ISC 5935 - Computational Tools for Finite Elements

Homework #3 - Solutions

Q1 This is the code snippet I used to compute the errors:

```
#
# Compute the L2 and H10 errors
#
  l2_error = 0.0
  h1_error = 0.0

  for e in range(n):

    l = e
    r = e + 1

    xl = x[l]
    xr = x[r]

#
#  Consider quadrature point Q: (0, 1, 2 ) in element E.
#
    for q in range ( 0, ng ):
#
#  Map XG and WG from [0,1] to
#      XQ and QQ in [XL,XR].
#
      xq = xl + xg[q] * ( xr - xl )
      wq = wg[q] * ( xr - xl )
#
#  Evaluate at XQ the basis functions and derivatives for XL and XR.
#
      phil = ( xr - xq  ) / ( xr - xl )
      philp = - 1.0 / ( xr - xl )

      phir = ( xq - xl ) / ( xr - xl )
      phirp = 1.0 / ( xr - xl )

#
#   Evaluate at XQ the finite element approximation
#
      u_loc  = u[l]*phil + u[r]*phir
      up_loc = u[l]*philp + u[r]*phirp
```

```
#
#  Evaluate the exact function and its derivative at XQ
#
      uex_loc  = exact_fn(xq)
      uexp_loc = exact_fn_der(xq)

#
# Update the numerical integral
#
      l2_error = l2_error + wq * (u_loc - uex_loc)**2.0
      h1_error = h1_error + wq * (up_loc - uexp_loc)**2.0

#
# Take square roots
#
  l2_error = np.sqrt(l2_error)
  h1_error = np.sqrt(h1_error)

  print ""
  print "L2 error = %.6g" % (l2_error)
  print "H1 error = %.6g" % (h1_error)
```

The function `exact_fn_der` evaluates the derivative of the exact function and is given by this:

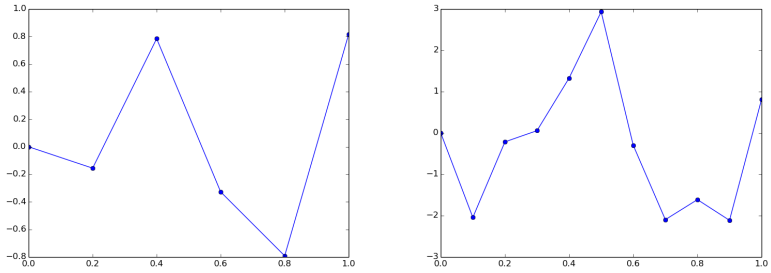
```
#
## EXACT_FN_DER evaluates the exact solution's derivative.
#
def exact_fn_der( x ):
    from math import exp
    value = ( 1 - x - x**2) * exp ( x )
    return value
```

Here are the errors I got.

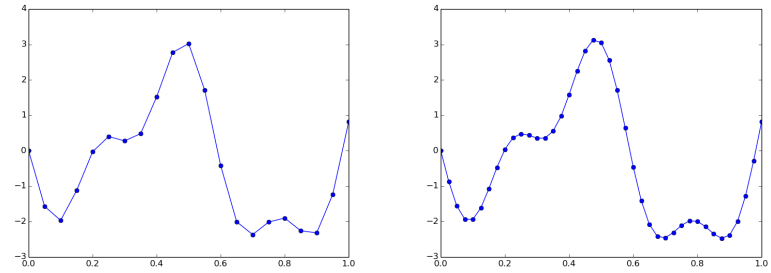
N	L^2 error	H^1 error
6	0.017362	0.275139
11	0.00438733	0.138818
21	0.00109978	0.0695662

Notice how, each time we halve the mesh size, the L^2 error reduces to a quarter of what it was. The H^1 error halves each time.

Q2 These are the figures of the solution. Successive plots look more and more similar, leading us to believe that our method is converging.



(a) $N = 6$ (b) $N = 11$



(c) $N = 21$ (d) $N = 41$

Figure 1: Finite element solutions computed with increasing spatial resolution.

Q3 This is the output

```
K : Matrix

 5  -5   0   0   0   0

-5  10  -5   0   0   0

 0  -5  10  -5   0   0

 0   0  -5  10  -5   0

 0   0   0  -5  10  -5

 0   0   0   0  -5   5

K: Eigenvalues
```

0.000 1.340 5.000 10.000 15.000 18.660

L : Matrix

-0.5 0.5 0.0 0.0 0.0 0.0

-0.5 -0.0 0.5 0.0 0.0 0.0

 0.0 -0.5 -0.0 0.5 0.0 0.0

 0.0 0.0 -0.5 -0.0 0.5 0.0

 0.0 0.0 0.0 -0.5 0.0 0.5

 0.0 0.0 0.0 0.0 -0.5 0.5

L: Eigenvalues

-0 + 0.87i, -0 + -0.87i, -0 + 0.50i, -0 + -0.50i, -0 + 0.00i, -0 + -0.00i,

M : Matrix

0.06667 0.03333 0.00000 0.00000 0.00000 0.00000

0.03333 0.13333 0.03333 0.00000 0.00000 0.00000

0.00000 0.03333 0.13333 0.03333 0.00000 0.00000

0.00000 0.00000 0.03333 0.13333 0.03333 0.00000

0.00000 0.00000 0.00000 0.03333 0.13333 0.03333

0.00000 0.00000 0.00000 0.00000 0.03333 0.06667

M: Eigenvalues

0.049 0.051 0.088 0.125 0.163 0.190

A : Matrix

4.567 -4.467 0.000 0.000 0.000 0.000

-5.467 10.133 -4.467 0.000 0.000 0.000

```
0.000 -5.467 10.133 -4.467 0.000 0.000

0.000 0.000 -5.467 10.133 -4.467 0.000

0.000 0.000 0.000 -5.467 10.133 -4.467

0.000 0.000 0.000 0.000 -5.467 5.567

A: Eigenvalues

18.688 + 0 i, 15.058 + 0 i, 10.100 + 0 i, 5.142 + 0 i, 0.164 + 0 i, 1.514 + 0 i,
```

Here is the code

```
#! /usr/bin/env python

# -*- coding: utf-8 -*-
"""
Created on Tue Sep 30 11:25:36 2014

@author: hans-werner
"""

def hw3q3():
    """ Investigate the semi-positive definiteness of system matrices """

    import numpy as np
    import scipy.linalg as la

    #
    # Define the mesh
    #
    a = 0.0
    b = 1.0
    N = 5
    x = np.linspace(a,b,N+1)

    #
    #  Set a 3 point quadrature rule on the reference interval [0,1].
    #
    ng = 3

    xg = np.array ( ( \
    0.11270166537925831148207346002 2, \
```

```
0.5, \
0.88729833462074168851792653997 8 ) )

wg = np.array ( ( \
5.0 / 18.0, \
8.0 / 18.0, \
5.0 / 18.0 ) )


#
# Compute the three system matrices
#
K = np.zeros([N+1,N+1])
L = np.zeros([N+1,N+1])
M = np.zeros([N+1,N+1])

#
# Loop over the elements
#
for e in range(N):

    xl = x[e]
    xr = x[e+1]

    #
    #  Consider quadrature point Q: (0, 1, 2 ) in element E.
    #
    for q in range ( 0, ng ):
        #
        #  Map XG and WG from [0,1] to
        #      XQ and QQ in [XL,XR].
        #
        xq = xl + xg[q] * ( xr - xl )
        wq = wg[q] * ( xr - xl )

        #
        #  Consider the I-th test function PHI(I,X) and
        #  its derivative PHI'(I,X).
        #
        for i_local in range ( 0, 2 ):
            i = i_local + e

            if ( i_local == 0 ):
                phii = ( xq - xr ) / ( xl - xr )
                phiip = 1.0 / ( xl - xr )
            else:
```

```
                phii = ( xq - xl ) / ( xr - xl )
                phiip = 1.0 / ( xr - xl )


            #
            #  Consider the J-th basis function PHI(J,X) and
            #  its derivative PHI'(J,X).
            #
            for j_local in range ( 0, 2 ):
                j = j_local + e

                if ( j_local == 0 ):
                    phij = ( xq - xr ) / ( xl - xr )
                    phijp = 1.0 / ( xl - xr )
                else:
                    phij = ( xq - xl ) / ( xr - xl )
                    phijp = 1.0 / ( xr - xl )

                #
                # Update the system matrices
                #
                K[i][j] = K[i][j] + wq*phiip*phijp
                L[i][j] = L[i][j] + wq*phii*phijp
                M[i][j] = M[i][j] + wq*phii*phij
#
# Combine the matrices to form the system matrix
A = K + L + M

#
# Compute the matrices' eigenvalues
#
lmd_K,_ = la.eigh(K)    # K is symmetric => use eigh
lmd_L,_ = la.eig(L)     # L is not symmetric => use eig
lmd_M,_ = la.eigh(M)    # M is symmetric => use eigh
lmd_A,_ = la.eig(A)     # A is not symmetric => use eig
#
# Print the Matrices and their eigenvalues
#
print " K : Matrix\n"
for i in range(N+1):
    for j in range(N+1):
        print "%3.0f" % (K[i][j]),
    print "\n"

print "K: Eigenvalues\n"
for i in range(N+1):
```

```
        print "%2.3f" % (lmd_K[i]),
    print "\n\n"

    print "L : Matrix \n"
    for i in range(N+1):
        for j in range(N+1):
            print "%4.1f" % (L[i][j]),
        print "\n"

    print "L: Eigenvalues\n"
    for i in range(N+1):
        print "%1.0f + %1.2fi, " % (lmd_L[i].real,lmd_L[i].imag),
    print "\n\n"

    print "M : Matrix\n"
    for i in range(N+1):
        for j in range(N+1):
            print "%4.5f" % (M[i][j]),
        print "\n"

    print "M: Eigenvalues\n"
    for i in range(N+1):
        print "%2.3f" % (lmd_M[i]),
    print "\n\n"

    print "A : Matrix\n"
    for i in range(N+1):
        for j in range(N+1):
            print "%5.3f" % (A[i][j]),
        print "\n"
    #
    # Note that since A is not symmetric, the eigenvalues might not be real
    #
    print "A: Eigenvalues\n"
    for i in range(N+1):
        print "%2.3f + %1.0f i," % (lmd_A[i].real,lmd_A[i].imag),
    print "\n\n"
#
#  If this script is called directly, then run it as a program.
#
if ( __name__ == '__main__' ):
  hw3q3( )
```