ISC 5935 - Computational Tools for Finite Elements

Homework#5: Solutions

Assigned 1 October 2014, Due 8 October 2014

http://people.sc.fsu.edu/jburkardt/classes/fem 2014/homework5.pdf

1. Contents of `bvp_01_mesh.xml`:

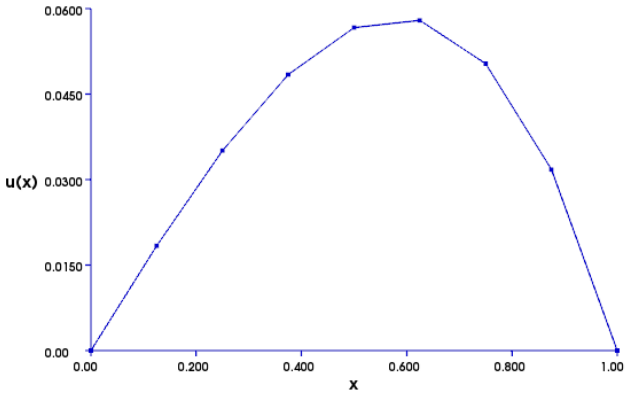
```
<?xml version="1.0"?>
<dolfin xmlns:dolfin="http://fenicsproject.org">
  <mesh celltype="interval" dim="1">
    <vertices size="9">
      <vertex index="0" x="0.000000000000000e+00" />
      <vertex index="1" x="1.250000000000000e-01" />
      <vertex index="2" x="2.500000000000000e-01" />
      <vertex index="3" x="3.750000000000000e-01" />
      <vertex index="4" x="5.000000000000000e-01" />
      <vertex index="5" x="6.250000000000000e-01" />
      <vertex index="6" x="7.500000000000000e-01" />
      <vertex index="7" x="8.750000000000000e-01" />
      <vertex index="8" x="1.000000000000000e+00" />
    </vertices>
    <cells size="8">
      <interval index="0" v0="0" v1="1" />
      <interval index="1" v0="1" v1="2" />
      <interval index="2" v0="2" v1="3" />
      <interval index="3" v0="3" v1="4" />
      <interval index="4" v0="4" v1="5" />
      <interval index="5" v0="5" v1="6" />
      <interval index="6" v0="6" v1="7" />
      <interval index="7" v0="7" v1="8" />
    </cells>
  </mesh>
</dolfin>
```

Contents of `bvp_01_u.xml`:

```
<?xml version="1.0"?>
<dolfin xmlns:dolfin="http://fenicsproject.org">
  <function_data size="9">
    <dof index="0" value="0" cell_index="0" cell_dof_index="0" />
    <dof index="1" value="0.018379439835948499" cell_index="0" cell_dof_index="1" />
    <dof index="2" value="0.035088583690214732" cell_index="1" cell_dof_index="1" />
    <dof index="3" value="0.048430969064432103" cell_index="2" cell_dof_index="1" />
    <dof index="4" value="0.056657390507543967" cell_index="3" cell_dof_index="1" />
    <dof index="5" value="0.057938496919442431" cell_index="4" cell_dof_index="1" />
    <dof index="6" value="0.050336133309191178" cell_index="5" cell_dof_index="1" />
```

```
    <dof index="7" value="0.031772978053653124" cell_index="6" cell_dof_index="1" />
    <dof index="8" value="0" cell_index="7" cell_dof_index="1" />
  </function_data>
</dolfin>
```

Plot

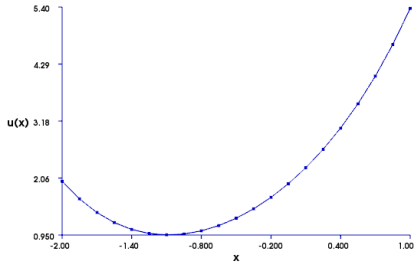


2. a) Weak form: Find u so that $u(-2) = 2$ and

$$\int_{-2}^{1} u'(x)v'(x)\,dx + \int_{-2}^{1} u(x)v(x)\,dx = \int_{-2}^{1} xv(x)\,dx + 5 * v(1)$$

for all $v \in V = \{v(x) \in H^1([-2,1]) : v(-2) = 0\}$.

Plot:

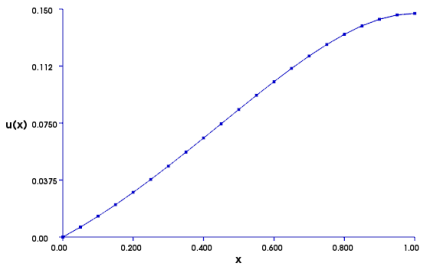


b) Weak form: Find u so that $u(0) = 0$ and

$$\int_{0}^{1} u'(x)v'(x)\,dx + \int_{0}^{1} u(x)v(x)\,dx + 2\int_{0}^{1} u'(x)v(x)\,dx = \int_{0}^{1} xv(x)\,dx$$

for all $v \in V = \{v(x) \in H^1([0,1]) : v(0) = 0\}$. Notice that since the Neumann condition is 0, we don't need to add any boundary integral.

Plot:



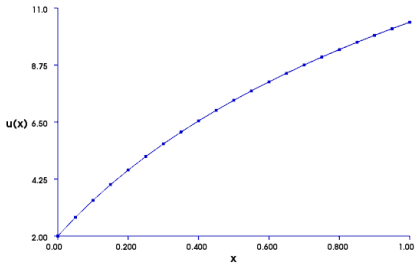
c) Weak form: Find u so that $u(0) = 2$ and

$$\int_0^1 (1+2x)u'(x)v'(x)\,dx - (1+2x)u'(x)v(x)\big|_{x=0}^{x=1} = \int_0^1 xv(x)\,dx, \text{ or}$$

$$\int_0^1 (1+2x)u'(x)v'(x)\,dx - 3*5*v(1) = \int_0^1 xv(x)\,dx$$

for all $v \in V = \{v(x) \in H^1([0,1]) : v(0) = 0\}$. Notice how the diffusion coefficient $1+2x$ also appears in the Neumann condition.

Plot:



Here is the code I used:

```
from dolfin import *

# ----------------------------------------------------------------------------
# Problem a):
# ----------------------------------------------------------------------------
# Mesh and Function Space
mesh = IntervalMesh(20,-2,1)
V    = FunctionSpace(mesh,"CG",1)

# Define Parameters
fa = Expression('x[0]')

# Define Dirichlet boundary conditions at the left endpoint
def q2a_boundary_left(x,on_boundary):
  return on_boundary and abs(x+2) < DOLFIN_EPS

bc = DirichletBC(V,Constant('2.0'),q2a_boundary_left)

# Formulate the weak form a(u,v) = L(v)
u = TrialFunction(V)
```

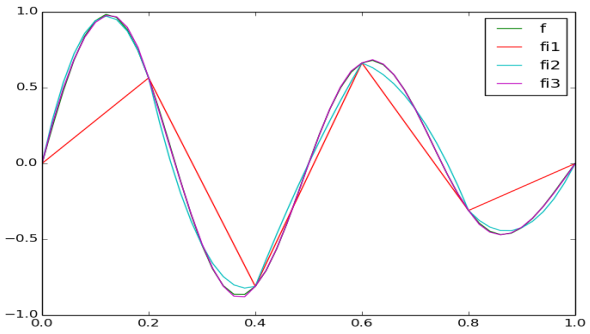
```
v = TestFunction(V)
a = inner(grad(u),grad(v))*dx + u*v*dx
L = fa*v*dx + Constant('5.0')*v*ds

# Solve the problem
u = Function(V)
solve(a==L,u,bc)

# Make a plot
plot(u,interactive=True)


# -----------------------------------------------------------------------------
# Problem b)
# -----------------------------------------------------------------------------
# Mesh and Function Space
mesh = IntervalMesh(20,0.0,1.0)
V    = FunctionSpace(mesh,"CG",1)

# Forcing term is the same as for a)

# Define the Dirichlet boundary condition on the left
def q2b_boundary_left(x,on_boundary):
  return on_boundary and abs(x) < DOLFIN_EPS

bc = DirichletBC(V,Constant('0.0'),q2b_boundary_left)

# Formulate the weak form a(u,v) = L(v)
u = TrialFunction(V)
v = TestFunction(V)
a = inner(grad(u),grad(v))*dx + u*v*dx + 2.0*u.dx(0)*v*dx
L = fa*v*dx

# Solve the problem
u = Function(V)
solve(a==L,u,bc)

# Make a plot
plot(u,interactive=True)


# -----------------------------------------------------------------------------
# Problem c)
# -----------------------------------------------------------------------------
# Mesh and Function Space is the same as before
```

```
# Define diffusion coefficient
q = Expression("1+2*x[0]")

# Define Dirichlet Boundary Conditions on the left endpoint
bc = DirichletBC(V,Constant('2.0'),q2b_boundary_left)

# Formulate the weak form
u = TrialFunction(V)
v = TestFunction(V)
a = inner(q*grad(u),grad(v))*dx
L = fa*v*dx + Constant('5.0')*q*v*ds

# Solve the problem
u = Function(V)
solve(a==L,u,bc)

# Make a plot
plot(u,interactive=True)
```

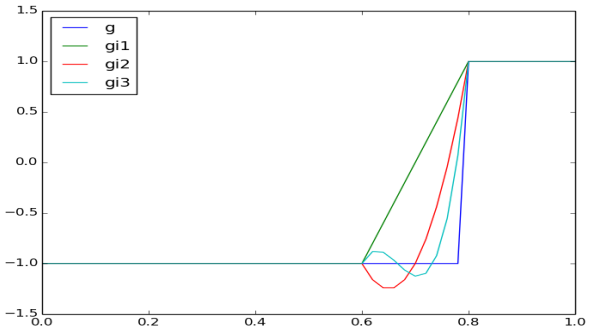
3. a) Dimensions of the various spaces:

Space	Dimension
V1	6
V2	11
V3	16

b) Plot of f and its interpolants:



Plot of g and its interpolants:



c) Output

```
V1: error(f) = 0.659224, error(g) = 1.800000
V2: error(f) = 0.095425, error(g) = 1.440000
V3: error(f) = 0.019384, error(g) = 1.071000
```

d) Output

Number of Cells	$\max \lvert f(x) - fi(x) \rvert$	$\max \lvert g(x) - gi(x) \rvert$
5	0.659224	1.8
10	0.172371	1.6
20	0.046843	1.2
40	0.011485	0.4

The error is

still pretty atrocious for interpolants of g. This error might be overly conservative, because there is only one cell in which the approximation is bad. The L^2 error might paint a rosier picture.

Here is a copy of the code

```
from dolfin import *
import matplotlib.pyplot as plt
import numpy

# ----------------------------------------------------------------------------
# Problem a)
# ----------------------------------------------------------------------------
# Define the mesh
mesh = UnitIntervalMesh(5)

# Define f and g
f = Expression("exp(-x[0]*x[0])*sin(4*pi*x[0])", cell = mesh.ufl_cell())
g = Expression("fabs(x[0]-pi/4)/(x[0]-pi/4)", cell = mesh.ufl_cell())

# Define the Function Spaces
V1 = FunctionSpace(mesh,"CG",1)
V2 = FunctionSpace(mesh,"CG",2)
V3 = FunctionSpace(mesh,"CG",3)

# Define the interpolants
# f
fi1 = interpolate(f,V1)
fi2 = interpolate(f,V2)
fi3 = interpolate(f,V3)
# g
gi1 = interpolate(g,V1)
gi2 = interpolate(g,V2)
gi3 = interpolate(g,V3)

# Print out the dimension of V1,V2, and V3
print "Dimension of V1 %d" % (V1.dim())
print "Dimension of V2 %d" % (V2.dim())
print "Dimension of V3 %d" % (V3.dim())


# ----------------------------------------------------------------------------
# Problem b)
# ----------------------------------------------------------------------------

def evaluate_fn(f,x):
```

```
  """ Evaluate a function at all points in a vector x
  Inputs:
  f - function
  x - numpy array

  Outputs:
  fv = f(xi) for all xi in x
  """
  n_pts = x.size
  fv = numpy.zeros([n_pts,1])
  for i in range(n_pts):
    fv[i] = f(x[i])

  return fv



# Plot f and its interpolants
x = numpy.linspace(0.0,1.0,51)
f_v   = evaluate_fn(f,x)
fi1_v = evaluate_fn(fi1,x)
fi2_v = evaluate_fn(fi2,x)
fi3_v = evaluate_fn(fi3,x)

plt.plot(x,f_v,x,fi1_v,x,fi2_v,x,fi3_v)
plt.legend( ('f','fi1','fi2','fi3'))


# Plot g and its interpolants
g_v   = evaluate_fn(g,x)
gi1_v = evaluate_fn(gi1,x)
gi2_v = evaluate_fn(gi2,x)
gi3_v = evaluate_fn(gi3,x)

plt.figure()
plt.plot(x,g_v,x,gi1_v,x,gi2_v,x,gi3_v)
plt.legend( ('g','gi1','gi2','gi3'), loc = 0)

# ------------------------------------------------------------------------
# Compute the maximum eror
# ------------------------------------------------------------------------
def compute_max_error(f_exact,f_approx,x):
  """ Compute the maximum absolute distance between f_exact and f_approx,
      based on the points given in x.
  """
  fe_v = evaluate_fn(f_exact,x)
```

```
  fa_v = evaluate_fn(f_approx,x)

  max_error = np.max(np.abs(fe_v - fa_v))
  return max_error

# f :
ef1 = compute_max_error(f,fi1,x)
ef2 = compute_max_error(f,fi2,x)
ef3 = compute_max_error(f,fi3,x)

# g :
eg1 = compute_max_error(g,gi1,x)
eg2 = compute_max_error(g,gi2,x)
eg3 = compute_max_error(g,gi3,x)

# Print
print "V1: error(f) = %f, error(g) = %f" % ( ef1,eg1 )
print "V2: error(f) = %f, error(g) = %f" % ( ef2,eg2 )
print "V3: error(f) = %f, error(g) = %f" % ( ef3,eg3 )



# ------------------------------------------------------------------------
# Problem d)
# ------------------------------------------------------------------------
cell_num_list = [5,10,20,40]

for n in cell_num_list:
  print "Number of cells = %d" %(n)
  mesh = UnitIntervalMesh(n)
  V    = FunctionSpace(mesh,"CG",1)

  fi   = interpolate(f,V)
  gi   = interpolate(g,V)

  error_fi = compute_max_error(f,fi,x)
  error_gi = compute_max_error(g,gi,x)

  print "|f - fi| = %f" % (error_fi)
  print "|g - gi| = %g" % (error_gi)
```