How TROLL Solves a Million Equations

Sparse-Matrix Techniques for Stacked-Time Solution of Perfect-Foresight Models

CEF 2008

Peter Hollinger
Intex Solutions, Inc.

Nonlinear Forward-Looking Model

$$g_{i,t}\,(y_{1,t+s},\dots\, y_{\mathrm{N},t+s},\, y_{1,t},\dots\, y_{\mathrm{N},t},\dots\, y_{1,t\text{-r}},\dots\, y_{\mathrm{N},t-r}) = 0$$

$\mathbf{g}_t\,(\mathbf{y}_{t\,+s},\dots,\, \mathbf{y}_t\,,\dots\, \mathbf{y}_{t-r}) = 0$ $\quad \mathbf{y}_t \equiv (\, y_{1,t}\,,\dots\, y_{\mathrm{N},t}\,)$ $\quad \mathbf{g}_t \equiv (\, g_{1,t}\,,\dots\, g_{\mathrm{N},t}\,)$

Note: can be transformed to 1-lag, 1-lead form:

$$\mathbf{g}_t\,(\mathbf{y}_{t\,+1},\, \mathbf{y}_t\,,\, \mathbf{y}_{t\text{-}1}) = 0$$

Stacked-Time Solution

Choose a long time-horizon, **T**, and stack equations:

$\mathbf{f}\,(\mathbf{z}) = 0$ $\qquad \mathbf{z} \equiv (\mathbf{y}_1,\dots\mathbf{y}_{\mathrm{T}})$ $\quad \mathbf{f} \equiv (\mathbf{g}_1,\dots\mathbf{g}_{\mathrm{T}})$

Given initial guess, $\mathbf{z}^{(0)}$, find $\mathbf{z}^*$ such that: $\mathbf{f}\,(\mathbf{z}^*) \approx 0$

Newton's Method

Solve $\mathbf{f}(\mathbf{z}) = 0$

Jacobian Matrix: $\mathbf{J} \equiv \{\partial f_i / \partial z_j\}$

Preprocessing: Incidence Matrix, Symbolic Derivatives

Newton step: $\Delta\mathbf{z}^{(k)} = -(\mathbf{J}|\mathbf{z}^{(k)})^{-1}\,\mathbf{f}(\mathbf{z}^{(k)})$

Newton iteration: $\mathbf{z}^{(k+1)} = \mathbf{z}^{(k)} - \mathbf{J}^{(k)-1}\,\mathbf{f}^{(k)}$

Damped Newton: $\mathbf{z}^{(k+1)} = \mathbf{z}^{(k)} - \alpha\,\mathbf{J}^{(k)-1}\,\mathbf{f}^{(k)}$

Jacobian Matrix Step: $\mathbf{J}^{-1}\,\mathbf{f}$

- Most expensive part of Newton's Method
- Often solved by LU factoring: **J = LU**

  cost = O(n^3) (standard "dense" LU)
- **J** can be large: n=NT for stacked-time

  e.g., 2000 eqns × 500 periods = 1,000,000 rows
- But **J** is very sparse:

  Most equations have only a few variables;
  the rest of the row is "hard" zero.

Main methods for solving sparse matrix

- Direct Sparse LU
  - reduce cost by skipping calculations with zeros
  - chose pivots to minimize fill-in but maintain accuracy
  - may be implemented in three stages:
    1) [Analyze and] **Factor** (slow)
    2) **Refactor** same pattern (fast, if pivots still OK)
    3) **Solve** (fast)
- Nonstationary Iterative Solver
  - iterations use matrix-vector product and "preconditioning"
  - preconditioner must give fast approximate solution
  - unsymmetric methods include FGMRES, CGS, BiCGstab

Sparse-LU Codes in TROLL

- MA30 (Duff & Reid, Harwell, circa 1980)
  - very fast Refactor stage
  - initial Factor step too slow for huge matrices
- UMFPACK 5.0 (Tim Davis, UFL, 2006)
  - fast Factor, but Refactor is no faster
  - modern code: takes advantage of BLAS
- PARDISO (Schenk & Gärtner, 2004)
  - bundled with Intel Math Kernel Library 9.1
  - fast Factor & fast Refactor
  - can be inaccurate on ill-conditioned system

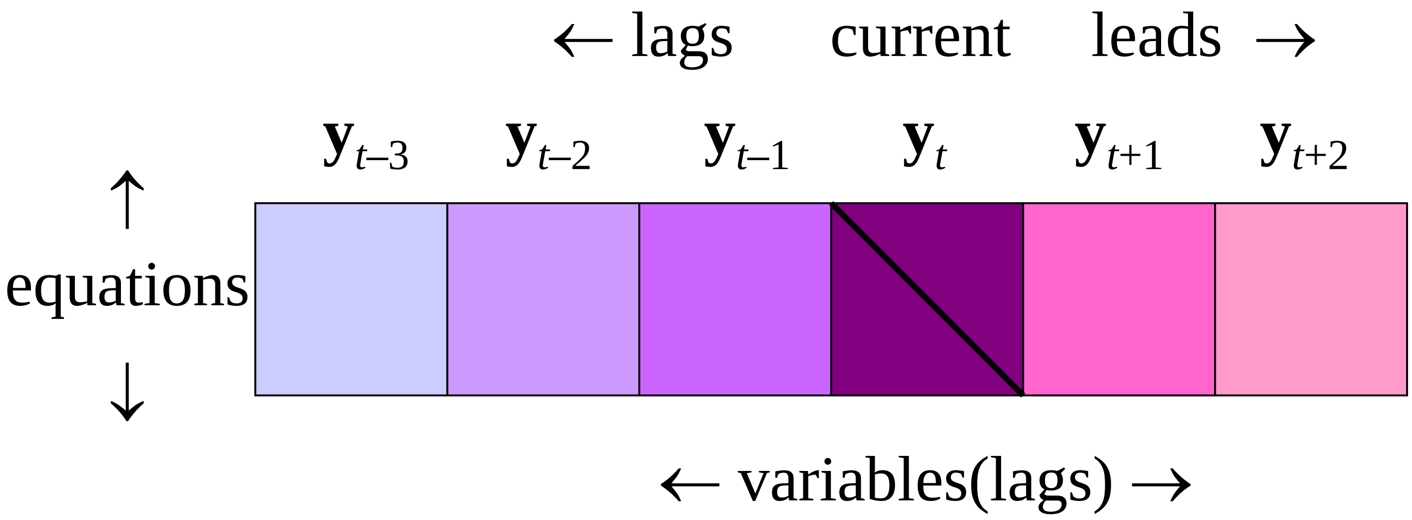
Sparse-LU Codes (cont.)

- “PARDCGS”: PARDISO with CGS feature
  - PARDISO for initial Factor and Solve
  - skip Refactor
  - solve by CGS, initial factoring as preconditioner
- “UMFITER”
  - UMFPACK for initial Factor and Solve
  - skip Refactor
  - solve: FGMRES, initial factoring as preconditioner

Both give extremely fast Refactor but slow Solve.

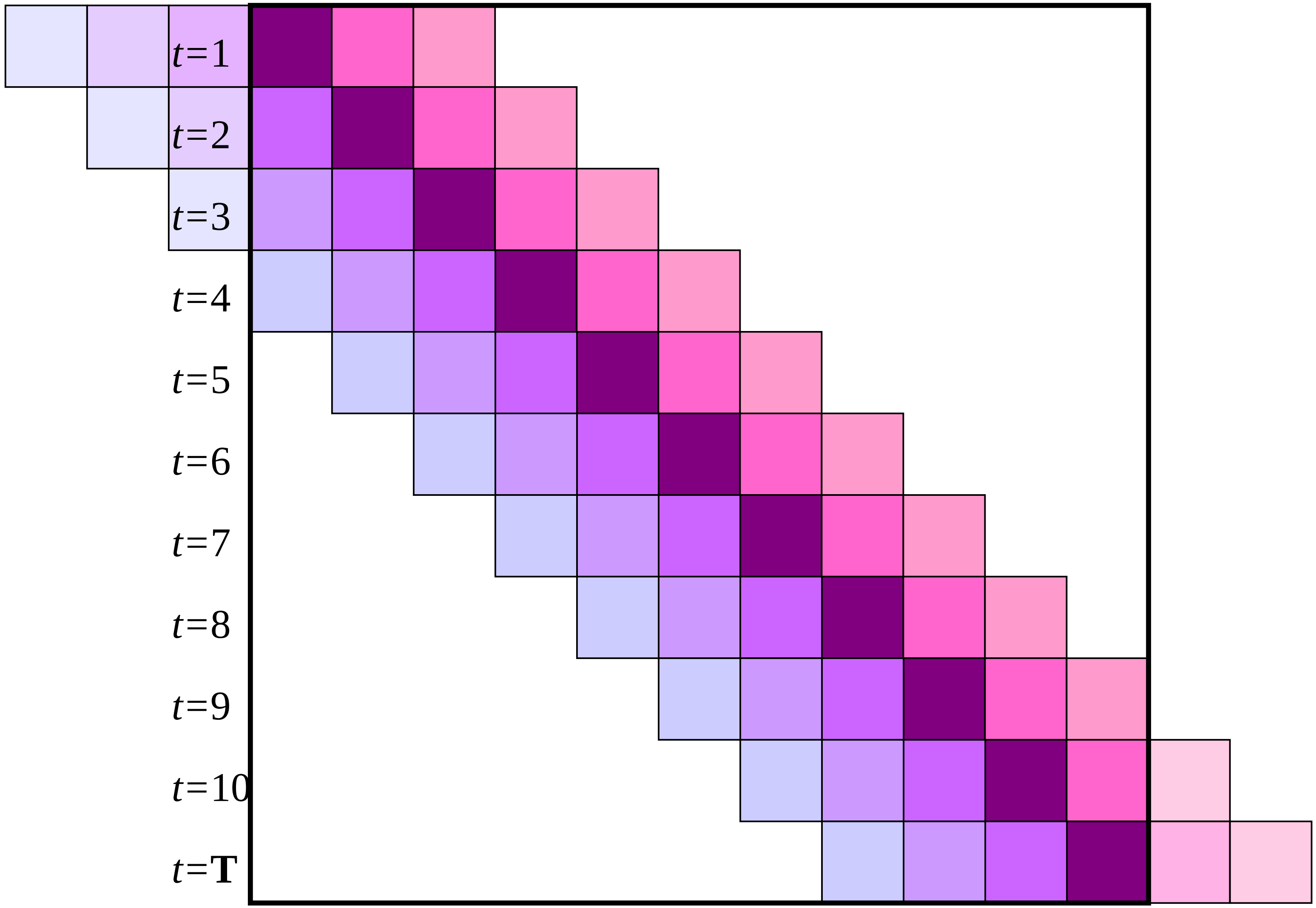
Symbolic Jacobian for Stacked-Time

Derivatives of N equations with respect to N variables at all leads and lags:



All squares are sparse, particularly the leads and lags.
"$\mathbf{y}_t$" square must be non-singular; diagonal can be made nonzero

Stacked Jacobian has a regular structure:



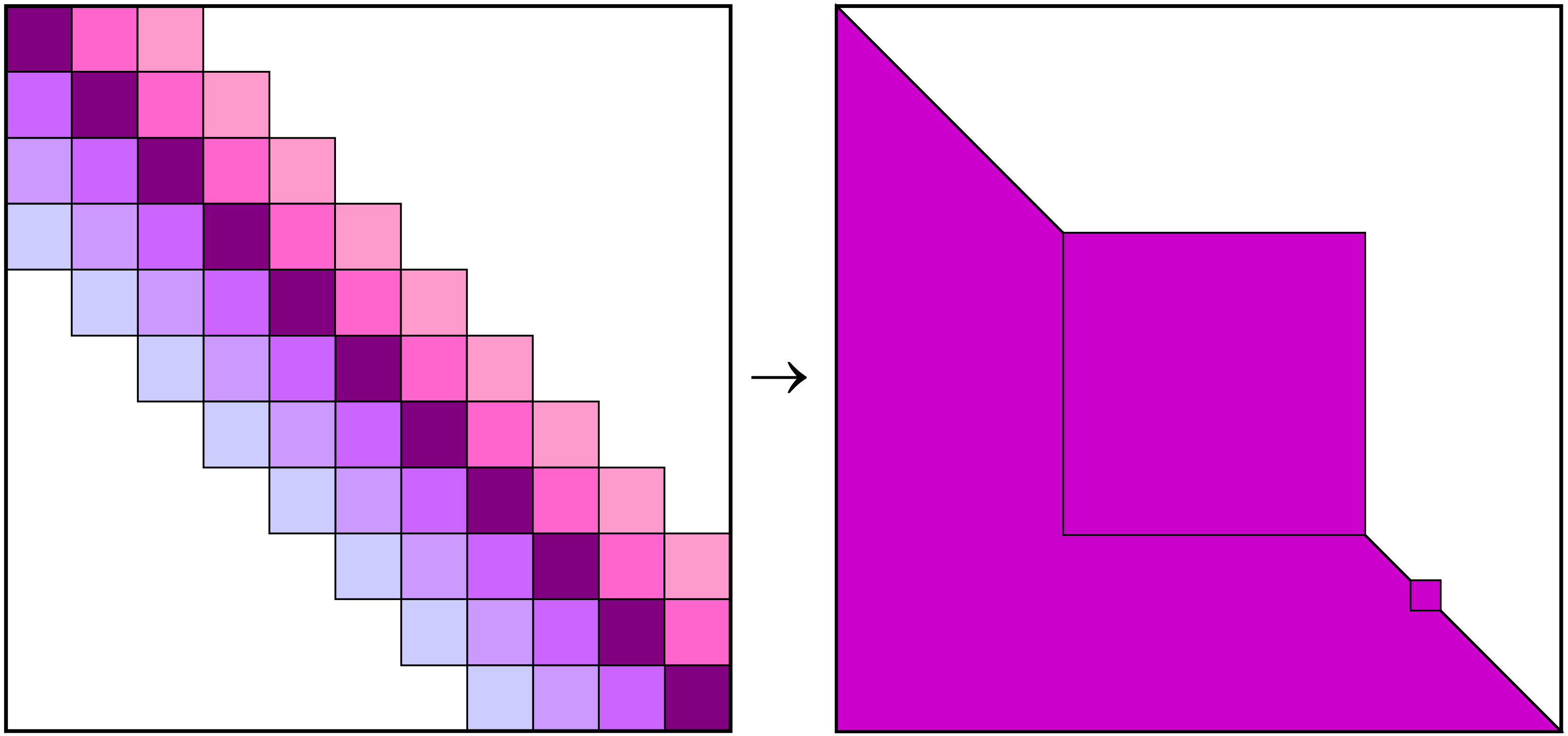
Each row of squares has the same pattern of nonzeros

Three ways to take advantage of sparsity:

1) **Full-Stack LU** (FSLU, aka “OLDSTACK”)
   - permute entire stack to block-triangular form
   - solve minimal simultaneous blocks with sparse-LU
   - one Solve per [Re]Factor

Full-Stack LU

Permute stack to minimal simultaneous blocks:



Typically one huge block

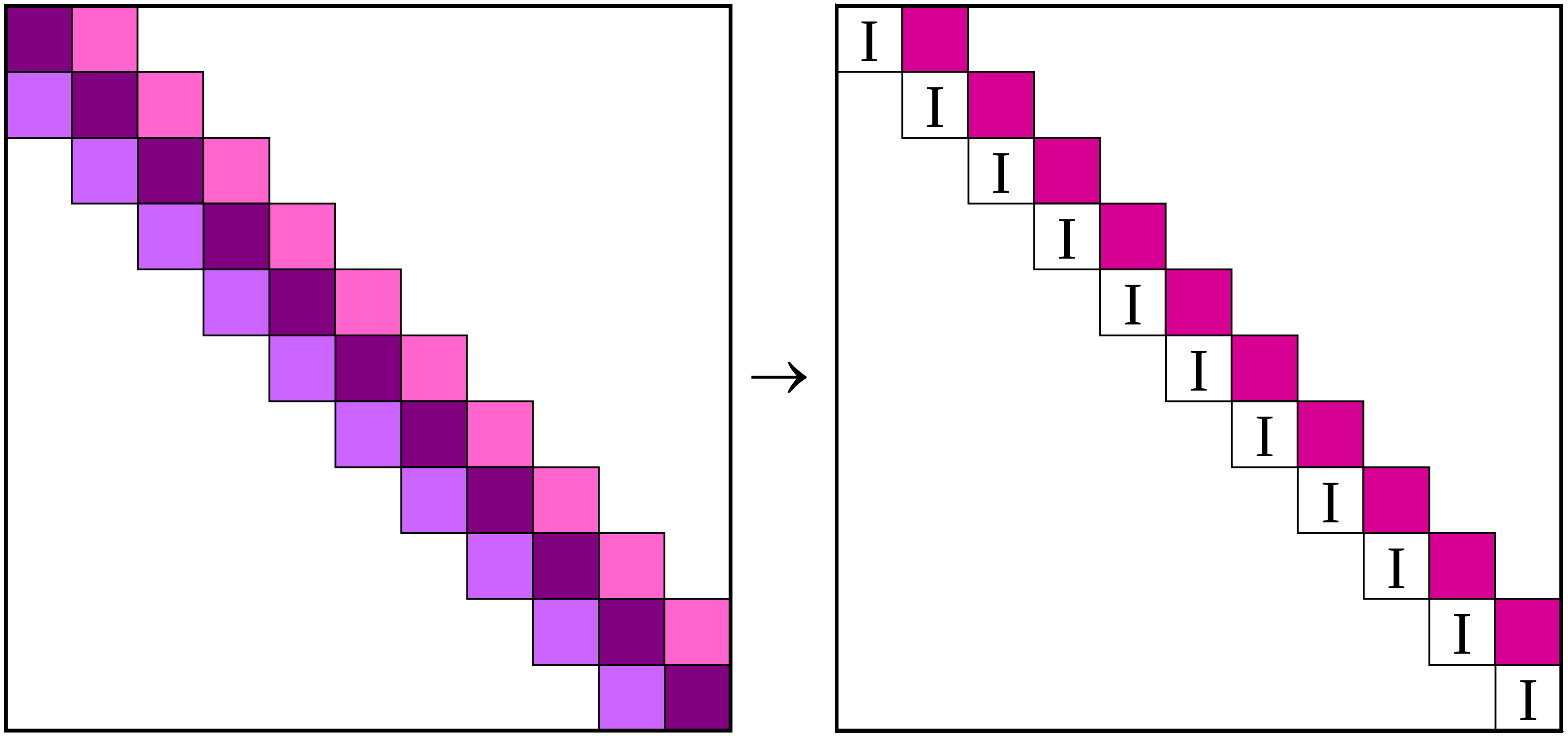
Three ways to take advantage of sparsity:

2) **Block-Band LU** (BBLU, aka “NEWSTACK”)

- triangularize period-by-period using sparse-LU
- blocks fill in (~10%); pattern becomes fixed
- numerical stability requires block-diagonal-dominance
- many Solve steps per [Re]Factor

Block-Band LU

Triangularize stack period-by-period:



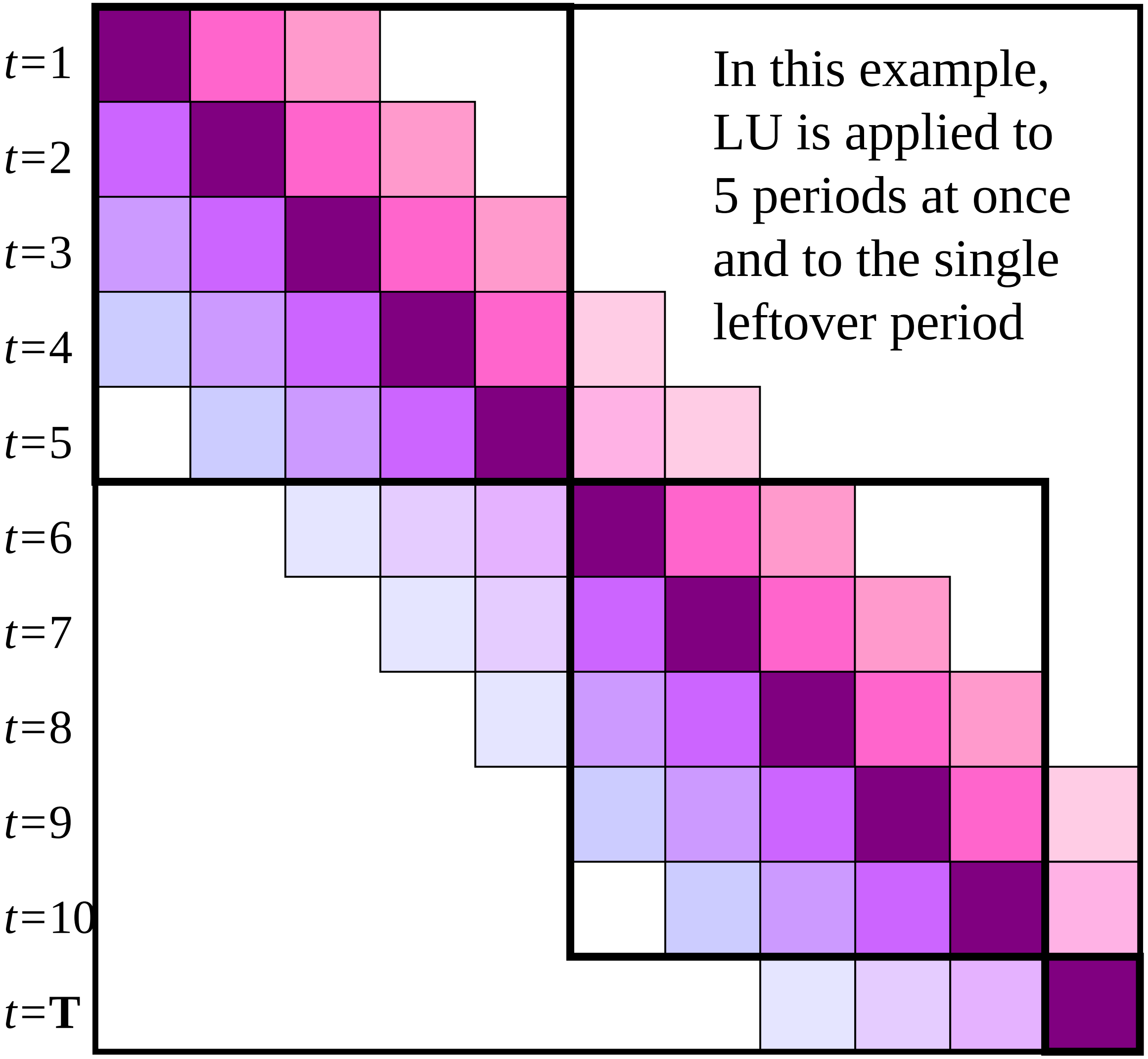
(while transforming RHS)
followed by block-back-solve

Three ways to take advantage of sparsity:

3) **IterStack**
  - nonstationary iterative method on entire stack (FGMRES)
  - sparse-LU on several periods at once as preconditioner
  - FGMRES may fail to converge: inaccurate Newton step
  - very many Solve steps per [Re]Factor

IterStack

FGMRES preconditioned by LU on several periods



In this example,
LU is applied to
5 periods at once
and to the single
leftover period

Sensible combinations

- FSLU needs fast [Re]Factor for huge matrix
  - UMFITER (faster than UMFPACK)
  - PARDCGS (faster than PARDISO)
- BBLU needs fast Solve; “small” matrices
  - MA30
- IterStack needs fast Solve; matrices not huge
  - MA30

Three Test Models

Variants of:

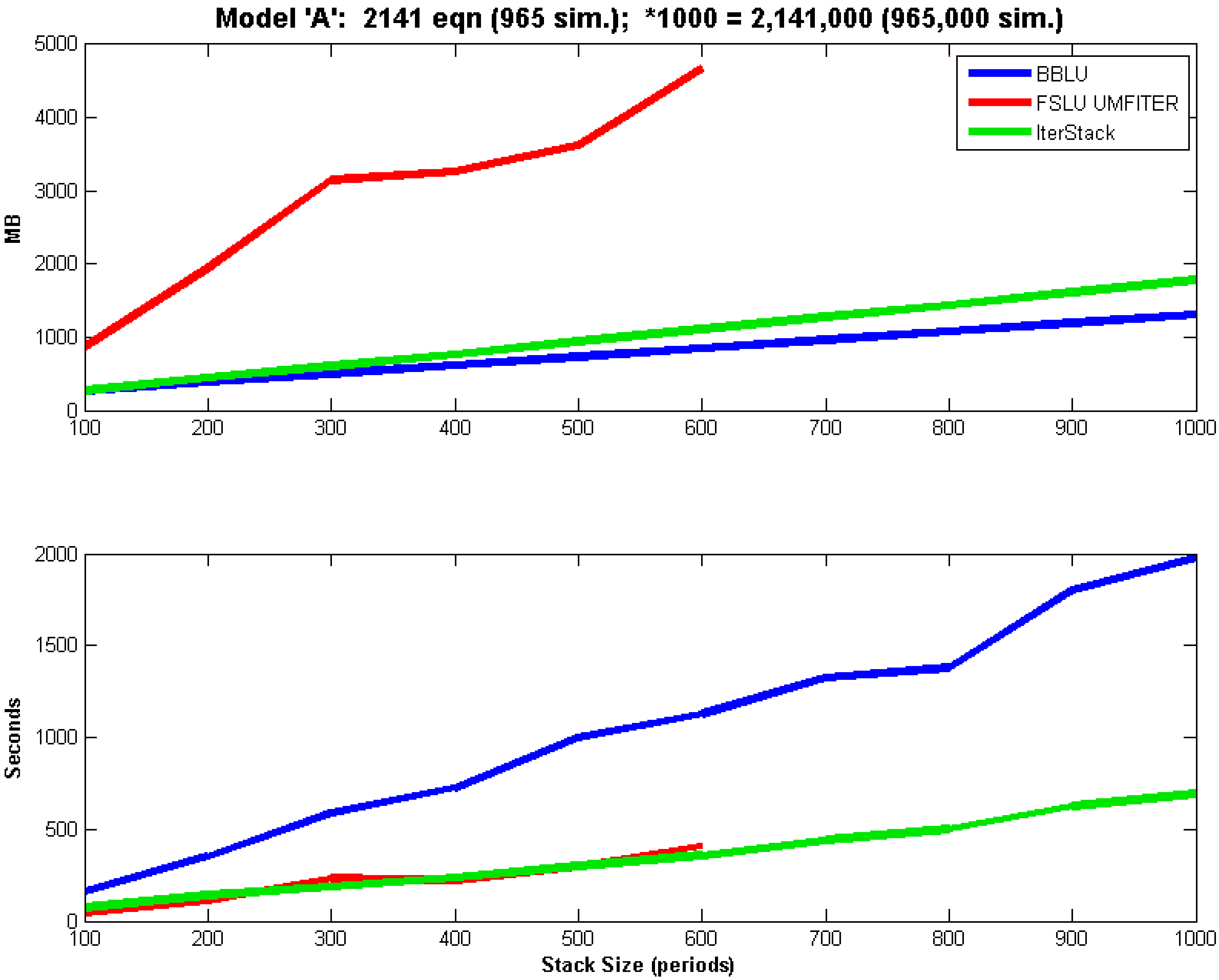
- GIMF from the IMF
- QUEST from the CEC
- QPM from the Bank of Canada

Dubbed ‘A’, ‘B’ and ‘C’ (no particular order)

	Equations	Simultaneous	Max Lag	Max Lead
A	2141	965	-2	2
B	1736	1252	-3	3
C	1037	870	-7	1

Model ‘A’ Summary

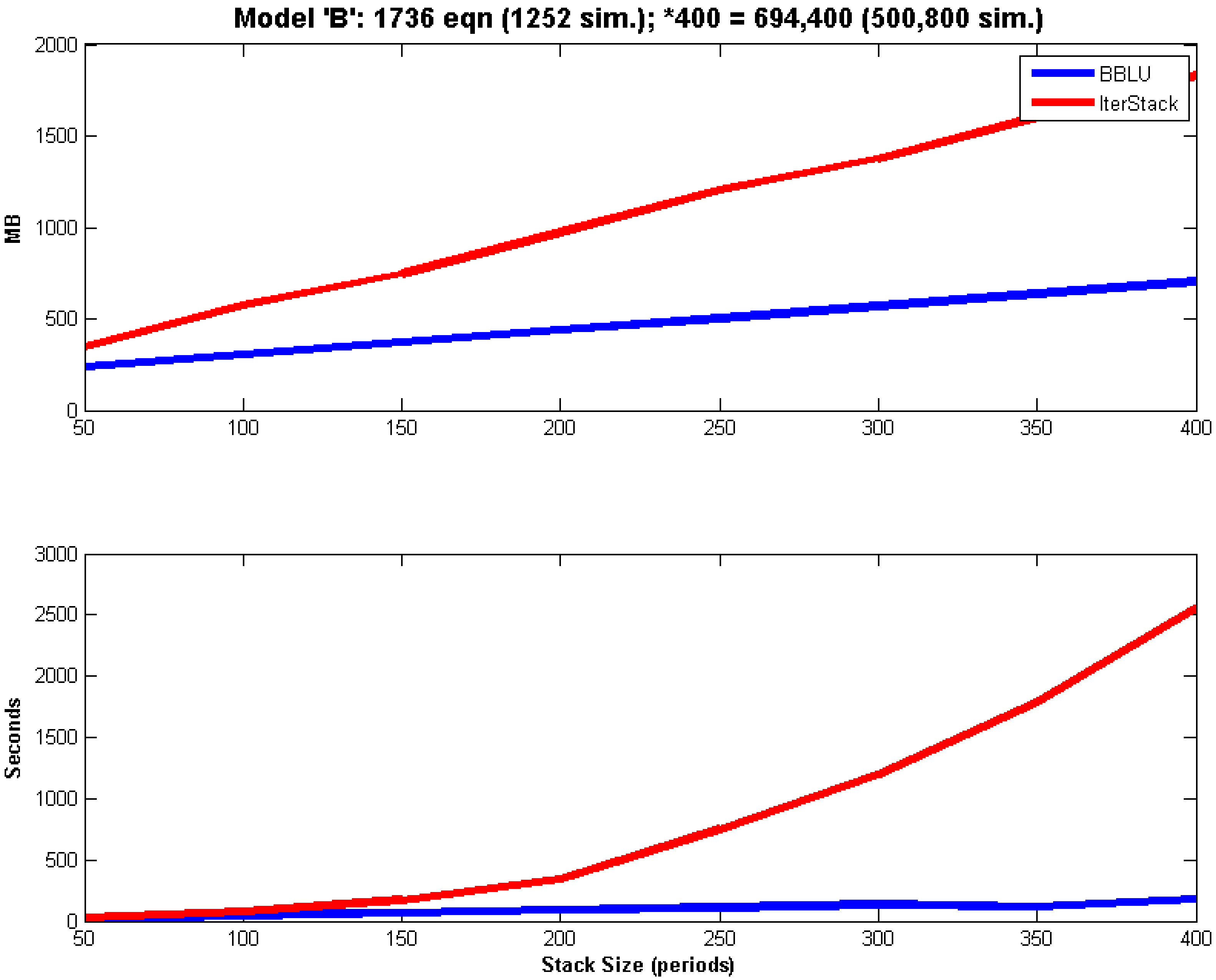
- PARDISO & PARDCGS failed
  - too ill-conditioned?
- UMFITER fast but ran out of memory
  - ~2GB by 200 periods
- IterStack worked very well
  - 50-100 FGMRES iterations (per Newton iter.)
  - about as fast as UMFITER, much less memory
- BBLU worked OK
  - less memory than IterStack, but a lot slower



Model 'A': 2141 eqn (965 sim.); *1000 = 2,141,000 (965,000 sim.)

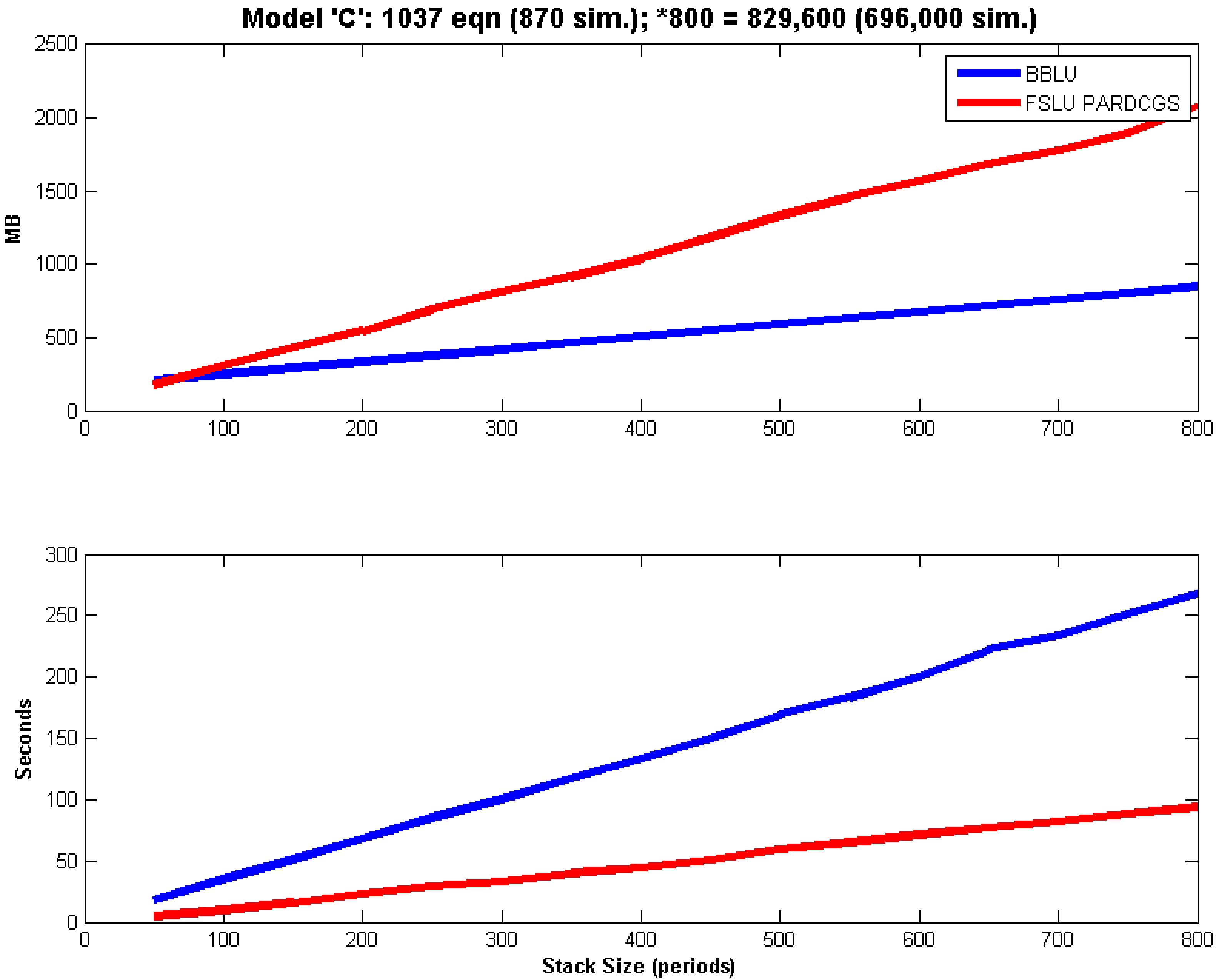
Model ‘B’ Summary

- PARDISO & PARDCGS failed
  - too ill-conditioned?
- UMFITER needed too much memory
  - >2GB by 200 periods
- BBLU worked very well
  - fast, moderate memory
  - converged in 4 Newton iterations
- IterStack needed too many FGMRES iters
  - ~500 FGMRES iterations, extra Newton iters (6)
  - needed extra restart space so extra memory


Model 'B': 1736 eqn (1252 sim.); *400 = 694,400 (500,800 sim.)
MB
2000
1500
1000
500
0
50
100
150
200
250
300
350
400
BBLU
IterStack
Seconds
3000
2500
2000
1500
1000
500
0
50
100
150
200
250
300
350
400
Stack Size (periods)

Model ‘C’ Summary

- UMFITER needed too much memory
  - >2GB by 200 periods
- IterStack failed
  - >500 FGMRES iterations, too inaccurate
- BBLU worked OK
  - moderate speed, modest memory
- PARDCGS worked very well
  - extremely fast
  - converged in 5 Newton iterations (same as BBLU)


Model 'C': 1037 eqn (870 sim.); *800 = 829,600 (696,000 sim.)

Conclusions

- No method is “best” for all models
  - need to experiment for each model

- IterStack somewhat disappointing
  - works well for some models, not others

- Modern sparse-LU codes make “OLDSTACK” competitive again.
  - but may need a lot of memory

Future Work

- IterStack improvements?
  - try other solvers (CGS, BiCGstab, QMR)
    - less memory than FGMRES
  - block-triangular preconditioner?
- Other Sparse-LU codes?
  - HSL MA50 (successor to MA30)
  - MUMPS
  - WSMP
- Use dynamics to shorten time horizon
  - linearized around the steady-state
  - will still need to be verified against full stack

The End

Presented at the 14th Annual Conference on
Computing in Economics and Finance
June 26-28, 2008
University of Sorbonne
Paris