

# HPC Python Tutorial: Introduction to PETSc4Py 4/23/2012

Instructor:

Yaakoub El Khamra, Research Associate, TACC

[yaakoub@tacc.utexas.edu](mailto:yaakoub@tacc.utexas.edu)

# What is PETSc?

“PETSc...is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It supports MPI, [shared memory pthreads](#), and [NVIDIA GPUs](#), as well as hybrid MPI-shared memory pthreads or MPI-GPU parallelism”

# PETSc's Role

Developing parallel, nontrivial PDE solvers that deliver high performance is still difficult and requires months (or even years) of concentrated effort. PETSc is a toolkit that can ease these difficulties and reduce the development time, but it is not a black-box PDE solver, nor a silver bullet.

Barry Smith

# What can you use PETSc for?

Everything

# Seriously though

- Scientific Computations: parallel linear algebra, in particular linear and nonlinear solvers
- Toolkit: Contains high level solvers, but also the low level tools to roll your own.
- Portable: Available on many platforms, basically anything that has MPI

Why use it? It's big, powerful, well supported.

# A bit more about PETSc

## What does it target?

- Serial and Parallel
- Linear and nonlinear
- Finite difference and finite element
- Structured and unstructured

## What is in PETSc?

- Linear system solvers (sparse/dense, iterative/direct)
- Nonlinear system solvers
- Tools for distributed matrices
- Support for profiling, debugging, graphical output
- And much much more

# Interfaces to your favorite packages

- Most packages can be automatically Downloaded
- Configured and Built (in \$PETSC\_DIR/externalpackages)
- Installed with PETSc
- Currently works for
  - Petsc4py
  - PETSc documentation utilities (Sowing, lgrind, c2html)
  - BLAS, LAPACK, BLACS, ScaLAPACK, PLAPACK
  - MPICH, MPE, Open MPI
  - ParMetis, Chaco, Jostle, Party, Scotch, Zoltan
  - MUMPS, Spooles, SuperLU, SuperLU\_Dist, UMFPack, pARMS
  - PaStiX, BLOPEX, FFTW, SPRNG
  - Prometheus, HYPRE, ML, SPAI
  - Sundials
  - Triangle, TetGen

# Want to learn more?

- PETSc Tutorial at TACC
- Regular tutorials by PETSc developers
- The PETSc website:  
<http://www.mcs.anl.gov/petsc/>
- The PETSc documentation:  
<http://www.mcs.anl.gov/petsc/documentation/index.html>
- PETSc source code examples in the tarball



# PETSc4Py

- Python bindings for PETSc, the Portable Extensible Toolkit for Scientific Computation
- Implemented with Cython
- A good friend of petsc4py is:
  - mpi4py: Python bindings for MPI, the Message Passing Interface
- Other two projects depend on petsc4py:
  - slepc4py: Python bindings for SLEPc, the Scalable Library for Eigenvalue Problem Computations
  - tao4py: Python bindings for TAO, the Toolkit for Advanced Optimization

# Python for control and logic

## C for local computation

- Decouple organization of storage from mathematical operations
  - Vectors are not arrays
- Lots of small arrays
  - get/setValues() methods
- Views into larger arrays
- Dense, local computation is cache/bandwidth efficient

# Note on Installations

- Old style:
  - Configure PETSc using `-download-petsc4py`
  - Downloaded to `externalpackages/petsc4py-version`
  - Demo code is there as well
  - Installed into PETSc lib directory
  - Add `$PETSC_DIR/$PETSC_ARCH/lib` to `PYTHONPATH`
- New style:
  - `pip install -install-options=-user petscp4y`
  - Uses `$PETSC_DIR` and `$PETSC_ARCH`
  - Installed into `$HOME/.local`
  - No additions to `PYTHONPATH`

# Components

- **Index Sets**: permutations, indexing into vectors, renumbering.
- **Vectors**: sequential and distributed.
- **Matrices**: sequential and distributed, sparse and dense.
- **Distributed Arrays**: regular grid-based problems.
- **Linear Solvers**: Krylov subspace methods.
- **Preconditioners**: sparse direct solvers, multigrid
- **Nonlinear Solvers**: line search, trust region, matrix-free.
- **Time steppers**: time-dependent, linear and nonlinear PDE's

# PETSc4Py Interface

- Using PETSc4Py is very similar to using MPI4Py
- Provides ALL PETSc functionality in a Pythonic way
- Manages all memory (creation/destruction)
- Visualization with [matplotlib](#)

# PETSc4Py Basic Operations

- Create a sparse matrix, set its size and type:

```
A = PETSc.Mat()
```

```
A.create(PETSc.COMM_WORLD)
```

```
A.setSizes([m*n, m*n])
```

```
A.setType('mpiaij')
```

- Create a linear solver and solve:

```
ksp = PETSc.KSP()
```

```
ksp.create(PETSc.COMM_WORLD)
```

```
ksp.setOperators(A)
```

```
ksp.setFromOptions()
```

```
ksp.solve(b, x)
```

# Example

```
import petsc4py, sys
petsc4py.init(sys.argv)

from petsc4py import PETSc

# grid size and spacing
m, n = 32, 32
hx = 1.0/(m-1)
hy = 1.0/(n-1)

# create sparse matrix
A = PETSc.Mat()
A.create(PETSc.COMM_WORLD)
A.setSizes([m*n, m*n])
A.setType('aij') # sparse

# precompute values for setting
# diagonal and non-diagonal entries
diagv = 2.0/hx**2 + 2.0/hy**2
offdx = -1.0/hx**2
offdy = -1.0/hy**2
```

```
# loop over owned block of rows on this
# processor and insert entry values
Istart, Iend = A.getOwnershipRange()
for I in xrange(Istart, Iend) :
    A[I,I] = diagv
    i = I//n # map row number to
    j = I - i*n # grid coordinates
    if i > 0 : J = I-n; A[I,J] = offdx
    if i < m-1: J = I+n; A[I,J] = offdx
    if j > 0 : J = I-1; A[I,J] = offdy
    if j < n-1: J = I+1; A[I,J] = offdy

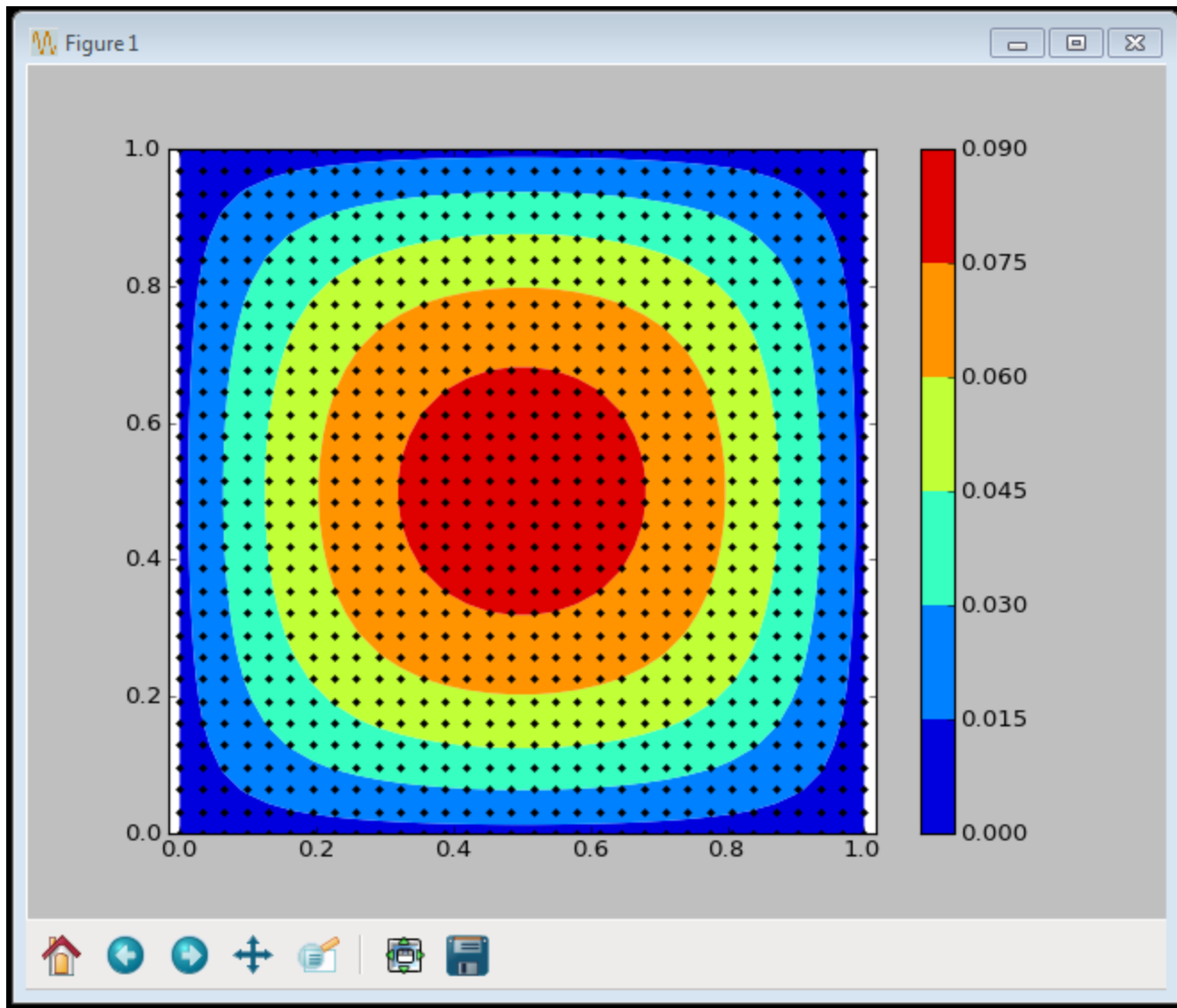
# communicate off-processor values
# and setup internal data structures
# for performing parallel operations
A.assemblyBegin()
A.assemblyEnd()
```

# Example (cont)

```
# create linear solver
ksp = PETSc.KSP()
ksp.create(PETSc.COMM_WORLD)
# use conjugate gradients
ksp.setType('cg')
# and incomplete Cholesky
ksp.getPC().setType('icc')
# obtain sol & rhs vectors
x, b = A.getVecs()
x.set(0)
b.set(1)
# and next solve
ksp.setOperators(A)
ksp.setFromOptions()
ksp.solve(b, x)
```

```
try:
    from matplotlib import pylab
except ImportError:
    raise SystemExit("matplotlib
not available")
from numpy import mgrid
X, Y = mgrid[0:1:1j*m,0:1:1j*n]
Z = x[...].reshape(m,n)
pylab.figure()
pylab.contourf(X,Y,Z)
pylab.plot(X.ravel(),Y.ravel(),'.k')
pylab.axis('equal')
pylab.colorbar()
pylab.show()
```





# Solid Fuel Ignition Problem

-Laplacian(u) - lambda \* exp(u) = 0,  $0 < x, y, z < 1$ ,

Boundary conditions:

$u = 0$  for  $x = 0, x = 1, y = 0, y = 1, z = 0, z = 1$

A finite difference approximation with the usual 7-point stencil is used to discretize the boundary value problem to obtain a nonlinear system of equations. The problem is solved in a 3D rectangular domain, using distributed arrays (DAs) to partition the parallel grid.

# Bratu3D: Bratu3D class

```
import sys, petsc4py
petsc4py.init(sys.argv)
```

```
from numpy import exp, sqrt
from petsc4py import PETSc
```

```
class Bratu3D(object):
```

```
    def __init__(self, da, lambda_):
        assert da.getDim() == 3
        self.da = da
        self.lambda_ = lambda_
        self.localX = da.createLocalVector()
```

```
    def formInitGuess(self, snes, X):
```

```
        #
        X.zeroEntries()
        corners, sizes = self.da.getCorners()
        x = X[...].reshape(sizes, order='F')
        #
        mx, my, mz = self.da.getSizes()
        hx, hy, hz = [1.0/m for m in [mx, my, mz]]
        lambda_ = self.lambda_
        scale = lambda_/(lambda_ + 1.0)
        #
        (xs, xe), (ys, ye), (zs, ze) = self.da.getRanges()
        for k in xrange(zs, ze):
            min_k = min(k, mz-k-1)*hz
            for j in xrange(ys, ye):
                min_j = min(j, my-j-1)*hy
                for i in xrange(xs, xe):
                    min_i = min(i, mx-i-1)*hx
                    if (i==0 or j==0 or k==0 or
                        i==mx-1 or j==my-1 or k==mz-1):
                        # boundary points
                        x[i, j, k] = 0.0
                    else:
                        # interior points
                        min_kij = min(min_i, min_j, min_k)
                        x[i, j, k] = scale*sqrt(min_kij)
```

```
    def formFunction(self, snes, X, F):
```

```
        #
        self.da.globalToLocal(X, self.localX)
        corners, sizes = self.da.getGhostCorners()
        x = self.localX[...].reshape(sizes, order='F')
        #
        F.zeroEntries()
        corners, sizes = self.da.getCorners()
        f = F[...].reshape(sizes, order='F')
        #
        mx, my, mz = self.da.getSizes()
        hx, hy, hz = [1.0/m for m in [mx, my, mz]]
        hxhyhz = hx*hy*hz
        hxhzdhy = hx*hz/hy;
        hyhzdhx = hy*hz/hx;
        hxhydhz = hx*hy/hz;
        lambda_ = self.lambda_
        #
        (xs, xe), (ys, ye), (zs, ze) = self.da.getRanges()
        for k in xrange(zs, ze):
            for j in xrange(ys, ye):
                for i in xrange(xs, xe):
                    if (i==0 or j==0 or k==0 or
                        i==mx-1 or j==my-1 or k==mz-1):
                        f[i, j, k] = x[i, j, k] - 0
                    else:
                        u = x[i, j, k] # center
                        u_e = x[i+1, j, k] # east
                        u_w = x[i-1, j, k] # west
                        u_n = x[i, j+1, k] # north
                        u_s = x[i, j-1, k] # south
                        u_u = x[i, j, k+1] # up
                        u_d = x[i, j, k-1] # down
                        u_xx = (-u_e + 2*u - u_w)*hyhzdhy
                        u_yy = (-u_n + 2*u - u_s)*hxhzdhy
                        u_zz = (-u_u + 2*u - u_d)*hxhydhz
                        f[i, j, k] = u_xx + u_yy + u_zz \
                            - lambda_*exp(u)*hxhyhz
```

# Bratu3D: Bratu3D class and DA Setup

```
def formJacobian(self, snes, X, J, P):  
    raise NotImplementedError  
    P.zeroEntries()  
    if J != P: J.assemble() # matrix-free operator  
    return PETSc.Mat.Structure.SAME_NONZERO_PATTERN
```

```
OptDB = PETSc.Options()  
  
N = OptDB.getInt('N', 16)  
lambda_ = OptDB.getReal('lambda', 6.0)  
do_plot = OptDB.getBool('plot', False)  
  
da = PETSc.DA().create([N, N, N])  
pde = Bratu3D(da, lambda_)  
  
snes = PETSc.SNES().create()  
F = da.createGlobalVector()  
snes.setFunction(pde.formFunction, F)  
  
fd = OptDB.getBool('fd', True)  
mf = OptDB.getBool('mf', False)  
if mf:  
    J = None  
    snes.setUseMF()  
else:  
    J = da.createMatrix()  
    snes.setJacobian(pde.formJacobian, J)  
    if fd:  
        snes.setUseFD()
```

# Bratu3D: SNES Solve and Plot

```
X = da.createGlobalVector()
pde.formInitGuess(None, X)

snes.getKSP().setType('cg')
snes.setFromOptions()
snes.solve(None, X)

U = da.createNaturalVector()
da.globalToNatural(X, U)
```

```
def plot(da, U):
    comm = da.getComm()
    scatter, U0 = PETSc.Scatter.toZero(U)
    scatter.scatter(U, U0, False,
PETSc.Scatter.Mode.FORWARD)
    rank = comm.getRank()
    if rank == 0:
        solution = U0[...]
        solution = solution.reshape(da.sizes,
order='f').copy()
        try:
            from matplotlib import pyplot
            pyplot.contourf(solution[:, :, N//2])
            pyplot.axis('equal')
            pyplot.show()
        except:
            raise
            pass
        comm.barrier()
        scatter.destroy()
        U0.destroy()

if do_plot: plot(da, U)

del pde, da, snes
del F, J, X, U
```

# Plot slice (:,:,N/2)

