Отчёт

Метод наименьших квадратов

Пусть $y_i, i = 1, \dots n$ набор скалярных экспериментальных данных, $x_i, i = 1, \dots n$ набор векторных экспериментальных данных и предполагается, что y зависит от x.

Вводится некоторая (в простейшем случае линейная) скалярная функция $f(x, \boldsymbol{b})$ которая определяется вектором неизвестных параметров $\boldsymbol{b}$.

Ставится задача найти вектор $\boldsymbol{b}$ такой, чтобы совокупность погрешностей

$$r_i = y_i - f(x, \boldsymbol{b})$$

была в некотором смысле минимальной.

Согласно методу наименьших квадратов решением этой задачи является вектор β, который минимизирует функцию

$$S(\beta) = \sum_{i=1}^{n} \big(y_i - f(x_i, \boldsymbol{b})\big)$$

Для решения задачи найдём стационарные точки $S(\boldsymbol{b})$, продифференцировав её по неизвестным параметрам $\boldsymbol{b}$, приравняв производные к нулю и решив полученную систему уравнений:

$$\sum_{t=1}^{n} \big(y_t - f(x_t, \boldsymbol{b})\big) \frac{\partial f(x_t, \boldsymbol{b})}{\partial b_i} = 0, \qquad i = 0, \dots m$$

Так как $f(x, \boldsymbol{b}) = b_0 + b_1 x + \dots + b_m x^m$, то

$$\frac{\partial f(x_t, \boldsymbol{b})}{\partial b_i} = x_t^i$$

Тогда

$$\sum_{t=1}^{n} \big(y_t - f(x_t, \boldsymbol{b})\big) x_t^i = 0$$

$$\sum_{t=1}^{n} f(x_t, \boldsymbol{b}) x_t^i = \sum_{t=1}^{n} y_t x_t^i$$

$$\sum_{t=1}^{n} \sum_{j=0}^{m} b_j x_t^j \, x_t^i = \sum_{t=1}^{n} y_t x_t^i$$

$$\sum_{j=0}^{m}\left(\sum_{t=1}^{n} x_t^{i+j}\right) * b_j = \sum_{t=1}^{n} y_t x_t^i$$

$\boldsymbol{Ab} = \boldsymbol{c}$ (*), где

$$\boldsymbol{A} = \left\{a_{i,j} = \sum_{t=1}^{n} x_t^{i+j}\right\}, \qquad \boldsymbol{c} = (c_i)^T$$

Вектор искомых параметров находится как решение СЛАУ (*)

Исходные данные

№ п/п	Тип ЛА	Max P, % от ном.	Начальная		Конечная	
			высота (м)	скорость (км/ч)	высота (м)	скорость (км/ч)
17	ТУ-154	105	700	350	9250	980

Программа для аппроксимации данных и построения полученных графиков

Программа была написана мной на языке программирования Python в среде разработки PyCharm с использованием графической библиотеки Matplotlib и математической библиотеки NumPy. Несмотря на низкую скорость работы, Python хорошо подходит для прототипирования. В связи с тем, что большая часть кода использованных библиотек написана на языке C, секции, относящиеся к решению СЛАУ (*) и построению графиков, оптимизированы.

Для первой части ДЗ я выбрал для аппроксимации зависимости ускорения свободного падения g, плотности атмосферы ρ, давления p и температуры T от геометрической высоты h. Но поскольку программа хорошо декомпозирована, то не составляет труда получить данные для других величин из других источников данных.

$$g = 9.806666 - 0.000003 * h$$

$$\rho = 1.21092 - 0.000105 * h$$

$$p = 98833.684451 - 9.962727 * h + 0.000272 * h^2$$

$$T = 291.673571 - 0.009307 * h$$

Графики функций приложены в конце отчёта.

Данные атмосферы задаются в столбцах таблицы формата .csv по следующим правилам:

1. Первая строка — название величины
2. Вторая строка — буквенное обозначение (с размерностью, для использования в качестве подписи оси)
3. Третья и далее — сами данные.

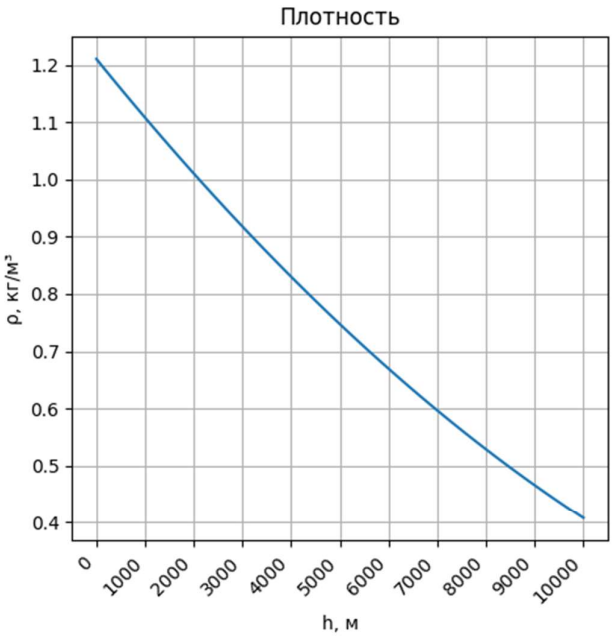
Для каждой величины собственная функция polyfill составляет из исходных данных компоненты СЛАУ (*), передаёт их функции np.linalg.solve, которая непосредственно решает систему и возвращает массив numpy вида [b0, b1, ..., bm], соответствующий функции $f(x, \boldsymbol{b})$

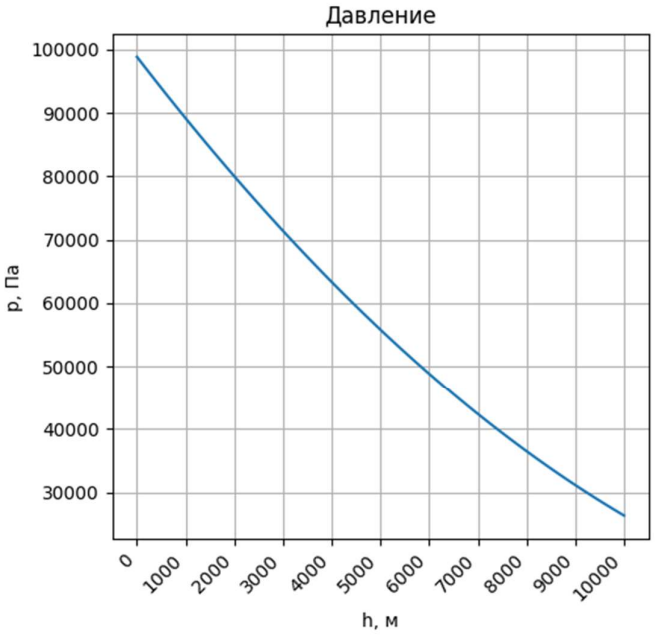
```
def polyfill(x: list, y: list, m: int):
    """
    Аппроксимирует зависимость между множествами x и y к полиному степени m
    :return: массив NumPy вида [b0, b1, ... bm], соответствующий полиному :raw-
html:`<br />` b0 + b1 * x + ... + bm * x ^ m
    """
    A_raw = [[sum(x[t] ** (i + j) for t in range(len(x))) for j in range(m + 1)]
for i in range(m + 1)]
    A = np.array(A_raw, dtype=float)
    c = np.fromiter((sum(y[t] * x[t] ** i for t in range(len(x))) for i in
range(m + 1)), dtype=float)
    b = np.linalg.solve(A, c)
    return b
```

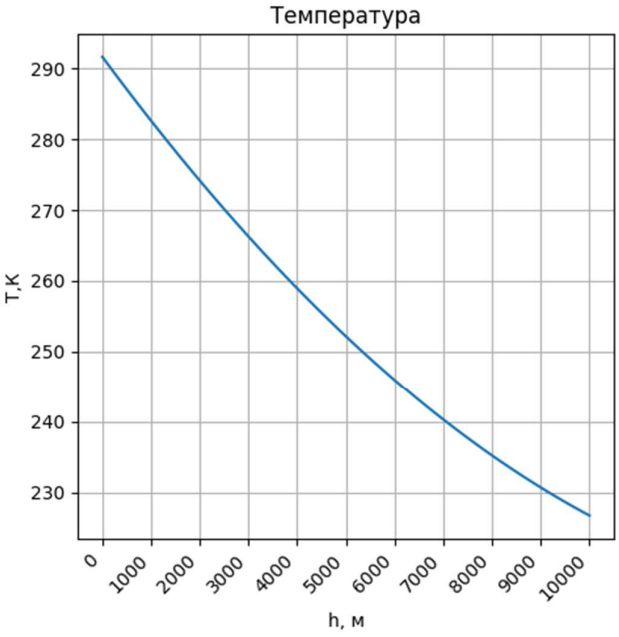
Для вычисления ординат точек при построении графиков используется собственная функция polyval:

```
def polyval(poly, x):
    result = 0
    for coeff in reversed(poly):
        result = result * x + coeff
    return result
```


Ускорение свободного падения
g, м/с²
9.805
9.800
9.795
9.790
9.785
9.780
9.775
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
h, м


Плотность
ρ, кг/м³
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
h, м


Давление
p, Па
100000
90000
80000
70000
60000
50000
40000
30000
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
h, м


Температура
T,K
290
280
270
260
250
240
230
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
h, м