SFEM_code_2

October 2, 2020

1 Simplification for FEM – Code tutorial

- Coding with Python in Jupyter notebook
- Basic elements in FEM code
- Convergence analysis for a concrete physical case

1.1 Python in Jupyter

- Brief introduction about Jupyter notebook
- Nesessary packages (Numpy, matplotlib)

[130]:
```
import numpy as np
from numpy.polynomial.legendre import leggauss # Gauss quadrature
import matplotlib.pyplot as plt
from scipy import integrate

%matplotlib inline
```

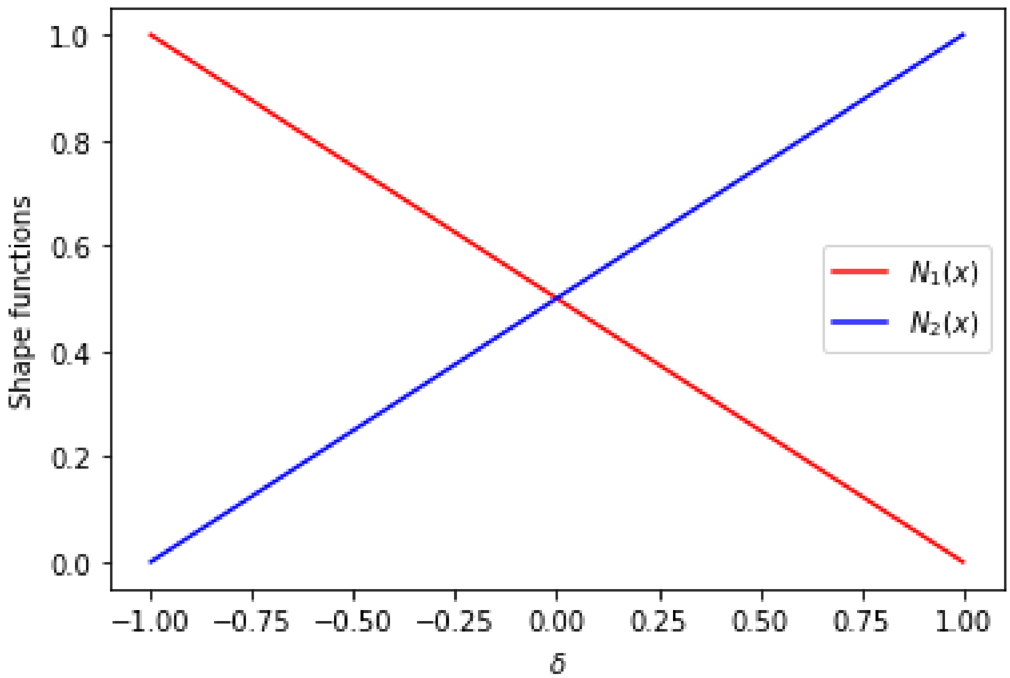
1.2 Basic elements in FEM code

- Numerical quadrature (Gauss Legendre)
- Shape functions linear, hieracchic)
- Elementary matrix and assembly
- Solve the linear system

[131]:
```
def gauss_legendre_quad(f, n, a, b):
    x, w = leggauss(n)
    sum_ = 0
    for k in range(len(x)):
        sum_ += w[k] * f(0.5*(b-a)*x[k]+0.5*(b+a))
    return 0.5 * (b-a) * sum_
```

1.2.1 Linear shape function

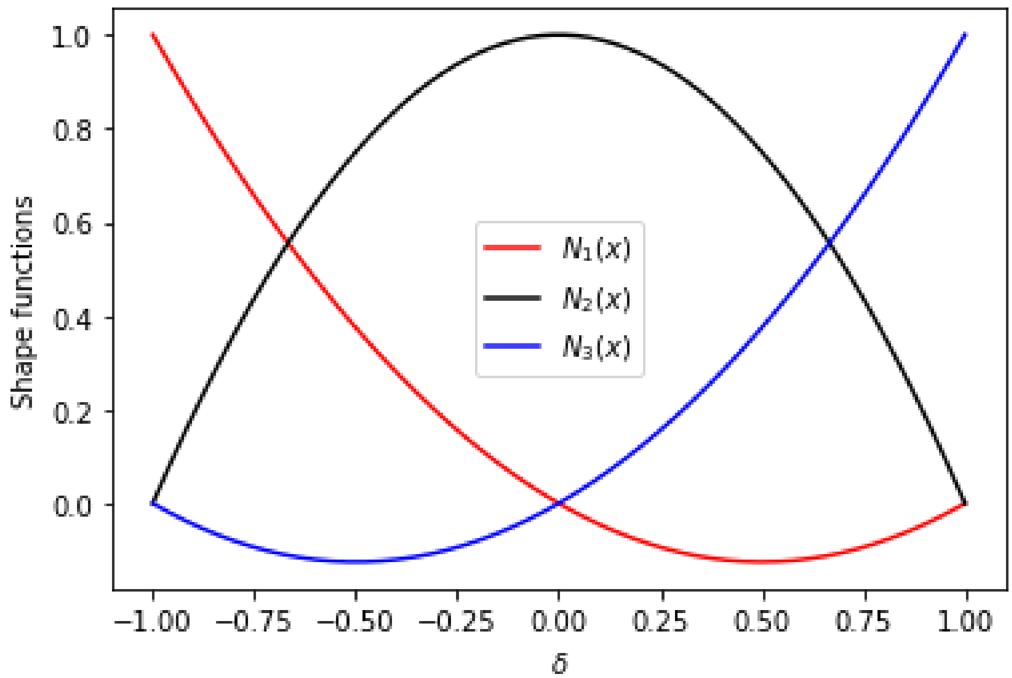
[132]:
```
N1 = lambda x: -x/2+1/2
N2 = lambda x: x/2+1/2
dN1 = lambda x: -1/2  # B1
dN2 = lambda x: 1/2  # B2
x = np.linspace(-1,1,200)
plt.plot(x,N1(x),'r',label='$N_1(x)$')
plt.plot(x,N2(x),'b',label='$N_2(x)$')
plt.xlabel('$\delta$');plt.ylabel('Shape functions')
plt.legend()
plt.show()
```



1.2.2 Quadratic shape function

[133]:
```
NN1 = lambda x: x**2/2 - x/2
NN2 = lambda x: -x**2+1
NN3 = lambda x: x**2/2 + x/2
x = np.linspace(-1,1,200)
plt.plot(x,NN1(x),'r',label='$N_1(x)$')
plt.plot(x,NN2(x),'k',label='$N_2(x)$')
plt.plot(x,NN3(x),'b',label='$N_3(x)$')
plt.xlabel('$\delta$');plt.ylabel('Shape functions')
```

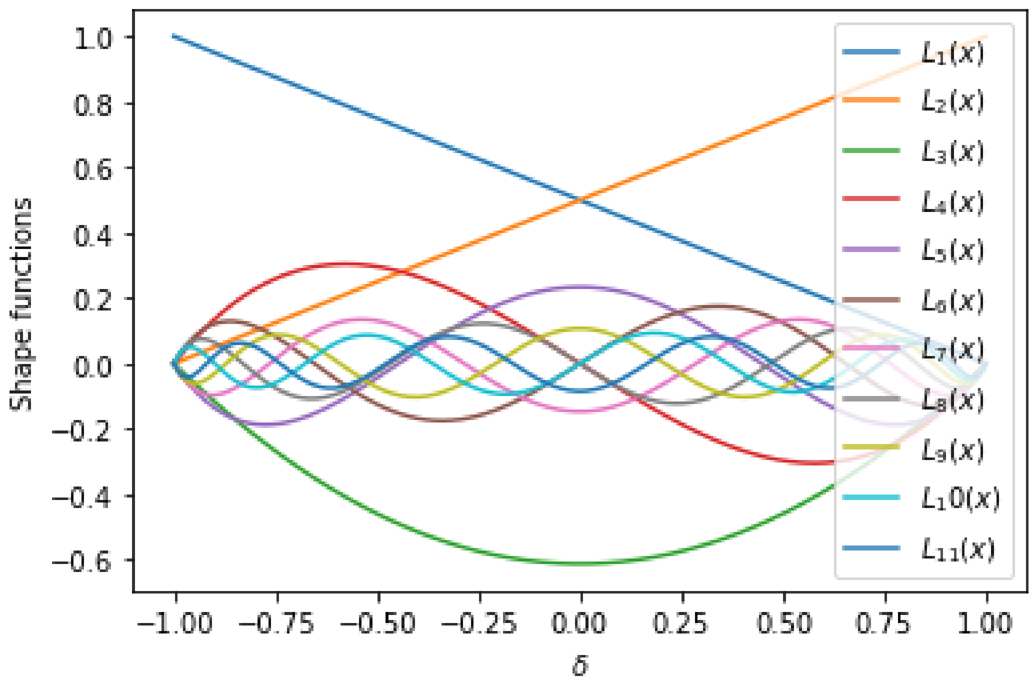
```
plt.legend()
plt.show()
```



1.2.3 High order shape function (Lobatto Hierarchical functions)

[134]:
```
L1 = lambda x: -1/2 * x + 1/2
L2 = lambda x: 1/2 * x + 1/2
L3 = lambda x: 1/(6**0.5) * (1.5*(x - 1)*(x + 1))
L4 = lambda x: 1/(10**0.5) * (5*x*(x - 1)*(x + 1)/2)
L5 = lambda x: 1/(14**0.5) * (7*(x - 1)*(x + 1)*(5*x**2 - 1)/8)
L6 = lambda x: 1/(18**0.5) * (9*x*(x - 1)*(x + 1)*(7*x**2 - 3)/8)
L7 = lambda x: 1/(22**0.5) * (11*(x - 1)*(x + 1)*(21*x**4 - 14*x**2 + 1)/16)
L8 = lambda x: 1/(26**0.5) * (13*x*(x - 1)*(x + 1)*(33*x**4 - 30*x**2 + 5)/16)
L9 = lambda x: 1/(30**0.5) * (15*(x - 1)*(x + 1)*(429*x**6 - 495*x**4 +␣
↪135*x**2 - 5)/128)
L10 = lambda x: 1/(34**0.5) * (17*x*(x - 1)*(x + 1)*(715*x**6 - 1001*x**4 +␣
↪385*x**2 - 35)/128)
L11 = lambda x: 1/(38**0.5) * (19*(x - 1)*(x + 1)*(2431*x**8 - 4004*x**6 +␣
↪2002*x**4 - 308*x**2 + 7)/256)
plt.plot(x,L1(x),label='$L_1(x)$')
plt.plot(x,L2(x),label='$L_2(x)$')
plt.plot(x,L3(x),label='$L_3(x)$')
plt.plot(x,L4(x),label='$L_4(x)$')
```

```
plt.plot(x,L5(x),label='$L_5(x)$')
plt.plot(x,L6(x),label='$L_6(x)$')
plt.plot(x,L7(x),label='$L_7(x)$')
plt.plot(x,L8(x),label='$L_8(x)$')
plt.plot(x,L9(x),label='$L_9(x)$')
plt.plot(x,L10(x),label='$L_10(x)$')
plt.plot(x,L11(x),label='$L_{11}(x)$')
plt.xlabel('$\delta$');plt.ylabel('Shape functions')
plt.legend()
plt.show()
```



1.3 ELementary matrix (Linear)

\$ K = dN*dN \$ Stiffness matrix

$$K = \begin{bmatrix} dN_1 dN_1 & dN_1 dN_2 \\ dN_2 dN_1 & dN_2 dN_2 \end{bmatrix}$$

\$ M = N * N \$ Mass matrix

$$M = \begin{bmatrix} N_1 N_1 & N_1 N_2 \\ N_2 N_1 & N_2 N_2 \end{bmatrix}$$

$$\int_0^\delta \phi_i(x)\phi_j(x)dx = \frac{h}{2}\int_{-1}^{1} N_i(\xi)N_j(\xi)d\xi, \quad \int_0^h \phi_i'(x)\phi_j'(x)dx = \frac{2}{h}\int_{-1}^{1} N_i'(\xi)N_j'(\xi)d\xi$$

[135]:
```
L = 2
nb_e = 100
h = L / nb_e
nb_dof = nb_e + 1  # number of degree of freedom (nodes)
```

[136]:
```
N = [N1, N2]
dN = [dN1, dN2]
K_e = np.zeros((2, 2))
M_e = np.zeros((2, 2))
for i in range(len(dN)):
    for j in range(len(dN)):
        f = lambda x: dN[i](x) * dN[j](x)
        K_e[i, j] = (2 / h) * gauss_legendre_quad(f, 5, -1, 1)
        g = lambda x: N[i](x) * N[j](x)
        M_e[i, j] = (h / 2) * gauss_legendre_quad(g, 5, -1, 1)
```

[137]:
```
K_e
```

[137]:
```
array([[ 50., -50.],
       [-50.,  50.]])
```

[138]:
```
M_e
```

[138]:
```
array([[0.00666667, 0.00333333],
       [0.00333333, 0.00666667]])
```

1.4 Assembly the elementary matrix

ELement by element in normal sort * Global stiffness matrix * Force matrix

[139]:
```
K = np.zeros((nb_dof, nb_dof))
for i in range(nb_e):
    K[i:i+2, i:i+2] += K_e - M_e
```

[140]:
```
K
```

[140]:
```
array([[ 49.99333333, -50.00333333,   0.        , …,   0.        ,
          0.        ,   0.        ],
       [-50.00333333,  99.98666667, -50.00333333, …,   0.        ,
          0.        ,   0.        ],
       [  0.        , -50.00333333,  99.98666667, …,   0.        ,
          0.        ,   0.        ],
       …,
       [  0.        ,   0.        ,   0.        , …,  99.98666667,
        -50.00333333,   0.        ],
```

```
       [  0.        ,   0.        ,   0.        , …, -50.00333333,
         99.98666667, -50.00333333],
       [  0.        ,   0.        ,   0.        , …,   0.        ,
        -50.00333333,  49.99333333]])
```

[141]:
```
F = np.zeros((nb_dof))
# F[0] = -1
```

[142]:
```
F
```

[142]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

1.4.1 Solve the linear system

[143]:
```
U = np.zeros((nb_dof))
# U[:] = np.linalg.solve(K, F)
```

2 Simplification for FEM – Coding tutorial (b)

S.WU

- FEM solution for a concrete mechanics problem
- Convergence of FEM solution

2.1 Problem description

A bar of length $2l$, cross-sectional area A and Young's modulus E. The bar is fixed at $x = 0$, subjected to linear body force cx and applied traction $\bar{t} = -cl^2/A$ at $x = 2l$ as shown in Fig as follow:

<img src="model.png",width =600>

[144]:
```
E = 10e4  # Young modulus Nm-2
A = 1.   # Section area
c = 1.   # Nm-2
l = 1. # m
x_nodes = np.linspace(0, 2*l, nb_dof)
```

[145]:
```
x_nodes
```

[145]: array([0. , 0.02, 0.04, 0.06, 0.08, 0.1 , 0.12, 0.14, 0.16, 0.18, 0.2 ,
        0.22, 0.24, 0.26, 0.28, 0.3 , 0.32, 0.34, 0.36, 0.38, 0.4 , 0.42,
        0.44, 0.46, 0.48, 0.5 , 0.52, 0.54, 0.56, 0.58, 0.6 , 0.62, 0.64,
        0.66, 0.68, 0.7 , 0.72, 0.74, 0.76, 0.78, 0.8 , 0.82, 0.84, 0.86,
        0.88, 0.9 , 0.92, 0.94, 0.96, 0.98, 1. , 1.02, 1.04, 1.06, 1.08,
        1.1 , 1.12, 1.14, 1.16, 1.18, 1.2 , 1.22, 1.24, 1.26, 1.28, 1.3 ,
        1.32, 1.34, 1.36, 1.38, 1.4 , 1.42, 1.44, 1.46, 1.48, 1.5 , 1.52,
        1.54, 1.56, 1.58, 1.6 , 1.62, 1.64, 1.66, 1.68, 1.7 , 1.72, 1.74,
        1.76, 1.78, 1.8 , 1.82, 1.84, 1.86, 1.88, 1.9 , 1.92, 1.94, 1.96,
        1.98, 2.])

Strong form is given

$$\frac{d}{dx}\left(AE\frac{du}{dx}\right) + cx = 0,$$

$$u(0) = 0,$$

$$\bar{t} = E\frac{du}{dx}n\,\Big|_{x=2l} = -\frac{cl^2}{A}$$

[146]:
```
u_ex = lambda x: c/(A*E)*(-x**3/6+l**2*x)
```

Derivation for the weak form (variational formulation)
Multiplication of test function $v \in U^0$ and integration by part from $(0, 2l)$

$$AEv\frac{du}{dx}\Big|_0^{2l} - \int_0^{2l} AE\frac{dv}{dx}\frac{du}{dx}dx + c\int_0^{2l} vxdx = 0 \tag{1}$$

$$\int_0^{2l} AE\frac{dv}{dx}\frac{du}{dx}dx = \int_0^{2l} vcxdx - vcl^2 \tag{2}$$

Thus, weak form for considered problem is given: find $u \in U$

$$\int_0^{2l} AE\frac{dv}{dx}\frac{du}{dx}dx = cl^2,\ \ v \in U^0 \tag{3}$$

2.1.1 Boundary conditions in considered problem

- Essential (Dirichlet) BCs $u(0) = 0$
- Natural (Neumman) BCs

[147]:
```
F_e = np.zeros((2))
for i in range(nb_e):
    ff = lambda x: ((x_nodes[i+1]-x)/h)*c*x
    gg = lambda x: ((x-x_nodes[i])/h)*c*x
    F_e[0]= gauss_legendre_quad(ff, 5, x_nodes[i], x_nodes[i+1])
```

```python
    F_e[1] = gauss_legendre_quad(gg, 5, x_nodes[i], x_nodes[i+1])
    F[i:i+2] += F_e
F[-1] += -c*l**2
F[0] = 0

K = np.zeros((nb_dof, nb_dof))
for i in range(nb_e):
    K[i:i+2, i:i+2] += A*E*K_e
K[0] = 0
K[:,0] = 0
K[0,0] = 1
U[:] = np.linalg.solve(K, F)
```

[148]:
```python
F
```

[148]:
```
array([ 0.00000000e+00,  4.00000000e-04,  8.00000000e-04,  1.20000000e-03,
        1.60000000e-03,  2.00000000e-03,  2.40000000e-03,  2.80000000e-03,
        3.20000000e-03,  3.60000000e-03,  4.00000000e-03,  4.40000000e-03,
        4.80000000e-03,  5.20000000e-03,  5.60000000e-03,  6.00000000e-03,
        6.40000000e-03,  6.80000000e-03,  7.20000000e-03,  7.60000000e-03,
        8.00000000e-03,  8.40000000e-03,  8.80000000e-03,  9.20000000e-03,
        9.60000000e-03,  1.00000000e-02,  1.04000000e-02,  1.08000000e-02,
        1.12000000e-02,  1.16000000e-02,  1.20000000e-02,  1.24000000e-02,
        1.28000000e-02,  1.32000000e-02,  1.36000000e-02,  1.40000000e-02,
        1.44000000e-02,  1.48000000e-02,  1.52000000e-02,  1.56000000e-02,
        1.60000000e-02,  1.64000000e-02,  1.68000000e-02,  1.72000000e-02,
        1.76000000e-02,  1.80000000e-02,  1.84000000e-02,  1.88000000e-02,
        1.92000000e-02,  1.96000000e-02,  2.00000000e-02,  2.04000000e-02,
        2.08000000e-02,  2.12000000e-02,  2.16000000e-02,  2.20000000e-02,
        2.24000000e-02,  2.28000000e-02,  2.32000000e-02,  2.36000000e-02,
        2.40000000e-02,  2.44000000e-02,  2.48000000e-02,  2.52000000e-02,
        2.56000000e-02,  2.60000000e-02,  2.64000000e-02,  2.68000000e-02,
        2.72000000e-02,  2.76000000e-02,  2.80000000e-02,  2.84000000e-02,
        2.88000000e-02,  2.92000000e-02,  2.96000000e-02,  3.00000000e-02,
        3.04000000e-02,  3.08000000e-02,  3.12000000e-02,  3.16000000e-02,
        3.20000000e-02,  3.24000000e-02,  3.28000000e-02,  3.32000000e-02,
        3.36000000e-02,  3.40000000e-02,  3.44000000e-02,  3.48000000e-02,
        3.52000000e-02,  3.56000000e-02,  3.60000000e-02,  3.64000000e-02,
        3.68000000e-02,  3.72000000e-02,  3.76000000e-02,  3.80000000e-02,
        3.84000000e-02,  3.88000000e-02,  3.92000000e-02,  3.96000000e-02,
       -9.80066667e-01])
```

[149]:
```python
K
```

[149]:
```
array([[ 1.e+00,  0.e+00,  0.e+00, …,  0.e+00,  0.e+00,  0.e+00],
       [ 0.e+00,  1.e+07, -5.e+06, …,  0.e+00,  0.e+00,  0.e+00],
       [ 0.e+00, -5.e+06,  1.e+07, …,  0.e+00,  0.e+00,  0.e+00],
```

```
 …,
 [ 0.e+00,  0.e+00,  0.e+00, …,  1.e+07, -5.e+06,  0.e+00],
 [ 0.e+00,  0.e+00,  0.e+00, …, -5.e+06,  1.e+07, -5.e+06],
 [ 0.e+00,  0.e+00,  0.e+00, …,  0.e+00, -5.e+06,  5.e+06]])
```

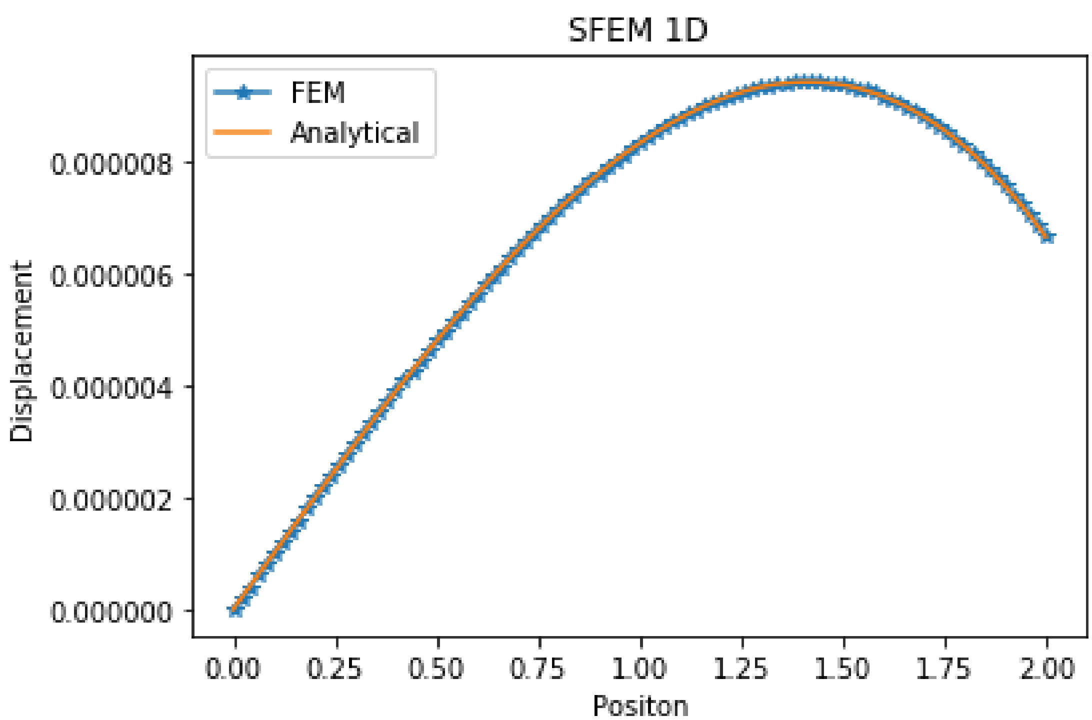
2.1.2 Mesh and element size

- L physical geometry
- h elements size
- nb_dof number of degree of freedom

2.2 Computational results

[150]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.set_title('SFEM 1D')
ax.set_xlabel('Positon')
ax.set_ylabel('Displacement')
ax.plot(x_nodes, U, '-*', label='FEM')
U_ex =  [u_ex(x) for x in x_nodes]
ax.plot(x_nodes, U_ex, '-', label='Analytical')
ax.legend()
```

[150]: <matplotlib.legend.Legend at 0x7f8f26696310>



[151]:
```
u_ex(0)
```

[151]: 0.0

[152]:
```
u_ex(x_nodes)
```

[152]:
```
array([0.00000000e+00, 1.99986667e-07, 3.99893333e-07, 5.99640000e-07,
       7.99146667e-07, 9.98333333e-07, 1.19712000e-06, 1.39542667e-06,
       1.59317333e-06, 1.79028000e-06, 1.98666667e-06, 2.18225333e-06,
       2.37696000e-06, 2.57070667e-06, 2.76341333e-06, 2.95500000e-06,
       3.14538667e-06, 3.33449333e-06, 3.52224000e-06, 3.70854667e-06,
       3.89333333e-06, 4.07652000e-06, 4.25802667e-06, 4.43777333e-06,
       4.61568000e-06, 4.79166667e-06, 4.96565333e-06, 5.13756000e-06,
       5.30730667e-06, 5.47481333e-06, 5.64000000e-06, 5.80278667e-06,
       5.96309333e-06, 6.12084000e-06, 6.27594667e-06, 6.42833333e-06,
       6.57792000e-06, 6.72462667e-06, 6.86837333e-06, 7.00908000e-06,
       7.14666667e-06, 7.28105333e-06, 7.41216000e-06, 7.53990667e-06,
       7.66421333e-06, 7.78500000e-06, 7.90218667e-06, 8.01569333e-06,
       8.12544000e-06, 8.23134667e-06, 8.33333333e-06, 8.43132000e-06,
       8.52522667e-06, 8.61497333e-06, 8.70048000e-06, 8.78166667e-06,
       8.85845333e-06, 8.93076000e-06, 8.99850667e-06, 9.06161333e-06,
       9.12000000e-06, 9.17358667e-06, 9.22229333e-06, 9.26604000e-06,
       9.30474667e-06, 9.33833333e-06, 9.36672000e-06, 9.38982667e-06,
       9.40757333e-06, 9.41988000e-06, 9.42666667e-06, 9.42785333e-06,
       9.42336000e-06, 9.41310667e-06, 9.39701333e-06, 9.37500000e-06,
       9.34698667e-06, 9.31289333e-06, 9.27264000e-06, 9.22614667e-06,
       9.17333333e-06, 9.11412000e-06, 9.04842667e-06, 8.97617333e-06,
       8.89728000e-06, 8.81166667e-06, 8.71925333e-06, 8.61996000e-06,
       8.51370667e-06, 8.40041333e-06, 8.28000000e-06, 8.15238667e-06,
       8.01749333e-06, 7.87524000e-06, 7.72554667e-06, 7.56833333e-06,
       7.40352000e-06, 7.23102667e-06, 7.05077333e-06, 6.86268000e-06,
       6.66666667e-06])
```

3 convergence analysis

[155]:
```
e_l2 = 0
for i in range(nb_e):
        u_ex_s = integrate.quad(u_ex, 0, 2)[0]
        x_trans = lambda x: (2 * x) / h - (x_nodes[i] + x_nodes[i+1])/h
          #  use of analytical lobatto expressions
        u_FE = lambda x: sum(N[j](x_trans(x)) * U[i + j] for j in range(len(N)))
        f_error = lambda x:(u_ex(x)-u_FE(x))**2
```

```
        e_l2_element = gauss_legendre_quad(f_error, 20, x_nodes[i],␣
 ↪x_nodes[i+1])
#        print(e_l2_element)
        e_l2 += e_l2_element
e_norm = np.sqrt(e_l2 / u_ex_s)
```

[156]:
```
e_norm
```

[156]: 1.632973722723116e-07