Задание 1.
Исходные данные:
Подберите скорость обучения (alpha) и количество итераций
Решение:

```
Python 3.8.10 (default, Sep 28 2021, 16:10:42)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>> X = np.array([ [1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 2, 5, 3, 0, 5, 10, 1, 2] ])
>>> X
array([[ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1],
       [ 1,  1,  2,  5,  3,  0,  5, 10,  1,  2]])
>>> X.shape
(2, 10)
>>> y = [45, 55, 50, 55, 60, 35, 75, 80, 50, 60]
>>> y_pred1 = 35 * np.ones(10) + X[1]*5
>>> y_pred2 = 40 * np.ones(10) + X[1]*7.5
>>> y_pred1, y_pred2
(array([40., 40., 45., 60., 50., 35., 60., 85., 40., 45.]), array([ 47.5,  47.5,  55. ,  77.5,  62.5,  40. ,
77.5, 115. ,  47.5,
        55. ]))
>>> err1 = np.sum(y - y_pred1)
>>> err2 = np.sum(y - y_pred2)
>>> err1, err2
(65.0, -60.0)
>>> mae_1 = np.sum(np.abs(y - y_pred1)) / 10
>>> mae_2 = np.sum(np.abs(y - y_pred2)) / 10
>>> mae_1, mae_2
(8.5, 9.0)
>>> mse_1 = np.mean((y - y_pred1) ** 2)
>>> mse_2 = np.mean((y - y_pred2) ** 2)
>>> mse_1, mse_2
(97.5, 188.75)
>>> X
array([[ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1],
       [ 1,  1,  2,  5,  3,  0,  5, 10,  1,  2]])
>>> X.shape
(2, 10)
>>> X.T
array([[ 1,  1],
       [ 1,  1],
       [ 1,  2],
       [ 1,  5],
       [ 1,  3],
       [ 1,  0],
       [ 1,  5],
       [ 1, 10],
       [ 1,  1],
       [ 1,  2]])
>>> X.T.shape
(10, 2)
>>> all(X @ y == np.dot(X, y))
```

```
True
>>> W = np.linalg.inv(np.dot(X, X.T)) @X @ y
>>> W
array([45.0625,  3.8125])
>>> y_pred3 = W[0] * X[0] + W[1] * X[1]
>>> y_pred3
array([48.875 , 48.875 , 52.6875, 64.125 , 56.5   , 45.0625, 64.125 ,
       83.1875, 48.875 , 52.6875])
>>> def calc_mae(y, y_pred):
...     err = np.mean(np.abs(y - y_pred))
...     return err
...
>>> def calc_mse(y, y_pred):
...     err = np.mean((y- y_pred) ** 2) # <=> 1/n * np.sum((y_pred - y) ** 2)
...     return err
...
>>> calc_mae(y, y_pred1), calc_mse(y, y_pred1)
(8.5, 97.5)
>>> calc_mae(y, y_pred2), calc_mse(y, y_pred2)
(9.0, 188.75)
>>> calc_mae(y, y_pred3), calc_mse(y, y_pred3)
(5.7875, 43.96874999999999)
>>> n = 10
>>> Q = 1/n * np.sum((y_pred3 - y) ** 2)
>>> alpha = 1e-2
>>> g = alpha * (1/n * 2 * np.sum(X[0] * (W[0] * X[0] - y)))
>>> W[0], W[0] - g
(45.062500000000014, 45.29125000000001)
>>> n = 20
>>> Q = 1/n * np.sum((y_pred3 - y) ** 2)
>>> alpha = 1e-2
>>> g = alpha * (1/n * 2 * np.sum(X[0] * (W[0] * X[0] - y)))
>>> W[0], W[0] - g
(1, 1.5550000000000002)
>>> n = 30
>>> Q = 1/n * np.sum((y_pred3 - y) ** 2)
>>> alpha = 1e-2
>>> g = alpha * (1/n * 2 * np.sum(X[0] * (W[0] * X[0] - y)))
>>> W[0], W[0] - g
(1, 1.37)
>>> n = 40
>>> Q = 1/n * np.sum((y_pred3 - y) ** 2)
>>> alpha = 1e-2
>>> g = alpha * (1/n * 2 * np.sum(X[0] * (W[0] * X[0] - y)))
>>> W[0], W[0] - g
(1, 1.2775)
>>> n = 50
>>> Q = 1/n * np.sum((y_pred3 - y) ** 2)
>>> alpha = 1e-2
>>> g = alpha * (1/n * 2 * np.sum(X[0] * (W[0] * X[0] - y)))
>>> W[0], W[0] - g
(1, 1.222)
```

```
>>> n = 60
>>> Q = 1/n * np.sum((y_pred3 - y) ** 2)
>>> alpha = 1e-2
>>> g = alpha * (1/n * 2 * np.sum(X[0] * (W[0] * X[0] - y)))
>>> W[0], W[0] - g
(1, 1.185)
>>> n = 70
>>> Q = 1/n * np.sum((y_pred3 - y) ** 2)
>>> alpha = 1e-2
>>> g = alpha * (1/n * 2 * np.sum(X[0] * (W[0] * X[0] - y)))
>>> W[0], W[0] - g
(1, 1.1585714285714286)
>>> n = 80
>>> Q = 1/n * np.sum((y_pred3 - y) ** 2)
>>> alpha = 1e-2
>>> g = alpha * (1/n * 2 * np.sum(X[0] * (W[0] * X[0] - y)))
>>> W[0], W[0] - g
(1, 1.13875)
>>> n = 90
>>> Q = 1/n * np.sum((y_pred3 - y) ** 2)
>>> alpha = 1e-2
>>> g = alpha * (1/n * 2 * np.sum(X[0] * (W[0] * X[0] - y)))
>>> W[0], W[0] - g
(1, 1.1233333333333333)
>>>
```

Задание 2.
Исходные данные:
В этом коде мы избавляемся от итераций по весам, но тут есть ошибка, исправьте ее
Решение:

```
Python 3.8.10 (default, Sep 28 2021, 16:10:42)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>> X = np.array([ [1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 2, 5, 3, 0, 5, 10, 1, 2] ])
>>> X
array([[ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1],
       [ 1,  1,  2,  5,  3,  0,  5, 10,  1,  2]])
>>> X.shape
(2, 10)
>>> y = [45, 55, 50, 55, 60, 35, 75, 80, 50, 60]
>>> y_pred1 = 35 * np.ones(10) + X[1]*5
>>> y_pred2 = 40 * np.ones(10) + X[1]*7.5
>>> y_pred1, y_pred2
(array([40., 40., 45., 60., 50., 35., 60., 85., 40., 45.]), array([ 47.5,  47.5,  55. ,  77.5,  62.5,  40. ,
77.5, 115. ,  47.5,
       55. ]))
>>> err1 = np.sum(y - y_pred1)
>>> err2 = np.sum(y - y_pred2)
>>> err1, err2
(65.0, -60.0)
>>> mae_1 = np.sum(np.abs(y - y_pred1)) / 10
```

```
>>> mae_2 = np.sum(np.abs(y - y_pred2)) / 10
>>> mae_1, mae_2
(8.5, 9.0)
>>> mse_1 = np.mean((y - y_pred1) ** 2)
>>> mse_2 = np.mean((y - y_pred2) ** 2)
>>> mse_1, mse_2
(97.5, 188.75)
>>> X
array([[ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1],
       [ 1,  1,  2,  5,  3,  0,  5, 10,  1,  2]])
>>> X.shape
(2, 10)
>>> X.T
array([[ 1,  1],
       [ 1,  1],
       [ 1,  2],
       [ 1,  5],
       [ 1,  3],
       [ 1,  0],
       [ 1,  5],
       [ 1, 10],
       [ 1,  1],
       [ 1,  2]])
>>> X.T.shape
(10, 2)
>>> all(X @ y == np.dot(X, y))
True
>>> W = np.linalg.inv(np.dot(X, X.T)) @X @ y
>>> W
array([45.0625,  3.8125])
>>> y_pred3 = W[0] * X[0] + W[1] * X[1]
>>> y_pred3
array([48.875 , 48.875 , 52.6875, 64.125 , 56.5   , 45.0625, 64.125 ,
       83.1875, 48.875 , 52.6875])
>>> def calc_mae(y, y_pred):
...     err = np.mean(np.abs(y - y_pred))
...     return err
...
>>> def calc_mse(y, y_pred):
...     err = np.mean((y- y_pred) ** 2) # <=> 1/n * np.sum((y_pred - y) ** 2)
...     return err
...
>>> calc_mae(y, y_pred1), calc_mse(y, y_pred1)
(8.5, 97.5)
>>> calc_mae(y, y_pred2), calc_mse(y, y_pred2)
(9.0, 188.75)
>>> calc_mae(y, y_pred3), calc_mse(y, y_pred3)
(5.7875, 43.96874999999999)
>>> n = X.shape[1]
>>> alpha = 1e-2
>>> W = np.array([1, 0.5])
>>> W
```

```
array([1. , 0.5])
>>> N = 10
>>> L = 0.01
>>> I = [1, 0.5]
>>> print(f'N = {n}, L = {alpha}, I = {W}')
N = 10, L = 0.01, I = [1.  0.5]
>>> for i in range(100):
...     y_pred = np.dot(W, X)
...     err = calc_mse(y, y_pred)
...     for k in range(len(W)):
...         W[k] -= alpha * (1/n * 2 * np.sum(X[k] * (y_pred - y)))
...     if i % 10 == 0:
...         alpha /= 1.1
...         print(f'I #{i}: W_mew = {W}, MSE = {round(err,2)}')
...
I #0: W_mew = [2.08 4.27], MSE = 3047.75
I #10: W_mew = [ 6.67106886 10.61676385], MSE = 749.71
I #20: W_mew = [ 9.49320908 10.25731657], MSE = 648.91
I #30: W_mew = [11.85740092  9.83349244], MSE = 570.46
I #40: W_mew = [13.86876921  9.46898661], MSE = 508.03
I #50: W_mew = [15.59085668  9.15672679], MSE = 457.73
I #60: W_mew = [17.07337653  8.88789585], MSE = 416.77
I #70: W_mew = [18.35601294  8.65530964], MSE = 383.06
I #80: W_mew = [19.47073522  8.45317196], MSE = 355.08
I #90: W_mew = [20.44350656  8.27677488], MSE = 331.65
>>>
```

Задание 3.

Исходные данные:

Вместо того, чтобы задавать количество итераций, задайте условие остановки алгоритма - когда ошибка за итерацию начинает изменяться ниже определенного порога (упрощенный аналог параметра tol в линейной регрессии в sklearn).

Решение:

```
Python 3.8.10 (default, Sep 28 2021, 16:10:42)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>> X = np.array([ [1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 2, 5, 3, 0, 5, 10, 1, 2] ])
>>> X
array([[ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1],
       [ 1,  1,  2,  5,  3,  0,  5, 10,  1,  2]])
>>> X.shape
(2, 10)
>>> y = [45, 55, 50, 55, 60, 35, 75, 80, 50, 60]
>>> y_pred1 = 35 * np.ones(10) + X[1]*5
>>> y_pred2 = 40 * np.ones(10) + X[1]*7.5
>>> y_pred1, y_pred2
(array([40., 40., 45., 60., 50., 35., 60., 85., 40., 45.]), array([ 47.5,  47.5,  55. ,  77.5,  62.5,  40. ,
77.5, 115. ,  47.5,
       55. ]))
>>> err1 = np.sum(y - y_pred1)
>>> err2 = np.sum(y - y_pred2)
```

```
>>> err1, err2
(65.0, -60.0)
>>> mae_1 = np.sum(np.abs(y - y_pred1)) / 10
>>> mae_2 = np.sum(np.abs(y - y_pred2)) / 10
>>> mae_1, mae_2
(8.5, 9.0)
>>> mse_1 = np.mean((y - y_pred1) ** 2)
>>> mse_2 = np.mean((y - y_pred2) ** 2)
>>> mse_1, mse_2
(97.5, 188.75)
>>> X
array([[ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1],
       [ 1,  1,  2,  5,  3,  0,  5, 10,  1,  2]])
>>> X.shape
(2, 10)
>>> X.T
array([[ 1,  1],
       [ 1,  1],
       [ 1,  2],
       [ 1,  5],
       [ 1,  3],
       [ 1,  0],
       [ 1,  5],
       [ 1, 10],
       [ 1,  1],
       [ 1,  2]])
>>> X.T.shape
(10, 2)
>>> all(X @ y == np.dot(X, y))
True
>>> W = np.linalg.inv(np.dot(X, X.T)) @X @ y
>>> W
array([45.0625,  3.8125])
>>> y_pred3 = W[0] * X[0] + W[1] * X[1]
>>> y_pred3
array([48.875 , 48.875 , 52.6875, 64.125 , 56.5   , 45.0625, 64.125 ,
       83.1875, 48.875 , 52.6875])
>>> def calc_mae(y, y_pred):
...     err = np.mean(np.abs(y - y_pred))
...     return err
...
>>> def calc_mse(y, y_pred):
...     err = np.mean((y- y_pred) ** 2) # <=> 1/n * np.sum((y_pred - y) ** 2)
...     return err
...
>>> calc_mae(y, y_pred1), calc_mse(y, y_pred1)
(8.5, 97.5)
>>> calc_mae(y, y_pred2), calc_mse(y, y_pred2)
(9.0, 188.75)
>>> calc_mae(y, y_pred3), calc_mse(y, y_pred3)
(5.7875, 43.96874999999999)
>>> n = X.shape[1]
```

```
>>> alpha = 1e-2
>>> W = np.array([1, 0.5])
>>> W
array([1. , 0.5])
>>> N = 10
>>> L = 0.01
>>> I = [1, 0.5]
>>> print(f'N = {n}, L = {alpha}, I = {W}')
N = 10, L = 0.01, I = [1.  0.5]
>>> for i in range(100):
...     y_pred = np.dot(W, X)
...     err = calc_mse(y, y_pred)
...     for k in range(W.shape[0]):
...         W[k] -= alpha * (1/n * 2 * np.sum(X[k] * (y_pred - y)))
...     W -= alpha * (1/n * 2 * np.sum(X * (y_pred -y)))
...     W_pred = W
...     if i % 10 == 0:
...         print(f'I #{i}: W_new = {W},MSE = {round(err,2)}')
...
I #0: W_new = [6.93 9.12],MSE = 3047.75
I #10: W_new = [12.13868515 10.14042547],MSE = 570.77
I #20: W_new = [15.7611049   9.44419993],MSE = 461.23
I #30: W_new = [18.98497043  8.82457608],MSE = 374.46
I #40: W_new = [21.8541324   8.27312591],MSE = 305.74
I #50: W_new = [24.40761684  7.78234866],MSE = 251.3
I #60: W_new = [26.68015593  7.34556883],MSE = 208.19
I #70: W_new = [28.7026605  6.9568454],MSE = 174.04
I #80: W_new = [30.50264041  6.610891  ],MSE = 146.99
I #90: W_new = [32.10457879  6.30299998],MSE = 125.57
>>>
```

Сравнивая 2 и 3 задание по скорости выполнения и получения значений, то при анализе этих двух алгоритмов, мы замечаем, что алгоритм в задании 2 отрабатывает быстрее чем алгоритм в третьем задании.