Examples of Maximum Likelihood Estimation and Optimization in R

Joel S Steele

Univariate example

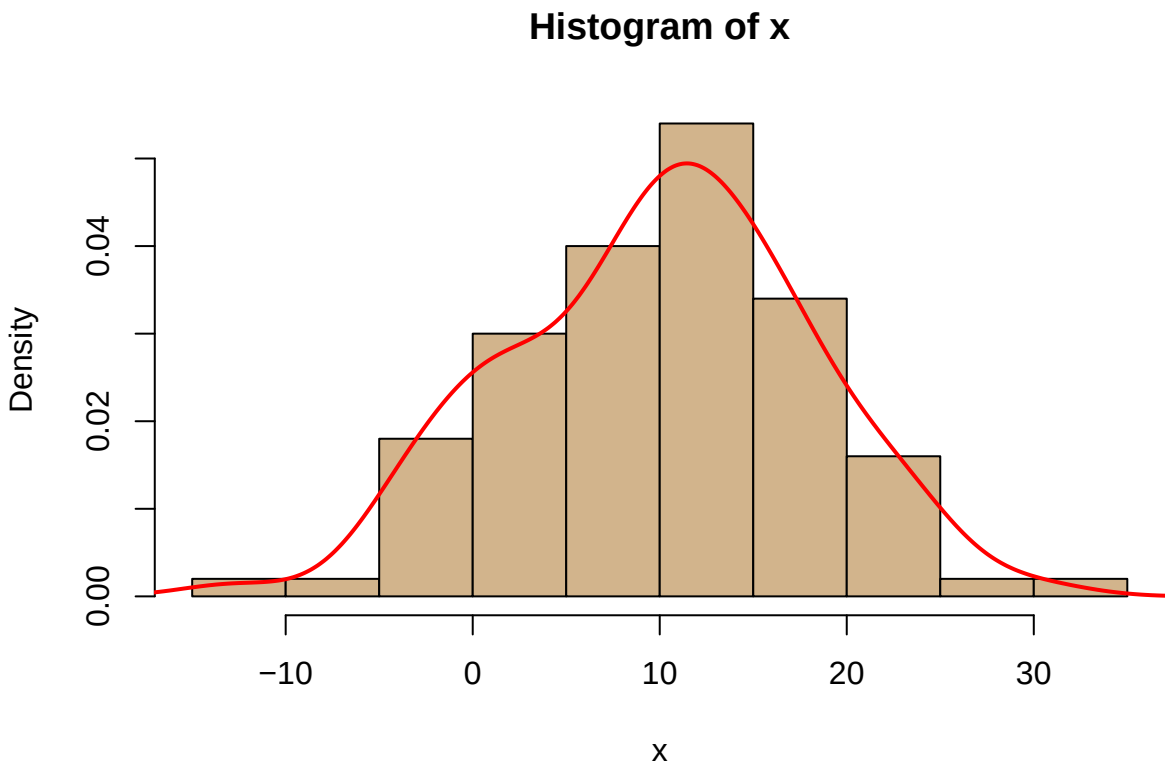
Here we see how the parameters of a function can be minimized using the *optim* function. As a first step, we simulate some data and specify some **known** values for the parameters.

```
set.seed(1123) # for consistency set the seed explicitly.

#first simulate some normal data with expected mean of 0 and sd of 1
x = rnorm(100)
# scale the data the way that we would like
x = x/sd(x) * 8 # sd of 8
x = x-mean(x) + 10 # mean of 10
c('mean'=mean(x),'sd'=sd(x)) # double check
```

```
mean   sd
  10    8
```

```
# histogram (in the fashion of SPSS)
hist(x, freq=FALSE,col='tan')
lines(density(x),col='red',lwd=2)
```



Fun with Likelihood Functions

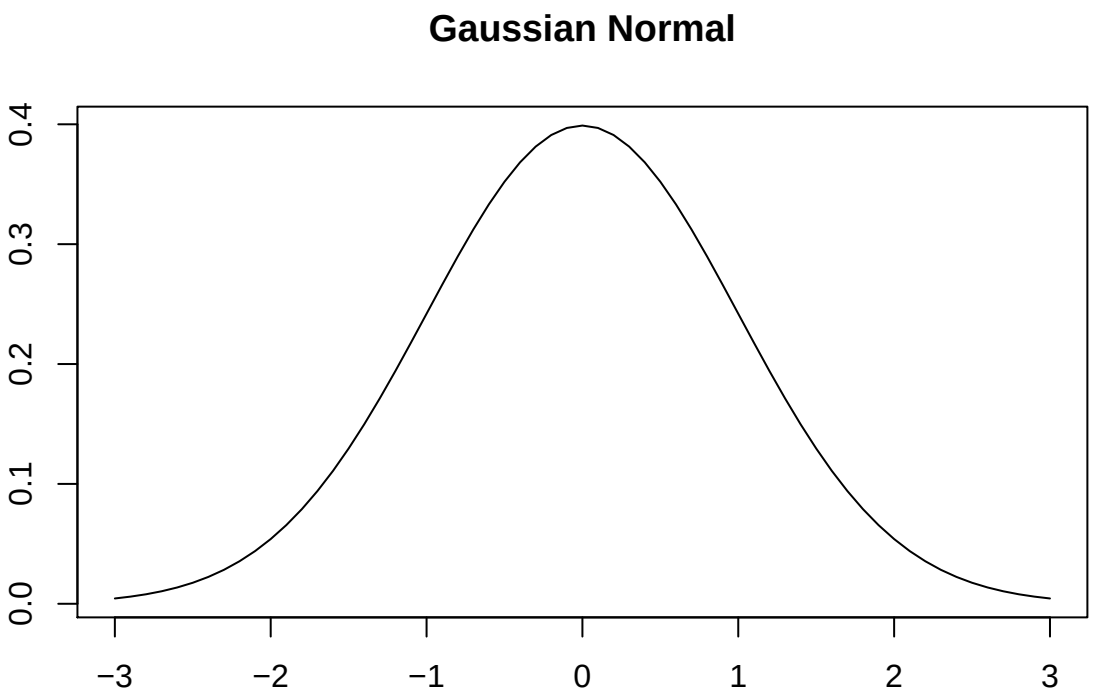
Since these data are drawn from a Normal distribution, $\sim \mathcal{N}(\mu, \sigma^2)$, we will use the Gaussian Normal distribution function for fitting.

$$f(x_i|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(-\frac{(x_i-\mu)^2}{2\sigma^2}\right)}$$

This specifications represents how to compute the likelihood for a single value x_i. That means, we can get the value of the function for any particular input, x_i, if we supply the parameters μ and σ^2.

Here is a plot of what the function produces if we plug in values $x = -3, \ldots, 3$ with use parameters of $\mu = 0$ and $\sigma = 1$.

```
# specify the single value normal probability function
norm_lik = function(x, m, s){
  y = 1/sqrt(2*pi*s^2)*exp((-1/(2*s^2))*(x-m)^2)
}
# and plot it just to make sure
plot(seq(-3,3,.1),sapply(seq(-3,3,.1),FUN=norm_lik,m=0,s=1),type='l',
     ylab='',xlab='', main='Gaussian Normal')
```



Likelihood of a set of values. The function specification changes when we are dealing with an entire set of observations. From basic probability, we know that, if the observations are independent, their *joint* probability is the product of their individual probabilities. So, for our set of observations, we compute the probability value of each point, and then multiply them all together to get the probability of the entire sample.

What does this mean? Well, we literally multiply each obtained value from the function. The result is,

$$\begin{aligned} f(x_1, x_2, \ldots, x_n|\mu, \sigma^2) &= \prod_i^n f(x_i|\mu, \sigma^2) \\ &= f(x_1|\mu, \sigma^2) \times f(x_2|\mu, \sigma^2) \times \ldots \times f(x_n|\mu, \sigma^2) \end{aligned}$$

Using our Gaussian function this translates to,

$$\begin{aligned} &= \prod_i^n \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(-\frac{(x_i-\mu)^2}{2\sigma^2}\right)} \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(-\frac{(x_1-\mu)^2}{2\sigma^2}\right)} \times \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(-\frac{(x_2-\mu)^2}{2\sigma^2}\right)} \times \ldots \times \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(-\frac{(x_n-\mu)^2}{2\sigma^2}\right)}. \end{aligned}$$

This product can be simplified somewhat. To help illustrate we will take advantage of the fact that the product operator, $\prod_i^n$, can be distributed algebraically.

$$\prod_i^n \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(-\frac{(x_i-\mu)^2}{2\sigma^2}\right)} = \prod_i^n \left[\frac{1}{\sqrt{2\pi\sigma^2}}\right] \times \prod_i^n \left[e^{\left(-\frac{(x_i-\mu)^2}{2\sigma^2}\right)}\right]$$

Thus, we can deal with the portions one at a time.

First portion. First, we see that the $\frac{1}{\sqrt{2\pi\sigma^2}}$ term does not involve the observation x_i, which makes it a constant. We also know that taking the product of a constant, is just the constant multiplied by itself the number of times it is included, in our case n times. So, we can express the first portion of the joint probability as,

$$\prod_i^n \frac{1}{\sqrt{2\pi\sigma^2}} = \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n.$$

Alternatively, we can re-express the fraction,

$$\frac{1}{\sqrt{2\pi\sigma^2}} = (2\pi\sigma^2)^{-\frac{1}{2}}.$$

This is helpful, since we remember that raising a power to a power is the same as multiplying the powers together, $(a^b)^c = a^{bc}$. This means that the product of the first term can be simplified as the fraction to the power n. Again, this is the same as multiplying the powers together. The result is,

$$\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n = \left((2\pi\sigma^2)^{-\frac{1}{2}}\right)^n = (2\pi\sigma^2)^{(-\frac{1}{2})\times(n)} = (2\pi\sigma^2)^{(-\frac{n}{2})}.$$

Second portion. Okay, on to the $e^{\left(-\frac{(x_i-\mu)^2}{2\sigma^2}\right)}$ part. First, we can re-express the entire power portion as $(-\frac{1}{2\sigma^2}) \times (x_i - \mu)^2$, so this can be rewritten as $e^{\left(-\frac{1}{2\sigma^2}(x_i-\mu)^2\right)}$.

It is important to recognize that if we have a base number, raised to a power, multiplied by the same base number, raised to a different power, this is equal to the base raised to the sum of the two powers. For example

$$2^2 \times 2^3 = (2 \times 2) \times (2 \times 2 \times 2) = 2^{2+3} = 2^5 = 32.$$

We can take the product of our exponential part and sum over x_i because,

$$\begin{aligned}\prod_i^n e^{\left(-\frac{1}{2\sigma^2}(x_i-\mu)^2\right)} &= e^{\left(-\frac{1}{2\sigma^2}(x_1-\mu)^2\right)} \times e^{\left(-\frac{1}{2\sigma^2}(x_2-\mu)^2\right)} \times \ldots \times e^{\left(-\frac{1}{2\sigma^2}(x_n-\mu)^2\right)} \\ &= e^{\left[-\frac{1}{2\sigma^2}(x_1-\mu)^2+-\frac{1}{2\sigma^2}(x_2-\mu)^2+\ldots+-\frac{1}{2\sigma^2}(x_n-\mu)^2\right]}\end{aligned}$$

Now, factor out the common multiple $-\frac{1}{2\sigma^2}$

$$\begin{aligned}&= e^{\left(-\frac{1}{2\sigma^2}\left[(x_1-\mu)^2+(x_2-\mu)^2+\ldots+(x_n-\mu)^2\right]\right)} \\ &= e^{\left(-\frac{1}{2\sigma^2}\sum_i^n(x_i-\mu)^2\right)}.\end{aligned}$$

Knowing all of this, we can express the *joint likelihood* of all our observations using the *Gaussian* curve,

$$\begin{aligned}f(x_1, x_2, ..., x_n|\mu, \sigma^2) &= \prod_i^n (2\pi\sigma^2)^{-\frac{1}{2}} e^{\left(-\frac{1}{2\sigma^2}(x_i-\mu)^2\right)} \\ &= (2\pi\sigma^2)^{-\frac{n}{2}} e^{\left(-\frac{1}{2\sigma^2}\sum_i^n(x_i-\mu)^2\right)}.\end{aligned}$$

But as you can imagine, if the probabilities are less than 1, then the product of a bunch of these is going to be **SUPER** small. It's not that big of a deal for the math, at least symbolically, but dealing with repeated multiplication of small things is *tedious* and *error prone*, for both humans and computers alike. Practically speaking, a computer has a limit on how small it can represent things and still be accurate.

Fortunately, we may, or may not, remember a special property of *logs*, that the *log* function can turn a product into sum—this will be illustrated below. So, by taking the *log* of the likelihood function we can make the computation much easier while still keeping the same functional relations among the parameter in our original likelihood function.

Quick and dirty logs

Just a refresher, *logs* are meant to show the number of times a number, the *base*, is to be multiplied by itself to get a particular value. Put another way, the answer of the *log* function represents what power of the *base* is needed to get the input value. For example, if the base is 10, and input value is 10, then the answer of the *log* function is 1, because $10^1 = 10$, and so $log_{10}(10) = 1$.

In order to show some of the other properties of *logs* we will work with an easy example. We will use 100, which can be expressed the following **equivalent** ways.

$$\begin{aligned}100 &= 10^2 \\ &= 10 \times 10 \\ &= 1000/10\end{aligned}$$

So, let's work with *log* with a base of 10, this means we are interested in what power to raise 10 to in order to produce the result of 100.

$$\begin{aligned}\text{if} \quad & 10^2 = 100 \\ \text{then} \quad & log_{10}(100) = 2\end{aligned}$$

As we can see, 2 is the answer for base 10. Below we present 3 of the basic properties of *logs*. These are not all of the properties, just the ones that are important for our illustration.

We assume base 10 for the following rules:

power rule

$log(A^n) = n \times log(A)$

- $log(10^2) = 2 \times log(10) = 2 \times 1 = 2$

product rule

$log(A \times B) = log(A) + log(B)$

- $log(10 \times 10) = log(10) + log(10) = 1 + 1 = 2$

quotient rule

$log(\frac{A}{B}) = log(A) - log(B)$

- $log(\frac{1000}{10}) = log(1000) - log(10) = 3 - 1 = 2$

Log likelihood derivation

So, why does this matter? Well, because we are interested in fitting our previous function of the likelihood of a set of data, but we don't want to cause our computer to start to smoke computing very small numbers. If we take the *log* of the likelihood function we get another function that preserves our main properties, but that will also turn our product into a sum.

We will take the *log* of this joint probability version from above. In this case it is easiest to use a base of e for the log of the likelihood, or *natural log*, ln which equals log_e—and remember, this means $ln(e) = 1$. This makes the exponential part much easier to understand. Here are the steps for expressing the new log-likelihood function,

$$
\begin{aligned}
ln(f(x_1, x_2, ..., x_n|\mu, \sigma^2)) &= ln\left[(2\pi\sigma^2)^{-\frac{n}{2}} e^{\left(-\frac{1}{2\sigma^2}\sum_i^n (x_i-\mu)^2\right)}\right] \\
\text{by the } \textbf{product rule} &= ln\left[(2\pi\sigma^2)^{-\frac{n}{2}}\right] + ln\left[e^{\left(-\frac{1}{2\sigma^2}\sum_i^n (x_i-\mu)^2\right)}\right] \\
\text{by the } \textbf{power rule} &= \left[\left(-\tfrac{n}{2}\right) ln(2\pi\sigma^2)\right] + \left[\left(-\tfrac{1}{2\sigma^2}\textstyle\sum_i^n (x_i-\mu)^2\right) ln(e)\right]
\end{aligned}
$$

simplify and we get

$$
\mathcal{L}(X|\mu, \sigma^2) = -\left(\tfrac{n}{2}\right) ln(2\pi\sigma^2) - \tfrac{1}{2\sigma^2}\textstyle\sum_i^n (x_i-\mu)^2
$$

Computer specification

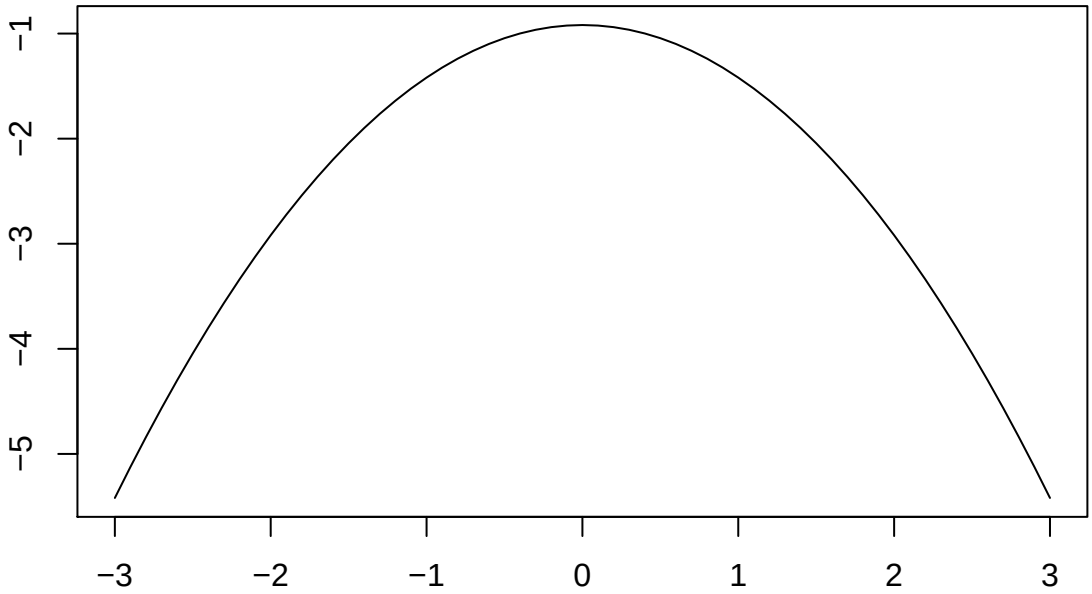
Great, now we need to translate this into something that a computer can understand. Make sure you are careful with your specification, parenthesis can matter... A LOT!

```
llik = function(x,par){
  m=par[1]
  s=par[2]
  n=length(x)
# log of the normal likelihood
# -n/2 * log(2*pi*s^2) + (-1/(2*s^2)) * sum((x-m)^2)
  ll = -(n/2)*(log(2*pi*s^2)) + (-1/(2*s^2)) * sum((x-m)^2)
# return the negative to maximize rather than minimize
  return(-ll)
}
```

There is something important to note about the specification above. Notice that the return value is forced to be negative. Why? Well, most optimization routines are designed to minimize whatever function they are given. This means that it will look for the *lowest* point on our curve, representing the minimum value, rather than the *highest*, which is the maximum that we want. By returning the negative of the function, this reflects the function along the x-axis, but we still end up at the same point.

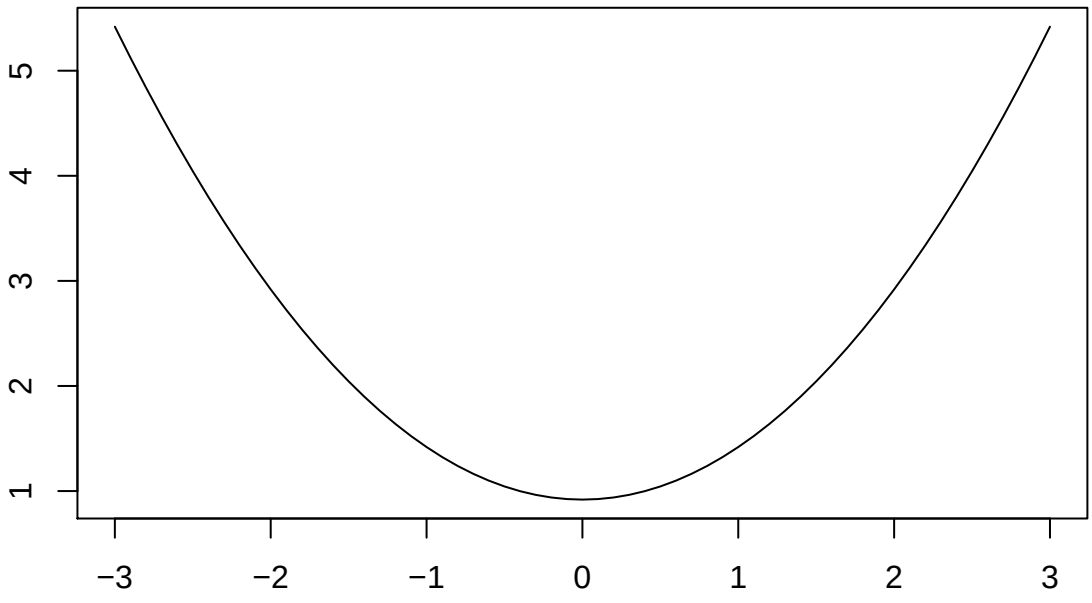
To illustrate the difference, the two versions are plotted below. The first represents what would happen if the return values were positive.

```
# log likelihood curve
plot(seq(-3,3,.1),-1*sapply(seq(-3,3,.1),FUN=llik,par=c(0,1)),type='l',
     ylab='',xlab='')
```



The second plot represents what we pass to the *optim()* function. Notice the lowest point in the second plot is exactly where the highest point was in the first plot.

```
# negative log likelihood curve
# just to see what the funciton produces we can plot it.
plot(seq(-3,3,.1),sapply(seq(-3,3,.1),FUN=llik,par=c(0,1)),type='l',
     ylab='',xlab='')
```



Optimization using *optim()*

Now for the good stuff. Here we use *optim* to minimize our log likelihood function for the two parameters μ, σ. The *optim* function needs some help with where exactly to start its search, in this example we supply a set of starting values, or an *initial guess*. We use a start value of 0.5 for each parameter.

```
# call optim with the starting values 'par',
# the function (here 'llik'),
# and the observations 'x'
res0 = optim(par=c(.5,.5), llik, x=x)
```

Below both results, *direct* and *optim()*, are compared side-by-side.

```
print(kable(
  cbind('direct'=c('mean'=mean(x),'sd'=sd(x)),
        'optim'=res0$par),digits=3))
```

	direct	optim
mean	10	10.002
sd	8	7.976

Bivariate example (Regression)

Within the regression framework, we are most interested in using a linear combination of parameters and variables to explain variance in our outcome of interest. The basic model takes the form of a line.

$$y = ax + b$$

or a more common expression in regression,

$$y_i = b_0 + b_1 x_i + \epsilon_i$$

Where b_0 and b_1 represent the intercept and slope respectively.

Parameter estimation. As you may remember from an earlier statistics course, we can use the *least squares* criteria to find the optimal estimates of both the intercept and slope. However, it may be instructive to see a small example of exactly how such a function is *minimized*.

Hand computation with Calculus

Example:

Say that you are interested in whether or not a mother's level of education relates to her child's high school GPA. Let's make up some data to illustrate.

The (totally made up) data:

- Mother's education: $X = [0, 1, 3, 4]$
- HS GPA: $Y = [3.0, 3.2, 3.3, 3.7]$
  - point $1 = (0, 3.0)$
  - point $2 = (1, 3.2)$
  - point $3 = (3, 3.3)$
  - point $4 = (4, 3.7)$

Here, we use the form of the equation for a line specified as, $y = ax + b$, where a is the slope, and b is the intercept. Our approach is to minimize the sum of squared errors, thus what we need now, is to define the error term. For this we use the expected value, $ax + b$, minus the observed y.

The error equation: $\epsilon = ax + b - y$

To minimize the sum of squared error, we take this function and square it

$$\sum_i \epsilon_i^2 = \sum_i (ax_i + b - y_i)^2$$

Using our data, this sum of squared errors can now be expressed as:

$$\begin{array}{lll} SS_e & = [(0a + b - 3.0)^2 & \text{values from point 1} \\ & +(1a + b - 3.2)^2 & \text{values from point 2} \\ & +(3a + b - 3.3)^2 & \text{values from point 3} \\ & +(4a + b - 3.7)^2] & \text{values from point 4} \end{array}$$

Simplify and expand

$$\begin{array}{ll} SS_e & = [(b - 3.0)(b - 3.0)+ \\ & (a + b - 3.2)(a + b - 3.2)+ \\ & (3a + b - 3.3)(3a + b - 3.3)+ \\ & (4a + b - 3.7)(4a + b - 3.7)] \end{array}$$

Multiply through

$$\begin{array}{ll} SS_e & = [(b^2 - 6b + 9)+ \\ & (a^2 + ab - 3.2a + ab + b^2 - 3.2b - 3.2a - 3.2b + 10.24)+ \\ & (9a^2 + 3ab - 9.9a + 3ab + b^2 - 3.3b - 9.9a - 3.3b + 10.89)+ \\ & (16a^2 + 4ab - 14.8a + 4ab + b^2 - 3.7b - 14.8a - 3.7b + 13.69)] \end{array}$$

Collect similar terms within each sub-expression

$$\begin{aligned} SS_e &= [(b^2 - 6b + 9) + \\ &\quad (a^2 + 2ab - 6.4a + b^2 - 6.4b + 10.24) + \\ &\quad (9a^2 + 6ab - 19.8a + b^2 - 6.6b + 10.89) + \\ &\quad (16a^2 + 8ab - 29.6a + b^2 - 7.4b + 13.69)] \end{aligned}$$

Combine all sub-expressions and collect common terms

$$\begin{aligned} SS_e &= a^2 + 9a^2 + 16a^2 \\ &\quad + b^2 + b^2 + b^2 + b^2 \\ &\quad - 6.4a - 19.8a - 29.6a \\ &\quad - 6b - 6.4b - 6.6b - 7.4b \\ &\quad + 2ab + 6ab + 8ab \\ &\quad + 9 + 10.24 + 10.89 + 13.69 \end{aligned}$$

Simplify common terms

$$\begin{aligned} SS_e &= 26a^2 \\ &\quad + 4b^2 \\ &\quad - 55.8a \\ &\quad - 26.4b \\ &\quad + 16ab \\ &\quad + 43.82 \end{aligned}$$

The result is the equation for the sum of squared errors, using our four observed points,

$$SS_e = 26a^2 + 4b^2 - 55.8a - 26.4b + 16ab + 43.82.$$

For our next step, we take the partial derivative of this equation with respect to each parameter. This will tell us how the function changes conditional on each parameter.

To begin, we take the partial derivative of the function SS_e with respect to a. We do the same for the parameter b later. It is important to note, when we differentiate the equation based on a, we only need to consider those terms that have a in them. We will be using the power rule, which states $\frac{d}{dx} = (x^n) = n \cdot x^{n-1}$.

$$\begin{aligned} SS_e \text{ wrt } a &= 26a^2 & -55.8a & & +16ab \\ \tfrac{\partial SS_e}{\partial a} &= 26\left(2 \cdot a^1\right) & -55.8\left(1 \cdot a^0\right) & & +16b\left(1 \cdot a^0\right) \\ \tfrac{\partial SS_e}{\partial a} &= 26(2 \cdot a) & -55.8(1 \cdot 1) & & +16b(1 \cdot 1) \end{aligned}$$

$$\tfrac{\partial SS_e}{\partial a} = 52a - 55.8 + 16b$$

This represents how the function changes with respect to a. In order to find the point where the function stops changing, we rearrange this equation to look like our equation for a line, then set this equal to zero. This gives us the minimum point for the equation, or where the change stops.

$$\begin{aligned} \tfrac{\partial SSe}{\partial a} &= 52a + 16b - 55.8 \\ 0 &= 52a + 16b - 55.8 \end{aligned}$$

Next we repeat for the same process for the parameter b.

$$\begin{aligned} SS_e \text{ wrt } b &= 4b^2 - 26.4b + 16ab \\ \\ \frac{\partial SSe}{\partial b} &= 8b - 26.4 + 16a \\ \frac{\partial SSe}{\partial b} &= 16a + 8b - 26.4 \\ 0 &= 16a + 8b - 26.4 \end{aligned}$$

Equation 1 (how the function changes with respect to a)

$$0 = 52a + 16b - 55.8.$$

Equation 2 (how the function changes with respect to b)

$$0 = 16a + 8b - 26.4.$$

Solve for a in Equation 1

$$\frac{(55.8 - 16b)}{52} = a.$$

Plug a into Equation 2 and solve for b

$$\begin{aligned} 0 &= 16 \cdot \left(\tfrac{(55.8-16b)}{52}\right) + 8b - 26.4 \\ 0 &= 16 \cdot \tfrac{55.8}{52} - 16 \cdot \tfrac{16b}{52} + 8b - 26.4 \end{aligned}$$

move all of the b terms to one side of the equation

$$\begin{aligned} 26.4 - 16 \cdot \tfrac{55.8}{52} &= -16 \cdot \tfrac{16b}{52} + \tfrac{416b}{52} \\ 9.23077 &= \tfrac{160b}{52} \\ 9.23077 \cdot 52 &= 160b \\ 480 &= 160b \\ \tfrac{480}{160} &= b \end{aligned}$$

the intercept estimate

$$3 = b.$$

Plug b into our equation for a from above

$$\begin{aligned} \tfrac{(55.8-16\cdot 3)}{52} &= a \\ \tfrac{(55.8-48)}{52} &= a \\ \tfrac{7.8}{52} &= a \\ 0.15 &= a. \end{aligned}$$

Based on our analytic solution $a = 0.15$ and $b = 3$.

So, the best fitting line is

$$y = 0.15x + 3$$

Written in the traditional regression form, we have

$$\hat{y} = 3 + 0.15x.$$

Let's confirm our findings using the *lm()* function from R.

```r
# our totally made up data
MomEd = c(0, 1, 3, 4)
HSGPA = c(3.0, 3.2, 3.3, 3.7)
# fit a linear model
coef(lm(HSGPA ~ MomEd)->lm0)
```

```
(Intercept)       MomEd
       3.00        0.15
```

Using numerical optimization

Another method to get the parameters estimates involves defining the loss function and minimizing it using an optimization routine. This is exactly what was done above in the univariate example. Our loss function of choice is the *minimum sum of squares* which compares our predictions to the observed. Let's specify the loss function for this model.

```r
# sum of squares function
SS_min = function(data,par){
  b0=par[1]
  b1=par[2]
  loss = with(data, sum((b0+b1*x - y)^2))
  return(loss)
}
# data on mom's ed and HS GPA from above
dat=data.frame(x=MomEd,y=HSGPA)
# min resid sum of squares
res1a = optim(par=c(.5,.5),SS_min, data=dat)
# direct comparison
print(kable(cbind('lm()'=coef(lm0),'SS_min'=res1a$par),digits=4))
```

	lm()	SS_min
(Intercept)	3.00	3.0002
MomEd	0.15	0.1500

With this loss function we can minimize any data that we believe follow a bivariate linear model. Below is another example with simulated data.

First, we will simulate some bivariate data.

```r
# slope effect
b1 = .85
# simulated data
x = 1:60
dat = data.frame(x=x,y=(x*b1)+rnorm(60))
# from the lm() function
lm1 = lm(y~x, data=dat)
lm1
```

```
Call:
```

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

Coefficients:
(Intercept)            x
     0.1099       0.8475
```

Now that we have an idea of what we are looking for, we can see how *optim()* compares.

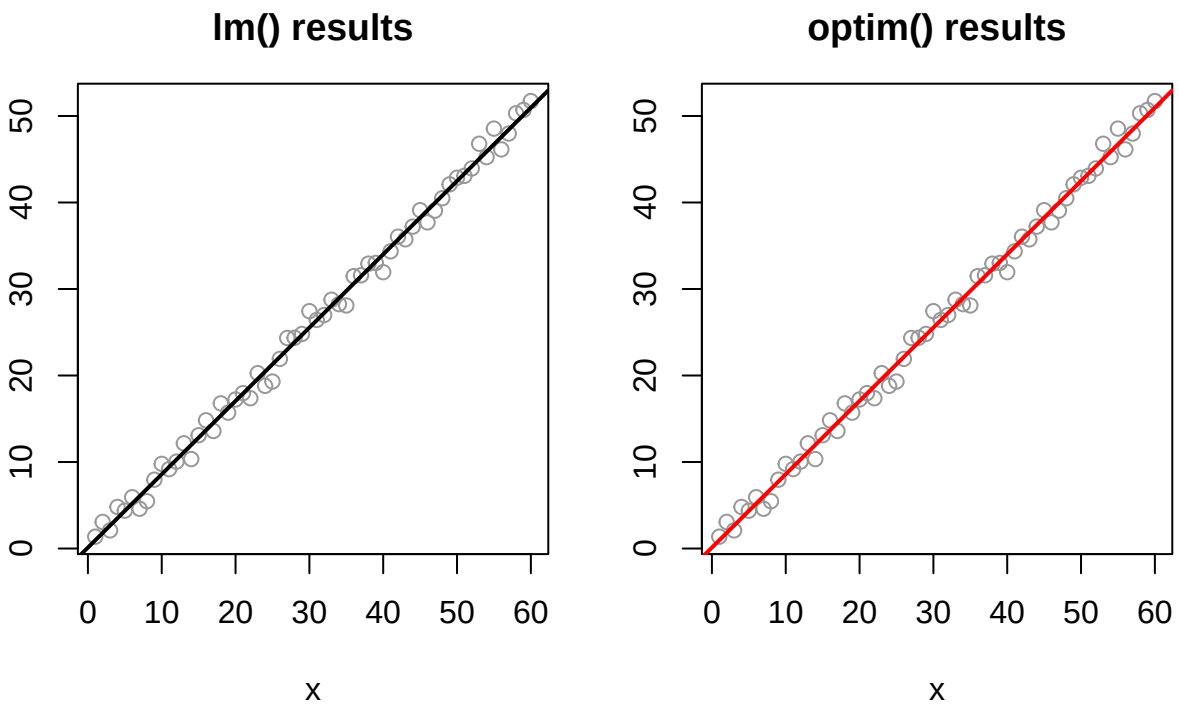
```
# different start values
res1 = optim(par=c(.01,1),SS_min, data=dat)
res1$par
```

```
[1] 0.1103062 0.8474895
```

How did we do compared to the *lm()* function? To visualize we can plot the data and superimpose the regression line over the top.

```
op=par(mfrow=c(1,2),mar=c(3,3,3,1),pty='s')
# scatterplot
with(dat,plot(x,y,col='grey60',main='lm() results'))
# regression line from lm()
abline(lm1,lwd=2)

with(dat,plot(x,y,col='grey60',main='optim() results'))
# from the Min SS
abline(res1$par[1],res1$par[2],col='red',lty=1,lwd=2)
```



```r
par(op)
```

Not bad at all.

Multivariable approach — Multiple Linear Regression

Below we see how the same approach can be used with multiple variables. Here however we are not dealing with a multivariate model, rather a multi-variable model. This is a simple multiple linear regression.

To begin we simulate some data with known values.

```
#============================================================================#
# adapted from:
#     http://www.mail-archive.com/r-sig-teaching@r-project.org/msg00066.html
#============================================================================#
sim_regression_data <- function(betas, means=0, stdvs=1, n=1000, rsqr=.8){
  #number of predictors
    nvar <- length(betas);
    #are means specified
    if(length(means) != nvar){
        means <- rep(0,nvar);
    }
    #are standard devs specified
    if(length(stdvs) != nvar){
        stdvs <- rep(1,nvar);
    }
    #random data setup based on supplied means and standard devs
    xs <- matrix(NA,nrow=n,ncol=nvar);
    for(i in 1:nvar){
        xs[,i] <- rnorm(n,mean=means[i],sd=stdvs[i]);
    }
    #random data scaled by beta weights
    yhat <- xs %*% betas
    #sum of square residuals
    ssr <- sum((yhat - mean(yhat))^2);
    #name things
    xscols <- c(paste('x',1:nvar,sep=''));
    colnames(xs) <- xscols;
    xs <- as.data.frame(xs);
    #error time
    err <- rnorm(n);
    lm_err <- lm(as.formula(paste("err ~ ",paste("xs",xscols,sep="$",collapse=" + "))));
    err <- resid(lm_err);
    #a little magic to get the r-square we want
    numr <- 1 - rsqr;
    err <- err * sqrt(numr/rsqr * ssr/(sum(err^2)));
    #add this to yhat to get observed y
    y <- yhat + err;
    #package it all up and ship it!
    data.df <- as.data.frame(cbind(y,xs));
    names(data.df) <- c('y',xscols);
    return(data.df);
}
# Example with known values
betas <- c(.6,  .8, .9, .65);
means <- c(100, 35, 10, 150);
stdvs <- c(15,  12,  5,  22);
simdat <- sim_regression_data(betas, means, stdvs, n=120, rsqr=.75);
```

```r
lm0 = lm(y ~ x1 + x2 + x3 + x4, data=simdat)
summary(lm0);
```

```
Call:
lm(formula = y ~ x1 + x2 + x3 + x4, data = simdat)

Residuals:
    Min      1Q  Median      3Q     Max
-34.239  -7.035   1.595   7.462  27.709

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.113e-14  1.049e+01   0.000        1
x1           6.000e-01  6.980e-02   8.596 4.75e-14 ***
x2           8.000e-01  8.902e-02   8.986 5.95e-15 ***
x3           9.000e-01  2.067e-01   4.354 2.92e-05 ***
x4           6.500e-01  5.400e-02  12.036  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 11.45 on 115 degrees of freedom
Multiple R-squared:   0.75, Adjusted R-squared:  0.7413
F-statistic: 86.25 on 4 and 115 DF,  p-value: < 2.2e-16
```

```r
# sum of squares function
SS_min = function(data,par){
  b0=par[1]
  b1=par[2]
  b2=par[3]
  b3=par[4]
  b4=par[5]
  ll = with(data, sum((b0 + b1*x1 + b2*x2 + b3*x3 + b4*x4 - y)^2))
  return(ll)
}
# min resid sum of squares
res1 = optim(par=rep(.5,ncol(simdat)),SS_min, data=simdat)
```

```r
print(kable(cbind('lm results'=coef(lm0), 'optim results'=res1$par),digits=3))
```

	lm results	optim results
(Intercept)	0.00	-0.237
x1	0.60	0.601
x2	0.80	0.800
x3	0.90	0.901
x4	0.65	0.651