Fitting data

Contents

How to estimate the best fitting function to a scatter plot in R?

http://stackoverflow.com/questions/15042435/how-to-estimate-the-best-fitting-function-to-a-scatter-plot-in-r/15045035#15045035

I have scatterplot of two variables, for instance this: I would like to find the function that better fits the relation between these two variables. to be precise I would like to compare the fitting of three models: linear, exponential and logarithmic. I was thinking about fitting each function to my values, calculate the likelihoods in each case and compare the AIC values. But I don't really know how or where to start. Any possible help about this would be extremely appreciated.

```r
# find the function that better fits the relation between these two variables.
x<-c(0.108,0.111,0.113,0.116,0.118,0.121,0.123,0.126,0.128,0.131,0.133,0.136)

y<-c(-6.908,-6.620,-5.681,-5.165,-4.690,-4.646,-3.979,-3.755,-3.564,-3.558,-3.272,-3.073)
```

http://stackoverflow.com/a/15045035

Here is an example of comparing five models. Due to the form of the first two models we are able to use `lm` to get good starting values. (Note that models using different transforms of `y` should not be compared so we should not use `lm1` and `lm2` as comparison models but only for starting values.)

Now run an `nls` for each of the first two. After these two models we try polynomials of various degrees in `x`. Fortunately lm and nls use consistent `AIC` definitions (although its not necessarily true that other R model fitting functions have consistent AIC definitions) so we can just use `lm` for the polynomials.

Finally we plot the data and fits of the first two models.

The lower the AIC the better so `nls1` is best followed by `lm3.2` following by nls2.

```r
lm1 <- lm(1/y ~ x)
nls1 <- nls(y ~ 1/(a + b*x), start = setNames(coef(lm1), c("a", "b")))
AIC(nls1)
```

```
[1] -2.390924
```

```r
lm2 <- lm(1/y ~ log(x))
nls2 <- nls(y ~ 1/(a + b*log(x)), start = setNames(coef(lm2), c("a", "b")))
AIC(nls2)
```

```
[1] -1.29101
```

```
lm3.1 <- lm(y ~ x)
AIC(lm3.1) # 13.43161
```

```
[1] 13.43161
```

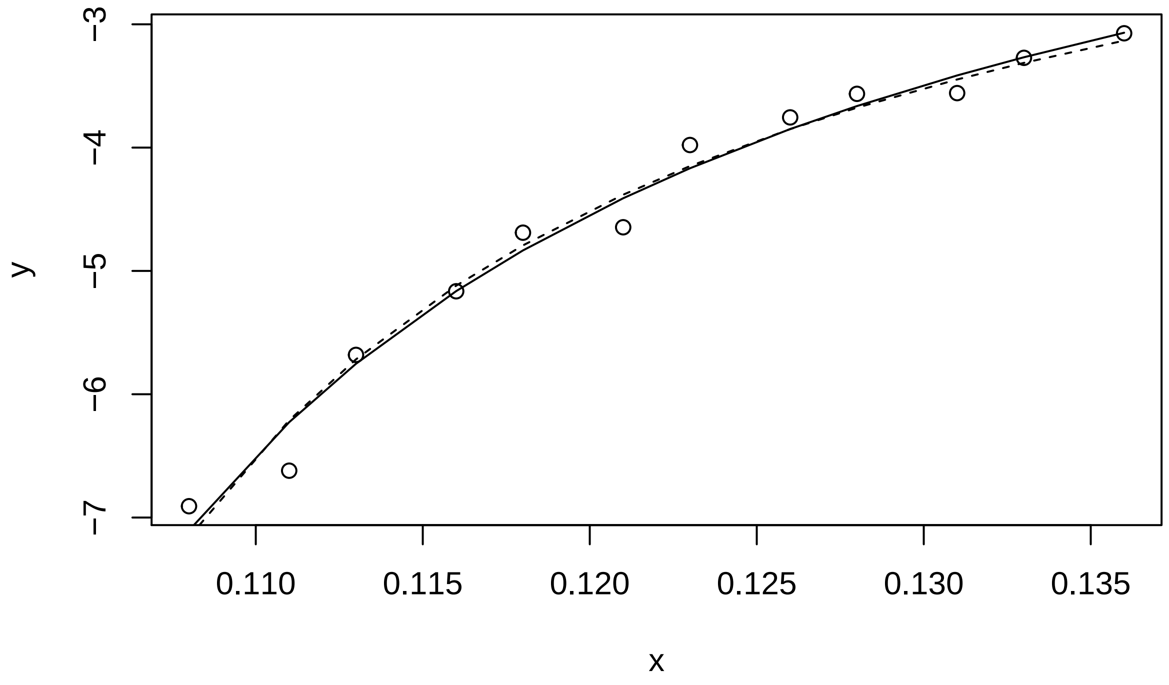
```
lm3.2 <- lm(y ~ poly(x, 2))
AIC(lm3.2) # -1.525982
```

```
[1] -1.525982
```

```
lm3.3 <- lm(y ~ poly(x, 3))
AIC(lm3.3) # 0.1498972
```

```
[1] 0.1498972
```

```
plot(y ~ x)
lines(fitted(nls1) ~ x, lty = 1) # solid line
lines(fitted(nls2) ~ x, lty = 2) # dashed line
```



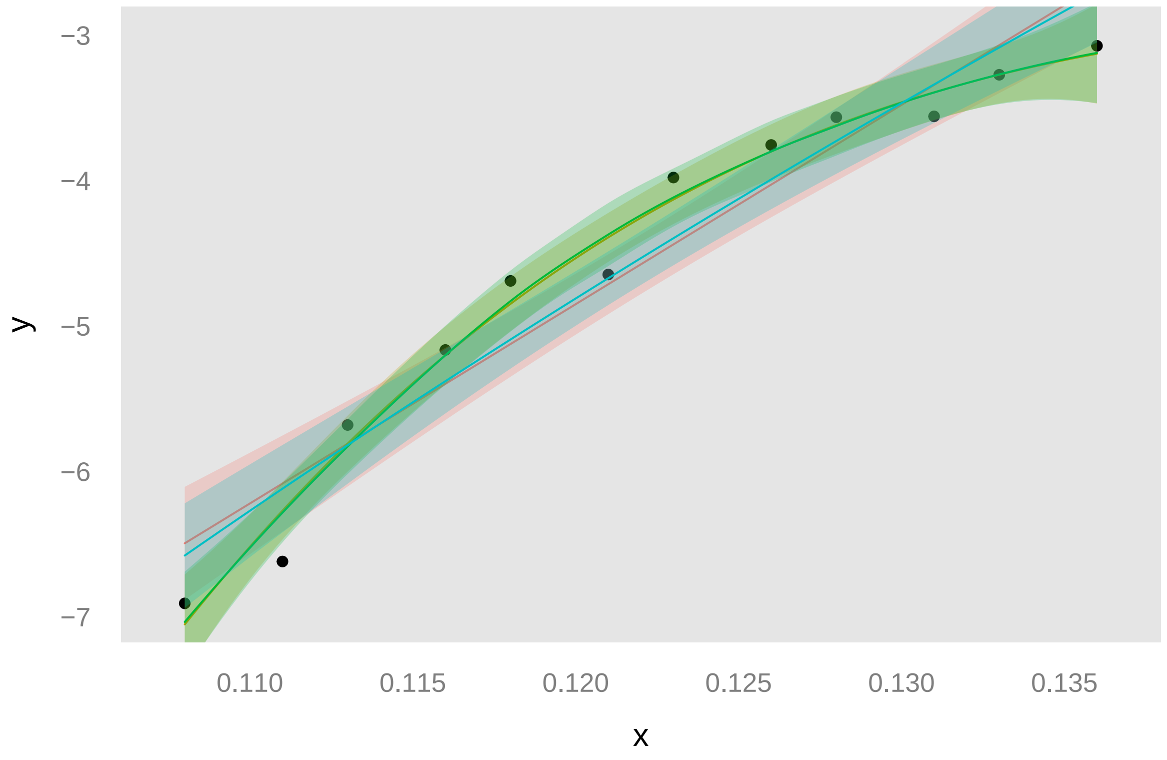
ADDED a few more models and subsequently fixed them up and changed notation. Also to follow up on Ben Bolker's comment we can replace AIC everywhere above with AICc from the AICcmodavg package.

http://stackoverflow.com/a/15044001

I would begin by an explanqtory plots, something like this :

```
library(latticeExtra)
library(grid)

x<-c(0.108,0.111,0.113,0.116,0.118,0.121,0.123,0.126,0.128,0.131,0.133,0.136)
y<-c(-6.908,-6.620,-5.681,-5.165,-4.690,-4.646,-3.979,-3.755,-3.564,-3.558,-3.272,-3.073)

dat <- data.frame(y=y,x=x)
```

```
xyplot(y ~ x,data=dat,par.settings = ggplot2like(),
       panel = function(x,y,...){
         panel.xyplot(x,y,...)
       })+
  layer(panel.smoother(y ~ x, method = "lm"), style =1)+  ## linear
  layer(panel.smoother(y ~ poly(x, 3), method = "lm"), style = 2)+  ## cubic
  layer(panel.smoother(y ~ x, span = 0.9),style=3)  + ### loeess
  layer(panel.smoother(y ~ log(x), method = "lm"), style = 4)  ## log
```



```
summary(lm(y~poly(x,3),data=dat))
```

```

Call:
lm(formula = y ~ poly(x, 3), data = dat)

Residuals:
     Min       1Q   Median       3Q      Max
-0.34959 -0.04177  0.04965  0.13216  0.15772

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -4.57592    0.05675 -80.631 6.24e-13 ***
poly(x, 3)1  4.11670    0.19659  20.940 2.84e-08 ***
poly(x, 3)2 -0.99374    0.19659  -5.055 0.000983 ***
poly(x, 3)3  0.09201    0.19659   0.468 0.652269
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1966 on 8 degrees of freedom
Multiple R-squared:  0.9831,    Adjusted R-squared:  0.9767
```

```
F-statistic: 154.8 on 3 and 8 DF,  p-value: 2.013e-07
```

```r
summary(lm(y ~ log(x), data=dat))
```

```
Call:
lm(formula = y ~ log(x), data = dat)

Residuals:
    Min      1Q  Median      3Q     Max 
-0.5008 -0.2467  0.0820  0.2203  0.4173 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)   30.775      2.776   11.09 6.14e-07 ***
log(x)        16.784      1.317   12.74 1.66e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3255 on 10 degrees of freedom
Multiple R-squared:  0.942, Adjusted R-squared:  0.9362 
F-statistic: 162.3 on 1 and 10 DF,  p-value: 1.66e-07
```

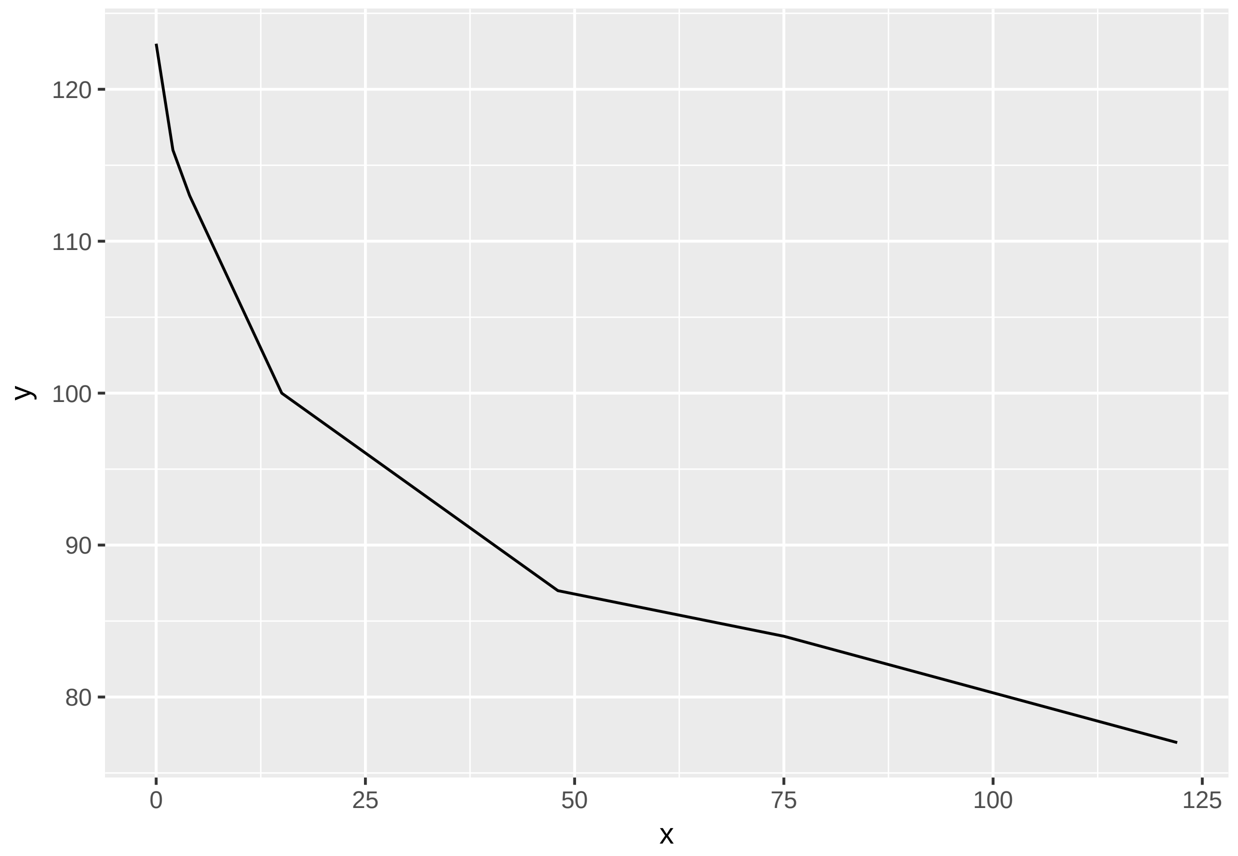
looks like you need a cubic model.

Fitting a function in R

http://stackoverflow.com/questions/11844522/fitting-a-function-in-r

I have a few datapoints (x and y) that seem to have a logarithmic relationship. Now I would like to find an underlying function that fits the graph and allows me to infer other datapoints (i.e. 3 or 82). I read about lm and nls but I'm not getting anywhere really. Can someone point me in the right direction on what to do from here?

```r
library(ggplot2)

# Importing your data
mydata <- read.table(text='
    x   y
1   0 123
2   2 116
3   4 113
4  15 100
5  48  87
6  75  84
7 122  77', header=T)

x <- mydata$x
y <- mydata$y

qplot(x, y, data=mydata, geom="line")
```



experimenting with the data

```r
lm1 <- lm(1/y ~ x)
nls1 <- nls(y ~ 1/(a + b*x), start = setNames(coef(lm1), c("a", "b")))
AIC(nls1)
```

```
[1] 48.56064
```

```r
# this will give error
lm2 <- lm(1/y ~ log(x))
```

```
Error in lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...): NA/NaN/Inf in 'x'
```

```r
nls2 <- nls(y ~ 1/(a + b*log(x)), start = setNames(coef(lm2), c("a", "b")))
```

```
Error in nls(y ~ 1/(a + b * log(x)), start = setNames(coef(lm2), c("a", : singular gradient
```

```r
AIC(nls2)
```

```
[1] -1.29101
```

```r
lm3.1 <- lm(y ~ x)
AIC(lm3.1) # 13.43161
```

```
[1] 52.4484
```

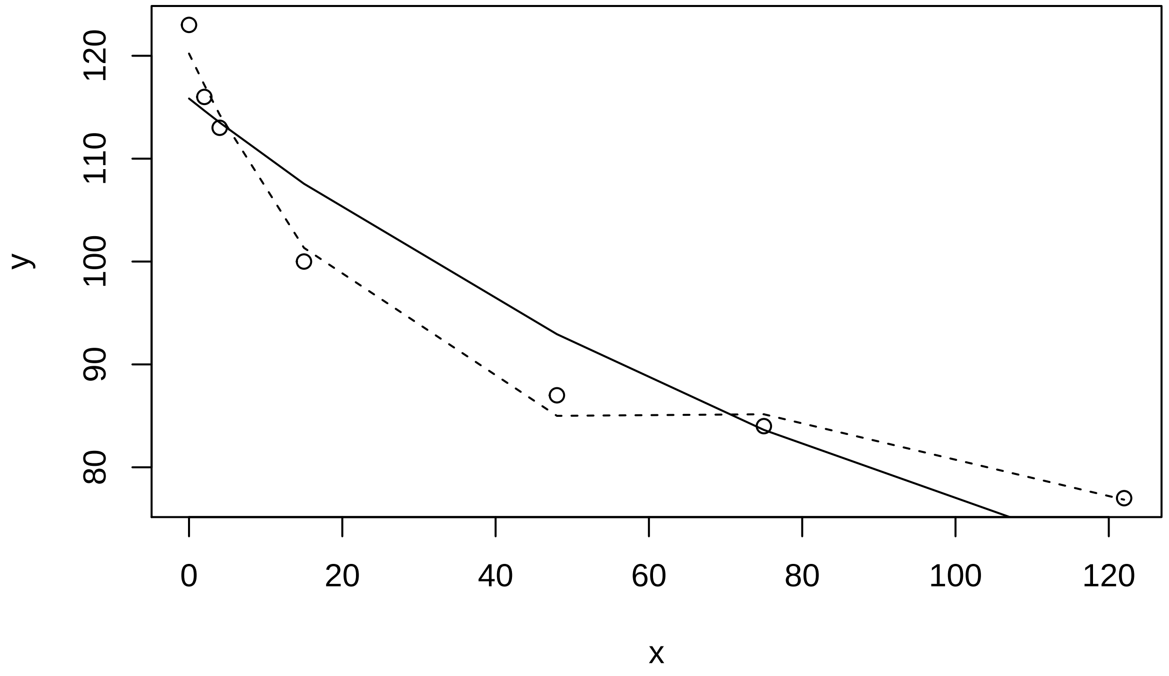
```r
lm3.2 <- lm(y ~ poly(x, 2))
AIC(lm3.2) # -1.525982
```

```
[1] 46.14859
```

```
lm3.3 <- lm(y ~ poly(x, 3))
AIC(lm3.3) # 0.1498972
```

```
[1] 36.42926
```

```
plot(y ~ x)
lines(fitted(nls1) ~ x, lty = 1) # solid line
lines(fitted(lm3.3) ~ x, lty = 2) # dashed line
```



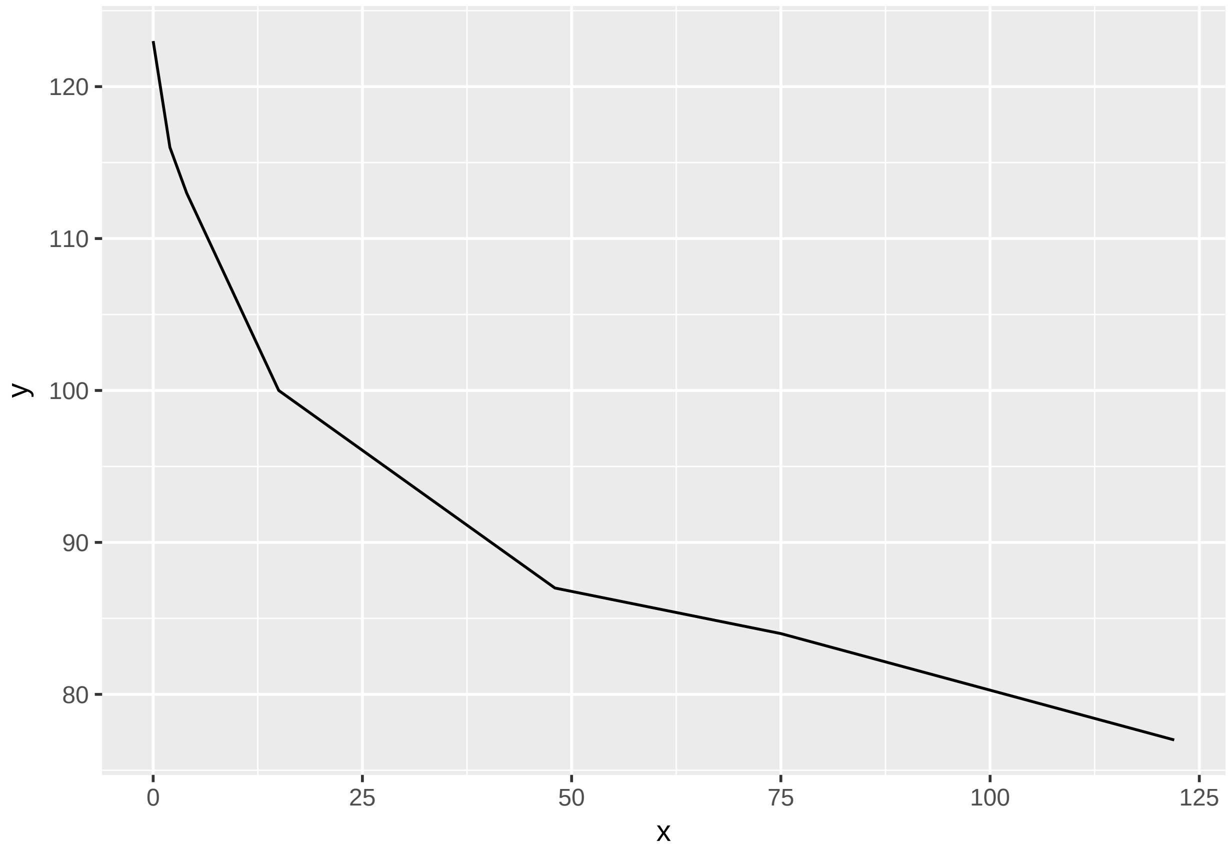
```
summary(lm(y ~ poly(x, 3), data = mydata))
```

```

Call:
lm(formula = y ~ poly(x, 3), data = mydata)

Residuals:
      1       2       3       4       5       6       7 
 2.7856 -1.1621 -1.2831 -1.3317  1.9978 -1.1559  0.1494 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 100.0000     0.9227 108.376 1.73e-06 ***
poly(x, 3)1 -39.9480     2.4413 -16.364 0.000497 ***
poly(x, 3)2  14.7236     2.4413   6.031 0.009139 ** 
poly(x, 3)3  -8.8033     2.4413  -3.606 0.036606 *  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.441 on 3 degrees of freedom
Multiple R-squared:  0.9906,    Adjusted R-squared:  0.9813 
F-statistic: 105.7 on 3 and 3 DF,  p-value: 0.001536
```

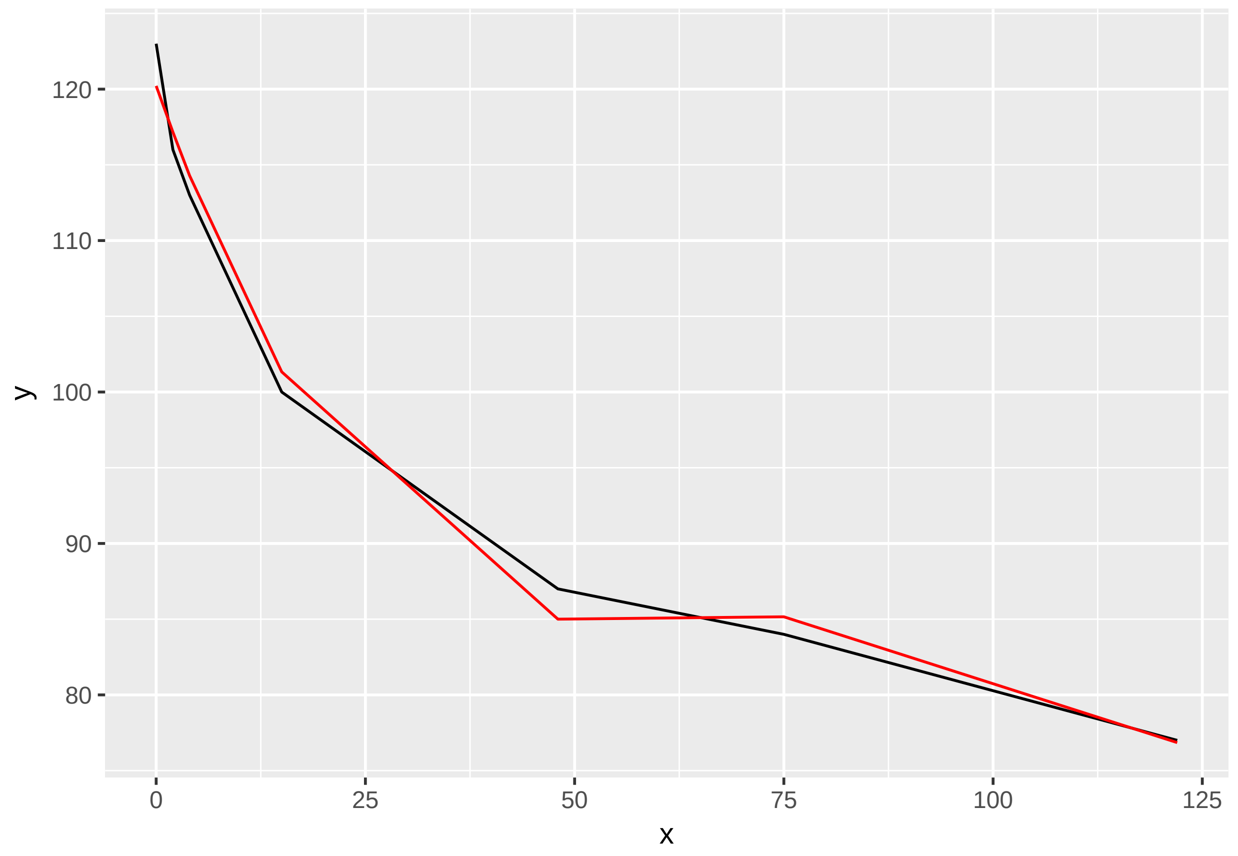
http://stackoverflow.com/a/11846275

Maybe using a cubic specification for your model and estimating via lm would give you a good fit.

```
library(ggplot2)

# Importing your data
dataset <- read.table(text='
    x   y
1   0 123
2   2 116
3   4 113
4  15 100
5  48  87
6  75  84
7 122  77', header=T)

# I think one possible specification would be a cubic linear model
y.hat <- predict(lm(y~x+I(x^2)+I(x^3), data=dataset))
# estimating the model and obtaining the fitted values from the model

qplot(x, y, data=dataset, geom="line") # your plot black lines
```



```
last_plot() + geom_line(aes(x=x, y=y.hat), col=2) # the fitted values red lines
```



```r
# It fits good.
```

Fitting polynomial model to data in R

http://stackoverflow.com/questions/3822535/fitting-polynomial-model-to-data-in-r

I've read the answers to this question and they are quite helpful, but I need help particularly in R.

I have an example data set in R as follows:

```r
x <- c(32,64,96,118,126,144,152.5,158)
y <- c(99.5,104.8,108.5,100,86,64,35.3,15)
```

I want to fit a model to these data so that y = f(x). I want it to be a 3rd order polynomial model.

How can I do that in R?

Additionally, can R help me to find the best fitting model?

http://stackoverflow.com/a/3822706

To get a third order polynomial in x (x^3), you can do

```r
lm(y ~ x + I(x^2) + I(x^3))
```

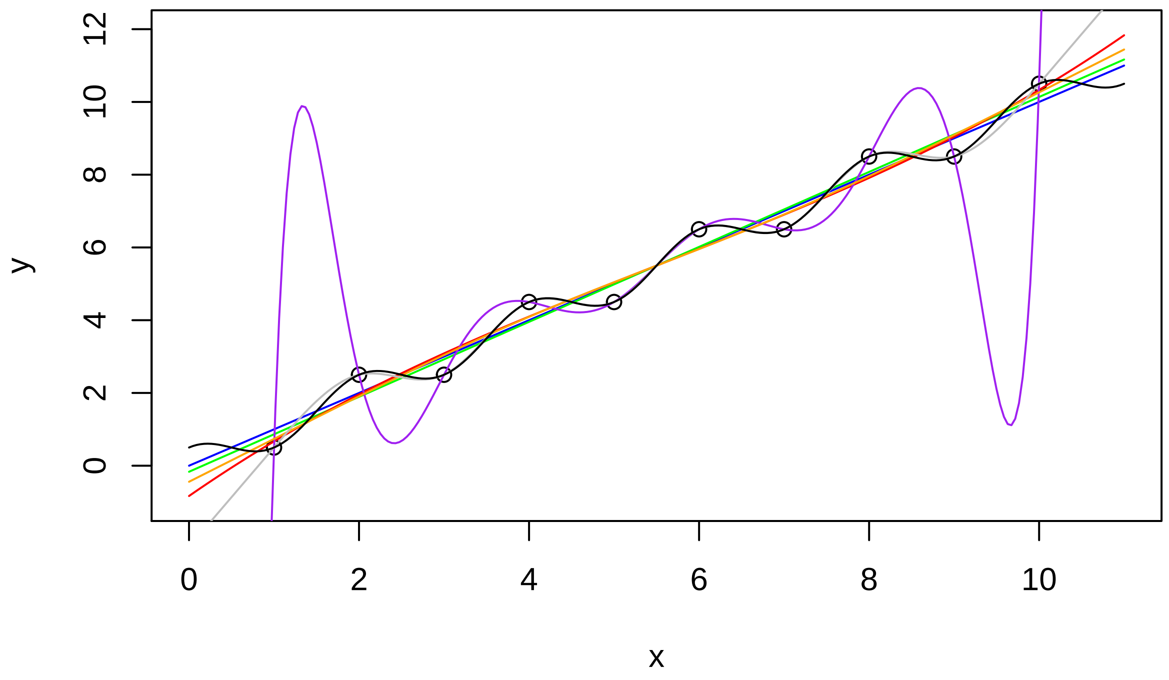
```

Call:
lm(formula = y ~ x + I(x^2) + I(x^3))
```

```
Coefficients:
(Intercept)            x       I(x^2)       I(x^3)
  1.269e+02   -1.626e+00    2.910e-02   -1.468e-04
```

http://stackoverflow.com/a/3824248

Which model is the "best fitting model" depends on what you mean by "best". R has tools to help, but you need to provide the definition for "best" to choose between them. Consider the following example data and code:

```r
x <- 1:10
y <- x + c(-0.5,0.5)

plot(x,y, xlim=c(0,11), ylim=c(-1,12))

fit1 <- lm( y~offset(x) -1 )
fit2 <- lm( y~x )
fit3 <- lm( y~poly(x,3) )
fit4 <- lm( y~poly(x,9) )

library(splines)
fit5 <- lm( y~ns(x, 3) )
fit6 <- lm( y~ns(x, 9) )

fit7 <- lm( y ~ x + cos(x*pi) )

xx <- seq(0,11, length.out=250)
lines(xx, predict(fit1, data.frame(x=xx)), col='blue')
lines(xx, predict(fit2, data.frame(x=xx)), col='green')
lines(xx, predict(fit3, data.frame(x=xx)), col='red')
lines(xx, predict(fit4, data.frame(x=xx)), col='purple')
lines(xx, predict(fit5, data.frame(x=xx)), col='orange')
lines(xx, predict(fit6, data.frame(x=xx)), col='grey')
lines(xx, predict(fit7, data.frame(x=xx)), col='black')
```



Which of those models is the best? arguments could be made for any of them (but I for one would not want to use the purple one for interpolation).

```
AIC(fit1)
```

```
[1] 16.51583
```

```
AIC(fit2)
```

```
[1] 20.20811
```

```
AIC(fit3)
```

```
[1] 23.40768
```

```
AIC(fit4)
```

```
[1] -Inf
```

```
AIC(fit5)
```

```
[1] 23.76997
```

```
AIC(fit6)
```

```
[1] -Inf
```

```
AIC(fit7)
```

```
[1] -659.2871
```

http://stackoverflow.com/a/3824080

Regarding the question 'can R help me find the best fitting model', there is probably a function to do this, assuming you can state the set of models to test, but this would be a good first approach for the set of n-1 degree polynomials:

```r
polyfit <- function(i) x <- AIC(lm(y ~ poly(x, i)))

as.integer(optimize(polyfit, interval = c(1,length(x)-1))$minimum)
```

```
[1] 2
```

Notes

- The validity of this approach will depend on your objectives, the assumptions of optimize() and AIC() and if AIC is the criterion that you want to use,
- polyfit() may not have a single minimum. check this with something like: for (i in 2:length(x)-1) print(polyfit(i))
- I used the as.integer() function because it is not clear to me how I would interpret a non-integer polynomial.
- for testing an arbitrary set of mathematical equations, consider the 'Eureqa' program reviewed by Andrew Gelman here

Update Also see the stepAIC function (in the MASS package) to automate model selection.

http://stackoverflow.com/a/12428183

The easiest way to find the best fit in R is to code the model as:

```r
lm.1 <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4))
lm.1
```

```


Call:
lm(formula = y ~ x + I(x^2) + I(x^3) + I(x^4))

Coefficients:
(Intercept)            x       I(x^2)       I(x^3)       I(x^4)
 -8.333e-01    1.622e+00   -1.282e-01    7.770e-03   -1.163e-16
```

After using step down AIC regression

```r
lm.s <- step(lm.1)
```

```
Start:  AIC=-4.97
y ~ x + I(x^2) + I(x^3) + I(x^4)

         Df Sum of Sq    RSS     AIC
- I(x^4)  1  0.000000 2.2378 -6.9711
- I(x^3)  1  0.002032 2.2398 -6.9620
- I(x^2)  1  0.009881 2.2476 -6.9270
- x       1  0.189611 2.4274 -6.1578
<none>                2.2378 -4.9711

Step:  AIC=-6.97
y ~ x + I(x^2) + I(x^3)

         Df Sum of Sq    RSS     AIC
- I(x^2)  1   0.18256 2.4203 -8.1869
- I(x^3)  1   0.18648 2.4242 -8.1707
<none>                2.2378 -6.9711
- x       1   1.24260 3.4804 -4.5545
```

```
Step:  AIC=-8.19
y ~ x + I(x^3)

         Df Sum of Sq     RSS      AIC
- I(x^3)  1    0.0039  2.4242 -10.1707
<none>                 2.4203  -8.1869
- x       1   11.6923 14.1126   7.4448

Step:  AIC=-10.17
y ~ x

       Df Sum of Sq    RSS     AIC
<none>               2.424 -10.171
- x     1    87.576 90.000  23.972
```

Curve fitting this data in R?

http://stackoverflow.com/questions/14645199/curve-fitting-this-data-in-r

For a few days I've been working on this problem and I'm stuck ...

I have performed a number of Monte Carlo simulations in R which gives an output y for each input x and there is clearly some simple relationship between x and y, so I want to identify the formula and its parameters. But I can't seem to get a good overall fit for both the 'Low x' and 'High x' series, e.g. using a logarithm like this:

```
data <- read.table(text='
      x          y
1   0.2 -0.7031864
2   0.3 -1.0533648
3   0.4 -1.3019655
4   0.5 -1.4919278
5   0.6 -1.6369545
6   0.7 -1.7477481
7   0.8 -1.8497117
8   0.9 -1.9300209
9   1.0 -2.0036842
10  1.1 -2.0659970
11  1.2 -2.1224324
12  1.3 -2.1693986
13  1.4 -2.2162889
14  1.5 -2.2548485
15  1.6 -2.2953162
16  1.7 -2.3249750
17  1.8 -2.3570141
18  1.9 -2.3872684
19  2.0 -2.4133978
20  2.1 -2.4359624
21  2.2 -2.4597122
22  2.3 -2.4818787
23  2.4 -2.5019371
24  2.5 -2.5173966
25  2.6 -2.5378936
26  2.7 -2.5549524
```

```
27  2.8 -2.5677939
28  2.9 -2.5865958
29  3.0 -2.5952558
30  3.1 -2.6120607
31  3.2 -2.6216831
32  3.3 -2.6370452
33  3.4 -2.6474608
34  3.5 -2.6576862
35  3.6 -2.6655606
36  3.7 -2.6763866
37  3.8 -2.6881303
38  3.9 -2.6932310
39  4.0 -2.7073198
40  4.1 -2.7165035
41  4.2 -2.7204063
42  4.3 -2.7278532
43  4.4 -2.7321731
44  4.5 -2.7444773
45  4.6 -2.7490365
46  4.7 -2.7554178
47  4.8 -2.7611471
48  4.9 -2.7719188
49  5.0 -2.7739299
50  5.1 -2.7807113
51  5.2 -2.7870781
52  5.3 -2.7950429
53  5.4 -2.7975677
54  5.5 -2.7990999
55  5.6 -2.8095955
56  5.7 -2.8142453
57  5.8 -2.8162046
58  5.9 -2.8240594
59  6.0 -2.8272394
60  6.1 -2.8338866
61  6.2 -2.8382038
62  6.3 -2.8401935
63  6.4 -2.8444915
64  6.5 -2.8448382
65  6.6 -2.8512086
66  6.7 -2.8550240
67  6.8 -2.8592950
68  6.9 -2.8622220
69  7.0 -2.8660817
70  7.1 -2.8710430
71  7.2 -2.8736998
72  7.3 -2.8764701
73  7.4 -2.8818748
74  7.5 -2.8832696
75  7.6 -2.8833351
76  7.7 -2.8891867
77  7.8 -2.8926849
78  7.9 -2.8944987
79  8.0 -2.8996780
```

```
80  8.1 -2.9011012
81  8.2 -2.9053911
82  8.3 -2.9063661
83  8.4 -2.9092228
84  8.5 -2.9135426
85  8.6 -2.9101730
86  8.7 -2.9186316
87  8.8 -2.9199631
88  8.9 -2.9199856
89  9.0 -2.9239220
90  9.1 -2.9240167
91  9.2 -2.9284608
92  9.3 -2.9294951
93  9.4 -2.9310985
94  9.5 -2.9352370
95  9.6 -2.9403694
96  9.7 -2.9395336
97  9.8 -2.9404153
98  9.9 -2.9437564
99 10.0 -2.9452175', header = TRUE)

x <- data$x
y <- data$y
```

I have also tried to fit (log10(x), 10^y) instead, which gives a good fit but the reverse transformation doesn't fit (x, y) very well.

Can anyone solve this?

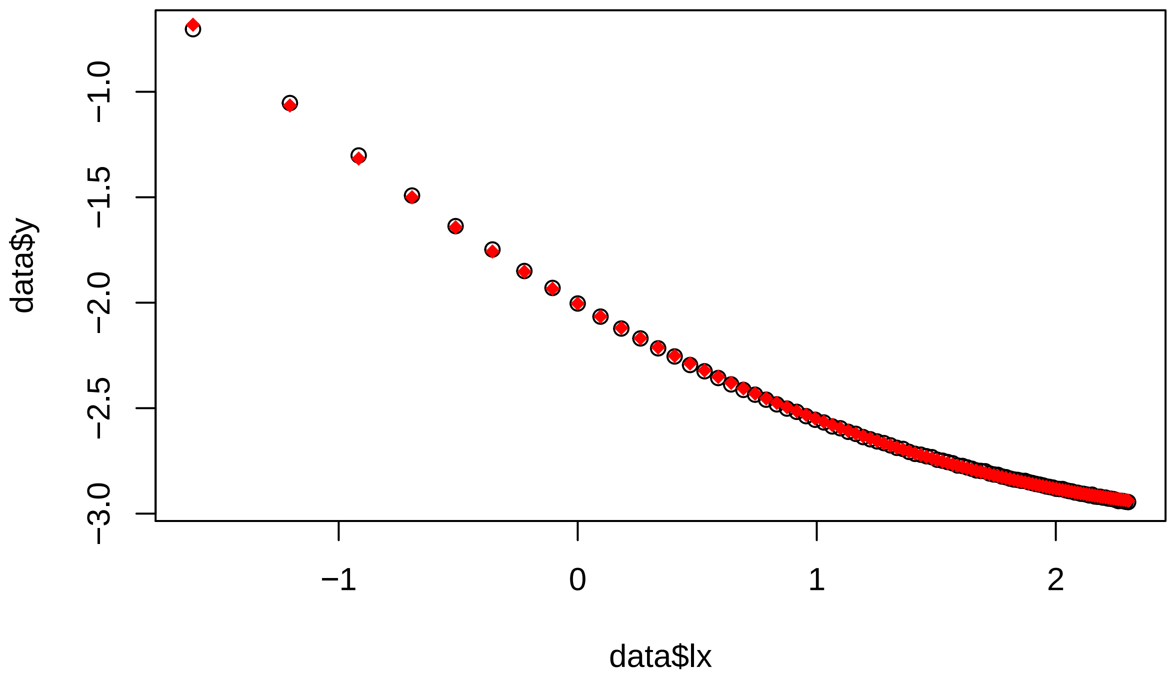
Please explain how you found the solution.

http://stackoverflow.com/a/14646118

Without an idea of the underlying process you may as well just fit a polynomial with as many components as you like. You don't seem to be testing a hypothesis (eg, gravitational strength is inverse-square related with distance) so you can fish all you like for functional forms, the data is unlikely to tell you which one is 'right'.

So if I read your data into a data frame with x and y components I can do:

```
data$lx = log(data$x)
plot(data$lx, data$y) # needs at least a cubic polynomial
m1 = lm(y~poly(lx,3), data=data) # fit a cubic
points(data$lx, fitted(m1), pch = 18, col = "red")
```



and the fitted points are pretty close. Change the polynomial degree from 3 to 7 and the points are identical. Does that mean that your Y values are really coming from a 7-degree polynomial of your X values? No. But you've got a curve that goes through the points.

At this scale, you may as well just join adjacent points up with a straight line, your plot is so smooth. But without underlying theory of why Y depends on X (like an inverse square law, or exponential growth, or something) all you are doing is joining the dots, and there are infinite ways of doing that.

Fitting a curve to specific data

http://stackoverflow.com/questions/14190883/fitting-a-curve-to-specific-data I have the following data in my thesis:

```
28 45
91 14
102 11
393 5
4492 1.77
```

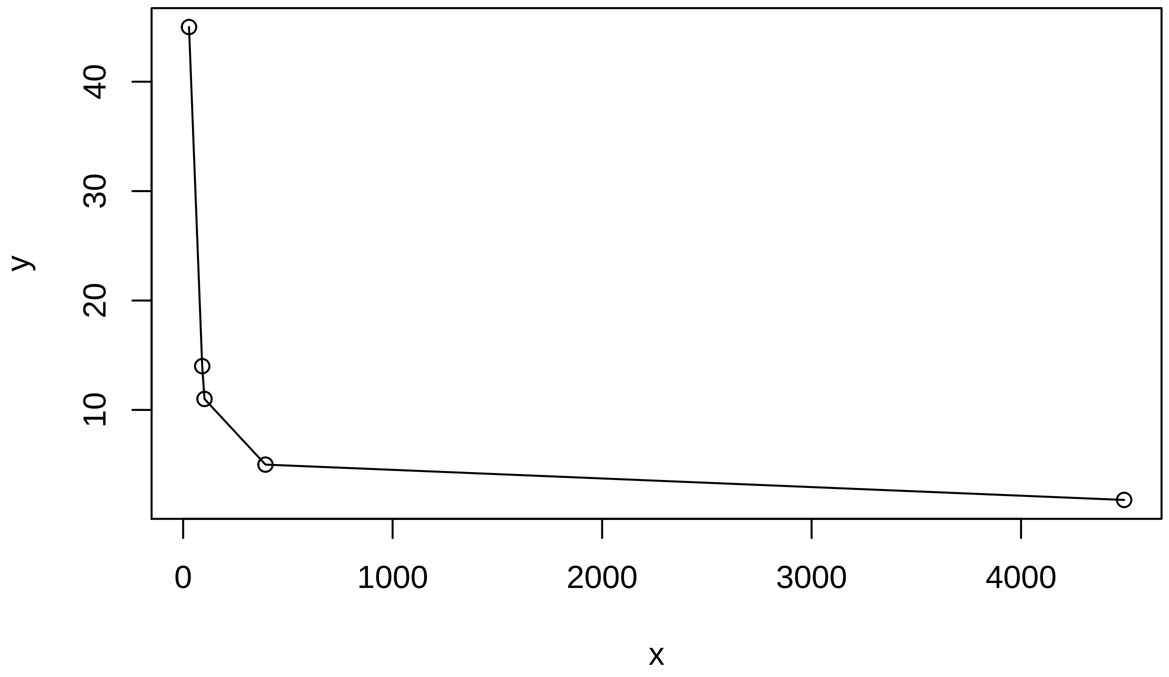
I need to fit a curve into this.

http://stackoverflow.com/a/15050715

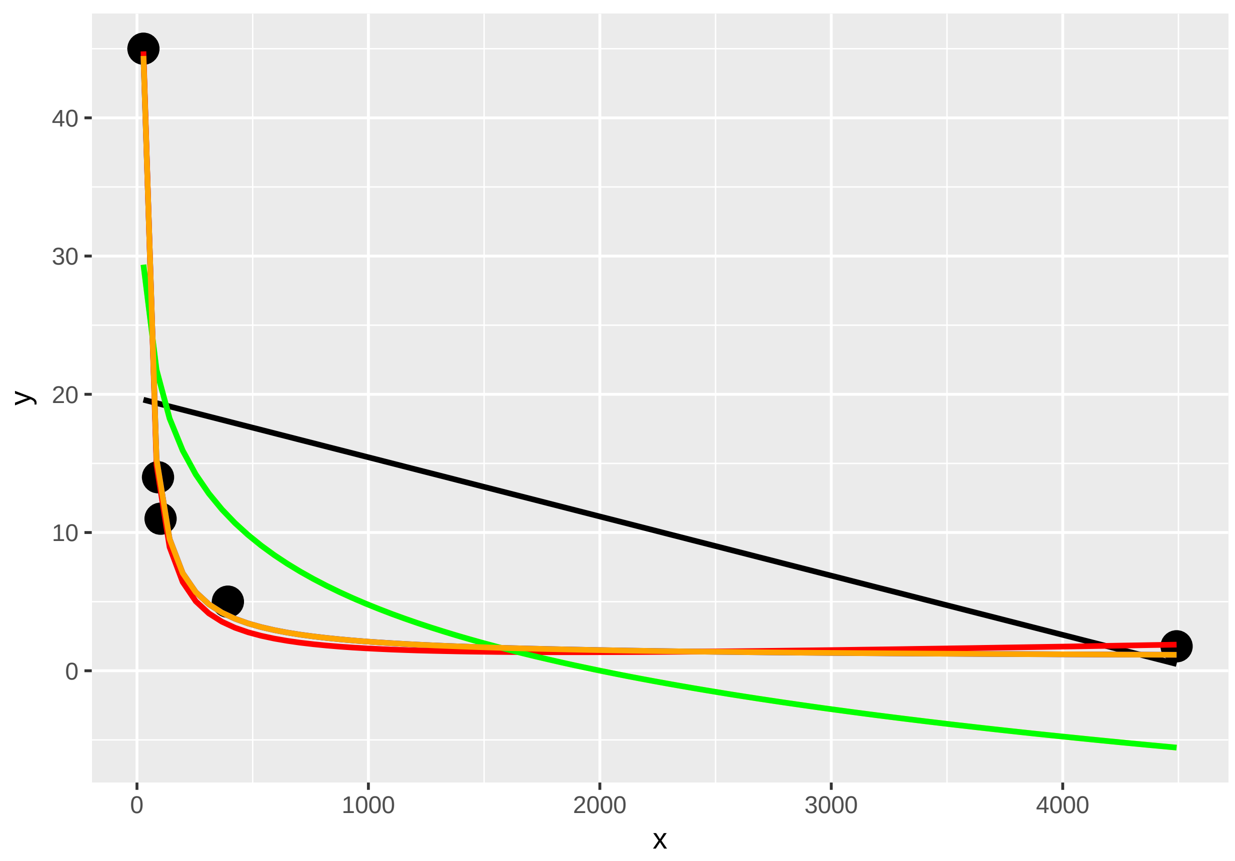
Just in case R is an option, here's a sketch of two methods you might use.

First method: evaluate the goodness of fit of a set of candidate models This is probably the best way as it takes advantage of what you might already know or expect about the relationship between the variables.

```
# read in the data
dat <- read.table(text= "x y
28 45
91 14
102 11
393 5
4492 1.77", header = TRUE)

# quick visual inspection
plot(dat); lines(dat)
```



```
 # a smattering of possible models... just made up on the spot
    # with more effort some better candidates should be added
# a smattering of possible models...
models <- list(
  lm(y~x, data = dat),
  lm(y~I(1/x), data=dat),
  lm(y ~ log(x), data = dat),
  nls(y ~ I(1/x*a) + b*x, data = dat, start = list(a = 1, b = 1)),
  nls(y ~ (a + b*log(x)), data=dat,
      start = setNames(coef(lm(y ~ log(x), data=dat)), c("a", "b"))),
  nls(y ~ I(exp(1)^(a + b * x)), data=dat, start = list(a=0,b=0)),
  nls(y ~ I(1/x*a)+b, data=dat, start = list(a=1,b=1))
)

# have a quick look at the visual fit of these models
library(ggplot2)
    ggplot(dat, aes(x, y)) + geom_point(size = 5) +
  stat_smooth(method = "lm", formula = as.formula(models[[1]]), size = 1, se = FALSE, colour = "black")
  stat_smooth(method = "lm", formula = as.formula(models[[2]]), size = 1, se = FALSE, colour = "blue")
  stat_smooth(method = "lm", formula = as.formula(models[[3]]), size = 1, se = FALSE, colour = "yellow"
  stat_smooth(method = "nls", formula = as.formula(models[[4]]), data=dat, start = list(a=0,b=0), size
  stat_smooth(method = "nls", formula = as.formula(models[[5]]), data=dat, start = setNames(coef(lm(y ~
  stat_smooth(method = "nls", formula = as.formula(models[[6]]), data=dat, start = list(a=0,b=0), size
  stat_smooth(method = "nls", formula = as.formula(models[[7]]), data=dat, start = list(a=0,b=0), size
```



The orange curve looks pretty good. Let's see how it ranks when we measure the relative goodness of fit of these models are...

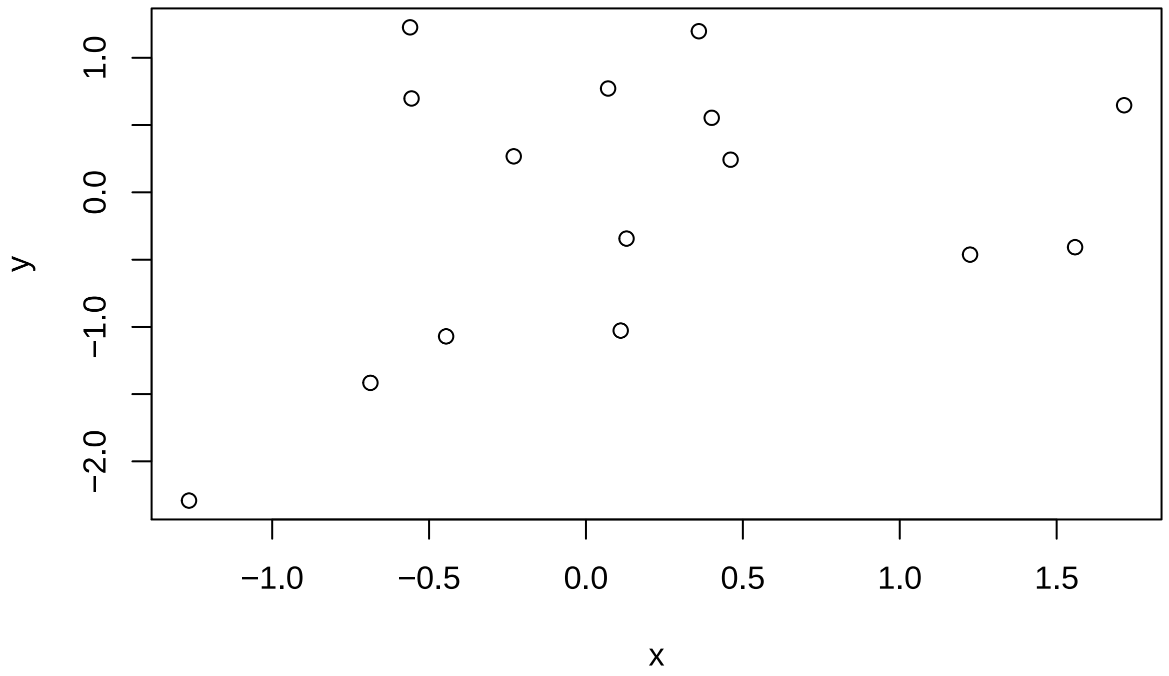
```
# calculate the AIC and AICc (for small samples) for each
# model to see which one is best, ie has the lowest AIC
library(AICcmodavg);
library(plyr);
library(stringr)

ldply(models, function(mod){ data.frame(AICc = AICc(mod), AIC = AIC(mod), model = deparse(formula(mod)))
```

```
      AICc      AIC                      model
1 70.23024 46.23024                      y ~ x
2 44.37075 20.37075                 y ~ I(1/x)
3 67.00075 43.00075                 y ~ log(x)
4 43.82083 19.82083     y ~ I(1/x * a) + b * x
5 67.00075 43.00075       y ~ (a + b * log(x))
6 52.75748 28.75748 y ~ I(exp(1)^(a + b * x))
7 44.37075 20.37075         y ~ I(1/x * a) + b
```

```
# y ~ I(1/x * a) + b * x is the best model of those tried here for this curve
# it fits nicely on the plot and has the best goodness of fit statistic
# no doubt with a better understanding of nls and the data a better fitting
# function could be found. Perhaps the optimisation method here might be
# useful also: http://stats.stackexchange.com/a/21098/7744
```

Predicting example

```
set.seed(123)
x <- rnorm(15)
y <- x + rnorm(15)
plot(x,y)
```



```
## Predictions
set.seed(123)
x <- rnorm(15)
y <- x + rnorm(15)
predict(lm(y ~ x))
```

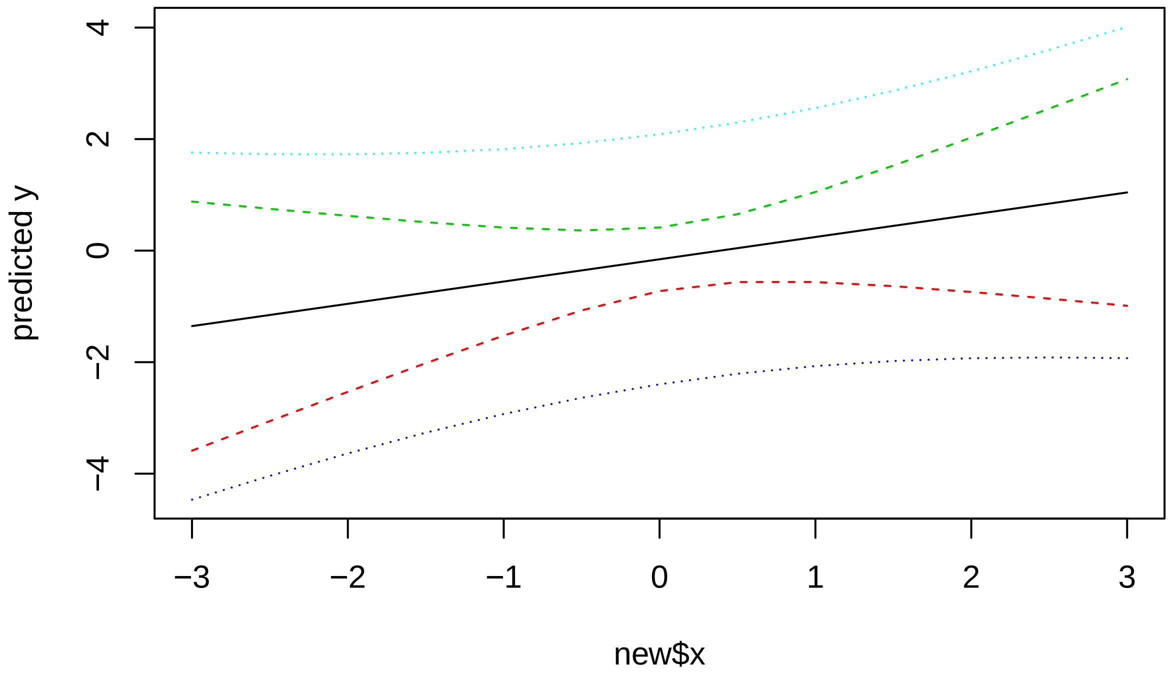
```
           1            2            3            4            5 
-0.378908173 -0.246994325  0.467447472 -0.126906972 -0.103431790 
           6            7            8            9           10 
 0.529892902  0.029013356 -0.660304220 -0.429380460 -0.333054106 
          11           12           13           14           15 
 0.333804973 -0.011364719  0.004992858 -0.110862227 -0.377057250 
```

```
new <- data.frame(x = seq(-3, 3, 0.5))
predict(lm(y ~ x), new, se.fit = TRUE)
```

```
$fit
          1           2           3           4           5           6 
-1.35320075 -1.15351171 -0.95382266 -0.75413362 -0.55444457 -0.35475552 
          7           8           9          10          11          12 
-0.15506648  0.04462257  0.24431161  0.44400066  0.64368970  0.84337875 
         13 
 1.04306779 

$se.fit
        1         2         3         4         5         6         7 
```

```
1.0336229 0.8808905 0.7307076 0.5850410 0.4483149 0.3317728 0.2636929
        8         9        10        11        12        13
0.2817241 0.3735998 0.5001589 0.6411824 0.7889515 0.9402914

$df
[1] 13

$residual.scale
[1] 1.003951
```

```
pred.w.plim <- predict(lm(y ~ x), new, interval = "prediction")
pred.w.clim <- predict(lm(y ~ x), new, interval = "confidence")
matplot(new$x, cbind(pred.w.clim, pred.w.plim[,-1]),
        lty = c(1,2,2,3,3), type = "l", ylab = "predicted y")
```



Using drhov vs friction factor data

```
# load the data from the curve
data <- read.table(text='
drhov   ff
3.42747 0.24203100
3.88830 0.16944600
4.85782 0.10225300
6.11430 0.07001020
8.41248 0.04450280
13.22850    0.02606930
24.30940    0.01258930
34.96940    0.00868380
45.34020    0.00659713
72.90190    0.00419354', header = TRUE)
```

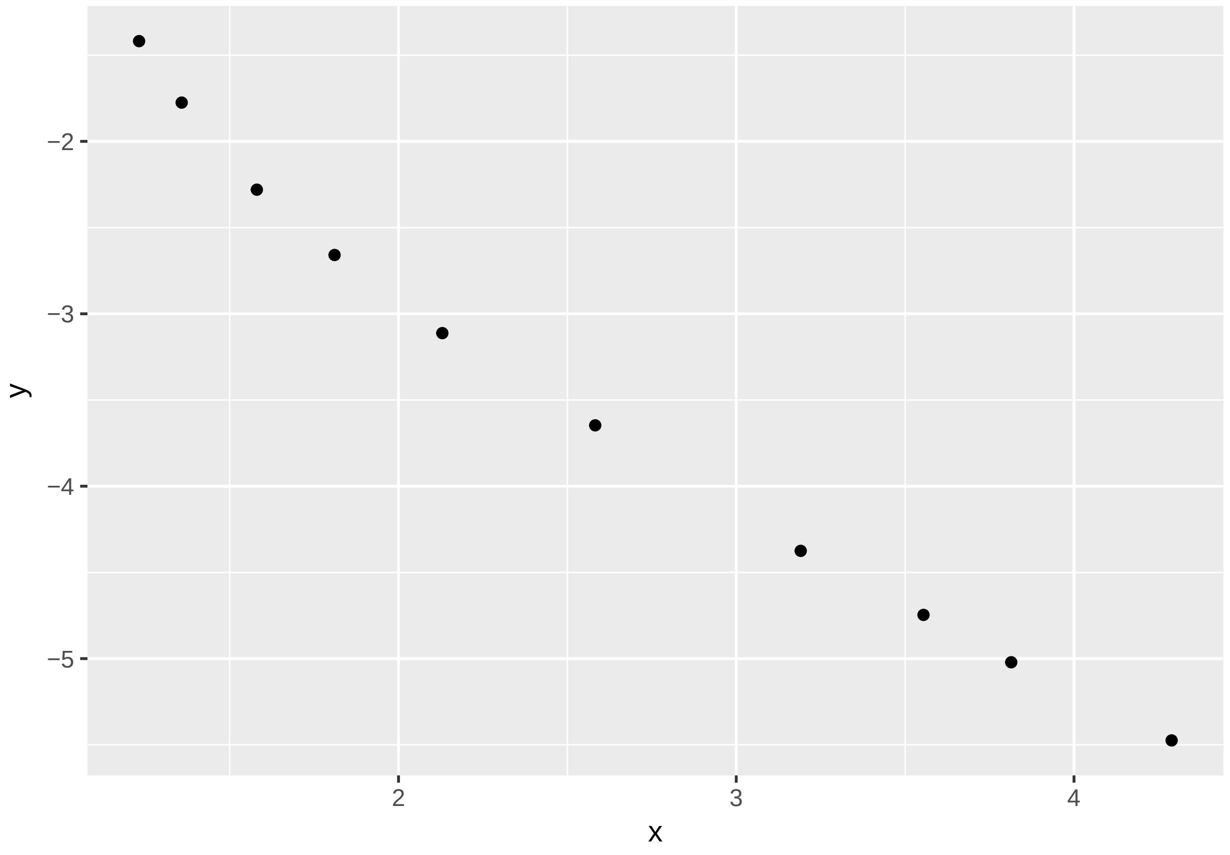
```
data
```

```
      drhov          ff
1   3.42747 0.24203100
2   3.88830 0.16944600
3   4.85782 0.10225300
4   6.11430 0.07001020
5   8.41248 0.04450280
6  13.22850 0.02606930
7  24.30940 0.01258930
8  34.96940 0.00868380
9  45.34020 0.00659713
10 72.90190 0.00419354
```

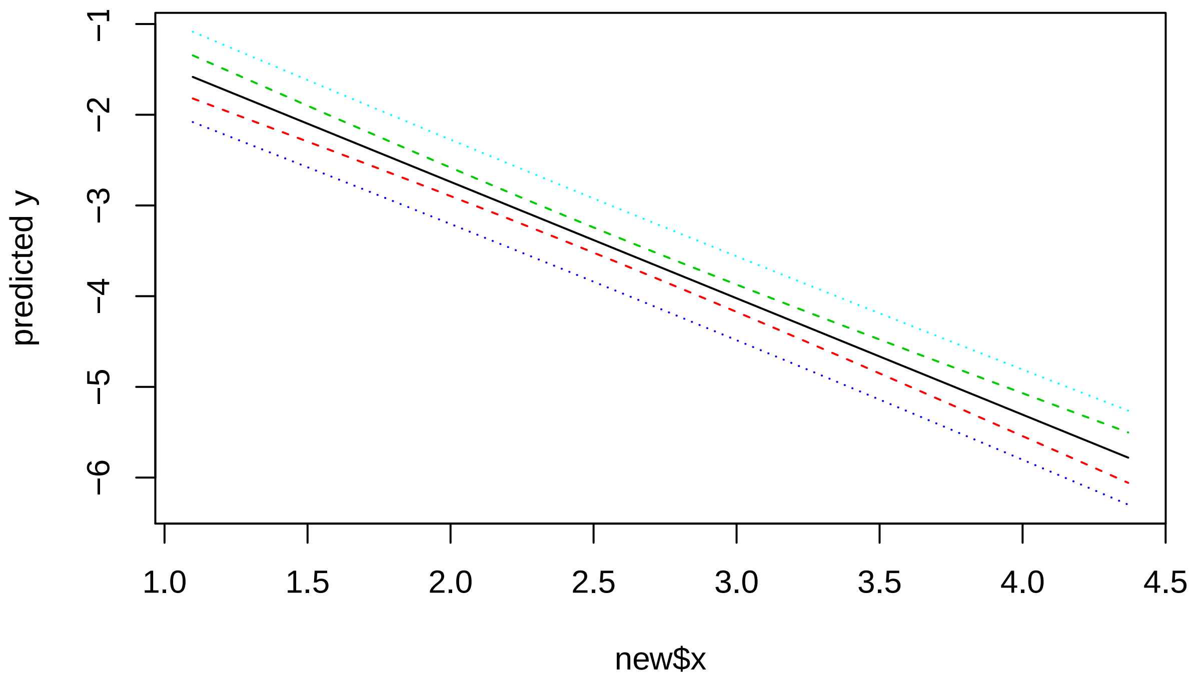
```
# convert the data to logarithmic
x <- log(data$drhov)
y <- log(data$ff)

models <- vector("list", 5)

predict(lm(y ~ x))
```

```
        1         2         3         4         5         6         7
-1.753990 -1.915873 -2.201548 -2.496749 -2.906218 -3.487093 -4.267940
        8         9        10
-4.734545 -5.067833 -5.677277
```

```
qplot(x,y)
```

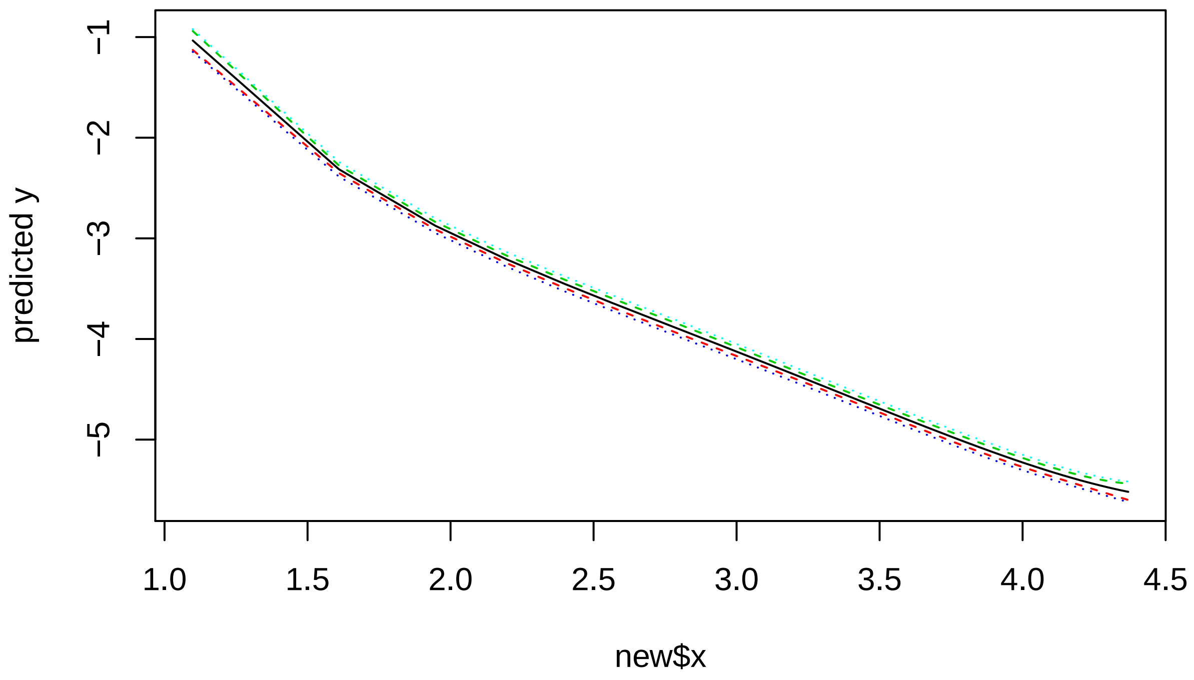


```r
# linear model y(x)
model <- lm(y ~ x)

# build a test dataframe
# new <- data.frame(x = seq(1, 4.5, 0.1))
new <- data.frame(x = log(seq(3, 80, 2)))     # dataframe with drhov sequence

prediction  <- predict(model, new, se.fit = TRUE)
pred.w.plim <- predict(model, new, interval = "prediction")
pred.w.clim <- predict(model, new, interval = "confidence")

matplot(new$x, cbind(pred.w.clim, pred.w.plim[,-1]),
        lty = c(1,2,2,3,3), type = "l", ylab = "predicted y")
```



```r
models[[1]] = model
```

```r
# linear model y = x + x^2 + x^3 + x^4
model <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4))

# dataframe with values to test prediction
new <- data.frame(x = log(seq(3, 80, 2)))     # dataframe with drhov sequence

pred.x4     <- predict(model, new, se.fit = TRUE)

pred.w.plim <- predict(model, new, interval = "prediction")
pred.w.clim <- predict(model, new, interval = "confidence")
matplot(new$x, cbind(pred.w.clim, pred.w.plim[,-1]),
        lty = c(1,2,2,3,3), type = "l", ylab = "predicted y")
```



```
models[[2]] = model
```

```
models
```

```
[[1]]

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
    -0.1732      -1.2833


[[2]]

Call:
lm(formula = y ~ x + I(x^2) + I(x^3) + I(x^4))

Coefficients:
(Intercept)            x       I(x^2)       I(x^3)       I(x^4)
    5.39208     -9.37645      4.11292     -0.90219      0.07352


[[3]]
NULL

[[4]]
NULL

[[5]]
NULL
```

```r
library(ggplot2)
## Test the model

# enter the values on the x axis: drhov, rounded to 2 decimals
drhov <- round(data$drhov, 2)

# call the model
model <- models[[2]]
summary(model)
```

```

Call:
lm(formula = y ~ x + I(x^2) + I(x^3) + I(x^4))

Residuals:
        1         2         3         4         5         6         7 
 0.015512 -0.009665 -0.023573  0.007556  0.012444  0.013999 -0.034784 
        8         9        10 
 0.006893  0.017679 -0.006062 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  5.39208    0.62650   8.607 0.000349 ***
x           -9.37645    1.07643  -8.711 0.000330 ***
I(x^2)       4.11292    0.65078   6.320 0.001461 ** 
I(x^3)      -0.90219    0.16489  -5.471 0.002778 ** 
I(x^4)       0.07352    0.01489   4.936 0.004336 ** 
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.0241 on 5 degrees of freedom
Multiple R-squared:  0.9998,    Adjusted R-squared:  0.9997 
F-statistic:  7823 on 4 and 5 DF,  p-value: 1.129e-09
```

```r
model$call            # print the formula for the model
```

```
lm(formula = y ~ x + I(x^2) + I(x^3) + I(x^4))
```

```r
# create the dataframe of the log(drhov)
new.df <- data.frame(x = c(log(drhov)))

# get prediction values
result <- predict(model, new.df, se.fit = TRUE)
result$fit
```

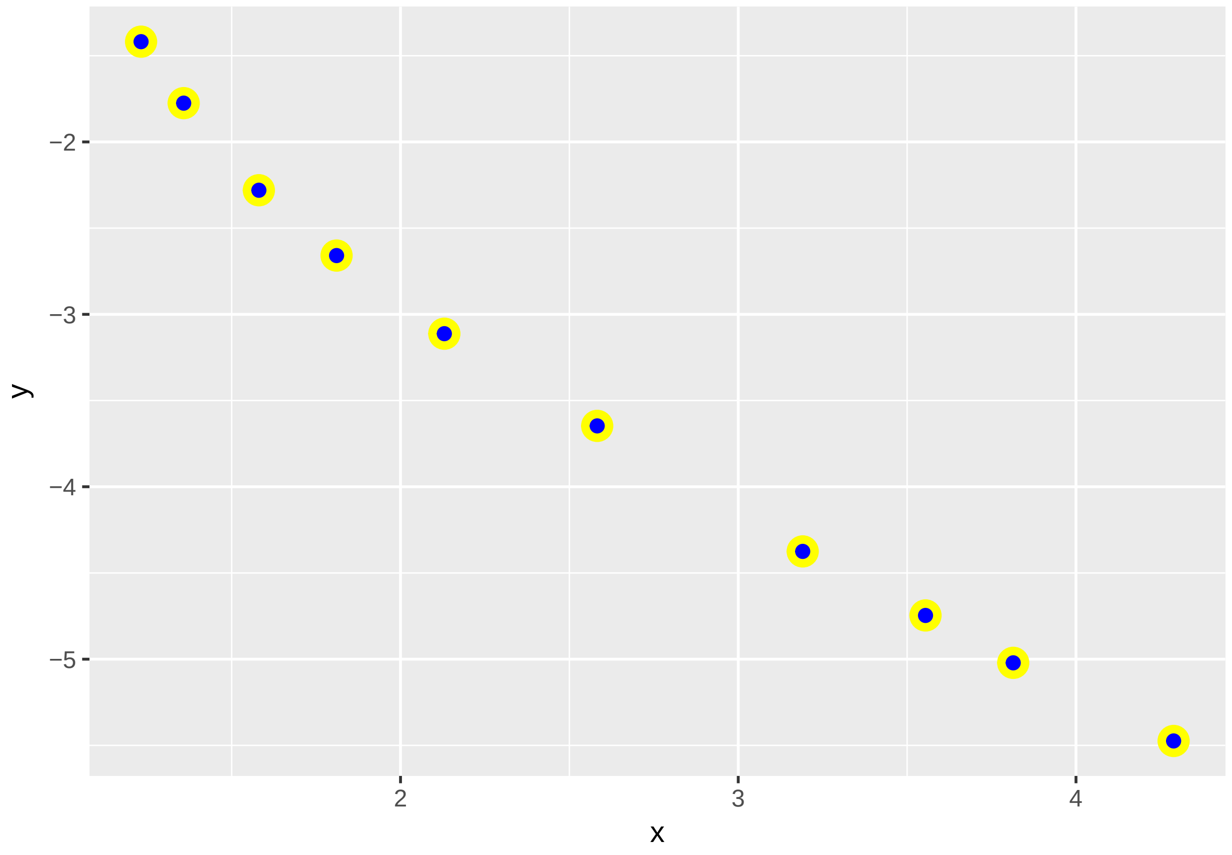
```
        1         2         3         4         5         6         7 
-1.436268 -1.766632 -2.257619 -2.665538 -3.124266 -3.661123 -4.340152 
        8         9        10 
-4.753209 -5.038795 -5.468130 
```

```r
# get back the friction factor after e^result
exp(result$fit)
```

```
          1           2           3           4           5           6 
0.237813707 0.170907707 0.104599291 0.069561951 0.043969213 0.025703639 
          7           8           9          10 
```

```
0.013034542 0.008623980 0.006481555 0.004219114
```

```r
# plot the experimental data
g <- ggplot(data, aes(x, y)) + geom_point(size = 5, col = "yellow")

# add the points resulting from the model applied
g + geom_point(method = "lm",
               formula = as.formula(model),
               size = 2, se = TRUE, colour = "blue")
```



```r
g + geom_point(log(drhov), result$fit)
```

```
Error: ggplot2 doesn't know how to deal with data of class numeric
```

```r
ggplot(aes(x = log(drhov), y = result$fit))
```

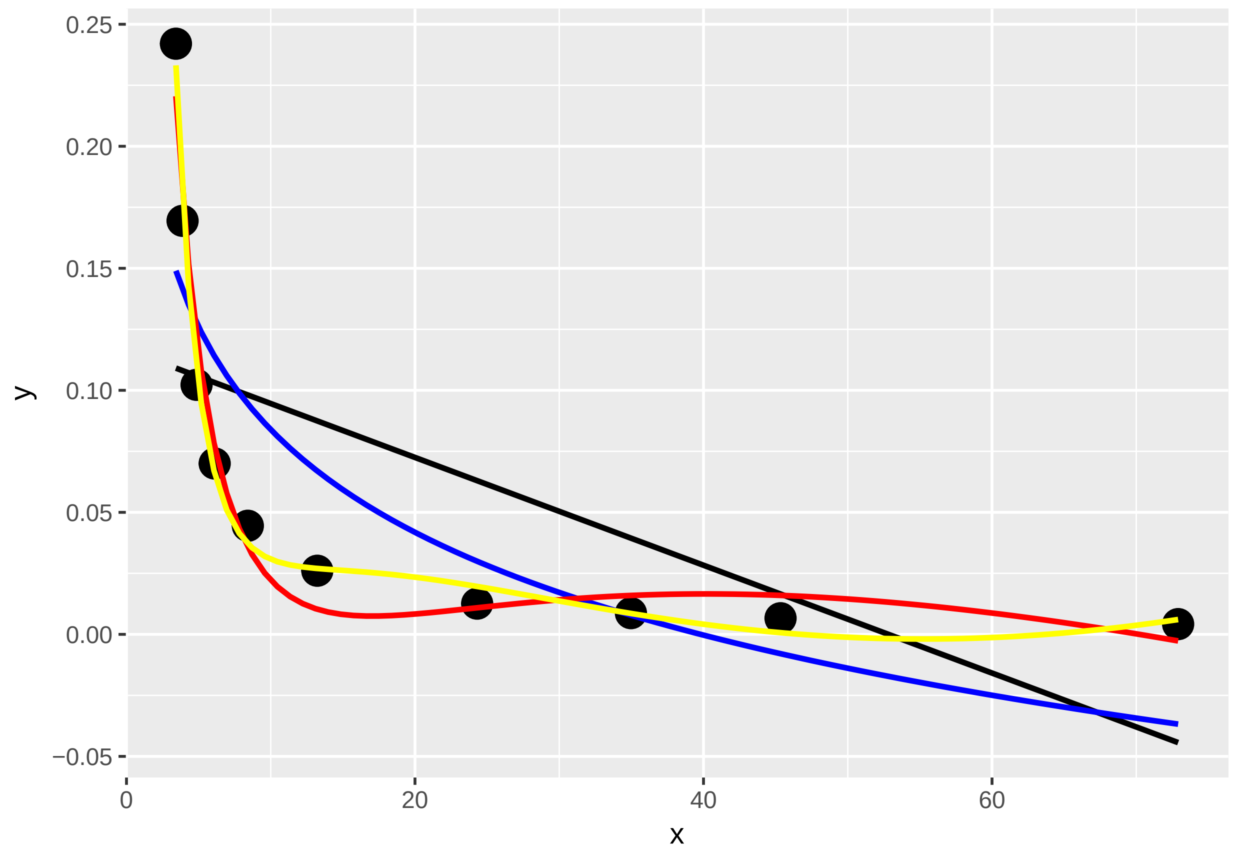
```
Error: ggplot2 doesn't know how to deal with data of class uneval
```

Interpolation without linearizing data

```r
# load the data from the curve
data <- read.table(text='
drhov   ff
3.42747 0.24203100
3.88830 0.16944600
4.85782 0.10225300
6.11430 0.07001020
8.41248 0.04450280
13.22850    0.02606930
24.30940    0.01258930
34.96940    0.00868380
45.34020    0.00659713
72.90190    0.00419354', header = TRUE)

x <- data$drhov
y <- data$ff
data
```

```
      drhov         ff
1   3.42747 0.24203100
2   3.88830 0.16944600
3   4.85782 0.10225300
4   6.11430 0.07001020
5   8.41248 0.04450280
6  13.22850 0.02606930
7  24.30940 0.01258930
8  34.96940 0.00868380
9  45.34020 0.00659713
10 72.90190 0.00419354
```

```
# model <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4))
models <- list(
  lm(y ~ x, data = data),
  lm(y ~ log(x), data = data),
  lm(y ~ log(x) + I(log(x)^2) + I(log(x)^3) ),
  lm(y ~ log(x) + I(log(x)^2) + I(log(x)^3) + I(log(x)^4) ),
  lm(y ~ log(x) + I(log(x)^2) + I(log(x)^3) + I(log(x)^4)  + I(log(x)^5) )

)

# have a quick look at the visual fit of these models
library(ggplot2)
ggplot(data, aes(x, y)) + geom_point(size = 5) +
  stat_smooth(method = "lm", formula = as.formula(models[[1]]), size = 1, se = FALSE, colour = "black")
  stat_smooth(method = "lm", formula = as.formula(models[[2]]), size = 1, se = FALSE, colour = "blue") +
      stat_smooth(method = "lm", formula = as.formula(models[[3]]), size = 1, se = FALSE, colour = "red
      stat_smooth(method = "lm", formula = as.formula(models[[4]]), size = 1, se = FALSE, colour = "yel
```



```
    stat_smooth(method = "lm", formula = as.formula(models[[5]]), size = 1, se = FALSE, colour = "gre
```

```
geom_smooth: na.rm = FALSE
stat_smooth: method = lm, formula = y ~ log(x) + I(log(x)^2) + I(log(x)^3) + I(log(x)^4) + I(log(x)^5),
position_identity
```

```
models
```

```
[[1]]

Call:
lm(formula = y ~ x, data = data)

Coefficients:
(Intercept)            x
   0.116665    -0.002209


[[2]]

Call:
lm(formula = y ~ log(x), data = data)

Coefficients:
(Intercept)       log(x)
    0.22385     -0.06077
```

```
[[3]]

Call:
lm(formula = y ~ log(x) + I(log(x)^2) + I(log(x)^3))

Coefficients:
(Intercept)       log(x)  I(log(x)^2)  I(log(x)^3)  
    0.95810     -0.90145      0.28110     -0.02871  


[[4]]

Call:
lm(formula = y ~ log(x) + I(log(x)^2) + I(log(x)^3) + I(log(x)^4))

Coefficients:
(Intercept)       log(x)  I(log(x)^2)  I(log(x)^3)  I(log(x)^4)  
    1.80707     -2.39138      1.19468     -0.26190      0.02111  


[[5]]

Call:
lm(formula = y ~ log(x) + I(log(x)^2) + I(log(x)^3) + I(log(x)^4) + 
    I(log(x)^5))

Coefficients:
(Intercept)       log(x)  I(log(x)^2)  I(log(x)^3)  I(log(x)^4)  
    3.26302     -5.60689      3.89171     -1.33820      0.22626  
I(log(x)^5)  
   -0.01501  
```

It is difficult to discern which model fits the data better, if P(x^3) or P(x^5) make any difference.

```r
summary(models[[3]])
```

```

Call:
lm(formula = y ~ log(x) + I(log(x)^2) + I(log(x)^3))

Residuals:
      Min        1Q    Median        3Q       Max 
-0.019894 -0.008842 -0.002649  0.008192  0.021499 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept)  0.958095   0.121970   7.855 0.000225 ***
log(x)      -0.901450   0.153711  -5.865 0.001087 ** 
I(log(x)^2)  0.281096   0.059307   4.740 0.003193 ** 
I(log(x)^3) -0.028712   0.007154  -4.013 0.007012 ** 
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.01603 on 6 degrees of freedom
Multiple R-squared:  0.9736,    Adjusted R-squared:  0.9605 
```

```
F-statistic:  73.9 on 3 and 6 DF,  p-value: 3.963e-05
```

```r
summary(models[[5]])
```

```

Call:
lm(formula = y ~ log(x) + I(log(x)^2) + I(log(x)^3) + I(log(x)^4) +
    I(log(x)^5))

Residuals:
         1          2          3          4          5          6
 0.0033806 -0.0052114 -0.0010064  0.0042358  0.0005017 -0.0035580
         7          8          9         10
 0.0017294  0.0018653 -0.0023433  0.0004064

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.26302    0.40467   8.063  0.00128 **
log(x)      -5.60689    0.87858  -6.382  0.00309 **
I(log(x)^2)  3.89171    0.72741   5.350  0.00589 **
I(log(x)^3) -1.33820    0.28781  -4.650  0.00966 **
I(log(x)^4)  0.22626    0.05461   4.143  0.01434 *
I(log(x)^5) -0.01501    0.00399  -3.762  0.01975 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.004544 on 4 degrees of freedom
Multiple R-squared:  0.9986,    Adjusted R-squared:  0.9968
F-statistic: 565.8 on 5 and 4 DF,  p-value: 8.708e-06
```

Visually, the linearization of the data seems like the best choice.

Simple nonlinear least squares curve fitting in R

http://www.walkingrandomly.com/?p=5254

The R code used for this example comes from Barry Rowlingson, so huge thanks to him.

A question I get asked a lot is 'How can I do nonlinear least squares curve fitting in X?' where X might be MATLAB, Mathematica or a whole host of alternatives. Since this is such a common query, I thought I'd write up how to do it for a very simple problem in several systems that I'm interested in

The problem

```r
xdata = c(-2,-1.64,-1.33,-0.7,0,0.45,1.2,1.64,2.32,2.9)
ydata = c(0.699369,0.700462,0.695354,1.03905,1.97389,2.41143,1.91091,0.919576,-0.730975,-1.42001)
```

and you'd like to fit the function

$$F(p_1, p_2, x) = p_1 cos(p_2 x) + p_2 sin(p_1 x)$$

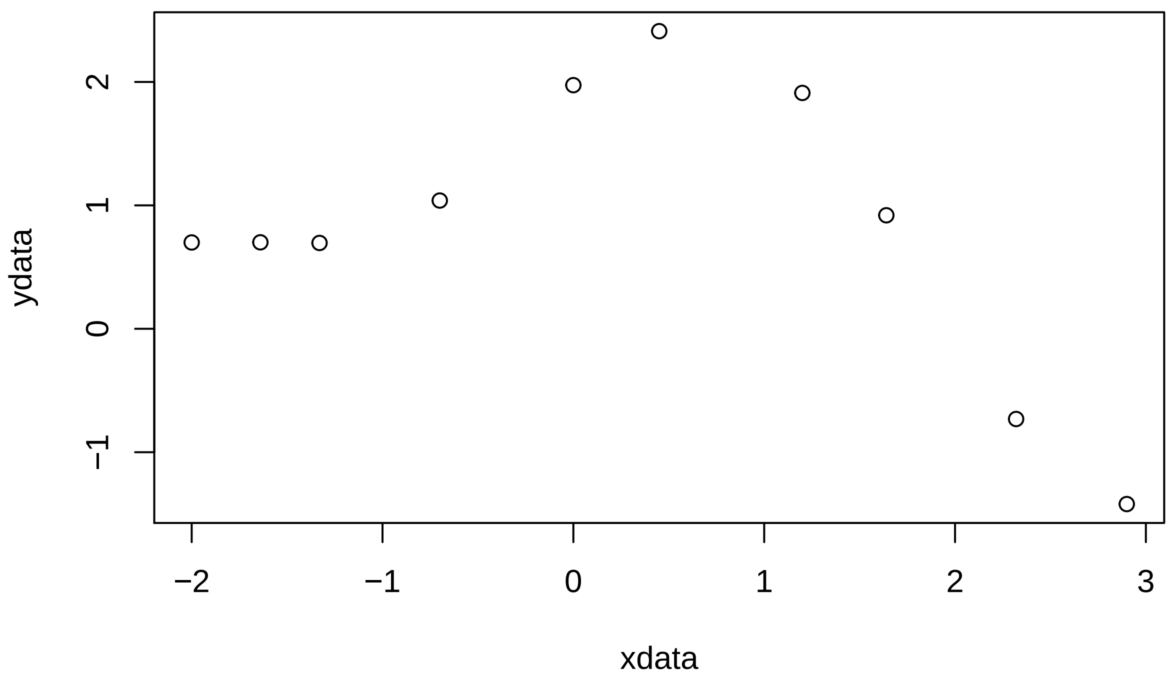
using nonlinear least squares. You're starting guesses for the parameters are p1=1 and P2=0.2

For now, we are primarily interested in the following results:

- The fit parameters
- Sum of squared residuals
- Parameter confidence intervals

plot data and build the model

```
# construct the data vectors using c()
xdata = c(-2,-1.64,-1.33,-0.7,0,0.45,1.2,1.64,2.32,2.9)
ydata = c(0.699369,0.700462,0.695354,1.03905,1.97389,2.41143,1.91091,0.919576,-0.730975,-1.42001)

# look at it
plot(xdata,ydata)
```



```
# some starting values
p1 = 1
p2 = 0.2

# do the fit
model = nls(ydata ~ p1*cos(p2*xdata) + p2*sin(p1*xdata), start = list(p1=p1, p2=p2))

# summarise
summary(model)
```

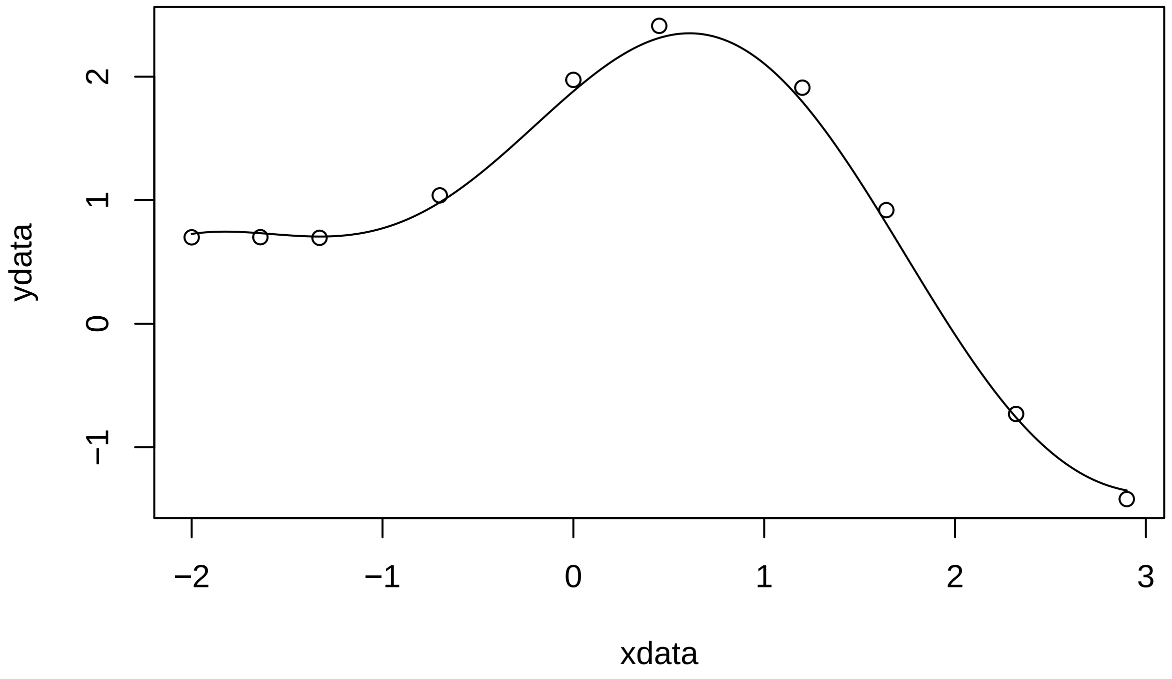
```
Formula: ydata ~ p1 * cos(p2 * xdata) + p2 * sin(p1 * xdata)

Parameters:
   Estimate Std. Error t value Pr(>|t|)
p1 1.881851   0.027430   68.61 2.27e-12 ***
p2 0.700230   0.009153   76.51 9.50e-13 ***
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.08202 on 8 degrees of freedom

Number of iterations to convergence: 7
Achieved convergence tolerance: 2.189e-06
```

See the fit

Draw the fit on the plot by getting the prediction from the fit at 200 x-coordinates across the range of xdata

```
new = data.frame(xdata = seq(min(xdata), max(xdata), len=200))

plot(xdata, ydata)
lines(new$xdata, predict(model, newdata=new))
```



other fitness analysis

Getting the sum of squared residuals is easy enough:

```
sum(resid(model)^2)   #0.0538127
```

```
[1] 0.0538127
```

Finally, lets get the parameter confidence intervals.

```
confint(model)
```

```
        2.5%     97.5%
p1 1.8206081 1.9442365
p2 0.6794193 0.7209843
```

```
#             2.5%     97.5%
#      p1 1.8206081 1.9442365
#      p2 0.6794193 0.7209843
```