In [1]:
```
import sys
from scipy import linalg as LA
import math
import numpy as np
import blfunc as bl
from IPython.display import display

from sympy import *
from sympy import symbols
from sympy import init_printing
init_printing()

np.set_printoptions(precision= 4, suppress = True)
```

basis/shape functions along with their derivatives and integrals, used later in analyses

In [2]:
```
x = symbols('x') # the indepedent variable
phi1, phi1_p, phi1_pp = symbols('phi1 phi1_p phi1_pp') # first test function and its derivatives
phi2, phi2_p, phi2_pp = symbols('phi2 phi2_p phi2_pp') # second test function and its derivatives
phi3, phi3_p, phi3_pp = symbols('phi3 phi3_p phi3_pp') # third test function and its derivatives
```

In [3]:
```
phi1 = x*(1 - 0.5*x)
phi1_p = expand(diff(phi1, x, 1))
phi1_pp = expand(diff(phi1, x, 2))
display(phi1)
display(phi1_p)
display(phi1_pp)
```

$x(-0.5x+1)$

$-1.0x+1$

-1.0

In [4]:
```
phi2 = x*(1-x)**2
phi2_p = expand(diff(phi2, x, 1))
phi2_pp = expand(diff(phi2, x, 2))
display(phi2)
display(phi2_p)
display(phi2_pp)
```

$x(-x+1)^2$

$3x^2-4x+1$

$6x-4$

In [5]:
```
phi3 = x
phi3_p = expand(diff(phi3, x, 1))
phi3_pp = expand(diff(phi3, x, 2))
display(phi3)
display(phi3_p)
display(phi3_pp)
```

x

1

0

In [6]:
```python
def phi(j, x):
    if j == 0:
        out = x * (1-x/2.0)
    elif j == 1:
        out = x*(1-x)**2
    elif j == 2:
        out = x
    else:
        sys.exit("Bad Value, Test Function Not Defined")
    return out


def phi_prime(j, x):
    if j == 0:
        out = 1 - x
    elif j == 1:
        out = 1 - 4*x + 3*x**2
    elif j == 2:
        out = 1
    else:
        sys.exit("Bad Value, Test Function Not Defined")
    return out

def phi_int(j, x):
    if j == 0:
        out = -(1/6.0)*x**3 + (1/2.0)*x**2
    else:
        out = 0.25 * x ** 4 - (2/3.0)*x**3 + 0.5*x**2
    return out

def phi_2prime(j, x):
    if j == 0:
        out = -1
    else:
        out = -4 + 6*x
    return out

def f(x):
    return -1.8*math.pi*math.cos(1.8*math.pi*x) + math.sin(1.8*math.pi*x) * (2 + (1.8* math.pi)**2 * x)

def f_int(x):
    t1 = 1.8*math.pi*x**2*math.cos(1.8*math.pi)
    t2 = 1.8*math.pi*x*math.cos(1.8*math.pi)
    t3 = 1.8*math.pi*x*math.cos(1.8*math.pi*x)
    t4 = 10/(9.0 *math.pi) * math.cos(1.8*math.pi*x)
    return -t1 - t2 + t3 - t4
```

a.i) Collocation Method

Calculate stiffness matrix and forcing vector

In [7]:
```python
x = [1/3.0, 2/3.0]
term = 2

K = np.zeros((term,term))
f_vect = np.zeros((term,1))

# boundary conditions
u0 = 0
u1 = 1

for j in xrange(term):
    for i in xrange(term):

        # calc stiffness matrix
        t1 = -(phi_prime(i, x[j]) + x[j]*phi_2prime(i, x[j]))
        t2 = 2*phi(i, x[j])
        K[i, j] = t1 + t2

    # calc forcing vector
    L_u = -u1 + 2*u0 + 2 * u1 * x[j]
    f_vect[j] = f(x[j]) - L_u
```

In [8]:
```python
print K
```

```
[[ 0.2222  1.2222]
 [ 0.963   0.4815]]
```

In [9]:
```
print f_vect
```

```
[[ 14.1204]
 [ -9.4646]]
```

In [10]:
```
alpha_collocation = LA.solve(K, f_vect)
print alpha_collocation
```

```
[[-17.1657]
 [ 14.6741]]
```

a.ii) Subdomain Method

Calculate the integral of f star

In [11]:
```
x = symbols('x')
u1_sym = 1.8*pi*cos(1.8*pi) # accounts for BCTs
f_sym = -1.8*pi*cos(1.8*pi*x) + sin(1.8*pi*x) * (2 + (1.8* pi)**2 * x)
f_star = f_sym - (-u1_sym + 2*u1_sym*x)
f_star_int = f_star.integrate(x)
display(f_star_int)
```

$$-1.8\pi x^2\cos(1.8\pi) - 1.8\pi x\cos(1.8\pi x) + 1.8\pi x\cos(1.8\pi) - \frac{1.11111111111111}{\pi}\cos(1.8\pi x)$$

Calculate stiffness matrix and forcing vector

In [12]:
```
x = [0, 0.5, 1]
term = 2

K = np.zeros((term,term))
f_vect = np.zeros((term,1))

for j in xrange(term):
    for i in xrange(term):
        # calc stiffness matrix

        #lower domain limit
        l_t1 = -x[j] * phi_prime(i, x[j])
        l_t2 = phi_int(i, x[j])

        #upper domain limit
        u_t1 = -x[j+1] * phi_prime(i, x[j+1])
        u_t2 = phi_int(i, x[j+1])

        #calc difference between domain limits
        t1 = u_t1 - l_t1
        t2 = 2*(u_t2 - l_t2)

        K[i, j] = t1 + t2

    # calc forcing vector
    f_vect[j] =  f_int(x[j+1]) - f_int(x[j])
```

In [13]:
```
print K
```

```
[[-0.0417  0.7083]
 [ 0.2396 -0.0729]]
```

In [14]:
```
print f_vect
```

```
[[-5.4302]
 [ 0.9228]]
```

In [15]:
```
alpha_subdomain = LA.solve(K, f_vect)
print alpha_subdomain
```

```
[[ 1.5463]
 [-7.5752]]
```

a.iii) The Least Squares method

In [16]:
```
x = symbols('x')
K11 = float(integrate((-x+0.5*x**2+x+2*x*(1-0.5*x))**2,(x,0,1)))
display(K11)
```

0.883333333333

In [17]:
```
K12 = float(integrate((-x+0.5*x**2+x+2*x*(1-0.5*x))*(-1+4*x-3*x**2 - x*(6*x-4)+2*x*(1-x)**2),(x,0,1)))
display(K12)
```

0.0166666666667

In [18]:
```
K22 = float(integrate((-1+4*x-3*x**2 - x*(6*x-4)+2*x*(1-x)**2)**2,(x,0,1)))
display(K22)
```

0.704761904762

In [19]:
```
f1 = float(integrate((-x+0.5*x**2+x+2*x*(1-0.5*x)) * f_star, (x,0,1)))
display(f1)
```

−9.44811806672

In [20]:
```
f2 = float(integrate((-1+4*x-3*x**2 - x*(6*x-4)+2*x*(1-x)**2)* f_star, (x,0,1)))
display(f2)
```

11.2468543006

In [21]:
```
K = np.array([[K11,K12],[K12,K22]])
f_vect = np.array([[float(f1)], [float(f2)]])
print "Stiffness matrix:"
print K
print "forcing vector:"
print f_vect
```

```
Stiffness matrix:
[[ 0.8833  0.0167]
 [ 0.0167  0.7048]]
forcing vector:
[[ -9.4481]
 [ 11.2469]]
```

In [22]:
```
alpha_ls = LA.solve(K, f_vect)
display(alpha_ls)
```

```
array([[-11.002 ],
       [ 16.2186]])
```

C. Find the 1, 2, and 3 termm approximate solutions via the Galerkin method

1 term approximation

In [23]:
```
K11_galerkin = float(integrate(x * phi1_p * phi1_p + 2 * phi1 * phi1,(x, 0 , 1)))
display(K11_galerkin)
```

0.35

In [24]:
```
f = -1.8*pi*cos(1.8*pi*x) + sin(1.8*pi*x) * (2 + (1.8* pi)**2 * x)
f1_galerkin = float(integrate(phi1 * f,(x, 0, 1)))
display(f1_galerkin)
```

−2.78506181384

In [25]:
```
alpha_g1 = f1_galerkin / K11_galerkin
print alpha_g1
```

```
-7.95731946811
```

2 term approximation

In [26]:
```
K22_galerkin = float(integrate(x * phi2_p * phi2_p + 2 * phi2 * phi2,(x, 0, 1)))
display(K22_galerkin)
```

0.052380952381

In [27]:
```
K12_galerkin = float(integrate(x * phi2_p * phi1_p + 2 * phi2 * phi1,(x, 0, 1)))
display(K12_galerkin)
```

0.0333333333333

In [28]:
```
f2_galerkin = float(integrate(phi2 * f,(x, 0, 1)))
display(f2_galerkin)
```

0.381149580866

In [29]:
```
K2t_galerkin = np.array([[K11_galerkin, K12_galerkin],[K12_galerkin, K22_galerkin]])
f2t_galerkin = np.array([[f1_galerkin], [f2_galerkin]])
alpha_g2 = LA.solve(K2t_galerkin, f2t_galerkin)
print alpha_g2
```

```
[[ -9.2084]
 [ 13.1364]]
```

3 term approximation

In [30]:
```
K13_galerkin = float(integrate(x * phi3_p * phi1_p + 2 * phi3 * phi1,(x, 0, 1)))
display(K13_galerkin)
```

0.583333333333

In [31]:
```
K23_galerkin = float(integrate(x * phi3_p * phi2_p + 2 * phi3 * phi2,(x, 0, 1)))
display(K23_galerkin)
```

−0.0166666666667

In [32]:
```
K33_galerkin = float(integrate(x * phi3_p * phi3_p + 2 * phi3 * phi3,(x, 0, 1)))
display(K33_galerkin)
```

1.16666666667

In [33]:
```
f3_galerkin = float(integrate(phi3 * f,(x, 0, 1)))
display(f3_galerkin)
```

−5.51933542211

In [34]:
```
K3t_galerkin = np.array([[K11_galerkin, K12_galerkin, K13_galerkin],[K12_galerkin, K22_galerkin, K23_galerkin],
                         [K13_galerkin, K23_galerkin, K33_galerkin]])
f3t_galerkin = np.array([[f1_galerkin], [f2_galerkin], [f3_galerkin]])
alpha_g3 = LA.solve(K3t_galerkin, f3t_galerkin)
print alpha_g3
```

```
[[-10.6622]
 [ 14.3176]
 [  0.8048]]
```

D. Compare approximate solutions

Pointwise comparison of displacements

In [35]:
```
import matplotlib.pyplot as plt
```

In [36]:
```
# plotting domain
low_bound = 0
delta = 0.05
up_bound = 1 + delta

x = np.arange(low_bound, up_bound, delta)
```

Exact Solution

In [37]:
```
exact = np.zeros(1.05/0.05)
exact = np.sin(1.8*np.pi*x)
```

Collocation Function

In [38]:
```
def col_func(x, delta):
    if x == 0:
        out = 0
    elif x == 1:
        last = alpha_collocation[0]*phi(0,x - delta) + alpha_collocation[1]*phi(1,x - delta)
        out = last + 1.8*np.pi*np.cos(1.8*np.pi) * delta
    else:
        out = alpha_collocation[0]*phi(0,x) + alpha_collocation[1]*phi(1,x)
    return out
```

Subdomain Function

In [39]:
```
def sub_func(x, delta):
    if x == 0:
        out = 0
    elif x == 1:
        last = alpha_subdomain[0]*phi(0,x - delta) + alpha_subdomain[1]*phi(1,x - delta)
        out = last + 1.8*np.pi*np.cos(1.8*np.pi) * delta
    else:
        out = alpha_subdomain[0]*phi(0,x) + alpha_subdomain[1]*phi(1,x)
    return out
```

Least Squares Function

In [40]:
```
def ls_func(x, delta):
    if x == 0:
        out = 0
    elif x == 1:
        last = alpha_ls[0]*phi(0,x - delta) + alpha_ls[1]*phi(1,x - delta)
        out = last + 1.8*np.pi*np.cos(1.8*np.pi) * delta
    else:
        out = alpha_ls[0]*phi(0,x) + alpha_ls[1]*phi(1,x)
    return out
```

Galerkin Functions

In [50]:
```
def g1_func(x, delta):
    if x == 0:
        out = 0
    elif x == 1:
        last = -7.95731946811*phi(0,x - delta)
        out = last + 1.8*np.pi*np.cos(1.8*np.pi) * delta
    else:
        out = -7.95731946811*phi(0,x)
    return out

def g2_func(x, delta):
    if x == 0:
        out = 0
    elif x == 1:
        last = alpha_g2[0]*phi(0,x - delta) + alpha_g2[1]*phi(1,x - delta)
        out = last + 1.8*np.pi*np.cos(1.8*np.pi) * delta
    else:
        out = alpha_g2[0]*phi(0,x) + alpha_g2[1]*phi(1,x)
    return out

def g3_func(x, delta):
    if x == 0:
        out = 0
    elif x == 1:
        last = alpha_g3[0]*phi(0,x - delta) + alpha_g3[1]*phi(1,x - delta) + alpha_g3[1]*phi(2,x - delta)
        out = last + 1.8*np.pi*np.cos(1.8*np.pi) * delta
    else:
        out = alpha_g3[0]*phi(0,x) + alpha_g3[1]*phi(1,x) + alpha_g3[1]*phi(2,x - delta)
    return out
```

In [52]:
```
collocation = np.zeros(len(x))
subdomain = np.zeros(len(x))
LS = np.zeros(len(x))
g1 = np.zeros(len(x))
g2 = np.zeros(len(x))
g3 = np.zeros(len(x))

for i in xrange(len(x)):
    # for plotting
    collocation[i] = col_func(x[i], delta)
    subdomain[i] = sub_func(x[i], delta)
    LS[i] = ls_func(x[i], delta)
    g1[i] = g1_func(x[i], delta)
    g2[i] = g2_func(x[i], delta)
    g3[i] = g3_func(x[i], delta)
```

In [ ]:
```
fig1 = plt.figure()
plt.plot(x, exact, 'r--', label = 'Exact Solution')
plt.plot(x, collocation, 'bs', label = 'Collocation')
plt.plot(x, subdomain, 'g^', label = 'Subdomain')
plt.plot(x, LS, 'r^', label = 'Least Squares')
plt.plot(x, g1, 'rx', label = 'Galerkin (1 Term)')
plt.plot(x, g2, 'bx', label = 'Galerkin (2 Term)')
plt.plot(x, g3, 'gx', label = 'Galerkin (3 Term)')

plt.legend(loc = 2, fontsize = 10)
plt.xlabel('Location In Domain')
plt.ylabel('Displacement')
plt.show()
```

Evaluate norms

In [ ]:
```
x = symbols('x')
u = np.sin(1.8*np.pi*x)
phi_1 = x*(1-0.5*x)
phi_2 = x*(1-x)**2
phi_3 = x
```

In [ ]:
```
```