Software Engineering in Practice in the Scientific Python Community

Chris Kees and Aron Ahmadia
Coastal and Hydraulics Laboratory
US Army Engineer R&D Center

&

The Scientific Python Community
Outline

- Motivating example -- proteus (http://proteus.usace.army.mil)
- Numerical arrays -- numpy (http://www.numpy.org)
- Community repositories -- github (http://github.com) bitbucket (http://bitbucket.com).
- Self-documenting code -- Sphinx (http://sphinx-doc.org/)
- Continuous performance monitoring -- vbench (http://pydata.github.io/vbench/)
- Domain Specific Languages - FENICS (http://fenicsproject.org/documentation/tutorial/fundamentals.html)
- Web notebooks -- IPythonNotebook (http://ipython.org/notebook.html)
- Scientific Packaging -- HashDist (http://hashdist.github.io)
Coastal, Hydraulic, and Hydrologic Modeling

- Many challenges ahead:
  - Demand for fresh water
  - Growth and development on coastlines
  - Climate change
  - Sealevel rise
- Need to understand processes from very small to very large scales (soil grains to continental scale hydrology)
- PDE's are a major part of the language of hydrodynamic processes [Water Resources (Miller_Dawson_etal_2013.pdf)]
Proteus

- Designed as a prototype for new models & methods
- Chose Python for its rapid prototyping capabilities
- Original authors (Matthew Farthing and I) had 10 years of experience developing parallel PDE solvers in C++
- Turned into a long-term R&D platform
  - Parallelized using PETSc (mpi4py and petsc4py)
  - Many 2D and 3D models
  - Runs on super computers at HPCMP and TACC
Example

The equation we want to solve is

\[ m_t + \nabla \cdot (f - a \nabla u) + r = 0 \]

where \( u \) is the unknown solution and the coefficients have the specific forms

\[ m(u) = Mu \]
\[ f(u) = Bu \]
\[ a(u) = A \]
\[ r(u) = Cu \]
Define the equations
In [4]: class LADR(TransportCoefficients.TC_base):
    """
The coefficients of the linear advection-diffusion equation
    """
    def __init__(self,M,A,B,C):
        TransportCoefficients.TC_base.__init__(self,
            nc=1, #number of components
            variableNames=['u'],
            mass     = {0:0:'linear'},
            advection = {0:0:'linear'},
            diffusion = {0:0:{0:'constant'}},
            potential = {0:0:'u'},
            reaction  = {0:0:'linear'})
        self.M=M;
        self.A=A;
        self.B=B;
        self.C=C;
    def evaluate(self,t,c):
        c[('m',0)][:]       = self.M*c[('u',0)]
        c[('dm',0,0)][:]    = self.M
        c[('f',0)][:,...,0] = self.B[0]*c[('u',0)]
        c[('f',0)][:,...,1] = self.B[1]*c[('u',0)]
        c[('df',0,0)][:,...,0] = self.B[0]
        c[('df',0,0)][:,...,1] = self.B[1]
        c[('a',0,0)][:,...,0] = self.A[0][0]
        c[('a',0,0)][:,...,1] = self.A[1][1]
        c[('r',0)][:]      = self.C*c[('u',0)]
        c[('dr',0,0)][:]   = self.C
Define the physics
In [5]: from proteus import default_p as p  
# physics  
p.name = "lad_2d"  
p.nd = 2; # Two dimensions  
p.L=(1.0,1.0);  
p.T=1.0  

p.coefficients=LADR(M=1.0,  
    A=[[0.001,0.0],  
        [0.0,0.001]],  
    B=[2.0,1.0],  
    C=0.0)  

def getDBC(x,flag):  
    if x[0] == 0.0 or x[1] == 0.0:
        return lambda x,t: 1.0
    elif x[0] == p.L[0] or x[1] == p.L[1]:
        return lambda x,t: 0.0
    else:
        return None

p.dirichletConditions = {0:getDBC}  
p.advectiveFluxBoundaryConditions = {}  
p.diffusiveFluxBoundaryConditions = {0:{} }  
p.periodicDirichletConditions = None  

class IC:
    def __init__(self):  
        pass  
    def uOfXT(self,x,t):
        if x[0] <= 0.0 or x[1] <= 0.0:  
            return 1.0
        else:
            return 0.0

p.initialConditions  = {0:IC()}

Define numerics
In [6]:
from proteus import default_n as n
import proteus as pr
n.timeIntegration = pr.TimeIntegration.BackwardEuler_cfl
n.stepController = pr.StepControl.Min_dt_cfl_controller
n.runCFL = 1.0
n.elementQuadrature = pr.Quadrature.SimplexGaussQuadrature(p.nd, 3)
.n.femSpaces = {0:pr.FemTools.C0_AffineLinearOnSimplexWithNodalBasis}
.n.elementBoundaryQuadrature = pr.Quadrature.SimplexGaussQuadrature(p.nd-1,3)
.n.subgridError = pr.SubgridError.AdvectionDiffusionReaction_ASGS(p.coefficients,
    p.nd, lag=False)

n.shockCapturing = pr.ShockCapturing.ResGradQuad_SC(p.coefficients,
    p.nd,
    shockCapturingFactor=0.99,
    lag=True)

n.numericalFluxType = pr.NumericalFlux.Advection_DiagonalUpwind_Diffusion_SIPG_exterior
.n.nnx = 41; n.nny = 41
.n.tnList = [float(i)/40.0 for i in range(11)]
.n.matrix = pr.LinearAlgebraTools.SparseMatrix
.n.multilevelLinearSolver = pr.LinearSolvers.KSP_petsc4py
.n.l_atol_res = 1.0e-8
.n.parallelPartitioningType = pr.MeshTools.MeshParallelPartitioningTypes.element
.n.nLayersOfOverlapForParallel = 1
.n.periodicDirichletConditions = None
Solve and plot
In [7]:

```python
from proteus import default_s, default_so
so = default_so
so.name = p.name
so.sList = [default_s]
so.tnList = n.tnList
ns = NumericalSolution.NS_base(so, [p], [n], so.sList, opts)
failed = ns.calculateSolution('ladr_run1')
assert not failed
import pylab
x = ns.modelList[0].levelModelList[-1].mesh.nodeArray[:, 0]
y = ns.modelList[0].levelModelList[-1].mesh.nodeArray[:, 1]
triangles = ns.modelList[0].levelModelList[-1].mesh.elementNodesArray
u = ns.modelList[0].levelModelList[-1].u[0].dof
pylab.tricontourf(x, y, triangles, u)
pylab.show()
```
Video format or MIME type is not supported.
Scientific Python Community Codes and Practices

- Proteus depends on many open source packages (and is itself open source)
- We also copied many practices of those packages
- Now we'll take a tour of some of these packages and practices
Numpy

- The defacto standard numerical array type in Python
- Built on LAPACK, BLAS, and C math
In [8]:
import numpy as np
x = np.array([1.0, 2.0, 3.0, 4.0])
y = 2*x
z = x+y
A = np.vstack([x, y, z, x+y+z])
print x, y, z, A

[[ 1.  2.  3.  4.]
 [ 2.  4.  6.  8.]
 [ 3.  6.  9. 12.]
 [ 1.  2.  3.  4.]]
In [9]: A.shape
   np.linalg.eigvals(A)

Out[9]: array([  0.00000000e+00,   3.80000000e+01,  -1.77635684e-15,
         0.00000000e+00])
A.shape
np.linalg.eigvals(A)
In [10]:
A[1,0:1] = 0.0
A[2,0:2] = 0.0
A[3,0:3] = 0.0
x = np.linalg.solve(A,y)
print y - np.dot(A,x)

[ 0.  0.  0.  0. ]
More linear algebra resources

- numpy (http://www.numpy.org)
- mpi4py (https://bitbucket.org/mpi4py)
- petsc4py (https://bitbucket.org/petsc/petsc4py)
Putting version control into practice

- Git is great but community projects require repository servers
- After years of maintaining our own servers, we switched to github (http://github.com/erdc-cm) and bitbucket (http://bitbucket.com/erdc-cm)
- These tools come with project-integrated support for collaboration
  - Issue tracker
  - Wiki
  - Static website hosting
  - Developer Team management
  - Also easy to add hooks for continuous integration
Documentation

• No free lunches; it's just plain hard to do well
• But, you should have something to show for your effort
• Generate documentation from code (i.e. keep docs in the code)
• Sphinx and Doxygen are two good tools
  ■ Generate multiple formats from text (with markup)
  ■ Extract code structure
  ■ References across packages
In [11]:

```python
def foo(a,b):
    r""
    Add a and b

    This implementation is based on the algorithm in
    (Smarty 56)

    :param a: numpy array of length n
    :param b: numpy array of length n

    ..todo Add tests
    """
    return a+b
```

In [12]: foo
Out[12]: <function __main__.foo>

Documentation examples

- matplotlib (http://matplotlib.org)
- proteus (http://proteus.usace.army.mil)
Testing

- Part of code verification process
- Doing it right requires development outside the scope of most scientific projects
- [nose](https://nose.readthedocs.org/en/latest/) fills some of those gaps
In [13]:
    def test_foo():
        nose.tools.eq_(4, foo(1, 3))
In [14]: !cd ~/proteus/src/test && nosetests -v.

test_vec_create ... ok
test_mat_create ... ok
test_vec_scalar_math ... ok
test_mat_vec_math ... ok
test_norm_zero[l2Norm] ... ok
test_norm_zero[l1Norm] ... ok
test_norm_zero[lInfNorm] ... ok
test_norm_zero[l2NormAvg] ... FAIL

test_norm_zero[wl2Norm] ... ok
test_norm_zero[wl1Norm] ... ok
test_norm_zero[wlInfNorm] ... ok

test_norm_zero[energyNorm] ... ok

test_norm_homogeneity[l2Norm] ... ok
test_norm_homogeneity[l1Norm] ... ok
test_norm_homogeneity[lInfNorm] ... ok

test_norm_homogeneity[l2NormAvg] ... ok

test_norm_homogeneity[wl2Norm] ... ok
test_norm_homogeneity[wl1Norm] ... ok

test_norm_homogeneity[wlInfNorm] ... ok

test_norm_homogeneity[energyNorm] ... ok

test_norm_triangle_inequality[l2Norm] ... ok
test_norm_triangle_inequality[l1Norm] ... ok
Continuous Integration + performance

- Testing is great but performance is critical for HPC
- vbench (http://pydata.github.io/vbench/)
Continuous Integration + coverage

- skimage example (https://coveralls.io/builds/926102)
Optimization Considerations: DSL's

- Discrete Weak Form Language: FENICS (http://fenicsproject.org/documentation/tutorial/fundamentals.html)
- Continuous Strong Form Language: IBVP (https://github.com/ibvp/ibvp)
Package Management: HashDist

- Scientific software distribution is hard
  - multiple languages
  - mix of source-based and vendor-supplied packages
  - Long build times on supercomputers
- HashDist grew out of multiple users experience (failing at software distribution)
Package definition example: openblas

extends: [base_package]

sources:
- url: https://github.com/xianyi/OpenBLAS/archive/v0.2.8.tar.gz
  key: tar.gz:gwpvzutaj3lw3f5qm2nzxbdklgq5akbq

build_stages:
- name: make_and_install
  after: prologue
  handler: bash
  bash:
    |
    make {{extra_flags}}
    make {{extra_flags}} PREFIX=${ARTIFACT} install

when_build_dependency:
  - {set: 'BLAS_LDFLAGS', value: '-L${ARTIFACT}/lib -lopenblas'}
Host packages:

when_build_dependency:
- {set:'MPICC', value:'{{HOST_MPICC}}'}
- {set:'MPICXX', value:'{{HOST_MPICXX}}'}
- {set:'MPIF77', value:'{{HOST_MPIF77}}'}
- {set:'MPIF90', value:'{{HOST_MPIF90}}'}
- {set:'MPIEXEC', value:'{{HOST_MPIEXEC}}'}
Stack specification

extends:
- file: debian.yaml

packages:

  launcher:
  cmake:
    use: host-cmake
  python:
    host: false
    build_with: |
      bzip2, sqlite
  blas:
    use: host-blas
  daetk:
  mpi:
    use: mpich
  nose:
  hdf5:
  ipython:
  matplotlib:
  petsc:
    build_with: |
      parmetis
    download: |
Stateless package management

- A central feature of HashDist is that each package "build artifact" and software profile is identified by a cryptographic hash of the package inputs (source+upstream deps+recipe)
- A proven approach copied from the Nix project with modifications for working on HPC systems ("host packages")
- This design choices leads to many desirable features
  - Easy/fast switching between multiple built versions of packages and profiles
  - Reproducibility of builds
Getting started with HashDist

- HashDist Landing Page (http://hashdist.github.io)
- HashDist on Github (http://github.com/hashdist/hashdist)
IPython Notebook

- This talk is one example of a web notebook
- A notebook is a multi-language text document
- It allows mixing of code and documentation
- IPython also contains a powerful parallel support
Thanks!