Computing Challenge 2014
Preliminary Results

UO Economics Macro Group
April 25, 2014

Computing Challenge 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:

Owyang, Piger and Wall (2012), "Forecasting National Recessions Using State Level Data"

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 γ as a predictor selection vector that indicates which predictors are used. Collect these predictors in the vector $X_{\gamma,t-h}$.

Forecasting Model

We use a probit model to link $X_{\gamma,t-h}$ to S_t:

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

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

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 γ.
- Estimate γ along with model parameters.
- A Bayesian approach to estimation, implemented using a Gibbs Sampler.

Posterior Simulation via Gibbs Sampler

- Define $\mathbf{y} = (y_1, y_2, \cdots, y_T)$ and $\mathbf{S} = (S_1, S_2, \cdots, S_T)$
- The Gibbs Sampler is then implemented in two blocks:
  1. Draw from $\pi(\mathbf{y}|\rho_\gamma, \gamma, \mathbf{S})$
     - Involves looping and drawing from truncated normal distributions.
  2. Draw from $\pi(\rho_\gamma, \gamma|\mathbf{y}, \mathbf{S}) = \pi(\rho_\gamma, \gamma|\mathbf{y})$
     - Metropolis-Hastings step as in Holmes and Held (2006).
     - This step involves computing a multivariate pdf for a 613×1 vector of random variables.
- Draws converge to draws from $\pi(\rho_\gamma, \gamma|\mathbf{S})$

Data

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

Computation Time

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.

	Julia	**Matlab**	**Python**	**Gauss**
Seconds	13.4	16.6	27.0	30.1
Relative	1.0	1.2	2.0	2.2

Computation Time

The log marginal likelihood is a key input into computation time

$$-log(|\bar{V}|) - y'(\bar{V})^{-1}y$$

Total Computation Time for Calculation of Marginal Likelihood

	Julia	**Matlab**	**Python**	**Gauss**
Seconds	4.8	12.9	19.7	27.2
Percent of Total	36%	78%	73%	90%

Improving on Built in Matrix Algebra Routines

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:

- $y'(\bar{V})^{-1}y$
- $|\bar{V}|$

Improving on Built in Matrix Algebra Routines

Nathan Kubota's Approach:

$$
\begin{aligned}
L &= chol(\bar{V}) \\
y'(\bar{V})^{-1}y &= y'(L'L)^{-1}y \\
y'(\bar{V})^{-1}y &= (L^{-1}y)'L^{-1}y \\
y'(\bar{V})^{-1}y &= B'B
\end{aligned}
$$

where:

$$
\begin{aligned}
B &= L^{-1}y \\
LB &= y
\end{aligned}
$$

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: \

Improving on Built in Inverse Routines

Once we have computed $L = chol(\bar{V})$ 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.

Computation Time

Results with Tricks to Speed Calculation of Marginal Likelihood

	Gauss	**Matlab**	**Julia**	**Python**
Seconds	3.1	3.6	4.0	6.4
Improvement	90%	78%	70%	76%

Computation Time

What about Compiling?

	Gauss	Matlab	Python-Compiled	Julia	Python
Seconds	3.1	3.6	3.7	4.0	6.4
Relative	1.0	1.2	1.2	1.3	2.1

Computation Time

What about a Longer Run?

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

	Minutes	Relative
Gauss - Baseline	71.9	13.5
Gauss	5.3	1.0
Python-Compiled	5.3	1.0
Julia	6.2	1.2
Matlab	8.6	1.6
Python	10.3	1.9

Conclusion

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)