

# 732G12 Data Mining

## Föreläsning 8

---

Johan Alenlöv

IDA, Linköping University, Sweden

- Icke-linjär regression
- Splines

Vi backar tillbaka till början av kursen och vanlig linjär regression i en variabel.

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0, \sigma^2).$$

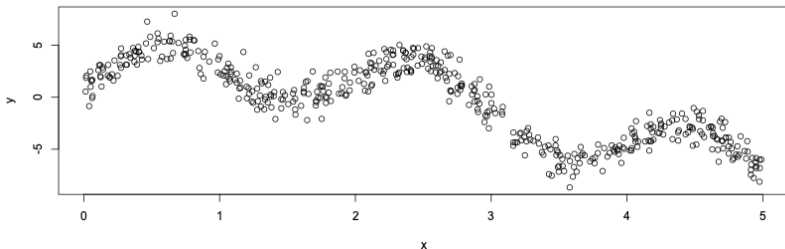
Vad gör vi om en linjär funktion inte passar till datan?

# Icke-linjär regression

Vi backar tillbaka till början av kursen och vanlig linjär regression i en variabel.

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0, \sigma^2).$$

Vad gör vi om en linjär funktion inte passar till datan?

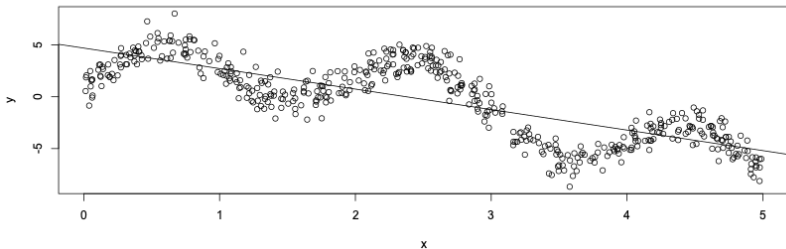


# Icke-linjär regression

Vi backar tillbaka till början av kursen och vanlig linjär regression i en variabel.

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0, \sigma^2).$$

Vad gör vi om en linjär funktion inte passar till datan?



Ett alternativ är att modellera som ett polynom,

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \dots + \beta_p x_i^p + \varepsilon_i.$$

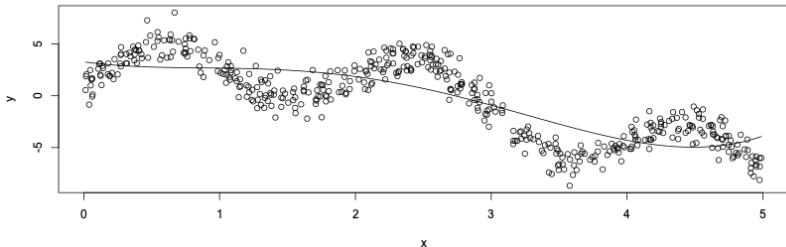
Men vilken ordning ska vi ha?

# Polynom regression

Ett alternativ är att modellera som ett polynom,

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \dots + \beta_p x_i^p + \varepsilon_i.$$

Men vilken ordning ska vi ha?

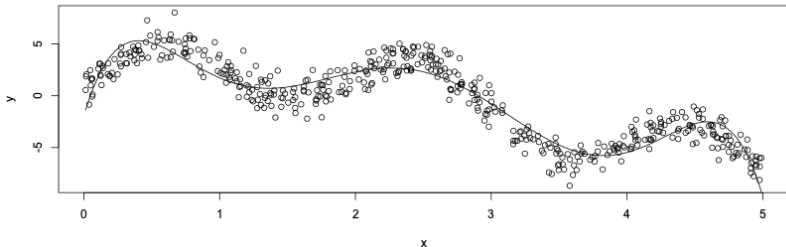


# Polynom regression

Ett alternativ är att modellera som ett polynom,

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \dots + \beta_p x_i^p + \varepsilon_i.$$

Men vilken ordning ska vi ha?



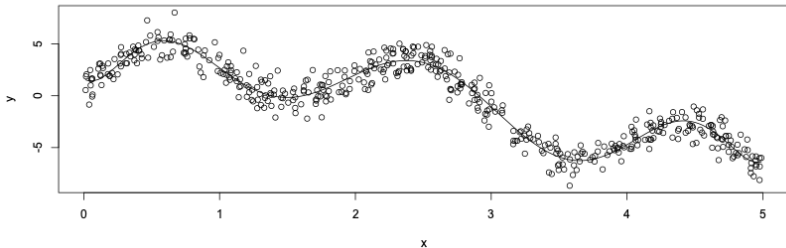


# Polynom regression

Ett alternativ är att modellera som ett polynom,

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \dots + \beta_p x_i^p + \varepsilon_i.$$

Men vilken ordning ska vi ha?



Ett alternativ är att använda en steg-funktion,

$$y_i = \beta_1 \mathbb{I}(x_i \in C_1) + \beta_2 \mathbb{I}(x_i \in C_2) + \dots + \beta_k \mathbb{I}(x_i \in C_k) + \varepsilon_i.$$

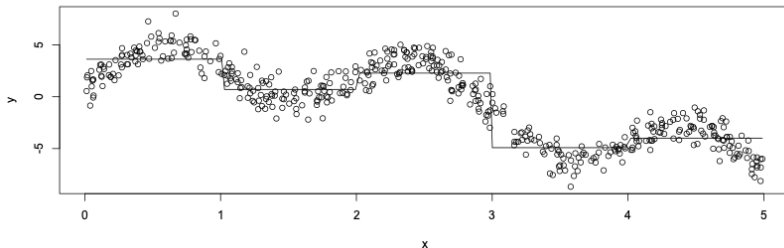
Vilka områden ska vi välja?

# Steg funktion

Ett alternativ är att använda en steg-funktion,

$$y_i = \beta_1 \mathbb{I}(x_i \in C_1) + \beta_2 \mathbb{I}(x_i \in C_2) + \dots + \beta_k \mathbb{I}(x_i \in C_k) + \varepsilon_i.$$

Vilka områden ska vi välja?

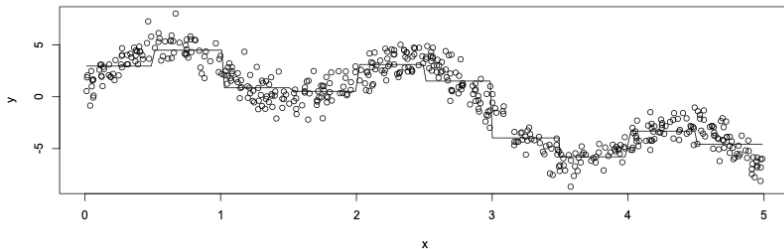


# Steg funktion

Ett alternativ är att använda en steg-funktion,

$$y_i = \beta_1 \mathbb{I}(x_i \in C_1) + \beta_2 \mathbb{I}(x_i \in C_2) + \dots + \beta_k \mathbb{I}(x_i \in C_k) + \varepsilon_i.$$

Vilka områden ska vi välja?



Fördelar:

- Generellt applicerbara.
- Enkelt att skala upp komplexiteten (fler polynom, fler områden).

Nackdelar:

- Svårt i högre dimensioner.
- Svårt att prediktera utanför data.

Gemensamt för dessa metoder är att de bygger på basfunktioner.

Givet ett gäng funktioner  $(b_1(x), b_2(x), \dots, b_k(x))$  kan vi göra regression som

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \dots + \beta_k b_k(x_i) + \varepsilon_i.$$

Observera att vi har ett linjärt problem (i basfunktionerna), vi kan enkelt skatta parametrarna  $\beta_i$  som i vanliga linjär regression.

Basfunktioner kommer i många olika former:

- Polynom:  $b_i(x) = x^i$ .
- Stegfunktioner:  $b_i(x) = \mathbb{I}(x \in C_i)$  där  $C_i$  är något intervall.
- Andra funktioner är också vanliga, t.ex. log, exp, sin, cos, osv.

Ett problem: Behöver ofta väldigt många basfunktioner för att väl anpassa data. Leder ofta till överanpassning.

# Stegvisa basfunktioner

Idé: Istället för att anpassa ett polynom (eller annan funktion) till hela datarymden. Anpassa en funktion per intervall.

Modellen blir då:

$$y_i = \begin{cases} h_0(x_i) + \varepsilon_i, & x_i \in C_0 \\ \vdots & \vdots \\ h_K(x_i) + \varepsilon_i, & x_i \in C_K, \end{cases}$$

där varje funktion  $h_i$  ges av

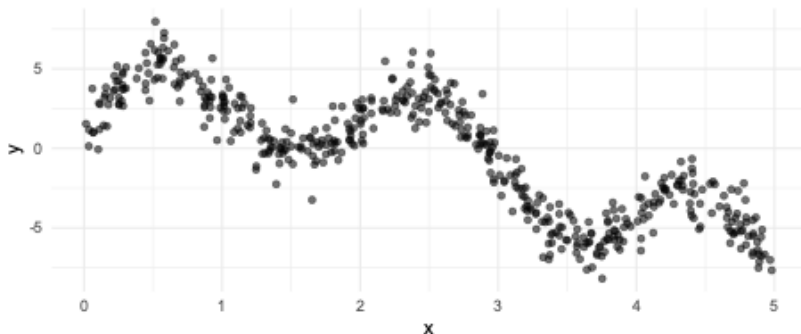
$$h_i(x) = \beta_{0,i} + \beta_{1,i}b_1(x) + \dots + \beta_{k,i}b_k(x).$$

Vi löser helt enkelt en massa "simpel" problem på olika intervall.



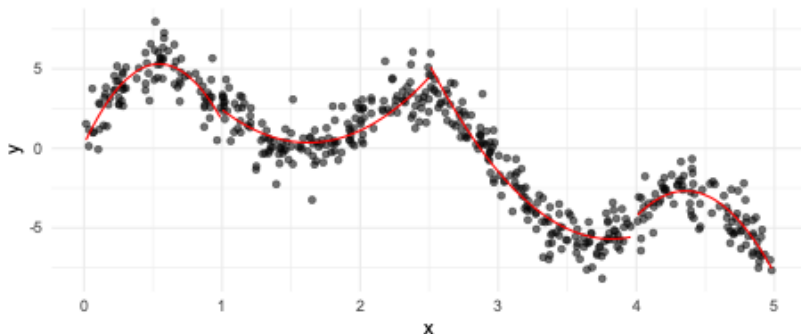
# Stegvisa basfunktioner - Exempel

Vi delar upp data i fyra intervall.  $x < 1$ ,  $1 \leq x < 2.5$ ,  $2.5 \leq x < 4$  och  $4 \leq x$  och skattar en kvadratisk funktion i varje intervall.



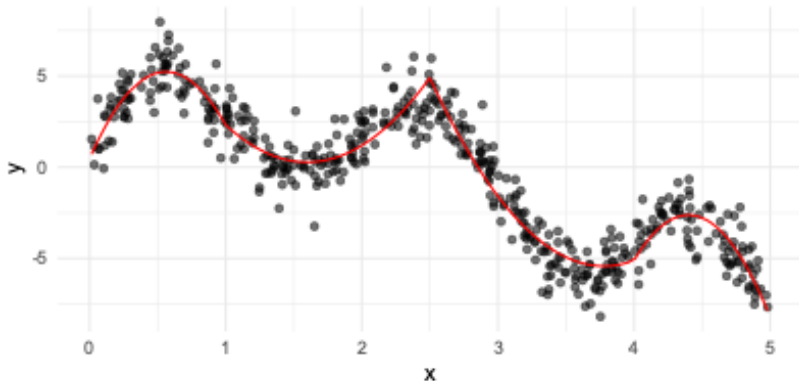
# Stegvisa basfunktioner - Exempel

Vi delar upp data i fyra intervall.  $x < 1$ ,  $1 \leq x < 2.5$ ,  $2.5 \leq x < 4$  och  $4 \leq x$  och skattar en kvadratisk funktion i varje intervall.



## Stegvisa basfunktioner - Exempel

Vi delar upp data i fyra intervall.  $x < 1$ ,  $1 \leq x < 2.5$ ,  $2.5 \leq x < 4$  och  $4 \leq x$  och skattar en kvadratisk funktion i varje intervall.



# Regression Splines

Stegvis polynom regression som i exemplet innan är en form av Regression Splines.

Målet är att anpassa en stegvis polynomfunktion med begränsningar i skarvarna för att få kontinuitet, kontinuerliga derivator osv.

Termonologi:

**knot (knot)** : Antal brytpunkter som vi använder oss av.

**frihetsgrader (degrees of freedom)** : Antal parametrar som vi kan välja.

**trunkerad polynombas (truncated power basis)** : en funktion  $h(x, \xi)$  som ges av

$$h(x, \xi) = (x - \xi)_+^p = \begin{cases} (x - \xi)^p & \text{if } x > \xi, \\ 0 & \text{otherwise.} \end{cases}$$

Vårt mål: Skatta en stegvis polynomfunktion av grad- $d$  under begränsningen att den ska vara kontinuerlig (kanske även kontinuerliga derivator).

Ett grad- $d$  polynom regression med  $K$  knutar kan modelleras med basfunktioner som

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \dots + \beta_k b_k(x_i) + \varepsilon_i,$$

för ett lämpligt val av basfunktioner. Observera att  $k \neq K$ .

För att få till ett grad- $d$  polynom med rätt egenskaper skapar vi en modell som

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \dots + \beta_d x_i^d \\ + \beta_{d+1} h(x_i, \xi_1) + \beta_{d+2} h(x_i, \xi_2) + \dots + \beta_{d+K} h(x_i, \xi_K) + \varepsilon_i,$$

där  $h(x, \xi) = (x - \xi)^d$ .

Antal frihetsgrader:

- För ett grad- $d$  polynom har vi  $d + 1$  frihetsgrader.
- Med  $K$  knutar ( $K + 1$  intervall) får vi  $(K + 1) \cdot (d + 1)$  frihetsgrader.
- Regression splines ger  $d + 1 + K$  frihetsgrader.

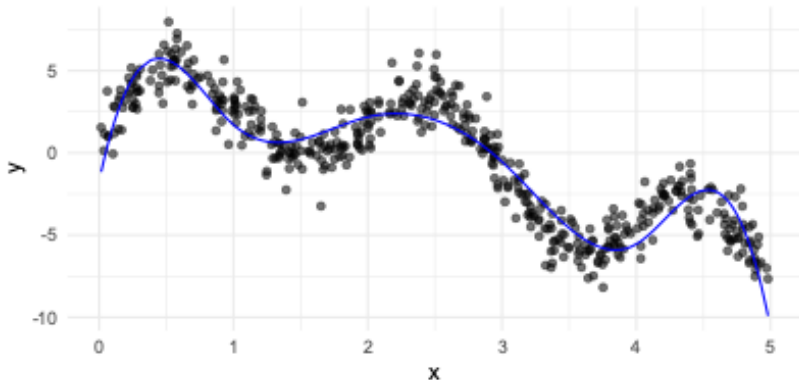
# Regression splines - Sammanfattning

- Att dela upp och skatta oberoende polynom i varje intervall ger diskontinuerlig funktion.
- För varje begränsning (kontinuerlig, kontinuerliga derivator) minskar antalet frihetsgrader.
- Vilken grad ska vi välja?
  - Vanligt med kubiska polynom (grad-3).
- Hur välja antal knutar och deras placeringar?
- Skattningar utanför intervallet blir dåliga.

**Natural spline** : Läger till ett extra krav, att funktionen ska vara linjär i ändarna (innan första och efter sista knuten).
- Jämfört med vanlig polynomregression skulle vi behöva grad  $d + k$  för samma antal frihetsgrader.

# Natural Splines - Exempel

Vi modellerar med kubiska splines med knutar i 1, 2, 3 och 4.

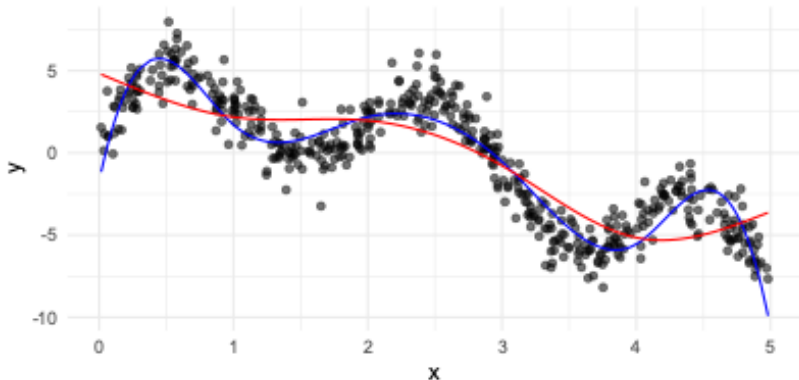




# Natural Splines - Exempel

Vi modellerar med kubiska splines med knutar i 1, 2, 3 och 4.

Lägger till Natural Splines med samma knutar.

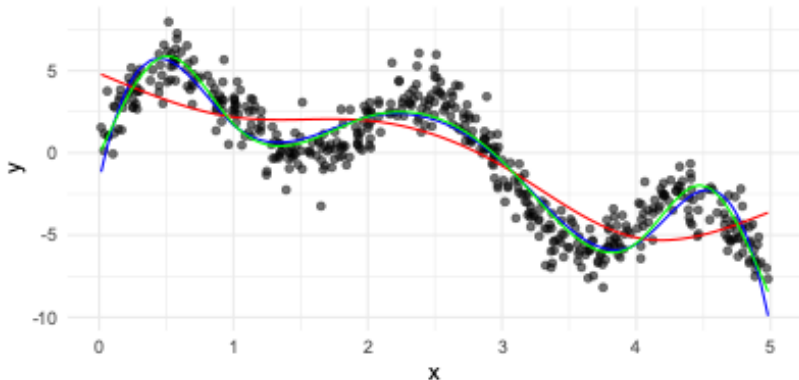


# Natural Splines - Exempel

Vi modellerar med kubiska splines med knutar i 1, 2, 3 och 4.

Lägger till Natural Splines med samma knutar.

Lägger till nya knutar i 0.5 och 4.5 och kör natural splines.

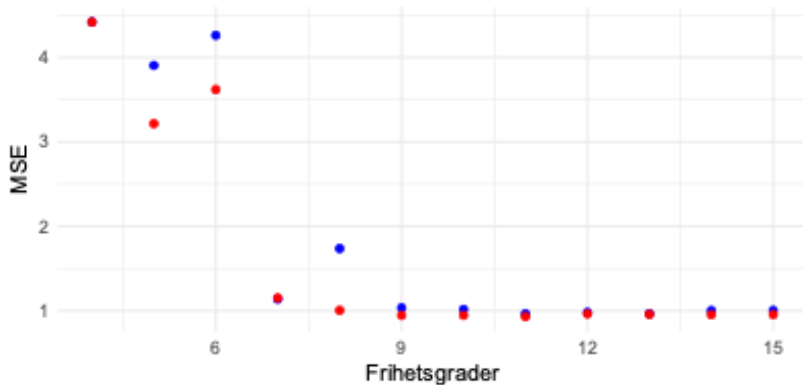


# Antal knutar och deras placeringar

- Funktionen kommer ha mest flexibilitet vid knutarna.
  - Knutarna tillåter parametrarna att ändras.
  - Placera fler knutar där du tror att det behövs.
- Specificera antalet frihetsgrader och sprid ut knutarna jämnt.
  - Givet en förbestämd grad  $d$  så är antalet frihetsgrader  $d + 1 + k$
  - Fler frihetsgrader ger fler knutar.
- Använd kors-validering för att bestämma antalet.

# Antalet knutar och deras placeringar - Exempel

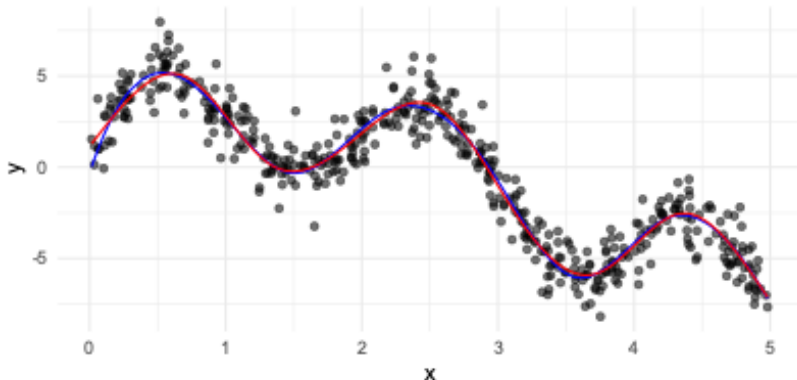
Vi använder samma data som tidigare och beräknar MSE för olika frihetsgrader



# Antalet knutar och deras placeringar - Exempel

Bästa modellen ser vi 8 frihetsgrader för natural splines och 9 frihetsgrader för cubic splines.

Det ger 4 respektive 5 knutar.



# Smoothing Splines

Vårt mål är att hitta en funktion  $g(x)$  som passar vår data väl.

Ett generellt sätt att göra detta på är att hitta  $g$  som minimerar

$$\text{RSS} = \sum_{i=1}^n (y_i - g(x_i))^2$$

Problem: Utan begränsningar på  $g$  kan vi alltid få RSS till 0.

Vad vi vill ha är en funktion som passar data bra men också är "mjuk".

# Mellanspel - Derivator

Givet en funktion  $g(x)$ , låt  $g'(x)$  beteckna dess förstaderivata och  $g''(x)$  dess andraderivata.

- Förstaderivatan  $g'$  är lutningen (hastigheten) av en kurva.
- Andraderivatan  $g''$  är hur lutningen ändrar sig (acceleration).

Vi kan se andraderivatan som ett mått på funktionens grovhet (roughness).

Ett sätt att mäta grovhet av en funktion är att summera andraderivatan över funktionen (kontinuerlig funktion = integral)

$$\int g''(x)^2 dx.$$

Linjär funktion har  $g''(x) = 0$  vilket är en helt mjuk funktion.

Om  $g(x)$  rör sig väldigt mycket kommer integralen bli väldigt stor.

# Smoothing Splines

Vi kan använda måttet för att skatta en funktion som har en viss "mjukhet".

Hitta funktionen  $g(x)$  som minimerar

$$L_{\text{Sm.Spl.}} = \sum_{i=1}^n (y_i - g(x_i))^2 + \lambda \int g''(x) dx.$$

Använder förlust funktion + straffterm likt regularisering vi sett tidigare i Ridge och LASSO.

- Om  $\lambda = 0$  har vi ingen begränsning.
- Om  $\lambda = \infty$  får vi linjär funktion.



$$L_{\text{Sm.Spl.}} = \sum_{i=1}^n (y_i - g(x_i))^2 + \lambda \int g''(x) dx.$$

**Teorem:** Funktionen  $g(x)$  som optimerar förlustfunktionen är stegvis kubiskt med knutar i (unika) punkterna  $(x_1, x_2, \dots, x_n)$  och linjär utanför detta område.

Resultatet är naturliga splines, men inte samma som om man gjort optimeringen som tidigare med samma knutar.

Det blir en "krympt" version av dessa naturliga splines där krympningen beror på  $\lambda$ .

# Smoothing Splines - Sammanfattning

- Efter optimering får vi kubisk natural splines.
- En knut i varje datapunkt.
- För många frihetsgrader? ( $4 + n$ )
- $\lambda$  krymper parameterarna vilket ändrar den effektiva frihetsgraden  $df_\lambda$ .
  - $\lambda = 0$  ger  $df_\lambda = 4 + n$ .
  - $\lambda = \infty$  ger  $df_\lambda = 2$ .
  - Ofta
- Hur välja  $\lambda$ ?

Eftersom smoothing splines resulterar i natural splines med ett fixt antal knutar med ridge regularisering kan vi skriva

$$\hat{\mathbf{g}}_\lambda = \mathbf{S}_\lambda \mathbf{y},$$

där  $\hat{\mathbf{g}}_\lambda$  är lösningen för vårt problem.

Matrisen  $\mathbf{S}_\lambda$  kallas smoothing matrisen.

Se Kap 5.4 i Elements of Statistical Learning för tekniska detaljer.

En intressant detalj från formeln är att hur vi hittar  $\hat{\mathbf{g}}_\lambda$  beror bara på data  $x_i$  och  $\lambda$  **inte** på  $\mathbf{y}$ .

Ett annat resultat är att vi kan definera effektiva frihetsgraderna som

$$df_\lambda := \text{tr}(\mathbf{S}_\lambda) = \sum_{i=1}^n \{\mathbf{S}_\lambda\}_{ii}.$$

Eftersom knutarna positioner är förbestämda  $x_i$  så behöver vi bara bestämma  $\lambda$ .

Det visar sig (igen see Elements of Statistical Learning för detaljer) att vi kan beräkna leave-one out cross-validation felet ( $RSS_{cv}$ ) väldigt effektivt,

$$RSS_{cv} = \sum_{i=1}^n (y_i - \hat{g}_{\lambda}^{(-i)}(x_i))^2 = \sum_{i=1}^n \left[ \frac{y_i - \hat{g}_{\lambda}(x_i)}{1 - \{\mathbf{S}_{\lambda}\}_{ii}} \right]^2.$$

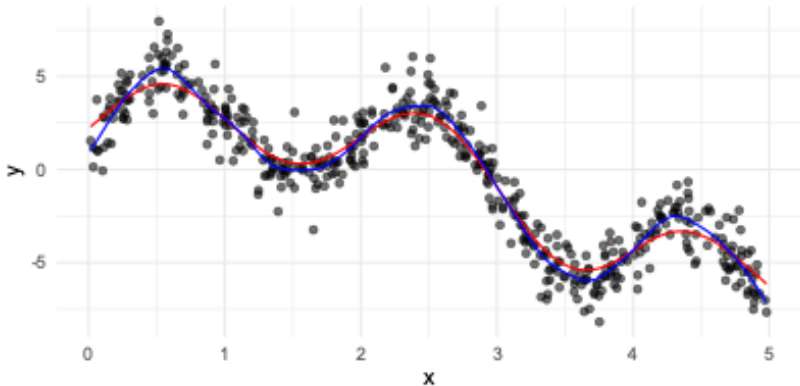
Observera: Vi behöver bara funktionen anpassad på all träningsdata för att beräkna detta.

# Smoothing Splines - Exempel

Använder samma data som tidigare.

Skattar smoothing splines med effektiva frihetsgrader 9. (Blå)

Använder också LOOCV för att hitta bästa värdet 20.6. (Röd)



# Smoothing Splines för klassificering

Vi har hittills gjort smoothing splines för regression, vi kan såklart använda metoden för klassificering.

För logistisk regression har vi

$$p(x) = \mathbb{P}(Y = 1|X = x) = \frac{e^{f(x)}}{1 + e^{f(x)}},$$

där  $f(x)$  är en linjär funktion.

Vi kan såklart använda smoothing splines istället.

Om vi skriver upp optimeringsproblemet får vi log-likelihood (med straffterm)

$$\ell = \sum_{i=1}^n \left[ y_i f(x_i) - \log(1 + e^{f(x_i)}) \right] - \frac{1}{2} \lambda \int \{f''(x)\}^2 dx.$$

Vilket igen kan visas resultera i natural splines med knutar i datapunkterna.



# Flerdimensionella Splines

Hittills har vi pratat om endimensionella modeller. Alla går utäka till flerdimensionella modeller.

Antag att vi har  $x \in \mathbb{R}^2$  och att vi har basfunktioner  $h_{1k}(x_1)$ ,  $k = 1, \dots, K_1$  för första koordinaten och motsvarande  $h_{2k}(x_2)$ ,  $k = 1, \dots, K_2$  för andra koordinaten.

Vi kan då definera den  $K_1 \times K_2$  dimensionella tensor produkt basen

$$g_{jk}(x) = h_{1j}(x_1)h_{2k}(x_2), j = 1, \dots, K_1 \quad k = 1, \dots, K_2.$$

Dessa baser kan vi använda för att beskriva vår tvådimensionella funktion

$$g(x) = \sum_{j=1}^{K_1} \sum_{k=1}^{K_2} g_{jk}(x).$$

# Flerdimensionella Smoothing Splines

Vi kan igen sätta upp smoothing splines problemet för dessa flerdimensionella problem. Om  $x_i \in \mathbb{R}^d$  ställer vi upp problemet

$$\min_g \sum_{i=1}^n \{y_i - g(x_i)\}^2 + \lambda J[g],$$

där  $J[g]$  är en lämplig straffterm.

Om  $d = 2$  har vi

$$J[g] = \int \int_{\mathbb{R}^2} \left[ \left( \frac{\partial^2 g(x)}{\partial x_1^2} \right)^2 + 2 \left( \frac{\partial^2 g(x)}{\partial x_1 \partial x_2} \right)^2 + \left( \frac{\partial^2 g(x)}{\partial x_2^2} \right)^2 \right] dx_1 dx_2$$

# Flerdimensionella Smoothing Splines

Precis som för endimensionella smoothing splines har vi

- $\lambda = 0$  ger en funktion som interpolerar alla datapunkter.
- $\lambda = 0$  ger ett plan.
- Andra värden ger en lösning som kan representeras som en linjärfunktion av basfunktioner med koefficient från ridge regression.

Lösningen är inte kubiska basfunktioner (som i endimensionella fallet) utan **radiala basfunktioner** (radial basis functions).

Också kallade **thin-plate splines**.