Description

This notebook intends to gather all the functionalities you'll have to implement for assignment 2.2.

Load libraries

In [1]:
```
import numpy as np
import igl
import meshplot as mp
import time

import sys as _sys
_sys.path.append("../src")
from elasticsolid import *
from elasticenergy import *
from matplotlib import gridspec
import matplotlib.pyplot as plt

shadingOptions = {
    "flat":True,
    "wireframe":False,
}

rot = np.array(
    [[1, 0, 0 ],
     [0, 0, 1],
     [0, -1, 0 ]]
)
```

Load mesh

Several meshes are available for you to play with under `data/` : `ball.obj` , `dinosaur.obj` , and `beam.obj` . You can also uncomment the few commented lines below to manipulate a simple mesh made out of 2 tetrahedra.

In [2]:

```
# v, _, _, t, _, _ = igl.read_obj("../data/dinosaur.obj")
v, _, _, t, _, _ = igl.read_obj("../data/beam.obj")

# t = np.array([
#         [0, 1, 2, 3],
#         [1, 2, 3, 4]
#     ])
# v = np.array([
#     [0., 0., 0.],
#     [1., 0., 0.],
#     [0., 1., 0.],
#     [0., 0., 1.],
#     [2/3, 2/3, 2/3]
# ])

be = igl.edges(igl.boundary_facets(t))
e = igl.edges(t)

aabb = np.max(v, axis=0) - np.min(v, axis=0)
length_scale = np.mean(aabb)

p = mp.plot(v @ rot.T, t, shading=shadingOptions)
```

Linear/Non-Linear Elastic Solid

Instantiation

We first specify the elasticity model to use for the elastic solid, as well as pinned vertices, and volumetric forces.

In [3]:

```
rho      = 131  # [kg.m-3]
young    = 10e6 # [Pa]
poisson = 0.2
force_mass = np.zeros(shape = (3,))
force_mass[2] = - rho * 9.81

minX     = np.min(v[:, 0])
pin_idx = np.arange(v.shape[0])[v[:, 0] < minX + 0.2*aabb[0]]

lin_ee     = LinearElasticEnergy(young, poisson)
neo_ee     = NeoHookeanElasticEnergy(young, poisson)

lin_solid = ElasticSolid(v, t, lin_ee, rho=rho, pin_idx=pin_idx, f_mass=for
neo_solid = ElasticSolid(v, t, neo_ee, rho=rho, pin_idx=pin_idx, f_mass=for

solid = neo_solid
```

Deform the mesh

This should now involve elastic forces computation.

In [4]:
```
v_def = v.copy()
v_def[:, 0] *= 2.
solid.update_def_shape(v_def)

p = mp.plot(solid.v_def @ rot.T, solid.t, shading=shadingOptions)
p.add_points(solid.v_def[solid.pin_idx, :] @ rot.T, shading={"point_color":
forcesScale = 2 * np.max(np.linalg.norm(solid.f_ext, axis=1))
p.add_lines(solid.v_def @ rot.T, (solid.v_def + solid.f_ext / forcesScale)
p.add_edges(v @ rot.T, be, shading={"line_color": "blue"})
```

Out[4]:
```
3
```

Find equilibrium

We compare different methods: number of steps, computation time.

In [5]:
```
n_steps = 1000
thresh  = 1.
v_init = v.copy()
```

Gradient descent

In [6]:
```
report_GD = equilibrium_convergence_report_GD(solid, v_init, n_steps, 5e-5,
energies_el_GD  = report_GD['energies_el']
energies_ext_GD = report_GD['energies_ext']
energy_GD       = energies_el_GD + energies_ext_GD
residuals_GD    = report_GD['residuals']
times_GD        = report_GD['times']
idx_stop_GD     = report_GD['idx_stop']
v_def_GD        = solid.v_def.copy()

# Lastly, plot the resulting shape
p = mp.plot(v_def_GD @ rot.T, solid.t, shading=shadingOptions)
forcesScale = 2 * np.max(np.linalg.norm(solid.f_ext, axis=1))
p.add_lines(v_def_GD @ rot.T, (solid.v_def + solid.f_ext / forcesScale) @ r
p.add_points(v_def_GD[pin_idx, :] @ rot.T, shading={"point_color":"black",
p.add_edges(v @ rot.T, be, shading={"line_color": "blue"})
```

```
/opt/notebooks/assignment_2_2/notebook/../src/elasticenergy.py:88: RuntimeW
arning: invalid value encountered in log
  j = np.log(np.linalg.det(jac))
/opt/notebooks/assignment_2_2/notebook/../src/elasticenergy.py:103: Runtime
Warning: invalid value encountered in log
  self.logJ = np.log(np.linalg.det(jac))
```

Out[6]:
```
3
```

BFGS

In [7]:

```
report_BFGS = equilibrium_convergence_report_BFGS(solid, v_init, n_steps, 1
energies_el_BFGS  = report_BFGS['energies_el']
energies_ext_BFGS = report_BFGS['energies_ext']
energy_BFGS       = energies_el_BFGS + energies_ext_BFGS
residuals_BFGS    = report_BFGS['residuals']
times_BFGS        = report_BFGS['times']
idx_stop_BFGS     = report_BFGS['idx_stop']
v_def_BFGS        = solid.v_def.copy()

# Lastly, plot the resulting shape
p = mp.plot(v_def_BFGS @ rot.T, solid.t, shading=shadingOptions)
forcesScale = 2 * np.max(np.linalg.norm(solid.f_ext, axis=1))
p.add_lines(v_def_BFGS @ rot.T, (solid.v_def + solid.f_ext / forcesScale) @
p.add_points(v_def_BFGS[pin_idx, :] @ rot.T, shading={"point_color":"black"
p.add_edges(v @ rot.T, be, shading={"line_color": "blue"})
```

```
/opt/conda/envs/gc_course_env/lib/python3.9/site-packages/numpy/linalg/lina
lg.py:2158: RuntimeWarning: invalid value encountered in det
  r = _umath_linalg.det(a, signature=signature)
```

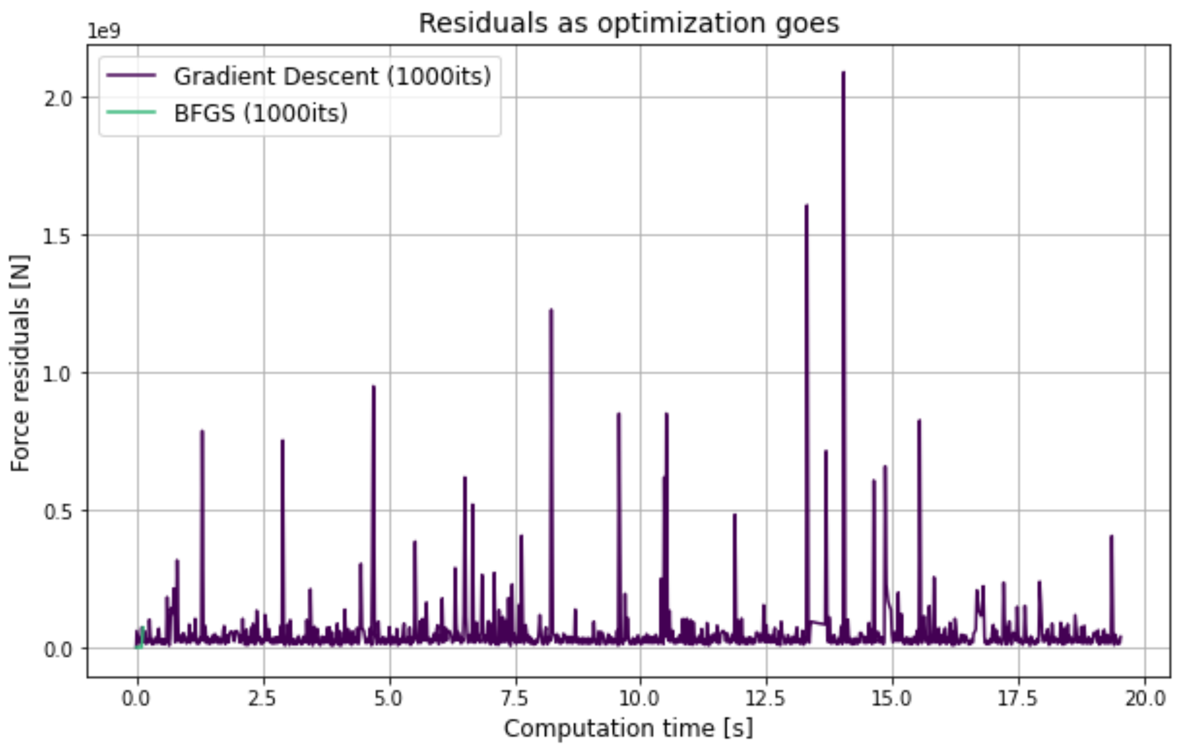
Out[7]:

```
3
```

Compare the different algorithms

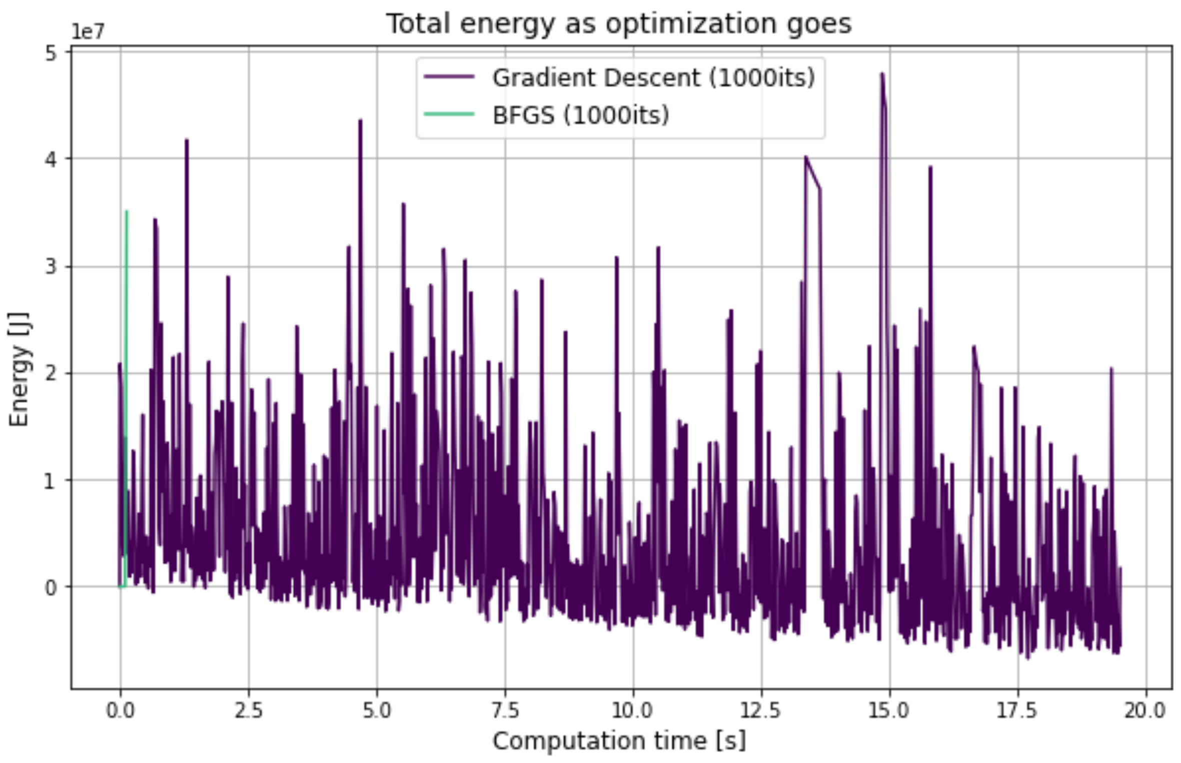
In [8]:

```
cmap   = plt.get_cmap('viridis')
colors = cmap(np.linspace(0., 1., 4))

gs = gridspec.GridSpec(nrows=1, ncols=1, width_ratios=[1], height_ratios=[1
fig = plt.figure(figsize=(10, 6))

axTmp = plt.subplot(gs[0, 0])
axTmp.plot(times_GD[:idx_stop_GD+1], residuals_GD[:idx_stop_GD+1], c=colors
           label="Gradient Descent ({:}its)".format(idx_stop_GD))
axTmp.plot(times_BFGS[:idx_stop_BFGS+1], residuals_BFGS[:idx_stop_BFGS+1],
           label="BFGS ({:}its)".format(idx_stop_BFGS))
y_lim = axTmp.get_ylim()
axTmp.set_title("Residuals as optimization goes", fontsize=14)
axTmp.set_xlabel("Computation time [s]", fontsize=12)
axTmp.set_ylabel("Force residuals [N]", fontsize=12)
axTmp.set_ylim(y_lim)
plt.legend(fontsize=12)
plt.grid()
plt.show()
```



In [9]:

```
gs = gridspec.GridSpec(nrows=1, ncols=1, width_ratios=[1], height_ratios=[1
fig = plt.figure(figsize=(10, 6))

axTmp = plt.subplot(gs[0, 0])
axTmp.plot(times_GD[:idx_stop_GD+1], energy_GD[:idx_stop_GD+1], c=colors[0]
           label="Gradient Descent ({:}its)".format(idx_stop_GD))
axTmp.plot(times_BFGS[:idx_stop_BFGS+1], energy_BFGS[:idx_stop_BFGS+1], c=c
           label="BFGS ({:}its)".format(idx_stop_BFGS))
y_lim = axTmp.get_ylim()
axTmp.set_title("Total energy as optimization goes", fontsize=14)
axTmp.set_xlabel("Computation time [s]", fontsize=12)
axTmp.set_ylabel("Energy [J]", fontsize=12)
axTmp.set_ylim(y_lim)
plt.legend(fontsize=12)
plt.grid()
plt.show()
```



Total energy as optimization goes
1e7
5
4
3
2
1
0
Energy [J]
Gradient Descent (1000its)
BFGS (1000its)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Computation time [s]

```
In [ ]:
```