A Regression Simulation as a Workflow Example

Andrew Raim[*]

1 Introduction

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

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

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

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

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 $n \in \{50, 100, 200\}$. Each combination of (σ, n) will be referred to as a "level" of the simulation. For each level, 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 $\boldsymbol{\theta}$ 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

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

The end result will be a table with the following form.

	$n = 50$	$n = 100$	$n = 200$
$\sigma = 1$	$\ldots$	$\ldots$	$\ldots$
$\sigma = 2$	$\ldots$	$\ldots$	$\ldots$
$\sigma = 3$	$\ldots$	$\ldots$	$\ldots$

This simple simulation can be run within an R script in a matter of seconds. We can create nested loops to iterate through the crossed levels of σ and n, and then loop through the R repetitions. Now imagine a more involved study where computing each estimate takes on the order 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 so that unanticipated results can be investigated after the run. This motivates the following workflow:

- Each level of the simulation will be placed into its own dedicated folder with a launcher script. Any logging and output relevant to the level will be saved here as well.
- The folders and launcher scripts will be generated by a script that we write.

[*]andrew.raim@gmail.com

- We will utilize one or more "workers", whose job is to search the folders for available work. Each folder is reserved by 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, check log files, run the launcher manually, and load the results.
- We will write a script to extract results from the folders and produce the desired results (a table of MSEs).

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 generate the workload and in post-processing. We may therefore take advantage of multiple CPUs to process our workload faster without requiring more sophisticated tools available for parallel and distributed computing. In contrast, individual computations which are too CPU or memory intensive for a single processor or compute node may benefit greatly from such tools; for example, the pbdR package provides an MPI implementation in R while the `parallel` package supports parallel computing which offers less control over communication but is simpler to use.

2 Workflow Implementation

The following scripts are used to implement our workflow for the regression simulation. They are specific to this simulation and would need to be rewritten or customized for other studies.

- `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 one level of the simulation. At the beginning, it checks to ensure that variables `N_sim`, `beta_true`, `sigma_true`, and `xmat_file` have been set. A sleep command is issued 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`.
- `analyze.R` handles post-processing after the simulation is completed. A table of MSEs is presented as the final result.

To execute the workflow and carry out the simulation, we use the `worker` script included in the `src` folder of this repo. The `worker` seeks out simulation levels which have not yet run and attempts to run them. There are currently two versions of `worker` which have similar functionality but different system requirements. The two versions should not be used simultaneously in the same study because they employ different file locking mechanisms.

1. The Bash version (`worker.sh`) is specific to Linux, but requires a fairly minimal environment; it makes use of tools such as `flock` for file locking and the `bc` calculator.
2. The Python version (`worker.py`) is more portable but requires Python 3.3 or higher.

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 value.

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

```
N_sim = 200
beta_true = c(-1,1)
sigma_true = 2
```

```
xmat_file = "../xmat-n1.rds"

source("../sim.R")
```

It is a good practice to run one launch script manually to verify that our generated code is correct: all variables should be set and all external files and data should be in the appropriate place. Here we assume the `sigma1_n1` folder is our current working directory, and that `sim.R` and `xmat-n1.rds` are 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 path to our generated simulation folders is located at `/path/to/study`.

We will demonstrate the Python version of the worker, but use of the Bash version is similar. First make sure the worker script is executable.

```
$ chmod +x /path/to/worker.py
```

The following command invokes one worker.

```
$ /path/to/worker.py -p '/path/to/study/sigma*_n*' -c 'R CMD BATCH launch.R'
```

- Option `-p` is used to specify one or more patterns used to identify work folders (as opposed to other folders which we should ignore). Arguments are interpreted as Bash patterns in both the Python and Bash versions of the worker script. Single quotes are used to prevent patterns from being expanded by the shell before being passed to `worker.py`.
- Option `-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, and quit.

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

```
2022-02-11 14:41:28 - Worker ID: 7de5ef0ff86a45cd66f1088bcb627585
2022-02-11 14:41:28 - Working directory: /sim-util/examples/regression-sim
2022-02-11 14:41:28 - Searching 1 patterns for available work
2022-02-11 14:41:28 - Searching pattern[0]: sigma*_n*
2022-02-11 14:41:28 - Lockfile in sigma2_n2 exists, skipping
2022-02-11 14:41:28 - Lockfile in sigma1_n2 exists, skipping
2022-02-11 14:41:28 - Lockfile in sigma1_n3 exists, skipping
2022-02-11 14:41:36 - Lockfile in sigma2_n1 acquired
2022-02-11 14:41:38 - Processed 1 jobs and worked for 0.002735 total hours so far
2022-02-11 14:41:38 - 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 only upon successful creation does the worker attempt to execute the job. If a job has failed and you would like it to be attempted 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: 7de5ef0ff86a45cd66f1088bcb627585
```

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. Example PBS and

Slurm submission scripts are provided for this example in the files `worker.qsub` and `worker.slurm`. We can launch several workers through PBS, for example, using the following.

```
$ qsub worker.qsub
100000.localhost
$ qsub worker.qsub
100001.localhost
$ qsub worker.qsub
100002.localhost
```

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 a persistent session.

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