HWA1, help document

Lukas Arnroth

7 juli 2017

Program to generate data

In the code below you find the program that will generate the data for this assignment. You can choose to copy it here or simply use the .R file on the student portal.

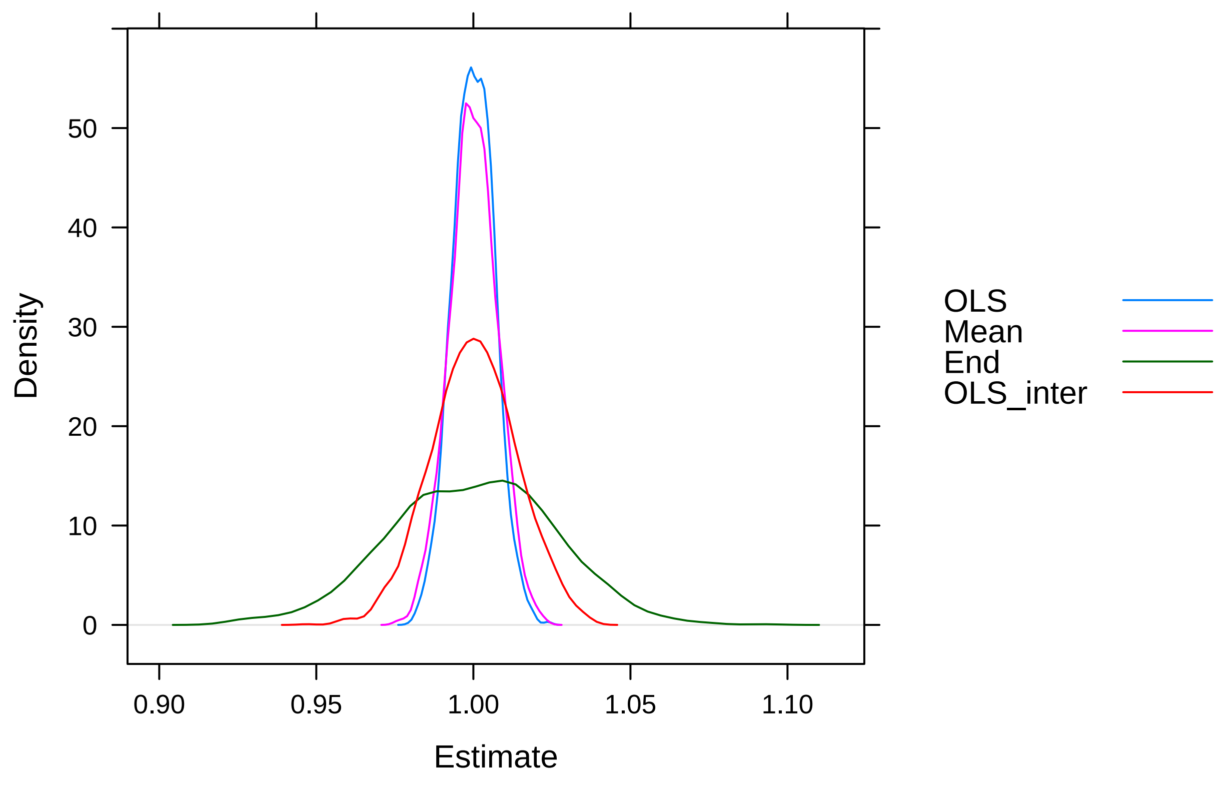
```
library(reshape2)

### Simulation program starts

# set size of sample by assigning a value to n which
# will be used later in the program
n <- 40

# Number of replications
reps <- 2000

# Set seed of random number generator
set.seed(123456) # hello

# Parameters of the regression
beta_1 <- 0
beta_2 <- 1
sigma_2 <- 1

# Generate t, which is the independent variable
# (wrote x as in the eviews code)
x <- 1:n # 1, ..., 40
xx = x*x # 1^2, ..., 40^2
sumxx<- sum(xx) # sum of squares of x


# Create matrix to store the results from the simulations
# We want one row (nrow) per repetition and one column for
# each estimator.
out <- matrix(nrow = reps, ncol = 4, dimnames = list(NULL, c("OLS", "Mean",
                                                              "End", "OLS_inter")))

# generate Y and residuals

for(i in 1:reps){
  # residuals
  u = rnorm(n)*sqrt(sigma_2)
  # data generating process of y, the dependent variable
  y = beta_1 + beta_2*x + u
  yx = y*x
  # estimator 1, replication i
  out[i, 1] <- sum(yx)/sumxx
```

```r
  # estimator 2, replication i
  out[i, 2] <- mean(y)/mean(x)
  # estimator 3, replication i
  # n'th Y and T
  out[i, 3] <- y[n]/n
  # estimator 4, replication i
  # this one is 1 all the time, so not even an estimate?
  out[i, 4] <- cov(y, x)/var(x)
}

# The melt function is from the reshape2 package. It makes the
# matrix into a data frame which is more suitable for plotting in ggplot
data <- melt(out, value.name = "Estimate")
# calling 'data' you notice that the matrix has been transformed into a data frame
# with 3 variables, first one is the time(rowname), second is estimator
# name (column name)  and third is the estimate (elements of the out matrix)
```

Ploting the Results

Learning how to plot in R can take some time to get used to. You will have more freedom in creating your plots than you're used to. Here I will provide you with code that allows you to create the necessary plots for this assignment. Firstly, we have the density plots of each estimate. This can be created in several ways, the most simple is done using the 'lattice' package.

```r
library(lattice)

# Estimate is the variable we want to plot
# and we want to create a seperate line for
# each estimate, we simply group by the
# Var2 column which is the type of estimator
densityplot(~Estimate, data = data, groups = Var2,
            plot.points = FALSE, ref = TRUE,
            auto.key = list(space = "right"))
```



Interpreting the results is up to you!

Fitting a Linear Model

Here we use the y and x vector from the data generation code. We will use the lm() function of R to fit a linear model. This topic is covered in section 11 of the R introduction. Say that we would like to estimate the following model

$$y = \beta_0 + x * \beta_1 \tag{1}$$

First thing that you should pay attention to is the fact that "=" is the same as "<-" in the R language, meaning that the equality sign is reserved as assignment to an object in the global environment. Instead of using "=" as in equation 1, we will use the tilde, "~".

```r
# specify the linear model and store it in the object "m"
# default setting is to use intercept
m <- lm(y ~ x)
m
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Coefficients:
## (Intercept)            x
##      0.2082       0.9904
```

Calling the object after fitting gives the estimated parameters. For more detailed information from model estimation we can use the summary() function on the lm-object. This is covered in section 11.3 of the R introduction.

```r
summary(m)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.9157 -0.8476 -0.0788  0.7538  3.3544
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.20824    0.39305    0.53    0.599
## x            0.99040    0.01671   59.28   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.22 on 38 degrees of freedom
## Multiple R-squared:  0.9893, Adjusted R-squared:  0.989
## F-statistic:  3514 on 1 and 38 DF,  p-value: < 2.2e-16
```

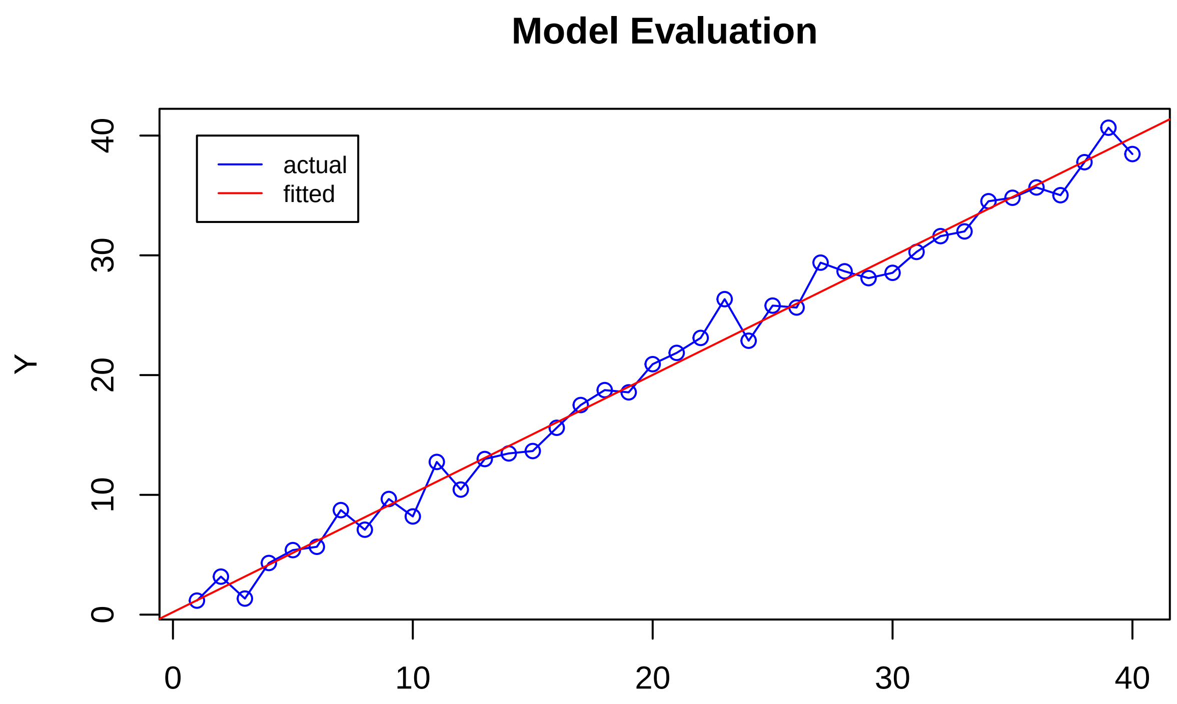
Ploting the Results from Model Fit

Next we will move on to evaluation through plotting. Ploting in R is frustrating and rewarding all at once. It takes some time to get a working knowledge of how R works when it comes to plots but you have great freedom in ploting using packages such as ggplot2 (although this package is outside the scope of this course). I would like to evaluate the fitted values against actual values, aswell as investigate the residuals. We should start by putting everything we need into one data frame. We will need the actual y, x, fitted y and the residuals from the estimation. To combine all these into a data frame we will use the data.frame() function. This is quite straight forward. The hard part here is extracting what we want from the lm-object. We will use the fitted.values() and residuals() functions in order to get these values.

```r
plot_data <- data.frame(y = y,
                        x = x,
                        fitted = fitted.values(m),
                        resid = residuals(m))
```

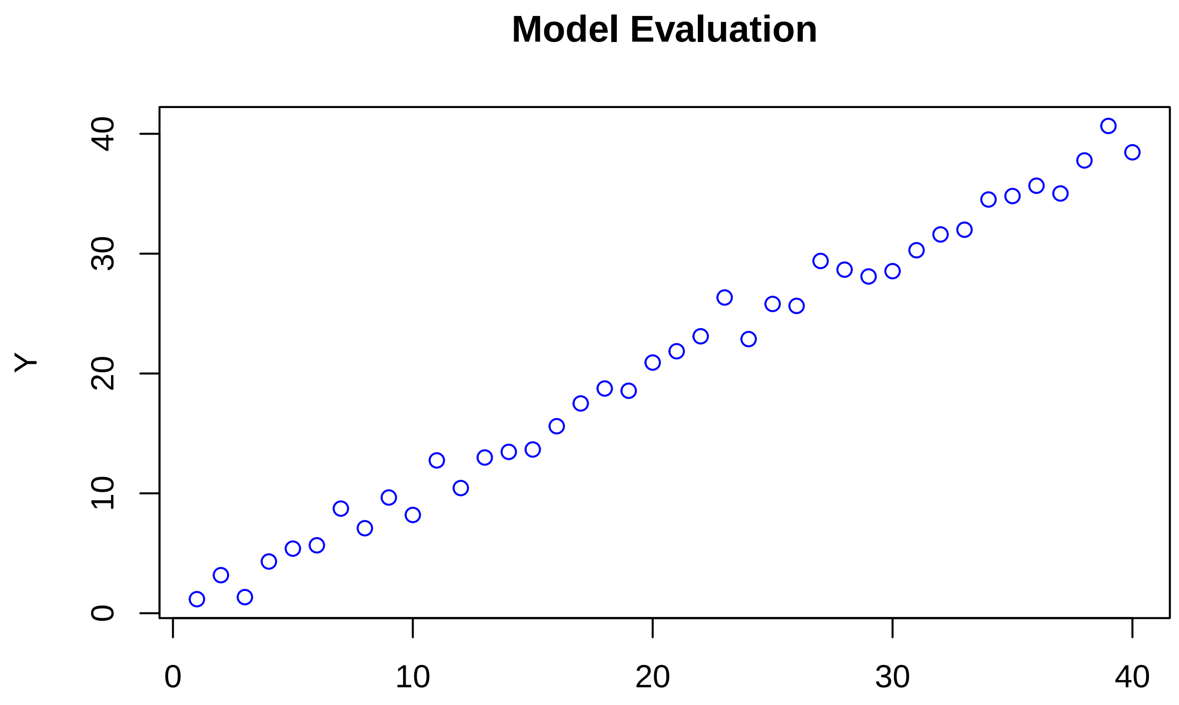
In the code above we create a data frame with 4 variables. The names of these are specified on the left hand side of the equality sign. The object assigned to these are on the right hand side. Now, lets begin by ploting the fitted values against the actual values. Before moving further I strongly suggest you look through section 12.1.1 of the R introduction to understand the plot() function. Notice here that in the first line of code we supply the function with two vectors.

```r
plot(y = plot_data$y, x = plot_data$x, col = "blue", ylab = "Y", xlab = "",
        main = "Model Evaluation") # plot actual y against x
lines(plot_data$y, col = "blue") # create line between the points of x
abline(m, col = "red") # add fitted y's from the linear model object
# adding legend, 40 is x-coordinate of legend, col are the colours, and lty specifies type of icon
# cex specifies the size of the legend window
legend(40, c("actual", "fitted"), col = c("blue", "red"), lty=c(1,1), cex = 0.75)
```

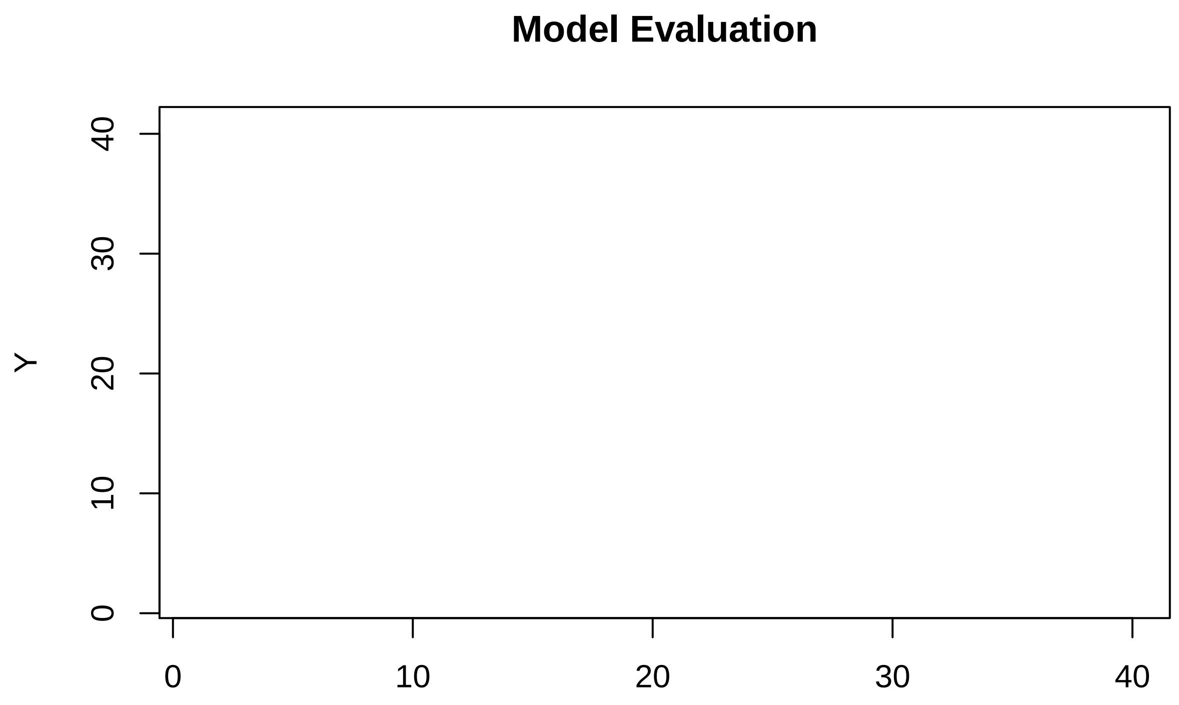


Personally I don't care very much for the points along the blue line. If I want to remove them I need to tell the plot() function to do nothing in the beginning. The circles are actually created by the plot() function before adding the lines to the plot. This is done by adding another parameter to the plot() call: *type* = "*n*". Let's have a look at the plot() call alone first

```r
# without specifying type = "n" plot() produces a scatter plot when supplied two vectors
plot(y = plot_data$y, x = plot_data$x, col = "blue", ylab = "Y", xlab = "",
        main = "Model Evaluation") # plot actual y against x
```

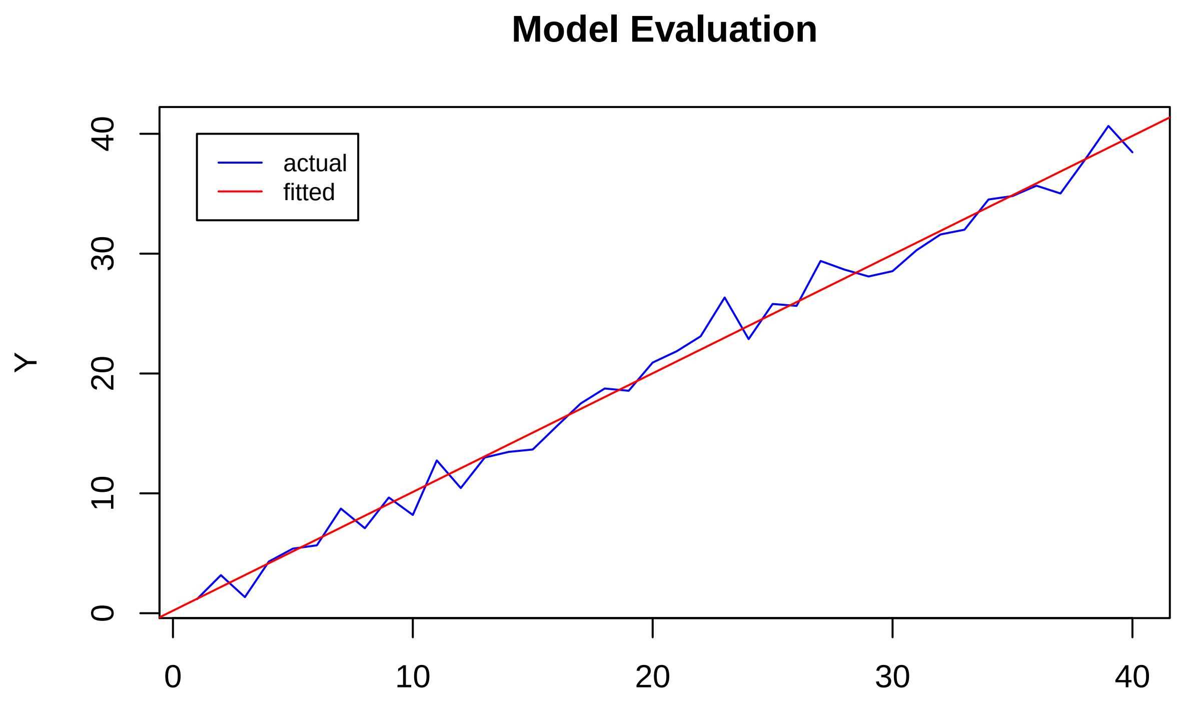


```
# specifying type = "n" overloads the scatterplot "problem". Essentially giving us
# a blank canvas against which we can add the lines
plot(y = plot_data$y, x = plot_data$x, col = "blue", ylab = "Y", xlab = "",
        main = "Model Evaluation", type = "n") # plot actual y against x
```



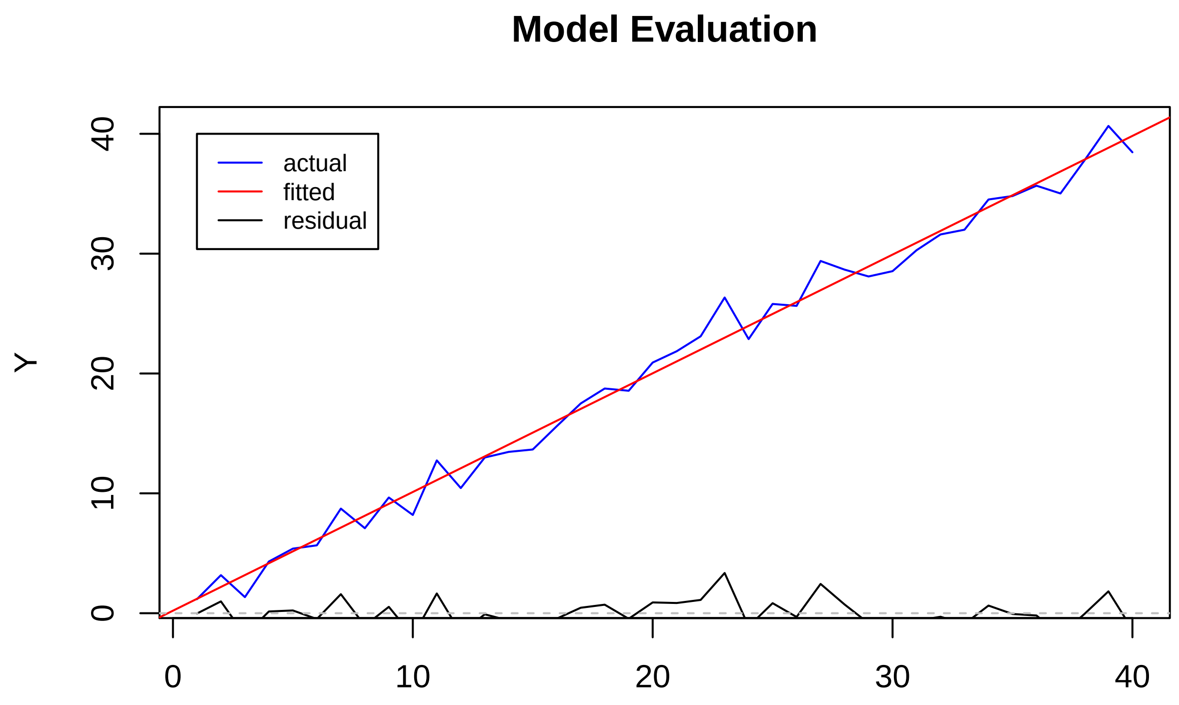
Using the last version of the plot() call we can start adding the lines

```r
plot(y = plot_data$y, x = plot_data$x, col = "blue", ylab = "Y", xlab = "",
        main = "Model Evaluation", type = "n")
lines(plot_data$y, col = "blue")
abline(m, col = "red")
legend(40, c("actual", "fitted"), col = c("blue", "red"), lty=c(1,1), cex = 0.75)
```



Now I would like to include the residuals in this plot and see if they seem to fluctuate randomly around 0. So the best way seems to simply add the residuals as a new line to the plot and also include a horizontal line at $y = 0$.

```r
plot(y = plot_data$y, x = plot_data$x, col = "blue", ylab = "Y", xlab = "",
        main = "Model Evaluation", type = "n")
lines(plot_data$y, col = "blue")
abline(m, col = "red")
lines(plot_data$resid, col = "black")
# abline with intercept(a) 0 and slope(b) 0. lty = 2 makes it dotted and more discrete
abline(a = 0, b = 0, col = "gray", lty = 2)
# we should ofcourse include the new residual line in the legend
# you can combare the legend of the new plot with the old one and
# figure out how it works!
legend(40, c("actual", "fitted", "residual"),
       col = c("blue", "red", "black"),
       lty=c(1,1,1), cex = 0.75)
```



The residual line is only visible for positive values, what went wrong? The reason for this is that the y-axis by default goes between the minimum and maximum value of y with some slight modifications.

```r
min(plot_data$y);max(plot_data$y)
```

```
## [1] 1.170899
```

```
## [1] 40.65649
```

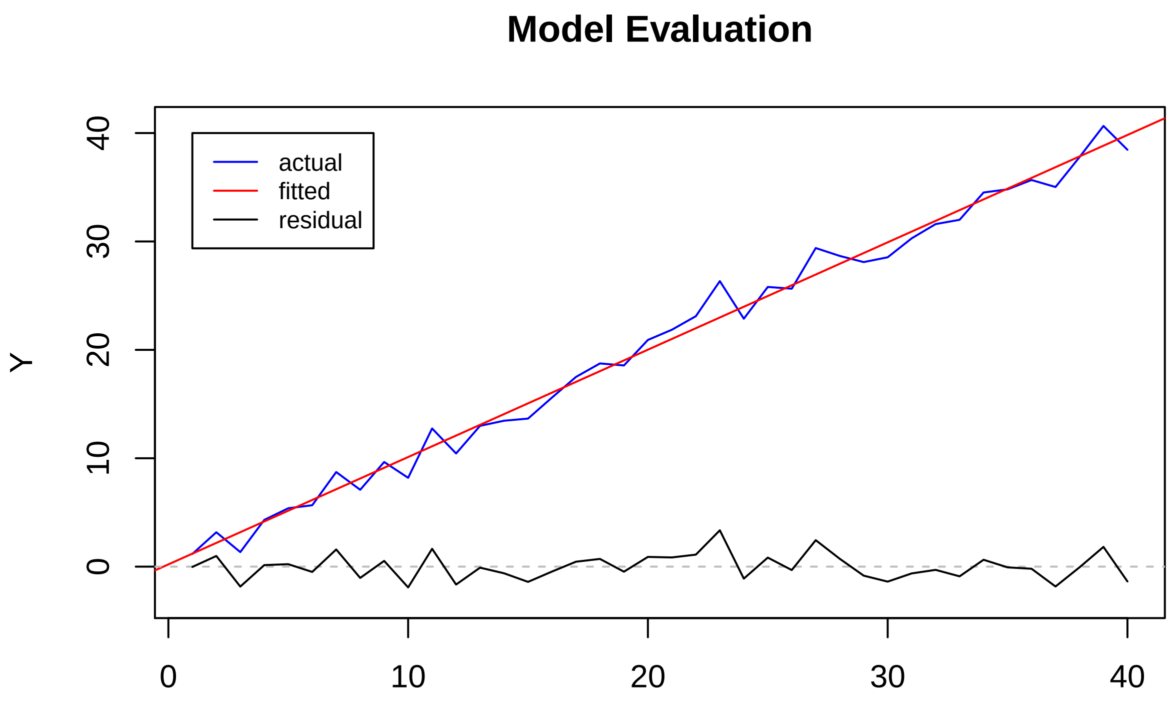
We would want this plot to begin a little under the minimum value of the residuals

```r
min(plot_data$resid)
```

```
## [1] -1.91574
```

Let's try letting the y-axis start at -3.

```r
# ylim changes the values of the y-axis
plot(y = plot_data$y, x = plot_data$x, col = "blue", ylab = "Y", xlab = "",
        main = "Model Evaluation", type = "n", ylim = c(-3, max(y)))
lines(plot_data$y, col = "blue")
abline(m, col = "red")
abline(a = 0, b = 0, col = "gray", lty = 2)
lines(plot_data$resid, col = "black")
legend(40, c("actual", "fitted", "residual"),
       col = c("blue", "red", "black"),
       lty=c(1,1,1), cex = 0.75)
```


Model Evaluation
Y
actual
fitted
residual
0
10
20
30
40
0
10
20
30
40