Assignment 5: Parallel Computing

Ben Aldridge

March 23, 2013

Contents

List of Figures

1 Introduction

In civil and environmental engineering, statistical analysis of different problems is beginning to gain momentum. However, it is still rarely used, especially in practice. This is mostly due to a lack of knowledge, but also because of a lack of tools that incorporate statistics into their analysis. This paper presents the very beginning of a package with the aim of providing engineering tools that include the use of statistical techniques.

Many problems in engineering lend themselves to statistical analysis. One of the specialities that lends itself most to statistical analysis is geotechnical engineering. This field involves the study of how soil behaves when subjected to different types of loading. This becomes very difficult in practice since soil properties can change very quickly in the field. This distribution of properties is traditionally dealt with by choosing a representative value, commonly the mean or a conservative mean, and then performing analysis with this value. While this is easier, it also creates deceptive results, as we are not taking into consideration the probability that the material is completely different somewhere else. There are several methods that can be used in the statistical analysis of geotechnical problems, but only a few are discussed here, and only one is implemented into the package at this moment.

The package presented, entitled 'REngine', is an attempt by the author to begin brining free, open-source statistical tools to the engineering community. R, is a perfect choice for this since it is designed for statistical approaches, and its similarity to MATLAB, which is used a lot in the engineering community. R hits a middle zone between programs such as SAS, and MATLAB by having a statistical approach to problems, while still being easy to use for scientific computing.

2 Slope Stability Analysis

One area of interest to geotechnical engineers is the analysis of the stability of an earth slope, like the one seen in Figure 1. This slope could be anything from a dam or levee, to a slope at the side of a road. It may also have multiple soil types layered throughout the slope. The basic method of analysis is to create a 'failure surface', as shown by the arc through the slope in Figure 1, and see what the factor of safety is for that failure surface. The factor of safety for slope stability is defined as a ratio of the strength available to the strength required. To compute this, the area above the failure surface is divided into a number of 'slices'. This can also be seen in Figure 1. Each slice has a weight, which causes a driving force (lowers the strength available, shown by the downward arrows), and a force holding the slice from moving (as shown by the angled arrows along the arc). Each force can be calculated by several different methods, and then combined to get the factor of safety for that failure surface. To find the minimum factor of safety for the slope, the failure surface is changed by changing the radius and center point of the circle, until a minimum factor of safety is

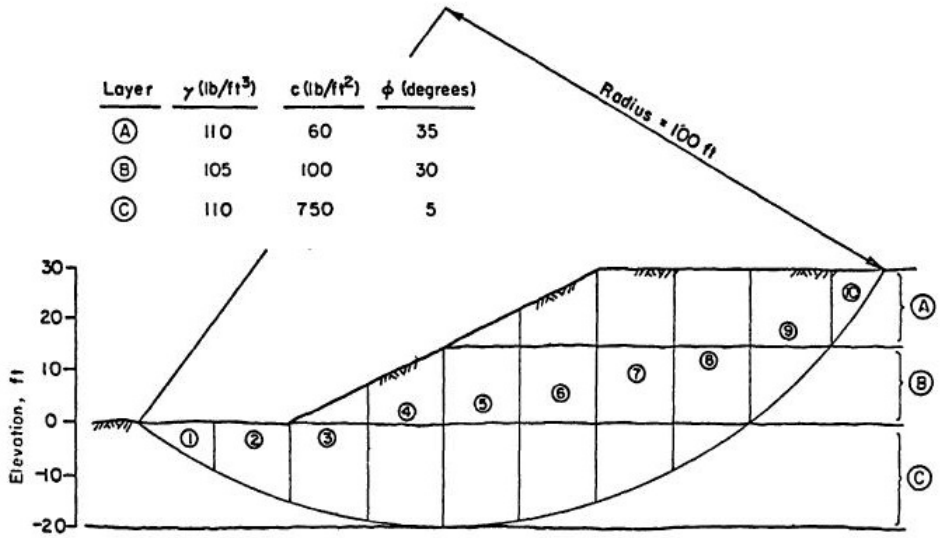

Figure 1: Example Slope

found. In traditional analysis, if this value is above about 1.3 (depends based on the risk associated with the slope failing), we say the slope is 'safe'. Below this, to about 1.1, the slope may still be deemed safe based on the engineers judgement. Lower then 1.1 and the slope is usually deemed not safe.

Many unknowns go into these models, the largest of which being the model itself. There are many aspects of slope behavior that are not covered by the method of slices, no matter how the analysis is done. Accounting for this probabilistically requires calibration with real data, and was not done for this package. The next largest unknown are the soil properties. While usually samples are taken for this type of analysis, properties within the slope can vary wildly. Several methods can be used to compensate for this variation. In this packages, a simple approach is taken in which the standard deviation and mean of each property are randomly varied, with a minimum factor of safety being found for each of this variations. The factors of safeties are then combined via,

$$P_f = 1 - \Phi[\beta]$$

$$\beta = \frac{\overline{F_s} - 1}{\sigma(F_s)}$$

These equations are assuming that the factor of safety has a normal distribution. It can also be done assuming a log-normal distribution.

This value is much better, even though the analysis is simple. This value gives us a probability of failure, which takes into account the variability of the soil.

However, it is not perfect. One limitation is that the analysis requires the model to be run multiple times, which has an associated cost. If we wanted to get rid of this problem, we would need to use random variables and create a closed form solution. While this would not be impossible, it is not robust, and would need to be done for every method of slices analysis method. Another problem is that the model assumes that we have a representative sample of the soil. While this will usually be true if enough sample are taken, it will not be true in many cases. To compensate for this, we would need to use geostatistics to find the probability that we have a representative sample, and then include that in our analysis. In this package, this was not done due to time constraints, but is planned to be added in the future.

3 Discussion of the Package and Associated Code

The function consists of three main functions.

1. The 'material' function, which produces a material data frame
2. The 'geom' function, which produces a geometry data frame
3. The 'SlopeStability' function, which performs the analysis discussed in the previous section

Using these functions, we can perform a full slope stability analysis as discussed above. Much of what is discussed in the following section is also contained in the help files, but the following sections are mostly to discuss problems with the current code, and what may be included in the futre.

3.1 The material Function

This function is meant to produce an object that gives the properties of each material. As it is right now, it is very flexible and serves as a building block for the whole package. It allows for the inclusion of any properties and any number of materials as desired, as long as each material has the same properties. If different properties are to be used, a new material data frame must be made. This was done so that each material data frame would represent a material set for a specific problem, without having to make dozens of different functions. The only catch is that a specific naming nomenclature must be followed. For example, in the slope stability problem, the properties 'cohesion_eff' (effective cohesion), 'FrictionAnlge_eff' (effective friction anlge), and 'UnitWeight' (unit weight), must be included as shown here. These names will most likely be changed to create a more memorable system at some point in the future.
Another future addition will be to make this into a class. This will be nice as it will allow for the creation of methods for other functions. It will also allow future functions to identify the elements being passed to it with greater ease.
Efficiency is not much of an issue in this function as it does relatively simple tasks. Code for the function can be found in the Appendix.

3.2 The geom Function

This function shares many similarities to the material function. It is design to allow for the user to input the desired geometry of the problem. Again, this function is very flexible as it is designed to be a building block for the whole package. The user can give any x and y coordinates of a polygon and give each polygon a tag that represents its name. The difficulty with this function has been converting it so that plotting may occur. To plot polygons, the points have to be placed in the correct order, either going clockwise or counter-clockwise. Much effort was put into sorting the values in the data frame so that they were automatically like this, but to no avail. Further investigation may lead to a method for doing this.
It may be better however to change from using this function, and possibly the material function, to the 'sp' package. This package was unfortunately discovered to late to implement into the current iteration of the package, but it is made for handling spatial problems, like we have in many engineering problems. It is not certain whether this package will work for the packages needs, but initial investigations are promising.
Again, the efficiency of the geom function is not much of an issue as it does relatively simple tasks. Code can be found in the Appendix.

3.3 The SlopeStability Function

This is the main function in the current iteration of the package. It does the analysis as discussed in the Slope Stability Analysis section. It takes the geometry of the slope and the properties of the slope material and calculates the probability of failure. To do this, several subfunctions are used to do various tasks.
The first subfunction is the topLayer function. This function takes the geom data frame and finds the points that are on the surface of the model. This is done by simply sorting the values and finding the largest y values. The code can be seen in the Appendix.
Another subfunction is the 'F' function. This function calculates the factor of safety (F in this case) given material properties, slice geometry, and the location of the failure surface. Only one analysis method is currently available, which is the OMS (ordinary methods of slices) method. Code was written for the Bishop method, but difficulties in using uniroot to iteratively find the value of F (since F can not directly be solved for in this method) caused this method to be abandonded for now. Research was done into using Spencer's Method, but again it required an iterative solution, so it was abandonded. In the end, OMS was the only method fully implemented since it did not require an iterative solution. One problem, that was found late in the development in the OMS method was that the slope at the bottom of the slice, which is used in the sine and cosine functions, was causing negative factors of safety, which is not possible. This may have been the reason the uniroot function was not behaving as expected, but that has not been tested yet. Each method has its own function, which

does all the simple calculations, in vectorized form. This allowed them to be very fast. It should be noted that the reason the OMS method does not require iteration is because it is simple, and was developed before computers. For this reason, it is also the least accurate.
One other subfunction is the cirIntercept function. This function uses uniroot to find the intersection of the failure surface with the slope surface, so that slices can be made.
Overall the function runs fairly fast. The main slowdown for the function is the iteration over the different variations of soil properties. The timing of the SlopeStability function at different number of variations can be seen in Figure 2. To help with the calculation speed of the function, the variations are run over in parallel, on machines where this is an option. At a future date, an option will be added for user control (if desired) of this portion. However, for now it is automated with the number of nodes in the cluster being equal to the number of cores, or at least two if no cores are found. It has also been limited to only 8 cores as a max.

The function was much faster than I was anticipating, only taking about one minute to run 1000 variations. It is also nice to see that the relation looks linear between the number of runs and the time, meaning that this can be scaled fairly well. Further speedup will be needed for higher numbers of variations, and can be obtained by possibly writing some of the functions in C. Also, examination of the code would most likely reveal other areas of speedup within the R code itself.
The problem used for this simulation (Figure 2), is shown in Figure 3. This also shows off the base for the future function to plot this simulation. Many more features are needed (including making the actual method/function), but it is a start. As can be seen, this is a very simple model, which brings up the main problem with the current function.
As it is now, the function is nice, but can only be used on extremely simple models. As it is now, only one material can be used, and only one geometry can be given (the geometry can be changed but you can't have to geometries together, even if they were the same material). These can be added, but require significant complication of the code. It was decided early on that it would be better to focus on making a simple model that worked, and then making it more complicated. Since the first task has been completed, complication can be done now.

One other major limitation of the code is in the search for the minimum factor of safety. As it is now, the user must be very careful to not give the function a range of values for the circle center point or the radius that could result in a circle that went outside the geometry. This could be a circle that went below the base of the model or one that was so small that it did not enter the slope. Again, this is not super difficult to do, but requires significant complication of the code. Again, this is now a priority.

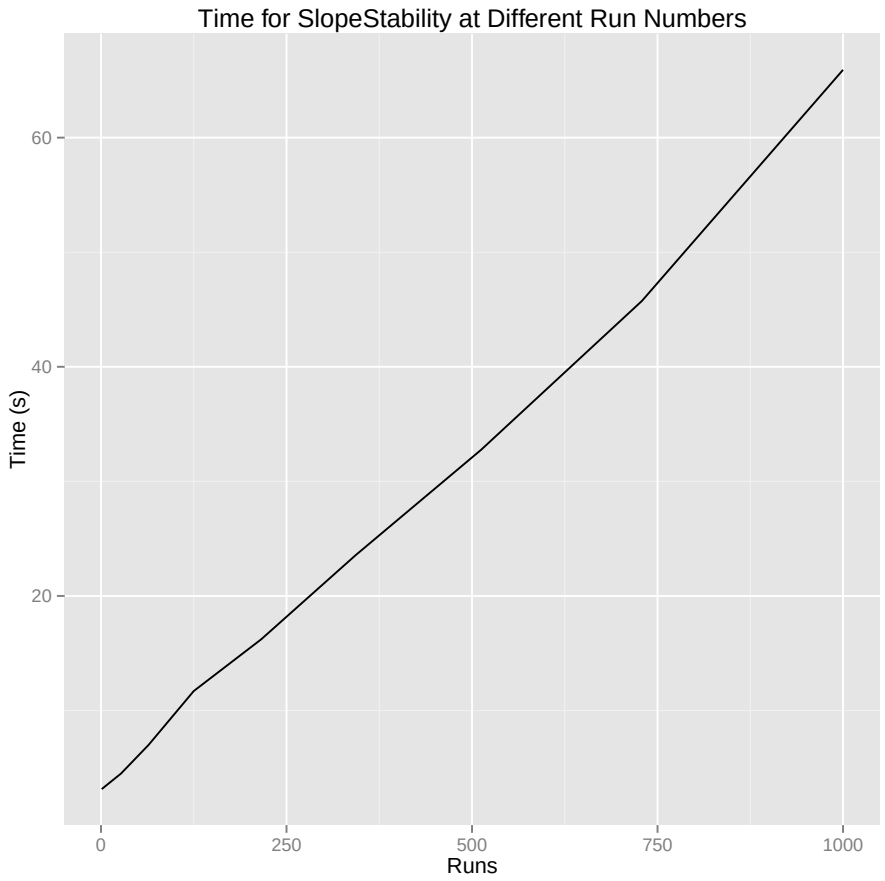

Figure 2: Time for SlopeStability at Different Run Numbers

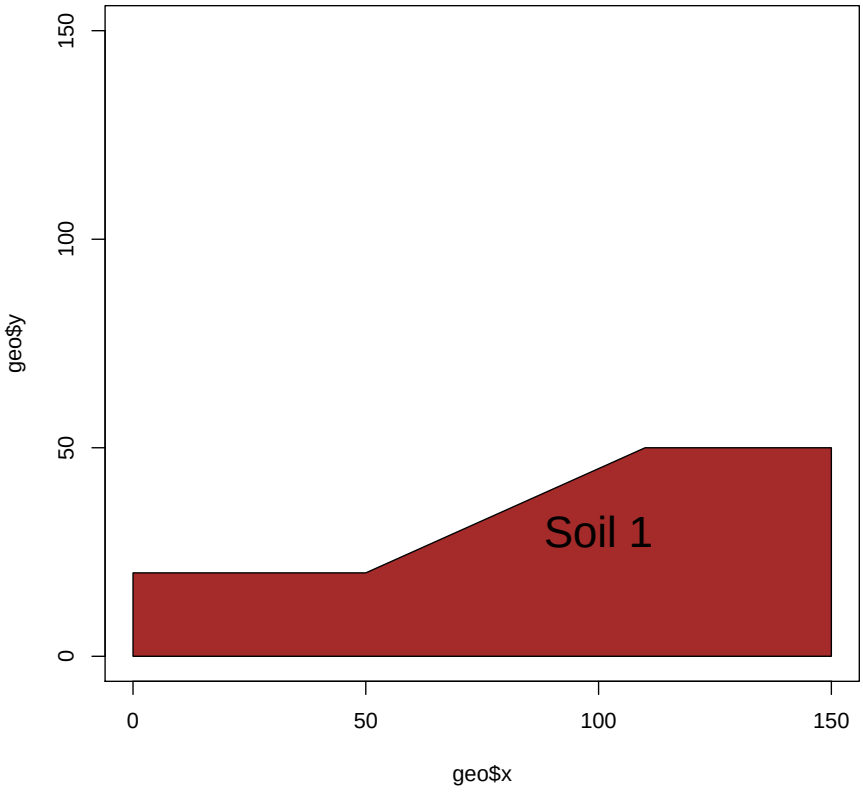

Figure 3: Model of example problem

4 Future Tasks

Many future tasks remain. Some have been mentioned above, while others are seen below. However, the main goal for this initial part was to develop a framework for the package. This was done with the geom and material functions. While simple, these functions provide an approach to many engineering problems, and will be useful in many applications.
Some main tasks to be done in the near future include,

1. Plotting of the geometry object
2. Making the SlopeStability function more robust
3. Add more geotechnical engineering problems, such as soil consolidation (from weights being put on top of soil) and soil liquefaction analysis. In soil liquefaction, many empirical relations are used, but again, only representative values are chosen, even though the data exists to use a probabilistic analysis.
4. Extend to other engineering problems, such as traffic flow, waste water treatment, structural analysis, among others. Some will be added that will not necessarily be used with statistical techniques, but will be usefull in a set of engineering tools.

5 References

In this package, a method was shown for converting the factor of safety to a probability of failure. This was found in 'Reliability-Based Design in Geotechnical Engineering: Computations and Applications' edited by Kok-Kwang Phoon. Many general concepts were also learned from this and will be implemented in the future. Also used for many of its concepts was 'Applied Statistics and Probability for Engineers' by Douglas C. Montgomery and George C. Runger.

6 Appendix

6.1 Code: material

```
material = function(...)
{
    require(moments)
    D = list(...)
    data.frame(MEAN = sapply(D,mean),
               SD = sapply(D, sd),
               SKEWNESS = sapply(D, skewness),
               KURTOSIS = sapply(D, kurtosis),
```

```
                N = sapply(D, length))
}
```

6.2 Code: geom

```
geom = function(...)
{
    D = list(...)
    N = names(D)
    out = mapply(FUN = function(l, n)
        {
            temp = data.frame(name = rep(n,length(l["x"]))
                , x = l["x"], y = l["y"])

        }, D, N, SIMPLIFY = FALSE)

    do.call(rbind, out)
}
```

6.3 Code: SlopeStability

```
SlopeStability = function(geom, materials, xrange, yrange
    , Rrange,
n = 10, N = 10,
method = "oms",
startX = mean(xrange),
startY = mean(yrange),
startR = mean(Rrange))
{

    #NEED TO LIMIT R TO THE SIZE OF THE AREA (SO CIRCLE
        IS CONTAINED)
    up = c(max(xrange), max(yrange), max(Rrange))
    low = c(min(xrange), min(yrange), min(Rrange))

    #Find the points that are at the surface
    tl = topLayer(geom)

    #Optimize over F======
    #Create value sets

    if(N==1)
    {
        fae = materials["FrictionAngle_eff","MEAN"]*pi
            /180
```

```
    ce = materials["cohesion_eff","MEAN"]
    g = materials["UnitWeight","MEAN"]
    pp = 0
}else
{
    fae = rnorm(n = N, mean = (materials["
        FrictionAngle_eff","MEAN"]*pi)/180, sd = (
        materials["FrictionAngle_eff","SD"]*pi)/180)
    ce = rnorm(n = N, mean = materials["cohesion_eff
        ","MEAN"], sd = materials["cohesion_eff","SD
        "])
    g = rnorm(n = N, mean = materials["UnitWeight","
        MEAN"], sd = materials["UnitWeight","SD"])
    pp = 0
}

vals = list(fae, ce, g, pp)
values = do.call(expand.grid, vals)
names(values) = c("fae","ce","g","pp")

#Run in parallel

num = detectCores()
if(num<2)
{
    num = 2
}
if(num>8)
{
    num = 8
}else
{
    num = 4
}
cluster = makeCluster(spec = num)

clusterExport(cl = cluster, varlist = c("values","
    method","n","up","low","Rrange","tl", "startX","
    startY","startR","F","oms","cirIntercept"), envir
    = environment())

fs = clusterApplyLB(cl = cluster, 1:dim(values)[1],
    function(i)
            {
                fae = values[i,"fae"]
                ce = values[i,"ce"]
```

```
                    g = values[i,"g"]
                    pp = values[i,"pp"]

                    temp = optim(par = c(startX, startY,
                        startR), TL = tl, fn = F, FAEr =
                        fae, cE = ce, u = pp, gam = g, met
                        = method, n = n, method = "L-BFGS-B
                        ", upper = up, lower = low)
                    temp$value
                })

    stopCluster(cluster)

    FS = unlist(fs)

    #Generate probability of failure (if applicable)
    if(N>1)
    {
    beta = (mean(FS) - 1)/sd(FS)
    Pf = 1 - pnorm(beta)

}else
{
    warning("Only one run done, returning Factor of
        Safety")
    FS
}
}
```

6.4 Code: topLayer

```
topLayer = function(geom)
{
    temp = lapply(unique(geom$x), function(val)
        {
            y = max(geom[geom$x==val,"y"])
            c(val,y)
        })
    out = as.data.frame(do.call(rbind, temp))
    names(out) = c("x","y")
    out[order(out$x), ]

}
```

6.5 Code: cirIntercept

```
cirIntercept = function(Xc, Yc, R, TopLayer)
{
temp = lapply(1:(dim(TopLayer)[1]-1), function(i)
    {
        m = (TopLayer$y[i+1] - TopLayer$y[i]) / (
            TopLayer$x[i+1] - TopLayer$x[i])
        b = TopLayer$y[i] - m * TopLayer$x[i]

        f = function(x, Xc, Yc, m, b, R)
    {
        (x - Xc)^2 + (m * x + b - Yc)^2 - R^2
    }

        x = try(uniroot(f, interval = c(TopLayer$x[i],
            TopLayer$x[i+1]), Xc = Xc, Yc = Yc, m = m, b =
            b, R = R), silent = TRUE)
#browser()
        if(class(x)=="try-error")
    {
        data.frame(int.x = NA, int.y =  NA, m = m, b = b,
            xrange_low = TopLayer$x[i], xrange_high =
            TopLayer$x[i+1])
    }else
    {
        y = m * x$root + b
        data.frame(int.x = x$root, int.y =  y, m = m, b =
            b, xrange_low = TopLayer$x[i], xrange_high =
            TopLayer$x[i+1])
    }
    })
do.call(rbind, temp)
}
```

6.6 Code: oms

```
oms = function(sliceCoords, FA_eff_radians, c_eff, gamma,
    u)
{
    b = diff(sliceCoords$x) #width of slices
    alpha = diff(sliceCoords$y_low)/b #angle at bottom of
        slice

    alphac = (cos(alpha)>0)*alpha + (cos(alpha)<0)*(alpha
        +pi)
```

```
    alphas = (sin(alpha)>0)*alpha + (sin(alpha)<0)*(alpha
        +pi)

    l = b/cos(alphac)

    h = mapply(FUN = function(a,b)
    {
        mean(c(a,b))
        }, (sliceCoords$y_high - sliceCoords$y_low)[1:(
            length(sliceCoords$y_low) - 1)], (
            sliceCoords$y_high - sliceCoords$y_low)[2:(
            length(sliceCoords$y_low))]) #mean height of
            the slice. This could be improved to calc W

    W = gamma*h*b

    out = sum(c_eff*l + (W*cos(alphac) - u*l)*tan(
        FA_eff_radians))/sum(W*sin(alphas))
    if(out<0)
    {
        browser()
    }
    out
}
```