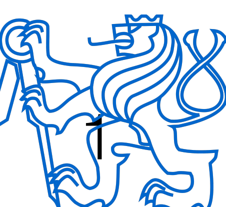
VIR lab 2

Non-linear regression and computational graphs

Karel Zimmermann
Czech Technical University in Prague
Faculty of Electrical Engineering, Department of Cybernetics



Assignment

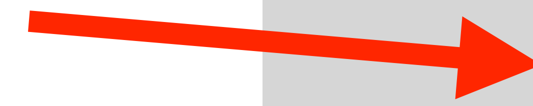
$$\mathcal{D} = \{\mathbf{x}_1, y_1 \ldots \mathbf{x}_N, y_N\}$$

```
pts = np.load('pts.npy')
```

Assignment

$$\mathcal{D} = \{\mathbf{x}_1, y_1 \ldots \mathbf{x}_N, y_N\}$$

$$f(\mathbf{x}_i, \mathbf{w}) = \frac{1}{1 + e^{-(\mathbf{w}_0 * \mathbf{x}_i + \mathbf{w}_1)}}$$



```
pts = np.load('pts.npy')

for i in range(30):
    f = …       # use torch.exp()
```

Assignment

$$\mathcal{D} = \{\mathbf{x}_1, y_1 \ldots \mathbf{x}_N, y_N\}$$

$$f(\mathbf{x}_i, \mathbf{w}) = \frac{1}{1 + e^{-(\mathbf{w}_0 * \mathbf{x}_i + \mathbf{w}_1)}}$$

$$\mathscr{L}(\mathbf{w}) = \sum_i (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$$

```
pts = np.load('pts.npy')

for i in range(30):
    f = …       # use torch.exp()
    loss = … # use torch.sum()
```

Assignment

$$\mathcal{D} = \{\mathbf{x}_1, y_1 \ldots \mathbf{x}_N, y_N\}$$

$$f(\mathbf{x}_i, \mathbf{w}) = \frac{1}{1 + e^{-(\mathbf{w}_0 * \mathbf{x}_i + \mathbf{w}_1)}}$$

$$\mathscr{L}(\mathbf{w}) = \sum_i (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathscr{L}(\mathbf{w})$$

```
pts = np.load('pts.npy')

for i in range(30):
    f = …      # use torch.exp()
    loss = … # use torch.sum()

    loss.backward()
    with torch.no_grad():
        w -= learning_rate * w.grad
    w.grad.zero_()
```

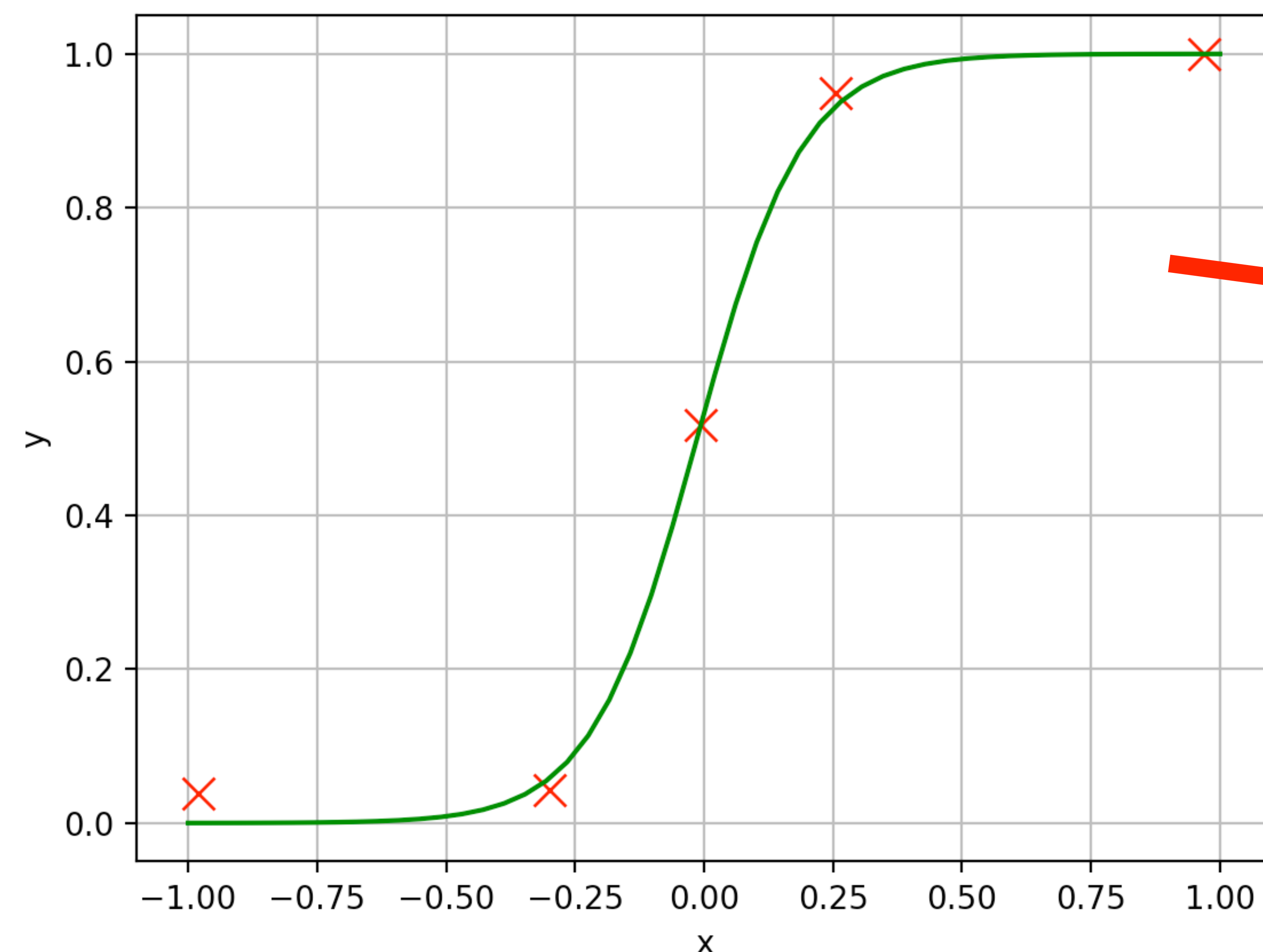
Assignment

$$\mathcal{D} = \{\mathbf{x}_1, y_1 \ldots \mathbf{x}_N, y_N\}$$

$$f(\mathbf{x}_i, \mathbf{w}) = \frac{1}{1 + e^{-(\mathbf{w}_0 * \mathbf{x}_i + \mathbf{w}_1)}}$$

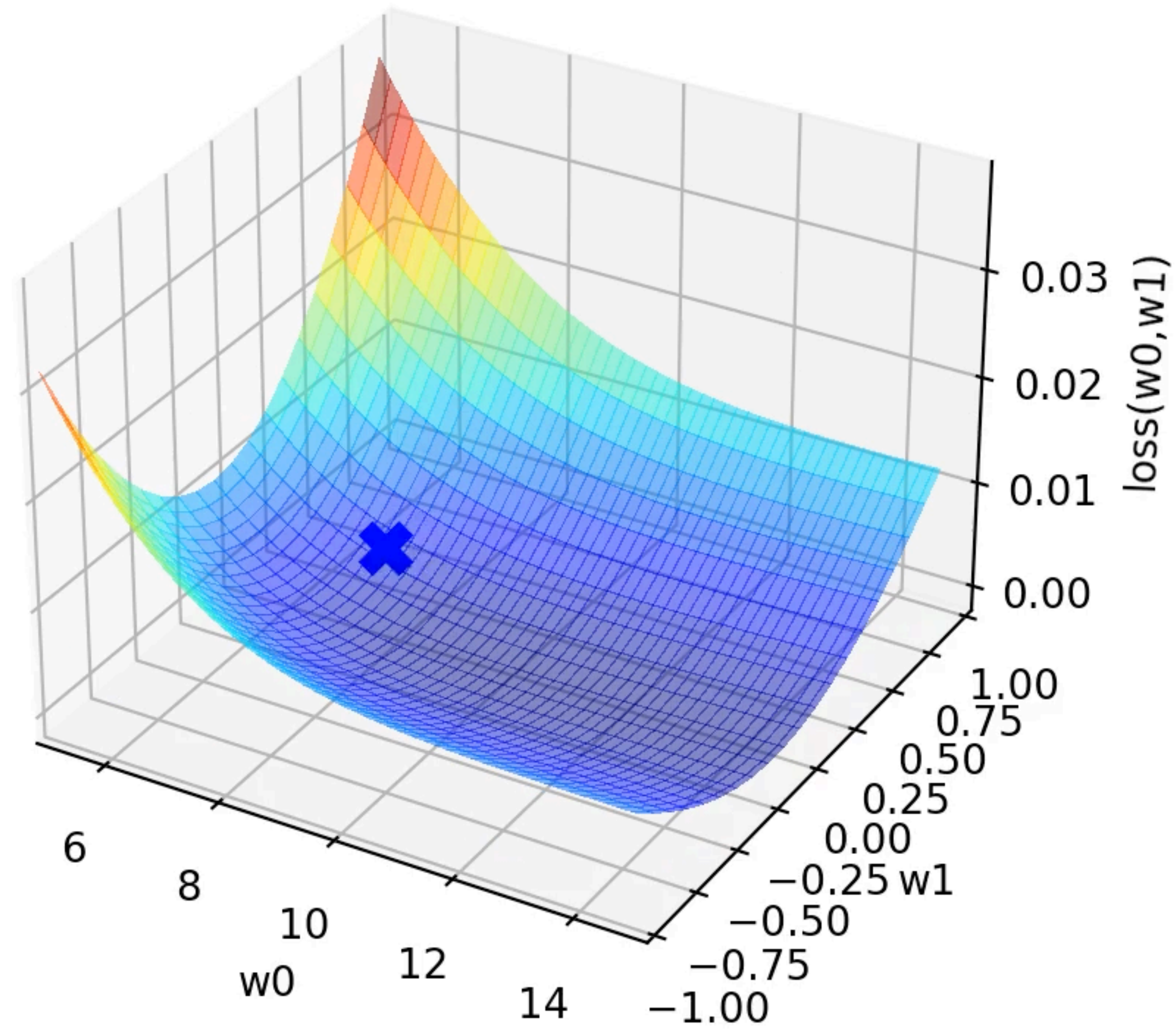
$$\mathscr{L}(\mathbf{w}) = \sum_i (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathscr{L}(\mathbf{w})$$

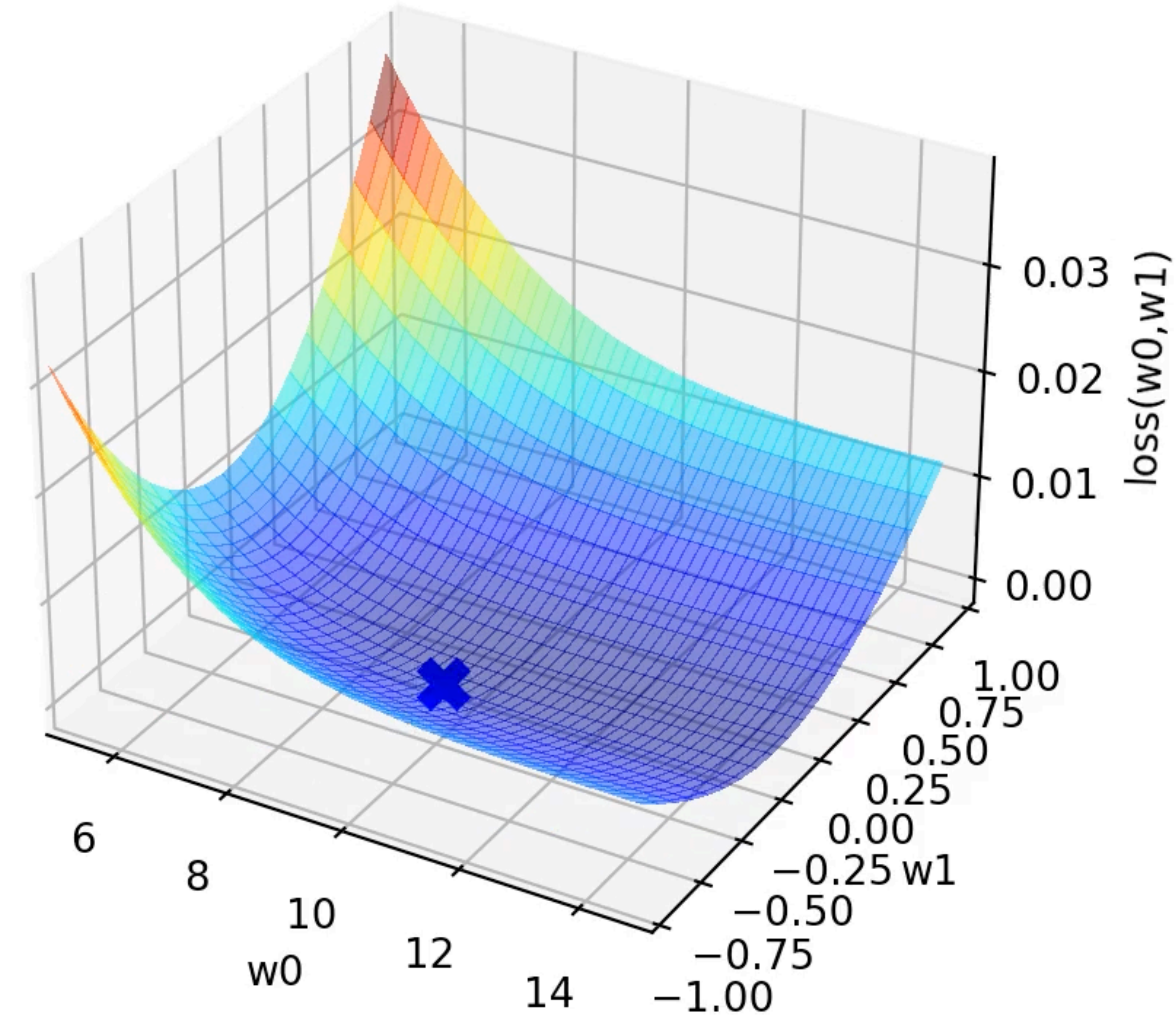



```
pts = np.load('pts.npy')

for i in range(30):
    f = …        # use torch.exp()
    loss = … # use torch.sum()

    loss.backward()
    with torch.no_grad():
        w -= learning_rate * w.grad
    w.grad.zero_()

    # visualize result
```

- What is the reasonable learning rate?



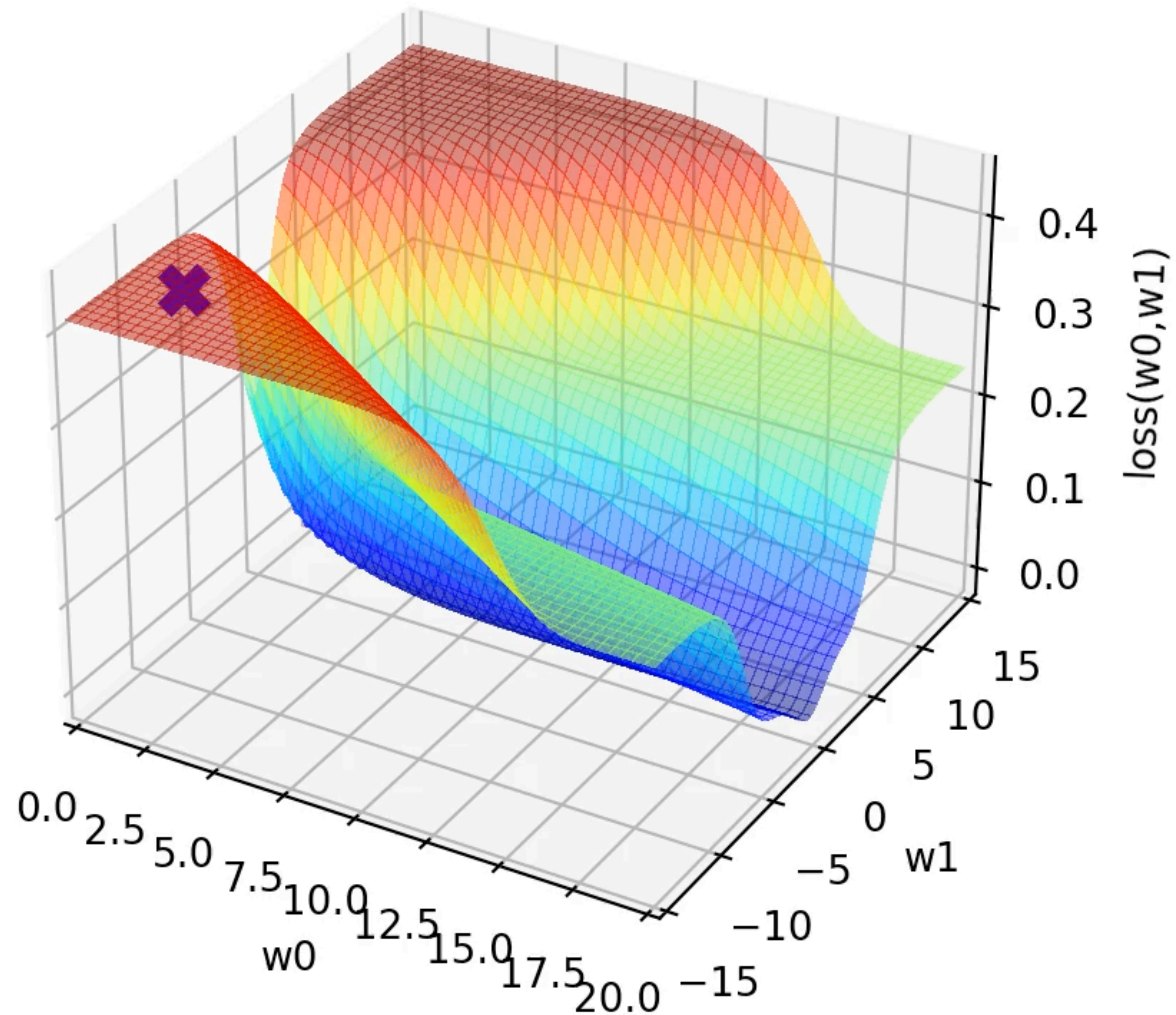
LSQ
$\alpha = 50$



LSQ
$\alpha = 80$

Should not we use different learning rate along different axis w0,w1?

- What is the reasonable learning rate?



How does the larger scale looks like

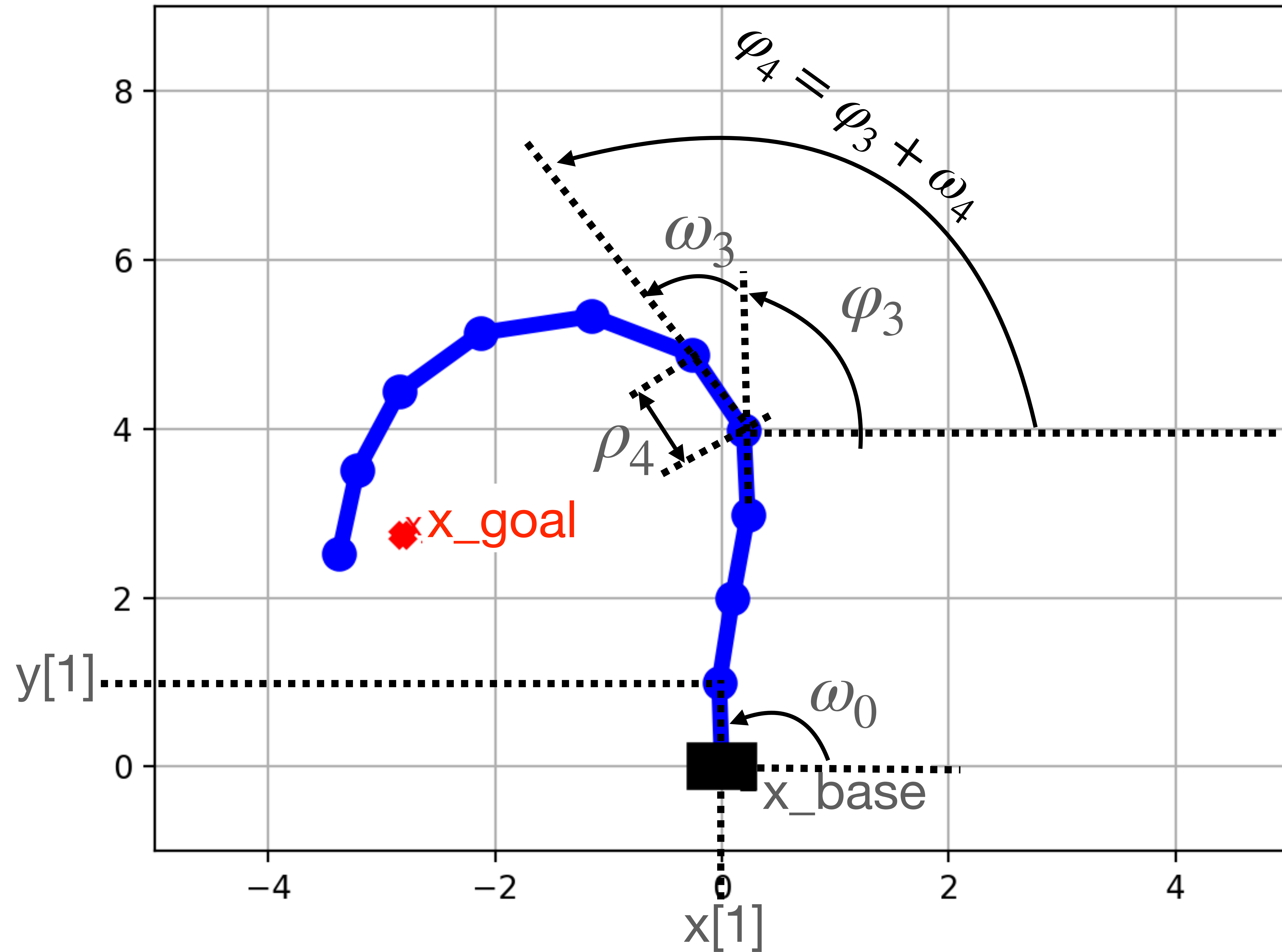
Assignment

- Is it linear least squares?
- Is the loss quadratic?
- How many steps required for full Newton method for quadratic loss?

- Non-linear least squares, GD, SGD
- mean / sum
- landscape w0/w1

- Computational graph + backprop (is it DAG?)
- Derive grad of sigmoid
- Jacobian and vector-Jacobian

Homework HW1



implement DKT

<u>input:</u> joints ω_i, links ρ_i, x_base

<u>output:</u> 2x(K+1) matrix

$$\mathbf{x} = \begin{bmatrix} \text{x_base[0]} & \text{x[1]} & \text{x[2]} & \\ \text{x_base[1]}, & \text{y[1]}, & \text{y[2]} & \cdots \end{bmatrix}$$

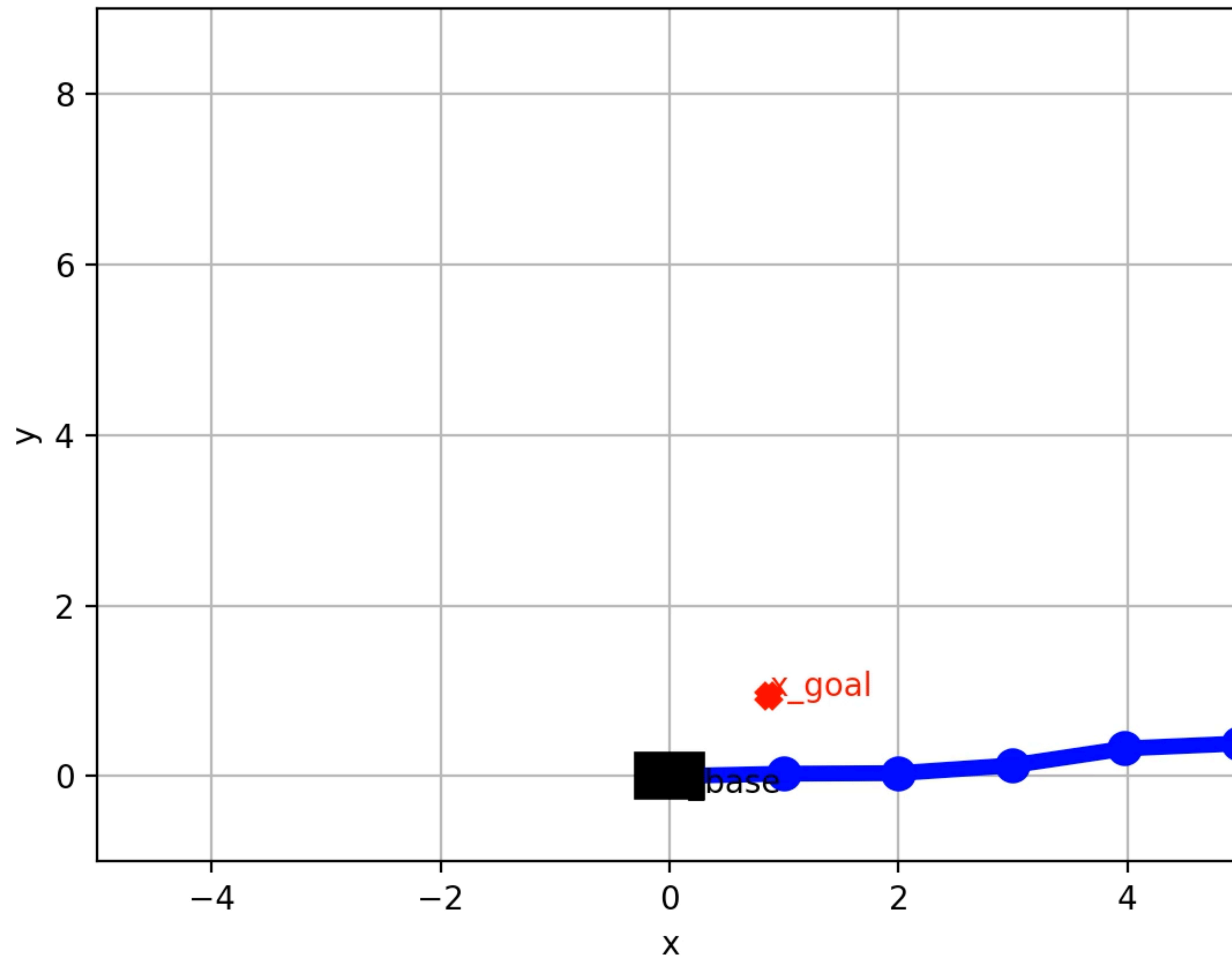
Homework HW1

```
def dkt(joints, links, x_base):

pos_x, pos_y, omega  = x_base[0], x_base[1], 0
x = [torch.stack((pos_x, pos_y))]
for k in range(LENGTH):
    omega = ... # omega of (k+1)-th joint as a function of previous omega and joints[k]
    pos_x = ... # x-coordinate of (k+1)-th joint as a function of previous pos_x, omega and links[k]
    pos_y = ... # y-coordinate of (k+1)-th joint as a function of previous pos_y, omega and links[k]
    x.append(torch.stack((pos_x, pos_y)))
return torch.stack(x, dim=1)
```

```
optimizer = optim.SGD([joints,links], lr=0.005)

for i in range(100):
    optimizer.zero_grad()
    x = dkt(joints, links, x_base)
    loss = ... # define loss e.g. ||x_last-x_goal||
    loss.backward()
    optimizer.step()
```

Homework HW1

Task 1 (5p): gradient ikt / opt. control

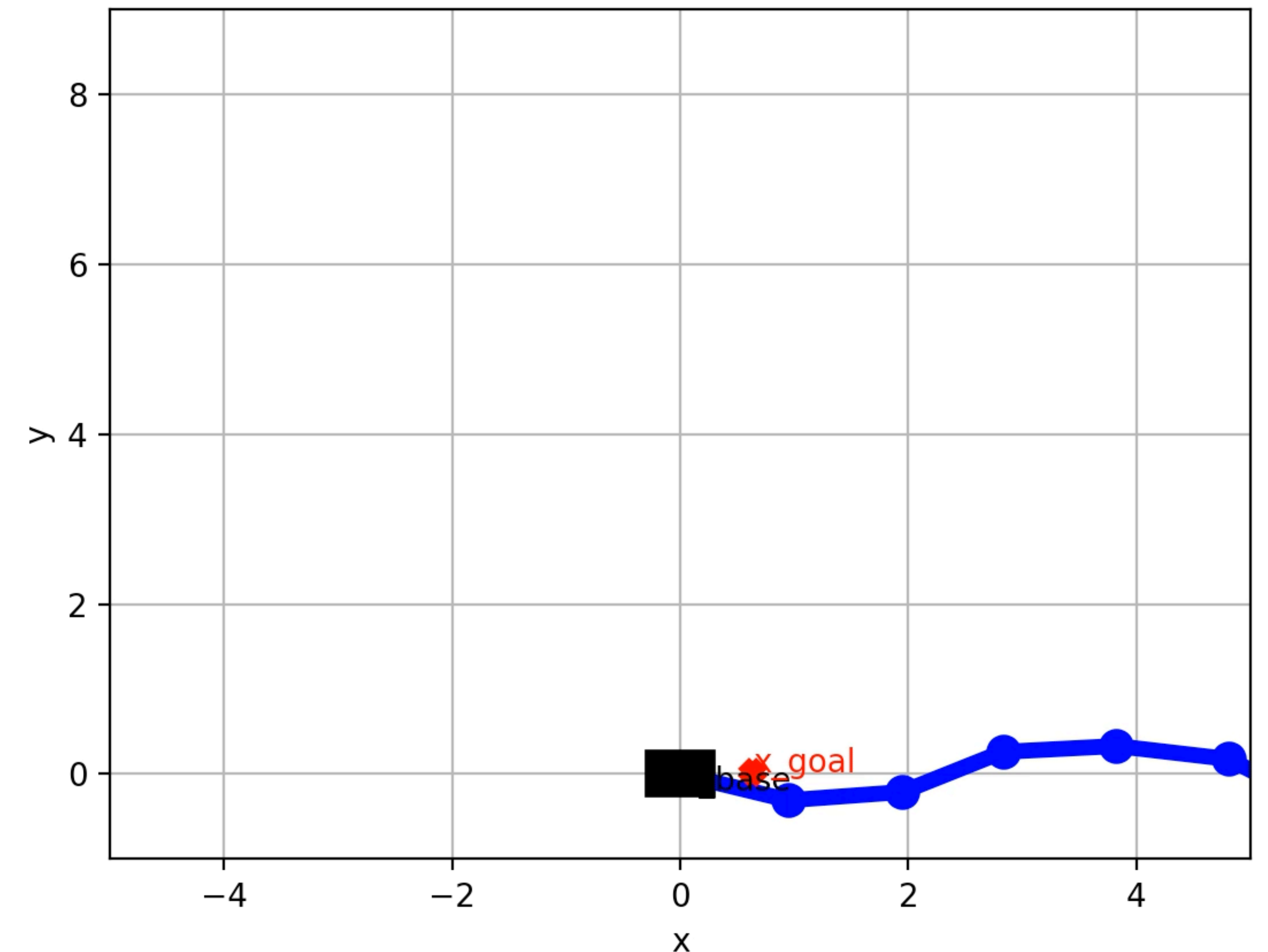


known: x_goal, x_base, links

to be estimated: **joints**

```
optimizer = optim.SGD([joints])
```

gradient calibration/ opt. design



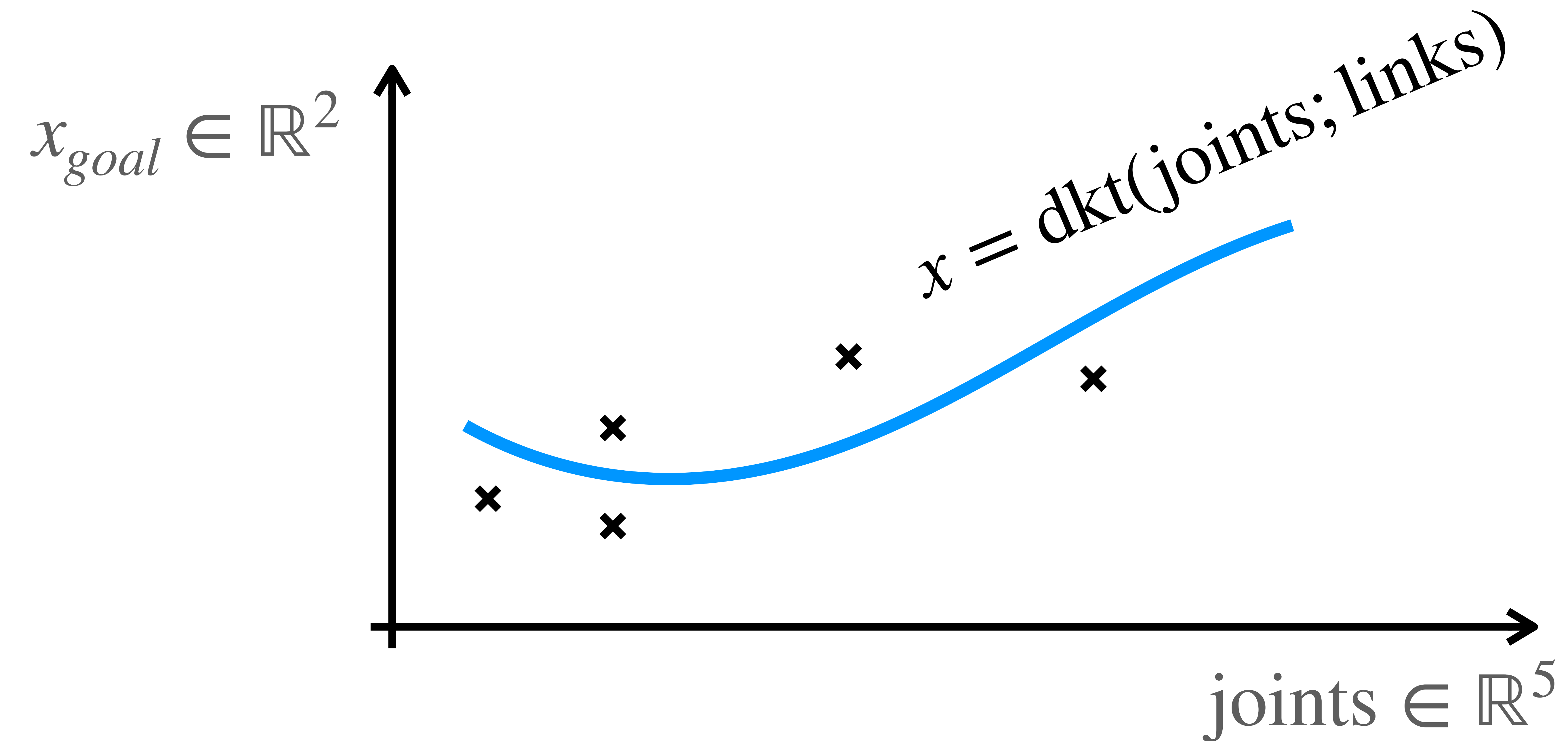
known: x_goal, x_base, joints

to be estimated: **links**

```
optimizer = optim.SGD([links])
```

Homework HW1

Task 2: calibration of 2D manipulator (1 points)



input: X_GOAL… 2x60 matrix of end-effector positions

JOINTS….. 5x60 matrix of corresponding joint angles

outputs: links …….. 5-dimensional vector of links length such that

$$\arg\min_{links} \sum_i \|\text{dkt}(\text{joints}^i; \text{links}) - x^i_{goal}\|$$

Homework HW1

Task 2: calibration of 2D manipulator (2 points)