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 2012
Golden, CO  June 18–22, 2012
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?
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
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
We will illustrate options using

PETSc SNES ex5, located at
$PETSC_DIR/src/snes/examples/tutorials/ex5.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?
I am not going to discuss nonlinear systems today, however if Newton is failing, contact petsc-maint@mcs.anl.gov
I am not going to discuss nonlinear systems today, however if Newton is failing, contact petsc-maint@mcs.anl.gov
A Krylov solver builds a small model of a linear operator $A$, using a subspace defined by

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

where $r$ is the initial residual.

The small system is solved directly, and the solution is projected back to the original space.
A Krylov solver builds a small model of a linear operator $A$, using a subspace defined by

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

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
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
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
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:

- 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_FIELDSPSPLIT**
- 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_FIELDSPSPLIT`

- 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?
How do I improve?

- Look at what you have
- Back off in steps
Use \texttt{-snes_view} or \texttt{-ksp_view} to output a description of the solver:

\begin{verbatim}
KSP Object: (fieldsplit_0_) 1 MPI processes
type: fgmres
  GMRES: restart=100, using Classical (unmodified) Gram-Schmidt Orthogonalization with no iterative refinement
  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
\end{verbatim}
What did the convergence look like?

Use `-snes_monitor and -ksp_monitor, or -log_summary:`
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
How do I improve?

Look at what you have

What did the convergence look like?

Use \texttt{-snes\_monitor} and \texttt{-ksp\_monitor}, or \texttt{-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
How do I improve?

Look at what you have

What did the convergence look like?

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

<table>
<thead>
<tr>
<th>Event</th>
<th>Count</th>
<th>Time (sec)</th>
<th>Flops</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Max Ratio</td>
<td>Max Ratio</td>
<td>Ratio</td>
</tr>
<tr>
<td>KSPSetUp</td>
<td>12</td>
<td>1.0</td>
<td>3.6259e-03</td>
<td>1.0</td>
</tr>
<tr>
<td>KSPSolve</td>
<td>3</td>
<td>1.0</td>
<td>4.8937e-01</td>
<td>1.0</td>
</tr>
<tr>
<td>SNESSSolve</td>
<td>1</td>
<td>1.0</td>
<td>4.9477e-01</td>
<td>1.0</td>
</tr>
</tbody>
</table>
How do I improve?
Look at what you have

Look at timing

Use `-log_summary`:

<table>
<thead>
<tr>
<th>Event</th>
<th>Time (sec)</th>
<th>Flops</th>
<th>---</th>
<th>Global</th>
<th>---</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Max</td>
<td>Ratio</td>
<td>Max</td>
<td>Ratio</td>
<td>%T</td>
<td>%f</td>
</tr>
<tr>
<td>VecMDot</td>
<td>1.8904e-03</td>
<td>1.0</td>
<td>2.49e+04</td>
<td>1.0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>MatMult</td>
<td>2.1026e-03</td>
<td>1.0</td>
<td>2.65e+05</td>
<td>1.0</td>
<td>0</td>
<td>29</td>
</tr>
<tr>
<td>PCApply</td>
<td>4.6001e-01</td>
<td>1.0</td>
<td>7.78e+05</td>
<td>1.0</td>
<td>58</td>
<td>84</td>
</tr>
<tr>
<td>KSPSetUp</td>
<td>3.6259e-03</td>
<td>1.0</td>
<td>0.00e+00</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>KSPSolve</td>
<td>4.8937e-01</td>
<td>1.0</td>
<td>8.93e+05</td>
<td>1.0</td>
<td>61</td>
<td>97</td>
</tr>
<tr>
<td>SNESsolve</td>
<td>4.9477e-01</td>
<td>1.0</td>
<td>9.22e+05</td>
<td>1.0</td>
<td>62100</td>
<td>0</td>
</tr>
</tbody>
</table>

Use `-log_summary_python` to get this information as a Python module
How do I improve?

Look at what you have

Look at timing

Use `-log_summary`:

<table>
<thead>
<tr>
<th>Event</th>
<th>Time (sec)</th>
<th>Flops</th>
<th>---</th>
<th>Global ---</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Max</td>
<td>Ratio</td>
<td>Max</td>
<td>Ratio</td>
<td>%T</td>
</tr>
<tr>
<td>VecMDot</td>
<td>1.8904e-03</td>
<td>1.0</td>
<td>2.49e+04</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>MatMult</td>
<td>2.1026e-03</td>
<td>1.0</td>
<td>2.65e+05</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>PCApply</td>
<td>4.6001e-01</td>
<td>1.0</td>
<td>7.78e+05</td>
<td>1.0</td>
<td>58</td>
</tr>
<tr>
<td>KSPSetUp</td>
<td>3.6259e-03</td>
<td>1.0</td>
<td>0.00e+00</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>KSPSolve</td>
<td>4.8937e-01</td>
<td>1.0</td>
<td>8.93e+05</td>
<td>1.0</td>
<td>61</td>
</tr>
<tr>
<td>SNESolve</td>
<td>4.9477e-01</td>
<td>1.0</td>
<td>9.22e+05</td>
<td>1.0</td>
<td>62100</td>
</tr>
</tbody>
</table>

Use `-log_summary_python` to get this information as a Python module
3  How do I improve?
   - Look at what you have
   - Back off in steps
Weaken the KSP

GMRES $\Rightarrow$ 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 $\Rightarrow$ ILU

- $\text{pc$\_type ilu}$
- Less storage and work

In parallel,

- Hypre $\text{pc$\_type hypre$ \text{pc$\_hypre$\_type euclid}$
- Block-Jacobi $\text{pc$\_type bjacobi$ \text{sub$\_pc$\_type ilu}$
- Additive Schwarz $\text{pc$\_type asm$ \text{sub$\_pc$\_type ilu}$

Default for MG smoother is Chebychev/SOR(2)
Weaken the PC

LU $\Rightarrow$ 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 $\Rightarrow$ 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)
How do I improve?

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
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`
The common block preconditioners for Stokes require only options:

The Stokes System

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

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

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
```


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`

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
```

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`


The common block preconditioners for Stokes require only options:

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

\[
\text{PC} = \begin{pmatrix}
I & 0 \\
B^T A^{-1} & I
\end{pmatrix}
\begin{pmatrix}
\hat{A} & 0 \\
0 & \hat{S}
\end{pmatrix}
\begin{pmatrix}
I & A^{-1} B \\
0 & I
\end{pmatrix}
\]
All block preconditioners can be *embedded* in MG using only options:

- `pc_type mg -pc_mg_levels 5 -pc_mg_galerkin`
- `mg_levels_pc_type fieldsplit`
- `mg_levels_pc_field_split_type`

System on each Coarse Level

\[
R \begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix} P
\]
How do I improve?

Stokes example

All block preconditioners can be *embedded* in MG using only options:

- `pc_type mg` - `pc_mg_levels 5` - `pc_mg_galerkin`
- `mg_levels_pc_type fieldsplit`
- `mg_levels_pc_field_split_type additive`
- `mg_levels_fieldsplit_0_pc_type gamg`
- `mg_levels_fieldsplit_0_ksp_type preonly`
- `mg_levels_fieldsplit_1_pc_type jacobi`
- `mg_levels_fieldsplit_1_ksp_type preonly`

Smoother

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

M. Knepley (UC)
All block preconditioners can be \textit{embedded} in MG using only options:

- \texttt{-pc_type mg -pc_mg_levels 5 -pc_mg_galerkin}
- \texttt{-mg_levels_pc_type fieldsplit}
- \texttt{-mg_levels_pc_field_split_type multiplicative}

- \texttt{-mg_levels_fieldsplit_0_pc_type gamg}
- \texttt{-mg_levels_fieldsplit_0_ksp_type preonly}

- \texttt{-mg_levels_fieldsplit_1_pc_type jacobi}
- \texttt{-mg_levels_fieldsplit_1_ksp_type preonly}

\begin{equation*}
\text{Smoother}
\begin{pmatrix}
\hat{A} & B \\
0 & I
\end{pmatrix}
\end{equation*}
All block preconditioners can be *embedded* in MG using only options:

- `pc_type mg -pc_mg_levels 5 -pc_mg_galerkin`
- `mg_levels_pc_type fieldsplit`
- `mg_levels_pc_field_split_type schur`
- `mg_levels_fieldsplit_0_pc_type gamg`
- `mg_levels_fieldsplit_0_ksp_type preonly`
- `mg_levels_fieldsplit_1_pc_type none`
- `mg_levels_fieldsplit_1_ksp_type minres`
- `mg_levels_pc_fieldsplit_schur_factorization_type diag`

\[
\begin{pmatrix}
\hat{A} & 0 \\
0 & -\hat{S}
\end{pmatrix}
\]
Stokes example

All block preconditioners can be *embedded* in MG using only options:

```
-pc_type mg -pc_mg_levels 5 -pc_mg_galerkin
-mg_levels_pc_type fieldsplit
-mg_levels_pc_field_split_type schur

-mg_levels_fieldsplit_0_pc_type gamg
-mg_levels_fieldsplit_0_ksp_type preonly

-mg_levels_fieldsplit_1_pc_type none
-mg_levels_fieldsplit_1_ksp_type minres

-mg_levels_pc_fieldsplit_schur_factorization_type lower
```

\[
\begin{pmatrix}
\hat{A} & 0 \\
B^T & \hat{S}
\end{pmatrix}
\]
All block preconditioners can be *embedded* in MG using only options:

-pc_type mg -pc_mg_levels 5 -pc_mg_galerkin
-mg_levels_pc_type fieldsplit
-mg_levels_pc_field_split_type schur
-mg_levels_fieldsplit_0_pc_type gamg
-mg_levels_fieldsplit_0_ksp_type preonly
-mg_levels_fieldsplit_1_pc_type none
-mg_levels_fieldsplit_1_ksp_type minres
-mg_levels_pc_fieldsplit_schur_factorization_type upper

\[
\begin{pmatrix}
\hat{A} & B \\
0 & \hat{S}
\end{pmatrix}
\]
All block preconditioners can be *embedded* in MG using only options:

- `pc_type mg -pc_mg_levels 5 -pc_mg_galerkin`
- `mg_levels_pc_type fieldsplit`
- `mg_levels_pc_field_split_type schur`
- `mg_levels_fieldsplit_0_pc_type gamg`
- `mg_levels_fieldsplit_0_ksp_type preonly`
- `mg_levels_fieldsplit_1_pc_type lsc`
- `mg_levels_fieldsplit_1_ksp_type minres`
- `mg_levels_pc_fieldsplit_schur_factorization_type upper`
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?
Okay, Computer Boy, Can you do this for a real PyLith Example?
First, we try LU on the whole problem (solver01.cfg)

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

FAIL

This is due to the saddle point introduced to handle the fault.
First, we try LU on the whole problem (solver01.cfg)

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

FAIL

This is due to the saddle point introduced to handle the fault.
First, we try LU on the whole problem (solver01.cfg)

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

FAIL

This is due to the saddle point introduced to handle the fault.
Next, we split fields using `PC_FIELDSPLIT` (solver02.cfg)

```python
[pylithapp.time_dependent.formulation]
split_fields = True
matrix_type = aij
[pylithapp.petsc]
snes_view = true
ksp_monitor_true_residual = true
fs_pc_type = fieldsplit
fs_pc_fieldsplit_real_diagonal = true
fs_pc_fieldsplit_type = additive
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
```

Does not converge because preconditioner is not strong enough
Next, we split fields using `PC_FIELDSPSPLIT` (solver02.cfg)

```python
[pylithapp.timedependent.formulation]
split_fields = True
matrix_type = aij

[pylithapp.petsc]
snes_view = true
ksp_monitor_true_residual = true
fs_pc_type = fieldsplit
fs_pc_fieldsplit_real_diagonal = true
fs_pc_fieldsplit_type = additive
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
```

Does not converge because preconditioner is not strong enough
We need to use a full Schur factorization (solver03.cfg)

```plaintext
fs_pc_type = fieldsplit
fs_pc_fieldsplit_real_diagonal = 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.
We need to use a full Schur factorization (solver03.cfg)

```plaintext
fs_pc_type = fieldsplit
fs_pc_fieldsplit_real_diagonal = 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.
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
```
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  2.198105129707e-13
Linear solve converged due to CONVERGED_RTOL iterations 1
1  SNES  Function  norm  1.146157083300e-13
We can add a user defined preconditioner for the Schur complement (solver04.cfg)

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

Improves to

<table>
<thead>
<tr>
<th></th>
<th>SNES Function norm</th>
<th>KSP Residual norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.547533880440e-02</td>
<td>1.158385264203e-02</td>
</tr>
<tr>
<td></td>
<td>Linear solve converged due to CONVERGED_RTOL iterations 24</td>
<td>Linear solve converged due to CONVERGED_RTOL iterations 25</td>
</tr>
<tr>
<td>1</td>
<td>5.404218646700e-14</td>
<td>2.200824144647e-14</td>
</tr>
</tbody>
</table>

Linear solve converged due to CONVERGED_RTOL iterations 1

And gets much better for larger problems.
You can back off the Schur complement tolerance \((\text{solver05.cfg})\)
\[
\text{fs_fieldsplit\_1\_ksp\_rtol} = 1.0\times 10^{-5}
\]
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 1.743099082900e-07
  - Linear solve converged due to CONVERGED_RTOL iterations 15
- 2 KSP Residual norm 9.111124467571e-13
  - Linear solve converged due to CONVERGED_RTOL iterations 2
- 1 SNES Function norm 2.316774353785e-11
You can back off the primal LU solver (solver06.cfg)

\[
\begin{align*}
fs\_fieldsplit\_0\_ksp\_type & = \text{preonly} \\
fs\_fieldsplit\_0\_pc\_type & = \text{ml}
\end{align*}
\]

at the cost of many more iterates

\[
\begin{align*}
0 \quad \text{SNES Function norm} & = 1.547533880440e-02 \\
\cdots \\
34 \quad \text{SNES Function norm} & = 1.094751648499e-09 \\
0 \quad \text{KSP Residual norm} & = 1.044862482330e-09 \\
1 \quad \text{KSP Residual norm} & = 1.026476859438e-11 \\
2 \quad \text{KSP Residual norm} & = 2.352619621602e-13 \\
3 \quad \text{KSP Residual norm} & = 4.901870841230e-15 \\
4 \quad \text{KSP Residual norm} & = 1.028487933615e-16 \\
5 \quad \text{KSP Residual norm} & = 2.250903096143e-18 \\
6 \quad \text{KSP Residual norm} & = 5.050245895484e-20 \\
35 \quad \text{SNES Function norm} & = 4.074830594018e-10
\end{align*}
\]

Nonlinear solve converged due to CONVERGED_FNORM_ABS iterations 35
You can restore the behavior with a lower tolerance (solver07.cfg)

```plaintext
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 1.158385274961e-02
  Linear solve converged due to CONVERGED_RTOL iterations 10
1  KSP Residual norm 1.744541880226e-07
  Linear solve converged due to CONVERGED_RTOL iterations 15
2  KSP Residual norm 1.585882433753e-12
  Linear solve converged due to CONVERGED_RTOL iterations 16
3  KSP Residual norm 1.222018988543e-17
  Linear solve converged due to CONVERGED_RTOL iterations 3
1  SNES Function norm 5.034307203820e-11