Numerical Treatment of Differential Equations: Homework 10

Truman Ellis

April 18, 2011

We wish to demonstrate that

$$\inf_{p\in Q_{k,k}} \|u-p\|_{m,2,E} \leq Ch^{k+1-m} |u|_{k+1,2,E}$$

for $k=1$ and both $m=0,1$. Thus we need to show that

$$\inf_{p\in Q_{1,1}} \|u-p\|_{0,2,E} \leq Ch^2 \left(\|u_{xx}\|_{0,2,E}^2 + \|u_{xy}\|_{0,2,E}^2 + \|u_{yy}\|_{0,2,E}^2\right)^{\frac{1}{2}},$$

and

$$\inf_{p\in Q_{1,1}} \left(\|u-p\|_{0,2,E}^2 + \|u_x-p_x\|_{0,2,E}^2 + \|u_y-p_y\|_{0,2,E}^2\right)^{\frac{1}{2}} \leq Ch \left(\|u_{xx}\|_{0,2,E}^2 + \|u_{xy}\|_{0,2,E}^2 + \|u_{yy}\|_{0,2,E}^2\right)^{\frac{1}{2}}.$$

In order to do this, I put together the skeleton of a finite element code. I chose to use an exact function:

$$u_{ex}(x,y) = \sin(\pi x)\sin(2\pi y)\,.$$

Then I prescribed the nodal degrees of freedom to be the exact solution at the nodal points. Then I was able to loop over each element and numerically integrate the square of the error by third order Gaussian Quadrature. Summing the contributions from all elements and taking the square root gave me the global error at each resolution. When we plot the log of the error vs $\log(h)$, the other terms on the right hand side of the inequality are just constants that shift the plot up. Thus we can demonstrate the correct behavior by looking at the slope of the plots. As expected, we get second order accuracy for $m=0$ and first order for $m=1$.

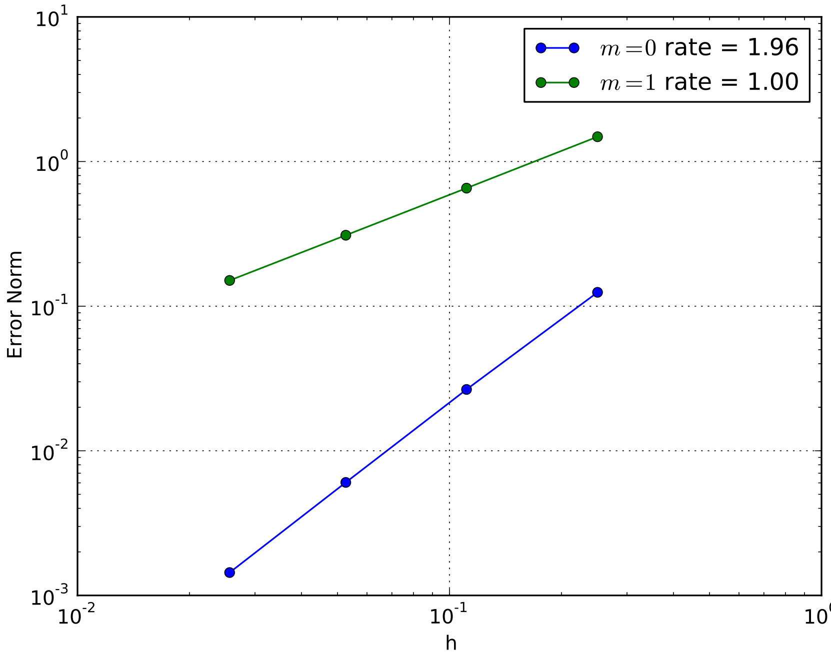

Figure 1: Error norms for $m = 0, 1$

```
# Code to test the convergence of Qk,k polynomials
# Written by: Truman Ellis
# Numerical Treatment of Differential Equations
# Spring 2011

from pylab import *
close('all')

# Define Initial Refinement
nx = 5
ny = 5

# Define number of refinements
nref = 3

# Define problem domain
xmin = 0.
xmax = 1.
ymin = 0.
ymax = 1.

# Define exact solution for convergence tests
def exact(x, y):
    return sin(pi*x)*sin(2*pi*y)

def gradexact(x, y):
    return array([pi*cos(pi*x)*sin(2*pi*y), 2*pi*sin(pi*x)*cos(2*pi*y)])

# Define element and node mapping
def mapping(i, j, nx):
    return i + j*nx

# Q1 basis functions
def Q1basis(x, y):
    return array([0.25*(1-x)*(1-y),\
                  0.25*(x+1)*(1-y),\
                  0.25*(x+1)*(y+1),\
                  0.25*(1-x)*(y+1)])
def GradQ1basis(x, y):
    return array([[-0.25*(1-y), -0.25*(1-x)],\
                   [0.25*(1-y), -0.25*(x+1)],\
                   [0.25*(y+1),  0.25*(x+1)],\
                  [-0.25*(y+1),  0.25*(1-x)]])

#Jacobian of transformation
def Jacobian((X,Y), (x,y)):
    return array([[0.25*((1-y)*(X[1]-X[0])+(y+1)*(X[2]-X[3])), \
                   0.25*((1-y)*(Y[1]-Y[0])+(y+1)*(Y[2]-Y[3]))],\
                  [0.25*((1-x)*(X[3]-X[0])+(x+1)*(X[2]-X[1])), \
                   0.25*((1-x)*(Y[3]-Y[0])+(x+1)*(Y[2]-Y[1]))]])

# Define quadrature points and weights
# Third order Gauss-Legendre Quadrature
np = 9
P = array([[-sqrt(3./5.), -sqrt(3./5.)],\
           [-sqrt(3./5.), 0],\
           [-sqrt(3./5.), sqrt(3./5.)],\
           [0, -sqrt(3./5.)],\
           [0, 0],\
           [0, sqrt(3./5.)],\
           [sqrt(3./5.), -sqrt(3./5.)],\
           [sqrt(3./5.), 0],\
           [sqrt(3./5.), sqrt(3./5.)]])
W = ([25./81., 40./81, 25./81, 40./81, 64./81, 40./81, 25./81, 40./81, 25./81])

# Calculate error in element defined by nodal points (X,Y) and DOFs u
def ElementError((X,Y), u):
    error = 0
    graderror = 0
    for i in range(0,np):
        # Local points
        xl = P[i,0]
        yl = P[i,1]
        # Global points
        xg = dot(Q1basis(xl,yl), X)
        yg = dot(Q1basis(xl,yl), Y)
        J = Jacobian((X,Y), (xl,yl))
        # m = 0 Error
        error += W[i]*(exact(xg,yg) - dot(Q1basis(xl,yl),u))**2*det(J)
        # m =1 Error
        gex = gradexact(xg, yg)
        gQ1 = dot(inv(J),dot(GradQ1basis(xl,yl).T,u))
        graderror += error
        graderror += W[i]*((gex[0]-gQ1[0]))**2*det(J)
```

```
        graderror += W[i]*((gex[1]-gQ1[1]))**2*det(J)
    return error, graderror

# Solve and refine
GlobalError = zeros(nref+1)
GradGlobalError = zeros(nref+1)
h = zeros(nref+1)
for r in range(0,nref+1):
    if r > 0:
        nx *= 2
        ny *= 2
    h[r] = 1./(nx-1)

    # Construct grid
    x = linspace(xmin,xmax,nx)
    y = linspace(ymin,ymax,ny)
    X, Y = meshgrid(x,y)
    X = X.flatten()
    Y = Y.flatten()
    NE = (nx-1)*(ny-1)

    # Compute mesh topology
    topo = zeros((NE, 4), dtype=int32)
    for i in range(0, nx-1):
        for j in range(0, ny-1):
            topo[mapping(i,j,nx-1)] = [i + j*nx, \
                                       i + 1 + j*nx, \
                                       (j+1)*nx + i + 1, \
                                       (j+1)*nx + i]

    # Compute approximate solution
    u = zeros( nx*ny )
    for i in range(0,nx):
        for j in range(0, ny):
            u[mapping(i,j,nx)] = exact(x[i], y[j])

    # Calculate global error
    for e in range(0,NE):
        LocalError, GradLocalError = ElementError( (X[topo[e]],Y[topo[e]]), u[topo[e]] )
        GlobalError[r] += LocalError
        GradGlobalError[r] += GradLocalError
    GlobalError[r] = sqrt(GlobalError[r])
    GradGlobalError[r] = sqrt(GradGlobalError[r])

# Plot results
figure()
hold(True)
loglog(h,GlobalError,'-o')
(m,b) = polyfit(log(h),log(GlobalError),1)
loglog(h,GradGlobalError,'-o')
(dm,b) = polyfit(log(h),log(GradGlobalError),1)
legend(('$m=0$ rate = %.2f'%(m,), '$m=1$ rate = %.2f'%(dm,)), loc='best')
xlabel('h')
ylabel('Error Norm')
grid(True)
show()
```