Introduction

In this lab we are comparing influence of outliers in training data to performance of Linear Regression with Gradient Descent.

Introduction

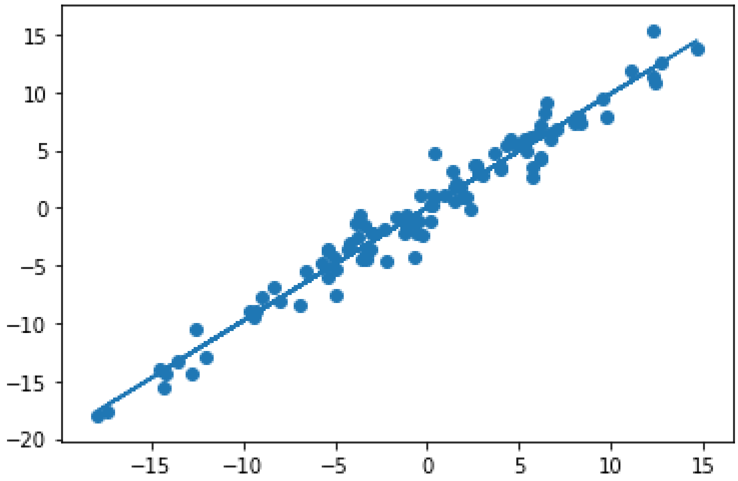
To perform all experiment we created a framework:

```
def f(w, x):
    return w[0] * x + w[1]


def Q(w, x, y):
    return ((w[0] * x + w[1] - y)**2).sum() / 2


def grad_Q(w, x, y):
    return np.array([((w[0] * x + w[1] - y) * x).sum(),
                     (w[0] * x + w[1] - y).sum()])


#GRADIENT DESCENT WITH ADAPTATIVE ALPHA AND FIXED STEP
def grad_descent(f,
                 grad_f,
                 w0,
                 x,
                 y,
                 fdif_threshold=1e-20,
                 grad_threshold=1e-20,
                 alfa_threshold=1e-20):

    w1 = w0
    f1 = f(w1, x, y)

    w_list = []
    w_list.append(w1)
    gradf_list = []

    loop_out = True
    loop_alfa = True
    j = 0
    while loop_out:
        j = j + 1
        gradf = grad_f(w1, x, y)
        if np.linalg.norm(gradf) <= grad_threshold:
            return w2, np.array(w_list)

        alfa = 1.0
        i = 0
        while loop_alfa:
            i = i + 1
```

```
            if alfa < alfa_threshold:
                w_list.append(w2)
                return w2, np.array(w_list)
            else:
                w2 = w1 - alfa * gradf
                f2 = f(w2, x, y)
                if f2 < f1:
                    loop_alfa = False
                else:
                    alfa = alfa / 2
        loop_alfa = True
        w_list.append(w2)
        if abs(f2 - f1) <= fdif_threshold:
            return w2, np.array(w_list)

        w1 = w2
        f1 = f2


def prepare_data():
    m = [0., 0.]
    angle = 45 * math.pi / 180
    rot = np.array([[math.cos(angle), -math.sin(angle)],
                    [math.sin(angle), math.cos(angle)]])
    lamb = np.array([[100, 0], [0, 1]])
    s = np.matmul(rot, np.matmul(lamb, rot.transpose()))
    c = np.random.multivariate_normal(m, s, 100)
    return c


def test_lin_reg(c, Q, grad_Q, f):
    x = c[:, 0]
    y = c[:, 1]
    w = np.array([1, 1])

    w, _ = grad_descent(Q,
                        grad_Q,
                        w,
                        x,
                        y,
                        fdif_threshold=1e-20,
                        grad_threshold=1e-20,
                        alfa_threshold=1e-20)
    plt.plot(x, f(w, x))
    plt.scatter(x, y)
    plt.show()
    return w
```

Experiment 1

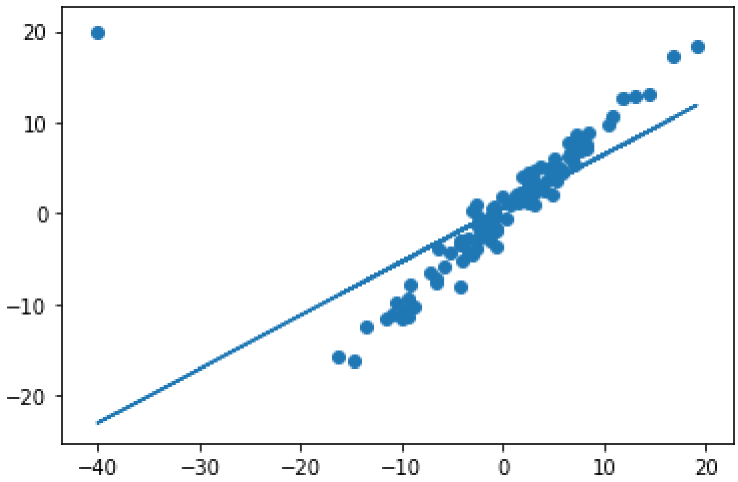
```
c = prepare_data()
test_lin_reg(c, Q, grad_Q, f)
```



With this data our Linear Regression works very good, output parameters are: 0.98565216, 0.02705736 which are really close to original 1.0, 0.0

Experiment 2

After adding outlier results are way worse than previous:



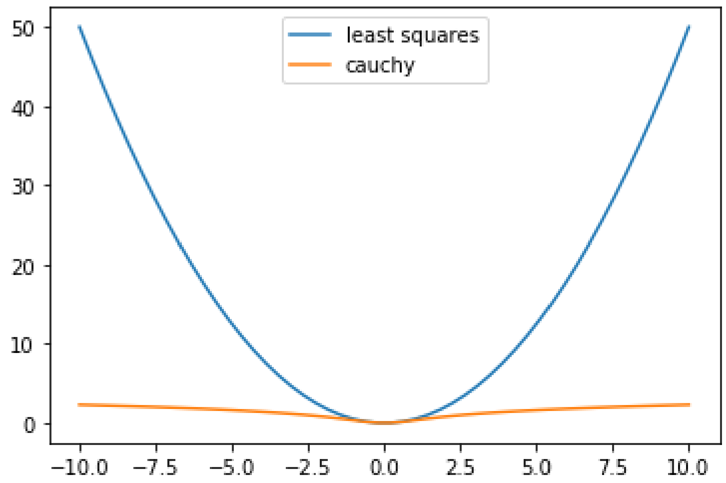
Output parameters are now 0.59021427, 0.56617796 it is due to that LR is trying to minimize the big difference between line and an outlier, which results in line tilt.

Robust functions

We are using two different cost functions Cauchy function and Least Squares function, and measure their sensitivity to outliers.

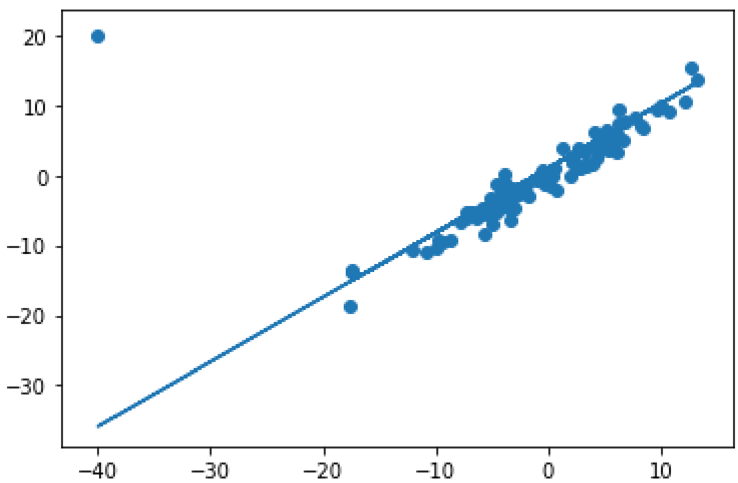
Experiment 1

```python
def p(u, type, c=1.0):
    if type == 'ls':
        return u**2 / 2
    elif type == 'cauchy':
        return (c**2 / 2) * np.log(1 + (u / c)**2)

linsp = np.linspace(-10, 10)
plt.plot(linsp, p(linsp, type='ls'), label='least squares')
plt.plot(linsp, p(linsp, type='cauchy'), label='cauchy')
plt.legend(loc='upper center')
plt.show()
```



Experiment 2

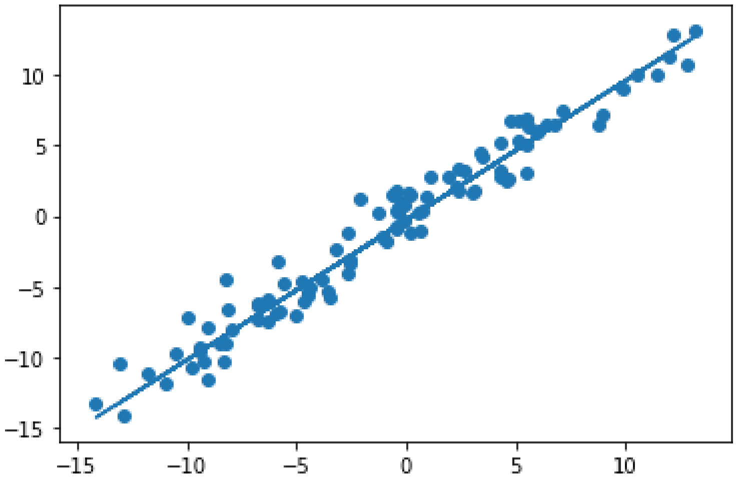
```python
def Q_cauchy(w, x, y, c=1.0):
    e = w[0] * x + w[1] - y
    return (p(e, type='cauchy', c=1.0)).sum()
c = prepare_data_outlier()
test_lin_reg(c, Q_cauchy, grad_Q, f)
```

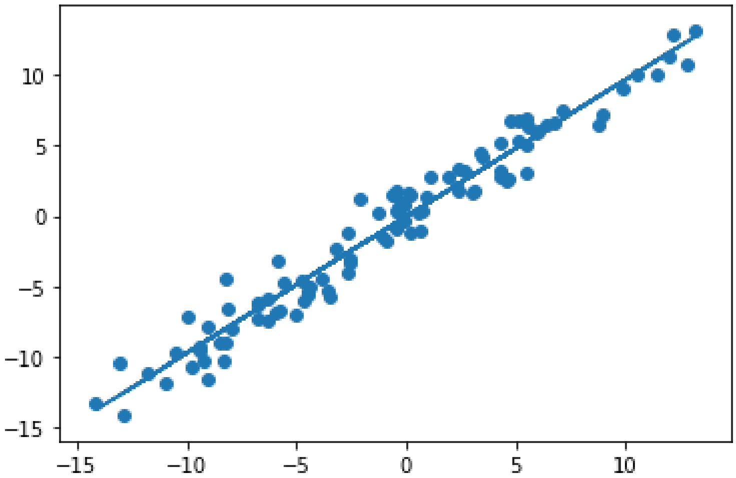


As we can see `Q_couchy` function is more resistant to outliers and results in better approximation of data model in Linear Regression.

Experiment 3

```
def Q_ls(w, x, y):
    e = w[0] * x + w[1] - y
    return (p(e, type='ls')).sum()

c = prepare_data()
res_cauchy = test_lin_reg(c, Q_cauchy, grad_Q, f)
res_ls = test_lin_reg(c, Q_ls, grad_Q, f)
print("res_cauchy:", res_cauchy, '\nres_lsquar:', res_ls)
```



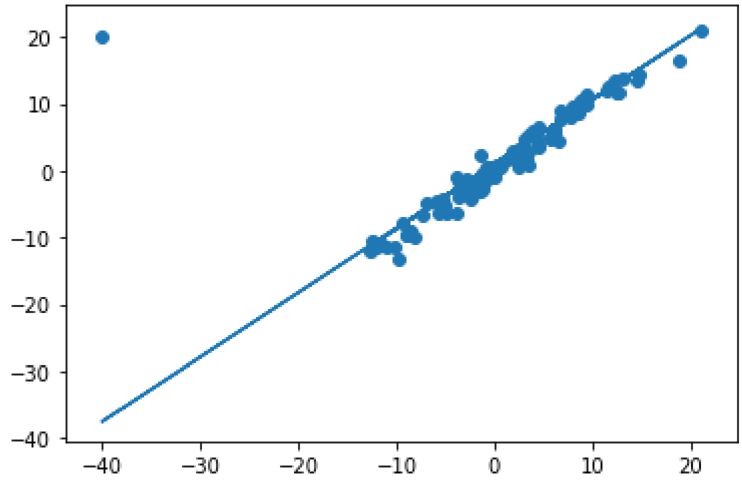


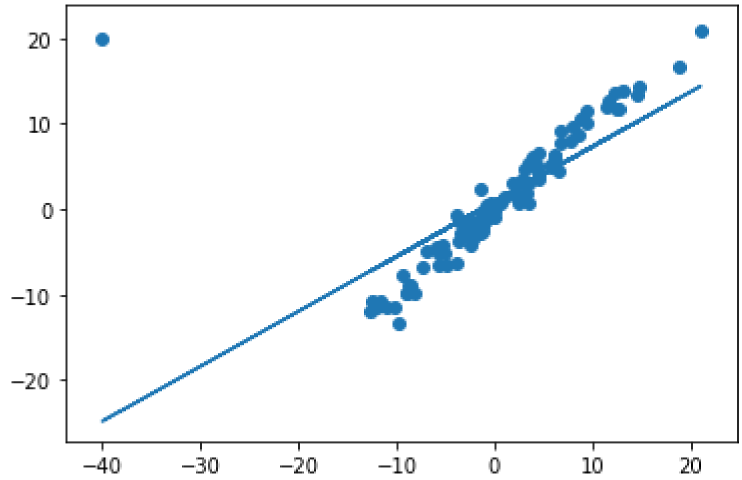
```
res_cauchy: [ 0.98932349 -0.30437448]
res_lsquar: [ 0.96924139 -0.05255234]
```

As we can see there is no big difference, both methods works good.

Experiment 4

```
c = prepare_data_outlier()
res_cauchy = test_lin_reg(c, Q_cauchy, grad_Q, f)
res_ls = test_lin_reg(c, Q_ls, grad_Q, f)
print("res_cauchy:", res_cauchy, '\nres_lsquar:', res_ls)
```



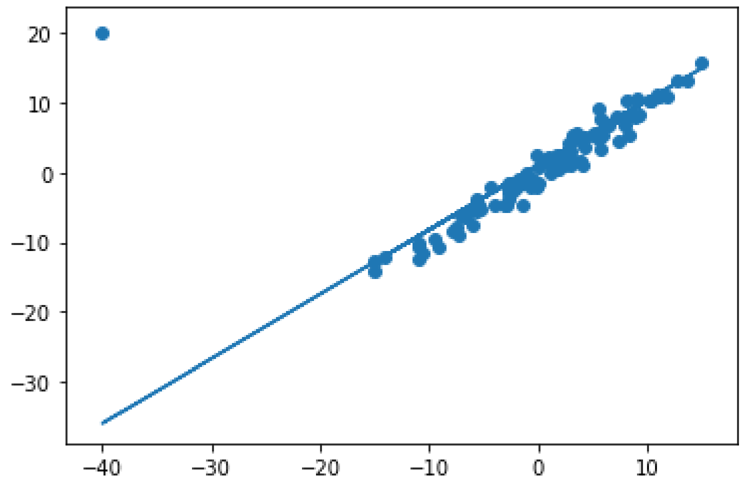


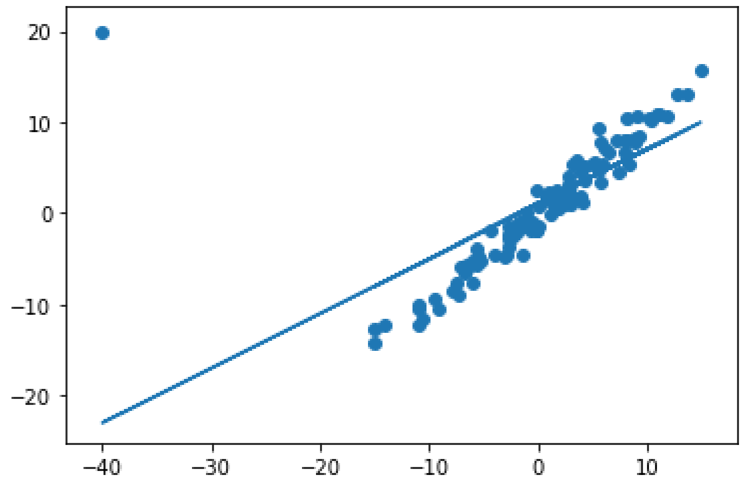

```
res_cauchy: [0.96295414 0.99961562]
res_lsquar: [0.64209666 0.87688545]
```

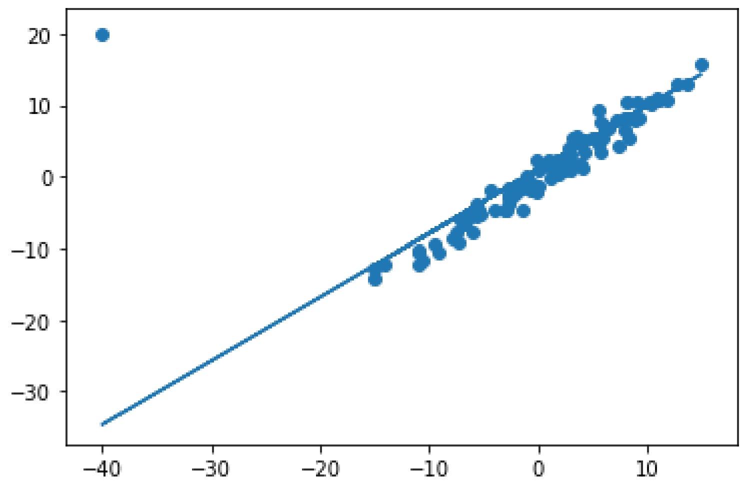
As expected Cauchy function is more robust than the quadratic function, outlier doesn't affect Linear Regression with Cauchy function that much.

Experiment 5

```
c = prepare_data_outlier()
def Q_cauchy(w, x, y, c=1.0):
    return (p(w[0] * x + w[1] - y, type='cauchy', c=c)).sum()
res_cauchy_1 = test_lin_reg(c, Q_cauchy, grad_Q, f)

def Q_cauchy(w, x, y, c=100.0):
    return (p(w[0] * x + w[1] - y, type='cauchy', c=c)).sum()
res_cauchy_100 = test_lin_reg(c, Q_cauchy, grad_Q, f)

def Q_cauchy(w, x, y, c=0.01):
    return (p(w[0] * x + w[1] - y, type='cauchy', c=c)).sum()
res_cauchy_1_100 = test_lin_reg(c, Q_cauchy, grad_Q, f)

print("res_cauchy_1:", res_cauchy_1, '\nres_cauchy_100:', res_cauchy_100,
'\nres_cauchy_1_100:', res_cauchy_1_100)
```



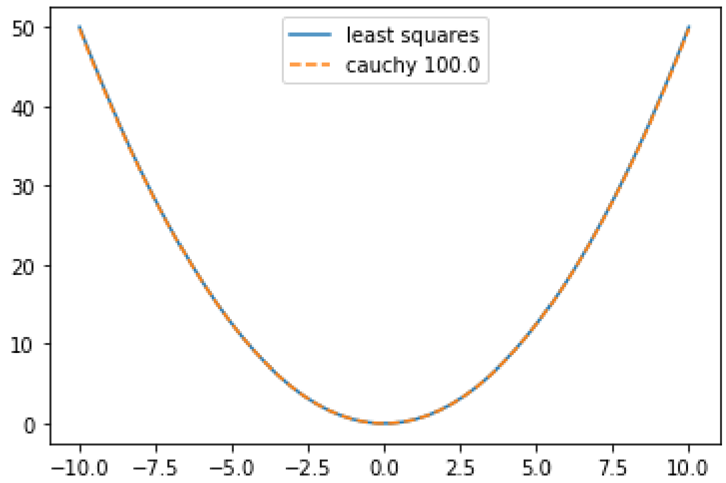





```
res_cauchy_1: [0.92243765 0.99847253]
res_cauchy_100: [0.60080138 0.9755706 ]
res_cauchy_1_100: [0.89075543 0.99774552]
```

As we can see Couchy with c=100 perform as bad as least squares, lets plot it.

```
linsp = np.linspace(-10, 10)
plt.plot(linsp, p(linsp, type='ls'), label='least squares')
plt.plot(linsp, p(linsp, type='cauchy', c=100.0), label='cauchy 100.0', ls="--")
plt.legend(loc='upper center')
plt.show()
```

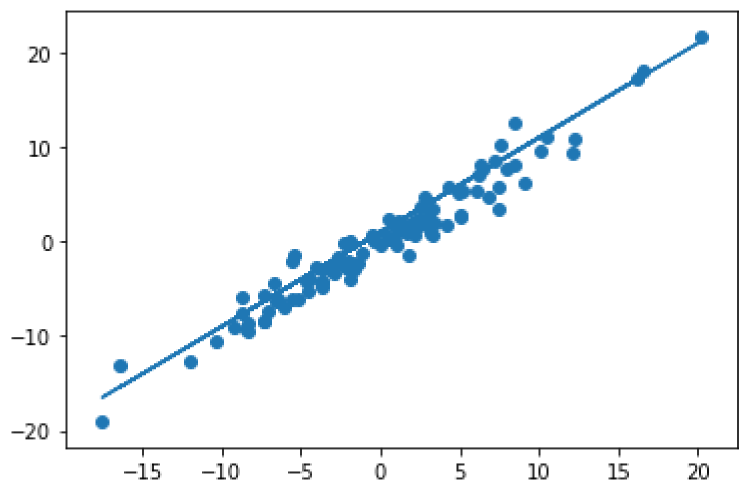


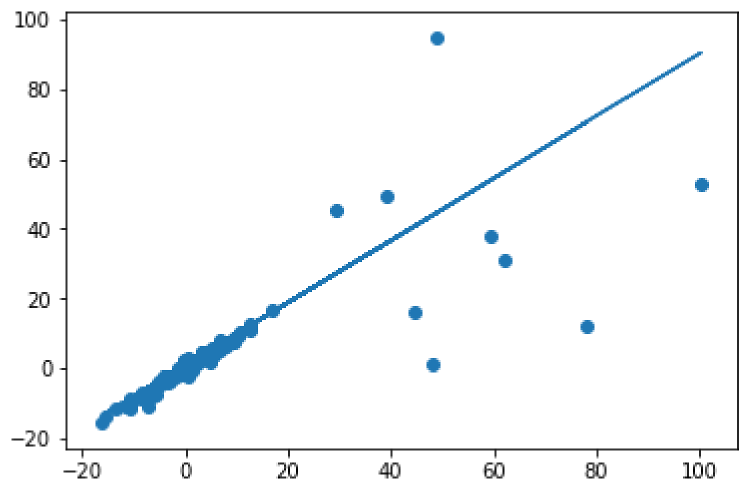
As we can see they behave the same, that induces such behavior.

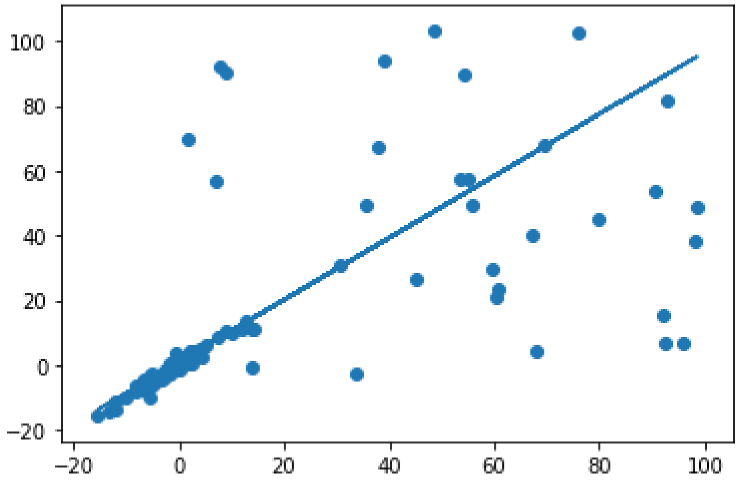
Experiment 6

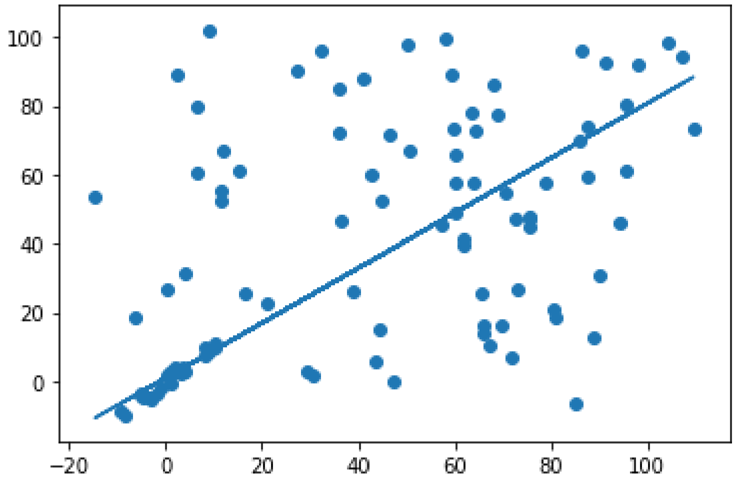
We have performed experiment where each set of data had one more malformed point - outlier. Below, we presents some of them.

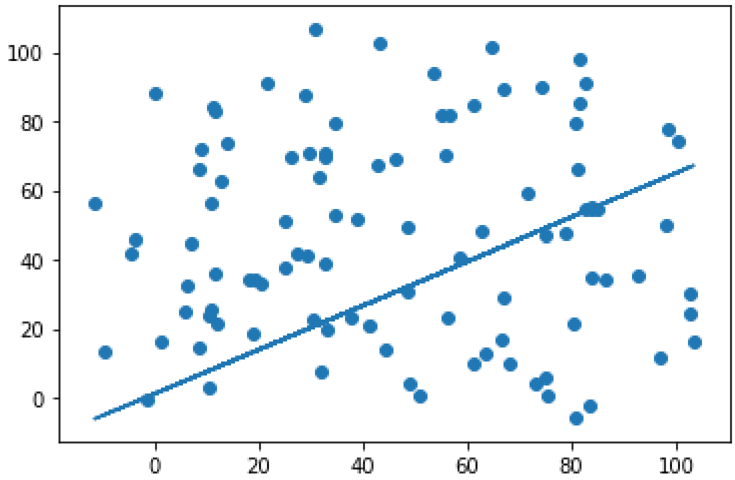
```
for i in dim:
    c = prepare_data_outliers(i)
    w = test_lin_reg(c, Q_cauchy, grad_Q, f)
    res.append(np.linalg.norm(w-w_ref))
```







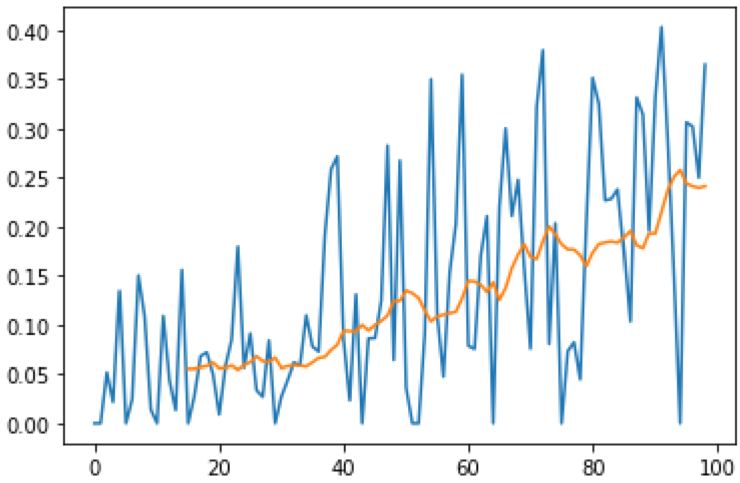




We have measured norm of difference between outputed w and its real value.

```
import matplotlib.pyplot as plt
plt.plot(l)
plt.plot([np.mean(l[i-15:i]) for i in range(len(l))])
```

This is the plot of its values (blue) and mean with window equals 15



After 40% of the data is malformed the distance between expected value starts growing.