

г. Санкт-Петербург 2024 г.

# Вычислительная реализация задачи

Линейная аппроксимация:

y =

n = 11

x in [0;2]

h = 0.2

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| xi | -2 | -1.8 | -1.6 | -1.4 | -1.2 | -1 | -0.8 | -0.6 | -0.4 | -0.2 | 0 |
| yi | -0.42 | -0.53 | -0.67 | -0.82 | -0.95 | -1.0 | -0.94 | -0.77 | -0.53 | -0.27 | 0.0 |

φ(x) = a + b \* x

Вычисляем суммы: sx = -11, sxx = 15.4, sy = -6.9, sxy = 7.63

φ(x) = -0.46 + 0.17 \* x

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| xi | -2 | -1.8 | -1.6 | -1.4 | -1.2 | -1 | -0.8 | -0.6 | -0.4 | -0.2 | 0 |
| yi | -0.42 | -0.53 | -0.67 | -0.82 | -0.95 | -1.0 | -0.94 | -0.77 | -0.53 | -0.27 | 0.0 |
| φ (xi) | -0.8 | -0.77 | -0.73 | -0.7 | -0.66 | -0.63 | -0.6 | -0.56 | -0.53 | -0.49 | -0.46 |
| (φ (xi)- yi)^2 | 0.144 | 0.056 | 0.004 | 0.015 | 0.082 | 0.137 | 0.118 | 0.043 | 0 | 0.05 | 0.212 |

σ = = **0.278**

Квадратичная аппроксимация:

y =

n = 11

x in [0;2]

h = 0.2

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| xi | -2 | -1.8 | -1.6 | -1.4 | -1.2 | -1 | -0.8 | -0.6 | -0.4 | -0.2 | 0 |
| yi | -0.42 | -0.53 | -0.67 | -0.82 | -0.95 | -1.0 | -0.94 | -0.77 | -0.53 | -0.27 | 0.0 |

φ(x) = a + b \* x + c \* x^2

Вычисляем суммы: sx = -11, sxx = 15.4, sxxx = -24.2, sxxxx = 40.53,

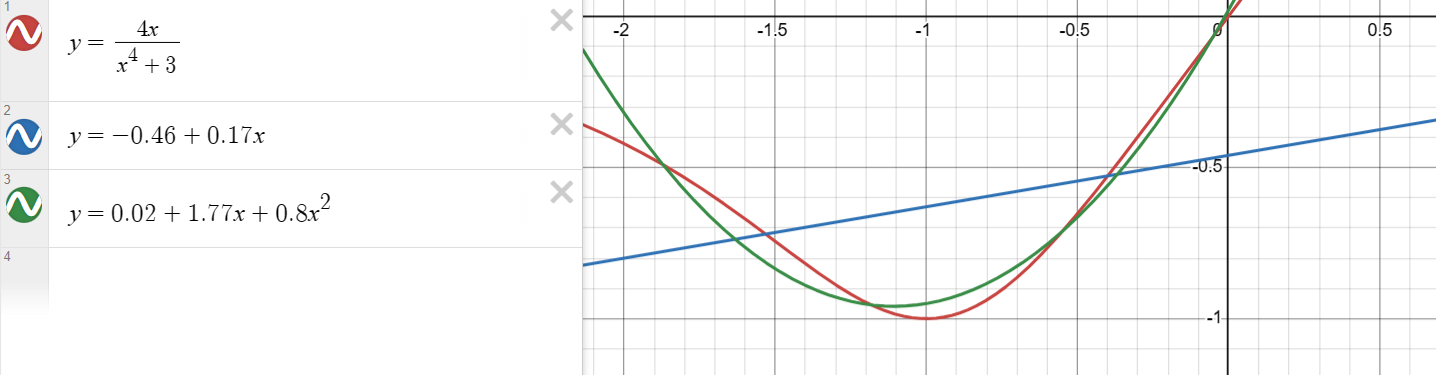
sy = -6.9, sxy = 7.63, sxxy = -10.06

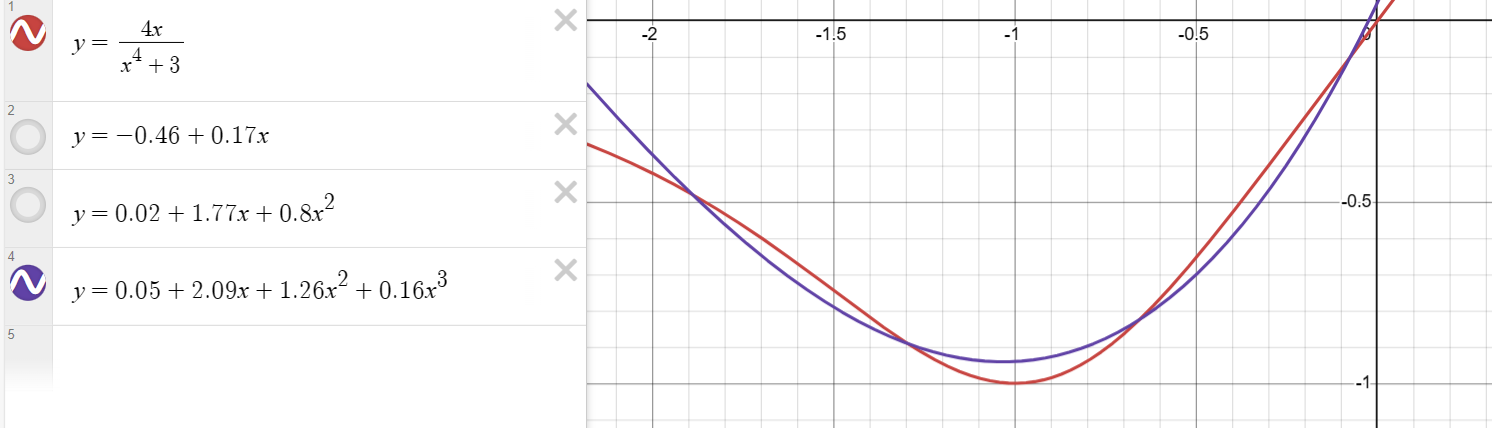
φ(x) = 0.02 + 1.77 \* x + 0.8 \* x ^ 2

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| xi | -2 | -1.8 | -1.6 | -1.4 | -1.2 | -1 | -0.8 | -0.6 | -0.4 | -0.2 | 0 |
| yi | -0.42 | -0.53 | -0.67 | -0.82 | -0.95 | -1.0 | -0.94 | -0.77 | -0.53 | -0.27 | 0.0 |
| φ (xi) | -0.32 | -0.57 | -0.76 | -0.89 | -0.95 | -0.95 | -0.88 | -0.75 | -0.56 | -0.3 | 0.02 |
| (φ (xi)- yi)^2 | 0.01 | 0.002 | 0.009 | 0.005 | 0.0 | 0.003 | 0.003 | 0.0 | 0.001 | 0.001 | 0.0 |

σ = = **0.056**

У квадратичной аппроксимации среднеквадратичное отклонение меньше, поэтому это приближение наилучшее.





Для кубической аппроксимации: σ = = **0.045**

# Программная реализация задачи

**import** **inspect**

**from** **math** **import** sqrt, exp, log

**import** **matplotlib.pyplot** **as** **plt**

**def** **calc\_det2**(A):

**return** A[**0**][**0**] \* A[**1**][**1**] - A[**0**][**1**] \* A[**1**][**0**]

**def** **solve2**(A, B):

n = **2**

det = calc\_det2(A)

det1 = calc\_det2([[B[r], A[r][**1**]] **for** r **in** range(n)])

det2 = calc\_det2([[A[r][**0**], B[r]] **for** r **in** range(n)])

x1 = det1 / det

x2 = det2 / det

**return** x1, x2

**def** **calc\_det3**(A):

pos = A[**0**][**0**] \* A[**1**][**1**] \* A[**2**][**2**] + \

A[**0**][**1**] \* A[**1**][**2**] \* A[**2**][**0**] + \

A[**0**][**2**] \* A[**1**][**0**] \* A[**2**][**1**]

neg = A[**0**][**2**] \* A[**1**][**1**] \* A[**2**][**0**] + \

A[**0**][**1**] \* A[**1**][**0**] \* A[**2**][**2**] + \

A[**0**][**0**] \* A[**1**][**2**] \* A[**2**][**1**]

**return** pos - neg

**def** **solve3**(A, B):

n = **3**

det = calc\_det3(A)

det1 = calc\_det3([[B[r], A[r][**1**], A[r][**2**]] **for** r **in** range(n)])

det2 = calc\_det3([[A[r][**0**], B[r], A[r][**2**]] **for** r **in** range(n)])

det3 = calc\_det3([[A[r][**0**], A[r][**1**], B[r]] **for** r **in** range(n)])

x1 = det1 / det

x2 = det2 / det

x3 = det3 / det

**return** x1, x2, x3

**def** **calc\_det4**(A):

n = **4**

sign = **1**

r = **0**

res = **0**

**for** c **in** range(n):

A\_ = [[A[r\_][c\_] **for** c\_ **in** range(n) **if** c\_ != c]

**for** r\_ **in** range(n) **if** r\_ != r]

res += sign \* A[r][c] \* calc\_det3(A\_)

sign \*= -**1**

**return** res

**def** **solve4**(A, B):

n = **4**

det = calc\_det4(A)

det1 = calc\_det4([[B[r], A[r][**1**], A[r][**2**], A[r][**3**]] **for** r **in** range(n)])

det2 = calc\_det4([[A[r][**0**], B[r], A[r][**2**], A[r][**3**]] **for** r **in** range(n)])

det3 = calc\_det4([[A[r][**0**], A[r][**1**], B[r], A[r][**3**]] **for** r **in** range(n)])

det4 = calc\_det4([[A[r][**0**], A[r][**1**], A[r][**2**], B[r]] **for** r **in** range(n)])

x1 = det1 / det

x2 = det2 / det

x3 = det3 / det

x4 = det4 / det

**return** x1, x2, x3, x4

**def** **linear\_approximation**(xs, ys, n):

sx = sum(xs)

sxx = sum(x \*\* **2** **for** x **in** xs)

sy = sum(ys)

sxy = sum(x \* y **for** x, y **in** zip(xs, ys))

a, b = solve2(

[

[n, sx],

[sx, sxx]

],

[sy, sxy])

**return** **lambda** x: a + b \* x, a, b

**def** **quadratic\_approximation**(xs, ys, n):

sx = sum(xs)

sxx = sum(x \*\* **2** **for** x **in** xs)

sxxx = sum(x \*\* **3** **for** x **in** xs)

sxxxx = sum(x \*\* **4** **for** x **in** xs)

sy = sum(ys)

sxy = sum(x \* y **for** x, y **in** zip(xs, ys))

sxxy = sum(x \* x \* y **for** x, y **in** zip(xs, ys))

a, b, c = solve3(

[

[n, sx, sxx],

[sx, sxx, sxxx],

[sxx, sxxx, sxxxx]

],

[sy, sxy, sxxy]

)

**return** **lambda** x: a + b \* x + c \* x \*\* **2**, a, b, c

**def** **cubic\_approximation**(xs, ys, n):

sx = sum(xs)

sxx = sum(x \*\* **2** **for** x **in** xs)

sxxx = sum(x \*\* **3** **for** x **in** xs)

sxxxx = sum(x \*\* **4** **for** x **in** xs)

sxxxxx = sum(x \*\* **5** **for** x **in** xs)

sxxxxxx = sum(x \*\* **6** **for** x **in** xs)

sy = sum(ys)

sxy = sum(x \* y **for** x, y **in** zip(xs, ys))

sxxy = sum(x \* x \* y **for** x, y **in** zip(xs, ys))

sxxxy = sum(x \* x \* x \* y **for** x, y **in** zip(xs, ys))

a, b, c, d = solve4(

[

[n, sx, sxx, sxxx],

[sx, sxx, sxxx, sxxxx],

[sxx, sxxx, sxxxx, sxxxxx],

[sxxx, sxxxx, sxxxxx, sxxxxxx]

],

[sy, sxy, sxxy, sxxxy]

)

**return** **lambda** x: a + b \* x + c \* x \*\* **2** + d \* x \*\* **3**, \

a, b, c, d

**def** **exponential\_approximation**(xs, ys, n):

ys\_ = list(map(log, ys))

\_, a\_, b\_ = linear\_approximation(xs, ys\_, n)

a = exp(a\_)

b = b\_

**return** **lambda** x: a \* exp(b \* x), a, b

**def** **logarithmic\_approximation**(xs, ys, n):

xs\_ = list(map(log, xs))

\_, a\_, b\_ = linear\_approximation(xs\_, ys, n)

a = a\_

b = b\_

**return** **lambda** x: a + b \* log(x), a, b

**def** **power\_approximation**(xs, ys, n):

xs\_ = list(map(log, xs))

ys\_ = list(map(log, ys))

\_, a\_, b\_ = linear\_approximation(xs\_, ys\_, n)

a = exp(a\_)

b = b\_

**return** **lambda** x: a \* x \*\* b, a, b

**def** **calc\_measure\_of\_deviation**(xs, ys, fi, n):

epss = [fi(x) - y **for** x, y **in** zip(xs, ys)]

**return** sum((eps \*\* **2** **for** eps **in** epss))

**def** **calc\_standard\_deviation**(xs, ys, fi, n):

**return** sqrt(sum(((fi(x) - y) \*\* **2** **for** x, y **in** zip(xs, ys))) / n)

**def** **calc\_pearson\_correlation\_coefficient**(xs, ys, n):

av\_x = sum(xs) / n

av\_y = sum(ys) / n

**return** sum((x - av\_x) \* (y - av\_y) **for** x, y **in** zip(xs, ys)) / \

sqrt(sum((x - av\_x) \*\* **2** **for** x **in** xs) \*

sum((y - av\_y) \*\* **2** **for** y **in** ys))

**def** **calc\_coefficient\_of\_determination**(xs, ys, fi, n):

av\_fi = sum(fi(x) **for** x **in** xs) / n

**return** **1** - sum((y - fi(x)) \*\* **2** **for** x, y **in** zip(xs, ys)) / sum((y - av\_fi) \*\* **2** **for** y **in** ys)

**def** **get\_str\_content\_of\_func**(func):

str\_func = inspect.getsourcelines(func)[**0**][**0**]

**return** str\_func.split('lambda x: ')[-**1**].split(',')[**0**].strip()

**def** **draw\_plot**(a, b, func, dx=**0.1**):

xs, ys = [], []

a -= dx

b += dx

x = a

**while** x <= b:

xs.append(x)

ys.append(func(x))

x += dx

plt.plot(xs, ys, 'g')

**def** **main**(xs, ys, n):

**if** all(map(**lambda** x: x > **0**, xs)) **and** all(map(**lambda** x: x > **0**, ys)):

approximation\_funcs = [

linear\_approximation,

power\_approximation,

exponential\_approximation,

logarithmic\_approximation,

quadratic\_approximation,

cubic\_approximation

]

**else**:

approximation\_funcs = [

linear\_approximation,

quadratic\_approximation,

cubic\_approximation

]

best\_sigma = float('inf')

best\_apprxmt\_f = None

**for** apprxmt\_f **in** approximation\_funcs:

**print**(apprxmt\_f.\_\_name\_\_, ": ")

fi, \*coeffs = apprxmt\_f(xs, ys, n)

s = calc\_measure\_of\_deviation(xs, ys, fi, n)

sigma = calc\_standard\_deviation(xs, ys, fi, n)

**if** sigma < best\_sigma:

best\_sigma = sigma

best\_apprxmt\_f = apprxmt\_f

r2 = calc\_coefficient\_of\_determination(xs, ys, fi, n)

**print**('fi(x) =', get\_str\_content\_of\_func(fi))

tmp = '(a, b, c)' **if** len(coeffs) == **3** **else** '(a, b)'

**print**(f'coeffs {tmp}:', list(map(**lambda** cf: round(cf, **4**), coeffs)))

**print**(f'S = {s:.5f}, σ = {sigma:.5f}, R2 = {r2:.5f}')

**if** apprxmt\_f **is** linear\_approximation:

**print**('r =', calc\_pearson\_correlation\_coefficient(xs, ys, n))

plt.title(apprxmt\_f.\_\_name\_\_)

draw\_plot(xs[**0**], xs[-**1**], fi)

**for** i **in** range(n):

plt.scatter(xs[i], ys[i], c='r')

plt.xlabel("X")

plt.ylabel("Y")

plt.show()

**print**('-' \* **50**)

**print**(f'best\_func: {best\_apprxmt\_f.\_\_name\_\_}')

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

case = **2**

**if** case == **1**:

xs = [**1.2**, **2.9**, **4.1**, **5.5**, **6.7**, **7.8**, **9.2**, **10.3**]

ys = [**7.4**, **9.5**, **11.1**, **12.9**, **14.6**, **17.3**, **18.2**, **20.7**]

**elif** case == **2**:

xs = [**1.1**, **2.3**, **3.7**, **4.5**, **5.4**, **6.8**, **7.5**]

ys = [**3.5**, **4.1**, **5.2**, **6.9**, **8.3**, **14.8**, **21.2**]

**elif** case == **3**:

xs = [**1.1**, **2.3**, **3.7**, **4.5**, **5.4**, **6.8**, **7.5**]

ys = [**2.73**, **5.12**, **7.74**, **8.91**, **10.59**, **12.75**, **13.43**]

**else**:

h = **0.2**

x0 = -**2**

n = **11**

xs = [round(x0 + i \* h, **2**) **for** i **in** range(n)]

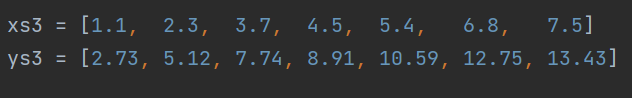
f = **lambda** x: **4** \* x / (x \*\* **4** + **3**)

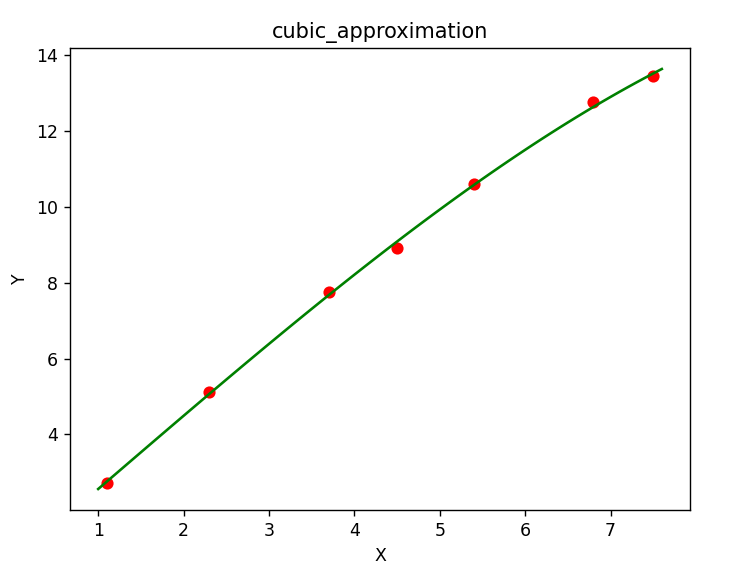
ys = [round(f(x), **2**) **for** x **in** xs]

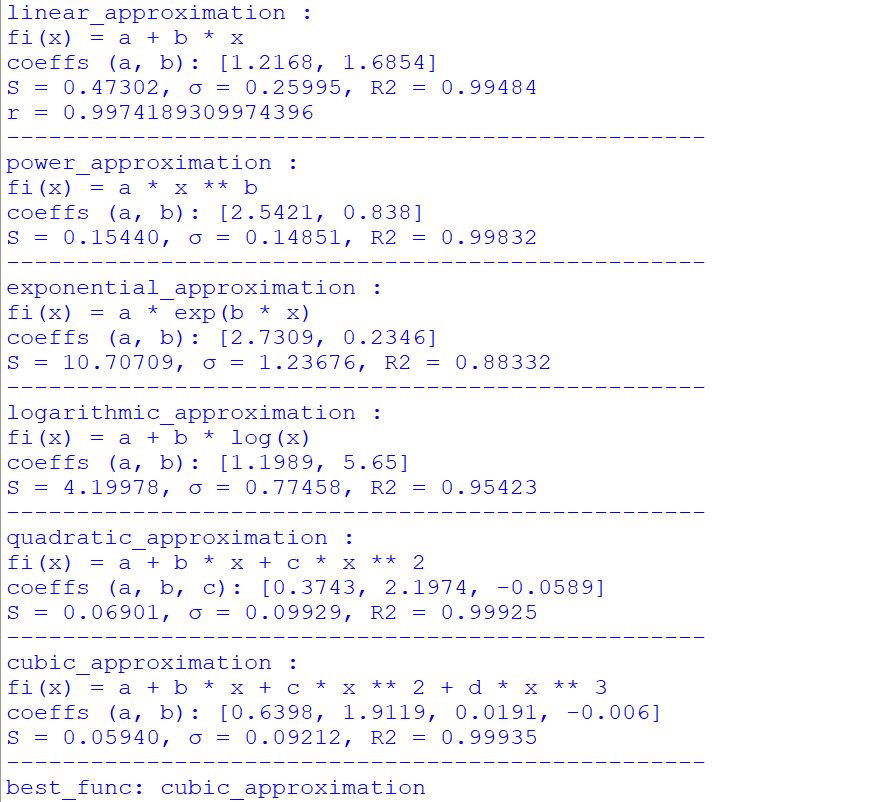
n = len(xs)

main(xs, ys, n)

# Тестовые данные







# Вывод

В ходе лабораторной работы я познакомился с аппроксимацией функции методом наименьших квадратов.

Метод наименьших квадратов – хороший метод, определяются параметры, при которых значения аппроксимирующей функции приблизительно совпадали бы со значениями исходной функции. В качестве аппроксимирующих функций обычно берут многочлены. Чем выше степень многочлена, тем точнее. Можно использовать экспоненциальные, логарифмические, степенные функции (сводя их преобразованиями к линейному аппроксимированию). Аппроксимирующая функций проходит в ближайшей близости от точек из заданного массива данных.