Homework 11. Due by 5pm on Wednesday 11/18.

Parallel computing using a Linux cluster.

The following directions take you through running a multi-processor test script for the Statistics allocation on the UM Flux cluster. It would be helpful if you could send me a short email to say if you got all the way through or where you got stuck, or any other useful feedback. You're welcome to consult your peers.

1. **Getting a flux account, an Mtoken and an allocation**. You should already have a flux account and Mtoken. You should all have your uniquenames listed on the `stats_flux` allocation, so you have permission to submit jobs to this queue. Let me know if any issues arise with these.

2. **Logging on to flux**. Once you have a flux account and an Mtoken, you can enter the cluster through a login node, which is designed primarily for submitting jobs to the rest of the cluster

   `http://arc-ts.umich.edu/flux/using-flux/login-nodes/`

   You should not run big jobs on the login node, but you can do a little bit of light data analysis and code testing, as well as moving files around.

   You can log into flux using ssh. For security, you can only log in from a University machine, so if you're at home you have to access flux via first logging into login.itd.umich.edu, or via a VPN. It may be simplest to log in by from one of the department Linux terminals, using

   ```
   ssh  -l [your unique name] flux-login.engin.umich.edu -X
   ```

   The "-X" flag allows ssh to open up remote X windows on your local machine. If your machine is running X windows (i.e., if it is a Linux machine) then applications run remotely will look just like they are running on your machine. For example, you can plot data using R on the flux login node and you'll see the resulting analysis without having to export the data out of flux.

3. **Moving files to and from flux**. One way to do this is by `scp`. For example

   ```
   scp gompertzTest.R [your unique name]@flux-login.engin.umich.edu:
   ```

   copies `gompertzTest.R` from your current machine into your home directory on flux, and

   ```
   scp [your unique name]@flux-login.engin.umich.edu:sims.Rda .
   ```

copies `sims.Rda` back from flux to your machine. You can also use `scp -r` to copy directories.

4. **Modules and R packages**. Software must be loaded from modules. The command `module avail` shows what is available and `module list` tells you which are already loaded. We are going to remove the default R module and replace it with one that was constructed to be (i) sufficiently up-to-date for recent versions of the `pomp` package; (ii) consistent with an installed version of `Rmpi` that facilitates communication between multiple R processes.

   ```
   module rm R
   module load Rmpi/R-3.1.1/0.6-5
   ```

   Then, running R on the login node, install the R packages `pomp` and `doParallel` into the default location on your user file space, e.g., using the R command

   ```
   install.packages(c("pomp","doParallel"))
   ```

   As for the last homework, you may have to create a personal R library as follows:

   ```
   Warning in install.packages("pomp") :
     'lib = "/usr/cac/rhel6/R/3.1.1-gcc/lib64/R/library"' is not writable
   Would you like to use a personal library instead?  (y/n) y
   Would you like to create a personal library
   ~/R/x86_64-unknown-linux-gnu-library/3.1.1
   to install packages into?  (y/n) y
   ```

5. **Checking the queue**. `qstat` lists all jobs which are running (R), queued (Q) or completed (C). Type `man qstat`, or use google, to find more than you need to know. We are more interested in the stats allocation, and to check on that we need `module load moab` in order to run

   ```
   showq -w acct=stats_flux
   ```

   Another useful thing to try is `mdiag -a stats_flux`, which shows you who has access to the `stats_flux` allocation and the maximum number of cores accessible to it.

6. **Cluster courtesy**. You can see from `showq` who is currently running jobs, and if some very long jobs are running. It is proper behavior to email (or talk) to people running large jobs to request that they let you have a fair share of the resources. Sometimes, people do not realize that their jobs are much larger or longer than intended. If nobody else is currently running jobs, it is permissible to use the whole allocation yourself, since you can assume that people who may later want to share the resources with you will let you know.

7. **Submitting a job to the queue**. Download `gompertzTest.R` and `runTest.pbs` onto your local machine from the Stats 810 website, and then transfer them into your home directory on flux. `runTest.pbs` is a pbs script, i.e., a set of instructions for the queue written in the Portable Batch System language. The last line of `runTest.pbs` runs R in batch mode, with input file `gompertzTest.R`. Please edit the line

   ```
   #PBS -M ionides@umich.edu
   ```

   to send email concerning the job to your own account! To run the script, type

   ```
   qsub runTest.pbs
   ```

   This computes 100 simulations from the gompertz model, using the C code for the Gompertz model that we saw previously. This toy example runs very quickly, but, of course, the computational framework is applicable to much bigger tasks.

8. **Running an interactive cluster job**. For debugging R code, it is useful to be able to run the commands in the R source file sequentially, by pasting them into an interactive R session. To do this in a cluster environment, an interactive job can be submitted by

   ```
   qsub -I -V -A stats_flux -l procs=5,qos=flux,walltime=1:00:00 -q flux
   ```

   When this job starts, you get a command line prompt on the master host of the job. You could type `less $PBS_NODEFILE` to list the nodes you have access to, and check that you have successfully initated your session. Then, you could start an interactive R session (just type `R`), enter each line of `gompertzTest.R` sequentially, and investigate the resulting R objects.

9. **Using `foreach` for embarrasingly parallel computations**. Many statistical applications of cluster computing, involving simulation or bootstrap methods, simply require a parallel for loop. Once you can set up a `foreach` command such as the one in `gompertzTest.R`, for most statistical purposes you have mastered cluster computing!

10. **Examine the results**. After copying the results back from flux (or working on the login node) you could plot some of the results, e.g.,

    ```
    require(pomp)
    load("sims.Rda")
    par(mfrow=c(3,4))
    for(i in 1:12) plot(x=time(r[[i]]),y=states(r[[i]]),xlab="t",ylab="X")
    ```