Computing Challenge 2014
Preliminary Results

UO Economics Macro Group
April 25, 2014
Goal: Provide some evidence on which technical computing languages are fastest for some of the types of computations that economists use.

Languages considered (so far):

- Gauss 14.0.7 (Aptech)
- Matlab R2014a (MathWorks)
- Python 2.7.6
- Julia 0.2
Econometric Model

The laboratory will be estimation of an econometric model based on:


A probit model with covariate uncertainty.

Primary elements of estimation are large(ish) data samples, loops, random number generation, and lots of matrix algebra.
Econometric Model

- $S_t \in \{0, 1\}$ is a binary random variable.
- Our objective is to forecast $S_t$ using predictors available to a forecaster at the end of month $t - h$, collected in the vector $X_{t-h}$.
- We consider the possibility that only a subset of the predictors in $X_{t-h}$ is used.
- Define $\gamma$ as a predictor selection vector that indicates which predictors are used. Collect these predictors in the vector $X_{\gamma, t-h}$.
We use a probit model to link $X_{\gamma, t-h}$ to $S_t$:

$$y_t = \alpha + X'_{\gamma, t-h} \beta_{\gamma} + u_t; \quad u_t \sim \text{i.i.d.} \mathcal{N}(0, 1)$$

$$S_t = 1 \quad \text{if} \quad y_t \geq 0.$$ 

It follows that:

$$\Pr[S_t = 1 | \rho_{\gamma}, \gamma] = \Phi \left( \alpha + X'_{\gamma, t-h} \beta_{\gamma} \right)$$
Model Uncertainty

- There is uncertainty about which predictors should be used in the forecasting model. This is uncertainty about the true value of $\gamma$.

- Estimate $\gamma$ along with model parameters.

- A Bayesian approach to estimation, implemented using a Gibbs Sampler.
Posterior Simulation via Gibbs Sampler

- Define \( y = (y_1, y_2, \cdots, y_T) \) and \( S = (S_1, S_2, \cdots, S_T) \)

- The Gibbs Sampler is then implemented in two blocks:

  1. **Draw from** \( \pi (y | \rho_\gamma, \gamma, S) \)
     - Involves looping and drawing from truncated normal distributions.

  2. **Draw from** \( \pi (\rho_\gamma, \gamma | y, S) = \pi (\rho_\gamma, \gamma | y) \)
     - Metropolis-Hastings step as in Holmes and Held (2006).
     - This step involves computing a multivariate pdf for a \( 613 \times 1 \) vector of random variables.

- Draws converge to draws from \( \pi (\rho_\gamma, \gamma | S) \)
Data set is monthly measured from June 1960 to June 2011 (613 data points).

56 potential predictors.

All estimations performed on MacBook Pro with:
- 2.7 GHZ Intel Core i7 Processor
- 16 GB Memory
Baseline Results:

Based on “global best practices” for an economist who needs to code.

400 total Gibbs Simulations (200 burn in)

Timing based on average of 10 runs.

<table>
<thead>
<tr>
<th></th>
<th>Julia</th>
<th>Matlab</th>
<th>Python</th>
<th>Gauss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seconds</td>
<td>13.4</td>
<td>16.6</td>
<td>27.0</td>
<td>30.1</td>
</tr>
<tr>
<td>Relative</td>
<td>1.0</td>
<td>1.2</td>
<td>2.0</td>
<td>2.2</td>
</tr>
</tbody>
</table>
The log marginal likelihood is a key input into computation time

\[-log(|\tilde{V}|) - y'(\tilde{V})^{-1}y\]

Total Computation Time for Calculation of Marginal Likelihood

<table>
<thead>
<tr>
<th></th>
<th>Julia</th>
<th>Matlab</th>
<th>Python</th>
<th>Gauss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seconds</td>
<td>4.8</td>
<td>12.9</td>
<td>19.7</td>
<td>27.2</td>
</tr>
<tr>
<td>Percent of Total</td>
<td>36%</td>
<td>78%</td>
<td>73%</td>
<td>90%</td>
</tr>
</tbody>
</table>
A key to improving performance is to improve on built in routines used to calculate the log multivariate normal pdf.

The two key calculations are:

- $y'(\tilde{V})^{-1}y$
- $|\tilde{V}|$
Nathan Kubota’s Approach:

\[ L = \text{chol}(\tilde{V}) \]
\[ y'(\tilde{V})^{-1}y = y'(L'L)^{-1}y \]
\[ y'(\tilde{V})^{-1}y = (L^{-1}y)'L^{-1}y \]
\[ y'(\tilde{V})^{-1}y = B'B \]

where:

\[ B = L^{-1}y \]
\[ LB = y \]

Efficient routines exist to solve this system of linear equations (that is solve for B) without inverting L. These are especially fast since L is lower triangular.

Matlab: linsolve(); Gauss: qrtsol(); Python: dtrtrs(); Julia: \
Once we have computed $L = \text{chol}(\bar{V})$ we can also quickly compute $|\bar{V}|$ as the squared product of the diagonal elements of $L$.

Additional gains can be found by saving $L$ and reusing where possible.
## Results with Tricks to Speed Calculation of Marginal Likelihood

<table>
<thead>
<tr>
<th></th>
<th>Gauss</th>
<th>Matlab</th>
<th>Julia</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seconds</td>
<td>3.1</td>
<td>3.6</td>
<td>4.0</td>
<td>6.4</td>
</tr>
<tr>
<td>Improvement</td>
<td>90%</td>
<td>78%</td>
<td>70%</td>
<td>76%</td>
</tr>
</tbody>
</table>
## Computation Time

### What about Compiling?

<table>
<thead>
<tr>
<th></th>
<th>Gauss</th>
<th>Matlab</th>
<th>Python-Compiled</th>
<th>Julia</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seconds</td>
<td>3.1</td>
<td>3.6</td>
<td>3.7</td>
<td>4.0</td>
<td>6.4</td>
</tr>
<tr>
<td>Relative</td>
<td>1.0</td>
<td>1.2</td>
<td>1.2</td>
<td>1.3</td>
<td>2.1</td>
</tr>
</tbody>
</table>
What about a Longer Run?

40,000 total Gibbs Simulations (20,000 burn in)

<table>
<thead>
<tr>
<th></th>
<th>Minutes</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gauss - Baseline</td>
<td>71.9</td>
<td>13.5</td>
</tr>
<tr>
<td>Gauss</td>
<td>5.3</td>
<td>1.0</td>
</tr>
<tr>
<td>Python-Compiled</td>
<td>5.3</td>
<td>1.0</td>
</tr>
<tr>
<td>Julia</td>
<td>6.2</td>
<td>1.2</td>
</tr>
<tr>
<td>Matlab</td>
<td>8.6</td>
<td>1.6</td>
</tr>
<tr>
<td>Python</td>
<td>10.3</td>
<td>1.9</td>
</tr>
</tbody>
</table>
Preliminary Conclusions

- Don’t trust built in functions. They may be very inefficient for your problem (or any problem.)

- Don’t invert that matrix!

- You took a linear algebra class. Use it!

- For this problem, Gauss, Python (with compiling), and Julia seem to be the fastest for large runs.
Open Questions

• Gains from using graphical processor? (Nathan)

• Gains from parallelizing?

• Why does Matlab bog down on long runs? I suspect something with draws from truncated normal. More concerning if this is something more systematic. (Rich)