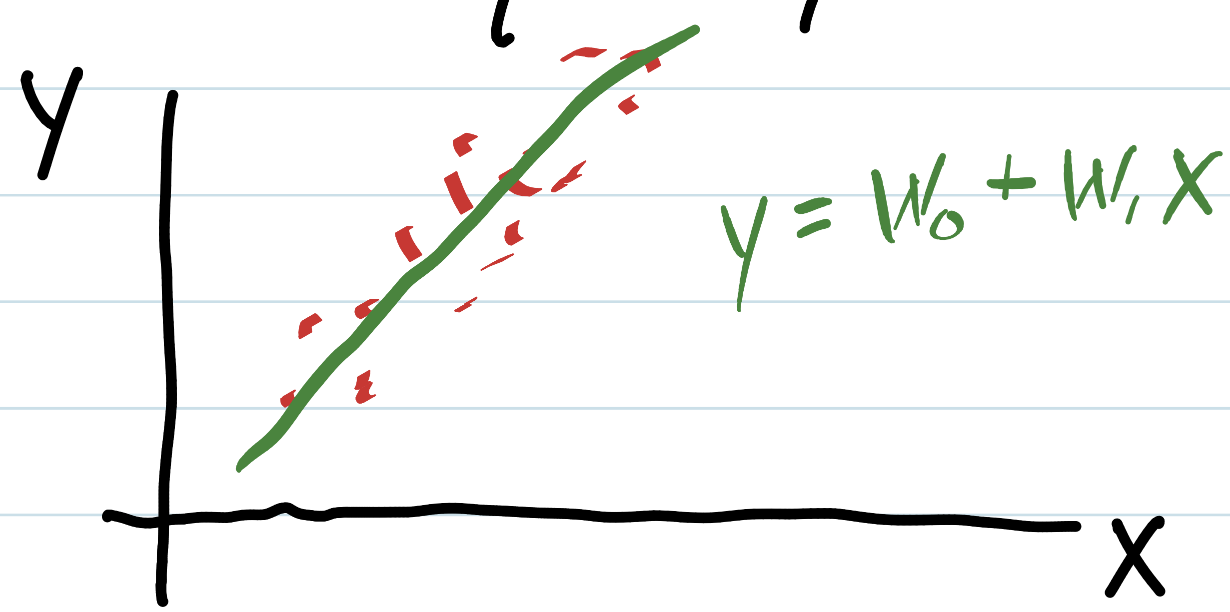
Locally Weighted Regression

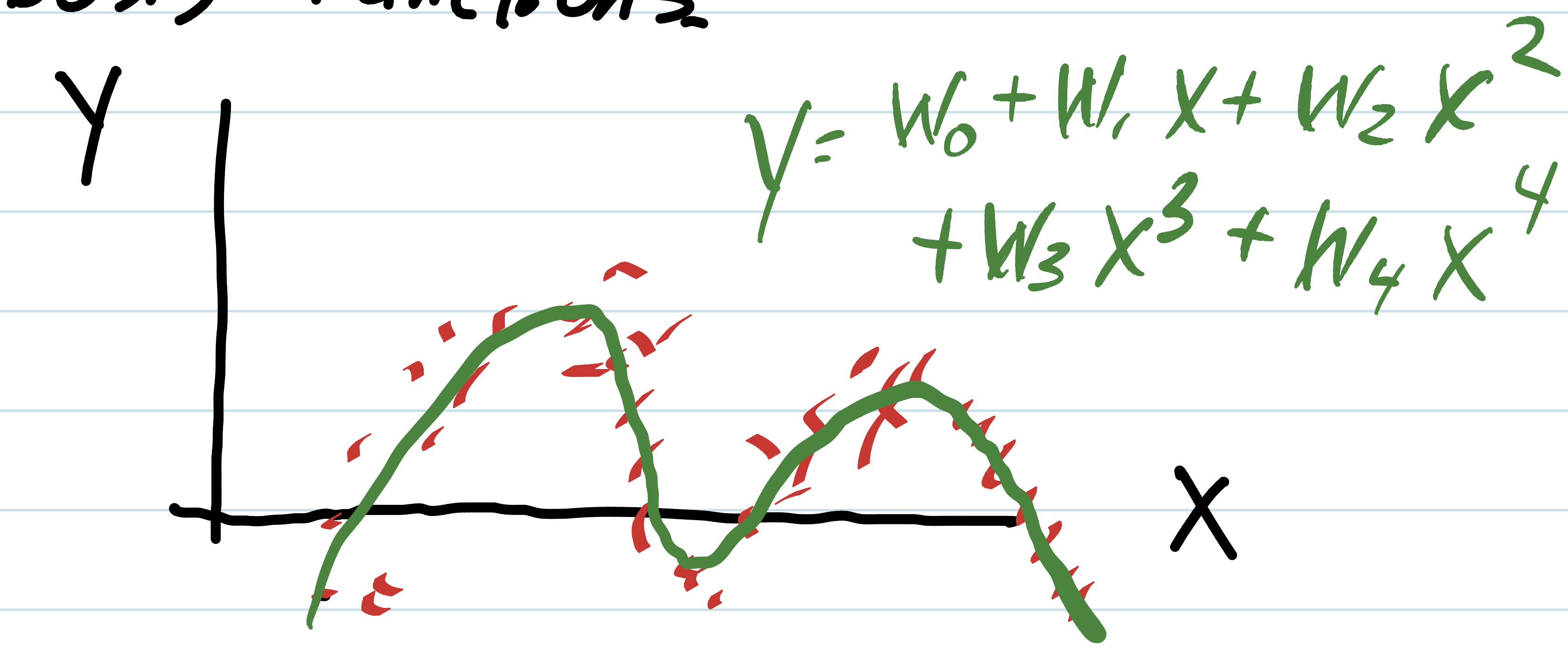
In "plain old" linear regression we used a model that is a linear combination of basis functions.

$$y = w^T \phi(x)$$

We assumed that this model was valid $\forall$ x, or in other words globally.

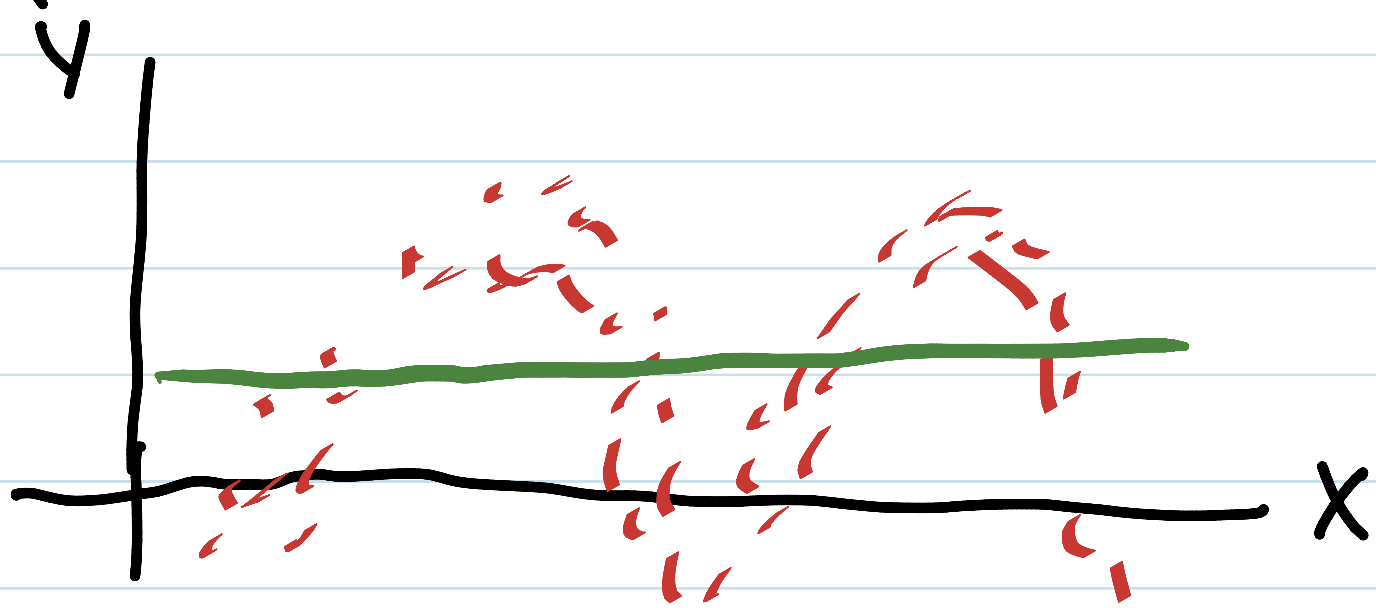
This generally works fine, but as the target function gets more complex you need more basis functions.





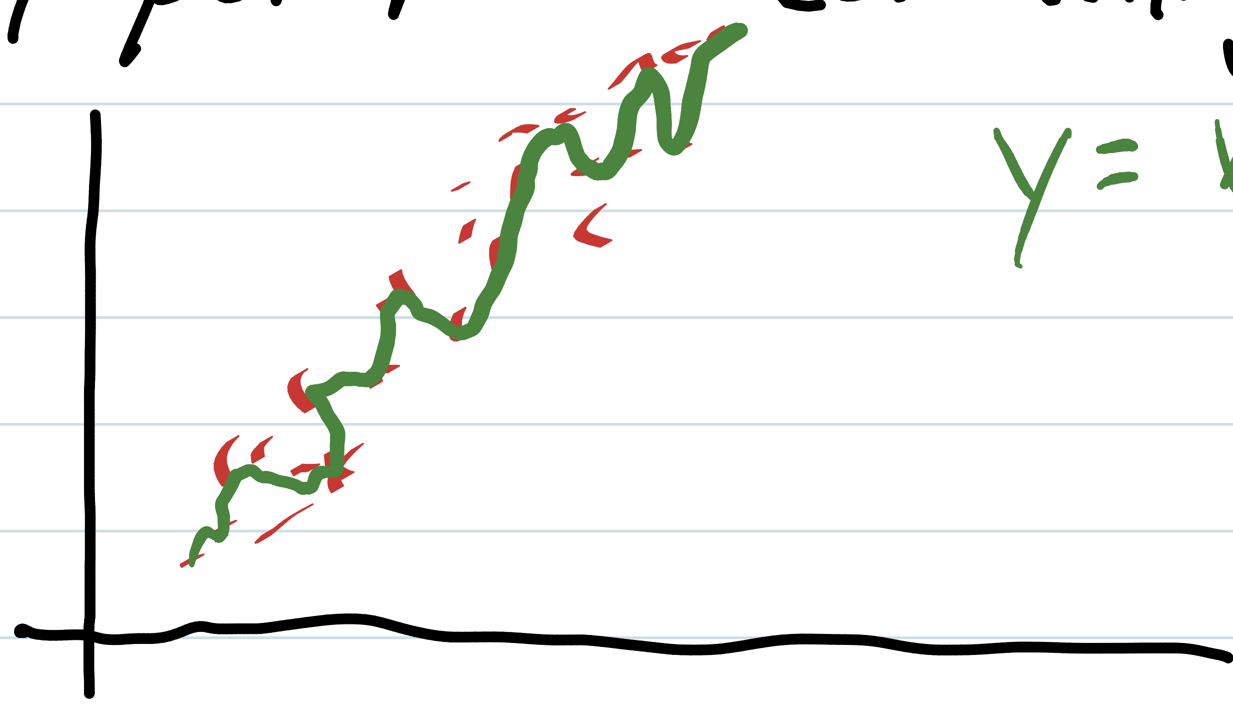
Problems with linear regression that uses a global model.

1) too few parameters (can't fit data well)



$y = w_0 + w_1 x$

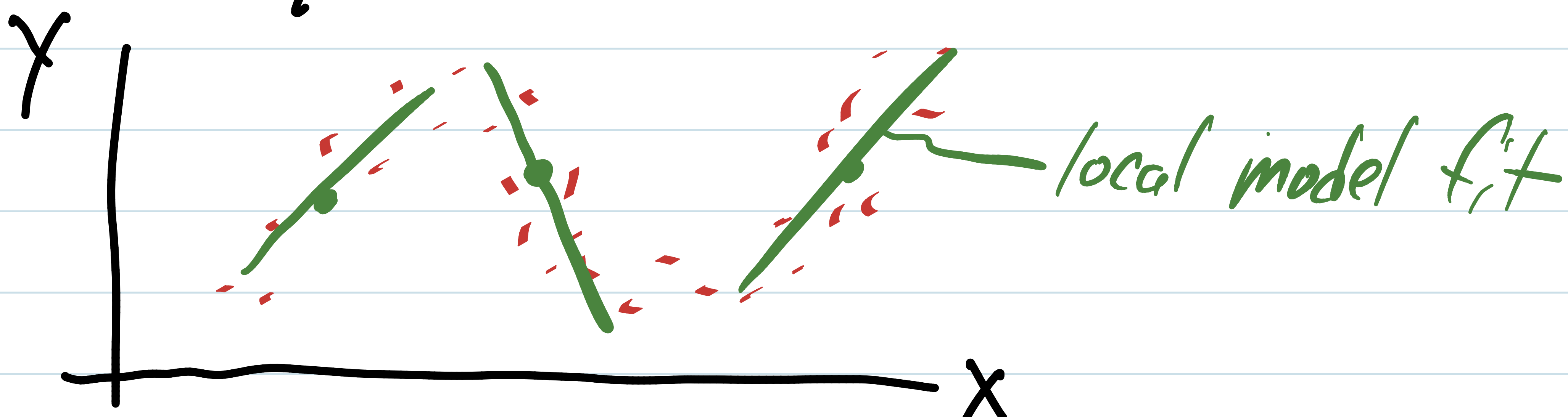
2) too many parameters (overfitting)



$y = w_0 + w_1 x \ldots\ldots$

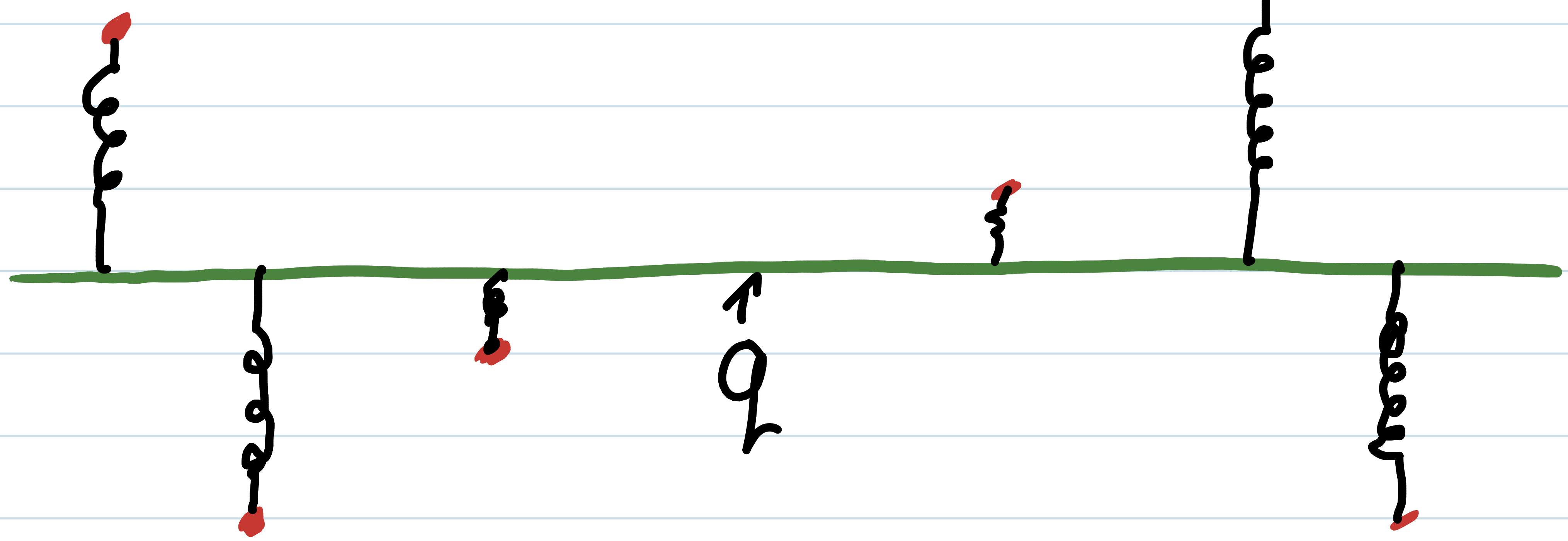
$w_{10} x^{10}$

Locally-weighted regression is the first of what we call non-parametric methods or data-based or lazy methods. We use a simple model (usually linear in the parameters) that is valid near a point of interest.



Unweighted regression gives points x_i that are far from a point of interest q (query point) the same weight as points close to the query point.

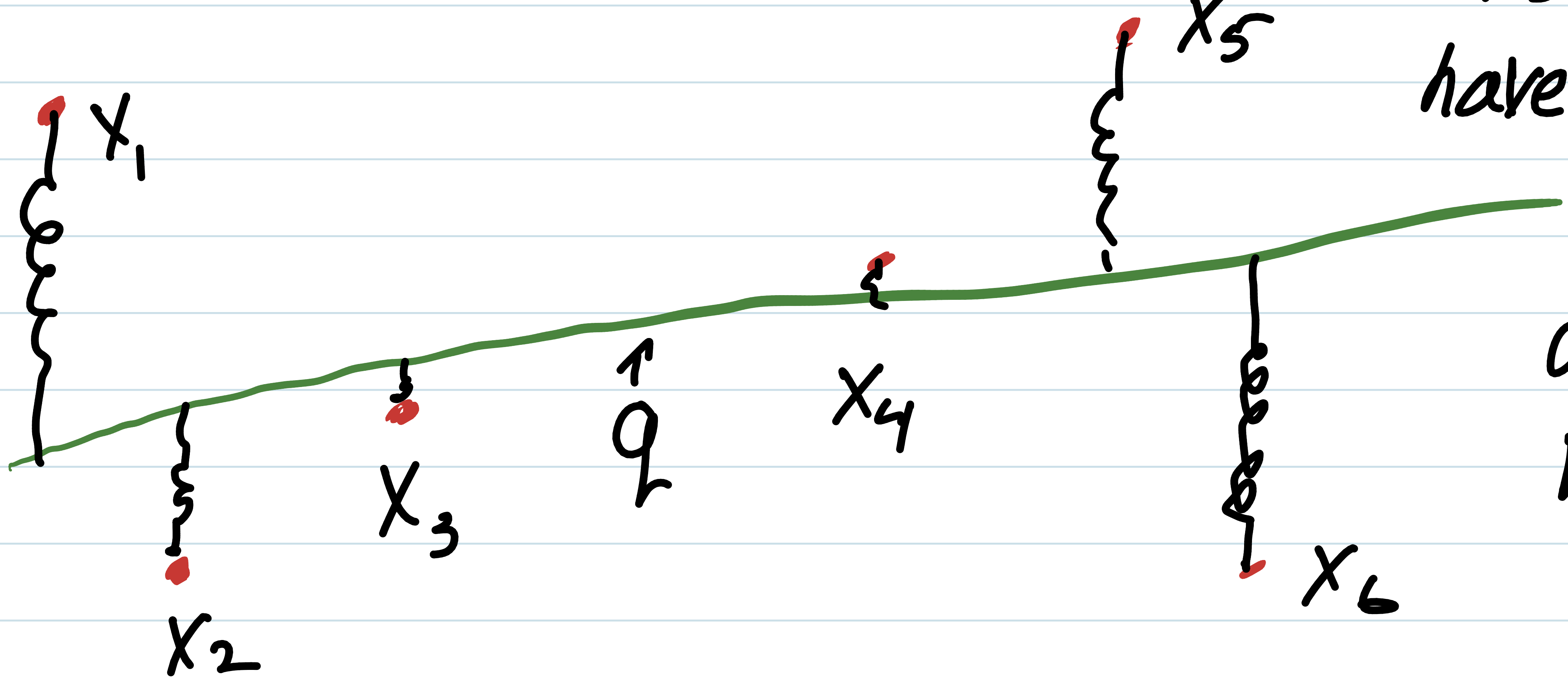
Example: all data points have springs with equal stiffness



unweighted regression cost function

$$C = \sum_i \Big(\underbrace{w^T x_i}_{\text{linear model}} - \underbrace{t_i}_{\text{measured output}}\Big)^2$$

Alternatively we might give more weight to points close to the query point.



x_3 and x_4 have stiffer springs and greater weight

locally weighted cost function

$$C(q) = \sum_i (w^T x_i - t_i)^2 K(d(x_i, q))$$

$K(\ )$ is called a weighting or kernel function (spring stiffness)

$d(x_i, q)$ is called a distance function

Just like with unweighted regression we want to pick w
to minimize the cost.

Let's define the weight $s_i = K(d(x_i, q))$

For input vector $x \in \mathbb{R}^k$ and n data points

$$\underline{X} = \begin{bmatrix} x_{11} & \dots & x_{1k} \\ & \vdots & \\ x_{n1} & \dots & x_{nk} \end{bmatrix}$$

← individual input vector

data matrix

and $t = [t_1 \ \dots \ t_n]^T$

Define z_i as the product of s_i and x_i

$$z_i = s_i x_i$$ weighted input

and $\underline{Z} = S\underline{X}$ where $S_{ii} = s_i$ on the diagonals

weighted input matrix $\in \mathbb{R}^{n \times k}$ and 0 everywhere else

$$S = \begin{bmatrix} s_1 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & s_2 & & & & & & 0 \\ 0 & & s_3 & & & & & 0 \\ 0 & & & s_4 & & & & 0 \\ 0 & & & & \ddots & & & 0 \\ \vdots & & & & & & & \vdots \\ 0 & & & & & & & s_n \end{bmatrix}$$

$$\underset{n \times k}{\underline{Z}} = \underset{n \times n}{S} \times \underset{n \times k}{\underline{X}}$$

$$V_i = s_i t_i$$ weighted output

$$V = S\vec{t}$$ weighted output vector $\in \mathbb{R}^n$

Then just like with unweighted regression we get a prediction of the output.

$$\hat{y}(q) = q^T \underbrace{\underbrace{(Z^TZ)^{-1}Z^T}_{\text{weighted input terms}} \underbrace{V}_{\text{weighted outputs}}}_{\text{parameters}}$$

mean of predictive distribution

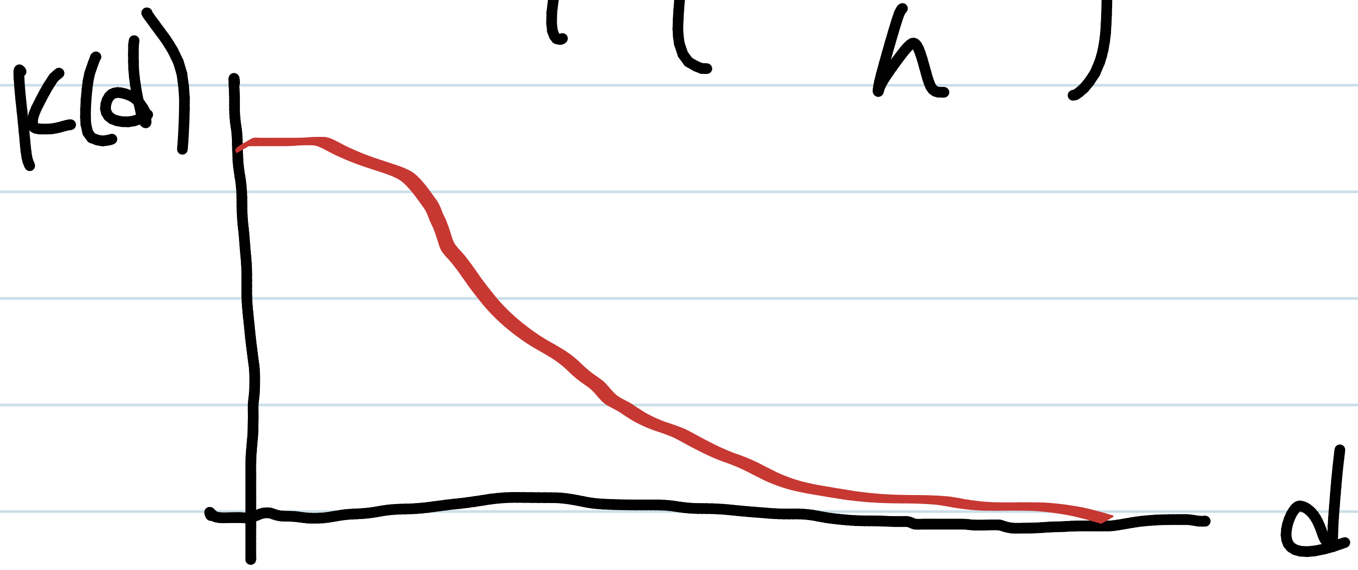
query input

Distance function: typically Euclidean

$$d_E(x, q) = \sqrt{(x-q)^T(x-q)}$$

Kernel or weighting function

$$K(d) = \exp\left(\frac{-d^2}{h}\right)$$



Gassian Weighting

h is a parameter to tune (scaling factor)

small h wiggly data

big h smooth data

Estimate the variance

Training data come from a stochastic process

$$y_i = f(x_i) + N(0, \sigma^2)$$

The estimate is

$$\underbrace{\hat{y}(q)}_{\text{estimated output}} = \underbrace{\left[(Z^T Z)^{-1} Z^T S y\right]^T}_{\text{parameter vector}} \underbrace{q}_{\text{query input}} = b_q^T \bar{y} = \sum_{i=1}^{n} b_i(q) y_i$$

$$E[\hat{y}(q)] = E[b_q^T \bar{y}] = b_q^T E[\bar{y}] = \sum_{i=1}^{n} b_i(q) f(x_i)$$

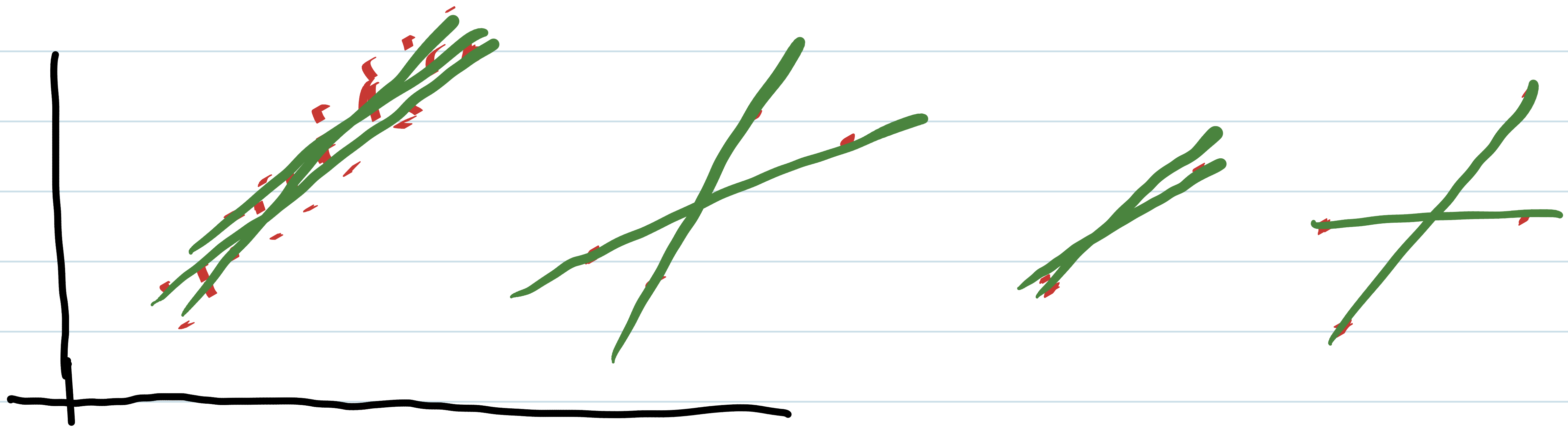
$$Var[\hat{y}(q)] = E[\hat{y}(q) - E[\hat{y}(q)]]^2 = \sum_{i=1}^{n} s_i^2 \sigma^2$$

The variance of the actual value of y.

$$Var[y_{new}(q)] = \sigma^2 + \sigma^2 b_q^T b_q$$

$$\mathrm{Var}[y_{new}(q)] = \sigma^2 + \underbrace{\sigma^2 b_q^T b_q}_{\text{Variance in my best estimate}}$$

Variance in the process



We need an estimate of $\sigma^2(q)$

$$\sigma^2(q) = \frac{\sum_{i=1}^{n} r_i(q)}{n_{LWR} - p_{LWR}}$$

$$r_i = \overbrace{W^T(q) z_i(q)}^{\hat{y}} - v_i(q)$$

Weighted error

$$n_{LWR} = \sum_{i=1}^{n} s_i^2$$

weighted # of samples

$$p_{LWR} = \sum_{i=1}^{n} s_i^2 \, z_i^T (Z^T Z)^{-1} z_i$$ weighted number of parameters

How to do LWR

You need

1) a local model $w^T x$

2) a distance function

3) a kernel function

The right choice for any of the three depends on the other two.

Local models: Usually a polynomial function of the input

$y = w_0$ constant

$y = w_0 + w_1 x$ linear

$y = w_0 + w_1 x + w_2 x^2$ quadratic

The model should capture the curvature of the local data.

How "local" depends on the distance & kernel functions.

Distance functions

unweighted Euclidean distance

$$d_E(x,q) = \sqrt{\sum_i (x_i - q_i)^2} = \sqrt{(x-q)^T (x-q)}$$

diagonally weighted Euclidean distance

$$d_m(x,q) = \sqrt{\sum_j \left(m_j(x_j - q_j)\right)^2} = \sqrt{(x-q)^T M^T M (x-q)}$$

$$M = \begin{bmatrix} m_1 & 0 & 0 \\ 0 & m_2 \ddots & 0 \\ 0 & 0 & m_\ell \end{bmatrix}$$

diagonal matrix where m_j is a weight for the j^{th} input dimension

You have to pick m_j. Its a tuning hyperparameter.

fully-weighted Euclidean distance

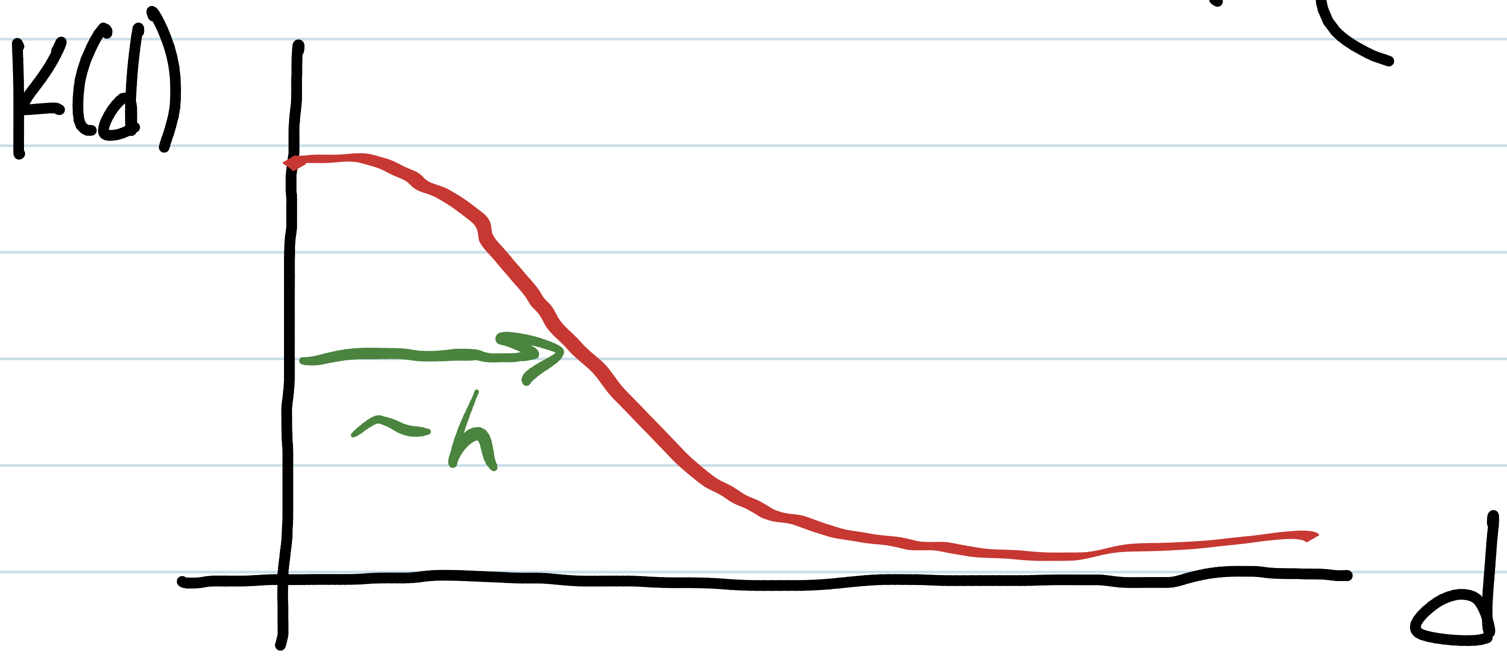
$$d_m(x,q) = \sqrt{(x-q)^T M^T M (x-q)}$$

where M is an arbitrary matrix

Kernel Functions

1) Gaussian $K(d) = \exp\left(\frac{-d^2}{h}\right)$

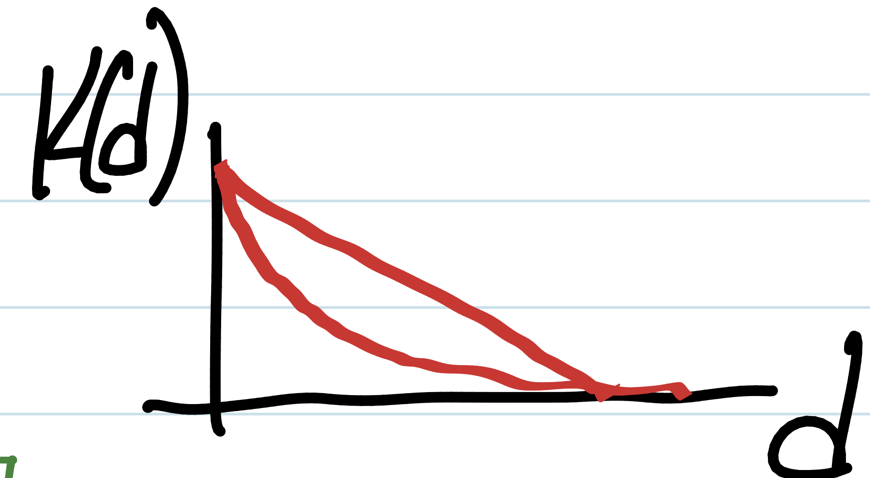
h is a bandwidth hyper-parameter to be tuned.



2) Inverse distance

$K(d) = \frac{1}{d^m}$ or $\boxed{K(d) = \frac{1}{1+d^m}}$

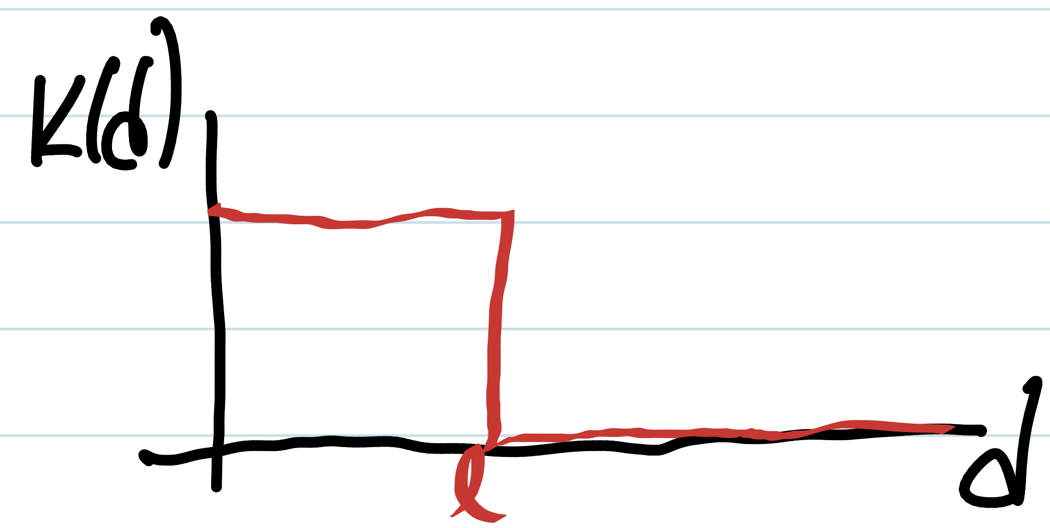
$= \infty$ at $d = 0$



eg $\frac{1}{d}$, $\frac{1}{d^2}$ $\frac{1}{d^3}$

3) Uniform distance $K(d) = \begin{cases} 1 & \text{if } |d| < \ell \\ 0 & \text{otherwise} \end{cases}$

disregards far off data



Picking hyperparameters M and h

M pick based on relative range of input dimensions

e.g $x = [\theta \ \dot{\theta}]$ $\quad \theta \in [0 \ 2\pi]$

$\dot{\theta} \in [-4\pi \ 4\pi]$

might pick $M = \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix}$

h pick to minimize cross-validation error

What is cross-validation error?

leave-one-out cross-validation error (LOOCV)

if you have n data points, use $n-1$ to predict the other remaining point and repeat leaving out a different point each time.

$$MSE^{cv} = \sum_{i=1}^{n} \underbrace{\left(e_j^{cv}\right)^2}_{\text{cross validation error for a specific left-out data point}}$$

Single Pendulum example

output: τ

input: $x = [1 \ q \ \dot{q} \ \ddot{q}]^T$

pick a linear model

$$y = w_0 + w_1 q + w_2 \dot{q} + w_3 \ddot{q} = w^T x$$

pick a Gaussian Kernel

$$K(d) = \exp\left(\frac{-d^2}{h}\right)$$

pick a Euclidean distance function

$$d(x, q) = \sqrt{(x-q)^T M^T M (x-q)}$$

pick M to scale

$1 \quad q \quad \dot{q} \quad \ddot{q}$

How to pick a query? It depends.

For robot trajectory following you might have a desired trajectory and want to compute torques to move along the trajectory.