Status of Python Bindings for OPM-Flow

R. Klöfkorn, Tor Harald Sandve, Michael Sargado, Håkon Hægland

Eichstätt, Feb 4, 2020

This work is partially supported by equinor

Some results will be published: Andreas Dedner, Robert Klöfkorn.
A Python Framework for Solving Advection-Diffusion Problems, FVCA9, 2020.

Outline

1. Status of Python Bindings for Flow
2. Python – Possible Future Applications
3. Summary and Outlook

Outline

Python bindings for OPM-Flow

Report steps in Python: SimulatorFullyImplicitBlackoilEbos.hpp

Python code

```
import sunbeam
from simulators import simulators
sim = simulators.BlackOilSimulator()
parsed = sunbeam.parse(
     'SPE1CASE1.DATA', recovery=[('PARSE_RANDOM_SLASH',
         sunbeam.action.ignore)])
sim.set_eclipse_state(parsed._state())
sim.set_deck(parsed._deck())
sim.set_schedule(parsed._schedule())
sim.set_summary_config(parsed._summary_config())
sim.step_init()
done = False
while not done:
    sim.step()  # One report step..
    sim.step_cleanup()
```

Split C++ `run` **method in the file above into** `run_init` , `run_step` , `run_last_step` .

Python bindings: Expose simulator state to Python

Next, we tried to expose the fluid state of C++ code to the Python script. We can read and set primary variables, but currently only read fluid state variables:

Python code

```
while not done:
    sim.step()   # One report step..
    # Primary var meaning:  0 = Sw_po_Sg, 1 = Sw_po_Rs, 2 = Sw_pg_Rv
    pvm = sim.get_primary_variables_meaning()
    # NB: only:  water_saturation, pressure, or composition
    pres = sim.get_primary_variable(idx_name='pressure')
    # Currently we can get one of the following variables
    # Sw, Sg, So, pw, po, pg, Rs, Rv, rho_w, rhow_o, rho_g
    oil_pres = sim.get_fluidstate_variable(name='po')
    pres = 1.01 * pres    # e.g.:  increase pressure by
    sim.set_primary_variable(idx_name='pressure', value=pres)
    # NB: Not implemented yet:
    #sim.set_fluidstate_variable(name='Rs', value=Rs)
```

Python bindings: Coupling of tools

Then we added reading/writing of porosity:

Python code

```
while not done:
      sim.step()  # One report step..
      poro = sim.get_porosity()
      print(poro)
      poro = poro*1.01 # E.g.  increase by 1
      sim.set_porosity(poro)  # update porosity for next report step
```

Use Python to couple different simulation tools!

Outline

Model described in Python using UFL

Compressible Euler Model in 2d

```
class Model:
    def toPrim(U): # auxiliary function
        v = as_vector( [U[i]/U[0] for i in range(1,3)] )
        pressure = 0.4*(U[3]-dot(v,v)*U[0]/2)
        return U[0], v, pressure
    def F_c(t,x,U):
        rho, v, p = Model.toPrim(U)
        return as_matrix([
           [rho*v[0],         rho*v[1]],
           [rho*v[0]*v[0]+p, rho*v[0]*v[1]],
           [rho*v[0]*v[1],   rho*v[1]*v[1]+p],
           [(U[3]+p)*v[0],    (U[3]+p)*v[1]] ])
    boundary = {range(1,5): lambda t,x,U: U}
    def maxLambda(t,x,U,n):
        rho, v, p = Model.toPrim(U)
        return abs(dot(v,n)) + sqrt(1.4*p/rho)
```

Create an operator

setup of case

```
gridView = structuredGrid([-1,-1],[1,1],[40,40])
space    = dgonb( gridView, order=3, dimRange=4)
uh       = space.interpolate([1.4,0,0,1], name='uh')
# this is where the magic happens
operator = femDGOperator(Model, space, limiter=None)
```

time loop

```
stepper = femdgStepper(order=3, operator=operator)
t = 0
while t < 0.1:
    operator.setTime(t)
    t += stepper(uh, dt=0.001)
```

Generated Code

femoperator

```
PYBIND11_MODULE( femoperator_b963c02389dad7ffdc4e16848899b3c2, module )
{
  using pybind11::operator""_a;
  pybind11::module cls0 = module;
  {
    typedef Dune::Fem::DGOperator< Dune::Fem::AdaptiveDiscreteFunction<..
    Dune::Python::insertClass< Dune::Fem::SpaceOperatorInterface< Dune::
    auto cls = Dune::Python::insertClass< Dune::Fem::DGOperator< Dune::F
    Dune::FemPy::registerOperator( cls0, cls );
    cls.def( "estimateMark",
             []( DuneType &self,
             const typename DuneType::DestinationType &u,
             const double dt) { self.estimateMark(u, dt); } );
  }
}
```

generated files and python modules

```
femoperator_b963c02389dad7ffdc4e16848899b3c2.cc
femoperator_b963c02389dad7ffdc4e16848899b3c2.so
modelFemDgAdv_nvff4652a5e9dec7e4e107ae79de4ff29d_19659fe2f2f74bcfb5d532f1a0d38be9.c
modelFemDgAdv_nvff4652a5e9dec7e4e107ae79de4ff29d_19659fe2f2f74bcfb5d532f1a0d38be9.s
```

Autogenerated Model

generated flux method

```
template< class Point >
void flux ( const Point &x, const DRangeType &u,
            const DJacobianRangeType &du, RJacobianRangeType &result ) const
{
  (result[ 0 ])[ 0 ] = u[ 0 ] * (u[ 1 ] / u[ 0 ]);
  (result[ 0 ])[ 1 ] = u[ 0 ] * (u[ 2 ] / u[ 0 ]);
  (result[ 1 ])[ 0 ] = 0.4 * (u[ 3 ] + -1 * ((u[ 0 ] * ((u[ 1 ] ...
  (result[ 1 ])[ 1 ] = (u[ 0 ] * (u[ 1 ] / u[ 0 ])) * (u[ 2 ] / ...
  (result[ 2 ])[ 0 ] = (u[ 0 ] * (u[ 1 ] / u[ 0 ])) * (u[ 2 ] / ...
  (result[ 2 ])[ 1 ] = 0.4 * (u[ 3 ] + -1 * ((u[ 0 ] * ((u[ 1 ] ...
  (result[ 3 ])[ 0 ] = (u[ 3 ] + 0.4 * (u[ 3 ] + -1 * ((u[ 0 ] *...
  (result[ 3 ])[ 1 ] = (u[ 3 ] + 0.4 * (u[ 3 ] + -1 * ((u[ 0 ] *...
}
```

Comparison original and python/autogenerated code

Table 1: Performance comparison of the C++ and the Python code for a simple test example solving the Euler equations in 2d with an explicit time stepping.

	SPGrid		
code \ #el	1024	4096	16384
C++	7.19	57.45	464.85
Python	7.04	56.29	457.23
C++ / Python	1.02	1.02	1.017

	ALUGrid		
code \ #el	1024	4096	16384
C++	12.72	106.08	884.28
Python	13.32	110.48	924.81
C++ / Python	0.955	0.96	0.956

Outline

Summary and Outlook

Python bindings

- Initial Python bindings are there

Summary and Outlook

NORCE

Python bindings

- Initial Python bindings are there
- **Ongoing:** Coupling with other tools coming this year

Summary and Outlook

NORCE

Python bindings

- Initial Python bindings are there
- **Ongoing:** Coupling with other tools coming this year
- **Future:** Use UFL to create smaller OPM setup cases

Thank you for your attention.