

Lecture 1 - 3D linear elasticity - code

```
import sympy as sp
import numpy as np
import pyvista as pv
import matplotlib.pyplot as plt

def plot_matrix_heatmap(K_num, zero_color='lightgray', cmap='viridis', dpi=150, atol=1e-12):

    # Make a copy, mask zeros
    data = np.array(K_num, dtype=float)
    mask = np.isclose(data, 0.0, atol=atol)
    data_masked = np.ma.array(data, mask=mask) # mask zeros

    # Create a colormap and set color for masked (i.e. zero) entries
    cmap_mod = plt.get_cmap(cmap).copy()
    cmap_mod.set_bad(zero_color) # color for masked entries

    fig, ax = plt.subplots(dpi=dpi)
    im = ax.imshow(data_masked, cmap=cmap_mod, interpolation='none', origin='upper')
    plt.colorbar(im, label='Stiffness Value')
    plt.title(f'Element Stiffness Matrix Heatmap ({data.shape[0]}x{data.shape[1]})')
    plt.xlabel('DOF Index')
    plt.ylabel('DOF Index')

    # Draw gridlines between cells
    n, m = data.shape
    ax.set_xticks(np.arange(-0.5, m, 1), minor=False)
    ax.set_yticks(np.arange(-0.5, n, 1), minor=False)
    ax.set_xticklabels([])
    ax.set_yticklabels([])
    ax.grid(color='k', linestyle='--', linewidth=1)

    plt.show()
```

```
# Problem parameters

# Cuboid dimensions [mm]
L, W, H = sp.Integer(100.0), sp.Integer(50.0), sp.Integer(5.0)

# Material properties
E_val = 7e4 # MPa
v_val = 0.3

# Traction load magnitude [N/mm2] to be applied along Z direction
traction = 0.5
```

From beam theory, we expect displacement to be

$$\delta_{\max} = \frac{PL^3}{3EI}$$

where $P = \text{traction} \times WH$ is the force in Newtons and $I = WH^3/12$ is the second moment of area

```
# Expected result from beam theory
I = (W * H**3) / 12 # Second moment of area
```

```

P = traction * W * H # Total load on the beam
δ_theory = (P * L**3) / (3 * E_val * I) # Deflection at free end
print(f'Expected deflection from beam theory: {δ_theory:.4f} mm')

```

Expected deflection from beam theory: 1.1429 mm

```

ξ, η, ζ = sp.symbols('ξ η ζ')
E, v = sp.symbols('E v')

```

```

# all combinations of ±1 for (ξ_a, η_a, ζ_a)
nodes_natural_coordinates = [
    [-1, -1, -1],
    [ 1, -1, -1],
    [ 1,  1, -1],
    [-1,  1, -1],
    [-1, -1,  1],
    [ 1, -1,  1],
    [ 1,  1,  1],
    [-1,  1,  1],
]
N = [(1 + ξ*ξa)*(1 + η*ηa)*(1 + ζ*ζa)/sp.Integer(8) for (ξa, ηa, ζa) in nodes_natural_coordinates]
N

```

```

[(1 - ζ)*(1 - η)*(1 - ξ)/8,
 (1 - ζ)*(1 - η)*(ξ + 1)/8,
 (1 - ζ)*(η + 1)*(ξ + 1)/8,
 (1 - ζ)*(1 - ξ)*(η + 1)/8,
 (1 - η)*(1 - ξ)*(ζ + 1)/8,
 (1 - η)*(ζ + 1)*(ξ + 1)/8,
 (ζ + 1)*(η + 1)*(ξ + 1)/8,
 (1 - ξ)*(ζ + 1)*(η + 1)/8]

```

```

# Map from natural to physical coordinates
x, y, z = L/2 * (ξ + 1), W/2 * (η + 1), H/2 * (ζ + 1)

```

```

# Create a pv.UnstructuredGrid from nodes_natural_coordinates and the functions x, y, z
# Use a single hex element connectivity

```

```

nodes = np.array([[x.subs({ξ: ξa, η: ηa, ζ: ζa}).evalf(),
                  y.subs({ξ: ξa, η: ηa, ζ: ζa}).evalf(),
                  z.subs({ξ: ξa, η: ηa, ζ: ζa}).evalf()] for (ξa, ηa, ζa) in
nodes_natural_coordinates], dtype=np.float64)

```

```

cells = np.hstack([[8], np.arange(8)]) # 8 nodes per hexahedron
cell_types = np.array([pv.CellType.HEXAHEDRON])
mesh = pv.UnstructuredGrid(cells, cell_types, nodes)

```

```
# Point labels
```

```

points = mesh.points # shape (n_nodes, 3)
labels = [str(i) for i in range(points.shape[0])]

```

```
# OBS: set export LIBGL_ALWAYS_SOFTWARE=1 before running this script if you get an OpenGL error
```

```

pv.start_xvfb() # starts a virtual framebuffer
pv.OFF_SCREEN = True # force off-screen rendering
pl = pv.Plotter()
pl.add_mesh(mesh, show_edges=True, color='lightgray', opacity=0.5)
pl.add_point_labels(points, labels,

```

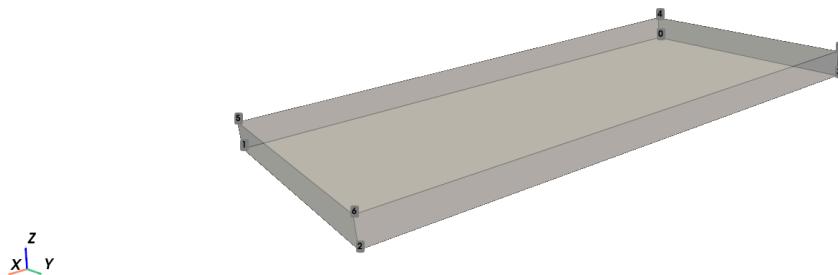
```

        point_size=5,           # size of the point glyph
        font_size=12,            # font size of the labels
        text_color='black',
        italic=False, bold=True,
        always_visible=True)

pl.add_axes(line_width=3,   # thickness of arrow-axes
            cone_radius=0.06,
            shaft_length=0.9,
            tip_length=0.3,
            label_size=(0.5, 0.2))

pl.show()

```



```

# find the Jacobian matrix d(x,y,z)/d(ξ,η,ζ)
J = sp.Matrix([[sp.diff(x, ξ), sp.diff(x, η), sp.diff(x, ζ)],
               [sp.diff(y, ξ), sp.diff(y, η), sp.diff(y, ζ)],
               [sp.diff(z, ξ), sp.diff(z, η), sp.diff(z, ζ)]])
J

```

$$\begin{bmatrix} 50 & 0 & 0 \\ 0 & 25 & 0 \\ 0 & 0 & \frac{5}{2} \end{bmatrix}$$

```
J_inv = J.inv()
J_inv
```

$$\begin{bmatrix} \frac{1}{50} & 0 & 0 \\ 0 & \frac{1}{25} & 0 \\ 0 & 0 & \frac{2}{5} \end{bmatrix}$$

```

# dN_i/ dx = (dN_i / dξ) * (dξ / dx), etc.
dN_dξ = [sp.diff(Na, ξ) for Na in N]
dN_dη = [sp.diff(Na, η) for Na in N]
dN_dζ = [sp.diff(Na, ζ) for Na in N]

# now apply the chain rule with J_inv
dN_dx = []
dN_dy = []
dN_dz = []
for i in range(len(N)):

    # vector of reference derivatives for shape i
    dN_ref = sp.Matrix([dN_dξ[i], dN_dη[i], dN_dζ[i]])

```

```

# multiply by the inverse Jacobian
dN_phys = J_inv * dN_ref

# extract physical derivatives
dN_dx.append(dN_phys[0])
dN_dy.append(dN_phys[1])
dN_dz.append(dN_phys[2])

dN_dx

```

```

[-(1 - ζ)*(1 - η)/400,
 (1 - ζ)*(1 - η)/400,
 (1 - ζ)*(η + 1)/400,
 -(1 - ζ)*(η + 1)/400,
 -(1 - η)*(ζ + 1)/400,
 (1 - η)*(ζ + 1)/400,
 (ζ + 1)*(η + 1)/400,
 -(ζ + 1)*(η + 1)/400]

```

```
# Build B matrix as function of (ξ,η,ζ)
```

```

def B_sym():
    B = sp.zeros(6, 3*8)
    for a in range(8):
        i = 3*a
        B[0, i+0] = dN_dx[a]
        B[1, i+1] = dN_dy[a]
        B[2, i+2] = dN_dz[a]
        B[3, i+0] = dN_dy[a]
        B[3, i+1] = dN_dx[a]
        B[4, i+1] = dN_dz[a]
        B[4, i+2] = dN_dy[a]
        B[5, i+0] = dN_dz[a]
        B[5, i+2] = dN_dx[a]
    return B

```

```
B_sym_expr = B_sym()
B_sym_expr
```

$$\begin{bmatrix} -\frac{(1-\zeta)(1-\eta)}{400} & 0 & 0 & \frac{(1-\zeta)(1-\eta)}{400} & 0 & 0 & \frac{(1-\zeta)(\eta+1)}{400} & 0 & 0 & -\frac{(1-\zeta)}{\zeta} \\ 0 & -\frac{(1-\zeta)(1-\xi)}{200} & 0 & 0 & -\frac{(1-\zeta)(\xi+1)}{200} & 0 & 0 & \frac{(1-\zeta)(\xi+1)}{200} & 0 & 0 \\ 0 & 0 & -\frac{(1-\eta)(1-\xi)}{20} & 0 & 0 & -\frac{(1-\eta)(\xi+1)}{20} & 0 & 0 & 0 & -\frac{(\eta+1)(\xi+1)}{20} \\ -\frac{(1-\zeta)(1-\xi)}{200} & -\frac{(1-\zeta)(1-\eta)}{400} & 0 & -\frac{(1-\zeta)(\xi+1)}{200} & \frac{(1-\zeta)(1-\eta)}{400} & 0 & \frac{(1-\zeta)(\xi+1)}{200} & \frac{(1-\zeta)(\eta+1)}{400} & 0 & \frac{(1-\zeta)}{2\zeta} \\ 0 & -\frac{(1-\eta)(1-\xi)}{20} & -\frac{(1-\zeta)(1-\xi)}{200} & 0 & -\frac{(1-\eta)(\xi+1)}{20} & -\frac{(1-\zeta)(\xi+1)}{200} & 0 & -\frac{(\eta+1)(\xi+1)}{20} & \frac{(1-\zeta)(\xi+1)}{200} & 0 \\ -\frac{(1-\eta)(1-\xi)}{20} & 0 & -\frac{(1-\zeta)(1-\eta)}{400} & -\frac{(1-\eta)(\xi+1)}{20} & 0 & \frac{(1-\zeta)(1-\eta)}{400} & -\frac{(\eta+1)(\xi+1)}{20} & 0 & \frac{(1-\zeta)(\eta+1)}{400} & -\frac{(1-\xi)}{2\zeta} \end{bmatrix}$$

```

# 4) define elasticity matrix D (isotropic, e.g. E=1, nu=0.3 for simplicity)
lam = E*v / ((1+v)*(1-2*v))
mu = E / (2*(1+v))
D = sp.zeros(6,6)
for i in range(3):
    D[i,i] = lam + 2*mu
    for j in range(3):
        if i!=j:
            D[i,j] = lam
for i in range(3,6):
    D[i,i] = mu

```

D

$$\begin{bmatrix} \frac{E\nu}{(1-2\nu)(\nu+1)} + \frac{2E}{2\nu+2} & \frac{E\nu}{(1-2\nu)(\nu+1)} & \frac{E\nu}{(1-2\nu)(\nu+1)} & 0 & 0 & 0 \\ \frac{E\nu}{(1-2\nu)(\nu+1)} & \frac{E\nu}{(1-2\nu)(\nu+1)} + \frac{2E}{2\nu+2} & \frac{E\nu}{(1-2\nu)(\nu+1)} & 0 & 0 & 0 \\ \frac{E\nu}{(1-2\nu)(\nu+1)} & \frac{E\nu}{(1-2\nu)(\nu+1)} & \frac{E\nu}{(1-2\nu)(\nu+1)} + \frac{2E}{2\nu+2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{E}{2\nu+2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{E}{2\nu+2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{E}{2\nu+2} \end{bmatrix}$$

```
# 5) form integrand for stiffness: integrand = B^T D B * detJ --> I am adding detJ here for convenience
detJ = sp.det(J)
detJ
```

3125

```
# Build integrand
K_sym = (B_sym_expr.T * D * B_sym_expr) * detJ
K_sym.simplify()
K_sym # 24 x 24 symbolic stiffness matrix
```

$$\begin{aligned}
& \frac{5E\left((\zeta-1)^2(\eta-1)^2(\nu-1)+(\zeta-1)^2(4\nu-2)(\xi-1)^2+(\eta-1)^2(400\nu-200)(\xi-1)^2\right)}{256(\nu+1)(2\nu-1)} \\
& - \frac{5E(\zeta-1)^2(\eta-1)(\xi-1)}{256(\nu+1)(2\nu-1)} \\
& - \frac{25E(\zeta-1)(\eta-1)^2(\xi-1)}{128(\nu+1)(2\nu-1)} \\
& \frac{5E\left((1-\nu)(\zeta-1)^2(\eta-1)^2-2(\zeta-1)^2(2\nu-1)(\xi-1)(\xi+1)-200(\eta-1)^2(2\nu-1)(\xi-1)(\xi+1)\right)}{256(\nu+1)(2\nu-1)} \\
& \frac{5E(\zeta-1)^2(\eta-1)(2\nu(\xi+1)-(2\nu-1)(\xi-1))}{256(\nu+1)(2\nu-1)} \\
& \frac{25E(\zeta-1)(\eta-1)^2(2\nu(\xi+1)-(2\nu-1)(\xi-1))}{128(\nu+1)(2\nu-1)} \\
& \frac{5E\left(-(1-\nu)(\zeta-1)^2(\eta-1)(\eta+1)+2(\zeta-1)^2(2\nu-1)(\xi-1)(\xi+1)+200(\eta-1)(\eta+1)(2\nu-1)(\xi-1)(\xi+1)\right)}{256(\nu+1)(2\nu-1)} \\
& \frac{5E(\zeta-1)^2(-2\nu(\eta-1)(\xi+1)+(\eta+1)(2\nu-1)(\xi-1))}{256(\nu+1)(2\nu-1)} \\
& \frac{25E(\zeta-1)(\eta-1)(\eta+1)(-2\nu(\xi+1)+(2\nu-1)(\xi-1))}{128(\nu+1)(2\nu-1)} \\
& \frac{5E\left((1-\nu)(\zeta-1)^2(\eta-1)(\eta+1)+(2-4\nu)(\zeta-1)^2(\xi-1)^2-200(\eta-1)(\eta+1)(2\nu-1)(\xi-1)^2\right)}{256(\nu+1)(2\nu-1)} \\
& \frac{5E(\zeta-1)^2(\xi-1)(2\nu(\eta-1)-(\eta+1)(2\nu-1))}{256(\nu+1)(2\nu-1)} \\
& \frac{25E(\zeta-1)(\eta-1)(\eta+1)(\xi-1)}{128(\nu+1)(2\nu-1)} \\
& \frac{5E\left((1-\nu)(\zeta-1)(\zeta+1)(\eta-1)^2+(200-400\nu)(\eta-1)^2(\xi-1)^2-2(\zeta-1)(\zeta+1)(2\nu-1)(\xi-1)^2\right)}{256(\nu+1)(2\nu-1)} \\
& \frac{5E(\zeta-1)(\zeta+1)(\eta-1)(\xi-1)}{256(\nu+1)(2\nu-1)} \\
& \frac{25E(\eta-1)^2(\xi-1)(2\nu(\zeta-1)-(\zeta+1)(2\nu-1))}{128(\nu+1)(2\nu-1)} \\
& \frac{5E((1-\nu)(\zeta-1)(\zeta+1)(\eta-1)^2+2(\zeta-1)(\zeta+1)(2\nu-1)(\xi-1)(\xi+1)+200(\eta-1)^2(2\nu-1)(\xi-1)(\xi+1))}{256(\nu+1)(2\nu-1)} \\
& \frac{5E(\zeta-1)(\zeta+1)(\eta-1)(-2\nu(\xi+1)+(2\nu-1)(\xi-1))}{256(\nu+1)(2\nu-1)} \\
& \frac{25E(\eta-1)^2(-2\nu(\zeta-1)(\xi+1)+(\zeta+1)(2\nu-1)(\xi-1))}{128(\nu+1)(2\nu-1)} \\
& \frac{5E((1-\nu)(\zeta-1)(\zeta+1)(\eta-1)^2-2(\zeta-1)(\zeta+1)(2\nu-1)(\xi-1)(\xi+1)-200(\eta-1)(\eta+1)(2\nu-1)(\xi-1)(\xi+1))}{256(\nu+1)(2\nu-1)} \\
& \frac{5E(\zeta-1)(\zeta+1)(2\nu(\eta-1)(\xi+1)-(\eta+1)(2\nu-1)(\xi-1))}{256(\nu+1)(2\nu-1)} \\
& \frac{25E(\eta-1)(\eta+1)(2\nu(\zeta-1)(\xi+1)-(\zeta+1)(2\nu-1)(\xi-1))}{128(\nu+1)(2\nu-1)} \\
& \frac{5E\left(-(1-\nu)(\zeta-1)(\zeta+1)(\eta-1)^2+2(\zeta-1)(\zeta+1)(2\nu-1)(\xi-1)(\xi+1)+200(\eta-1)(\eta+1)(2\nu-1)(\xi-1)^2\right)}{256(\nu+1)(2\nu-1)} \\
& \frac{5E(\zeta-1)(\zeta+1)(\xi-1)(-2\nu(\eta-1)+(\eta+1)(2\nu-1))}{256(\nu+1)(2\nu-1)} \\
& \frac{25E(\eta-1)(\eta+1)(\xi-1)(-2\nu(\zeta-1)+(\zeta+1)(2\nu-1))}{128(\nu+1)(2\nu-1)} \\
& \frac{5E(\zeta-1)^2(-2\nu(\zeta-1)(\eta+1)+(\zeta+1)(2\nu-1))}{64(\nu+1)(2\nu-1)} \\
& \frac{25E(\zeta-1)(\eta-1)^2(8(\zeta-1)(\xi+1)-(\zeta-1)^2(\eta-1)(\xi+1))}{512(\nu+1)(2\nu-1)} \\
& \frac{5E\left(8(1-\nu)(\zeta-1)^2(\xi-1)(\xi+1)-(\zeta-1)^2(\eta-1)(\eta+1)(2\nu-1)-400(\zeta-1)(\eta-1)(\eta+1)(2\nu-1)\right)}{512(\nu+1)(2\nu-1)} \\
& \frac{25E(\zeta-1)(\eta-1)(\xi-1)(\xi+1)}{64(\nu+1)(2\nu-1)} \\
& \frac{5E(\zeta-1)^2(-2\nu(\eta+1)(\xi-1)+(\eta-1)(2\nu-1))}{256(\nu+1)(2\nu-1)} \\
& \frac{25E(\zeta-1)(\xi-1)^2(8(\zeta-1)(\xi+1)-(\zeta-1)^2(\eta-1)(\eta+1)(2\nu-1)-400(\zeta-1)(\eta-1)(\eta+1)(2\nu-1))}{512(\nu+1)(2\nu-1)} \\
& \frac{25E(\zeta-1)(\xi-1)^2(2\nu(\eta+1)-(\eta-1)(2\nu-1))}{64(\nu+1)(2\nu-1)} \\
& \frac{5E(\zeta-1)(\zeta+1)(\eta-1)(\xi-1)}{256(\nu+1)(2\nu-1)} \\
& \frac{25E(\zeta-1)(\zeta+1)(\xi-1)(\xi+1)}{512(\nu+1)(2\nu-1)} \\
& \frac{5E(\zeta-1)(\zeta+1)(2\nu(\eta-1)(\xi-1)-(\eta-1)(2\nu-1))}{256(\nu+1)(2\nu-1)} \\
& \frac{25E(\zeta-1)(\zeta+1)(2\nu(\zeta-1)(\eta+1)-(\zeta+1)(2\nu-1))}{64(\nu+1)(2\nu-1)} \\
& \frac{5E(\zeta-1)(\zeta+1)(\xi-1)(-2\nu(\eta+1)+(\eta-1)(2\nu-1))}{256(\nu+1)(2\nu-1)} \\
& \frac{25E(\zeta-1)(\zeta+1)(\xi-1)(-2\nu(\zeta-1)+(\zeta+1)(2\nu-1))}{64(\nu+1)(2\nu-1)} \\
& \frac{5E\left(-8(1-\nu)(\zeta-1)(\zeta+1)(\xi-1)^2+(\zeta-1)(\zeta+1)(\eta-1)(\eta+1)(2\nu-1)\right)}{512(\nu+1)(2\nu-1)} \\
& \frac{25E(\eta-1)(\xi-1)^2(2\nu(\zeta-1)-(\zeta+1)(2\nu-1))}{64(\nu+1)(2\nu-1)} \\
& \frac{5E(\zeta-1)(\zeta+1)(\eta-1)(-2\nu(\xi-1)+(\xi+1)(2\nu-1))}{256(\nu+1)(2\nu-1)} \\
& \frac{25E(\zeta-1)(\zeta+1)(2\nu(\eta-1)(\xi-1)-(\eta-1)(2\nu-1))}{64(\nu+1)(2\nu-1)} \\
& \frac{5E(\zeta-1)(\zeta+1)(\xi-1)(-2\nu(\zeta-1)+(\zeta+1)(2\nu-1))}{256(\nu+1)(2\nu-1)} \\
& \frac{25E(\zeta-1)(\zeta+1)(\xi-1)(-2\nu(\zeta-1)+(\zeta+1)(2\nu-1))}{64(\nu+1)(2\nu-1)}
\end{aligned}$$

```
# Replace E and nu with numeric values
K_sym = K_sym.subs({E: E_val, v: v_val})
```

```
# 6) lambdify K integrand for numeric evaluation
f_K = sp.lambdify((xi, eta, zeta), K_sym, 'numpy')
```

```
def integrate_K(n_gauss=2):
    # get Gauss points and weights for 1D
    if n_gauss == 1:
        points = [0.0]
        weights = [2.0]
    elif n_gauss == 2:
        p = 1/np.sqrt(3)
```

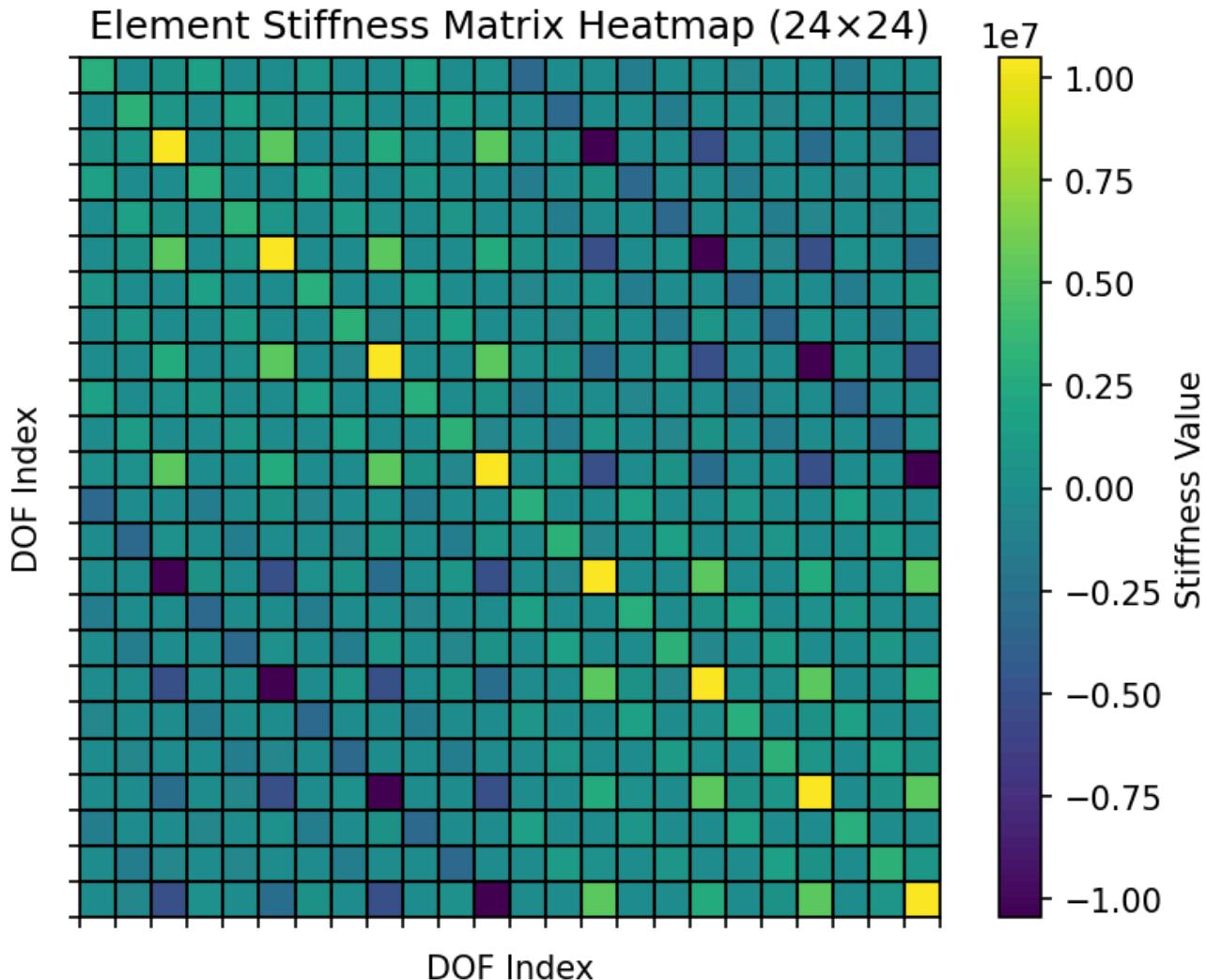
```

    points = [-p, p]
    weights = [1.0, 1.0]
else:
    raise ValueError("extend as needed")
K_e = np.zeros((24,24))
for xi, wxi in zip(points, weights):
    for eta, weta in zip(points, weights):
        for zeta, wzeta in zip(points, weights):
            K_e += f_K(xi, eta, zeta) * (wxi*weta*wzeta)
return K_e

# Example: compute for E=210e9, nu=0.3 with 2x2x2 Gauss
K_num = integrate_K(n_gauss=2)

plot_matrix_heatmap(K_num)

```



```

# LOAD VECTOR COMPUTATION

# Determinant of the surface Jacobian: can be obtained the cross product of the two tangent
# vectors on the surface
# on the surface  $\xi=1$ , parametrize with  $(\eta, \zeta)$ 
# surface Jacobian magnitude
x_surf = x.subs(xi, 1)
y_surf = y.subs(xi, 1)
z_surf = z.subs(xi, 1)
dxdeteta = sp.Matrix([sp.diff(x_surf, eta),
                     sp.diff(y_surf, eta),
                     sp.diff(z_surf, eta)])

```

```

        sp.diff(z_surf, η)])
dxdzeta = sp.Matrix([sp.diff(x_surf, ζ),
                     sp.diff(y_surf, ζ),
                     sp.diff(z_surf, ζ)])
J_surf = (dxdzeta.cross(dxdzeta)).norm()
J_surf

```

$$\frac{125}{2}$$

```

# From N: list of 8 expressions corresponding to shape functions, compute the matrix [N] = [N1 0 0
N2 0 0 ... N8 0 0; 0 N1 0 0 N2 0 ... 0 N8; 0 0 N1 0 0 N2 ... 0 0 N8]
def N_matrix():
    N_mat = sp.zeros(3, 3*8)
    for a in range(8):
        i = 3*a
        N_mat[0, i+0] = N[a]
        N_mat[1, i+1] = N[a]
        N_mat[2, i+2] = N[a]
    return N_mat

N_mat_expr = N_matrix()

# For the boundary, I only care about the entries cooresponding to N1, N2, N5 and N6 (with Na
ranging from 0 to 7)
boundary_dof_indices = [1, 2, 5, 6]

N_surf = N_mat_expr.subs({ξ: 1})
N_surf


$$\begin{bmatrix} 0 & 0 & 0 & \frac{(1-\zeta)(1-\eta)}{4} & 0 & 0 & \frac{(1-\zeta)(\eta+1)}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{(1-\eta)(\zeta+1)}{4} & 0 \\ 0 & 0 & 0 & 0 & \frac{(1-\zeta)(1-\eta)}{4} & 0 & 0 & \frac{(1-\zeta)(\eta+1)}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{(1-\eta)(\zeta+1)}{4} \\ 0 & 0 & 0 & 0 & 0 & \frac{(1-\zeta)(1-\eta)}{4} & 0 & 0 & \frac{(1-\zeta)(\eta+1)}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$


t = sp.Matrix([[0], [0], [-traction]]) # force load vector on the surface
surface_integrand = N_surf.transpose() * t * J_surf
traction_surf = sp.lambdify((η, ζ), surface_integrand, 'numpy')

def integrate_surface_load(n_gauss=2):
    if n_gauss == 2:
        p = 1/np.sqrt(3)
        pts = [-p, p]
        wts = [1.0, 1.0]
    else:
        raise ValueError("extend as needed")
    Fe = np.zeros((24, 1))
    for ηi, wη in zip(pts, wts):
        for ζj, wζ in zip(pts, wts):
            Fe += traction_surf(ηi, ζj) * (wη*wζ)
    return Fe

loads_vec = integrate_surface_load(n_gauss=2)
loads_vec.T

array([[ 0. ,  0. ,  0. ,  0. ,  0. , -31.25,  0. ,  0. ,
       -31.25,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,
       0. , -31.25,  0. ,  0. , -31.25,  0. ,  0. ,  0. ]])
```

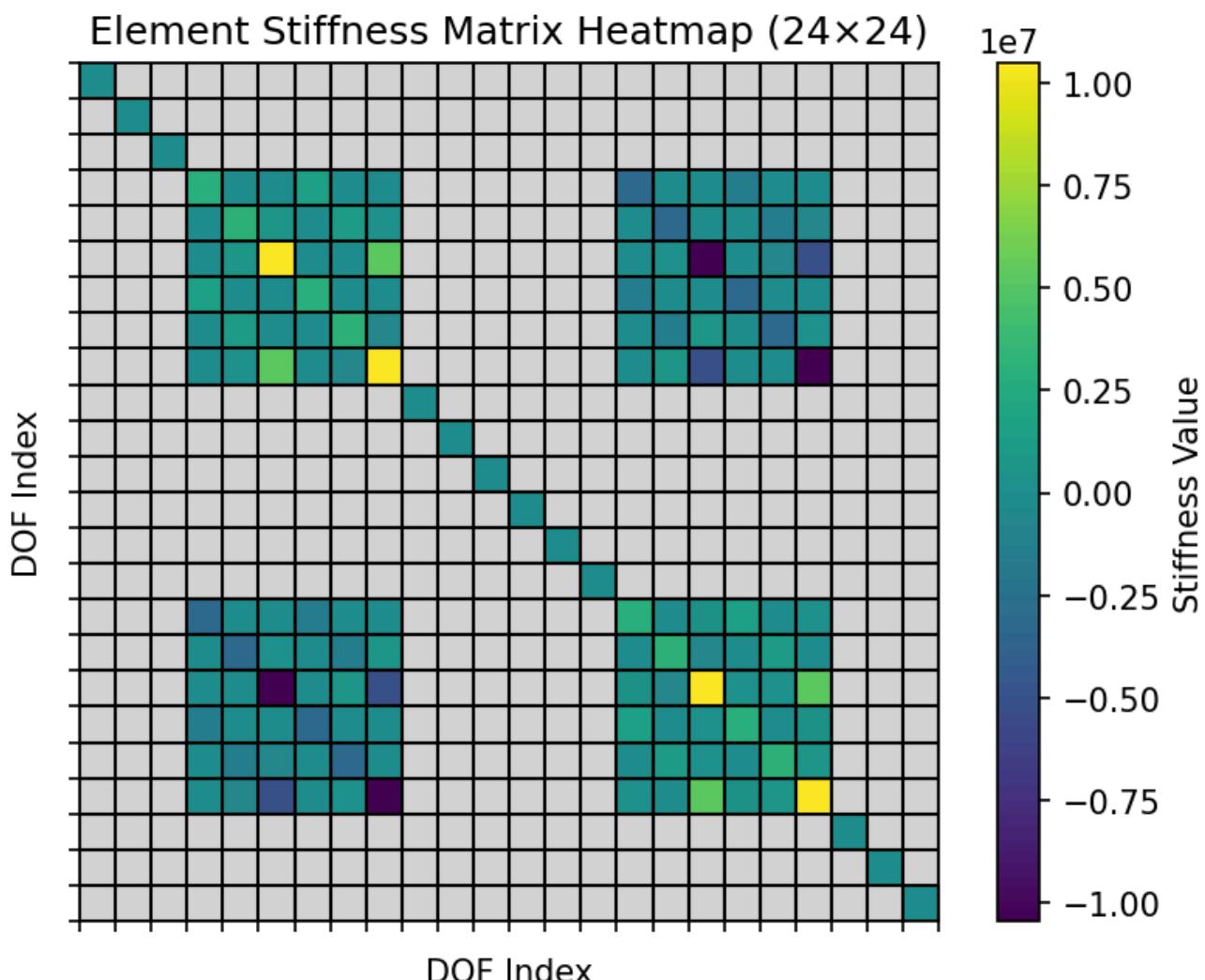
```
loads_vec.T
```

```
array([[ 0. ,  0. ,  0. ,  0. ,  0. , -31.25,  0. ,  0. ,
       -31.25,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,
       0. , -31.25,  0. ,  0. , -31.25,  0. ,  0. ,  0. ]])
```

```
# DIRICHLET CONDITIONS
# Suppose node numbering 0-7 internally; boundary_nodes = [0, 3, 4, 7] # (1,4,5,8 in 1-based)
boundary_nodes = [0, 3, 4, 7]
# Convert to DOF indices
fixed_dofs = []
for n in boundary_nodes:
    fixed_dofs += [3*n + 0, 3*n + 1, 3*n + 2] # x,y,z DOFs

for dof in fixed_dofs:
    # zero out row and column
    K_num[dof, :] = 0
    K_num[:, dof] = 0
    # set diagonal to 1
    K_num[dof, dof] = 1
    # zero force
    loads_vec[dof] = 0

plot_matrix_heatmap(K_num)
```



```
# Solve for displacements: K u = f
displacements = np.linalg.solve(K_num, loads_vec)

# Add displacements to the pyvista grid for visualization
mesh.point_data['Displacements'] = displacements.reshape(-1, 3)
mesh.save('deformed_mesh.vtk')
```

```
np.linalg.norm(displacements.reshape(-1,3)).max()
```

```
np.float64(0.014771265101577533)
```