Using JAX-FEM to Model AVIC Gel Stiffness Modification

Arushi Sadam, Gabriel Peery, and Michael Sacks

August 9, 2024

Primary Study Goals

1. Learn JAXFEM basics for hyperelastic problems.
2. Run test problems and verify.
3. Develop forward model of 'gel' problem (cuboidal region with cavity withe prescribed displacements) using real AVIC geometry.
4. Test both unmodified and prescribed α fields.
5. Check for speed and accuracy compared FENICS solutions.
6. Wrap up and document.

Additional things if time permits

1. Build an inverse model formulation for the 'gel' problem using real AVIC geometries.
2. Go for the 'cell' model to obtain stress fiber behaviors.
3. Explore implementation of JUPYTER LAB notebooks (JLNB).
4. Explore use of PYVISTA/TRAME for 3D graphics directly embedded in the JLNB.

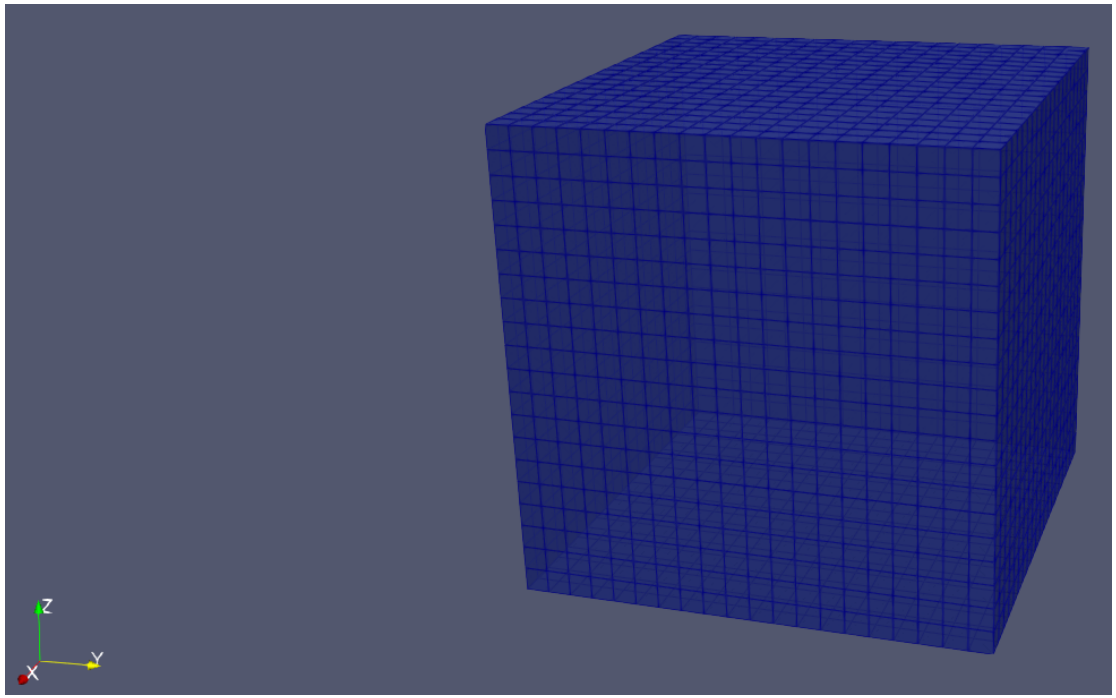
Current ToDo

1. Run basic hyperelasticity demo. ✓
2. Modify Hyperelasticity demo to instead uniaxially stretch a cube ✓
3. Setup JAX-FEM with CudaNN.
4. Determine and represent the Jacobian of the deformation gradient as a mesh grid across the deformed object ✓
5. Calculate the effective force required to stretch a beam across different stretch ratios & compare results with the ideal analytical solution ✓
6. Model a homogeneous hydrogel where a stretched sphere represents an activated cell

General tests

Twisted Cube Structure

- 1x1x1 Cube
- 20x20x20 hexahedral elements
- Back end is fixed, front end is twisted clockwise
- Note: units not provided



Twisted Demo Cube (more perspectives below)

Model Math/Boundary Conditions

Cube Surface Abbreviations:

- Ω_0 represents the cube's reference position.
- $\partial\Omega_0$ is the cube's surface in the reference position.
- $\partial\Omega_{\text{fixed},0}$ is the cube's fixed surface (back face) in the reference position.
- $\partial\Omega_{\text{twisted},0}$ is the cube's twisted surface (front face) in the reference position.

Isochoric Neohookean Model:

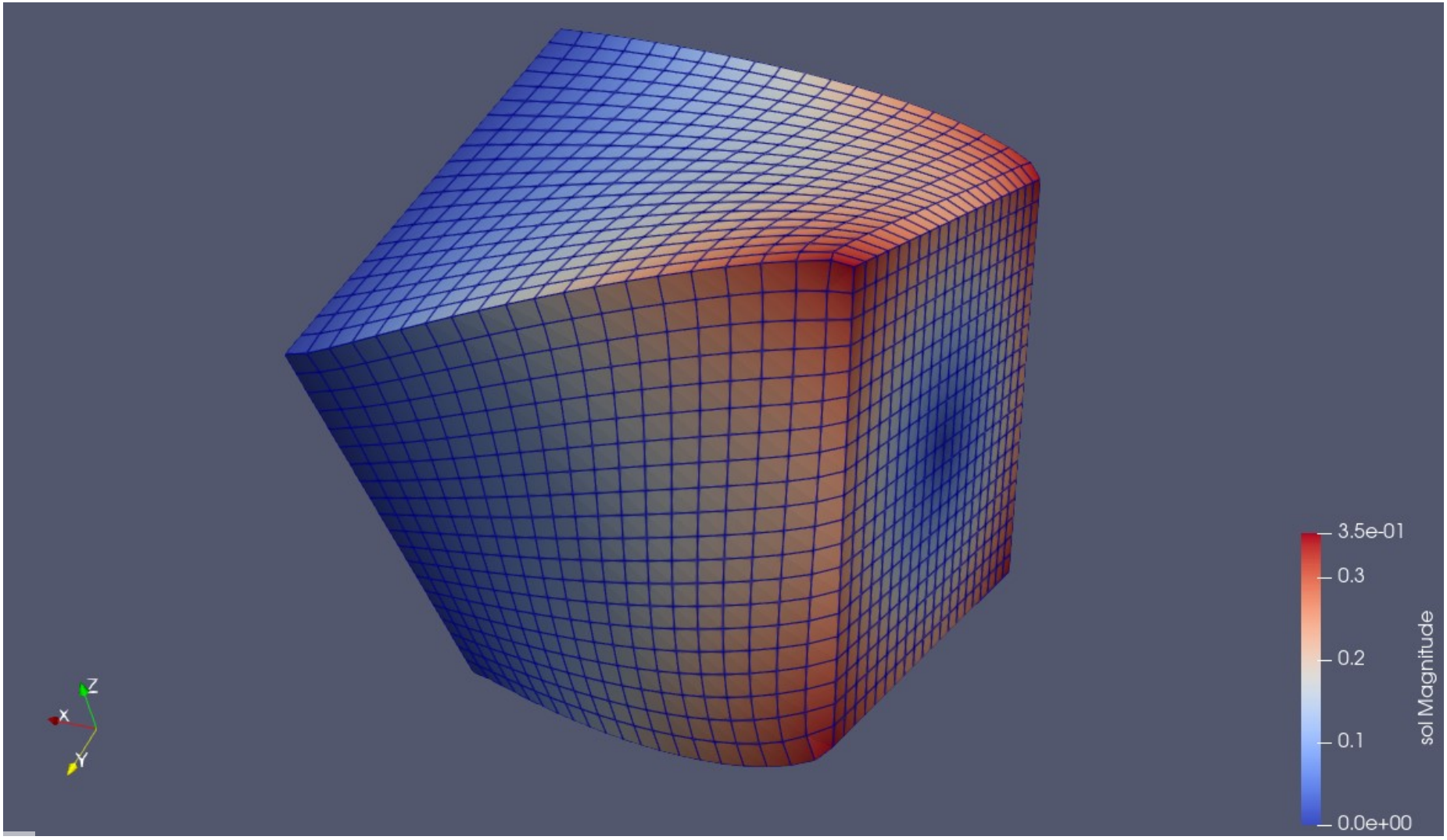
- $$\psi = \frac{\mu}{2}\left(J^{-\frac{2}{3}}I_1 - 3\right) + \frac{\kappa}{2}(J-1)^2$$

  where μ and κ are the Lamé parameters, $I_1 = \text{tr}(C)$ is the 1st invariant of the Right Cauchy-Green deformation tensor, and $J = \det(F)$ where F is the deformation tensor.
- $\mu \approx 0.3846$, $\kappa \approx 0.8333$

Boundary Conditions of Twisted Cube:

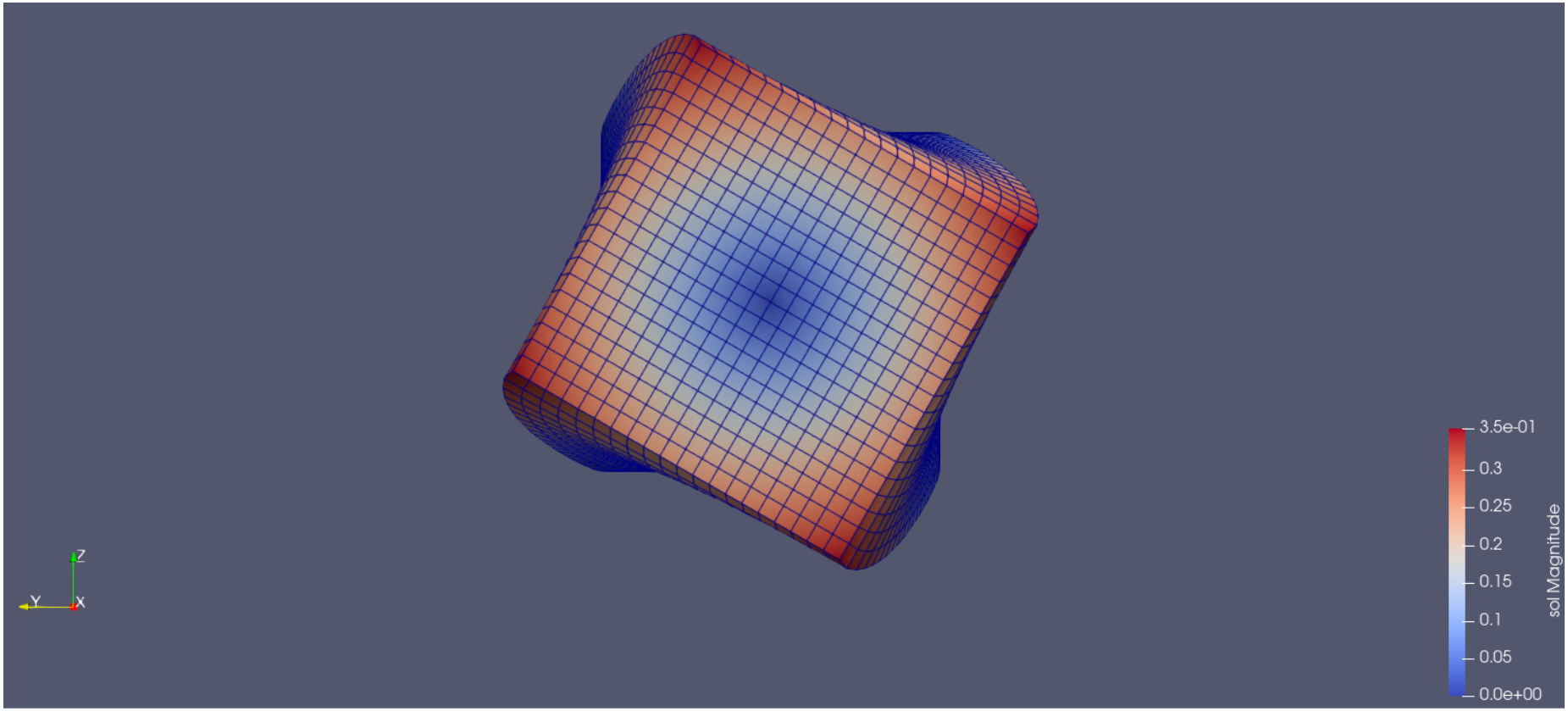
- $\mathbf{u}(x) = 0,\ x \in \partial\Omega_{\text{fixed},0}$
- $$\mathbf{u}(x) = \begin{bmatrix} u_1(x) \\ u_2(x) \\ u_3(x) \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{\frac{1}{2}+(y-0.5)\cos(\frac{\pi}{3})-(z-0.5)\sin(\frac{\pi}{3})-y}{2} \\ \frac{\frac{1}{2}+(y-0.5)\sin(\frac{\pi}{3})+(z-0.5)\cos(\frac{\pi}{3})-z}{2} \end{bmatrix},\ x \in \partial\Omega_{\text{twisted},0}$$

Results (displaced twisted cube, side view)



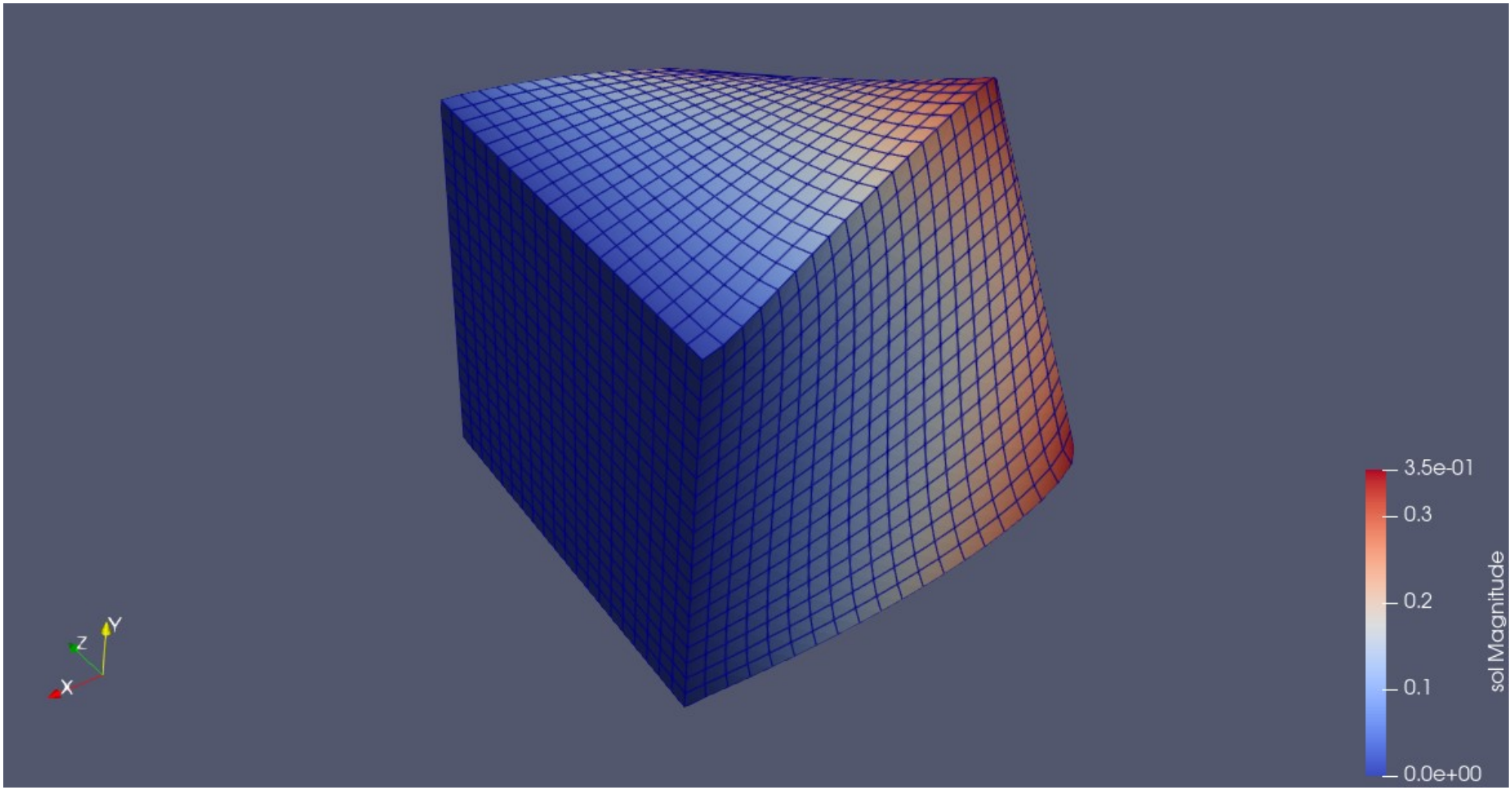
Side View

Results (displaced twisted cube, front view)



Front View

Results (displaced twisted cube, back view)

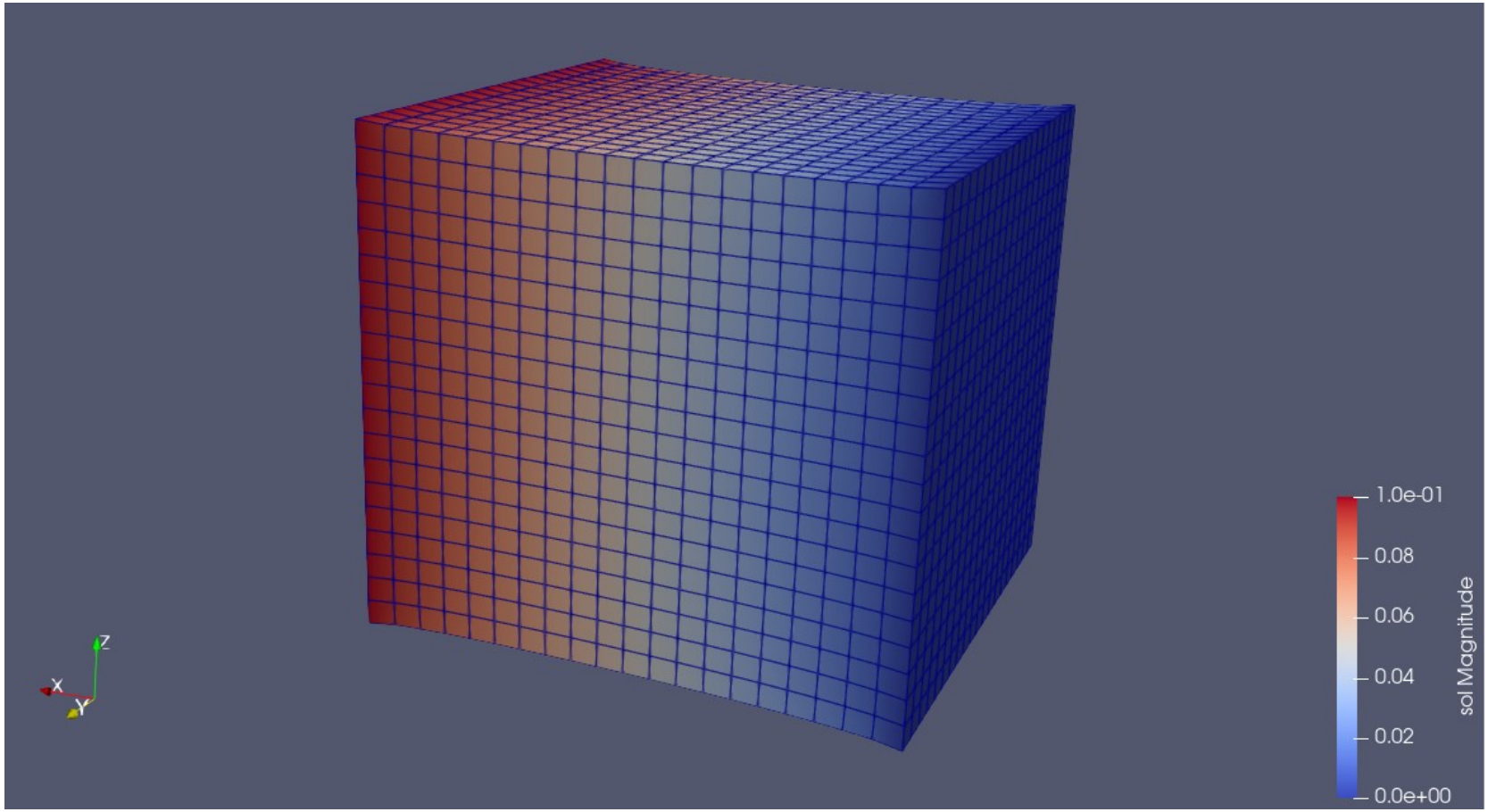


Back View

2) Stretched Cube (Modified Demo)

Model Math/Boundary Conditions

- Same cube structure & Isochoric Neohookean Model as in the JAX-FEM hyperelasticity demo.
- One side is fixed, opposite side is extended along x axis (by 10%)



Displaced Stretched Cube

Boundary Conditions cont.

- $\mathbf{u}(x) = \begin{bmatrix} u_1(x) \\ u_2(x) \\ u_3(x) \end{bmatrix} = \begin{bmatrix} 0.1 * x \\ 0 \\ 0 \end{bmatrix}$, $x \in \partial\Omega_{\text{pulled},0}$ ($\partial\Omega_{\text{pulled},0}$ is the stretched cube face)
- $\mathbf{u}(x) = \begin{bmatrix} u_1(x) \\ u_2(x) \\ u_3(x) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$, $x \in \partial\Omega_{\text{fixed},0}$

Implementation: Boundary Conditions

For comparison, the twisted cube boundary conditions were:

```
def dirichlet_val_x2(point):
    return (0.5 + (point[1] - 0.5) * np.cos(np.pi / 3.) -
            (point[2] - 0.5) * np.sin(np.pi / 3.) - point[1])/2.
def dirichlet_val_x3(point):
    return (0.5 + (point[1] - 0.5) * np.sin(np.pi / 3.) +
            (point[2] - 0.5) * np.cos(np.pi / 3.) - point[2]) / 2.
dirichlet_bc_info = [[left] * 3 + [right] * 3, [0, 1, 2] * 2,
                     [zero_dirichlet_val, dirichlet_val_x2,
                     dirichlet_val_x3] + [zero_dirichlet_val] * 3]
```

Note specifically the last three lines of the `dirichlet_bc_info`:

- `[left] * 3 + [right] * 3` specifies that the Dirichlet boundary conditions are applied to the left and right faces of the cube, for all three spatial dimensions (x, y, z).
- `[0, 1, 2] * 2` indicates which components of the displacement vector $\mathbf{u}(x)$ are being constrained. Here, `0, 1, 2` correspond to the x, y, and z components respectively. This pattern `[0, 1, 2]` is repeated twice to match the left and right faces.
- `[zero_dirichlet_val dirichlet_val_x2, dirichlet_val_x3] + [zero_dirichlet_val] * 3`, specifies the boundary value functions to be applied. For the left face, `zero_dirichlet_val` is applied to the x component, `dirichlet_val_x2` to the y component, and `dirichlet_val_x3` to the z component. For the right face, `zero_dirichlet_val` is applied to all three components.

Implementation: Boundary Conditions cont.

The twisted cube demo has zero change on the x component of the cell and trigonometric functions on the y and z components (for twisting in the yz plane). To stretch the cube in the x direction, simply change the boundary conditions to be a linear stretch for the x component (`pull_dirichlet_val_x` function) and 0 for the y and z components. The stretched code boundary conditions are thus:

```
def zero_dirichlet_val(point):
    return 0.
def pull_dirichlet_val_x(point):
    return .1*point[0]
# Apply zero Dirichlet boundary values on the left side
# and pulling on the right side along the x-axis.
dirichlet_bc_info = [[left] * 3 + [right] * 3,
                     [zero_dirichlet_val, zero_dirichlet_val,
                     zero_dirichlet_val] +
                     [pull_dirichlet_val_x, zero_dirichlet_val,
                     zero_dirichlet_val]]
```

3) Representing J

Motivation & Implementation

Motivation

- J is the volume ratio or Jacobian determinant of the deformation gradient F. $J = \det(F)$.
- $J(X,t) > 0$ for all X and t, where X is the reference position and t is time, due to the impenetrability of matter (no negative volume) (Holzapfel, section 2.4, p. 74).

Implementation

- Same Isochoric Neohookean model as before. J values are calculated for twisted cube and stretched cubes shown above.
- Create function `get_j(F)`: takes deformation gradient matrix (F) as input and calculates/returns the determinant J.
- Create function `get_f(u_grad)`: takes displacement/solution gradient matrix and outputs F, per $F = I + \nabla U_{\text{grad}}$.

```
def get_j(F):
    return np.linalg.det(F)

def get_f(u_grad):
    I = np.identity(u_grad.shape[0])
    F = u_grad + I
    return F
```

Implementation cont.

- Create a new J matrix (`j_mat`) with dimensions number of cells (`num_cells`, 8000 for cube example) by number of nodes per cell (`num_points`, 8 for cube example). Loop through each cell's vertices, and update the `j_mat` with the corresponding J value for the specific cell at a specific node.

```
ug_s = u_grads.shape
j_mat = np.zeros(ug_s[:2])
num_cells = ug_s[0] # 8000
num_points = ug_s[1]
for i in range(num_cells):
    for j in range(num_points):
        j_mat[i, j] = get_j(get_f(u_grads[i, j]))
```

- Paraview mesh only represents global nodal values; our current `j_mat` has local J values (values for each node for each cell). Neighboring cells share common nodes; instead of a `num_cells` by `num_nodes` `j_mat`, we need a 1D array of unique vertices with shape `num_vertices` (21*21*21 = 9261 for the cube example) by 1.
- `cells` is a matrix that contains the node indices each cell is connected to, while `points` defines the spatial locations of each node. In `generate_mesh.py`, function `cells_out` is defined to access these two matrices:

```
def cells_out():
    return cells, points
```

Implementation cont.

- Per local cell/node in `cells`, the associated global node is determined. Then, since some nodes have multiple J values (contributed by different cells), the average of the J values is assigned to the specific node. `local_j` ultimately stores the J values per global node (matrix has shape `num_vertices` by 1).

```
cells, points = cells_out()
num_repeat = np.zeros(len(points))
local_j = np.zeros(len(points))
for r in range(cells.shape[0]):
    for c in range(cells.shape[1]):
        point = points[cells[r, c]]
        ind = np.where(np.all(points == point, axis=1))
        local_j = local_j.at[ind].add(j_mat[r, c])
        num_repeat = num_repeat.at[ind].add(1)
local_j = local_j / num_repeat
```

Implementation cont.

- To allow for multiple nodal meshes to be displayed in one Paraview file, the `utils.py` file was edited to accept lists of dictionaries as mesh information.

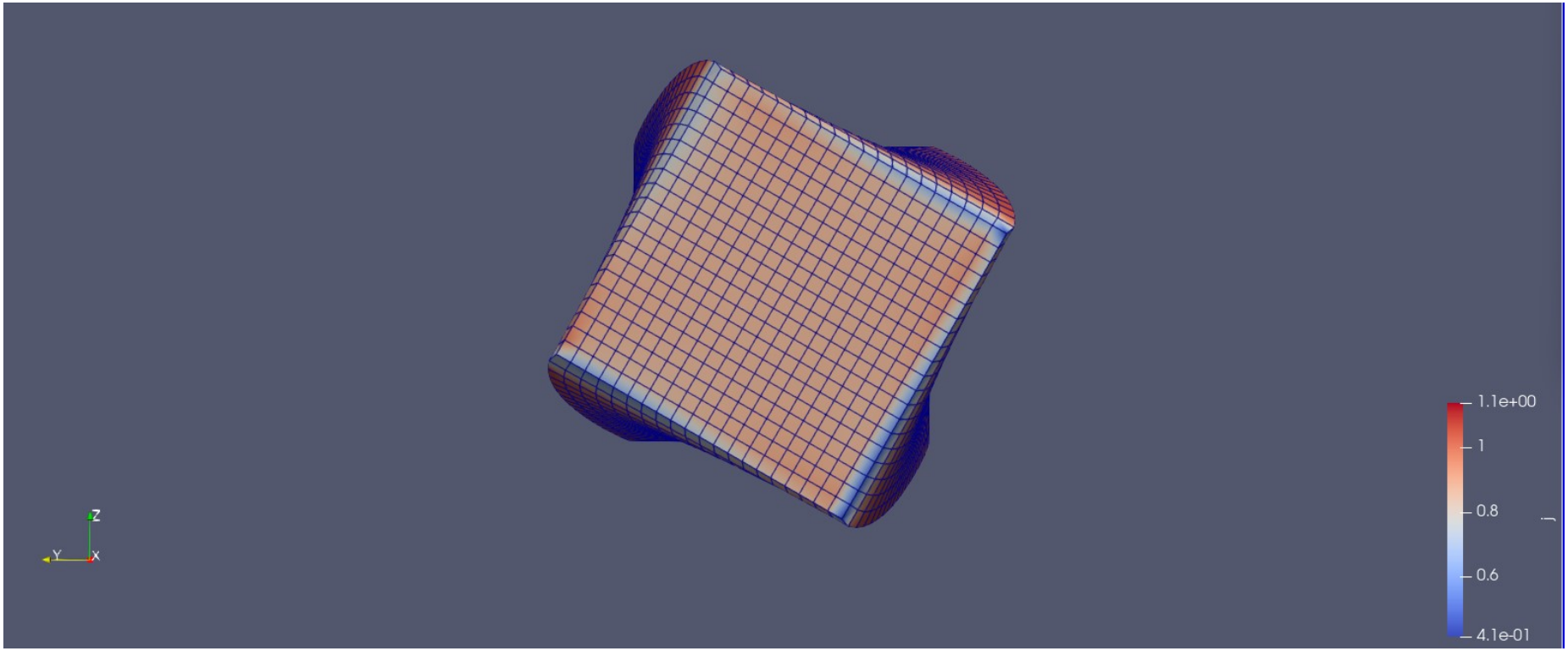
```
if point_infos is not None:
for point_info in point_infos:
    for name, data in point_info.items():
        assert len(data) == len(sol), "point data wrong shape!"
```

- The example save solution command is:

```
save_sol(problem.fes[0], sol, vtk_path,
             point_infos = [{"j":local_j}])
```

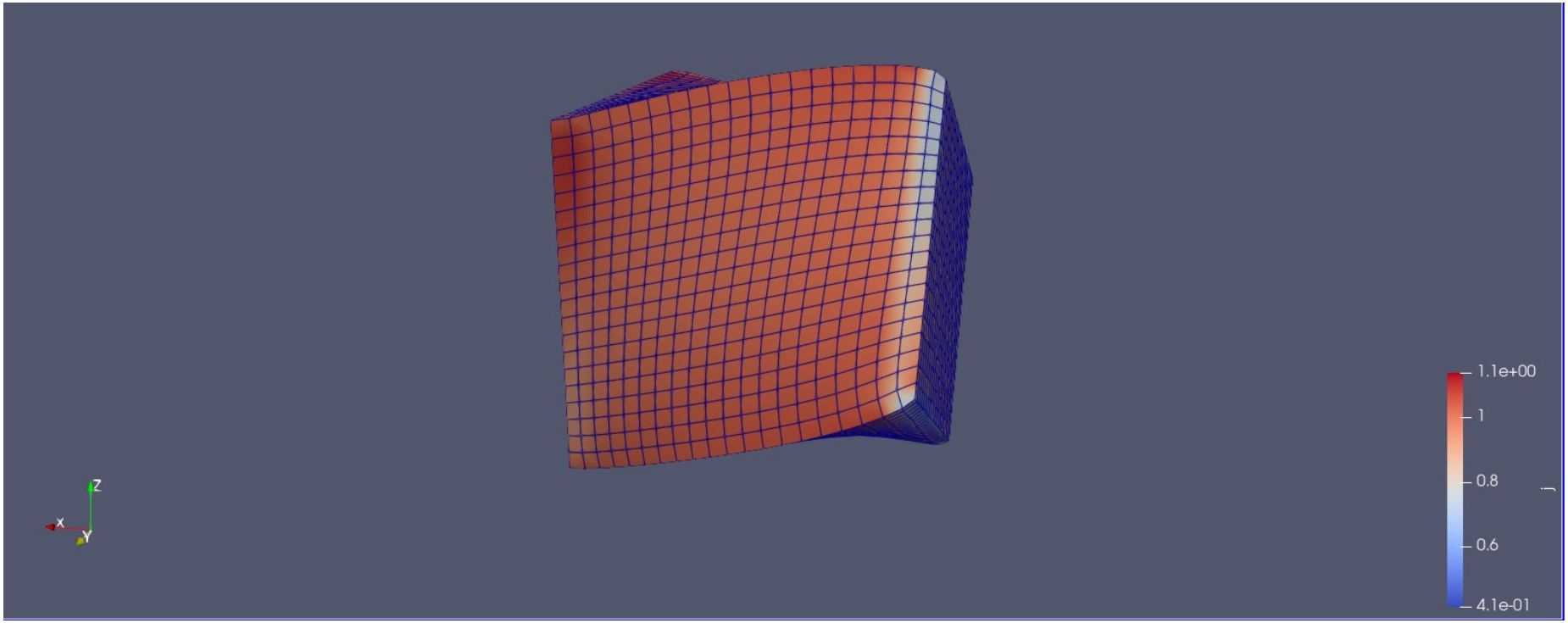
Results: Twisted Cube Front View

For the JAX-FEM Demo cube (that is twisted), the deformed cube has the following J values:



Front View

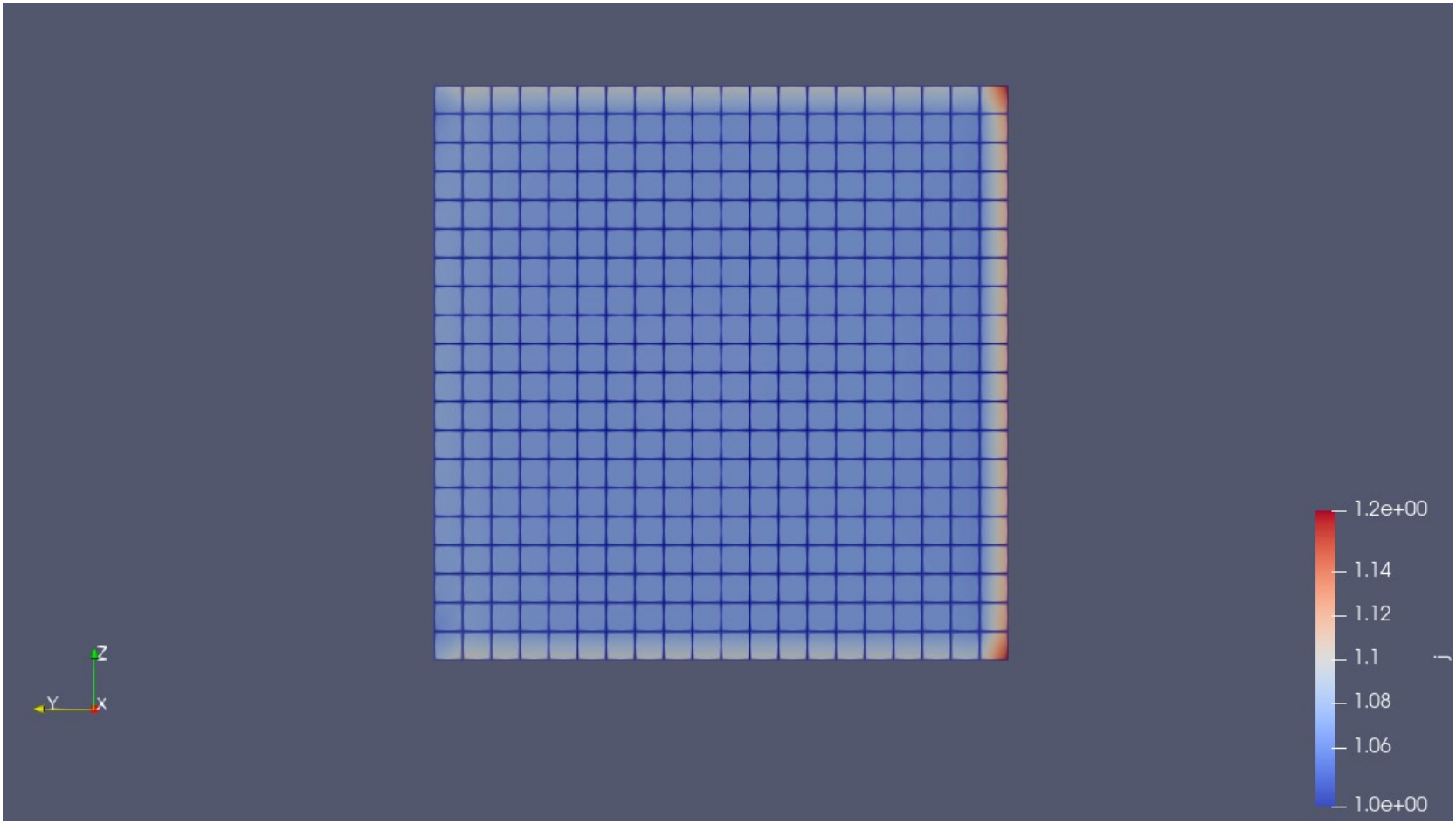
Results: Twisted Cube Side View



Side View

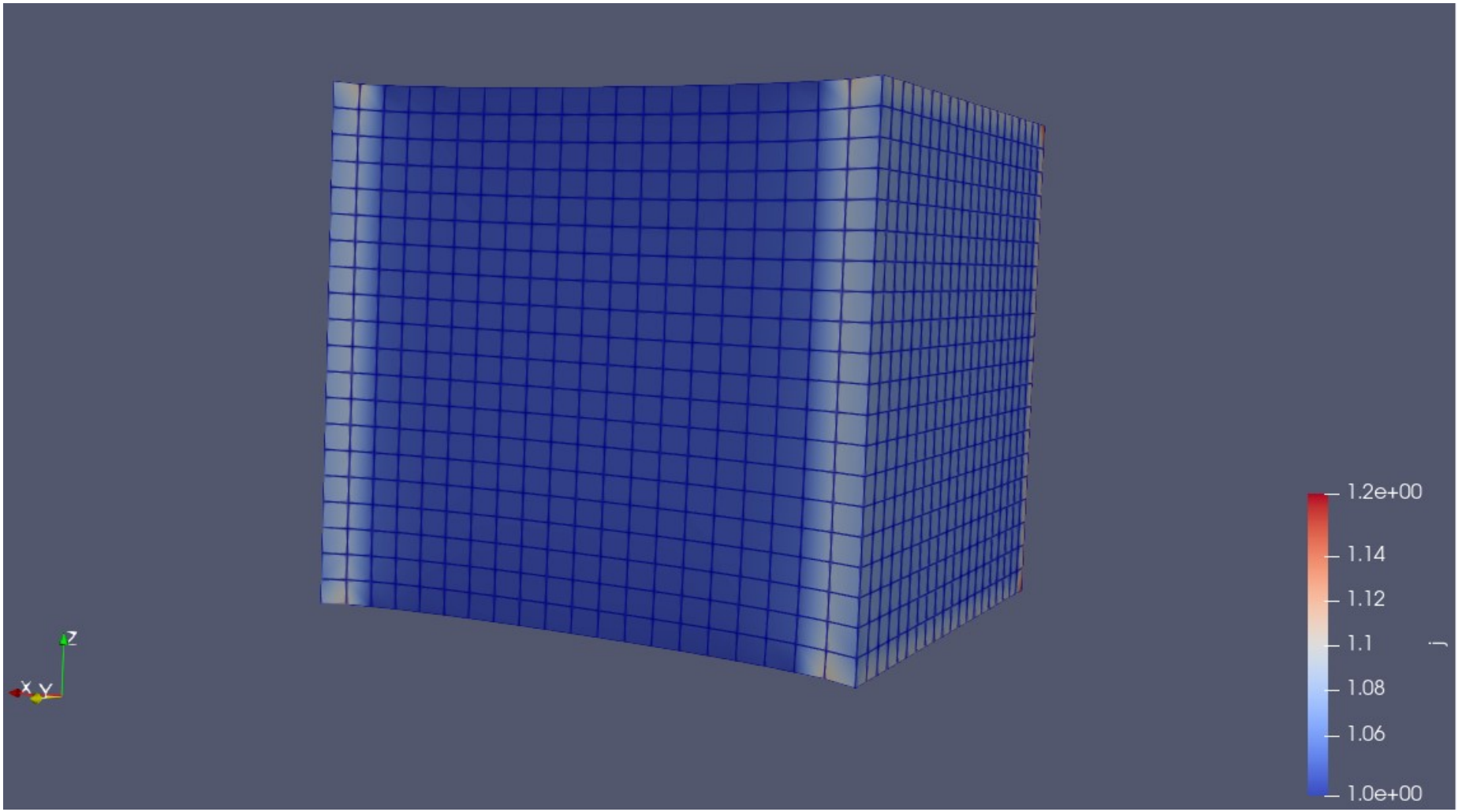
Results: Stretched Cube Front View

The stretched cube has the following J values:



Front View

Results: Stretched Cube Side View

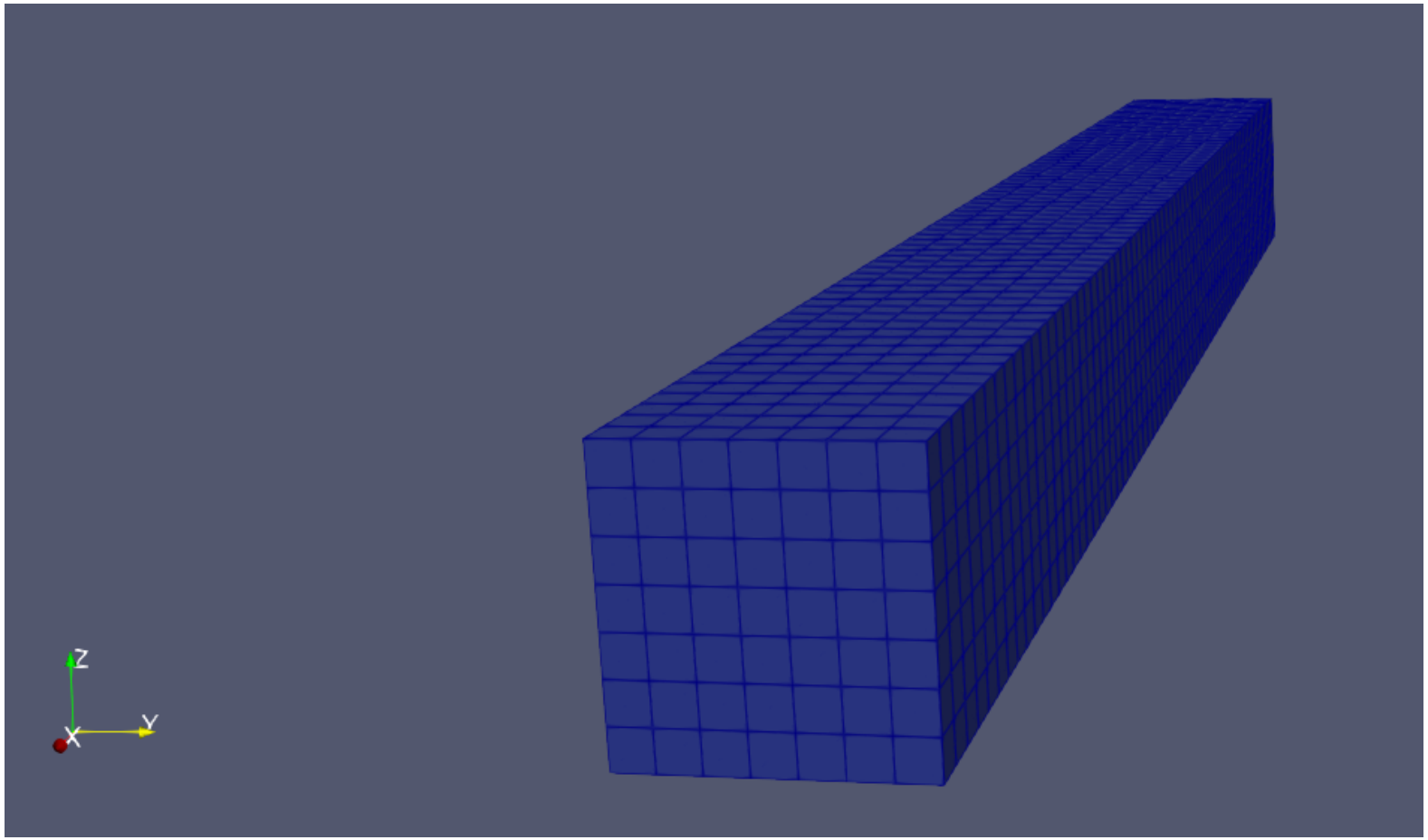


Side View

4) Calculating Effective Force to Stretch a Beam

Stretched Beam Structure

- 10x1x1 Beam
- 70x7x7 hexahedral elements
- Back end is fixed, front end is along in the x axis



Displacement

Stretched Beam Structure cont.

- Ω_0 represents the beam's reference position.
- $\partial\Omega_0$ is the beam's surface in the reference position.
- $\partial\Omega_{\text{fixed},0}$ is the beam's fixed surface in the reference position.
- $\partial\Omega_{\text{pulled},0}$ is the beam's pulled surface in the reference position.

Model Math/Boundary Conditions

- Same Isochoric Neohookean Model as in the JAX-FEM hyperelasticity demo.
- $\mathbf{u}(x) = \begin{bmatrix} u_1(x) \\ u_2(x) \\ u_3(x) \end{bmatrix} = \begin{bmatrix} (\lambda - 1) \cdot \mathbf{e}_x \\ 0 \\ 0 \end{bmatrix}$, $x \in \partial\Omega_{\text{pulled},0}$, $\lambda \in [1, 1.3]$, $\lambda = l/L$ (l = current beam length, L = reference beam length)
- $\mathbf{u}(x) = \begin{bmatrix} u_1(x) \\ u_2(x) \\ u_3(x) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$, $x \in \partial\Omega_{\text{fixed},0}$

Theoretical Solution

- P is the 1st Piola-Kirchhoff stress tensor.
- $\mathbf{n}$ is the outward unit normal vector.
- F_{eff} is the effective force in the x-direction of the stretched surface (scalar value).
- $\mathbf{P} = \frac{\partial \psi}{\partial \mathbf{F}}$.
- $F_{\text{eff}} = \int_{\partial \Omega_{\text{stretched},0}} (\mathbf{P} \cdot \mathbf{n}) \cdot \mathbf{n} \, d\partial\Omega_{\text{stretched}}$.
- $\mathbf{n} = \mathbf{e}_x$ on $\partial \Omega_{\text{stretched},0}$.
- A_0 is the area of the stretched surface.

σ_{11} is 1,1 component of Cauchy Stress tensor

λ is stretch

$$\sigma_{11} = 2C_1 \left(\lambda^2 - \frac{1}{\lambda} \right)$$

Deriving the effective force F_{eff} we compute...
We assume, in line with Wikipedia's derivation,

$$[\mathbf{F}] = \begin{bmatrix} \lambda & 0 & 0 \\ 0 & \frac{1}{\sqrt{\lambda}} & 0 \\ 0 & 0 & \frac{1}{\sqrt{\lambda}} \end{bmatrix}$$

$$\implies J = 1$$

$$\implies \mathbf{P} = J\sigma\mathbf{F}^{-T} = \sigma\mathbf{F}^{-T}$$

$$\implies P_{11} = \frac{\sigma_{11}}{\lambda} = 2C_1 \left(\lambda - \frac{1}{\lambda^2} \right)$$

Theoretical Solution cont.

Assuming P_{11} is uniform on $\partial\Omega_{\text{stretched},0}$:

$$
\begin{aligned}
F_{\text{eff}} &= \int_{\partial\Omega_{\text{stretched},0}} (\mathbf{P}\cdot\mathbf{n})\cdot\mathbf{n}\, d\partial\Omega_{\text{stretched}} \\
&= \int_{\partial\Omega_{\text{stretched},0}} (\mathbf{P}\cdot\mathbf{e}_x)\cdot\mathbf{e}_x\, d\partial\Omega_{\text{stretched}} \\
&= \int_{\partial\Omega_{\text{stretched},0}} P_{11}\, d\partial\Omega_{\text{stretched},0} \\
&= P_{11}\cdot A_0 \\
&= \mu\cdot A_0\left(\lambda - \frac{1}{\lambda^2}\right)
\end{aligned}
$$

Observed Solution (via JAX-FEM)

Since $F_{\text{eff}} = \int_{\partial\Omega_{\text{stretched},0}} P_{11}\, d\partial\Omega_{\text{stretched},0}$, F_{eff} on the stretched face can be determined by:

1. Determining the 1st Piola-Kirchhoff Tensor ($\mathbf{P}$) across the (global) nodes (in all 3 dimensions)
2. Selecting and summing the nodes specific to the stretched face
3. Calculating the L_2 norm or magnitude of the resulting face-specific $\mathbf{P}$ tensor.

Determining P

To calculate $\mathbf{P}$ on each node, a strategy similar to calculating the `j_mat` (p. 14) is used. The 1st Piola-Kirchhoff Tensor is calculated and returned via the function `first_Pk_stress(u_grad)` (defined in the `HyperElasticity` class). For each cell's node, a local $\mathbf{P}$ matrix is formed (`P_mat`):

```
ug_s = u_grads.shape
P_mat = np.zeros(ug_s) # num_cells, num_nodes_per_cell,
                                    num_vectors, num_dim
num_cells = ug_s[0] # num_cells
num_points = ug_s[1] # num_nodes_per_cell
for i in range(num_cells):
    for j in range(num_points):
        P_mat = P_mat.at[i,j,:,:].set(first_PK_stress(u_grads[i,j]))
```

Determining P cont.

To convert the local P_mat (with shape num_cells=70*7*7=3430 x num_points = 8 x num_vectors=3 x num_dim=3)) to a global matrix (with shape num_nodes=71*8*8=4544 x num_vectors=3 x num_dim=3), again, a similar strategy as the local-to-global conversion for the J matrix is used:

```
cells, points = cells_out()
num_repeat_p = np.zeros(len(points))
local_p = np.zeros((len(points),3,3)) # num points x num vectors
                                      x num dim
for r in range(cells.shape[0]):
    for c in range(cells.shape[1]):
        point = points[cells[r,c]]
        ind = np.where(np.all(points == point, axis = 1))
        local_p = local_p.at[ind].add(P_mat[r,c,:,:])
        num_repeat_p = num_repeat_p.at[ind].add(1)
local_p = local_p/num_repeat_p[:, np.newaxis, np.newaxis]
```

Determining P cont.

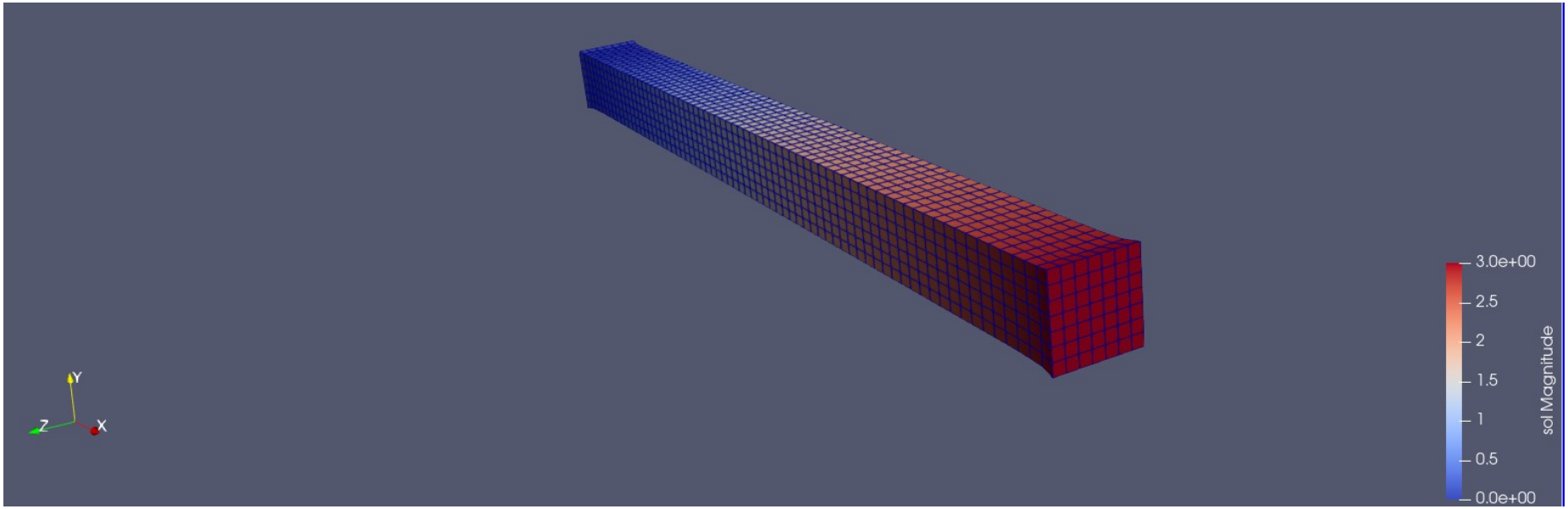
To select the end points of the **P** matrix, the indices of points that have x value of Lx (the max beam length in the x direction) are used. Then, only the x direction vectors are selected.

```
end_ind = np.argwhere(points[:,0] == Lx)
all_p_face = local_p[end_ind,:,0]
all_p_facex = all_p_face[:,0]
```

Finally, since the stretched face area (A_0) is 1 and the force vectors point almost entirely in the direction of stretch **e_x**, the L2 norm of the averaged x vectors is taken to determine the final net magnitude of effective force.

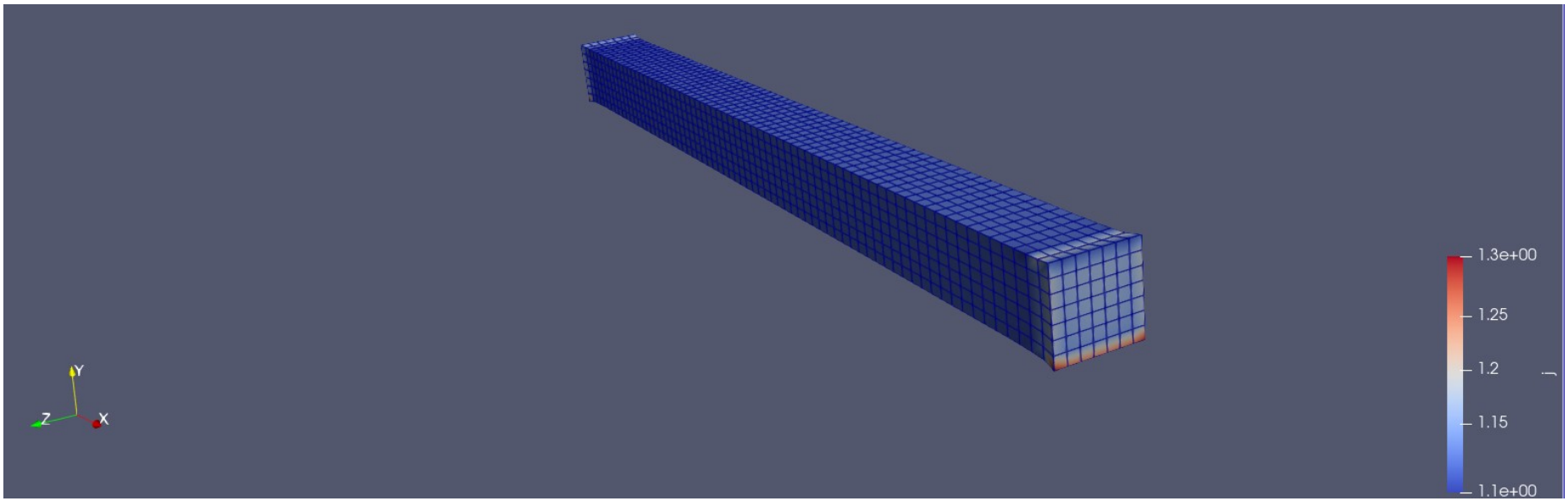
```
xyz = np.mean(all_p_facex, axis = 0)
F_mag = np.linalg.norm(xyz)]
```

Results: Beam Displacement



Displacement

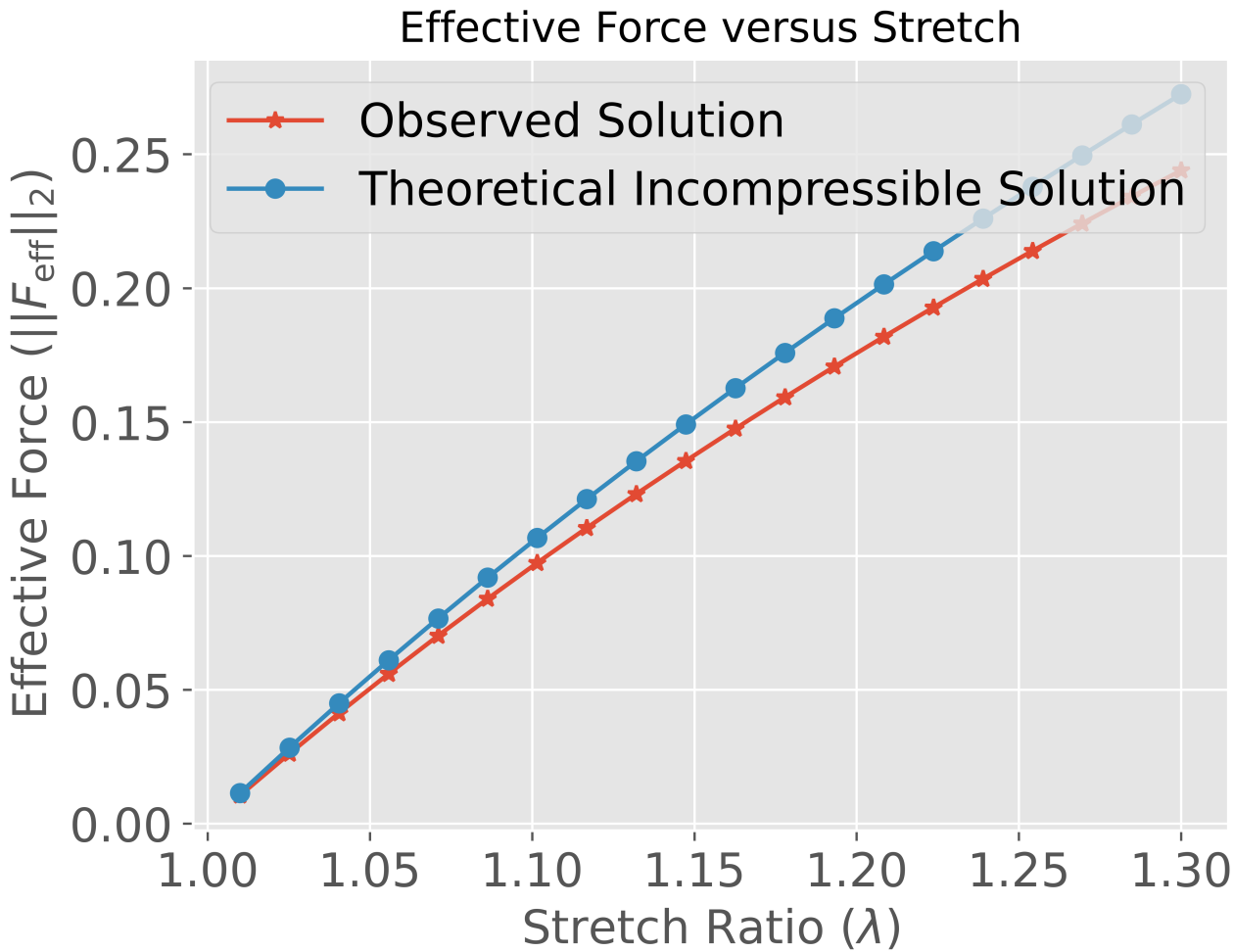
Results: Beam J values



J

Results: Force vs Stretch

The effective force (magnitude) versus different stretch values is:

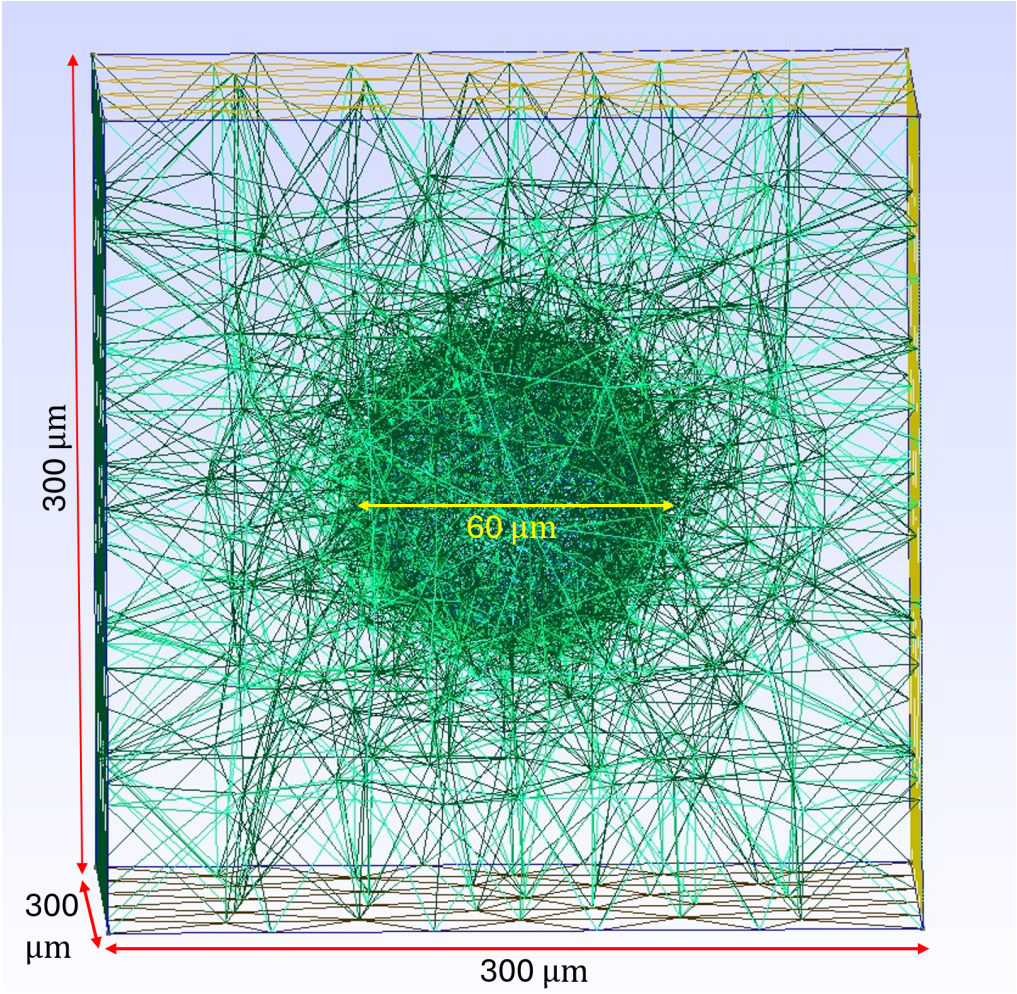


RMSE $\approx$ 0.016740 (for 20 values).

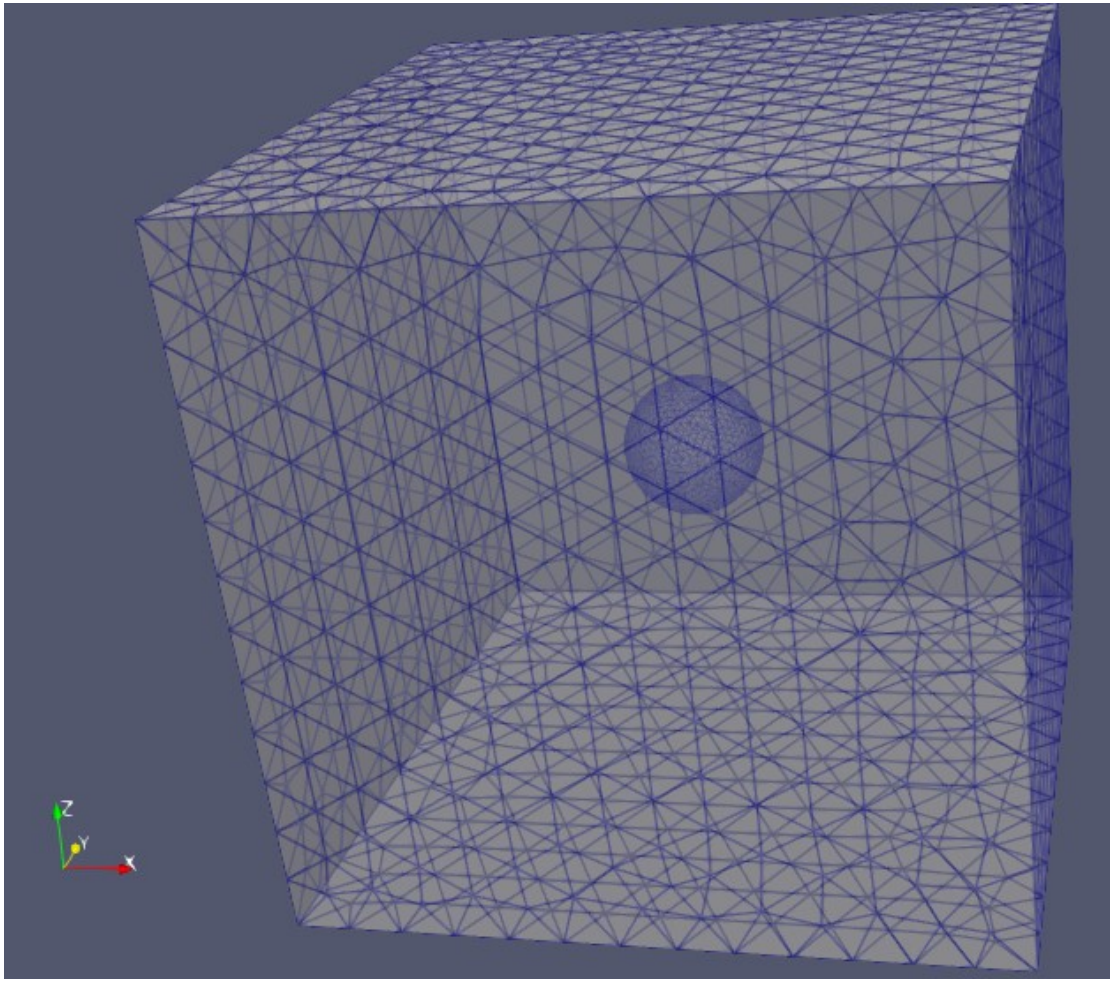
5) Model Hydrogel with Sphere as Cell

Gel/Cell Structure

1. 300 x 300 x 300 μm cube (hydrogel) (30 μm element spacing)
2. 30 μm radius sphere (cell) (1 μm element spacing) at gel center
3. 30,974 tetrahedral elements



Initial & Boundary Conditions



- Gel surface (cube exterior) is held in place
- Cell (sphere) is stretched along the x axis (1.1x) while maintaining volume

Model Math/Boundary Conditions

Surface Abbreviations:

- $\Omega_{c,0}$ represents the cube's/gel's reference position.
- $\partial\Omega_{\text{exterior},0}$ is the cube's exterior surface (all 6 sides of cube) in the reference position.
- $\partial\Omega_{\text{interior},0}$ is the cube's interior surface (sphere) in the reference position.

Strain Energy Equation/Model:

- $$\psi = C_1\left(I_1 - 3 - 2\ln(J)\right) + D_1\left(\ln(J)\right)^2$$

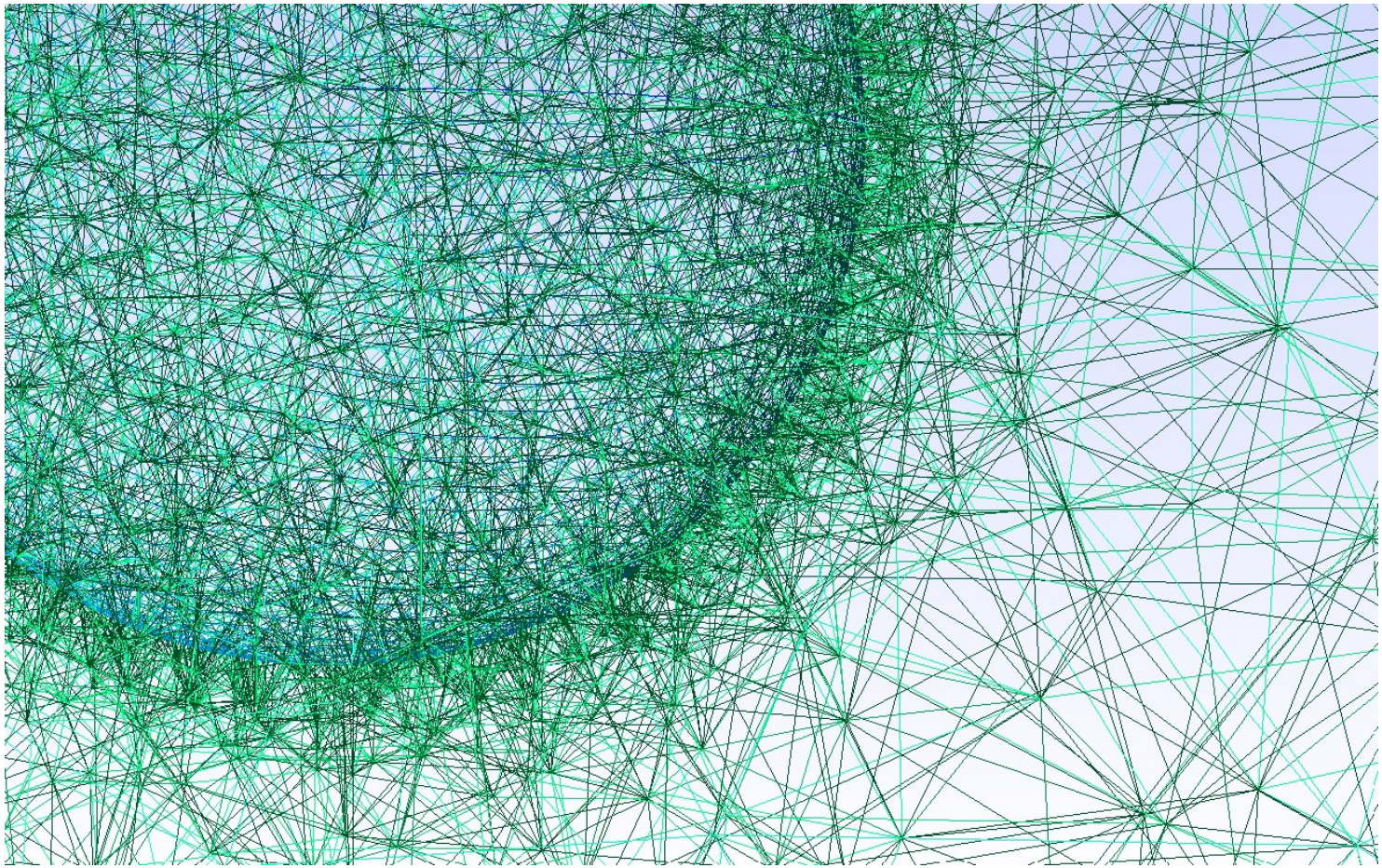
  where C_1 and D_1 are the constants, $I_1 = \text{tr}(C)$ is the 1st invariant of the Right Cauchy-Green deformation tensor, and $J = \det(F)$ where F is the deformation tensor.
- $C_1 = 50Pa$, $D_1/C_1 = 1000$

Boundary Conditions of Gel:

- $\mathbf{u}(x) = 0,\ x \in \partial\Omega_{\text{exterior},0}$
- $\mathbf{u}(x) = \begin{bmatrix} u_1(x) \\ u_2(x) \\ u_3(x) \end{bmatrix} = \begin{bmatrix} .1x \\ (\frac{1}{\sqrt{1.1}} - 1)y \\ (\frac{1}{\sqrt{1.1}} - 1)z \end{bmatrix},\ x \in \partial\Omega_{\text{interior},0}$

Implementation: Generate Mesh

Mesh created via Gmsh and OpenCASCADE. A zoomed in section of the mesh:



Implementation: Read-in Mesh

The JAX-FEM demos create meshes based on the problem; however, in this case, we read in the created mesh (from the previous slide) by creating a new mesh in `generate_mesh.py`.

The function `read_in_mesh(msh_file, cell_type)` is defined as:

```
def read_in_mesh(msh_file, cell_type):
    # msh_file is string of mesh file, cell_type is TET4 (in this example)
    mesh = meshio.read(msh_file)
    global cells, points
    points = mesh.points # (num_total_nodes, dim)
    cells = mesh.cells_dict[cell_type] # (num_cells, num_nodes)
    out_mesh = meshio.Mesh(points=points, cells={cell_type: cells})
    return out_mesh
```

In the main script (that states the problem), to read in the mesh, use:

```
meshio_mesh = read_in_mesh("sphere.msh", cell_type)
mesh = Mesh(meshio_mesh.points, meshio_mesh.cells_dict[cell_type])
```

Implementation: Selecting Boundaries

$\partial\Omega_{\text{exterior},0}$ is defined as all sides with x, y or z values equal to $\frac{\text{cube side length}}{2}$ (gel cube is centered at origin):

```
box_length = 300 # cube side length, in micro meters
tol = .01 # tolerance for position

# Define boundary locations.
def cube(point):
    return np.logical_or(
        np.isclose(np.abs(point[0]), box_length / 2., atol=1e-5),
        np.logical_or(
            np.isclose(np.abs(point[1]), box_length / 2., atol=1e-5),
            np.isclose(np.abs(point[2]), box_length / 2., atol=1e-5)
        )
    )
```

$\partial\Omega_{\text{interior},0}$ is defined as all points 30 μm away from the origin (cell is centered in gel):

```
r = 30
def sphere_surface(point):
    return np.isclose(np.linalg.norm(point) , r, atol=tol)
```

Implementation: Boundary Conditions

Assuming uniform gel stiffness, to ensure no volume change of the cell: the cell is stretched 1.1x in the x direction and squished/contracted by $\frac{1}{\sqrt{1.1}}$x in the y and z directions.

The dirichlet boundary conditions are implemented as:

```
dirichlet_bc_info = [[cube] * 3 + [sphere_surface] * 3,
                     [0, 1, 2] * 2,
                     # zero displacement on exterior:
                     [zero_dirichlet_val, zero_dirichlet_val,
                     zero_dirichlet_val]
                     # displacements on interior:
                     + [surface_x, surface_y, surface_z]]
```

Implementation: Boundary Conditions cont.

Three variables a, b, and c are defined to indicate the stretch factors in the x, y, and z dimensions:

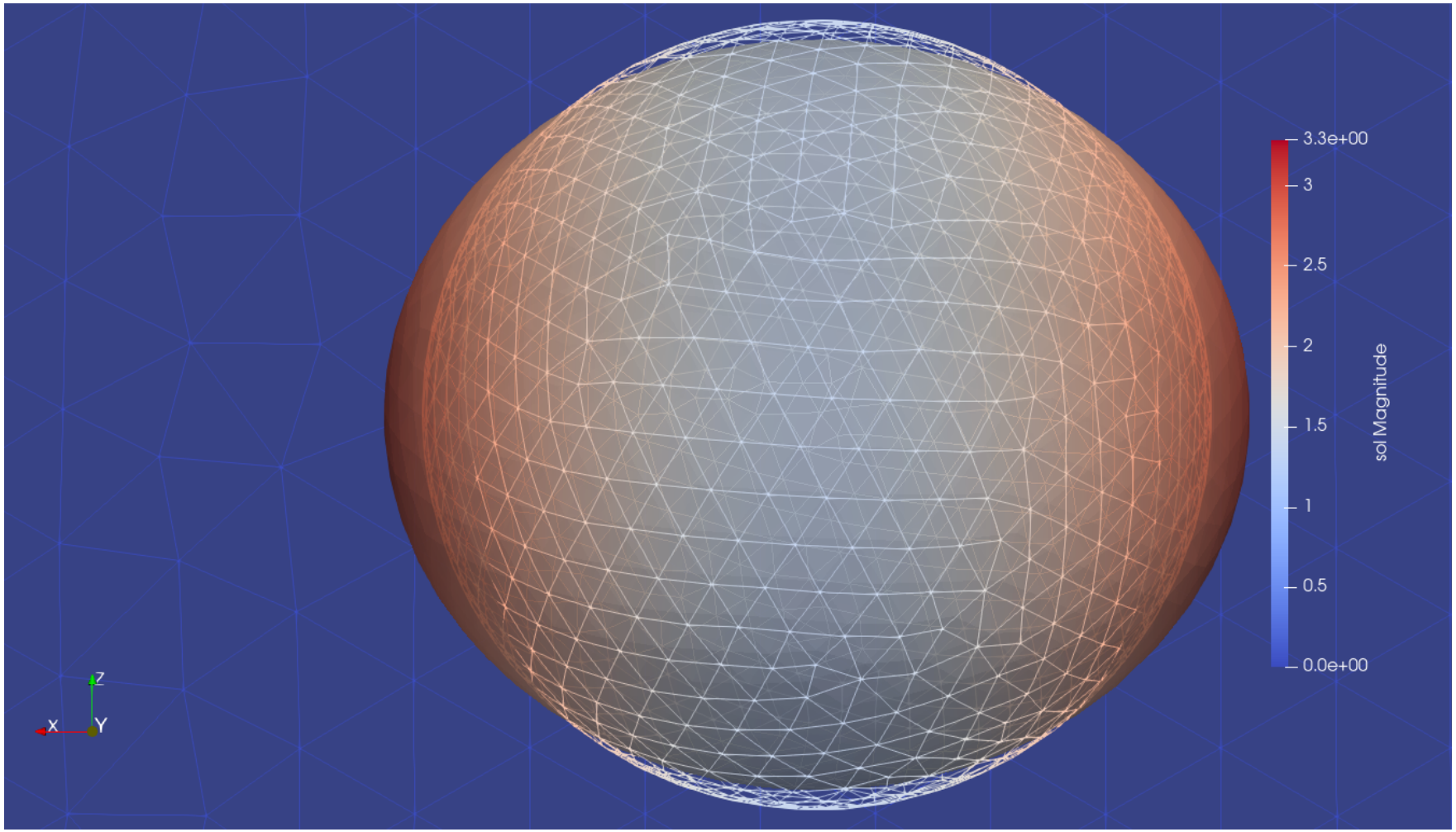
```
a = 1.1 - 1 # x
b = 1/sqrt(1.1) - 1 # y
c = 1/sqrt(1.1) - 1 # z
```

Note that `dirichlet_bc_info` applies displacements (on top of the current position), this is why a, b, and c are 1 less than the original 3 stretch factors (1.1, $\frac{1}{\sqrt{1.1}}$, $\frac{1}{\sqrt{1.1}}$ defined on previous slide). Each point is multiplied by the respective stretch factor:

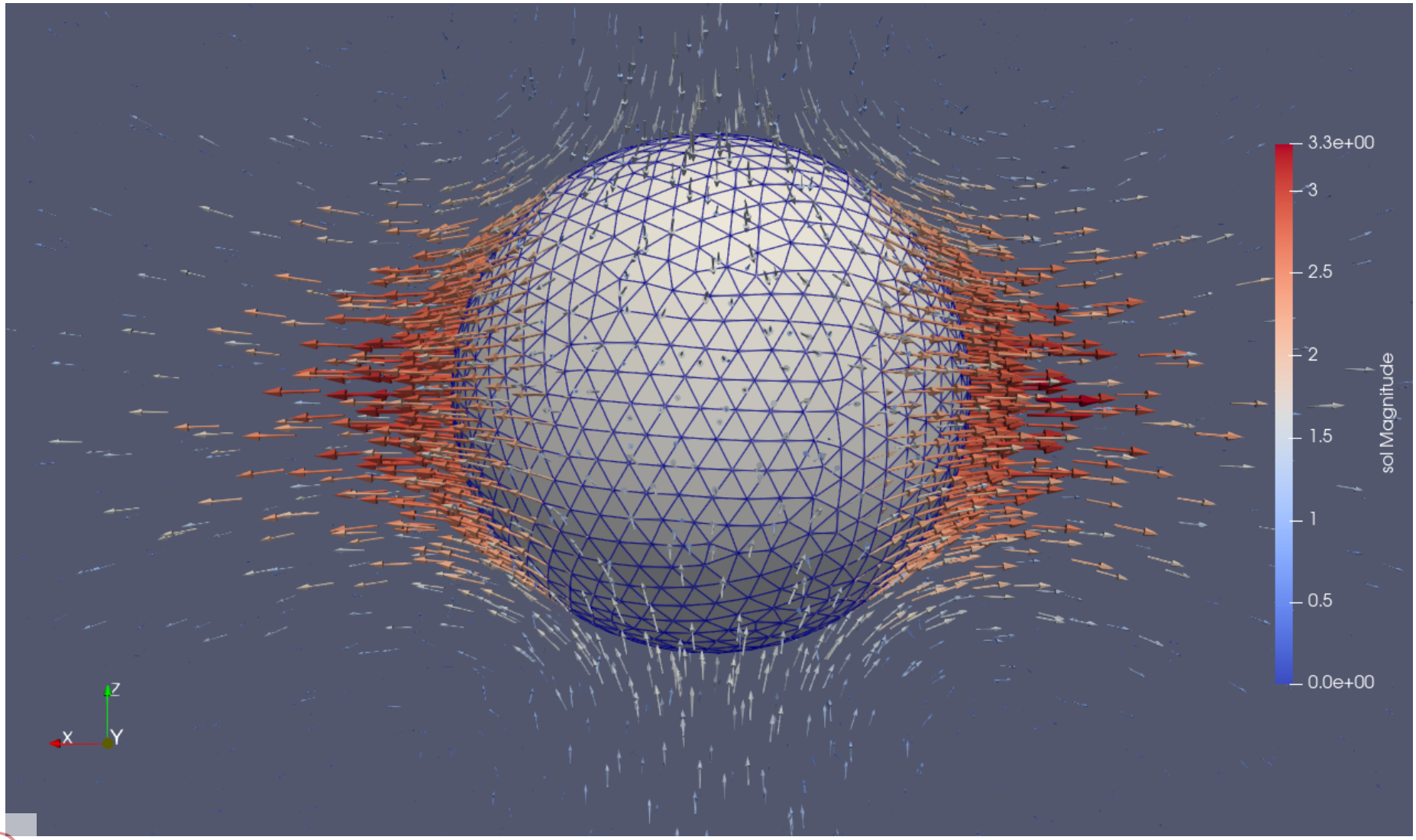
```
def surface_x(point):
    return point[0]*a

def surface_y(point):
    return point[1]*b

def surface_z(point):
    return point[2]*c
```

Results: Reference vs Deformed Configurations

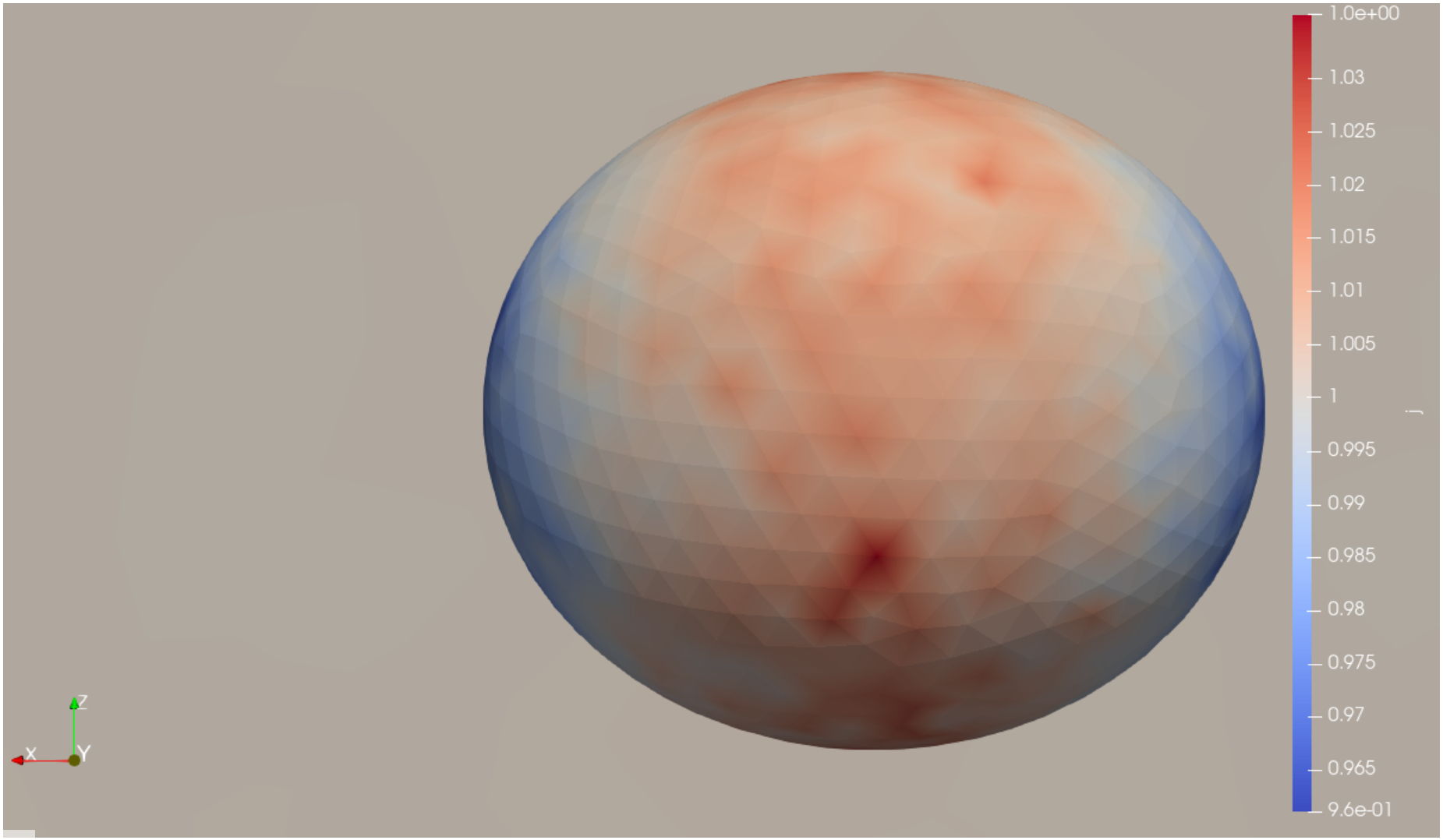
Wireframe represents reference configuration.



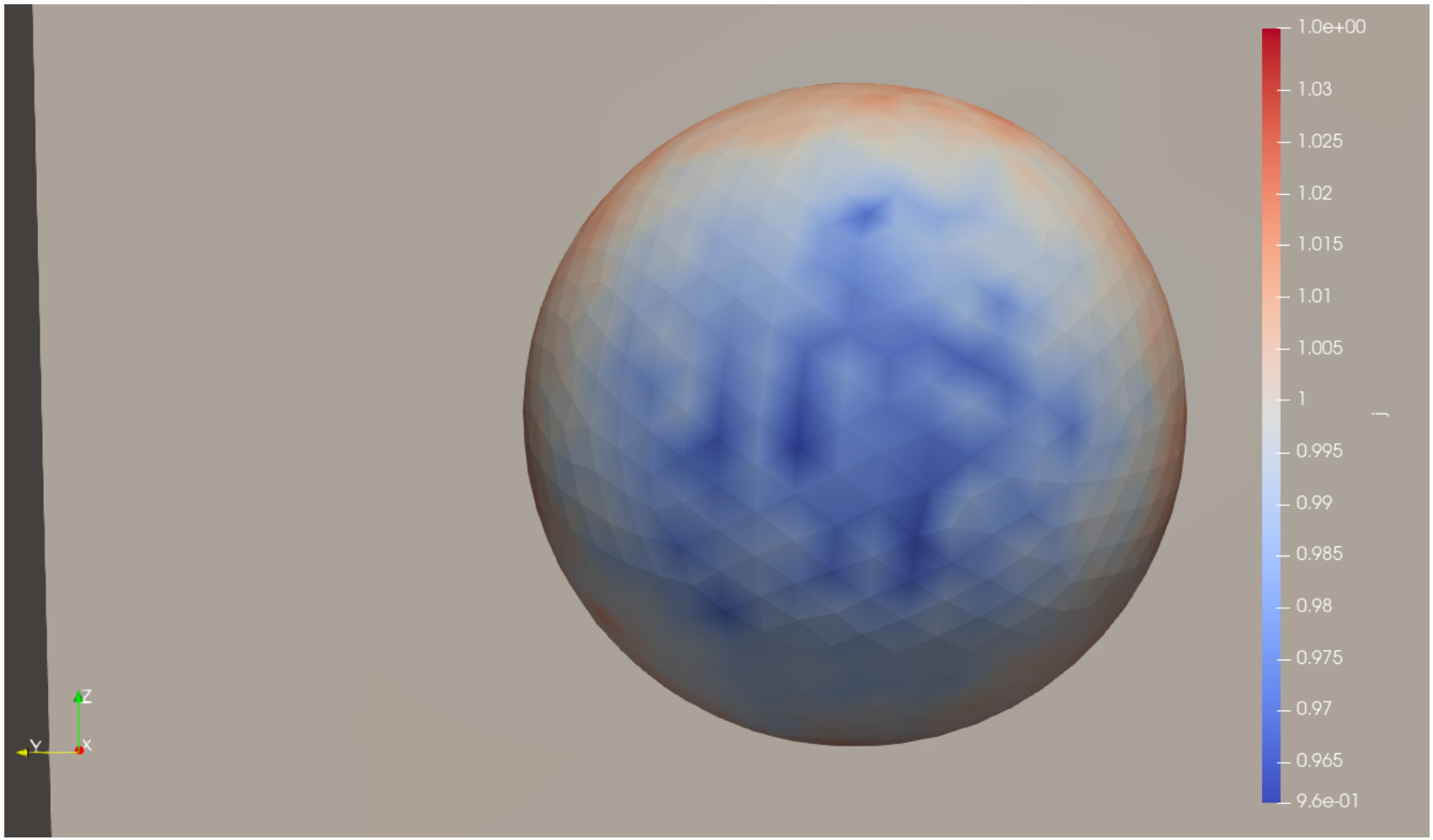
Results: Displacement Field



Results: J

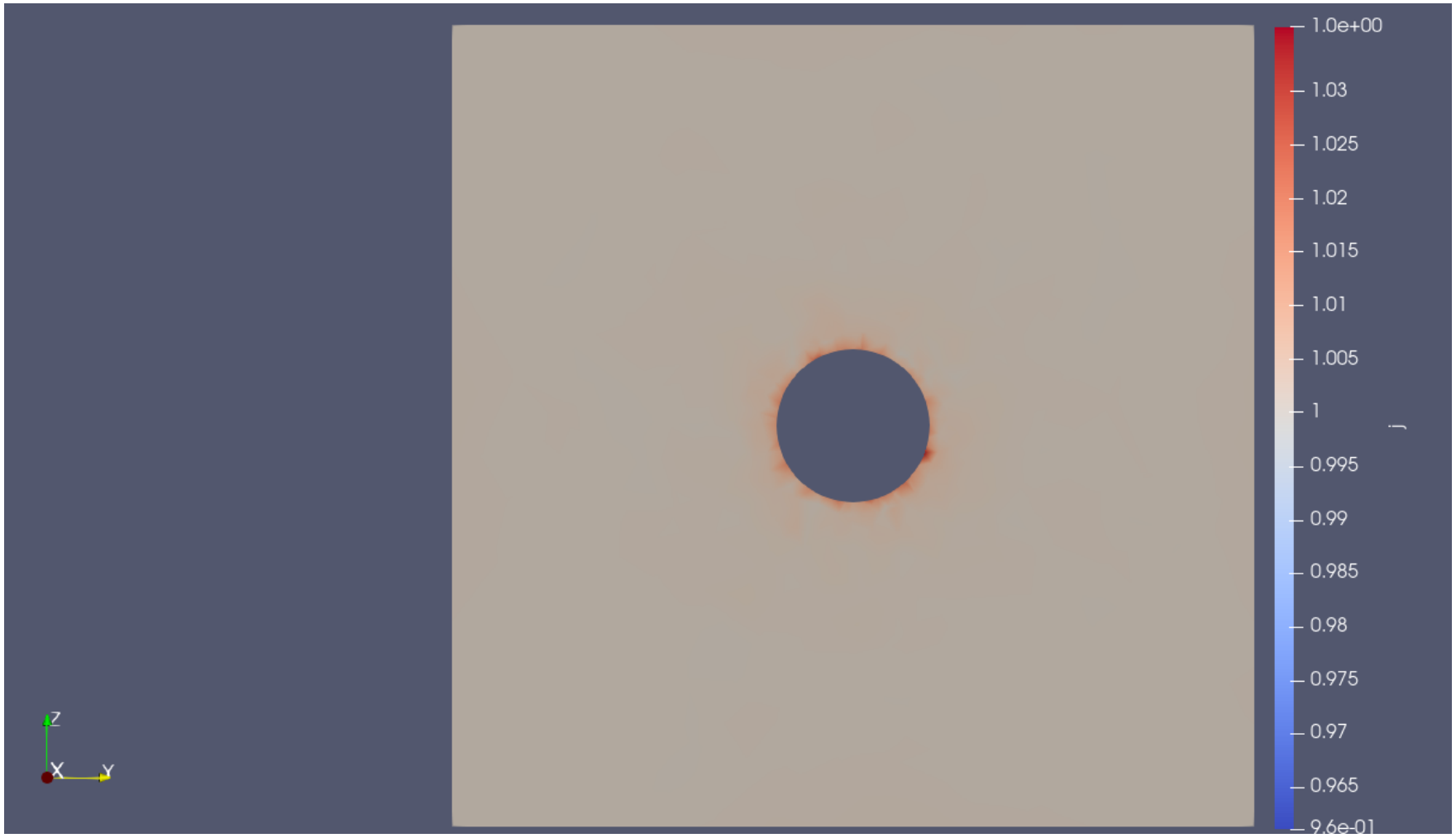

1.0e+00
1.03
1.025
1.02
1.015
1.01
1.005
1
0.995
0.99
0.985
0.98
0.975
0.97
0.965
9.6e-01
J
Z
X
Y

Results: J cont.



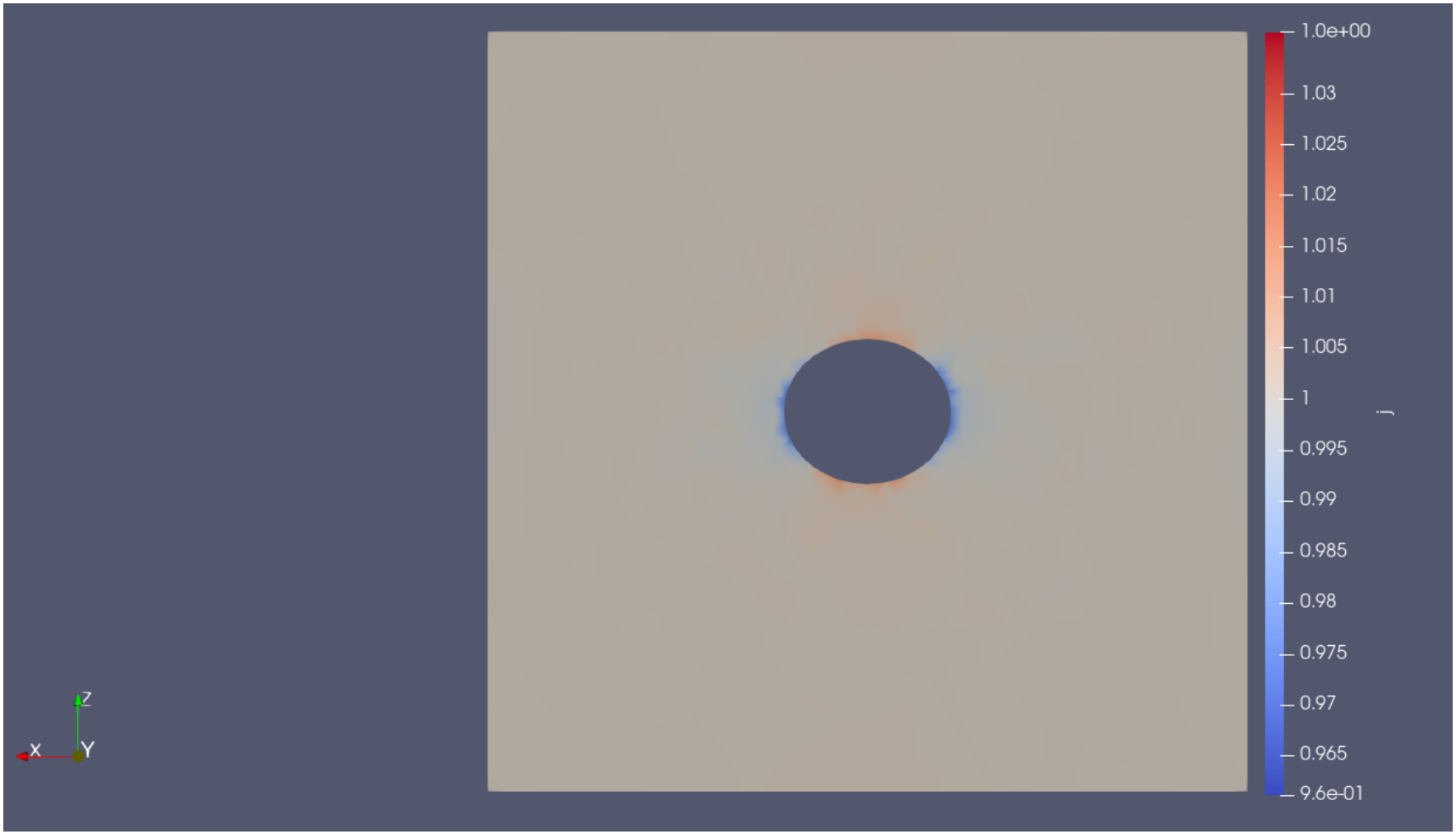
Results: J cross section

J across gel (yz plane cross section):



Results: J cross section cont.

J across gel (xz plane cross section):



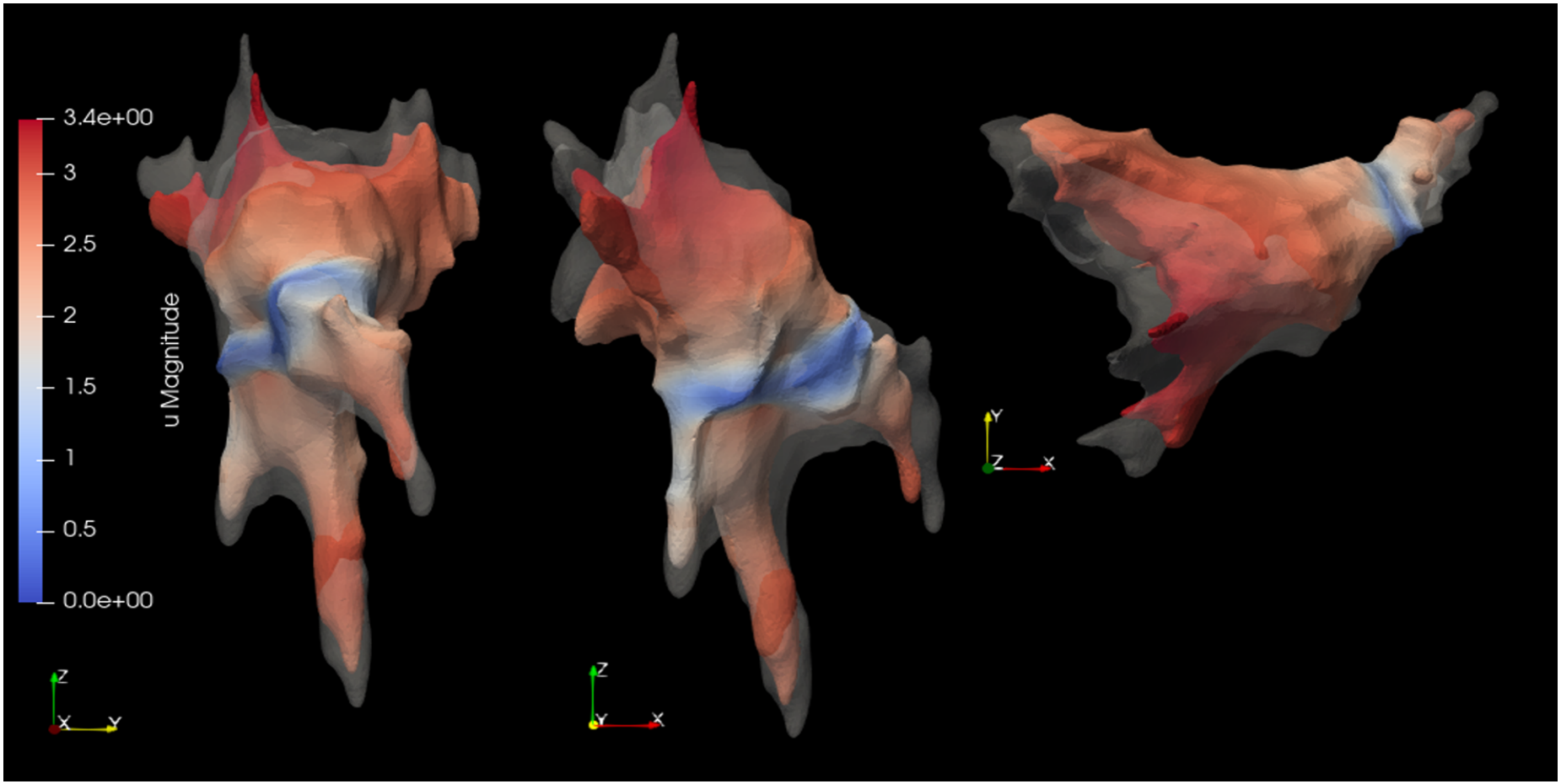
Forward Model on Test Cell Geometry

Boundary Conditions

- d Ω_O represents the gel cube exterior.
- dΩ_I represents the cell exterior (where protrusions occur).
- $\mathbf{u}_{\text{exp}}$ represents the expected displacement of the cell exterior.

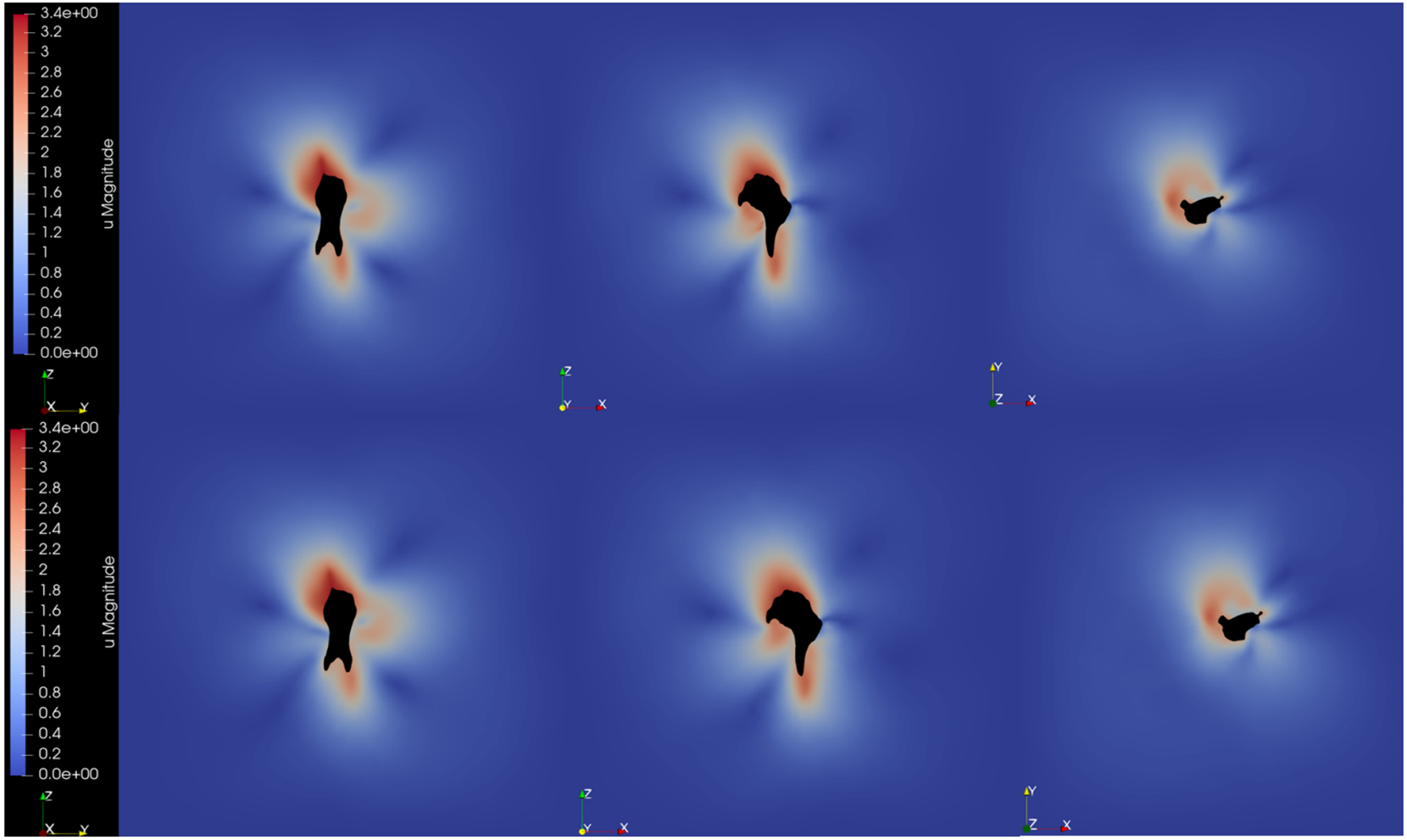
$$\mathbf{u}(\mathbf{x}_0) = \mathbf{0}, \qquad \mathbf{x}_0 \in \partial\Omega_O$$
$$\mathbf{u}(\mathbf{x}_0) = \mathbf{u}_{\text{exp}}(\mathbf{x}_0), \qquad \mathbf{x}_0 \in \partial\Omega_I$$

Boundary Conditions cont.



JAX-FEM forward model solution. Gray denotes reference configuration.

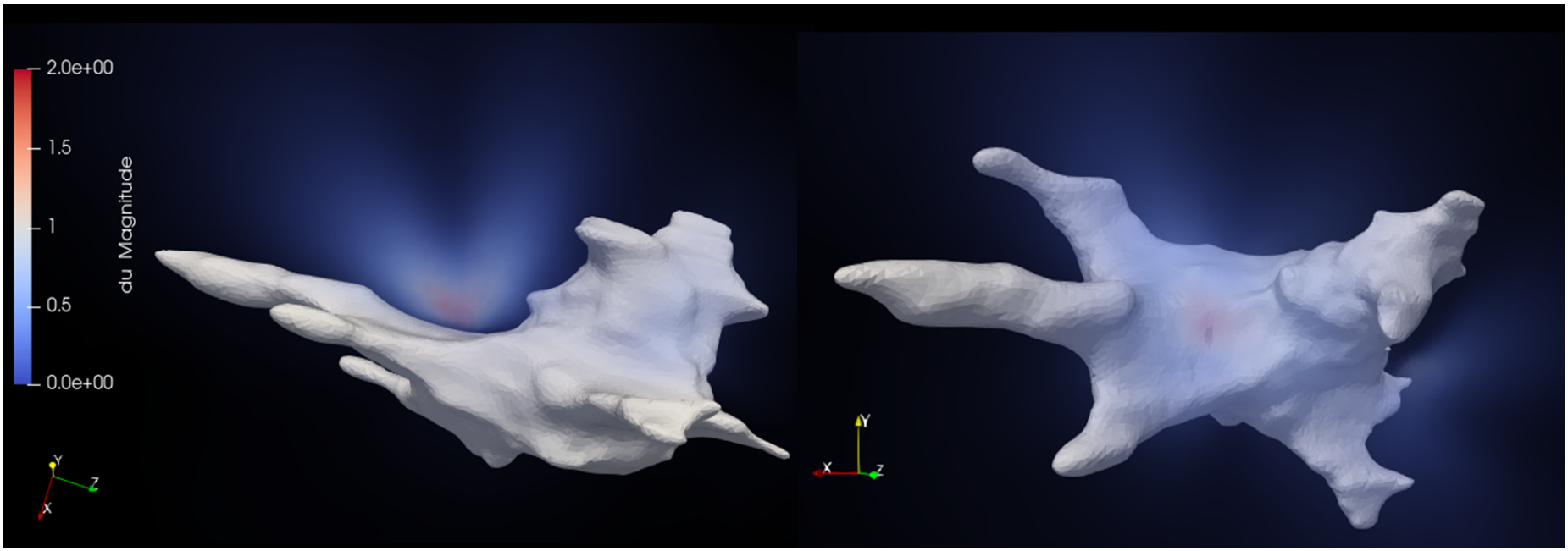
Results ($\mathbf{u}$ field)



$\mathbf{u}$ difference between JAX-FEM(top) and FEniCS (bottom) for 1.96E+04 elements

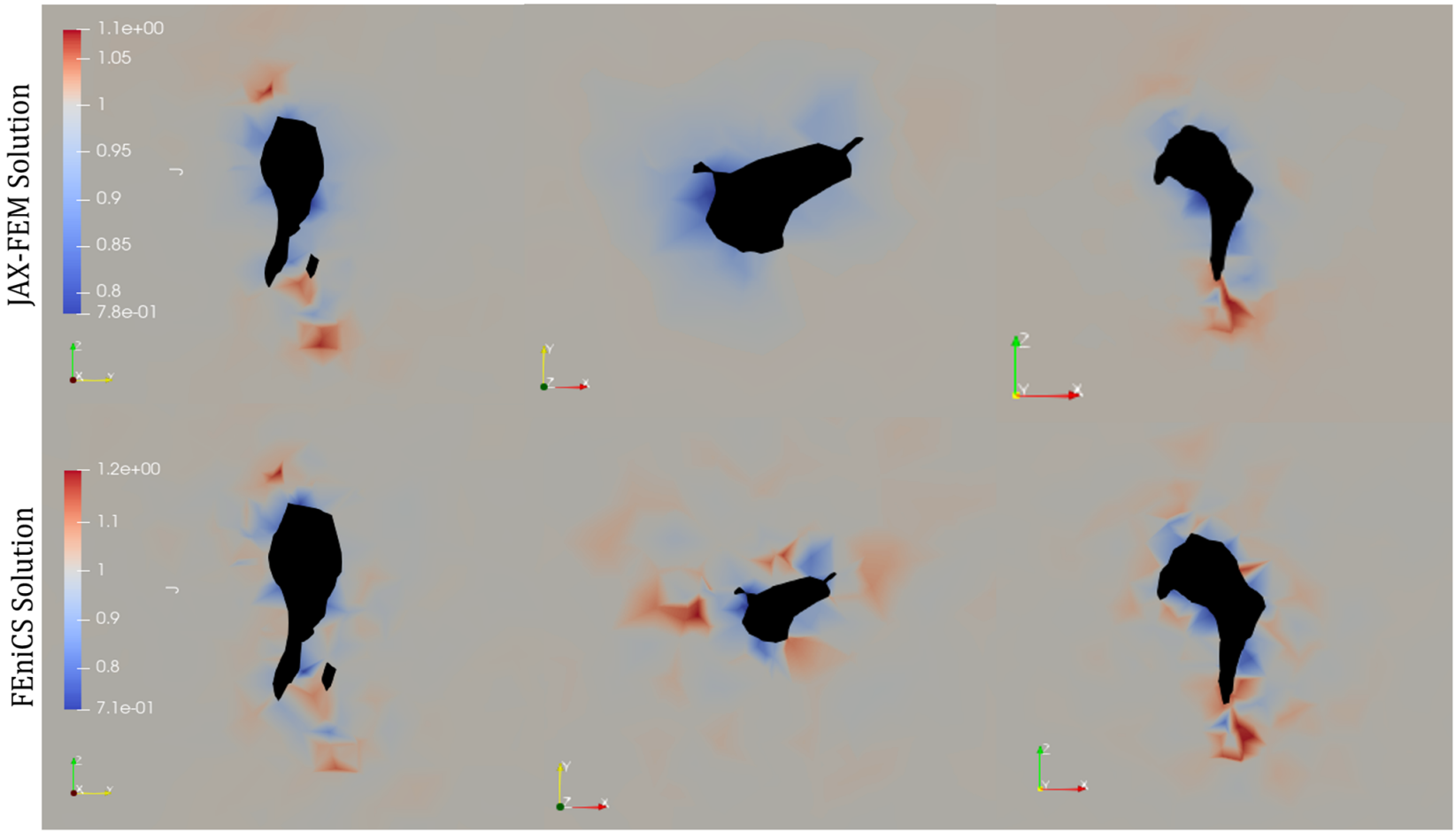
Results (u field) cont.

Note: Residual value (L2 norm of residual vector) does not decrease below 10^3; will need to evaluate if this is due to the preconditioner difference (using Incomplete LU, ILU, instead of Algebraic Multigrid, AMG).



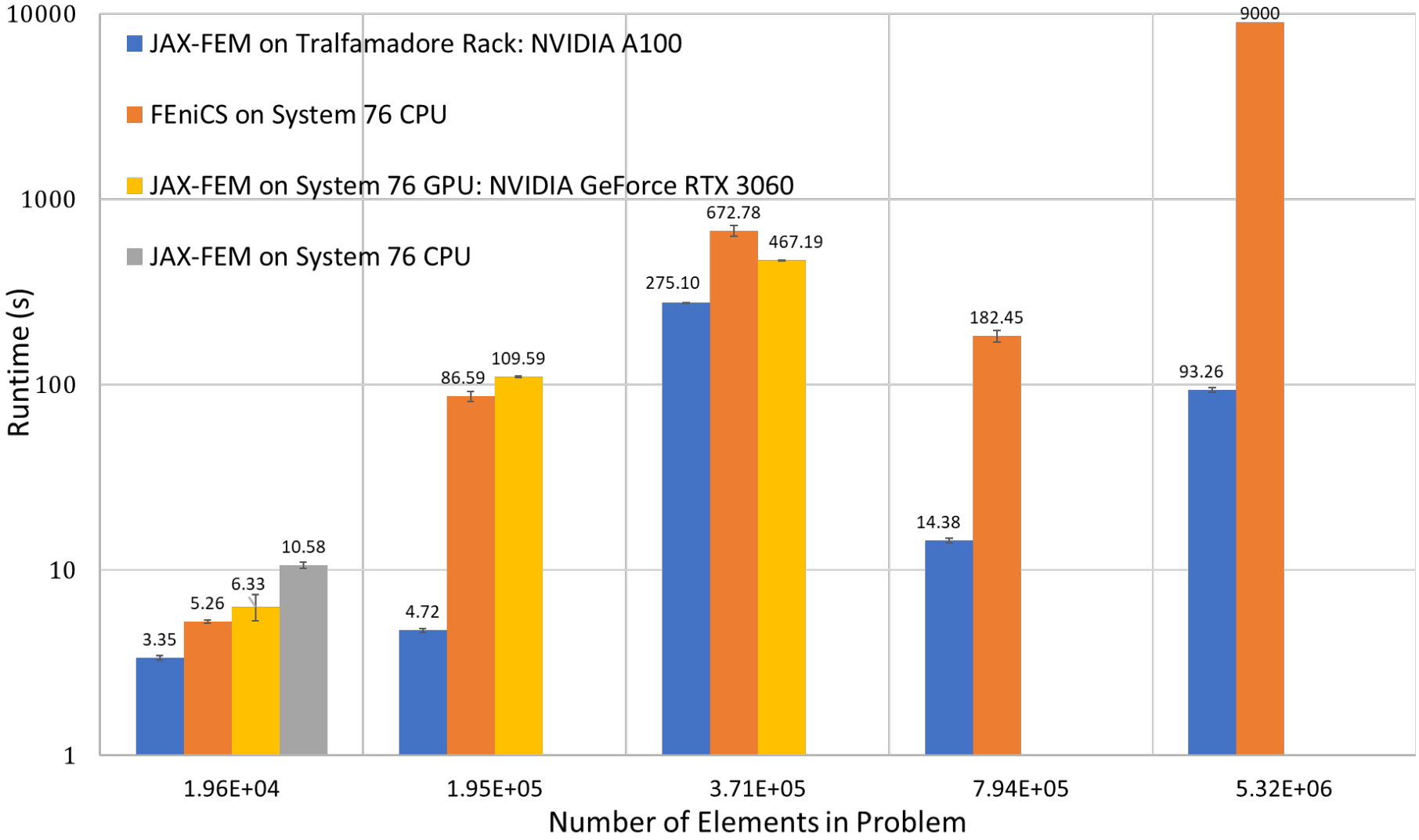
Solution difference between JAX-FEM and FEniCS in reference configuration.

Results: Jacobian of Deformation Gradient (J) field)



J = det(**F**). Jacobian field cross-sections of JAX-FEM and FEniCS solutions (should be 1 per incompressibility).

Results: Forward Model Runtime Comparision (JAX-FEM vs. FEniCS)



Runtime comparision between JAX-FEM and FEniCS. Note logarithmic runtime scale.

Inverse Model on Test Cell Geometry

Introduction & Objective

The inverse model identifies the modulus field with minimal error in simulated nodal displacements, with objective as below.

$$\phi = \int_\Omega \xi : \xi \, d\Omega + \gamma \int_\Omega (\nabla\alpha \cdot \nabla\alpha)^{\frac{1}{2}} \, d\Omega$$

$$\xi = \mathbf{C}_{target} - \mathbf{C}_{simulated}$$

$$\mathbf{C} := \mathbf{F}^T\mathbf{F} \text{ (the Cauchy-Green tensor)}$$

$$\Omega = \text{reference gel domain}$$

$$\gamma = \text{regularization}$$

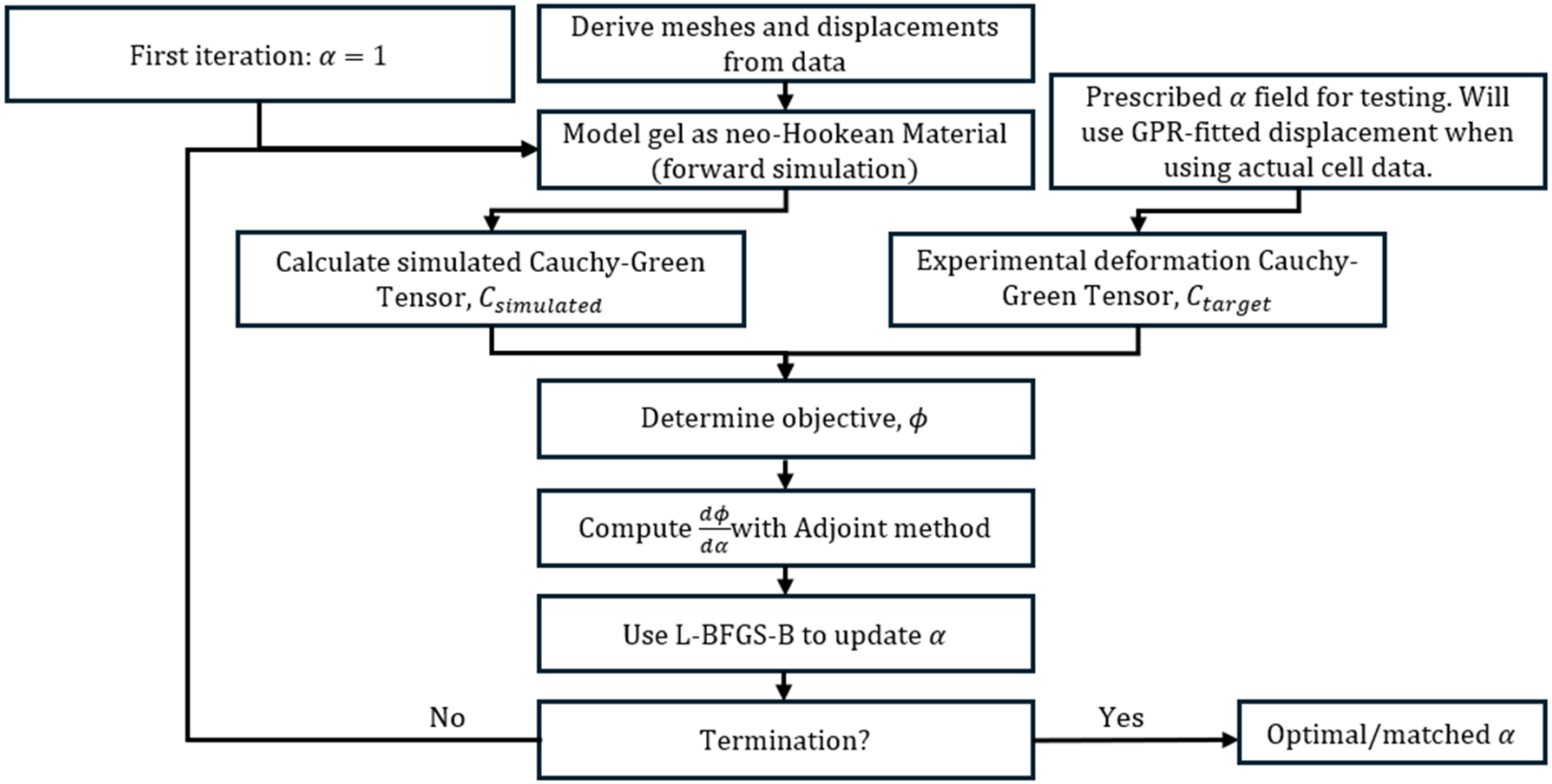
$$\hat{\alpha} = \min_{\alpha(\text{DoFs})} \phi$$

Ideally, Tikhonov regularization would be used instead: $\int_\Omega (\nabla\alpha \cdot \nabla\alpha) \, d\Omega$

Inverse Model Objective

Due to the piecewise constant alpha field (natively supported in JAX-FEM), Tikhonov regularization would cause divergence of the objective.
SciPy's optimize-minimize function is used with the L-BFGS-B (Limited-Memory Broyden-Fletcher-Goldfarb-Shanno algorithm) optimization algorithm. While BFGS stores a dense n x n approximation of the inverse Hessian (2nd derivative) matrix, L-BFGS-B stores only a few vectors that represent the approximation implicitly, making it well suited for inverse models with many degrees of freedom (DoFs).

Pipeline



Inverse Model Pipeline

Adjoint Method

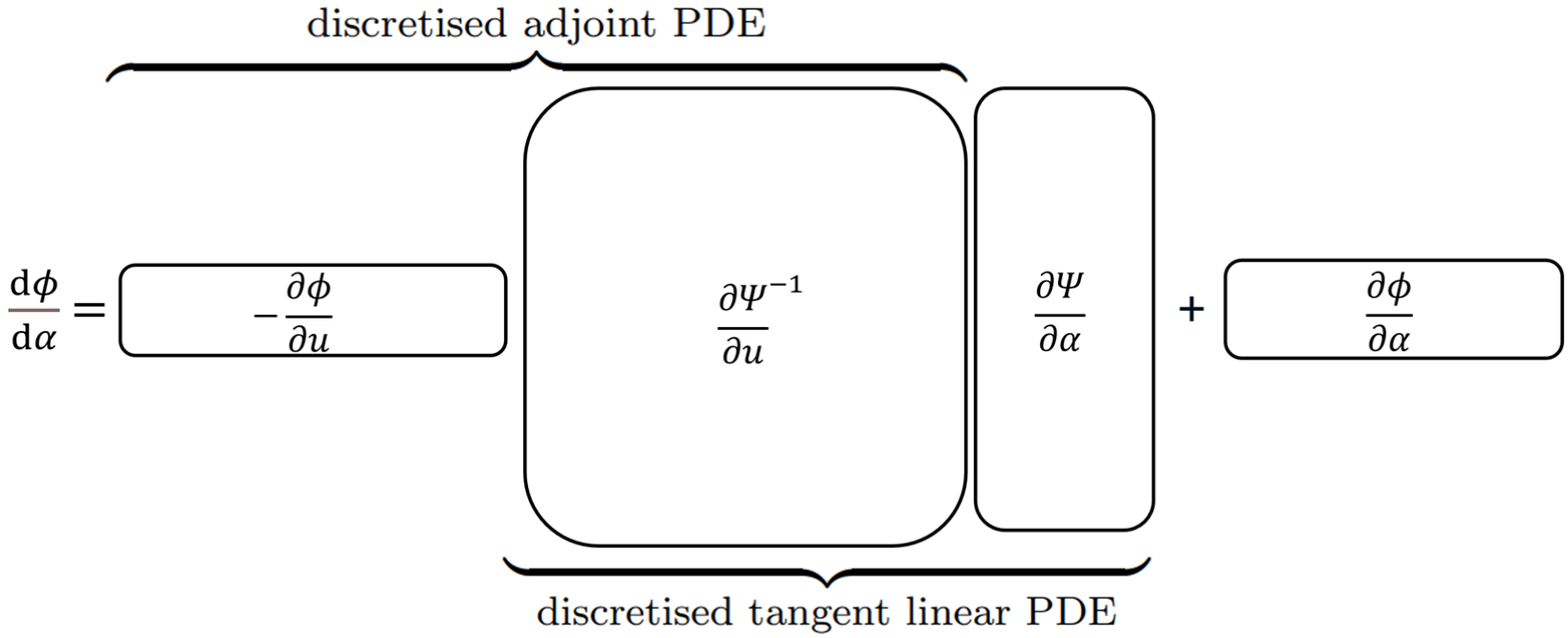
The objective gradient is determined with an adjoint method, allowing for more accuracy and efficiency than with a finite difference gradient approximation.
The strain energy density, Ψ, provides a PDE relating the solution (u, the displacement field) and the optimization parameter (α, the stiffness field), such that $\Psi(u, \alpha) = 0$. Setting derivatives to zero (first-order optimality), we have:

$$\frac{d\phi}{d\alpha} = \frac{\partial \phi}{\partial u}\frac{du}{d\alpha} + \frac{\partial \phi}{\partial \alpha}$$

$$\frac{d\Psi}{d\alpha} = \frac{\partial \Psi}{\partial u}\frac{du}{d\alpha} + \frac{\partial \Psi}{\partial \alpha} = 0$$

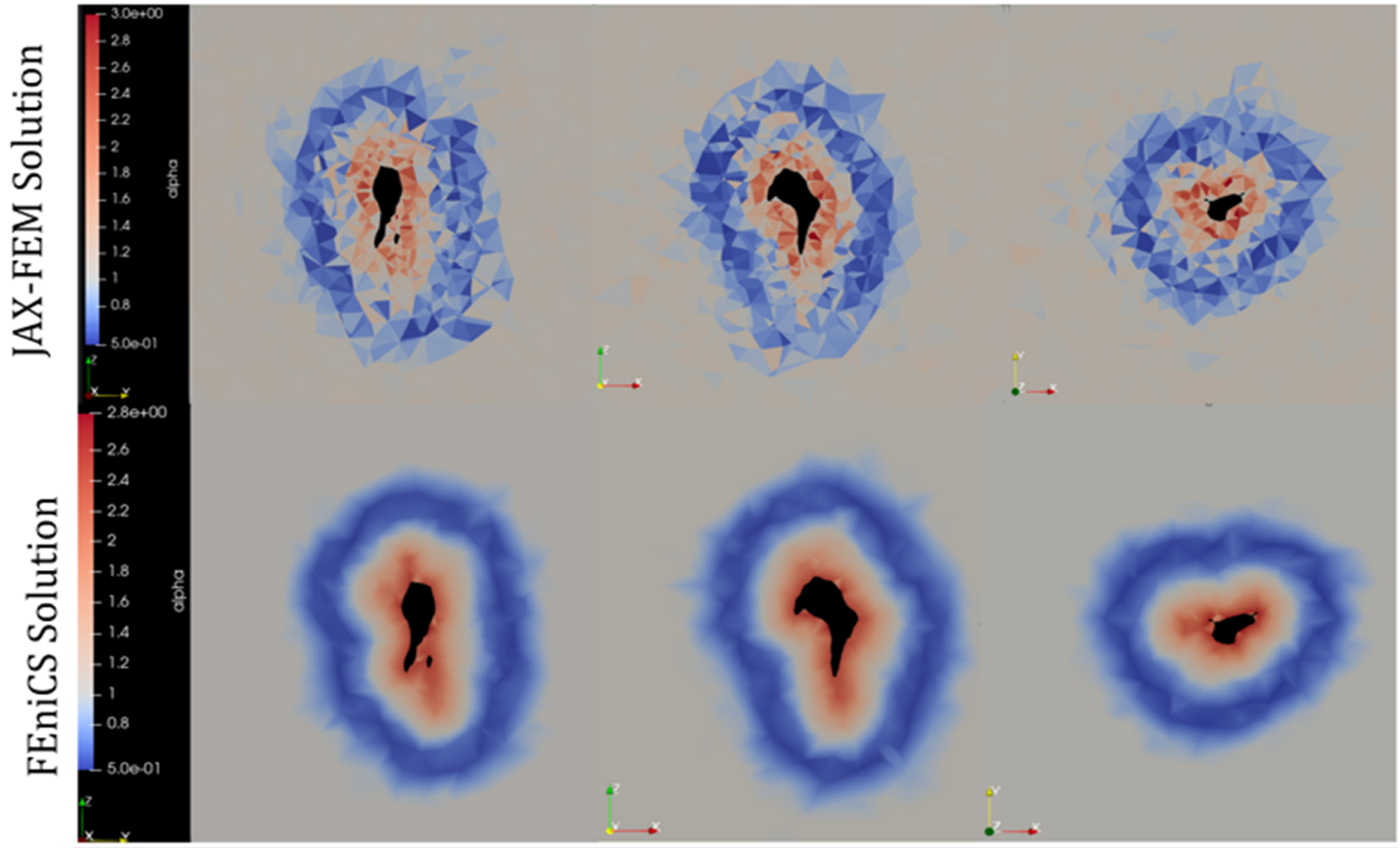
$$\implies \frac{du}{d\alpha} = -[\frac{\partial \Psi}{\partial u}]^{-1}\frac{\partial \Psi}{\partial \alpha}$$

Adjoint Method Visualization



The adjoint PDE uses vector-matrix computations, while the tangent linear PDE requires expensive matrix-matrix computations.

Results: Recovered α field



Piece-wise constant alpha field will be a future implementation. alpha field on 19K element problem, 700 iterations. 3806.77 seconds or 1.057 hrs for JAX-FEM vs 50 minutes for FEniCS.

Future Steps

To-Do

1. Implement a stronger preconditioner to decrease residual below 10**-9.
2. Implement a piecewise linear field with Tikhonov regularization for the objective in the inverse model.
3. Run the inverse model on a real cell to recover the α field.
4. Develop advanced AVIC stress fiber models.

Citations

- Deepmodeling. "Jax-Fem/Demos/Hyperelasticity at Main · Deepmodeling/Jax-FEM." GitHub, github.com/deepmodeling/jax-fem/tree/main/demos/hyperelasticity. Accessed 14 June 2024.
- Holzapfel, Gerhard A. Nonlinear Solid Mechanics: A Continuum Approach for Engineering. John Wiley & Sons, 2000.
- "Neo-Hookean Solid." Wikipedia, Wikimedia Foundation, 1 May 2024, en.wikipedia.org/wiki/Neo-Hookean_solid.