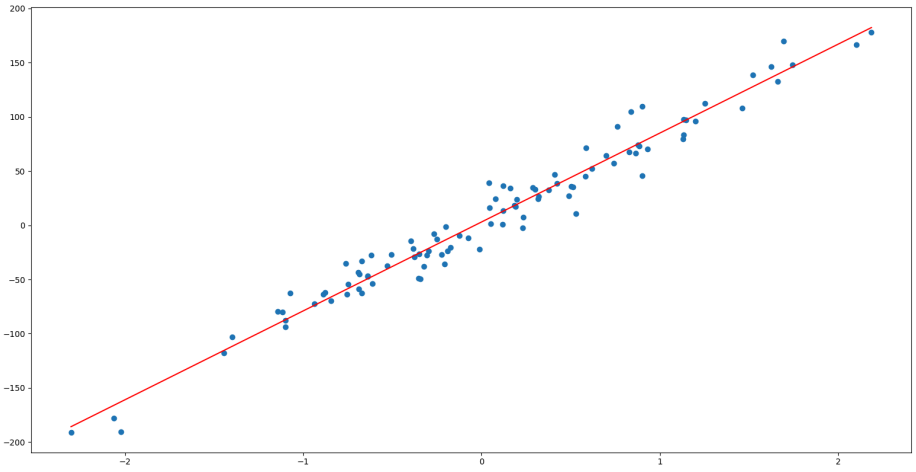
3.2 Линейная регрессия

- **Задача**: на основе наблюдаемых точек построить прямую, которая отображает связь между двумя или более переменными .abs
- **Регрессия** пытается “подогнать” функцию к наблюдаемым данным, чтобы спрогнозировать новые данные
- **Линейная регрессия** подгоняем данные к прямой лини, пытаемся установить линейную связь между переменными и предсказать новые данные

Линейная регрессия

```
import numpy as np
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Создаем искусственный набор данных для задачи регрессии.
# n_samples=100: 100 примеров (точек).
# n_features=1: 1 признак (ось X).
# n_informative=1: 1 признак является значимым.
# n_targets=1: 1 целевая переменная (ось Y).
# noise=1: Уровень шума (разброс точек вокруг идеальной линии).
# random_state=1: Для воспроизводимости результатов.
features, target = make_regression(
    n_samples=100, n_features=1, n_informative=1,
    n_targets=1, noise=15, random_state=1
)

# print(features)
# [[-1.07296862]
#  [-0.52817175]
#  [ 1.62434536]
#  [ 0.86540763]
#  [ 1.74481176]
#  [-2.3015387 ]
#  [-0.61175641]]

# print(target)
# [-38.59253069  15.79052414  21.79200288  20.41166767  27.17541562
```

```
#  -38.59030346 -25.40609661]

model = LinearRegression().fit(features, target)

# Рисуем исходные данные в виде диаграммы рассеяния (точек)
plt.scatter(features, target)

# Для построения линии регрессии:
# 1. Создаем равномерный диапазон значений по оси X
x = np.linspace(features.min(), features.max(), 100)
# 2. Рассчитываем предсказания Y, используя формулу y = kx + b,
#    где 'k' - это model.coef_[0], а 'b' - это model.intercept_
# Рисуем линию регрессии красным цветом
y_pred = model.coef_[0] * x + model.intercept_
x = np.linspace(features.min(), features.max(), 100)
plt.plot(x, model.coef_[0] * x + model.intercept_, 'r')

plt.show()
```



Простая линейная регрессия

Линейная подразумевает линейную зависимость - это значит что если одна переменная увеличивается или уменьшается, то и результат уменьшается пропорционально.

Плюсы	Минусы
прогнозирование на новых данных	точки, обучаемых данных не будут точно лежать на прямой (шум) ⇒ область погрешности

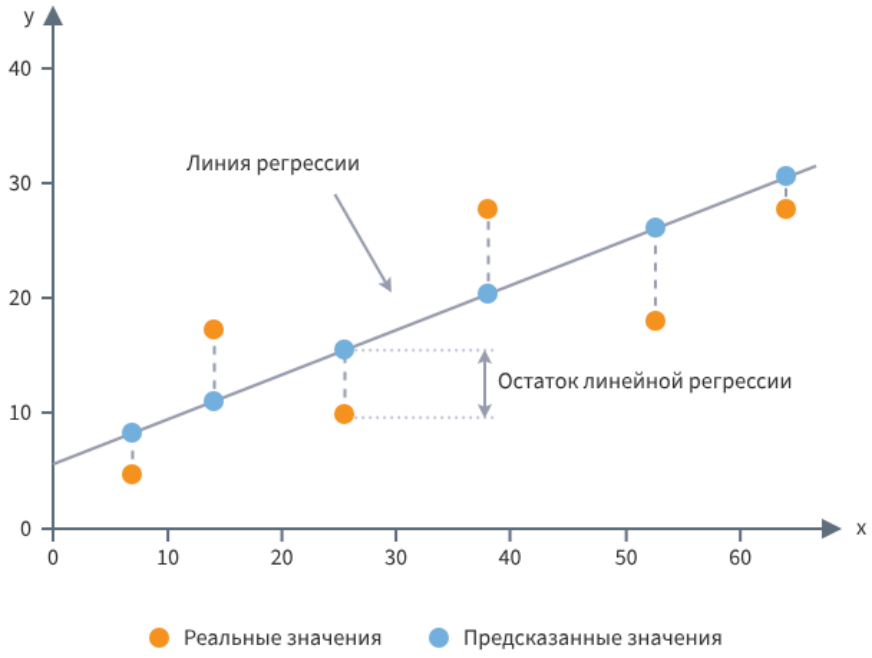
Плюсы	Минусы
анализ взаимного влияния переменных друг на друга	не позволяет делать прогнозы вне диапазона имеющихся данных

Данные, на основании которых разрабатывается модель - это выборка из совокупности, хотелось бы, чтобы это была РЕПРЕЗЕНТАТИВНАЯ выборка.

Каким образом подбирать линию, которая проходит через некоторое множество точек?

Необходимо провести линию которая будет лучше всего проходит через все точки - метод наименьших квадратов (**МНК**).

- **Остатки (отклонения / ошибки)** - расстояния между точками данных и ближайшими по вертикали точками на прямой. Если выше примой, то остаток положительный, если ниже - отрицательный.



OX - features; OY - target

Задача состоит в минимизации остатков ⇒ то разрыв между прямой и точками будет минимальным

Набор остатков: $R_1, R_2, ..., R_n$

- Можно минимизировать сумму $\sum_{i=0}^{n} R_i$, но это плохая идея, так как положительны и отрицательные остатки будут друг друга компенсировать

- Можно складывать абсолютные значения $\sum_{i=0}^{n} |R_i|$, но методы связанные с нахождением прямой связаны с нахождением производной, но производной в точке $(0, 0)$ не существует, придется дорабатывать метод
- Чаще всего используют $\sum_{i=0}^{n} R_i^2$, тогда наша функция - парабола, где производная определена в каждой точке

Обучение модели сводится к минимизации (функция потерь)

Аналитическое решение

Для $y = w_0 + w_1$ существует несколько решений, одно из важных - это наличие аналитического решения.

Решение можно разбить на две группы

Численные

Они легче и доступнее, но требуют большей вычислительной мощности. Дает неточные решения с некоторой погрешностью

Иногда это решение является единственным

Аналитические

1. Метод наименьших квадратов

Правильные (опираются на математическую теорию) сложные, а иногда и просто невозможные.

Дана табличная функция (x_i, y_i) и нам нужно минимизировать все ее остатки, то есть нам надо найти w_0 и w_1 (МНК):

$$w_1 = \frac{n \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

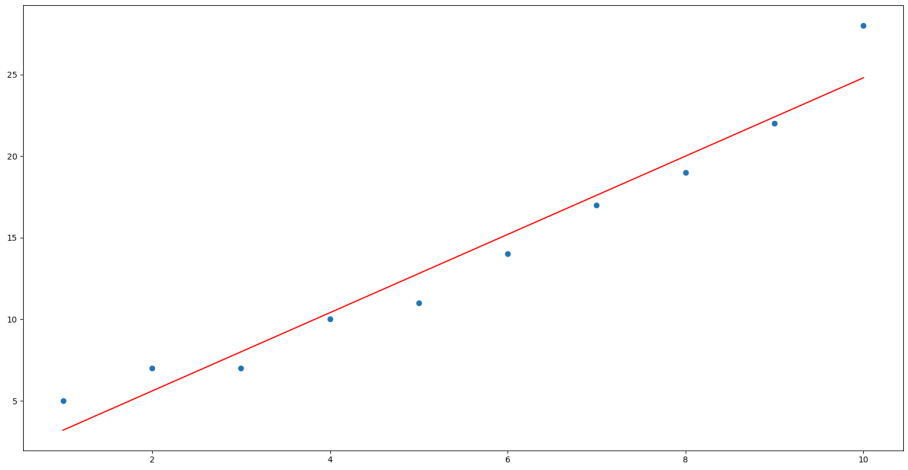
$$w_0 = \frac{\sum_{i=1}^{n} y_i}{n} - w_1 \frac{\sum_{i=1}^{n} x_i}{n}$$

n - число точек

Практическая реализация

```
import numpy as np
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

data = np.array(
   [
```

```
        [1, 5], [2, 7], [3, 7], [4, 10], [5, 11],
        [6, 14], [7, 17], [8, 19], [9, 22], [10, 28],
    ]
)

x = data[:, 0]
y = data[:, 1]
n = len(x)

w_1 = (n * np.sum(x * y) - np.sum(x) * np.sum(y)) / (n * np.sum(x**2) - (np.sum(x))**2)
w_0 = np.mean(y) - w_1 * np.mean(x)

print(w_1)
# 2.4
print(w_0)
# 0.8000000000000007

plt.scatter(x, y)

line_y = w_1 * x + w_0
plt.plot(x, line_y, color='red')

plt.show()
```



2. Метод обратных матриц

уравнение прямой :

$$y = w_1 x + w_0$$

Вектор коэффициентов:

$$\vec{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

Матрица признаков (X) :

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}$$

Вектор целевых значений (Y) :

$$\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

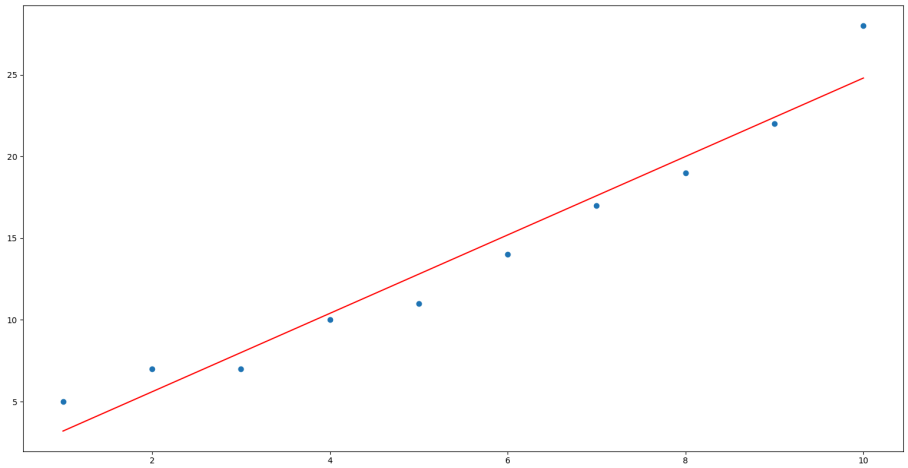
Тогда наше решение будет иметь вид:

$$\vec{w} = \left(X^T X\right)^{-1} X^T y$$

Практическая реализация

```
import numpy as np
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from numpy.linalg import inv

data = np.array(
   [
      [1, 5], [2, 7], [3, 7], [4, 10], [5, 11],
      [6, 14], [7, 17], [8, 19], [9, 22], [10, 28],
   ]
)

x = data[:, 0]
y = data[:, 1]

x_1 = np.vstack([x, np.ones([len(x)])]).T
w = inv(x_1.transpose() @ x_1) @ (x_1.transpose() @ y)
```

```
print(w)
# [2.4 0.8]

plt.scatter(x, y)

line_y = w[0] * x + w[1]
plt.plot(x, line_y, color='red')

plt.show()
```



3. Метод разложения матриц

QR - разложение матрицы:

$$X = Q \cdot R$$

Матрица признаков (X) :

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}$$

Ортогональная матрица (Q) :

$$Q = \begin{bmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \\ \vdots & \vdots \\ q_{n1} & q_{n2} \end{bmatrix}$$

Верхнетруегольная матрица (R) :

$$R = \begin{bmatrix} r_{11} & r_{12} \\ 0 & r_{22} \end{bmatrix}$$

Решение для векторных коэффициентов :

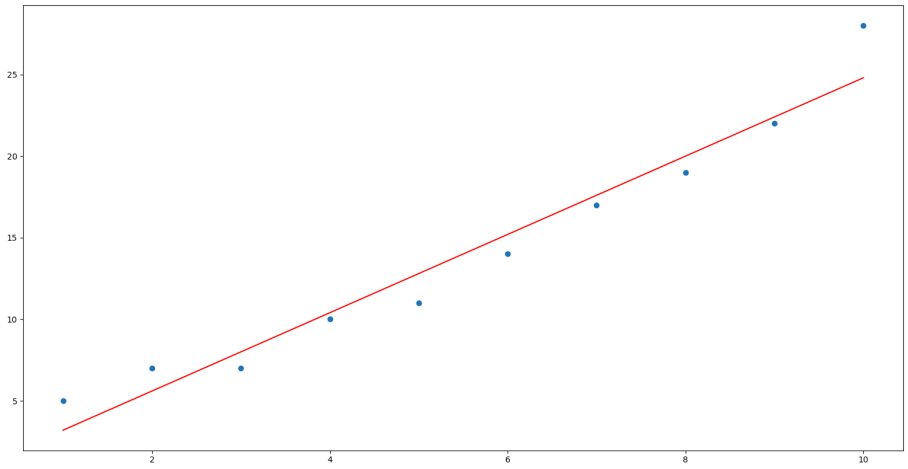
$$\vec{w} = R^{-1} \cdot Q^T \cdot y$$

Данный метод позволяет минимизировать ошибку вычислений, которая возникают при использовании компьютеров

Практическая реализация:

```
import numpy as np
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from numpy.linalg import inv, qr

data = np.array(
   [
      [1, 5], [2, 7], [3, 7], [4, 10], [5, 11],
      [6, 14], [7, 17], [8, 19], [9, 22], [10, 28],
   ]
)

x = data[:, 0]
y = data[:, 1]

x_1 = np.vstack([x, np.ones([len(x)])]).T

Q, R = qr(x_1)
w = (inv(R) @ Q.transpose()) @ y

print(w)
# [2.4 0.8]

plt.scatter(x, y)

line_y = w[0] * x + w[1]
```

```
plt.plot(x, line_y, color='red')

plt.show()
```



4. Градиентный способ

Это метод оптимизации, где используются производные и итерации.

Частные производные по одному из параметров позволяет определить угловой коэффициент и изменение параметра, выполняется в ту сторону, где он max или min.

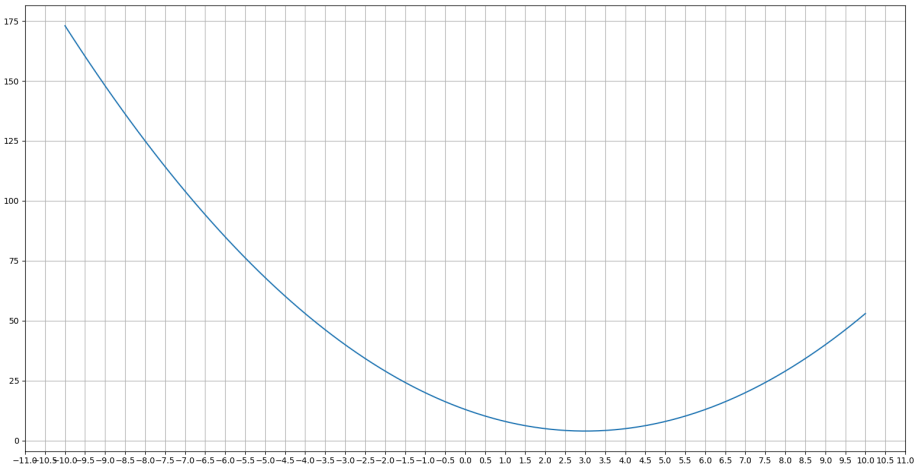
Для больших угловых коэффициентов делается более широкий шаг, для малых - менее широкий.

Ширина шага обычно вычисляется как доля от углового коэффициента и напрямую связана со скоростью обучения. Чем выше скорость, тем быстрее будет работать система, но делается это за счет снижения точности, чем ниже скорость, тем больше времени затратим на обучение, но точность будет выше.

Как это делается:

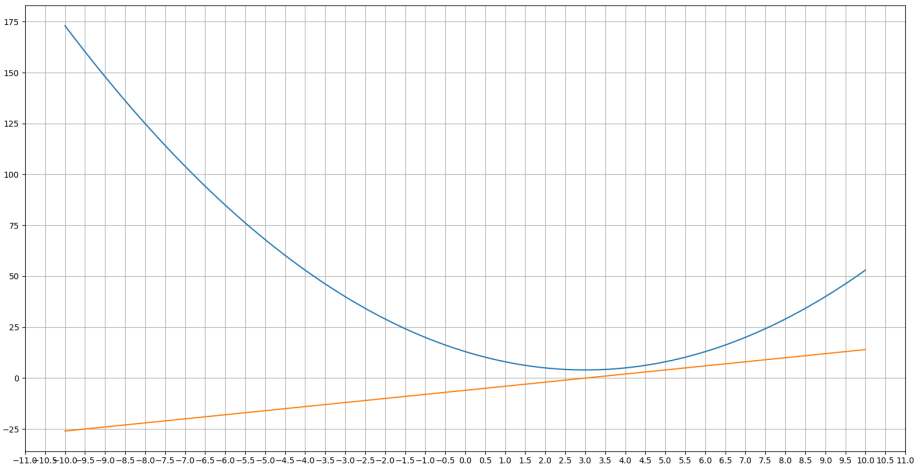
$$f(x) = (x - 3)^2 + 4$$

найти x, где $f(x)$ минимальна



Поиск производной $f'(x)$:

$$f'(x) = 2(x - 3)$$



Как ищется минимум в градиентном способе ?

```
import numpy as np
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from numpy.linalg import inv, qr
import random
```

```
def f(x):
    return (x - 3) ** 2 + 4


def dx_f(x):
    return 2 * (x - 3)


L = 0.001

iteration = 100_000
x = random.randint(0, 5)

for i in range(iteration):
    d_x = dx_f(x)
    x -= L * d_x

print(x, f(x))
# 2.999999999999889 4.0
```

Как градиентный спуск работает с линейной регрессией ?

$$y = w_0 + w_1 x$$

Задана табличная функция (x_i, y_i), тогда функция потерь:

$$E(w_1, w_0) = \sum_{i=1}^{n} R_i^2 = \sum_{i=1}^{n} (y_i - (w_0 + w_1 x))^2$$

Чтобы воспользоваться градиентным способом нужно найти частные производные:

$$\frac{d}{dw_0} E(w_1, w_0) \qquad\qquad \frac{d}{dw_1} E(w_1, w_0)$$

$$E(w_1, w_0) = \sum_{i=1}^{n} (y_i^2 - 2y_i(w_0 + w_1 x) - (w_0 + w_1 x)^2) =$$
$$= \sum_{i=1}^{n} (y_i^2 - 2y_i w_0 - 2y_i w_1 x - w_0^2 - 2w_0 w_1 x - w_1^2 x^2)$$

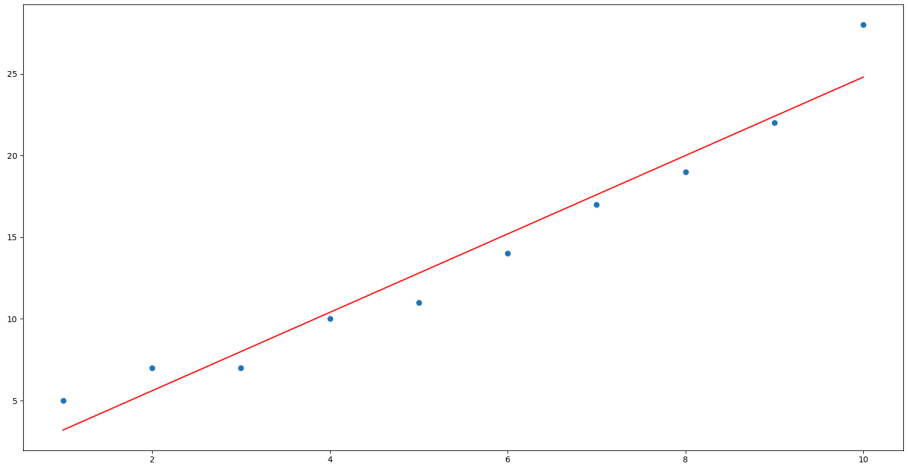
$$1) \quad \frac{d}{dw_0} E(w_1, w_0) = \sum_{i=1}^{n} (2y_i - 2w_0 - 2w_1 x) = 2 \sum_{i=1}^{n} (y_i - w_0 - w_1 x)$$

$$2)\ \ \frac{d}{dw_1}E(w_1, w_0) = \sum_{i=1}^{n}(-2y_i x - 2w_0 x - 2w_1 x^2) = 2\sum_{i=1}^{n} x_i(-y_i - w_0 - w_1 x)$$

Практическая реализация

```
import numpy as np
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from numpy.linalg import inv, qr
import random


data = np.array(
    [
        [1, 5], [2, 7], [3, 7], [4, 10], [5, 11],
        [6, 14], [7, 17], [8, 19], [9, 22], [10, 28],
    ]
)

x = data[:, 0]
y = data[:, 1]
n = len(x)

w1 = 0.0
w0 = 0.0

L = 0.001
iteration = 100_000

for i in range(iteration):
    D_w0 = -2 * sum((y[i] - (w0 + w1 * x[i])) for i in range(n))
    D_w1 = -2 * sum(x[i] * (y[i] - (w0 + w1 * x[i])) for i in range(n))

    w0 -= L * D_w0
    w1 -= L * D_w1

print(w0, w1)
# 0.7999999999999812 2.400000000000003

plt.scatter(x, y)

line_y = w1 * x + w0
```

```
plt.plot(x, line_y, color='red')
plt.show()
```



Трёхмерный график поверхности ошибки

```
import numpy as np
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from numpy.linalg import inv, qr
import random


def E(w1, w0, x, y):
    return sum((y[i] - (w0 + w1 * x[i])) ** 2 for i in range(len(x)))


data = np.array(
    [
        [1, 5], [2, 7], [3, 7], [4, 10], [5, 11],
        [6, 14], [7, 17], [8, 19], [9, 22], [10, 28],
    ]
)

x = data[:, 0]
y = data[:, 1]
n = len(x)
```

```
w1 = np.linspace(-10, 10, 100)
w0 = np.linspace(-10, 10, 100)

W1, W0 = np.meshgrid(w1, w0)

EW = E(W1, W0, x, y)

fig = plt.figure()
ax = plt.axes(projection="3d")

ax.plot_surface(W1, W0, EW)

w1_fit = 2.4
w0_fit = 0.8
E_fit = E(w1_fit, w0_fit, x, y)
ax.scatter3D(w1_fit, w0_fit, E_fit, color="red")

plt.show()
```