732G12 Laboration 2: extra material

Josef Wilzén

29 augusti 2024

Funktionen lm()

I R så använder vi funktionen `lm()` för att skatta regressionsmodeller, lm = linear model. Denna funktion kommer att användas mycket under kursen. Kolla dokumentationen med `?lm`. I denna laboration från R-kursen under "Frivillig fördjupning: Introduktion till linjär regression" så finns det exempel på hur `lm()` kan användas.

R är ett objektorienterat programmeringsspråk, och funktionen `lm()` returnerar objekt av klassen "`lm`", vilket har formen av en lista. Så det är enkelt att plocka önskade delar från den listan vid behov. Det finns en mängd funktioner kopplade till objekt av klassen "`lm`":

- `coef()`: Ger regressionskoefficienter
- `residuals()`: Ger residualvektorn
- `fitted()`: Ger vektorn med anpassade värden ($\hat{y}$)
- `summary()`: Ger en sammanfattande analys av regressionsmodellen. Funktionen returnerar ett objekt av klassen "`summary.lm`". Kolla `?summary.lm` i dokumentationen. `coef()` funkar på dessa objekt.
- `anova()`: Ger anova-tabellen för modellen
- `predict()`: gör prediktioner för (nya) x-värden, alltså beräknar $\hat{y}$ för givna x-värden. Kan även beräkna konfidensintervall och prediktionsintervall för $\hat{y}$. Se `?predict.lm()` för detaljer.
- `plot()`: Ger olika diagnostiska plottar, se `?plot.lm` för detaljer.
- `confint()`: Beräknar konfidensintervall för regressionskoefficienterna
- `model.matrix()`: skapar olika typer av designmatriser som kan användas i `lm()`. Denna funktion använder vi innan vi anropar `lm()`.
- Funktionerna `lm_diagnostics()` och `model_diagnostics()` ger utvärderingsplottar för residualerna. Dessa finns här: länk

Det är också användbart att använda `str()` på lm-objekt. Kolla i ?`lm()` under Value för att se vilka olika delar som finns i objektet.

Övningsuppgifter

Vi kod finns här: länk. Hitta filen ni söker och klicka sen på knappen "Raw". Sen kan ni spara filen med tex `Ctrl + S` . Det går även att läsa in filer direkt i R om man använder url:en som fås när ni har klickat på "Raw". Exempel: läsa in funktionerna i filen "lm_diagnostics.R", då kan vi köra:

```
> source(
+ "https://raw.githubusercontent.com/STIMALiU/732G12_DM/master/labs/lm_diagnostics.R"
+ )
```

1 Diagnos och utvärdering av linjär regression

There exist several useful functions in R for residual diagnostics. Some examples are:

- The `lm.influence()` function, `dffits()`, see documentation: `?lm.influence()`
- A tutorial over common diagnostics tools in R: Regression Model Diagnostics
- A tutorial over common diagnostics tools in R: Regression Diagnostics
- R package `olsrr`, see the vignette: Measures of Influence

2 Modellera icke-linjära samband med linjär regression

2.1 Introduktion

Det finns många sätt som vi kan utöka den vanliga linjära regressionsmodellen för att modellera icke-linjära samband mellan de förklarande variablerna och responsvariabeln. Notera att detta är att stort fält med en rik litteratur, med många ofta avancerade regressionsmodeller. Vi ska här undersöka några enkla utökningar av linjär regression.

Vi kan beskriva regressionsmodeller som

$$y_i = f(x_i) + E \qquad E \sim N\left(0, \sigma^2\right)$$

där $f(x)$ är en funktion som modellerar sambandet mellan x och y.

I fallet med den linjära regressionsmodellen:

$$f(x_i) = \beta_0 + \beta_1 x_i \Leftrightarrow y_i = \beta_0 + \beta_1 x_i + E$$

I detta falla så är $f(x)$ en *linjär* funktion. Vi kan såklart låta $f()$ bero av fler variabler:

$$f(x_{i,1}, x_{i,2}) = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2}$$

eller mer generellt

$$f(\boldsymbol{x}_i) = \boldsymbol{x}_i^T \boldsymbol{\beta}$$

där $\boldsymbol{x}^T = (1, x_{i,1}, x_{i,2}, \ldots, x_{i,p})$ och $\boldsymbol{\beta}^T = (\beta_0, \beta_1, \beta_2, \ldots, \beta_0)$.

1. Vi utgår från att vi har *en* förklarande variabel här, x, men många av de saker som presenteras nedan går att relativt enkelt utökas till flera förklarande variabler.
2. För att undersöka vad för typ av samband som finns mellan x och y så är det oftast lättast plotta vår data i en scatter plot. Då kan se vad för typ av samband det finns mellan x och y. Ibland så kan vi göra lämpliga transformationer av y för att få sambanden mellan x och y mer linjärt, tex logaritmtransformation[1].
   (a) Är de oberoende? Dvs värdena på y ändras inte när värden på x ändras.
   (b) Linjärt beroende: y ökar eller minskar linjärt när värden på x ökar.
   (c) Icke-linjärt beroende: Här kan vi ha många olika varianter på hur y ändras när x ändras. Exempel på icke-linjära samband är:
   $$y = x^2 + E \qquad y = sin(x) + E \qquad y = tanh(x) + 0.1 \cdot x^3 + E$$
3. Om vi har skäl att tro att x och y har ett icke-linjärt samband, så kan vi skapa en eller flera nya variabler genom att använda icke-linjära transformationer på x. Ett vanligt exempel är en polynomtransformation[2]: $x^2, x^3, \ldots, x^h$ Sen använder vi dessa som vanliga kovariater i en linjär regressionsmodell. Vi får då att
   $$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \ldots + \beta_h x^h \tag{1}$$
   Vi kan spara dessa transformationer i nya variabler $z_1 = x$, $z_2 = x^2$, $z_3 = x^3$ osv, och sen använder vi dessa i vår designmatris $X = (1, z_1, z_2, \ldots, z_h)$, och modellen blir då $y = X\beta + E$.

[1] Notera: $log()$ syftar på logartimen med bas e i laborationerna
[2] Notera: skapar vi polynom av x så vill vi i regel centrera eller standardisera x först.

4. Vi antar då att y beror linjärt på de nya variablerna z_1 till z_h, och vi kan skatta β med OLS. Så vårt modellantagande är att vi antar att y beror linjärt på de nya variablerna, men att y beror *icke-linjärt* på de ursprungliga variablen x. Så denna idé att vi har variabler, som vi sen transformerar icke-linjärt och som vi sen använder som de nya variablerna i vår designmatris kan vi generalisera till andra transformationer än polynomtransformationer.

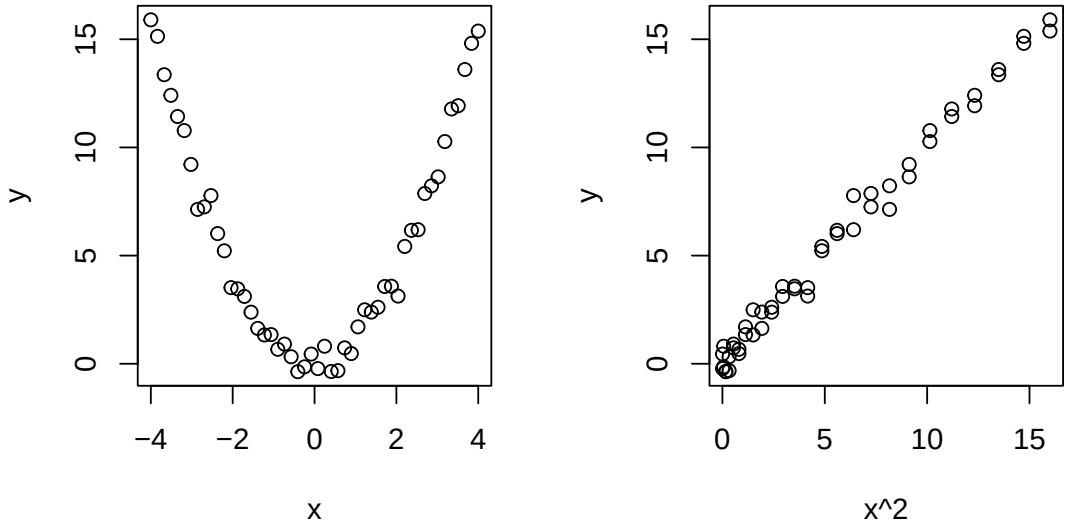
5. Notera: Polynom av av tillräckligt hög ordning kan anpassa kontinuerliga funktioner godtyckligt bra. Så om vi har tillräckligt högt h i funktionen i (1) kan vi i många fall få en bra anpassning av y. Dock så ökar risken för överanpassning ju högre h vi har. Det finns vissa problem med höga ordningens polynom (känsliga för outliers tex), ofta så väljer vi tredje eller fjärdegradspolynom som högst.

6. Vi låter $z = g(x)$ vara en funktion som transformerar x. Exempel:

   (a) $z = g(x) = log(x)$
   (b) $z = g(x) = exp(x)$
   (c) $z = g(x) = sin(x)$
   (d) $z = g(x) = tanh(x)$
   (e) $z = g(x) = \sqrt{x}$
   (f) $z = g(x) = 0.5 \cdot log(x) + 2 \cdot sin(x)$

7. Se ett exempel nedan. Sambandet mellan x och y är kvadratiskt, men mellan x^2 och y är sambandet linjärt.

```
> x<-seq(-4,4,length=50)
>
> y<-x^2+rnorm(50,sd=0.5)
> par(mfrow=c(1,2))
> plot(x,y)
> plot(x^2,y)
```



8. Som vi ser ovan kan vi skapa en mängd olika transformationer beroende på hur vi definerar $g(x)$. Vi kan tänka oss att vi skapar flera transformationer av x, vi kan kalla transformation nummer j

för $z_j = g_j(x)$, alla våra transformationer blir då:

$$z_j = g_j(x) \qquad j = 1, \ldots, J \quad \Leftrightarrow$$

$$z_1 = g_1(x), z_2 = g_2(x), \ldots, z_J = g_J(x)$$

9. När vi skapar nya variabler på detta sätt så kan vi kalla dessa för *basfunktioner* (basis functions) eller baser. Så vi skapar en eller flera basfunktioner som baseras på den ursprungliga variablen x. Vi skriver

$$y = f(x) + E$$

$$f(x) = \beta_0 + \sum_{j=1}^{J} \beta_j \cdot g_j(x) \tag{2}$$

$$= \beta_0 + \beta_1 \cdot g_1(x) + \beta_2 \cdot g_2(x) + \ldots + \beta_J \cdot g_J(x)$$

$$= \beta_0 + \beta_1 \cdot z_1 + \beta_2 \cdot z_2 + \ldots + \beta_J \cdot z_J$$

10. Eftersom funktionen $f(x)$ i ekvation (2) är linjär med avssende på basfunktionerna $g_j(x)$ kan vi enkelt skatta β med OLS. Ekvation (2) ger oss ett ramverk för att modellera många olika typer av icke-linjära samband mellan x och y. Nedan kommer vi titta närmare på ett antal fall där vi kan använda oss av detta ramverk.

2.2 Stegfunktionen

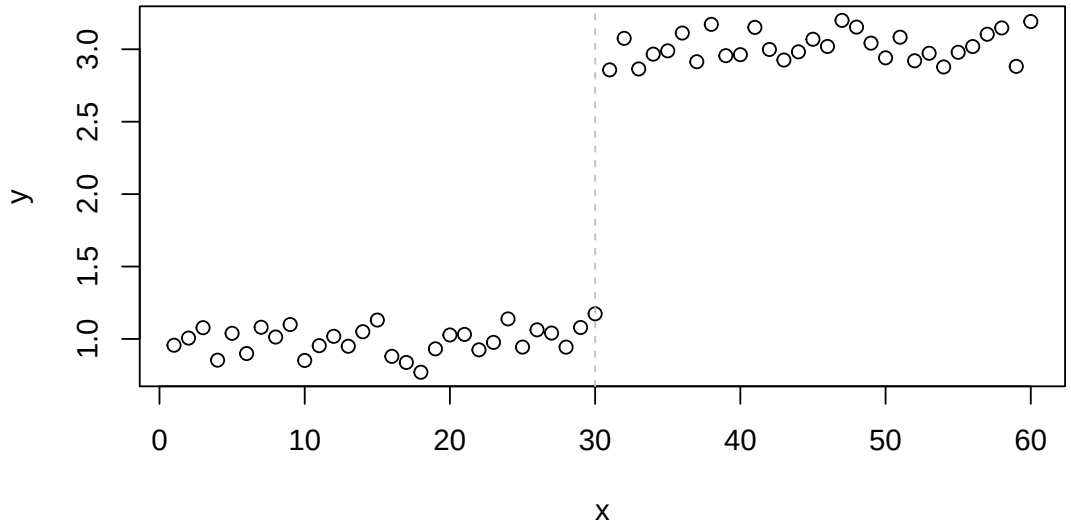
1. Stegfunktionen: En vanlig funktion inom matematiken är stegfunktionen $u(x)$, som defineras som

$$u(x) = \begin{cases} 0 & om\, x \leq a \\ 1 & om\, x > a \end{cases}$$

där a är en konstant som vi väljer. Stegfunktionen är användbar om vi har plötsliga nivåskillnader eller skiften i y.

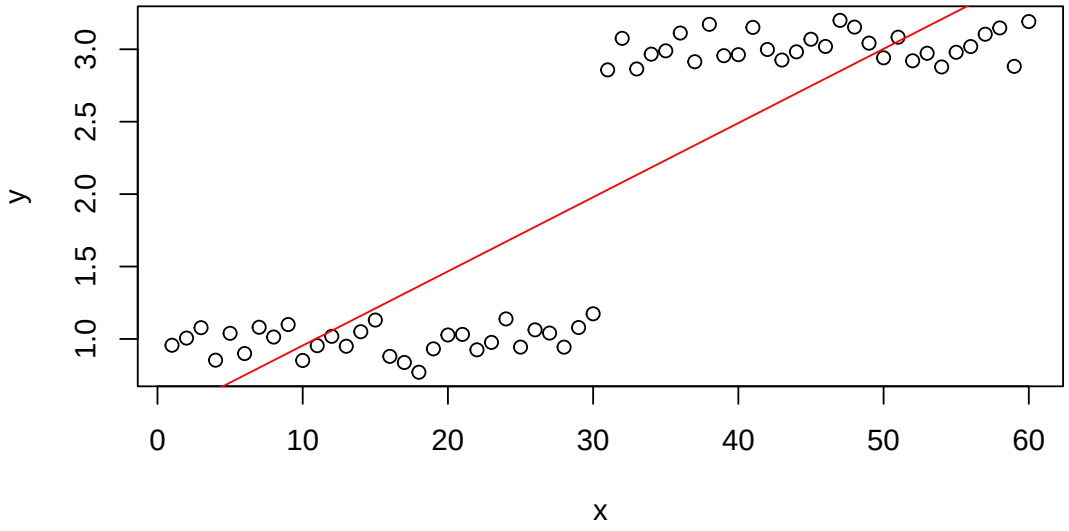
2. Se figuren nedan. Vi har en tydlig nivåskillnad i y när $x = 30$, vilket är vid den streckade linjen.

```
> x<-1:60
> set.seed(2322)
> y<-c(rnorm(n=30,mean=1,sd=0.1),rnorm(n=30,mean=3,sd=0.1))
> plot(x,y)
> abline(v=30,lty="dashed",col="gray")
```



3. Vi testar nu att skatta en enkel linjär regression för x och y.

```
> lm_obj1<-lm(y~x)
> summary(lm_obj1)


Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-0.9331 -0.3964 -0.0457  0.3639  0.9942

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.44280    0.13291    3.33   0.0015 **
x            0.05120    0.00379   13.51   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.508 on 58 degrees of freedom
Multiple R-squared:  0.759,Adjusted R-squared:  0.755
F-statistic:  183 on 1 and 58 DF,  p-value: <2e-16

> y_hat<-fitted(lm_obj1)
> plot(x,y)
> lines(x,y_hat,col="red")
```

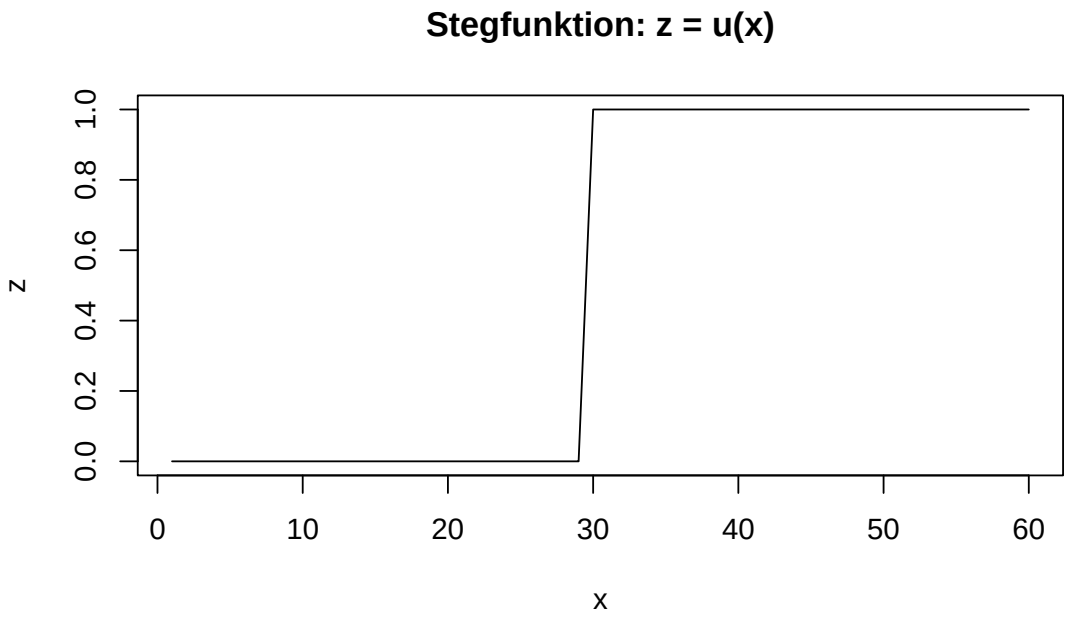


4. Vi ser att vi får signifikanta parameterar, men vi ser när vi plottar $\hat{y}$ tillsammans med data att vi har en mycket dålig anpassning. Vi testar nu att skapa den nya variablen

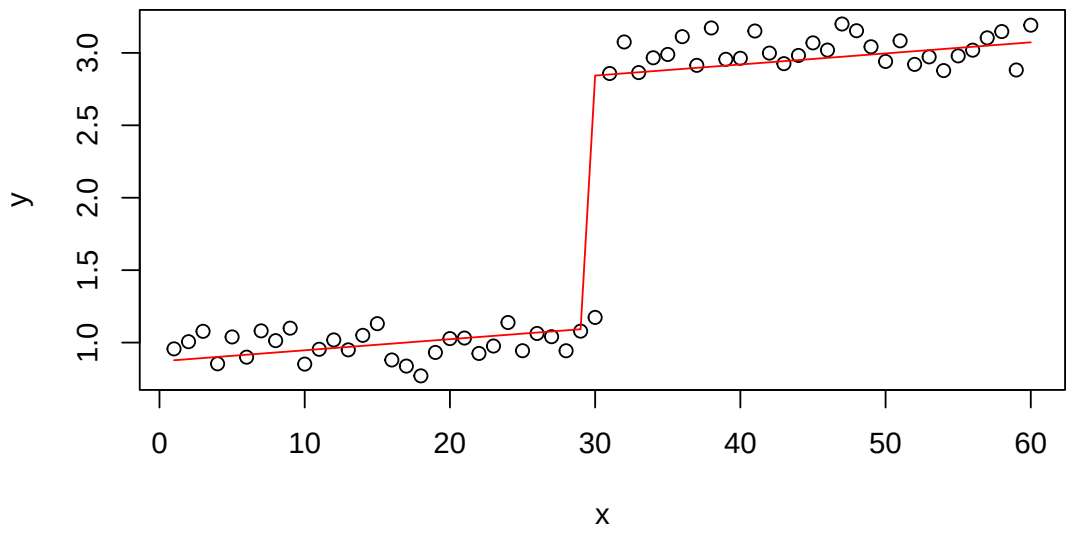
$$z = u(x) = \begin{cases} 0 & om\, x \leq 30 \\ 1 & om\, x > 30 \end{cases}$$

och lägger till den vår modell.

```
> z<-ifelse(x<30,0,1)
> plot(x,z,t="l",main="Stegfunktion: z = u(x)")
```



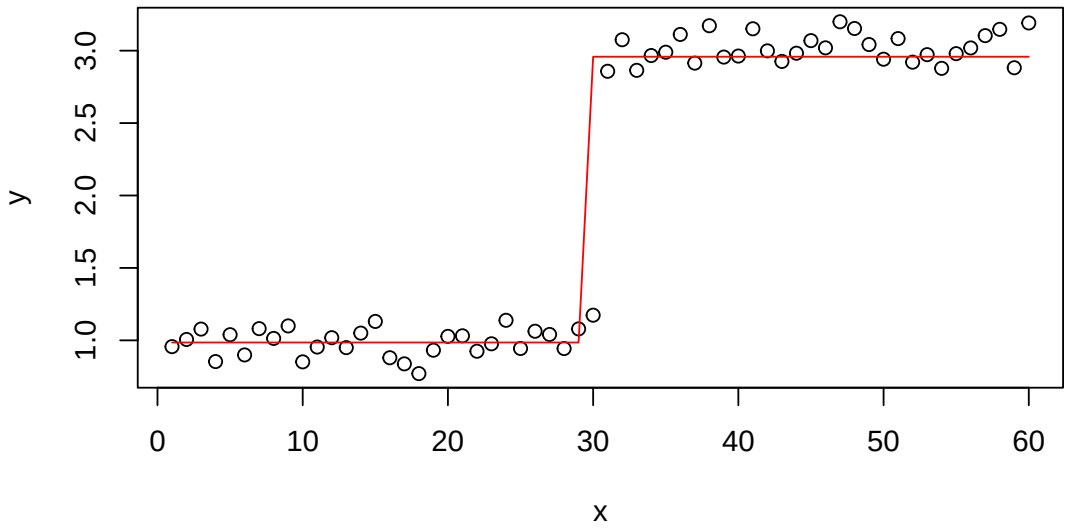
```
> lm_obj2<-lm(y~x+z)
> summary(lm_obj2)


Call:
lm(formula = y ~ x + z)

Residuals:
    Min      1Q  Median      3Q     Max
-1.6696 -0.0493  0.0245  0.0962  0.2675

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.87031    0.07271   11.97   <2e-16 ***
x            0.00765    0.00373    2.05    0.045 *
z            1.74356    0.12918   13.50   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.25 on 57 degrees of freedom
Multiple R-squared:  0.943,Adjusted R-squared:  0.941
F-statistic:  468 on 2 and 57 DF,  p-value: <2e-16

> y_hat<-fitted(lm_obj2)
> plot(x,y)
> lines(x,y_hat,col="red")
```



5. Vi har nu en tydligt bättre anpassning för $\hat{y}$. Vi ser nu att x inte längre är signifikant, men att z är tydligt signifikant. Vi testar nu att bara ha med z som förklarande variabel

```
> lm_obj3<-lm(y~z)
> summary(lm_obj3)
```

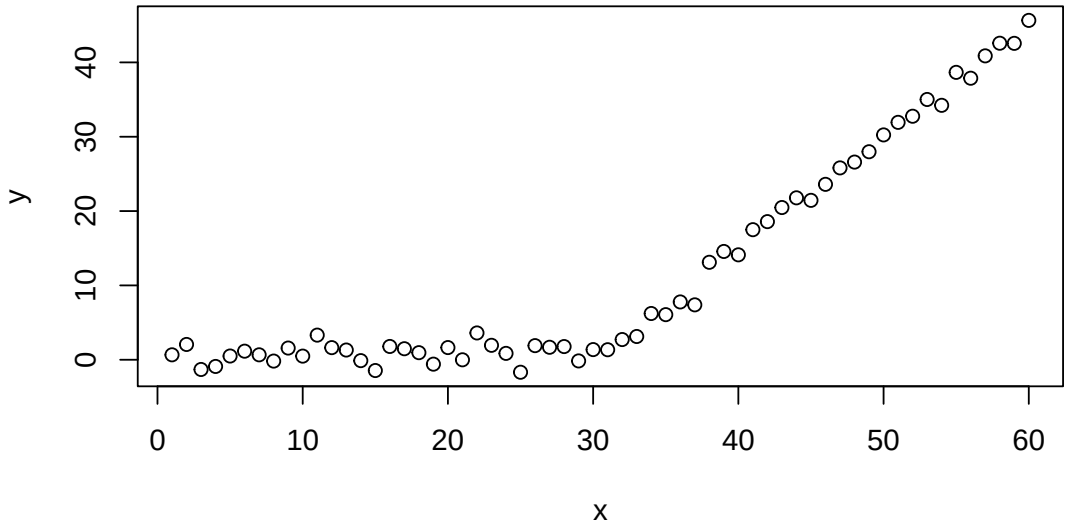
```
Call:
lm(formula = y ~ z)

Residuals:
    Min      1Q  Median      3Q     Max
-1.7843 -0.0414  0.0262  0.0943  0.2413

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.9851     0.0478    20.6   <2e-16 ***
z             1.9731     0.0664    29.7   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.257 on 58 degrees of freedom
Multiple R-squared:  0.938,Adjusted R-squared:  0.937
F-statistic:  882 on 1 and 58 DF,  p-value: <2e-16

> y_hat<-fitted(lm_obj3)
> plot(x,y)
> lines(x,y_hat,col="red")
```



6. Vi ser nu att vi har en ännu bättre anpassning till data, och att z är mycket signifikant. Den sista modellen är skattar den rätta modellen, den som data generades ifrån.

7. Stegfunktioner kan hjälpa oss att modellera många olika sorters funktioner $f(x)$. Vi kan få vår modell att ”bete sig” annorlunda i olika regioner av x-värden. Se datan nedan.

```
> x<-1:60
> set.seed(755)
> y<-ifelse(x<=30,1,1.5*(x[31:60]-30))+rnorm(n=60,sd=1)
> #y#<-c(rnorm(n=30,mean=1,sd=0.1),0.1*x[31:60]+rnorm(n=30,mean=0,sd=0.1))
> plot(x,y)
```



8. Här har vi en konstantfunktion fram till $x = 30$ och sen verkar vi ha en linjär funktion som med avseende på x. Vi skapar igen variabeln

$$z = u(x) = \begin{cases} 0 & om\, x \leq 30 \\ 1 & om\, x > 30 \end{cases}$$

och sen skattar vi två olika modeller:

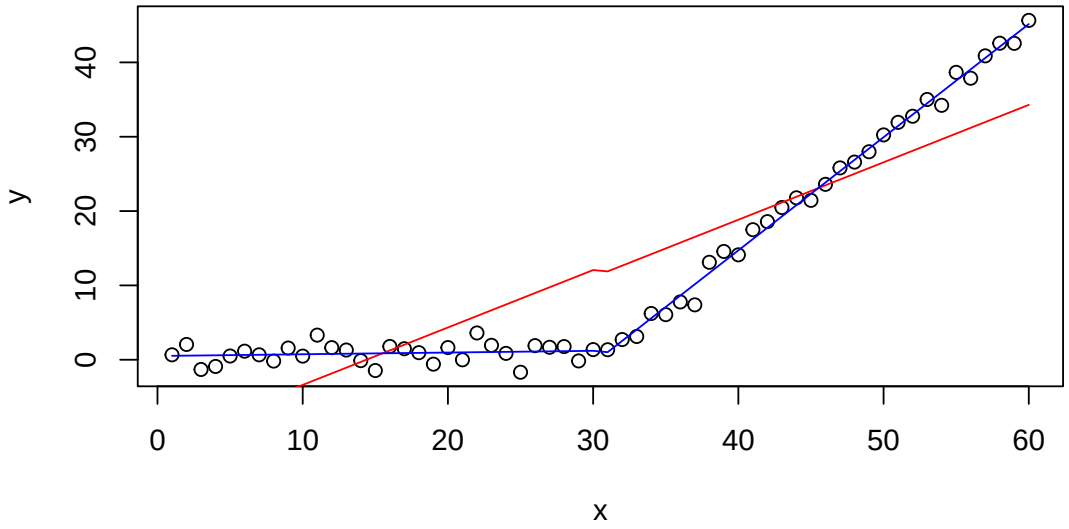
$$y = \beta_0 + \beta_1 x + \beta_2 z + E$$

$$y = \beta_0 + \beta_1 x + \beta_2 z + \beta_3 x \cdot z + E$$

Den andra modellen har en interaktionsterm mellan x och z. Vi testar att skatt dessa modeller.

```
> x<-1:60
> z<-ifelse(x<=30,0,1)
> xz<-x*z
> model1<-lm(y~x+z)
> model2<-lm(y~x+z+xz)
>
> summary(model1)


Call:
lm(formula = y ~ x + z)

Residuals:
    Min      1Q  Median      3Q     Max
-11.454  -4.822  -0.009   5.779  11.623

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -11.111      1.990   -5.58  6.8e-07 ***
x              0.773      0.101    7.67  2.4e-10 ***
```

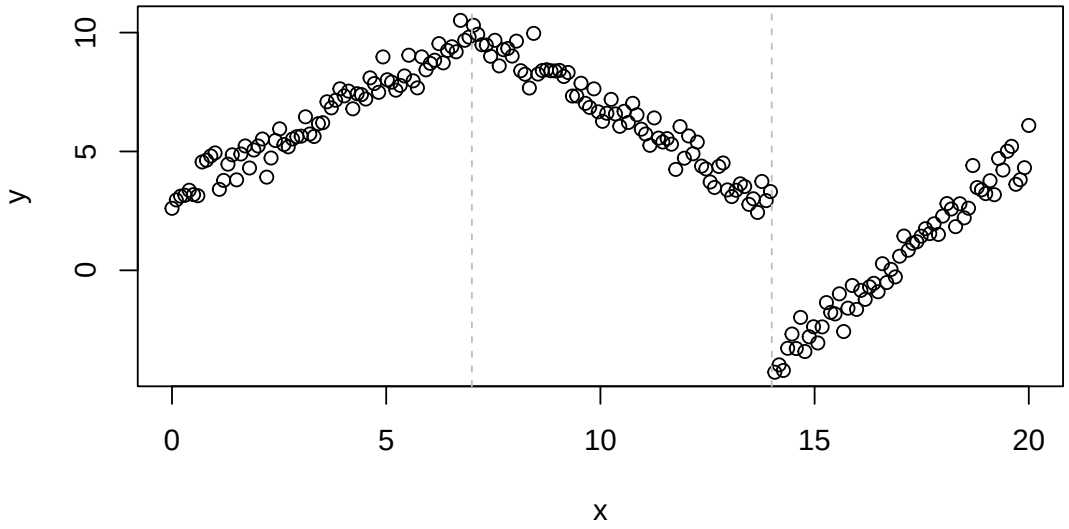
```
z             -0.956      3.490   -0.27     0.79
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.75 on 57 degrees of freedom
Multiple R-squared:  0.795,Adjusted R-squared:  0.788
F-statistic:  111 on 2 and 57 DF,  p-value: <2e-16

> summary(model2)


Call:
lm(formula = y ~ x + z + xz)

Residuals:
   Min     1Q Median     3Q    Max
-2.765 -0.874  0.235  0.699  2.591

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.5047     0.4371    1.15     0.25
x             0.0232     0.0246    0.94     0.35
z           -46.6706     1.2214  -38.21   <2e-16 ***
xz            1.4988     0.0348   43.04   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.17 on 56 degrees of freedom
Multiple R-squared:  0.994,Adjusted R-squared:  0.994
F-statistic: 3.09e+03 on 3 and 56 DF,  p-value: <2e-16

> y_hat1<-fitted(model1)
> y_hat2<-fitted(model2)
>
> plot(x,y)
> lines(x,y_hat1,col="red") # utan interaktion
> lines(x,y_hat2,col="blue") # med interaktion
```



9. Den röda linjen är anpassade värden för modellen utan interaktion och den blå linjen är anpassade värden för modellen med interaktion. Här ser vi att den modellen som har med interaktionen mellan x och en lämplig stegfunktion kan modellera y väl. Vi kan ha olika lutning på linjen i olika regioner av x.

10. Vi kan generalisera detta till mer komplicerade situationer. Sen data nedan. Vi har tydliga brott vid $x = 7$ och $x = 14$. Vi kan använda den informationen när vi skapar en regressionsmodell.

```
> x<-seq(0,20,length=200)
> y1<-3+x
> y2<-17+-1*x
> y3<--25+1.5*x
> set.seed(345)
> y<-ifelse(x<7,y1,ifelse(x<14,y2,y3))+rnorm(200,sd=0.5)
> plot(x,y)
> abline(v=c(7,14),lty="dashed",col="gray")
```

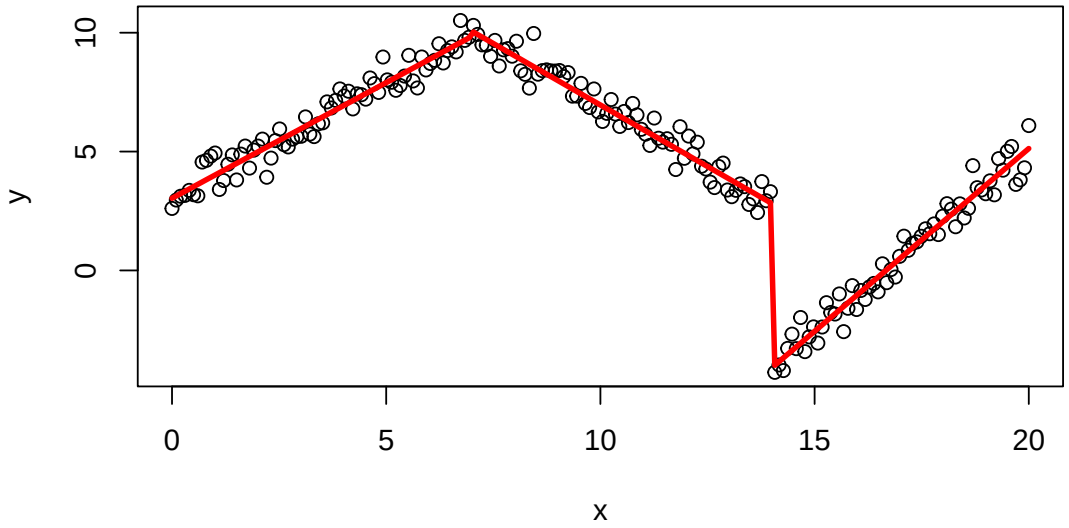


11. Vi skattar en modell med två stegfunktioner, och interaktioner mellan dessa och x.

```
> # stegfunktioner:
> z1<-ifelse(x<=7,1,0)
> z2<-ifelse(x<=14,1,0)
> # interaktioner:
> xz1<-x*z1
> xz2<-x*z2
> # skatta modell:
> model<-lm(y~x+z1+z2+xz1+xz2)
> summary(model)


Call:
lm(formula = y ~ x + z1 + z2 + xz1 + xz2)

Residuals:
    Min      1Q  Median      3Q     Max
-1.2667 -0.3531 -0.0268  0.3156  1.3956

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -25.6246     0.6463   -39.6   <2e-16 ***
x             1.5376     0.0377    40.7   <2e-16 ***
z1          -14.2725     0.3422   -41.7   <2e-16 ***
z2           42.9326     0.7214    59.5   <2e-16 ***
xz1           2.0078     0.0424    47.4   <2e-16 ***
xz2          -2.5725     0.0482   -53.4   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.509 on 194 degrees of freedom
Multiple R-squared:  0.98,Adjusted R-squared:  0.979
F-statistic: 1.89e+03 on 5 and 194 DF,  p-value: <2e-16
```

12. Vi ser att alla variabler är signifikanta och behövs för modelleringen. Vi tittar på anpassade värden nedan, och vi ser att vi har en bra anpassning av sambandet mellan x och y.

```
> plot(x,y)
> lines(x,fitted(model),col="red",lwd=3)
```



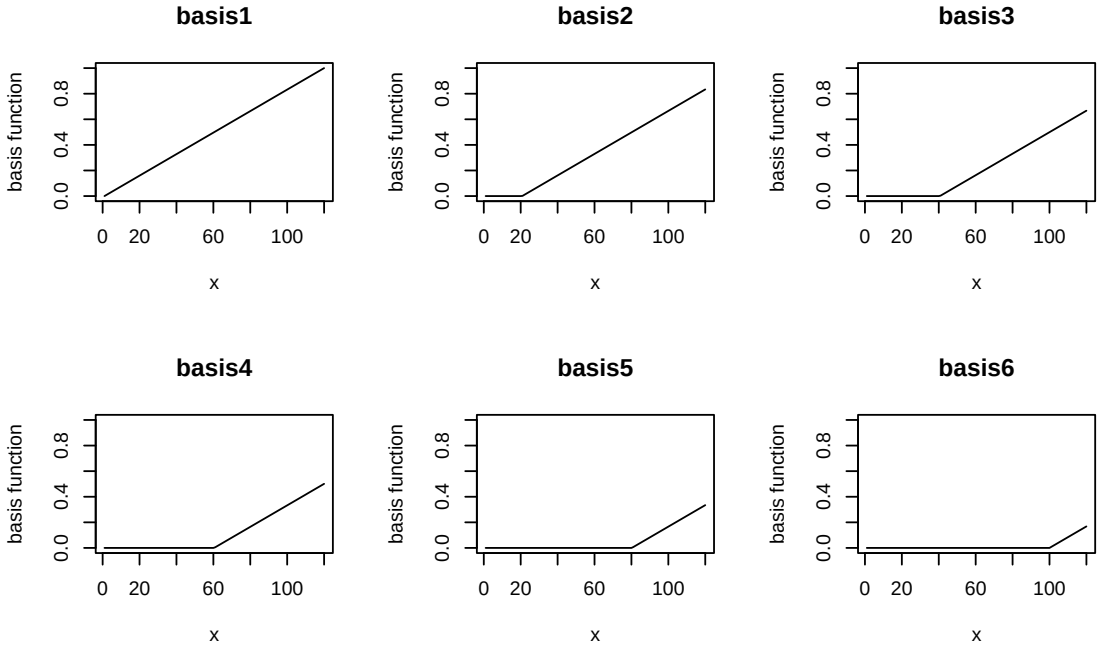
2.3 Trunkerade polynombaser

1. Trunkerade polynombaser: ovan hade vi ett fall där vi hade olika linjära samband i olika regioner av x: Ett samband för $x <= 7$, ett för $7 < x \leq 14$ och ett för $14 < x$. Vi kan generalisera denna approach genom att använda en klass av basfunktioner som kallas trunkerade polynombaser (truncated power basis).

2. Trunkerade polynombaser är polynom, som är ”avstängda” en för en del av värdena på x och sen är de bara ”vanliga polynom”. Man brukar ha en grupp av sådana basfunktioner. Detta är exempel på en variant av *splinemodell*. Vi kollar hur de ser ut med koden nedan. Vi använder en funktion som finns här.

```
> # läs in funktion
> source(
+ "https://raw.githubusercontent.com/STIMALiU/732G12_DM/master/labs/trunc_power_basis.R"
+ )
>
> # x är en vektor med data
> # no_basis är antal basfunktioner
> basis_list<-trunc_power_basis(x = 1:120,no_basis = 6,type = "linear",show_plot = FALSE)
> str(basis_list)

List of 2
 $ basis_mat: num [1:120, 1:6] 0 0.0084 0.0168 0.0252 0.0336 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
```

```
  .. ..$ : chr [1:6] "basis1" "basis2" "basis3" "basis4" ...
 $ x        : int [1:120] 1 2 3 4 5 6 7 8 9 10 ...

> par(mfrow=c(2,3))
> for(i in 1:6){
+         plot(basis_list$x,basis_list$basis_mat[,i],t="l",xlab="x",
+                 ylab="basis function",ylim=c(0,1),main=colnames(basis_list$basis_mat)[i])
+ }
```

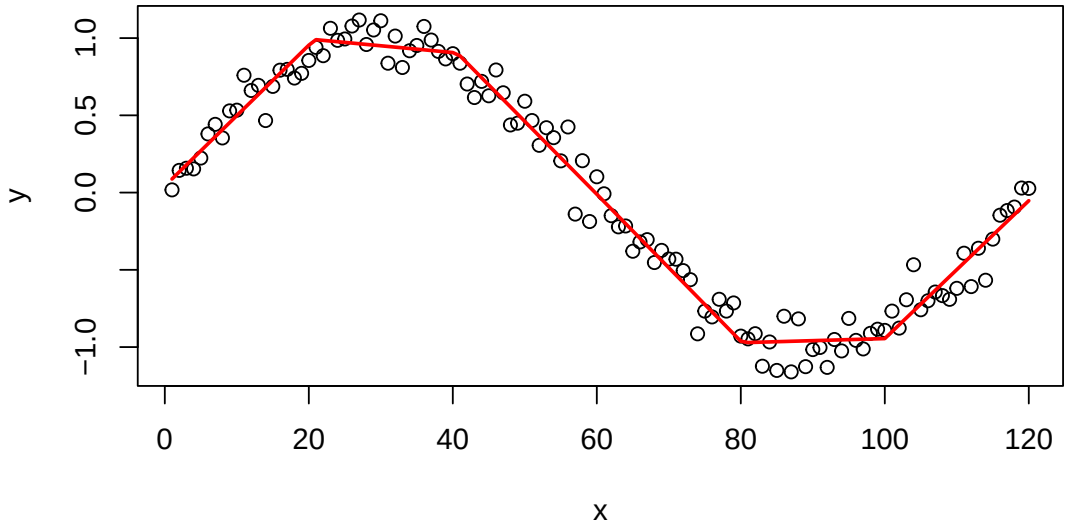


3. Ovan ser vi att den första basfunktionen är en vanlig linjär funktion. Den andra är ”avstängd” (=trunkerad) i början, sen är den en vanlig linjär funktion. Alla de andra är mer och mer trunkerade (större andel som är noll). Default är att trunkeringspunkterna är jämt spridda inom x variationsområde. Trunkeringspunkterna kallas knutar (knots), och vi kan välja dessa manuellt om vi vill. Då ger vi en numerisk vektor till argumentet `knots=`. Om vi använder dessa basfunktioner i en regressionsmodell så har vi exempel på en spline-modell. Det finns många spline-modeller, och detta är en av de enklare.

4. Vi testar att använda trunkerade polynombaser för att modellera en sinusfunktion med brus.

```
> x<-1:120
> basis_list<-trunc_power_basis(x = 1:120,no_basis = 6,type = "linear",show_plot = FALSE)
> set.seed(2423)
> y<-sin(pi*x/60)+rnorm(n = 120,sd=0.1)
> plot(x,y)
> df<-data.frame(y=y,basis_list$basis_mat)
> head(df)

         y    basis1 basis2 basis3 basis4 basis5 basis6
1 0.017374 0.0000000      0      0      0      0      0
2 0.143701 0.0084034      0      0      0      0      0
3 0.157536 0.0168067      0      0      0      0      0
4 0.153171 0.0252101      0      0      0      0      0
5 0.223521 0.0336134      0      0      0      0      0
6 0.379642 0.0420168      0      0      0      0      0
```

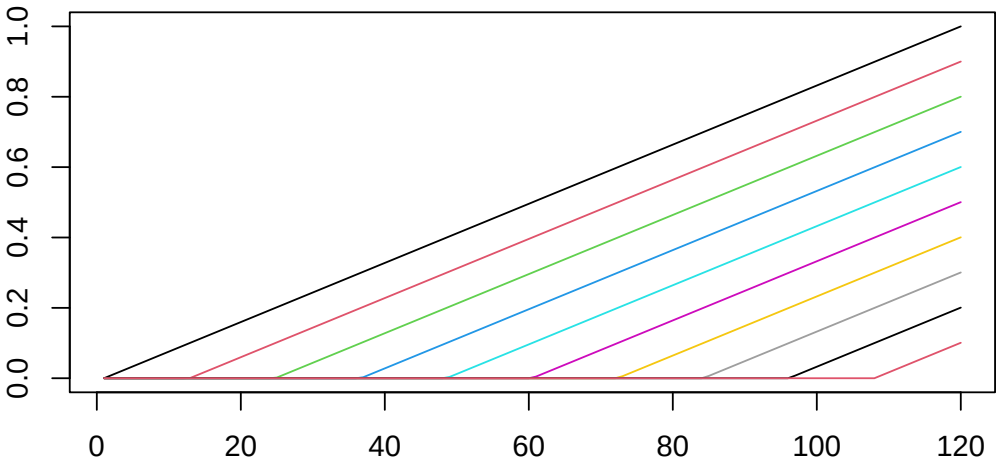
```
> tail(df)

             y  basis1  basis2  basis3  basis4  basis5  basis6
115 -0.301239 0.95798 0.79160 0.62521 0.45882 0.29244 0.12605
116 -0.146450 0.96639 0.80000 0.63361 0.46723 0.30084 0.13445
117 -0.116284 0.97479 0.80840 0.64202 0.47563 0.30924 0.14286
118 -0.093748 0.98319 0.81681 0.65042 0.48403 0.31765 0.15126
119  0.029527 0.99160 0.82521 0.65882 0.49244 0.32605 0.15966
120  0.027135 1.00000 0.83361 0.66723 0.50084 0.33445 0.16807

> A<-lm(y~.,data=df)
> plot(x,y)
> lines(x,fitted(A),col="red",lwd=2)
```



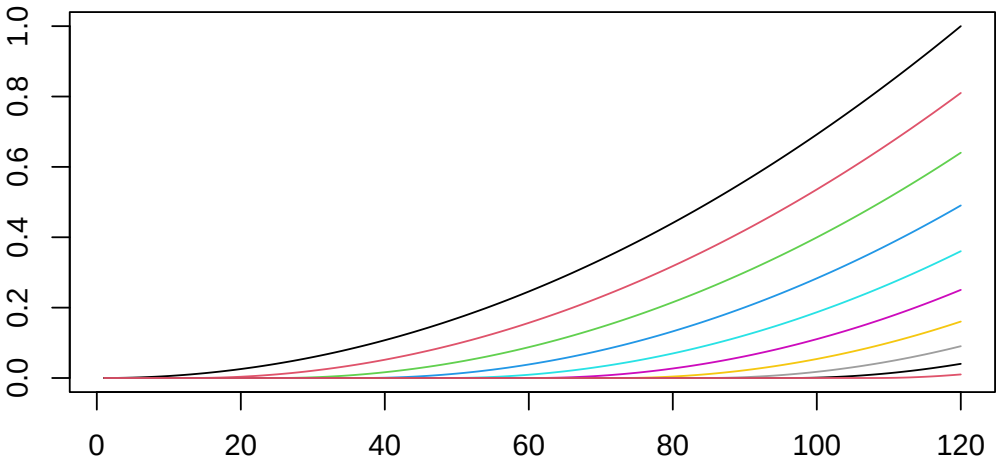
5. Vi ser att med 6 stycken linjära trunkerade polynombaser så får vi en ok anpassning av y. Ökar vi antalet basfunktioner så får vi en mer flexibel modell, men riskerar också att få överanpassning, dvs vi börjar modellera bruset i data. Testa hur anpassningen blir om vi har `no_basis = 3`, `no_basis = 10`, `no_basis = 100`. Vilken av dessa modeller är du mest nöjd med? Beräkna AIC för alla modeller, vilken har lägst värde?

6. Vi kan välja om vi vill ha linjära baser eller kvadratiska baser. De linjära kan vara bättre om det är skarpa skiftningar i y när x ändrar sig. De kvadratiska är bättre om y varierar mer mjukt när x ändrar sig.

```
> # x är en vektor med data
> # no_basis är antal basfunktioner
> # vi plottar alla baserna i samma plot:
> # linjära baser:
> basis_list1<-trunc_power_basis(x = 1:120,no_basis = 10,type = "linear",show_plot = TRUE)
```




```
> # kvadratiska baser:
> basis_list2<-trunc_power_basis(x = 1:120,no_basis = 10,type = "quadratic",show_plot = TRUE)
```



7. Testa att modellera sinusfunktionen ovan med kvadratiska trunkerade polynombaser. Testa låta `no_basis` vara 5, 10 och 30. Vilken anpassning är ni mest nöjda med?

8. Har vi många basfunktioner i vår ”fulla modell” så är det ofta bra att använda metoder för automatiskt modellval. Här kan krympningsmetoder som ridge eller LASSO användas. Andra alternativ är framåtvalsmetoden eller Stepwise forward selection[3]. Har vi många basfunktioner och/eller många observationer så rekomenderas paketet `bigstep` , se här.

[3] Varför är inte backward elimination så bra här?

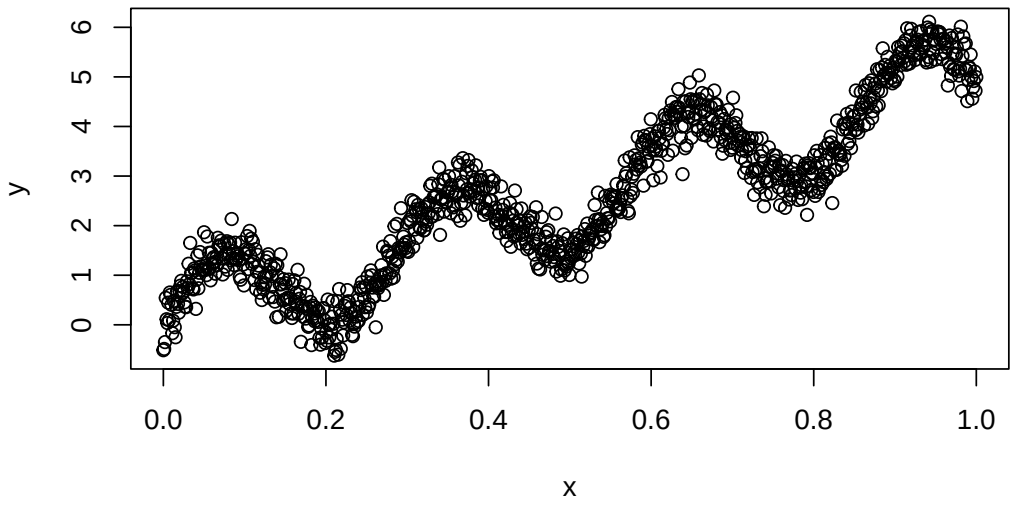
9. Generera data nedan. Skapa sedan en modell där ni modeller y med hjälp av trunkerade polynombaser, och där ni väljer basfunktioner med Stepwise forward selection med BIC[4] som kriterie.

   (a) Använd 100 linjära trunkerade polynombaser som den fulla modellen. Ta reda på hur många basfunktioner som blev kvar i den slutgiltiga modellen. Plotta anpassade värden tillsammans med data. Hur bra blev anpassningen?

   (b) Använd 100 kvaderade trunkerade polynombaser som den fulla modellen. Ta reda på hur många basfunktioner som blev kvar i den slutgiltiga modellen. Plotta anpassade värden tillsammans med data. Hur bra blev anpassningen?

   (c) Lösningsförslag finns här.

```
> x<-seq(0,1,length=1000)
> set.seed(64)
> y<-sin(pi*x*7)+x*5+rnorm(n = 1e3,sd=0.3)
> plot(x,y)
```



10. Upprepa uppgiften ovan men använd: ridge, lasso och elasticnet med $\alpha = 0.5$ för att välja modell. Använd korsvalidering för att välja λ.

2.4 Consinusbaser

En annan variant av basfunktioner som kan modellera icke-linjära funktioner är cosinusbaser. De defineras som

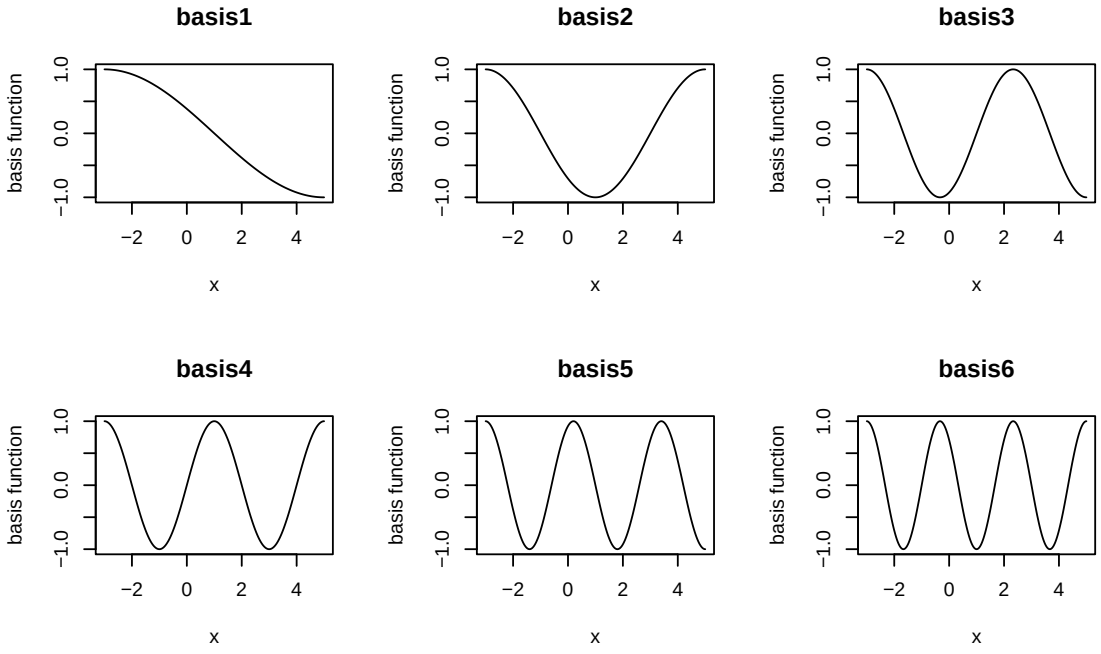
$$x_{min} = min\,(x) \qquad x_{max} = max\,(x) \qquad L = x_{max} - x_{min}$$
$$g_h\,(x) = cos\left(\frac{\pi \cdot h \cdot (x - x_{min})}{L}\right) \qquad h = 1, 2, \ldots, H \tag{3}$$

där x är en vektor med vår förklarande variabel, x_{min} är det minsta värdet av x och x_{max} är det största värdet av x. h är ordningen för basen, och H är den högsta ordningen. Vi kan likna h med graden i ett polynom. Så vi kan givet vår förklarande variabel skapa H stycken basfunktioner baserat på ekvation (3). Desto högre H är desto mer flexibel och icke-linjär funktion kan vi modellera med dessa baser. Samtidigt ökar risken för överanpassning ju högre H som vi har. En fördel med dessa basfunktioner är att de är okorrelerade med varandra.

[4]Notera: om vi har många variabler i vår modell så tenderar AIC att välja en för stor modell, då är ofta BIC eller AICc bättre att använda.

1. Vi testar att skapa några cosinusbaser och ser hur de ser ut. Vi använder en funktion som finns här.

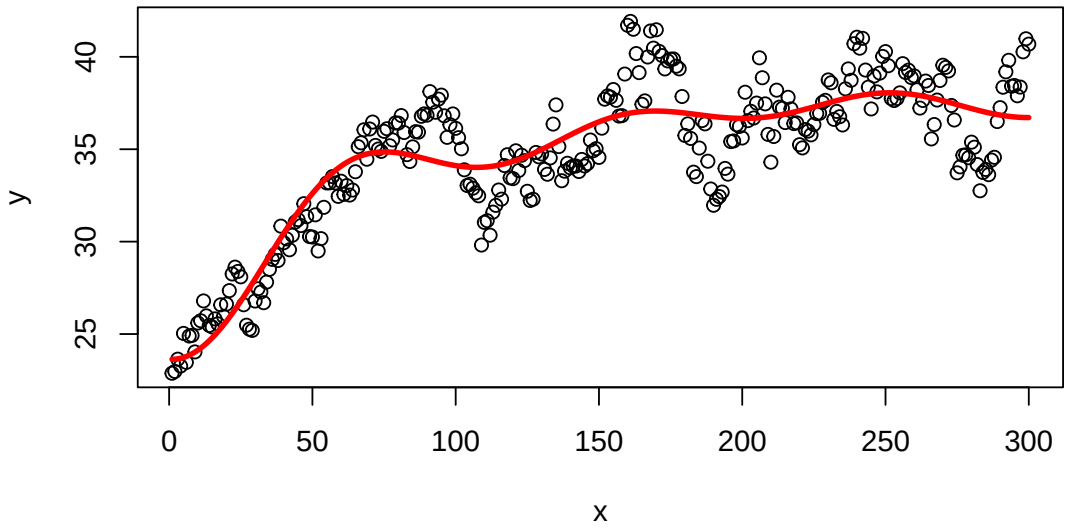
```
> source(
+ "https://raw.githubusercontent.com/STIMALiU/732G12_DM/master/labs/cosine_basis.R"
+ )
>
> x<-seq(-3,5,length=300)
> # order: högsta ordningen på cosinusbaserna
> cos_list<-cosine_basis(x = x,order = 6)
> dim(cos_list$basis_mat)

[1] 300   6

> par(mfrow=c(2,3))
> for(i in 1:6){
+         plot(cos_list$x,cos_list$basis_mat[,i],t="l",xlab="x",ylab="basis function",ylim=c(-1,
+ }
```



2. Vi ser att den första basen har en halv period mellan x_{min} och x_{max}. Sen ju högre ordning som basen har desto fler svängningar har den mellan x_{min} och x_{max}. Vi testar att använda dessa basfunktioner för att modellera en observerad y-variabel.

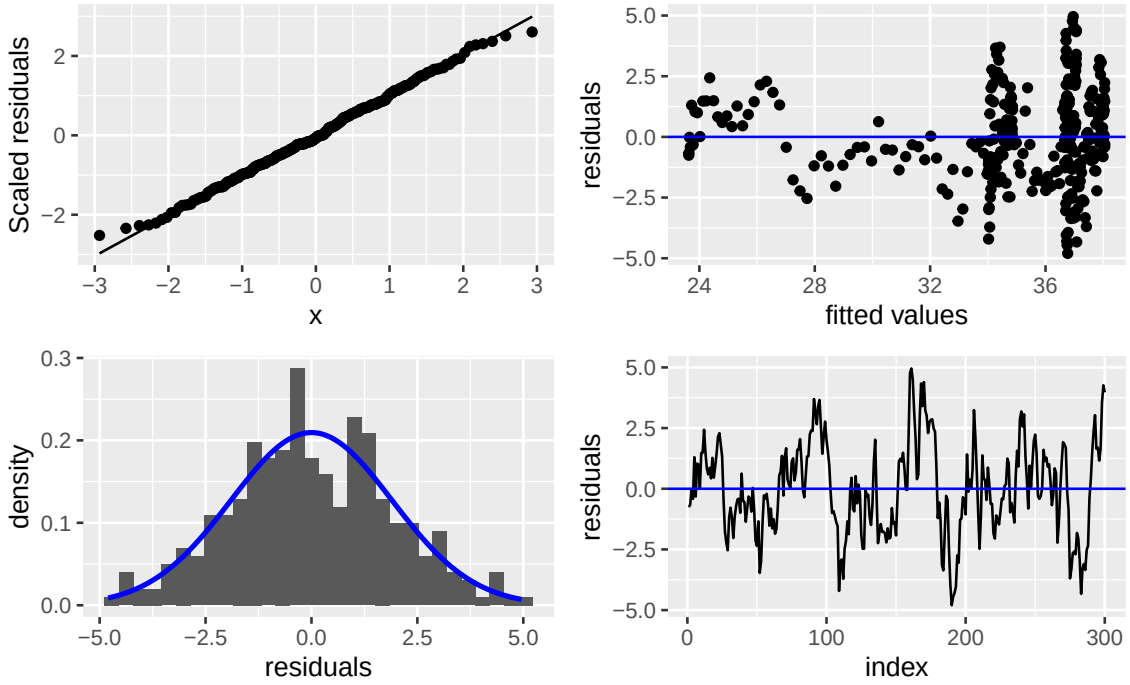
```
> n_obs<-300
> x<-1:n_obs
> set.seed(4325)
> y<-arima.sim(model = list(ar=0.999),n = n_obs,sd=1)
> plot(x,y,t="p")
>
> # skapar baser:
> cos_list<-cosine_basis(x = x,order = 6)
> # anpassar en modell
> df<-data.frame(y=y,cos_list$basis_mat)
```

```
> A<-lm(y~.,data = df)
> summary(A)


Call:
lm(formula = y ~ ., data = df)

Residuals:
   Min     1Q Median     3Q    Max
-4.800 -1.288 -0.129  1.322  4.958

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   34.710      0.111  312.54  < 2e-16 ***
basis1        -4.135      0.157  -26.37  < 2e-16 ***
basis2        -2.106      0.157  -13.43  < 2e-16 ***
basis3        -1.323      0.157   -8.44  1.5e-15 ***
basis4        -1.368      0.157   -8.72  < 2e-16 ***
basis5        -1.081      0.157   -6.90  3.3e-11 ***
basis6        -1.064      0.157   -6.79  6.4e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.92 on 293 degrees of freedom
Multiple R-squared:  0.794,Adjusted R-squared:  0.789
F-statistic:  188 on 6 and 293 DF,  p-value: <2e-16

> y_hat<-fitted(A)
> plot(x,y,t="p")
> lines(x,y_hat,col="red",lwd=3)
```

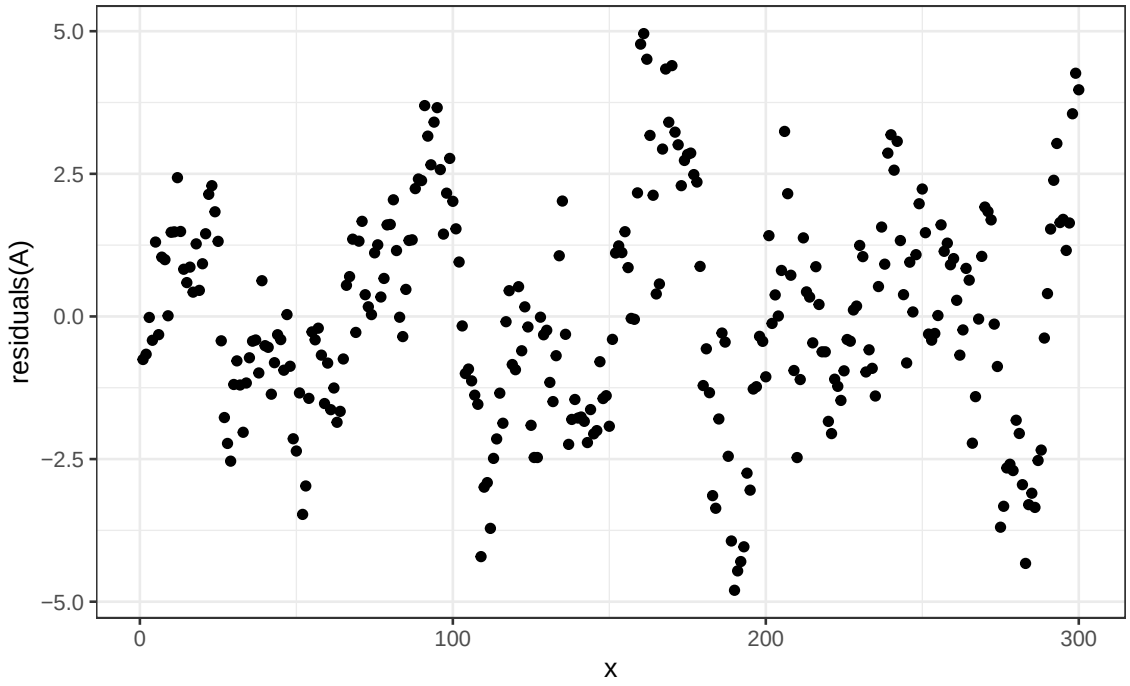


3. Vi ser att vi har en bättre anpassning en en vanlig linjär modell med bara x som kovariat. Dock så kan man önska en bättre anpassning. Vi gör residualanalys:

```
> suppressWarnings({
+ lm_diagnostics(A)
+ })

`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
> # förklarande variabeln mot residualer
> qplot(x,residuals(A))+theme_bw()
```



4. Vi ser att det finns bereoende kvar mellan x och y. Tex ser vi tydliga mönster i fitted values mot residuals.

5. Anpassa nu en linjär regression med cosinusbaser med `order = 10` och `order = 30` till data ovan. Hur blir anpassningen?

6. Skapa sedan en modell där ni modeller y med hjälp av cosinusbaser, och där ni väljer basfunktioner med Stepwise forward selection med BIC som kriterie. Ha `order = 100` som den fulla modellen. Hur blir anpassningen? Hur många basfunktioner väljs?

7. Upprepa uppgiften ovan men använd: ridge, lasso och elasticnet med $\alpha = 0.5$ för att välja modell. Använd korsvalidering för att välja λ.

2.5 Sammanfattning

1. När vi ska modellera icke-linjära samband mellan x och y, så är det ofta bra att skapa transformationer av x, olika alternativ:

   (a) Ofta vill vi använda en metod för att "automatiskt" transformera x, exempel är splines eller lokal regression.

   (b) Vi kan välja ett antal manuella transformationer av x som passar vårt problem, tex $log\,(x)$, $sin\,(\pi \cdot x)$, x^2

      i. Testa att plotta den nya transformerade variabeln mot y. Om du ser ett linjärt mönster i plotten så är troligen transformationen lämplig. Detta funkar ofta bra om vi har "ett tydligt mönster" mellan x och y. Se tex uppgift 7 i sektion 2.1, när vi har ett kvadratiskt samband. Men är sambandet mellan x och y mer komplicerat så kan så är det ofta svårt att hitta *en* transformation som gör att x passar y. Men det kan fortfarande vara så att flera transformer av x tillsammans kan modellera y på ett bra sätt.

   (c) Använda oss av en eller flera stegfunktioner. Även interaktioner med dessa och tex x kan vara användbara.

   (d) Polyom av x

   (e) Trunkerade polynom av x (en variant av enkla splines)

   (f) Cosinusbaser av x

   (g) Någon kombination av ovanstående transformationer

2. När vi använder oss av många basfunktioner, som också kan vara komplicerade, då är det bra att tänka på:

   (a) Ofta är vi inte intresserade för inferens för enskilda basfunktioner (ibland kan vi vara det), utan ofta fokuserar vi mer på vilken *funktion* som basfunktionerna tillsammas skapar. Vi fokuserar på hur bra de anpassade värden blir för modellen. Vi kan testa om en grupp basfunktioner behövs genom att göra lämpliga partiella f-test.

   (b) Det finns ju fler och ju mer avancerade basfunktioner som vi använder, desto större är risken för *överanpassning*. Så ofta vill vi dela in data i en träningsdel och en valideringsdel[5], och sen beräkna MSE för båda dessa delar. Då kan vi jämföra olika mängder av basfunktioner, och se vilken som har lägst MSE på valideringsdata.

   (c) Det är bra att plotta anpassade värden för att se hur bra de modellerar y. Det är viktigt att göra residualanalys för att se om vi har lyckats modellera relationen mellan x och y på ett bra sätt. Det är bra att göra residualanalys på både träningsdelen och valideringsdelen.

   (d) Vi kan använda olika metoder för automatiskt modellval.

      i. Ofta är krympningsmetoder att föredra (ridge, LASSO etc)

      ii. Andra alternativ: framåtvalsmetoden eller Stepwise forward selection. Har vi många basfunktioner och/eller många observationer så rekomenderas paketet `bigstep`, se här.

[5] Notera: Vi skattar alltid modellen på träningsdata. Vi använder valideringsdata för utvärdering.