Working with 

A University of Queensland Advanced Workshop

Session 8: Going Non-linear

Bill Venables, CSIRO/Data61, Dutton Park

Rhetta Chappell, Griffith University

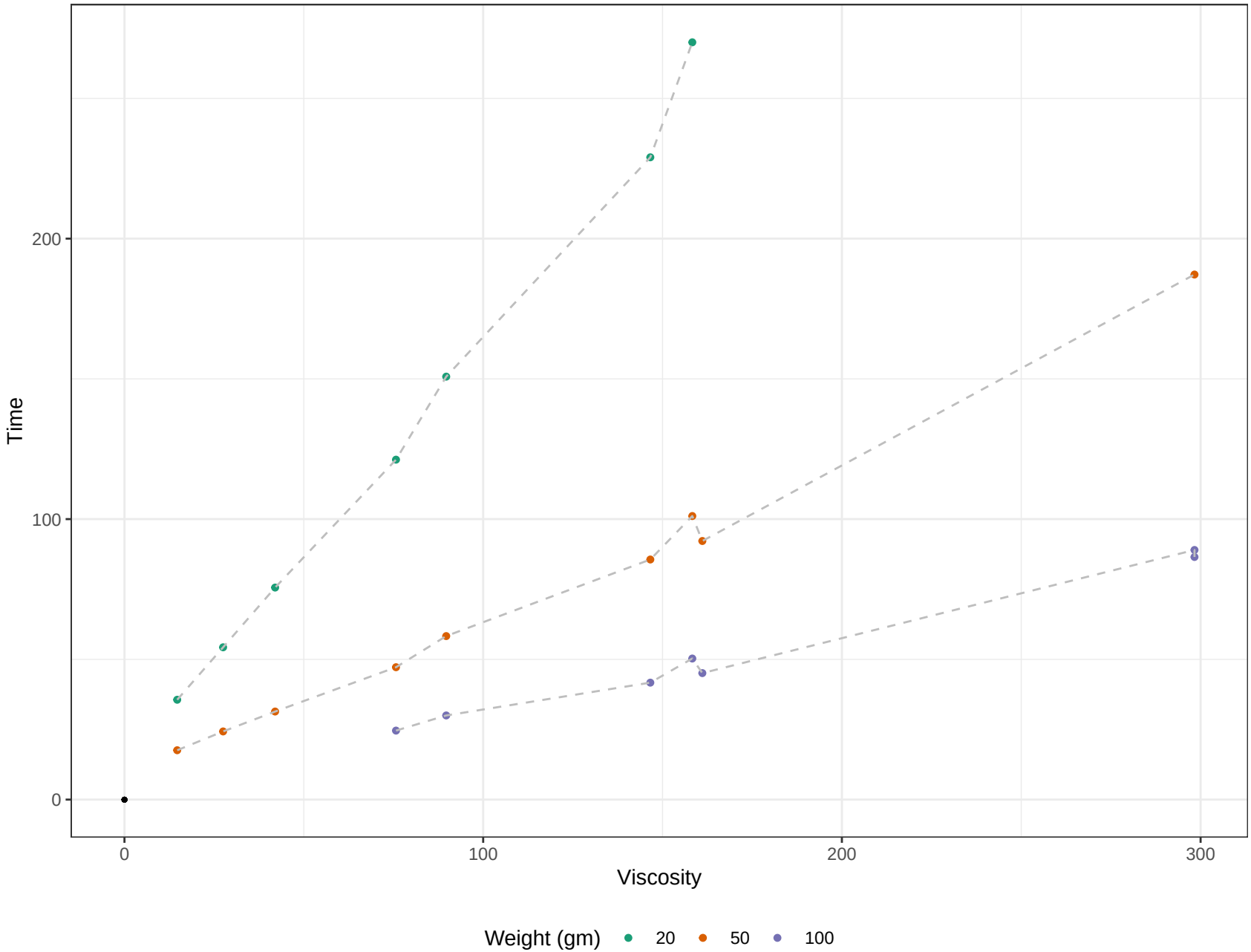
2–5 February, 2021

Contents

1 Stormer viscometer revisited

Look again at the data:

```
p0 <- ggplot(Stormer) +
  aes(x = Viscosity, y = Time, group = factor(Weight), colour = factor(Weight)) +
  geom_point(size = 1.25)  + theme_bw() +
  geom_line(linetype = "dashed", colour = "grey") +
  scale_color_brewer(palette = "Dark2", name = "Weight (gm)") +
  theme(legend.position = "bottom")

p0  + geom_point(aes(x = 0, y = 0), inherit.aes = FALSE, size = 0.7)
```


Time
200
100
0
0
100
200
300
Viscosity
Weight (gm)
20
50
100

The model from theory is

$$\text{Time} = \frac{\beta \text{ Viscosity}}{\text{Weight} - \theta} + \varepsilon$$

where ε is a $N(0, \sigma^2)$ measurement.

- If we knew θ this would be a regression (through the origin).
- Not knowing θ makes this a *non-linear* regression.
- β is known as a *linear parameter*. Special algorithms are available for non-linear models with one or more linear parameters.

Parameters are estimated by least squares ($\equiv$ maximum likelihood), but the process is necessarily iterative, and the algorithms need a bit of help.

- Starting approximate values are needed for the non-linear parameters,
- If the partial derivatives of the model function with respect to the parameters are also supplied, this can improve the process.

1.1 Finding initial values and first fit

This usually requires either good inside knowledge – or some ingenuity.

Ignore ε for now, multiply through by (Weight $- \theta$) and re-arrange the terms. This gives the "pseudo-model":

$$\text{Time} \times \text{Weight} \approx \beta \text{ Viscosity} + \theta \text{ Time}$$

This looks like a linear regression. Let's fit it despite the outrage:

```
storm_iv <- lm(I(Time*Weight) ~ 0 + Viscosity + Time, Stormer)
(init <- setNames(coef(storm_iv), c("beta", "theta")))
```

```
     beta     theta
28.875541  2.843728
```

Fitting the model is now easy:

```
storm_nl <- nls(Time ~ beta*Viscosity/(Weight-theta), data=Stormer, start=init)
round(summary(storm_nl)$coefficients, digits = 3)
```

```
      Estimate Std. Error t value Pr(>|t|)
beta    29.401      0.916  32.114    0.000
theta    2.218      0.666   3.333    0.003
```

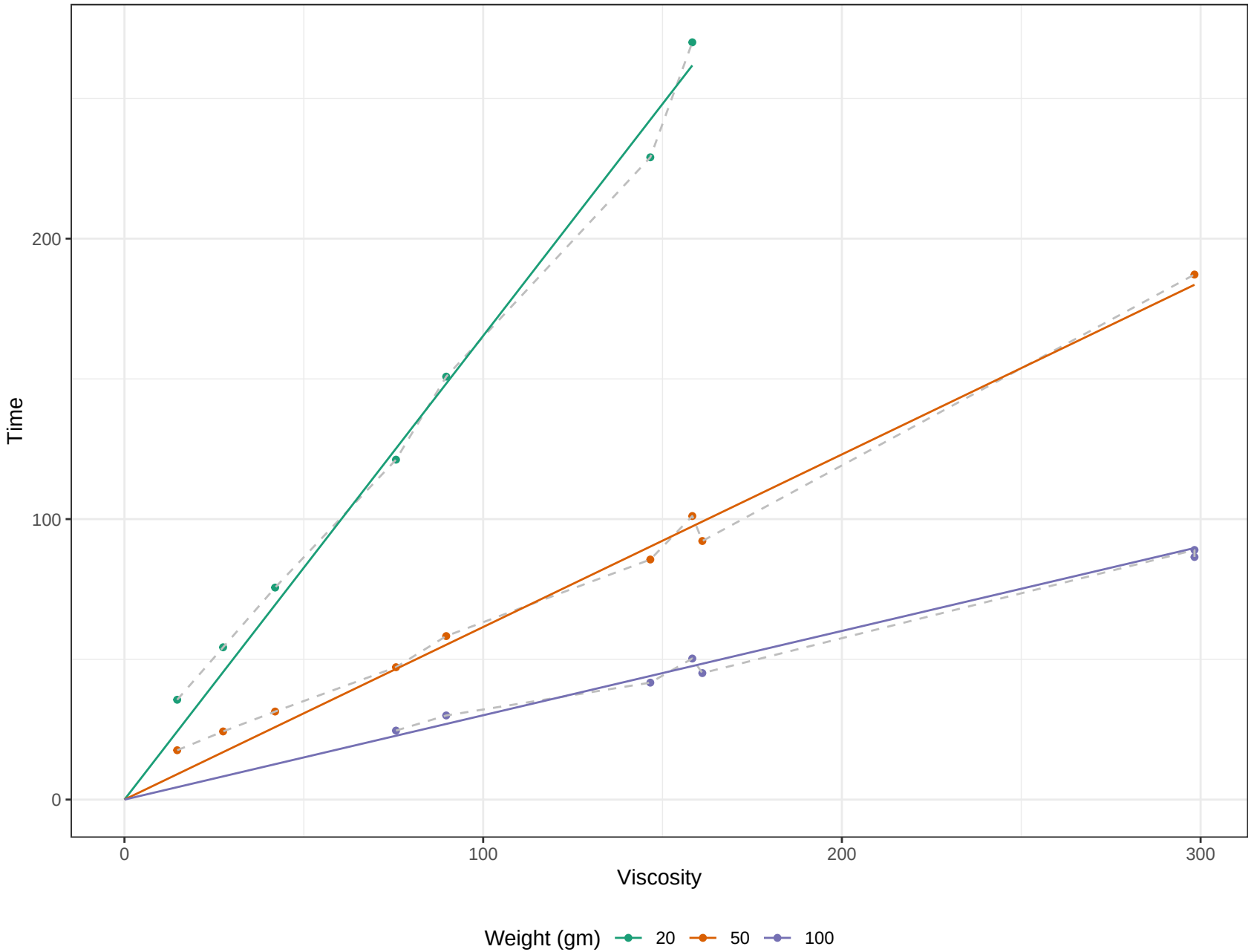
1.2 Inspecting the fit

Put the model predictions on the graph of the data:

```
m <- with(Stormer, tapply(Viscosity, Weight, max))
w <- as.numeric(names(m))
pStormer <- rbind(cbind(Viscosity = 0,          Weight = w),
                  cbind(Viscosity = unname(m), Weight = w)) %>%
  data.frame()
pStormer$Time <- predict(storm_nl, newdata = pStormer)
pStormer
```

```
  Viscosity Weight     Time
1       0.0     20   0.0000
2       0.0     50   0.0000
3       0.0    100   0.0000
4     158.3     20 261.7417
5     298.3     50 183.5512
6     298.3    100  89.6936
```

```
p0 + geom_line(data = pStormer)
```


Time
200
100
0
0
100
200
300
Viscosity
Weight (gm)
20
50
100

1.3 Adding derivatives

This extension is *required* for non-linear random effect models.

```
(stormer <- deriv(~ b*V/(W - t), namevec = c("b", "t"),
                  function.arg = function(V, W, b, t) {}) %>% fix_deriv())

function (V, W, b, t)
{
    .expr1 <- b * V
    .expr2 <- W - t
    .value <- .expr1/.expr2
    .grad <- array(0, c(length(.value), 2L), list(NULL, c("b",
        "t")))
    .grad[, "b"] <- V/.expr2
    .grad[, "t"] <- .expr1/.expr2^2
    .actualArgs <- as.list(match.call())[colnames(.grad)]
    if (all(vapply(.actualArgs, is.name, NA))) {
        colnames(.grad) <- .actualArgs
    }
    attr(.value, "gradient") <- .grad
    .value
}
```

Putting the function into action:

```
storm_nl2 <- nls(Time ~ stormer(Viscosity, Weight, beta, theta), Stormer,
                 start = init)
summary(storm_nl2)


Formula: Time ~ stormer(Viscosity, Weight, beta, theta)

Parameters:
      Estimate Std. Error t value Pr(>|t|)
beta   29.4013     0.9155  32.114  < 2e-16
theta   2.2183     0.6655   3.333  0.00316

Residual standard error: 6.268 on 21 degrees of freedom

Number of iterations to convergence: 3
Achieved convergence tolerance: 6.376e-08
```

Wouldn't it be great if we could put the initial value process into code as well? We can!

1.4 Self-starting models

This is something of a detailed topic, but important if you work in this area.

```
SSstormer <- selfStart(model = ~ b*v/(w - c), parameters = c("b", "c"),
                       initial = function(mCall, data, LHS, ...) {## '...' needed
                         t <- eval(LHS, data)                      ## in R 4.0.4
                         v <- eval(mCall[["v"]], data)
                         w <- eval(mCall[["w"]], data)
                         b <- coef(lm(I(w*t) ~ 0 + v + t))
                         setNames(b, mCall[c("b", "c")])
                       },
                       template = function(v, w, b, c) {})
storm_nl3 <- nls(Time ~ SSstormer(Viscosity, Weight, beta, theta),
                 data = Stormer)
coef(storm_nl3)
```

```
     beta     theta
29.401257  2.218274
```

1.5 Removing the kinks

One possible reason for the (apparently) systematic 'kniks' in the data might be that the assumed viscosities contained measurement errors. This suggests we at least *look at* a model of the form

$$\text{Time} = \frac{\beta\,(\text{Viscosity} + \phi)}{\text{Weight} - \theta} + \varepsilon = \frac{(\beta\phi) + \beta\ \text{Viscosity}}{\text{Weight} - \theta} + \varepsilon$$

The initial value 'pseudo-model' is only mildly changed:

$$\text{Time} \times \text{Weight} \approx \beta^{\star} + \beta\ \text{Viscosity} + \theta\ \text{Time}$$

where $\beta^{\star} = \beta\phi$ is a simple re-parametrisation. This allows us to devise an initial value function and a self-starting model.

```
SSstormer2 <- selfStart(~ b*(v + f)/(w - c),
                        parameters = c("b", "c", "f"),
                        initial = function (mCall, data, LHS, ...) {
                          t <- eval(LHS, data)
                          v <- eval(mCall[["v"]], data)
                          w <- eval(mCall[["w"]], data)
                          b <- coef(lm(I(w * t) ~ 1 + v + t))  ## incl intercept
                          b <- c(b[2:3], b[1]/b[2])            ## change back
                          setNames(b, mCall[c("b", "c", "f")])
                        },
                        template = function(v, w, b, c, f) {}) %>% fix_deriv()
storm_nl4 <- nls(Time ~ SSstormer2(Viscosity, Weight, beta, theta, phi),
                 data = Stormer)
coef(storm_nl4)

     beta     theta       phi
28.684857  1.638091  6.631461
```

This suggests there could be a negative bias of $\phi = 6.63$. A more thorough investigation may allow ϕ to vary for the different "known" viscosities.

1.5.1 Random effects

Rather than fit a different ϕ for each viscosity, a more realistic approach with such a small dataset is to allow ϕ to have a random component, $\phi + \delta$, where δ varies over the different viscosities, $\delta \sim \mathrm{N}\left(0, \sigma_\delta^2\right)$.

The fitting function is *lme4::nlemr*, and the main point of difference is the way the formula specifies the random effects. Self-starting model functions still work (provided they are adjusted by *fix_deriv()*!) but do not generate starting values. These must be supplied, but need only be supplied for the fixed effect parts.

```
storm_nle <- nlmer(
  Time ~ SSstormer2(Viscosity,Weight,beta,theta,phi) ~ (phi|Viscosity), ## 3-form
  data = Stormer, start = coef(storm_nl4))                              ## start!
rbind(`base model`             = c(coef(storm_nl), phi = 0),
      perturbed                = coef(storm_nl4),
      `nlmer (fixed effects)` = fixef(storm_nle)) %>% round(4)
```

```
                         beta  theta    phi
base model            29.4013 2.2183 0.0000
perturbed             28.6849 1.6381 6.6315
nlmer (fixed effects) 28.7877 1.6967 5.5690
```

The BLUPs ($\phi + \delta$) are shown below

```
(B <- coef(storm_nle)$Viscosity)
```

```
          beta    theta        phi
14.7  28.78774 1.696709  8.5654336
27.5  28.78774 1.696709  7.6457594
42    28.78774 1.696709  6.5682620
75.7  28.78774 1.696709  2.0889463
89.7  28.78774 1.696709  6.5352599
146.6 28.78774 1.696709 -0.8472456
158.3 28.78774 1.696709 12.5876087
161.1 28.78774 1.696709 -2.4813585
298.3 28.78774 1.696709  9.4580533
```

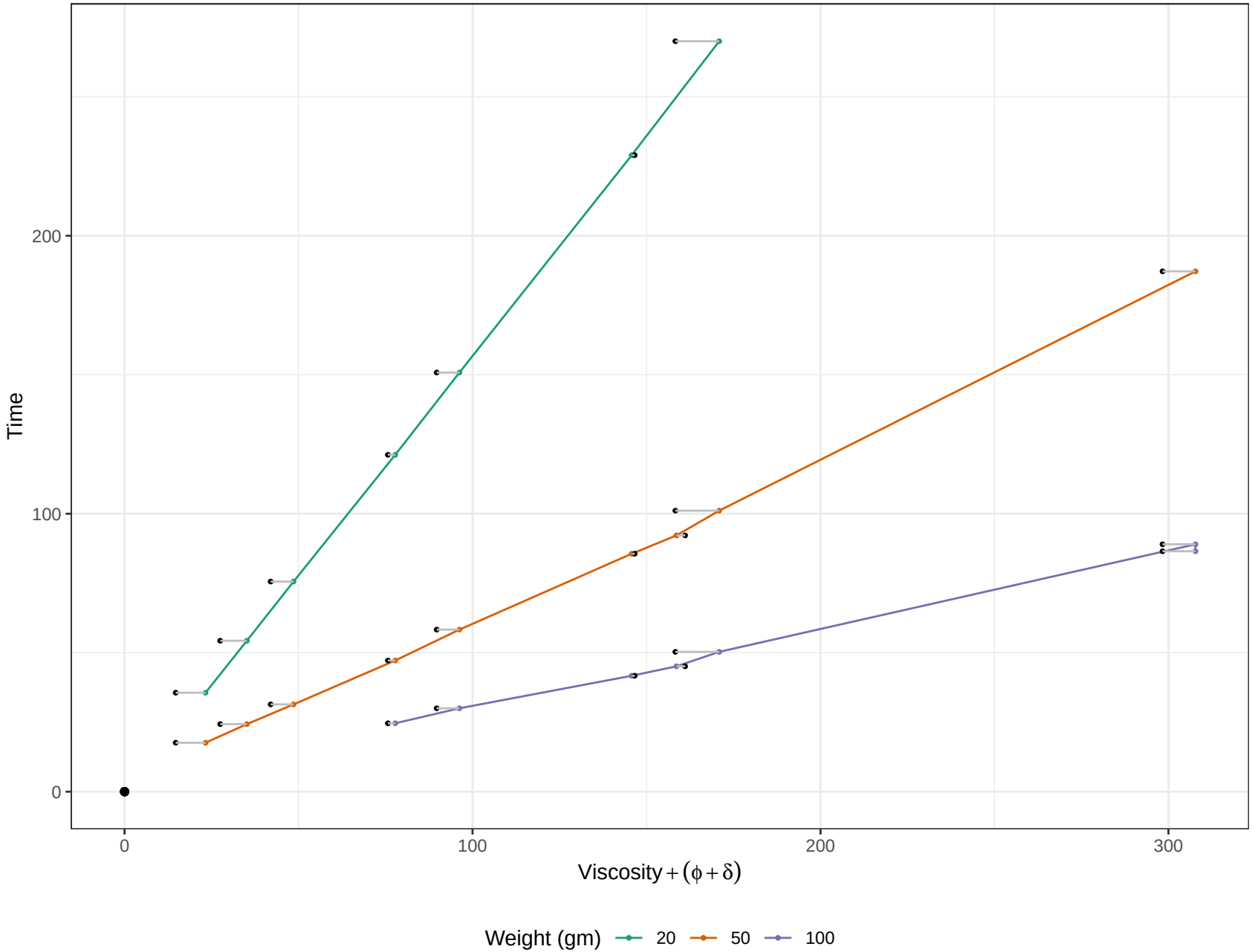
We now adjust the viscosity values by adding on these BLUPs to see to what extent they "straighten out the kinks" in the graphic.

```
phi <- setNames(B[["phi"]], rownames(B))
Stormer <- within(Stormer, {
  Viscosity_phi <- Viscosity + phi[factor(Viscosity)]
})
```

The model has one additional parameter, ϕ, as the random effects, δ, have the logical status of residuals. The model is not realistic, but intended merely to raise possibilities: if there are errors in the viscosities, which one might be in error and by how much?

The diagnostic graphic is below. It shows the magnitude of the adjustments and the effect on the data.

```
p0 <- ggplot(Stormer) + aes(x = Viscosity_phi, y = Time) +
  geom_point(aes(colour = factor(Weight)), size = 0.7) +
  geom_line(aes(colour = factor(Weight))) +
  geom_point(aes(x = Viscosity), size = 0.7) +
  geom_segment(aes(xend = Viscosity, yend = Time), colour = "grey") +
  geom_point(aes(x = 0, y = 0)) +
  xlab(expression(Viscosity + (phi + delta))) +
  scale_colour_brewer(palette = "Dark2", name = "Weight (gm)") +
  theme_bw() + theme(legend.position = "bottom")
p0
```


Time
200
100
0
0
100
200
300
Viscosity + (ϕ + δ)
Weight (gm)
20
50
100

2 Weight loss and exponential growth

The weight loss gives the weight of an obese patient on a medically controlled diet over approximately 9 months. Our goal is to investigate what is the likely future track of the weight of the patient.

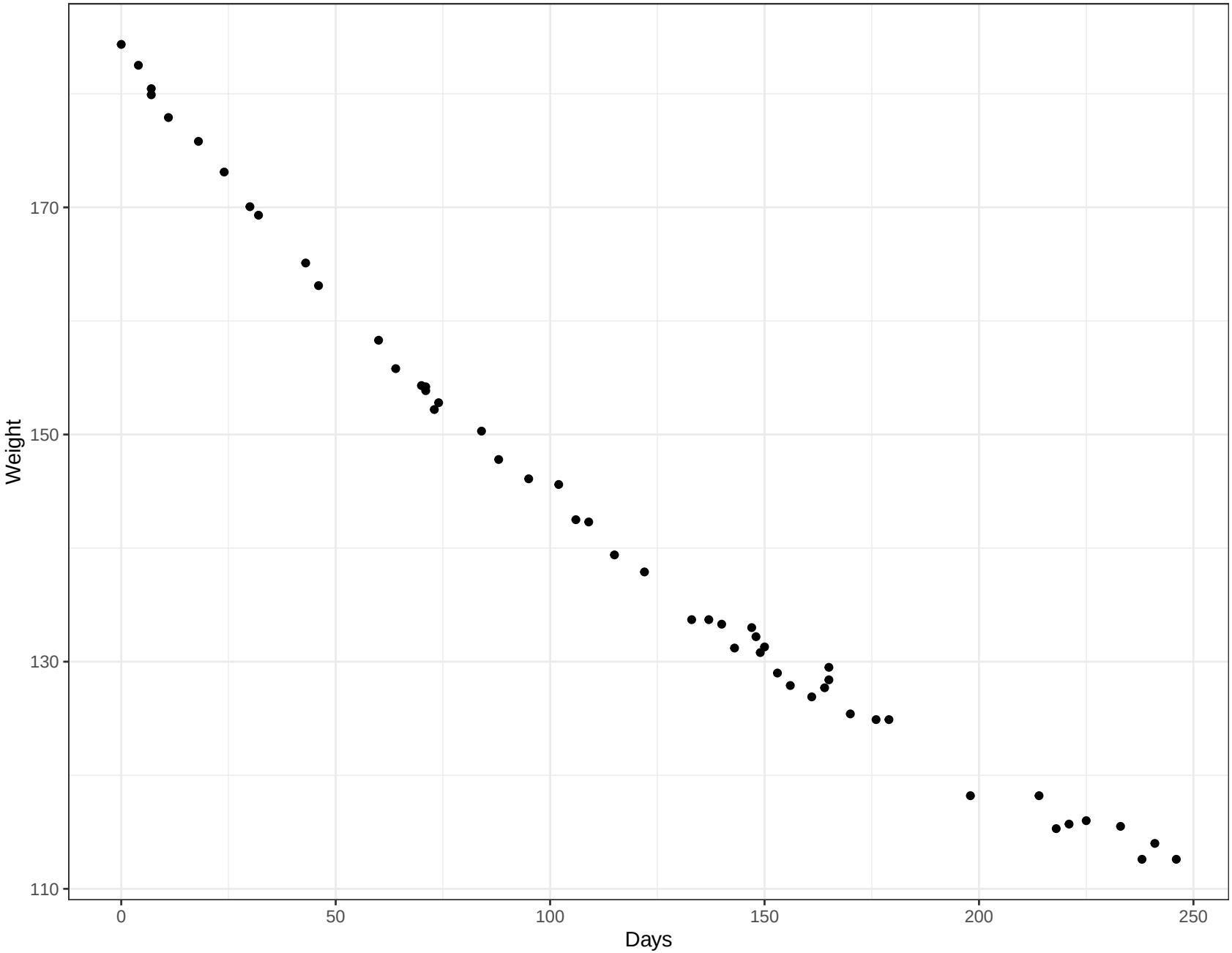
The model we have in mind is a negative exponential growth curve:

$$\text{Weight} = \beta_0 + \beta_1 \exp(-\text{Days}/\theta) + \varepsilon$$

Note that $\beta_0 + \beta_1$ is the weight at day 0 and β_0 is the (projected) ultimate lean weight. So β_1 is the amount of weight to be lost. The parameter θ is related to the "half-life" of the diet: if $\text{Days} = \theta \log 2$ then the mean weight is $\beta_0 + \beta_1 \frac{1}{2}$, i.e. half the 'losable' weight has been lost.

First look at the data:

```
p0 <- ggplot(wtloss) + aes(x = Days, y = Weight) + geom_point() + theme_bw()
p0
```

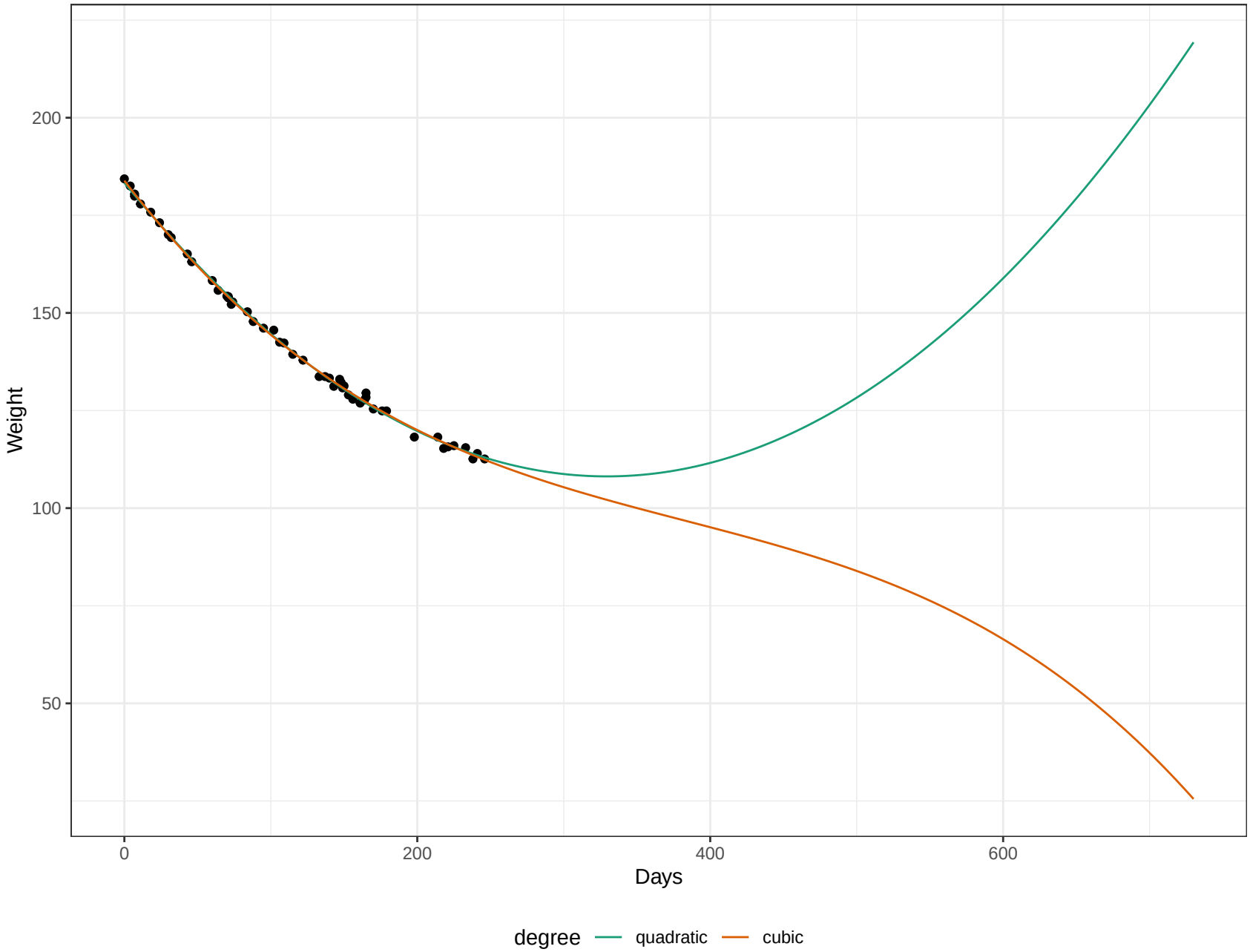

Weight
170
150
130
110
0
50
100
150
200
250
Days

Polynomial models work well within the range of the diet, but fail badly outside.

Bariatric physicians suggest that if a patient can maintain a weight for two years the chances of remaining at the lower weight are much greater.

So let's push it!

```
pWtloss <- data.frame(Days = 0:730)  ## two years
pWtloss <- within(pWtloss, {
  quadratic <- predict(lm(Weight ~ poly(Days, 2), wtloss), pWtloss)
  cubic     <- predict(lm(Weight ~ poly(Days, 3), wtloss), pWtloss)
})
pwt <- pWtloss %>% pivot_longer(cols = c(quadratic, cubic),
                                names_to = "degree",
                                values_to = "Weight")%>%
  mutate(degree = factor(degree, levels = cq(quadratic, cubic))) ## cq() ...

p0 + geom_line(data = pwt, aes(colour = degree)) +
  scale_colour_brewer(palette = "Dark2") +
  theme(legend.position = "bottom")
```


200
150
100
50
Weight
0
200
400
600
Days
degree
quadratic
cubic

Now turn to the non-linear model.

$$\text{Weight} = \beta_0 + \beta_1 \exp\left(-\text{Days}/\theta\right) + \varepsilon$$

To get initial values, one technique is

- Get good estimates of the curve at three equally-spaced time points,
- Equate these to the model to give three equations,
- Solve for the three parameters β_0, β_1 and θ.

For this model the equations can be solved algebraically, which is the idea behind the self-starting model we will define.

Note that if we solve the equations for θ first, the other two parameters are *linear parameters*, and initial values for them can then be found by linear regression.

Further details behind the code are left as a puzzle for the student!

2.1 A self-starting model

Our self-starting model is as follows:

```
SSnegexp <- selfStart(model = ~ b0 + b1*exp(-t/theta),
                      parameters = c("b0", "b1", "theta"),
                      initial = function(mCall, data, LHS, ...) {
                        t <- eval(mCall[["t"]], data)
                        y <- eval(LHS, data)
                        r <- range(t)
                        d <- (r[2] - r[1])/4
                        yf <- predict(lm(y ~ poly(t, 2)),
                                      data.frame(t = r[1] + (1:3)*d))
                        ratio <- (yf[1] - yf[2])/(yf[2] - yf[3])
                        stopifnot(ratio > 0)
                        theta <- d/log(ratio)
                        b <- coef(lm(y ~ 1 + exp(-t/theta)))
                        setNames(c(b, theta), mCall[c("b0", "b1", "theta")])
                      }) %>% fix_deriv()
```

Now we try it out!

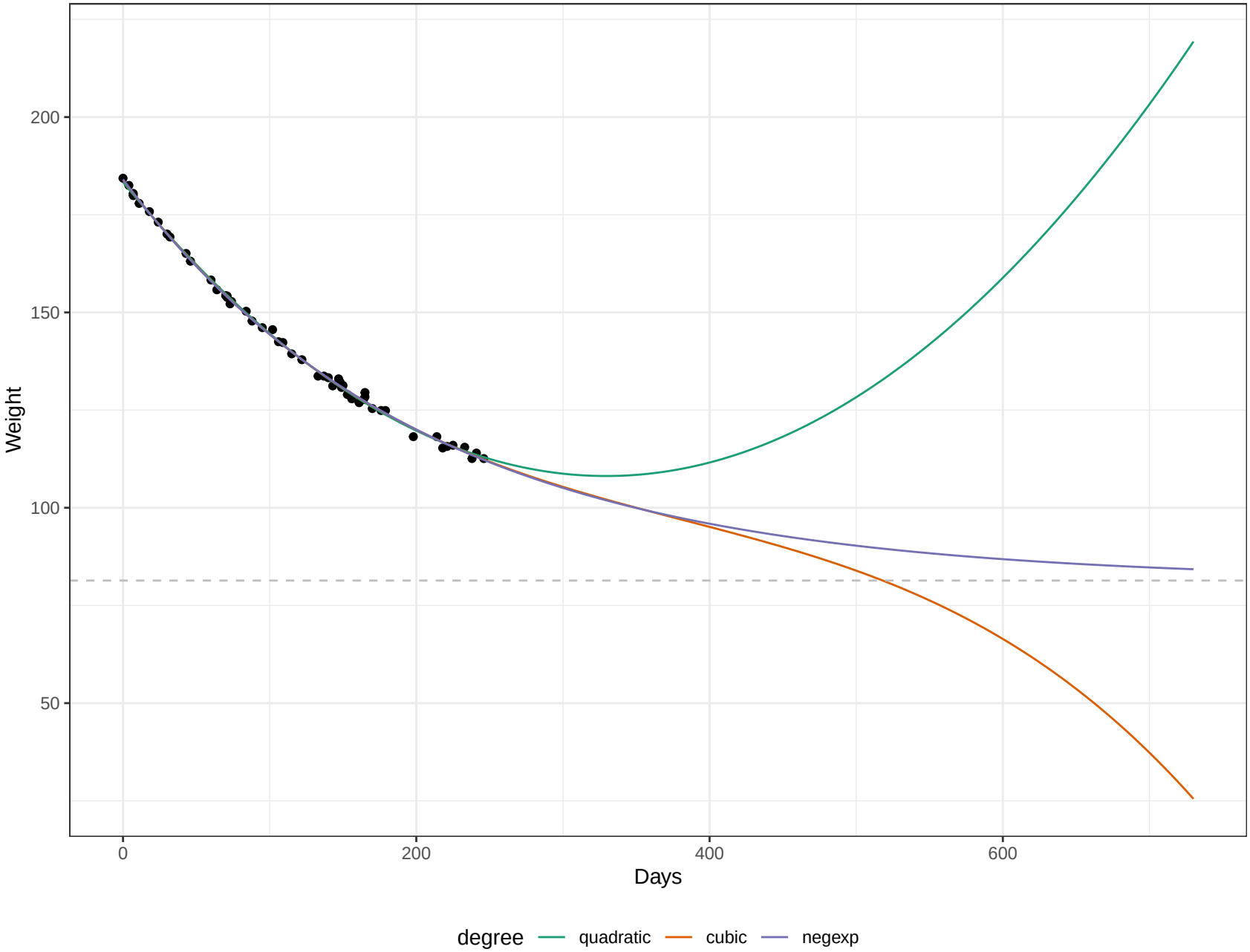
```
wtloss_ne <- nls(Weight ~ SSnegexp(Days, b0, b1, theta), data = wtloss,
                 trace = TRUE)

39.24501 :   81.32989 102.72323 204.88339
39.2447 :   81.3740 102.6840 204.7328
39.2447 :   81.37382 102.68412 204.73338

summary(wtloss_ne)


Formula: Weight ~ SSnegexp(Days, b0, b1, theta)

Parameters:
      Estimate Std. Error t value Pr(>|t|)
b0      81.374      2.269   35.86   <2e-16
b1     102.684      2.083   49.30   <2e-16
theta  204.733      7.638   26.80   <2e-16

Residual standard error: 0.8949 on 49 degrees of freedom

Number of iterations to convergence: 2
Achieved convergence tolerance: 3.898e-08
```

The model estimates imply that

- The stable weight is $\hat{\beta}_0 = 81.37$ kilograms,
- The amount of weight to be lost is $\hat{\beta}_1 = 102.68$ kilograms,
- The half-life of the diet is $\hat{\theta} \times \log 2 = 204.73 \times 0.69 \approx 142$ days. In other words, half the *remaining* weight to be lost, will be lost every 20 weeks (according to the model!).

We now look at how the model compares in extrapolation to the polynomial models:

```
pWtloss$negexp <- predict(wtloss_ne, pWtloss)
pwt <- pWtloss %>% pivot_longer(cols = c(quadratic, cubic, negexp),
                                names_to = "degree",
                                values_to = "Weight") %>%
  mutate(degree = factor(degree, levels = cq(quadratic, cubic, negexp))) ## cq()
p0 + geom_line(data = pwt, aes(colour = degree)) +
  geom_hline(yintercept = coef(wtloss_ne)[["b0"]],colour = "grey",
             linetype = "dashed")+
  scale_colour_brewer(palette = "Dark2") + theme(legend.position = "bottom")
```


200
150
100
50
Weight
0
200
400
600
Days
degree
quadratic
cubic
negexp

2.2 Progressive model fitting

What the patient would like to know is what the model is suggesting progressively as the regime proceeds.

First a few preparatory steps:

```
period <- 28 ## Recalibrate every 4 weeks

W0 <- with(wtloss, Weight[Days == 0])
tab <- cbind(Days           = 0,
             n              = 1,
             Weight         = W0,
             Final          = NA,
             `Yet to lose`  = NA,
             `Half life`    = NA)

end <- with(wtloss, max(Days))  ## Latest data available

full_model <- nls(Weight ~ SSnegexp(Days, W0, w, theta), wtloss)
```

Fit models to the progressive data sets and present the results:

```
days <- 0
while(days < end) {
  days <- min(end, days + period)
  n <- with(wtloss, sum(Days <= days))  ## sample size
  m <- update(full_model, data = filter(wtloss, Days <= days))
  b <- coef(m)
  w <- predict(m, data.frame(Days = days))
  tab <- rbind(tab, unname(c(days, n, w, b[1], w-b[1], log(2)*b[3])))
}
tab <- data.frame(tab, check.names = FALSE) %>%
  within({
    Change <- c(NA, diff(Weight))
  }) %>%
  select(Days, n, Weight, Change, everything())

booktabs(tab, digits = 0) %>% print(include.rownames = FALSE)
```

Days	n	Weight	Change	Final	Yet to lose	Half life
0	1	184				
28	7	172	-12	163	9	22
56	11	160	-12	87	73	135
84	19	149	-10	83	67	140
112	24	141	-8	94	48	121
140	29	133	-8	77	56	150
168	40	127	-6	83	44	139
196	43	121	-6	82	39	140
224	47	115	-6	78	37	149
246	52	112	-3	81	31	142

3 Muscle shortening

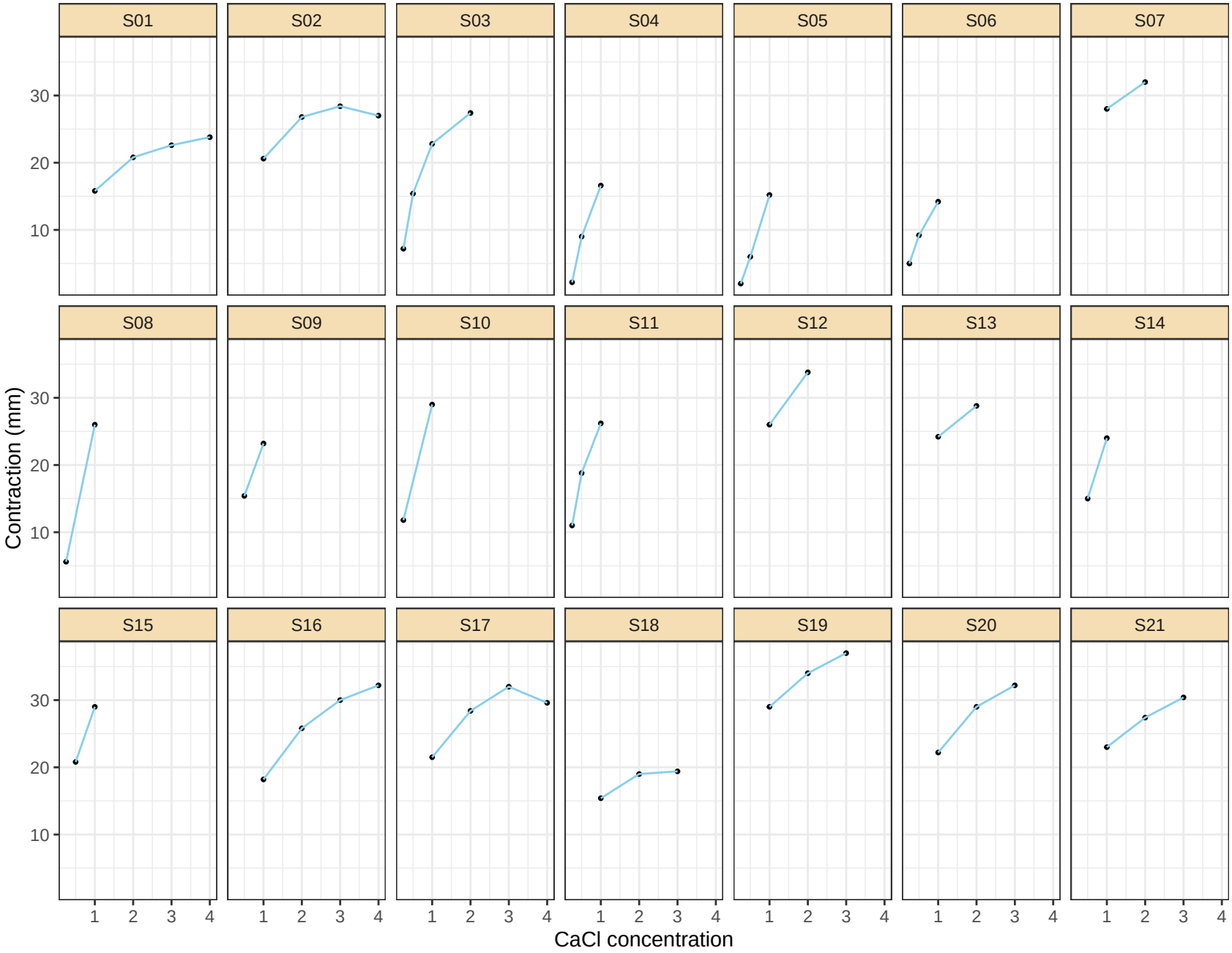
From *?MASS::muscle*:

> The purpose of this experiment was to assess the influence of calcium in solution on the contraction of heart muscle in rats. The left auricle of 21 rat hearts was isolated and on several occasions a constant-length strip of tissue was electrically stimulated and dipped into various concentrations of calcium chloride solution, after which the shortening of the strip was accurately measured as the response.
>
> Linder, A., Chakravarti, I. M. and Vuagnat, P. (1964) Fitting asymptotic regression curves with different asymptotes. In *Contributions to Statistics. Presented to Professor P. C. Mahalanobis on the occasion of his 70th birthday*, ed. C. R. Rao, pp. 221–228. Oxford: Pergamon Press

The data is also provided in `WWRData`. First look at the data

The data are very sparse and fragmentary:

```
p0 <- ggplot(muscle) + aes(x = Conc, y = Length) +
  geom_point(size = 0.7) +
  xlab("CaCl concentration") + ylab("Contraction (mm)") +
  facet_wrap(~ Strip, nrow = 3) +
  theme_bw() +
  theme(strip.background = element_rect(fill = "wheat"))
p0 + geom_line(colour = "sky blue")
```

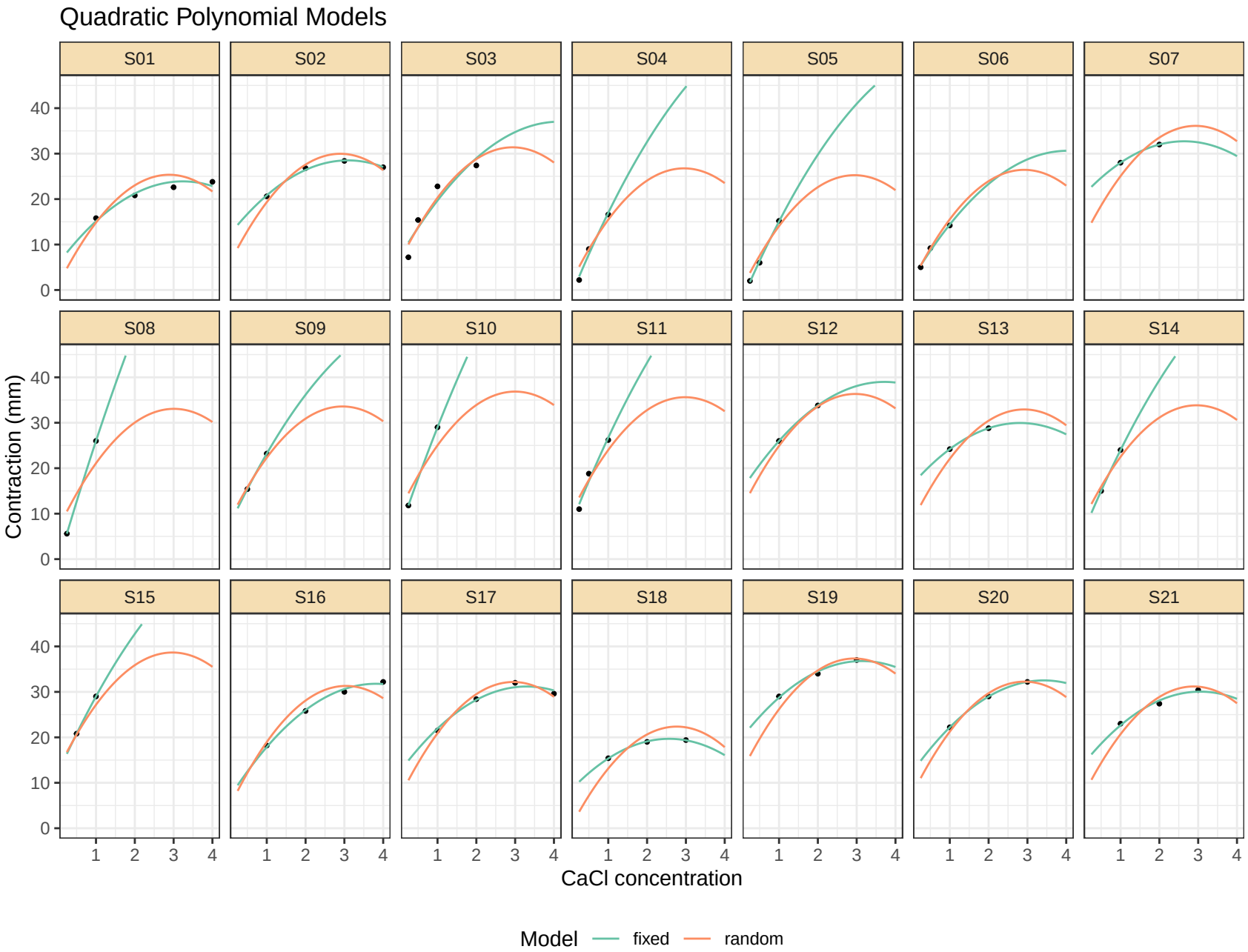

S01
S02
S03
S04
S05
S06
S07
S08
S09
S10
S11
S12
S13
S14
S15
S16
S17
S18
S19
S20
S21
Contraction (mm)
10
20
30
1
2
3
4
CaCl concentration

3.1 Linear models

Before going to asymptotic regressions, consider fitting quadratic polynomials with separate intercepts and slopes, but with a fixed coefficient of x^3. We could do this with *fixed* or *random* coefficients.

The two superficially similar models behave *very* differently!

```
mfix <- lm(Length ~ 0 + Strip/Conc + I(Conc^2/2), data = muscle)
mran <- lmer(Length ~ Conc + I(Conc^2/2) + (1 + Conc|Strip), data = muscle)
pMuscle <- with(muscle, expand.grid(
  Strip = levels(Strip), Conc = seq(min(Conc), max(Conc), length = 100))
)
pMuscle <- within(pMuscle, {
  fixed  <- predict(mfix, pMuscle)
  random <- predict(mran, pMuscle, re.form = ~(1 + Conc|Strip))
})
pms <- pMuscle %>% pivot_longer(cols = c(fixed, random), names_to = "Model",
                                values_to = "Length")
p0 + geom_line(data = pms, aes(colour = Model)) +
  scale_colour_brewer(palette = "Set2") + ylim(0, 45) +
  theme(legend.position = "bottom") + labs(title = "Quadratic Polynomial Models")
```


Quadratic Polynomial Models
S01
S02
S03
S04
S05
S06
S07
S08
S09
S10
S11
S12
S13
S14
S15
S16
S17
S18
S19
S20
S21
Contraction (mm)
0
10
20
30
40
1
2
3
4
CaCl concentration
Model
fixed
random

3.2 Gompertz growth curve model

The suggested model in the original publication is a Gompertz growth curve, with the mean of the form

$$\mu = \exp\left(\beta_0 - \beta_1 \rho^x\right)$$

where typically $0 < \rho < 1$. Writing $\theta = -1/\log \rho$, (so $\rho = e^{-1/\theta}$), which is positive, the model may also be written as

$$\log \mu = \beta_0 - \beta_1 \exp\left(-\frac{x}{\theta}\right)$$

I.e. the model is a negative exponential model in the log scale. This provides a simple way to write an initial value function, and hence a self-starting model.

The sign change, $-\beta_1$, usually means the function will *increase* to an asymptotic value rather than *decrease* as was the case with the weight loss model.

A self-starting Gompertz growth curve model is below. (There is already an *SSgompertz* function in the stats package which uses another parametrisation.)

```
SSgompertz2 <- selfStart(model = ~ exp(b0 - b1*rho^x),
                         parameters = c("b0", "b1", "rho"),
                         initial = function(mCall, data, LHS, ...) {
                           x <- eval(mCall[["x"]], data)
                           y <- eval(LHS, data)
                           r <- range(x)
                           d <- (r[2] - r[1])/4
                           yf <- predict(lm(log(y) ~ poly(x, 2)),
                                         data.frame(x = r[1] + (1:3)*d))
                           ratio <- ((yf[3] - yf[2])/(yf[2] - yf[1]))
                           stopifnot(ratio > 0)
                           rho <- ratio^(1/d)
                           b <- coef(lm(log(y) ~ 1 + I(-rho^x)))
                           setNames(c(b, rho), mCall[c("b0", "b1", "rho")])
                         }) %>% fix_deriv()
```

(This is included in the WWRCourse package as well.)

3.2.1 Fixed and random effect models

We now fit the equivalent of the fixed effect only polynomial model. In this case the ρ parameter will be the same for all *Strip*s, but the parameters β_0 and β_1 will be particular to the strip.

```
m0 <- nls(Length ~ SSgompertz2(Conc, b0, b1, rho), data = muscle) ## Ignore Strip
b <- coef(m0)
init <- list(b0 = rep(b[["b0"]], 21),  ## must be a list for indexed parameters
             b1 = rep(b[["b1"]], 21),
             rho = b[["rho"]])
fgomp <- nls(Length ~ exp(b0[Strip] - b1[Strip]*rho^Conc), data = muscle,
             start = init)
rgomp <- nlmer(Length ~ SSgompertz2(Conc, b0, b1, rho) ~ b0 + b1|Strip,
               data = muscle, start = coef(m0))
```

Prediction is easy for the fixed effect model:

```
pMuscle <- within(pMuscle, {
  fixed <- predict(fgomp, pMuscle)  ## replaces the previous component
})
```

But the random effect model requires some work "by hand".

First look at fixed and random effects:

```
fixef(rgomp)

       b0        b1       rho
3.3837000 2.3874884 0.1251406

ranef(rgomp)$Strip

               b0          b1
S01 -0.245936474  0.61555165
S02 -0.054992932  0.09811076
S03 -0.002929187 -0.24580502
S04 -0.180944693  0.65433264
....
S17  0.024607303  0.09332962
S18 -0.345450248  0.67900129
S19  0.201775909 -0.53959596
S20  0.047235282  0.04036690
S21  0.009873066 -0.11066725
```

The coefficients (= “parameters” for non-linear models) combine the two:

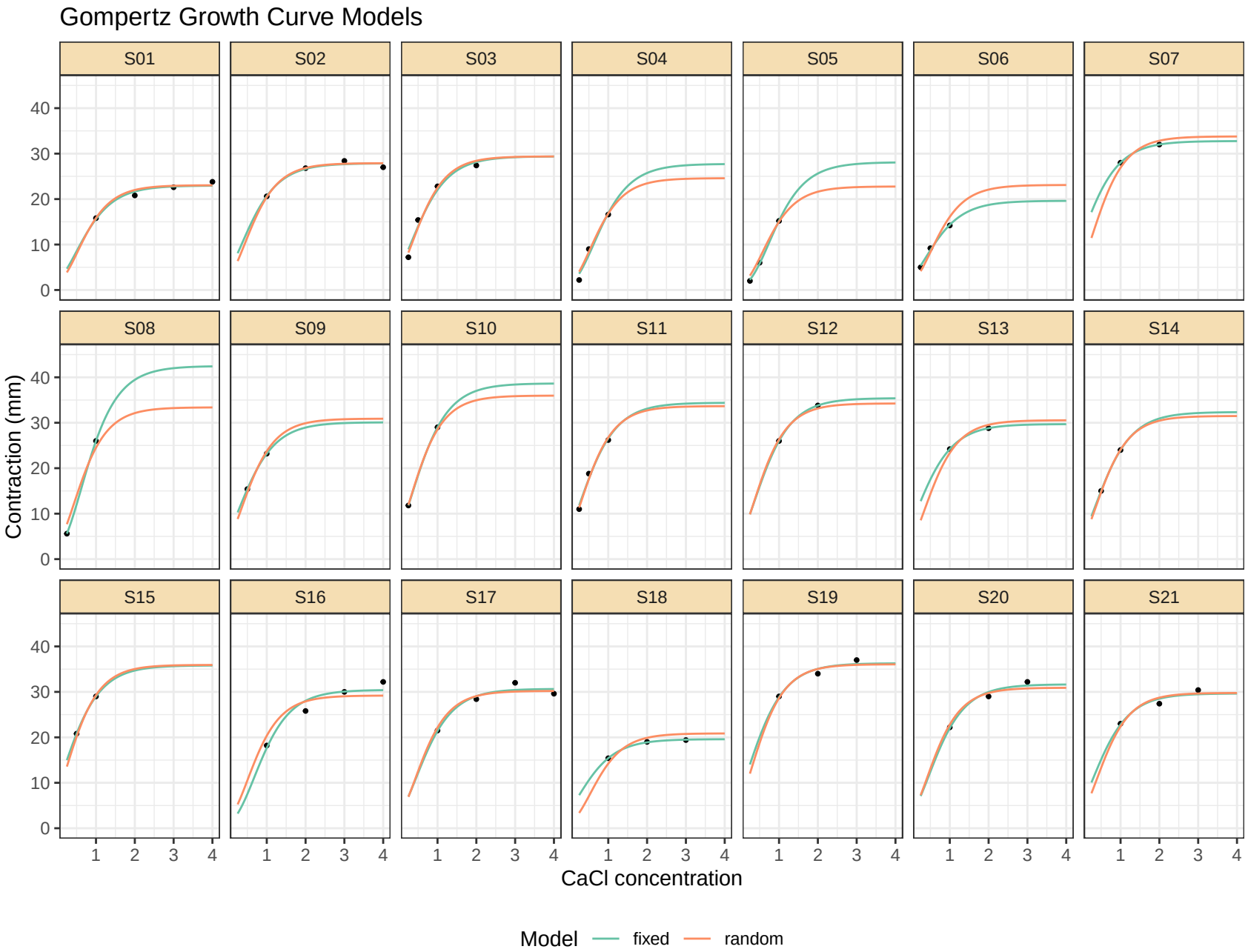
```
(B <- coef(rgomp)$Strip)

           b0       b1       rho
S01 3.137764 3.003040 0.1251406
S02 3.328707 2.485599 0.1251406
....
S20 3.430935 2.427855 0.1251406
S21 3.393573 2.276821 0.1251406
```

Make a data frame extending this with the *Strip* labels included and merge it with *pMuscle*. Then complete the computation by hand:

```
B <- B %>% rownames_to_column("Strip") %>% untibble()
pMuscle <- pMuscle %>% left_join(B, by = "Strip") %>%
  mutate(random = exp(b0 - b1*rho^Conc), Strip = factor(Strip)) %>%
  select(Strip, Conc, fixed, random)
```

Now we can repeat the same display as with the polynomial models.

```
pms <- pMuscle %>% pivot_longer(cols = c(fixed, random), names_to = "Model",
                                values_to = "Length")
p0 + geom_line(data = pms, aes(colour = Model)) +
  scale_colour_brewer(palette = "Set2") + ylim(0, 45) +
  theme(legend.position = "bottom") + labs(title = "Gompertz Growth Curve Models")
```

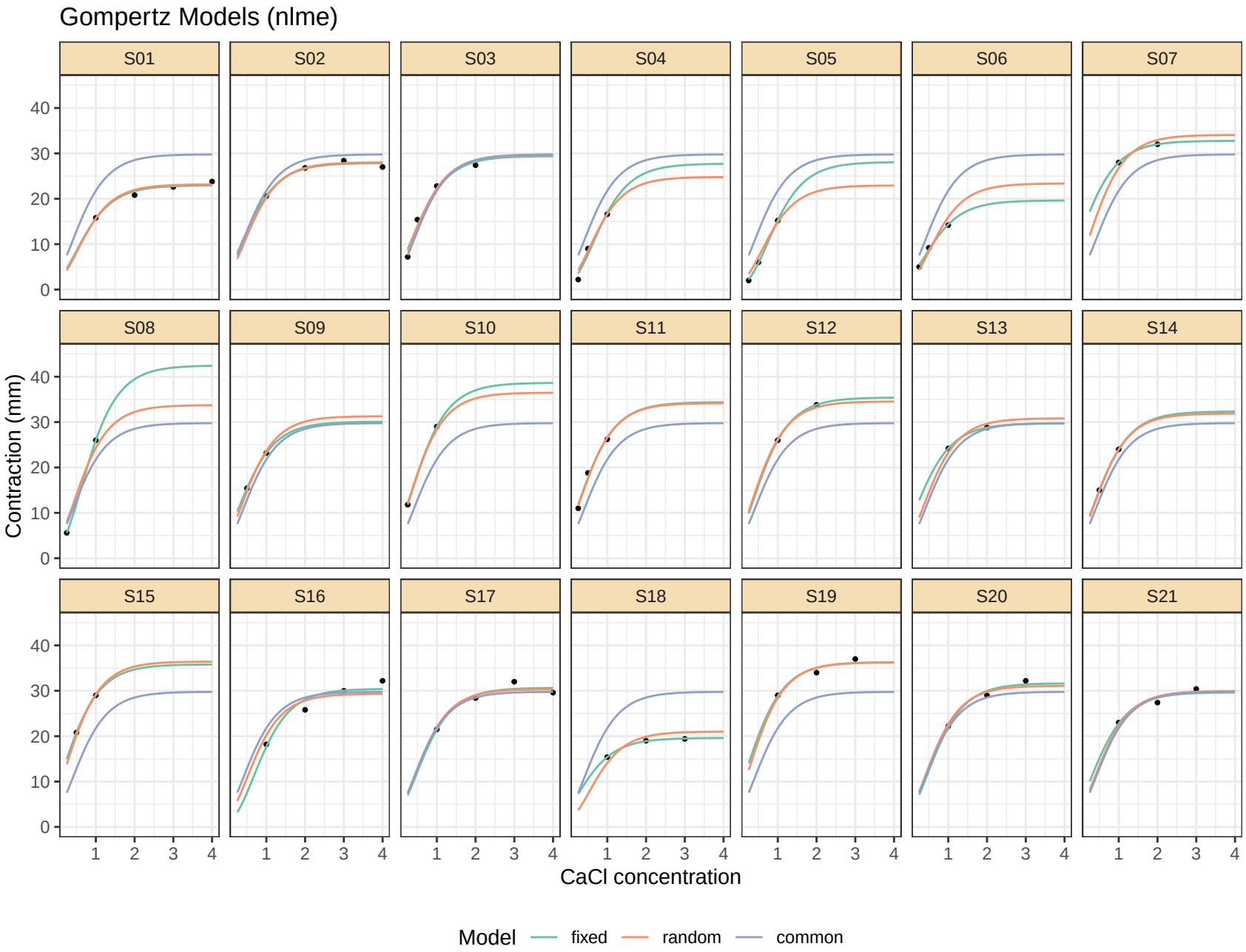

Gompertz Growth Curve Models
S01
S02
S03
S04
S05
S06
S07
S08
S09
S10
S11
S12
S13
S14
S15
S16
S17
S18
S19
S20
S21
Contraction (mm)
0
10
20
30
40
1
2
3
4
CaCl concentration
Model
fixed
random

3.3 Epilogue

- The relationship was known to be asymptotic, so the polynomial models are at best a first step. It is clear that the "shrinkage" effect of the random effect model has conferred more stability on the result even for these models.
- The green curves (fixed model) in the final display correspond to the model fitted *by hand* in the original 1964 article. Forcing the relationship to be asymptotic has clearly given them some stability. (The non-linear fitting technology was not available in 1964, of course!)
- Random effect models are often claimed to be very effective for handling sparse or "scrappy" data. There seems to be good empirical evidence for this.
- An older package, *nlme*, (which is part of core **R**) has some pro's and con's relative to *lmer*. It has a different calling interface, much more complicated than that of *lmer*, but offers some convenience functions

as well. This includes codepredict methods for non-linear models. The code to fit our Gompertz random effect model and predict is given below

```
suppressPackageStartupMessages({
  library(nlme)
})
rgomp <- nlme(Length ~ SSgompertz2(Conc, b0, b1, rho), data = muscle,
              fixed = b0 + b1 + rho ~ 1, random = b0 + b1 ~ 1|Strip,
              start = coef(m0))
pMuscle <- within(pMuscle, {
  fixed  <- predict(fgomp, pMuscle)  ## replaces the component (not needed!)
  random <- predict(rgomp, pMuscle, level = 1)  ## replaces the previous component
  common <- predict(rgomp, pMuscle, level = 0)  ## same for all strips (fixef)
})
pms <- pMuscle %>% pivot_longer(cols = c(fixed, random, common),
                                names_to = "Model", values_to = "Length") %>%
  mutate(Model = factor(Model, levels = cq(fixed, random, common)))
p0 + geom_line(data = pms, aes(colour = Model)) +
  scale_colour_brewer(palette = "Set2") + ylim(0, 45) +
  theme(legend.position = "bottom") + labs(title = "Gompertz Models (nlme)")
```


Gompertz Models (nlme)
S01
S02
S03
S04
S05
S06
S07
S08
S09
S10
S11
S12
S13
S14
S15
S16
S17
S18
S19
S20
S21
Contraction (mm)
CaCl concentration
Model
fixed
random
common

Session information

Date: 2021-01-29

- R version 4.0.3 (2020-10-10), `x86_64-pc-linux-gnu`
- Running under: `Ubuntu 20.04.1 LTS`
- Matrix products: default
- BLAS: `/usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0`
- LAPACK: `/usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0`
- Base packages: base, datasets, graphics, grDevices, methods, stats, utils
- Other packages: dplyr 1.0.3, english 1.2-5, forcats 0.5.1, ggplot2 3.3.3, ggthemes 4.2.4, gridExtra 2.3, knitr 1.31, lattice 0.20-41, lme4 1.1-26, Matrix 1.3-2, nlme 3.1-151, patchwork 1.1.1, purrr 0.3.4, readr 1.4.0, scales 1.1.1, stringr 1.4.0, tibble 3.0.5, tidyr 1.1.2, tidyverse 1.3.0, WWRCourse 0.2.3, WWRData 0.1.0, WWRGraphics 0.1.2, WWRUtilities 0.1.2, xtable 1.8-4
- Loaded via a namespace (and not attached): assertthat 0.2.1, backports 1.2.1, boot 1.3-26, broom 0.7.3, cellranger 1.1.0, cli 2.2.0, colorspace 2.0-0, compiler 4.0.3, crayon 1.3.4, DBI 1.1.1, dbplyr 2.0.0, digest 0.6.27, ellipsis 0.3.1, evaluate 0.14, fansi 0.4.2, farver 2.0.3, fractional 0.1.3, fs 1.5.0, generics 0.1.0, glue 1.4.2, grid 4.0.3, gtable 0.3.0, haven 2.3.1, highr 0.8, hms 1.0.0, httr 1.4.2, iterators 1.0.13, jsonlite 1.7.2, labeling 0.4.2, lazyData 1.1.0, lifecycle 0.2.0, lubridate 1.7.9.2, magrittr 2.0.1, MASS 7.3-53, minqa 1.2.4, modelr 0.1.8, munsell 0.5.0, nloptr 1.2.2.2, parallel 4.0.3, PBSmapping 2.73.0, pillar 1.4.7, pkgconfig 2.0.3, R6 2.5.0, randomForest 4.6-14, RColorBrewer 1.1-2, Rcpp 1.0.6, readxl 1.3.1, reprex 1.0.0, rlang 0.4.10, rpart 4.1-15, rstudioapi 0.13, rvest 0.3.6, SOAR 0.99-11, splines 4.0.3, statmod 1.4.35, stringi 1.5.3, tidyselect 1.1.0, tools 4.0.3, vctrs 0.3.6, withr 2.4.1, xfun 0.20, xml2 1.3.2