Details of AD Model Builder's Covariance Calculations

Cole Monnahan — monnahc@uw.edu September 19, 2012

Abstract

This short document details (1) the functions that AD Model Builder (ADMB) uses to bound parameters, (2) the method for calculating a bounded covariance matrix, and (3) what is stored in the binary admodel.hes file and how the user can utilize this information for defining their own correlation matrix for the MCMC jump function. This information may be useful to ADMB users who wish to understand this process better, or those who wish for more control in order to experiment with making the MCMC more efficient by manually changing the correlation matrix.

1 Introduction

ADMB is a powerful tool for calculating maximum likelihood estimates (MLE) for complex, large, non-linear models. Besides providing asymptotic variance estimates for parameters, it also has a built-in method for drawing from a posterior distribution using a Markov chain Monte Carlo (MCMC). By default the MCMC proposal function is a multivariate normal distribution with a covariance matrix determined by the curvature at the MLE.

However, for models with many parameters, obtaining MCMC convergence can be very difficult and take weeks or months, making the Bayesian approach infeasible. One of the most common issues that arises is when parameters are highly correlated in the model and that correlation varies throughout the parameter space, which causes the jump function to not explore the possible space very efficiently. To get around this problem it may be useful for users to be able to replace the model-estimated covariance matrix with a modified covariance matrix with lower correlations between certain parameters. In general, having more control over the covariance matrix used would allow the user to explore more possibilities to improve convergence time. In order for the user to execute this it must be possible to extract the covariance matrix calculated in ADMB and understand how it is calculated from the Hessian. These calculations have not been documented and are poorly understood, making this task difficult. This document details the calculations used by ADMB and provides a method by which the user can override the default ADMB behavior and specify their own covariance matrix.

2 Bounded Parameters

ADMB has the built in capability to bound parameters within a certain range when building a model. These parameters are declared as type <code>init_bounded_number</code> in the PARAME-TER section, followed by a minimum and maximum value (and lastly a phase). Typically these are used for naturally bounded parameters such as probabilities, but the ADMB manual also recommends bounding parameters to "ensure that the minimization is stable" (1-22). Starting values for optimization are by default the average of the min and max values provided by the user as starting values although these can be set manually in the PRELIMINARY_CALCS_SECTION or by using a .pin file (see the manual for further information).

Understanding how ADMB accomplishes the bounding procedure is vital to understanding the Hessian and covariance matrices created by the program, since these matrices are calculated in the unbounded space. That is, if one does not know how parameters are bounded, then they cannot interpret or use the covariance matrix. I'll briefly detail the transformations used internally in ADMB, as well as how the Delta method is applied to convert from the unbounded to bounded covariance matrix.

2.1 Bounded Transformations

When a parameter is specified to be of type init_bounded_number ADMB will use one of two transformations internally that map the real line to the bounds specified by the user. Which transformation is used for all parameters in the model depends on a variable called the Hybrid_bounded_flag, which for this document I'll shorten to hbf for convenience. Regardless of the value of hbf, ADMB will correctly optimize in the bounded space and carry out variance estimates using the delta method. Thus for each of the two transformations the inverse and derivative functions are also used. Here I specify the two transformations as f and g but in reality in the source code they are determined using an if statement within the function declaration (see R code below). I find it easier to think about, and present the two functions separately.

Let a and b be the minimum and maximum bounds, respectively, then the transformations are:

$$g \colon [-1,1] \mapsto [a,b] \tag{1}$$

$$g(x,a,b) = a + (b-a)\left(\frac{1}{2}\sin\left(\frac{\pi x}{2}\right) + \frac{1}{2}\right)$$
 (2)

$$g^{-1} = \left(\frac{2a}{\pi}\right) \cdot \sin\left(\frac{2(x-a)}{(b-a)} - 1\right) \tag{3}$$

$$g' = (b - a)\left(\frac{\pi}{4}\right)\cos\left(\frac{x\pi}{2}\right) \tag{4}$$

and

$$f: [-\infty, \infty] \mapsto [a, b] \tag{5}$$

$$f(x, a, b) = a + \frac{b - a}{1 + e^{-x}} \tag{6}$$

$$f^{-1} = -\log\left(\frac{b-x}{x-a}\right) \tag{7}$$

$$f' = \frac{(b-a)e^{-x}}{(1+e^{-x})^2} \tag{8}$$

As mentioned above which of these functions used by ADMB depends on the value of hbf, as shown below.

Bounding Functions =
$$\begin{cases} g, g^{-1}, g' & \text{if } hbf = 0 \\ f, f^{-1}, f' & \text{if } hbf = 1 \end{cases}$$
 (9)

Note that hbf is controllable by the user by passing an option argument to the executable: -hbf. However ADMB calculates the bounded standard errors correctly regardless of the value of hbf, so it is more important just to be aware of what its value is, rather than worrying about setting it manually (unless for other reasons).

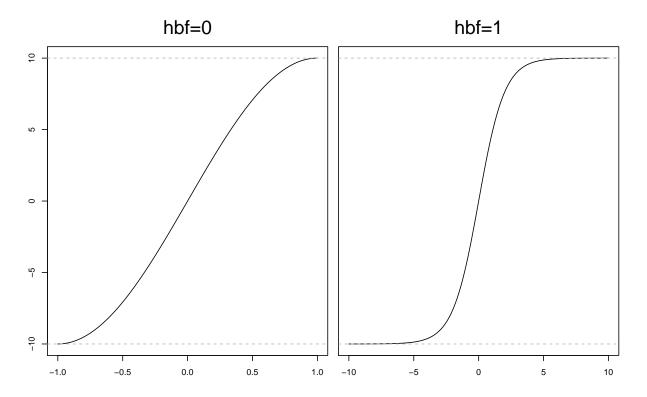


Figure 1: Plots of the two transformations where a = -10 and b = 10. Note that the domain (-1,1) in the first plot covers the entire bounded range, while in the second plot this domain is $(-\infty,\infty)$.

2.2 R Functions

The above functions are written below in R, and match the naming convention used in the ADMB source code, such that boundp is the transformation, boundpin is the inverse, and ndfboundp is the derivative. Note that these are simplified versions of the actual transformations in that they leave out any of the penalty calculations as well as checks to ensure numerical stability (for example see the boundp source code).

```
boundp <- function(x, minb, maxb, hbf){</pre>
    ## The internal transformations used in ADMB depending on the value of the
    ## Hybrid_bounded_flag (hbf) value.
    if(hbf==1)
        result <- minb+(maxb-minb)/(1+exp(-x))
    else if(hbf==0)
        result <-minb+(maxb-minb)*(.5*sin(x*pi/2)+.5)
    else stop("Invalid hbf value, should be 0 or 1")
    return(result)
}
boundpin <- function(x, minb, maxb, hbf) {</pre>
    ## The inverse of the transformation
    if(hbf==1)
        result <- -log( (maxb-x)/(x-minb) )
    else if(hbf==0)
        result < asin(2*(x-minb)/(maxb-minb)-1)/(pi/2)
    else stop("Invalid hbf value, should be 0 or 1")
    return(result)
}
ndfboundp <- function(x, minb, maxb, hbf) {</pre>
    ## The derivative used to find the "scales"
    if(hbf==1)
        result <- (maxb-minb)*exp(-x)/(1+exp(-x))^2
    else if(hbf==0)
        result <- (maxb-minb)*.5*pi/2*cos(x*pi/2)
    else stop("Invalid hbf value, should be 0 or 1")
    return(result)
}
```

2.3 Delta Method

For all problems, the Hessian is the matrix of second mixed derivatives evaluated at the MLE in unbounded space. Thus the covariance matrix needs to be corrected so that it reflects the covariance of the bounded space, which is where SEs should be calculated (the same as

the MLE). Let x be a parameter in unbounded space, and y = f(x) the same parameter in the bounded space and that f is the transformation used. Then the Delta method is used to estimate the variance of y and covariance between two parameters. Note that if there is no transformation (i.e. the identity transformation) then the scale term is simply 1 for that parameter

$$Var\left[\hat{y}\right] = Var\left[f(\hat{x})\right] \tag{10}$$

$$\approx \operatorname{Var}\left[\hat{x}\right] \cdot f'(\hat{x})^2 \tag{11}$$

$$Cov [\hat{y}_1, \, \hat{y}_2] = Cov [f(\hat{x}_1), \, f(\hat{x}_2)]$$

$$\approx Cov [\hat{x}_1, \, \hat{x}_2] \cdot f'(\hat{x}_1) \cdot f'(\hat{x}_2)$$
(12)

The variances in unbounded space ($\operatorname{Var}[\hat{x}]$) are readily available from the covariance matrix. The terms $f'(\hat{x})$ are thus critical to estimating parameter variances and are easily calculated from above. In the ADMB source code these are usually referred to as scales. Presumably because these terms are so important to the internal functioning of ADMB, they are stored directly with the Hessian information written to file in the admodel.hes file. This is a key observation because it allows the user to extract these terms from file and convert the covariance matrix in unbounded space (which is what is written to file) to one in bounded space. Naturally ADMB does this calculation automatically, but knowledge of the calculations also allows for calculating in the opposite direction – that is from bounded to unbounded space. An example of this would be specifying a correlation matrix in bounded space for the MCMC jump function to use, and then backward calculating what the covariance matrix would be in unbounded space. If done correctly, when ADMB calculates the correlation matrix internally it will recreate the one supplied by the user.

3 The admodel.hes file

ADMB creates two temporary binary files to store information about the curvature in unbounded space: admodel.hes and admodel.cov. Note that these files maintain this name regardless of the model name, unlike many other important files generated upon execution. (Also note that they are considered temporary so the "clean directory" command in the ADMB-IDE (i.e. emacs) will delete them). Both files actually contain most of the same information, but I will focus here on the .hes file since it is created even if the Hessian is not positive definite and a valid covariance matrix cannot be calculated; however the Hessian can always be inverted to determine the covariance (although if it is not positive definite it won't be a valid covariance matrix – ADMB will throw an error regarding this when optimizing).

3.1 File Contents

ADMB writes the .hes file with information sequentially, so it needs to be recovered in the same way. The following table describes the contents and storage type associated with the .hes file.

Content	Description	Type	Size
n	Number of parameters (not including sd_variables)	Integer	1
H	The Hessian matrix, as vector of elements	Numeric	n^2
hbf	The Hybrid_bounded_flag, dictating which transformation was used.	Integer	1
scale	The "scale" used in the Delta method, one for each variable.	numeric	n

Table 1: The contents of the admodel.hes file.

Thus a typical file contains elements $\{n, H, hbf, scale\}$, not coincidentally what is needed to calculate the bounded variance. Using the knowledge laid out above, the following R function will read these pieces of information into a list.

```
getADMBHessian <- function(){</pre>
    ## This function reads in all of the information contained in the
    ## admodel.hes file. Some of this is needed for relaxing the covariance
    ## matrix, and others just need to be recorded and rewritten to file so ADMB
    ## "sees" what it's expecting.
    filename <- file("admodel.hes", "rb")</pre>
    on.exit(close(filename))
    num.pars <- readBin(filename, "integer", 1)</pre>
    hes.vec <- readBin(filename, "numeric", num.pars^2)</pre>
    hes <- matrix(hes.vec, ncol=num.pars, nrow=num.pars)
    hybrid_bounded_flag <- readBin(filename, "integer", 1)
    scale <- readBin(filename, "numeric", num.pars)</pre>
    result <- list(num.pars=num.pars, hes=hes,
                    hybrid_bounded_flag=hybrid_bounded_flag, scale=scale)
    return(result)
}
```

The admodel.cov file is identical in structure to the admodel.hes file except that instead of storing the Hessian it stores the covariance matrix. The getADMBHessian function above is easily converted to work with the admodel.cov file. Note that the admodel.hes file will be written even if it is not positive definite. This can be an important distinction if the user has non-positive definite Hessian but would still like to manually define a covariance matrix for use in running an MCMC.

4 Example

These concepts can be demonstrated with a very simple example. First, consider the "Simple" example that comes with ADMB and is just a linear regression with a being the slope and b being the intercept. After running ADMB on this model, the function above extracts the following information:

```
> getADMBHessian()
$num.pars
[1] 2
```

\$hes

[1] 1 1

```
[,1] [,2]
[1,] 102.79718 17.550738
[2,] 17.55074 5.014496

$hybrid_bounded_flag
[1] 0

$scale
```

Using this Hessian, it is possible to calculate the covariance matrix and then the standard errors and correlation matrix as follows.

And, as expected this matches the ADMB calculations in the .cor file.

```
The logarithm of the determinant of the hessian = 5.33488 index name value std dev 1 2 1 a 1.9091e+000 1.5547e-001 1.0000 2 b 4.0782e+000 7.0394e-001 -0.7730 1.0000
```

Now, if the a and b parameters are bounded within ADMB, such that they are declared as

```
init_bounded_number a(-10,10)
init_bounded_number b(-10,10)
```

and repeat the above calculations we get the same values in the .cor file (which we should, since bounding the problem does not change the MLE values or correlations). However, if we try to calculate the covariance matrix from the admodel.hes file there is a discrepancy:

Notice that the standard errors are incorrect, but that the correlation matrix is correct. This is because the scale terms cancel out in calculating the correlation matrix. However, it should be clear that the covariance matrix (and thus the standard errors) are not matching because this matrix is in **unbounded space**, whereas we want it to be in bounded space. As detailed above, this can be accomplished by utilizing the scale values from the admodel.hes file.

Now the standard errors match again, thus we have calculated the bounded covariance matrix.

5 User-specified Correlation Matrix

From the calculations above in the R code, it should be apparent how a user can manually specify a correlation matrix and force ADMB to use it when running an MCMC [R code to accomplish this is still being developed, but a preliminary copy can be attained from the author via email]. Given a specific matrix, just calculate the unbounded covariance matrix using the standard errors and scales from the binary files. When finished, the new matrix can be written back to a binary file and ADMB will read this in and use that new matrix in the MCMC jump function. An important note is that ADMB uses the admodel.cov and not the admodel.hes file when initiating the MCMC, so be sure to write the correct file and make sure to include the other values (i.e. n, H, hbf, scales) as detailed in the table above. Following are steps outlining how a user would go about specifying their own matrix:

- 1. Run optimization
- 2. Specify correlation matrix and rewrite the admodel.cov file.
- 3. Run an MCMC with command line option -mcmc N -mcsave N -noest -nohess, and possibly -pin to direct it to a file with starting parameters.
- 4. Evaluate and generate the chain with -mceval

ADMB will still "rescale" the stepsize to achieve an optimal convergence, unless the option—mcnoscale is passed.

The user should be aware that there are built-in options aimed at increasing the efficiency of the MCMC chain, such as -mcgrope and -mcrb although they are undocumented and don't leave the user with the same level of control. However, it would be prudent to explore these options first as they may be sufficient for the user's needs.