TDT4173 Machine Learning

Regression techniques

Norwegian University of Science and Technology

Helge Langseth
IT-VEST 310
helgel@idi.ntnu.no

NTNU

Outline

Announcements

- **Assignment 2 is out.**
  This is the "BIG" one, so do spend some time on it!
- **The Lecture next week (September 21*st*) is cancelled.**
  Get up early and work on the big assignment instead!
- We have to have a **reference group** after all.
  I need 3 (well, at least 2) volunteers.

Summary-points from last lesson

- **Ensemble methods:**
  - Bagging
  - Boosting
- **SVMs:**
  - Linear separators
  - The dual problem – solution using constrained optimization
  - Non-separable subspaces
  - Nonlinearity and kernels

Linear regression

- As usual, we have examples $(\boldsymbol{x}_i, y_i)$.
- We require $\boldsymbol{x}_i \in \mathbb{R}^d$, so the instance space is the (appropriately sized) real space.
- Furthermore: **the target is numeric, too**: $y_i \in \mathbb{R}$.

Examples:

- Voltage $\rightarrow$ Temperature
- Outside temp. + Speed $\rightarrow$ Power consumption of electrical car
- Robot arm controls $\rightarrow$ Torque at effectors

Linear regression - setup

Assume a **linear model** from $\boldsymbol{X}$ to Y, i.e.

$$\begin{aligned} Y &\leftarrow \beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d + \text{``Noise``} \\ Y &\leftarrow \boldsymbol{\beta}^\intercal \boldsymbol{x} + \epsilon,\ \epsilon \sim \mathcal{N}(0, \sigma^2) \end{aligned}$$

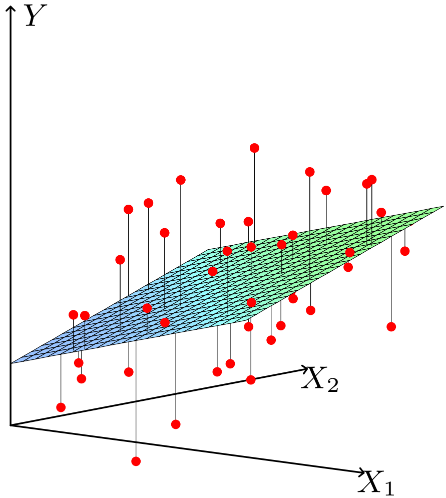
Notice! Use a trick of appending a '1' to $x_1, \ldots, x_d$ for the vector notion to work, i.e., $\boldsymbol{x} = [1, x_1, x_2, \ldots, x_d]^\intercal$.)

We call this linear...

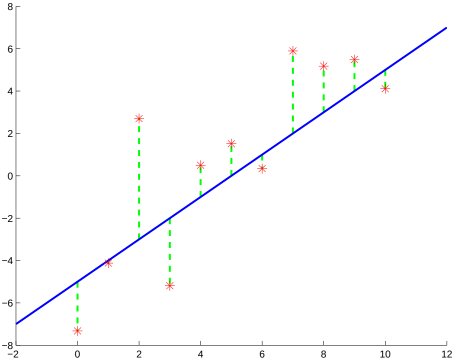
- As long as it is linear in the (unknown) weights $\boldsymbol{\beta}$.
- Even if it is nonlinear in the (known) x-values.
- So, the model $Y = \beta_0 + \beta_1 x + \beta_2 x^2 + \epsilon$ is a **linear** regression model – because it is linear in each β_i.

What we try to do. . .

Find the plane which most closely approximates the training data:



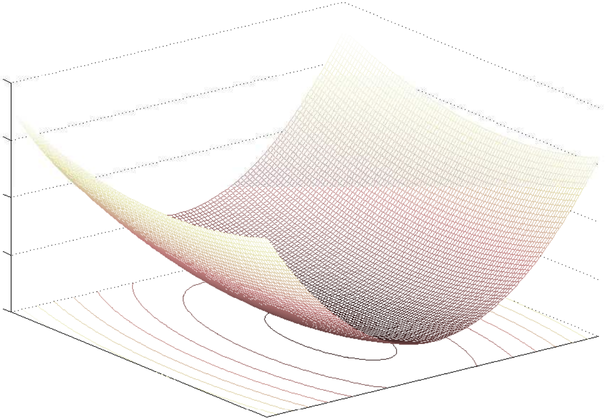
How to choose a “close“ approximation



We define:

$$\mathbb{E}[\vec{\beta}] = \frac{1}{2}\sum_{r=1}^{N}(y_r - \hat{y_r})^2 = \frac{1}{2}\sum_{r=1}^{N}(y_r - \boldsymbol{\beta}^{\mathsf{T}}\boldsymbol{x}_r)^2$$

Gradient Descent – Motivation



Question: How can we minimise this error function without knowing the whole shape?
Obvious answer: Walk "downhills"!

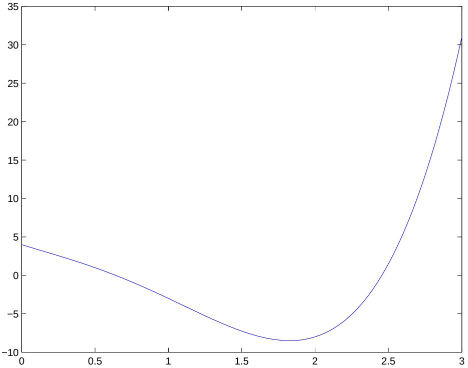
Gradient Descent – The setup

- We want to find the value x which minimises $f(x)$.
- To avoid evaluating the whole function we use an iterative approach:
  - Guess a value for x, and calculate the derivative $f'(x)$.
  - Make a new guess for x based on these calculations — Move "downhill".
  - Keep iterating...
- **Intuition:**
  - If the derivative is small (in absolute value) we are close from the minimising point.
  - ...and if it is zero we are done.
  - On the other hand, we are far away if the derivative is large (in absolute value).

Solution:
Use the update rule $x_{i+1} \leftarrow x_i - \eta \cdot f'(x_i)$. $\eta > 0$ is the learning rate.

Gradient Descent – Example

Minimize $f(x) = 2x^4 - 5x^3 + 2x^2 - 6x + 4$ with $\eta = 0.025$.

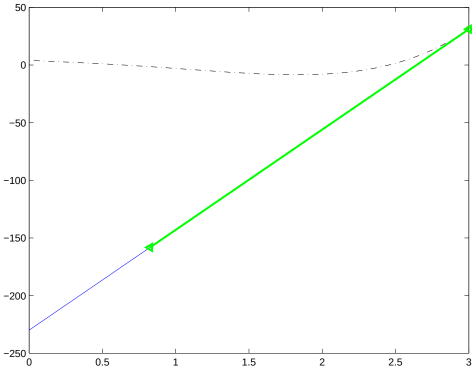


The $f(x)$ has a minimum at $x = 1.8261$. Let's try to find it...

DEMO: `gradient.m`

Gradient Descent – Example

Minimize $f(x) = 2x^4 - 5x^3 + 2x^2 - 6x + 4$ with $\eta = 0.025$.

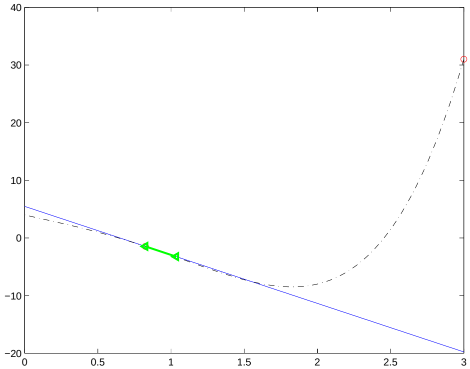


Starting from $x_0 = 3$ and finding $f'(3) = 87$:

$$\begin{aligned} x_1 &= x_0 - \eta f'(x_0) \\ &= 3 - 0.025 \cdot 87 = 0.8250 \end{aligned}$$

Gradient Descent – Example

Minimize $f(x) = 2x^4 - 5x^3 + 2x^2 - 6x + 4$ with $\eta = 0.025$.

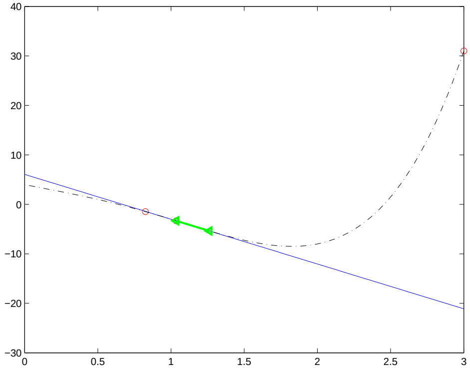


Going from $x_1 = 0.8250$ with $f'(0.8250) = -8.4172$:

$$\begin{aligned} x_2 &= x_1 - \eta f'(x_1) \\ &= 0.825 - 0.025 \cdot (-8.4172) = 1.0354 \end{aligned}$$

Gradient Descent – Example

Minimize $f(x) = 2x^4 - 5x^3 + 2x^2 - 6x + 4$ with $\eta = 0.025$.

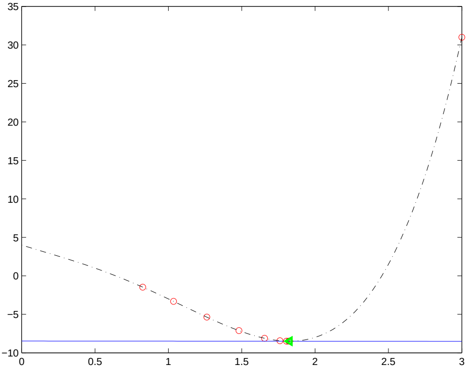


Going from $x_2 = 1.0354$ with $f'(1.0354) = -9.0592$:

$$x_3 = 1.0354 - 0.025 \cdot (-9.0592) = 1.2619$$

Gradient Descent – Example

Minimize $f(x) = 2x^4 - 5x^3 + 2x^2 - 6x + 4$ with $\eta = 0.025$.

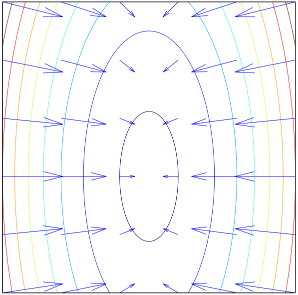


…and finally going from $x_{10} = 1.8260$ with $f'(1.8260) = -0.0034$:

$$x_{11} = 1.8260 - 0.025 \cdot (-0.0034) = 1.8261$$

…*and we are done.*

Gradient Descent – in higher dimensions



Recall that

- The **gradient** of a surface $\mathbb{E}[\vec{\beta}]$ is a vector in the direction the curve grows the most (calculated at $\vec{\beta}$).
- The gradient is calculated as $\nabla\mathbb{E}[\vec{\beta}] \equiv \left[\frac{\partial\mathbb{E}}{\partial\beta_0}, \frac{\partial\mathbb{E}}{\partial\beta_1}, \cdots \frac{\partial\mathbb{E}}{\partial\beta_d}\right]$.

Training rule: $\Delta\vec{\beta} = -\eta\cdot\nabla\mathbb{E}[\vec{\beta}]$, i.e., $\Delta\beta_i = -\eta\cdot\frac{\partial\mathbb{E}}{\partial\beta_i}$.

Gradient Descent and linear regression

We need the $\frac{\partial \mathbb{E}}{\partial \beta_i}$ for $i = 0, \ldots, d$:

$$\begin{aligned}\frac{\partial \mathbb{E}}{\partial \beta_i} &= \frac{\partial}{\partial \beta_i}\left\{\frac{1}{2}\sum_{r=1}^{N}\left(y_r - \sum_{j=0}^{d}\beta_j\, x_{r,j}\right)^2\right\} \\ &= -\sum_{r=1}^{N}\left(y_r - \sum_{j=0}^{d}\beta_j\, x_{r,j}\right)\cdot x_{r,i}\end{aligned}$$

(Each $\boldsymbol{x}_r$ is redefine to have a new element 1 as its zeroth element.)

Training rule: $\beta_i \leftarrow \beta_i + \eta \times \sum_{r=1}^{N}\left(y_r - \boldsymbol{\beta}^\mathsf{T}\boldsymbol{x}_r\right)\cdot x_{r,i}$

DEMO: `regexample.m`

The normal equations

Using vector and matrix derivatives, we can also optimise $\boldsymbol{\beta}$ directly:

$$\boldsymbol{\beta} \leftarrow \left(\boldsymbol{X}^\intercal \boldsymbol{X}\right)^{-1} \boldsymbol{X}^\intercal \boldsymbol{y}$$

Here

- Let $\boldsymbol{\beta}$ be the vector of parameters: $\boldsymbol{\beta} = [\beta_0, \ldots, \beta_d]^\intercal$.
- Let $\boldsymbol{y}$ be the observed target-values: $\boldsymbol{y} = [y_1, \ldots, y_N]^\intercal$.
- Let $\boldsymbol{X}$ be the matrix of observed explanatory variables, i.e. a matrix of size $N \times (d+1)$, where element (r, i) is explanatory variable $i = 0, \ldots, d$ in observation r (and again: $x_{r,0} = 1$ is appended to each $\boldsymbol{x}$ for this to work).

Regression with polynomials

- Say I believe in a model $Y = \beta_0 + \beta_1 \cdot x + \beta_2 \cdot x^2$.
- I have the data:

x	y
0	0.376
1	0.934
2	4.265
3	8.837

- How can I proceed?

Regression with polynomials

- Say I believe in a model $Y = \beta_0 + \beta_1 \cdot x + \beta_2 \cdot x^2$.
- I **create** the data:

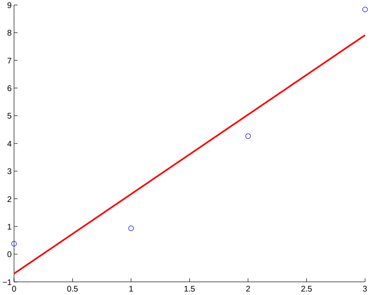
x			y
1	0	0	0.376
1	1	1	0.934
1	2	4	4.265
1	3	9	8.837

- **. . . and run along as before.**

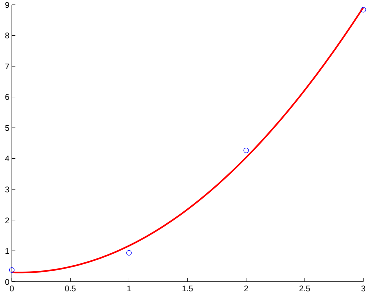
Regression with polynomials

Different models – Different results

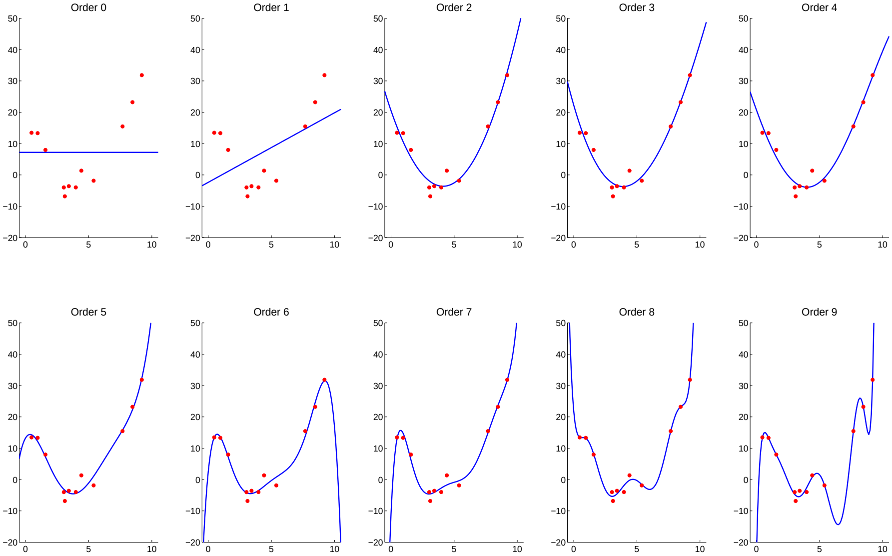
$$Y = \beta_0 + \beta_1 \cdot x$$



$$Y = \beta_0 + \beta_1 \cdot x + \beta_2 \cdot x^2$$



Regression with polynomials: Overfitting



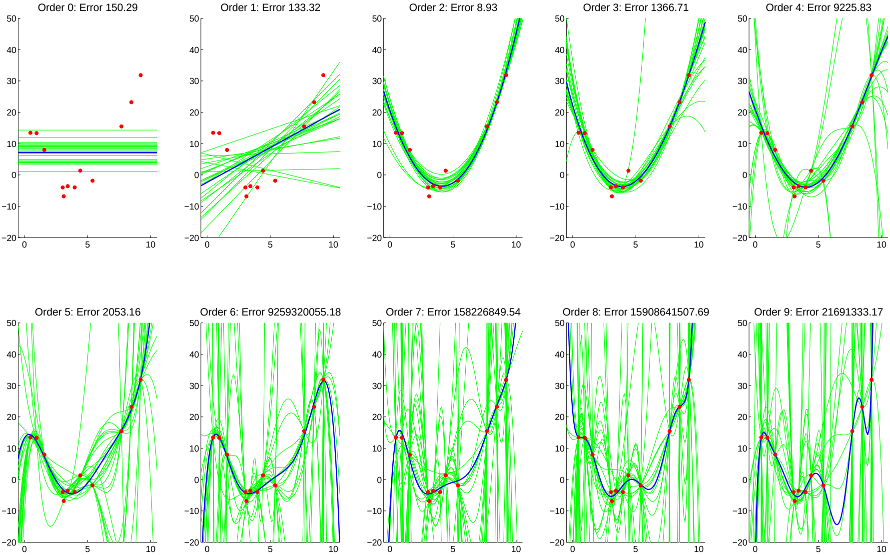
Regression with polynomials: Overfitting

One idea to assess if we overfit:

1. Create N datasets using bootstrapping (ref. last lecture)
2. For each bootstrapped dataset:
   - Generate each candidate regression model (order 0, 1, 2, $\ldots$).
   - Calculate the generalisation error on the original dataset.
3. Choose the model with the best averaged generalisation error.

DEMO: `polyboot.m`

Regression with polynomials: Overfitting



Regression with polynomials: Overfitting

Better/quicker method:

- Define an error function, which penalises complexity:

$$\begin{aligned}\mathbb{E}(\boldsymbol{\beta}) &= \text{Prediction error} + \lambda \times \text{Complexity penalty} \\ &= \frac{1}{2}\sum_{r=1}^{N}\left(y_r - \sum_{j=0}^{d}\beta_j\, x_{r,j}\right)^2 + \lambda \times f(\boldsymbol{\beta})\end{aligned}$$

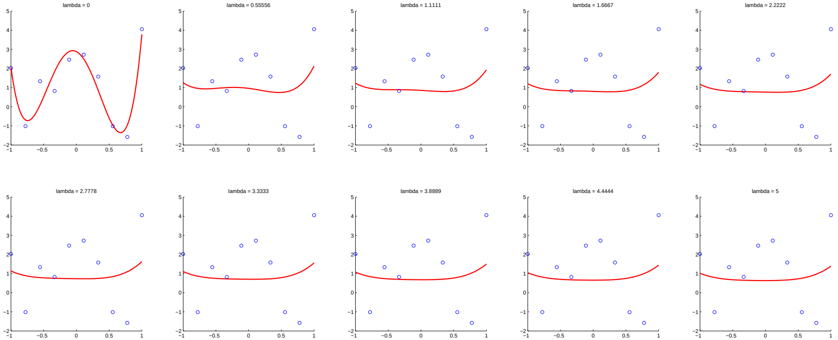
- Choose $\boldsymbol{\beta}$ to minimise the error, either using closed-form equations or gradient descent.

Ridge regression

Ridge regression uses

$$\mathbb{E}(\boldsymbol{\beta}) = \frac{1}{2}\sum_{r=1}^{N}\left(y_r - \sum_{j=0}^{d}\beta_j\, x_{r,j}\right)^2 + \lambda \times \sum_{j=0}^{d}\beta_j^2$$

Chosen to react to big β-values, hence reduces overfitting.



Lasso regression

Lasso regression uses

$$\mathbb{E}(\boldsymbol{\beta}) = \frac{1}{2}\sum_{r=1}^{N}\left(y_r - \sum_{j=0}^{d}\beta_j\, x_{r,j}\right)^2 + \lambda \times \sum_{j=0}^{d}|\beta_j|$$

Chosen to force as many β-values as possible to be zero.

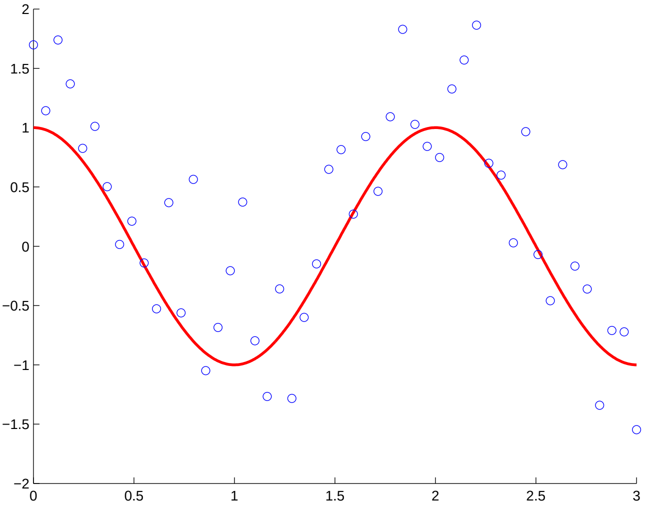
Where Ridge has a closed form solution

$$\boldsymbol{\beta} \leftarrow \left(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{X}^\top \boldsymbol{y},$$

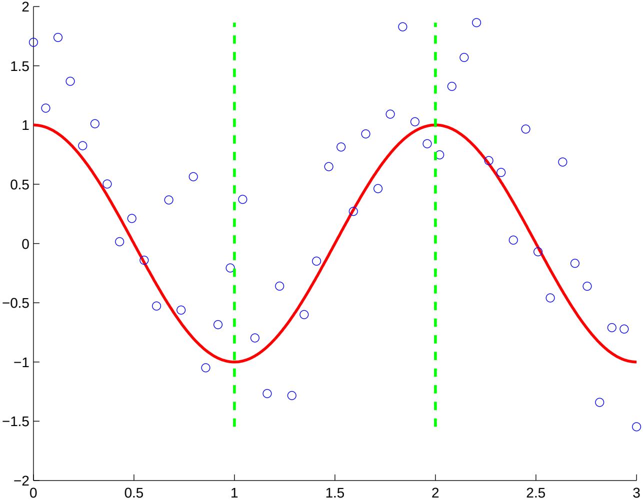
the Lasso is solved by gradient descent.

DEMO: `L1reg.m`

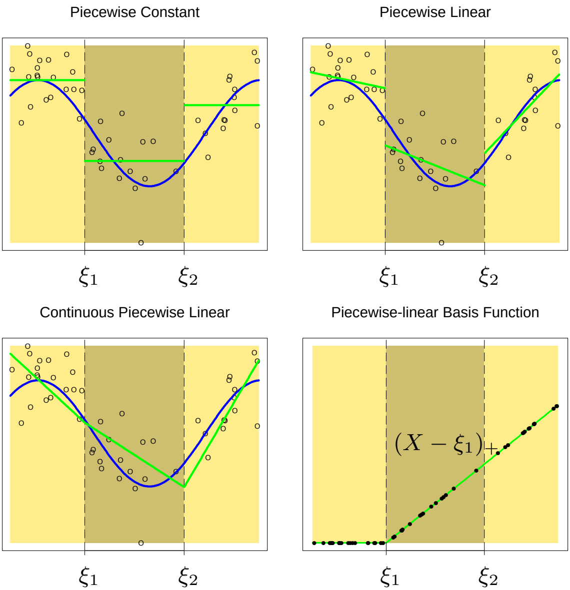
The problem: Some models are just not "stationary"



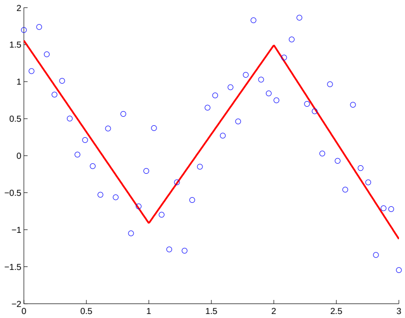
The problem: Some models are just not "stationary"

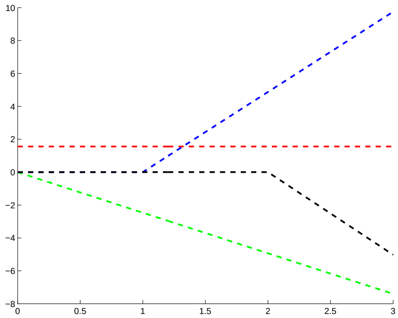


Strategies for non-uniformity



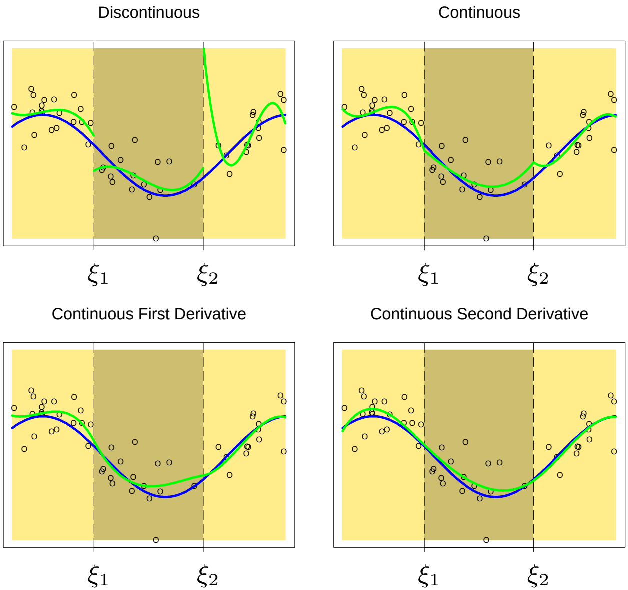
Piecing together the cosine function





- Approximation guaranteed to be continuous
- Not smooth
- Can we do better?

Higher order functions



Cubic spline setup

Setup:

- Define "split points" (ξ_j-values) – typically a split is defined at every datapoint
- "Basis functions" $(x - \xi_j)^3_+$ defined for each split point.
- The global model $Y \leftarrow \beta_0 + \sum_j \beta_j \, (x - \xi_j)^3_+ + \text{Noise}$ can be fitted "as usual".

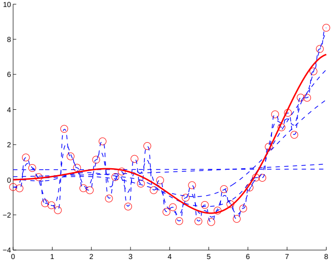
Goodies:

- The model i smooth – even has continuous second derivatives.
- Robust methods (B-splines) are available – scales fairly well.
- Good statistical properties

DEMO: `splines.m`

Kernel-regression: A "non-parametric" solution

- Instead of defining a model for Y, kernel regression just assumes that $Y(x)$ is smooth in x.
- Thus, $Y(x_{\text{New}})$ is a weighted average of observed y_i values, with weights depending on difference between x_i and x_{New}.
- Often used weight function
  $k(|x_{\text{New}} - x_i|, \sigma) = \exp\left(\frac{-(x_{\text{New}}-x_i)^2}{2\sigma^2}\right)$; σ is the "bandwidth".



DEMO: `kernel.m`

Setting the "extra" parameters

- We have seen some new parameters:
  - kernel bandwidth (σ)
  - tradeoff between error and complexity (λ)
  - number of polynomials (d)
  - spline-knots ($\boldsymbol{\xi}$)

How can we find good values for these?

Setting the "extra" parameters

- We have seen some new parameters:
  - kernel bandwidth (σ)
  - tradeoff between error and complexity (λ)
  - number of polynomials (d)
  - spline-knots ($\boldsymbol{\xi}$)

Use cross-validation:

For each potential model (parameter setting):

- Define cross validation sets
- Calculate cross-validation error
- Choose the best one!

Summary

- Linear regression
- Gradient descent
- Overfitting
- Splines
- Non-parametric smoothing

Remember: Next week is off!

I have other arrangements I must attend to next week, so

the lecture on Sept 21th is cancelled.

Spend your new-found extra time on the "big" assignment.