PRACE PETSc Tutorial

CSC - IT Center for Science, Espoo, Finland

2-3 May, 2019

Part 5: Linear System Solvers

Václav Hapla
ETH Zürich

Linear system solvers

(KSP = Krylov SPace solvers)

```
KSP ksp;
KSPType type = KSPCG;
MPI_Comm comm = PETSC_COMM_WORLD;
```

- **create**: `KSPCreate(comm, &ksp);`
- **type**: `KSPSetType(ksp, type);  /* default=KSPGMRES */`
- **options:** `KSPSetFromOptions(ksp);`
- **dealloc**: `KSPDestroy(&ksp);`

Solving linear system

```
Mat A, B;  Vec b, x;
/* ... initialize A, b, x ... */
KSPSetOperators(ksp, A, B);
KSPSolve(ksp,b,x);
```

- A **defines** the linear **system**
- B is used for **preconditioner construction**
  (e.g. explicit approximation of implicit A, passed to incomplete factorization)
- Usually A = B

Convergence tolerances

```
KSPSetTolerances(ksp, rtol, atol, dtol, maxit);
```

- `rtol` = the **relative** convergence tolerance
  stop if `residual < rtol * norm(RHS)`
- `atol` = **absolute** convergence tolerance
  stop if `resisual < atol`
- `dtol` = **divergence** tolerance
  stop if `residual > dtol * norm(RHS)`
- `maxit` = **maximum** number of **iterations**
- all can be set to `PETSC_DEFAULT`
- **options database**: `-ksp_rtol 1e-6 -ksp_divtol 1e5 -ksp_atol 1e-12 -ksp_max_it 2000`

Preconditioners

- many built-in and interfaced preconditioners
- ILU, block Jacobi, sparse aproximate inverse ...
- can be composed in additive or multiplicative way (`PCCOMPOSITE`)

```
KSP ksp;
PC pc;
PCType pctype=PCILU;
...
KSPGetPC(ksp,&pc);
PCSetType(pc,pctype);
```

Try out various solvers

```
KSPType ksptype = KSPCG;
KSP ksp;
...
KSPSetType(ksp,ksptype);
```

- `type = {KSPRICHARDSON, KSPCG, KSPGCR, KSPGMRES, ...}`
- command-line:
  `-ksp_type {richardson,cg,gcr,gmres,...}`
- see manual pages of `KSPSetType`, `KSPType`, and concrete types

Try out various preconditioners

```
PCType pctype = PCILU;
PC pc;
KSP ksp;
...
KSPGetPC(ksp,&pc);
PCSetType(pc,pctype);
```

- `type={PCJACOBI,PCBJACOBI,PCILU,PCICC...}`
- command-line:
  `-pc_type {richardson,cg,gcr,gmres,...}`
- see manual pages of `PCSetType`, `PCType`, and concrete types

Options prefixes (1)

- Sometimes there can be **multiple instances** of the **same class** that we want to control **separately**.
- **code:**

```
KSP ksp1, ksp2, ksp3;
...
KSPSetOptionsPrefix(ksp2, "ksp2_");  /* called only on ksp2 */
```

- **options:**

```
-ksp_rtol 1e-8          # sets rel. tol. of all unprefixed (ksp1 and ksp3)
-ksp2_ksp_rtol 1e-4     # sets rel. tol. of ksp2
```

- `KSPView()` prints the prefix, e.g.

```
KSP Object: (ksp2_) 1 MPI processes
```

Options prefixes (2)

- It's easy to play around with many different solver types and settings.

```
-ksp2_ksp_type cg -ksp2_pc_type bjacobi -ksp2_ksp_rtol 1e-3 \
-ksp2_ksp_converged_reason \
-ksp2_sub_ksp_type richardson -ksp2_sub_pc_type icc -ksp2_sub_ksp_converged_reason
```

- There can be built-in prefixes related composed solver/preconditioners, consult documentation.
  - e.g. `sub_` above points to `PCBJACOBI` block-wise `KSP/PC`
- To overcome lengthy prefixes, one can use handy `-prefix_push/-prefix_pop`:

```
-prefix_push ksp2_ \
  -ksp_type cg -pc_type bjacobi -ksp_rtol 1e-3 -ksp_converged_reason \
  -prefix_push sub_ \
    -ksp_type richardson -pc_type icc -ksp_converged_reason \
  -prefix_pop \
-prefix_pop
```

Direct solvers

- direct solvers = special case of preconditioned iterative solvers
- just one iteration with application of „ultimate preconditioner“, i.e. forward & backward substitution of a complete factor

```
KSPSetType(ksp, KSPPREONLY);
PCSetType(pc, PCLU);     /* or PCCHOLESKY */
```

Iterative/preconditioned/direct solvers

method	PCType	KSPType
pure iterative	none	cg, gmres, gcr, richardson,...
preconditioned iterative	ilu, icc, jacobi, sor, ...	cg, gmres, gcr, richardson,...
direct	lu, cholesky	preonly

MatSolverPackage

- PETSc built-in factorization routines are not very efficient but there are interfaces to several external tools (MUMPS, SuperLU_Dist, PaStiX, ...)
- for current list, search for MATSOLVER* at https://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/Mat/index.html

```
PC pc;
MatSolverPackage pkg = MATSOLVERMUMPS;
PCFactorSetMatSolverPackage(pc,pkg);
```

- `pkg={MATSOLVERMUMPS,MATSOLVERUMFPACK,...}`
- command-line:
  `-pc_factor_mat_solver_package {mumps,umfpack,...}`
- see manual pages of `PCFactorSetMatSolverType`, `MatSolverType`, and concrete packages

Low-level access to factorization

```
PC pc;  Mat F,B,X;  Vec b,x;
PCSetType(pc,PCCHOLESKY);                                   /* or PCLU */
PCFactorSetMatSolverPackage(pc,MATSOLVERMUMPS);
      /* or e.g. MATSOLVERSTRUMPACK, see MatMatSolve() manual page */
PCSetUp(pc);
PCFactorGetMatrix(pc,&F);              /* F contains factor(s), not A */
/* vector-vector solve */
MatSolve(F,b,x);                                    /* Solve A*x  = b */
MatSolveTranspose(F,b,x);                           /* Solve A'*x = b */
/* matrix-matrix solve */
MatMatSolve(F,B,X);                                 /* Solve A*X  = B */
MatMatSolveTranspose(F,B,X);                        /* Solve A'*X = B */
MatMatTransposeSolve(F,B,X);                        /* Solve A*X' = B */
```

Thanks for your attention.