

PETSc for Python

<http://petsc4py.googlecode.com>

Lisandro Dalcin¹
dalcinl@gmail.com

Christopher Kees²
cekees@gmail.com

¹Centro Internacional de Métodos Computacionales en Ingeniería
Consejo Nacional de Investigaciones Científicas y Técnicas
Santa Fe, Argentina

²Coastal and Hydraulics Laboratory
US Army Engineer Research and Development Center
Vicksburg, MS

SIAM CSE 2011
February 28 – March 4, 2011
Reno, Nevada

Outline

Overview

Vectors & Matrices

Linear Solvers

Nonlinear Solvers

Interoperability

Performance

Overview

Vectors & Matrices

Linear Solvers

Nonlinear Solvers

Interoperability

Performance

What is **petsc4py**?

Python bindings for **PETSc**, the *Portable Extensible Toolkit for Scientific Computation*.

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*.

Implementation

Implemented with Cython <http://www.cython.org>

- ▶ Code base far easier to write, maintain, and extend.
- ▶ Faster than other solutions (mixed Python and C codes).
- ▶ Easier to cross language boundaries (reuse C/C++/Fortran).

Implementation - Cython [1]

```
cdef import from "petsc.h":  
  
    ctypedef int    PetscInt  
    ctypedef float  PetscScalar  
  
    ctypedef void*  PetscVec  "Vec"  
    ctypedef void*  PetscMat  "Mat"  
    ctypedef void*  PetscKSP  "KSP"  
  
    int VecDot(PetscVec x, PetscVec y, PetscScalar *val)  
    int MatMult(PetscMat A, PetscVec x, PetscVec y)  
    int KSPSolve(PetscKSP ksp, PetscVec b, PetscVec x)  
  
cdef inline int CHKERR(int ierr) except -1:  
    if ierr != 0:  
        raise RuntimeError("error code %d" % ierr)  
    return 0
```

Implementation - Cython [2]

```
cdef class Vec:
    cdef PetscVec vec
    ...
    def dot(self, Vec other):
        cdef PetscScalar val
        ierr = VecDot(self.vec, other.vec, &val); CHKERR(ierr)
        return val
    ...
cdef class Mat:
    cdef PetscMat mat
    ...
    def mult(self, Vec x, Vec y):
        ierr = MatMult(self.mat, x.vec, y.vec); CHKERR(ierr)
    ...
cdef class KSP:
    cdef PetscKSP ksp
    ...
    def solve(self, Vec b, Vec x):
        ierr = KSPSolve(self.ksp, b.vec, x.vec); CHKERR(ierr)
```

Features – PETSc 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.
- ▶ **Timesteppers:** time-dependent, linear and nonlinear PDE's.

Features – Interoperability

Support for wrapping other PETSc-based codes.

- ▶ You can use **Cython** (from `petsc4py` `cimport` PETSc).
- ▶ You can use **SWIG** (*typemaps* provided).
- ▶ You can use **F2Py** (fortran attribute).

(more on this later ...)

Features – Easy installation

```
$ virtualenv --no-site-packages ENV
```

```
$ source ENV/bin/activate
```

```
(ENV) $ pip install slepc4py
```

```
Downloading/unpacking slepc4py
```

```
Downloading/unpacking petsc4py (from slepc4py)
```

```
Downloading/unpacking slepc (from slepc4py)
```

```
Downloading/unpacking petsc (from petsc4py->slepc4py)
```

```
Downloading/unpacking numpy (from petsc4py->slepc4py)
```

```
Installing collected packages: numpy, petsc, petsc4py,  
slepc, slepc4py
```

```
(ENV) $
```

Overview

Vectors & Matrices

Linear Solvers

Nonlinear Solvers

Interoperability

Performance

Vectors (Vec) – Conjugate Gradients Method

$cg(A, x, b, i_{max}, \epsilon)$:

$i \leftarrow 0$

$r \leftarrow b - Ax$

$d \leftarrow r$

$\delta_0 \leftarrow r^T r$

$\delta \leftarrow \delta_0$

while $i < i_{max}$ and

$\delta > \delta_0 \epsilon^2$ do :

$q \leftarrow Ad$

$\alpha \leftarrow \frac{\delta}{d^T q}$

$x \leftarrow x + \alpha d$

$r \leftarrow r - \alpha q$

$\delta_{old} \leftarrow \delta$

$\delta \leftarrow r^T r$

$\beta \leftarrow \frac{\delta}{\delta_{old}}$

$d \leftarrow r + \beta d$

$i \leftarrow i + 1$

```
1 def cg(A, b, x, imax=50, eps=1e-6):
2     """
3     A, b, x : matrix, rhs, solution
4     imax   : maximum iterations
5     eps    : relative tolerance
6     """
7     # allocate work vectors
8     r = b.duplicate()
9     d = b.duplicate()
10    q = b.duplicate()
11    # initialization
12    i = 0
13    A.mult(x, r)
14    r.aprx(-1, b)
15    r.copy(d)
16    delta_0 = r.dot(r)
17    delta = delta_0
18    # enter iteration loop
19    while (i < imax and
20           delta > delta_0 * eps**2):
21        A.mult(d, q)
22        alpha = delta / d.dot(q)
23        x.aprx(+alpha, d)
24        r.aprx(-alpha, q)
25        delta_old = delta
26        delta = r.dot(r)
27        beta = delta / delta_old
28        d.aprx(beta, r)
29        i = i + 1
30    return i, delta**0.5
```

Matrices (Mat) [1]

```
1  from petsc4py import PETSc
2
3  # grid size and spacing
4  m, n = 32, 32
5  hx = 1.0/(m-1)
6  hy = 1.0/(n-1)
7
8  # create sparse matrix
9  A = PETSc.Mat()
10 A.create(PETSc.COMM_WORLD)
11 A.setSizes([m*n, m*n])
12 A.setType('aij') # sparse
13 A.setFromOptions()
14
15 # precompute values for setting
16 # diagonal and non-diagonal entries
17 diagv = 2.0/hx**2 + 2.0/hy**2
18 offdx = -1.0/hx**2
19 offdy = -1.0/hy**2
```

Matrices (Mat) [2]

```
1  # loop over owned block of rows on this
2  # processor and insert entry values
3  Istart, Iend = A.getOwnershipRange()
4  for I in range(Istart, Iend) :
5      A[I,I] = diagv
6      i = I//n      # map row number to
7      j = I - i*n # grid coordinates
8      if i> 0 : J = I-n; A[I,J] = offdx
9      if i< m-1: J = I+n; A[I,J] = offdx
10     if j> 0 : J = I-1; A[I,J] = offdy
11     if j< n-1: J = I+1; A[I,J] = offdy
12
13 # communicate off-processor values
14 # and setup internal data structures
15 # for performing parallel operations
16 A.assemblyBegin()
17 A.assemblyEnd()
```

Overview

Vectors & Matrices

Linear Solvers

Nonlinear Solvers

Interoperability

Performance

Linear Solvers (KSP+PC)

```
1  # create linear solver,
2  ksp = PETSc.KSP()
3  ksp.create(PETSc.COMM_WORLD)
4
5  # use conjugate gradients method
6  ksp.setType('cg')
7  # and incomplete Cholesky
8  ksp.getPC().setType('icc')
9
10 # obtain sol & rhs vectors
11 x, b = A.getVecs()
12 x.set(0)
13 b.set(1)
14
15 # and next solve
16 ksp.setOperators(A)
17 ksp.setFromOptions()
18 ksp.solve(b, x)
```

Overview

Vectors & Matrices

Linear Solvers

Nonlinear Solvers

Interoperability

Performance

Nonlinear Solvers (SNES) [1]

```
1  from petsc4py import PETSc
2  from numpy import exp
3
4  m, n = 32, 32 # grid sizes
5  alpha = 6.8  # parameter
6
7  def Bratu2D(snes, X, F, alpha, m, n):
8      # NumPy array <- Vec
9      x = X.array.reshape(m, n)
10     f = F.array.reshape(m, n)
11     # setup 5-points stencil
12     u = x[1:-1, 1:-1] # center
13     uN = x[1:-1, :-2] # north
14     uS = x[1:-1, 2: ] # south
15     uW = x[ :-2, 1:-1] # west
16     uE = x[2:, 1:-1] # east
17     # compute nonlinear function
18     hx = 1.0/(m-1) # x grid spacing
19     hy = 1.0/(n-1) # y grid spacing
20     f[:, :] = x
21     f[1:-1, 1:-1] = \
22         (2*u - uE - uW) * (hy/hx) \
23         + (2*u - uN - uS) * (hx/hy) \
24         - alpha * exp(u) * (hx*hy)
```

Nonlinear Solvers (SNES) [2]

```
1  # create nonlinear solver
2  snes = PETSc.SNES().create()
3  # register the function in charge of
4  # computing the nonlinear residual
5  f = PETSc.Vec().createSeq(m*n)
6  snes.setFunction(Bratu2D, f, args=(alpha, m, n))
7
8  # configure the nonlinear solver
9  # to use a matrix-free Jacobian
10 snes.setUseMF(True)
11 snes.getKSP().setType('cg')
12 snes.setFromOptions()
13
14 # solve the nonlinear problem
15 b = None # rhs = 0
16 x = f.duplicate() # solution
17 snes.solve(b, x)
```

Overview

Vectors & Matrices

Linear Solvers

Nonlinear Solvers

Interoperability

Performance

Interoperability – Cython

```
1  from petsc4py.PETSc cimport *
2
3  cdef extern from "MyPDE.h":
4      ctypedef struct Params:
5          double alpha, beta, gamma
6          int FormFunction(PetscDA da, PetscVec x, PetscVec F, Params *p)
7          int FormJacobian(PetscDA da, PetscVec x, PetscMat J, Params *p)
8
9  cdef inline Params getParams(dict params):
10     cdef Params p
11     p.alpha = params.get('alpha', 0.0)
12     p.beta  = params.get('beta',  1.0)
13     p.gamma = params.get('gamma', 2.0)
14     return p
15
16 def computeFunction(DA da, Vec X, Vec F, **params):
17     cdef Params p = getParams(params)
18     ierr = FormFunction(da.da, X.vec, F.vec, &p)
19     assert ierr != 0
20
21 def computeJacobian(DA da, Vec X, Mat J, **params):
22     cdef Params p = getParams(params)
23     ierr = FormJacobian(da.da, X.vec, J.mat, &p)
24     assert ierr != 0
```

Interoperability – SWIG

```
1  %module MyPDE
2
3  %include petsc4py/petsc4py.i
4
5  %{
6  #include "MyPDE.h"
7  %}
8
9  typedef struct Params {
10     double alpha;
11     double beta;
12     double gamma;
13 } Params;
14
15 PetscErrorCode FormInitGuess(DA da, Vec x, Params *p);
16 PetscErrorCode FormFunction(DA da, Vec x, Vec F, Params *p);
17 PetscErrorCode FormJacobian(DA da, Vec x, Mat J, Params *p);
```

Interoperability – F2Py

```
1  python module MyPDE
2  interface
3
4      subroutine FormInitGuess(da, x, params, ierr)
5          integer, intent(in)      :: da, x
6          real(kind=8), intent(in) :: params(3)
7          integer, intent(out)     :: ierr
8      end subroutine FormInitGuess
9
10     subroutine FormFunction(da, x, F, params, ierr)
11         integer, intent(in)      :: da, x, F
12         real(kind=8), intent(in) :: params(3)
13         integer, intent(out)     :: ierr
14     end subroutine FormFunction
15
16     subroutine FormJacobian(da, x, J, params, ierr)
17         integer, intent(in)      :: da, x, J
18         real(kind=8), intent(in) :: params(3)
19         integer, intent(out)     :: ierr
20     end subroutine FormJacobian
21
22 end interface
23 end python module MyPDE
```

Overview

Vectors & Matrices

Linear Solvers

Nonlinear Solvers

Interoperability

Performance

Performance

Consider the following diffusive, unsteady, non-linear, scalar problem in the unit cube $\Omega = (x, y, z) = (0, 1)^3$

- ▶ PDE, boundary and initial conditions:

$$\begin{aligned} \frac{\partial \phi}{\partial t} - \nabla \cdot (\kappa(\phi) \nabla \phi) &= G \quad \text{on } \Omega \times (0, T] \\ \frac{\partial \phi}{\partial \mathbf{n}} &= 0 \quad \text{at } \partial\Omega \times [0, T] \\ \phi &= \phi^0 \quad \text{at } t = 0, \end{aligned} \tag{1}$$

- ▶ Nonlinear diffusion and line source term:

$$\begin{aligned} \kappa(\phi) &= \begin{cases} 1 & \text{if } \phi \geq 0 \\ \frac{1}{1+\phi^2} & \text{if } \phi < 0 \end{cases} \\ G(x = \frac{1}{4}, y = \frac{1}{4}, \frac{1}{2} < z < 1) &= -300 \end{aligned} \tag{2}$$

Performance –Discretization

- ▶ finite differences in space
- ▶ backward Euler in time

$$\frac{1}{\Delta t}(\phi_{i,j,k}^{n+1} - \phi_{i,j,k}^n) - L_{i,j,k}(\kappa, \phi^{n+1}) - G_{i,j,k} = 0 \quad (3)$$

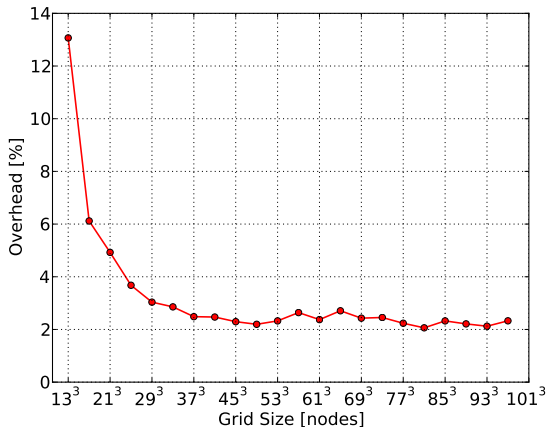
$$\begin{aligned} L_{i,j,k}(\kappa, \phi) = & \frac{\kappa[i - \frac{1}{2}]}{(\Delta x_1)^2} \phi[i - 1] - \frac{\kappa[i - \frac{1}{2}] + \kappa[i + \frac{1}{2}]}{(\Delta x_1)^2} \phi[0] + \frac{\kappa[i + \frac{1}{2}]}{(\Delta x_1)^2} \phi[i + 1] + \\ & + \frac{\kappa[j - \frac{1}{2}]}{(\Delta x_2)^2} \phi[j - 1] - \frac{\kappa[j - \frac{1}{2}] + \kappa[j + \frac{1}{2}]}{(\Delta x_2)^2} \phi[0] + \frac{\kappa[j + \frac{1}{2}]}{(\Delta x_2)^2} \phi[j + 1] + \\ & + \frac{\kappa[k - \frac{1}{2}]}{(\Delta x_3)^2} \phi[k - 1] - \frac{\kappa[k - \frac{1}{2}] + \kappa[k + \frac{1}{2}]}{(\Delta x_3)^2} \phi[0] + \frac{\kappa[k + \frac{1}{2}]}{(\Delta x_3)^2} \phi[k + 1] \end{aligned} \quad (4)$$

(notation: $[0] = (i, j, k)$, $[i \pm \frac{1}{2}] = (i \pm \frac{1}{2}, j, k)$, and so on)

Performance – Python+F90 versus C+F90

Basically a measure of Python calling overhead ...

- ▶ equivalent Python and C codes for the drivers
- ▶ common Fortran 90 code for the nonlinear residual
- ▶ use matrix-free for the nonlinear problems



Do not hesitate to ask for help ...

- ▶ Mailing List: `petsc-users@mcs.anl.gov`
- ▶ Mail&Chat: `dalcin1@gmail.com`
- ▶ Project Page:
`http://petsc4py.googlecode.com`

Thanks!