Introduction to PETSc

SNES

Fall, 2010

Nonlinear Solvers (SNES)

SNES: Scalable Nonlinear Equations Solvers

- Application code interface
- Choosing the solver
- Setting algorithmic options
- Viewing the solver
- Determining and monitoring convergence
- Matrix-free solvers
- User-defined customizations

solvers: nonlinear

Nonlinear Solvers

Goal: For problems arising from PDEs, support the general solution of *F(u) = 0*

User provides:

- Code to evaluate *F(u)*
- Code to evaluate Jacobian of *F(u)* (optional)
  - or use sparse finite difference approximation
  - or use automatic differentiation
    - AD support via collaboration with P. Hovland and B. Norris
    - via automated interface to ADIFOR and ADIC (see http://www.mcs.anl.gov/autodiff)

solvers: nonlinear

Nonlinear Solvers (SNES)

- Newton-based methods, including
  - Line search strategies
  - Trust region approaches (using TAO)
  - Pseudo-transient continuation
  - Matrix-free variants
- Can customize all phases of solution process

solvers:
nonlinear

Sample Nonlinear Application: Driven Cavity Problem

\- Lap(U) - Grad_y(Omega) = 0

\- Lap(V) + Grad_x(Omega) = 0

\- Lap(Omega) + Div([U*Omega,V*Omega]) - GR*Grad_x(T) = 0

\- Lap(T) + PR*Div([U*T,V*T]) = 0

petsc/src/snes/examples/tutorials/ex19.c

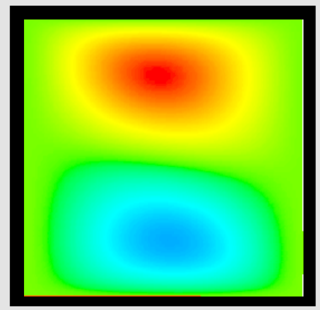
solvers:
nonlinear

Sample Nonlinear Application: Driven Cavity Problem

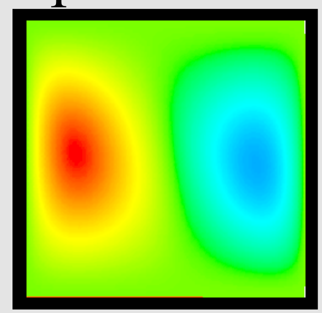
- Velocity-vorticity formulation
- Flow driven by lid and/or bouyancy
- Logically regular grid, parallelized with DAs
- Finite difference discretization
- source code:

  petsc/src/snes/examples/tutorials/ex19.c

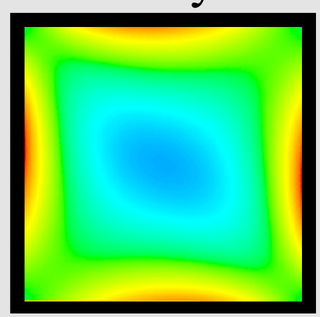
Solution Components



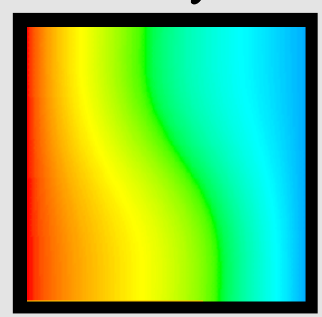
velocity: u



velocity: v



vorticity: z



temperature: T

Application code author: D. E. Keyes

solvers: nonlinear

Basic Nonlinear Solver Code (C/C++)

```
SNES   snes;                        /*  nonlinear solver context  */
Mat    J;                           /*  Jacobian matrix  */
Vec    x, F;                        /*  solution, residual vectors  */
int    n, its;                      /*  problem dimension, number of iterations  */
ApplicationCtx  usercontext;        /*  user-defined application context  */

...
MatCreate(PETSC_COMM_WORLD,PETSC_DECIDE,PETSC_DECIDE,n,n, &J);
MatSetFromOptions(J);
VecCreate(PETSC_COMM_WORLD, &x);
VecSetSizes(x,PETSC_DECIDE,n);
VecSetFromOptions(x);
VecDuplicate(x, &F);

SNESCreate(PETSC_COMM_WORLD, &snes);
SNESSetFunction(snes, F, EvaluateFunction, usercontext);
SNESSetJacobian(snes,J,J, EvaluateJacobian, usercontext);
SNESSetFromOptions(snes);
SNESSolve(snes, PETSC_NULL, x );
SNESDestroy(snes);
```

solvers:
nonlinear

Solvers Based on Callbacks

- User provides routines to perform actions that the library requires. For example,
  - SNESSetFunction(SNES,...)
    - uservector - vector to store function values
    - userfunction - name of the user's function
    - usercontext - pointer to private data for the user's function
- Now, whenever the library needs to evaluate the user's nonlinear function, the solver may call the application code directly with its own local state.
- usercontext: serves as an application context object. Data are handled through such opaque objects; the library never sees irrelevant application data.

important concept

solvers: nonlinear

Sample Application Context: Driven Cavity Problem

```
typedef struct {
   /* - - - - - - - - - - - - - -  basic application data  - - - - - - - - - - - - - - - - - */
   double        lid_velocity, prandtl;   /* problem parameters */
   double        grashof;                 /* problem parameters */
   int           mx, my;                  /* discretization parameters */
   int           mc;                      /* number of DoF per node */
   int           draw_contours;           /* flag - drawing contours */
   /* - - - - - - - - - - - - - -   parallel data  - - - - - - - - - - - - - - - - - - - - - - - */
   MPI_Comm      comm;                    /* communicator */
   DA            da;                      /* distributed array */
   Vec           localF, localX;          /* local ghosted vectors */
} AppCtx;
```

solvers:
nonlinear

Sample Function Evaluation Code: Driven Cavity Problem

```
UserComputeFunction(SNES snes, Vec X, Vec F, void *ptr)
{
   AppCtx   *user = (AppCtx *) ptr;          /* user-defined application context */
   int       istart, iend, jstart, jend;     /* local starting and ending grid points */
   Scalar    *f;                             /* local vector data */
    ….
    /* (Code to communicate nonlocal ghost point data not shown) */
    VecGetArray( F, &f );
    /* (Code to compute local function components; insert into f[ ] shown on next slide) */
    VecRestoreArray( F, &f );
    ….
return 0;
}
```

solvers:
nonlinear

Finite Difference Jacobian Computation

- Compute and explicitly store Jacobian via 1st-order FD
  - Dense: -snes_fd, SNESDefaultComputeJacobian()
  - Sparse via colorings: MatFDColoringCreate(), SNESDefaultComputeJacobianColor()
- Matrix-free Newton-Krylov via 1st-order FD, no preconditioning unless specifically set by user
  - -snes_mf
- Matrix-free Newton-Krylov via 1st-order FD, user-defined preconditioning matrix
  - -snes_mf_operator

solvers: nonlinear

Uniform access to all linear and nonlinear solvers

- -ksp_type [cg, gmres, bcgs, tfqmr,…]
- -pc_type [lu, ilu, jacobi, sor, asm,…]
- -snes_type [ls,…]

- -snes_line_search <line search method>
- -snes_ls <parameters>
- -snes_convergence <tolerance>
- etc...

solvers: nonlinear

Customization via Callbacks: Setting a user-defined line search routine

SNESSetLineSearch(SNES snes,int(*ls)(…),void *lsctx)

Available line search routines ls( ) include:

- SNESCubicLineSearch( ) - cubic line search
- SNESQuadraticLineSearch( ) - quadratic line search
- SNESNoLineSearch( ) - full Newton step
- SNESNoLineSearchNoNorms( ) - full Newton step but calculates no norms (faster in parallel, useful when using a fixed number of Newton iterations instead of usual convergence testing)
- YourOwnFavoriteLineSearchRoutine( )

SNES: Review of Basic Usage

- SNESCreate( ) - Create SNES context
- SNESSetFunction( ) - Set function eval. routine
- SNESSetJacobian( ) - Set Jacobian eval. routine
- SNESSetFromOptions( ) - Set runtime solver options for [SNES,SLES, KSP,PC]
- SNESSolve( ) - Run nonlinear solver
- SNESView( ) - View solver options actually used at runtime (alternative: -snes_view)
- SNESDestroy( ) - Destroy solver

solvers: nonlinear

SNES: Review of Selected Options

Functionality	Procedural Interface	Runtime Option
Set nonlinear solver	SNESSetType( )	-snes_type [ls,tr,umls,umtr,…]
Set monitoring routine	SNESSetMonitor( )	-snes_monitor -snes_monitor_true_residual, …
Set convergence tolerances	SNESSetTolerances( )	-snes_rtol <rt> -snes_atol <at> -snes _max_its <its>
Set line search routine	SNESSetLineSearch( )	-sne_ls [cubic,quadratic,…]
View solver options	SNESView( )	-snes_view
Set linear solver options	SNESSetKSP( ) SNESGetKSP() KSPGetPC( )	-ksp_type <ksptype> -ksp_rtol <krt> -pc_type <pctype> …

And many more options...

solvers:
nonlinear