Introduction to PETSc: Solvers

Serge Van Criekingen

Maison de la Simulation

May 15, 2013

PETSc Solvers : Introduction

PETSc specializes in Krylov-type iterative solvers, but offers interfaces for external direct solvers (Mumps, PaStiX, SuperLU) and other iterative solvers (Hypre, Trilinos/ML,...).

Types :

- KSP $\equiv$ Krylov solver
- PC $\equiv$ Preconditioner

Note : no specific type for direct solver, in fact handled as "special case" of KSP (!) - see below.

PETSc Solvers : Create & Set Matrix

```
KSPCreate(MPI_Comm comm, KSP *ksp);
```

PETSc Solvers : Create & Set Matrix

```
KSPCreate(MPI_Comm comm, KSP *ksp);
```

```
KSPSetOperators(KSP ksp, Mat A, Mat precondBase,
                               MatStructure flag);
```

`A` = system matrix

`precondBase` = base matrix to derive the preconditioner (typically A itself)

`flag` = `SAME_PRECONDITIONER`, `SAME_NONZERO_PATTERN`, `DIFFERENT_NONZERO_PATTERN` (ignored if only one solve)

PETSc Solvers : Create & Set Matrix

```
KSPCreate(MPI_Comm comm, KSP *ksp);
```

```
KSPSetOperators(KSP ksp, Mat A, Mat precondBase,
                                 MatStructure flag);
```

`A` = system matrix

`precondBase` = base matrix to derive the preconditioner (typically A itself)

`flag` = `SAME_PRECONDITIONER`, `SAME_NONZERO_PATTERN`, `DIFFERENT_NONZERO_PATTERN`
(ignored if only one solve)

Note : `A` can be a shell matrix $\Rightarrow$ matrix-free methods.

PETSc Solvers : Set Solution Method

```
KSPSetType(KSP ksp, KSPType kspType);
KSPSetTolerances(KSP ksp,
          real rtol, real atol, real dtol, int maxits);
```

`kspType = KSPCG, KSPGMRES, KSPBCGS, KSPMINRES, ...`

`rtol, atol, dtol` = relative, absolute, divergence tolerance

PETSc Solvers : Set Solution Method

```
KSPSetType(KSP ksp, KSPType kspType);
KSPSetTolerances(KSP ksp,
          real rtol, real atol, real dtol, int maxits);
```

`kspType = KSPCG, KSPGMRES, KSPBCGS, KSPMINRES, ...`

`rtol, atol, dtol` = relative, absolute, divergence tolerance

or for run-time specification :

```
KSPSetFromOptions(KSP ksp);
```

and program launched using :

```
  mpirun   ...     -ksp_type <method> -ksp_rtol <rtol>
```

PETSc Solvers : Solve & After

```
KSPSetUp(KSP ksp);
```

To solve `A x = b` :

```
KSPSolve(KSP ksp,Vec b,Vec x);
```

- `x` overwritten with answer.
- initial guess `x=0` unless `KSPSetInitialGuessNonzero` before solve.

PETSc Solvers : Solve & After

```
KSPSetUp(KSP ksp);
```

To solve `A x = b` :

```
KSPSolve(KSP ksp,Vec b,Vec x);
```

- `x` overwritten with answer.
- initial guess `x=0` unless `KSPSetInitialGuessNonzero` before solve.

After solve :

`KSPGetConvergedReason` (e.g., `rtol` achieved)
`KSPGetIterationNumber`
`KSPGetResidualNorm`
`KSPDestroy`

PETSc Solvers : Preconditioning (PC type)

```
PCSetType(PC pc, PCType pcType);
```

with `pcType = PCNONE, PCJACOBI, PCSOR, PCILU, PCASM,...`

PETSc Solvers : Preconditioning (PC type)

```
PCSetType(PC pc, PCType pcType);
```

with `pcType = PCNONE, PCJACOBI, PCSOR, PCILU, PCASM,...`

Example with `PCJACOBI` :

```
KSPgetPC(ksp,&pc);
PCSetType(pc,PCJACOBI);
PCSetUp(pc);
```

PETSc Solvers : Preconditioning (PC type)

```
PCSetType(PC pc, PCType pcType);
```

with `pcType = PCNONE, PCJACOBI, PCSOR, PCILU, PCASM,...`

Example with `PCSOR` :

```
KSPgetPC(ksp,&pc);
PCSetType(pc,PCSOR);
PCSetUp(pc);
```

PETSc Solvers : Preconditioning (PC type)

```
PCSetType(PC pc, PCType pcType);
```

with pcType = PCNONE, PCJACOBI, PCSOR, PCILU, PCASM,...

Example with PCILU :

```
KSPgetPC(ksp,&pc);
PCSetType(pc,PCILU);
PCFactorSetLevels(pc,level_of_fill);
PCSetUp(pc);
```

PETSc Solvers : Preconditioning (PC type)

Example with PCASM :

```
KSPgetPC(ksp,&pc);
PCSetType(pc,PCASM);
PCASMSetOverlap(pc,overlap);
PCSetUp(PC pc);
PCASMGetSubKSP(pc, &n_local, &first_local,
                                        &subKSP[]);
for (i=0; i<nlocal; i++){
      KSPSetType(subKSP(i),KSPPREONLY);
      KSPGetPC(subKSP(i),&subPC);
      PCSetType(subPC,PCSOR);
}
```

PETSc Solvers : Preconditioning (PC type)

Example with `PCASM` :

```
KSPgetPC(ksp,&pc);
PCSetType(pc,PCASM);
PCASMSetOverlap(pc,overlap);
PCSetUp(PC pc);
PCASMGetSubKSP(pc, &n_local, &first_local,
                                        &subKSP[]);
for (i=0; i<nlocal; i++){
      KSPSetType(subKSP(i),KSPPREONLY);
      KSPGetPC(subKSP(i),&subPC);
      PCSetType(subPC,PCSOR);
}
```

NB : `KSPPREONLY` = default "sub-type"

PETSc Solvers : Direct Methods

In PETSc, direct methods are special cases of Krylov methods (KSP), with only the PCLU preconditioner applied :

```
KSPSetType(ksp,KSPPREONLY);
KSPgetPC(ksp, *pc);
PCSetType(pc, PCLU);
```

PETSc Solvers : Direct Methods

In PETSc, direct methods are special cases of Krylov methods (KSP), with only the PCLU preconditioner applied :

```
KSPSetType(ksp,KSPPREONLY);
KSPgetPC(ksp, *pc);
PCSetType(pc, PCLU);
```

The PETSc built-in PCLU works only in sequential.
For parallel direct methods, use external solvers :

```
PCFactorSetMatSolverPackage(pc, MATSOLVERPASTIX);
```

PETSc Solvers : Direct Methods

In PETSc, direct methods are special cases of Krylov methods (KSP), with only the PCLU preconditioner applied :

```
KSPSetType(ksp,KSPPREONLY);
KSPgetPC(ksp, *pc);
PCSetType(pc, PCLU);
```

The PETSc built-in PCLU works only in sequential.
For parallel direct methods, use external solvers :

```
PCFactorSetMatSolverPackage(pc, MATSOLVERMUMPS);
```

PETSc Solvers : Direct Methods

In PETSc, direct methods are special cases of Krylov methods (KSP), with only the PCLU preconditioner applied :

```
KSPSetType(ksp,KSPPREONLY);
KSPgetPC(ksp, *pc);
PCSetType(pc, PCLU);
```

The PETSc built-in PCLU works only in sequential.
For parallel direct methods, use external solvers :

```
PCFactorSetMatSolverPackage(pc, MATSOLVERSUPERLU_DIST);
```

PETSc Solvers : Direct Methods

In PETSc, direct methods are special cases of Krylov methods (KSP), with only the PCLU preconditioner applied :

```
KSPSetType(ksp,KSPPREONLY);
KSPgetPC(ksp, *pc);
PCSetType(pc, PCLU);
```

The PETSc built-in PCLU works only in sequential.
For parallel direct methods, use external solvers :

```
PCFactorSetMatSolverPackage(pc, MATSOLVERSUPERLU_DIST);
```

For run-time specification :

```
PCSetFromOptions(pc);
```

PETSc Solvers : Viewing

```
KSPView(KSP ksp, PETSC_VIEWER_STDOUT_WORLD)
```

Example output with *BCGS* and *PCSOR* :

```
KSP Object: 8 MPI processes
  type: bcgs
  maximum iterations=1000000, initial guess is zero
  tolerances:  relative=1e-08, absolute=1e-50, divergence=10000
  left preconditioning
  using DEFAULT norm type for convergence test
PC Object: 8 MPI processes
  type: sor
    SOR: type = local_symmetric, iterations = 1, local iterations = 1, omega =
  linear system matrix = precond matrix:
  Matrix Object:   8 MPI processes
    type: mpibaij
    rows=57600, cols=57600, bs=4
    total: nonzeros=1142400, allocated nonzeros=1612800
    total number of mallocs used during MatSetValues calls =0
        block size is 4
```

Solve the system defined by one of the matrices you built previously and a random right-hand-side vector (use `VecSetRandom`).

Compare various iterative solution method

- for symmetric matrices : `CG`, `MinRes`
- for non-symmetric matrices : `GMRES`, `BiCGStab`

changing the method at run-time.

Compute the norm of the residual $||Ax - b||$ yourself
and compare with the one given by `PETSc`
(use `KSPDefaultConvergedSetUIRNorm`).

Compare the results with an external direct solver, e.g. MUMPS.