

## Домашние задание №2

In [1]:

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 plt.rcParams.update({'font.size': 14})
```

In [2]:

```
1 def calc_mse(y, y_pred):
2     err = np.mean((y - y_pred)**2)
3     return err
```

In [3]:

```
1
2 X = np.array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
3               [1, 1, 2, 1, 3, 0, 5, 10, 1, 2], # стаж
4               [500, 700, 750, 600, 1450,          # средняя стоимость занятия
5               800, 1500, 2000, 450, 1000],
6               [1, 1, 2, 1, 2, 1, 3, 3, 1, 2]]) # квалификация репетитора
7
8 y = [45, 55, 50, 59, 65, 35, 75, 80, 50, 60] # средний балл ЕГЭ (целевая переменная)
9 n = X.shape[1]
```

**1. Постройте график зависимости весов всех признаков от  $\lambda$  в L2-регуляризации (на данных из урока).**

In [4]:

```
1 # Стандартизируем признаки
2 X_st = X.copy()
3 X_st = X_st.astype(np.float64)
4
5 for i in range(1, 4):
6     X_st[i] = (X_st[i] - X_st[i].mean()) / X_st[i].std()
7
8 # Напишем функцию градиентного спуска с L2 регуляризацией
9 def eval_model_reg2(X, y, iterations, lambda_, alpha=1e-4):
10     np.random.seed(42)
11     W = np.random.randn(X.shape[0])
12     n = X.shape[1]
13     for i in range(1, iterations + 1):
14         y_pred = np.dot(W, X)
15         err = calc_mse(y, y_pred)
16         W -= alpha * (1/n * 2 * np.dot((y_pred - y), X.T) + 2 * lambda_ * W)
17     return W
```

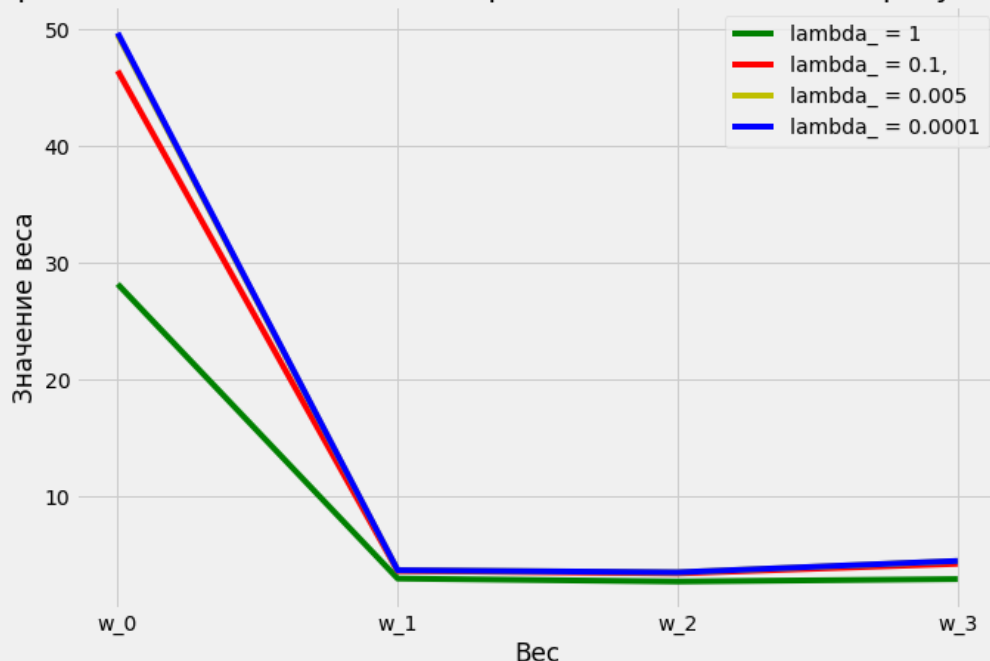
In [5]:

```
1 # Получим коэффициенты при разных Lambda
2 W_all = []
3 for i in (1, 0.1, 0.005, 0.0001):
4     W_all.append(list(eval_model_reg2(X_st, y, iterations=1000, alpha=1e-3, lambda_ = i)
5 W_all_2 = pd.DataFrame(W_all, columns = ("w_0", "w_1", "w_2", "w_3"))
```

In [6]:

```
1 # Нарисуем график зависимости весов всех признаков от Lambda в L2-регуляризации (на дан
2 plt.style.use('fivethirtyeight')
3 plt.figure(figsize=(10, 7))
4 color = ["g", "r", "y", "b"]
5 label = ["lambda_ = 1", "lambda_ = 0.1,", "lambda_ = 0.005", "lambda_ = 0.0001"]
6 for i in range(4):
7     plt.plot(W_all_2.loc[i], color = color[i], label=label[i])
8     plt.xlabel('Вес')
9     plt.ylabel('Значение веса')
10    plt.title("График зависимости весов всех признаков от lambda в L2-регуляризации")
11    plt.legend()
```

График зависимости весов всех признаков от lambda в L2-регуляризации



**2. Можно ли к одному и тому же признаку применить сразу и нормализацию, и стандартизацию?**

In [7]:

```
1 n = X.shape[1]
2 X = np.array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
3               [1, 1, 2, 1, 3, 0, 5, 10, 1, 2], # стаж
4               [500, 700, 750, 600, 1450, # средняя стоимость занятия
5               800, 1500, 2000, 450, 1000]])
6
7 y = [45, 55, 50, 59, 65, 35, 75, 80, 50, 60] # средний балл ЕГЭ (целевая переменная,
```

Посмотрим на коэффициенты и MSE до масштабирования

In [8]:

```
1 w = np.linalg.inv(X @ X.T) @ X @ y
2 w
```

Out[8]:

```
array([4.25569284e+01, 2.62631471e+00, 8.22015726e-03])
```

In [9]:

```
1 y_pred = np.dot(w, X)
2 calc_mse(y, y_pred)
```

Out[9]:

```
42.983380986418894
```

Попробуем последовательно применить сначала нормализацию, а затем стандартизацию. Потом, наоборот и посмотрим, что будет происходить с данными, коэффициентами и ошибкой обучения.

### Проведем нормализацию исходных признаков

In [10]:

```
1 X_norm = X.copy()
2 X_norm = X_norm.astype(np.float64)
3 X_norm
```

Out[10]:

```
array([[1.00e+00, 1.00e+00, 1.00e+00, 1.00e+00, 1.00e+00, 1.00e+00,
        1.00e+00, 1.00e+00, 1.00e+00, 1.00e+00],
       [1.00e+00, 1.00e+00, 2.00e+00, 1.00e+00, 3.00e+00, 0.00e+00,
        5.00e+00, 1.00e+01, 1.00e+00, 2.00e+00],
       [5.00e+02, 7.00e+02, 7.50e+02, 6.00e+02, 1.45e+03, 8.00e+02,
        1.50e+03, 2.00e+03, 4.50e+02, 1.00e+03]])
```

In [11]:

```
1 X_norm[1] = (X_norm[1] - X_norm[1].min()) / (X_norm[1].max() - X_norm[1].min())
2 X_norm[1]
```

Out[11]:

```
array([0.1, 0.1, 0.2, 0.1, 0.3, 0. , 0.5, 1. , 0.1, 0.2])
```

In [12]:

```
1 X_norm[2] = (X_norm[2] - X_norm[2].min()) / (X_norm[2].max() - X_norm[2].min())
2 X_norm[2]
```

Out[12]:

```
array([0.03225806, 0.16129032, 0.19354839, 0.09677419, 0.64516129,
        0.22580645, 0.67741935, 1. , 0. , 0.35483871])
```

In [13]:

```
1 X_norm[2] = (X_norm[2] - X_norm[2].min()) / (X_norm[2].max() - X_norm[2].min())
2 X_norm[2]
```

Out[13]:

```
array([0.03225806, 0.16129032, 0.19354839, 0.09677419, 0.64516129,
       0.22580645, 0.67741935, 1.          , 0.          , 0.35483871])
```

In [14]:

```
1 X_norm[1] = (X_norm[1] - X_norm[1].min()) / (X_norm[1].max() - X_norm[1].min())
2 X_norm[1]
```

Out[14]:

```
array([0.1, 0.1, 0.2, 0.1, 0.3, 0. , 0.5, 1. , 0.1, 0.2])
```

In [15]:

```
1 w_norm = np.linalg.inv(X_norm @ X_norm.T) @ X_norm @ y
2 print(f"Коэффициенты после нормализации исходных признаков: {w_norm}")
3 y_pred = np.dot(w_norm, X_norm)
4 print(f"MSE после нормализации исходных признаков: {calc_mse(y, y_pred)}")
```

Коэффициенты после нормализации исходных признаков: [46.25599918 26.26314715 1  
2.74124375]

MSE после нормализации исходных признаков: 42.98338098641885

## Проведем стандартизацию нормализованных признаков

In [16]:

```
1 X_norm_st = X_norm.copy().astype(np.float64)
2 X_norm_st[1] = (X_norm_st[1] - X_norm_st[1].mean()) / X_norm_st[1].std()
3 X_norm_st[1]
```

Out[16]:

```
array([-0.57142857, -0.57142857, -0.21428571, -0.57142857,  0.14285714,
       -0.92857143,  0.85714286,  2.64285714, -0.57142857, -0.21428571])
```

In [17]:

```
1 X_norm_st[2] = (X_norm_st[2] - X_norm_st[2].mean()) / X_norm_st[2].std()
2 X_norm_st[2]
```

Out[17]:

```
array([-0.97958969, -0.56713087, -0.46401617, -0.77336028,  0.97958969,
       -0.36090146,  1.08270439,  2.11385144, -1.08270439,  0.05155735])
```

In [18]:

```
1 w_norm_st = np.linalg.inv(X_norm_st @ X_norm_st.T) @ X_norm_st @ y
2 print(f"Коэффициенты после стандартизации нормализованных признаков: {w_norm_st}")
3 y_pred = np.dot(w_norm_st, X_norm_st)
4 print(f"MSE после стандартизации нормализованных признаков: {calc_mse(y, y_pred)}")
```

Коэффициенты после стандартизации нормализованных признаков: [57.4  
7.3536812 3.98592874]

MSE после стандартизации нормализованных признаков: 42.98338098641888

Теперь проведем масштабирование в обратном порядке

Проведем стандартизацию исходных признаков

In [19]:

```
1 X_st = X.copy()
2 X_st = X_st.astype(np.float64)
3 X_st
```

Out[19]:

```
array([[1.00e+00, 1.00e+00, 1.00e+00, 1.00e+00, 1.00e+00, 1.00e+00,
        1.00e+00, 1.00e+00, 1.00e+00, 1.00e+00],
       [1.00e+00, 1.00e+00, 2.00e+00, 1.00e+00, 3.00e+00, 0.00e+00,
        5.00e+00, 1.00e+01, 1.00e+00, 2.00e+00],
       [5.00e+02, 7.00e+02, 7.50e+02, 6.00e+02, 1.45e+03, 8.00e+02,
        1.50e+03, 2.00e+03, 4.50e+02, 1.00e+03]])
```

In [20]:

```
1 X_st[1] = (X_st[1] - X_st[1].mean()) / X_st[1].std()
2 X_st[1]
```

Out[20]:

```
array([-0.57142857, -0.57142857, -0.21428571, -0.57142857, 0.14285714,
       -0.92857143, 0.85714286, 2.64285714, -0.57142857, -0.21428571])
```

In [21]:

```
1 X_st[2] = (X_st[2] - X_st[2].mean()) / X_st[2].std()
2 X_st[2]
```

Out[21]:

```
array([-0.97958969, -0.56713087, -0.46401617, -0.77336028, 0.97958969,
       -0.36090146, 1.08270439, 2.11385144, -1.08270439, 0.05155735])
```

In [22]:

```
1 w_st = np.linalg.inv(X_st @ X_st.T) @ X_st @ y
2 print(f"Коэффициенты после стандартизации исходных признаков: {w_st}")
3 y_pred = np.dot(w_st, X_st)
4 print(f"MSE после w_norm_st: {calc_mse(y, y_pred)}")
```

Коэффициенты после стандартизации исходных признаков: [57.4 7.3536812 3.98592874]

MSE после w\_norm\_st: 42.98338098641888

**Проведем нормализацию стандартизированных признаков**

In [23]:

```
1 X_st_norm = X_st.copy()
2 X_st_norm
```

Out[23]:

```
array([[ 1.          ,  1.          ,  1.          ,  1.          ,  1.          ,
         1.          ,  1.          ,  1.          ,  1.          ,  1.          ],
       [-0.57142857, -0.57142857, -0.21428571, -0.57142857,  0.14285714,
        -0.92857143,  0.85714286,  2.64285714, -0.57142857, -0.21428571],
       [-0.97958969, -0.56713087, -0.46401617, -0.77336028,  0.97958969,
        -0.36090146,  1.08270439,  2.11385144, -1.08270439,  0.05155735]])
```

In [24]:

```
1 X_st_norm[1] = (X_st_norm[1] - X_st_norm[1].min()) / (X_st_norm[1].max() - X_st_norm[1].min())
2 X_st_norm[1]
```

Out[24]:

```
array([0.1, 0.1, 0.2, 0.1, 0.3, 0. , 0.5, 1. , 0.1, 0.2])
```

In [25]:

```
1 X_st_norm[2] = (X_st_norm[2] - X_st_norm[2].min()) / (X_st_norm[2].max() - X_st_norm[2].min())
2 X_st_norm[2]
```

Out[25]:

```
array([0.03225806, 0.16129032, 0.19354839, 0.09677419, 0.64516129,
        0.22580645, 0.67741935, 1.          , 0.          , 0.35483871])
```

In [26]:

```
1 w_st_norm = np.linalg.inv(X_st_norm @ X_st_norm.T) @ X_st_norm @ y
2 print(f"Коэффициенты после нормализации стандартизированных признаков: {w_st_norm}")
3 y_pred = np.dot(w_st_norm, X_st_norm)
4 print(f"MSE после нормализации стандартизированных признаков: {calc_mse(y, y_pred)}")
```

Коэффициенты после нормализации стандартизированных признаков: [46.25599918 2 6.26314715 12.74124375]

MSE после нормализации стандартизированных признаков: 42.98338098641885

Сравним коэффициенты при разных комбинациях масштабирования

In [27]:

```
1 print(f"Коэффициенты после нормализации исходных признаков: {w_norm}")
2 print(f"Коэффициенты после нормализации стандартизированных признаков: {w_st_norm}")
3 print(f"Коэффициенты после стандартизацию исходных признаков: {w_st}")
4 print(f"Коэффициенты после стандартизации нормализованных признаков: {w_norm_st}")
```

Коэффициенты после нормализации исходных признаков: [46.25599918 26.26314715 1  
2.74124375]

Коэффициенты после нормализации стандартизированных признаков: [46.25599918 2  
6.26314715 12.74124375]

Коэффициенты после стандартизацию исходных признаков: [57.4 7.3536812  
3.98592874]

Коэффициенты после стандартизации нормализованных признаков: [57.4  
7.3536812 3.98592874]

Сравним признаки при разных комбинациях масштабирования

In [28]:

```
1 print(X_norm[2])
2 print(X_st_norm[2])
3 print(X_norm_st[2])
4 print(X_st[2])
```

[0.03225806 0.16129032 0.19354839 0.09677419 0.64516129 0.22580645  
0.67741935 1. 0. 0.35483871]

[0.03225806 0.16129032 0.19354839 0.09677419 0.64516129 0.22580645  
0.67741935 1. 0. 0.35483871]

[-0.97958969 -0.56713087 -0.46401617 -0.77336028 0.97958969 -0.36090146  
1.08270439 2.11385144 -1.08270439 0.05155735]

[-0.97958969 -0.56713087 -0.46401617 -0.77336028 0.97958969 -0.36090146  
1.08270439 2.11385144 -1.08270439 0.05155735]

## Вывод

Смысла в одновременном применении нормализации и стандартизации нет, так как после нормализации стандартизированных признаков мы получаем те же признаки, что и после нормализации исходных признаков. Тот же принцип действует и в обратном случае. После стандартизации нормализованных признаков, мы получаем те же признаки, что и после стандартизации исходных. Поэтому нужно просто применять либо нормализацию либо стандартизацию.

## 3. \*Напишите функцию наподобие `eval_model_reg2`, но для применения L1-регуляризации.

In [29]:

```
1 n = X.shape[1]
2 X = np.array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
3               [1, 1, 2, 1, 3, 0, 5, 10, 1, 2], # стаж
4               [500, 700, 750, 600, 1450, # средняя стоимость занятия
5               800, 1500, 2000, 450, 1000],
6               [1, 1, 2, 1, 2, 1, 3, 3, 1, 2]]) # квалификация репетитора
7
8 y = [45, 55, 50, 59, 65, 35, 75, 80, 50, 60] # средний балл ЕГЭ (целевая переменная,
```

In [30]:

```
1 # Стандартизируем признаки
2 X_st = X.copy()
3 X_st = X_st.astype(np.float64)
4
5 for i in range(1, 4):
6     X_st[i] = (X_st[i] - X_st[i].mean()) / X_st[i].std()
7
8 # Напишем функцию градиентного спуска с L1 регуляризацией
9 def eval_model_reg1(X, y, iterations, lambda_, alpha=1e-4):
10     np.random.seed(42)
11     W = np.random.randn(X.shape[0])
12     n = X.shape[1]
13     for i in range(1, iterations + 1):
14         y_pred = np.dot(W, X)
15         err = calc_mse(y, y_pred)
16         W -= alpha * (1/n * 2 * np.dot((y_pred - y), X.T) + lambda_ * np.sign(W))
17     return W
```

In [31]:

```
1 # Посчитаем коэффициенты
2 w_L1 = eval_model_reg1(X_st, y, iterations=10000, alpha=1e-3, lambda_ = 0.001)
```

In [32]:

```
1 # Посчитаем MSE
2 y_pred = np.dot(w_L1, X_st)
3 calc_mse(y, y_pred)
```

Out[32]:

39.80711256982036

#### 4. \*Сравните на графиках изменение весов признаков от lambda в L1-регуляризации и L2-регуляризации (на данных из урока).

In [33]:

```
1 n = X.shape[1]
2 X = np.array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
3               [1, 1, 2, 1, 3, 0, 5, 10, 1, 2], # стаж
4               [500, 700, 750, 600, 1450, # средняя стоимость занятия
5               800, 1500, 2000, 450, 1000],
6               [1, 1, 2, 1, 2, 1, 3, 3, 1, 2]]) # квалификация репетитора
7
8 y = [45, 55, 50, 59, 65, 35, 75, 80, 50, 60] # средний балл ЕГЭ (целевая переменная,
```

In [34]:

```
1 # Стандартизируем признаки
2 X_st = X.copy()
3 X_st = X_st.astype(np.float64)
4
5 for i in range(1, 4):
6     X_st[i] = (X_st[i] - X_st[i].mean()) / X_st[i].std()
```



In [35]:

```
1 # Получим коэффициенты при разных Lambda
2 W_all = []
3 for i in (1, 0.1, 0.005, 0.0001):
4     W_all.append(list(eval_model_reg1(X_st, y, iterations=1000, alpha=1e-3, lambda_= i)
5 W_all_1 = pd.DataFrame(W_all, columns = ("w_0", "w_1", "w_2", "w_3"))
```

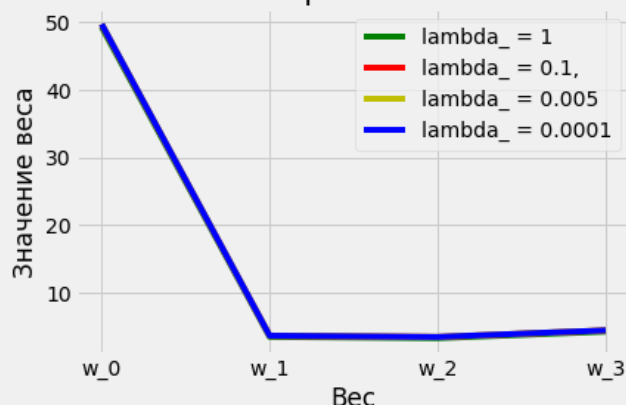
In [36]:

```
1 # Получим коэффициенты при разных Lambda
2 W_all = []
3 for i in (1, 0.1, 0.005, 0.0001):
4     W_all.append(list(eval_model_reg2(X_st, y, iterations=1000, alpha=1e-3, lambda_= i)
5 W_all_2 = pd.DataFrame(W_all, columns = ("w_0", "w_1", "w_2", "w_3"))
```

In [37]:

```
1 # Нарисуем график зависимости весов всех признаков от Lambda в L1-регуляризации (на дан
2 plt.style.use('fivethirtyeight')
3 plt.figure(figsize=(6, 4))
4 color = ["g", "r", "y", "b"]
5 label = ["lambda_ = 1", "lambda_ = 0.1,", "lambda_ = 0.005", "lambda_ = 0.0001"]
6 for i in range(4):
7     plt.plot(W_all_1.loc[i], color = color[i], label=label[i])
8     plt.xlabel('Вес')
9     plt.ylabel('Значение веса')
10    plt.title("График зависимости весов всех признаков от lambda в L1-регуляризации")
11    plt.legend()
```

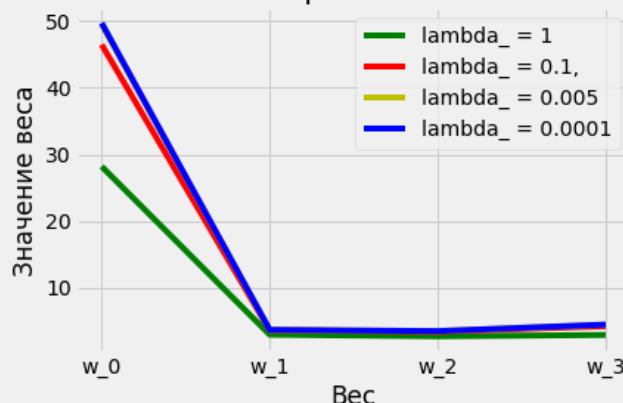
График зависимости весов всех признаков от lambda в L1-регуляризации



In [38]:

```
1 # Нарисуем график зависимости весов всех признаков от lambda в L2-регуляризации (на дан
2 plt.style.use('fivethirtyeight')
3 plt.figure(figsize=(6, 4))
4 color = ["g", "r", "y", "b"]
5 label = ["lambda_ = 1", "lambda_ = 0.1,", "lambda_ = 0.005", "lambda_ = 0.0001"]
6 for i in range(4):
7     plt.plot(W_all_2.loc[i], color = color[i], label=label[i])
8     plt.xlabel('Вес')
9     plt.ylabel('Значение веса')
10    plt.title("График зависимости весов всех признаков от lambda в L2-регуляризации")
11    plt.legend()
```

График зависимости весов всех признаков от lambda в L2-регуляризации



#### Вывод:

На существующем наборе данных мы не видим существенного отличия на графиках изменение весов признаков от lambda в L1-регуляризации, как это наблюдается при L2 регуляризации.

**5. \*Постройте графики зависимости весов двух признаков (стаж и стоимость занятия) от количества итераций для градиентного спуска и для стохастического градиентного спуска (на данных из урока).**

In [39]:

```
1 n = X.shape[1]
2 X = np.array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
3               [1, 1, 2, 1, 3, 0, 5, 10, 1, 2], # стаж
4               [500, 700, 750, 600, 1450, # средняя стоимость занятия
5               [800, 1500, 2000, 450, 1000]])
6
7 y = [45, 55, 50, 59, 65, 35, 75, 80, 50, 60] # средний балл ЕГЭ (целевая переменная,
```

In [40]:

```
1 # Стандартизируем признаки
2 X_st = X.copy()
3 X_st = X_st.astype(np.float64)
4
5 for i in range(1, 3):
6     X_st[i] = (X_st[i] - X_st[i].mean()) / X_st[i].std()
```

In [41]:

```
1 def eval_model_GD(X, y, iterations, lambda_, alpha=1e-4):
2     W_all_GD = []
3     np.random.seed(42)
4     W = np.random.randn(X.shape[0])
5     n = X.shape[1]
6     for i in range(iterations + 1):
7         y_pred = np.dot(W, X)
8         err = calc_mse(y, y_pred)
9         W -= (alpha * 1/n * 2 * np.dot(X, (np.dot(W, X) - y)))
10        W_all_GD.append(list(W[[1, 2]]))
11    #         print(i, W_all_GD)
12    return W_all_GD
```

In [42]:

```
1 # Метод стохастического градиентного спуска (mini-batch SGD)
2 def eval_model_SGD(X, y, iterations, qty_in_batch, alpha=1e-4):
3     np.random.seed(42)
4     W_all_SGD = []
5     W = np.random.randn(X.shape[0]) # начальное приближение весов
6     n = X.shape[1] # число наблюдений
7     n_batch = n // qty_in_batch
8     if n % qty_in_batch != 0:
9         n_batch += 1
10    for i in range(1, iterations + 1):
11        for b in range(n_batch):
12            start_ = qty_in_batch * b
13            end_ = qty_in_batch * (b + 1)
14            X_tmp = X[:, start_ : end_]
15            y_tmp = y[start_ : end_]
16            y_pred_tmp = np.dot(W, X_tmp)
17            err = calc_mse(y_tmp, y_pred_tmp)
18            W -= alpha * (1/n * 2 * np.dot((y_pred_tmp - y_tmp), X_tmp.T))
19            W_all_SGD.append(list(W[[1, 2]]))
20    return W_all_SGD
```

In [43]:

```
1 # Получим коэффициенты для градиентного спуска на разных итерациях
2 W_all_GD = eval_model_GD(X_st, y, iterations=100, alpha=1e-3, lambda_ = 0.001)
3 W_all_GD = pd.DataFrame(W_all_GD)
```

In [44]:

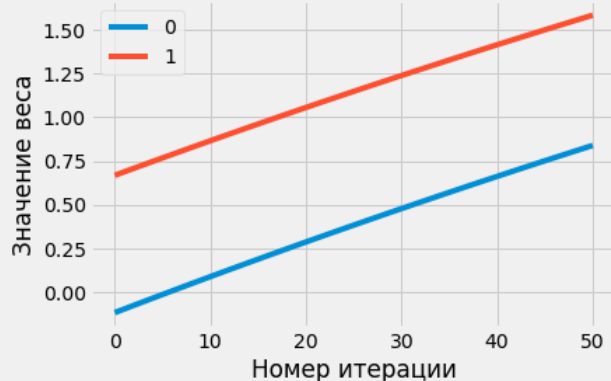
```
1 # Получим коэффициенты для стохастического градиентного спуска на разных итерациях
2 W_all_SGD = eval_model_SGD(X_st, y, iterations=100, qty_in_batch = 2, alpha=1e-3)
3 W_all_SGD = pd.DataFrame(W_all_SGD)
```

In [45]:

```
1 # Построим график для градиентного спуска на разных итерациях
2 plt.style.use('fivethirtyeight');
3 plt.figure(figsize=(6, 4));
4 W_all_GD.loc[:50].plot();
5 plt.xlabel('Номер итерации');
6 plt.ylabel('Значение веса');
7 plt.title("График зависимости весов от количества итераций для градиентного спуска");
```

<Figure size 432x288 with 0 Axes>

График зависимости весов от количества итераций для градиентного спуска



In [46]:

```
1 # Построим график для стохастического градиентного спуска на разных итерациях
2 W_all_SGD.loc[:50].plot();
3 plt.xlabel('Номер итерации')
4 plt.ylabel('Значение веса')
5 plt.title("График зависимости весов от количества итераций для стохастического градиентного спуска");
```

График зависимости весов от количества итераций для стохастического градиентного спуска



### Вывод:

Видим, что при стохастическом градиентном спуске изменения весов носят неровный характер, в отличие от простого градиентного спуска

