Improving Computational Efficiency

Tyler Ransom

Duke University

June 18, 2012

Plan

Discuss how to improve computational efficiency in the following ways:

- shell scripts
- providing analytical gradient
- saving results along the way
- compiled routines
- mex files
- Matlab coder

Shell scripts

- By now you should be comfortable with doing batch submission on the Duke econ cluster (with the `matsub` command)
- We'll discuss how to automate this process using shell scripts
- Shell scripts are analogous to do-files or m-files
  - They just execute linux commands instead of Stata or Matlab commands

Anatomy of a Shell Script

- First set of lines is like a LaTeX preamble
  - Sets up options and gives the system some context
- Last set of lines is where user inputs commands
- Comments in shell scripts are done with #
- "Preamble" options are initialized with "#!" or "#$" so comments shouldn't start with those characters

Why use a shell script?

- Allows a user to automate processes, minimizing typographical errors
- Users can have the Duke cluster send an email message when the job is finished
- Users can create custom names for their scripts (instead of “matsub” or “statasub”)
- If users are performing computations via other languages like C++ or Fortran, shell scripts are the only way to submit those types of jobs to the cluster

Shell script example: Matlab

```
#!/bin/sh
#$ -S /bin/sh
#$ -N "my shell script name" [default is shell script
filename]
#$ -cwd -j y
#$ -V
#$ -M my_net_id@duke.edu -m e
/usr/local/bin/matlab -nodesktop -nodisplay -nosplash
-nojvm < my_m_file.m > my_m_file.log
```

Options you should know about

- "`#$ -N`" tells the system what name you want
- "`#$ -M my_net_id@duke.edu -m e`" tells the system you would like to email your duke address ("`-m e`" indicates it should only send email upon completion and not at the start)
- NOTE: you can specify multiple email addresses, but they all must be duke econ usernames
- I don't touch the other options, but you can find out more about what they do at `https://help.econ.duke.edu/wiki/help:batchcluster`

Shell script example: Stata

```
#!/bin/sh
#$ -S /bin/sh
#$ -N "job name"
#$ -cwd -j y
#$ -V
#$ -M my_net_id@duke.edu -m
/usr/local/stata11/stata-se -b do my_do_file.do
```

Shell script example: Fortran/C++

```
#!/bin/sh
#$ -S /bin/sh
#$ -N “job name”
#$ -cwd -j y
#$ -V #$ -M my_net_id@duke.edu -m
path_to_my_fortran_or_C++_exectuable
```

Providing Analytical Gradient

- Often, you will need to estimate a large-scale problem that takes an extremely long time to solve with `fminunc`
- Fminunc runs much more quickly when the user supplies the analytical gradient of the objective function
- Matlab doesn't have to numerically approximate the gradient, which speeds things up

Syntax for providing analytical gradient

- The analytical gradient is defined in the objective function and passed to `fminunc` as a second output of the function, e.g.

```
function [like, grad] = mylikelihood(b,X,y);
like = ...;
grad = ...;
end
```

Syntax for providing analytical hessian

- Users may also provide the analytical hessian to `fminunc`. The syntax for this is very similar to that of the gradient:

```
function [like, grad,hess] = mylikelihood(b,X,y);
like = ...;
grad = ...;
hess = ...;
end
```

Analytical gradient/hessian

- As with starting values, the more information a user provides fminunc, the better/more quickly the optimizer will operate
- It takes a lot of "programming time" to formulate the gradient and/or hessian, but it could have huge payoffs in terms of "computational time"
- Tradeoff between spending more time coding and spending more time waiting for code to run
- Matlab has the optimization option `'DerivativeCheck'` which allows the user to check how close her analytical gradient is to Matlab's numerical approximation

Saving results along the way

- One huge headache saver when you're doing a computationally intensive problem is to save your results along the way
- E.g. every 50 iterations, update the results matrix
- Then, if you encounter an unforeseen problem down the line, you still have the estimates within the past 50 iterations, which will allow you to pick up where you left off
- This can be implemented in a number of ways. I prefer generating a global variable that keeps track of the iteration count

Compiled Routines

Matlab has a number of built-in estimation functions, e.g.

- `regress` : OLS regression
- `glmfit` : GLS regression with a variety of models
  - Logit/Probit
  - Poisson
- `mnrfit` : multinomial regression (for multinomial choice models)
  - Logit/Probit
- `nlinfit` : nonlinear least squares

Compiled Routines

Advantages

- Much faster than `fminunc` (or any other optimizer)
- Includes helpful output like standard errors, sum of squared residuals, F-test for joint significance, etc.
- Works well with other compiled routines to get predicted residuals, confidence intervals, etc.

No sense in reinventing the wheel

- Unless your instructor/professor tells you to (so you'll learn what's actually going on)

Compiled Routines

- If you need to estimate a certain type of model, chances are there is a compiled routine in Matlab already
- However, most work in structural applied micro requires you to come up with your own model, which may not be compatible with a compiled routine
- You should still try to use compiled routines whenever possible, because they provide a huge bang for your buck

Mex files

- Matlab has the ability to link with compiled languages like C and Fortran
- A Matlab function that calls a C/Fortran function is called a mex file (Make EXecutable)
- The mex file acts exactly like a Matlab function, except it is written in a compiled language so it gets evaluated much more quickly and efficiently
- There is really no limit to how you can use Mex files with Matlab

Mex files

What you need to run a mex file:

- C/Fotran compiler that is compatible with Matlab
- Knowledge of C/Fortran (since the bulk of the mex file is written in this language)
- Gateway Routine (interfaces C/C++ and MATLAB)
- Computational Routine (written in C/C++/Fortran that performs the computations you want executed at faster speeds)
- Preprocessor Macros (for building platform-independent code)

Note: the first element of an array in C/Fortran is 0 but in Matlab it is 1

Compiled Languages

- If code runs so much more quickly and efficiently in compiled languages like C and Fortran, why not always use them for computationally burdensome jobs?
- Just as there is a tradeoff between “programming time” and “computation time” there is also a tradeoff between “computational efficiency” and “user friendliness”
- It is much more difficult to debug code and interactively work with data in compiled languages

Matlab Coder

- Mathworks has a program called Matlab Coder which coverts matlab scripts in to C++ files.
- This allows users to completely test and debug code in Matlab, then port that code over into C++ in a form that can be run on multiple platforms
- This resource is currently unavailable in the Duke Econ cluster, but it will become available sometime soon
- This is perhaps the best way to speed up Matlab code, and doesn't require an extensive knowledge of C++ to do it