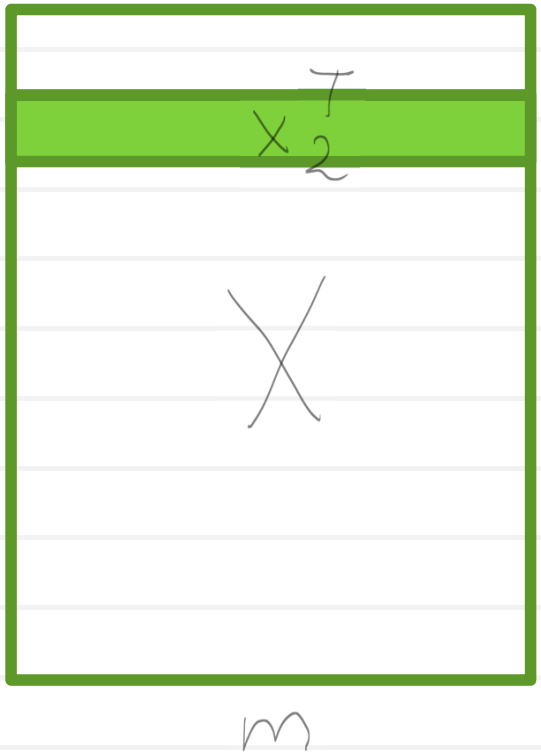
Lecture 10: Least Squares

Linear least squares setting

Least squares the most widely used data fitting model in statistics, science, engineering, etc.

Data matrix (aka observation matrix):



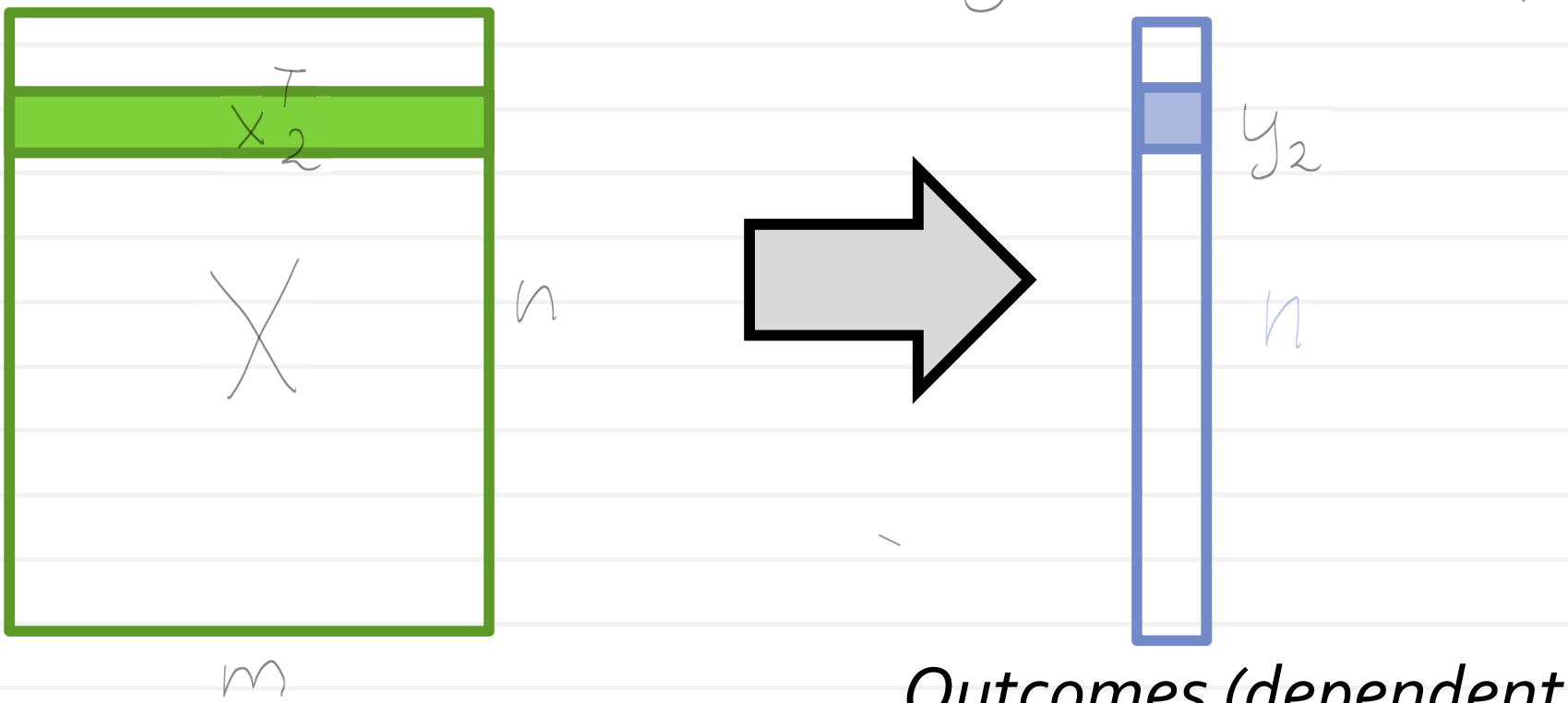
$$X = [x_1 \; x_2 \ldots x_n]^T$$

$$x_i \in \mathbb{R}^m$$

a vector of input variables (*observations*, *independent variables*, *features*, *predictors*, *regressors*)

Linear least squares setting

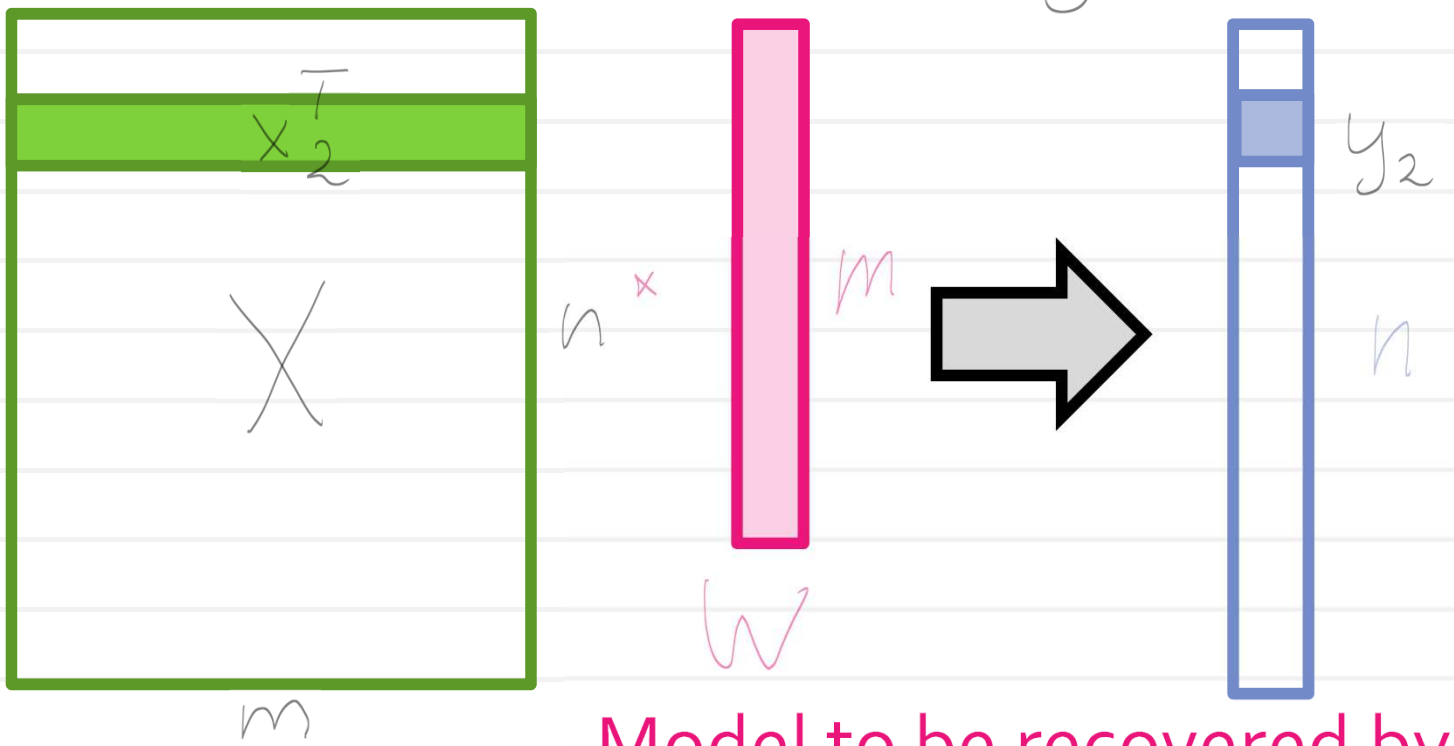
$$X = [X_1 \; X_2 \; \dots \; X_n]^T \qquad y = [y_1 \; y_2 \; \dots \; y_n]^T \in \mathbb{R}^n$$



Outcomes (dependent variables, response)

Linear least squares setting

$X = [x_1 \; x_2 \ldots x_n]^T$ $\quad y = [y_1 \; y_2 \ldots y_n]^T \in \mathbb{R}^n$



Model to be recovered by optimization.

$x_1^T w = y_1$

$x_2^T w = y_2$

$\ldots$

$x_n^T w = y_n$

$X w = y +$ "noise"

Linear least squares setting

$X w = y +$ "noise"

w that gives the best fit (in the *least-quadratic sense*) is then recovered by:

$$\min_w \frac{1}{2} \|Xw - y\|_2^2 = F(w)$$

$$\min_w \frac{1}{2} \sum_{i=1}^{n} (x_i^T \cdot w - y_i)^2$$

Residual vector:

$$r(w) = \begin{bmatrix} x_1^T w - y_1 \\ \vdots \\ x_n^T w - y_n \end{bmatrix}$$

We can rewrite:

$$\min_w \frac{1}{2} \|r(w)\|_2^2$$

Computer tomography

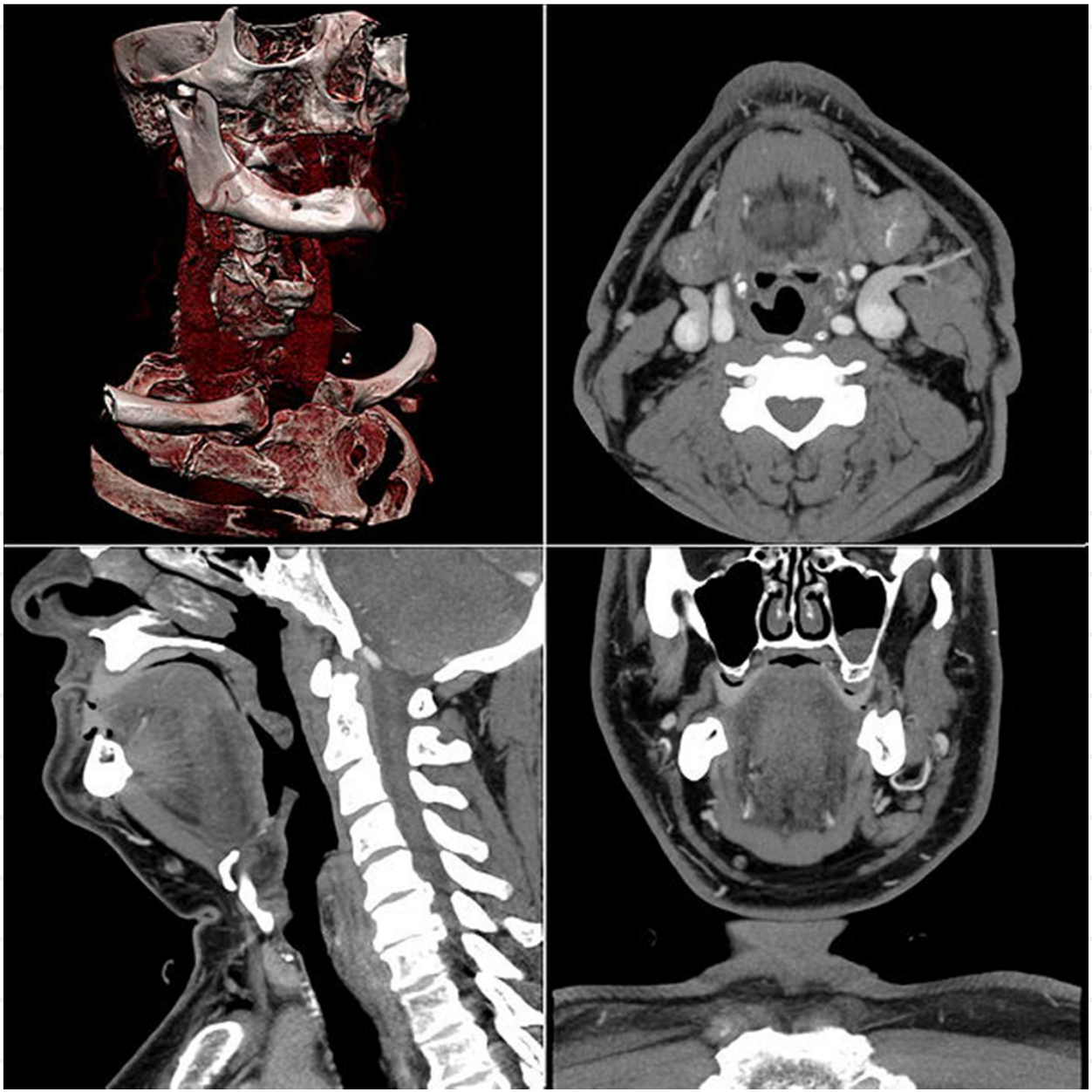


Image from wikipedia

Computer tomography

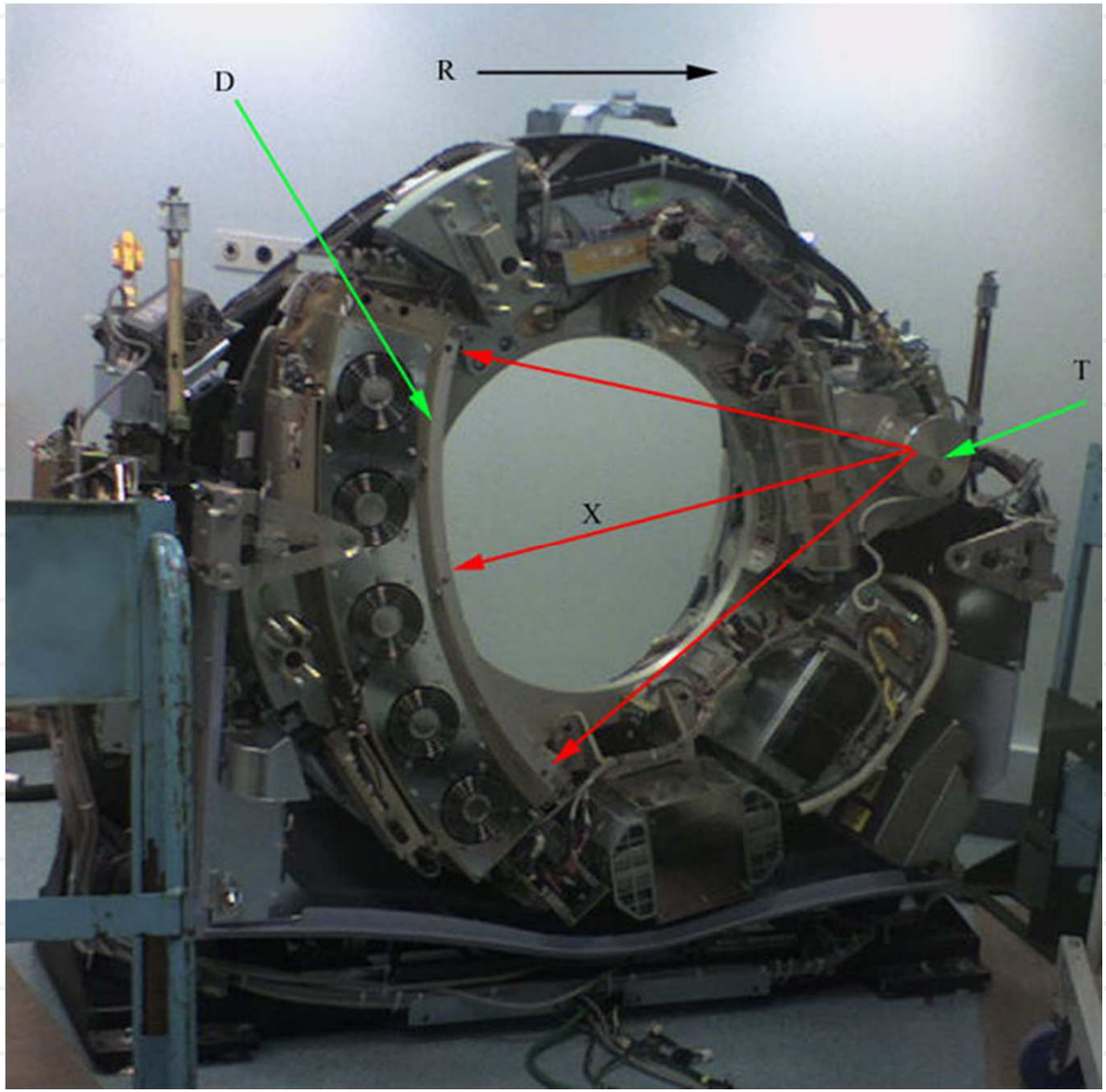
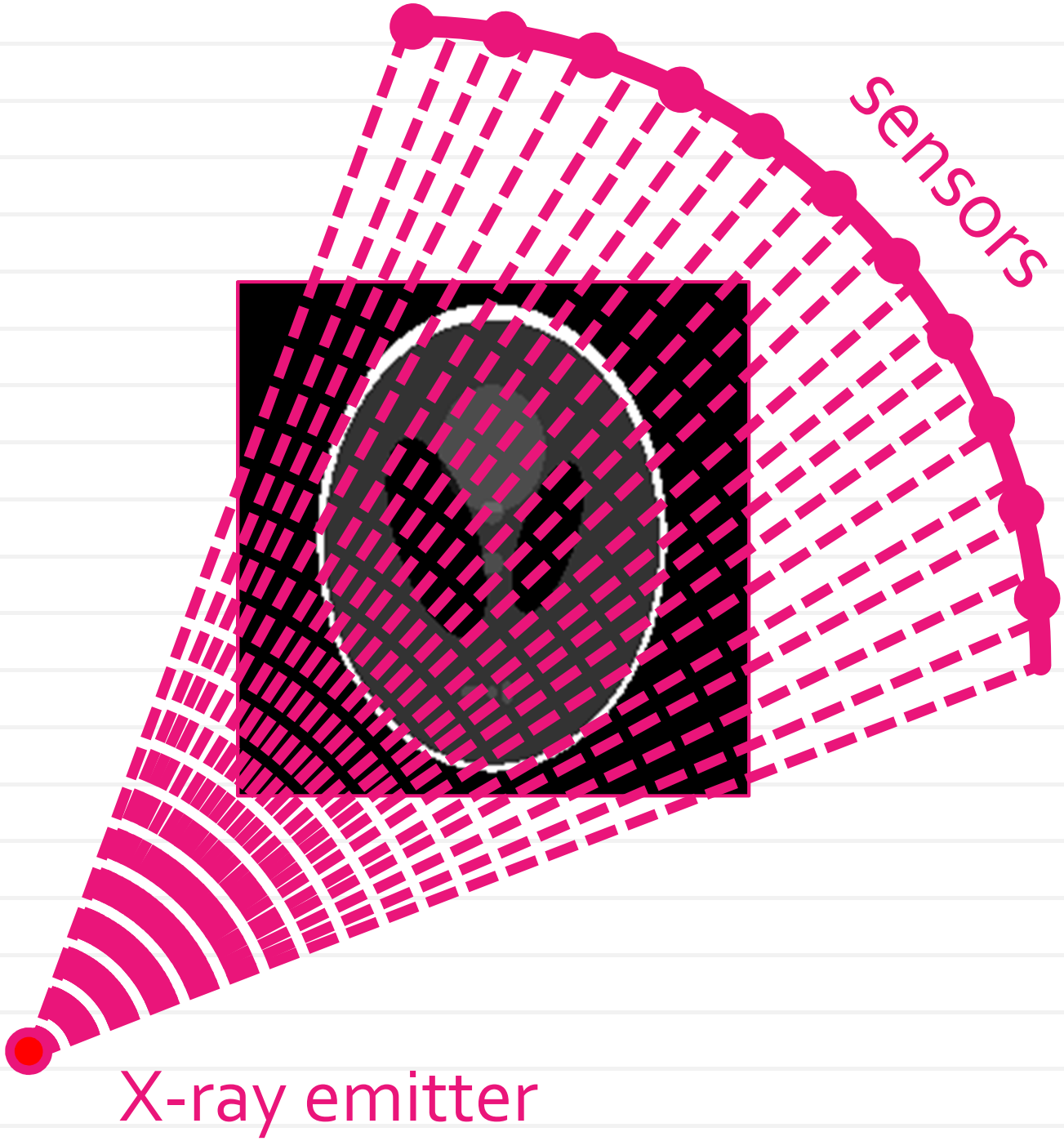
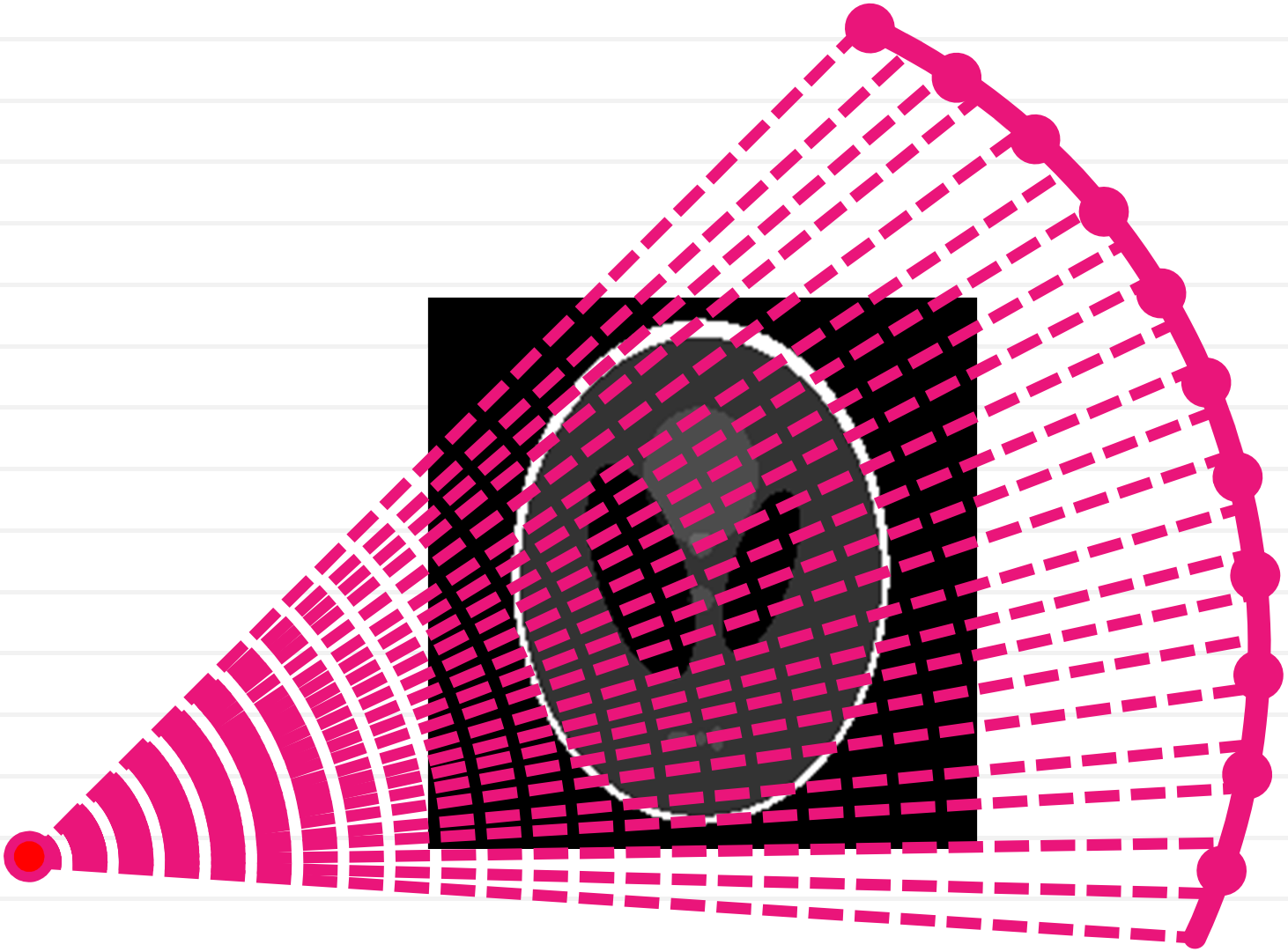


Image from wikipedia

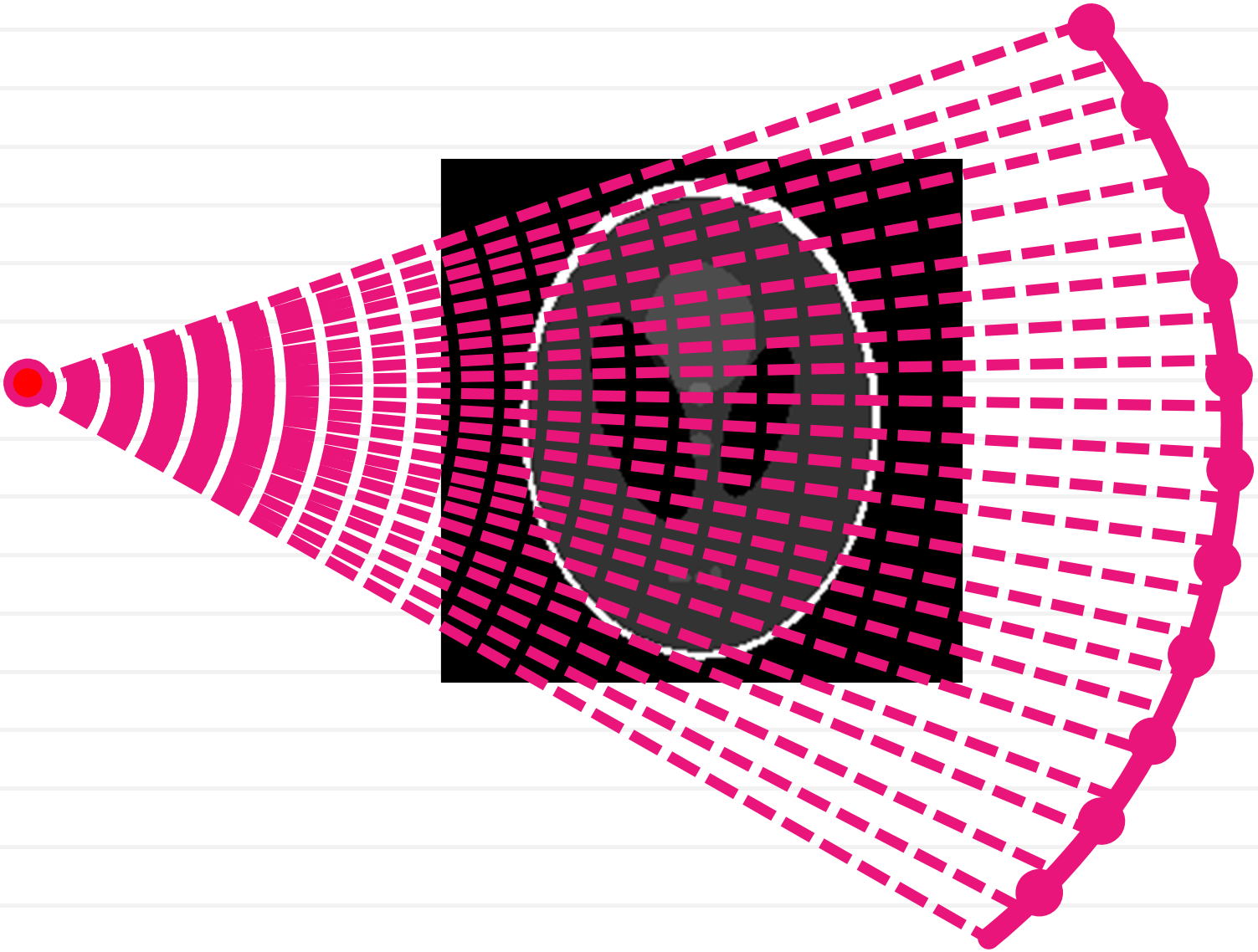
Computer tomography



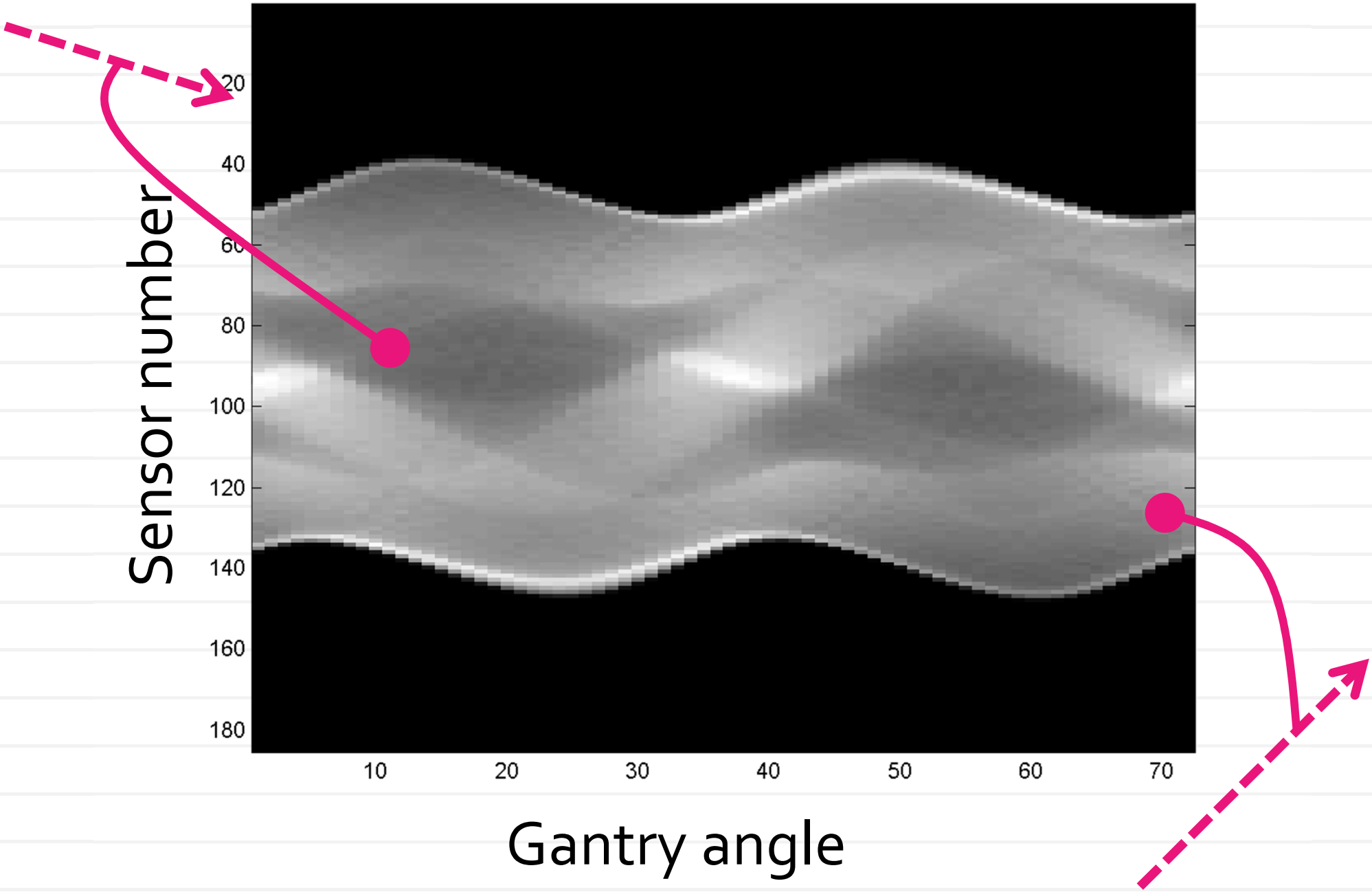
Computer tomography



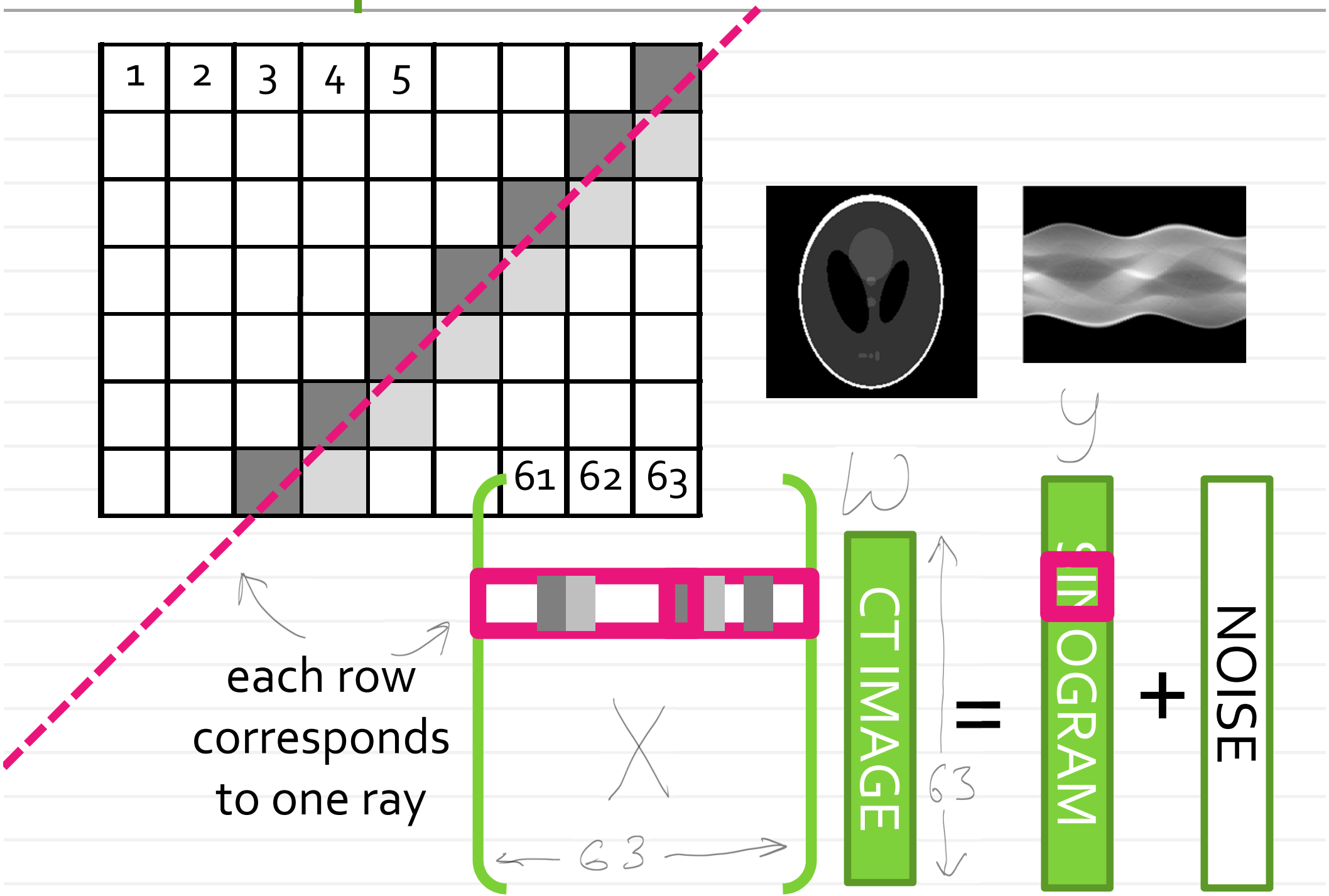
Computer tomography



A sinogram


Sensor number
20
40
60
80
100
120
140
160
180
10
20
30
40
50
60
70
Gantry angle

The least squares model



Finding the minimum

$$\min_{w} \frac{1}{2} \| Xw - y \|_2^2$$

Differentiating w.r.t. w and setting the gradient to zero, we get:

$$(Xw - y)^T X = 0$$

After algebra, we get *normal equations*:

$$X^T X w = X^T y$$

If normal equations are *full rank*, then:

$$w = (X^T X)^{-1} X^T y$$

Adding non-linearity

We can adapt linear least squares to fitting non-linear functions, by transforming the input x non-linearly:

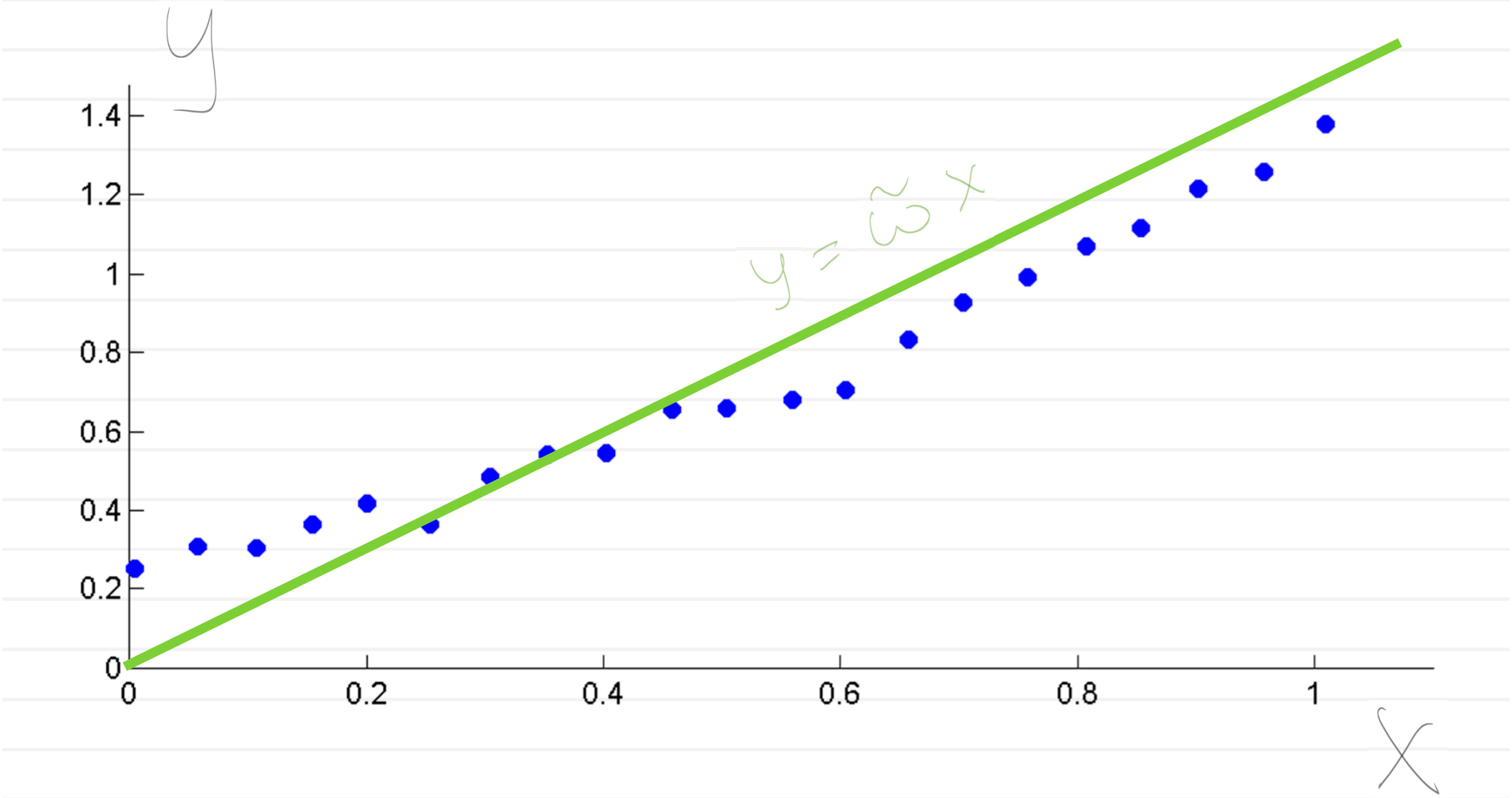
$$\mathbb{R}^{m'} \ni x \longrightarrow \phi(x) \in \mathbb{R}^{m}$$

Our data matrix then becomes:

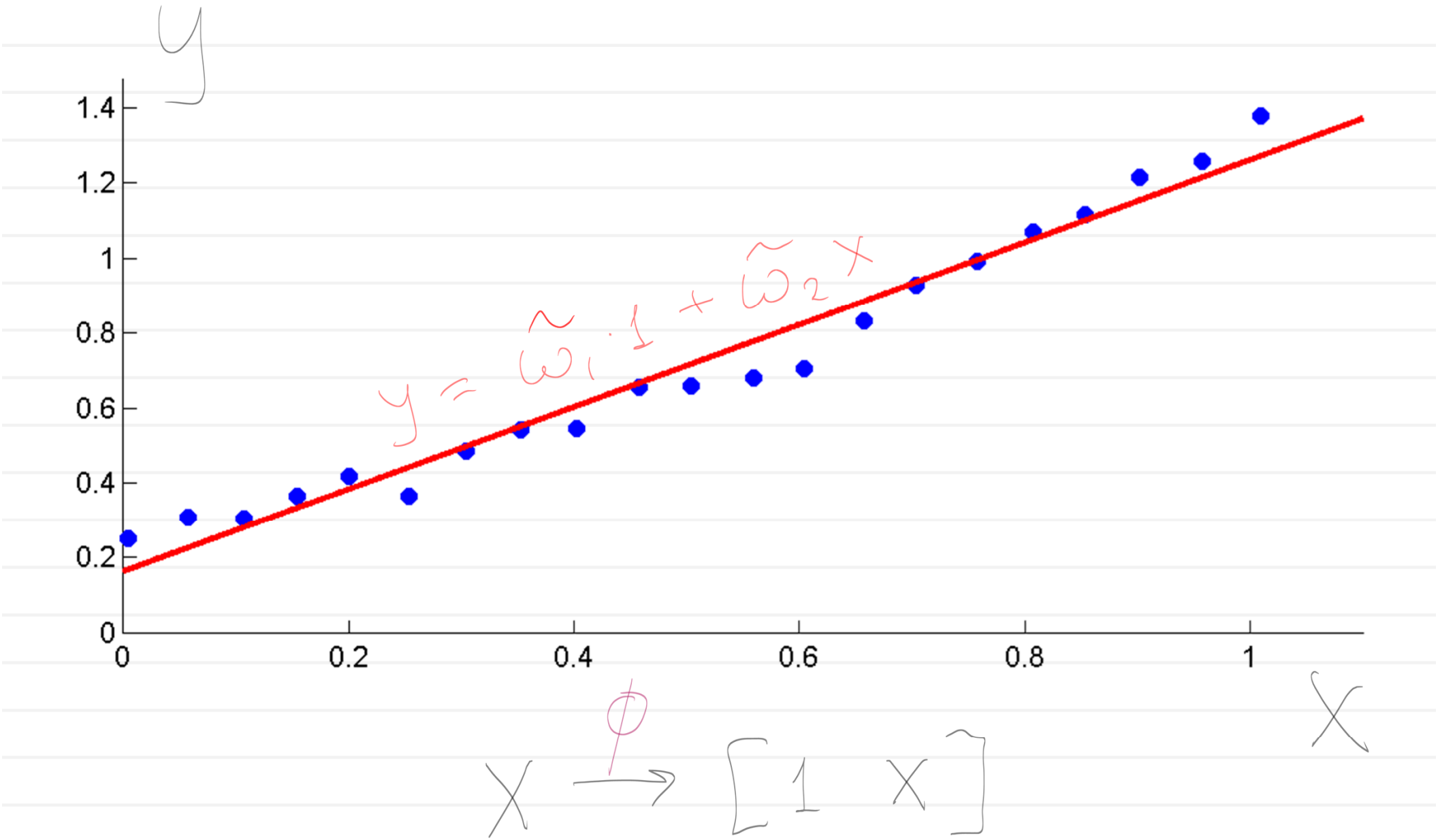
$$X = \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_m(x_1) \\ & \cdots & & \\ \phi_1(x_n) & \phi_2(x_n) & \cdots & \phi_m(x_n) \end{bmatrix}$$

Then, linear least squares proceed as before.

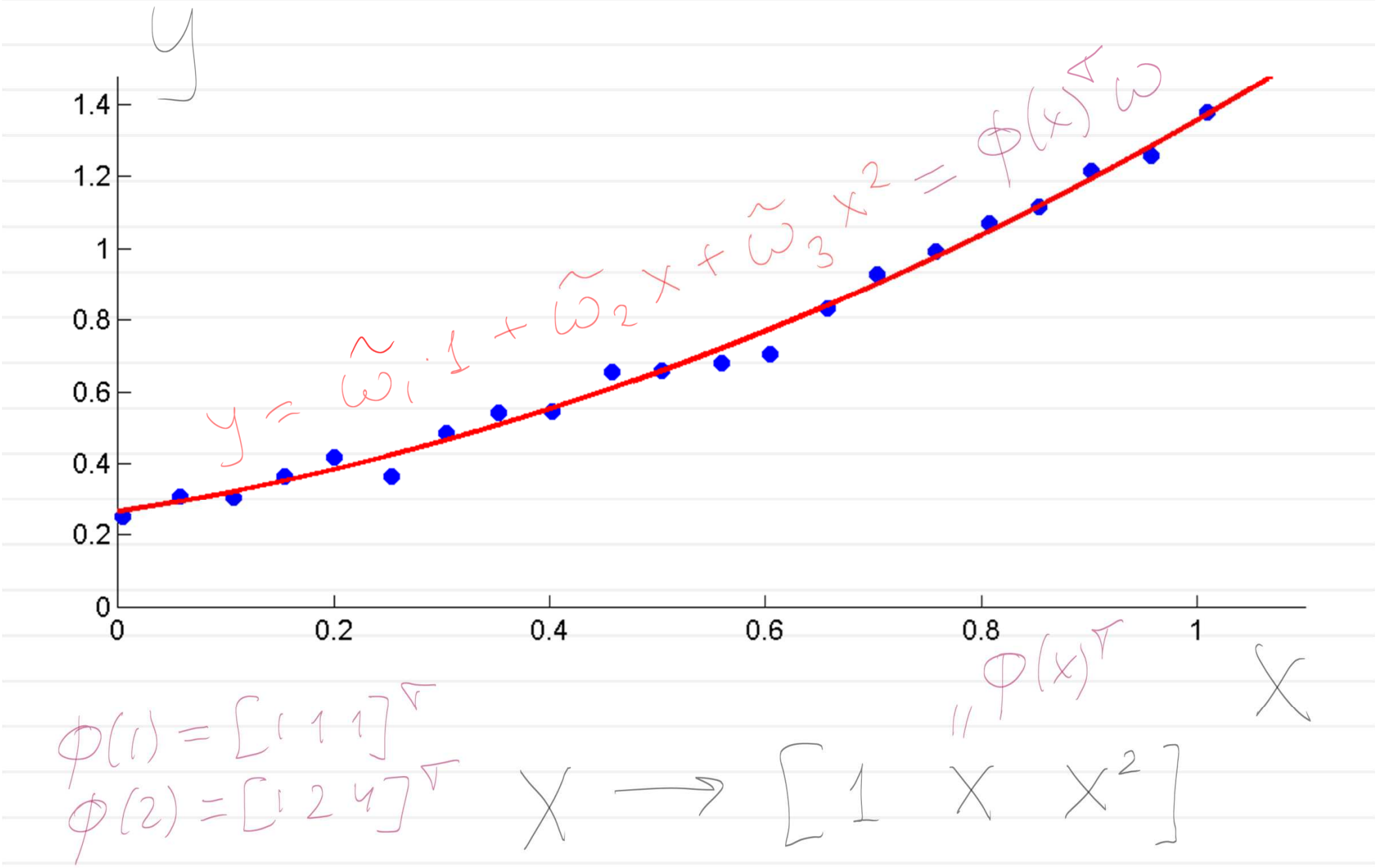
Example fitting the data



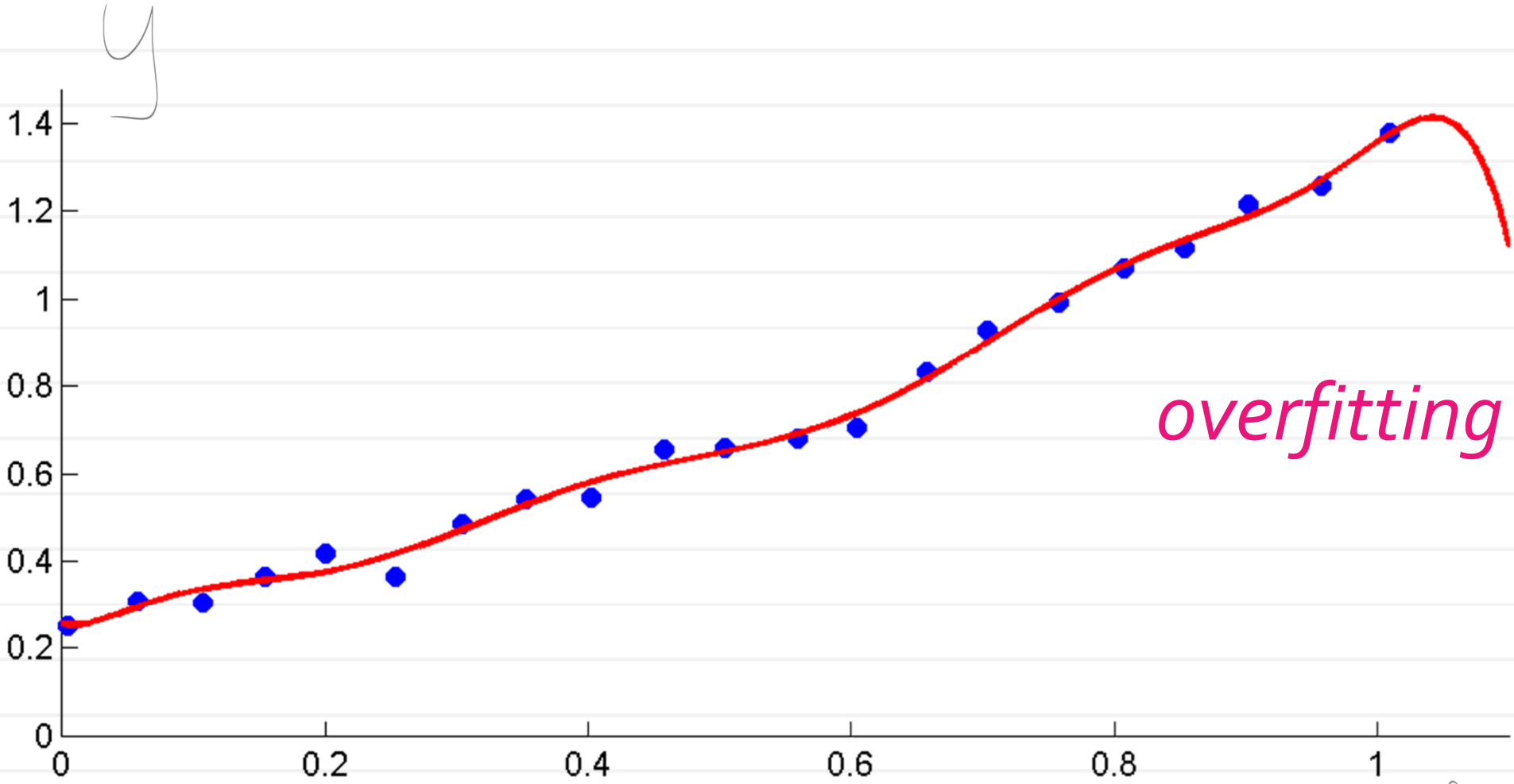
Example fitting the data



Example fitting the data



Example fitting the data



$x \rightarrow [1 \quad x \quad \ldots \quad x^{10}]$

Choosing the right model

The most straight-forward and commonly used method is *validation*:

1. Split data into training and validation
2. Fit all models on training
3. Check how well do they perform on validation
4. Pick the best performer
5. Reestimate the model using full set

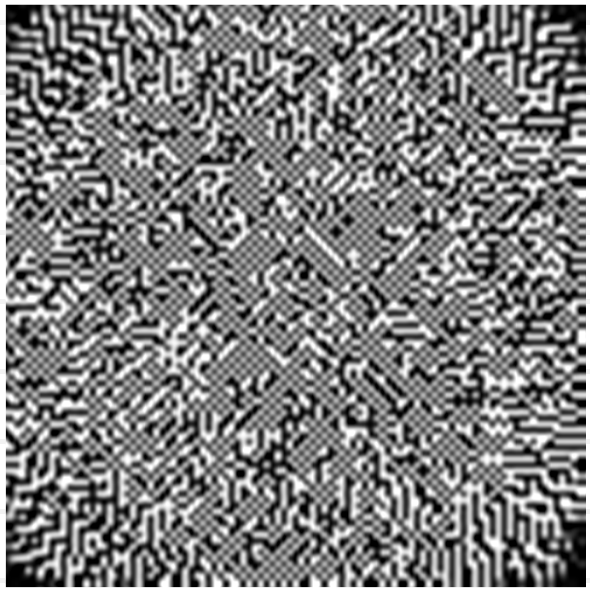
Overfitting in computer tomography

Tomographic reconstruction suffers from overfitting as well:

"True" phantom used to generate the data:

Reconstruction (no noise):

Reconstruction with noise:



(Not fully) Bayesian approach

Our model (in the linear case):

$$X w = y + \text{“noise”}$$

Let us look at: $p(w \mid X, y)$

Using Bayes rule:

$$p(w \mid X, y) = \frac{1}{p(X, y)} p(X, y \mid w)\, p(w)$$

posterior *likelihood* *prior*

We will now model the factors and then try to find w that maximizes the probability.

(Not fully) Bayesian approach

$$\max_{\omega} P(\omega \mid X, y)$$

$$\max_{\omega} \frac{1}{P(X,y)} P(X,y \mid \omega) P(\omega)$$

$$\min_{\omega} -\log P(X,y \mid \omega) - \log(P(\omega))$$

log-likelihood *log-prior*

Different choices of log-likelihood and log-prior give rise to different regression methods with different statistical and computational performance

Tikhonov-regularized least-squares

$$X w = y + \text{“noise”}$$

$$\min_w -\log P(X, y \mid w) - \log(P(w))$$

log-likelihood *log-prior*

We assume that the noise is independent between measurements and Gaussian. Then our likelihood is:

$$P(X, y \mid w) \propto \prod_i \exp\left(-\frac{\| y_i - x_i^T w \|_2^2}{2\sigma^2}\right)$$

We assume that w is drawn from an isotropic zero-mean Gaussian prior:

$$w \sim \frac{1}{Z} \exp\left(-\lambda \| w \|_2^2\right)$$

Thus we get:

$$\min_w \frac{1}{2} \| Xw - y \|_2^2 + \frac{\mu}{2} \| w \|_2^2 \qquad \mu = 2\lambda\sigma^2$$

Finding the minimum

$$\min_{\omega} \frac{1}{2} \|X\omega - y\|_2^2 + \frac{1}{2}\mu \|\omega\|_2^2$$

Differentiating w.r.t. w and setting the gradient to zero, we get:

$$(X\omega - y)^T X + \mu I \omega = [0 \; \dots \; 0]$$

$$(X\omega - y)^T X + \mu I \omega = 0$$

After algebra, we get *normal equations*:

$$(X^T X + \mu I)\omega = X^T y$$

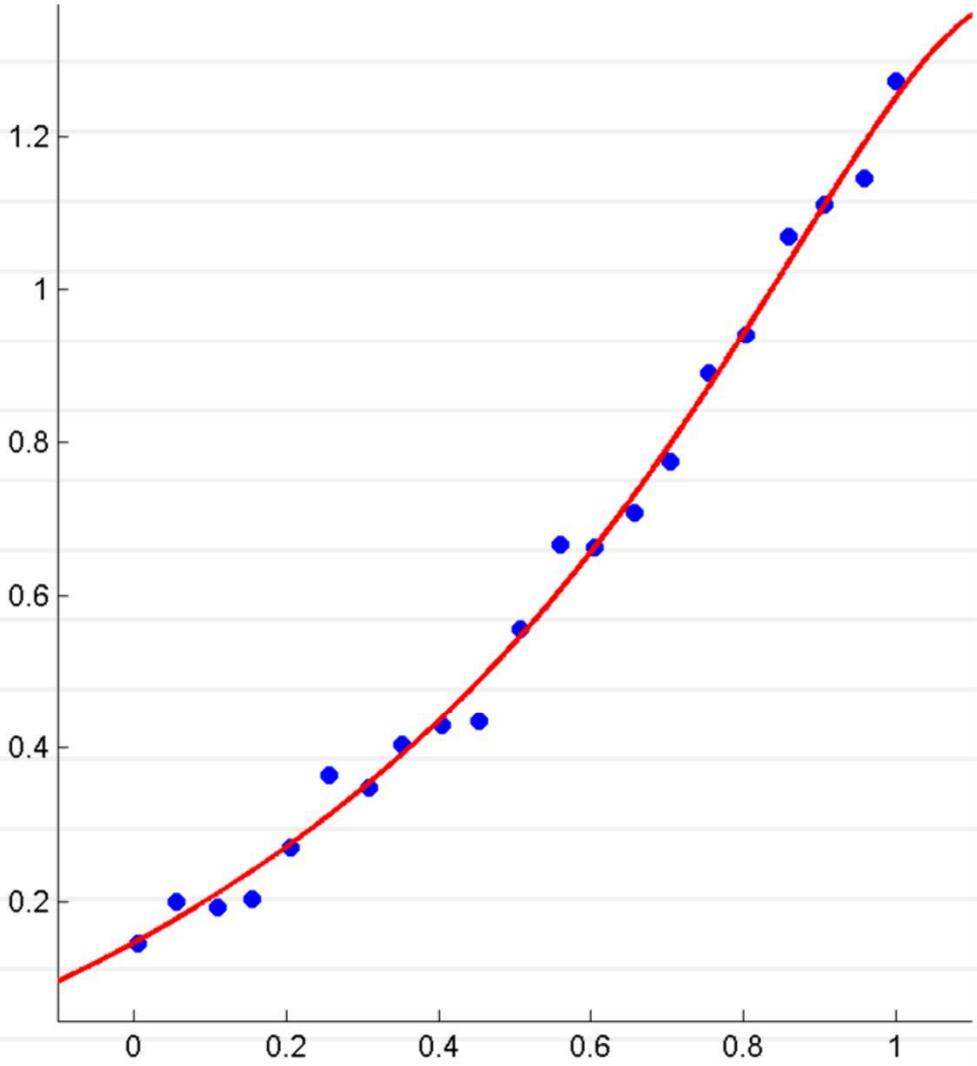
We can express *w* as:

$$\omega = (X^T X + \mu I)^{-1} X^T y$$

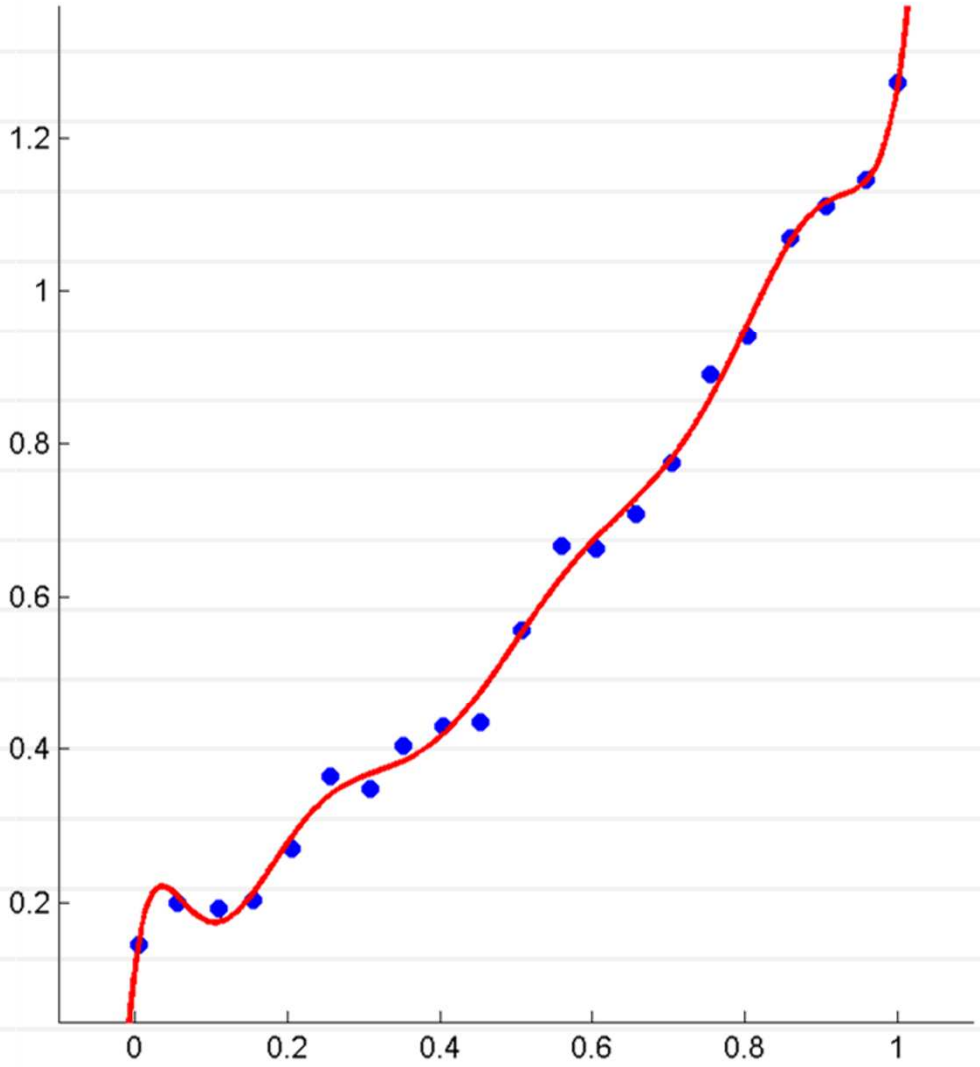
Fitting with regularization

Fitting degree 10 polynomial:

$x \rightarrow [1 \quad x \quad \ldots \quad x^{10}]$

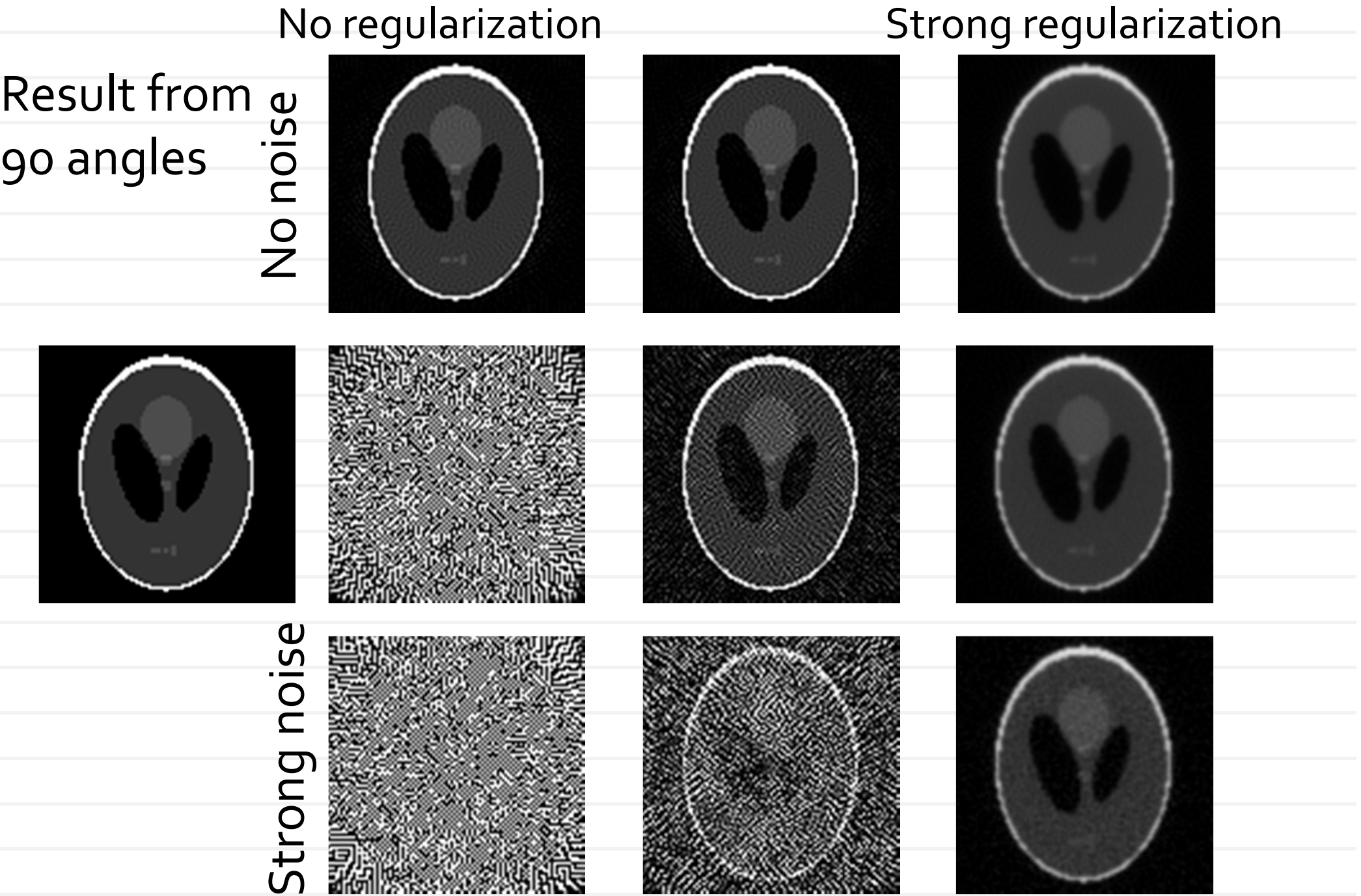


$+ 0.01 \; \|w\|_2^2$



No regularization

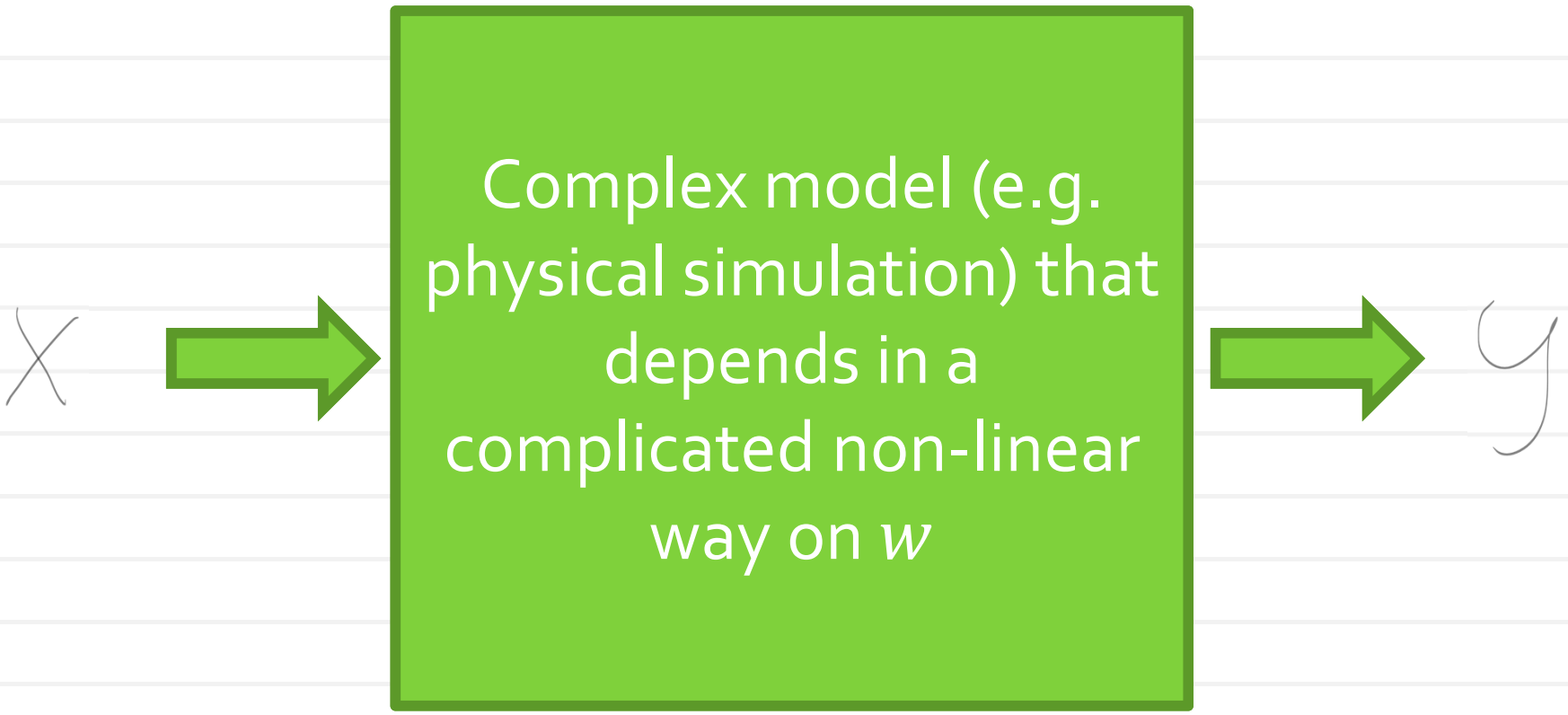
Tomography with regularization



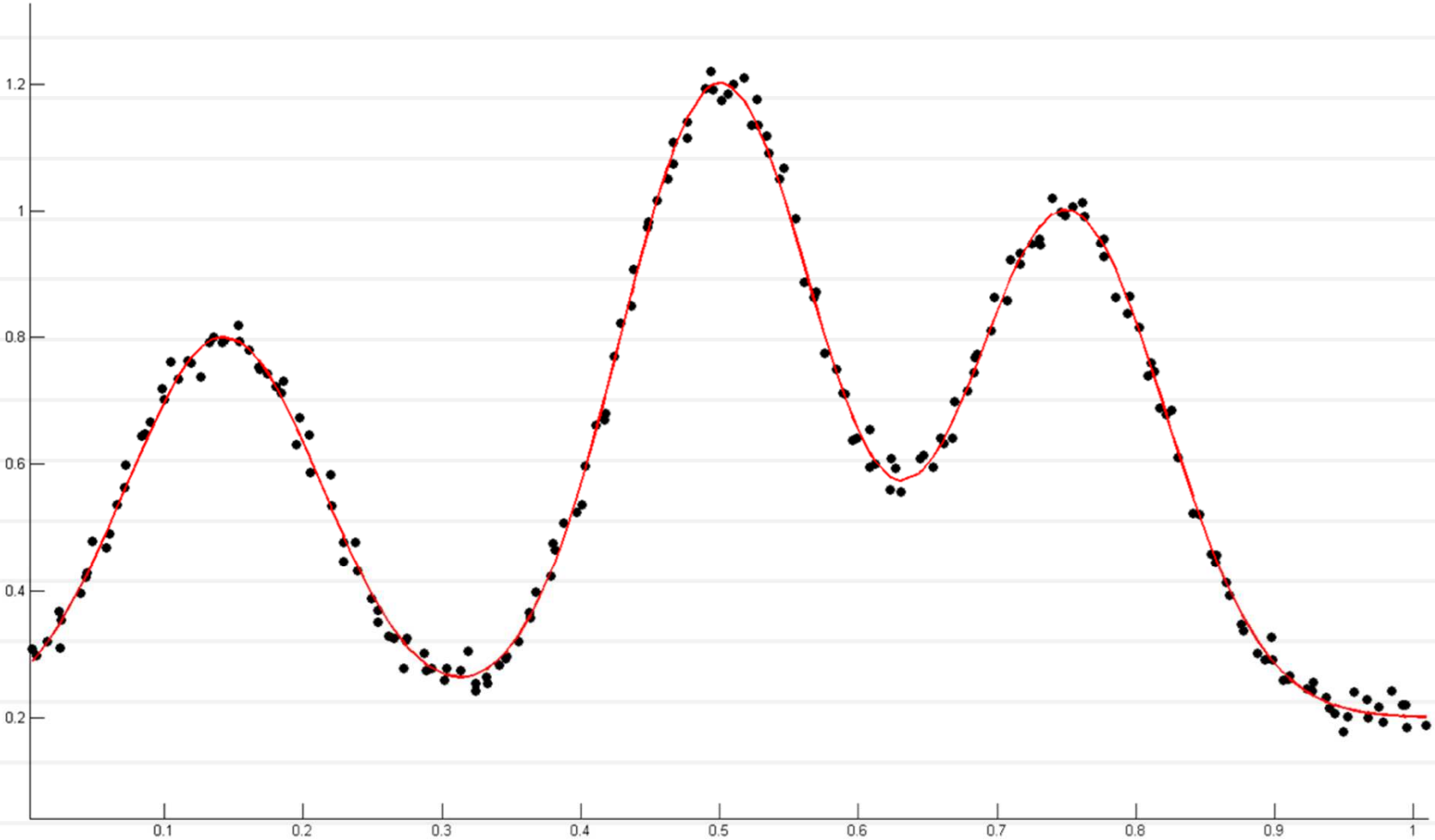
Non-linear least squares

Typical usage scenario:

$f(x; w)$



Nonlinear curve fitting



$$f(x;\omega) = \omega_1 e^{-100(x-\omega_2)^2} + \omega_3 e^{-100(x-\omega_4)^2} + \omega_5 e^{-100(x-\omega_6)^2}$$

Nonlinear curve fitting

$$f(x;\omega) = \omega_1 e^{-100(x-\omega_2)^2} + \omega_3 e^{-100(x-\omega_4)^2} + \omega_5 e^{-100(x-\omega_6)^2}$$

Assume that the noise is Gaussian, then to recover *w* we need to solve:

$$\min_{\omega} \frac{1}{2} \sum_{i=1}^{n} \left(y_i - f(x_i;\omega) \right)^2 = F(\omega)$$

$$\min_{\omega} \frac{1}{2} \| r(\omega) \|^2 \qquad r(\omega) = \begin{bmatrix} f(x_1;\omega) - y_1 \\ \dots \\ f(x_n;\omega) - y_n \end{bmatrix}$$

Non-linear least squares

$$\min_{\omega} \frac{1}{2} \sum_{i=1}^{n} \left(y_i - f(x_i; \omega) \right)^2$$

The gradient of the objective function is:

$$\frac{dF(\omega)}{d\omega} = \nabla F(\omega) = J^T r(\omega)$$

, where the Jacobian J is:

$$J = \begin{bmatrix} \frac{\partial f(x_1;\omega)}{\partial \omega_1} & \cdots & \frac{\partial f(x_1;\omega)}{\partial \omega_m} \\ \frac{\partial f(x_n;\omega)}{\partial \omega_1} & \cdots & \frac{\partial f(x_n;\omega)}{\partial \omega_m} \end{bmatrix}$$

Non-linear LS : gradient descent

We can try a standard gradient descent. We use a first order approximation in the vicinity of the current estimate w^t:

$$F(\omega_t + \Delta\omega) = F(\omega_t) + \nabla F(\omega_t)^T \Delta\omega$$

We then do steps along the gradient according some schedule s_t (or using line search):

$$\omega_{t+1} = \omega_t - s_t \, J(\omega_t)^T r(\omega_t)$$

Reminder: $\nabla F(\omega) = J(\omega)^T r(\omega)$

Non-linear LS: second-order method

Gradient descent is slow. Instead, we can get much faster convergence by looking at a second-order approximation:

$$F(\omega_t+\Delta\omega) = F(\omega_t) + \nabla F(\omega_t)^T \Delta\omega + \frac{1}{2}\Delta\omega^T \nabla^2 F(\omega_t)\Delta\omega$$

Reminder: $\nabla F(\omega) = J(\omega)^T r(\omega)$

Let us estimate the matrix of second order derivatives:

$$\nabla(\nabla F(\omega)) = \frac{d\, J(\omega)^T}{d\omega}\cdot r(\omega) + J(\omega)^T J(\omega)$$

Or, in more details:

$$\left[\nabla^2 F(\omega)\right]_{i,j} = \sum_k \frac{\partial^2 f(x_k;\omega_t)}{\partial\omega_i\cdot\partial\omega_j} r_k(\omega_t) + \sum_k \frac{\partial f(x_k;\omega_t)}{\partial\omega_i}\cdot\frac{\partial f(x_k;\omega_t)}{\partial\omega_j}$$

Simplifying the second-order term

$$\left[\nabla^2 F(\omega)\right]_{i,j} = \sum_k \frac{\partial^2 f(x_k;\omega_t)}{\partial\omega_i \cdot \partial\omega_j} r_k(\omega_t) + \sum_k \frac{\partial f(x_k;\omega_t)}{\partial\omega_i} \cdot \frac{\partial f(x_k;\omega_t)}{\partial\omega_j}$$

- Computing green is much easier than red (*n* first order derivatives vs n^2 second-order derivatives per each square)
- The second-order term is approximated with the green term only (red term is omitted).
- This approximation is better for almost linear functions (second derivatives are small) or close to a good minimum (the residual vector is small).

Properties of the approximate Hessian

$$\nabla^2 F(\omega) \simeq \frac{d J(\omega)^T}{d\omega} \cdot r(\omega) + J(\omega)^T J(\omega)$$

- Easily computable
- Easily invertible (when Jacobian is sparse)

$$\nabla^2 F(\omega) \cdot \Delta\omega = -\nabla F(\omega)$$

- Positive (semi)-definite
- We are replacing one approximation with another

Non-linear LS: Gauss-Newton method

At each step, the reduced second-order approximation is:

$$F(\omega_t + \Delta\omega) = F(\omega_t) + \nabla F(\omega_t) \cdot \Delta\omega + \frac{1}{2} \Delta\omega^T J(\omega_t)^T J(\omega_t) \Delta\omega$$

Equate the gradient to zero:

$$J(\omega_t)^T r(\omega_t) + J(\omega_t)^T J(\omega_t) \cdot \Delta\omega = 0$$

A non-linear version of **normal equations** (consider linear normal equations for a linear function f(x, w) = $x^T w$ at the origin)

Reminder: normal equations in the linear case

$$X^T y + X^T X \Delta\omega = 0$$

Non-linear LS: Gauss-Newton method

$$\min_{\omega} \frac{1}{2} \sum_{i=1}^{n} \left(y_i - f(x_i; \omega) \right)^2$$

Iterative algorithm:

1. Solve

$$J(\omega_t)^T r(\omega_t) + J(\omega_t)^T J(\omega_t) \cdot \Delta\omega = 0$$

2. Update the current position:

$$\omega_{t+1} = \omega_t + \Delta\omega$$

Problem: after two approximations some steps are too large and may lead to the increase/very slow decrease/oscillations.

Levenberg-Marquardt method

$$\min_{\omega} \frac{1}{2} \sum_{i=1}^{n} \left(y_i - f(x_i; \omega) \right)^2$$

On each step of Gauss-Newton we optimize:

$$F(\omega_t + \Delta\omega) = F(\omega_t) + \nabla F(\omega_t) \cdot \Delta\omega + \frac{1}{2} \Delta\omega^T J(\omega_t)^T J(\omega_t) \Delta\omega$$

The approximation makes sense in the vicinity.
Prevent the steps that are too large by adding an extra term (introducing the **"trust region"**):

$$\min_{\Delta\omega} F(\omega_t) + \nabla F(\omega_t) \cdot \Delta\omega + \frac{1}{2} \Delta\omega^T J(\omega_t)^T J(\omega_t) \Delta\omega + \frac{1}{2} \lambda \|\Delta\omega\|^2$$

Levenberg-Marquardt method

$$\min_{\Delta\omega} F(\omega_t) + \nabla F(\omega_t)\cdot\Delta\omega + \frac{1}{2}\Delta\omega^T J(\omega_t)^T J(\omega_t)\Delta\omega + \frac{1}{2}\lambda\|\Delta\omega\|^2$$

Equate the gradient to zero:

$$J(\omega_t)^T r(\omega_t) + J(\omega_t)^T J(\omega_t)\cdot\Delta\omega + \lambda I\Delta\omega = 0$$

We arrive at the ***Levenberg-Marquardt*** algorithm:

1. Solve

$$J(\omega_t)^T r(\omega_t) + J(\omega_t)^T J(\omega_t)\cdot\Delta\omega + \lambda I\Delta\omega = 0$$

2. Update the current position: $\omega_{t+1} = \omega_t + \Delta\omega$

3. If the value is decreased "enough", decrease λ.
Otherwise, step back and increase it.

Interpretation 1

$$\nabla^2 F(\omega) = \frac{d J(\omega)^T}{d\omega} \cdot r(\omega) + J(\omega)^T J(\omega) + \lambda I$$

- Easily computable
- Easily invertible (when Jacobian is sparse)

$$\nabla^2 F(\omega) \cdot \Delta\omega = -\nabla F(\omega)$$

- **Positive definite, well conditioned**

Interpretation 2

$$J(\omega_t)^T r(\omega_t) + J(\omega_t)^T J(\omega_t) \cdot \Delta\omega + \lambda I \Delta\omega = 0$$

$$\Delta\omega = -\left(J(\omega_t)^T J(\omega_t) + \lambda_t I\right)^{-1} J^T r(\omega_t)$$

How does the size of the extra term affects the algorithm?

λ^t is small $\quad \Delta\omega \approx -\left(J(\omega_t)^T J(\omega_t)\right)^{-1} J(\omega_t)^T r(\omega_t)$

This is Gauss-Newton!

λ^t is large $\quad \Delta\omega \approx -\frac{1}{\lambda} J(\omega_t)^T r(\omega_t) = -\frac{1}{\lambda} \nabla F(\omega_t)$

This is gradient descent!

Depending on the size of the extra term, Levenberg-Marquardt switches between gradient descent like behaviour and Newton method-like behaviour.