APA-L1

September 14, 2018

1 APA Laboratori 1 (TEMA 1) - Ridge and standard regression

In [1]:
```
options(repr.plot.width=6, repr.plot.height=6)
```

Començarem fent bàsicament l'exemple introductori del TEMA 1 (ajust polinòmic)

In [2]:
```
# per igualar els resultats de tothom (esperem!)
set.seed (7)
```

Variables de control

In [3]:
```
N <- 20
a <- 0
b <- 1
sigma.quadrat <- 0.3^2
```

Noteu que si poseu una expressió entre parèntesis, R l'avalúa i mostra el resultat (per això poso tants parèntesis). Proveu:

In [4]:
```
pep <- 3

(pep <- 3)
```

3

Generació de la mostra TR (de training) de mida N, inputs x_n uniformes en (0,1)
El sort() és per claredat, no té cap importància
Noteu que a classe era N=10 per pura simplicitat del dibuix a la pissarra!

In [5]:
```
x <- sort(runif(N, a,b))
t <- sin(2*pi*x) + rnorm(N, mean=0, sd=sqrt(sigma.quadrat))
(sample <- data.frame(input=x,target=t))
```

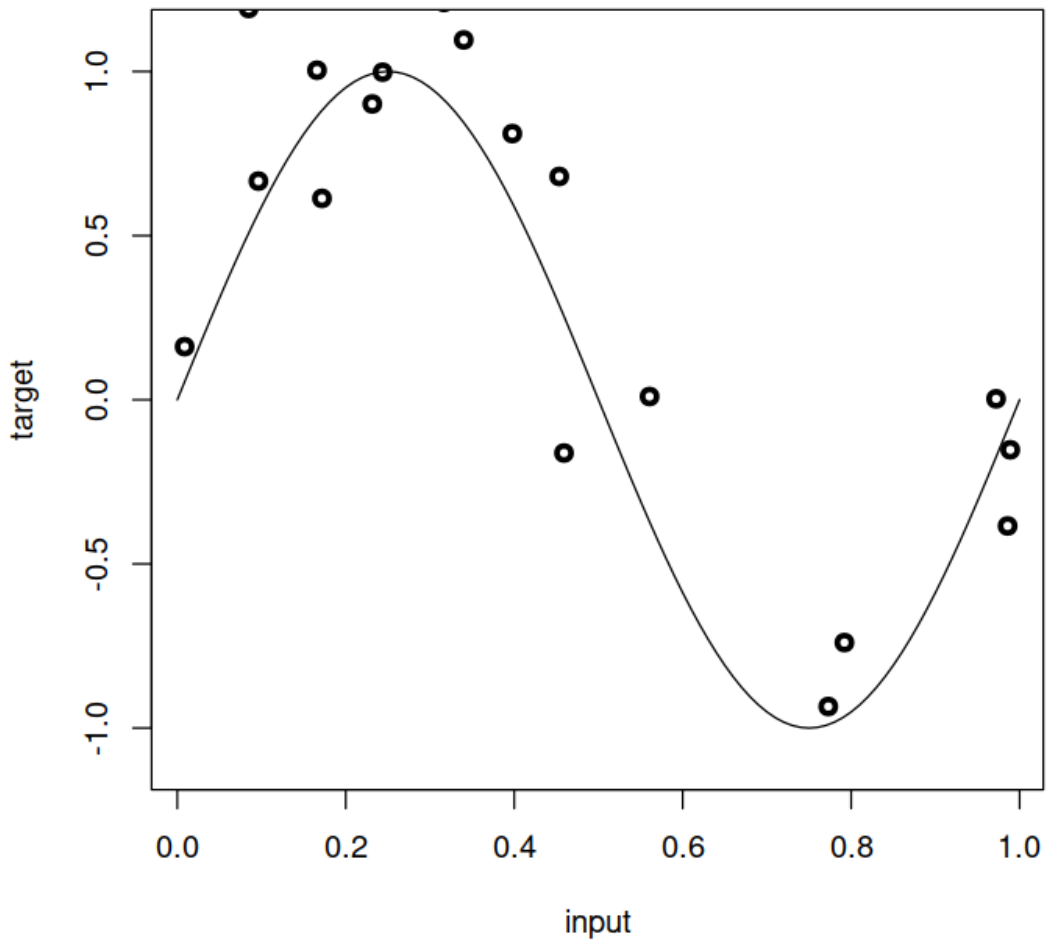
input	target
0.00870460	0.161761221
0.06974868	1.239375485
0.08470071	1.191857490
0.09630154	0.666034392
0.11569778	1.233414311
0.16585548	1.003769916
0.17174808	0.613404837
0.23147710	0.901036673
0.24374939	0.997782159
0.31658480	1.210203790
0.34006235	1.096043045
0.39774545	0.810788765
0.45344777	0.680132800
0.45910367	-0.162258051
0.56066587	0.009863772
0.77281195	-0.934487757
0.79201043	-0.739680581
0.97206250	0.002887112
0.98573709	-0.384412376
0.98890930	-0.152447739

```
In [6]: dim(sample)
        attach(sample)
```

1. 20 2. 2

Dibuixem les dades i la funció "target" (la part regular, que voldríem detectar)

```
In [7]: plot(input,target, lwd=3, ylim = c(-1.1, 1.1))
        curve (sin(2*pi*x), a, b, add=TRUE, ylim = c(-1.1, 1.1))
```



Generació de la mostra de validacio (VA) de mida N.valid, inputs x equiespaiats en (a,b) la usarem per fer prediccions

```
In [8]: N.valid <- 1000

        x.valid <- sort(runif(N.valid, a,b))
        t.valid <- sin(2*pi*x.valid) + rnorm(N.valid, mean=0, sd=sqrt(sigma.quadrat))
        valid.sample <- data.frame(input=x.valid,target=t.valid)
```

Regressió lineal amb grau M=1, per començar, model lineal

```
In [9]: model <- glm (target ~ input, data = sample, family = gaussian)

        model # ens diu que el model és y(x) = -1.606ůx + 1.133
```

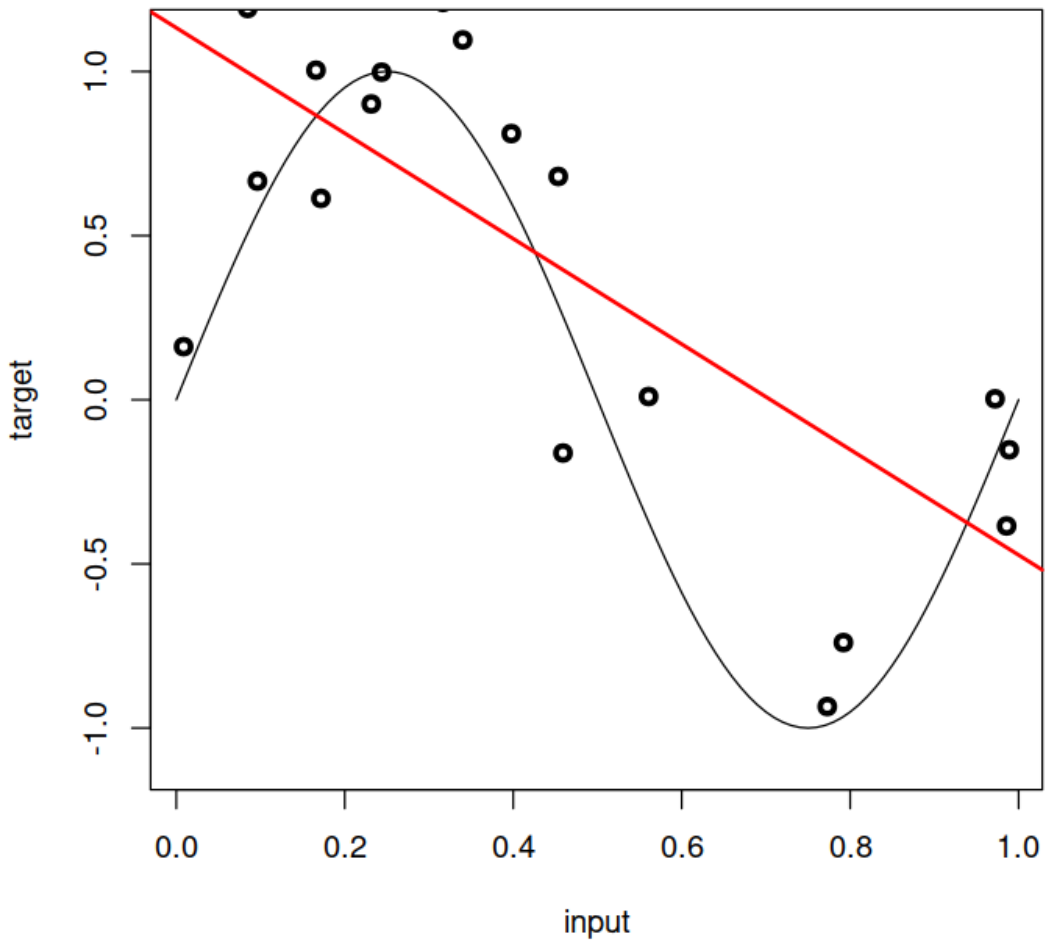
```
Call:  glm(formula = target ~ input, family = gaussian, data = sample)

Coefficients:
(Intercept)        input
      1.133       -1.606

Degrees of Freedom: 19 Total (i.e. Null);  18 Residual
Null Deviance:     9.082
Residual Deviance: 3.81  AIC: 29.59
```

fem-lo predir les dades TR (les usades per trobar el model) i calculem l'error quadràtic mitjà

```
In [10]: (prediccio <- predict(model, data=sample))
```

1 1.11889670138465 **2** 1.02086222660092 **3** 0.99684982685251 **4** 0.978219335997735 **5** 0.947069715027615 **6** 0.866518342297673 **7** 0.857055062684359 **8** 0.761132514392282 **9** 0.74142368415277 **10** 0.624452773120077 **11** 0.586748720490253 **12** 0.494111849399919 **13** 0.404656041710228 **14** 0.395572887798918 **15** 0.232467845373552 **16** -0.108230709123742 **17** -0.1390627397969 **18** -0.42821954159293 **19** -0.45018041443979 **20** -0.455274858795909

```
In [11]: plot(input,target, lwd=3, ylim = c(-1.1, 1.1))
         curve (sin(2*pi*x), a, b, add=TRUE, ylim = c(-1.1, 1.1))
         abline(model,col="red", lwd=2)
```




```
In [12]: (mean.square.error <- sum((target - prediccio)^2)/N)
```

0.190477152391008

alternativament, glm() ens el calcula

```
In [13]: (mean.square.error <- model$deviance/N)
```

0.190477152391008

si mireu el help de glm() fent ?glm --que és una rutina molt potent i general, i serveix per molts tipus de regressions-- veureu que diu:

"deviance": up to a constant, minus twice the maximized log-likelihood

això ho hem vist a classe, en el cas de regressió lineal (el cas que ens ocupa), hem obtingut la suma dels error quadràtics (la deviance, en argot estadístic), per minimització del "minus log-likelihood".

si us surt, la frase "Where sensible, the constant is chosen so that a saturated model has deviance zero" vol dir simplement que un model interpolador (que prediu exactament els targets) tindrà deviance zero.

com a curiositat, 'null.deviance' és l'error d'un model "null" (que només te el terme independent), i correspón al model M=0; l'AIC el veurem més endavant (o no)

```
In [14]: ?glm
```

```
In [15]: model$null.deviance/N
```

0.454107413181705

Dit tot això, reprenem el fil i tornem a l'error quadràtic mitjà. Resulta que és convenient treballar amb la seva arrel, per obtenir així la "llargada" del vector d'errors

```
In [16]: (root.mse <- sqrt(model$deviance/N))
```

0.436436882482459

però és encara millor normalitzar l'error, dividint per la variança dels targets, obtenint el que s'anomena NMSE (normalized mean squared error)

```
In [17]: (NMSE <- model$deviance/((N-1)*var(target)))
```

0.419453959265781

Interpretació:

1. Òbviament NMSE >= 0, però no té cota superior definida
2. un model constant que predigui la mitjana dels targets (de fet, el millor model constant), tindria un NMSE = 1.
3. models amb NMSE > 1 són per tant horribles; un model es comença a considerar acceptable a partir de NMSE < 0.2
4. observeu que, notant que l'error quadràtic no és més que l'estimació de la variança dels targets, el NMSE es pot veure com la fracció de la variança dels targets no explicada (capturada) per les prediccions del model. Per exemple, un NMSE = 0.13 correspon a un model capaç d'explicar el 87% de la variabilitat del target.

El nostre model M=1 explica per tant només el 58% de la variabilitat del target.

Regressió cúbica (polinomi de grau M=3, continua sent un model lineal, penseu-lo)

```
In [18]: model <- glm (target ~ poly(input, 3, raw=TRUE), data = sample, family = gaussian)
```

veiem-ne el model: summary() dóna més informació

```
In [19]: summary(model)
```

```
Call:
glm(formula = target ~ poly(input, 3, raw = TRUE), family = gaussian,
    data = sample)

Deviance Residuals:
```

```
     Min        1Q    Median        3Q       Max
-0.54238  -0.21388   0.02159   0.24464   0.43526

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)                   0.3265     0.2234   1.461 0.163247
poly(input, 3, raw = TRUE)1   8.6171     2.1342   4.038 0.000954 ***
poly(input, 3, raw = TRUE)2 -26.5899     5.2419  -5.073 0.000113 ***
poly(input, 3, raw = TRUE)3  17.5885     3.4666   5.074 0.000113 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

(Dispersion parameter for gaussian family taken to be 0.09093278)

    Null deviance: 9.0821  on 19  degrees of freedom
Residual deviance: 1.4549  on 16  degrees of freedom
AIC: 14.342

Number of Fisher Scoring iterations: 2
```

Deviance Residuals són les diferències al quadrat punt a punt. El que se'ns mostra són els estadístics (min, median, etc) de les 20 que tenim. els coeficients del polinomi (del model) són:

```
In [20]: model$coefficients
```

(Intercept) 0.326476789186326 **poly(input, 3, raw = TRUE)1** 8.61710098347513 **poly(input, 3, raw = TRUE)2** -26.5898536910208 **poly(input, 3, raw = TRUE)3** 17.5885350039958

o sigui el polinomi és y(x) = 17.5885350ůx^3 -26.5898537ůx^2 -8.6171010ůx + 0.3264768

Std. Error és la incertesa de cada coeficient, relativa al valor del coeficient; en aquest cas són molt altes, cosa que és deguda a que només tenim 20 punts

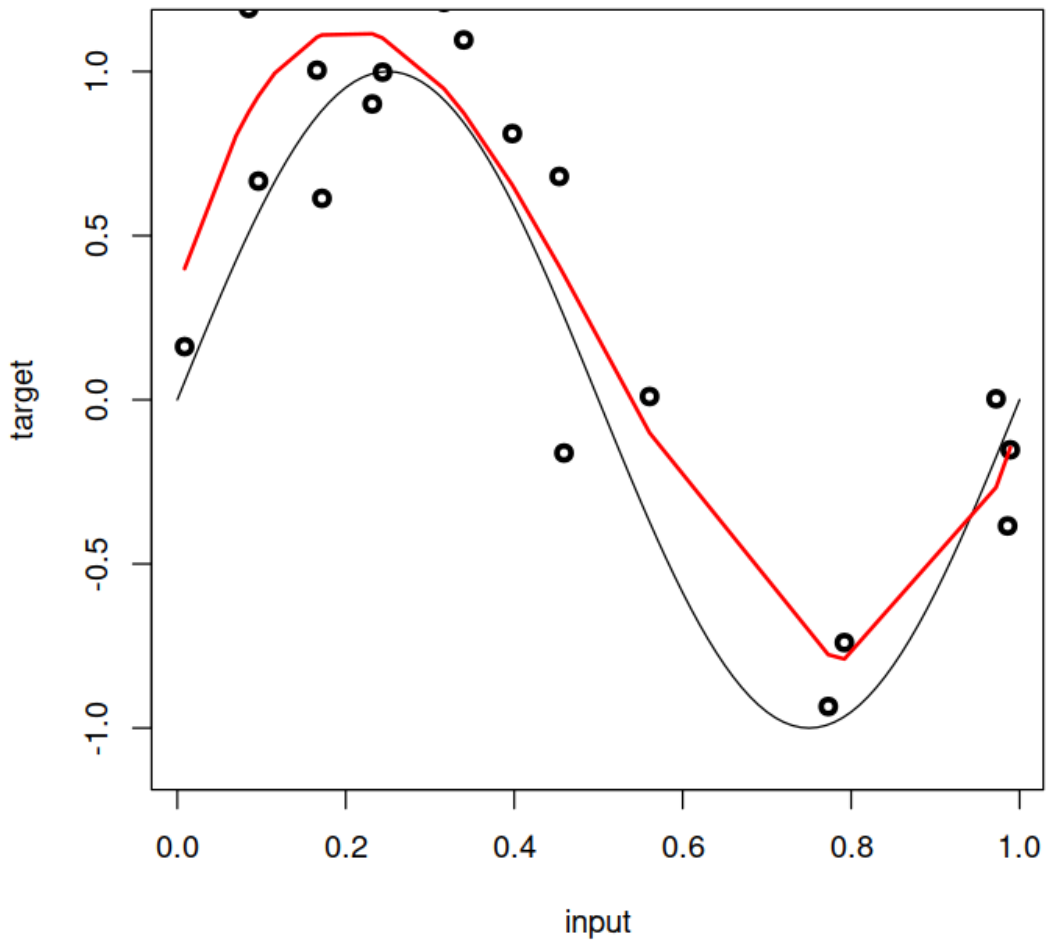
La darrera columna és un test sobre la probabilitat de que cada coeficient sigui en realitat zero (i per tant, l'entrada x corresponent no té relació amb el target). Per interpretar-ho, mireu el codi inferior:

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Entrades x amb '***' són imprescindibles, després '**', etc. Tot això, suposant, és clar, que el model és correcte (que no és el cas, com sabem, perquè l'hem dissenyat nosaltres)

Ara ho dibuixem tot de nou

```
In [21]: # dades de TR
         plot(input,target, lwd=3, ylim = c(-1.1, 1.1))
         # part regular a modelar
         curve (sin(2*pi*x), a, b, add=TRUE, ylim = c(-1.1, 1.1))
         # el model obtingut
         points(input, predict(model), type="l", col="red", lwd=2)
```



calculem l'error normalitzat NRMSE a la mostra de TR

```
In [22]: (NMSE <- model$deviance/((N-1)*var(target)))
```

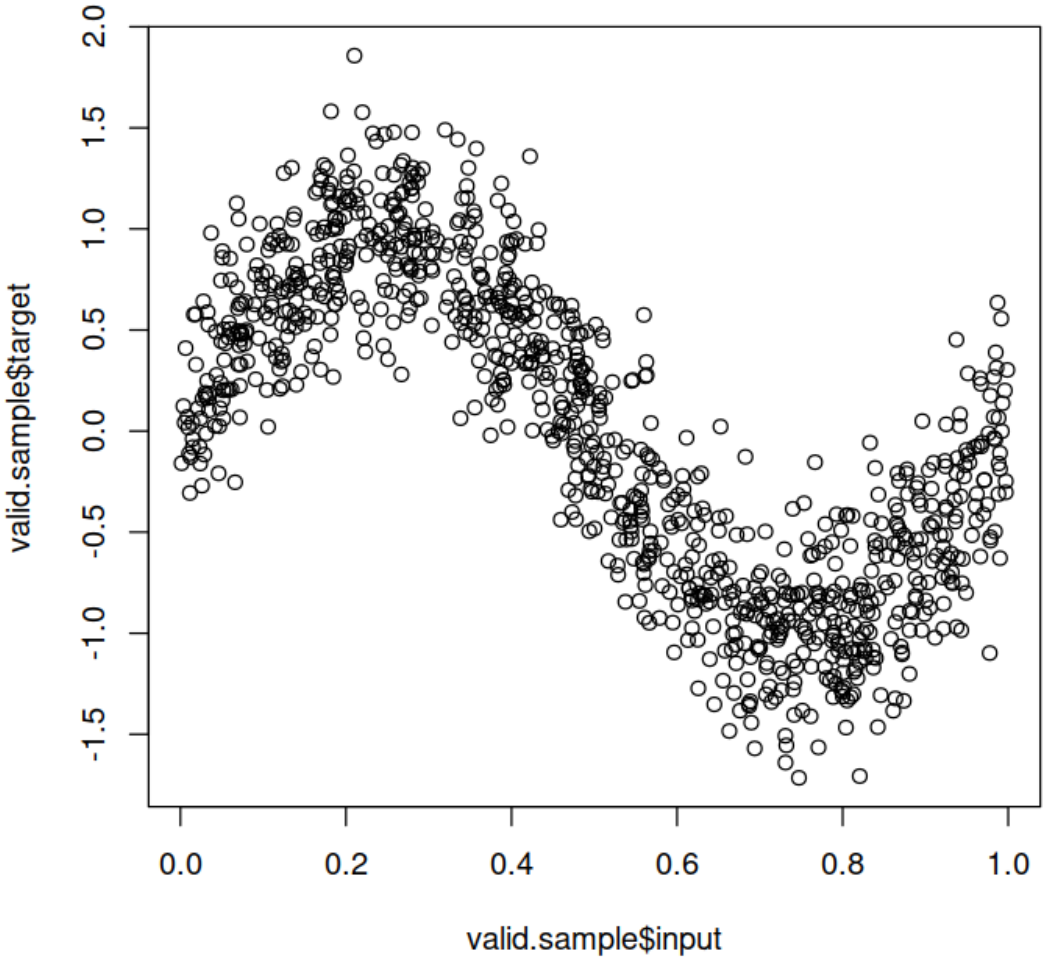
0.160196075319881

gairebé 84%, ha millorat força ... per cert, aquesta quantitat és el famós coeficient R^2 d'ajust en regressió (expressat com a 0.84)

Ara veurem com calcular l'error normalitzat a la mostra de validacio hem de fer-ho explícitament, ja que predict() només dóna les prediccions, no els errors

fem un cop d'ull primer a les dades de VA

```
In [23]: plot(valid.sample$input, valid.sample$target)
```



i calculem l'error

```
In [24]: prediccions <- predict (model, newdata=valid.sample)
         (NMSE.valid <- sum((valid.sample$target - prediccions)^2)/((N.valid-1)*var(valid.sample
```

0.233558782843144

Quan un model és incorrecte, el seu error de predicció és alt. En cas de models sobreajustats, sol ser molt superior al de TR. En cas de models infraajustats (com és el cas ara), tots dos són elevats i similars. Aquest és un model raonable i per tant tos dos errors són baixos i similars.

Ara farem la mateixa simulació que a classe, fent regressió polinòmica, de graus M des de p a q (poden ser arbitraris, els fixarem en 1 i N-1)

desem coeficients del polinomi (del model) i els error de training i validacio

```
In [25]: p <- 1
         q <- N-1
```

```
coef <- list()
model <- list()
norm.mse.train <- NULL
norm.mse.valid <- NULL

for (i in p:q)
{
  model[[i]] <- glm(target ~ poly(input, i, raw=TRUE), data = sample, family = gaussian

  coef[[i]] <- model[[i]]$coefficients
  norm.mse.train[i] <- model[[i]]$deviance/((N-1)*var(target))

  prediccions <- predict (model[[i]], newdata=valid.sample)
  norm.mse.valid[i] <- sum((valid.sample$target - prediccions)^2)/((N.valid-1)*var(vali
}
```

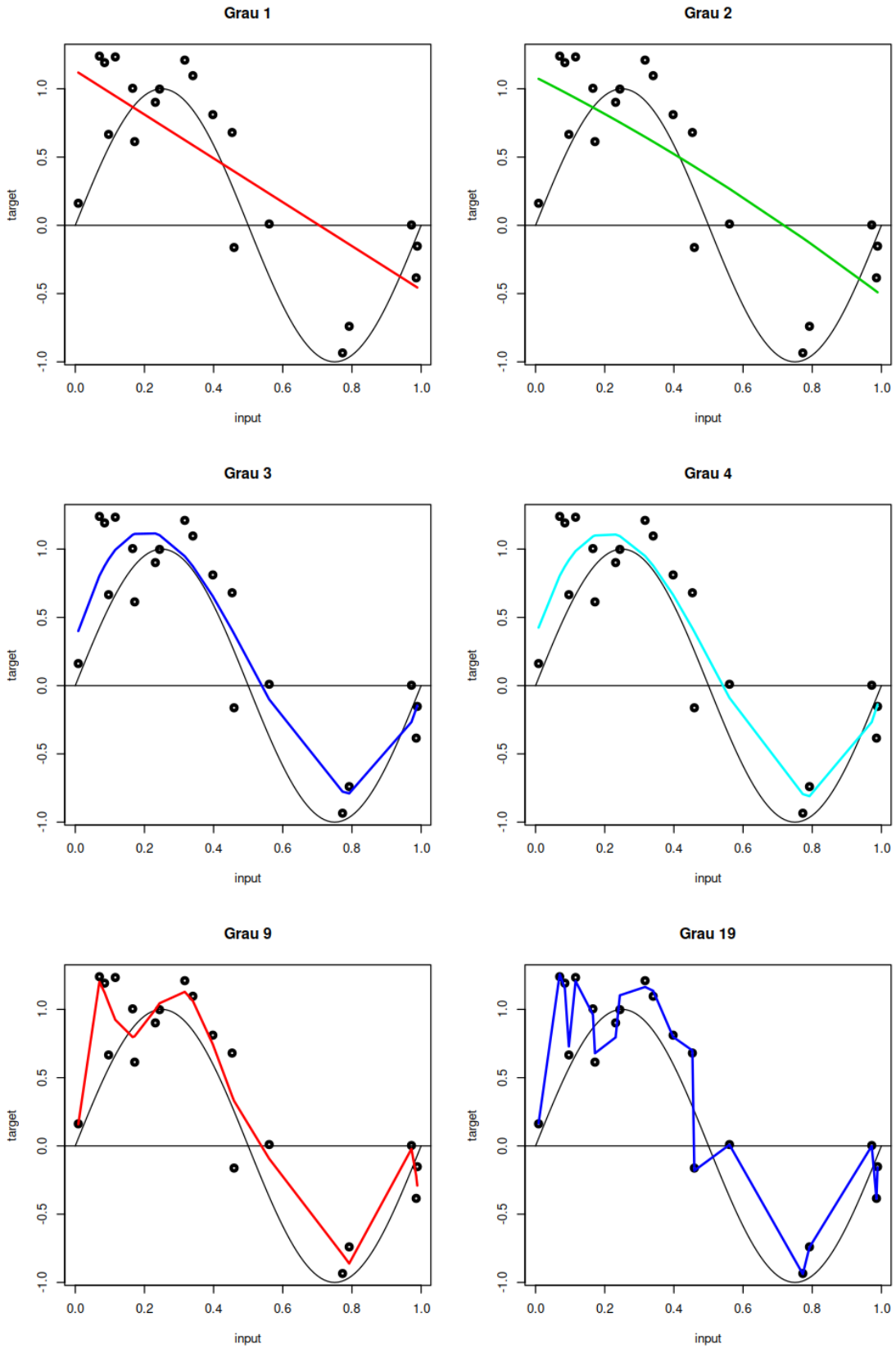
```
Warning message in predict.lm(object, newdata, se.fit, scale = 1, type = ifelse(type == :
prediction from a rank-deficient fit may be misleadingWarning message in predict.lm(object, newd
prediction from a rank-deficient fit may be misleadingWarning message in predict.lm(object, newd
prediction from a rank-deficient fit may be misleadingWarning message in predict.lm(object, newd
prediction from a rank-deficient fit may be misleading
```

ho arrepleguem tot en una matriu

```
In [26]: resultats <- cbind (Grau=p:q, Coeficients=coef, Error.train=norm.mse.train, Error.valid
```

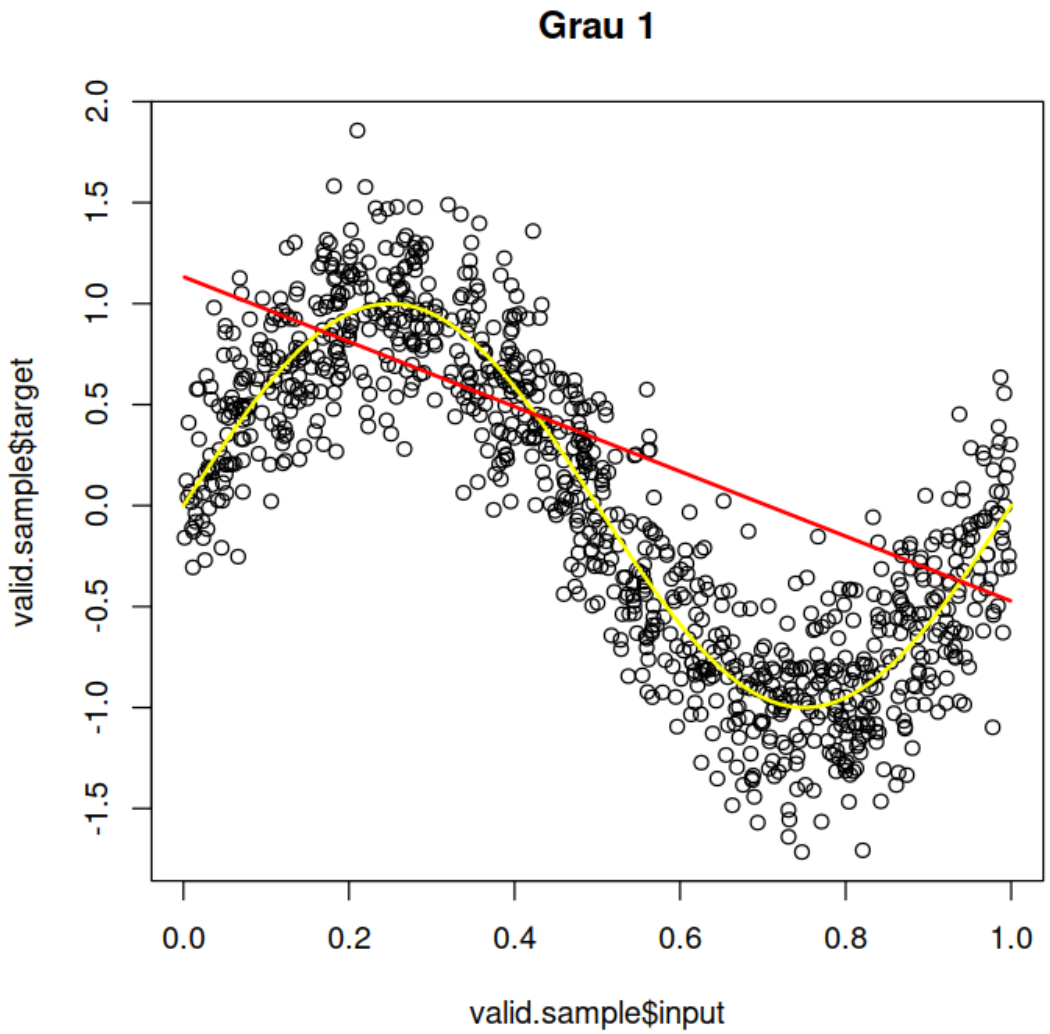
dibuixem 6 dels models (graus 1,2,3,4,9,19) contra les dades de training

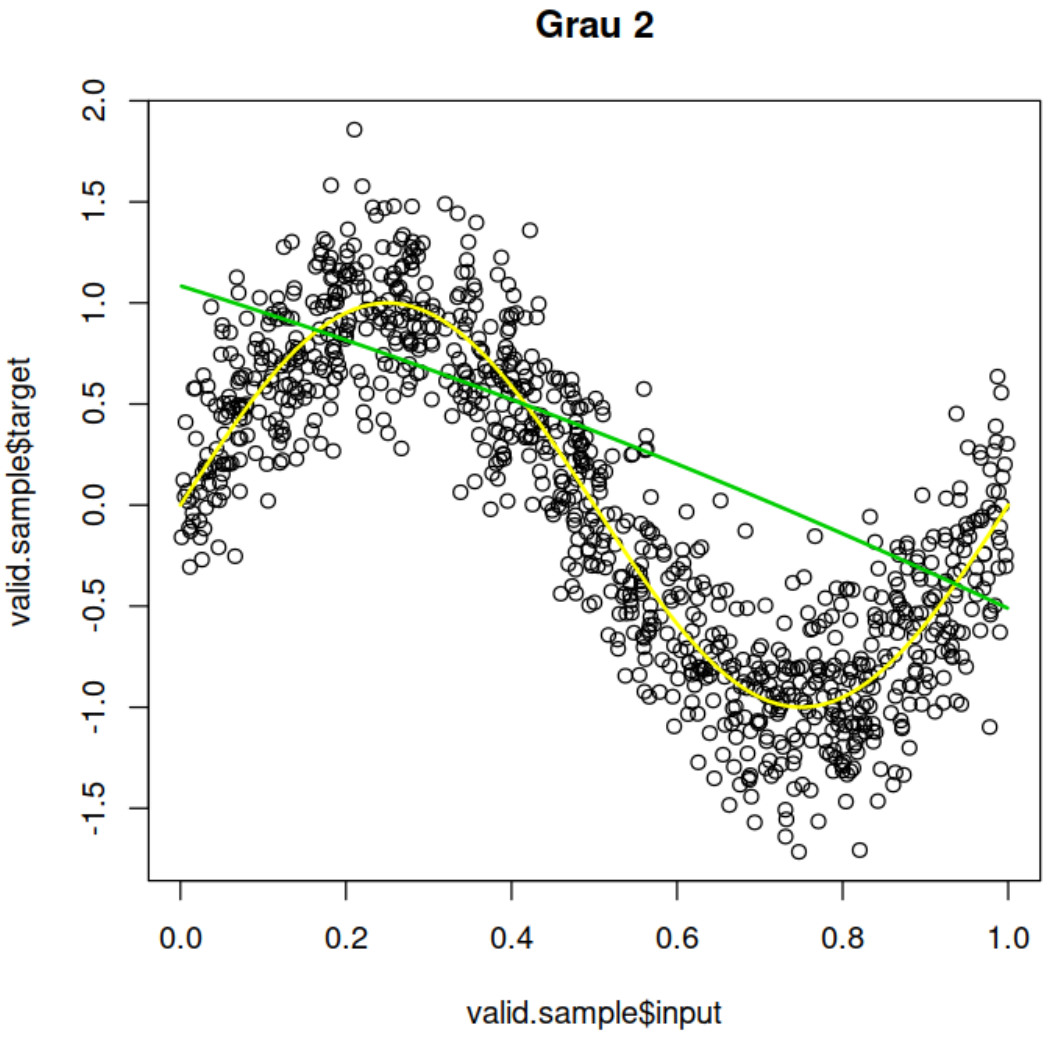
```
In [27]: options(repr.plot.width=8, repr.plot.height=12)
         # això crea una graella (grid) de 3x2
         par(mfrow=c(3, 2))

         graus <- c(1,2,3,4,9,19)

         for (i in graus)
         {
           plot(input,target, lwd=3)
           curve (sin(2*pi*x), a, b, add=TRUE)
           abline(0,0)
           points(input, predict(model[[i]]), type="l", col=25+i, lwd=2)
           title (main=paste('Grau',i))
         }
```

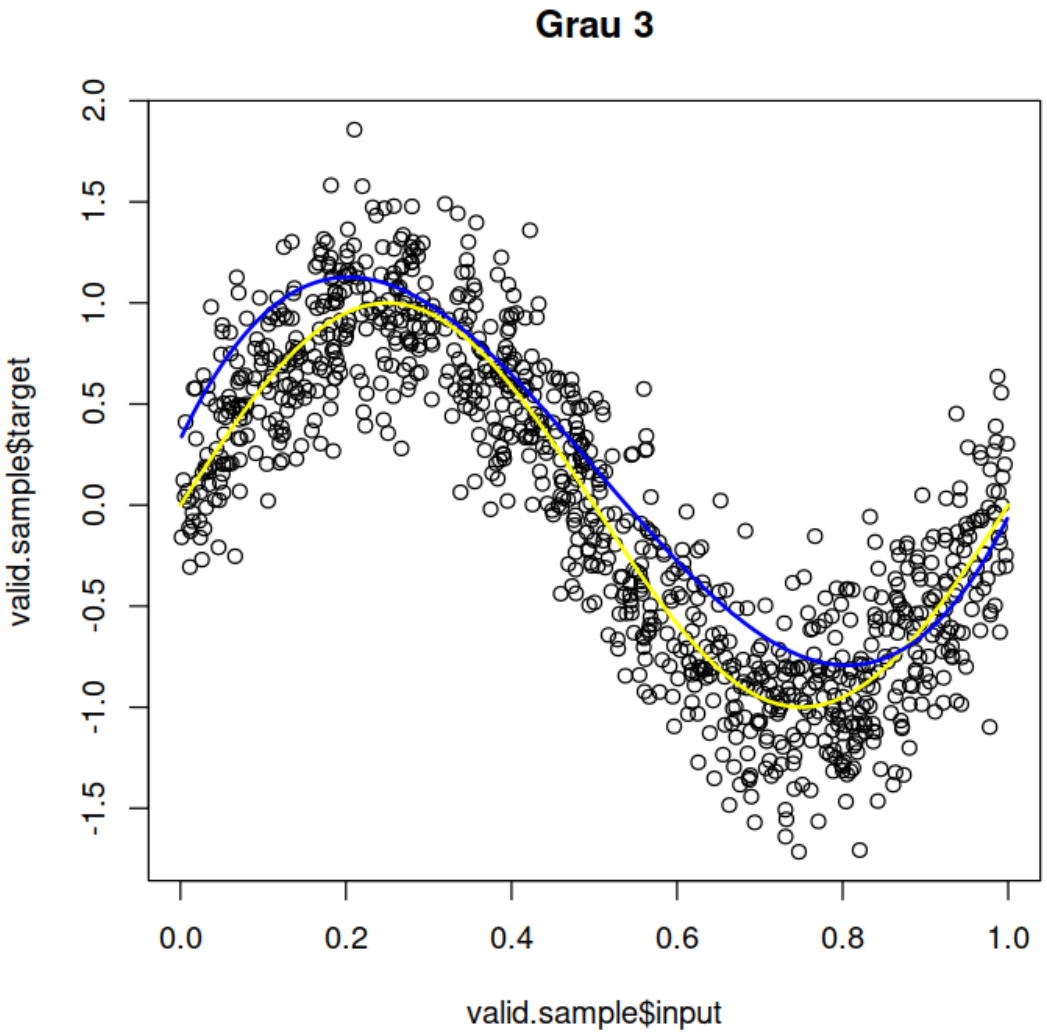

Grau 1
Grau 2
Grau 3
Grau 4
Grau 9
Grau 19
target
input
-1.0
-0.5
0.0
0.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0

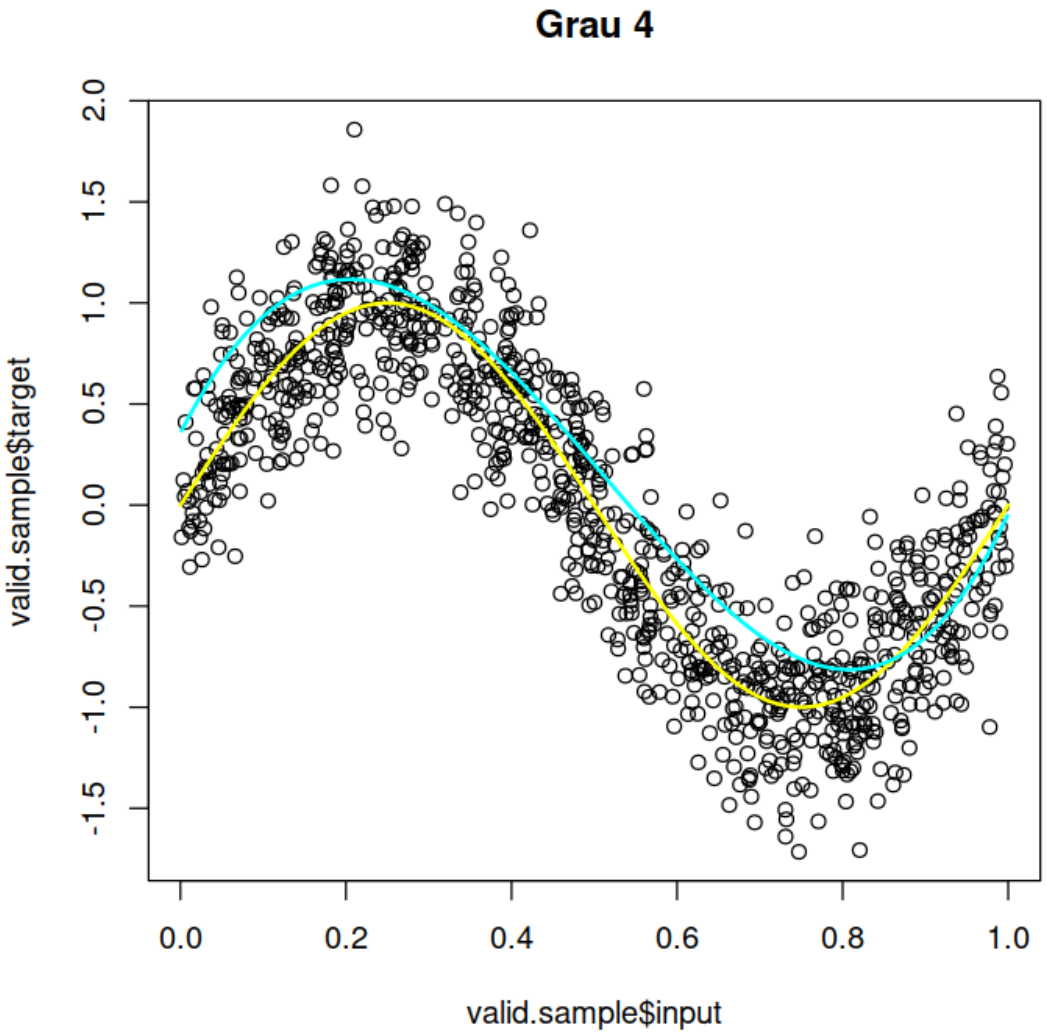
Ara dibuixem les prediccions dels mateixos models contra les dades de validacio la funció a modelar està sempre en groc, per referència

```
In [28]: options(repr.plot.width=6, repr.plot.height=6)
         for (i in graus)
         {
           plot(valid.sample$input, valid.sample$target)
           curve (sin(2*pi*x), a, b, add=TRUE, col='yellow',lwd=2)
           points(valid.sample$input,
                  predict(model[[i]], newdata=valid.sample),
                  type="l", col=25+i, lwd=2)
           title (main=paste('Grau',i))
         }
```

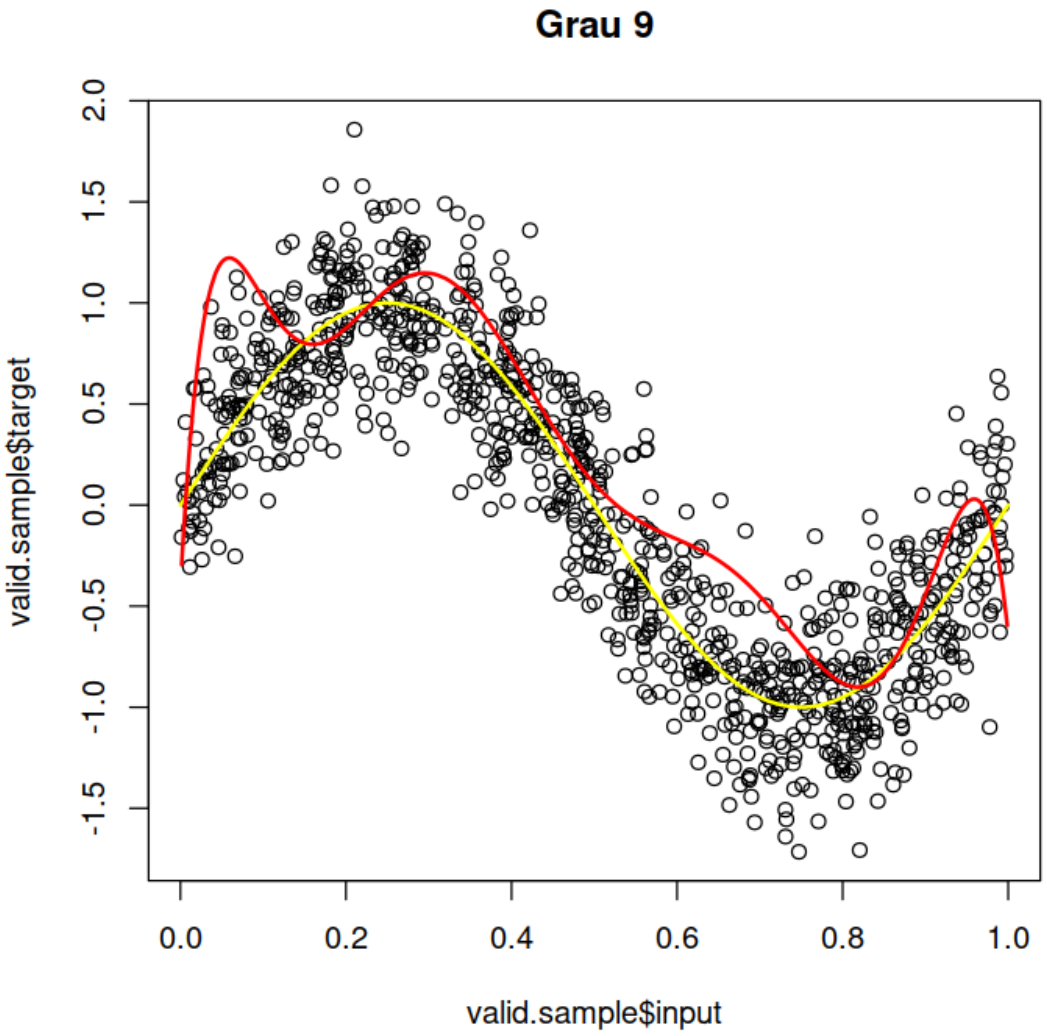


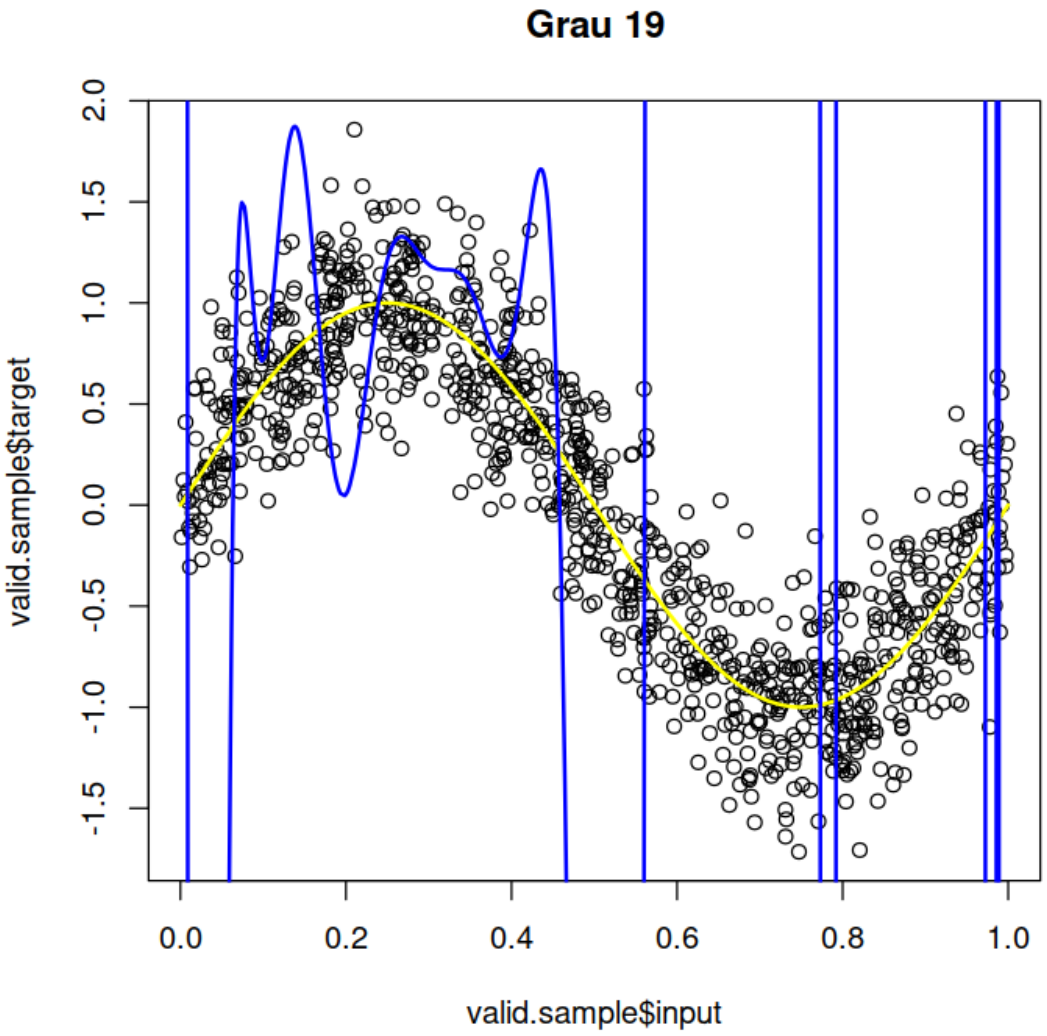

Grau 2
valid.sample$target
valid.sample$input
2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
0.0
0.2
0.4
0.6
0.8
1.0


Grau 3
valid.sample$target
valid.sample$input
2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
0.0
0.2
0.4
0.6
0.8
1.0




```
Warning message in predict.lm(object, newdata, se.fit, scale = 1, type = ifelse(type == :
prediction from a rank-deficient fit may be misleading
```


Grau 9
valid.sample$target
2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
0.0
0.2
0.4
0.6
0.8
1.0
valid.sample$input



Ara farem una gràfica similar a la de classe: error de TR i error de VA junts, en funció del grau M; observarem els fenòmens de sobreajust i infraajust molt clarament

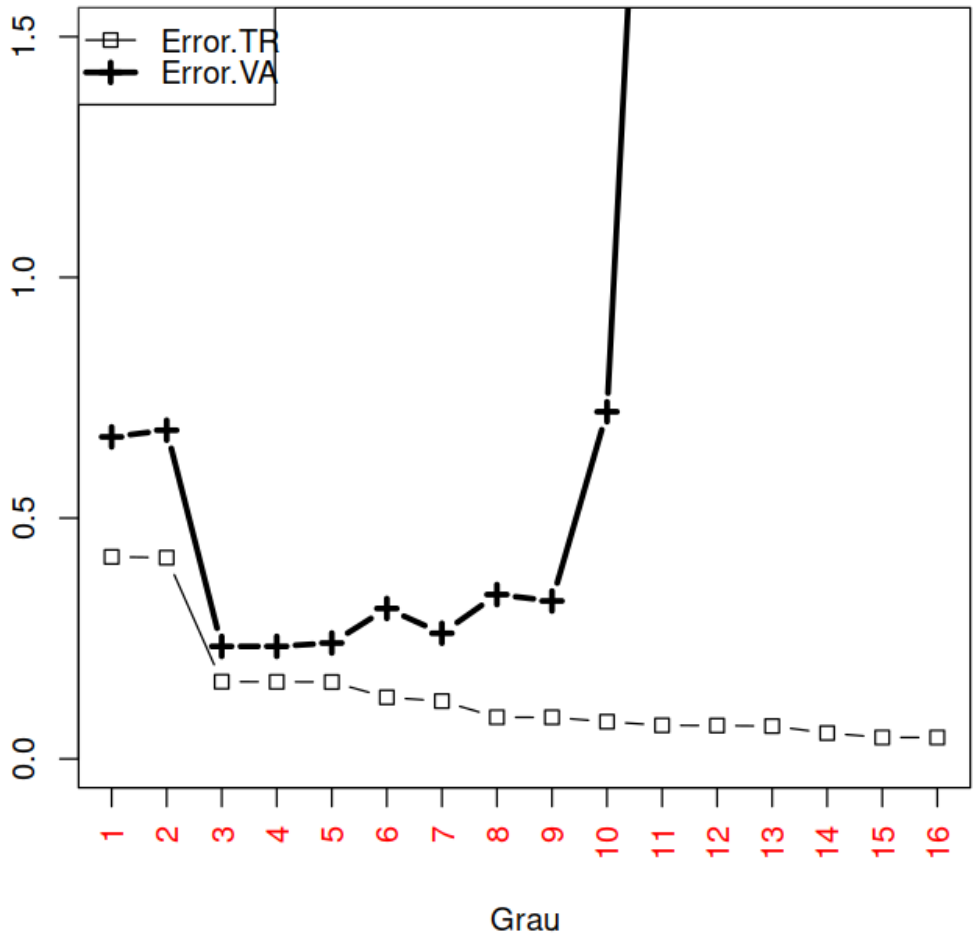
Ometem els graus a partir del 17 perquè els valors es disparen i caurien fora del dibuix; però podeu mirar els resultats numèrics a la matriu:

```
In [29]: (r <- data.frame(resultats[,-2]))

         par(mfrow=c(1,1))
```

Grau	Error.train	Error.valid
1	0.419454	0.668733
2	0.4179419	0.6825725
3	0.1601961	0.2335588
4	0.1599038	0.233627
5	0.1596058	0.2405754
6	0.1277075	0.3123137
7	0.1199549	0.2608449
8	0.0863783	0.3412057
9	0.08621885	0.3278474
10	0.07713956	0.7208899
11	0.06956677	2.922974
12	0.069142	4.265301
13	0.06808642	3.883657
14	0.0535703	108.7034
15	0.04439567	4493.7
16	0.04439567	4493.7
17	0.04046699	123036.6
18	0.04046699	123036.6
19	0.004332349	24791887

preparar un plot buit i omplir-lo

```
In [30]: plot(1:20, 1:20, xlim=c(1,16), ylim=c(0,1.5), type = "n", xlab="Grau", ylab="", xaxt="n

         axis(1, at=1:16,labels=1:16, col.axis="red", las=2)
         points (x=r$Grau[1:16], y=r$Error.train[1:16], type='b', pch=0)
         points (x=r$Grau[1:16], y=r$Error.valid[1:16], type='b', pch=3, lwd=3)

         legend(x="topleft", legend=c("Error.TR", "Error.VA"), pch=c(0, 3), lwd=c(1, 3))
```



Per un pèl, el menor error de VA és per M=3, per tant el métode vist a classe seleccionaria el model correcte (se'n diu "model selection"). Noteu que el "model selection" és bàsicament la detecció de la complexitat adequada al problema (la millor en aquest cas, --és a dir, dins els polinomis-- correspón a un de cúbic). Més endavant tractarem el tema de la "complexitat" d'un model des d'un de vista més general.

Ara investigarem els propis coeficients: veurem que tot coeficient del mateix grau va fent-se gran (en valor absolut) a mesura que puja el grau màxim (tret del coeficient de grau 0 o Intercept, que més o menys es manté constant, donat que intenta seguir la mitjana dels targets, que no varia)

Estudiem només fins al grau 10, per fer la taula manegable

```
In [31]: coefs.table <- matrix (nrow=10, ncol=9)

         for (i in 1:10)
           for (j in 1:9)
```

```
        coefs.table[i,j] <- coef[[j]][i]

    coefs.table
```

1.132876	1.0844778	0.3264768	0.3568835	0.3268113	8.553415e-02	-2.946476e-02	-3.66102
-1.605962	-1.2794397	8.6171010	7.9996913	9.0296205	2.260371e+01	3.179382e+01	6.94311
NA	-0.3165401	-26.5898537	-23.7058024	-31.9938065	-1.980077e+02	-3.424321e+02	-1.09247
NA	NA	17.5885350	13.0179027	37.4790430	8.149175e+02	1.704025e+03	7.67887
NA	NA	NA	2.2875104	-27.1108740	-1.685570e+03	-4.332365e+03	-2.81791
NA	NA	NA	NA	12.2488011	1.632314e+03	5.698220e+03	5.77370
NA	NA	NA	NA	NA	-5.869543e+02	-3.688041e+03	-6.66134
NA	NA	NA	NA	NA	NA	9.285284e+02	4.04487
NA	NA	NA	NA	NA	NA	NA	-1.00493
NA	NA	NA	NA	NA	NA	NA	NA

si cap a la finestra, movem-ne els marges: és important veure la taula d'una sola peça

Interpretació: 1. Les files són els coeficients dels diferents graus 1..M i les columnes els graus M; per exemple, [3,4] és el coeficient de grau 3 del polinomi M=4 2. tot coeficient del mateix grau va fent-se gran (en valor absolut) a mesura que puja el grau màxim; per exemple, el coeficient de grau 2 passa de 1 a 8-9, 22-32, 69, 74, etc. 3. Això implica que a mesura que creix M, *tots* els coeficients es fan molt grans. I ens suggereix que, tot i sobreestimar el grau M, podríem controlar l'ajust si poguéssim limitar aquest creixement: això porta a la tècnica de regularització (que en estadística es coneix com a "ridge regression")

Ara comprovarem que el sobreajust disminueix amb N !!!

generació de la mostra de training de mida N, inputs x equiespaiats en (a,b)

```
In [32]: # abans eren 20, la resta és exactament igual
         N.big <- 100

         x <- seq(a,b,length.out=N.big)
         t <- sin(2*pi*x) + rnorm(N.big, mean=0, sd=sqrt(sigma.quadrat))
         big.sample <- data.frame(input=x,target=t)
         attach(big.sample)
```

```
The following objects are masked from sample:

    input, target
```

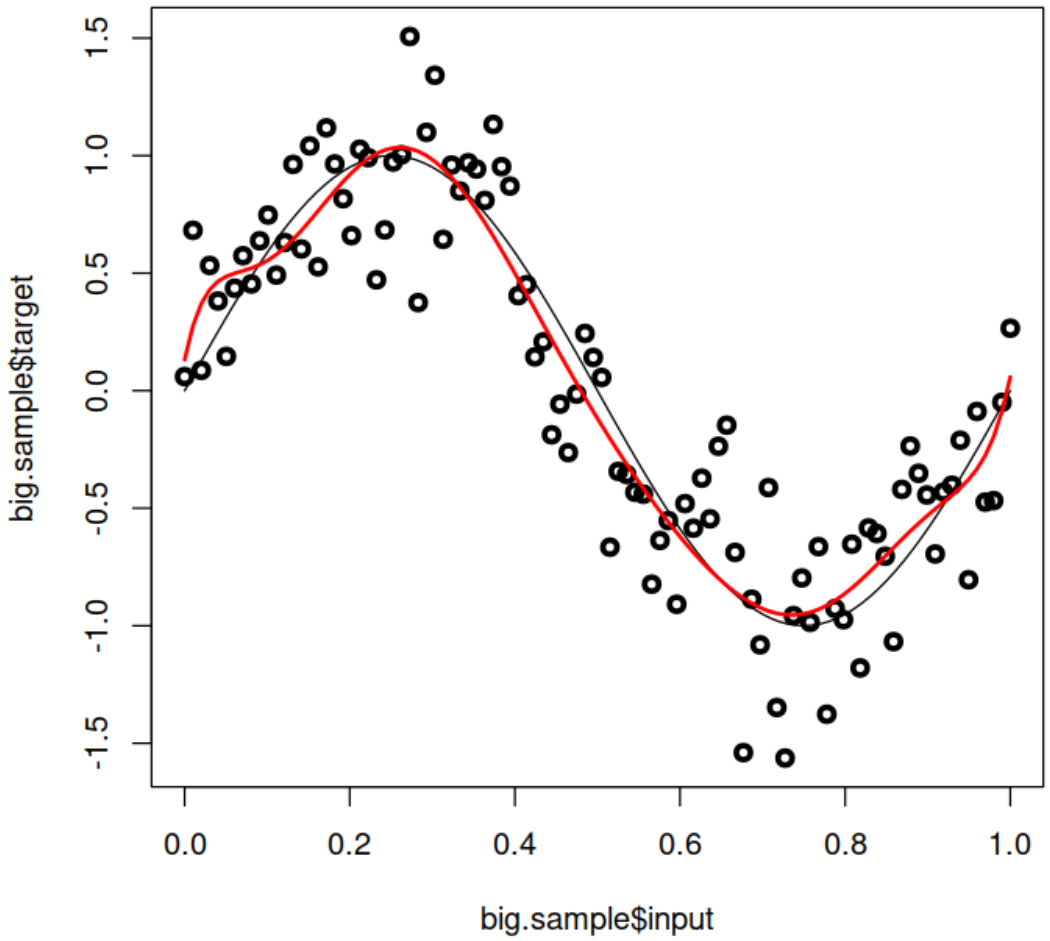
```
In [33]: model <- glm(target ~ poly(input, 9, raw=TRUE), data = big.sample, family = gaussian)

         (nmse.train <- model$deviance/((N.big-1)*var(target)))
```

0.1369170814225

```
In [34]: prediccions <- predict (model, newdata=valid.sample)
         (nmse.valid <- sum((valid.sample$target - prediccions)^2)/((N.big-1)*var(valid.sample$t
```

1.62627281255304

```
In [35]: par(mfrow=c(1, 1))
         plot(big.sample$input,big.sample$target, lwd=3)
         curve (sin(2*pi*x), a, b, add=TRUE)
         points(big.sample$input, predict(model), type="l", col="red", lwd=2)
```



Sí: estem modelant el TR amb M=9 i no sobreajusta tant (compareu-lo amb la gràfica que hem generat abans)

Conclusió: a mesura que N es fa gran el problema del sobreajust s'alleuja força. En general, però, el concepte "N gran" depèn de quantes variables x usem per modelar i, a més, no és usual tenir control sobre N; per tant, en la pràctica caldrà restringir la mida dels coeficients.

2 Ridge Regression

Ara passem a regressió ridge per re-ajustar el TR més gran (N=100)

```
In [36]: library(MASS)

         par(mfcol=c(1,1))
```

Aquesta és la idea de la regularització; partim d'una especificació de model sobradament complexe (M=12) i limitem explícitament la mida (en valor absolut) dels coeficients, via el paràmetre lambda (constant de regularització)

La gràcia ara és estimar un bon valor per lambda: veurem més endavant que es pot fer de manera molt eficient quan el model és lineal. Provem doncs dins una seqüència de valors molt llarga:

```
In [37]: lambdes <- seq(0.001,0.5,0.001)

         length(lambdes)
```

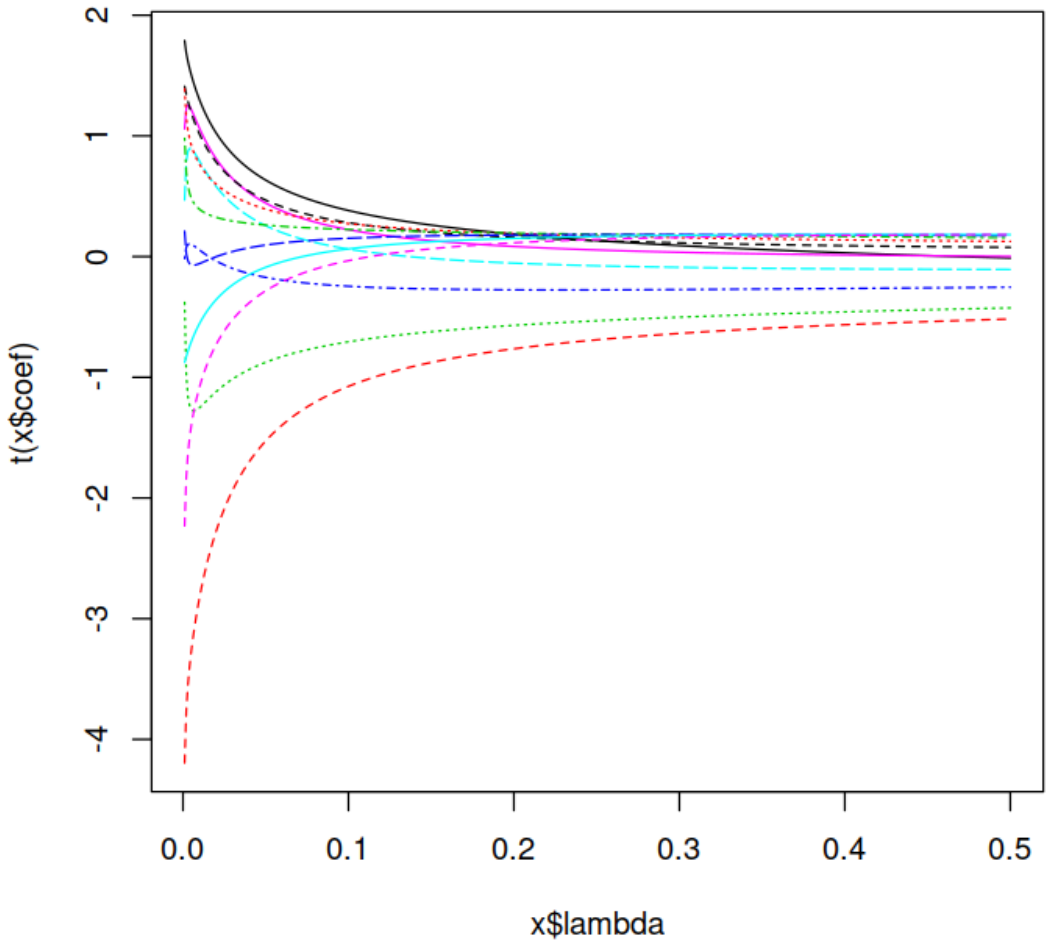
500

aquest seria el model "estàndar" (sense regularitzar)

```
In [38]: model <- glm (target ~ poly(input, 12, raw=TRUE), data = sample, family = gaussian)
```

aquest seria el model regularitzat

```
In [39]: model.ridge <- lm.ridge (model, lambda = lambdes)
         plot(model.ridge, lty=1:3)
```



El que veiem a la gràfica (que és molt xula) és com tots els coeficients (dels graus 1 al 12) se'n van a zero a mesura que regularitzem més (lambdes més altes)

Igual que la crida anterior, però ara seleccionem la millor lambda (que és la que té un error quadràtic regularitzat menor); podríem haver fet només aquesta crida directament:

```
In [40]: select( lm.ridge(model, lambda = lambdes) )

modified HKB estimator is 1.36268e-13
modified L-W estimator is 2.12222
smallest value of GCV  at 0.008
```

hi ha varis criteris per seleccionar la millor lambda de manera eficient; a classe (quan toqui) veurem el GCV

dóna lambda = 0.008, per tant ajustem un nou model (que serà el definitiu) amb aquest valor:

```
In [41]: model.final <- lm.ridge (model,lambda=0.008)
```

Compareu els dos grups de coeficients del nostre model M=12 (estàndar i regularitzats)

```
In [42]: # M=12 (estàndar)
         coef(model)
```

(Intercept) -0.59173630614867 **poly(input, 12, raw = TRUE)1** 104.718851383364 **poly(input, 12, raw = TRUE)2** -2262.49180722151 **poly(input, 12, raw = TRUE)3** 24440.4650271985 **poly(input, 12, raw = TRUE)4** -153604.747731663 **poly(input, 12, raw = TRUE)5** 582043.282582646 **poly(input, 12, raw = TRUE)6** -1256837.54695595 **poly(input, 12, raw = TRUE)7** 1081481.09836579 **poly(input, 12, raw = TRUE)8** 1407725.45111879 **poly(input, 12, raw = TRUE)9** -4952182.09549588 **poly(input, 12, raw = TRUE)10** 5752026.12615818 **poly(input, 12, raw = TRUE)11** -3189031.48719114 **poly(input, 12, raw = TRUE)12** 706099.800973201

```
In [43]: # M=12 (regularitzat)
         coef(model.final)
```

1 0.566885472687149 **poly(input, 12, raw = TRUE)1** 4.29750602828978 **poly(input, 12, raw = TRUE)2** -8.76607795467751 **poly(input, 12, raw = TRUE)3** -3.77698948337141 **poly(input, 12, raw = TRUE)4** 0.228698189835508 **poly(input, 12, raw = TRUE)5** 2.58740521491071 **poly(input, 12, raw = TRUE)6** 3.53997750126452 **poly(input, 12, raw = TRUE)7** 3.45477766739867 **poly(input, 12, raw = TRUE)8** 2.65866549498171 **poly(input, 12, raw = TRUE)9** 1.38256585048371 **poly(input, 12, raw = TRUE)10** -0.226681281609164 **poly(input, 12, raw = TRUE)11** -2.07873990499899 **poly(input, 12, raw = TRUE)12** -4.11722102863199

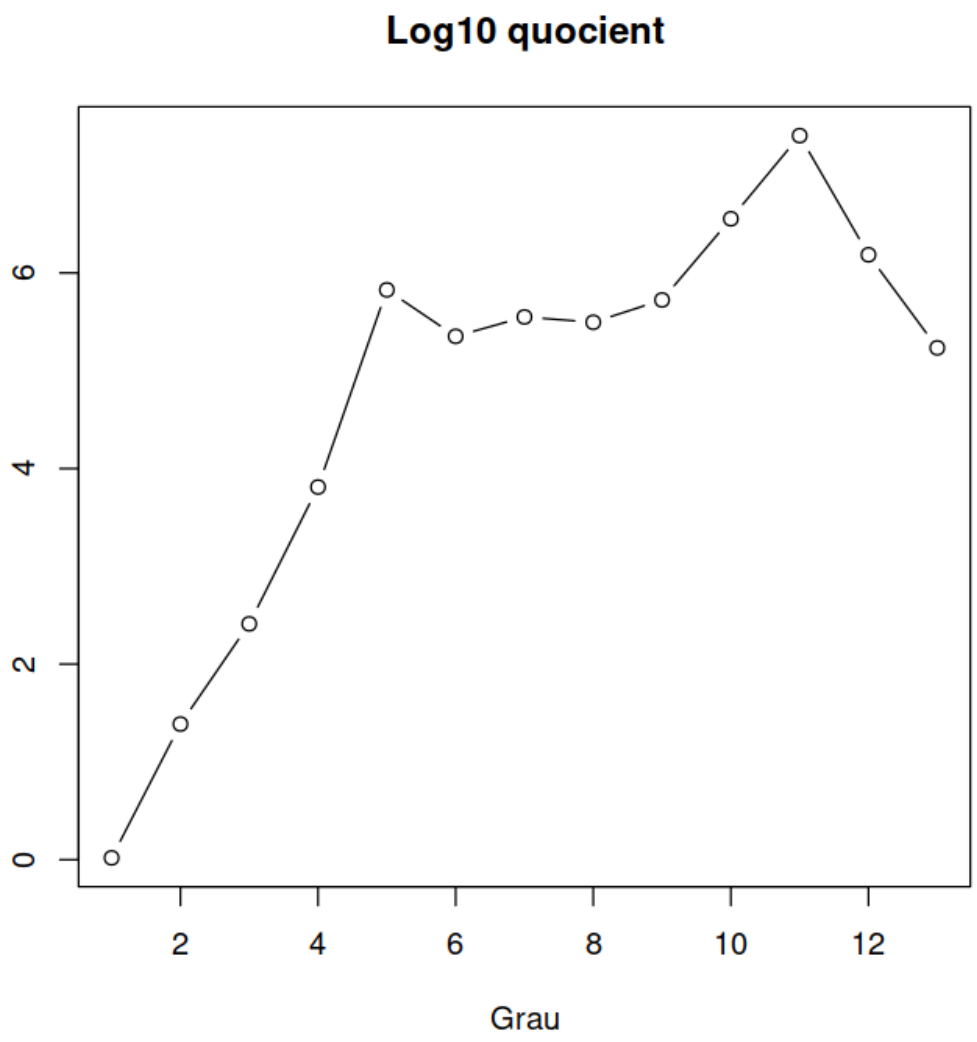
```
In [44]: sqrt(sum(coef(model)^2)) / sqrt(sum(coef(model.final)^2))
```

654241.918153639

per tenir una idea millor, calcularem el quocient entre uns i altres. Com sabreu de classe, el coeficient de grau 0 (Intercept) no es regularitza; per això model.final*coefnoenseldna*, *percoef*()*s*.*Engeneralsmillorusarelsmtodesquepermetenaccediralscampsd*′*unobjecteque hi*′*directament*′*via*

Millor fem un plot logarítmic, per veure els ordres de magnitud

```
In [45]: plot (log10(abs(coef(model) / coef(model.final))),
                xlim=c(1,13), xlab="Grau", ylab="", main="Log10 quocient", type="b")
```



Queda per veure que el model regularitzat no és pitjor que l'inicial (no sigui que per penalitzar-li els coeficients ara sigui un model pitjor)

primer calculem l'error pel mètode estàndar (sense regularitzar) com fins ara

```
In [46]: prediccions.classic <- predict (model, newdata=valid.sample)

In [47]: (NMSE.VA.classic <- sum((valid.sample$target - prediccions.classic)^2)/((N.valid-1)*var
```

4.26530128551942

dóna terrible, com ja imaginavem

Malauradament la funció predict() no accepta un objecte lm.ridge() --això és "culpa" dels programadors de lm.ridge(), que no van fer-ho--, no de predict(); total, que ho hem de fer nosaltres:

primer posem els coeficients en format "xulo", amb els noms de classe

```
In [48]: (c <- setNames(coef(model.final), paste0("c_", 0:12)))
```

c_0 0.566885472687149 **c_1** 4.29750602828978 **c_2** -8.76607795467751 **c_3** -3.77698948337141 **c_4** 0.228698189835508 **c_5** 2.58740521491071 **c_6** 3.53997750126452 **c_7** 3.45477766739867 **c_8** 2.65866549498171 **c_9** 1.38256585048371 **c_10** -0.226681281609164 **c_11** -2.07873990499899 **c_12** -4.11722102863199

ara calculem les potències (els graus) per les dades x

```
In [49]: pots <- outer (X=valid.sample$input, Y=0:12, FUN="^")
```

ara calculem el propi polinomi, multiplicant coeficients per potències i sumant-ho tot (és a dir, calculem el producte escalar entre coeficients i potències)

```
In [50]: prediccions.regul <- pots %*% c
```

finalment ja podem calcular l'error d'aquestes prediccions

```
In [51]: (NMSE.VA.regul <- sum((valid.sample$target - prediccions.regul)^2)/((N.valid-1)*var(val
```

0.23687070966002

dóna 0.2368707, mentre que el model anterior, trobat per prova i error de M donava 0.2335588; és a dir, són dos models igualment bons, però el regularitzat s'ha trobat mitjançant un mecanisme de control de complexitat que és independent del métode (només afecta la funció d'error) i el paràmetre lambda és més fàcil d'ajustar i menys sensible a fluctuacions