

# Using PETSc Solvers in PyLith

Matthew Knepley, Brad Aagaard, and Charles Williams

Computation Institute  
University of Chicago

Department of Molecular Biology and Physiology  
Rush University Medical Center

CDM 2013

Golden, CO June 23–28, 2013



**We want to enable users to**  
assess solver performance,  
and optimize solvers  
for particular problems.

We want to enable users to  
assess solver performance,  
and optimize solvers  
for particular problems.

We want to enable users to  
assess solver performance,  
and optimize solvers  
for particular problems.

# Outline

- 1 Controlling the Solver
- 2 Where do I begin?
- 3 How do I improve?
- 4 Can We Do It?
- 5 Nonlinear Systems

# 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 given to PyLith

```
--petsc.ksp_type=gmres
```

```
--petsc.start_in_debugger
```

# 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 given to PyLith

```
--petsc.ksp_type=gmres
```

```
--petsc.start_in_debugger
```

# Examples

We will illustrate options using

PETSc SNES **ex19**, located at

```
$PETSC_DIR/src/snes/examples/tutorials/ex19.c
```

and

PyLith Example **hex8**, located at

```
$PYLITH_DIR/examples/3d/hex8/
```



# Outline

- 1 Controlling the Solver
- 2 Where do I begin?**
- 3 How do I improve?
- 4 Can We Do It?
- 5 Nonlinear Systems

# What is a Krylov solver?

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

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

where  $r$  is the initial residual.

The small system is solved 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

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

where  $r$  is the initial residual.

The small system is solved 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

- 1 Controlling the Solver
- 2 Where do I begin?
- 3 How do I improve?**
  - Look at what you have
  - Back off in steps
- 4 Can We Do It?
- 5 Nonlinear Systems

# 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
SNESolve	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 MF/s
	Max	Ratio	Max	Ratio	%T	%f	%M	%L	%R	
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
SNESolve	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

# Look at timing

Use `-log_summary`:

Event	Time (sec)		Flops		--- Global ---					Total MF/s
	Max	Ratio	Max	Ratio	%T	%f	%M	%L	%R	
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
SNESolve	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  $\implies$  BiCGStab

- `-ksp_type bcgs`
- Less storage
- Fewer dot products (less work)
- Variants `-ksp_type bcgs1` and `-ksp_type ibcgs`

Complete **Table** of Solvers and Preconditioners

# 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  $\implies$  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  $\implies$  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)

# 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 optimal 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
```

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

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
```

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

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_fieldsplit_schur_factorization_type diag
```

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

May and Moresi, Preconditioned iterative methods for Stokes flow problems arising in computational geodynamics, 2008.

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_fieldsplit_schur_factorization_type lower
```

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

May and Moresi, Preconditioned iterative methods for Stokes flow problems arising in computational geodynamics, 2008.



# 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_fieldsplit_schur_factorization_type upper
```

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

May and Moresi, Preconditioned iterative methods for Stokes flow problems arising in computational geodynamics, 2008.

# 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
```

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

May and Moresi, Preconditioned iterative methods for Stokes flow problems arising in computational geodynamics, 2008.

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 use 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

- 1 Controlling the Solver
- 2 Where do I begin?
- 3 How do I improve?
- 4 Can We Do It?**
- 5 Nonlinear Systems

Okay, Computer Boy,  
Can you do this for  
a real PyLith Example?

Example 3D Hex8 `step10.cfg`

First, we try LU on the whole problem (`solver01.cfg`)

```
[pylithapp.petsc]  
snes_view = true  
pc_type = lu
```

FAIL

This is due to the saddle point introduced to handle the fault.

Example 3D Hex8 `step10.cfg`

First, we try LU on the whole problem (`solver01.cfg`)

```
[pylithapp.petsc]  
snes_view = true  
pc_type = lu
```

**FAIL**

This is due to the saddle point introduced to handle the fault.



Example 3D Hex8 `step10.cfg`

First, we try LU on the whole problem (`solver01.cfg`)

```
[pylithapp.petsc]  
snes_view = true  
pc_type = lu
```

**FAIL**

This is due to the saddle point introduced to handle the fault.

# Example 3D Hex8 `step10.cfg`

Next, we split fields using `PC_FIELDSPLIT` (`solver02.cfg`)

```
[pylithapp.timedependent.formulation]
split_fields = True
matrix_type = aij
[pylithapp.petsc]
snes_view = true

fs_pc_type = fieldsplit
fs_pc_use_amat = true
fs_pc_fieldsplit_type = additive
fs_fieldsplit_0_ksp_type = preonly
fs_fieldsplit_0_pc_type = lu
fs_fieldsplit_1_ksp_type = preonly
fs_fieldsplit_1_pc_type = jacobi
```

Converges slowly because preconditioner is not strong enough

Example 3D Hex8 `step10.cfg`

Next, we split fields using `PC_FIELDSPLIT` (`solver02.cfg`)

```
[pylithapp.timedependent.formulation]
split_fields = True
matrix_type = aij
[pylithapp.petsc]
snes_view = true

fs_pc_type = fieldsplit
fs_pc_use_amat = true
fs_pc_fieldsplit_type = additive
fs_fieldsplit_0_ksp_type = preonly
fs_fieldsplit_0_pc_type = lu
fs_fieldsplit_1_ksp_type = preonly
fs_fieldsplit_1_pc_type = jacobi
```

Converges slowly because preconditioner is not strong enough

Example 3D Hex8 `step10.cfg`

We need to use a full Schur factorization (`solver03.cfg`)

```
fs_pc_type = fieldsplit
fs_pc_use_amat = true
fs_pc_fieldsplit_type = schur
fs_pc_fieldsplit_schur_factorization_type = full
fs_fieldsplit_0_ksp_type = preonly
fs_fieldsplit_0_pc_type = lu
fs_fieldsplit_1_ksp_type = gmres
fs_fieldsplit_1_ksp_rtol = 1.0e-11
fs_fieldsplit_1_pc_type = jacobi
```

Works in one iterate! This is good for checking the physics.

Example 3D Hex8 `step10.cfg`

We need to use a full Schur factorization (`solver03.cfg`)

```
fs_pc_type = fieldsplit
fs_pc_use_amat = true
fs_pc_fieldsplit_type = schur
fs_pc_fieldsplit_schur_factorization_type = full
fs_fieldsplit_0_ksp_type = preonly
fs_fieldsplit_0_pc_type = lu
fs_fieldsplit_1_ksp_type = gmres
fs_fieldsplit_1_ksp_rtol = 1.0e-11
fs_fieldsplit_1_pc_type = jacobi
```

Works in one iterate! This is good for checking the physics.

# Example 3D Hex8 `step10.cfg`

We can add a user defined preconditioner for the Schur complement (`solver04.cfg`)

```
[pylithapp.timedependent.formulation]
use_custom_constraint_pc = True
[pylithapp.petsc]
fs_pc_fieldsplit_schur_precondition = user
```

# Example 3D Hex8 `step10.cfg`

We can add a user defined preconditioner for the Schur complement (`solver04.cfg`)

```
[pylithapp.timedependent.formulation]
use_custom_constraint_pc = True
[pylithapp.petsc]
fs_pc_fieldsplit_schur_precondition = user
```

## The initial convergence

```
0 SNES Function norm 1.547533880440e-02
  Linear solve converged due to CONVERGED_RTOL iterations 30
0 KSP Residual norm 1.158385264202e-02
  Linear solve converged due to CONVERGED_RTOL iterations 30
1 KSP Residual norm 2.231623131220e-13
  Linear solve converged due to CONVERGED_RTOL iterations 1
1 SNES Function norm 1.146037096697e-13
```

## Example 3D Hex8 `step10.cfg`

We can add a user defined preconditioner for the Schur complement (`solver04.cfg`)

```
[pylithapp.timedependent.formulation]
use_custom_constraint_pc = True
[pylithapp.petsc]
fs_pc_fieldsplit_schur_precondition = user
```

improves to

```
0 SNES Function norm 1.547533880440e-02
  0 KSP Residual norm 1.547533880440e-02
    Linear solve converged due to CONVERGED_RTOL iterations 24
  1 KSP Residual norm 4.395395819238e-14
    Linear solve converged due to CONVERGED_RTOL iterations 1
1 SNES Function norm 2.195069233327e-14
```

and gets much better for larger problems.



Example 3D Hex8 `step10.cfg`

You can back off the Schur complement tolerance (`solver05.cfg`)

```
fs_fieldsplit_1_ksp_rtol = 1.0e-05
```

at the cost of more iterates

```
0 SNES Function norm 1.547533880440e-02
  Linear solve converged due to CONVERGED_RTOL iterations 10
0 KSP Residual norm 1.158385275006e-02
  Linear solve converged due to CONVERGED_RTOL iterations 10
1 KSP Residual norm 1.743099082956e-07
  Linear solve converged due to CONVERGED_RTOL iterations 15
2 KSP Residual norm 9.111124472508e-13
  Linear solve converged due to CONVERGED_RTOL iterations 2
1 SNES Function norm 2.316766461963e-11
```

# Example 3D Hex8 `step10.cfg`

You can back off the primal LU solver (`solver06.cfg`)

```
fs_fieldsplit_0_ksp_type = preonly
fs_fieldsplit_0_pc_type  = gamg
```

at the cost of many more iterates

```
0 SNES Function norm 1.547533880440e-02
...
29 SNES Function norm 1.027184332531e-09
  0 KSP Residual norm 1.117454082561e-09
  1 KSP Residual norm 1.497082251939e-10
  2 KSP Residual norm 1.933577738149e-11
  3 KSP Residual norm 3.591884182205e-12
  4 KSP Residual norm 9.523637205261e-13
  5 KSP Residual norm 2.783009003067e-13
  6 KSP Residual norm 5.326205770835e-14
  7 KSP Residual norm 9.571296676103e-15
  8 KSP Residual norm 2.136358416887e-15
  9 KSP Residual norm 3.573852925652e-16
 10 KSP Residual norm 5.446314621304e-17
 11 KSP Residual norm 1.225127873021e-17
 12 KSP Residual norm 1.176269330132e-18
```

# Example 3D Hex8 `step10.cfg`

You can restore the behavior with a lower tolerance (`solver07.cfg`)

```
fs_fieldsplit_0_ksp_type = gmres
fs_fieldsplit_0_ksp_rtol = 5.0e-10
```

but it is quite sensitive to the tolerance.

```
0 SNES Function norm 1.547533880440e-02
  Linear solve converged due to CONVERGED_RTOL iterations 10
0 KSP Residual norm 1.158385275017e-02
  Linear solve converged due to CONVERGED_RTOL iterations 10
1 KSP Residual norm 1.757091032722e-07
  Linear solve converged due to CONVERGED_RTOL iterations 15
2 KSP Residual norm 1.654457025683e-12
  Linear solve converged due to CONVERGED_RTOL iterations 16
3 KSP Residual norm 2.248597744072e-17
Linear solve converged due to CONVERGED_RTOL iterations 3
1 SNES Function norm 3.052480752043e-10
```

# Outline

- 1 Controlling the Solver
- 2 Where do I begin?
- 3 How do I improve?
- 4 Can We Do It?
- 5 Nonlinear Systems**

# Driven Cavity Problem

## SNES ex19.c

```
./ex19 -lidvelocity 100 -grashof 1e2  
-da_grid_x 16 -da_grid_y 16 -da_refine 2  
-snes_monitor_short -snes_converged_reason -snes_view
```

# Driven Cavity Problem

## SNES ex19.c

```
./ex19 -lidvelocity 100 -grashof 1e2  
-da_grid_x 16 -da_grid_y 16 -da_refine 2  
-snes_monitor_short -snes_converged_reason -snes_view
```

```
lid velocity = 100, prandtl # = 1, grashof # = 100  
0 SNES Function norm 768.116  
1 SNES Function norm 658.288  
2 SNES Function norm 529.404  
3 SNES Function norm 377.51  
4 SNES Function norm 304.723  
5 SNES Function norm 2.59998  
6 SNES Function norm 0.00942733  
7 SNES Function norm 5.20667e-08  
Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 7
```

# Driven Cavity Problem

## SNES ex19.c

```
./ex19 -lidvelocity 100 -grashof 1e4  
-da_grid_x 16 -da_grid_y 16 -da_refine 2  
-snes_monitor_short -snes_converged_reason -snes_view
```

# Driven Cavity Problem

## SNES ex19.c

```
./ex19 -lidvelocity 100 -grashof 1e4  
-da_grid_x 16 -da_grid_y 16 -da_refine 2  
-snes_monitor_short -snes_converged_reason -snes_view
```

```
lid velocity = 100, prandtl # = 1, grashof # = 10000  
0 SNES Function norm 785.404  
1 SNES Function norm 663.055  
2 SNES Function norm 519.583  
3 SNES Function norm 360.87  
4 SNES Function norm 245.893  
5 SNES Function norm 1.8117  
6 SNES Function norm 0.00468828  
7 SNES Function norm 4.417e-08  
Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 7
```



# Driven Cavity Problem

## SNES ex19.c

```
./ex19 -lidvelocity 100 -grashof 1e5  
-da_grid_x 16 -da_grid_y 16 -da_refine 2  
-snes_monitor_short -snes_converged_reason -snes_view
```

# Driven Cavity Problem

## SNES ex19.c

```
./ex19 -lidvelocity 100 -grashof 1e5  
-da_grid_x 16 -da_grid_y 16 -da_refine 2  
-snes_monitor_short -snes_converged_reason -snes_view
```

```
lid velocity = 100, prandtl # = 1, grashof # = 100000
```

```
0 SNES Function norm 1809.96
```

```
Nonlinear solve did not converge due to DIVERGED_LINEAR_SOLVE iterations C
```

# Driven Cavity Problem

## SNES ex19.c

```
./ex19 -lidvelocity 100 -grashof 1e5  
-da_grid_x 16 -da_grid_y 16 -da_refine 2 -pc_type lu  
-snes_monitor_short -snes_converged_reason -snes_view
```

```
lid velocity = 100, prandtl # = 1, grashof # = 100000  
0 SNES Function norm 1809.96  
1 SNES Function norm 1678.37  
2 SNES Function norm 1643.76  
3 SNES Function norm 1559.34  
4 SNES Function norm 1557.6  
5 SNES Function norm 1510.71  
6 SNES Function norm 1500.47  
7 SNES Function norm 1498.93  
8 SNES Function norm 1498.44  
9 SNES Function norm 1498.27  
10 SNES Function norm 1498.18  
11 SNES Function norm 1498.12  
12 SNES Function norm 1498.11  
13 SNES Function norm 1498.11  
14 SNES Function norm 1498.11  
...
```

# Why isn't SNES converging?

- The Jacobian is wrong (maybe only in parallel)
  - Check with `-snes_type test` and `-snes_mf_operator -pc_type lu`
- The linear system is not solved accurately enough
  - Check with `-pc_type lu`
  - Check `-ksp_monitor_true_residual`, try right preconditioning
- The Jacobian is singular with inconsistent right side
  - Use **MatNullSpace** to inform the **KSP** of a known null space
  - Use a different Krylov method or preconditioner
- The nonlinearity is just really strong
  - Run with `-info` or `-snes_ls_monitor` to see line search
  - Try using trust region instead of line search `-snes_type tr`
  - Try grid sequencing if possible `-snes_grid_sequence`
  - Use a continuation

# Nonlinear Preconditioning

**PC** preconditions **KSP**

```
-ksp_type gmres
```

```
-pc_type richardson
```

**SNES** preconditions **SNES**

```
-snes_type ngmres
```

```
-npc_snes_type nrichardson
```

# Nonlinear Preconditioning

**PC** preconditions **KSP** **SNES** preconditions **SNES**

`-ksp_type gmres`

`-snes_type ngmres`

`-pc_type richardson`

`-npc_snes_type nrichardson`

# Nonlinear Use Cases

## Warm start Newton

```
-snes_type newtonls  
-npc_snes_type nrichardson -npc_snes_max_it 5
```

## Cleanup noisy Jacobian

```
-snes_type ngmres -snes_ngmres_m 5  
-npc_snes_type newtonls
```

## Additive-Schwarz Preconditioned Inexact Newton

```
-snes_type aspin -snes_npc_side left  
-npc_snes_type nasm -npc_snes_nasm_type restrict
```

# Nonlinear Preconditioning

Also called *globalization*

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short  
-snes_type newtonls -snes_converged_reason  
-pc_type lu
```

```
lid velocity = 100, prandtl # = 1, grashof # = 50000  
 0 SNES Function norm 1228.95  
 1 SNES Function norm 1132.29  
 2 SNES Function norm 1026.17  
 3 SNES Function norm 925.717  
 4 SNES Function norm 924.778  
 5 SNES Function norm 836.867  
  ⋮  
21 SNES Function norm 585.143  
22 SNES Function norm 585.142  
23 SNES Function norm 585.142  
24 SNES Function norm 585.142  
  ⋮
```



# Nonlinear Preconditioning

Also called *globalization*

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short  
-snes_type fas -snes_converged_reason  
-fas_levels_snes_type gs -fas_levels_snes_max_it 6
```

```
lid velocity = 100, prandtl # = 1, grashof # = 50000
```

```
0 SNES Function norm 1228.95
```

```
1 SNES Function norm 574.793
```

```
2 SNES Function norm 513.02
```

```
3 SNES Function norm 216.721
```

```
4 SNES Function norm 85.949
```

```
Nonlinear solve did not converge due to DIVERGED_INNER iterations 4
```

# Nonlinear Preconditioning

Also called *globalization*

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
-snes_type fas -snes_converged_reason
-fas_levels_snes_type gs -fas_levels_snes_max_it 6
-fas_coarse_snes_converged_reason
```

```
lid velocity = 100, prandtl # = 1, grashof # = 50000
 0 SNES Function norm 1228.95
   Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 12
 1 SNES Function norm 574.793
   Nonlinear solve did not converge due to DIVERGED_MAX_IT its 50
 2 SNES Function norm 513.02
   Nonlinear solve did not converge due to DIVERGED_MAX_IT its 50
 3 SNES Function norm 216.721
   Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 22
 4 SNES Function norm 85.949
   Nonlinear solve did not converge due to DIVERGED_LINE_SEARCH its 42
Nonlinear solve did not converge due to DIVERGED_INNER iterations 4
```

# Nonlinear Preconditioning

Also called *globalization*

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
-snes_type fas -snes_converged_reason
-fas_levels_snes_type gs -fas_levels_snes_max_it 6
-fas_coarse_snes_linesearch_type basic
-fas_coarse_snes_converged_reason
```

```
lid velocity = 100, prandtl # = 1, grashof # = 50000
 0 SNES Function norm 1228.95
   Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 6
  :
47 SNES Function norm 78.8401
   Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 5
48 SNES Function norm 73.1185
   Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 6
49 SNES Function norm 78.834
   Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 5
50 SNES Function norm 73.1176
   Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 6
  :
```

# Nonlinear Preconditioning

Also called *globalization*

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
-snes_type nrichardson -npc_snes_max_it 1 -snes_converged_reason
-npc_snes_type fas -npc_fas_coarse_snes_converged_reason
-npc_fas_levels_snes_type gs -npc_fas_levels_snes_max_it 6
-npc_fas_coarse_snes_linesearch_type basic
```

```
lid velocity = 100, prandtl # = 1, grashof # = 50000
 0 SNES Function norm 1228.95
   Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 6
 1 SNES Function norm 552.271
   Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 27
 2 SNES Function norm 173.45
   Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 45
  :
43 SNES Function norm 3.45407e-05
   Nonlinear solve converged due to CONVERGED_SNORM_RELATIVE its 2
44 SNES Function norm 1.6141e-05
   Nonlinear solve converged due to CONVERGED_SNORM_RELATIVE its 2
45 SNES Function norm 9.13386e-06
   Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 45
```

# Nonlinear Preconditioning

Also called *globalization*

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
-snes_type ngmres -npc_snes_max_it 1 -snes_converged_reason
-npc_snes_type fas -npc_fas_coarse_snes_converged_reason
-npc_fas_levels_snes_type gs -npc_fas_levels_snes_max_it 6
-npc_fas_coarse_snes_linesearch_type basic
```

```
lid velocity = 100, prandtl # = 1, grashof # = 50000
 0 SNES Function norm 1228.95
   Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 6
 1 SNES Function norm 538.605
   Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 13
 2 SNES Function norm 178.005
   Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 24
  :
27 SNES Function norm 0.000102487
   Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE its 2
28 SNES Function norm 4.2744e-05
   Nonlinear solve converged due to CONVERGED_SNORM_RELATIVE its 2
29 SNES Function norm 1.01621e-05
   Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 29
```

# Nonlinear Preconditioning

Also called *globalization*

```
./ex19 -lidvelocity 100 -grashof 5e4 -da_refine 4 -snes_monitor_short
-snes_type ngmres -npc_snes_max_it 1 -snes_converged_reason
-npc_snes_type fas -npc_fas_coarse_snes_converged_reason
-npc_fas_levels_snes_type newtonls -npc_fas_levels_snes_max_it 6
-npc_fas_levels_snes_linesearch_type basic
-npc_fas_levels_snes_max_linear_solve_fail 30
-npc_fas_levels_ksp_max_it 20 -npc_fas_levels_snes_converged_reason
-npc_fas_coarse_snes_linesearch_type basic
lid velocity = 100, prandtl # = 1, grashof # = 50000
0 SNES Function norm 1228.95
  Nonlinear solve did not converge due to DIVERGED_MAX_IT its 6
  :
  Nonlinear solve converged due to CONVERGED_SNORM_RELATIVE its 1
  :
1 SNES Function norm 0.1935
2 SNES Function norm 0.0179938
3 SNES Function norm 0.00223698
4 SNES Function norm 0.000190461
5 SNES Function norm 1.6946e-06
Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 5
```

# Other Solver Issues

I am not going to discuss  
fault friction solves today,

however if it is failing,

contact

[cig-short@geodynamics.org](mailto:cig-short@geodynamics.org)

# Other Solver Issues

I am not going to discuss  
fault friction solves today,

however if it is failing,  
contact

[cig-short@geodynamics.org](mailto:cig-short@geodynamics.org)