The Portable Extensible Toolkit for Scientific Computing

Matthew Knepley

Mathematics and Computer Science Division
Argonne National Laboratory

Computation Institute
University of Chicago

PETSc Hands-On
13th Workshop on the DOE ACTS Collection
LBNL, Berkeley, CA August 14–17, 2012

ARGONNE NATIONAL LABORATORY

We want to enable users to,

assess solver performance,

and optimize solvers
for particular problems.

Main Point

We want to enable users to,

assess solver performance,

and optimize solvers
for particular problems.

Main Point

We want to enable users to,

assess solver performance,

and optimize solvers
for particular problems.

Outline

Controlling PETSc

All of PETSc can be controlled by options

`-ksp_type gmres`

`-start_in_debugger`

All objects can have a prefix, `-velocity_pc_type jacobi`

All PETSc options can be namespaced

`-sub_ksp_type bicg`

`-fieldsplit_1_pc_type jacobi`

Controlling PETSc

All of PETSc can be controlled by options

`-ksp_type gmres`

`-start_in_debugger`

All objects can have a prefix, `-velocity_pc_type jacobi`

All PETSc options can be namespaced

`-sub_ksp_type bicg`

`-fieldsplit_1_pc_type jacobi`

Examples

We will illustrate options using

PETSc SNES ex5, located at

`$PETSC_DIR/src/snes/examples/tutorials/ex5.c`

and

PETSc SNES ex62, located at

`$PETSC_DIR/src/snes/examples/tutorials/ex62.c`

Outline

1. Controlling the Solver
2. Where do I begin?
3. How do I improve?
4. Debugging
5. Examples

Nonlinear Systems

I am not going to discuss nonlinear systems today,

however if Newton is failing,

contact

petsc-maint@mcs.anl.gov

Nonlinear Systems

I am not going to discuss nonlinear systems today,

however if Newton is failing,

contact

petsc-maint@mcs.anl.gov

What is a Krylov solver?

A Krylov solver builds a small model of a linear operator A, using a subspace defined by

$$\mathcal{K}(A, r) = \mathrm{span}\{r, Ar, A^2 r, A^3 r, \ldots\}$$

where r is the initial residual.

The small system is sovled directly, and the solution is projected back to the original space.

What is a Krylov solver?

A Krylov solver builds a small model of a linear operator A, using a subspace defined by

$$\mathcal{K}(A, r) = \mathrm{span}\{r, Ar, A^2 r, A^3 r, \ldots\}$$

where r is the initial residual.

The small system is sovled directly, and the solution is projected back to the original space.

What Should I Know About Krylov Solvers?

- They can handle *low-mode* errors
- They need preconditioners
- They do a lot of inner products

What is a Preconditioner?

A preconditioner M changes a linear system,

$$M^{-1}Ax = M^{-1}b$$

so that the effective operator is $M^{-1}A$, which is hopefully better for Krylov methods.

- Preconditioner should be inexpensive
- Preconditioner should accelerate convergence

What is a Preconditioner?

A preconditioner M changes a linear system,

$$M^{-1}Ax = M^{-1}b$$

so that the effective operator is $M^{-1}A$, which is hopefully better for Krylov methods.

- Preconditioner should be inexpensive
- Preconditioner should accelerate convergence

What is a Preconditioner?

A preconditioner M changes a linear system,

$$M^{-1}Ax = M^{-1}b$$

so that the effective operator is $M^{-1}A$, which is hopefully better for Krylov methods.

- Preconditioner should be inexpensive
- Preconditioner should accelerate convergence

Always start with LU

Always, always start with LU:

- No iterative tolerance
- (Almost) no condition number dependence
- Check for accidental singularity

In parallel, you need a 3rd party package

- MUMPS (`--download-mumps`)
- SuperLU (`--download-superlu_dist`)

Always start with LU

Always, always start with LU:

- No iterative tolerance
- (Almost) no condition number dependence
- Check for accidental singularity

In parallel, you need a 3rd party package

- MUMPS (`--download-mumps`)
- SuperLU (`--download-superlu_dist`)

What if LU fails?

LU will fail for

- Singular problems
- Saddle-point problems

For saddles use PC_FIELDSPLIT

- Separately solves each field
- Decomposition is automatic in PyLith
- Autodetect with -pc_fieldsplit_detect_saddle_point
- Exact with full Schur complement solve

What if LU fails?

LU will fail for

- Singular problems
- Saddle-point problems

For saddles use `PC_FIELDSPLIT`

- Separately solves each field
- Decomposition is automatic in PyLith
- Autodetect with `-pc_fieldsplit_detect_saddle_point`
- Exact with full Schur complement solve

Outline

Outline

3. How do I improve?
   - Look at what you have
   - Back off in steps

What solver did you use?

Use `-snes_view` or `-ksp_view` to output a description of the solver:

```
KSP Object:         (fieldsplit_0_)         1 MPI processes
  type: fgmres
    GMRES: restart=100, using Classical (unmodified) Gram-
      Schmidt Orthogonalization with no iterative refinemen
    GMRES: happy breakdown tolerance 1e-30
  maximum iterations=1, initial guess is zero
  tolerances:  relative=1e-09, absolute=1e-50,
    divergence=10000
  right preconditioning
  has attached null space
  using UNPRECONDITIONED norm type for convergence test
```

What did the convergence look like?

Use `-snes_monitor` and `-ksp_monitor`, or `-log_summary`:

What did the convergence look like?

Use `-snes_monitor` and `-ksp_monitor`, or `-log_summary`:

```
0 SNES Function norm 0.207564
1 SNES Function norm 0.0148968
2 SNES Function norm 0.000113968
3 SNES Function norm 6.9256e-09
4 SNES Function norm < 1.e-11
```

What did the convergence look like?

Use `-snes_monitor` and `-ksp_monitor`, or `-log_summary`:

```
0 KSP Residual norm 1.61409
    Residual norms for mg_levels_1_ solve.
    0 KSP Residual norm 0.213376
    1 KSP Residual norm 0.0192085
  Residual norms for mg_levels_2_ solve.
  0 KSP Residual norm 0.223226
  1 KSP Residual norm 0.0219992
    Residual norms for mg_levels_1_ solve.
    0 KSP Residual norm 0.0248252
    1 KSP Residual norm 0.0153432
  Residual norms for mg_levels_2_ solve.
  0 KSP Residual norm 0.0124024
  1 KSP Residual norm 0.0018736
1 KSP Residual norm 0.02282
```

What did the convergence look like?

Use `-snes_monitor` and `-ksp_monitor`, or `-log_summary`:

```
Event          Count       Time (sec)        Flops         Total
             Max Ratio  Max        Ratio   Max   Ratio Mflop/s
KSPSetUp        12 1.0 3.6259e-03 1.0 0.00e+00 0.0       0
KSPSolve         3 1.0 4.8937e-01 1.0 8.93e+05 1.0       2
SNESSolve        1 1.0 4.9477e-01 1.0 9.22e+05 1.0       2
```

Look at timing

Use `-log_summary`:

```
Event            Time (sec)        Flops        --- Global ---   Total
               Max       Ratio    Max   Ratio %T %f %M %L %R   MF/s
VecMDot     1.8904e-03 1.0 2.49e+04 1.0  0  3  0  0  0     13
MatMult     2.1026e-03 1.0 2.65e+05 1.0  0 29  0  0  0    126
PCApply     4.6001e-01 1.0 7.78e+05 1.0 58 84  0  0 64      2
KSPSetUp    3.6259e-03 1.0 0.00e+00 0.0  0  0  0  0  4      0
KSPSolve    4.8937e-01 1.0 8.93e+05 1.0 61 97  0  0 90      2
SNESSolve 4.9477e-01 1.0 9.22e+05 1.0 62100  0  0 92      2
```

Use `-log_summary_python` to get this information as a Python module

Look at timing

Use `-log_summary`:

Event	Time (sec)		Flops		--- Global ---					Total
	Max	Ratio	Max	Ratio	%T	%f	%M	%L	%R	MF/s
VecMDot	1.8904e-03	1.0	2.49e+04	1.0	0	3	0	0	0	13
MatMult	2.1026e-03	1.0	2.65e+05	1.0	0	29	0	0	0	126
PCApply	4.6001e-01	1.0	7.78e+05	1.0	58	84	0	0	64	2
KSPSetUp	3.6259e-03	1.0	0.00e+00	0.0	0	0	0	0	4	0
KSPSolve	4.8937e-01	1.0	8.93e+05	1.0	61	97	0	0	90	2
SNESSolve	4.9477e-01	1.0	9.22e+05	1.0	62	100	0	0	92	2

Use `-log_summary_python` to get this information as a Python module

Outline

3. How do I improve?
   - Look at what you have
   - Back off in steps

Weaken the KSP

GMRES $\Longrightarrow$ BiCGStab

- `-ksp_type bcgs`
- Less storage
- Fewer dot products (less work)
- Variants `-ksp_type bcgsl` and `-ksp_type ibcgs`

Complete Table of Solvers and Preconditioners

Weaken the PC

LU $\Longrightarrow$ ILU

- `-pc_type ilu`
- Less storage and work

In parallel,

- Hypre `-pc_type hypre -pc_hypre_type euclid`
- Block-Jacobi `-pc_type bjacobi -sub_pc_type ilu`
- Additive Schwarz `-pc_type asm -sub_pc_type ilu`

Default for MG smoother is Chebychev/SOR(2)

Weaken the PC

LU $\implies$ ILU

- `-pc_type ilu`
- Less storage and work

In parallel,

- Hypre `-pc_type hypre -pc_hypre_type euclid`
- Block-Jacobi `-pc_type bjacobi -sub_pc_type ilu`
- Additive Schwarz `-pc_type asm -sub_pc_type ilu`

Default for MG smoother is Chebychev/SOR(2)

Weaken the PC

LU $\Longrightarrow$ ILU

- `-pc_type ilu`
- Less storage and work

In parallel,

- Hypre `-pc_type hypre -pc_hypre_type euclid`
- Block-Jacobi `-pc_type bjacobi -sub_pc_type ilu`
- Additive Schwarz `-pc_type asm -sub_pc_type ilu`

Default for MG smoother is Chebychev/SOR(2)

Algebraic Multigrid (AMG)

- Can solve elliptic problems
  - Laplace, elasticity, Stokes
- Works for unstructured meshes
- `-pc_type gamg`, `-pc_type ml`, `-pc_type hypre -pc_hypre_type boomeramg`
- CRUCIAL to have an accurate near-null space
  - MatSetNearNullSpace()
  - PyLith provides this automatically
- Use `-pc_mg_log` to put timing in its own log stage

PC_FieldSplit

- Separate solves for block operators
  - Physical insight for subsystems
  - Have optimial PCs for simpler equations
  - Suboptions `fs_fieldsplit_0_*`
- Flexibly combine subsolves
  - Jacobi: `fs_pc_fieldsplit_type = additive`
  - Gauss-Siedel: `fs_pc_fieldsplit_type = multiplicative`
  - Schur complement: `fs_pc_fieldsplit_type = schur`

Stokes example

The common block preconditioners for Stokes require only options:

The Stokes System $\begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix}$

Stokes example

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

$$\begin{pmatrix} \hat{A} & 0 \\ 0 & I \end{pmatrix}$$

Cohouet & Chabard, Some fast 3D finite element solvers for the generalized Stokes problem, 1988.

Stokes example

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
```

PC

$$\begin{pmatrix} \hat{A} & B \\ 0 & I \end{pmatrix}$$

Elman, Multigrid and Krylov subspace methods for the discrete Stokes equations, 1994.

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 none
-fieldsplit_1_ksp_type minres
```

PC

$$\begin{pmatrix} \hat{A} & 0 \\ 0 & -\hat{S} \end{pmatrix}$$

```
-pc_fieldsplit_schur_factorization_type diag
```

May and Moresi, Preconditioned iterative methods for Stokes flow problems arising in computational geodynamics, 2007.
Olshanskii, Peters, and Reusken, Uniform preconditioners for a parameter dependent saddle point problem with application to generalized Stokes interface equations, 2006.

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 none
-fieldsplit_1_ksp_type minres
```

PC

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

```
-pc_fieldsplit_schur_factorization_type lower
```

May and Moresi, Preconditioned iterative methods for Stokes flow problems arising in computational geodynamics, 2007.

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 none
-fieldsplit_1_ksp_type minres
```

PC

$$\begin{pmatrix} \hat{A} & B \\ 0 & \hat{S} \end{pmatrix}$$

```
-pc_fieldsplit_schur_factorization_type upper
```

May and Moresi, Preconditioned iterative methods for Stokes flow problems arising in computational geodynamics, 2007.

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

$$\begin{pmatrix} \hat{A} & B \\ 0 & \hat{S}_{\mathrm{LSC}} \end{pmatrix}$$

```
-pc_fieldsplit_schur_factorization_type upper
```

May and Moresi, Preconditioned iterative methods for Stokes flow problems arising in computational geodynamics, 2007.
Kay, Loghin and Wathen, A Preconditioner for the Steady-State N-S Equations, 2002.
Elman, Howle, Shadid, Shuttleworth, and Tuminaro, Block preconditioners based on approximate commutators, 2006.

Stokes example

The common block preconditioners for Stokes require only options:

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

PC

$$\begin{pmatrix} I & 0 \\ B^T A^{-1} & I \end{pmatrix} \begin{pmatrix} \hat{A} & 0 \\ 0 & \hat{S} \end{pmatrix} \begin{pmatrix} I & A^{-1}B \\ 0 & I \end{pmatrix}$$

Why us FGMRES?

Flexible GMRES (FGMRES) allows a different preconditioner at each step:

- Takes twice the memory
- Needed for iterative PCs
- Avoided sometimes with a careful PC choice

Outline

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

- Put a breakpoint in `PetscError()` to catch errors as they occur
- PETSc tracks memory overwrites at both ends 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
  - Print unfreed memory on `PetscFinalize()` with `-malloc_dump`
- Simply the best tool today is valgrind
  - It checks memory access, cache performance, memory usage, etc.
  - http://www.valgrind.org
  - Need `-trace-children=yes` when running under MPI

Outline

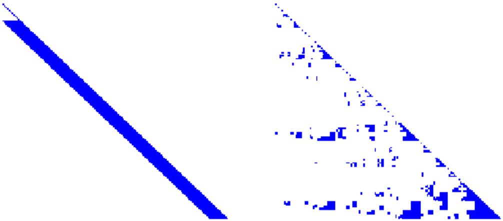
Try it out

```
$ cd $PETSC_DIR/src/snes/examples/tutorials && make ex5
```

- `$ ./ex5 -da_grid_x 10 -da_grid_y 10 -par 6.7 -snes_monitor -{ksp,snes}_converged_reason -snes_view`
- `$ ./ex5 -da_grid_x 10 -da_grid_y 10 -par 6.7 -snes_monitor -{ksp,snes}_converged_reason -snes_view -mat_view_draw -draw_pause 0.5`
- `$ ./ex5 -da_grid_x 10 -da_grid_y 10 -par 6.7 -snes_monitor -{ksp,snes}_converged_reason -snes_view -mat_view_draw -draw_pause 0.5 -pc_type lu -pc_factor_mat_ordering_type natural`
- Use `-help` to find other ordering types

Sample output

```
0 SNES Function norm 1.139460779565e+00
Linear solve converged due to CONVERGED_RTOL iterations 1
1 SNES Function norm 4.144493702305e-02
Linear solve converged due to CONVERGED_RTOL iterations 1
2 SNES Function norm 6.309075568032e-03
Linear solve converged due to CONVERGED_RTOL iterations 1
3 SNES Function norm 3.359792279909e-04
Linear solve converged due to CONVERGED_RTOL iterations 1
4 SNES Function norm 1.198827244256e-06
Linear solve converged due to CONVERGED_RTOL iterations 1
5 SNES Function norm 1.545029314765e-11
```



Sample output (SNES and KSP)

```
SNES Object: 1 MPI processes
  type: ls
    line search variant: CUBIC
    alpha=1.000000000000e-04, maxstep=1.000000000000e+08, minlambd
    damping factor=1.000000000000e+00
  maximum iterations=50, maximum function evaluations=10000
  tolerances: relative=1e-08, absolute=1e-50, solution=1e-08
  total number of linear solver iterations=5
  total number of function evaluations=6
  KSP Object:   1 MPI processes
    type: gmres
      GMRES: restart=30, using Classical (unmodified) Gram-Schmidt
      GMRES: happy breakdown tolerance 1e-30
    maximum iterations=10000, initial guess is zero
    tolerances:  relative=1e-05, absolute=1e-50, divergence=10000
    left preconditioning
    using PRECONDITIONED norm type for convergence test
```

Sample output (PC and Mat)

```
PC Object:   1 MPI processes
  type: lu
    LU: out-of-place factorization
    tolerance for zero pivot 2.22045e-14
    matrix ordering: nd
    factor fill ratio given 5, needed 2.95217
      Factored matrix follows:
        Matrix Object:           1 MPI processes
          type: seqaij
          rows=100, cols=100
          package used to perform factorization: petsc
          total: nonzeros=1358, allocated nonzeros=1358
          total number of mallocs used during MatSetValues calls
            not using I-node routines
  linear system matrix = precond matrix:
  Matrix Object:     1 MPI processes
    type: seqaij
    rows=100, cols=100
    total: nonzeros=460, allocated nonzeros=460
    total number of mallocs used during MatSetValues calls =0
      not using I-node routines
```

In parallel

- `$ mpiexec -n 4 ./ex5 -da_grid_x 10 -da_grid_y 10 -par 6.7 -snes_monitor -{ksp,snes}_converged_reason -snes_view -sub_pc_type lu`
- How does the performance change as you
  - vary the number of processes (up to 32 or 64)?
  - increase the problem size?
  - try inexact subdomain solve?
  - try overlapping method: `-pc_type asm -pc_asm_overlap 2`
  - simulate a big machine: `-pc_asm_blocks 512`
  - change the Krylov method: `-ksp_type ibcgs`
  - use algebraic multigrid: `-pc_type hypre`
  - use smoothed aggregation multigrid: `-pc_type ml`

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 multiplicati
  -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

```
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
```

User Projects

Create a new code based upon SNES Example 5.

1. Create a new directory
   - `mkdir -p /home/knepley/proj/newsim/src`
2. Copy the source
   - `cp ex5.c /home/knepley/proj/newsim/src`
   - Add `myStuff.c` and `myStuff2.F`
3. Create a PETSc makefile

```
bin/ex5:  src/ex5.o src/myStuff.o src/myStuff2.o
  ${CLINKER} -o $@ $^ ${PETSC_SNES_LIB}
include ${PETSC_DIR}/conf/variables
include ${PETSC_DIR}/conf/rules
```

To get the project ready-made

`hg clone http://petsc.cs.iit.edu/petsc/tutorials/SimpleTutorial newsim`