Introduction to petsc4py

Overview

- petsc4py is a Pythonic interface to PETSc (pronounced: "pet-see"), the Portable Extensible Toolkit for Scientific Computing.

- PETSc is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations.

  ▶ Details

- PETSc and petsc4py have been designed to be as easy as possible to use for beginners while still delivering the performance expected of high performance numerical codes.

  ▶ Details

Outline

- Dive into petsc4py - Solving the Poisson Equation in 3 Dimensions
  - Vec - distributed vectors
  - Mat - distributed matrices and matrix operators
  - KSP - Krylov subspace methods
  - PC - preconditioners and direct solvers
- More on PETSc objects
  - DM - distributed meshes
  - DMDA - structured distributed meshes
  - SNES - Newton-like methods for nonlinear systems
  - TS - time integrators and pseudo-transient continuation methods
- Putting it all together - Steady thermal and lid-driven cavity flow

Dive into petsc4py

Laplace's and Poisson's Equations

▶ Details

\[-\nabla^2u = 0\]

Note the usage of the minus in our definition, allowing us to deal with the Laplacian as a positive definite operator.

The inhomogenous potential equation with sources is well known as Poisson's equation:

\[-\nabla^2u = f\]

Discretization of Poisson's Equation

For this exercise, we select the second-order finite difference approximation to the solution, source, and Laplace operator:

$$\mathtt{x} \approx u, \\ \mathtt{b} \approx f, \\ \mathtt{A} \approx \nabla^2u$$

We would now like to solve the following system of linear equations: $$\mathtt{A}\mathtt{x} = \mathtt{b}$$

▶ Details

The PETSc Mat object

- The PETSc `Mat` object is a generic discrete linear operator.
- Layout is compressed row storage (Yale) by default
- (Parallel) rows are distributed across processors
- To every extent possible, `Mat` abstracts the *matrix data layout* from the code

Construction of a Python shell Mat object

▶ Details

```python
# number of nodes in each direction
# excluding those at the boundary
n = 32
# grid spacing
h = 1.0/(n+1)
A = PETSc.Mat().create()
A.setSizes([n**3, n**3])
A.setType('python')
shell = Del2Mat(n) # shell context
A.setPythonContext(shell)
A.setUp()
```

The PETSc Vec object

- The PETSc `Vec` object is a generic element of a discrete vector space.
- Scalar elements of the vector are distributed across processors in the same manner that `Mat` rows are
- Supports many common mathematical operations in conjunction with the `Mat` object
- To every extent possible, `Vec` abstracts the *parallel array layout* from the code

Setting up solution and source Vecs

▶ Details

```
x, b = A.getVecs()
# set the initial guess to 0
x.set(0.0)
# set the right-hand side to 1
b.set(1.0)
```

The PETSc KSP object

- The PETSc `KSP` object is a generic solution strategy for a system of linear equations
- The `KSP` object supports serial and parallel, iterative and direct linear solvers
- The name comes from Krylov Subspace, which refers to the collection of vectors spanned by the space: \([b Ab A^2b ... A^nb]\), and which is the basis for many iterative methods
- To every extent possible,`KSP` abstracts the *solver strategy* from the code

Setting up the Krylov solver (KSP)

▶ Details

```
ksp = PETSc.KSP().create()
ksp.setType('cg')
```

▶ Details

```
pc = ksp.getPC()
pc.setType('none')
```

▶ Details

```
ksp.setOperators(A)
ksp.setFromOptions()
ksp.solve(b, x)
```

Some notes on the syntactic sugar in petsc4py and matplotlib

The `poisson3d.py` demo contains two examples of syntactic sugar that should be noted:

▶ Details

```
X, Y =  mgrid[0:1:1j*n,0:1:1j*n]
```

▶ Details

```
Z = x[...].reshape(n,n,n)[:,:,n/2-2]
```

Exercise and Demonstration

▶ Details

DM - distributed meshes

- The PETSc `DM` object is a generic geometric discretization of a physical space

  ▶ Details

- When a stencil extent has been set, the `DM` object can build and maintain `local` and `global` vectors
  - `global` vectors perfectly decompose the spatial discretization
  - `local` vectors contain "ghost" values, read-only values needed by each processor to compute on its local data

- The `DMDA` object, which implements `DM` for simple Cartesian discretizations, provides more intuition on how `DM` works

DMDA - structured distributed meshes

- The PETSc `DMDA` object is a generic geometric discretization of a structured Cartesian physical space

  ▶ Details

- The `DMDA` object can represent structured grids in 1, 2, or 3 dimensions, and also provides convenient array pointers for computing on the boundaries and interior

- To every extent possible, the `DMDA` abstracts the *parallel data layout* from the code

SNES - Newton-like methods for nonlinear systems

- The PETSc `SNES` object is a generic solution strategy for a nonlinear system of equations of the form:

\[F(x) = 0 \]

▶ Details

- In the simplest case, the user provides a structured domain and a routine for computing \(F(x)\), and PETSc does the rest by approximating the Jacobian: `-snes_fd`

- `SNES` also supports matrix-free strategies: `-snes_mf`

- To every extent possible, `SNES` abstracts the *nonlinear solver strategy* from the code

TS - time integrators

- The PETSc `TS` object is a generic solution strategy for ordinary differential equations and differential algebraic equations

- Provides Forward Euler, Backward Euler, and a variety of Runge-Kutta and Strong Stability

Preserving time integrators

- Can be used to solve equations in steady-state via pseudo-timestepping
- To every extent possible, `TS` abstracts the *time-stepping strategy* from the code

Putting it all together - Steady thermal and lid-driven cavity flow