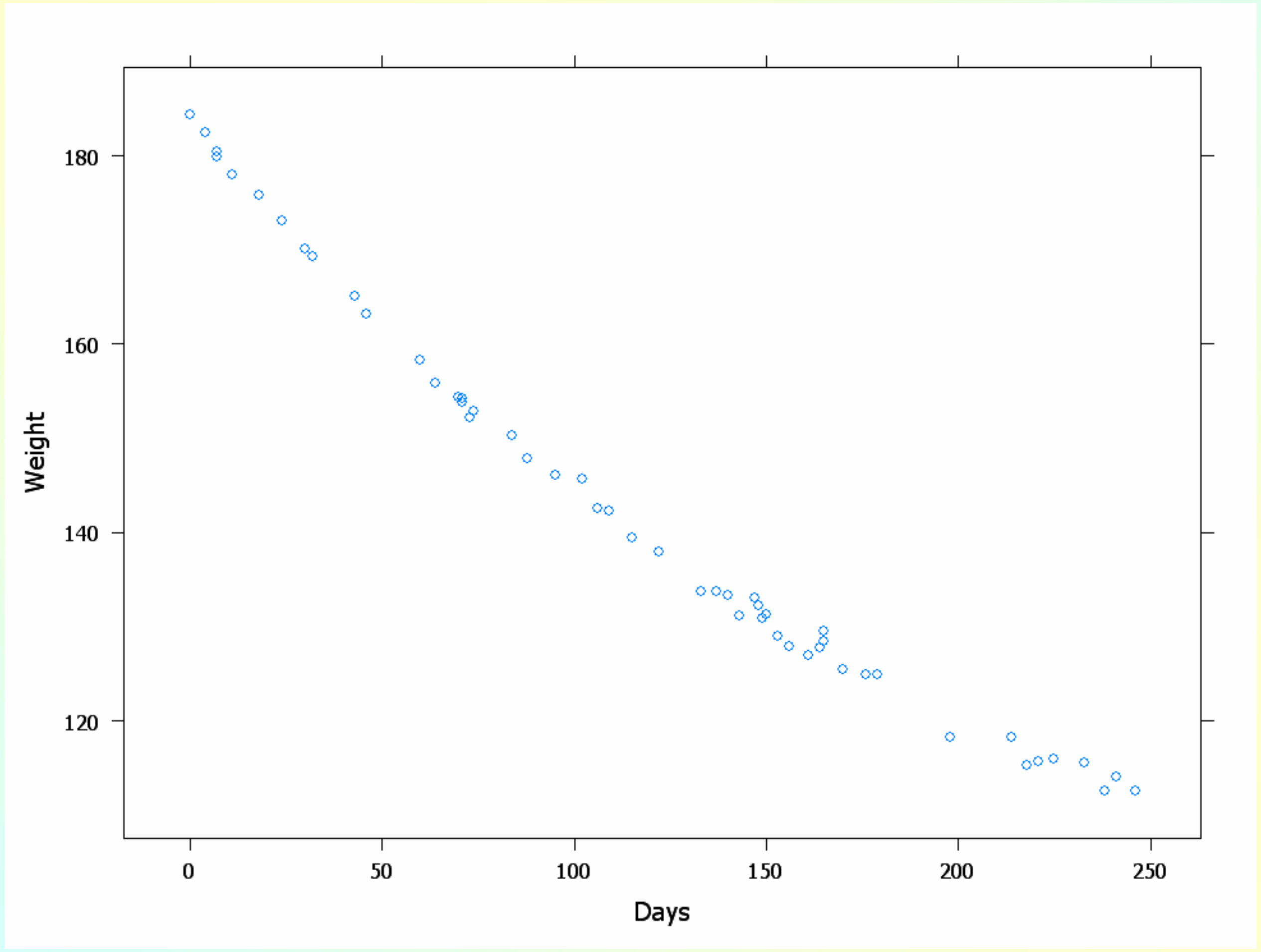
Session 09

Non-linear regression

An example: the weight loss data

```
library(MASS)
xyplot(Weight ~ Days, wtloss)
```



Natural models

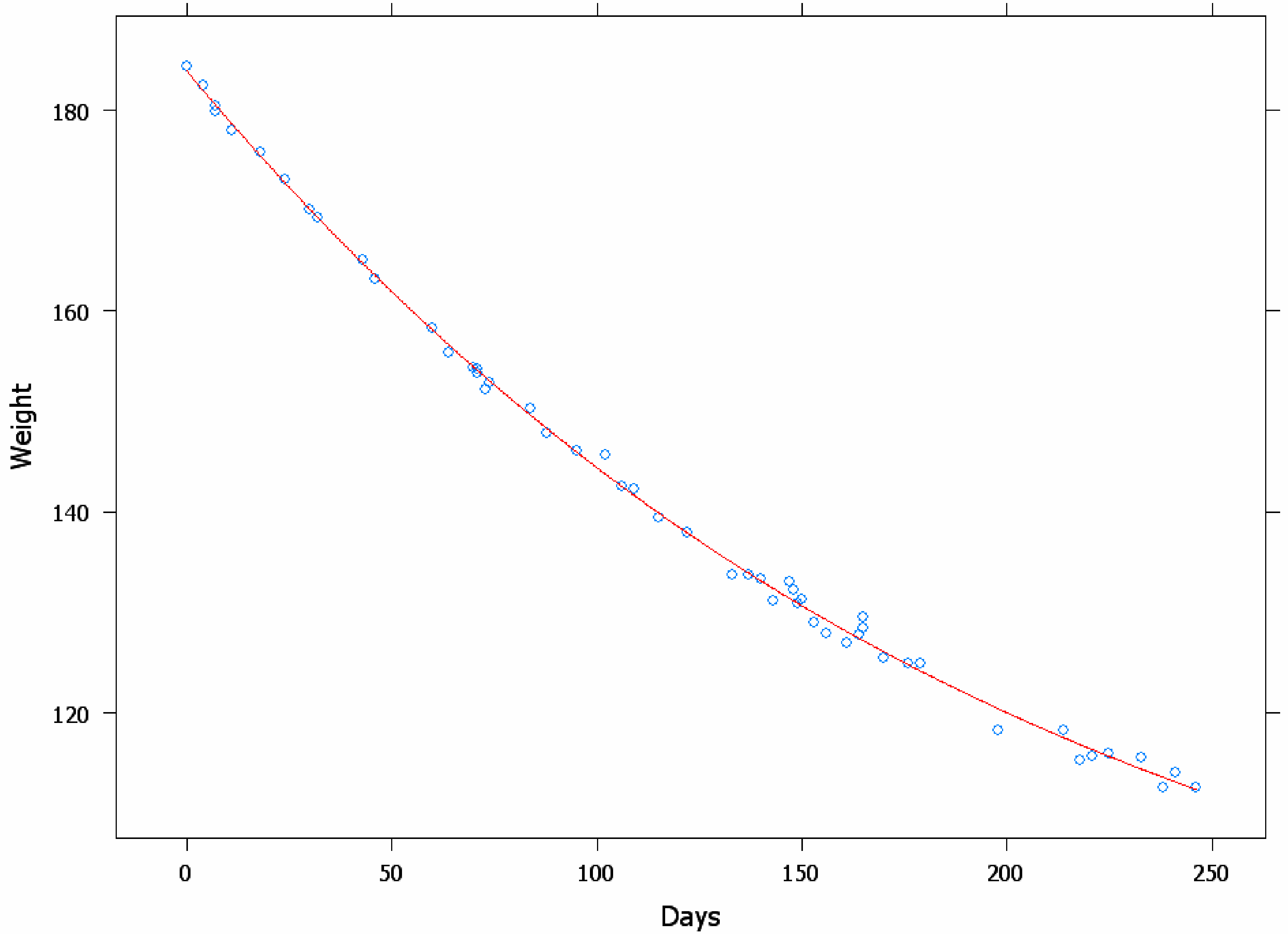
- Polynomials?
  - Quadratic – too pessimistic?
  - Cubic – no evidence yet of an inflexion
- Non-linear?

$$W_i = \beta_0 + \beta_1 2^{-t_i/\tau} + \varepsilon_i$$

- Advantages:
  - Some evidence from dietary theory
  - Clear advantages in short-term extrapolation
- Disadvantage: harder to fit.

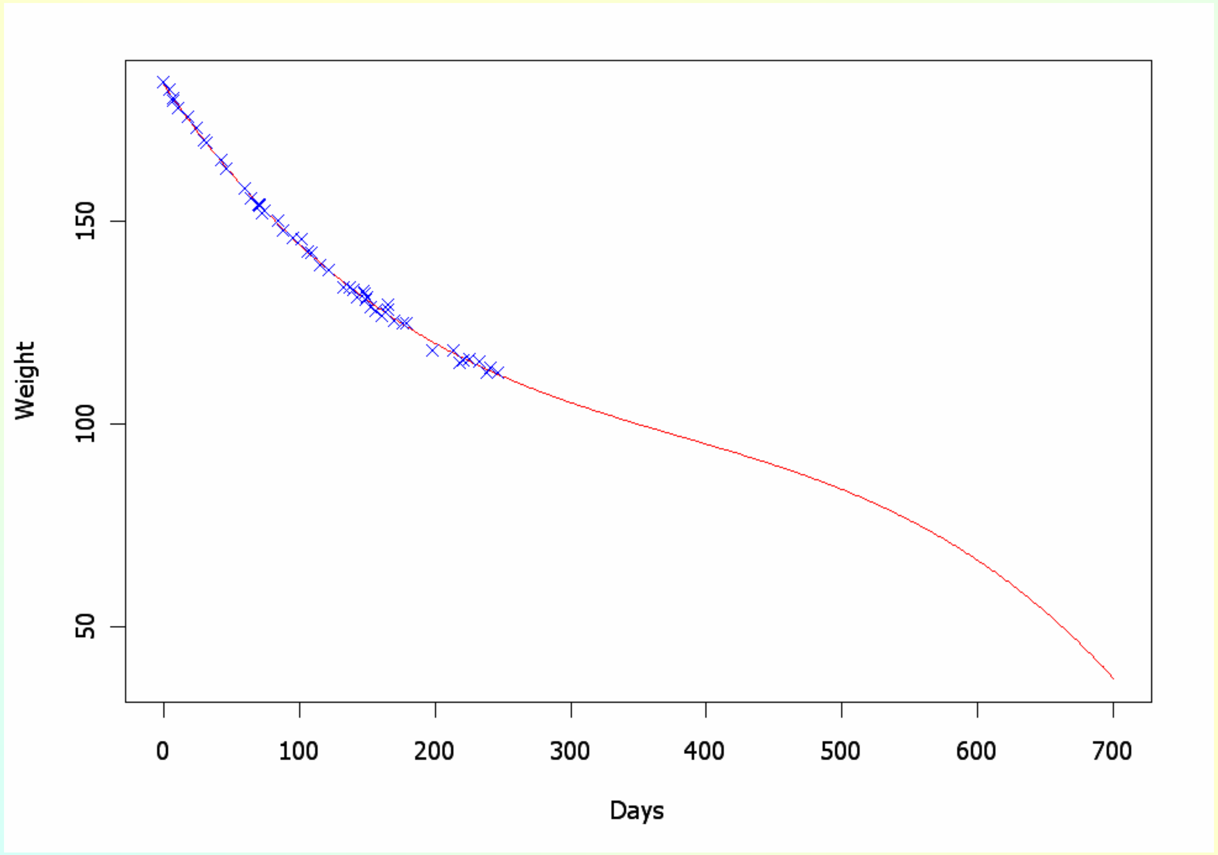
An initial empirical fit

```
wt.lm1 <- lm(Weight ~ poly(Days, 3), wtloss)

Wtloss <- with(wtloss,
  data.frame(Days = seq(min(Days), max(Days), len
  = 1000), Weight = NA))
Wtloss <- rbind(Wtloss, wtloss)
Wtloss <- Wtloss[order(Wtloss$Days), ]

Wtloss$pWeight <- predict(wt.lm1, Wtloss)

library(lattice)
xyplot(Weight ~ Days, Wtloss,
  panel = function(x, y, ...) {
      panel.xyplot(x, y, ...)
      panel.lines(x, Wtloss$pWeight, col = "red")
})
```


Weight
180
160
140
120
0
50
100
150
200
250
Days

What happens under extrapolation?

```
plot(0:700, predict(wt.lm1, data.frame(Days = 0:700)),
  type = "l", xlab = "Days", ylab = "Weight", col = "red")
with(wtloss, points(Days, Weight, pch = 4, col = "blue"))
```



Initial values 1

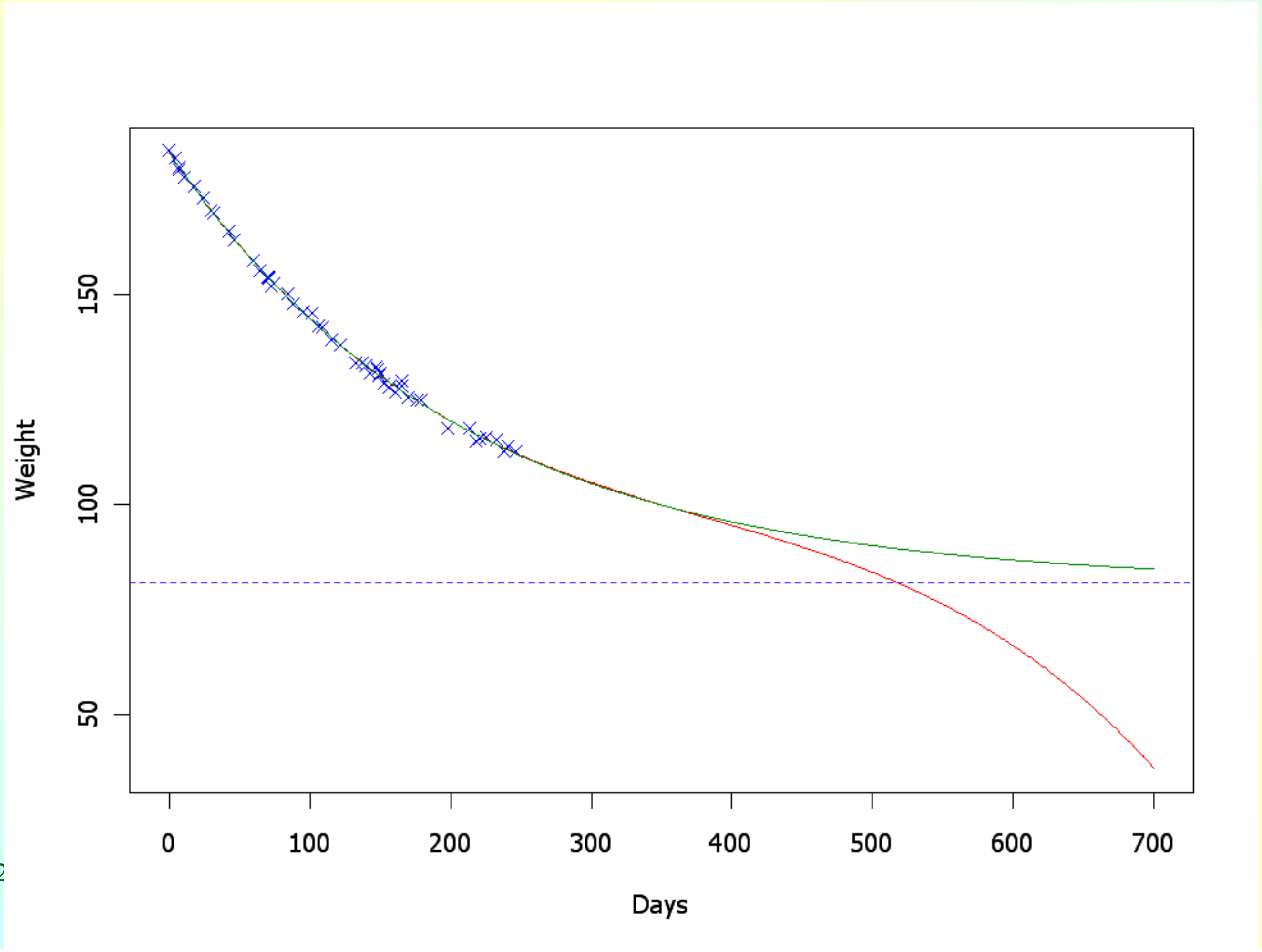
First algorithm: guess them!

```
wt.nls0 <-
   nls(Weight ~ b0 + b1 * 2^( - Days/tau),
   wtloss,
   start = c(b0 = 70, b1 = 122, tau = 150),
   trace = T)
```

```
769.5897  :   70 122 150
 42.10227 :   81.57397 102.48311 142.58520
 39.24473 :   81.3761  102.6819  141.9020
 39.2447  :   81.3738  102.6841  141.9104
```

Extrapolation under the non-linear model

```
lines(0:700, predict(wt.nls0, data.frame(Days = 0:700)),
             col = "green4")
abline(h = coef(wt.nls0)[1], lty = "dashed", col = "blue")
```



Initial values 2

Second algorithm: calculate approximate values

$$\eta_0 = \beta_0 + \beta_1 2^{-t_0/\tau}$$

$$\eta_1 = \beta_0 + \beta_1 2^{-(t_0+\delta)/\tau}$$

$$\eta_2 = \beta_0 + \beta_1 2^{-(t_0+2\delta)/\tau}$$

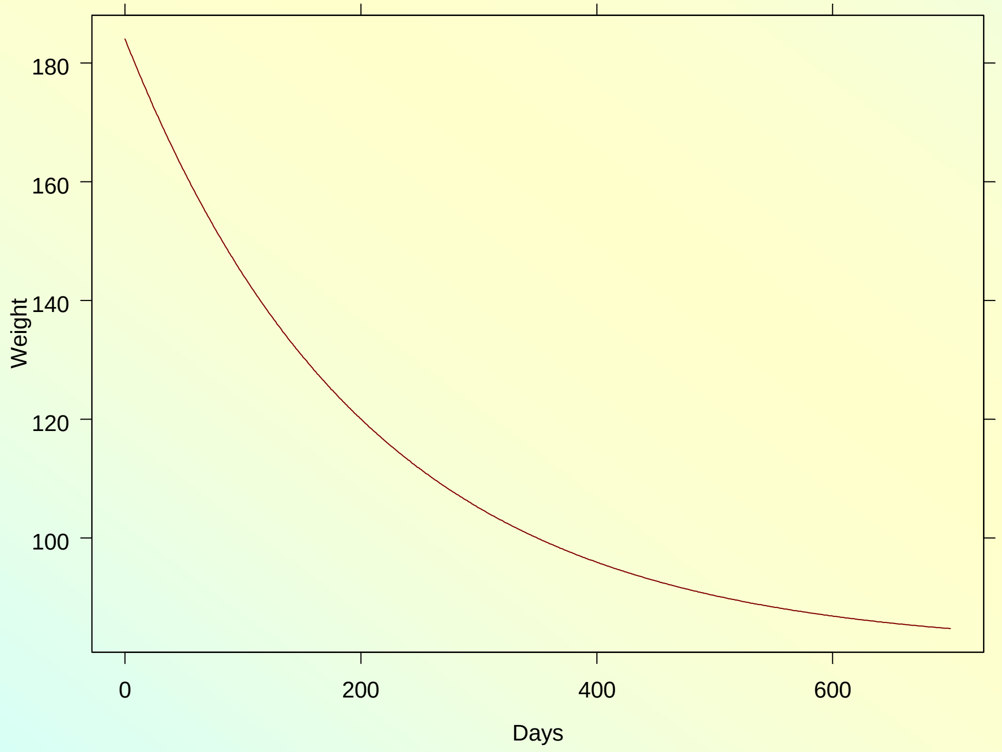
$$\tau = \frac{\delta \log(2)}{\log\left((\eta_1 - \eta_0)/(\eta_2 - \eta_1)\right)}$$

Initial values 2

```
rg <- range(wtloss$Days)
del <- 0.25*(rg[2] - rg[1])
eta <- predict(wt.lm1, data.frame(Days = rg[1] +
   1:3*del))
deta <- diff(eta)
t0 <- as.vector(del*log(2)/log(deta[1]/deta[2]))
ab <- as.vector(coef(lm(Weight ~ I(2^(-Days/t0)),
   wtloss)))
init <- c(b0 = ab[1], b1 = ab[2], tau = t0)
wt.nls1 <- nls(Weight ~ b0+b1*2^(-Days/tau), wtloss,
   start = init, trace = T)
39.46115 :   82.51384 101.67121 139.21188
39.24485 :   81.39093 102.66630 141.87092
39.2447  :   81.37376 102.68416 141.91049
```

Now what happens under extrapolation?

```
tmp <- data.frame(Days = 0:700)
tmp$Weight <- predict(wt.nls1, tmp)
xyplot(Weight ~ Days, tmp, type="l")
```



Cementing it inside a function

```
ival <- function(y, x) {
  rg <- range(x)
  del <- 0.25 * (rg[2] - rg[1])
  fm <- lm(y ~ x + I(x^2))
  dat <- data.frame(x = rg[1] + 1:3 * del)
  eta <- predict(fm, dat)
  deta <- diff(eta)
  t0 <- as.vector((del * log(2))/log(deta[1]/deta[2]))
  ab <- as.vector(coef(lm(y ~ I(2^( - x/t0)))))
  c(b0 = ab[1], b1 = ab[2], tau = t0)
}
with(wtloss, ival(Weight, Days))
      b0        b1       tau
 81.3299 102.7232 142.0143
```

Derivatives

- The non-linear least squares fitting algorithm can use first derivatives if they are available.
- The regression function must be an S function that returns the values with the derivatives as an attribute.
- There is an important tool for doing the differentiation and providing the function in one step.

```
negexp <- deriv(~ b0 + b1*2^(-x/tau),
   c("b0", "b1", "tau"),function(x, b0, b1, tau) {})

wt.nls2 <- nls(Weight ~ negexp(Days, b0, b1, tau),
  wtloss, start = with(wtloss, ival(Weight, Days)),
  trace = T)

39.24501 :   81.3299 102.7232 142.0143
39.2447  :   81.3740 102.6840 141.9099
39.2447  :   81.37382 102.68412 141.91037
```

```
> negexp
function (x, b0, b1, tau) {
    .expr3 <- 2^(-x/tau)
    .value <- b0 + b1 * .expr3
    .grad <- array(0, c(length(.value),
  3L), list(NULL, c("b0", "b1", "tau")))
    .grad[, "b0"] <- 1
    .grad[, "b1"] <- .expr3
    .grad[, "tau"] <- b1 * (.expr3 *
  (log(2) * (x/tau^2)))
    attr(.value, "gradient") <- .grad
    .value
}
```

Putting it all together: Self-starters

- The regression (S-)function may also include an attribute giving an algorithm for finding initial values
- If this is possible, it makes fitting such non-linear models very convenient, and if the initial value routine is good, very secure
- Writing such a self-starting regression function requires us to use some obscure conventions. This is necessary because it has to cohere to the model fitting function itself.
- We are about to write a function that will not be called by us, but by **`nls()`** itself.

```
SSival <- function(mCall, data, LHS)
{
#
# y ~ b0 + b1*2^(-x/tau)
#
  x <- eval(mCall[["x"]], data)
  y <- eval(LHS, data)
  rg <- range(x)
  del <- 0.25 * (rg[2] - rg[1])
  fm <- lm(y ~ x + I(x^2))
  dat <- data.frame(x = rg[1] + 1:3 * del)
  eta <- predict(fm, dat)
  deta <- diff(eta)
  t0 <- as.vector((del * log(2))/log(deta[1]/deta[2]))
  ab <- as.vector(coef(lm(y ~ I(2^( - x/t0)))))
   pars <- list(ab[1], ab[2], t0)
   names(pars) <- mCall[c("b0", "b1", "tau")]
   pars
}
```

Trying it out

```
SSnegexp <- selfStart(model = ~ b0 + b1*2^(-x/tau),
   initial = SSival, parameters = c("b0", "b1", "tau"),
   template = function(x, b0, b1, tau) {})

wt.nls3 <- nls(Weight ~ SSnegexp(Days, a, b, t),
  wtloss, trace = T)
```

```
39.24501 :   81.3299  102.7232  142.0143
39.2447  :   81.3740  102.6840  141.9099
39.2447  :   81.37382 102.68412 141.91037
```

- Looks pretty good!

The weight loss data: summary

```
summary(wt.nls3)

Formula: Weight ~ SSnegexp(Days, a, b, t)

Parameters:
    Value Std. Error t value
a  81.374    2.26899 35.8635
b 102.684    2.08275 49.3021
t 141.910    5.29449 26.8033

Residual standard error: 0.894937 on 49 degrees of freedom

Correlation of Parameter Estimates:
       a      b
b -0.989
t -0.986  0.956
```

Second example: Stormer Viscometer

- The regression is of the form

$$T = \frac{\beta v}{w - \theta} + \varepsilon$$

- Ignoring the error, simplifying and re-writing gives

$$wT = \beta v + \theta T$$

- The rest is easy: regress y = wT on x1 = V and x2 = T with no intercept

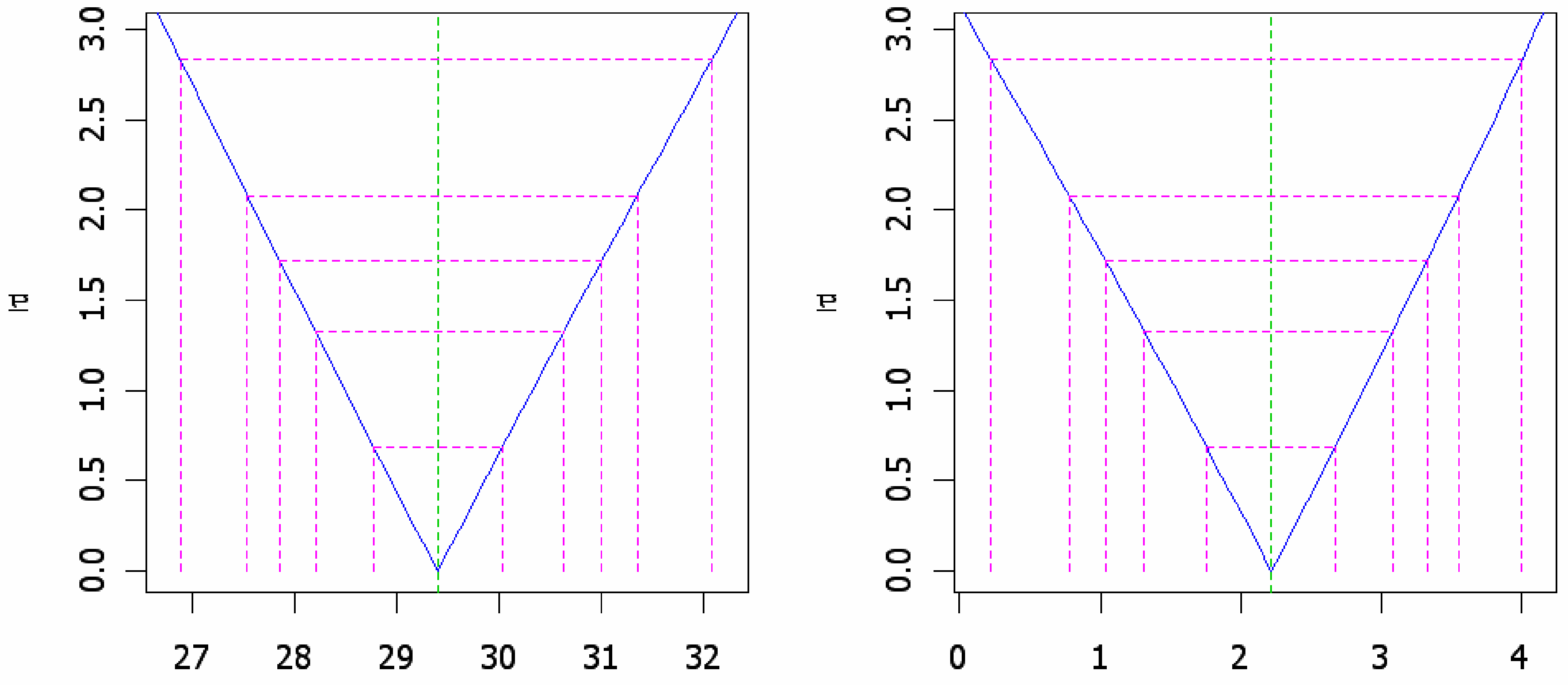
Trying out the stormer example

```
SSival.storm <- function(mCall, data, LHS) {
#
# y ~ b*x1/(x2-theta)
#
   x1 <- eval(mCall[["x1"]], data)
   x2 <- eval(mCall[["x2"]], data)
   y <- eval(LHS, data)
   cf <- lsfit(cbind(x1, y), x2*y, int = F)$coef
   names(cf) <- mCall[c("b", "theta")]
   cf
}
storm <- selfStart(model = ~ b*x1/(x2-theta),
   initial = SSival.storm, parameters = c("b", "theta"),
   template = function(x1, x2, b, theta) {})

storm.nls1 <- nls(Time ~ storm(Viscosity, Wt, beta, tau), stormer,
   trace=T)

885.3645 :  28.875541  2.843728
825.1098 :  29.393464  2.233276
825.0514 :  29.401327  2.218226
825.0514 :  29.401257  2.218274
```

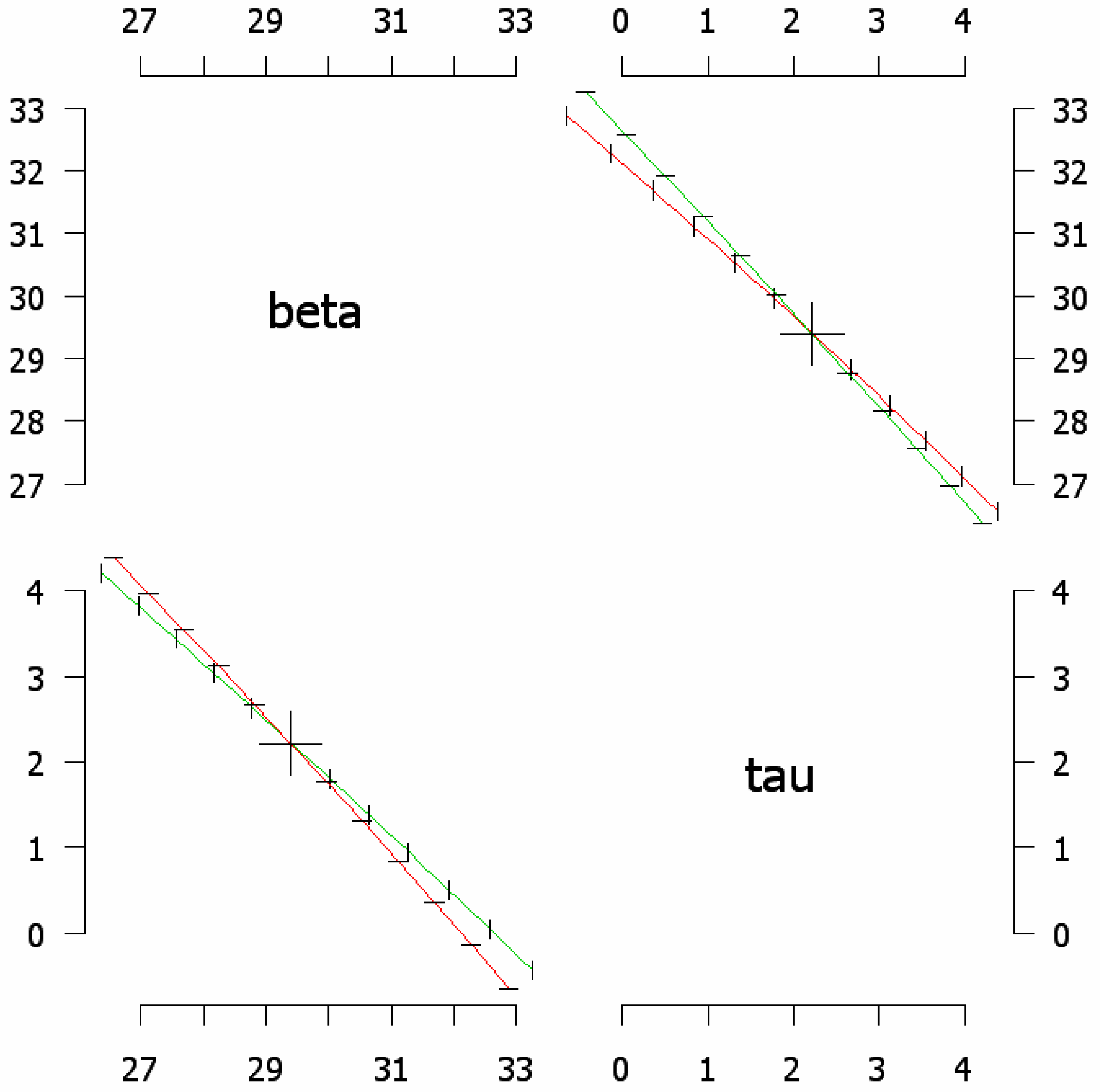
Confidence intervals

- Marginal intervals are available as estimate ± 2SE in much the same way as LM/GLM
- Joint intervals are sometimes useful. These need to be constructed individually
- *Profiling* can also be used to find likelihood ratio intervals and to give some idea of joint behaviour of the estimates

```
storm.nls1$call$trace <- FALSE # turn off the trace
par(mfrow = c(1,2))
storm.pf <- profile(storm.nls1)
plot(storm.pf)
par(mfrow = c(1,1))
pairs(storm.pf)
```


0.0
0.5
1.0
1.5
2.0
2.5
3.0
27
28
29
30
31
32
0
1
2
3
4

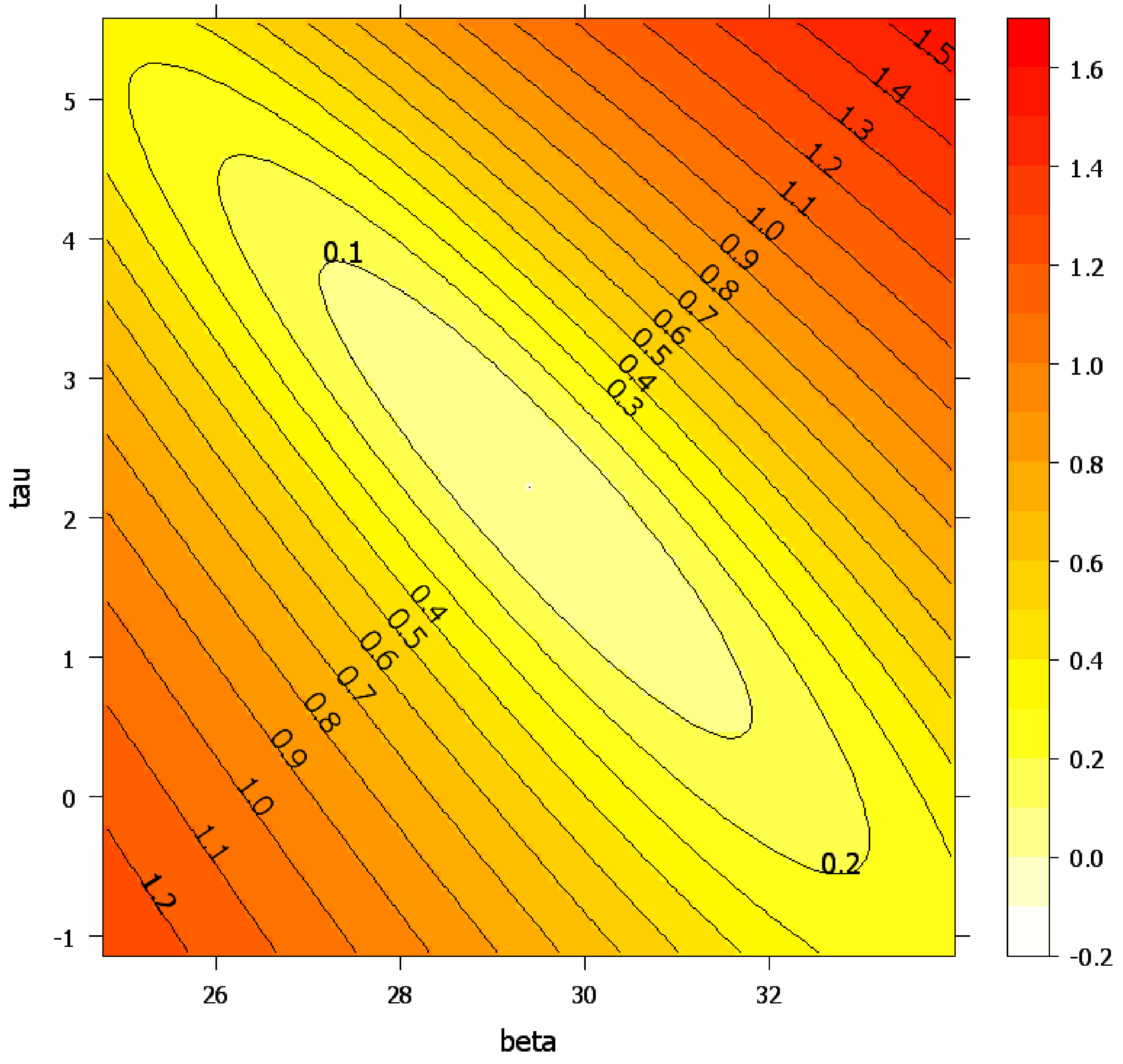
Time~storm(Viscosity, Wt, beta, tau)



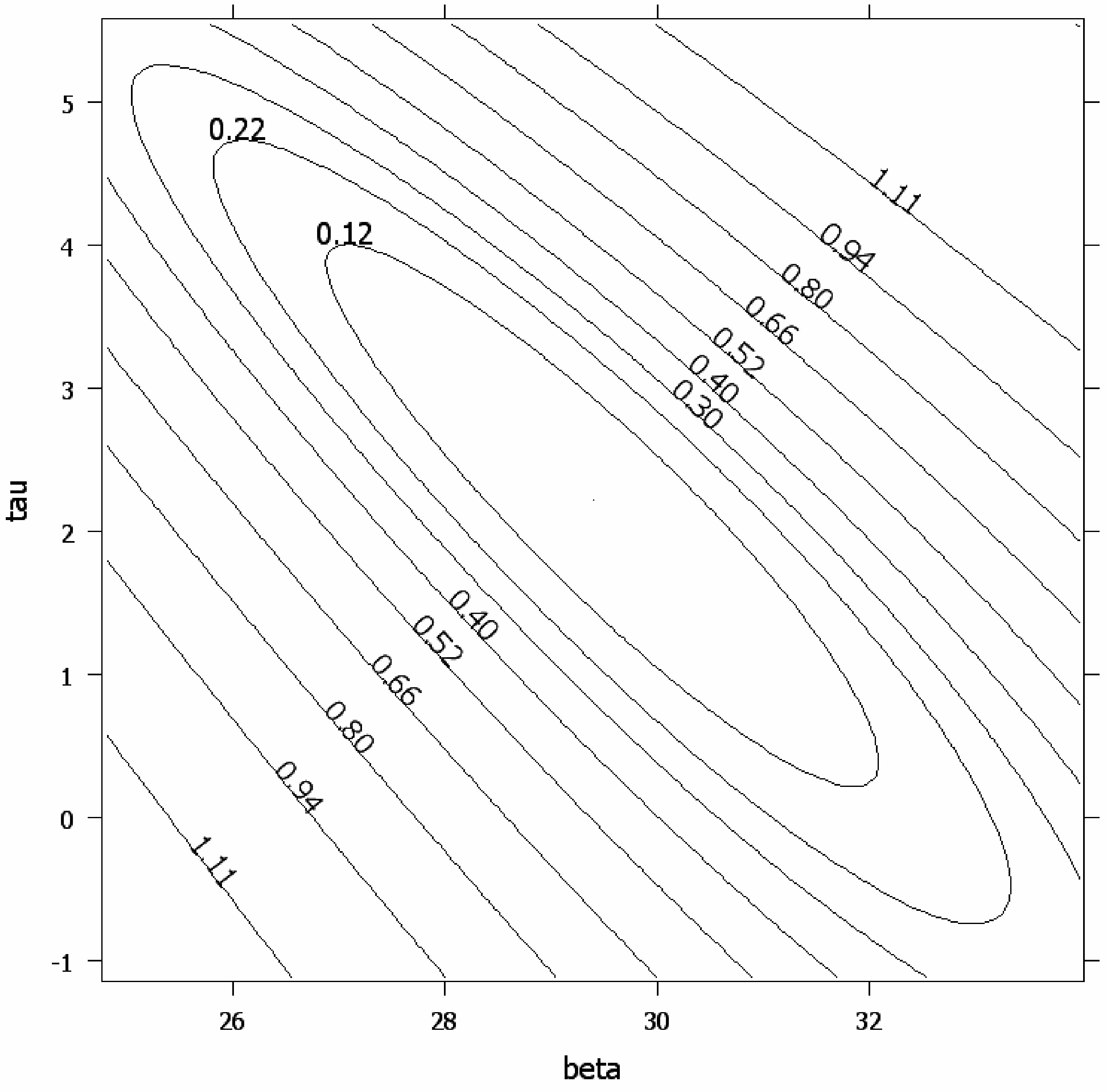
Notes

- Behaviour of the log-likelihood surface near the MLE appears to be quadratic, implying that the approximate confidence intervals are probably reasonable
- The parameter estimates are highly correlated.
- For more detailed behaviour we need to look at the bivariate surface directly

```
b <- coef(storm.nls1)
se <- sqrt(diag(vcov(storm.nls1)))

del <- seq(-5, 5, length = 101)
Stm <- expand.grid(beta = b[1] + del*se[1], tau = b[2]
   + del*se[2])

Regfn <- with(stormer,
  outer(Viscosity, Stm$beta)/
             outer(Wt, Stm$tau, "-"))
res <- matrix(stormer$Time, nrow(stormer), nrow(Stm)) -
   Regfn
Stm$RSS <- colSums(res*res)/1000
Stm$RSS <- with(Stm, log10(1 + RSS - min(RSS)))

levelplot(RSS ~ beta + tau, Stm, contour = TRUE, pretty
   = TRUE,
  col.regions = rev(heat.colors(100)))

cont <- round(quantile(Stm$RSS,0:10/10), 2)
contourplot(RSS ~ beta+tau, Stm, at=cont)
```



0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
1.1
1.2
1.3
1.4
1.5
tau
beta
26
28
30
32
-1
0
1
2
3
4
5
-0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6


0.22
0.12
1.11
0.94
0.80
0.66
0.52
0.40
0.30
0.40
0.52
0.66
0.80
0.94
1.11
tau
5
4
3
2
1
0
-1
26
28
30
32
beta

Bootstrap intervals

- Three possible ways of using bootstrap ideas on non-linear regression:
  - Bootstrap on the observations themselves: Could lead to cases where the model is non-estimable.
  - Bootstrap on the residuals, centred or uncentred
  - The "Bayesian bootstrap": give each observation a variable weight (as the ordinary bootstrap does) and vary the weights. It can be shown that an exp(1) weight is appropriate. Rubin, D. B. (1981). The Bayesian Bootstrap. *Annals of Statistics*, 9, 130-134

A demonstration: the weights are “about right”

```
m <- v <- numeric(100)
for(j in 1:100) {
  sam <- factor(sample(1:50, 50, rep = T), levels =
   1:50)
  sam <- table(sam)
  m[j] <- mean(sam)
  v[j] <- var(sam)
}
mean(m)
[1] 1
mean(v)
[1] 1.013061
rv <- rexp(5000, 1)
mean(rv)
[1] 0.9899044
var(rv)
[1] 0.9623089
```

Trying it out

```
B <- matrix(NA, 1000, 2)

for(i in 1:1000) {
  w <- rexp(nrow(stormer), 1)
  tmp <- try(update(storm.nls1, weights = w))
  if(class(tmp)[1] != "try-error")
    B[i,  ] <- coef(tmp)
}
```

```
> colMeans(B)
[1] 29.344850  2.278037
> var(B)
           [,1]       [,2]
[1,]  0.3468216 -0.3090304
[2,] -0.3090304  0.3718592
```

```
B1 <- matrix(NA, 1000, 2)
rs <- scale(resid(storm.nls1), scale = F)
yt <- stormer$Time - rs
for(i in 1:1000) {
  ThisY <- as.vector(yt + sample(rs, rep = T))
  tmp <- try(update(storm.nls1, ThisY ~ .))
  if(class(tmp)[1] != "try-error")
    B1[i,  ] <- coef(tmp)
}
> colMeans(B1)
[1] 30.10174  1.94715
> var(B1)
           [,1]       [,2]
[1,]  0.7475719 -0.4950940
[2,] -0.4950940  0.3882688
```

Final comparisons

```
> apply(B, 2, quantile, c(1, 39)/40)
          [,1]      [,2]
2.5%  28.09367 0.9321281
97.5% 30.52142 3.3800514

> apply(B1, 2, quantile, c(1, 39)/40)
          [,1]      [,2]
2.5%  28.37939 0.7236612
97.5% 31.69766 3.1115609

> tau <- qt(0.975, nrow(stormer)-2)
> b <- coef(storm.nls1)
> se <- sqrt(diag(vcov(storm.nls1)))
>
> rbind(b - tau*se, b + tau*se)
         beta      tau
[1,] 27.49730 0.834246
[2,] 31.30521 3.602302
```

Final comments

- Non-linear regression offers a way of integrating empirical and “process” models
- If kinds of non-linear regression are to be repeatedly done, some investment in selfStart models pays off.
- Bootstrap confidence intervals for the parameters can be relatively straightforward, if the model is small and the data set manageable
- "Tests of fit" are not automatically available. Graphical examination of residuals is important here, as elsewhere.