Overfitting, Regularization, Cross-validation

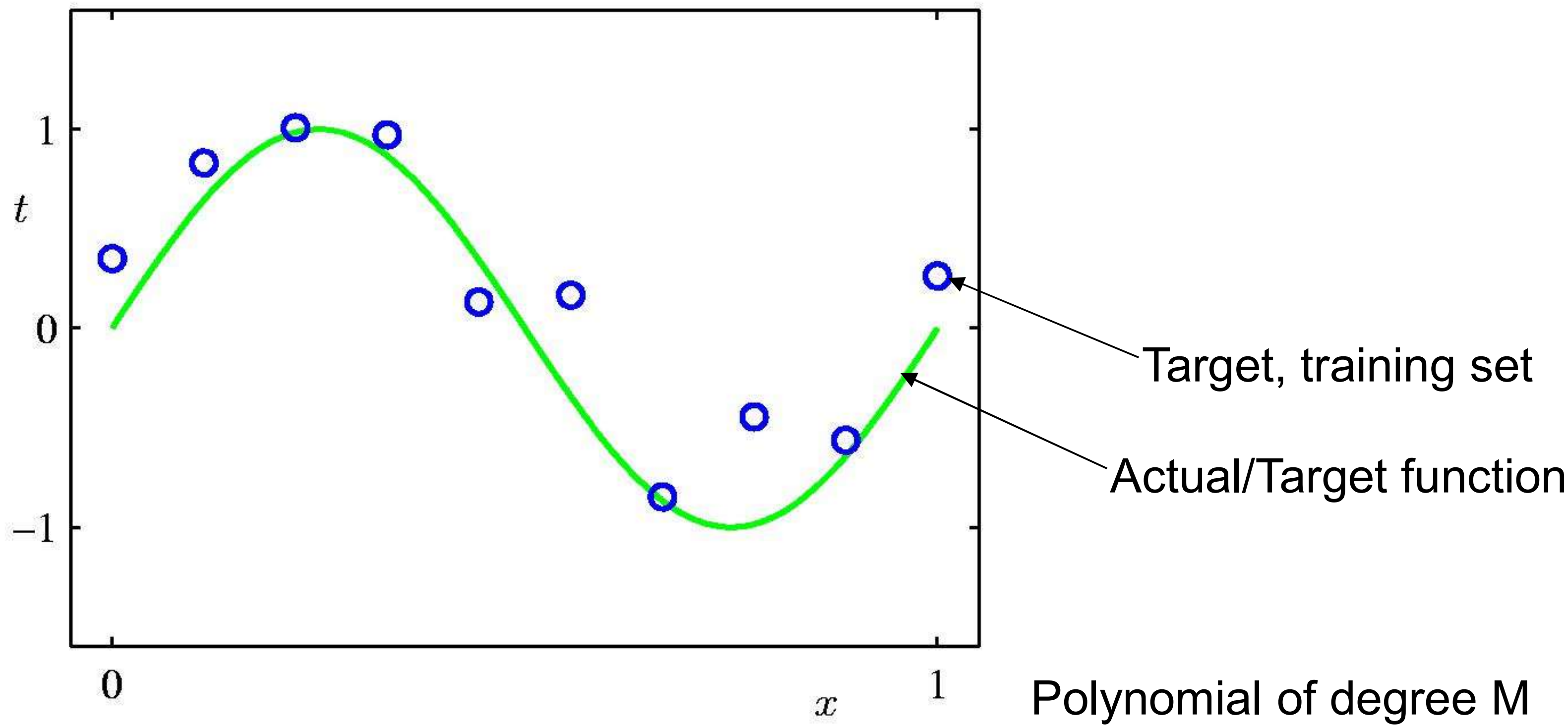
Dr. Wojtek Kowalczyk

wojtek@liacs.nl

Overfitting, Regularization, Cross-validation

1. Overfitting: model learns “small details” of the training set and is unable to correctly classify cases of the test set (usually: too many parameters/degrees of freedom)
2. Regularization: preventing overfitting by imposing some constraints on values or the number of model parameters
3. Cross-validation: monitoring the error both on the training and the test set

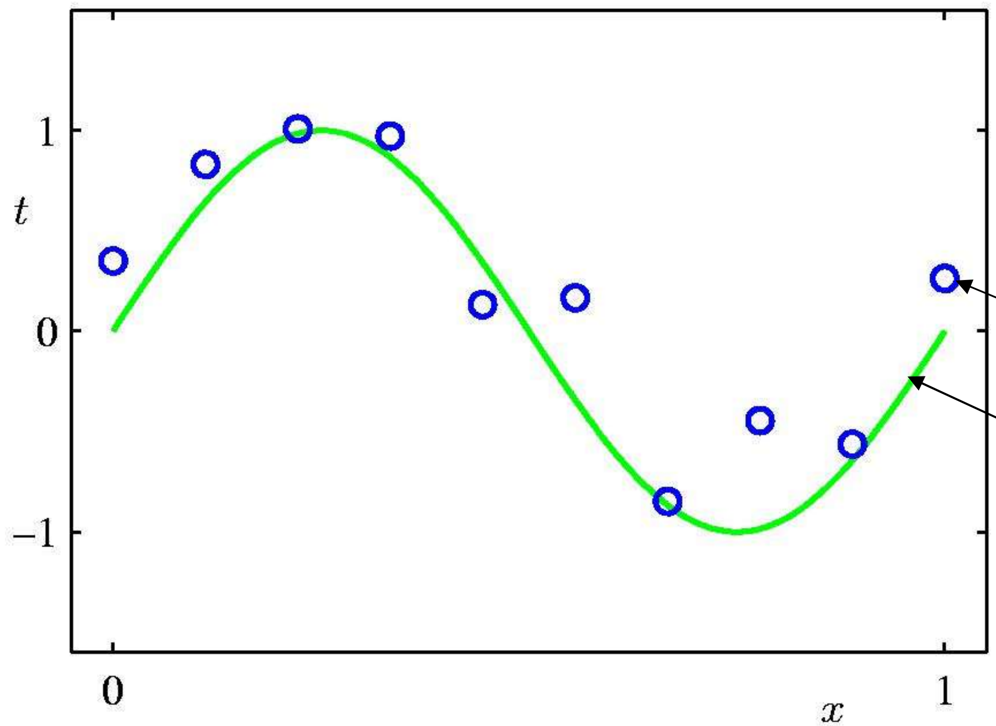
Regression: Polynomial Curve Fitting



Polynomial of degree M

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

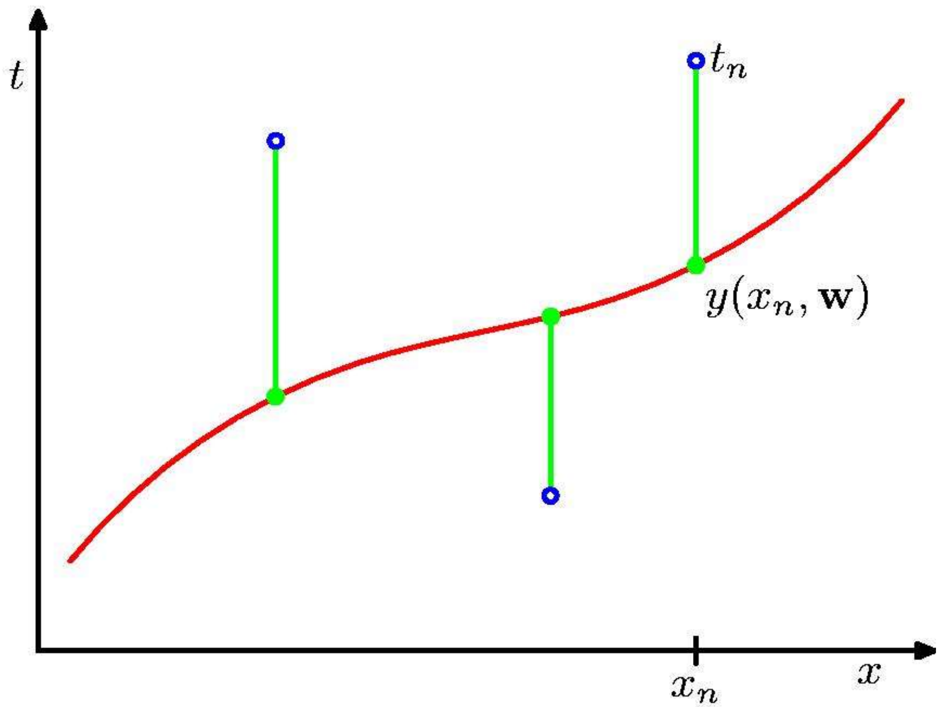
Regression: Polynomial Curve Fitting



Polynomial of degree M

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

Sum-of-Squares Error Function



$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\}^2$$

Through optimization,
we need to find the model (red) y(x,w) that minimizes the error function.

E_{SSE}(w): Sum squared error: 2E(w)

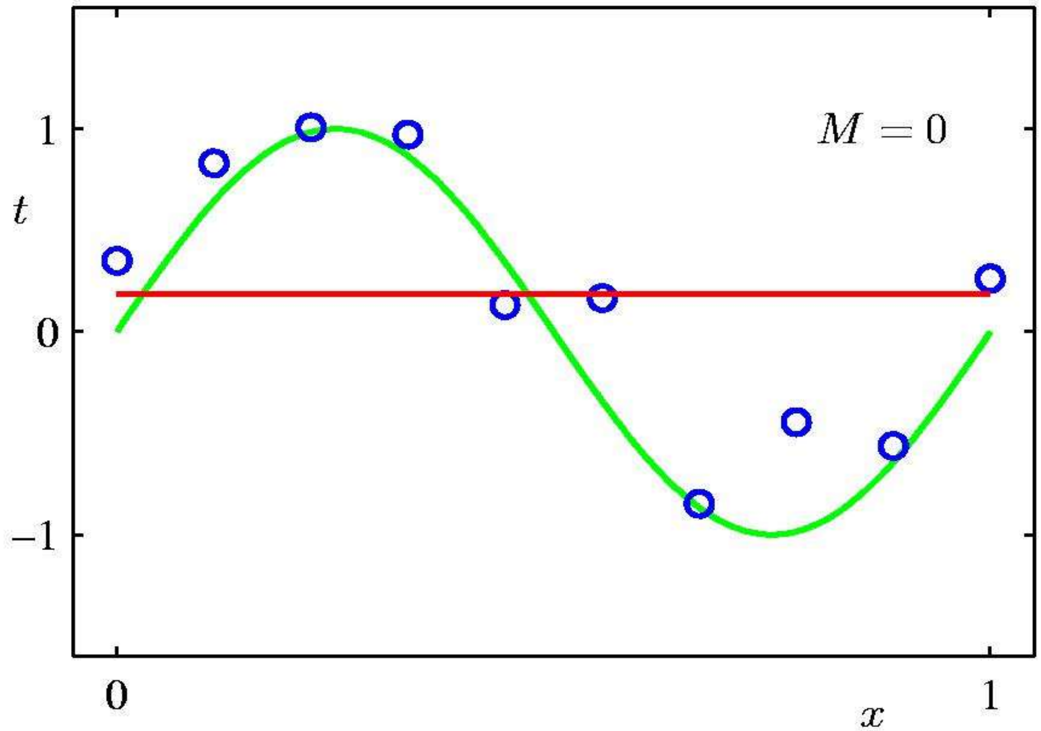
E_{RMS}(w): Root mean squared error

$$E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^\star)/N}$$

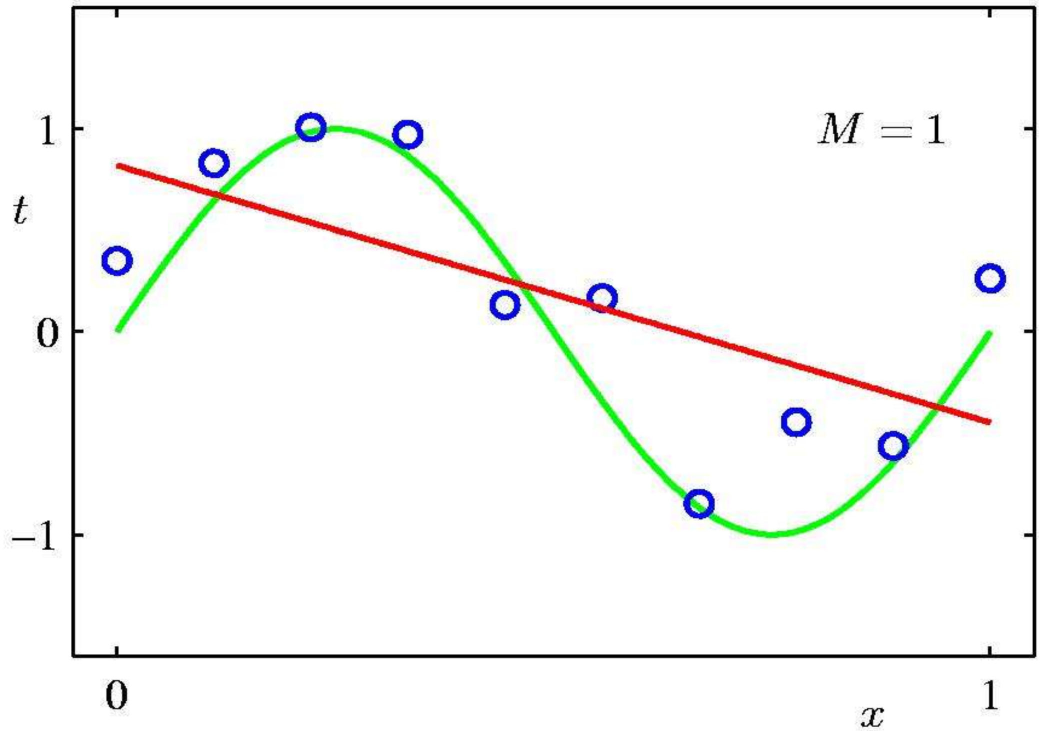
Finding optimal coefficients:
the ***polyfit.m*** function.

|x-y| instead of (x-y)2?
Then error wouldn't be "smooth"…

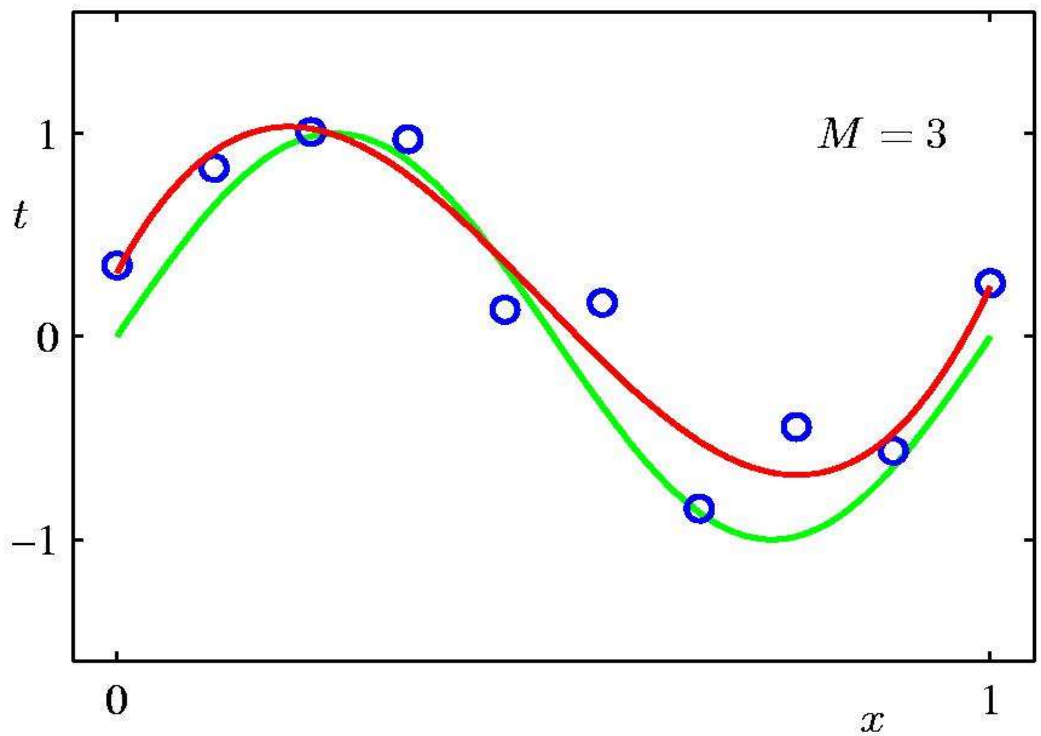
0^{th} Order Polynomial



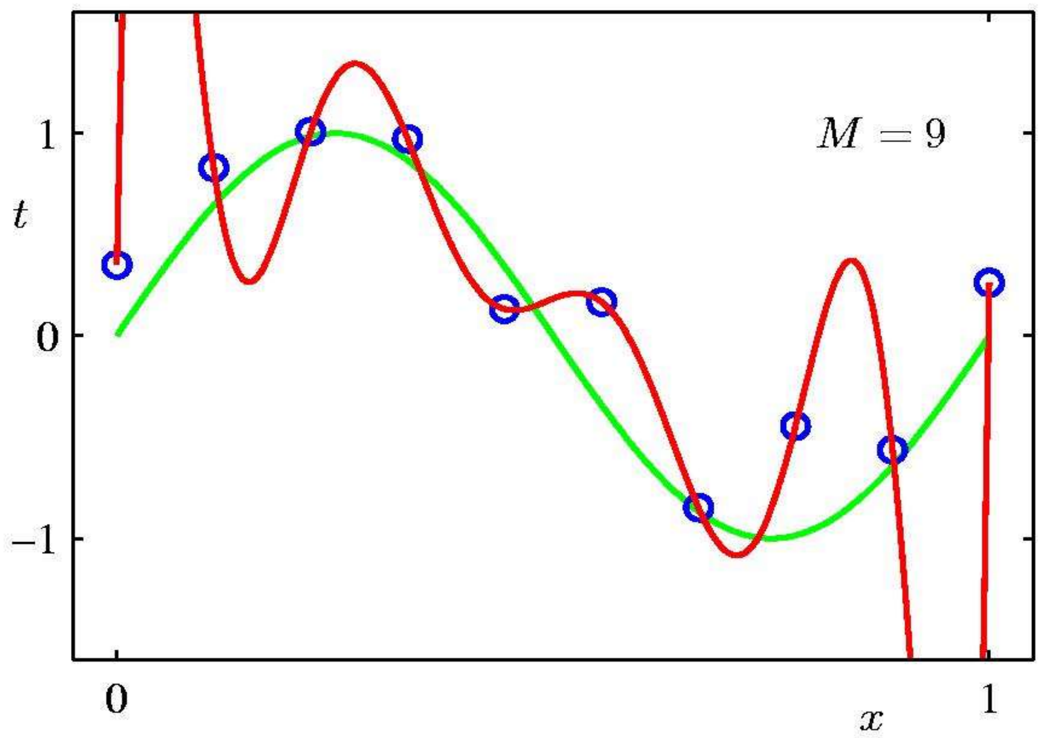
1^{st} Order Polynomial



3rd Order Polynomial



9th Order Polynomial (over-fitting)



Polynomial Coefficients

	$M=0$	$M=1$	$M=3$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

REGULARIZATION

Regularization is a powerful technique of limiting overfitting.

The key idea: somehow enforce the absolute values of model parameters to be relatively small (in our case: coefficients of the polynomial).

For example: add to the error function an extra term: "the sum of squared coefficients of your model":

$$\lambda \sum_{i=0}^{M} w_i^2$$

where λ a tunable parameter that controls the size "punishment" for too big values of coefficients.

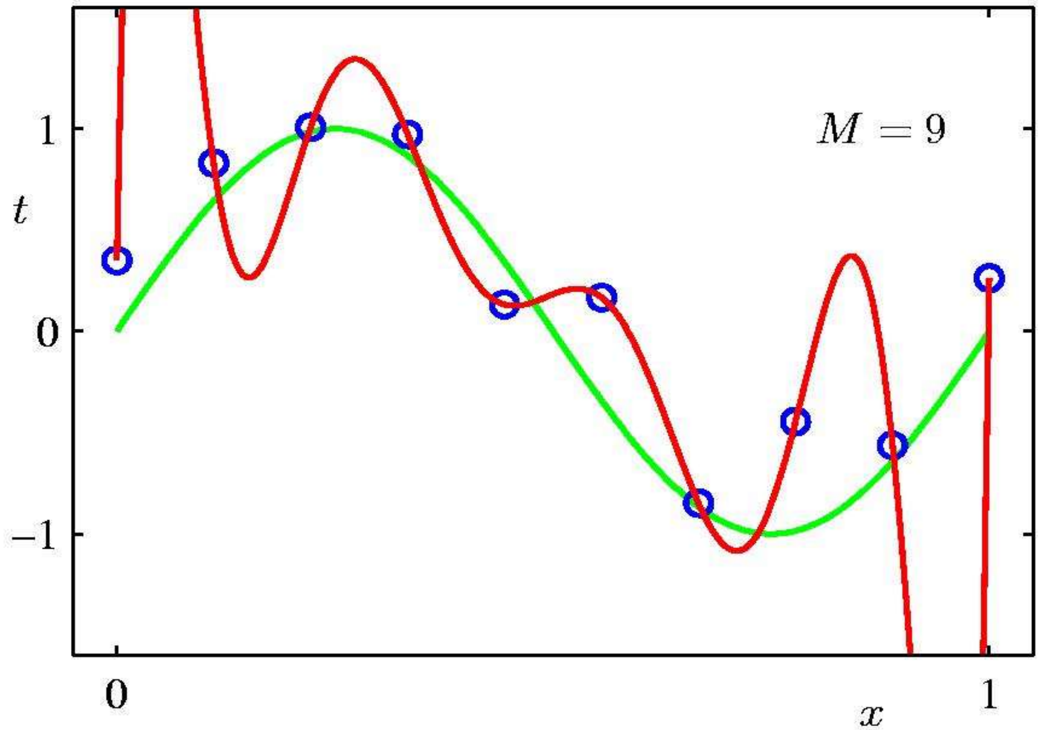
Regularization

- **Penalize large coefficient values**

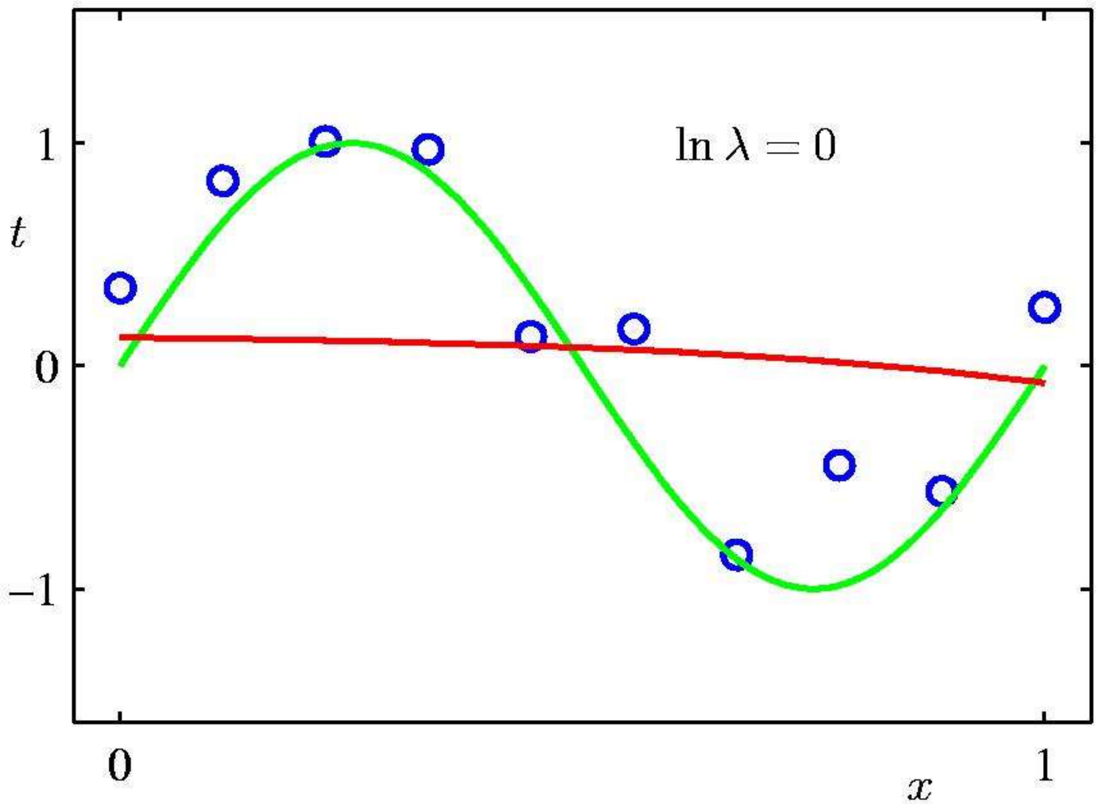
$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

- **Minimize training error while keeping the weights small. This is known as:**
  - shrinkage,
  - ridge regression,
  - weight decay (neural networks)

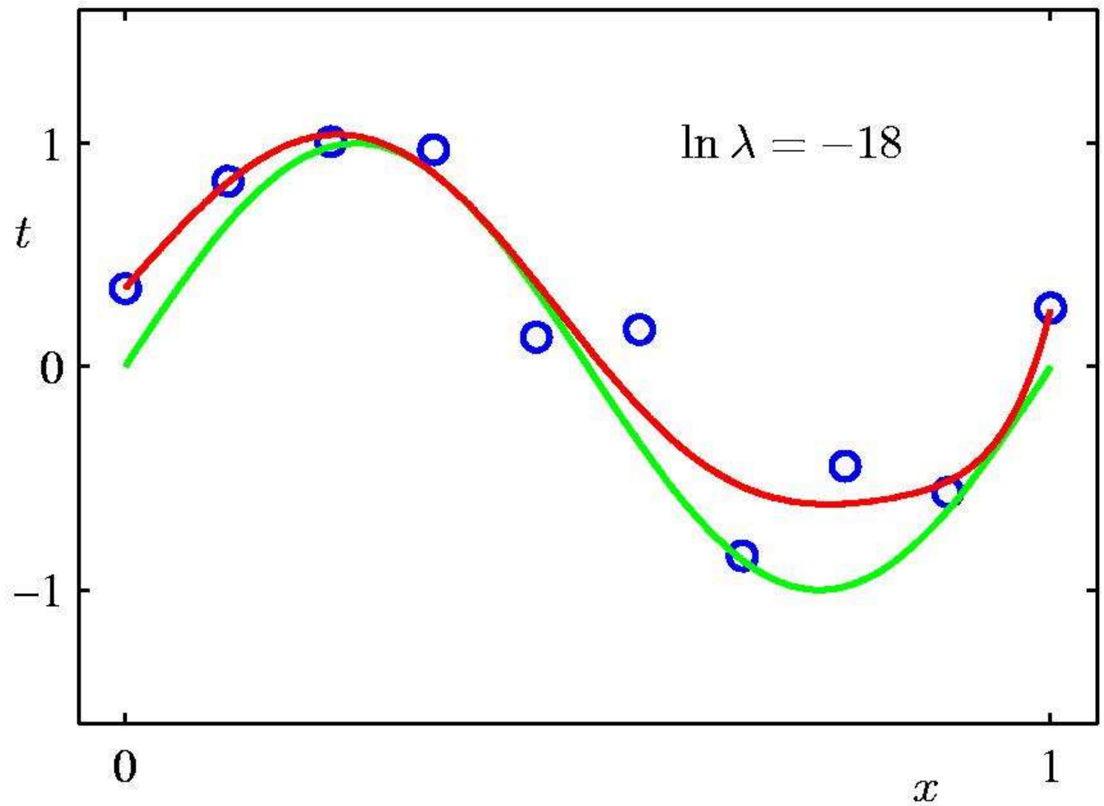
Regularization (M=9; λ=0)



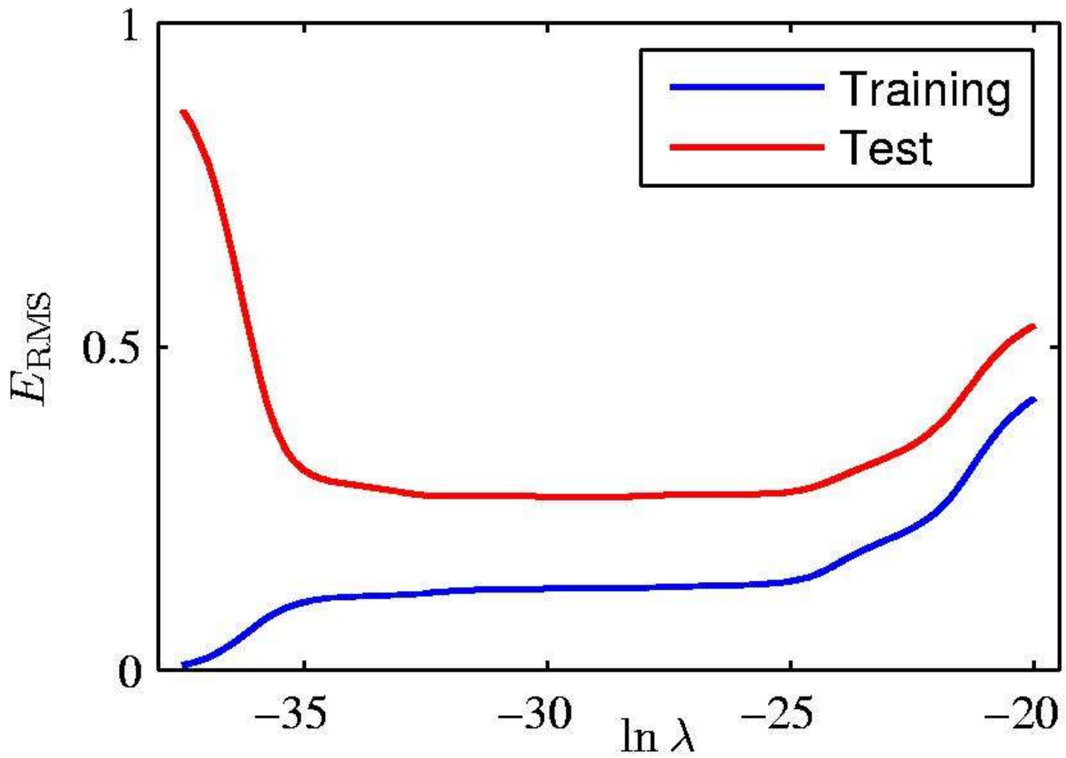
Regularization (d=9; λ=1)



Regularization (M=9; λ=1.5230e-08)



Regularization: error vs. lambda



Polynomial Coefficients

	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

TASK 1

Try to reconstruct (with help of Python) all the plots shown in the previous slides. In concreto, start with the function

$$y(x)=0.5+0.4*\sin(2*\pi*x), \text{ for } x \text{ in } [0, 1].$$

Generate two noisy sets of n points (train and test) that will be used for approximating y, for n=9, 15, 100. The x-coordinates should be uniformly distributed in [0,1], y-coordinates should be contaminated with Gaussian noise with mean=0, std=0.05:

Task 2

Consider a unit cube in n-dimensional space $U_n=[0,1]^n$,
and an n-dimensional unit ball B_n that is included in U_n:

$$B_n=\{(x_1, x_2, ..., x_n) \mid (x_1-0.5)^2+(x_2-0.5)^2+...+(x_n-0.5)^2 < 0.5^2\}$$

for n=1, 2, ..., 100.

a) find the number of vertices ("corners") of U_n, **Corners(n)**

b) calculate the length of the longest diagonal of U_n, **Diag(n)**

c) estimate (or calculate) the volume of B_n, **VolumeB(n)**

Hint: generate 10^6 points, uniformly distributed in U_n (points=rand(10^6,n);) and check how many of them are in B_n

d) calculate the volume of the "0.01-skin" of U_n: **VolumeS(n)** = 1^n-(1-2*0.01)^n

e) For **n= 2, 4, 8, …., 1024** generate 1000 points in $\mathbf{U_n}$, uniformly distributed, find distances between all pairs of these points, and produce a histogram these distances.

Produce a single script that generates all these figures.