The Portable Extensible Toolkit for Scientific Computing

Matthew Knepley

Mathematics and Computer Science Division
Argonne National Laboratory

Computation Institute
University of Chicago

PETSc Tutorial
13th Workshop on the DOE ACTS Collection
LBNL, Berkeley, CA     August 14–17, 2012
Never believe *anything*,

unless you can run it.
Never believe *anything*,

unless you can run it.
Outline

1. Getting Started with PETSc
   - What is PETSc?
   - Who uses PETSc?
   - Stuff for Windows
   - How can I get PETSc?
   - How do I Configure PETSc?
   - How do I Build PETSc?
   - How do I run an example?
   - How do I get more help?

2. SNES ex62

3. Solvers

4. FieldSplit
Getting Started with PETSc

What is PETSc?
Who uses PETSc?
Stuff for Windows
How can I get PETSc?
How do I Configure PETSc?
How do I Build PETSc?
How do I run an example?
How do I get more help?
Unit Objectives

- Introduce PETSc
- Download, Configure, Build, and Run an Example
- Empower students to learn more about PETSc
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
Ask Questions!!

- Helps me understand what you are missing
- Helps you clarify misunderstandings
- Helps others with the same question
How We Can Help at the Tutorial

- Point out relevant documentation
- Quickly answer questions
- Help install
- Guide design of large scale codes
- Answer email at petsc-maint@mcs.anl.gov
How We Can Help at the Tutorial

- Point out relevant documentation
- Quickly answer questions
- Help install
- Guide design of large scale codes
- Answer email at petsc-maint@mcs.anl.gov
How We Can Help at the Tutorial

- Point out relevant documentation
- Quickly answer questions
- Help install
- Guide design of large scale codes
- Answer email at petsc-maint@mcs.anl.gov
How We Can Help at the Tutorial

- Point out relevant documentation
- Quickly answer questions
- Help install
- Guide design of large scale codes
- Answer email at petsc-maint@mcs.anl.gov
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
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.

— Barry Smith
You want to think about how you decompose your data structures, how you think about them globally. [...] If you were building a house, you’d start with a set of blueprints that give you a picture of what the whole house looks like. You wouldn’t start with a bunch of tiles and say, “Well I’ll put this tile down on the ground, and then I’ll find a tile to go next to it.” But all too many people try to build their parallel programs by creating the smallest possible tiles and then trying to have the structure of their code emerge from the chaos of all these little pieces. You have to have an organizing principle if you’re going to survive making your code parallel.

(http://www.rce-cast.com/Podcast/rce-28-mpich2.html)
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 XT5, BG/Q, NVIDIA Fermi, Earth Simulator
  - Loosely coupled systems, such as networks of workstations
    - IBM, Mac, iPad/iPhone, PCs running Linux or Windows

PETSc History
- Begun September 1991
- Over 60,000 downloads since 1995 (version 2)
- Currently 400 per month

PETSc Funding and Support
- Department of Energy
  - SciDAC, MICS Program, AMR Program, INL Reactor Program
- National Science Foundation
  - CIG, CISE, Multidisciplinary Challenge Program
What is PETSc?

Timeline

- PETSc-1
- MPI-1
- Barry
- 1991

- PETSc-2
- MPI-2
- Bill
- 1995

- PETSc-3
- Satish
- 2000

- Dinesh
- 2005

- Hong
- 2010

- Kris
- Matt
- Victor
- Dmitry
- Lisandro
- Jed
The PETSc Team

Bill Gropp
Barry Smith
Satish Balay
Jed Brown
Matt Knepley
Lisandro Dalcin
Hong Zhang
Victor Eijkhout
Dmitry Karpeev
1 Getting Started with PETSc

- What is PETSc?
- Who uses PETSc?
- Stuff for Windows
- How can I get PETSc?
- How do I Configure PETSc?
- How do I Build PETSc?
- How do I run an example?
- How do I get more help?
Who Uses PETSc?

Computational Scientists

- **Earth Science**
  - PyLith (CIG)
  - Underworld (Monash)
  - Magma Dynamics (LDEO, Columbia)

- **Subsurface Flow and Porous Media**
  - STOMP (DOE)
  - PFLOTRAN (DOE)
Who Uses PETSc?

Computational Scientists

- CFD
  - Fluidity
  - OpenFOAM
  - freeCFD
  - OpenFVM

- MicroMagnetics
  - MagPar

- Fusion
  - NIMROD
Who Uses PETSc?

Algorithm Developers

- **Iterative methods**
  - Deflated GMRES
  - LGMRES
  - QCG
  - SpecEst

- **Preconditioning researchers**
  - Prometheus (Adams)
  - ParPre (Eijkhout)
  - FETI-DP (Klawonn and Rheinbach)
Algorithm Developers

- Finite Elements
  - PETSc-FEM
  - libMesh
  - Deal II
  - OOFEM

- Other Solvers
  - Fast Multipole Method (PetFMM)
  - Radial Basis Function Interpolation (PetRBF)
  - Eigensolvers (SLEPc)
  - Optimization (TAO)
PETSc has run implicit problems with over 500 billion unknowns
- UNIC on BG/P and XT5
- PFLOTRAN for flow in porous media

PETSc has run on over 290,000 cores efficiently
- UNIC on the IBM BG/P Intrepid at ANL
- PFLOTRAN on the Cray XT5 Jaguar at ORNL

PETSc applications have run at 22 Teraflops
- Kaushik on XT5
- LANL PFLOTRAN code
PETSc has run implicit problems with over 500 billion unknowns
- UNIC on BG/P and XT5
- PFLOTRAN for flow in porous media

PETSc has run on over 290,000 cores efficiently
- UNIC on the IBM BG/P Intrepid at ANL
- PFLOTRAN on the Cray XT5 Jaguar at ORNL

PETSc applications have run at 22 Teraflops
- Kaushik on XT5
- LANL PFLOTRAN code
What Can We Handle?

- PETSc has run implicit problems with over 500 billion unknowns
  - UNIC on BG/P and XT5
  - PFLOTRAN for flow in porous media

- PETSc has run on over 290,000 cores efficiently
  - UNIC on the IBM BG/P Intrepid at ANL
  - PFLOTRAN on the Cray XT5 Jaguar at ORNL

- PETSc applications have run at 22 Teraflops
  - Kaushik on XT5
  - LANL PFLOTRAN code
PyLith

- Multiple problems
  - Dynamic rupture
  - Quasi-static relaxation
- Multiple models
  - Nonlinear visco-plastic
  - Finite deformation
  - Fault constitutive models
- Multiple meshes
  - 1D, 2D, 3D
  - Hex and tet meshes
- Parallel
  - PETSc solvers
  - Sieve mesh management

\*Aagaard, Knepley, Williams\*
Multiple Mesh Types

- Triangular
- Tetrahedral
- Rectangular
- Hexahedral
Magma Dynamics

- Couples scales
  - Subduction
  - Magma Migration
- Physics
  - Incompressible fluid
  - Porous solid
  - Variable porosity
- Deforming matrix
  - Compaction pressure
- Code generation
  - FEniCS
- Multiphysics Preconditioning
  - PETSc FieldSplit

\[a\] Katz, Speigelman
Magma Dynamics

- Couples scales
  - Subduction
  - Magma Migration
- Physics
  - Incompressible fluid
  - Porous solid
  - Variable porosity
- Deforming matrix
  - Compaction pressure
- Code generation
  - FEniCS
- Multiphysics Preconditioning
  - PETSc FieldSplit

\(^{a}\)Katz, Speigelman
Fracture Mechanics

- Full variational formulation
  - Phase field
  - Linear or Quadratic penalty

- Uses TAO optimization
  - Necessary for linear penalty
  - Backtacking

- No prescribed cracks
  - Arbitrary crack geometry
  - Arbitrary intersections

- Multiple materials
  - Composite toughness

\[ ^a \text{Bourdin} \]
Fracture Mechanics

1 Bourdin

\[ M. \text{Knepley (UC)} \]
Vortex Method

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re
- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates
- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner
- Parallelism
  - MPI
  - GPU

\(^a\text{Cruz, Yokota, Barba, Knepley}\)
Vortex Method

$\text{t} = 100$

- **Incompressible Flow**
  - Gaussian vortex blobs
  - High Re

- **PetFMM**
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates

- **PetRBF**
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner

- **Parallelism**
  - MPI
  - GPU

\[ ^a \text{Cruz, Yokota, Barba, Knepley} \]
Vortex Method

t = 200

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re

- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates

- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner

- Parallelism
  - MPI
  - GPU

\(^a\)Cruz, Yokota, Barba, Knepley
Vortex Method
\( t = 300 \)

- **Incompressible Flow**
  - Gaussian vortex blobs
  - High Re
- **PetFMM**
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates
- **PetRBF**
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner
- **Parallelism**
  - MPI
  - GPU

\(^a\)Cruz, Yokota, Barba, Knepley
Vortex Method

\( t = 400 \)

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re

- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates

- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner

- Parallelism
  - MPI
  - GPU

\(^a\text{Cruz, Yokota, Barba, Knepley}\)
Incompressible Flow
- Gaussian vortex blobs
- High Re

PetFMM
- 2D/3D domains
- Automatic load balancing
- Variety of kernels
- Optimized with templates

PetRBF
- Variety of RBFs
- Uses PETSc solvers
- Scalable preconditioner

Parallelism
- MPI
- GPU

\(^a\text{Cruz, Yokota, Barba, Knepley}\)
Incompressible Flow
- Gaussian vortex blobs
- High Re

PetFMM
- 2D/3D domains
- Automatic load balancing
- Variety of kernels
- Optimized with templates

PetRBF
- Variety of RBFs
- Uses PETSc solvers
- Scalable preconditioner

Parallelism
- MPI
- GPU

\(^a\)Cruz, Yokota, Barba, Knepley
Vortex Method

\( t = 700 \)

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re
- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates
- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner
- Parallelism
  - MPI
  - GPU

\(^acruz, yokota, barba, knepley\)
Incompressible Flow
- Gaussian vortex blobs
- High Re

PetFMM
- 2D/3D domains
- Automatic load balancing
- Variety of kernels
- Optimized with templates

PetRBF
- Variety of RBFs
- Uses PETSc solvers
- Scalable preconditioner

Parallelism
- MPI
- GPU

\(^a\)Cruz, Yokota, Barba, Knepley
Gravity Anomaly Modeling

- Potential Solution
  - Kernel of inverse problem
  - Needs optimal algorithm

- Implementations
  - Direct Summation
  - FEM
  - FMM

- Parallelism
  - MPI
  - 4000+ cores
  - All methods scalable

---

\[ a \]  

May, Knepley
FEniCS-Apps
Rheagen

- **Rheologies**
  - Maxwell
  - Grade 2
  - Oldroyd-B

- **Stabilization**
  - DG
  - SUPG
  - EVSS
  - DEVSS
  - Macroelement

- **Automation**
  - FIAT (elements)
  - FFC (weak forms)

\(^a\)Terrel

Who uses PETSc?

PETSc

ACTS '12 27 / 105
Rheologies
- Maxwell
- Grade 2
- Oldroyd-B

Stabilization
- DG
- SUPG
- EVSS
- DEVSS
- Macroelement

Automation
- FIAT (elements)
- FFC (weak forms)
Real-time Surgery

- **Brain Surgery**
  - Elastic deformation
  - Overlaid on MRI
  - Guides surgeon

- **Laser Thermal Therapy**
  - PDE constrained optimization
  - Per-patient calibration
  - Thermal inverse problem

---

*a*Warfield, Ferrant, et.al.
Brain Surgery
- Elastic deformation
- Overlaid on MRI
- Guides surgeon

Laser Thermal Therapy
- PDE constrained optimization
- Per-patient calibration
- Thermal inverse problem

Fuentes, Oden, et.al.
Getting Started with PETSc

- What is PETSc?
- Who uses PETSc?

Stuff for Windows

- How can I get PETSc?
- How do I Configure PETSc?
- How do I Build PETSc?
- How do I run an example?
- How do I get more help?
Questions for Windows Users

- Have you installed cygwin?
  - Need python, make, and build-utils packages

- Will you use the GNU compilers?
  - If not, remove `link.exe`
  - If MS, check compilers from `cmd window` and use `win32fe`

- Which MPI will you use?
  - You can use `--with-mpi=0`
  - If MS, need to install MPICH2
  - If GNU, can use `--download-mpich`
Getting Started with PETSc

- What is PETSc?
- Who uses PETSc?
- Stuff for Windows

How can I get PETSc?
- How do I Configure PETSc?
- How do I Build PETSc?
- How do I run an example?
- How do I get more help?
The latest tarball is on the PETSc site
- 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
Getting Started with PETSc

How can I get PETSc?

Cloning PETSc

- The full development repository is open to the public
  - http://petsc.cs.iit.edu/petsc/petsc-dev
  - http://petsc.cs.iit.edu/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

- We also make release repositories available
  - http://petsc.cs.iit.edu/petsc/releases/petsc-3.3
  - http://petsc.cs.iit.edu/petsc/releases/BuildSystem-3.3
Unpacking PETSc

- Just clone development repository
  - `hg clone http://petsc.cs.iit.edu/petsc/petsc-dev petsc-dev`
  - `hg clone -rrelease-3.3 petsc-dev petsc-3.3`

  or

- Unpack the tarball
  - `tar xzf petsc.tar.gz`
Exercise 1

Download and Unpack PETSc!
Outline

1 Getting Started with PETSc
   • What is PETSc?
   • Who uses PETSc?
   • Stuff for Windows
   • How can I get PETSc?
   • How do I Configure PETSc?
     • How do I Build PETSc?
     • How do I run an example?
     • How do I get more help?
Configuring PETSc

- Set $PETSC_DIR$ to the installation root directory
- Run the configuration utility
  - $PETSC_DIR$/configure
  - $PETSC_DIR$/configure -help
  - $PETSC_DIR$/configure -download-mpich
  - $PETSC_DIR$/configure -prefix=/usr
- There are many examples on the installation page
- Configuration files are in $PETSC_DIR/$PETSC_ARCH/conf
  - Configure header is in $PETSC_DIR/$PETSC_ARCH/include
  - $PETSC_ARCH$ has a default if not specified
Configuring PETSc

- You can easily reconfigure with the same options
  
  ```
  ./$PETSC_ARCH/conf/reconfigure-$PETSC_ARCH.py
  ```

- Can maintain several different configurations
  
  ```
  ./configure -PETSC_ARCH=linux-fast
  -with-debugging=0
  ```

- All configuration information is in the logfile
  
  ```
  ./$PETSC_ARCH/conf/configure.log
  ```

  ALWAYS send this file with bug reports
Configuring PETSc for FEM

$PETSC_DIR/configure
   –download-triangle –download-ctetgen
   –download-chaco –download-parmetis
   –download-scientificpython –download-fiati –download-generator
   –with-cuda
   –with-cudac=’nvcc -m64’ –with-cuda-arch=sm_10
   –with-cusp-dir=/PETSc3/multicore/cusp
   –with-thrust-dir=/PETSc3/multicore/thrust
   –with-cuda-only
   –with-precision=single
$PETSC_DIR/configure

–download-triangle –download-ctetgen
–download-chaco –download-parmetis
–download-scientificpython –download-fiat –download-generator
–with-cuda
–with-cudac=’nvcc -m64’ –with-cuda-arch=sm_10
–with-cusp-dir=/PETSc3/multicore/cusp
–with-thrust-dir=/PETSc3/multicore/thrust
–with-cuda-only
–with-precision=single
$PETSC_DIR/configure
   --download-triangle --download-ctetgen
   --download-chaco --download-parmetis
   --download-scientificpython --download-fiat --download-generator
   --with-cuda
   --with-cudac='nvcc -m64' --with-cuda-arch=sm_10
   --with-cusp-dir=/PETSc3/multicore/cusp
   --with-thrust-dir=/PETSc3/multicore/thrust
   --with-cuda-only
   --with-precision=single
Configuring PETSc for FEM

$PETSC_DIR/configure

  –download-triangle –download-ctetgen
  –download-chaco –download-parmetis
  –download-scientificpython –download-fiat –download-generator
  –with-cuda
  –with-cudac=’nvcc -m64’ –with-cuda-arch=sm_10
  –with-cusp-dir=/PETSc3/multicore/cusp
  –with-thrust-dir=/PETSc3/multicore/thrust
  –with-cuda-only
  –with-precision=single
$PETSC_DIR/configure
  --download-triangle --download-ctetgen
  --download-chaco --download-parmetis
  --download-scientificpython --download-fiat --download-generator
  --with-cuda
  --with-cudac='nvcc -m64' --with-cuda-arch=sm_10
  --with-cusp-dir=/PETSc3/multicore/cusp
  --with-thrust-dir=/PETSc3/multicore/thrust
  --with-cuda-only
  --with-precision=single
Getting Started with PETSc

How do I Configure PETSc?

Configuring PETSc for FEM

$PETSC_DIR/configure
   --download-triangle --download-ctetgen
   --download-chaco --download-parmetis
   --download-scientificpython --download-fiat --download-generator
   --with-cuda
   --with-cudac='nvcc -m64' --with-cuda-arch=sm_10
   --with-cusp-dir=/PETSc3/multicore/cusp
   --with-thrust-dir=/PETSc3/multicore/thrust
   --with-cuda-only
   --with-precision=single
$PETSC_DIR/configure
   –download-triangle –download-ctetgen
   –download-chaco –download-parmetis
   –download-scientificpython –download-fiato –download-generator
   –with-cuda
   –with-cudac='nvcc -m64’ –with-cuda-arch=sm_10
   –with-cusp-dir=/PETSc3/multicore/cusp
   –with-thrust-dir=/PETSc3/multicore/thrust
   –with-cuda-only
   –with-precision=single
Starting in 2.2.1, some packages are automatically
- Downloaded
- Configured and Built (in $PETSC_DIR/externalpackages)
- Installed with PETSc

Currently works for
- petsc4py
- PETSc documentation utilities (Sowing, lgrind, c2html)
- BLAS, LAPACK, BLACS, ScaLAPACK, PLAPACK
- MPICH, MPE, OpenMPI
- ParMetis, Chaco, Jostle, Party, Scotch, Zoltan
- MUMPS, Spooles, SuperLU, SuperLU_Dist, UMFPack, pARMS
- BLOPEX, FFTW, SPRNG
- Prometheus, HYPRE, ML, SPAI
- Sundials
- Triangle, TetGen
- FIAT, FFC, Generator
- Boost
Configure your downloaded PETSc.
Outline

1. Getting Started with PETSc
   - What is PETSc?
   - Who uses PETSc?
   - Stuff for Windows
   - How can I get PETSc?
   - How do I Configure PETSc?
   - How do I Build PETSc?
   - How do I run an example?
   - How do I get more help?
Building PETSc

There are three valid ways to build PETSc:

- **Using recursive make starting in** `cd $PETSC_DIR`
  - `make`
  - `make install` if you configured with `--prefix`
  - Check build when done with `make test`

- **Using CMake**
  - Same `make`, `make install`, `make test`
  - Automatically enabled if CMake is found by configure
  - Handles dependencies

- **Experimental Python build**
  - `python ./config/builder2.py -help` for Python 2.7
  - `./config/builder.py` for older Python
  - Handles dependencies
Building PETSc

- Can build multiple configurations
  - `PETSC_ARCH=linux-fast make`
  - Libraries are in `$PETSC_DIR/$PETSC_ARCH/lib/`

- Complete log for each build is in logfile
  - `./$PETSC_ARCH/conf/make.log`
  - ALWAYS send this with bug reports

- (Deprecated) Can also build a subtree with recursive make
  - `cd src/snes; make`
  - `cd src/snes; make ACTION=libfast tree`
Build your configured PETSc.
Reconfigure PETSc to use ParMetis.

1. `linux-c-debug/conf/reconfigure-linux-c-debug.py`
   - `PETSC_ARCH=linux-parmetis`
   - `download-parmetis`

2. `PETSC_ARCH=linux-parmetis make`

3. `PETSC_ARCH=linux-parmetis make test`
1. Getting Started with PETSc
   - What is PETSc?
   - Who uses PETSc?
   - Stuff for Windows
   - How can I get PETSc?
   - How do I Configure PETSc?
   - How do I Build PETSc?
   - How do I run an example?
   - How do I get more help?
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
Running PETSc with Python

- Can run any PETSc example
  - `python ./config/builder2.py check
    $PETSC_DIR/src/snes/examples/tutorials/ex5.c`

- Checks against test output
  - Ignores if no output is present

- Can specify multiple files
  - `python ./config/builder2.py check
    [$PETSC_DIR/src/snes/examples/tutorials/ex5.c,extraFile.c]`

- Can also run using MPI directly
  - Use `-retain` to keep executable
  - `mpiexec ./$PETSC_ARCH/lib/lib-ex5/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 -n 4 ./ex5`

- You should never have to make MPI calls when using PETSc
  - Almost never
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
- Set PETSC_COMM_WORLD to put all of PETSc in a subcomm

Point-to-point communication
- Happens between two processes (like in MatMult())

Reduction or scan operations
- Happens among all processes (like in VecDot())
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
  - \texttt{-ksp\_monitor}, \texttt{graphically -ksp\_monitor\_draw}
- Can disable dynamically
  - \texttt{-ksp\_monitors\_cancel}
- Does not display subsolvers
  - \texttt{-snes\_monitor}
- Can use the true residual
  - \texttt{-ksp\_monitor\_true\_residual}
- Can display different subobjects
  - \texttt{-snes\_monitor\_residual, -snes\_monitor\_solution, -snes\_monitor\_solution\_update}
  - \texttt{-snes\_monitor\_range}
  - \texttt{-ksp\_gmres\_krylov\_monitor}
- Can display the spectrum
  - \texttt{-ksp\_monitor\_singular\_value}
Outline

1 Getting Started with PETSc
   • What is PETSc?
   • Who uses PETSc?
   • Stuff for Windows
   • How can I get PETSc?
   • How do I Configure PETSc?
   • How do I Build PETSc?
   • How do I run an example?
   • How do I get more help?
Getting More Help

- http://www.mcs.anl.gov/petsc
- Hyperlinked documentation
  - Manual
  - Manual pages for every method
  - HTML of all example code (linked to manual pages)
- FAQ
- Full support at petsc-maint@mcs.anl.gov
- High profile users
  - David Keyes
  - Marc Spiegelman
  - Richard Katz
  - Brad Aagaard
  - Aron Ahmadia
Outline

1. Getting Started with PETSc
2. SNES ex62
3. Solvers
4. FieldSplit
5. DM
The isoviscous Stokes problem

\[ \Delta \vec{u} - \nabla p = \vec{f} \]
\[ \nabla \cdot \vec{u} = 0 \]

on the square domain \( \Omega = [0, 1]^2 \).

The sides of the box may have
- Dirichlet, or
- homogeneous Neumann boundary conditions.
We discretize using finite elements on an unstructured mesh:

\[
(\nabla \vec{v}, \nabla \vec{u}) - (\nabla \cdot \vec{v}, p) = -(\vec{v}, \vec{f}) \\
(q, \nabla \cdot \vec{u}) = 0
\]

A finite element basis tabulation header is generated using

```
bin/pythonscripts/PetscGenerateFEMQuadrature.py
dim order dim 1 laplacian
dim order 1 1 gradient ex62.h
```

**dim**  The spatial dimension

**order**  The order of the Lagrange element

The code should be capable of using any FIAT element, but has not yet been tested for this.
Outline

1. Getting Started with PETSc
2. SNES ex62
3. Solvers
   - Objects
   - Design
4. FieldSplit
5. DM
Outline

3 Solvers
   - Objects
   - Design
SNES snes;

SNESCreate(comm, &snes);
SNESSetOptionsPrefix(snes, "foo_");
SNESSetFromOptions(snes);
/* Use snes */
SNESView(snes, PETSC_VIEWER_DRAW_WORLD);
SNESDestroy(snes);

- **SNES** is an opaque object (pointer to incomplete type)
  - Assignment, comparison, etc, are cheap
- **What’s up with this *Options* stuff?**
  - Allows the type to be set at runtime: `-foo_snes_type qn`
  - Inversion of Control similar to service locator pattern, related to dependency injection
  - Other options (performance and semantics) can be changed at runtime under `-foo_snes_`
Every object in PETSc supports a basic interface

<table>
<thead>
<tr>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create()</td>
<td>create the object</td>
</tr>
<tr>
<td>Get/SetName()</td>
<td>name the object</td>
</tr>
<tr>
<td>Get/SetType()</td>
<td>set the implementation type</td>
</tr>
<tr>
<td>Get/SetOptionsPrefix()</td>
<td>set the prefix for all options</td>
</tr>
<tr>
<td>SetFromOptions()</td>
<td>customize from the command line</td>
</tr>
<tr>
<td>SetUp()</td>
<td>preform other initialization</td>
</tr>
<tr>
<td>View()</td>
<td>view the object</td>
</tr>
<tr>
<td>Destroy()</td>
<td>cleanup object allocation</td>
</tr>
</tbody>
</table>

Also, all objects support the -help option.
Ways to set options

- Command line
- File name in the third argument of `PetscInitialize()`
  - `~/.petsrcrc`
  - `$PWD/.petsrcrc`
  - `$PWD/petsrcrc`
  - `PetscOptionsInsertFile()`
  - `PetscOptionsInsertString()`
  - `PETSC_OPTIONS` environment variable
  - command line option `-options_file [file]`
Outline

3 Solvers
- Objects
- Design
Always use **SNES** instead of **KSP**:

- No more costly than linear solver
- Can accommodate unanticipated nonlinearities
- Automatic iterative refinement
- Callback interface can take advantage of problem structure

Jed actually recommends **TS**...
Always use **SNES** instead of **KSP**:  
- No more costly than linear solver  
- Can accommodate unanticipated nonlinearities  
- Automatic iterative refinement  
- Callback interface can take advantage of problem structure  

Jed actually recommends **TS**...
What about TS?

Didn’t Time Integration Suck in PETSc?

Yes, it did . . .

until Jed, Emil, and Peter rewrote it
Didn’t Time Integration Suck in PETSc?

Yes, it did . . .

until Jed, Emil, and Peter rewrote it
Didn’t Time Integration Suck in PETSc?

Yes, it did ... 

until Jed, Emil, and Peter rewrote it
IMEX time integration in PETSc

Additive Runge-Kutta IMEX methods

\[ G(t, x, \dot{x}) = F(t, x) \]
\[ J_\alpha = \alpha G_{\dot{x}} + G_x \]

User provides:

- `FormRHSFunction(ts,t,x,F,void *ctx)`
- `FormIFunction(ts,t,x,xdot,G,void *ctx)`
- `FormIJacobian(ts,t,x,xdot,alpha,J,J_p,mstr,void *ctx)`

- Single step interface so user can have own time loop
- Choice of explicit method, e.g. SSP
- L-stable DIRK for stiff part \( G \)
- Orders 2 through 5, embedded error estimates
- Dense output, hot starts for Newton
- More accurate methods if \( G \) is linear, also Rosenbrock-W
- Can use preconditioner from classical “semi-implicit” methods
- Extensible adaptive controllers, can change order within a family
- Easy to register new methods: `TSARKIMEXRegister()`
Some TS methods

**TSSSPRK104** 10-stage, fourth order, low-storage, optimal explicit SSP Runge-Kutta $c_{eff} = 0.6$ (Ketcheson 2008)

**TSARKIMEX2E** second order, one explicit and two implicit stages, $L$-stable, optimal (Constantinescu)

**TSARKIMEX3** (and 4 and 5), $L$-stable (Kennedy and Carpenter, 2003)

**TSROSWRA3PW** three stage, third order, for index-1 PDAE, $A$-stable, $R(\infty) = 0.73$, second order strongly $A$-stable embedded method (Rang and Angermann, 2005)

**TSROSWRA34PW2** four stage, third order, $L$-stable, for index 1 PDAE, second order strongly $A$-stable embedded method (Rang and Angermann, 2005)

**TSROSWLLSSP3P4S2C** four stage, third order, $L$-stable implicit, SSP explicit, $L$-stable embedded method (Constantinescu)
1D nonlinear hyperbolic conservation laws

src/ts/examples/tutorials/ex9.c
./ex9 -da_grid_x 100 -initial 1 -physics shallow -limit minmod -ts_ssp_type rks2 -ts_ssp_nstages 8 -ts_monitor_solution

Stiff linear advection-reaction test problem

src/ts/examples/tutorials/ex22.c
./ex22 -da_grid_x 200 -ts_monitor_solution -ts_type rosw -ts_rosw_type ra34pw2 -ts_adapt_monitor

1D Brusselator (reaction-diffusion)

src/ts/examples/tutorials/ex25.c
./ex25 -da_grid_x 40 -ts_monitor_solution -ts_type rosw -ts_rosw_type 2p -ts_adapt_monitor
New methods in SNES

LS, TR  Newton-type with line search and trust region
NRichardson  Nonlinear Richardson, usually preconditioned
VIRS, VISS  reduced space and semi-smooth methods for variational inequalities
QN  Quasi-Newton methods like BFGS
NGMRES  Nonlinear GMRES
NCG  Nonlinear Conjugate Gradients
SORQN  SOR quasi-Newton
GS  Nonlinear Gauss-Seidel sweeps
FAS  Full approximation scheme (nonlinear multigrid)
MS  Multi-stage smoothers (in FAS for hyperbolic problems)
Shell  Your method, often used as a (nonlinear) preconditioner
### Recent PETSc functionality: Indicated by blue

#### Time Integrators
- Pseudo-Timestepping
- General Linear
- Runge-Kutta
- IMEX
- Strong Stability Preserving
- Rosenbrock-W

#### Nonlinear Algebraic Solvers
- Line Search Newton
- Trust Region Newton
- Quasi-Newton (BFGS)
- Nonlinear Gauss-Seidel
- Successive Substitutions
- Nonlinear MG (FAS)
- Nonlinear CG
- Active Set VI

#### Krylov Subspace Solvers
- Richardson
- GMRES
- Hierarchical Krylov
- BiCG Stabilized
- Chebychev
- TFQMR
- LSQR
- SYMMLQ
- CG
- IBCGS

#### Preconditioners
- Blocks (by field)
- Additive Schwarz
- ILU/ICC
- Schur Complement
- Algebraic Multigrid
- Geometric Multigrid

#### Matrices
- Compressed Sparse Row (AIJ)
- Block AIJ
- Matrix Blocks (MatNest)
- Symmetric Block AIJ
- Dense
- GPU & PThread Matrices

### Vectors
- Index Sets

In PETSc, objects at higher levels of abstraction use lower-level objects.
Solver use in SNES ex62

Solver code does not change for different algorithms:

```c
SNES snes;
Vec u, r;
PetscErrorCode ierr;

ierr = SNESCreate(PETSC_COMM_WORLD, &snes);CHKERRQ(ierr);
/* Specify residual computation */
ierr = SNESSetFromOptions(snes);CHKERRQ(ierr); /* Configure solver */
ierr = SNESSolve(snes, PETSC_NULL, u);CHKERRQ(ierr);
```

- **Never recompile!** all configuration is dynamic
- Factories are hidden from the user
- Type of nested solvers can be changed at runtime
I will omit error checking and declarations:

```c
SNESCreate(PETSC_COMM_WORLD, &snes);
/* Specify residual computation */
SNESSetFromOptions(snes); /* Configure solver */
SNESolve(snes, PETSC_NULL, u);
```
The configuration API can also be used:

```c
SNESCreate(PETSC_COMM_WORLD, &snes);
/* Specify residual computation */
SNESNGMRESRestartType(snes, SNES_NGMRES_RESTART_PERIODIC);
SNESetFromOptions(snes);
SNESolve(snes, PETSC_NULL, u);
```

- Ignored when not applicable (no ugly check)
- Type safety of arguments is retained
- No downcasting
Adding a prefix namespaces command line options:

```c
SNESCreate(PETSC_COMM_WORLD, &snes);
/* Specify residual computation */
SNESSetOptionsPrefix(snes, "stokes_");
SNESSetFromOptions(snes);
SNES Solve(snes, PETSC_NULL, u);
```

- `-stokes_snes_type qn` changes the solver type,
whereas `-snes_type qn` does not
Solver use in SNES ex62

User provides a function to compute the residual:

```c
SNESCreate(PETSC_COMM_WORLD, &snes);
SNESSetFunction(snes, r, FormFunction, &user);
SNESSetFromOptions(snes);
SNESolve(snes, PETSC_NULL, u);
```

\[ r = F(u) \]

- User handles parallel communication
- User handles domain geometry and discretization
**DM** allows the user to compute only on a local patch:

```c
SNESCreate(PETSC_COMM_WORLD, &snes);
SNESSetDM(snes, user.dm);
SNESSetFunction(snes, r, SNESDMComputeFunction, &user);
SNESSetFromOptions(snes);
SNESolve(snes, PETSC_NULL, u);

DMSetLocalFunction(user.dm, (DMLocalFunction1) FormFunctionLocal);
```

- Code looks serial to the user
- PETSc handles global residual assembly
Optionally, the user can also provide a Jacobian:

```c
SNESCreate(PETSC_COMM_WORLD, &snes);
SNESSetDM(snes, user.dm);
SNESSetFunction(snes, r, SNESDMComputeFunction, &user);
SNESSetJacobian(snes, A, J, SNESDMComputeJacobian, &user);
SNESSetFromOptions(snes);
SNESolve(snes, PETSC_NULL, u);

DMSetLocalFunction(user.dm, (DMLocalFunction1) FormFunctionLocal);
DMSetLocalJacobian(user.dm, (DMLocalJacobian1) FormJacobianLocal);
```

**SNES ex62 allows both**

- finite difference (JFNK), and
- FEM action

**versions of the Jacobian.**
The **DM** also handles storage:

```c
CreateMesh(PETSC_COMM_WORLD, &user, &user.dm);
DMCreateGlobalVector(user.dm, &u);
VecDuplicate(u, &r);
DMCreateMatrix(user.dm, MATAIJ, &J);
```

- DM can create local and global vectors
- Matrices are correctly preallocated
Use `SNESSetFromOptions()` so that everything is set dynamically

- **Set the type**
  - Use `-snes_type` (or take the default)

- **Set the preconditioner**
  - Use `-npc_snes_type` (or take the default)

- **Override the tolerances**
  - Use `-snes_rtol` and `-snes_atol`

- **View the solver to make sure you have the one you expect**
  - Use `-snes_view`

- **For debugging, monitor the residual decrease**
  - Use `-snes_monitor`
  - Use `-ksp_monitor` to see the underlying linear solver
Programming with Options

**ex55**: Allen-Cahn problem in 2D
- constant mobility
- triangular elements

Geometric multigrid method for saddle point variational inequalities:

```
./ex55 -ksp_type fgmres -pc_type mg -mg_levels_ksp_type fgmres
        -mg_levels_pc_type fieldsplit -mg_levels_pc_fieldsplit_detect_saddle_point
        -mg_levels_pc_fieldsplit_type schur -da_grid_x 65 -da_grid_y 65
        -mg_levels_pc_fieldsplit_factorization_type full
        -mg_levels_pc_fieldsplit_schur_precondition user
        -mg_levels_fieldsplit_1_ksp_type gmres -mg_coarse_ksp_type preonly
        -mg_levels_fieldsplit_1_pc_type none -mg_coarse_pc_type svd
        -mg_levels_fieldsplit_0_ksp_type preonly
        -mg_levels_fieldsplit_0_pc_type sor -pc_mg_levels 5
        -mg_levels_fieldsplit_0_pc_sor_forward -pc_mg_galerkin
        -snes_vi_monitor -ksp_monitor_true_residual -snes_atol 1.e-11
        -mg_levels_ksp_monitor -mg_levels_fieldsplit_ksp_monitor
        -mg_levels_ksp_max_it 2 -mg_levels_fieldsplit_ksp_max_it 5
```
Programming with Options

**ex55: Allen-Cahn problem in 2D**

Run flexible GMRES with 5 levels of multigrid as the preconditioner.

```
./ex55 -ksp_type fgmres -pc_type mg -pc_mg_levels 5
   -da_grid_x 65 -da_grid_y 65
```

Use the Galerkin process to compute the coarse grid operators

```
-pc_mg_galerkin
```

Use SVD as the coarse grid saddle point solver

```
-mg_coarse_ksp_type preonly -mg_coarse_pc_type svd
```
Programming with Options

ex55: Allen-Cahn problem in 2D

Run flexible GMRES with 5 levels of multigrid as the preconditioner

```
./ex55 -ksp_type fgmres -pc_type mg -pc_mg_levels 5
   -da_grid_x 65 -da_grid_y 65

Use the Galerkin process to compute the coarse grid operators
- pc_mg_galerkin

Use SVD as the coarse grid saddle point solver
- mg_coarse_ksp_type preonly -mg_coarse_pc_type svd
```
ex55: Allen-Cahn problem in 2D

Run flexible GMRES with 5 levels of multigrid as the preconditioner

```
./ex55 -ksp_type fgmres -pc_type mg -pc_mg_levels 5
   -da_grid_x 65 -da_grid_y 65
```

Use the Galerkin process to compute the coarse grid operators

```
-pc_mg_galerkin
```

Use SVD as the coarse grid saddle point solver

```
-mg_coarse_ksp_type preonly -mg_coarse_pc_type svd
```
ex55: Allen-Cahn problem in 2D

Run flexible GMRES with 5 levels of multigrid as the preconditioner

```
ex55 -ksp_type fgmres -pc_type mg -pc_mg_levels 5
    -da_grid_x 65 -da_grid_y 65
```

Use the Galerkin process to compute the coarse grid operators

```
-pc_mg_galerkin
```

Use SVD as the coarse grid saddle point solver

```
-mg_coarse_ksp_type preonly -mg_coarse_pc_type svd
```
ex55: Allen-Cahn problem in 2D

Smoothers: Flexible GMRES (2 iterates) with a Schur complement PC

- mg_levels_ksp_type fgmres - mg_levels_pc_fieldsplit_detect_saddle_point
- mg_levels_ksp_max_it 2 - mg_levels_pc_type fieldsplit
- mg_levels_pc_fieldsplit_type schur
- mg_levels_pc_fieldsplit_factorization_type full
- mg_levels_pc_fieldsplit_schur_precondition diag

Schur complement solver: GMRES (5 iterates) with no preconditioner

- mg_levels_fieldsplit_1_ksp_type gmres
- mg_levels_fieldsplit_1_pc_type none - mg_levels_fieldsplit_ksp_max_it 5

Shur complement action: Use only the lower diagonal part of A00
- mg_levels_fieldsplit_0_ksp_type preonly
- mg_levels_fieldsplit_0_pc_type sor
- mg_levels_fieldsplit_0_pc_sor_forward
ex55: Allen-Cahn problem in 2D

**Smooother: Flexible GMRES (2 iterates) with a Schur complement PC**

```plaintext
-mg_levels_ksp_type fgmres
-mg_levels_pc_fieldsplit_detect_saddle_point
-mg_levels_ksp_max_it 2
-mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type schur
-mg_levels_pc_fieldsplit_factorization_type full
-mg_levels_pc_fieldsplit_schur_precondition diag
```

Schur complement solver: GMRES (5 iterates) with no preconditioner

```plaintext
-mg_levels_fieldsplit_1_ksp_type gmres
-mg_levels_fieldsplit_1_pc_type none
-mg_levels_fieldsplit_ksp_max_it 5
```

Shur complement action: Use only the lower diagonal part of A00

```plaintext
-mg_levels_fieldsplit_0_ksp_type preonly
-mg_levels_fieldsplit_0_pc_type sor
-mg_levels_fieldsplit_0_pc_sor_forward
```
ex55: Allen-Cahn problem in 2D

Smooother: Flexible GMRES (2 iterates) with a Schur complement PC

- mg_levels_ksp_type fgmres -mg_levels_pc_fieldsplit_detect_saddle_point
- mg_levels_ksp_max_it 2 -mg_levels_pc_type fieldsplit
- mg_levels_pc_fieldsplit_type schur
- mg_levels_pc_fieldsplit_factorization_type full
- mg_levels_pc_fieldsplit_schur_precondition diag

Schur complement solver: GMRES (5 iterates) with no preconditioner

- mg_levels_fieldsplit_1_ksp_type gmres
- mg_levels_fieldsplit_1_pc_type none -mg_levels_fieldsplit_ksp_max_it 5

Shur complement action: Use only the lower diagonal part of A00

- mg_levels_fieldsplit_0_ksp_type preonly
- mg_levels_fieldsplit_0_pc_type sor
- mg_levels_fieldsplit_0_pc_sor_forward
Programming with Options

**ex55**: Allen-Cahn problem in 2D

**Smoother**: Flexible GMRES (2 iterates) with a Schur complement PC

```
-mg_levels_ksp_type fgmres -mg_levels_pc_fieldsplit_detect_saddle_point
-mg_levels_ksp_max_it 2 -mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type schur
-mg_levels_pc_fieldsplit_factorization_type full
-mg_levels_pc_fieldsplit_schur_precondition diag
```

**Schur complement solver**: GMRES (5 iterates) with no preconditioner

```
-mg_levels_fieldsplit_1_ksp_type gmres
-mg_levels_fieldsplit_1_pc_type none -mg_levels_fieldsplit_ksp_max_it 5
```

**Shur complement action**: Use only the lower diagonal part of A00

```
-mg_levels_fieldsplit_0_ksp_type preonly
-mg_levels_fieldsplit_0_pc_type sor
-mg_levels_fieldsplit_0_pc_sor_forward
```
Complete table of solvers

1. Sequential LU
   - ILUDT (SPARSEKIT2, Yousef Saad, U of MN)
   - EUCLID & PILUT (Hypre, David Hysom, LLNL)
   - ESSL (IBM)
   - SuperLU (Jim Demmel and Sherry Li, LBNL)
   - Matlab
   - UMFPACK (Tim Davis, U. of Florida)
   - LUSOL (MINOS, Michael Saunders, Stanford)

2. Parallel LU
   - MUMPS (Patrick Amestoy, IRIT)
   - SPOOLES (Cleve Ashcroft, Boeing)
   - SuperLU_Dist (Jim Demmel and Sherry Li, LBNL)

3. Parallel Cholesky
   - DSCPACK (Padma Raghavan, Penn. State)
   - MUMPS (Patrick Amestoy, Toulouse)
   - CHOLMOD (Tim Davis, Florida)

4. XYTlib - parallel direct solver (Paul Fischer and Henry Tufo, ANL)
3rd Party Preconditioners in PETSc

Complete table of solvers

1. Parallel ICC
   - BlockSolve95 (Mark Jones and Paul Plassman, ANL)

2. Parallel ILU
   - PaStiX (Faverge Mathieu, INRIA)

3. Parallel Sparse Approximate Inverse
   - Parasails (Hypre, Edmund Chow, LLNL)
   - SPAI 3.0 (Marcus Grote and Barnard, NYU)

4. Sequential Algebraic Multigrid
   - RAMG (John Ruge and Klaus Steuben, GMD)
   - SAMG (Klaus Steuben, GMD)

5. Parallel Algebraic Multigrid
   - Prometheus (Mark Adams, PPPL)
   - BoomerAMG (Hypre, LLNL)
   - ML (Trilinos, Ray Tuminario and Jonathan Hu, SNL)
1. Getting Started with PETSc
2. SNES ex62
3. Solvers
4. FieldSplit
5. DM
# The Great Solver Schism: Monolithic or Split?

## Monolithic
- Direct solvers
- Coupled Schwarz
- Coupled Neumann-Neumann (need unassembled matrices)
- Coupled multigrid
- Need to understand local spectral and compatibility properties of the coupled system

## Split
- Physics-split Schwarz (based on relaxation)
- Physics-split Schur (based on factorization)
  - approximate commutators
  - SIMPLE, PCD, LSC
  - segregated smoothers
  - Augmented Lagrangian
  - “parabolization” for stiff waves
- Need to understand global coupling strengths

- Preferred data structures depend on which method is used.
- Interplay with geometric multigrid.
FieldSplit Preconditioner

- **Analysis**
  - Use **ISes** to define **fields**
  - Decouples **PC** from problem definition

- **Synthesis**
  - Additive, Multiplicative, Schur
  - Commutes with Multigrid
FieldSplit Options

Analysis
- `-pc_fieldsplit_<split num>_fields 2,1,5`
- `-pc_fieldsplit_detect_saddle_point`

Synthesis
- `-pc_fieldsplit_type`
- `-pc_fieldsplit_real_diagonal`
  Use diagonal blocks of operator to build PC

Schur complements
- `-pc_fieldsplit_schur_precondition <self,user,diag>`
  How to build preconditioner for $S$
- `-pc_fieldsplit_schur_factorization_type <diag,lower,upper,full>`
  Which off-diagonal parts of the block factorization to use
The common block preconditioners for Stokes require only options:

The Stokes System

\[
\begin{pmatrix}
A & B \\
B^T & 0
\end{pmatrix}
\]
The common block preconditioners for Stokes require only options:

- `pc_type fieldsplit`
- `pc_field_split_type additive`
- `fieldsplit_0_pc_type ml`
- `fieldsplit_0_ksp_type preonly`
- `fieldsplit_1_pc_type jacobi`
- `fieldsplit_1_ksp_type preonly`

\[
PC = \begin{pmatrix}
\hat{A} & 0 \\
0 & I
\end{pmatrix}
\]

The common block preconditioners for Stokes require only options:

- `pc_type fieldsplit`
- `pc_field_split_type multiplic`
- `fieldsplit_0_pc_type hypre`
- `fieldsplit_0_ksp_type preonly`
- `fieldsplit_1_pc_type jacobi`
- `fieldsplit_1_ksp_type preonly`

The common block preconditioners for Stokes require only options:

- `pc_type fieldsplit`
- `pc_field_split_type schur`
- `fieldsplit_0_pc_type gamg`
- `fieldsplit_0_ksp_type preonly`
- `fieldsplit_1_pc_type none`
- `fieldsplit_1_ksp_type minres`
- `pc_fieldsplit_schur_factorization_type diag`


The common block preconditioners for Stokes require only options:

-\texttt{-pc\_type fieldsplit}
-\texttt{-pc\_field\_split\_type schur}
-\texttt{-fieldsplit\_0\_pc\_type gamg}
-\texttt{-fieldsplit\_0\_ksp\_type preonly}
-\texttt{-fieldsplit\_1\_pc\_type none}
-\texttt{-fieldsplit\_1\_ksp\_type minres}
-\texttt{-pc\_fieldsplit\_schur\_factorization\_type lower}

\[
\begin{pmatrix}
\hat{A} & 0 \\
B^T & \hat{S}
\end{pmatrix}
\]

The common block preconditioners for Stokes require only options:

```
-pc_type fieldsplit
-pc_field_split_type schur
-fieldsplit_0_pc_type gamg
-fieldsplit_0_ksp_type preonly
-fieldsplit_1_pc_type none
-fieldsplit_1_ksp_type minres
-pc_fieldsplit_schur_factorization_type upper
```

\[
PC = \begin{pmatrix}
\hat{A} & B \\
0 & \hat{S}
\end{pmatrix}
\]

Stokes example

The common block preconditioners for Stokes require only options:

- `pc_type fieldsplit`
- `pc_field_split_type schur`
- `fieldsplit_0_pc_type gamg`
- `fieldsplit_0_ksp_type preonly`
- `fieldsplit_1_pc_type lsc`
- `fieldsplit_1_ksp_type minres`
- `pc_fieldsplit_schur_factorization_type upper`

Elman, Howle, Shadid, Shuttleworth, and Tuminaro, Block preconditioners based on approximate commutators, 2006.
The common block preconditioners for Stokes require only options:

- `pc_type fieldsplit`
- `pc_field_split_type schur`
- `pc_fieldsplit_schur_factorization_type full`

\[
\begin{pmatrix}
I & 0 \\
B^T A^{-1} & I
\end{pmatrix}
\begin{pmatrix}
\hat{A} & 0 \\
0 & \hat{S}
\end{pmatrix}
\begin{pmatrix}
I & A^{-1} B \\
0 & I
\end{pmatrix}
\]
FEM Setup

./bin/pythonscripts/PetscGenerateFEMQuadrature.py
  2 2 2 1 laplacian
  2 1 1 1 gradient
  src/snes/examples/tutorials/ex62.h
Jacobi

```
ex62
-run_type full -bc_type dirichlet -show_solution 0
-refinement_limit 0.00625 -interpolate 1
-snes_monitor_short -snes_converged_reason
  -snes_view
-ksp_gmres_restart 100 -ksp_rtol 1.0e-9
  -ksp_monitor_short
-pc_type jacobi
```
Block diagonal

ex62

- run_type full -bc_type dirichlet -show_solution 0
- refinement_limit 0.00625 -interpolate 1
- snes_monitor_short -snes_converged_reason
  - snes_view
- ksp_type fgmres -ksp_gmres_restart 100
  - ksp_rtol 1.0e-9 -ksp_monitor_short
- pc_type fieldsplit -pc_fieldsplit_type additive
- fieldsplit_velocity_pc_type lu
- fieldsplit_pressure_pc_type jacobi
Block triangular

ex62

- run_type full -bc_type dirichlet -show_solution 0
- refinement_limit 0.00625 -interpolate 1
- snes_monitor_short -snes_converged_reason
  - snes_view
- ksp_type fgmres -ksp_gmres_restart 100
  - ksp_rtol 1.0e-9 -ksp_monitor_short
- pc_type fieldsplit -pc_fieldsplit_type multiplicative
- fieldsplit_velocity_pc_type lu
- fieldsplit_pressure_pc_type jacobi
Diagonal Schur complement

ex62

- run_type full -bc_type dirichlet -show_solution 0
- refinement_limit 0.00625 -interpolate 1
- snes_monitor_short -snes_converged_reason
  - snes_view
- ksp_type fgmres -ksp_gmres_restart 100
  - ksp_rtol 1.0e-9 -ksp_monitor_short
- pc_type fieldsplit -pc_fieldsplit_type schur
  - pc_fieldsplit_schur_factorization_type diag
- fieldsplit_velocity_ksp_type gmres
  - fieldsplit_velocity_pc_type lu
- fieldsplit_pressure_ksp_rtol 1e-10
  - fieldsplit_pressure_pc_type jacobi
Upper triangular Schur complement

```bash
ex62
  -run_type full -bc_type dirichlet -show_solution 0
  -refinement_limit 0.00625 -interpolate 1
  -snes_monitor_short -snes_converged_reason
    -snes_view
  -ksp_type fgmres -ksp_gmres_restart 100
    -ksp_rtol 1.0e-9 -ksp_monitor_short
  -pc_type fieldsplit -pc_fieldsplit_type schur
    -pc_fieldsplit_schur_factorization_type upper
  -fieldsplit_velocity_ksp_type gmres
    -fieldsplit_velocity_pc_type lu
  -fieldsplit_pressure_ksp_rtol 1e-10
    -fieldsplit_pressure_pc_type jacobi
```
Lower triangular Schur complement

```
ex62
  -run_type full -bc_type dirichlet -show_solution 0
  -refinement_limit 0.00625 -interpolate 1
  -snes_monitor_short -snes_converged_reason
    -snes_view
  -ksp_type fgmres -ksp_gmres_restart 100
    -ksp_rtol 1.0e-9 -ksp_monitor_short
  -pc_type fieldsplit -pc_fieldsplit_type schur
    -pc_fieldsplit_schur_factorization_type lower
  -fieldsplit_velocity_ksp_type gmres
    -fieldsplit_velocity_pc_type lu
  -fieldsplit_pressure_ksp_rtol 1e-10
    -fieldsplit_pressure_pc_type jacobi
```
Full Schur complement

ex62

- run_type full -bc_type dirichlet -show_solution 0
- refinement_limit 0.00625 -interpolate 1
- snes_monitor_short -snes_converged_reason
  - snes_view
- ksp_type fgmres -ksp_gmres_restart 100
  - ksp_rtol 1.0e-9 -ksp_monitor_short
- pc_type fieldsplit -pc_fieldsplit_type schur
  - pc_fieldsplit_schur_factorization_type full
- fieldsplit_velocity_ksp_type gmres
  - fieldsplit_velocity_pc_type lu
- fieldsplit_pressure_ksp_rtol 1e-10
  - fieldsplit_pressure_pc_type jacobi
Outline

1. Getting Started with PETSc
2. SNES ex62
3. Solvers
4. FieldSplit
5. DM
   - PetscSection and DMComplex
   - Vec and Mat Particulars
Outline

5 DM
- PetscSection and DMComplex
- Vec and Mat Particulars
What does a DM do?

- **Problem Definition**
  - Discretization/Dof mapping (**PetscSection**)
  - Residual calculation

- **Decomposition**
  - Partitioning, **DMCreateSubDM()**
  - Vec and Mat creation
  - Global ⇔ Local mapping

- **Hierarchy**
  - **DMCoarsen()** and **DMRefine()**
  - **DMInterpolate()** and **DMRestrict()**
  - Hooks for resolution-dependent data
PetscSection
What Is It?

Similar to PetscLayout, maps point \( \rightarrow \) (size, offset)
- Processes are replaced by points
  - Also what we might use for multicore PetscLayout
- Boundary conditions are just another PetscSection
  - Map points to number of constrained dofs
  - Offsets into integer array of constrained local dofs
- Fields are just another PetscSection
  - Map points to number of field dofs
  - Offsets into array with all fields
- Usable by all DM subclasses
  - Structured grids with DMDA
  - Unstructured grids with DMComplex
PETSc Solvers only understand Integers

Decouples Mesh From Discretization
- Mesh does not need to know how dofs are generated, just how many are attached to each point.
- It does not matter whether you use FD, FV, FEM, etc.

Decouples Mesh from Solver
- Solver gets the data layout and partitioning from Vec and Mat, nothing else from the mesh.
- Solver gets restriction/interpolation matrices from DM.

Decouples Discretization from Solver
- Solver only gets the field division, nothing else from discretization.
High Level Interface

DMComplexCreateSection(  
    DM dm, PetscInt dim, PetscInt numFields,  
    PetscInt numComp[], PetscInt numDof[],  
    PetscInt numBC, PetscInt bcField[], IS bcPoints[],  
    PetscSection *section);  

<table>
<thead>
<tr>
<th>Discretization</th>
<th>Dof/Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1 - P_0$</td>
<td>[2 0 0 0</td>
</tr>
<tr>
<td>$Q_2 - Q_1$</td>
<td>[2 2 0 0</td>
</tr>
<tr>
<td>$Q_2 - P_1^{\text{disc}}$</td>
<td>[2 2 0 0</td>
</tr>
</tbody>
</table>
PetscSection and DMComplex

How Do I Build One?

**Low Level Interface**

```c
PetscSectionCreate(PETSC_COMM_WORLD, &s);
PetscSectionSetNumFields(s, 2);
PetscSectionSetFieldComponents(s, 0, 3);
PetscSectionSetFieldComponents(s, 1, 1);
PetscSectionSetChart(s, cStart, vEnd);
for(PetscInt v = vStart; v < vEnd; ++v) {
    PetscSectionSetDof(s, v, 3);
    PetscSectionSetFieldDof(s, v, 0, 3);
}
for(PetscInt c = cStart; c < cEnd; ++c) {
    PetscSectionSetDof(s, c, 1);
    PetscSectionSetFieldDof(s, c, 1, 1);
}
PetscSectionSetUp(s);
```
**DMComplex** stands for a DM modeling a CW Complex

- Handles any kind of mesh
  - Simplicial
  - Hex
  - Hybrid
  - Non-manifold

- Small interface
  - Simple to input a mesh using the API

- Accepts mesh generator input
  - ExodusII, Triangle, TetGen, LaGriT, Cubit
The operations used in SNES ex62 get and set values from a Vec, organized by the DM and PetscSection

DMComplexVecGetClosure(
    DM dm, PetscSection section, Vec v, PetscInt point,
    PetscInt *csize, const PetscScalar *values[])

- Element vector on cell
- Coordinates on cell vertices

Used in FormFunctionLocal(),

for(c = cStart; c < cEnd; ++c) {
    const PetscScalar *x;

    DMComplexVecGetClosure(dm, PETSC_NULL, X, c, PETSC_NULL, &x);
    for(PetscInt i = 0; i < cellDof; ++i) {
        u[c*cellDof+i] = x[i];
    }
    DMComplexVecRestoreClosure(dm, PETSC_NULL, X, c, PETSC_NULL, &x);
}
The operations used in SNES ex62 get and set values from a Vec, organized by the DM and PetscSection

DMComplexVecGetClosure(
    DM dm, PetscSection section, Vec v, PetscInt point,
    PetscInt *csize, const PetscScalar *values[])

- Element vector on cell
- Coordinates on cell vertices

Used in FormFunctionLocal(),

```c
for(c = cStart; c < cEnd; ++c) {
    const PetscScalar *x;

    DMComplexVecGetClosure(dm, PETSC_NULL, X, c, PETSC_NULL, &x);
    for(PetscInt i = 0; i < cellDof; ++i) {
        u[c*cellDof+i] = x[i];
    }
    DMComplexVecRestoreClosure(dm, PETSC_NULL, X, c, PETSC_NULL, &x);
}
```
The operations used in SNES ex62 get and set values from a Vec, organized by the **DM** and **PetscSection**

DMComplexVecSetClosure(  
    DM dm, PetscSection section, Vec v, PetscInt point,  
    const PetscScalar values[], InsertMode mode)

DMComplexMatSetClosure(  
    DM dm, PetscSection section, PetscSection globalSection, Mat A, PetscInt point,  
    PetscScalar values[], InsertMode mode)

- **Element vector and matrix on cell**

**Used in** FormJacobianLocal(),

```c
for(c = cStart; c < cEnd; ++c) {
    DMComplexMatSetClosure(dm, PETSC_NULL, PETSC_NULL, JacP, c,  
        &elemMat[c*cellDof*cellDof], ADD_VALUES);
}
```
The operations used in SNES ex62 get and set values from a `Vec`, organized by the `DM` and `PetscSection`.

```c
DMComplexVecSetClosure(
    DM dm, PetscSection section, Vec v, PetscInt point,
    const PetscScalar values[], InsertMode mode)
```

```c
DMComplexMatSetClosure(
    DM dm, PetscSection section, PetscSection globalSection, Mat A, PetscInt point,
    PetscScalar values[], InsertMode mode)
```

- Element vector and matrix on cell

Used in `FormJacobianLocal()`,

```c
for(c = cStart; c < cEnd; ++c) {
    DMComplexMatSetClosure(dm, PETSC_NULL, PETSC_NULL, JacP, c,
                            &elemMat[c*cellDof*cellDof], ADD_VALUES);
}
```
The functions above are built upon

DMComplexGetTransitiveClosure(
    DM dm, PetscInt p, PetscBool useCone,
    PetscInt *numPoints, PetscInt *points[])

- Returns points and orientations
- Iterate over points to stack up the data in the array
$P_2 - P_1$ Stokes Example

Naively, we have

$$\text{cl}(\text{cell}) = \begin{bmatrix} f e_0 e_1 e_2 v_0 v_1 v_2 \end{bmatrix}$$

$$\text{x}(\text{cell}) = \begin{bmatrix} u_{e_0} v_{e_0} u_{e_1} v_{e_1} u_{e_2} v_{e_2} \\
 u_{v_0} v_{v_0} p_{v_0} u_{v_1} v_{v_1} p_{v_1} u_{v_2} v_{v_2} p_{v_2} \end{bmatrix}$$
$P_2 - P_1$ Stokes Example

We reorder so that fields are contiguous

\[
\chi'(\text{cell}) = \begin{bmatrix}
    u_{e_0} & v_{e_0} & u_{e_1} & v_{e_1} & u_{e_2} & v_{e_2} \\
    u_{v_0} & v_{v_0} & u_{v_1} & v_{v_1} & u_{v_2} & v_{v_2} \\
    p_{v_0} & p_{v_1} & p_{v_2}
\end{bmatrix}
\]
DMComplex
Basic Operations

- Cone
  - edge $\rightarrow$ endpoints
  - cell $\rightarrow$ faces

- Support
  - vertex $\rightarrow$ edges
  - face $\rightarrow$ cells

- Transitive Closure
  - cell $\rightarrow$ faces, edges, vertices

- Meet
  - cells $\rightarrow$ shared face

- Join
  - vertices $\rightarrow$ shared cell
Outline

5 DM

- PetscSection and DMComplex
- Vec and Mat Particulars
### Selected Vector Operations

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>VecAXPY(Vec y, PetscScalar a, Vec x)</td>
<td>( y = y + a \times x )</td>
</tr>
<tr>
<td>VecAYPX(Vec y, PetscScalar a, Vec x)</td>
<td>( y = x + a \times y )</td>
</tr>
<tr>
<td>VecWAYPX(Vec w, PetscScalar a, Vec x, Vec y)</td>
<td>( w = y + a \times x )</td>
</tr>
<tr>
<td>VecScale(Vec x, PetscScalar a)</td>
<td>( x = a \times x )</td>
</tr>
<tr>
<td>VecCopy(Vec y, Vec x)</td>
<td>( y = x )</td>
</tr>
<tr>
<td>VecPointwiseMult(Vec w, Vec x, Vec y)</td>
<td>( w_i = x_i \times y_i )</td>
</tr>
<tr>
<td>VecMax(Vec x, PetscInt *idx, PetscScalar *r)</td>
<td>( r = \max r_i )</td>
</tr>
<tr>
<td>VecShift(Vec x, PetscScalar r)</td>
<td>( x_i = x_i + r )</td>
</tr>
<tr>
<td>VecAbs(Vec x)</td>
<td>( x_i =</td>
</tr>
<tr>
<td>VecNorm(Vec x, NormType type, PetscReal *r)</td>
<td>( r = |x| )</td>
</tr>
</tbody>
</table>
It is sometimes more efficient to directly access local storage of a `Vec`.

- PETSc allows you to access the local storage with
  - `VecGetArray(Vec, double **[])`

- You must return the array to PETSc when you finish
  - `VecRestoreArray(Vec, double **[])`

- Allows PETSc to handle data structure conversions
  - Commonly, these routines are fast and do not involve a copy
VecGetArray in C

```c
Vec v;
PetscScalar *array;
PetscInt n, i;

VecGetArray(v, &array);
VecGetLocalSize(v, &n);
PetscSynchronizedPrintf(PETSC_COMM_WORLD,
    "First element of local array is %f\n", array[0]);
PetscSynchronizedFlush(PETSC_COMM_WORLD);
for(i = 0; i < n; ++i) {
    array[i] += (PetscScalar) rank;
}
VecRestoreArray(v, &array);
```
VecGetArray in F77

```c
#include "finclude/petsc.h"

Vec v;
PetscScalar array(1)
PetscOffset offset
PetscInt n, i
PetscErrorCode ierr

call VecGetArray(v, array, offset, ierr)
call VecGetLocalSize(v, n, ierr)
do i=1,n
   array(i+offset) = array(i+offset) + rank
end do
call VecRestoreArray(v, array, offset, ierr)
```
include "finclude/petsc.h90"

Vec v;
PetscScalar pointer :: array(:)
PetscInt n, i
PetscErrorCode ierr

call VecGetArrayF90(v, array, ierr)
call VecGetLocalSize(v, n, ierr)
do i=1,n
   array(i) = array(i) + rank
end do
call VecRestoreArrayF90(v, array, ierr)
with v as a:
    for i in range(len(a)):
        a[i] = 5.0*i
DMDAVecGetArray in C

```c
DM da;
Vec v;
DMDALocalInfo *info;
PetscScalar **array;

DMDAVecGetArray(da, v, &array);
for (j = info->ys; j < info->ys+info->ym; ++j) {
    for (i = info->xs; i < info->xs+info->xm; ++i) {
        u = x[j][i];
        uxx = (2.0 *u - x[j][i-1] - x[j][i+1])*hydhx;
        uyy = (2.0 *u - x[j-1][i] - x[j+1][i])*hxdhy;
        f[j][i] = uxx + uyy;
    }
}
DMDAVecRestoreArray(da, v, &array);
```
PETSc can help you:

- easily construct a code to test your ideas
  - Lots of code construction, management, and debugging tools
- scale an existing code to large or distributed machines
  - Using `FormFunctionLocal()` and scalable linear algebra
- incorporate more scalable or higher performance algorithms
  - Such as domain decomposition, fieldsplit, and multigrid
- tune your code to new architectures
  - Using profiling tools and specialized implementations
PETSc can help you:

- easily construct a code to test your ideas
  - Lots of code construction, management, and debugging tools

- scale an existing code to large or distributed machines
  - Using `FormFunctionLocal()` and scalable linear algebra

- incorporate more scalable or higher performance algorithms
  - Such as domain decomposition, fieldsplit, and multigrid

- tune your code to new architectures
  - Using profiling tools and specialized implementations
Conclusions

PETSc can help you:

- easily construct a code to test your ideas
  - Lots of code construction, management, and debugging tools

- scale an existing code to large or distributed machines
  - Using `FormFunctionLocal()` and scalable linear algebra

- incorporate more scalable or higher performance algorithms
  - Such as domain decomposition, fieldsplit, and multigrid

- tune your code to new architectures
  - Using profiling tools and specialized implementations
PETSc can help you:

- easily construct a code to test your ideas
  - Lots of code construction, management, and debugging tools
- scale an existing code to large or distributed machines
  - Using `FormFunctionLocal()` and scalable linear algebra
- incorporate more scalable or higher performance algorithms
  - Such as domain decomposition, fieldsplit, and multigrid
- tune your code to new architectures
  - Using profiling tools and specialized implementations
PETSc can help you:
- easily construct a code to test your ideas
  - Lots of code construction, management, and debugging tools
- scale an existing code to large or distributed machines
  - Using `FormFunctionLocal()` and scalable linear algebra
- incorporate more scalable or higher performance algorithms
  - Such as domain decomposition, fieldsplit, and multigrid
- tune your code to new architectures
  - Using profiling tools and specialized implementations