PETSc Tutorial

PETSc Team
Presented by Matthew Knepley

Mathematics and Computer Science Division
Argonne National Laboratory

AMCS E4302
Columbia University
October 11, 2006

Disclaimer

This PETSc Tutorial debuts Tonight!

Tutorial Goal

Enable students to develop new simulations with PETSc.

- Serial and Parallel

Tutorial Goal

Enable students to develop new simulations with PETSc.

- Serial and Parallel
- Linear and Nonlinear

Tutorial Goal

Enable students to develop new simulations with PETSc.

- Serial and Parallel
- Linear and Nonlinear
- Finite Difference and Finite Element

Tutorial Goal

Enable students to develop new simulations with PETSc.

- Serial and Parallel
- Linear and Nonlinear
- Finite Difference and Finite Element
- Structured and Unstructured

Tutorial Goal

Enable students to develop new simulations with PETSc.

- Serial and Parallel
- Linear and Nonlinear
- Finite Difference and Finite Element
- Structured and Unstructured
- Triangles and Hexes

Tutorial Goal

Enable students to develop new simulations with PETSc.

- Serial and Parallel
- Linear and Nonlinear
- Finite Difference and Finite Element
- Structured and Unstructured
- Triangles and Hexes
- Optimal Solvers

Tutorial Goal

Enable students to develop new simulations with PETSc.

- Serial and Parallel
- Linear and Nonlinear
- Finite Difference and Finite Element
- Structured and Unstructured
- Triangles and Hexes
- Optimal Solvers

Items in red not finished for tutorial

Outline

Part I

Creating a PETSc Application

How did PETSc Originate?

PETSc was developed as a Platform for Experimentation

We want to experiment with different

- Models
- Discretizations
- Solvers
- Algorithms (which blur these boundaries)

What I Need From You

- Tell me if you do not understand
- Tell me if an example does not work
- Suggest better wording or figures
- Followup problems at petsc-maint@mcs.anl.gov

How Can We Help?

- Provide documentation

- Answer email at petsc-maint@mcs.anl.gov

How Can We Help?

- Provide documentation
- Quickly answer questions

- Answer email at petsc-maint@mcs.anl.gov

How Can We Help?

- Provide documentation
- Quickly answer questions
- Help install

- Answer email at petsc-maint@mcs.anl.gov

How Can We Help?

- Provide documentation
- Quickly answer questions
- Help install
- Guide large scale flexible code development
- Answer email at petsc-maint@mcs.anl.gov

The Role of PETSc

Developing parallel, nontrivial PDE solvers that deliver high performance is still difficult and requires months (or even years) of concentrated effort.

PETSc is a toolkit that can ease these difficulties and reduce the development time, but it is not a black-box PDE solver, nor a silver bullet.

What is PETSc?

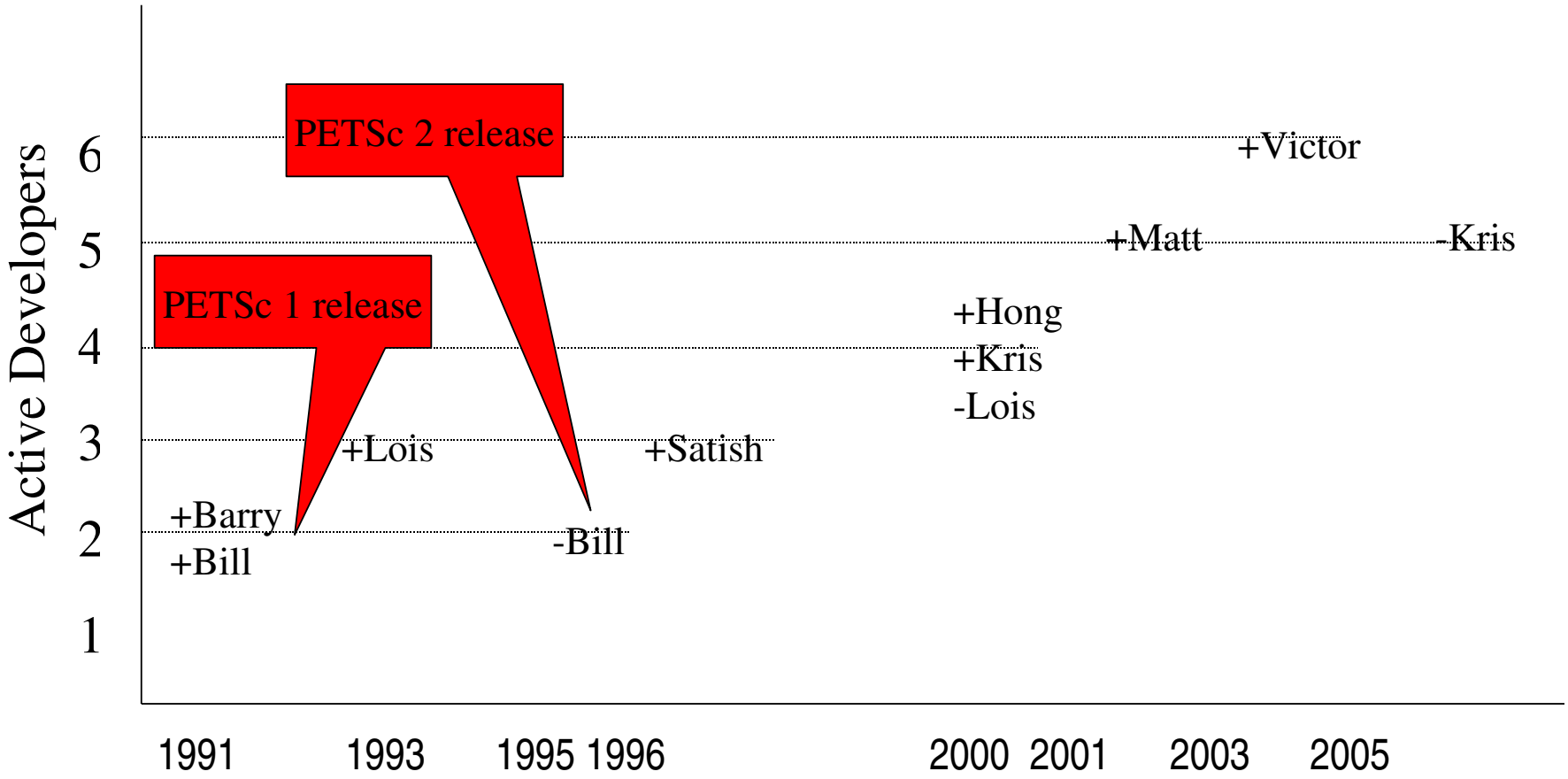
A freely available and supported research code

- Download from http://www.mcs.anl.gov/petsc
- Free for everyone, including industrial users
- Hyperlinked manual, examples, and manual pages for all routines
- Hundreds of tutorial-style examples
- Support via email: petsc-maint@mcs.anl.gov
- Usable from C, C++, Fortran 77/90, and Python

What is PETSc?

- Portable to any parallel system supporting MPI, including:
  - Tightly coupled systems
    - Cray T3E, SGI Origin, IBM SP, HP 9000, Sub Enterprise
  - Loosely coupled systems, such as networks of workstations
    - Compaq,HP, IBM, SGI, Sun, PCs running Linux or Windows
- PETSc History
  - Begun September 1991
  - Over 8,500 downloads since 1995 (version 2), currently 250 per month
- PETSc Funding and Support
  - Department of Energy
    - SciDAC, MICS Program
  - National Science Foundation
    - CIG, CISE, Multidisciplinary Challenge Program

Timeline


Active Developers
6
5
4
3
2
1
PETSc 2 release
PETSc 1 release
+Victor
+Matt
-Kris
+Hong
+Kris
-Lois
+Lois
+Satish
+Barry
+Bill
-Bill
1991
1993
1995
1996
2000
2001
2003
2005

What Can We Handle?

- PETSc has run problems with over 500 million unknowns
  - http://www.scconference.org/sc2004/schedule/pdfs/pap111.pdf

What Can We Handle?

- PETSc has run problems with over 500 million unknowns
  - http://www.scconference.org/sc2004/schedule/pdfs/pap111.pdf
- PETSc has run on over 6,000 processors efficiently
  - ftp://info.mcs.anl.gov/pub/tech_reports/reports/P776.ps.Z

What Can We Handle?

- PETSc has run problems with over 500 million unknowns
  - http://www.scconference.org/sc2004/schedule/pdfs/pap111.pdf
- PETSc has run on over 6,000 processors efficiently
  - ftp://info.mcs.anl.gov/pub/tech_reports/reports/P776.ps.Z
- PETSc applications have run at 2 Teraflops
  - LANL PFLOTRAN code

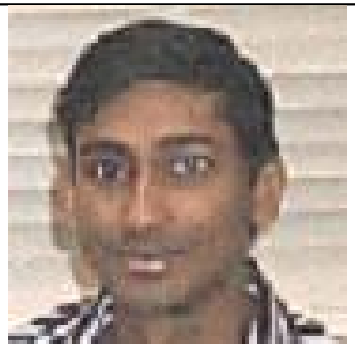
Who Uses PETSc?

- Computational Scientists
  - PyLith (TECTON), Underworld, Columbia group
- Algorithm Developers
  - Iterative methods and Preconditioning researchers
- Package Developers
  - SLEPc, TAO, MagPar, StGermain, DealII

The PETSc Team

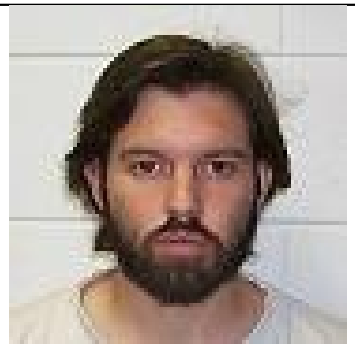

Bill Gropp


Barry Smith


Satish Balay


Dinesh Kaushik


Kris Buschelman


Matt Knepley


Hong Zhang


Victor Eijkhout


Lois McInnes

Downloading PETSc

- The latest tarball is on the PETSc site
  - ftp://ftp.mcs.anl.gov/pub/petsc/petsc.tar.gz
  - We no longer distribute patches (everything is in the distribution)
- There is a Debian package
- There is a FreeBSD Port
- There is a Mercurial development repository

Cloning PETSc

- The full development repository is open to the public
  - http://mercurial.mcs.anl.gov/petsc/petsc-dev
  - http://mercurial.mcs.anl.gov/petsc/BuildSystem
- Why is this better?
  - You can clone to any release (or any specific ChangeSet)
  - You can easily rollback changes (or releases)
  - You can get fixes from us the same day

Unpacking PETSc

- Just clone development repository
  - `hg clone http://mercurial.mcs.anl.gov/petsc/petsc-dev petsc-dev`
  - `hg clone -rRelease-2.3.1 petsc-dev petsc-2.3.1`

or

- Unpack the tarball
  - `tar xzf petsc.tar.gz`

Getting the Source

You will need the Developer copy of PETSc:

- Using Mercurial

```
hg clone http://mercurial.mcs.anl.gov/petsc/petsc-dev
cd petsc-dev/python
hg clone http://mercurial.mcs.anl.gov/petsc/BuildSystem
```

- Manual download

```
wget ftp://info.mcs.anl.gov/pub/petsc/petsc-dev.tar.gz .
```

and the tutorial source code:

- Using Mercurial

```
hg clone http://mercurial.mcs.anl.gov/petsc/Columbia06TutorialCode
```

- Manual download

```
wget ftp://info.mcs.anl.gov/pub/petsc/Columbia06TutorialCode.tar.gz .
```

Configuring PETSc

- Set `$PETSC_DIR` to the installation root directory
- Run the configuration utility
  - `$PETSC_DIR/config/configure.py`
  - `$PETSC_DIR/config/configure.py --help`
  - `$PETSC_DIR/config/configure.py --download-mpich`
- There are many examples on the installation page
- Configuration files are placed in `$PETSC_DIR/bmake/$PETSC_ARCH`
  - `$PETSC_ARCH` has a default if not specified

Configuring PETSc

- You can easily reconfigure with the same options
  - `./bmake/$PETSC_ARCH/configure.py`
- Can maintain several different configurations
  - `./config/configure.py -PETSC_ARCH=linux-fast --with-debugging=0`
- All configuration information is in `configure.log`
  - ALWAYS send this file with bug reports

Automatic Downloads

- Starting in 2.2.1, some packages are automatically
  - Downloaded
  - Configured and Built (in `$PETSC_DIR/externalpackages`)
  - Installed in PETSc
- Currently works for
  - PETSc documentation utilities (Sowing, lgrind, c2html)
  - BLAS, LAPACK, BLACS, ScaLAPACK, PLAPACK
  - MPICH, MPE, LAM
  - ParMetis, Chaco, Jostle, Party, Scotch
  - MUMPS, Spooles, SuperLU, SuperLU_Dist, UMFPack
  - Prometheus, HYPRE, ML, SPAI
  - Sundials
  - Triangle, TetGen
  - FIAT, FFC

Building PETSc

- Uses recursive make starting in `cd $PETSC_DIR`
  - `make`
  - Check build when done with `make test`
- Complete log for each build in `make_log_$PETSC_ARCH`
  - ALWAYS send this with bug reports
- Can build multiple configurations
  - `PETSC_ARCH=linux-fast make`
  - Libraries are in `$PETSC_DIR/lib/$PETSC_ARCH/`
- Can also build a subtree
  - `cd src/snes; make`
  - `cd src/snes; make ACTION=libfast tree`

Running PETSc

- Try running PETSc examples first
  - `cd $PETSC_DIR/src/snes/examples/tutorials`
- Build examples using make targets
  - `make ex5`
- Run examples using the make target
  - `make runex5`
- Can also run using MPI directly
  - `mpirun ./ex5 -snes_max_it 5`
  - `mpiexec ./ex5 -snes_monitor`

Using MPI

- The Message Passing Interface is:
  - a library for parallel communication
  - a system for launching parallel jobs (mpirun/mpiexec)
  - a community standard
- Launching jobs is easy
  - `mpiexec -np 4 ./ex5`
- You should never have to make MPI calls when using PETSc
  - Almost never

MPI Concepts

- Communicator
  - A context (or scope) for parallel communication ("Who can I talk to")
  - There are two defaults:
    - yourself (PETSC_COMM_SELF),
    - and everyone launched (PETSC_COMM_WORLD)
  - Can create new communicators by splitting existing ones
  - Every PETSc object has a communicator
- Point-to-point communication
  - Happens between two processes (like in `MatMult()`)
- Reduction or scan operations
  - Happens among all processes (like in `VecDot()`)

Alternative Memory Models

- Single process (address space) model
  - OpenMP and threads in general
  - Fortran 90/95 and compiler-discovered parallelism
  - System manages memory and (usually) thread scheduling
  - Named variables refer to the same storage
- Single name space model
  - HPF, UPC
  - Global Arrays
  - Titanium
  - Named variables refer to the coherent values (distribution is automatic)
- Distributed memory (shared nothing)
  - Message passing
  - Names variables in different processes are unrelated

Common Viewing Options

- Gives a text representation
  - `-vec_view`
- Generally views subobjects too
  - `-snes_view`
- Can visualize some objects
  - `-mat_view_draw`
- Alternative formats
  - `-vec_view_binary`, `-vec_view_matlab`, `-vec_view_socket`
- Sometimes provides extra information
  - `-mat_view_info`, `-mat_view_info_detailed`

Common Monitoring Options

- Display the residual
  - `-ksp_monitor`, graphically `-ksp_xmonitor`
- Can disable dynamically
  - `-ksp_cancelmonitors`
- Does not display subsolvers
  - `-snes_monitor`
- Can use the true residual
  - `-ksp_truemonitor`
- Can display different subobjects
  - `-snes_vecmonitor`, `-snes_vecmonitor_update`, `-snes_vecmonitor_residual`
  - `-ksp_gmres_krylov_monitor`
- Can display the spectrum
  - `-ksp_singmonitor`

Getting More Help

- http://www.mcs.anl.gov/petsc
- Hyperlinked documentation
  - Manual
  - Manual pages for evey method
  - HTML of all example code (linked to manual pages)
- FAQ
- Full support at petsc-maint@mcs.anl.gov
- High profile users
  - David Keyes
  - Rich Martineau
  - Richard Katz
  - Charles Williams

Following the Tutorial

Update to each new checkpoint (r0):

- `hg clone -r0 Columbia06TutorialCode code-test`

or

- `hg update 0`

Build the executable with `make`, and then run:

- `make runbratu`
- `make debugbratu`
- `make valbratu`
- `make NP=2 runbratu`
- `make EXTRA_ARGS="-pc_type jacobi" runbratu`

Code Update

Update to Revision 0

Initialization

- Call `PetscInitialize()`
  - Setup static data and services
  - Setup MPI if it is not already
- Call `PetscFinalize()`
  - Calculates logging summary
  - Shutdown and release resources
- Checks compile and link

Command Line Processing

- Check for an option
  - `PetscOptionsHasName()`
- Retrieve a value
  - `PetscOptionsGetInt()`, `PetscOptionsGetIntArray()`
- Set a value
  - `PetscOptionsSetValue()`
- Clear, alias, reject, etc.

Profiling

- Use `-log_summary` for a performance profile
  - Event timing
  - Event flops
  - Memory usage
  - MPI messages
- Call `PetscLogStagePush()` and `PetscLogStagePop()`
  - User can add new stages
- Call `PetscLogEventBegin()` and `PetscLogEventEnd()`
  - User can add new events

Part II

Creating a Simple 2D Mesh

Higher Level Abstractions

The PETSc `DA` class is a topology and dicretization interface.

- Structured grid interface
  - Fixed simple topology
- Supports stencils, communication, reordering
  - Limited idea of operators
- Nice for simple finite differences

The PETSc `Mesh` class is a topology interface.

- Unstructured grid interface
  - Arbitrary topology and element shape
- Supports partitioning, distribution, and global orders

Higher Level Abstractions

The PETSc `DM` class is a hierarchy interface.

- Supports multigrid
  - `DMMG` combines it with the `MG` preconditioner
- Abstracts the logic of multilevel methods

The PETSc `Section` class is a function interface.

- Functions over unstructured grids
  - Arbitrary layout of degrees of freedom
- Support distribution and assembly

Code Update

Update to Revision 2

Creating a DA

```
DACreate2d(comm, wrap, type, M, N, m, n, dof, s, lm[],
ln[], DA *da)
```

`wrap`: Specifies periodicity
- `DA_NONPERIODIC`, `DA_XPERIODIC`, `DA_YPERIODIC`, or `DA_XYPERIODIC`

`type`: Specifies stencil
- `DA_STENCIL_BOX` or `DA_STENCIL_STAR`

`M/N`: Number of grid points in x/y-direction

`m/n`: Number of processes in x/y-direction

`dof`: Degrees of freedom per node

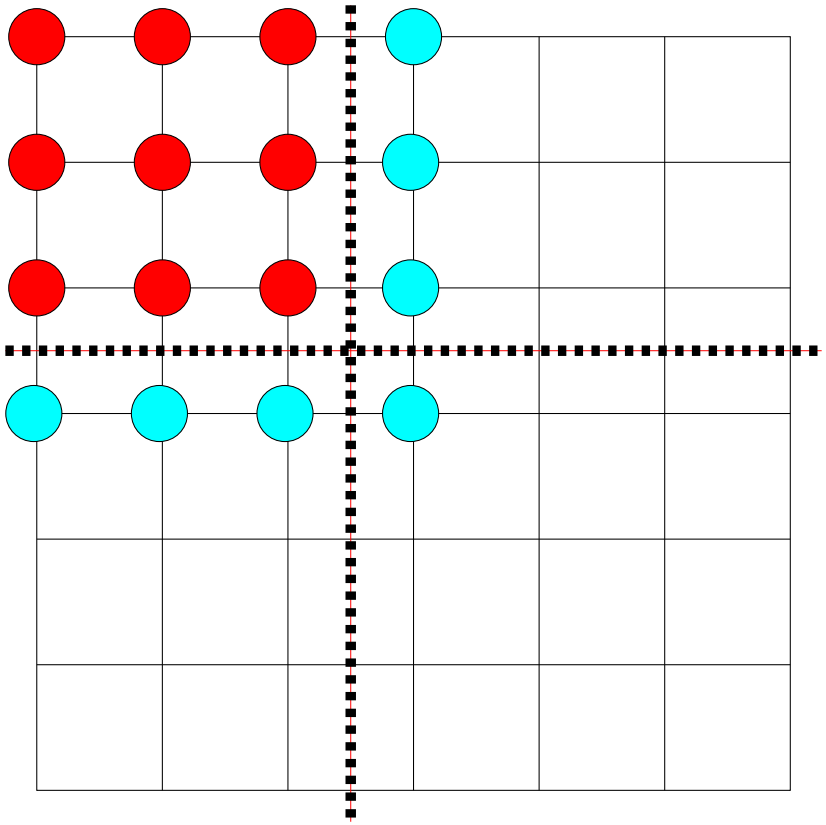
`s`: The stencil width

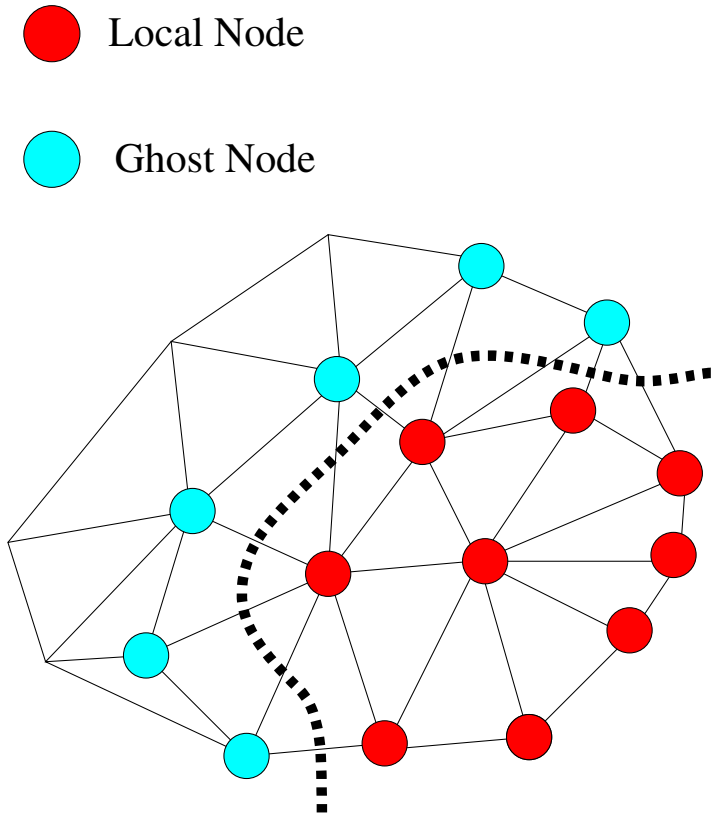
`lm/n`: Alternative array of local sizes
- Use `PETSC_NULL` for the default

Ghost Values

To evaluate a local function $f(x)$, each process requires

- its local portion of the vector x
- its ghost values, bordering portions of x owned by neighboring processes





DA Global Numberings

Proc 2			Proc 3	
25	26	27	28	29
20	21	22	23	24
15	16	17	18	19
10	11	12	13	14
5	6	7	8	9
0	1	2	3	4
Proc 0			Proc 1	

Natural numbering

Proc 2			Proc 3	
21	22	23	28	29
18	19	20	26	27
15	16	17	24	25
6	7	8	13	14
3	4	5	11	12
0	1	2	9	10
Proc 0			Proc 1	

PETSc numbering

DA Global vs. Local Numbering

- **Global**: Each vertex belongs to a unique process and has a unique id
- **Local**: Numbering includes ghost vertices from neighboring processes

Proc 2			Proc 3	
X	X	X	X	X
X	X	X	X	X
12	13	14	15	X
8	9	10	11	X
4	5	6	7	X
0	1	2	3	X
Proc 0			Proc 1	

Local numbering

Proc 2			Proc 3	
21	22	23	28	29
18	19	20	26	27
15	16	17	24	25
6	7	8	13	14
3	4	5	11	12
0	1	2	9	10
Proc 0			Proc 1	

Global numbering

Viewing the DA

- `make NP=1 EXTRA_ARGS="-da_view_draw -draw_pause -1" runbratu`
- `make NP=1 EXTRA_ARGS="-da_grid_x 10 -da_grid_y 10 -da_view_draw -draw_pause -1" runbratu`
- `make NP=4 EXTRA_ARGS="-da_grid_x 10 -da_grid_y 10 -da_view_draw -draw_pause -1" runbratu`

Correctness Debugging

- Automatic generation of tracebacks
- Detecting memory corruption and leaks
- Optional user-defined error handlers

Interacting with the Debugger

- Launch the debugger
  - `-start_in_debugger [gdb,dbx,noxterm]`
  - `-on_error_attach_debugger [gdb,dbx,noxterm]`
- Attach the debugger only to some parallel processes
  - `-debugger_nodes 0,1`
- Set the display (often necessary on a cluster)
  - `-display khan.mcs.anl.gov:0.0`

Debugging Tips

- Putting a breakpoint in `PetscError()` can catch errors as they occur
- PETSc tracks memory overwrites at the beginning and end of arrays
  - The `CHKMEMQ` macro causes a check of all allocated memory
  - Track memory overwrites by bracketing them with `CHKMEMQ`
- PETSc checks for leaked memory
  - Use `PetscMalloc()` and `PetscFree()` for all allocation
  - Option `-trmalloc` will print unfreed memory on `PetscFinalize()`

Memory Debugging

We can check for unfreed memory using:

```
make EXTRA_ARGS="-malloc_dump" runbratu
```

All options can be seen using:

```
make EXTRA_ARGS="-help" runbratu
```

Code Update

Update to Revision 3

Creating the Mesh

- File input
  - `MeshCreatePCICE()`
  - `MeshCreatePyLith()`
- Generation
  - `ALE::Generator::generateMesh()`
  - `ALE::Generator::refineMesh()`
- Partitioning and Distribution
  - `MeshDistribute()`

Code Update

Update to Revision 4

Viewing the Mesh

- `make NP=1 EXTRA_ARGS="-structured 0 -mesh_view_vtk" runbratu`
- `mayavi -d bratu.vtk -m SurfaceMap&`
- `make NP=4 EXTRA_ARGS="-structured 0 -mesh_view_vtk" runbratu`
- Viewable using Mayavi

Part III

Defining a Function

A DA is more than a Mesh

A DA contains topology, geometry, and an implicit Q1 discretization.

It is used as a template to create

- Vectors (functions)
- Matrices (linear operators)

DA Vectors

- The DA object contains only layout (topology) information
  - All field data is contained in PETSc `Vecs`
- Global vectors are parallel
  - Each process stores a unique local portion
  - `DACreateGlobalVector(DA da, Vec *gvec)`
- Local vectors are sequential (and usually temporary)
  - Each process stores its local portion plus ghost values
  - `DACreateLocalVector(DA da, Vec *lvec)`
  - includes ghost values!

Updating Ghosts

Two-step process enables overlapping computation and communication

- `DAGlobalToLocalBegin(da, gvec, mode, lvec)`
  - `gvec` provides the data
  - `mode` is either `INSERT_VALUES` or `ADD_VALUES`
  - `lvec` holds the local and ghost values
- `DAGlobalToLocal End(da, gvec, mode, lvec)`
  - Finishes the communication

The process can be reversed with `DALocalToGlobal()`.

DA Local Function

The user provided function which calculates the nonlinear residual in 2D has signature

```
PetscErrorCode (*lfunc)(DALocalInfo *info, PetscScalar **x,
                        PetscScalar **r, void *ctx)
```

`info`: All layout and numbering information

`x`: The current solution

- Notice that it is a multidimensional array

`r`: The residual

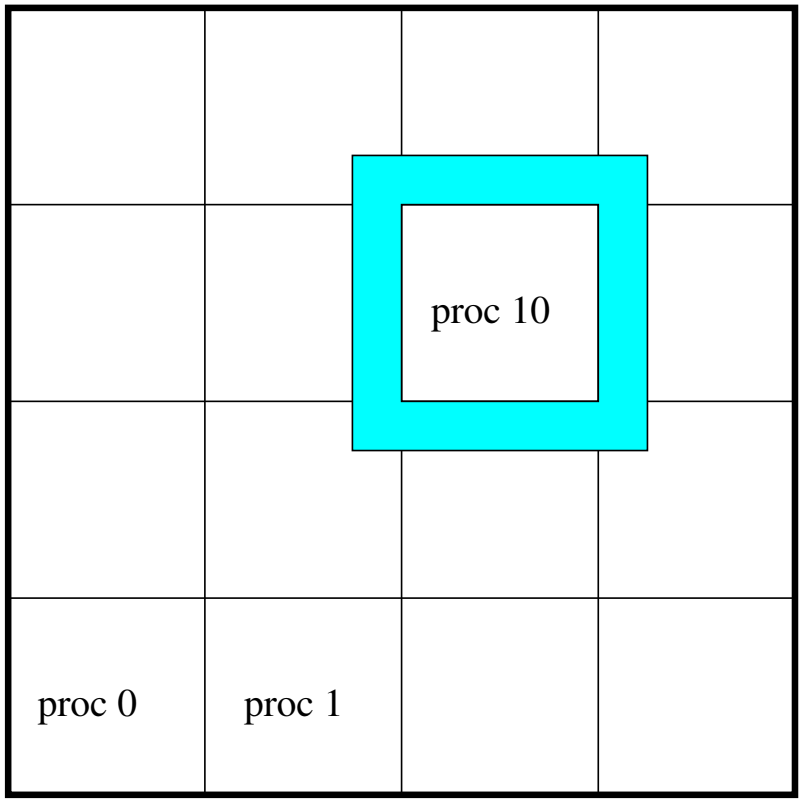
`ctx`: The user context passed to `DASetLocalFunction()`

The local DA function is activated by calling

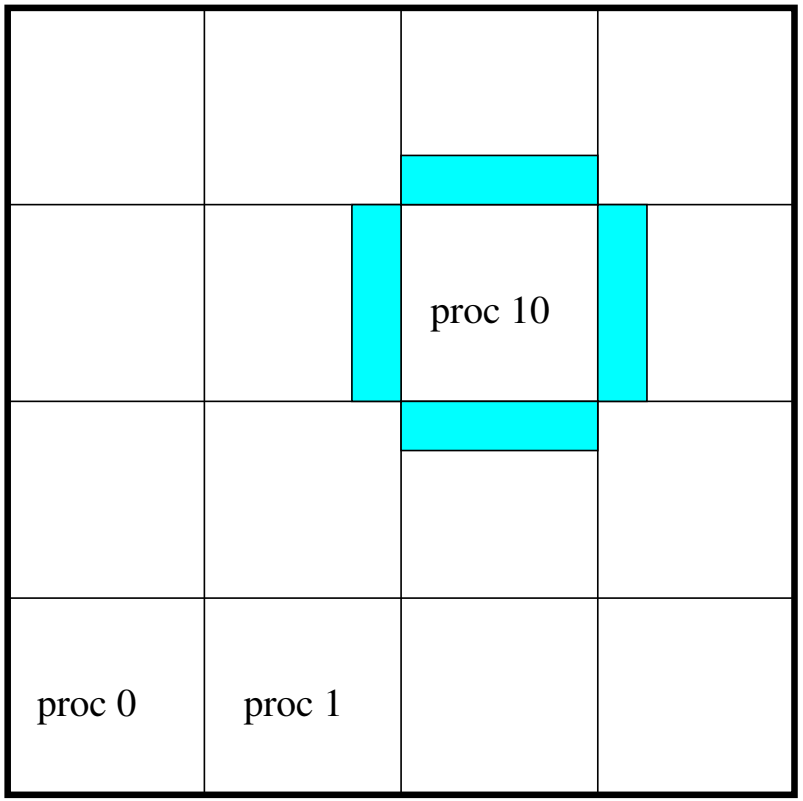
```
SNESSetFunction(snes, r, SNESDAFormFunction, ctx)
```

DA Stencils

Both the box stencil and star stencil are available.



Box Stencil



Star Stencil

Setting Values on Regular Grids

PETSc provides

```
MatSetValuesStencil(Mat A, m, MatStencil idxm[], n,
                    MatStencil idxn[], values[], mode)
```

- Each row or column is actually a `MatStencil`
  - This specifies grid coordinates and a component if necessary
  - Can imagine for unstructured grids, they are *vertices*
- The values are the same logically dense block in rows and columns

Code Update

Update to Revision 5

Structured Functions

- Functions takes values at the DA vertices
- Used as approximations to functions on the continuous domain
  - Values are really coefficients of linear basis
- User only constructs the local portion
- `make NP=2 EXTRA_ARGS="-test 1 -vec_view_draw -draw_pause -1" runbratu`

Code Update

Update to Revision 6

Sections

Sections associate data to submeshes

- Name comes from section of a fiber bundle
  - Generalizes linear algebra paradigm
- Define `restrict()`,`update()`
- Define `complete()`
- Assembly routines take a `Topology` and several `Sections`
  - This is called a `Bundle`

Section Types

Section can contain arbitrary values

- C++ interface is templated over value type
- C interface has three value types
  - `SectionReal`
  - `SectionInt`
  - `SectionPair`

Section can have arbitrary layout

- C++ interface can place unknowns on any Mesh entity (Sieve point)
- C interface has default layouts
  - `MeshGetVertexSectionReal()`
  - `MeshGetCellSectionReal()`

Viewing the Section

- `make EXTRA_ARGS="-test 1 -structured 0 -vec_view_vtk" runbratu`
  - Produces linear.vtk and cos.vtk
- Viewable with MayaVi, exactly as with the mesh.

Weak Forms

A *weak form* is the pairing of a function with an element of the *dual space*.

- Produces a number (by definition of the dual)
- Can be viewed as a "function" of the dual vector
- Used to define finite element solutions
- Require a dual space and integration rules

For example, for $f \in V$, we have the weak form

$$\int_\Omega \phi(\mathbf{x}) f(\mathbf{x}) dx \qquad \phi \in V^*$$

Code Update

Update to Revision 7

FIAT Integration

Finite Element Integrator and Tabulator by Rob Kirby

`http://www.fenics.org/fiat`

The `quadrature.fiat` file contains:

- An element (usually a family and degree) defined by FIAT
- A quadrature rule

Then `make` produces `quadrature.h` with:

- Quadrature points and weights
- Basis function evaluations at the quadrature points
- Basis function derivative evaluations at the quadrature points

FIAT is part of the FEniCS project, as is the PETSc Sieve module

Code Update

Update to Revision 8

Viewing a DA Weak Form

- We use Q1 finite elements and a *Galerkin* formulation
  - Uses a linear basis in each dimension
  - Should really use a fast tensor evaluation routine
- Could substitute exact integrals for quadrature
- `make EXTRA_ARGS="-test 1 -vec_view_draw -draw_pause -1" runbratu`

Debugging Assembly

On two processes, I get a SEGV!

So we try running with:

- `make NP=2 EXTRA_ARGS="-test 1 -vec_view_draw -draw_pause -1" debugbratu`

Debugging Assembly

On two processes, I get a SEGV!

So we try running with:

- `make NP=2 EXTRA_ARGS="-test 1 -vec_view_draw -draw_pause -1" debugbratu`
- Spawns one debugger window per process

Debugging Assembly

On two processes, I get a SEGV!

So we try running with:

- `make NP=2 EXTRA_ARGS="-test 1 -vec_view_draw -draw_pause -1" debugbratu`
- Spawns one debugger window per process
- SEGV on access to ghost coordinates

Debugging Assembly

On two processes, I get a SEGV!

So we try running with:

- `make NP=2 EXTRA_ARGS="-test 1 -vec_view_draw -draw_pause -1" debugbratu`
- Spawns one debugger window per process
- SEGV on access to ghost coordinates
- Fix by using a local ghosted vector
  - Update to Revision 9

Debugging Assembly

On two processes, I get a SEGV!

So we try running with:

- `make NP=2 EXTRA_ARGS="-test 1 -vec_view_draw -draw_pause -1" debugbratu`
- Spawns one debugger window per process
- SEGV on access to ghost coordinates
- Fix by using a local ghosted vector
  - Update to Revision 9
- Notice that we were already using ghosted assembly
  - Could eliminate this by reorganizing element traversal

Code Update

Update to Revision 10

Section Assembly

First we do local operations:

- Loop over cells
- Compute cell geometry
- Integrate each basis function to produce an element vector
- Call `SectionUpdateAdd()`
  - Note that this updates the *closure* of the cell

Then we do global operations:

- `SectionComplete()` exchanges data across overlap
  - C just adds nonlocal values (C++ is flexible)
- C++ interface allow completion over arbitrary overlaps

```
make NP=2 EXTRA_ARGS="-test 1 -structured 0 -vec_view_vtk" runbratu
```

Global and Local

Local (analytical)

- Discretization/Approximation
  - FEM integrals
  - FV fluxes
- Boundary conditions

Global and Local

Local (analytical)

- Discretization/Approximation
  - FEM integrals
  - FV fluxes
- Boundary conditions

Largely dim dependent
(e.g. quadrature)

Global and Local

Local (analytical)

- Discretization/Approximation
  - FEM integrals
  - FV fluxes
- Boundary conditions

Largely dim dependent
(e.g. quadrature)

Global (topological)

- Data management
  - Sections (local pieces)
  - Completions (assembly)
- Boundary definition
- Multiple meshes
  - Mesh hierarchies

Global and Local

Local (analytical)

- Discretization/Approximation
  - FEM integrals
  - FV fluxes
- Boundary conditions

Largely dim dependent
(e.g. quadrature)

Global (topological)

- Data management
  - Sections (local pieces)
  - Completions (assembly)
- Boundary definition
- Multiple meshes
  - Mesh hierarchies

Largely dim independent
(e.g. mesh traversal)

Part IV

Defining an Operator

Update to Revision 11

DA Operators

- Evaluate again only the local portion
  - No nice local array form without copies
- Use `MatSetValuesStencil()` to convert `(i,j,k)` to indices
- Notice we use J^{-1} to convert derivatives
- `make NP=1 EXTRA_ARGS="-test 1 -mat_view_draw -draw_pause -1" runbratu`

DA Local Jacobian

The user provided function which calculates the nonlinear residual in 2D has signature

```
PetscErrorCode (*lfunc)(DALocalInfo *info, PetscScalar **x,
                        Mat J, void *ctx)
```

`info`: All layout and numbering information

`x`: The current solution

`J`: The Jacobian

`ctx`: The user context passed to `DASetLocalFunction()`

The local DA function is activated by calling

```
SNESSetJacobian(snes, J, J, SNESDAComputeJacobian, ctx)
```

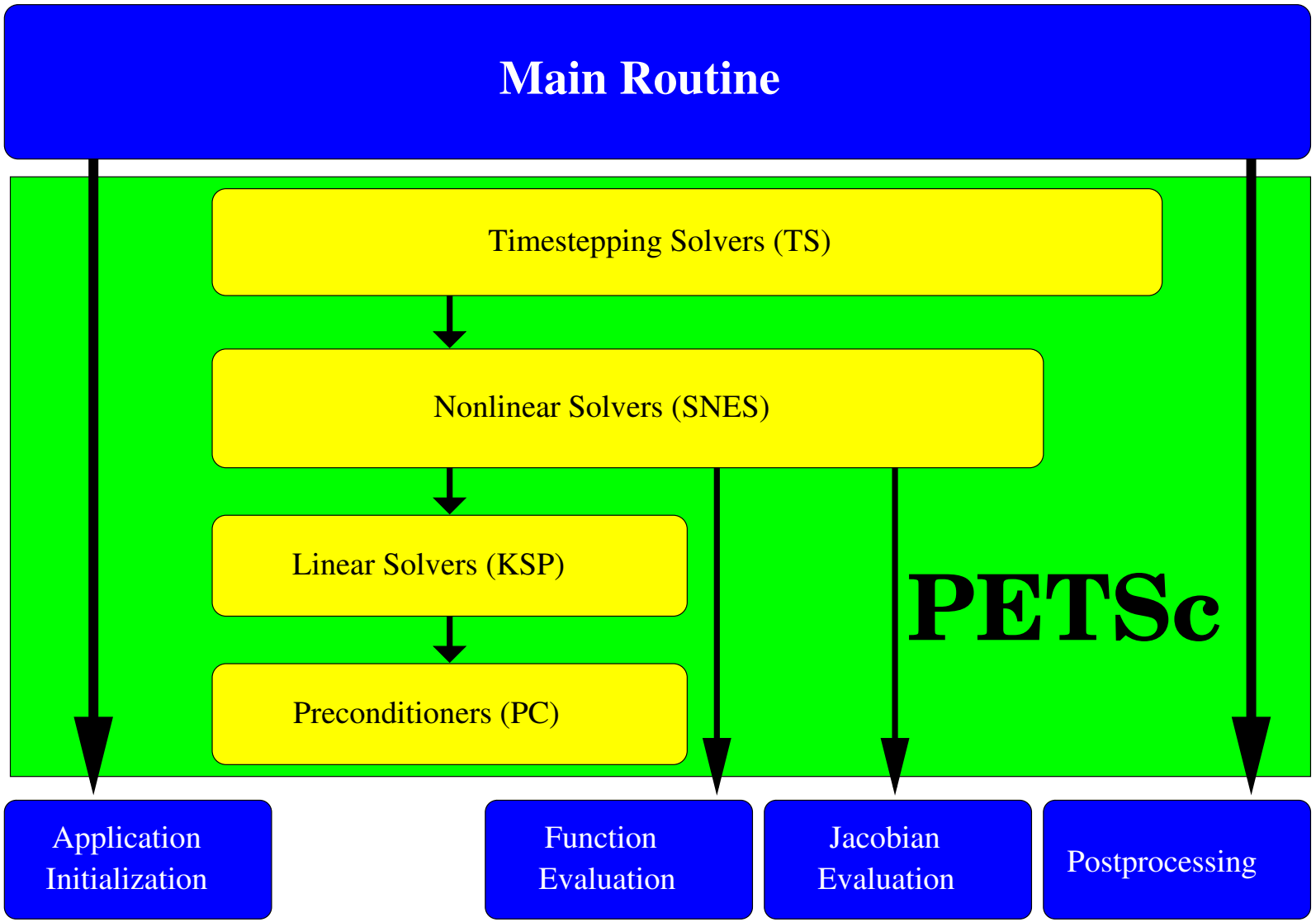
Mesh Operators

- We evaluate the local portion just as with functions
- Currently `updateOperator()` uses `MatSetValues()`
  - We should properly have `OperatorComplete()`
  - Also requires a Section, for layout, and a global variable order
- `make NP=1 EXTRA_ARGS="-test 1 -structured 0 -mat_view_draw -draw_pause -1" runbratu`

Part V

Solving Systems of Equations

Flow Control for a PETSc Application



SNESCallbacks

The SNES interface is based upon callback functions

- `SNESSetFunction()`
- `SNESSetJacobian()`

When PETSc needs to evaluate the nonlinear residual $F(x)$, the solver calls the user's function inside the application.

The user function get application state through the `ctx` variable. PETSc never sees application data.

SNES Function

The user provided function which calculates the nonlinear residual has signature

```
PetscErrorCode (*func)(SNES snes, Vec x, Vec r, void *ctx)
```

- `x`: The current solution
- `r`: The residual
- `ctx`: The user context passed to `SNESSetFunction()`
  - Use this to pass application information, e.g. physical constants

SNES Jacobian

The user provided function which calculates the Jacobian has signature

```
PetscErrorCode (*func)(SNES snes, Vec x, Mat *J, Mat *M,
                       MatStructure *flag, void *ctx)
```

- `x`: The current solution
- `J`: The Jacobian
- `M`: The Jacobian preconditioning matrix (possibly J itself)
- `ctx`: The user context passed to `SNESSetFunction()`
  - Use this to pass application information, e.g. physical constants
- Possible `MatrStructure` values are:
  - SAME_NONZERO_PATTERN, DIFFERENT_NONZERO_PATTERN, ...

Alternatively, you can use

- a builtin sparse finite difference approximation
- automatic differentiation
  - AD support via ADIC/ADIFOR (P. Hovland and B. Norris from ANL)

SNES Variants

- Line search strategies
- Trust region approaches
- Pseudo-transient continuation
- Matrix-free variants

Finite Difference Jacobians

PETSc can compute and explicitly store a Jacobian via 1st-order FD

- Dense
  - Activated by `-snes_fd`
  - Computed by `SNESDefaultComputeJacobian()`
- Sparse via colorings
  - Coloring is created by `MatFDColoringCreate()`
  - Computed by `SNESDefaultComputeJacobianColor()`

Can also use Matrix-free Newton-Krylov via 1st-order FD

- Activated by `-snes_mf` without preconditioning
- Activated by `-snes_mf_operator` with user-defined preconditioning
  - Uses preconditioning matrix from `SNESSetJacobian()`

Update to Revision 12

DMMG Integration with SNES

- DMMG supplies global residual and Jacobian to SNES
  - User supplies local version to DMMG
  - The `Rhs_*()` and `Jac_*()` functions in the example
- Allows automatic parallelism
- Allows grid hierarchy
  - Enables multigrid once interpolation/restriction is defined
- Paradigm is developed in unstructured work
  - Notice we have to scatter into contiguous global vectors (initial guess)
- Handle Neumann BC using `DMMGSetNullSpace()`

Part VI

Out of Flatland

Update to Revision 13

Structured Mesh Conversion

- Added new constructor call
- Added new local evaluation routines
  - `Rhs_Structured_3d()` and `Jac_Structured_3d()`
- Added new 3D source term

Unstructured Mesh Conversion

- Added new quadrature rule
- No need to change evaluation routines
  - Just need to pick the correct quadrature
- Added new 3D mesh files
  - Interfaces to TetGen (and soon TUMBLE) mesh generator

Part VII

The Undiscovered Country

What We Have Not Covered

- Unstructured hexes
- Nonlinearity
- Dirichlet BC
- Error Estimation
- Multigrid
- Semi-Lagrangian Schemes

References

- Documentation: http://www.mcs.anl.gov/petsc/docs
  - PETSc Users manual
  - Manual pages
  - Many hyperlinked examples
  - FAQ, Troubleshooting info, installation info, etc.
- Publications: http://www.mcs.anl.gov/petsc/publications
  - Research and publications that make use PETSc
- MPI Information: http://www.mpi-forum.org
- **Using MPI** (2nd Edition), by Gropp, Lusk, and Skjellum
- **Domain Decomposition**, by Smith, Bjorstad, and Gropp

Experimentation is Essential!

Proof is not currrently enough to examine solvers

- N. M. Nachtigal, S. C. Reddy, and L. N. Trefethen, *How fast are nonsymmetric matrix iterations?*, SIAM J. Matrix Anal. Appl., **13**, pp.778–795, 1992.
- Anne Greenbaum, Vlastimil Ptak, and Zdenek Strakos, *Any Nonincreasing Convergence Curve is Possible for GMRES*, SIAM J. Matrix Anal. Appl., **17** (3), pp.465–469, 1996.