# Аппроксимация методом наименьших квадратов на Python

Д.П. Худяков

6 июля 2021 г.

**Аннотация**

В данном документе приводится описание алгоритма аппроксимации данных полиномом n - степени от m переменных методом наименьших квадратов. Сначала будет рассмотрен алгоритм для полинома третьей степени и двух переменных, далее он будет обобщён с использованием рекурсивных функций. В конце будут приведены примеры использования алгоритма.

## 1 Аппроксимация МНК полиномом третьей степени, двух переменных

Полином имеет следующий вид:

$$y = y(x_1, x_2) = a_1 + a_2x_1 + a_3x_2 + a_4x_1^2 + a_5x_1x_2 + a_6x_2^2 + a_7x_1^3 + a_8x_1x_2^2 + a_9x_1x_2^2 + a_{10}x_2^3 \quad (1)$$

Задача состоит в минимизации функции - суммы разности квадратов, подбором параметров $a_i$:

$$F(\overline{a}) = \sum_{i=1}^{k}(y_i - y(x_{1i}, x_{2i}))^2 \quad (2)$$

где $\overline{a} = \{a_1; a_2; ...; a_{10}\}$, $k$ - количество экспериментальных точек, $y_i, x_{1i}, x_{2i}$ - экспериментальные значения.

Функция будет иметь минимум в точке, где производная равна нулю. Это условие можно записать в виде системы таким образом:

$$\begin{cases} \frac{\partial f(\overline{a})}{\partial a_1} = -2 \cdot \sum_{i=1}^{k}(y_i - y(x_{1i}, x_{2i})) \cdot 1 = 0 \\ \frac{\partial f(\overline{a})}{\partial a_2} = -2 \cdot \sum_{i=1}^{k}(y_i - y(x_{1i}, x_{2i})) \cdot x_{1i} = 0 \\ \cdots \\ \frac{\partial f(\overline{a})}{\partial a_{10}} = -2 \cdot \sum_{i=1}^{k}(y_i - y(x_{1i}, x_{2i})) \cdot x_{2i}^3 = 0 \end{cases} \quad (3)$$

После преобразований:

$$\begin{cases} \sum_{i=1}^{k} y(x_{1i}, x_{2i}) \cdot 1 = \sum_{i=1}^{k} y_i \cdot 1 \\ \sum_{i=1}^{k} y(x_{1i}, x_{2i}) \cdot x_{1i} = \sum_{i=1}^{k} y_i \cdot x_{1i} \\ \cdots \\ \sum_{i=1}^{k} y(x_{1i}, x_{2i}) \cdot x_{2i}^3 = \sum_{i=1}^{k} y_i \cdot x_{2i}^3 \end{cases} \quad (4)$$

Система урванений в матричном виде:

$$AX = B \tag{5}$$

Или:

$$\begin{pmatrix} \sum_{i=1}^{k} 1 \cdot 1 & \sum_{i=1}^{k} x_{1i} \cdot 1 & \sum_{i=1}^{k} x_{2i} \cdot 1 & \vdots & \sum_{i=1}^{k} x_{2i}^3 \cdot 1 \\ \sum_{i=1}^{k} 1 \cdot x_{1i} & \sum_{i=1}^{k} x_{1i} \cdot x_{1i} & \sum_{i=1}^{k} x_{2i} \cdot x_{1i} & \vdots & \sum_{i=1}^{k} x_{2i}^3 \cdot x_{1i} \\ \dots & \dots & \dots & \vdots & \dots \\ \sum_{i=1}^{k} 1 \cdot x_{2i}^3 & \sum_{i=1}^{k} x_{1i} \cdot x_{2i}^3 & \sum_{i=1}^{k} x_{2i} \cdot x_{2i}^3 & \vdots & \sum_{i=1}^{k} x_{2i}^3 \cdot x_{2i}^3 \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \\ \dots \\ a_{10} \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{k} y_i \cdot 1 \\ \sum_{i=1}^{k} y_i \cdot x_{1i} \\ \dots \\ \sum_{i=1}^{k} y_i \cdot x_{2i}^3 \end{pmatrix}$$

Формулу полинома можно записать таким образом:

$$y(x_{1i}, x_{2i}) = \sum_{i=0}^{i=3} \sum_{j=0}^{j=i} a_{\frac{1+i}{2} \cdot i + j + 1} \cdot x_1^{i-j} \cdot x_2^j \tag{6}$$

Замена $x_1^{i-j} \cdot x_2^j = \psi_k$, где $k = \frac{1+i}{2} \cdot i + j + 1$, позволит записать матрицы $A$ и $B$ так:

$$A = \begin{pmatrix} \sum_{i=1}^{k} \psi_{1i} \cdot \psi_{1i} & \sum_{i=1}^{k} \psi_{2i} \cdot \psi_{1i} & \sum_{i=1}^{k} \psi_{3i} \cdot \psi_{1i} & \vdots & \sum_{i=1}^{k} \psi_{10i} \cdot \psi_{1i} \\ \sum_{i=1}^{k} \psi_{1i} \cdot \psi_{2i} & \sum_{i=1}^{k} \psi_{2i} \cdot \psi_{2i} & \sum_{i=1}^{k} \psi_{3i} \cdot \psi_{2i} & \vdots & \sum_{i=1}^{k} \psi_{10i} \cdot \psi_{2i} \\ \dots & \dots & \dots & \vdots & \dots \\ \sum_{i=1}^{k} \psi_{1i} \cdot \psi_{10i} & \sum_{i=1}^{k} \psi_{2i} \cdot \psi_{10i} & \sum_{i=1}^{k} \psi_{3i} \cdot \psi_{10i} & \vdots & \sum_{i=1}^{k} \psi_{10i} \cdot \psi_{10i} \end{pmatrix}$$

$$B = \begin{pmatrix} \sum_{i=1}^{k} y_i \cdot \psi_{1i} \\ \sum_{i=1}^{k} y_i \cdot \psi_{2i} \\ \dots \\ \sum_{i=1}^{k} y_i \cdot \psi_{10i} \end{pmatrix}$$

Для составления матриц $A$ и $B$ достаточно реализовать функцию для нахождения $\psi_i$:

```
KI_LIST = [[1, 0], [2, 1], [4, 2], [7, 3]]


def psi(num_p, k, data):
    global KI_LIST
    i = 0
    for ki_curr in KI_LIST:
        if k >= ki_curr[0]:
            i = ki_curr[1]
        else:
            break
    j = k - 1 - i * (1 + i) / 2.0
    power = [i - j, j]
    return (data[num_p][0] ** power[0]) * (data[num_p][1] ** power[1])
```

Здесь data - массив экспериментальных точек, num_p - номер текущей точки, k - индекс $\psi$. KI_LIST составлен циклом по $i$ от 0 до 3 по формуле $[\frac{1+i}{2} i + 1, i]$.

Функции для составления матриц $A$ и $B$ представлены ниже:

```
def calc_a(exp_data, n):
    a = [[0.0] * n for i in range(n)]
    for n1 in range(1, n + 1):
        for n2 in range(1, n + 1):
            a_curr = 0.0
            for n3 in range(len(exp_data)):
                a_curr += psi(n3, n2, exp_data) * psi(n3, n1, exp_data)
            a[n1 - 1][n2 - 1] = a_curr
    return a


def calc_b(exp_data, n):
    b = []
    for n1 in range(1, n + 1):
        b_curr = 0.0
        for n2 in range(len(exp_data)):
            b_curr += exp_data[n2][2] * psi(n2, n1, exp_data)
        b.append(b_curr)
    return b
```

С использованием библиотеки NumPy алгоритм нахождения коэффициентов полинома имеет следующий вид:

```
a = calc_a(exp_data, n)
b = calc_b(exp_data, n)
res = np.linalg.solve(np.array(a), np.array(b))
```

# 2 Аппроксимация МНК полиномом n степени, m переменных

## 2.1 Представление полинома

Формула полинома n степени, m переменных:

$$\sum_{i_1=0}^{i_1=n}\sum_{i_2=0}^{i_2=i_1}\dots\sum_{i_{m-1}=0}^{i_{m-1}=i_{m-2}}\sum_{i_m=0}^{i_m=i_{m-1}} x_1^{i_1-i_2}\cdot x_2^{i_2-i_3}\cdot\ldots\cdot x_{m-1}^{i_{m-1}-i_m}\cdot x_m^{i_m} \tag{7}$$

Функции для записи такого полинома:

```
def get_ext_list(init_ext_list):
    print("init_ext_list = ", init_ext_list)
    ext_list = []
    for n in range(len(init_ext_list)):
        if n == len(init_ext_list) - 1:
            ext_list.append(init_ext_list[n])
        else:
            ext_list.append(init_ext_list[n] - init_ext_list[n + 1])
    print("ext_list = ", ext_list)
    return ext_list


def pol_mul(c_list, init_ext_list, str_res, curr_a):
```

```
    ext_list = get_ext_list(init_ext_list)
    str_add_v = " + (" + str(c_list[curr_a - 1]) + " * "
    for n_curr in range(len(ext_list)):
        str_add_v += "x" + str(n_curr) + "^" + str(ext_list[n_curr])
        if n_curr != len(ext_list) - 1:
            str_add_v += " * "
    str_res += str_add_v + ")"
    return str_res


def calc_pol(c_list, pol_ext, nv, n_call, ext_list, str_res, curr_a):
    n_call_new = n_call + 1
    new_ext_list = copy.deepcopy(ext_list)
    new_ext_list.append(0)
    for i in range(pol_ext + 1):
        new_ext_list[-1] = i
        if n_call_new == nv:
            curr_a += 1
            str_res = pol_mul(c_list, new_ext_list, str_res, curr_a)
        else:
            str_res, curr_a = calc_pol(c_list, i, nv, n_call_new,
                                       new_ext_list, str_res, curr_a)
    return str_res, curr_a


str_res, curr_a = calc_pol(c, pol_ext, nv, 0, [], "", 0)
```

В качестве аргументов на вход подаётся массив из коэффициентов полинома - c, степень полинома - pol_ext, количество переменных - nv и рекурсивно заполняемые переменные. Переменная n_call считает количество вызовов функции и используется во время проверки условия остановки - равенство количеству переменных. Переменная ext_list служит для хранения массива степеней переменных текущего слагаемого полинома. str_res - переменная для хранения результата. curr_a служит для хранения индекса текущего коэффициента, определяемого рекурсивно. Функция calc_pol вызывает саму себя, обновляя массив индексов каждой переменной, пока количество вызовов не превышает количество переменных, а при их равенстве вызывает функцию составления слагаемого полинома, передавая ей массив индексов. Фнкция составления слагаемого - pol_mul использует функцию get_ext_list для составления массива степеней согласно формуле (7).
В результате вызова функции calc_pol с такими параметрами:

```
calc_pol(["a1", "a2", "a3", "a4", "a5", "a6"], 2, 2, 0, [], "", 0)
```

Получена строка:
$polynomial = (a1 \cdot x_0^0 \cdot x_1^0) + (a2 \cdot x_0^1 \cdot x_1^0) + (a3 \cdot x_0^0 \cdot x_1^1) + (a4 \cdot x_0^2 \cdot x_1^0) + (a5 \cdot x_0^1 \cdot x_1^1) + (a6 \cdot x_0^0 \cdot x_1^2)$

## 2.2 Составление матриц A и B

Обозначив за $\phi_{ji}$ - произведение аргументов полинома при j-ом коэффициенте в i-ой точке, матрицу B можно записать таким образом:

$$B = \begin{pmatrix} \sum_{i=1}^{k} y_i \cdot \phi_{1i} \\ \sum_{i=1}^{k} y_i \cdot \phi_{2i} \\ \dots \\ \sum_{i=1}^{k} y_i \cdot \phi_{ji} \end{pmatrix}$$

Тогда функции для составления матрицы B будут выглядеть аналогично функциям для составления полинома:

```
def b_mul(b, exp_list, init_ext_list):
    sum_add_v_all = 0.0
    ext_list = get_ext_list(init_ext_list)
    for curr_exp in exp_list:
        sum_add_v = curr_exp[-1]
        for n_curr in range(len(ext_list)):
            sum_add_v *= curr_exp[n_curr] ** ext_list[n_curr]
        sum_add_v_all += sum_add_v
    b.append(sum_add_v_all)
    return b


def calc_b(exp_list, b, pol_ext, nv, n_call, ext_list):
    n_call_new = n_call + 1
    new_ext_list = copy.deepcopy(ext_list)
    new_ext_list.append(0)
    for i in range(pol_ext + 1):
        new_ext_list[-1] = i
        if n_call_new == nv:
            b = b_mul(b, exp_list, new_ext_list)
        else:
            b = calc_b(exp_list, b, i, nv, n_call_new, new_ext_list)
    return b


b = calc_b(exp_data, [], pol_ext, nv, 0, [])
```

где exp_list - массив эксперементальных точек, b - искомая матрица.
Матрица A при введенных обозначениях имеет следующий вид:

$$A = \begin{pmatrix} \sum_{i=1}^{k} \phi_{1i} \cdot \phi_{1i} & \sum_{i=1}^{k} \phi_{2i} \cdot \phi_{1i} & \sum_{i=1}^{k} \phi_{3i} \cdot \phi_{1i} & \vdots & \sum_{i=1}^{k} \phi_{ji} \cdot \phi_{1i} \\ \sum_{i=1}^{k} \phi_{1i} \cdot \phi_{2i} & \sum_{i=1}^{k} \phi_{2i} \cdot \phi_{2i} & \sum_{i=1}^{k} \phi_{3i} \cdot \phi_{2i} & \vdots & \sum_{i=1}^{k} \phi_{ji} \cdot \phi_{2i} \\ \dots & \dots & \dots & \vdots & \dots \\ \sum_{i=1}^{k} \phi_{1i} \cdot \phi_{ji} & \sum_{i=1}^{k} \phi_{2i} \cdot \phi_{ji} & \sum_{i=1}^{k} \phi_{3i} \cdot \phi_{ji} & \vdots & \sum_{i=1}^{k} \phi_{ji} \cdot \psi_{ji} \end{pmatrix}$$

Функции для составления матрицы A отличаются только количеством измерений в которых ведется расчет, для их учета введена переменная dim.

```
def a_mul(a, exp_list, init_ext_list):
    sum_add_v_all1 = 0.0
```

```
    ext_list = []
    for curr_init_list in init_ext_list:
        ext_list.append(get_ext_list(curr_init_list))
    for curr_exp in exp_list:
        sum_add_v = 1.0
        for curr_ext_list in ext_list:
            for n_curr in range(len(ext_list[0])):
                sum_add_v *= curr_exp[n_curr] ** curr_ext_list[n_curr]
        sum_add_v_all1 += sum_add_v
    a.append(sum_add_v_all1)
    return a


def calc_a(exp_list, a, pol_ext, nv, n_call, ext_list, dim, real_pol_ext):
    n_call_new = n_call + 1
    new_ext_list = copy.deepcopy(ext_list)
    new_ext_list[dim].append(0)
    for i in range(pol_ext + 1):
        new_ext_list[dim][-1] = i
        if (n_call_new == nv) and (dim != 0):
            a = calc_a(exp_list, a, real_pol_ext, nv, 0, new_ext_list,
                       dim - 1, real_pol_ext)
        elif (n_call_new == nv) and (dim == 0):
            a = a_mul(a, exp_list, new_ext_list)
        else:
            a = calc_a(exp_list, a, i, nv, n_call_new, new_ext_list,
                       dim, real_pol_ext)
    return a
```

Решение матричного уравнения проводится с помощью библиотеки NumPy.

```
c = np.linalg.solve(np.array(a), np.array(b))
```

# 3 Примеры

## 3.1 Пример 1

Результат аппроксимации функции $f(x) = \cos(x) - \frac{x}{2}$ на отрезке от 0 до $2\pi$ полиномом пятой степени представлен на Рис.1. А сам полином имеет вид:
$polynomial = (0.9816430700989729 \cdot x_0^0) + (-0.1963275611078208 \cdot x_0^1) + (-1.033741420304099 \cdot x_0^2)$
$+ (0.31366588837963355 \cdot x_0^3) + (-0.02496073831109367 \cdot x_0^4) + (4.1167951800195695e - 13 \cdot x_0^5)$

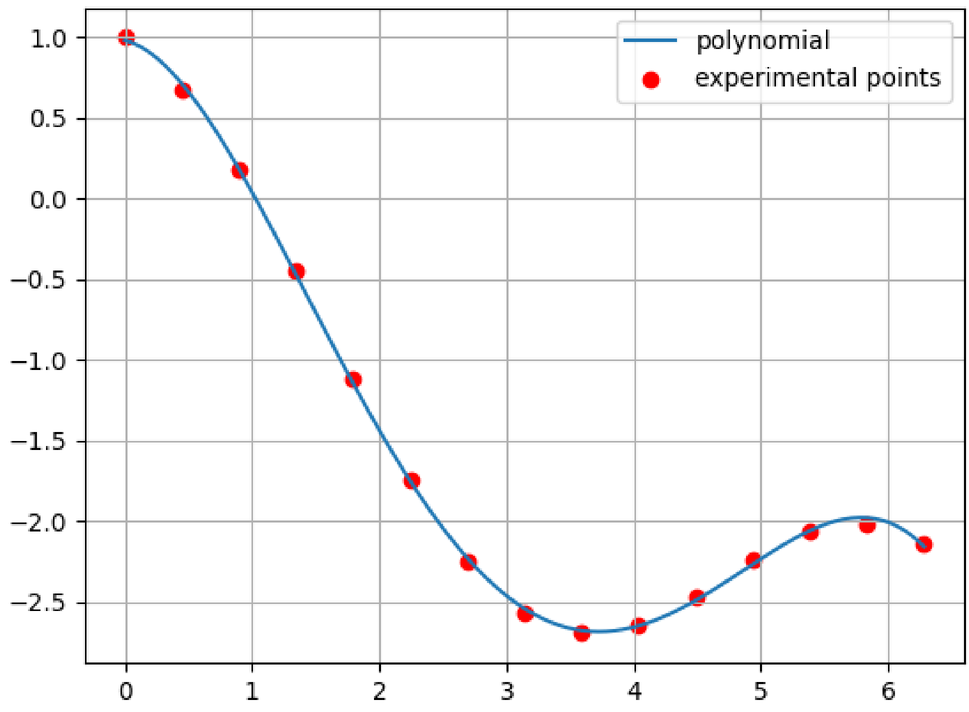


Рис. 1: График аппроксимирующей функции

## 3.2 Пример 2

Результат аппроксимации данных в трехмерном пространстве заданных в виде графика - Рис.2 представлен на Рис.3. Полином третьей степени двух переменных выглядит так:
$polynomial = (0.42196048238646816 \cdot x_0^0 \cdot x_1^0) + (0.5321981377555719 \cdot x_0^1 \cdot x_1^0) + (-2.142761529696963 \cdot x_0^0 \cdot x_1^1) + (-0.7492271056918415 \cdot x_0^2 \cdot x_1^0) + (1.6516540144139782 \cdot x_0^1 \cdot x_1^1) + (4.371662253412071 \cdot x_0^0 \cdot x_1^2) + (0.4619208853053831 \cdot x_0^3 \cdot x_1^0) + (-2.0296581753147445 \cdot x_0^2 \cdot x_1^1) + (1.842665698725525 \cdot x_0^1 \cdot x_1^2) + (-1.6722006702730612 \cdot x_0^0 \cdot x_1^3)$

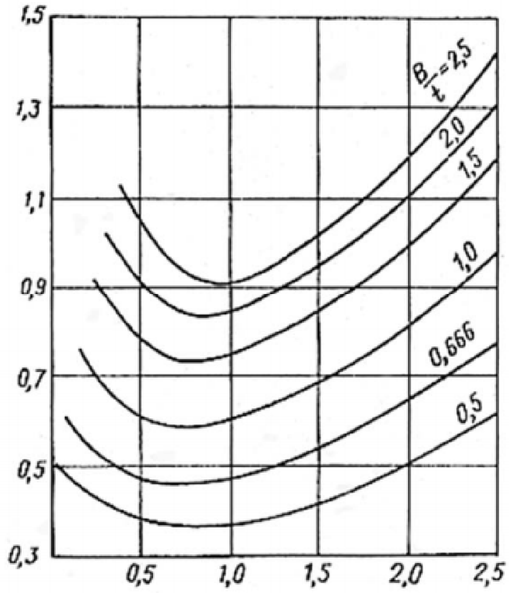


Рис. 2: Главная характеристика номинальных режимов

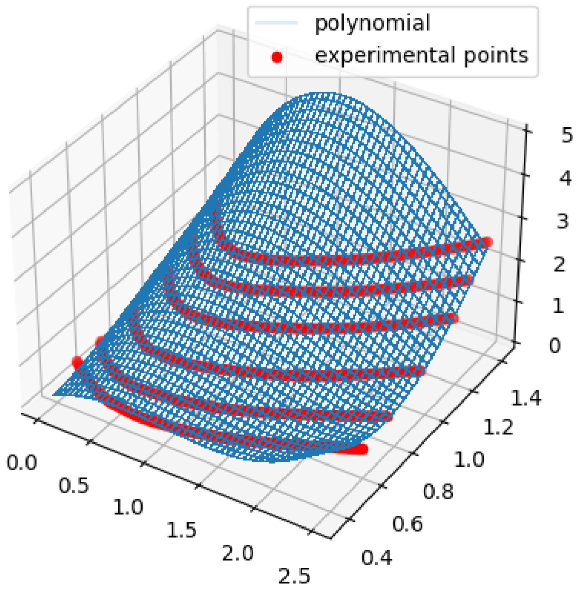


Рис. 3: График аппроксимирующей поверхности

## 3.3 Пример 3

Аппроксимация набора данных boston-housing о стоимости домов полиномом первой степени (линейная регрессия) показывает RMSE = 3.287, а коэффициенты этого полинома имеют следующие значения:
$a_1 = 3.55457677e + 01$
$a_2 = -1.06574337e - 01$
$a_3 = 4.91031831e - 02$

$a_4 = 4.91031831e - 02$
$a_5 = 2.50883963e + 00$
$a_6 = -1.75828551e + 01$
$a_7 = 3.82281505e + 00$
$a_8 = 1.05141853e - 02$
$a_9 = -1.43435667e + 00$
$a_{10} = 3.62353199e - 01$
$a_{11} = -1.54828341e - 02$
$a_{12} = -9.11362333e - 01$
$a_{13} = 9.71340993e - 03$
$a_{14} = -5.55041015e - 01$