Python Minisymposia

- MS52: Python-based Software for Solving Partial Differential Equations - Part I (NOW)
- MS62: Python-based Software for Solving Partial Differential Equations - Part II (Tue/4:30/Sierra 2)
- MS70: Python Software for Numerical Optimization (Wed/9:30/Sierra 2)
- MS80: Python in Scientific Computing - Part I (Wed/2:00/Sierra 2)
- MS89: Python in Scientific Computing - Part II (Wed/4:30/Sierra 2)
Lessons Learned and Open Issues from the Development of the Proteus Toolkit for Coastal and Hydraulics Modeling

https://adh.usace.army.mil/proteus

Chris Kees Matthew Farthing
christopher.e.kees@usace.army.mil
matthew.w.farthing@usace.army.mil

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

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

- Lea Jenkins (Clemson)
- John Chrispell (Tulane)
- Tim Kelley and Scott Pope (NCSU)
- Clint Dawson, Serge Prudhomme, Steve Mattis, Tim Povich (UT/ICES)
- Yuri Bazilevs (UCSD)
- Ido Akkerman, Stacy Howington, Amanda Hines, John Peters (ERDC)
Outline

- Overview
- Physics API
- Lessons Learned
What is Proteus?

- Proteus is a Python package for rapidly developing computer models and numerical methods.
- The implementation uses standard software engineering practices: object-oriented programming, loose coupling, iterative/incremental programming, “literate” programming.
- A strong boundary is maintained between physics implementation and numerical methods implementation (loose coupling).
- Has a layered API for model implementation. Highly optimized models for specific/detailed physics can be implemented by deriving from more generic models (iterative programming).
- Contains “wrapper” modules for a wide variety of 3rd party libraries (ADH, PETSc, triangle, tetgen,...)
History of Proteus

- USACE began development on the ADaptive Hydraulics (ADH) code in the 90’s.
- ADH is a C library/executable implementing parallel, $h$-adaptive, piecewise linear finite element methods for a variety of single-phase flow and transport models.
- In 2006 we started two research projects focused on continuum models of multi-phase flow at various scales.
- We decided to write a prototype for a new version of ADH with new models and methods as part of the research on multi-phase flow.
- Desired characteristics of the prototype: multi-level, multi-scale, multi-phase/component, variable-order, variable-continuity, highly modular, and customizable.
Equations Solved

- 2D & 3D incompressible Navier-Stokes (Unsteady/Steady, LES, RANS, VANS)
- 2D diffusive wave (overland flow)
- 2D shallow water
- 2D & 3D two-phase incompressible, immiscible flow (hybrid VOF/level set formulation with LES, etc.)
- 2D & 3D saturated groundwater
- 2D & 3D Richards’ equation (variably saturated groundwater, various constitutive models)
- 2D & 3D two-phase flow in porous media (continuum mixture formulation, incompressible or compressible)
- 2D & 3D density-dependent groundwater flow and salinity transport
- 2D & 3D eikonal equation (signed distance calculations)
- 2D & 3D linear elasticity
- 3D elastoplastic deformation (levee stability, Mohr-Coulomb material)
- 2D & 3D 6DOF solid/air/water interaction
- 1D, 2D, & 3D Poisson, Burgers, linear/nonlinear ADRE, Stokes, etc.
Framework

Geometry | Material Properties | Auxiliary Conditions

Physics Specification | Numerics Specification

Proteus Toolkit

FemTools
Reference Elements
Local Function Spaces
Space Mappings
Interpolation Conditions
Finite Element Spaces
Multigrid Projections

Assemblers
Generic Transport
Specialized Assemblers
Form compiler wrappers

Solvers
Newton-Krylov
Newton-Multigrid
Nonlinear Multigrid
Pseudo-transient
Specialized Solvers
Wrappers

Utilities
Archiving
Coprocessing
Geometry
Quadrature
Meshing

Time
Discretizations
Verified Numerics

- Continuous linear and quadratic polynomial spaces ($C^0 P^1$ and $C^0 P^2$) on simplicial elements (intervals, triangles, tetrahedra) with nodal (Lagrange) basis
- Continuous tensor product spaces ($C^0 Q^k$) on hexahedra with nodal basis
- Discontinuous complete polynomial spaces ($C^{-1} P^k$) on simplicial elements with monomial basis
- $P^1$ non-conforming simplicial elements (equivalent to Raviart-Thomas mixed element)
- Eulerian-Lagrangian Localized Adjoint Methods (ELLAMs) for advection-dominated processes
- Locally discontinuous Galerkin mixed elements with static condensation
- SIPG/NIPG/IIPG primal discontinuous elements
- Residual-based variational multiscale methods (RBVMS)
- Analytical Riemann solvers (numerical fluxes) for linear advection, two-phase flow in porous media, and shallow water
- Approximate Riemann solvers: Harten-Lax-van Leer (SWE), Rusanov (two-phase flow), Cheng-Shu (Hamilton-Jacobi)
- Velocity post-processing to enforce element-wise (local) conservation
Verification and Validation Test Problems

- Dam break experiments
- Marin free surface flow/object experiment
- Wigley hull tow tank experiment
- Beach erosion board
- Flow around a cylinder
- Driven cavity
- Rotating Gaussian
- Advection in a vortex
- Porous media, slope stability,...
- Poisseulle, Couette, and Decay of Vortex (low RE analytical solutions) 2D & 3D
Two-phase flow (prototype, parallel)
Two-phase flow (optimized, parallel, 2.5M tets)
Peer-Reviewed Verification and Validation


Some Characteristics of Popular Math Software

- Directed at an abstract formulation covering an important class of problems: $Ax = b, \ F(t, y, y') = 0$
- Have multiple layers of interfaces: `dgbsv` (simple interface), `dgbrtf + dgbtrs` (computational interface)
- Use robust and accurate numerics: LU with partial pivoting, BDF methods.
- Some separation between problem/data description and numerics (PetscMat, PetscKSP).
Popular Modeling Toolkit for PDE’s

- “The COMSOL multiphysics simulation environment facilitates all steps in the modeling process: defining your geometry, specifying your physics, meshing, solving and then post-processing your results.”
- “FEniCS is free software for automated solution of differential equations. We provide software tools for working with computational meshes, finite element variational formulations of PDEs, ODE solvers and linear algebra.”
- “…OpenFOAM is a flexible set of efficient C++ modules. These are used to build a wealth of: solvers, to simulate specific problems in engineering mechanics; utilities, to perform pre- and post-processing tasks ranging from simple data manipulations to visualisation and mesh processing; libraries, to create toolboxes that are accessible to the solvers/utilities, such as libraries of physical models.”
Our target problems are systems of nonlinear equations governing the transport of an abstract vector of components $u_j, j = 1, \ldots, n_c$:

$$\frac{\partial m^i}{\partial t} + \nabla \cdot \left( f^i - \sum_{k}^{n_c} a^{ik} \nabla \phi^k \right) + r^i + h^i(\nabla u) = 0$$

where $i = 1, \ldots n_c$. The large majority of models in hydrology are in this class.
Main elements of a computer model

- A set of PDE’s.
- A set of space-time domains.
- Initial/boundary conditions and material properties.
- Discretizations for PDE’s and solvers for finite dimensional systems.
- Auxiliary computations, post-processing schemes, visualization, archiving, ...

We divide these elements into the p-file (problem description module), n-file (numerics module), and batch file.
A simple example

For \((t, x, y) \in [0, T] \times [0, 1] \times [0, 1]\) find \(u\) such that

\[
(Mu)_t + \nabla \cdot [Bu - A\nabla u] = 0
\]
\[
u(0, x, y) = 0
\]
\[
u(t, x, 0) = u(t, 0, y) = 1
\]
\[
u(t, x, 1) = u(t, 1, x) = 0
\]
\[
M = 1
\]
\[
B = (1, 1)
\]
\[
A = 0.001I
\]
from proteus import *
from proteus.default_p import *
from adr import *

name = "ladr_2d_ldg"
nd = 2; L=(1.0,1.0,1.0); T=1.0
coefficients=LAD(M=1.0,

    A=[[0.01,0.0],
        [0.0,0.01]],
    B=[1.0,1.0])

def getDBC(x, flag):
    if x[0] == 0.0 or x[1] == 0.0:
        return lambda x,t: 1.0
    elif x[0] == 1.0 or x[1] == 1.0:
        return lambda x,t: 0.0
dirichletConditions = {0:getDBC}
diffusiveFluxBoundaryConditions = {0:{}}
sd=True
from proteus import *
from proteus.default_n import *
from ladr_2d_p import *
timeIntegration = BackwardEuler
femSpaces = {0:C0_AffineLinearOnSimplexWithNodalBasis}
elementQuadrature = SimplexGaussQuadrature(nd, 3)
elementBoundaryQuadrature = SimplexGaussQuadrature(nd-1, 3)
nnx=11; nny=11
tnList=[float(i)/10.0 for i in range(11)]
matrix = SparseMatrix
multilevelLinearSolver = LU
from proteus.TransportCoefficients import *
import numpy

class LAD(TC_base):
    def __init__(self,M,A,B):
        sdInfo = {(0,0):(numpy.arange(start=0,stop=3,
                          step=1,dtype='i'),
                   numpy.arange(start=0,stop=2,
                          step=1,dtype='i'))
        TC_base.__init__(self, nc=1,
                         variableNames=['u'],
                         mass={0:{0:'linear'}})
adr.py, cont’d

def evaluate(self, t, c):
    c[('m', 0)][:] = self.M * c[('u', 0)];
    c[('dm', 0, 0)][:] = self.M
    for i, u in enumerate(c[('u', 0)].flat):
        c[('f', 0)].flat[i*2+0] = self.B[0]*u
        c[('f', 0)].flat[i*2+1] = self.B[1]*u
        c[('df', 0, 0)].flat[i*2+0] = self.B[0]
        c[('df', 0, 0)].flat[i*2+1] = self.B[1]
        c[('a', 0, 0)].flat[i*2+0] = self.A[0][0]
        c[('a', 0, 0)].flat[i*2+1] = self.A[1][1]

advection={0:{0:'linear'}},
diffusion={0:{0:{0:'constant'}}},
potential={0:{0:'u'}},
reaction={0:{0:'linear'}},
hamiltonian={},
sparseDiffusionTensors=sdInfo)
Design Mistakes

- Coefficient storage dictionary layout is easy to use only in Python.
- Optimized C and Fortran coefficient routines have a very ugly, problem-specific interface.
- Optimized discretizations must use yet another interface.
Lessons Learned

- Error trapping and develop tools for physics should have been a higher priority
- Tutorials and examples for physics should have been a higher priority
- Config/build/distribute tools should have been a higher priority
Open Issues

- A new physics API
- UFL generation capability
- Config/build/dist tools
- Economics
- Social Psychology