An Example Workflow Featuring a Regression Simulation

Andrew Raim

1 Introduction

This example illustrates a workflow for a simple simulation. Consider a linear regression model with

$$y_i = \boldsymbol{x}_i^\top \boldsymbol{\beta} + \epsilon_i, \quad \epsilon_i \overset{\text{iid}}{\sim} \text{N}(0, \sigma^2), \quad i = 1, \ldots, n, \tag{1}$$

where $\boldsymbol{x}_i \in \mathbb{R}^d$ is a given covariate which is considered to be fixed. Let $\boldsymbol{\theta} = (\boldsymbol{\beta}, \sigma^2)$ represent the unknown parameters and let $\hat{\boldsymbol{\theta}} = (\hat{\boldsymbol{\beta}}, \hat{\sigma}^2)$ be the maximum likelihood estimator (MLE). Here the MLE has well-known closed form expressions

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}, \quad \hat{\sigma}^2 = \frac{1}{n} \boldsymbol{y}^\top (\boldsymbol{I} - \boldsymbol{H}) \boldsymbol{y},$$

where $\boldsymbol{X}$ is the $n \times d$ matrix with rows $\boldsymbol{x}_i^\top$ and $\boldsymbol{H} = \boldsymbol{X}(\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top$. A property of interest for the MLE is its mean-squared error

$$\text{MSE}(\hat{\boldsymbol{\theta}}, \boldsymbol{\theta}) = \text{E}\left[\|\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}\|^2 \right]$$

with respect to the true data generating parameter $\boldsymbol{\theta}$. We will prepare a small simulation to evaluate the MSE for $\beta = (-1, 1)$, $\sigma \in \{1, 2, 3\}$, and $n \in \{50, 100, 200\}$. For each combination of (σ, n) ("level" of the simulation), we will repeat the following for $r = 1, \ldots, R = 200$: first generate observations $\boldsymbol{y}^{(r)} = (y_1^{(r)}, \ldots, y_n^{(r)})$ from model (1) using the current level of $\boldsymbol{\theta}$ and n, then obtain the MLE $\hat{\boldsymbol{\theta}}^{(r)}$. Using the empirical sampling distribution of $\hat{\boldsymbol{\theta}}^{(1)}, \ldots, \hat{\boldsymbol{\theta}}^{(R)}$, we may then approximate

$$\text{MSE}(\hat{\boldsymbol{\theta}}, \boldsymbol{\theta}) \approx \frac{1}{R} \sum_{r=1}^{R} \|\hat{\boldsymbol{\theta}}^{(r)} - \boldsymbol{\theta}\|^2$$

This simple simulation can be run within an R script in a matter of seconds or minutes. We can create nested loops to iterate through the crossed levels of σ and n, and then the R repetitions. Now imagine a more involved study where computing each estimate takes a matter of minutes or hours. Also, instead of having only 9 levels of the simulation, perhaps we have hundreds. Rather than saving just the MLE repetitions, we may also want to save the generated data $\boldsymbol{y}^{(r)}$, and diagnostics from each fit. This motivates the following workflow:

- Each level of the simulation will be placed into its own dedicated folder with a launcher script. Any log files and output from the level will be saved here as well.
- The folders and launcher scripts will be generated by a script that we write.
- We will utilize one or more "workers", whose job is to search the folders for available work. A folder is reserved and has its launcher executed by at most one worker.
- If a level fails, or if we wish to investigate its result, we can manually enter the folder and run the launcher or load the results.
- We will write a script to extract results from the folders and produce the desired results.

This kind of "embarrassingly parallel" computing is typical in many statistical projects: no communication is needed across levels during the simulation. Coordination is only needed to initially generated the workload and in post-processing. We may therefore wish to take advantage of multiple CPUs to process our workload faster, without the extra sophistication of frameworks such as MPI or the `parallel` package.

2 Workflow Implementation

The following scripts are used to implement our workflow for the regression simulation:

- 'gen.R' generates the folder structure and creates a 'launch.R' script in each folder.
- 'util.R' contains several utility functions used by other scripts.
- 'sim.R' contains the main logic to run each level of the simulation. The program sleeps for a few moments after each repetition to give the feeling of a more computationally demanding simulation.
- 'launch.R' is responsible for setting variables specific to the simulation level and running 'sim.R'.
- 'worker.py' is a Python script that seeks simulation levels which have yet been run and attempts to run them.
- 'analyze.R' handles post-processing after the simulation is completed. A table of MSEs is presented as the final result.

All scripts mentioned above, except for `worker.py`, are specific to this simulation and must be rewritten or customized for other applications.

Let us run `gen.R` on the command line. The following displays will shown as a Linux prompt, but other environments should work similarly.

```
$ R CMD BATCH gen.R
```

The following folders and directories are produced.

```
sigma1_n1  sigma1_n3  sigma2_n2  sigma3_n1  sigma3_n3    xmat-n2.rds
sigma1_n2  sigma2_n1  sigma2_n3  sigma3_n2  xmat-n1.rds  xmat-n3.rds
```

Folders of the form `sigmaA_nB` correspond to the simulation level for the Ath value of σ and the Bth value of n, for $A, B \in \{1, 2, 3\}$. Files of the form `xmat-nB.rds` represent the design matrix $\boldsymbol{X}$ to be used when n is taken to be the Bth level.

Let's inspect `launch.R` in the folder `sigma1_n1`.

```
set.seed(1234)

beta_true = c(-1,1)
sigma_true = 2.000000
xmat_file = "../xmat-n1.rds"
N_sim = 200

source("../sim.R")
save.image("results.Rdata")
```

We could run this manually to verify that our code generation script is correct and that all resources are in the appropriate place. Note that it assumes the `sigma1_n1` folder is our current working directory. Furthermore, `sim.R` and `xmat-n1.rds` should be in the parent directory.

Once we are certain the code and folders are laid out correctly, we will want to use a worker(s) instead of running the study manually. Suppose

- The script 'worker.py' is located at '/path/to/worker.py'.

- The path to our generated simulation folders is located at '/path/to/study'.

The following command invokes one worker.

```
python3 /path/to/worker.py -b /path/to/study -p "sigma(.*)_n(.*)" -c 'R CMD BATCH launch.R'
```

Note that `-b` is used to specify one or more paths which contain work folders, `-p` is used to specify one or more patterns used to identify work folders (as opposed to other folders which we should ignore), and `-c` specifies a command to run if we find a work folder whose job has not yet been attempted. Use the `-h` flag to display a help message, including the command line format, for `worker.py`.

A worker logs its activity to stdout. This can be redirected to a file if desired.

```
2021-06-09 18:16:05 - Worker ID: 75e1421210cc92d62e5403313cab33f0
2021-06-09 18:16:05 - Working directory: /sim-util/examples/regression-sim
2021-06-09 18:16:05 - Searching 1 basepaths for available work
2021-06-09 18:16:05 - Basepath[0]: /sim-util/examples/regression-sim
                      Pattern: sigma(.*)_n(.*)
2021-06-09 18:16:05 - Lockfile in sigma2_n2 exists, skipping
2021-06-09 18:16:05 - Lockfile in sigma1_n2 exists, skipping
2021-06-09 18:16:05 - Lockfile in sigma1_n3 exists, skipping
2021-06-09 18:16:05 - Lockfile in sigma2_n1 acquired
2021-06-09 18:19:25 - Processed 1 jobs and worked for 0.055721 total hours so far
2021-06-09 18:19:25 - Lockfile in sigma3_n1 acquired
```

Let us look in `sigma2_n1`, which was completed by the worker above.

```
$ ls sigma2_n1
launch.R  launch.Rout  results.Rdata  worker.err  worker.lock  worker.out
```

The files `worker.out` and `worker.err` capture any output from stdout and stderr while running the job. The file `worker.lock` is placed as a marker to indicate that a job has been reserved by a worker; only one worker may create this file, and it can run the job only upon successful creation of the file. If a job fails and you would like a worker to consider it again, delete the corresponding `worker.lock`.

The file `worker.lock` contains the ID of the worker that reserved it. This can be linked back to the `Worker ID` reported in the log output, which may help if it is necessary to determine which worker ran a job.

```
$ cat sigma2_n1/worker.lock
Reserved by worker: 75e1421210cc92d62e5403313cab33f0
```

When working on a remote server, it may not be possible to keep a persistent session open for long periods that may be needed for a simulation study. In a setting where a scheduler such as PBS or Slurm is used to allocate jobs to compute nodes, each worker may be submitted as a job to the scheduler. In an environment without a scheduler, running workers via a terminal multiplexer such as tmux or GNU screen allows terminal sessions to detach and reattach as needed to provide some persistence.

After the simulation is complete, we may run `analyze.R` to extract the estimates from each folder and build a table of MSEs.

```
R> source("analyze.R")
               n1         n2         n3
sigma1 0.07391762 0.03495352 0.01905428
sigma2 0.71125615 0.30743114 0.18523518
sigma3 3.15877261 1.32028399 0.82559688
```

As we may expect, the MSE increases as σ increases with n fixed, but decreases when n is increased and σ is fixed.