Non-linear least Squares Fitting in Ecology and Evolution

Introduction and preliminaries

This chapter assumes that you have already seen the NLLS lecture (https://github.com/mhasoba/TheMulQuaBio/blob/master/Lectures/NLLS/Pawar_NLLS.pdf) and/or its recording (https://imperial.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=2aa5bbb4-e1d0-44a1-adea-a87000aab72f).

We will work with several NLLS examples here.

> *You should build a separate R scripts for each of these examples using the principles of good coding you learnt in the R week*. In particular, create separate `code`, `data`, `results` directories, and `setwd()` to `code`.
>
> For example, for the Allometry example below, create a new R script file called `Allometry_nlls.R` and save it to your `code` directory. Then build your script for tackling this particular example in that file.

You may work in RStudio or any other Code editor you prefer.

You will need the `nls.lm` R package, which you can install using the standard method (linux users, launch R in `sudo` mode first):

```
> install.packages("minpack.lm")
```

Now, load the necessary packages, and clear all variables and graphic devices:

```
In [2]:
```

```
rm(list = ls())
graphics.off()

library("minpack.lm") # for Levenberg-Marquardt nls fitting
library("ggplot2")
```

You will need to repeat these commands at the start of every new nlls script file you develop below.

Why use the nls.lm package?

The standard NLLS function in R, cals `nls` uses a less robust algorithm called the Gauss-Newton algorithm. Therefore, `nls` will often fail to fit your model to the data if you start off at starting values for the parameters that are too far from what the optimal vaues would be, especially if the "parameter space" is weirdly shaped, i.e., the model has a mathematical form that makes it hard to find parameter combinations that minimize the NLLS. If this does not makes ense, don't worry about it- just go with `nls_LM` from the `nls.lm` package instead of `nls`! If you are really curious, try substituting `nls` for `nls_LM` in the examples below and compare the results.

Allometric scaling

Let's start with a common and reasonably simple example from biology: allometric scaling (https://en.wikipedia.org/wiki/Allometry). We will look at allometric scaling of body weight vs. total body length in dragonflies and damselfiles.

Allometric relationships take the form:

$$y = ax^b$$

where x and y are morphological measures (body length and body weight respectively, in our current example), the constant is the value of y at body length $x = 1$ unit, and b is the scaling "exponent". This is also called a power-law, because y relates to x through a simple power.

First create a function object for the power law model:

In [3]:

```
powMod <- function(x, a, b) {
    return(a * x^b)
}
```

Now read in the data (https://raw.githubusercontent.com/mhasoba/TheMulQuaBio/master/Data/GenomeSize.csv) (first click on link and use "Save as" or `Ctrl+S` to download it as a csv):

In [4]:

```
MyData <- read.csv("../Data/GenomeSize.csv")

head(MyData)
```

Suborder	Family	Species	GenomeSize	GenomeSE	GenomeN	BodyWeight	TotalLength	HeadLength	Thorax
Anisoptera	Aeshnidae	Aeshna canadensis	2.20	NA	1	0.159	67.58	6.83	11.81
Anisoptera	Aeshnidae	Aeshna constricta	1.76	0.06	4	0.228	71.97	6.84	10.72
Anisoptera	Aeshnidae	Aeshna eremita	1.85	NA	1	0.312	78.80	6.27	16.19
Anisoptera	Aeshnidae	Aeshna tuberculifera	1.78	0.10	2	0.218	72.44	6.62	12.53
Anisoptera	Aeshnidae	Aeshna umbrosa	2.00	NA	1	0.207	73.05	4.92	11.11
Anisoptera	Aeshnidae	Aeshna verticalis	1.59	NA	1	0.220	66.25	6.48	11.64

Anisoptera (https://en.wikipedia.org/wiki/Dragonfly) are dragonflies, and Zygoptera (https://en.wikipedia.org/wiki/Damselfly) are Damselflies. The variables of interest are `BodyWeight` and `TotalLength`. Let's use the dragonflies data subset.

So subset the data accordingly and remove NAs:

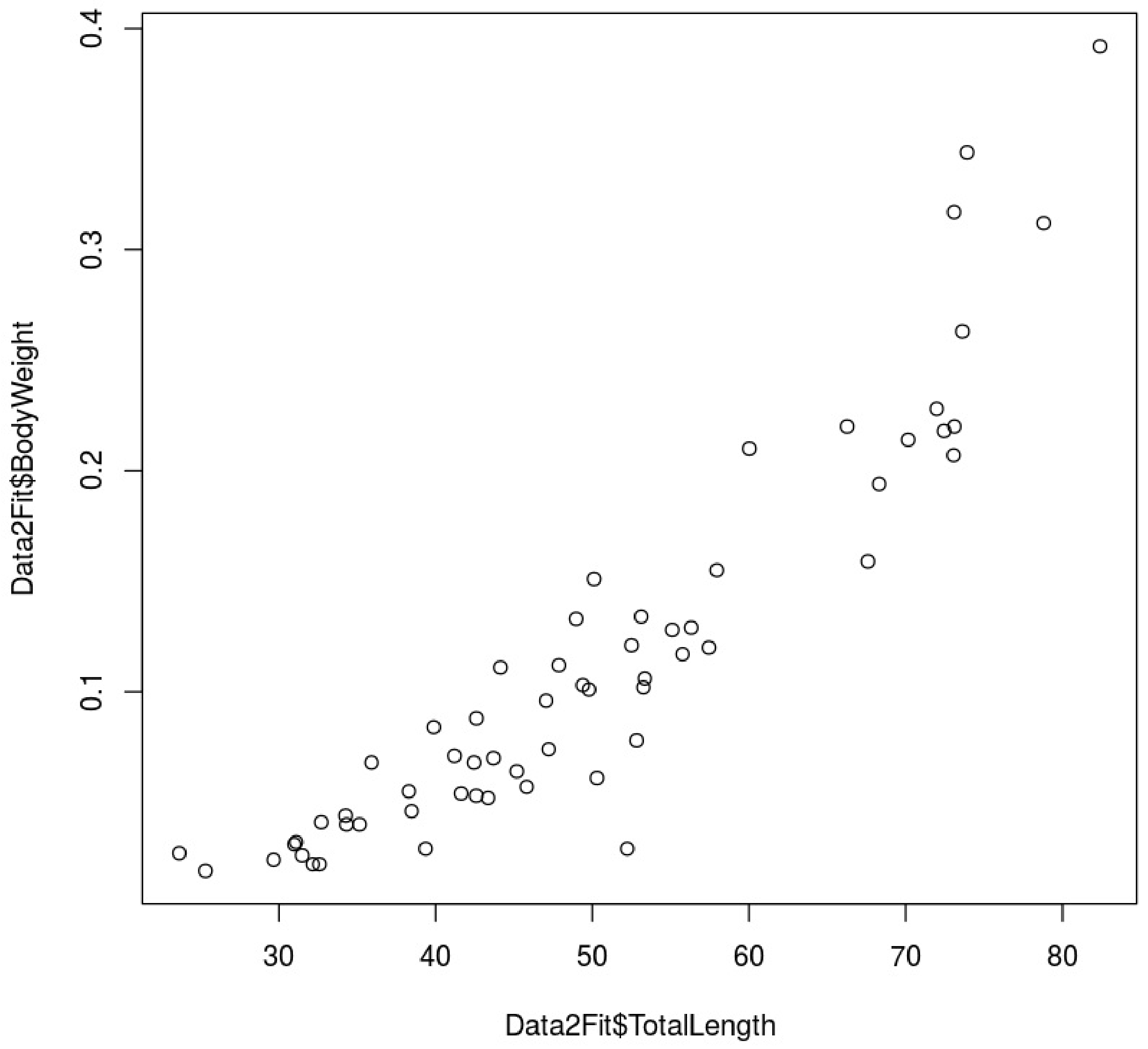
In [5]:

```
Data2Fit <- subset(MyData,Suborder == "Anisoptera")

Data2Fit <- Data2Fit[!is.na(Data2Fit$TotalLength),] # remove NA's
```

Plot it:

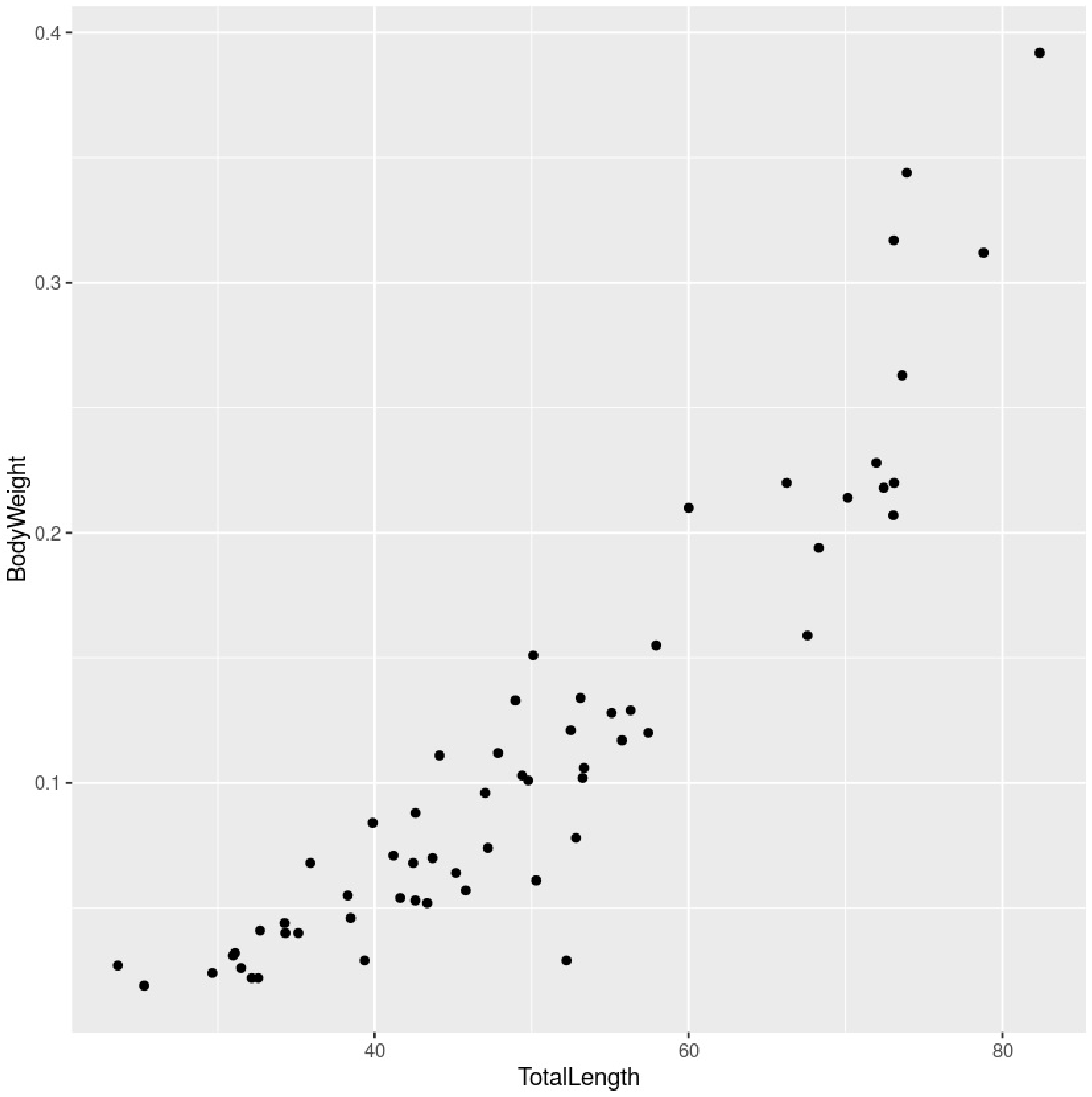
In [6]:

```
plot(Data2Fit$TotalLength, Data2Fit$BodyWeight)
```



In [7]:

```
ggplot(Data2Fit, aes(x = TotalLength, y = BodyWeight)) + geom_point() # or using ggplot!
```



Now fit the model to the data using NLLS:

In [8]:

```
PowFit <- nlsLM(BodyWeight ~ powMod(TotalLength, a, b), data = Data2Fit, start = list(a = .1, b = .1))
```

We can use `summary()` just like we would for a `lm()` fit object.

In [9]:

```
summary(PowFit)
```

```
Formula: BodyWeight ~ powMod(TotalLength, a, b)

Parameters:
   Estimate Std. Error t value Pr(>|t|)
a 3.941e-06  2.234e-06   1.764    0.083 .
b 2.585e+00  1.348e-01  19.174   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.02807 on 58 degrees of freedom

Number of iterations to convergence: 39
Achieved convergence tolerance: 1.49e-08
```

Most of the output is analogous to the output of an `lm()`. However, further statistucal inference here cannot be done using Analysis of Variance (ANOVA) (), because the mdoel is not a Linear Model. Try `anova(PowFit)`, and see what happens. The `Number of iterations to convergence: 39`, and `Achieved convergence tolerance: 1.49e-08` stem from the fact that NLLS requires computer simulations; revisit the Lecture (https://github.com/mhasoba/TheMulQuaBio/blob/master/Lectures/NLLS/Pawar_NLLS.pdf) for an explanation of this.

Now let's visualize the fit. For this, first we need to generate a vector of body lengths (the x-axis variable) for plotting:

In [10]:

```
Lengths <- seq(min(Data2Fit$TotalLength),max(Data2Fit$TotalLength),len=200)
```

Next, calculate the predicted line. For this, we will need to extract the coefficient from the model fit object using the `coef()` command.

In [11]:

```
coef(PowFit)["a"]
coef(PowFit)["b"]
```

a: 3.94068491030517e-06

b: 2.58504796772587

So, we can do the following:

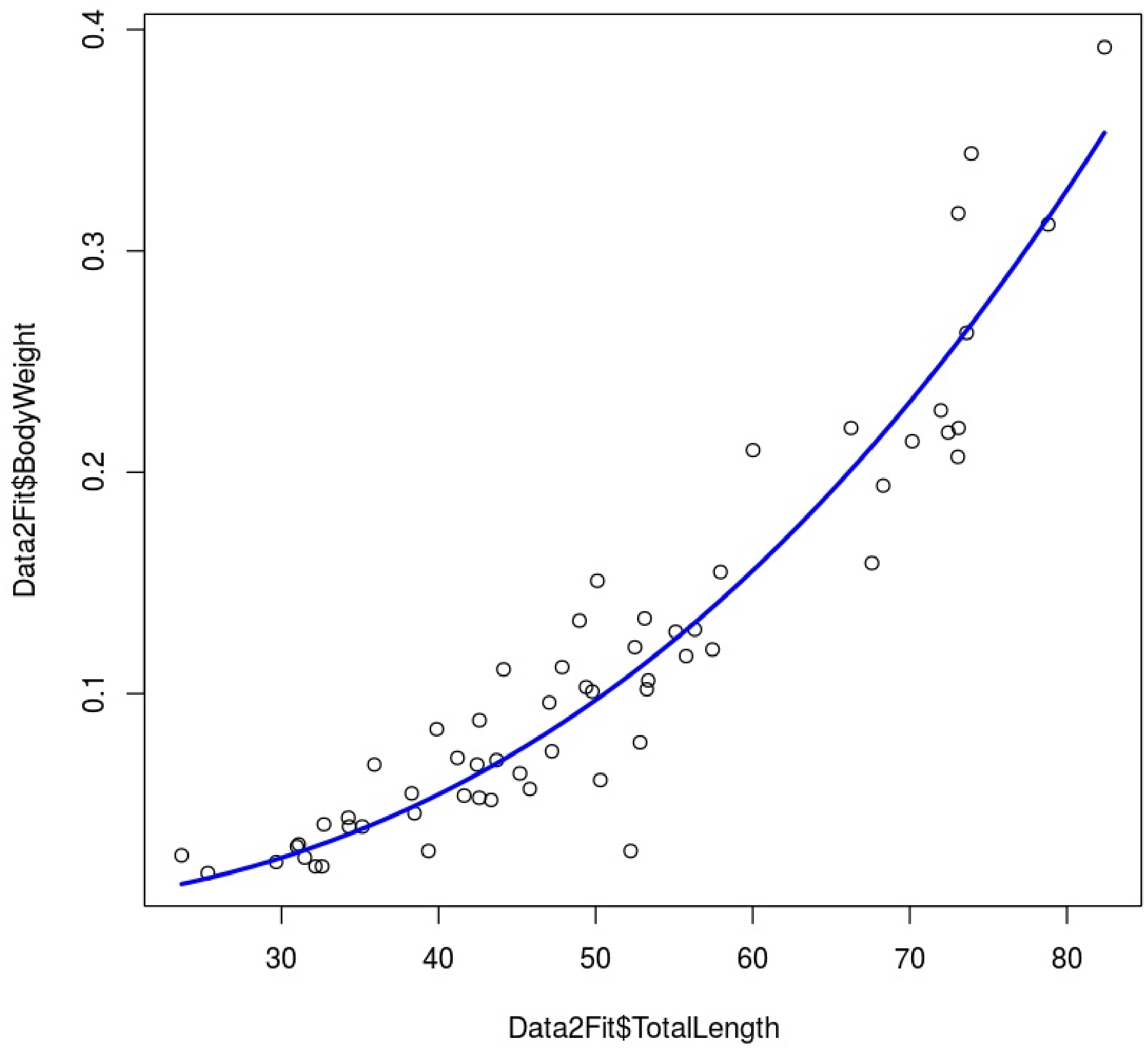
In [12]:

```
Predic2PlotPow <- powMod(Lengths,coef(PowFit)["a"],coef(PowFit)["b"])
```

Now plot the data and the fitted model line:

In [13]:

```
plot(Data2Fit$TotalLength, Data2Fit$BodyWeight)
lines(Lengths, Predic2PlotPow, col = 'blue', lwd = 2.5)
```



We can claculate the confidence intervals on the estimated parameters as we would in OLS fitting used for Linear Models:

In [14]:

```
confint(PowFit)
```

```
Waiting for profiling to be done...
```

	2.5%	**97.5%**
a	1.171935e-06	1.205273e-05
b	2.318292e+00	2.872287e+00

As you learnt before, a coefficient's CI should not include zero for it to be statistically significant (different from zero).

Exercises

(a) Make the same plot as above, fitted line and all, in `ggplot`, and add (display) the equation you estimated to your new (ggplot) plot. The equation is: $\text{Weight} = 3.94 \times 10^{-06} \times \text{Length}^{2.59}$

(b) Try playing with the starting values, and see if you can "break" the model fit -- that is the NLLS fitting does not converge on a solution.

(c) Repeat the model fitting (incuding a-b above) using the Zygoptera data subset.

(d) There is an alternative (and in fact, more commonly-used) approach for fitting the allometric model to data: using Oridinary Least Squares on bi-logarithamically transformed data. That is, if you take a log of both sides of the allometric equation we get,

$$\log(y) = \log(a) + b\log(x)$$

This is a straight line equation of the form $c = d + bz$, where $c = \log(c)$, $d = \log(a)$, $z = \log(x)$, and b is now the slope parameter. So you can use Ordinary Least Squares and the linear models framework (with `lm()`) in R to estimate the parameters of the allometric equation.

In this exercise, try comparing the NLLS vs OLS methods to se how much difference you get in the parameter estimates between them. For example, see the methods used in this paper by Cohen et al 2012 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3465447/).

(e) The allometry between Body weight and Length is not the end of the story, you have a number of other linear morphological measurements (`HeadLength`, `ThoraxLength`, `AdbdomenLength`, `ForewingLength`, `HindwingLength`, `ForewingArea`, and `HindwingArea`) that can also be investigated. In this exercise, you will try two lines of investigation (again, repeated separately for Dragonflies and Damselfiles):

```
(i) How do each of these measurs allometrically scale with Body length (obtain estimates of scaling co
nstant and exponent)? (Hint: you may want to use the `pairs()` command in R to get an overview of all
the pairs of potential scaling relationships.

(ii) Do any of the linear morphological measurements other than body length better predict Body weight
? That is, does body weight scale more tightly with a linear morphological measurement other than tota
l body length?
```

Comparing two models

How do we know that there isn't a better or alternative model that adequately explains the pattern in your dataset?

This is important consideration in all data analyses (and more generally, the scientific method!), so you must aim to compare your NLLS model with an one or more alternatives for a more extensive and reliable investigation of the problem.

Let's use model comparison to investigate whether the relationship between body weight and length we found above is indeed allometric. For this, we need an alternative model that can be fitted to the same data. Let's try a quadratic curve, which is of the form:

$$y = a + bx + cx^2$$

This can also capture curvature in data, just as the allometric equation. Note that this mode is linear in its parameters (a linear model), which You can fit to the simply data using your favorite `lm()` function:

In [15]:

```
QuaFit <- lm(BodyWeight ~ poly(TotalLength,2), data = Data2Fit)
```

And like before, we obtain the predicted values (but this time using the `predict.lm` function):

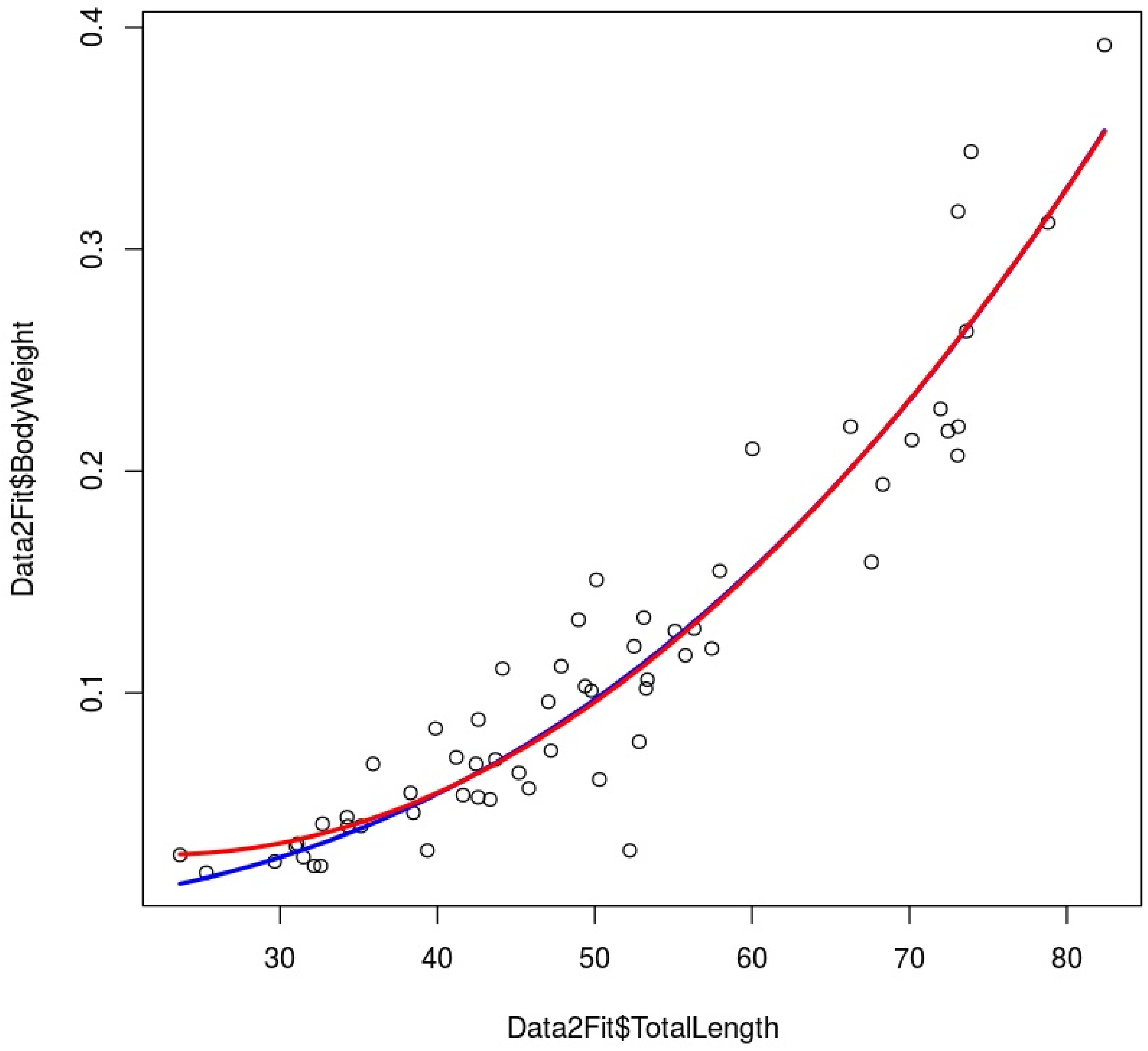
In [16]:

```
Predic2PlotQua <- predict.lm(QuaFit, data.frame(TotalLength = Lengths))
```

Now let's plot the two fitted models together:

In [17]:

```
plot(Data2Fit$TotalLength, Data2Fit$BodyWeight)
lines(Lengths, Predic2PlotPow, col = 'blue', lwd = 2.5)
lines(Lengths, Predic2PlotQua, col = 'red', lwd = 2.5)
```



Very similar fits, except that the quadratic model seems to get it wrong at the lower end of the data range. Let's do a proper/formal model comparison now to check which model better-fits the data.

Let's first calculate the R^2 values of the two fitted models:

In [18]:

```
RSS_Pow <- sum(residuals(PowFit)^2)  # Residual sum of squares
TSS_Pow <- sum((Data2Fit$BodyWeight - mean(Data2Fit$BodyWeight))^2)  # Total sum of squares
RSq_Pow <- 1 - (RSS_Pow/TSS_Pow)  # R-squared value

RSS_Qua <- sum(residuals(QuaFit)^2)  # Residual sum of squares
TSS_Qua <- sum((Data2Fit$BodyWeight - mean(Data2Fit$BodyWeight))^2)  # Total sum of squares
RSq_Qua <- 1 - (RSS_Qua/TSS_Qua)  # R-squared value

RSq_Pow
RSq_Qua
```

```
0.90054752976309

0.900302864503218
```

Not very useful. In general, R^2 is a good measure of model fit, but cannot be used for model selection -- epecially not here, given the tiby difference in the R^2's.

Instead, as discussed in the lecture, we can use the Akaike Information Criterion (AIC):

In [19]:

```
n <- nrow(Data2Fit) #set sample size
kPow <- length(coef(PowFit)) # get number of parameters in power law model
kQua <- length(coef(QuaFit)) # get number of parameters in quadratic model

AIC_Pow <- n * log((2 * pi) / n) + n + 2 + n * log(RSS_Pow) + 2 * kPow
AIC_Qua <- n * log((2 * pi) / n) + n + 2 + n * log(RSS_Qua) + 2 * kQua
AIC_Pow - AIC_Qua
```

```
-2.1474260812509
```

Of course, as you might have suspected, we can do this using an in-built function in R!

In [20]:

```
AIC(PowFit) - AIC(QuaFit)
```

```
-2.1474260812509
```

So which model wins? As we had dicussed in the NLLS lecture, a rule of thumb is that a AIC value difference (typically denoted as ΔAIC) > 2 is a acceptable cutoff for calling a winner. So the power law (allometric model) is a better fit here. Read the Johnson & Omland paper (https://github.com/mhasoba/TheMulQuaBio/blob/master/Readings/Modelling/JohnsonOmland2004.pdf) for more on model selection in Ecology and Evolution.

Exercises

(a) Calculate the Bayesian Information Criterion (BIC), also know as the Schwarz Criterion (see your Lecture notes and the Johnson & Omland paper (https://github.com/mhasoba/TheMulQuaBio/blob/master/Readings/Modelling/JohnsonOmland2004.pdf), and use ΔBIC to select the better fitting model.

(b) Fit a straight line to the same data and compare with the allometric and quadratic models.

(c) Repeat the model comparison (incuding 1-2 above) using the Damselflies (Zygoptera) data subset -- does the allometric mdoel still win?

(d) Repeat exercise (e)(i) and (ii) from the above set, biut with model comparison (e.g., again using a quadratic as a alternative model) to establish that the relationships are indeed allometric.

Fitting growth rate models

Let's up the ante and tackle a more complicated NLLS problem - Fitting population growth rates to data. In general, a population grows exponentially while its abundance is low and resources are not limiting. This growth then slows and eventually stops as resources become limiting. There may also be a time lag before the population growth really takes off at the start.

We will work with some Bacterial growth data that Tom Smith (https://mhasoba.pythonanywhere.com/pawarlab/default/people), a PhD student at Silwood, has been generating as part of his Dissertation research. Bacterial growth in batch culture follows a distinct set of phases; lag phase, exponential phase and stationary phase. During the lag phase a suite of transcriptional machinery is activated, including genes involved in nutrient uptake and metabolic changes, as bacteria prepare for growth. During the exponential growth phase, bacteria divide at a constant rate, the population doubling with each generation. When the carrying capacity of the media is reached, growth slows and the number of cells in the culture stabilises, beginning the stationary phase.

Traditionally, growth rate can be measured by plotting out cell numbers or culture density against time on a semi-log graph and fitting a straight line through the exponential growth phase - the slope of the line gives the maximum growth rate (r_{max}). Models have since been devised which we can use to describe the whole sigmoidal bacterial growth curve.

Let's first generate some "data" on the number of bacterial cells as a function of time taht we can play with:

In [21]:

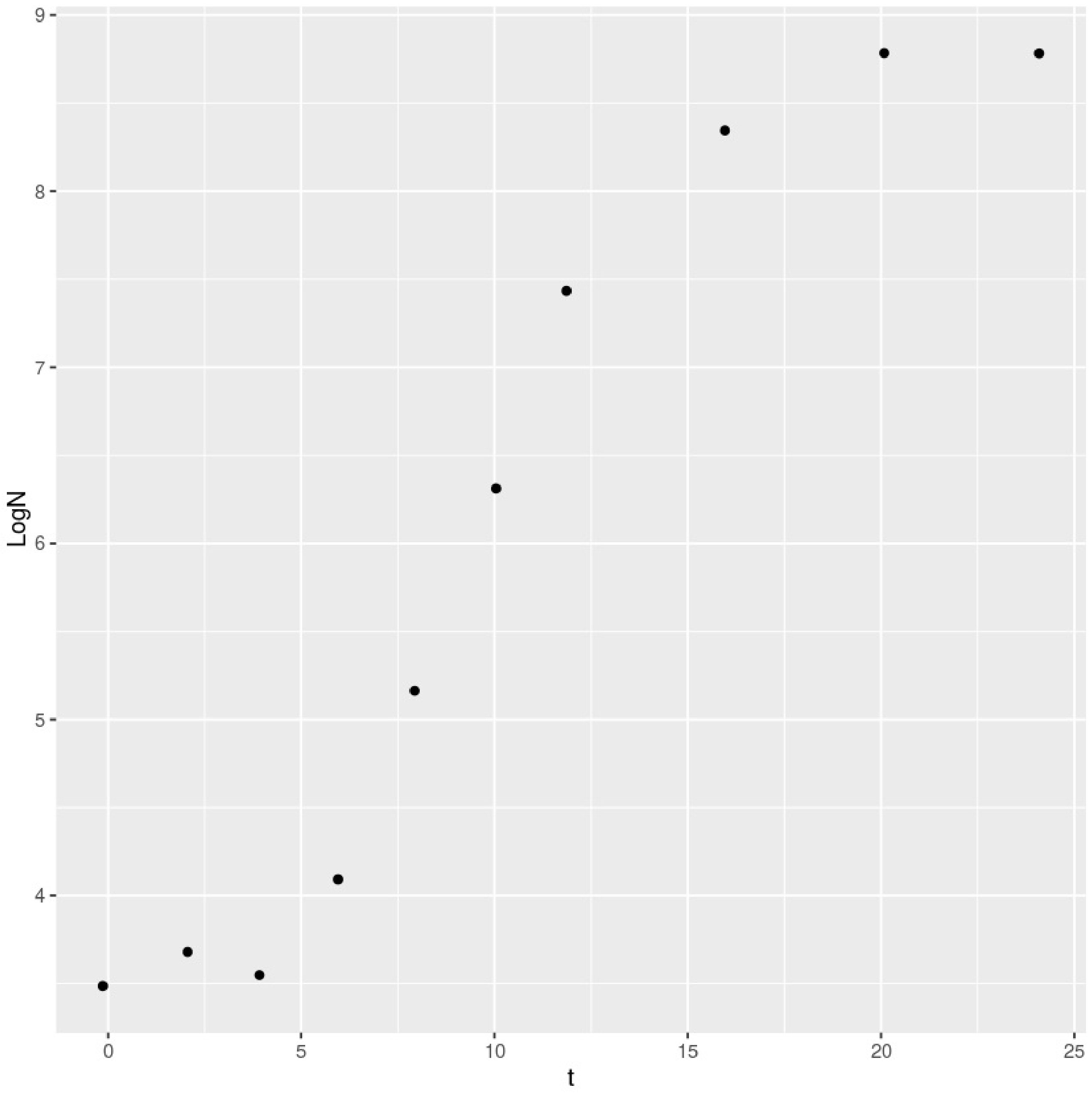
```
time <- c(0, 2, 4, 6, 8, 10, 12, 16, 20, 24) # timepoints, in hours
log_cells <- c(3.62, 3.62, 3.63, 4.14, 5.23, 6.27, 7.57, 8.38, 8.70, 8.69) # logged cell counts - more on th
is below

data <- data.frame(time, log_cells) + rnorm(length(time),sd=.1) # add some random error

names(data) <- c("t", "LogN")
```

We have added a vector of normally distributed errors to emulate random "sampling errors". Note also that the the assumption of normality of these errors underlies the statistical analyses of Ordinary NLLS fits just as it underies Ordinary Least Squares (your standard linear modelling).

Plot the data:

In [22]:

```
ggplot(data, aes(x = t, y = LogN)) + geom_point()
```



We will fit three growth models, all of which are known to fit such population growth data, especially in microbes. These are a modified Gompertz model (Zwietering et. al., 1990), the Baranyi model (Baranyi, 1993) and the Buchanan model (or three-phase logistic model; Buchanan, 1997). Given a set of cell numbers (N) and times (t), each growth model can be described in terms of:

N_0: Initial cell culture (Population) density (number of cells per unit volume)

N_{max}: Maximum culture density (aka "carrying capacity")

r_{max}: Maximum growth rate

t_{lag}: Duration of the lag phase before the population starts growing exponentially

First let's specify the model functions:

In [23]:

```
baranyi_model <- function(t, r_max, N_max, N_0, t_lag){  # Baranyi model (Baranyi 1993)
return(N_max + log10((-1+exp(r_max*t_lag) + exp(r_max*t))/(exp(r_max*t) - 1 + exp(r_max*t_lag) * 10^(N_max-N
_0))))
}


buchanan_model <- function(t, r_max, N_max, N_0, t_lag){ # Buchanan model - three phase logistic (buchanan 1
997)
  return(N_0 + (t >= t_lag) * (t <= (t_lag + (N_max - N_0) * log(10)/r_max)) * r_max * (t - t_lag)/log(10) +
           (t >= t_lag) * (t > (t_lag + (N_max - N_0) * log(10)/r_max)) * (N_max - N_0))
}


gompertz_model <- function(t, r_max, N_max, N_0, t_lag){  # Modified gompertz growth model (Zwietering 1990)
  return(N_0 + (N_max - N_0) * exp(-exp(r_max * exp(1) * (t_lag - t)/((N_max - N_0) * log(10)) + 1)))
}
```

It is important to note that we have written the funcions in log (to the base 10 - can also be base 2 or natural log) scale. This is because NLLS fitting often converges better in log scale. The interpretation of each of the the estimated/fitted paramters does not change if we take a log of the model's equation.

Now let's generate some starting values for the NLLS fitting. We did not pay much attention to what starting values we used in the above example on fitting an allometric model because the power-law model is easy to fit using NLLS, and starting far from the optimal parameters does not matter too much. Here, we derive the starting values by using the actual data:

In [24]:

```
N_0_start <- min(data$LogN)
N_max_start <- max(data$LogN)
t_lag_start <- data$t[which.max(diff(diff(data$LogN)))]
r_max_start <- max(diff(data$LogN))/mean(diff(data$t))
```

Now fit the models:

In [25]:

```
fit_baranyi <- nlsLM(LogN ~ baranyi_model(t = t, r_max, N_max, N_0, t_lag), data,
              list(t_lag=t_lag_start, r_max=r_max_start, N_0 = N_0_start, N_max = N_max_start))

fit_buchanan <- nlsLM(LogN ~ buchanan_model(t = t, r_max, N_max, N_0, t_lag), data,
                        list(t_lag=t_lag_start, r_max=r_max_start, N_0 = N_0_start, N_max = N_max_start))

fit_gompertz <- nlsLM(LogN ~ gompertz_model(t = t, r_max, N_max, N_0, t_lag), data,
                      list(t_lag=t_lag_start, r_max=r_max_start, N_0 = N_0_start, N_max = N_max_start))
```

```
Warning message in baranyi_model(t = t, r_max, N_max, N_0, t_lag):
“NaNs produced”Warning message in baranyi_model(t = t, r_max, N_max, N_0, t_lag):
“NaNs produced”Warning message in baranyi_model(t = t, r_max, N_max, N_0, t_lag):
“NaNs produced”
```

You might get a warning message such as:

```
Warning message in baranyi_model(t = t, r_max, N_max, N_0, t_lag):
“NaNs produced”
```

This just means that the Baranyi model generated some NaNs during the fitting procedure for the given data. You can ignore it in this case (but not always - sometimes these NaNs mean that the equation is wrongly written, or that it generates NaNs across the whole range of the x-values, in which case the model is inappropriate for these data).

Get the model summaries:

In [26]:

```
summary(fit_baranyi)
summary(fit_buchanan)
summary(fit_gompertz)
```

```
Formula: LogN ~ baranyi_model(t = t, r_max, N_max, N_0, t_lag)

Parameters:
      Estimate Std. Error t value Pr(>|t|)    
t_lag  5.23445    0.39510   13.25 1.14e-05 ***
r_max  1.34149    0.09418   14.24 7.48e-06 ***
N_0    3.54887    0.09488   37.40 2.44e-08 ***
N_max  8.65373    0.09067   95.45 8.91e-11 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.153 on 6 degrees of freedom

Number of iterations to convergence: 17 
Achieved convergence tolerance: 1.49e-08
```

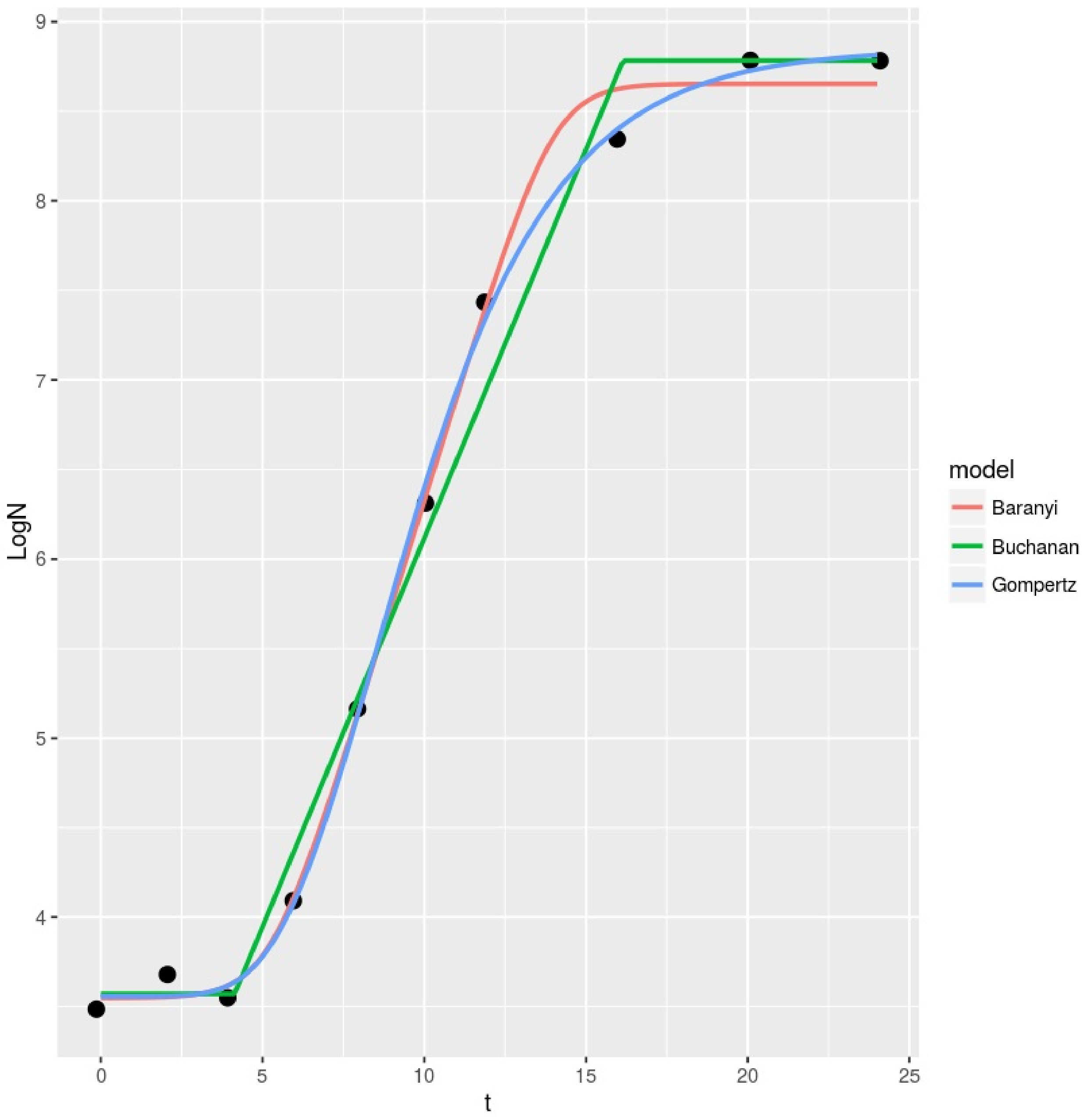
```
Formula: LogN ~ buchanan_model(t = t, r_max, N_max, N_0, t_lag)

Parameters:
      Estimate Std. Error t value Pr(>|t|)    
t_lag  4.13715    0.73469   5.631  0.00134 ** 
r_max  0.99942    0.08768  11.398 2.73e-05 ***
N_0    3.57192    0.16900  21.136 7.31e-07 ***
N_max  8.78296    0.20698  42.433 1.15e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.2927 on 6 degrees of freedom

Number of iterations to convergence: 6 
Achieved convergence tolerance: 1.49e-08
```

```
Formula: LogN ~ gompertz_model(t = t, r_max, N_max, N_0, t_lag)

Parameters:
      Estimate Std. Error t value Pr(>|t|)    
t_lag  5.47660    0.23460   23.34 4.05e-07 ***
r_max  1.46198    0.07087   20.63 8.44e-07 ***
N_0    3.55900    0.05991   59.41 1.53e-09 ***
N_max  8.84733    0.07520  117.64 2.54e-11 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.09735 on 6 degrees of freedom

Number of iterations to convergence: 9 
Achieved convergence tolerance: 1.49e-08
```

And see how the fits look:

In [27]:

```
timepoints <- seq(0, 24, 0.1)
baranyi_points <- baranyi_model(t = timepoints, r_max = coef(fit_baranyi)["r_max"], N_max = coef(fit_baranyi
)["N_max"],
                                N_0 = coef(fit_baranyi)["N_0"], t_lag = coef(fit_baranyi)["t_lag"])
buchanan_points <- buchanan_model(t = timepoints, r_max = coef(fit_buchanan)["r_max"], N_max = coef(fit_buch
anan)["N_max"],
                                N_0 = coef(fit_buchanan)["N_0"], t_lag = coef(fit_buchanan)["t_lag"])
gompertz_points <- gompertz_model(t = timepoints, r_max = coef(fit_gompertz)["r_max"], N_max = coef(fit_gomp
ertz)["N_max"],
                                N_0 = coef(fit_gompertz)["N_0"], t_lag = coef(fit_gompertz)["t_lag"])

df1 <- data.frame(timepoints, baranyi_points)
df1$model <- "Baranyi"
names(df1) <- c("t", "LogN", "model")

df2 <- data.frame(timepoints, buchanan_points)
df2$model <- "Buchanan"
names(df2) <- c("t", "LogN", "model")

df3 <- data.frame(timepoints, gompertz_points)
df3$model <- "Gompertz"
names(df3) <- c("t", "LogN", "model")

model_frame <- rbind(df1, df2, df3)

ggplot(data, aes(x = t, y = LogN)) +
  geom_point(size = 3) +
  geom_line(data = model_frame, aes(x = t, y = LogN, col = model), size = 1)
```



Exercises

(a) Calculate the confidence intervals on the parameters of each of the three fitted models, and use model selection (using AIC and/or BIC) as you did before to find the best-fitting model of the three

(b) Repeat the model comparison exercise 1000 times (You will have to write a loop), and determine whether one model gerally wins more often than the others. Note that each rub will generate a slightly different dataset, because we are adding a vector of random errors every time the "data" are generated.

(c) Repeat (2), but increase the error by increasing the standard deviation of the normal error distributon, and see if there are differences in the robustness of the models to sampling/experimental errors. You may also want to try chaning the distribution of the errors to some non-normal distribtion as see what happens.

(d) Fit some real data to these models! Import the following dataset on bacterial growth rates (https://github.com/mhasoba/TheMulQuaBio/blob/master/Data/example_growth_data.csv) into R:

In [28]:

```
BacData <- read.csv("../Data/example_growth_data.csv")

head(BacData)
tail(BacData)
```

ID	bacterial_genus	replicate	trait_name	trait_value	hour
Sch_AE103_02	Flavobacterium	1	Log(cells/mL)	5.301030	0
Sch_AE103_02	Flavobacterium	1	Log(cells/mL)	5.301030	5
Sch_AE103_02	Flavobacterium	1	Log(cells/mL)	6.991226	10
Sch_AE103_02	Flavobacterium	1	Log(cells/mL)	8.094820	15
Sch_AE103_02	Flavobacterium	1	Log(cells/mL)	8.358316	20
Sch_AE103_02	Flavobacterium	1	Log(cells/mL)	8.460296	25

	ID	bacterial_genus	replicate	trait_name	trait_value	hour
963	Wil_SP04_02	Bacillus	4	Log(cells/mL)	7.086360	25
964	Wil_SP04_02	Bacillus	4	Log(cells/mL)	7.322219	30
965	Wil_SP04_02	Bacillus	4	Log(cells/mL)	7.361728	35
966	Wil_SP04_02	Bacillus	4	Log(cells/mL)	7.322219	40
967	Wil_SP04_02	Bacillus	4	Log(cells/mL)	7.260071	45
968	Wil_SP04_02	Bacillus	4	Log(cells/mL)	7.292256	50

The column `trait_value` and `hour` are your variables of interest (log cell density and time), respectively. Note that the `ID` column will tell you which rows represent one separate growth experiment. Make sure you have a good look at the data first by plotting them up (idealy, in a loop).

That's the end of this NLLS tutorial. Hope you found it useful, and that it gives you a toe-hold for learning how to fit non-linear models to your own data.

Also, note that similar to linear models, you can use Non-linear Mixed Effects statistical methods to fit nonlinear models to data (e.g., using the `lme4` and/or `nlme` packages in R).

Readings and Resources

- Motulsky, Harvey, and Arthur Christopoulos. Fitting models to biological data using linear and nonlinear regression: a practical guide to curve fitting. OUP USA, 2004.
- Johnson, J. B. & Omland, K. S. 2004 Model selection in ecology and evolution. Trends Ecol. Evol. 19, 101–108.
- The NCEAS non-linear modelling working group (https://groups.nceas.ucsb.edu/non-linear-modeling/projects/OrangeTree)
- Mixed-Effects Models in S and S-PLUS (https://link.springer.com/book/10.1007/b98882)