In [1]:

```
import sympy as sp
from IPython.display import display
import numpy as np
import itertools

#input parameters
ndf = 1  # degrees of freedom
side = [2, 4, 8, 16]      #elements per side
numel =[4, 16, 64, 256]  #total number of elements in mesh
numnp = [(side[i]+1)**2 for i in range(len(numel))]   #total nod
es for each mesh
nen = 4   #nodes per element
numndpside = [3,5,9,17]   #nodes per side of mesh
a = 10   #lenght of one side
```

Define shape functions for a bilinear quadrilateral element

Construct local stiffness matrix

In [2]:

```
xi, eta = sp.symbols('xi eta')

def shape(x,y):
    N1 = sp.Rational(1,4)*(1-x)*(1-y)
    N2 = sp.Rational(1,4)*(1+x)*(1-y)
    N3 = sp.Rational(1,4)*(1+x)*(1+y)
    N4 = sp.Rational(1,4)*(1-x)*(1+y)
    return [N1, N2, N3, N4]
shape_fn = shape(xi,eta)
shape_mat = sp.zeros(len(shape_fn))

# Create elemental K_ij matrix in symbolic form:
for j in range(len(shape_fn)):
    for i in range(len(shape_fn)):
        shape_mat[j,i] = sp.integrate(sp.integrate((sp.diff(shap
e_fn[i],xi)*sp.diff(shape_fn[j],xi) +
                         sp.diff(shape_fn[i],eta)*sp.diff(shape_
fn[j],eta)),(xi,-1,1)),(eta,-1,1))

ID_2 = sp.zeros(ndf,(side[0]+1)**2)
# Convert symbolic form to numerical form
K_ij = np.reshape(shape_mat,(len(shape_fn),len(shape_fn)))

sp.pprint('[K_ij] :')
display(shape_mat)
display(K_ij)
print('')
```

```
[K_ij] :
```

$$\begin{bmatrix} \frac{2}{3} & -\frac{1}{6} & -\frac{1}{3} & -\frac{1}{6} \\ -\frac{1}{6} & \frac{2}{3} & -\frac{1}{6} & -\frac{1}{3} \\ -\frac{1}{3} & -\frac{1}{6} & \frac{2}{3} & -\frac{1}{6} \\ -\frac{1}{6} & -\frac{1}{3} & -\frac{1}{6} & \frac{2}{3} \end{bmatrix}\begin{bmatrix} \frac{2}{3} & -\frac{1}{6} & -\frac{1}{3} & -\frac{1}{6} \\ -\frac{1}{6} & \frac{2}{3} & -\frac{1}{6} & -\frac{1}{3} \\ -\frac{1}{3} & -\frac{1}{6} & \frac{2}{3} & -\frac{1}{6} \\ -\frac{1}{6} & -\frac{1}{3} & -\frac{1}{6} & \frac{2}{3} \end{bmatrix}$$

```
array([[2/3, -1/6, -1/3, -1/6],
       [-1/6, 2/3, -1/6, -1/3],
       [-1/3, -1/6, 2/3, -1/6],
       [-1/6, -1/3, -1/6, 2/3]], dtype=object)
```

In [3]:

```
#Create element matrix
e_4 = np.zeros((side[0]+1,side[0]+1))
e_16 = np.zeros((side[1]+1,side[1]+1))
e_64 = np.zeros((side[2]+1,side[2]+1))
e_256 = np.zeros((side[3]+1,side[3]+1))


counter_4 = 1
for i in range(side[0]+1):
    for j in range(side[0]+1):
        e_4[i,j] = counter_4
        counter_4 += 1


counter_16 = 1
for i in range(side[1]+1):
    for j in range(side[1]+1):
        e_16[i,j] = counter_16
        counter_16 += 1


counter_64 = 1
for i in range(side[2]+1):
    for j in range(side[2]+1):
        e_64[i,j] = counter_64
        counter_64 += 1


counter_256 = 1
for i in range(side[3]+1):
    for j in range(side[3]+1):
        e_256[i,j] = counter_256
        counter_256 += 1

e = [e_4,e_16,e_64,e_256]

for i in range(len(e)):
    print('element matrix = ', i+1)
    display(e[i])
    print('')
```

```
element matrix =  1

array([[1., 2., 3.],
       [4., 5., 6.],
       [7., 8., 9.]])

element matrix =  2

array([[ 1.,  2.,  3.,  4.,  5.],
       [ 6.,  7.,  8.,  9., 10.],
       [11., 12., 13., 14., 15.],
       [16., 17., 18., 19., 20.],
       [21., 22., 23., 24., 25.]])

element matrix =  3

array([[ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.]
,
       [10., 11., 12., 13., 14., 15., 16., 17., 18.]
,
       [19., 20., 21., 22., 23., 24., 25., 26., 27.]
,
       [28., 29., 30., 31., 32., 33., 34., 35., 36.]
,
       [37., 38., 39., 40., 41., 42., 43., 44., 45.]
,
       [46., 47., 48., 49., 50., 51., 52., 53., 54.]
,
       [55., 56., 57., 58., 59., 60., 61., 62., 63.]
,
       [64., 65., 66., 67., 68., 69., 70., 71., 72.]
,
       [73., 74., 75., 76., 77., 78., 79., 80., 81.]
])


element matrix =  4

array([[  1.,   2.,   3.,   4.,   5.,   6.,   7.,
8.,   9.,  10.,  11.,
         12.,  13.,  14.,  15.,  16.,  17.],
       [ 18.,  19.,  20.,  21.,  22.,  23.,  24.,  2
5.,  26.,  27.,  28.,
         29.,  30.,  31.,  32.,  33.,  34.],
       [ 35.,  36.,  37.,  38.,  39.,  40.,  41.,  4
2.,  43.,  44.,  45.,
         46.,  47.,  48.,  49.,  50.,  51.],
       [ 52.,  53.,  54.,  55.,  56.,  57.,  58.,  5
9.,  60.,  61.,  62.,
         63.,  64.,  65.,  66.,  67.,  68.],
       [ 69.,  70.,  71.,  72.,  73.,  74.,  75.,  7
6.,  77.,  78.,  79.,
         80.,  81.,  82.,  83.,  84.,  85.],
       [ 86.,  87.,  88.,  89.,  90.,  91.,  92.,  9
3.,  94.,  95.,  96.,
         97.,  98.,  99., 100., 101., 102.],
       [103., 104., 105., 106., 107., 108., 109., 11
0., 111., 112., 113.,
        114., 115., 116., 117., 118., 119.],
       [120., 121., 122., 123., 124., 125., 126., 12
7., 128., 129., 130.,
        131., 132., 133., 134., 135., 136.],
       [137., 138., 139., 140., 141., 142., 143., 14
4., 145., 146., 147.,
        148., 149., 150., 151., 152., 153.],
       [154., 155., 156., 157., 158., 159., 160., 16
1., 162., 163., 164.,
        165., 166., 167., 168., 169., 170.],
       [171., 172., 173., 174., 175., 176., 177., 17
8., 179., 180., 181.,
        182., 183., 184., 185., 186., 187.],
       [188., 189., 190., 191., 192., 193., 194., 19
5., 196., 197., 198.,
        199., 200., 201., 202., 203., 204.],
       [205., 206., 207., 208., 209., 210., 211., 21
2., 213., 214., 215.,
        216., 217., 218., 219., 220., 221.],
       [222., 223., 224., 225., 226., 227., 228., 22
9., 230., 231., 232.,
        233., 234., 235., 236., 237., 238.],
       [239., 240., 241., 242., 243., 244., 245., 24
6., 247., 248., 249.,
        250., 251., 252., 253., 254., 255.],
       [256., 257., 258., 259., 260., 261., 262., 26
3., 264., 265., 266.,
        267., 268., 269., 270., 271., 272.],
       [273., 274., 275., 276., 277., 278., 279., 28
0., 281., 282., 283.,
        284., 285., 286., 287., 288., 289.]])
```

In [4]:

```
"""Create the matrices IX, ID and LM"""

# ID matrices
ID_2 = sp.zeros(ndf,(side[0]+1)**2)
ID_4 = sp.zeros(ndf,(side[1]+1)**2)
ID_8 = sp.zeros(ndf,(side[2]+2)**2)
ID_16 = sp.zeros(ndf,(side[3]+2)**2)

ID_matrices = [ID_2, ID_4, ID_8, ID_16]

for n in range(len(ID_matrices)):
    for i in range(ndf):
        for j in range((side[n]+1)**2):
            ID_matrices[n][i,j] = j+1
```

In [5]:

```
# Create IX matrix
IX_2 = sp.zeros(nen,(side[0])**2)
IX_4 = sp.zeros(nen,(side[1])**2)
IX_8 = sp.zeros(nen,(side[2])**2)
IX_16 = sp.zeros(nen,(side[3])**2)
IX = [IX_2,IX_4,IX_8,IX_16]

m = 2
for i in range(len(IX)):
    c1 = 1
    for k in range(2):
        interim = np.reshape(e[i][0:numndpside[i]-1:1,k:numndpsi
de[i]-c1:1],(side[i]**2))
        for l in range(len(interim)):
            IX[i][k:l] = sp.Integer(interim[l])
        c1 -= 1
    c1 = 1
    for j in range(2):
        interim = np.reshape(e[i][1:numndpside[i]:1,c1:numndpsid
e[i]-j:1],(side[i]**2))
        for l in range(len(interim)):
            IX[i][j+2:l] = sp.Integer(interim[l])
        c1 -= 1

    print('IX_{}  = '.format(m))
    display(IX[i])
    print('')
    m = 2*m
```

```
IX_2  =
```

$$\begin{bmatrix} 1 & 2 & 4 & 5 \\ 2 & 3 & 5 & 6 \\ 5 & 6 & 8 & 9 \\ 4 & 5 & 7 & 8 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 4 & 5 \\ 2 & 3 & 5 & 6 \\ 5 & 6 & 8 & 9 \\ 4 & 5 & 7 & 8 \end{bmatrix}$$

```
IX_4  =
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 6 & 7 & 8 & 9 & 11 & 12 & 13 & 14 & 16 \\ 2 & 3 & 4 & 5 & 7 & 8 & 9 & 10 & 12 & 13 & 14 & 15 & 17 \\ 7 & 8 & 9 & 10 & 12 & 13 & 14 & 15 & 17 & 18 & 19 & 20 & 22 \\ 6 & 7 & 8 & 9 & 11 & 12 & 13 & 14 & 16 & 17 & 18 & 19 & 21 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 6 & 7 & 8 & 9 & 11 & 12 & 13 & 14 & 16 & 17 & 18 & 19 \\ 2 & 3 & 4 & 5 & 7 & 8 & 9 & 10 & 12 & 13 & 14 & 15 & 17 & 18 & 19 & 20 \\ 7 & 8 & 9 & 10 & 12 & 13 & 14 & 15 & 17 & 18 & 19 & 20 & 22 & 23 & 24 & 25 \\ 6 & 7 & 8 & 9 & 11 & 12 & 13 & 14 & 16 & 17 & 18 & 19 & 21 & 22 & 23 & 24 \end{bmatrix}$$

```
IX_8  =
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 10 & 11 & 12 & 13 & 1 \\ 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 11 & 12 & 13 & 14 & 1 \\ 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 20 & 21 & 22 & 23 & 2 \\ 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 19 & 20 & 21 & 22 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 \\ 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 \\ 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 \\ 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 \end{bmatrix}$$

```
IX_16  =
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 1 \\ 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 1 \\ 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 & 3 \\ 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 \\ 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 & 32 & 33 & 34 \\ 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 & 32 & 33 \end{bmatrix}$$

Since there is only one degree of freedom

$$[IX] = [LM]$$

$$[IX] = [LM]$$

In [6]:

```
# From the local stiffness matrix and [LM], compute the global s
tiffness matrix
K_IJ = []
for i in range(len(numnp)):
    K_IJ.append(np.zeros((ndf*numnp[i],ndf*numnp[i])))
# print('len(K_IJ) = ',len(K_IJ))
# print('len(IX) = ',len(IX))

for z in range(len(K_IJ)):
    for k in range(numel[z]):
        for j in range(ndf*nen):
            for i in range(ndf*nen):
                K_IJ[z][IX[z][i,k]-1,IX[z][j,k]-1] = K_IJ[z][IX[
z][i,k]-1,IX[z][j,k]-1] + shape_mat[i,j]
```

In [7]:

```
# find elements not on the boundary
free_element = []
for i in range(len(e)):
    fe = []
    interim_matrix = ((np.reshape(e[i][1:-1,1:-1],(side[i]-1,sid
e[i]-1)).astype(int)))
    for j in range(numndpside[i]-2):
        for k in range(numndpside[i]-2):
            fe.append(interim_matrix[j,k])
    free_element.append(fe)


# find element on the boundary
bc_element = []
for z in range(nen):
    bc_elem = []
    for i in range(numndpside[z]):
        for j in range(numndpside[z]):
            if (i==0):
                bc_elem.append((e[z][i,j]).astype(int))
            elif (i == numndpside[z]-1):
                bc_elem.append((e[z][i,j]).astype(int))
            elif (j==0):
                bc_elem.append((e[z][i,j]).astype(int))
            elif (j==numndpside[z]-1):
                bc_elem.append((e[z][i,j]).astype(int))
    bc_element.append(bc_elem)

for i in range(len(free_element)):
    print(free_element[i])
```

```
[5]
[7, 8, 9, 12, 13, 14, 17, 18, 19]
[11, 12, 13, 14, 15, 16, 17, 20, 21, 22, 23, 24, 25,
26, 29, 30, 31, 32, 33, 34, 35, 38, 39, 40, 41, 42,
43, 44, 47, 48, 49, 50, 51, 52, 53, 56, 57, 58, 59,
60, 61, 62, 65, 66, 67, 68, 69, 70, 71]
[19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
32, 33, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46,
47, 48, 49, 50, 53, 54, 55, 56, 57, 58, 59, 60, 61,
62, 63, 64, 65, 66, 67, 70, 71, 72, 73, 74, 75, 76,
77, 78, 79, 80, 81, 82, 83, 84, 87, 88, 89, 90, 91,
92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 104, 105,
106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 11
6, 117, 118, 121, 122, 123, 124, 125, 126, 127, 128,
129, 130, 131, 132, 133, 134, 135, 138, 139, 140, 14
1, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151,
152, 155, 156, 157, 158, 159, 160, 161, 162, 163, 16
4, 165, 166, 167, 168, 169, 172, 173, 174, 175, 176,
177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 18
9, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199,
200, 201, 202, 203, 206, 207, 208, 209, 210, 211, 21
2, 213, 214, 215, 216, 217, 218, 219, 220, 223, 224,
225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 23
5, 236, 237, 240, 241, 242, 243, 244, 245, 246, 247,
248, 249, 250, 251, 252, 253, 254, 257, 258, 259, 26
0, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270,
271]
```

In [8]:

```
# set the Dirichlet boundary conditions:
u_x = []
u_bc = []
for i in range(len(numel)):
    ubc = np.zeros((len(bc_element[i]),1))     # u at the bounda
ry conditions
#     interim = np.zeros((len(bc_element[i]),len(bc_element[i]))
)
    section = a/side[i]
    output = np.zeros(numndpside[i])
    for j in range(1,numndpside[i]):
        output[j] = output[j-1]+section
    u_x.append(output)
    for k in range(numndpside[i]):
        ubc[k] = output[k]*(10-output[k])
    u_bc.append(ubc)
print(u_x[1])
print('')
print(u_bc[1])
```

```
[ 0.   2.5  5.   7.5 10. ]

[[ 0.  ]
 [18.75]
 [25.  ]
 [18.75]
 [ 0.  ]
 [ 0.  ]
 [ 0.  ]
 [ 0.  ]
 [ 0.  ]
 [ 0.  ]
 [ 0.  ]
 [ 0.  ]
 [ 0.  ]
 [ 0.  ]
 [ 0.  ]
 [ 0.  ]]
```

In [9]:

```
# Shape K
# determine locations related to unknown u's within K_IJ
K_x = []

for z in range(nen):
    K_interim = np.zeros((len(free_element[z]),len(free_element[
z])))
    for i in range(len(free_element[z])):
        for j in range(len(free_element[z])):
            K_interim[i,j] = K_IJ[z][free_element[z][i]-1,free_e
lement[z][j]-1]
    K_x.append(K_interim)



# determine locations related to the bc's u's within K_IJ
K_bc = []

for z in range(nen):
    K_interim = np.zeros((len(free_element[z]),len(bc_element[z]
)))
    for i in range(len(free_element[z])):
        for j in range(len(bc_element[z])):
            K_interim[i,j] = -1*K_IJ[z][free_element[z][i]-1,bc_
element[z][j]-1]
    K_bc.append(K_interim)
```

In [10]:

```
# Compute the solutions of u not on the boundary
u = []
for z in range(nen):
    output = np.linalg.solve(K_x[z],np.dot(K_bc[z],u_bc[z]))
    u.append(output)
```

In [11]:

```
# Reshape u for graphing
u_solution = []
for z in range(nen):
    interim = np.zeros((numnp[z],1))
    for i in range(len(u[z])):
        interim[free_element[z][i]-1] = u[z][i]
    for j in range(len(u_bc[z])):
        interim[bc_element[z][j]-1] = u_bc[z][j]
    u_solution.append(interim)
    print(interim)
    print('')
```

```
[[ 0.   ]
 [25.   ]
 [ 0.   ]
 [ 0.   ]
 [ 3.125]
 [ 0.   ]
 [ 0.   ]
 [ 0.   ]
 [ 0.   ]]

[[ 0.        ]
 [18.75      ]
 [25.        ]
 [18.75      ]
 [ 0.        ]
 [ 0.        ]
 [ 7.88018433]
 [11.21111751]
 [ 7.88018433]
 [ 0.        ]
 [ 0.        ]
 [ 3.34821429]
 [ 4.73214286]
 [ 3.34821429]
 [ 0.        ]
 [ 0.        ]
 [ 1.22695853]
 [ 1.73531106]
 [ 1.22695853]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]]

[[ 0.        ]
 [10.9375    ]
 [18.75      ]
 [23.4375    ]
 [25.        ]
 [23.4375    ]
 [18.75      ]
 [10.9375    ]
 [ 0.        ]
 [ 0.        ]
 [ 6.84958413]
 [12.40234694]
 [15.87827712]
 [17.05611975]
 [15.87827712]
 [12.40234694]
 [ 6.84958413]
 [ 0.        ]
 [ 0.        ]
 [ 4.48489146]
 [ 8.22193465]
 [10.65908821]
 [11.49922738]
 [10.65908821]
 [ 8.22193465]
 [ 4.48489146]
 [ 0.        ]
 [ 0.        ]
 [ 2.95741137]
 [ 5.44785456]
 [ 7.0960234 ]
 [ 7.67092184]
 [ 7.0960234 ]
 [ 5.44785456]
 [ 2.95741137]
 [ 0.        ]
 [ 0.        ]
 [ 1.93449669]
 [ 3.57011361]
 [ 4.65887712]
 [ 5.04016986]
 [ 4.65887712]
 [ 3.57011361]
 [ 1.93449669]
 [ 0.        ]
 [ 0.        ]
 [ 1.2296445 ]
 [ 2.27094947]
 [ 2.96565179]
 [ 3.2093324 ]
 [ 2.96565179]
 [ 2.27094947]
 [ 1.2296445 ]
 [ 0.        ]
 [ 0.        ]
 [ 0.72403997]
 [ 1.33755623]
 [ 1.74721582]
 [ 1.89099985]
 [ 1.74721582]
 [ 1.33755623]
 [ 0.72403997]
 [ 0.        ]
 [ 0.        ]
 [ 0.33508509]
 [ 0.61908448]
 [ 0.80877875]
 [ 0.87537362]
 [ 0.80877875]
 [ 0.61908448]
 [ 0.33508509]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]]

[[ 0.        ]
 [ 5.859375  ]
 [10.9375    ]
 [15.234375  ]
 [18.75      ]
 [21.484375  ]
 [23.4375    ]
 [24.609375  ]
 [25.        ]
 [24.609375  ]
 [23.4375    ]
 [21.484375  ]
 [18.75      ]
 [15.234375  ]
 [10.9375    ]
 [ 5.859375  ]
 [ 0.        ]
 [ 0.        ]
 [ 4.49323849]
 [ 8.65228863]
 [12.27090314]
 [15.27948651]
 [17.64310451]
 [19.34221457]
 [20.36558506]
 [20.70732268]
 [20.36558506]
 [19.34221457]
 [17.64310451]
 [15.27948651]
 [12.27090314]
 [ 8.65228863]
 [ 4.49323849]
 [ 0.        ]
 [ 0.        ]
 [ 3.56751692]
 [ 6.9292274 ]
 [ 9.92617305]
 [12.45817456]
 [14.46878684]
 [15.92429857]
 [16.80469159]
 [17.09927811]
 [16.80469159]
 [15.92429857]
 [14.46878684]
 [12.45817456]
 [ 9.92617305]
 [ 6.9292274 ]
 [ 3.56751692]
 [ 0.        ]
 [ 0.        ]
 [ 2.86931172]
 [ 5.59606909]
 [ 8.05831817]
 [10.1649169 ]
 [11.85370734]
 [13.08439173]
 [13.83190694]
 [14.08253502]
 [13.83190694]
 [13.08439173]
 [11.85370734]
 [10.1649169 ]
 [ 8.05831817]
 [ 5.59606909]
 [ 2.86931172]
 [ 0.        ]
 [ 0.        ]
 [ 2.32235312]
 [ 4.53932727]
 [ 6.55632503]
 [ 8.2963321 ]
 [ 9.70151808]
 [10.73123999]
 [11.35898449]
 [11.56983604]
 [11.35898449]
 [10.73123999]
 [ 9.70151808]
 [ 8.2963321 ]
 [ 6.55632503]
 [ 4.53932727]
 [ 2.32235312]
 [ 0.        ]
 [ 0.        ]
 [ 1.88496327]
 [ 3.6891536 ]
 [ 5.33812416]
 [ 6.76835894]
 [ 7.92938816]
 [ 8.78380947]
 [ 9.30621369]
 [ 9.48194303]
 [ 9.30621369]
 [ 8.78380947]
 [ 7.92938816]
 [ 6.76835894]
 [ 5.33812416]
 [ 3.6891536 ]
 [ 1.88496327]
 [ 0.        ]
 [ 0.        ]
 [ 1.53069683]
 [ 2.99817535]
 [ 4.34326379]
 [ 5.51405723]
 [ 6.46786298]
 [ 7.1719265 ]
 [ 7.60334184]
 [ 7.74862819]
 [ 7.60334184]
 [ 7.1719265 ]
 [ 6.46786298]
 [ 5.51405723]
 [ 4.34326379]
 [ 2.99817535]
 [ 1.53069683]
 [ 0.        ]
 [ 0.        ]
 [ 1.2411078 ]
 [ 2.43217461]
 [ 3.52591872]
 [ 4.48014767]
 [ 5.25939345]
 [ 5.83582242]
 [ 6.18957996]
 [ 6.30881148]
 [ 6.18957996]
 [ 5.83582242]
 [ 5.25939345]
 [ 4.48014767]
 [ 3.52591872]
 [ 2.43217461]
 [ 1.2411078 ]
 [ 0.        ]
 [ 0.        ]
 [ 1.00253481]
 [ 1.96528077]
 [ 2.85041884]
 [ 3.62383152]
 [ 4.25643481]
 [ 4.72506451]
 [ 5.01297533]
 [ 5.11006939]
 [ 5.01297533]
 [ 4.72506451]
 [ 4.25643481]
 [ 3.62383152]
 [ 2.85041884]
 [ 1.96528077]
 [ 1.00253481]
 [ 0.        ]
 [ 0.        ]
 [ 0.80443717]
 [ 1.57727814]
 [ 2.28837606]
 [ 2.91034323]
 [ 3.41961936]
 [ 3.79725628]
 [ 4.02943444]
 [ 4.10776418]
 [ 4.02943444]
 [ 3.79725628]
 [ 3.41961936]
 [ 2.91034323]
 [ 2.28837606]
 [ 1.57727814]
 [ 0.80443717]
 [ 0.        ]
 [ 0.        ]
 [ 0.63843604]
 [ 1.25196763]
 [ 1.81677378]
 [ 2.3111146 ]
 [ 2.71617687]
 [ 3.01673309]
 [ 3.20161186]
 [ 3.26400083]
 [ 3.20161186]
 [ 3.01673309]
 [ 2.71617687]
 [ 2.3111146 ]
 [ 1.81677378]
 [ 1.25196763]
 [ 0.63843604]
 [ 0.        ]
 [ 0.        ]
 [ 0.49771089]
 [ 0.97609449]
 [ 1.41663446]
 [ 1.80238163]
 [ 2.1186114 ]
 [ 2.35335539]
 [ 2.49779913]
 [ 2.54655158]
 [ 2.49779913]
 [ 2.35335539]
 [ 2.1186114 ]
 [ 1.80238163]
 [ 1.41663446]
 [ 0.97609449]
 [ 0.49771089]
 [ 0.        ]
 [ 0.        ]
 [ 0.37658985]
 [ 0.73859908]
 [ 1.07204417]
 [ 1.36410027]
 [ 1.60359749]
 [ 1.78143185]
 [ 1.89088141]
 [ 1.927827  ]
 [ 1.89088141]
 [ 1.78143185]
 [ 1.60359749]
 [ 1.36410027]
 [ 1.07204417]
 [ 0.73859908]
 [ 0.37658985]
 [ 0.        ]
 [ 0.        ]
 [ 0.27025266]
 [ 0.5300617 ]
 [ 0.76940445]
 [ 0.97907731]
 [ 1.15105128]
 [ 1.27877084]
 [ 1.35738784]
 [ 1.38392766]
 [ 1.35738784]
 [ 1.27877084]
 [ 1.15105128]
 [ 0.97907731]
 [ 0.76940445]
 [ 0.5300617 ]
 [ 0.27025266]
 [ 0.        ]
 [ 0.        ]
 [ 0.17450206]
 [ 0.34226855]
 [ 0.49683273]
 [ 0.6322518 ]
 [ 0.74333625]
 [ 0.8258444 ]
 [ 0.87663616]
 [ 0.89378343]
 [ 0.87663616]
 [ 0.8258444 ]
 [ 0.74333625]
 [ 0.6322518 ]
 [ 0.49683273]
 [ 0.34226855]
 [ 0.17450206]
 [ 0.        ]
 [ 0.        ]
 [ 0.08557825]
 [ 0.16785535]
 [ 0.24366124]
 [ 0.31008146]
 [ 0.36456965]
 [ 0.4050433 ]
 [ 0.4299599 ]
 [ 0.43837195]
 [ 0.4299599 ]
 [ 0.4050433 ]
 [ 0.36456965]
 [ 0.31008146]
 [ 0.24366124]
 [ 0.16785535]
 [ 0.08557825]
 [ 0.        ]
 [ 0.          ]
 [ 0.          ]
 [ 0.          ]
 [ 0.          ]
 [ 0.          ]
 [ 0.          ]
 [ 0.          ]
 [ 0.          ]
 [ 0.          ]
 [ 0.          ]
 [ 0.          ]
 [ 0.          ]
 [ 0.          ]
 [ 0.          ]
 [ 0.          ]
 [ 0.          ]
 [ 0.          ]]
```

In [12]:

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

# Convert to x,y and z coords

Z = []
for i in range(nen):
    Z.append(np.reshape(u_solution[i],(numndpside[i],numndpside[
i])))

x_plot = []

for z in range(nen):
    output = []
    for i in range(len(u_x[z])):
        interim = u_x[z][i]
        output.append(interim)
    x_plot.append(output)
    y_plot = x_plot

    (X,Y) = np.meshgrid(x_plot[z],y_plot[z])
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    surf = ax.plot_surface(X,Y,Z[z],rstride=1,cstride=1,cmap='ma
gma',lw=0)
    fig.colorbar(surf)

    plt.savefig('node_{}.tif'.format(2**(z+1)))
    plt.show()
    plt.clf()

#     im = plt.imshow(Z[z], cmap='hot')
#     plt.colorbar(im, orientation='horizontal')
#     plt.show()
```

```
<Figure size 640x480 with 2 Axes>

<Figure size 640x480 with 0 Axes>

<Figure size 640x480 with 2 Axes>

<Figure size 640x480 with 0 Axes>

<Figure size 640x480 with 2 Axes>

<Figure size 640x480 with 0 Axes>

<Figure size 640x480 with 2 Axes>
```

In [13]:

```
# compute the midpoints to the approximate solution
uh_midpoint = []
for z in range(nen):
    mid = np.floor(numndpside[z]/2).astype(int)
    output = Z[z][mid,mid]
    uh_midpoint.append(output)
```

In [14]:

```
# compute exact solution
def u_exact(x,y):
    z = sp.symbols('z')
    output = 0
    for i in range(1,10):
        output1 = 0.2/np.sinh(i*np.pi)
        output2 = sp.integrate(z*(10-z)*sp.sin(i*np.pi*z/10), (z
, 0, 10))
        output3 = np.sin(i*np.pi*x/10)
        output4 = np.sinh(i*np.pi*(10-y)/10)
        output += output1*output2*output3*output4
    return output
u_midpoint_exact = (u_exact(5,5))
print(u_exact(5,5))
```

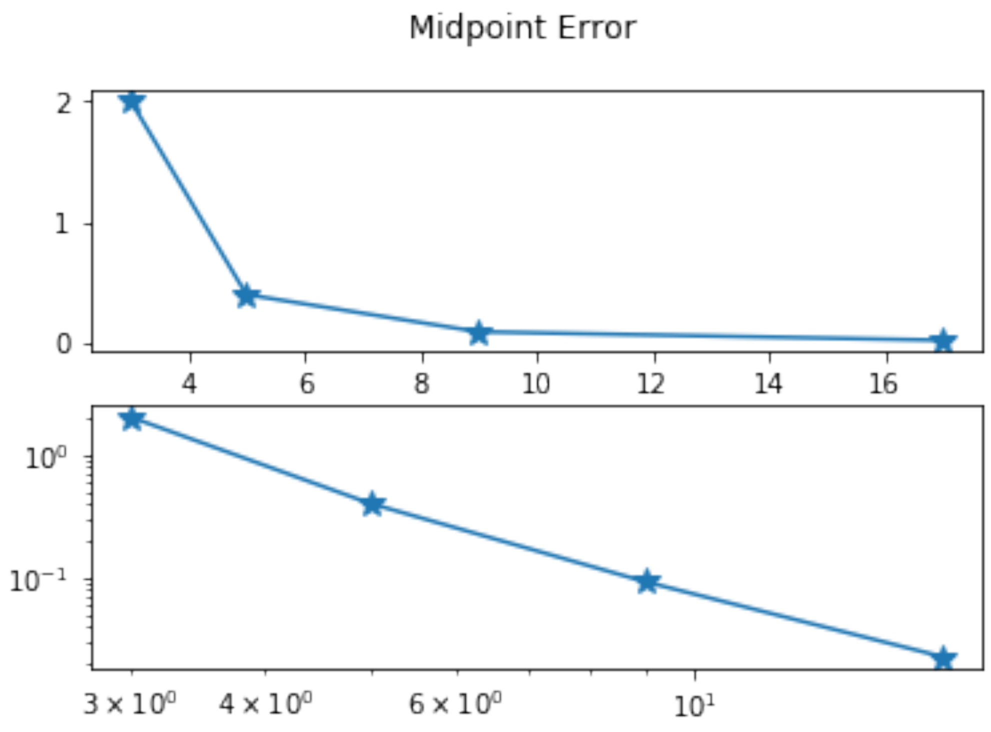
```
5.13286467244044
```

In [15]:

```
#compute the mid point error

midpoint_error = []
for z in range(nen):
    midpoint_error.append(np.absolute(uh_midpoint[z]-u_midpoint_
exact))

#plot
fig, axs = plt.subplots(2)
fig.suptitle('Midpoint Error')
axs[0].plot(numndpside[:], midpoint_error[:],markersize=10,marke
r='*')
axs[1].plot(numndpside[:], midpoint_error[:],markersize=10,marke
r='*')
axs[1].set_yscale('log')
axs[1].set_xscale('log')
plt.show()
```



In [ ]: