

Validating Model Fitting Code in R

General Strategy

We validate our model fitting code, so we can be sure it works. In addition, it is good to have some sense of any bias that might exist in estimating parameters or comparing models. By simulating data by running a model “forwards,” we can confirm that our code can recover true parameter values (with some error, of course) and identify true models.

The basic approach is to write a short function that both (1) simulates new data for a model and (2) fits the model to the newly simulated data. The parameter estimates (and/or log-likelihoods) can be stored for each batch of simulated data, and the population of estimates can then be plotted, to see if the true value is in the center of each distribution.

Gaussian Example

Suppose we have the model $y_i \sim \mathcal{N}(\alpha + \beta_1 x_i + \beta_2 z_i + \beta_3 x_i z_i, \sigma)$ —a linear regression model with two prediction variables and an interaction term—and we want to make sure our fitting code is working correctly. First, generate some x and z values to stand in for prediction variables:

```
x <- rnorm( 100 , 50 , 10 ) # 100 random normal values with mean 50 and sd 10
z <- rnorm( 100 , 20 , 2 ) # 100 random normal values with mean 20 and sd 2
```

Now assign some parameter values to try to recover with our code:

```
a <- 10
b1 <- 2
b2 <- -7
b3 <- 0.75
s <- 3
```

Now we write a function that simulates data from the model and fits the model to the simulated data. If the fitting code is working, we should get parameter estimates that are reasonably close to the true values we used to simulate the data (larger samples make this process more accurate).

```
modelrep <- function(a,b1,b2,b3,s,x,z) {
  # first simulate data from model
  y <- rnorm( length(x) , mean=a+b1*x+b2*z+b3*x*z , sd=s )
  # now fit model to simulated data in y
  m <- mle2( y ~ dnorm( mean=ka+kb1*x+kb2*z+kb3*x*z , sd=ks ) ,
            start=list(ka=mean(y),kb1=0,kb2=0,kb3=0,ks=sd(y)) , data=list(y=y,x=x,z=z) )
  # finally return coefficients from fitted model
  coef(m)
}
```

We are ready to go now. To repeat the above any arbitrary number of times, you can use the `replicate` function, which does exactly what it sounds like it does.

```
yreps <- replicate( 99 , modelrep(a,b1,b2,b3,s,x,z) )
```

The above repeats the simulation and fitting 99 times, using the parameter values we chose at the start. Note that we are explicitly passing the data in `x` and `z` to both the `modelrep` function and `mle2` (using the `data=list(y=y,x=x,z=z)`). We do this so R can find the data values reliably. If you know what “scoping” is, then this is an example of its impact. If not, don’t worry about it, but do use the procedures that I have used above.

After running the line of code with `replicate` in it, you get a big matrix of parameter estimates in `yreps`. Depending upon how fast your computer is, it might take a while for R to finish all 99 replicates. But once it does finish, each row of the matrix `yreps` is a parameter and each column is a replicate simulation. So if you type:

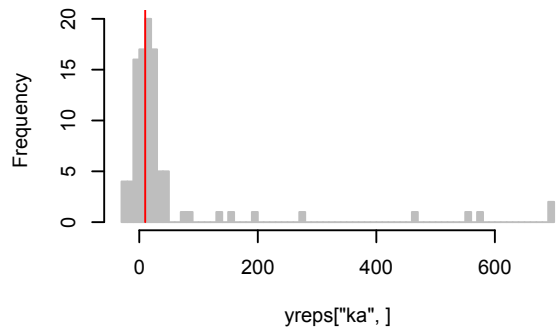
```
yreps[,50]
```

you will see the parameter estimates for the 50th simulation. We are going to plot the histograms of each parameter, to make sure each true parameter value is located centrally in each distribution of estimates.

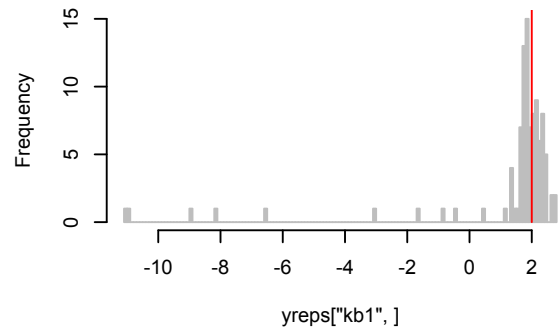
```
par(mfrow=c(3,2))
hist( yreps["ka",] , breaks=100 , col="gray" , border="gray" )
lines( c(a,a) , c(0,100) , col="red" )
hist( yreps["kb1",] , breaks=100 , col="gray" , border="gray" )
lines( c(b1,b1) , c(0,100) , col="red" )
hist( yreps["kb2",] , breaks=100 , col="gray" , border="gray" )
lines( c(b2,b2) , c(0,100) , col="red" )
hist( yreps["kb3",] , breaks=100 , col="gray" , border="gray" )
lines( c(b3,b3) , c(0,100) , col="red" )
hist( yreps["ks",] , breaks=100 , col="gray" , border="gray" )
lines( c(s,s) , c(0,100) , col="red" )
```

The figure that follows is what I got, on one particular run of 99 replicates. Each histogram below is for a single parameter. The red line in each is the location of the “true” value of the parameter, the value we chose at the start. Note that the red line is centrally located in each case, aside from the rare long tails in each plot. Those result from rare kinds of simulated data that lead to `mle2` screwing up the model fitting process. Remember: Maximum likelihood search isn’t guaranteed to find sensible estimates, and any particular simulated set of data might be highly unlikely, given the parameters, produced strange estimates. This is why you should replicate the analysis many times, preferably more than 99 even.

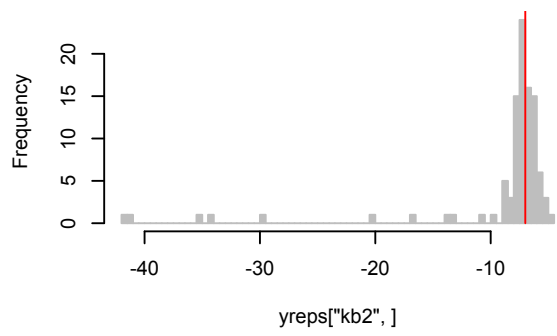
Histogram of yreps["ka",]



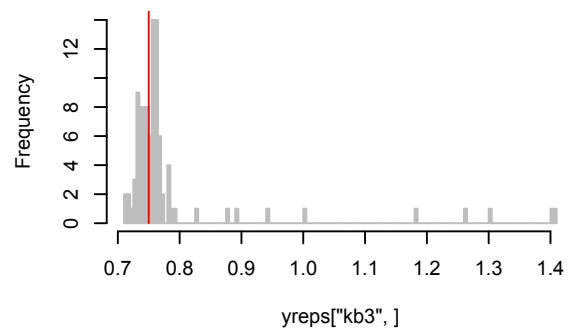
Histogram of yreps["kb1",]



Histogram of yreps["kb2",]



Histogram of yreps["kb3",]



Histogram of yreps["ks",]

