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 = $\frac{4x}{x^4+3}$

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	-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

$$\begin{cases} n*a + sx*b = sy \\ sx*a + sxx*b = sxy \end{cases} \begin{cases} 11*a - 11*b = -6.9 \\ -11*a + 15.4*b = 7.63 \end{cases} \begin{cases} 11*a - 11*b = -6.9 \\ 4.4*b = 0.73 \end{cases}$$

$$\begin{cases} b = 0.73/\,4.4 = 0.17 \\ 11a = -6.9 + 11*0.17 = -5.03 \end{cases} \begin{cases} b = 0.17 \\ a = -0.46 \end{cases}$$

φ(x) = -0.46 + 0.17 * x

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	-0.42	-0.53	-0.67	-0.82	-0.95	-1.0	-0.94	-0.77	-0.53	-0.27	0.0
φ (x_i)	-0.8	-0.77	-0.73	-0.7	-0.66	-0.63	-0.6	-0.56	-0.53	-0.49	-0.46
(φ (x_i)-y_i)^2	0.144	0.056	0.004	0.015	0.082	0.137	0.118	0.043	0	0.05	0.212

$$\sigma = \sqrt{\frac{\sum(\varphi\ (xi) - yi)^2}{n}} = \mathbf{0.278}$$

Квадратичная аппроксимация:

y = $\frac{4x}{x^4+3}$

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	-0.42	-0.53	-0.67	-0.82	-0.95	-1.0	-0.94	-0.77	-0.53	-0.27	0.0

$\varphi(x) = a + b * x + c * x\text{^}2$

Вычисляем суммы: sx = -11, sxx = 15.4, sxxx = -24.2, sxxxx = 40.53,
sy = -6.9, sxy = 7.63, sxxy = -10.06

$$\begin{cases} n*a + sx*b + sxx*c = sy \\ sx*a + sxx*b + sxxx*c = sxy \\ sxx*a + sxxx*b + sxxxx*c = sxxy \end{cases} \quad \begin{cases} 11*a - 11*b + 15.4*c = -6.9 \\ -11*a + 15.4*b - 24.2*c = 7.63 \\ 15.4*a - 24.2*b + 40.53*c = -10.06 \end{cases}$$

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

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

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

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

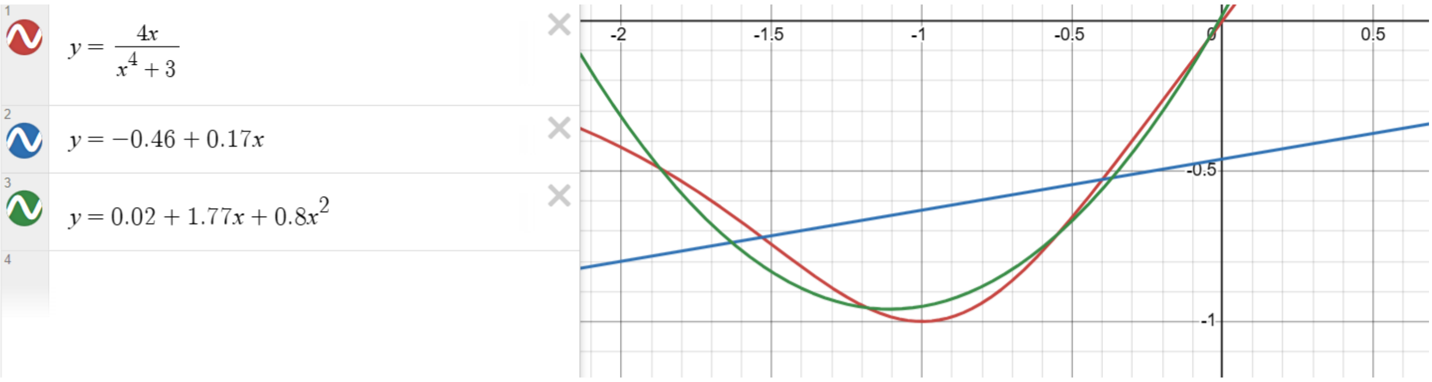
$$\begin{cases} a = 0.02 \\ b = 1.77 \\ c = 0.8 \end{cases}$$

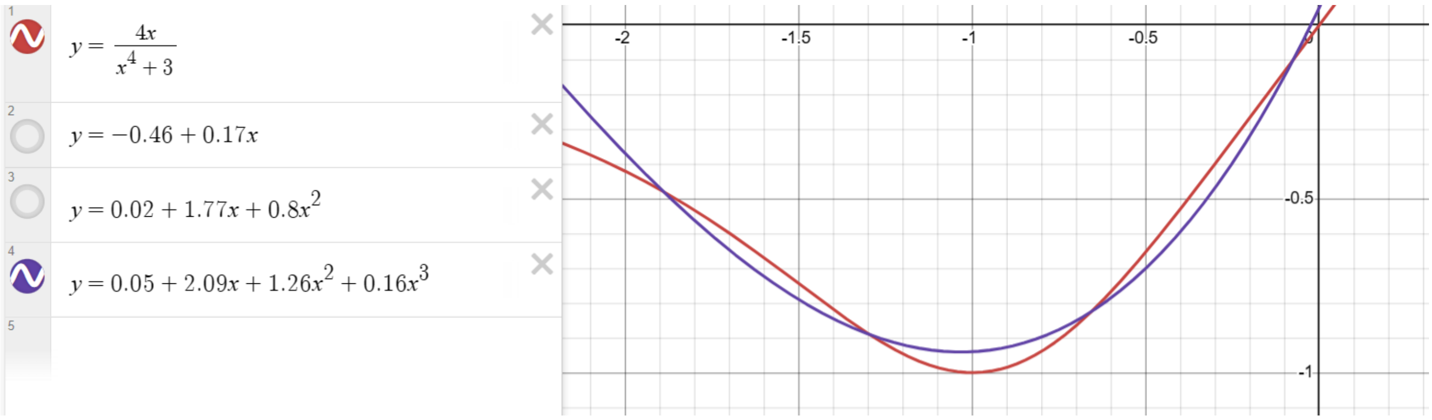
$\varphi(x) = 0.02 + 1.77 * x + 0.8 * x \text{ ^ } 2$

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	-0.42	-0.53	-0.67	-0.82	-0.95	-1.0	-0.94	-0.77	-0.53	-0.27	0.0
$\varphi(x_i)$	-0.32	-0.57	-0.76	-0.89	-0.95	-0.95	-0.88	-0.75	-0.56	-0.3	0.02
$(\varphi(x_i)-y_i)\text{^}2$	0.01	0.002	0.009	0.005	0.0	0.003	0.003	0.0	0.001	0.001	0.0

$\sigma = \sqrt{\frac{\sum(\varphi(xi) - yi)^2}{n}} = \mathbf{0.056}$

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





Для кубической аппроксимации: $\sigma = \sqrt{\frac{\sum(\varphi\ (xi) - yi)^2}{n}} = \mathbf{0.045}$

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):
```

```python
    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]
```

```python
        ],
        [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):
```

```python
    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))
```

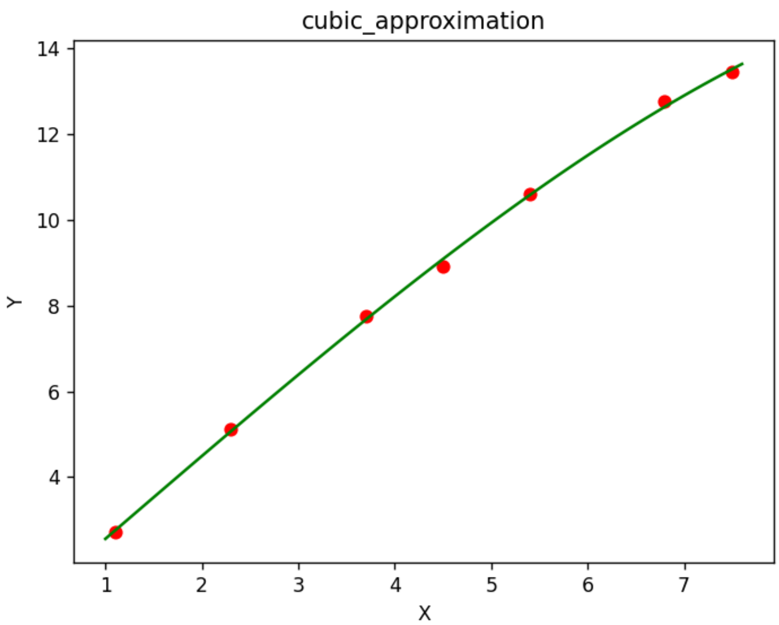
```
        plt.title(apprxmt_f.__name__)

        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

```
xs3 = [1.1,  2.3,  3.7,  4.5,  5.4,   6.8,   7.5]
ys3 = [2.73, 5.12, 7.74, 8.91, 10.59, 12.75, 13.43]
```



```
linear_approximation :
fi(x) = a + b * x
coeffs (a, b): [1.2168, 1.6854]
S = 0.47302, σ = 0.25995, R2 = 0.99484
r = 0.9974189309974396
--------------------------------------------------
power_approximation :
fi(x) = a * x ** b
coeffs (a, b): [2.5421, 0.838]
S = 0.15440, σ = 0.14851, R2 = 0.99832
--------------------------------------------------
exponential_approximation :
fi(x) = a * exp(b * x)
coeffs (a, b): [2.7309, 0.2346]
S = 10.70709, σ = 1.23676, R2 = 0.88332
--------------------------------------------------
logarithmic_approximation :
fi(x) = a + b * log(x)
coeffs (a, b): [1.1989, 5.65]
S = 4.19978, σ = 0.77458, R2 = 0.95423
--------------------------------------------------
quadratic_approximation :
fi(x) = a + b * x + c * x ** 2
coeffs (a, b, c): [0.3743, 2.1974, -0.0589]
S = 0.06901, σ = 0.09929, R2 = 0.99925
--------------------------------------------------
cubic_approximation :
fi(x) = a + b * x + c * x ** 2 + d * x ** 3
coeffs (a, b): [0.6398, 1.9119, 0.0191, -0.006]
S = 0.05940, σ = 0.09212, R2 = 0.99935
--------------------------------------------------
best_func: cubic_approximation
```

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.