Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Кафедра информатики

Отчет по лабораторной работе:

Лабораторная работа №3 “Переобучение и регуляризация”

Выполнил: Карп Александр Игоревич

магистрант кафедры информатики

группа №858641

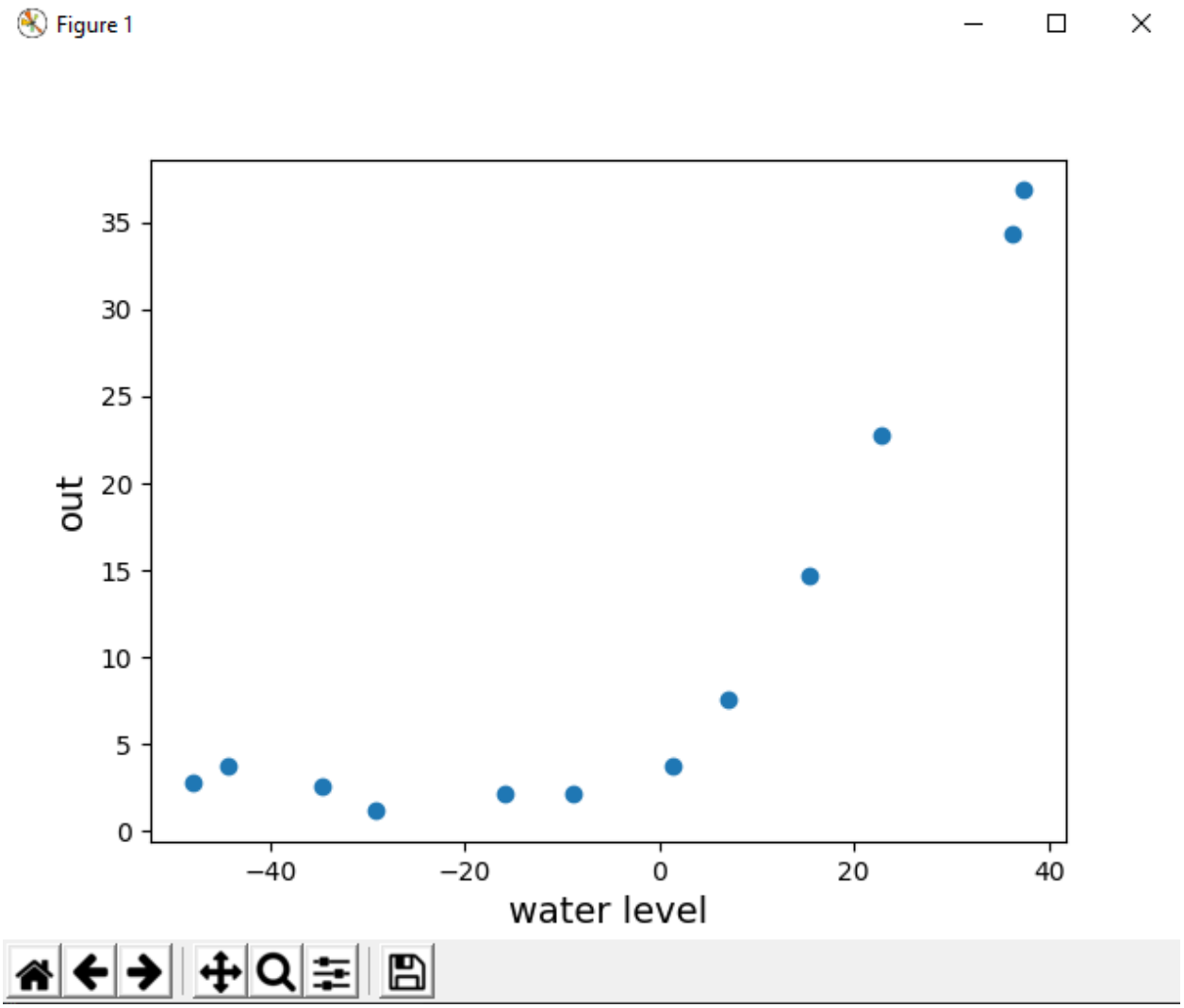
Минск 2019

1. Загрузите данные **ex3data1.mat** из файла.

```
# 1
dataset = sio.loadmat("ex3data1.mat")

x_train = pd.DataFrame(dataset["X"])
x_val = pd.DataFrame(dataset["Xval"])
x_test = pd.DataFrame(dataset["Xtest"])

y_train = dataset["y"].squeeze()
y_val = dataset["yval"].squeeze()
y_test = dataset["ytest"].squeeze()
```

2. Постройте график, где по осям откладываются X и y из обучающей выборки.

```
fig, ax = plt.subplots()
ax.scatter(x_train, y_train)
plt.xlabel("water level", fontsize=14)
plt.ylabel("out", fontsize=14)
plt.show()
```



3) Реализуйте функцию стоимости потерь для линейной регрессии с L2-регуляризацией.

```
def loss(theta, X, y, lmbd=0):
    units = np.full((len(X)), 1)  # единичный вектор
    return ((X @ theta - y) ** 2 @ units + lmbd * (np.sum(theta[1:] ** 2))) /
(len(y))
```

4) Реализуйте функцию градиентного спуска для линейной регрессии с L2-регуляризацией.

Воспользуемся алгоритмом Бройдена — Флетчера — Гольдфарба — Шанно:

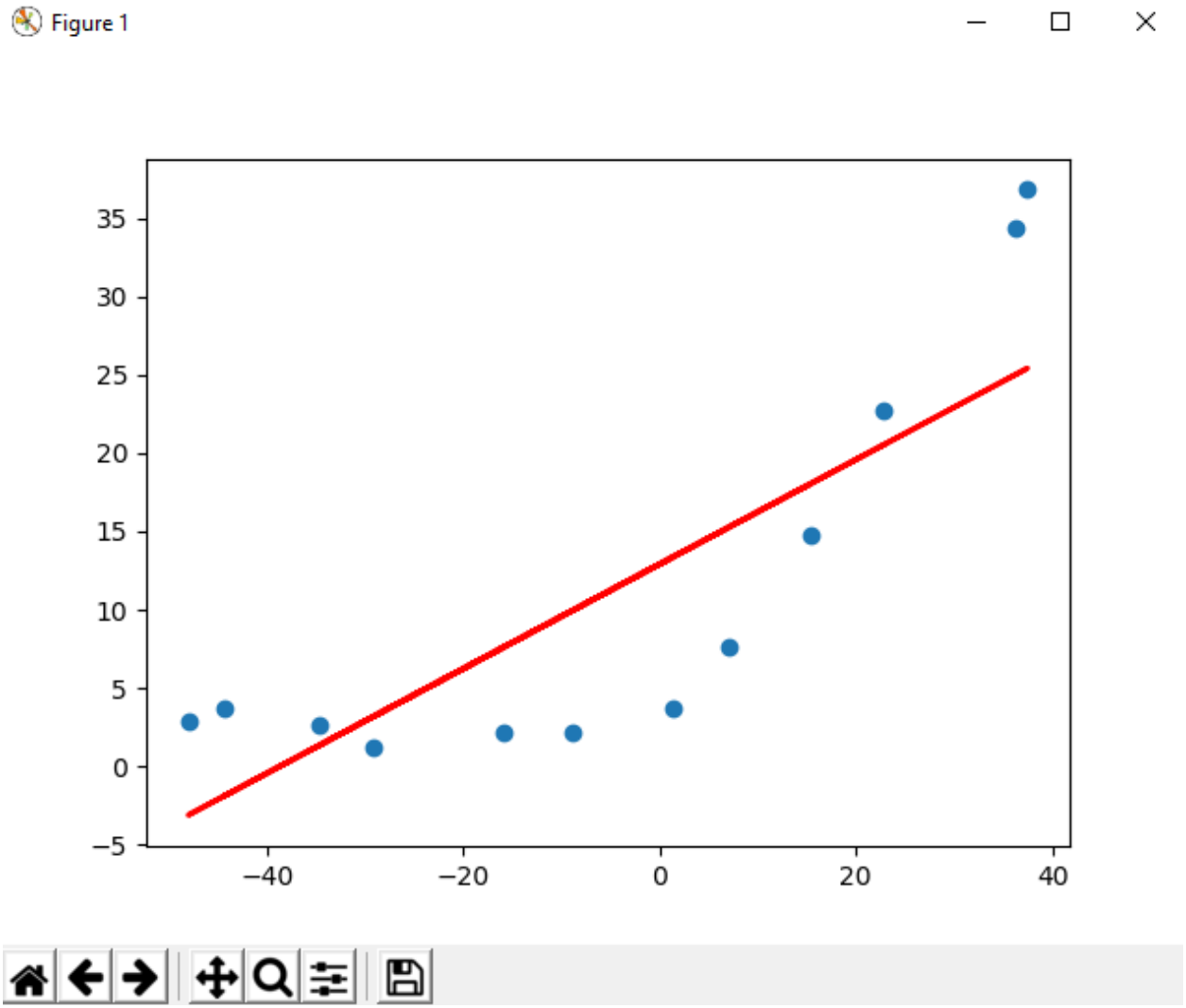
```
def gradient(theta, X, y, lmbd=0):
    diff =   np.abs(lmbd*theta)
    diff[0] = 0
    return (2 / len(y)) *( X.T @ (X @ theta - y) + diff )

theta_bfgs = optimize.fmin_bfgs(
    loss,
    theta.flatten(),
    gradient,
    (x_train_ones.values, y_train, 0)
)
```

θ: [13.08790351 0.36777923]

$Loss$: 44.74781299021783

5) Постройте модель линейной регрессии с коэффициентом регуляризации 0 и постройте график полученной функции совместно с графиком из пункта 2. Почему регуляризация в данном случае не сработает?

```
theta_bfgs = optimize.fmin_bfgs(
    loss,
    theta.flatten(),
    gradient,
    (x_train_ones.values, y_train, 0)
)
```

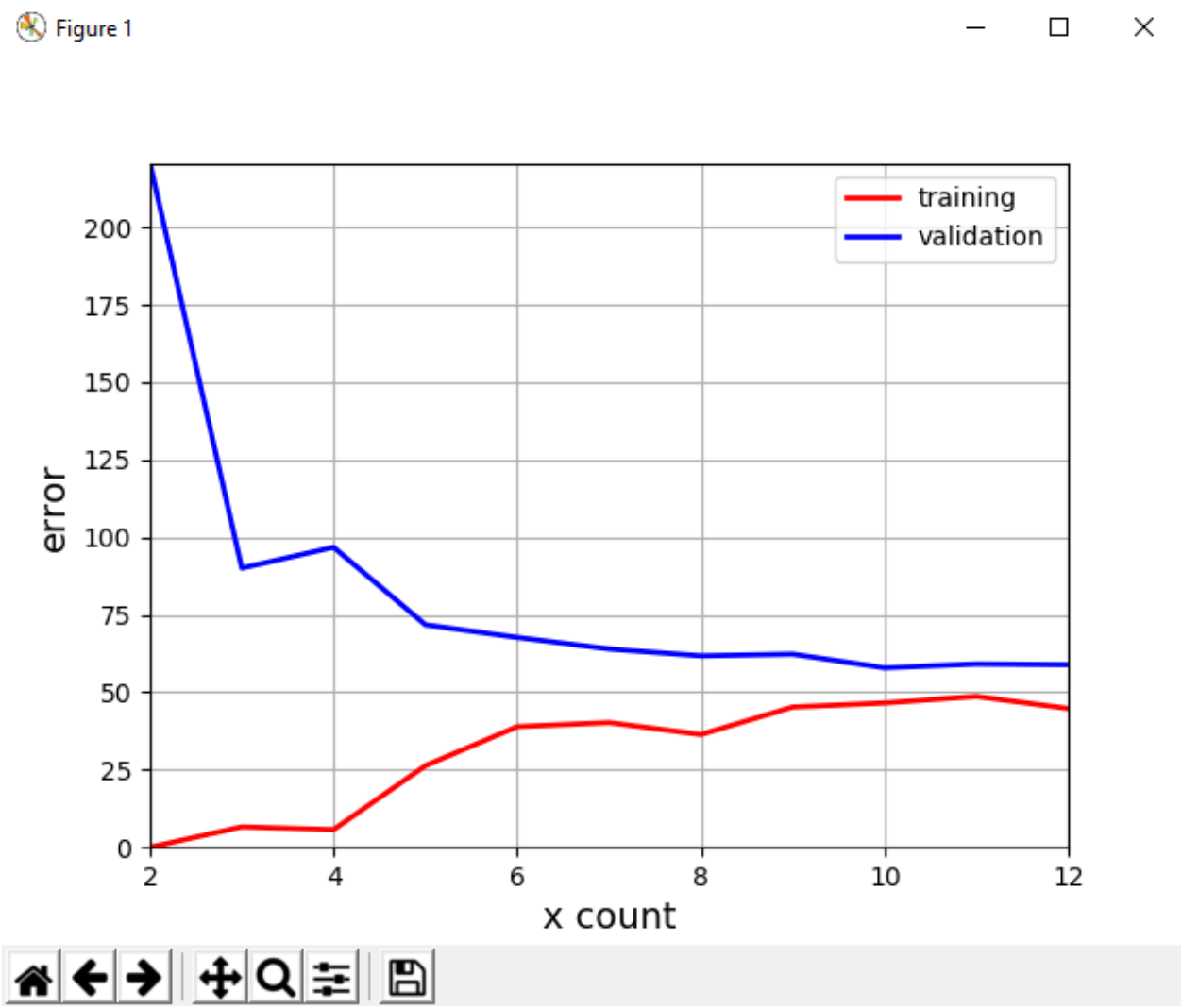


Регуляризация в данном случае не сработает в первую очереди из-за того, что наша функция линейна. Во-вторых, будь она нелинейна, регуляризация бы не сработала, так как регуляризационный коэффициент равен нулю.

6) Постройте график процесса обучения (learning curves) для обучающей и валидационной выборки. По оси абсцисс откладывается число элементов из обучающей выборки, а по оси ординат - ошибка (значение функции потерь) для обучающей выборки (первая кривая) и валидационной выборки (вторая кривая). Какой вывод можно сделать по построенному графику?

```
def learning_curves_chart(X_train, y_train, X_val, y_val, lambda_=0):
    m = len(y_train)
    train_err = np.zeros(m)
    val_err = np.zeros(m)
    theta = np.zeros(X_train.shape[1])
    for i in range(1, m):
        theta_bfgs = optimize.fmin_bfgs(
            loss,
```

```
            theta.flatten(),
            gradient,
            (X_train[0:i + 1, :], y_train[0:i + 1], lambda_)
        )
        train_err[i] = loss(theta_bfgs, X_train[0:i + 1, :], y_train[0:i +
1])
        val_err[i] = loss(theta_bfgs, X_val, y_val)
    plt.plot(range(2, m + 1), train_err[1:], c="r", linewidth=2)
    plt.plot(range(2, m + 1), val_err[1:], c="b", linewidth=2)
    plt.xlabel("x count", fontsize=14)
    plt.ylabel("error", fontsize=14)
    plt.legend(["training", "validation"])
    plt.axis([2, m, 0, max(max(val_err), max(train_err))])
    plt.grid()
    plt.show()
```



С ростом количества данных в обучающей выборке ошибка на тренировочном датасете растет, а на валидационном наоборот, уменьшается. Со временем обе ошибка стабилизируются и остаются на одном уровне. Исходные данные, судя по графику из п.2, не могут быть точно апроксимированы линейной функцией, из-за этого мы видим достаточно высокую ошибку.

7. Реализуйте функцию добавления p - 1 новых признаков в обучающую выборку (X^2, X^3, X^4, …, X^p).

```
def add_params(X_original, p=0):
    X_copy = X_original.copy()
    for i in range(2, p + 1):
        X_copy[i] = X_copy[0] ** i
    return X_copy
x_poly =add_params(x_train, 8)
```

8. Поскольку в данной задаче будет использован полином высокой степени, то необходимо перед обучением произвести нормализацию признаков.

```
def normalize(X_original):
    x_scaled = (X_original - train_means) / train_std
    df = pd.DataFrame(x_scaled)
    return df
train_means = x_poly.mean(axis=0)
train_std = np.std(x_poly, axis=0, ddof=1)
x_scaled = normalize(x_poly)
```

9. Обучите модель с коэффициентом регуляризации 0 и p = 8.

```
theta = np.zeros(x_scaled.shape[1])
theta_bfgs_scaled =  theta_bfgs = optimize.fmin_bfgs(
    loss,
    theta.flatten(),
    gradient,
    (x_scaled.values, y_train, 0)
)
```

10. Постройте график модели, совмещенный с обучающей выборкой, а также график процесса обучения. Какой вывод можно сделать в данном случае?

```
x = pd.DataFrame(np.linspace(min(x_train.values) - 5, max(x_train.values) +
5, 1000))

x_polynom = add_params(x, 8)
x_polynom = normalize(x_polynom)
x_polynom.insert(0, 'Ones', 1)

fig, ax = plt.subplots()
plt.scatter(x_train.values, y_train, color='red')
plt.plot(x, x_polynom @ theta_bfgs_scaled, linewidth=2)
plt.xlabel("water level", fontsize=14)
plt.ylabel("out", fontsize=14)
plt.show()
val = normalize(add_params(x_val, 8))
val.insert(0, 'Ones', 1)
learning_curves_chart(x_scaled.values, y_train, val.values, y_val, 0)
```

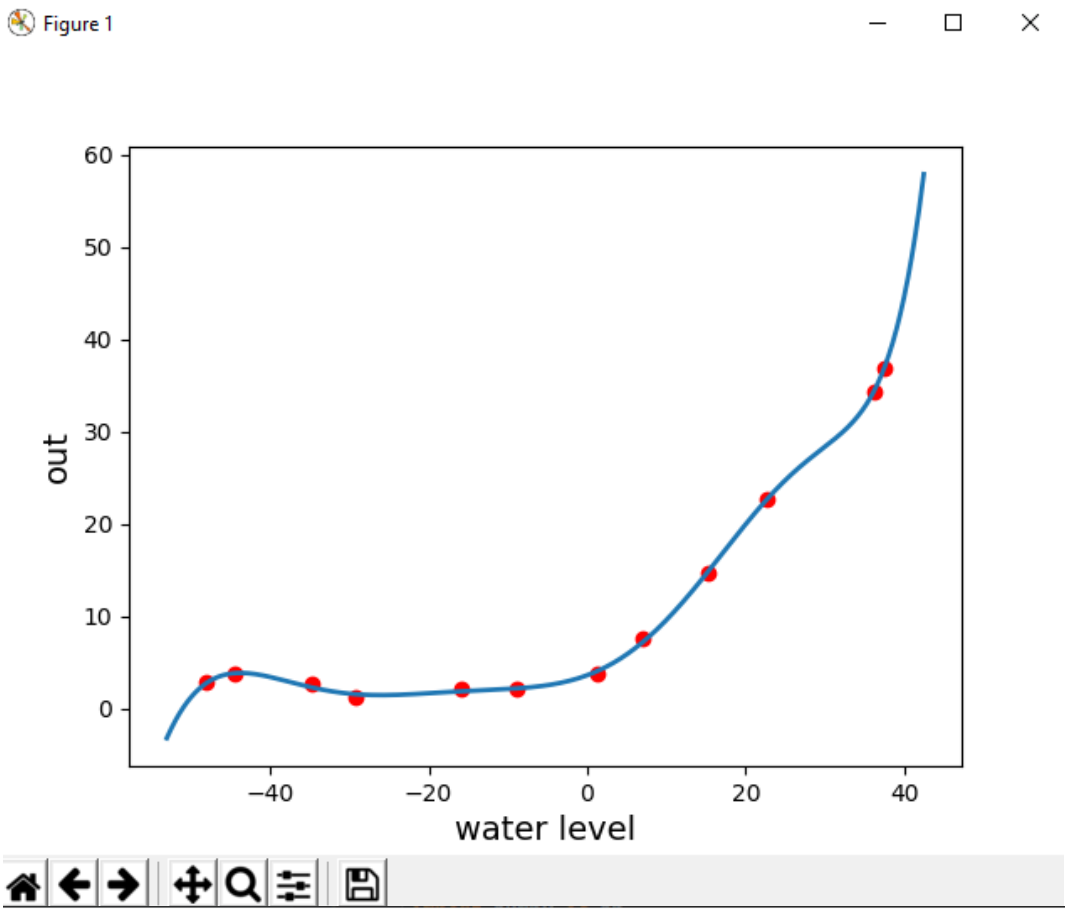


Figure 1
60
50
40
30
20
10
0
out
−40
−20
0
20
40
water level

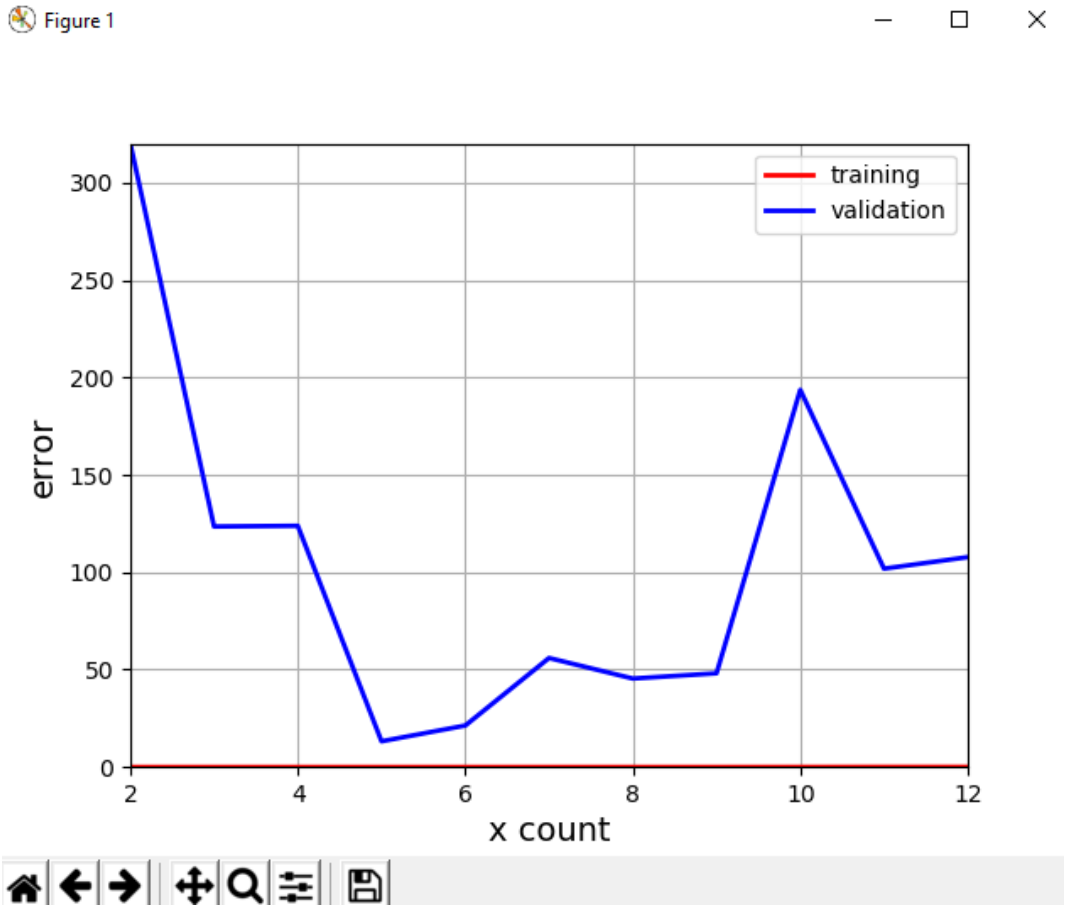


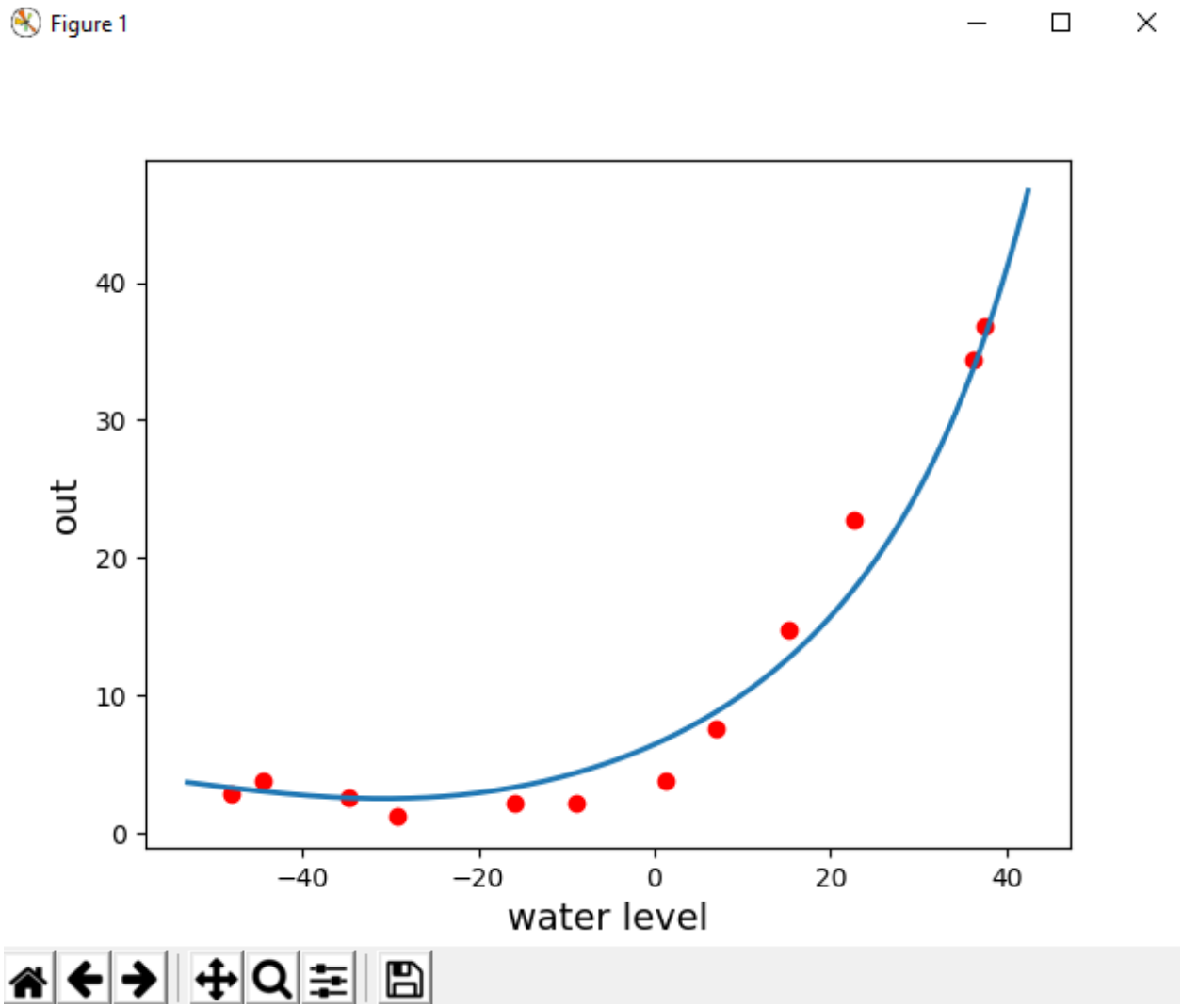
Figure 1
training
validation
300
250
200
150
100
50
0
error
2
4
6
8
10
12
x count

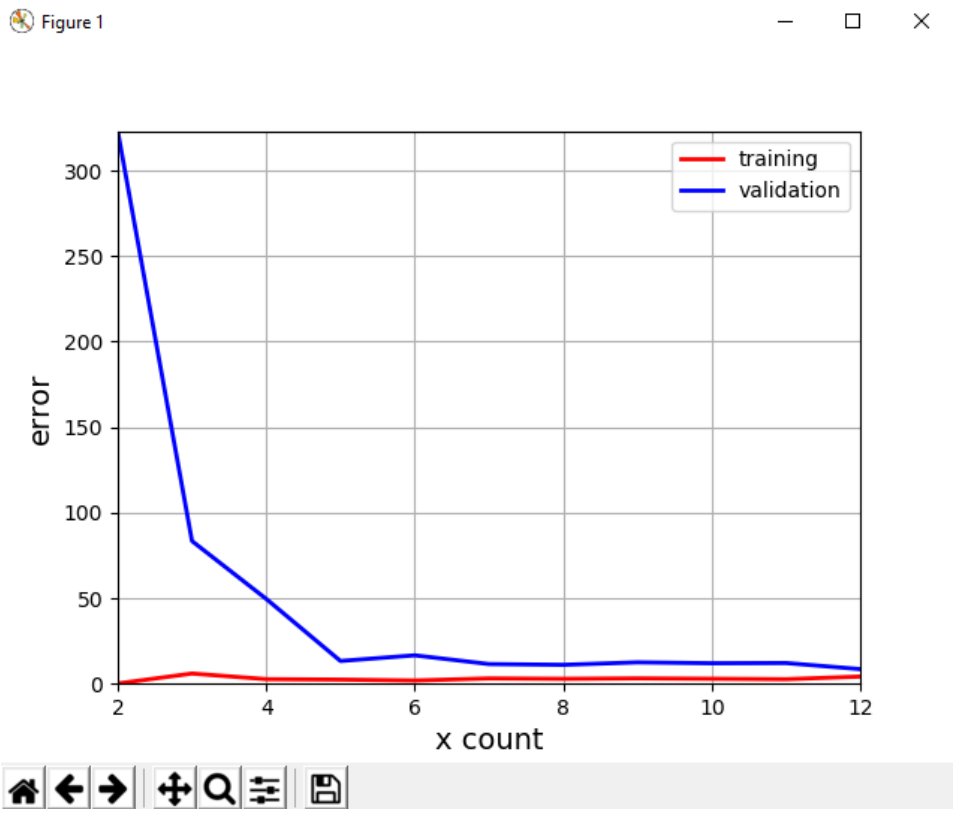
Здесь можно наблюдать, что модель идеально подстроилась под данные из тренировочного сета, но ошибка на валидационном сете непредсказуема, т.к. $\lambda = 0$, возникло переобучение

11. Постройте графики из пункта 10 для моделей с коэффициентами регуляризации 1 и 100. Какие выводы можно сделать?

```
# 11
theta_bfgs_scaled = optimize.fmin_bfgs(
    loss,
    theta.flatten(),
    gradient,
    (x_scaled.values, y_train, 1)
)

plt.scatter(x_train.values, y_train, color='red')
plt.plot(x, np.dot(x_polynom, theta_bfgs_scaled), linewidth=2)
plt.xlabel("water level", fontsize=14)
plt.ylabel("out", fontsize=14)
plt.show()

learning_curves_chart(x_scaled.values, y_train, val.values, y_val, 1)

theta_bfgs_scaled = optimize.fmin_bfgs(
    loss,
    theta.flatten(),
    gradient,
    (x_scaled.values, y_train, 100)
)
plt.scatter(x_train.values, y_train, color='red')
plt.plot(x, np.dot(x_polynom, theta_bfgs_scaled), linewidth=2)
plt.xlabel("water level", fontsize=14)
plt.ylabel("out", fontsize=14)
plt.show()
learning_curves_chart(x_scaled.values, y_train, val.values, y_val, 100)
```

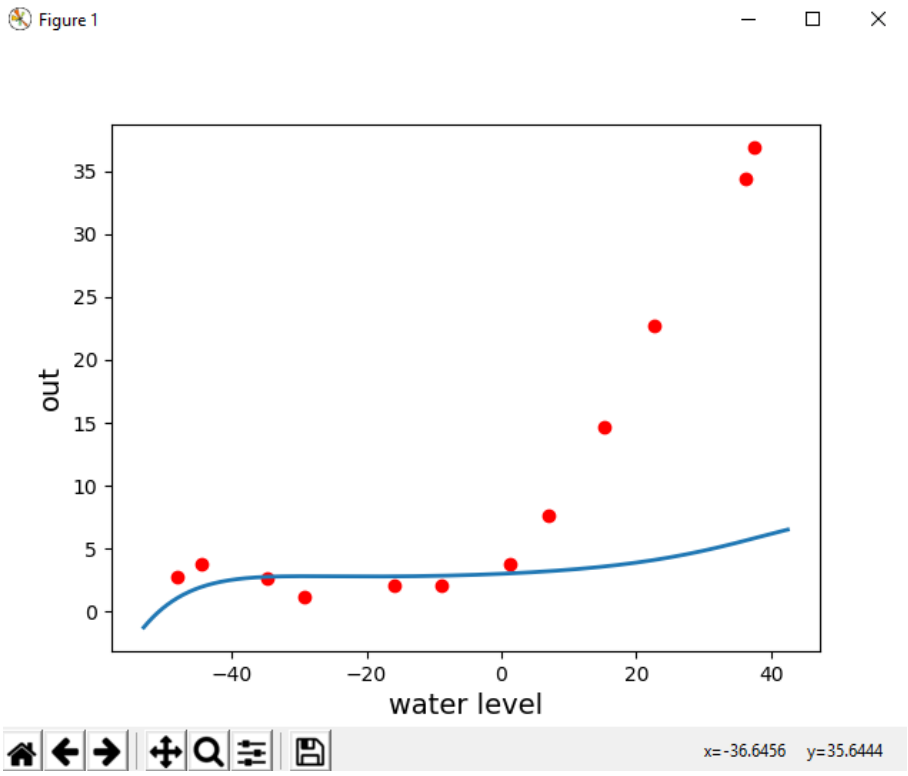
$\lambda = 1$

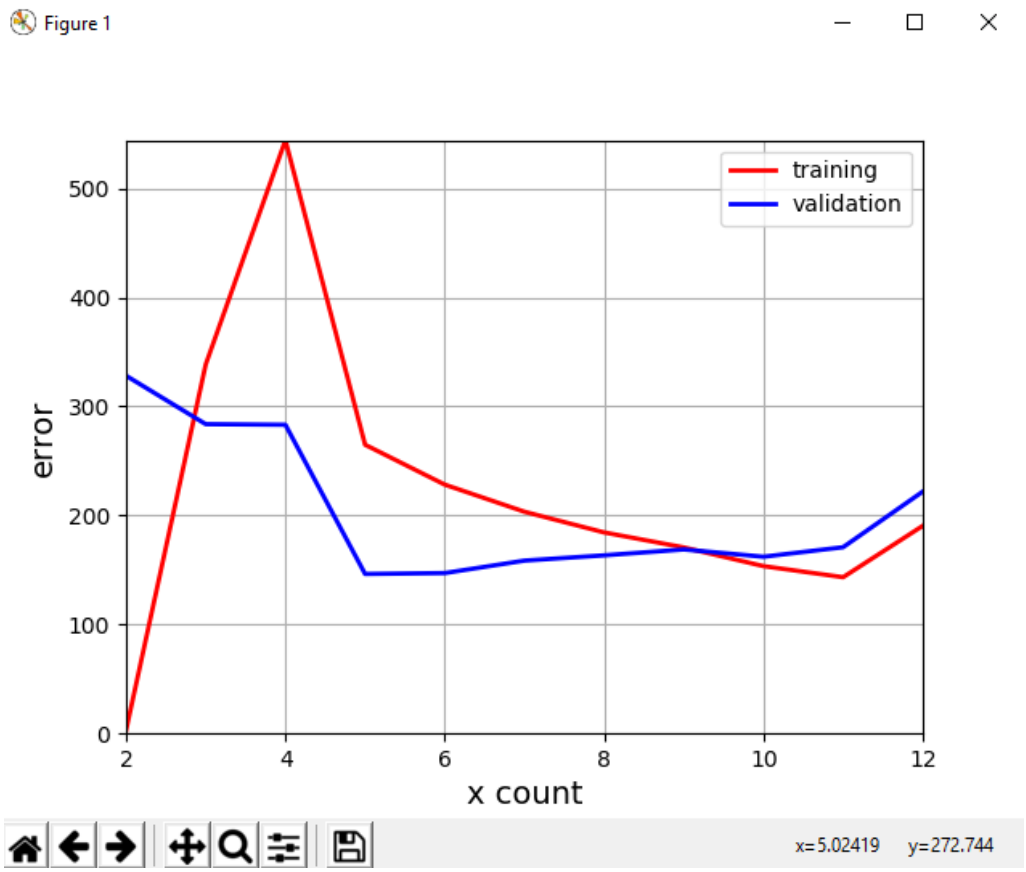




Как видно из первого графика, регуляризация «сгладила» нашу функцию, она теперь не идет строго по точкам из тренировочного датасета. Как видно из второго графика, регуляризация помогла нам избежать переобучения и ошибка как на тренировочном, так и на валидационном сете минимальна.

$$\lambda = 100$$

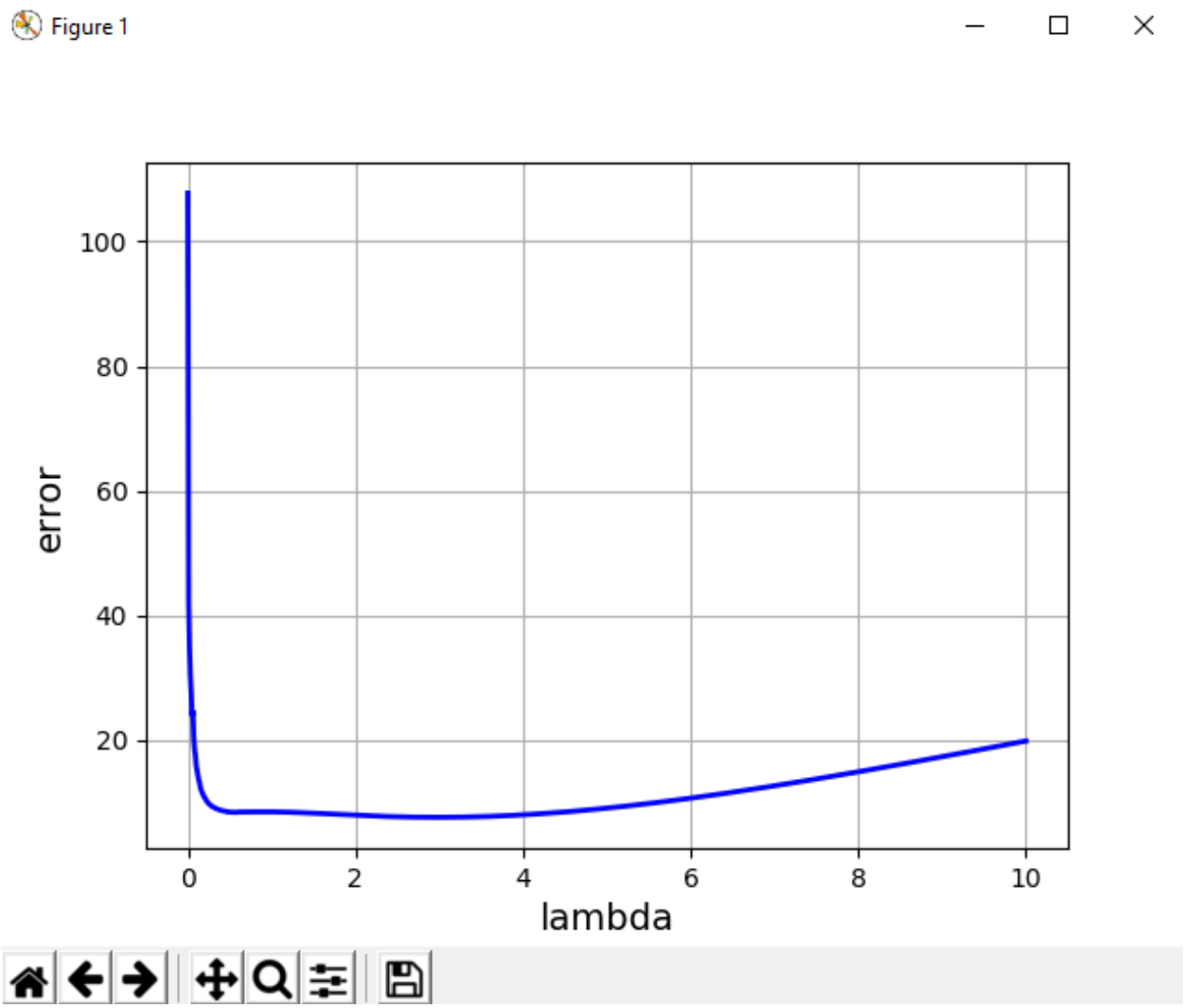




Как видно из графиков, штраф при таком коэффициенте регуляризации оказывается слишком большим, модель не может точно апроксимировать исходные данные

12) С помощью валидационной выборки подберите коэффиент регуляризации, который позволяет достичь наименьшей ошибки. Процесс подбора отразите с помощью графика (графиков).

```
# 12
lambda_values = np.linspace(0,10, 1000)
val_err = []
for lamb in lambda_values:
    theta_bfgs_scaled = optimize.fmin_bfgs(
        loss,
        theta.flatten(),
        gradient,
        (x_scaled.values, y_train, lamb)
    )
    val_err.append(loss(theta_bfgs_scaled, val, y_val))
plt.plot(lambda_values, val_err, c="b", linewidth=2)
plt.grid()
plt.xlabel("lambda", fontsize=14)
plt.ylabel("error", fontsize=14)
plt.show()
print(lambda_values[np.argmin(val_err)])
```



$$\lambda_{optimal} = 2.97$$

13) Вычислите ошибку (потерю) на контрольной выборке.

```
#13
theta = np.zeros(x_scaled.shape[1])
x_test = normalize(add_params(x_test, 8))
x_test.insert(0, "Ones", 1)
theta_bfgs_scaled = optimize.fmin_bfgs(
        loss,
        theta.flatten(),
        gradient,
        (x_scaled.values, y_train, 2.97)
)
print(loss(theta_bfgs_scaled, x_test, y_test))
```

$$F_{loss}(X, y)_{lambda=2.97} = 7.647$$

Выводы

В рамках этой лабораторной работы был изучен метод борьбы с переобучением моделей под названием регуляризация. Этот метод довольно прост: в функцию потерь добавляется еще один параметр, равный L2 норме вектора θ с настраиваемым коэффициентом. Суть этого коэффициента в том, что он штрафует модель за слишком большие вектора, при этом помогая модели справляться с выбросами в тренировочных данных и, тем самым, избегать переобучения.