Университет ИТМО
МФ КТиУ, Ф ПИиКТ

Лабораторная работа №4
Дисциплина «Вычислительная математика»

Аппроксимация функции методом наименьших квадратов

Выполнил
Аскаров Эмиль Рамилевич
Преподаватель:
Машина Екатерина Алексеевна

г. Санкт-Петербург
2024 г.

Вычислительная реализация задачи

Линейная аппроксимация:

y = $\frac{12x}{x^4+1}$

n = 11
x in [0;2]
h = 0.2

i	1	2	3	4	5	6	7	8	9	10	11
x_i	-2	-1.8	-1.6	-1.4	-1.2	-1	-0.8	-0.6	-0.4	-0.2	0
y_i	-1.412	1.879	-2.542	-3.47	-4.685	-6.0	-6.81	-6.374	-4.68	-2.396	0.0

φ(x) = a + b * x

Вычисляем суммы: sx = -11, sxx = 15.4, sy = -40.248, sxy = 38.377

$$\begin{cases} n*a+sx*b=sy \\ sx*a+sxx*b=sxy \end{cases} \quad \begin{cases} 11*a-11*b=-40.248 \\ -11*a+15.4*b=38.377 \end{cases}$$

$$\begin{cases} a=-4.084 \\ b=-0.425 \end{cases}$$

φ(x) = -4.084 – 0.425 * x

x_i	-2.0	-1.8	-1.6	-1.4	-1.2	-1.0	-0.8	-0.6	-0.4	-0.2	0.0
y_i	-1.412	-1.879	-2.542	-3.47	-4.685	-6.0	-6.81	-6.374	-4.68	-2.396	0.0
φ (x_i)	7.743	6.926	6.109	5.293	4.476	3.659	2.842	2.025	1.209	0.392	-0.425
delta	83.814	77.528	74.84	76.79	83.924	93.296	93.161	70.543	34.68	7.773	0.181

$\sigma = \sqrt{\frac{\sum(\varphi\ (xi)-yi)^2}{n}} = 2.602$

Квадратичная аппроксимация:

y = $\frac{12x}{x^4+1}$

n = 11
x in [0;2]
h = 0.2

i	1	2	3	4	5	6	7	8	9	10	11
x_i	-2	-1.8	-1.6	-1.4	-1.2	-1	-0.8	-0.6	-0.4	-0.2	0
y_i	-1.412	1.879	-2.542	-3.47	-4.685	-6.0	-6.81	-6.374	-4.68	-2.396	0.0

φ(x) = a + b * x + c * x^2
Вычисляем суммы: sx = -11, sxx = 15.4, sxxx = -24.2, sxxxx = 40.533,
sy = -40.248, sxy = 38.377, sxxy = -45.289

$$\begin{cases} n*a+sx*b+sxx*c=sy \\ sx*a+sxx*b+sxxx*c=sxy \\ sxx*a+sxxx*b+sxxxx*c=sxxy \end{cases} \begin{cases} 11*a-11*b+15.4*c=-40.248 \\ -11*a+15.4*b-24.2*c=38.377 \\ 15.4*-24.2*b+40.533*c=-45.289 \end{cases}$$

$$\begin{cases} 11*a=-6.9+11*b-15.4*c \\ 4.4*b-8.8*c=0.73 \\ 15.4*a-24.2*b+40.53*c=-10.06 \end{cases} \begin{cases} a=-0.63+b-1.4*c \\ 4.4*b-8.8*c=0.73 \\ 15.4*a-24.2*b+40.53*c=-10.06 \end{cases}$$

$$\begin{cases} a=-0.63+b-1.4*c \\ 4.4*b-8.8*c=0.73 \\ -9.7+15.4*b-21.56*c-24.2*b+40.53*c=-10.06 \end{cases}$$

$$\begin{cases} a=-0.63+b-1.4*c \\ 4.4*b-8.8*c=0.73 \\ -8.8*b+18.97*c=-0.36 \end{cases} \begin{cases} a=-0.63+b-1.4*c \\ 4.4*b=0.73+8.8*c \\ -1.46-17.6*c+18.97*c=-0.36 \end{cases}$$

$$\begin{cases} a=-0.63+b-1.4*c \\ 4.4*b=0.73+8.8*c \\ 1.37*c=1.1 \end{cases} \begin{cases} a=-0.63+b-1.4*c \\ 4.4*b=7.77 \\ 1.37*c=1.1 \end{cases} \begin{cases} a=-0.63+1.77-1.4*0.8 \\ b=1.77 \\ 1.37*c=1.1 \end{cases}$$

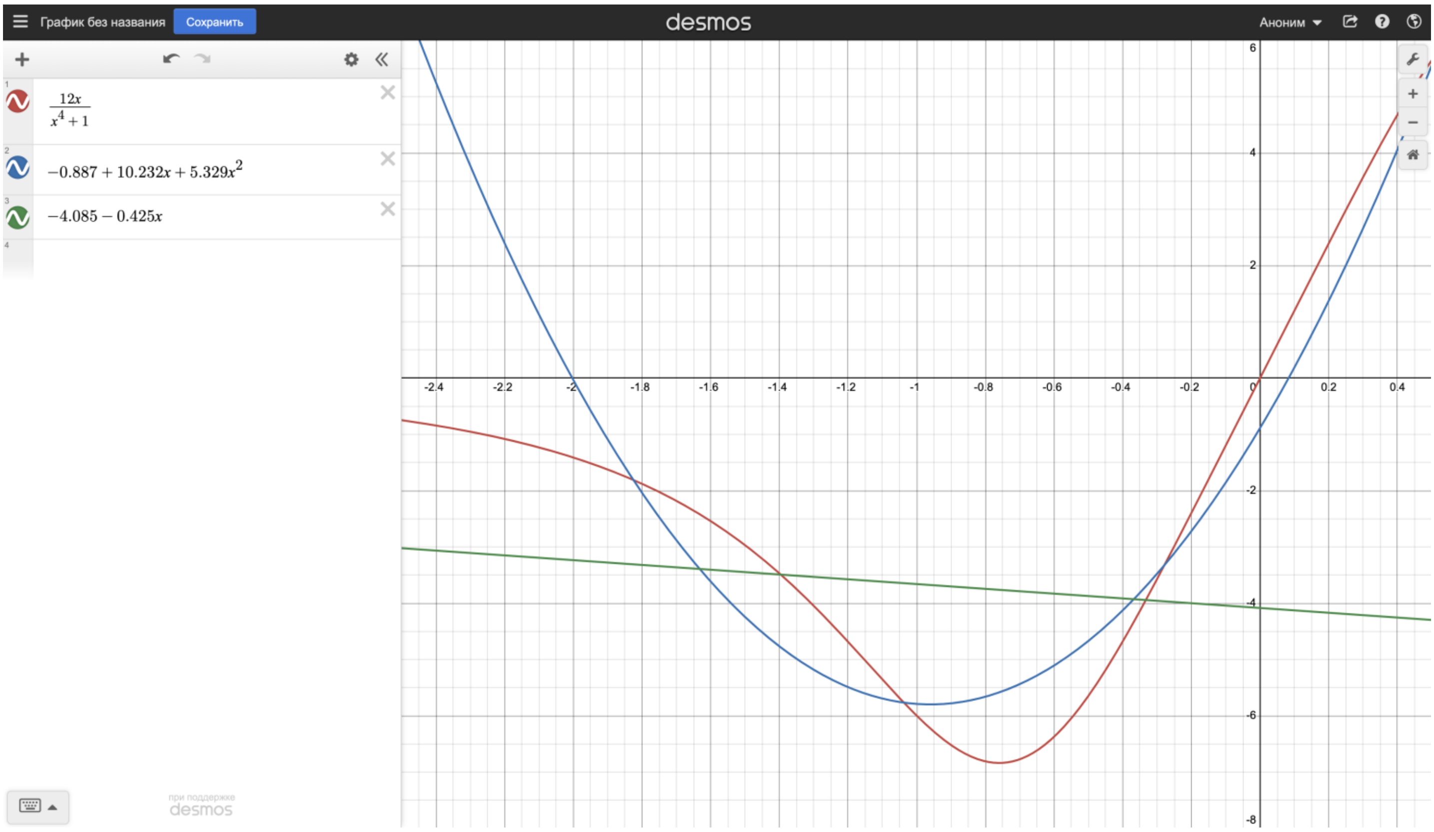
$$\begin{cases} a=-0.887 \\ b=10.232 \\ c=5.329 \end{cases}$$

φ(x) = 5.329 + 10.232 * x - 0.887 * x ^ 2

x_i	-2.0	-1.8	-1.6	-1.4	-1.2	-1.0	-0.8	-0.6	-0.4	-0.2	0.0
y_i	-1.412	-1.879	-2.542	-3.47	-4.685	-6.0	-6.81	-6.374	-4.68	-2.396	0.0
phi_i	-0.035	-2.039	-3.616	-4.767	-5.492	-5.79	-5.662	-5.108	-4.127	-2.72	-0.887
delta	1.896	0.026	1.153	1.682	0.651	0.044	1.318	1.603	0.306	0.105	0.787

$\sigma = \sqrt{\frac{\sum(\varphi\ (xi) - yi)^2}{n}} = \mathbf{0.933}$

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



Программная реализация задачи

```
import inspect
from math import sqrt, exp, log

import matplotlib.pyplot as plt


def add_col(m, col):
    for k, row in enumerate(m):
        row.append(col[k])
    return m


def remove_last_col(m):
    for k, row in enumerate(m):
        row.pop()
    return m


def plus(src, ind, m):
    for i in range(src + 1, len(m)):
        _plus(src, i, m, -m[i][ind] / m[src][ind])


def _plus(src, dest, m, mul: float = 1):
    for i in range(len(m[0])):
        m[dest][i] += m[src][i] * mul


def swap(src, dest, m):
    m[src], m[dest] = m[dest], m[src]


def rang(m):
    return sum(any(row) for row in m)
```

```python
def determinant(m, k):
    p = 1
    for i in range(len(m)):
        p *= m[i][i]
    return (-1) ** k * p


def solve(m):
    k = 0
    row = 0
    col = 0
    n = len(m)
    while col < n:
        for j in range(row, n):
            if m[j][col]:
                swap(j, row, m)
                k += row != j
                plus(row, col, m)
                row += 1
                break
        col += 1
    xs = []
    for i in range(len(m)):
        x = m[len(m) - i - 1][-1]
        for j in range(1, i + 2):
            if j == i + 1:
                x /= m[len(m) - i - 1][-j - 1]
            else:
                x -= m[len(m) - i - 1][-j - 1] * xs[j - 1]
        xs.append(x)
    return xs[::-1]


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))
    ext_matrix = add_col(
        [
            [n, sx],
            [sx, sxx]
        ],
        [sy, sxy]
    )

    a, b = solve(ext_matrix)
    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))
    ext_matrix = add_col(
        [
            [n, sx, sxx],
            [sx, sxx, sxxx],
            [sxx, sxxx, sxxxx]
        ],
        [sy, sxy, sxxy]
    )

    a, b, c = solve(ext_matrix)
```

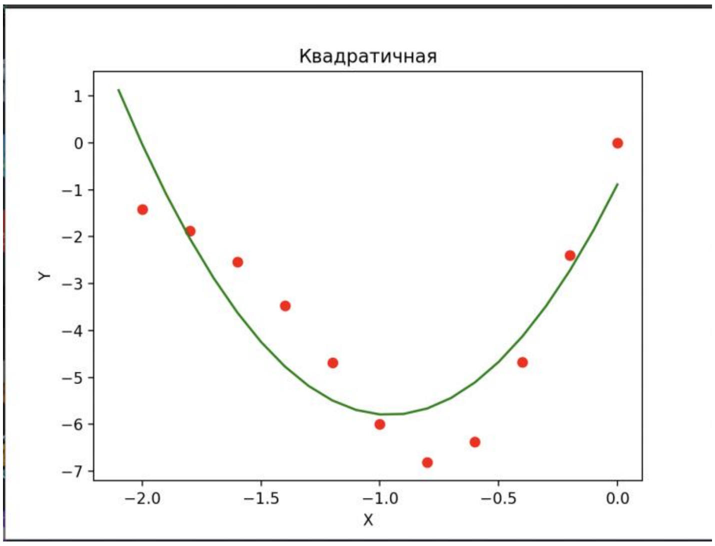
```python
    return lambda x: a + b * x + c * x ** 2, a, b, c


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))
    ext_matrix = add_col(
        [
            [n, sx, sxx, sxxx],
            [sx, sxx, sxxx, sxxxx],
            [sxx, sxxx, sxxxx, sxxxxx],
            [sxxx, sxxxx, sxxxxx, sxxxxxx]
        ],
        [sy, sxy, sxxy, sxxxy]
    )

    a, b, c, d = solve(ext_matrix)
    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_deviation(xs, ys, fi, n):
    return sum((eps ** 2 for eps in [fi(x) - y for x, y in zip(xs, ys)]))


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))
```

```python
def calc_coefficient_of_determination(xs, ys, fi, n):
    return 1 - sum((y - fi(x)) ** 2 for x, y in zip(xs, ys)) / (sum(fi(x) ** 2 for x in
xs) - sum(fi(x) for x in xs) ** 2 / n)


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 read_number(s: str):
    while True:
        try:
            return float(input(s))
        except Exception:
            continue


if __name__ == '__main__':
    read_number("Введите количество точек: ")
    xs = list(map(float, input().split()))
    ys = list(map(float, input().split()))
    if len(xs) != len(ys):
        print("Некорректные данные")

    n = len(xs)

    names = {
        linear_approximation: "Линейная",
        power_approximation: "Степенная",
        exponential_approximation: "Экспоненциальная",
        logarithmic_approximation: "Логарифмическая",
        quadratic_approximation: "Квадратичная",
        cubic_approximation: "Кубическая"
    }

    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(names[apprxmt_f], ": ")
        fi, *coeffs = apprxmt_f(xs, ys, n)
```

```python
        s = calc_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))
        print(f'coeffs:', 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(names[apprxmt_f])

        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: {names[best_apprxmt_f]}')
```

Тестовые данные

```
→  lab4 git:(main) × python3 main.py
Введите количество точек: 11
-2.0    -1.8    -1.6    -1.4    -1.2    -1.0    -0.8    -0.6    -0.4    -0.2    0.0
-1.412  -1.879  -2.542  -3.47   -4.685  -6.0    -6.81   -6.374  -4.68   -2.396  0.0
Линейная :
fi(x) = a + b * x
coeffs: [-4.0841, -0.4252]
S = 48.56409, σ = 2.10117, R2 = -60.04082
r = -0.12695833690235026
--------------------------------------------------
Квадратичная :
fi(x) = a + b * x + c * x ** 2
coeffs: [-0.8864, 10.2339, 5.3296]
S = 9.57058, σ = 0.93277, R2 = 0.75947
```



```
Линейная :
fi(x) = a + b * x
coeffs: [-4.0841, -0.4252]
S = 48.56409, σ = 2.10117, R2 = -60.04082
r = -0.12695833690235026
--------------------------------------------------------
Квадратичная :
fi(x) = a + b * x + c * x ** 2
coeffs: [-0.8864, 10.2339, 5.3296]
S = 9.57058, σ = 0.93277, R2 = 0.75947
--------------------------------------------------------
Кубическая :
fi(x) = a + b * x + c * x ** 2 + d * x ** 3
coeffs: [0.4356, 20.7361, 19.1, 4.5901]
S = 1.24048, σ = 0.33581, R2 = 0.97422
--------------------------------------------------------
best_func: Кубическая
```

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.