In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import math
```

3.b Plot of source function - f (x) = π^2[1 – sin(πx)]

In [2]:

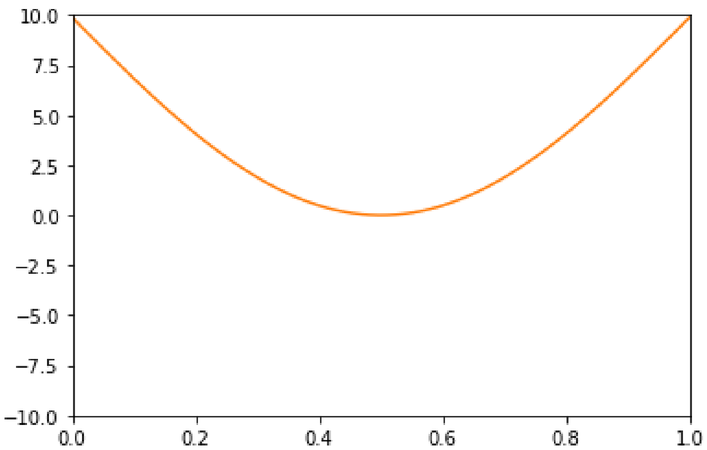
```
func = lambda x: np.square(np.pi) * (1. - np.sin(np.pi*x))
```

In [3]:

```
fig, ax = plt.subplots()
ax.axis([0,1,-10,10])
line, = ax.plot([])

xmin,xmax = ax.get_xlim()
npoints = fig.get_size_inches()[0]*fig.dpi
x = np.linspace([xmin], [xmax], int(npoints))
y = func(x)

plt.plot(x, y)
plt.show()
```



Plot of exact solution

In [6]:

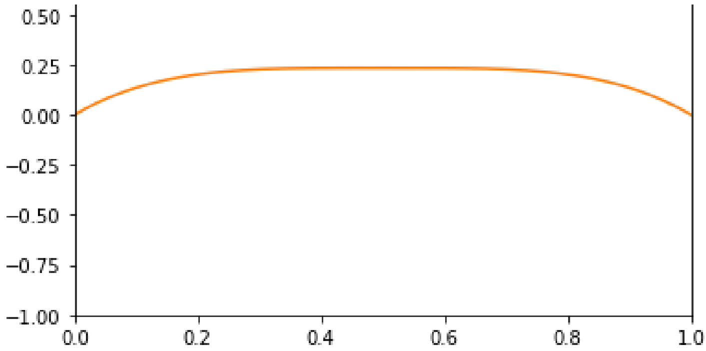
```
func = lambda x: (np.square(np.pi) * (x - np.square(x)) / 2) - np.sin(np.pi*x)
```

In [7]:

```
fig, ax = plt.subplots()
ax.axis([0,1,-1,1])
line, = ax.plot([])

xmin,xmax = ax.get_xlim()
npoints = fig.get_size_inches()[0]*fig.dpi
x = np.linspace([xmin], [xmax], int(npoints))
y = func(x)

plt.plot(x, y)
plt.show()
```



3.c Grid sizes

In [8]:

```
from scipy.sparse import diags
import numpy as np

def create_tri_diag(n):
    k = np.array([np.ones(n-1),-2*np.ones(n),np.ones(n-1)])
    offset = [-1,0,1]
    A = diags(k,offset).toarray()
    return A

def discretize_fx(h):
    func = lambda x: np.square(np.pi) * (1. - np.sin(np.pi*x))
    x = np.arange(0, 1, h)[1:]
    b = func(x)
    return b

def solve(A, b):
    return np.linalg.solve(A, b)

def discretize_true_solution(h):
    func = lambda x: (np.square(np.pi) * (x - np.square(x)) / 2) - np.sin(np.pi*x)
    x = np.arange(0, 1, h)[1:]
    discretized_true_sol = func(x)
    return discretized_true_sol
```

Grid size: 1/5

h = 1/5 => n = 4

A is a 4*4 matrix and t is 4-dimentional

In [31]:

```
h = 1/5
A = create_tri_diag(n=4) * np.square(5) * (-1)
b = discretize_fx(h=h)
approx = solve(A, b)
true = discretize_true_solution(h=h)
```

```
/home/ashutosh/Desktop/fau_venv/lib/python3.6/site-packages/ipykernel_launcher.py:5: Visi
bleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-
or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated.
If you meant to do this, you must specify 'dtype=object' when creating the ndarray
  """
```

Error = 0.118

In [32]:

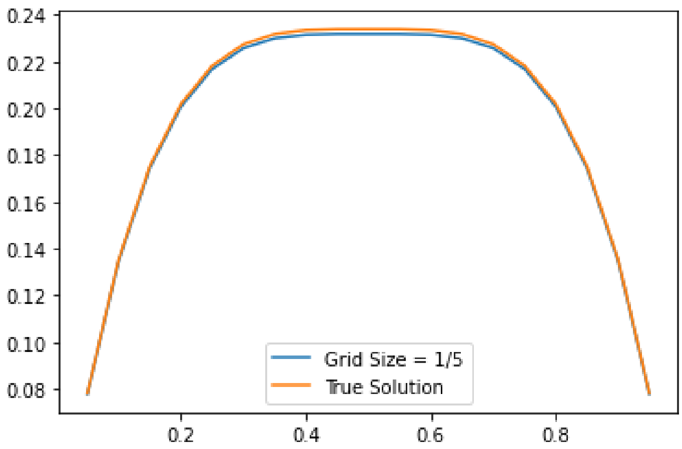
```
np.sum(np.abs(true - approx)) / np.sum(np.abs(true))
```

Out[32]:

0.11869348018555485

In [40]:

```
x = np.arange(0, 1, h)[1:]
plt.plot(x, approx, label='Grid Size = 1/5')
plt.plot(x, true, label='True Solution')
plt.legend()
plt.show()
```



Grid size: 1/10

h = 1/10 => n = 9

A is a 9*9 matrix and t is 9-dimentional

In [41]:

```
h = 1/10
A = create_tri_diag(n=9) * np.square(10) * (-1)
b = discretize_fx(h=h)
approx = solve(A, b)
true = discretize_true_solution(h=h)
```

```
/home/ashutosh/Desktop/fau_venv/lib/python3.6/site-packages/ipykernel_launcher.py:5: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
  """
```

Error = 0.02853

In [42]:

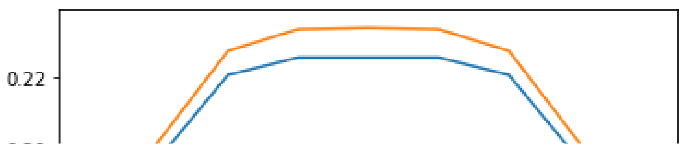
```
np.sum(np.abs(true - approx)) / np.sum(np.abs(true))
```

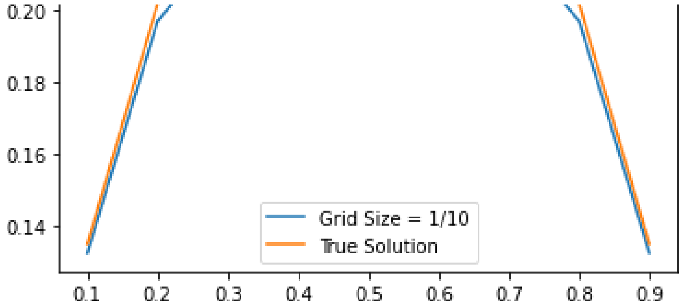
Out[42]:

0.028537531921092086

In [43]:

```
x = np.arange(0, 1, h)[1:]
plt.plot(x, approx, label='Grid Size = 1/10')
plt.plot(x, true, label='True Solution')
plt.legend()
plt.show()
```





Grid size: 1/20

h = 1/120 => n = 19

A is a 19*19 matrix and t is 19-dimentional

In [44]:

```
h = 1/20
A = create_tri_diag(n=19) * np.square(20) * (-1)
b = discretize_fx(h=h)
approx = solve(A, b)
true = discretize_true_solution(h=h)
```

```
/home/ashutosh/Desktop/fau_venv/lib/python3.6/site-packages/ipykernel_launcher.py:5: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
  """
```

Error = 0.0007

In [48]:

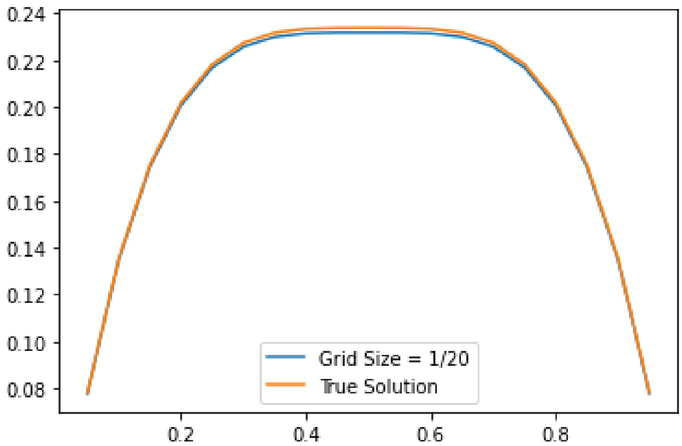
```
np.sum(np.abs(approx - true)) / np.sum(np.abs(true))
```

Out[48]:

```
0.007065980752937152
```

In [47]:

```
x = np.arange(0, 1, h)[1:]
plt.plot(x, approx, label='Grid Size = 1/20')
plt.plot(x, true, label='True Solution')
plt.legend()
plt.show()
```



3.d Estimatate of p for E = O(h^p)

In [51]:

In [51]:

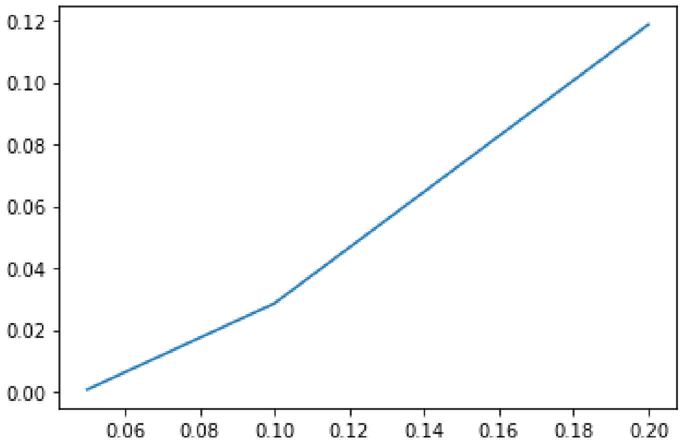
```
errors = [0.11869, 0.02853, 0.0007]
h_s1186 = [1/5, 1/10, 1/20]
```

In [52]:

```
plt.plot(h_s, errors)
```

Out[52]:

```
[<matplotlib.lines.Line2D at 0x7f20978b79b0>]
```



As we can see that relationship of h and error is almost linear

In [ ]: