Assignment 5: Sensitivity Analysis

An important part of being a good modeler is knowing how to perform sensitivity analysis (SA). Often the results of SA will simply confirm your intuition, but surprisingly often it will give you new insight into your model, even for fairly simple models. SA equates to understanding how model output responds to changes in model input. Common goals are (see Saltelli paper for more discussion):

1) Factor Prioritization (FP) - Identify which factors would provide the greatest reduction in output uncertainty if fixed. For example, will your modeling goals be better served by spending time and money to reduce uncertainty in factor A or B?
2) Factor Fixing (FF) - Identify which factors are non-influential, and therefore can be fixed without much change in the model. This is often the goal of 'screening' exercises, i.e. to screen out those factors that are not important
3) Variance Cutting (VC) - Identify a minimum subset of factors that one should fix to achieve a prescribed reduction in output uncertainty.
4) Factor Mapping (FM) - Identify which factors are mostly responsible for producing realizations of the output, Y, within different output classes (e.g., acceptable vs. non-acceptable levels of a contaminant).

There are several SA techniques, each with their own strengths and limitations. In general, simple techniques are suitable for most models, but more sophisticated techniques will be needed for models that are nonlinear, have several important factors, and/or have interactions between factors. As you know, this is often the case in environmental models. The suitability of a technique will also depend on the computational requirements of the models. For example, climate models take too long to run to make the more sophisticated SA techniques viable.

Here we will work with a fairly simple example used in Saltelli et al. (2004). The model computes the minimum height above ground that a bungee jumper will reach, based on his/her starting height from the ground (H), mass (M), and the number of strands in the bungee cord (s). The idea is that we want an exciting jump (low minimum height) that we can survive (height ¿ 0), and we don't know any of the input values perfectly. Let's start by defining the model, which can be determined based on the solution to a linear oscillator equation as
$hmin = H - (2 * 9.8 * M/(1.5 * s))$

To make calling the function easier, we are going to define it in R as a function which takes one vector with three elements as input, as opposed to three separate variables:

```
hmin = function(x) (x[1] - 2 * x[2] * 9.8 / (1.5 * x[3]))
```

One-at-a-time (OAT) local SA

A simple type of sensitivity analysis is to investigate each factor individually. One way to do this is to define a base case for all parameters, and then go one-at-a-time through each parameter and perturb it from its base value. This is called a local sensitivity analysis, since we are really only computing a partial derivative. If the effect of one factor depends on the value of other factors, then a local SA can be misleading. But often it will provide at least a qualitative view of which factors are important and which are not (i.e. screening). Let's try that for our model. We will first define a range of uncertainty for each parameter, and then test the effect of changing each parameter from its minimum to maximum value.

```
par.names = c("Height","Mass","# Strands")
par.mins = c(40, 67, 20)                          #DEFINE RANGE FOR PARAMETERS
par.maxs = c(60, 74, 40)
```

```
par.mids = .5*(par.mins + par.maxs)      #DEFINE MIDPOINTS FOR PARAMETERS
npars=length(par.mins)                   #NUMBER OF PARAMETERS

difs = numeric(length=npars)             #ARRAY TO HOLD RESULT
for (par in 1:npars) {
        input1 = par.mids; input2 = par.mids    #MAKE TWO DIFFERENT INPUT
            VECTORS
        input1[par] = par.mins[par]      #MAKE VALUE OF PARAM IN 1ST VECTOR =
            MIN VALUE
        input2[par] = par.maxs[par]      #MAKE VALUE OF PARAM IN 2ND VECTOR =
            MAX VALUE
        difs[par] = hmin(input2) - hmin(input1) #COMPUTE DIFFERENCE B/W MODEL
            OUTPUTS WITH MAX AND MIN PARAMETER VALUES
}

barplot(difs, xlab="Parameter", ylab = "Max - Min", names.arg = par.names)
          #MAKE PLOT
```

We can see from this that the s and H appear important factors, while M does not. Note that this is very dependent on the range we have defined for each parameter.

One-at-a-time (OAT) global SA

For many models, we may expect that the effect of uncertainty in a factor will depend on the values of (i.e. interact with) other factors, and so we will want to find a better measure of sensitivity than the local measure. A useful technique for this is the method of Morris. The Morris method is still an OAT approach, which means it takes only a few computations. It is described in detail in Saltelli et al. (2004). Here I have written a function in R to perform the Morris Method, following pp100-101 in the book. Note I assume that the distribution of each parameter is uniform, but this could be changed.

```
#FIRST DEFINE INPUTS INTO MODEL
k = npars
r = 10           #NUMBER OF TIMES TO COMPUTE EFFECT FOR EACH PARAMETER
p = 4            #NUMBER OF POSSIBLE LEVELS FOR EACH PARAMETER (SHOULD BE EVEN)
delta = p/(2*(p-1))      #INCREMENT TO ADJUST PARAMETER VALUE BY

#DEFINE FUNCTION TO SAMPLE FROM SET {0,1/(p-1),...1}
base.samp = function(p,n) }
        x = (p-1-p/2) * runif(n) #NEED TO LIMIT TO RANGE OF (0, 1-delta)
        round(x) / (p-1)
}

par.mins = c(40, 67, 20)                 #DEFINE RANGE FOR PARAMETERS
par.maxs = c(60, 74, 40)

#START FUNCTION
morris = function(k,r,p,delta,par.mins,par.maxs,fun) {
#NOTE THE LAST INPUT IS THE MODEL NAME FOR SENS ANALYSIS

par.range = par.maxs - par.mins
effects = array(dim = c(k,r))

for (r.ind in 1:r) {
        J = array(1,dim=c(k+1,k))        #AN ARRAY OF ONES
        B = lower.tri(J) * 1             #A LOWER TRIANGULAR ARRAY OF ONES
```

```
        xstar = base.samp(p,k)          #BASE VECTOR
        D = array(0,dim=c(k,k))         #AN ARRAY WITH EITHER 1 OR -1 IN
            DIAGONAL
        diag(D) = 1.0 -2*(runif(k) < .5)
        P = array(0,dim=c(k,k))         #A RANDOM PERMUTATION ARRAY
        diag(P) = 1
        P = P[,sample(c(1:k),k,replace=F)]
        Bstar =  (t(xstar*t(J)) + (delta/2) * ((2*B - J)%*%D + J)) %*% P
        y = numeric(length=(k+1))       #VECTOR TO HOLD MODEL OUTPUT
        for (i in 1:(k+1)) y[i] = fun(par.mins + par.range * Bstar[i,] )
        for (i in 1:k) {
                par.change = Bstar[i+1,] - Bstar[i,]    #FIND WHICH PARAMETER
                    CHANGED AND WHETHER UP OR DOWN
                i2 = which(par.change != 0)
                effects[i2,r.ind] = (y[i+1] - y[i]) * (1 - 2 * (par.change[i2]
                     < 0))
        }
}
effects         #RETURN THE k x r ARRAY OF COMPUTED EFFECTS
}

#NOW RUN THE MORRIS METHOD FOR hmin
effects = morris (k,r,p,delta,par.mins,par.maxs,hmin)
mu = apply(effects,1,mean)
sigma = apply(effects,1,sd)
mu.star = apply(abs(effects),1,mean)
plot(mu.star,sigma,pch = c("H","M","s"))
```

The interpretation of results is that factors with a large average absolute effect (as measured by mu.star) are generally more important than those that do not. At the same time, since we have measured the effect of each factor for different values of other factors, we can use sigma to estimate how much the effect of the factor depends on other factors. In this case, H and s appear important, while only s appears to depend on the value of other factors. Notice that the total number of model runs was $r * (k + 1)$.

Variance-based global SA

The screening methods described above do not provide a very intuitive quantitative measure of factor importance. Variance-based SA techniques offer the ability to provide a well-defined measure of sensitivity: the amount (i.e. fraction) of variance in output that is reduced if we were to know the factor perfectly. Again, the trick is to get a global measure by sampling for different values of the parameter space. But in this case we will not vary each one individually, but vary all of them together.

```
#The first step is to define the type of distributions for the inputs. Again,
    we will assume that they are independent , uniform distributions , with
    the following ranges :


par.mins = c(40 , 67, 20) # DEFINE RANGE FOR PARAMETERS
par.maxs = c(60 , 74, 40)
par.range = par.maxs - par.mins
k=length(par.range)
#Now sample nrun draws from each of these distributions to generate an nrun x
    k input matrix , called M

nrun = 5e3
M = array(runif(nrun*k),dim=c(nrun,k))          #CREATE U(0,1) n x k MATRIX
```

```
for (i in 1:k) M[,i] = M[,i] * par.range[i] + par.mins[i]       #TRANSFORM TO
    FIT RANGE OF EACH PAR

#Now we are going to make a different input matrix, M2, the same way
M2 = array(runif(nrun*k),dim=c(nrun,k))                  #CREATE U(0,1) n x k
    MATRIX
for (i in 1:k) M2[,i] = M2[,i] * par.range[i] + par.mins[i]     #TRANSFORM TO
    FIT RANGE OF EACH PAR

#Now we are going to make K different matrices, where in the jth matrix, all
    parameters are taken from M2 except parameter j is taken from M
NJ.list = list()
for (j in 1:k) {
        temp = M2
        temp[,j] = M[,j]
        NJ.list[[j]] = temp
}


#Now we compute the expected value (EY) and total variance (VY) of Y, for
    which we will use M2
#For computational reasons, we will estimate EY^2 using both M and M2
y1 = y2 = y3 = numeric(length = nrun)
for (i in 1:nrun) {
                y1[i] = fun(M[i,])
                y2[i] = fun(M2[i,])
}
EY=mean(y1)
VY=sum(y1*y1)/(nrun-1)-EY^2

#Now, to compute the first-order effects (S) of factor J, we need to compute
    the values UJ, as discussed in class
#To compute the total effects of each factor (ST), we use M2 instead of M in
    the cross product
#i.e. for S the only thing common b/w M and NJ is factor j, for ST the only
    thing different between M2 and NJ is factor j. This gives us the effect of
     all other factors combined, plus their interactions. So if we take 1 -
    that, it gives us the total effect of factor j

S = ST = numeric(length=k)
for (j in 1:k) {
        NJ = NJ.list[[j]]
        for (i in 1:nrun) y3[i] = fun(NJ[i,])
        UJ = sum(y1*y3) / (nrun-1)   #here everything but factor j is
            resampled
        UJ2 = sum(y2*y3) / (nrun-1)  #here only factor j is resampled
        S[j] = (UJ - EY^2) / VY
        ST[j] = 1.0 - (UJ2 - EY^2) / VY
}
```

Exercises

Problem 1: (3 pts) Rerun the Morris method for the bungee model, but using different ranges for the parameter values. Plot mu.star vs. sigma for these different parameter values. How are the values changed if the range and/or midpoint of the distributions changed?

Problem 2: (4 pts) Now we will run the Morris method on the monsoon model you worked with in assignment 2. Recall in that assignment you wrote a function that output a vector of annual rainfall amounts. First, modify that function so that the default number of years to run is only 200, instead of 6030. (This will allow the SA to run faster). Now write a function that returns the average rainfall of that vector. Apply the Morris method to this function with the following parameter ranges:

```
par.names = c("Pstrong","Pweak","tau","prmax","P_init")
par.mins=c(8,0,14,.7,.65)
par.maxs=c(10,2,21,.9,.85)
```

Use p=4, r=20, and $delta$=2/3. This should take about 60 seconds to run, so you can use the chance to stand and stretch a little. Plot mu.star vs. sigma for the five factors. What can you conclude about the importance of each factor?

Problem 3: Make a function that takes a function and parameter mins and maxs as input, and returns the main and total effects for each parameter using the variance-based global SA method. Apply this function to the bungee model in #1 above, and using nrun = 10,000. Make a figure showing both main and total effects and discuss whether this result is consistent with the results from the Morris Method for the same parameter ranges.

Note: The main and total effects should generally both be between 0 and 1. However, for computational reasons, sometimes factors that should have an effect close to zero will have slightly negative values for their main and/or total effects. So don't worry if some of your values come out negative.