Numerical Examples Using R

Contents

Nonlinear Regression

Nonlinear Regression (Starting Values Available)

The velocity of a chemical reaction (y) is modeled as a function of the concentration of the chemical (x). There are a total of 18 observations. The desired model is

$$y_i = \frac{\theta_0 x_i}{\theta_1 + x_i} + \epsilon_i.$$

To find reasonable starting values for θ_0 and θ_1, we take the inverse of the model expression (ignoring the error term) and fit the ordinary least-squares regression:

$$\frac{1}{y_i} = \frac{\theta_1 + x_i}{\theta_0 x_i} = \frac{1}{\theta_o} + \frac{\theta_1}{\theta_0}(\frac{1}{x_i}).$$

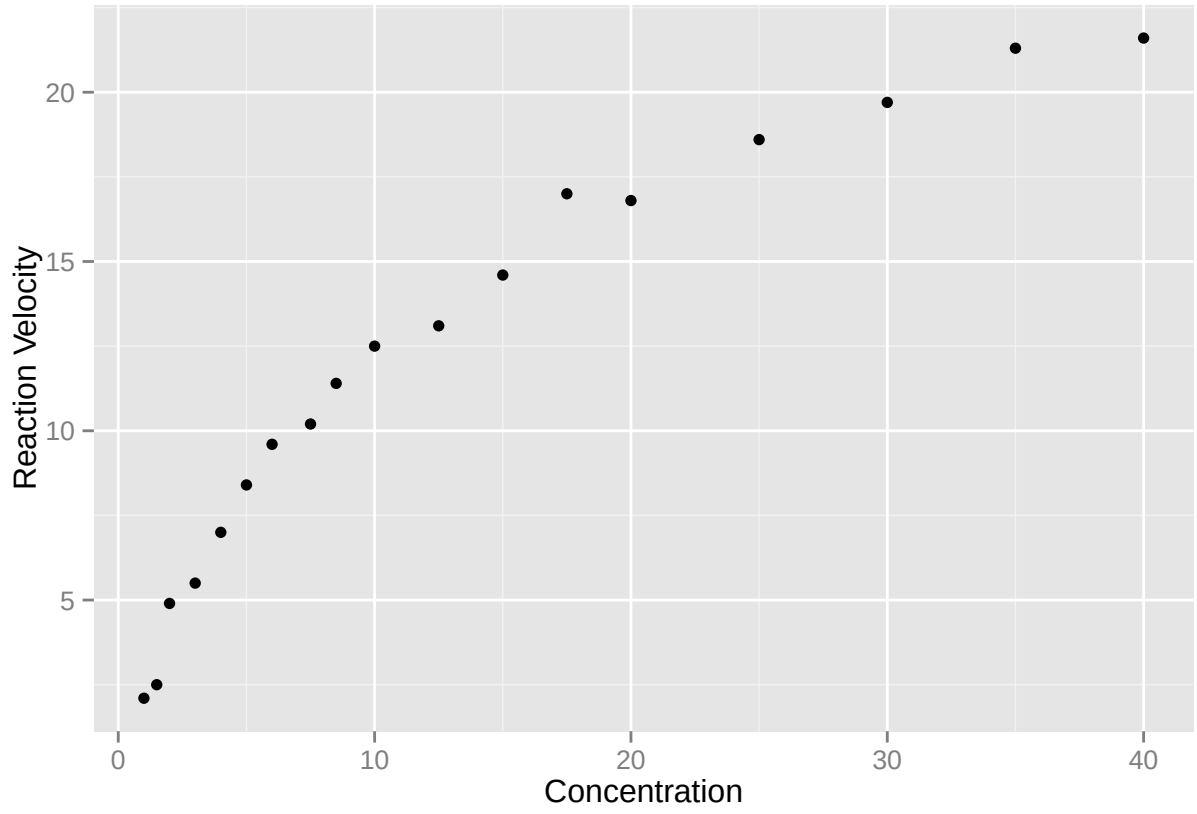
Descriptive analysis

```
enzyme <- read.table("datasets/enzyme.dat")
names(enzyme) <- c('y', 'x')
```

	y	x
1	2.1	1.0
2	2.5	1.5
3	4.9	2.0
4	5.5	3.0
5	7.0	4.0
6	8.4	5.0
7	9.6	6.0
8	10.2	7.5
9	11.4	8.5
10	12.5	10.0
11	13.1	12.5
12	14.6	15.0
13	17.0	17.5

	y	x
14	16.8	20.0
15	18.6	25.0
16	19.7	30.0
17	21.3	35.0
18	21.6	40.0

```
# plot the data
require(ggplot2)
p <- ggplot(data = enzyme, aes(x, y)) + geom_point() +
  labs(y = "Reaction Velocity", x = "Concentration")
p
```



Perform an OLS regression to find starting values

```
ols_fit <- lsfit(1/enzyme$x, 1/enzyme$y)
ols_fit$coefficients
```

```
##  Intercept          X
## 0.03375868 0.45401397
```

So the starting values for θ_0 and θ_1 are given by:

$$\theta_0^{(0)} = 1/\beta_0 = \frac{1}{0.03375868} = 29.62$$

$$\theta_1^{(0)} = \beta_1/\beta_0 = \frac{0.45401397}{0.03375868} = 13.45$$

Perform a non-linear regression with Gauss-Newton method

Fit the model The `nls()` function in *stats* package performs nonlinear (weighted) least-square estimates of the parameter of a nonlinear model. It can use a `formula` object to specify a model and any user-specified function can be used in the model. Because nonlinear models are sometimes complicated, here we showed how to use a customized `nlmodel()` function to specify the model.

To match the SAS output, `trace = T` is set to print out iteration history. Note that the first column corresponding to the objective function and the other columns corresponding to the parameter estimates for each iteration.

```
nlmodel <- function(x, theta0, theta1) theta0 * x / (theta1 + x)
nls_fit <- nls(y ~ nlmodel(x, theta0, theta1), data = enzyme,
               start = list(theta0 = 29.62, theta1 = 13.45), trace = T)
```

```
## 6.57116 :  29.62 13.45
## 4.303542 :  28.14228 12.59796
## 4.302271 :  28.13785 12.57534
## 4.302271 :  28.13708 12.57449
## 4.302271 :  28.13705 12.57445
```

```
nls_fit
```

```
## Nonlinear regression model
##   model: y ~ nlmodel(x, theta0, theta1)
##    data: enzyme
## theta0 theta1
##  28.14  12.57
##  residual sum-of-squares: 4.302
##
## Number of iterations to convergence: 4
## Achieved convergence tolerance: 4.304e-07
```

Parameter estimates Parameter estimates and other model summaries can be obtained using the `summary()` function.

```
sm <- summary(nls_fit, correlation = T)
sm$coefficients  # coefficients and their significance
```

```
##        Estimate Std. Error  t value     Pr(>|t|)
## theta0 28.13705  0.7279790 38.65091 3.137221e-17
## theta1 12.57445  0.7630534 16.47913 1.850253e-11
```

```
sm$correlation  # correlation matrix of parameters
```

```
##           theta0    theta1
## theta0 1.0000000 0.9366248
## theta1 0.9366248 1.0000000
```

To get individual confidence intervals on paramter estimates, we need to know the standard error of the estiamtes as well as the degree of freedom of the estimates of σ^2. That can be obtained from `df.residual()`.

```
rdf <- df.residual(nls_fit)
# C.I. for theta0
sm$coefficients[1, 1] + qt(c(.025, .975), rdf) * sm$coefficients[1, 2]
```

```
## [1] 26.5938 29.6803
```

```
# C.I. for theta1
sm$coefficients[2, 1] + qt(c(.025, .975), rdf) * sm$coefficients[2, 2]
```

```
## [1] 10.95685 14.19205
```

Other Diagnostics To compute the leverage, we need F and $(F'F)^{-1}$. The tangent plane hat matrix is $H = F(F'F)^{-1}F'$. The leverage values are the diagonal elements of the hat matrix.

```
cf <- nls_fit$m$gradient()  # cf: capital f matrix
sm$cov.unscaled  # (F'F)^(-1)
```

```
##          theta0   theta1
## theta0 1.970879 1.934914
## theta1 1.934914 2.165370
```

```
ch <- cf %*% sm$cov.unscaled %*% t(cf)

# diagonal of hat matrix
kable(diag(ch), row.names = T, col.names = c("LEV"))
```

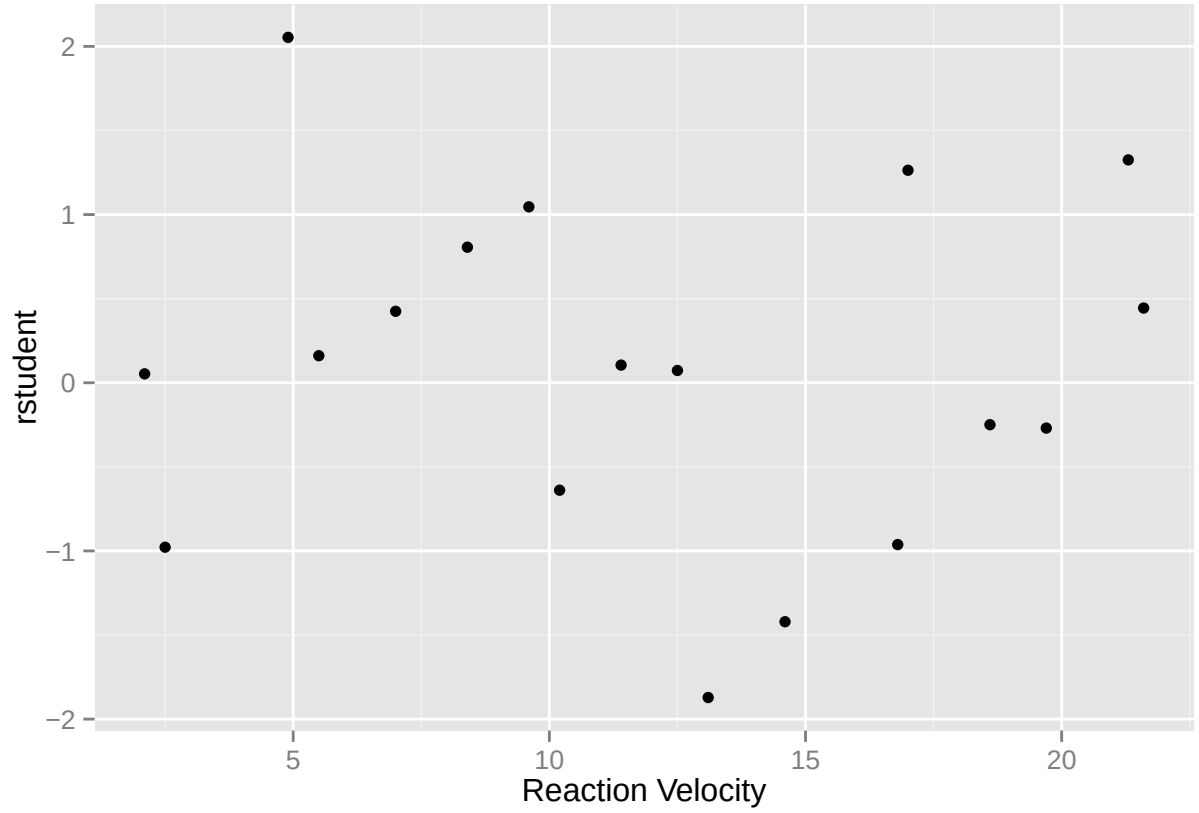
	LEV
1	0.0176537
2	0.0328108
3	0.0484048
4	0.0759531
5	0.0956214
6	0.1072992
7	0.1124450
8	0.1117837
9	0.1080296
10	0.1003662
11	0.0882287
12	0.0818877
13	0.0831978

	LEV
14	0.0919111
15	0.1271639
16	0.1783176
17	0.2379108
18	0.3010149

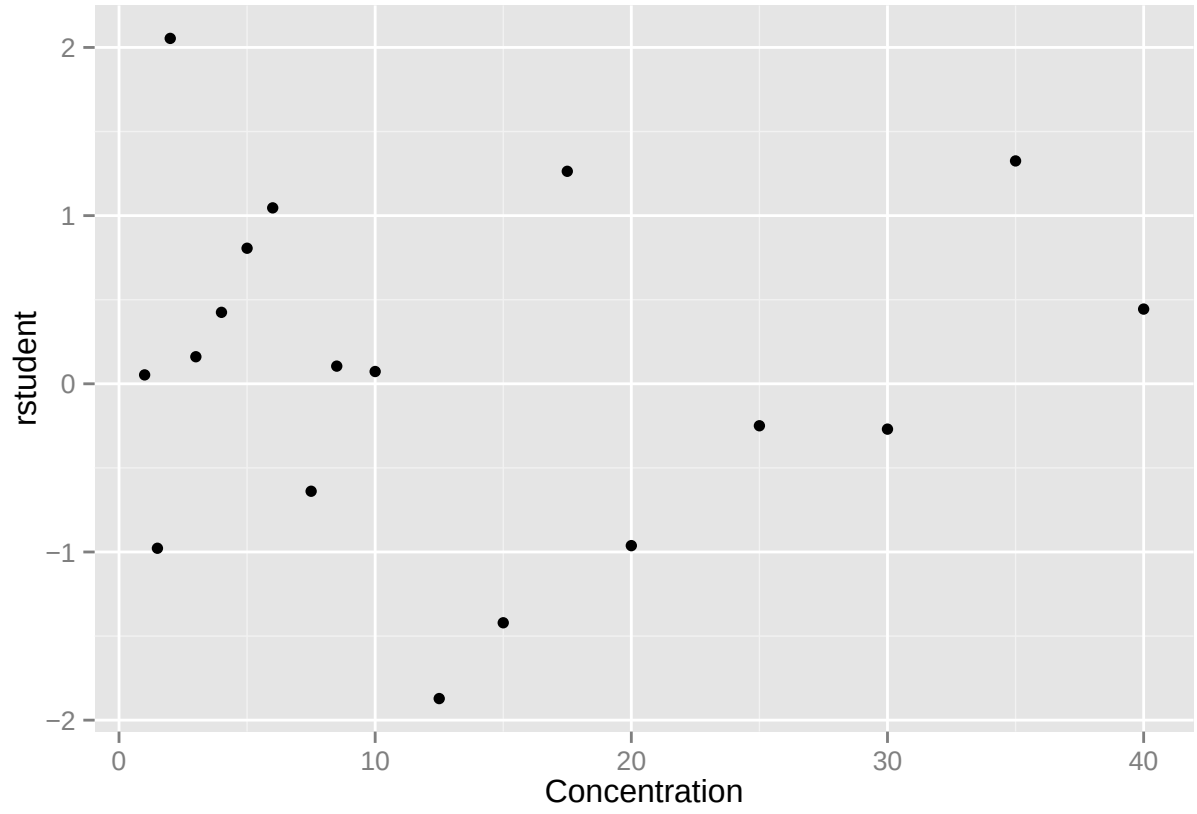
Studentized residual plots The studentized residual is computed as

$$r_i = \frac{e_i}{s\sqrt{1 - \hat{H}_{ii}}}$$

```
rstudent <- residuals(nls_fit) / (sm$sigma * sqrt(1 - diag(ch)))
enzyme <- data.frame(enzyme, rstudent)
qplot(y, rstudent, data = enzyme, geom = "point", xlab = "Reaction Velocity")
```




```
qplot(x, rstudent, data = enzyme, geom = "point", xlab = "Concentration")
```

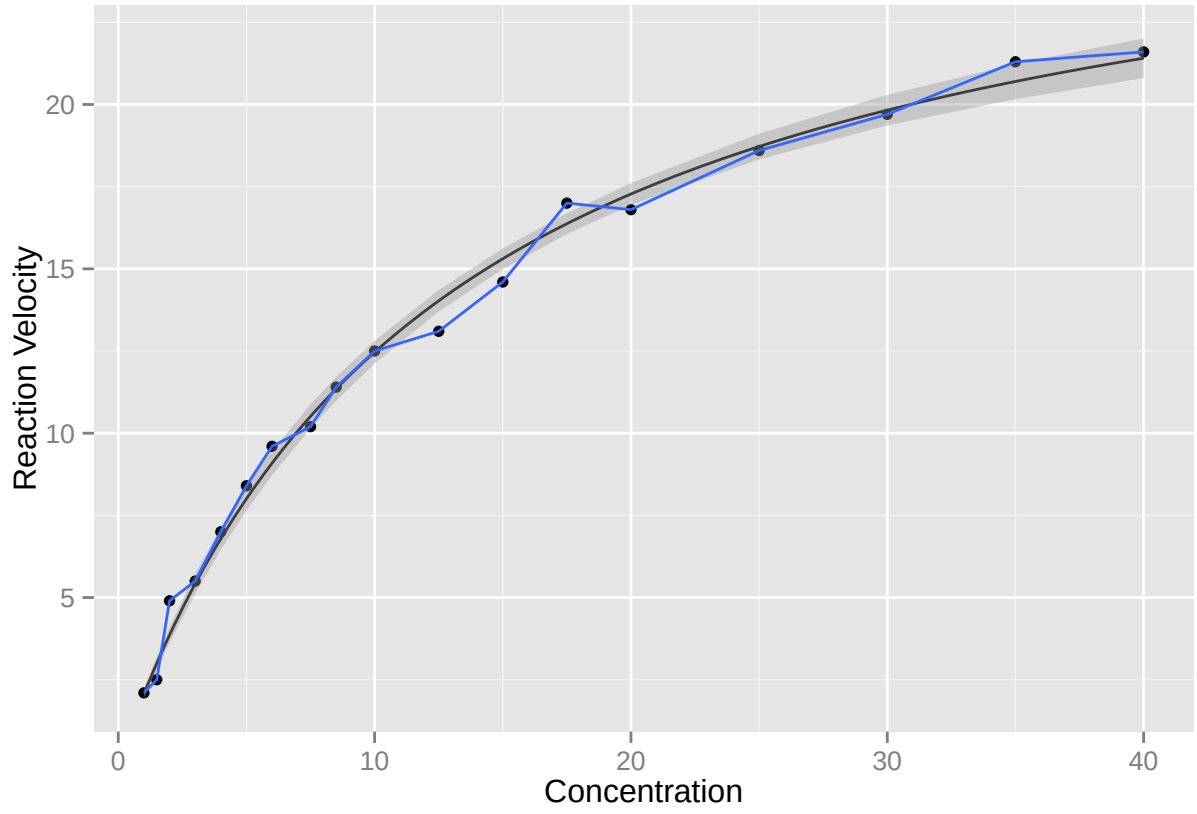


Confidence intervals for Y The confidence interval is computed by

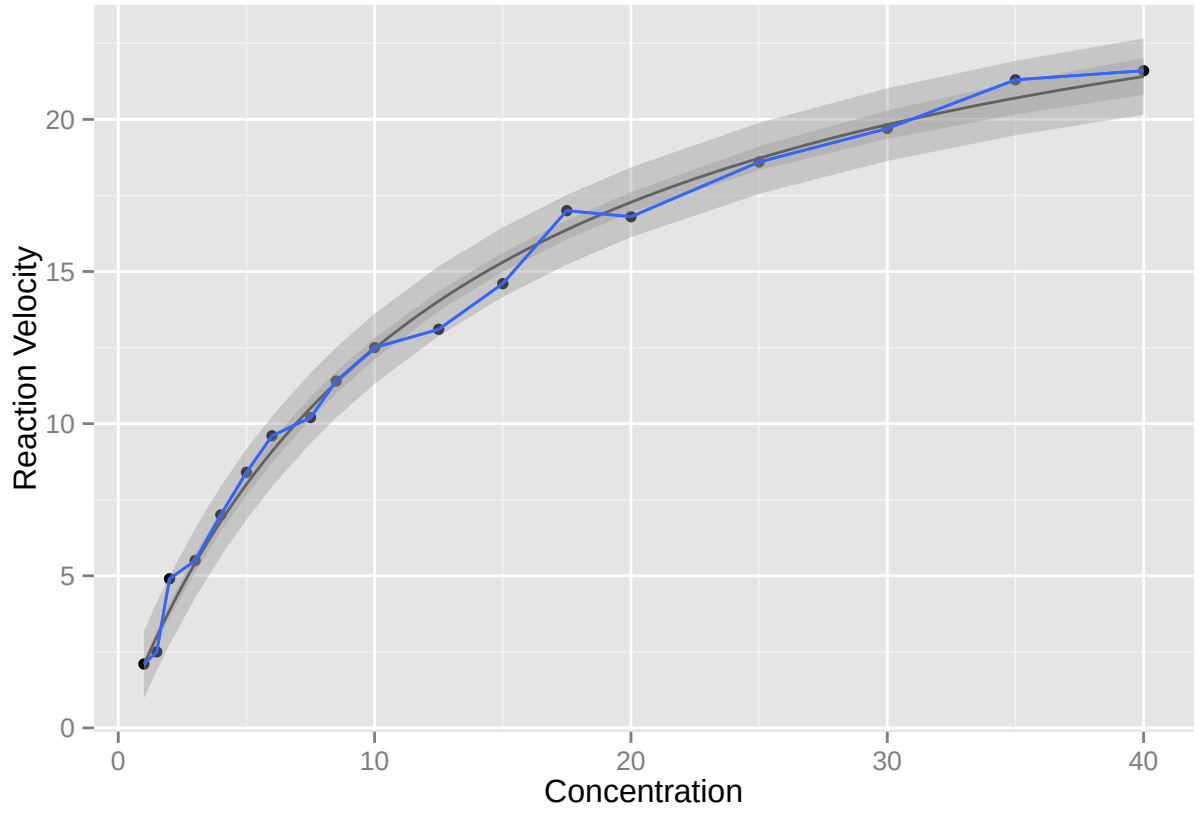
$$\hat{Y}_0 \pm t_{(n-p,1-\alpha/2)} s[f_0'(F'F)^{-1}f_0]^{1/2}.$$

Note that if we want confidence interval at original x values x_i, the definition for f_0 here is just the transpose of $F(\theta, x_i)$.

```
y0 <- fitted(nls_fit)
se <- apply(t(cf), 2, function(f0) {
  sm$sigma * {t(f0) %*% sm$cov.unscaled %*% f0}^(.5)
})

ll <- fitted(nls_fit) + qt(.025, df.residual(nls_fit)) * se
ul <- fitted(nls_fit) + qt(.975, df.residual(nls_fit)) * se
ci <- data.frame(enzyme, ll, ul)


p + stat_function(fun = nlmodel, args = as.list(coef(nls_fit))) +
  geom_smooth(aes(ymin = ll, ymax = ul), data = ci, stat="identity")
```




```
# prediction intervals are similar
se <- apply(t(cf), 2, function(f0) {
  sm$sigma * {1 + t(f0) %*% sm$cov.unscaled %*% f0}^(.5)
})

ll <- fitted(nls_fit) + qt(.025, df.residual(nls_fit)) * se
ul <- fitted(nls_fit) + qt(.975, df.residual(nls_fit)) * se
pi <- data.frame(enzyme, ll, ul)

p + stat_function(fun = nlmodel, args = as.list(coef(nls_fit))) +
  geom_smooth(aes(ymin = ll, ymax = ul), data = ci, stat="identity") +
  geom_smooth(aes(ymin = ll, ymax = ul), data = pi, stat="identity")
```



Nonlinear Regression (Grid-Search)

We will model the time evolution of an algal sample taken in the Adriatic Sea (Cavallini, 1993). Time (x) is expressed in days and biomass (y), which is a measure of growth, is measured in mm2 (what is actually measured is the surface covered by biomass in a microscopic sample). The data seem to follow a logistic curve:

$$y_i = \frac{\theta_0}{1 + \exp(-\theta_1(x_i - \theta_2))}$$

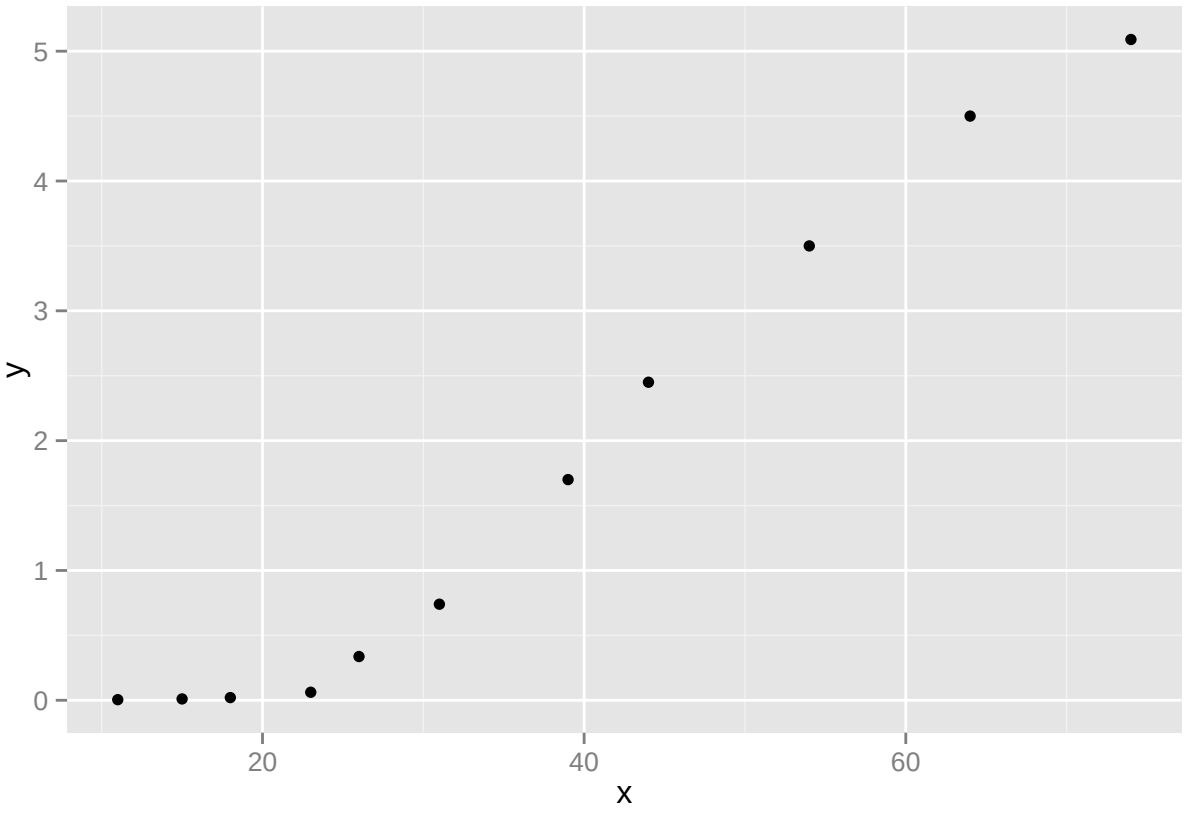
Descriptive analysis

```
algal <- data.frame(x = c(11, 15, 18, 23, 26, 31, 39, 44, 54, 64, 74),
                    y = c(.0048, .0105, .0207, .0619, .3370,
                          .7400, 1.7, 2.45, 3.5, 4.5, 5.09))
kable(algal, row.names = T)
```

	x	y
1	11	0.0048
2	15	0.0105
3	18	0.0207

	x	y
4	23	0.0619
5	26	0.3370
6	31	0.7400
7	39	1.7000
8	44	2.4500
9	54	3.5000
10	64	4.5000
11	74	5.0900

```
require(ggplot2)
p <- ggplot(data = algal, aes(x, y)) + geom_point()
p
```



Perform a nonlinear regression with the grid-search method

In R, we can perform the grid search and find the starting value manually.

```
# setup grid
grid <- expand.grid(theta0 = seq(0, 10, 2),
                    theta1 = seq(0, .5, .1),
                    theta2 = seq(40, 60, 5))
kable(head(grid))
```

theta0	theta1	theta2
0	0	40
2	0	40
4	0	40
6	0	40
8	0	40
10	0	40

We specify the nonlinear model and compute the sum of squares of error manually.

```
nlmodel <- function(x, theta) {
  theta[1] / (1 + exp(-theta[2] * (x - theta[3])))
}

sse <- apply(grid, 1, function(theta) {
  sum((algal$y - nlmodel(algal$x, theta))^2)
})
kable(head(data.frame(grid, sse)))  # only the head of the table is shown
```

theta0	theta1	theta2	sse
0	0	40	67.96616
2	0	40	42.13636
4	0	40	38.30656
6	0	40	56.47676
8	0	40	96.64696
10	0	40	158.81716

Final starting values:

```
grid[which.min(sse), ]
```

```
##    theta0 theta1 theta2
## 82      6    0.1     50
```

```
nls_fit <- nls(y ~ nlmodel(x, theta), data = algal, trace = T,
               start = list(theta = c(6, .1, 50)))
```

Fit the nonlinear model

```
## 0.6350226 :   6.0  0.1 50.0
## 0.2857354 :   5.0141669  0.1157241 45.8625826
## 0.2384547 :   5.0887283  0.1221786 45.7619359
## 0.2381781 :   5.0956237  0.1211468 45.7748836
## 0.2381669 :   5.0947305  0.1213394 45.7743463
## 0.2381665 :   5.0949589  0.1213007 45.7748138
## 0.2381664 :   5.0949235  0.1213077 45.7747829
## 0.2381664 :   5.0949315  0.1213063 45.7747980
## 0.2381664 :   5.0949302  0.1213066 45.7747966
```

```
nls_fit
```

```
## Nonlinear regression model
##   model: y ~ nlmodel(x, theta)
##    data: algal
##  theta1  theta2  theta3
##  5.0949  0.1213 45.7748
##  residual sum-of-squares: 0.2382
##
## Number of iterations to convergence: 8
## Achieved convergence tolerance: 1.925e-06
```

```
sm <- summary(nls_fit, correlation = T)
sm$coefficients
```

```
##          Estimate Std. Error  t value     Pr(>|t|)
## theta1  5.0949302 0.19766239 25.77592 5.505407e-09
## theta2  0.1213066 0.01159284 10.46393 6.044273e-06
## theta3 45.7747966 1.17097569 39.09116 2.015699e-10
```

```
sm$correlation
```

```
##            theta1     theta2     theta3
## theta1  1.0000000 -0.6865027  0.8166978
## theta2 -0.6865027  1.0000000 -0.6528791
## theta3  0.8166978 -0.6528791  1.0000000
```

```
# the output shows theta 1-3, it's actually theta 0-2
mapply(function(est, se) est + qt(c(.025, .975), df.residual(nls_fit)) * se,
       sm$coefficients[, 1], sm$coefficients[, 2], SIMPLIFY = F)
```

Confidence intervals for parameter estimates

```
## $theta1
## [1] 4.63912 5.55074
##
## $theta2
## [1] 0.09457346 0.14803972
##
## $theta3
## [1] 43.07452 48.47507
```

```
y0 <- fitted(nls_fit)
cf <- cf <- nls_fit$m$gradient()
se <- apply(t(cf), 2, function(f0) {
```

```
  sm$sigma * {t(f0) %*% sm$cov.unscaled %*% f0}^(.5)
})

ll <- fitted(nls_fit) + qt(.025, df.residual(nls_fit)) * se
ul <- fitted(nls_fit) + qt(.975, df.residual(nls_fit)) * se
ci <- data.frame(algal, ll, ul)

# prediction intervals are similar
se <- apply(t(cf), 2, function(f0) {
  sm$sigma * {1 + t(f0) %*% sm$cov.unscaled %*% f0}^(.5)
})

ll <- fitted(nls_fit) + qt(.025, df.residual(nls_fit)) * se
ul <- fitted(nls_fit) + qt(.975, df.residual(nls_fit)) * se
pi <- data.frame(algal, ll, ul)

p + stat_function(fun = nlmodel, args = list(coef(nls_fit))) +
  geom_smooth(aes(ymin = ll, ymax = ul), data = ci, stat="identity") +
  geom_smooth(aes(ymin = ll, ymax = ul), data = pi, stat="identity")
```

Confidence and prediction interval for Y