FEniCS: Discontinuous Galerkin Example

M. M. Sussman

sussmanm@math.pitt.edu

Office Hours: 11:10AM-12:10PM, Thack 622

May 12 – June 19, 2014

Discontinuous Galerkin Methods

- Beatrice Riviere, Discontinuous Galerkin Methods for Solving Elliptic and Parabolic Equations: Theory and Implementation. SIAM 2008. ISBN-10: 089871656X
- B. Cockburn, Discontinuous Galerkin Methods, **`https://www.ima.umn.edu/preprints/may2003/1921.pdf`**

Advantages and disadvantages

- Originally for “conservation laws”
- Convection-diffusion
- Elementwise conservation
- High order methods without large matrices
- Trick is to couple elements stably, accurately, efficiently.
- Require computations on facets, including jumps and averages

ExampleDG

- Equation
$$u \cdot \nabla \phi - \kappa \Delta \phi = f$$
- From **`undocumented/dg-advection-diffusion/python/demo_dg-advection-diffusion.py`**
- Dirichlet boundary on right side
- Dirichlet values $\sin(5\pi y)$
- Convecting velocity (-1,-0.4).

exampleDG.py code

```
"""
exampleDG.py, from
undocumented/dg-advection-diffusion/python/demo_dg-advection-diffusion.py

Steady state advection-diffusion equation,
discontinuous formulation using full upwinding.

Implemented in python from cpp demo by Johan Hake.
"""
from dolfin import *

class DirichletBoundary(SubDomain):
    def inside(self, x, on_boundary):
        return abs(x[0] - 1.0) < DOLFIN_EPS and on_boundary

# mesh
mesh = UnitSquareMesh(64,64)

# Defining the function spaces
V_dg = FunctionSpace(mesh, "DG", 1)
V_cg = FunctionSpace(mesh, "CG", 1)
V_u  = VectorFunctionSpace(mesh, "CG", 2)

# Create velocity Function
u = interpolate( Constant(("-1.","-0.4")), V_u)

# Test and trial functions
v   = TestFunction(V_dg)
phi = TrialFunction(V_dg)
```

`ExampleDG.py` code

```
# Diffusivity
kappa = Constant(0.0)

# Source term
f = Constant(0.0)
```

ExampleDG.py code

```
# Diffusivity
kappa = Constant(0.0)

# Source term
f = Constant(0.0)

# Penalty term
alpha = Constant(5.0)
```

ExampleDG.py code

```
# Diffusivity
kappa = Constant(0.0)

# Source term
f = Constant(0.0)

# Penalty term
alpha = Constant(5.0)

# Mesh-related functions
n = FacetNormal(mesh)
h = CellSize(mesh)
h_avg = (h('+') + h('-'))/2

# ( dot(v, n) + |dot(v, n)| )/2.0
un = (dot(u, n) + abs(dot(u, n)))/2.0
```

ExampleDG.py code

```
# Diffusivity
kappa = Constant(0.0)

# Source term
f = Constant(0.0)

# Penalty term
alpha = Constant(5.0)

# Mesh-related functions
n = FacetNormal(mesh)
h = CellSize(mesh)
h_avg = (h('+') + h('-'))/2

# ( dot(v, n) + |dot(v, n)| )/2.0
un = (dot(u, n) + abs(dot(u, n)))/2.0

# Bilinear form
a_int = dot(grad(v), kappa*grad(phi) - u*phi)*dx
```

ExampleDG.py code

```
# Diffusivity
kappa = Constant(0.0)

# Source term
f = Constant(0.0)

# Penalty term
alpha = Constant(5.0)

# Mesh-related functions
n = FacetNormal(mesh)
h = CellSize(mesh)
h_avg = (h('+') + h('-'))/2

# ( dot(v, n) + |dot(v, n)| )/2.0
un = (dot(u, n) + abs(dot(u, n)))/2.0

# Bilinear form
a_int = dot(grad(v), kappa*grad(phi) - u*phi)*dx

a_fac = kappa('+')*(alpha('+')/h('+'))*dot(jump(v, n), jump(phi, n))*dS \
      - kappa('+')*dot(avg(grad(v)), jump(phi, n))*dS \
      - kappa('+')*dot(jump(v, n), avg(grad(phi)))*dS
h, jump, n, avg, +, -
```

ExampleDG.py code

```
# Diffusivity
kappa = Constant(0.0)

# Source term
f = Constant(0.0)

# Penalty term
alpha = Constant(5.0)

# Mesh-related functions
n = FacetNormal(mesh)
h = CellSize(mesh)
h_avg = (h('+') + h('-'))/2

# ( dot(v, n) + |dot(v, n)| )/2.0
un = (dot(u, n) + abs(dot(u, n)))/2.0

# Bilinear form
a_int = dot(grad(v), kappa*grad(phi) - u*phi)*dx

a_fac = kappa('+')*(alpha('+')/h('+'))*dot(jump(v, n), jump(phi, n))*dS \
      - kappa('+')*dot(avg(grad(v)), jump(phi, n))*dS \
      - kappa('+')*dot(jump(v, n), avg(grad(phi)))*dS

a_vel = dot(jump(v), un('+')*phi('+') - un('-')*phi('-') )*dS \
      + dot(v, un*phi)*ds

a = a_int + a_fac + a_vel
```

ExampleDG.py code

```
# Linear form
L = v*f*dx

# Set up boundary condition (apply strong BCs)
g = Expression("sin(pi*5.0*x[1])")
bc = DirichletBC(V_dg, g, DirichletBoundary(), "geometric")

# Solution function
phi_h = Function(V_dg)

# Assemble and apply boundary conditions
A = assemble(a)
b = assemble(L)
bc.apply(A, b)

# Solve system
solve(A, phi_h.vector(), b)

# Project solution to a continuous function space
up = project(phi_h, V=V_cg)

file = File("temperature.pvd")
file << up

# Plot solution
plot(up, interactive=True)
```

Solution