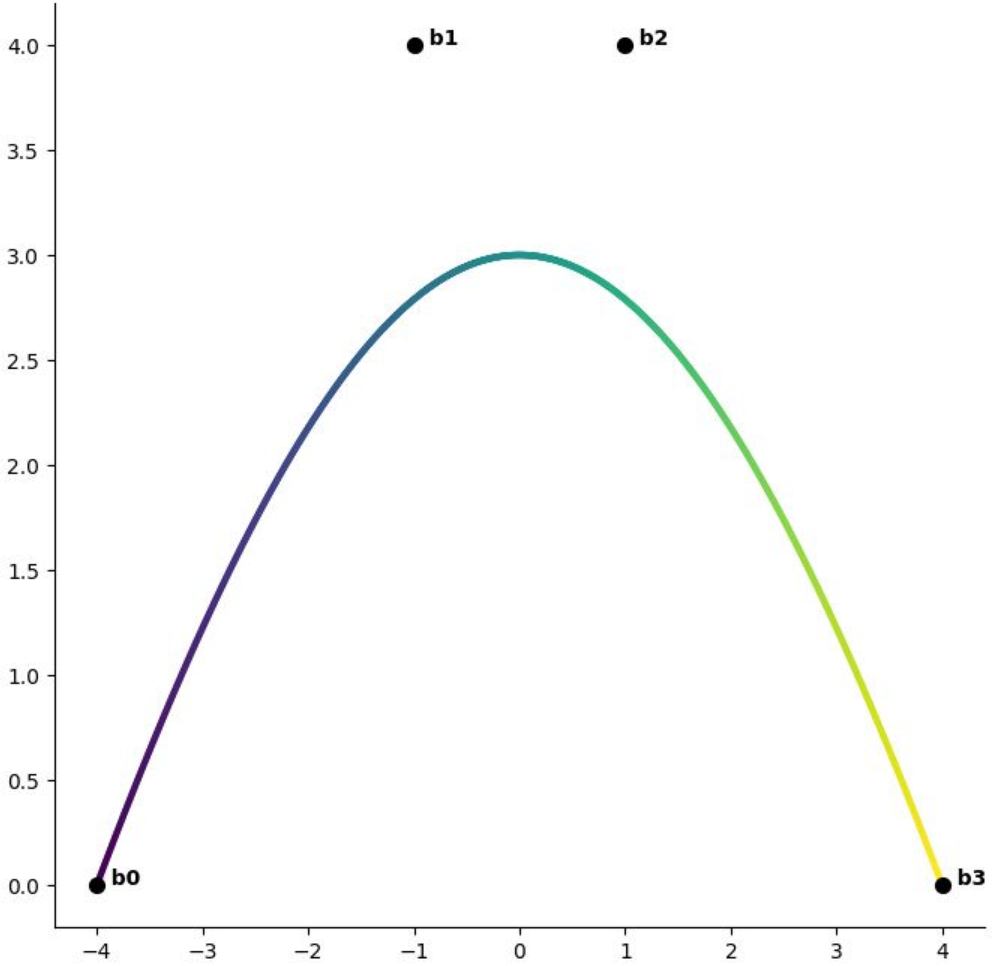
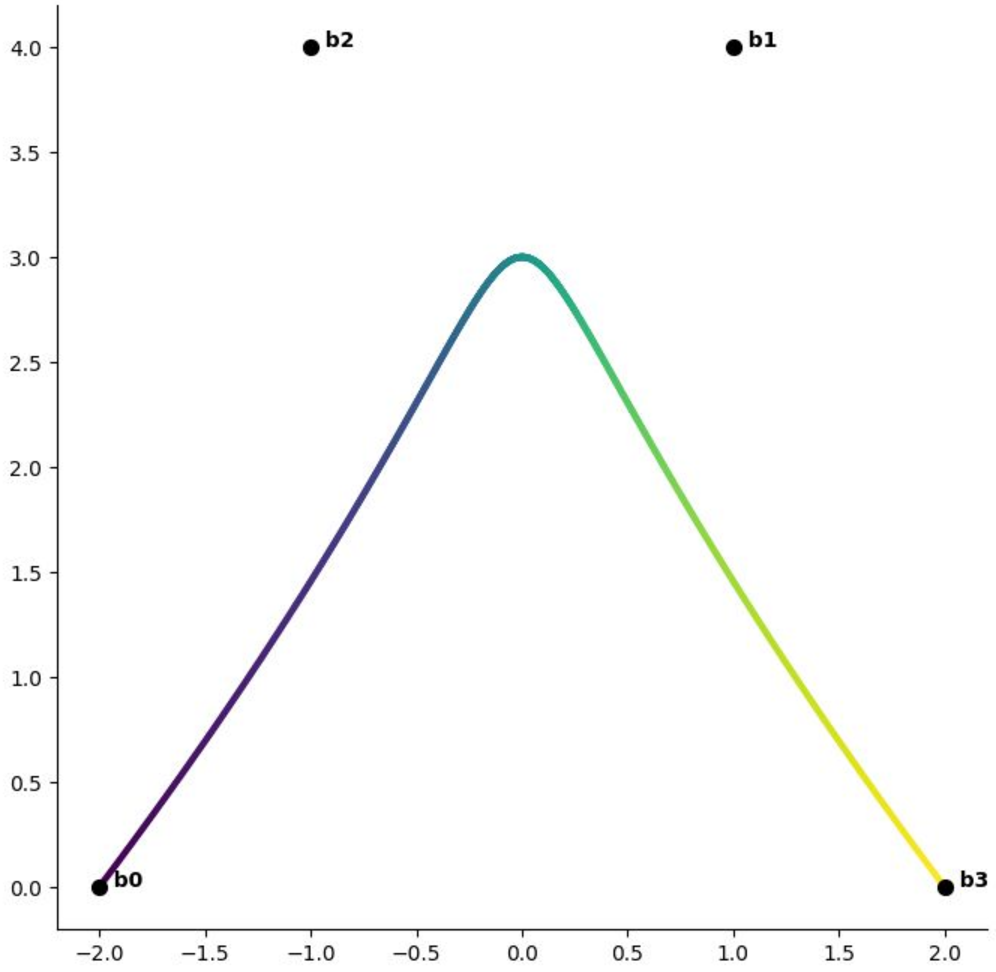
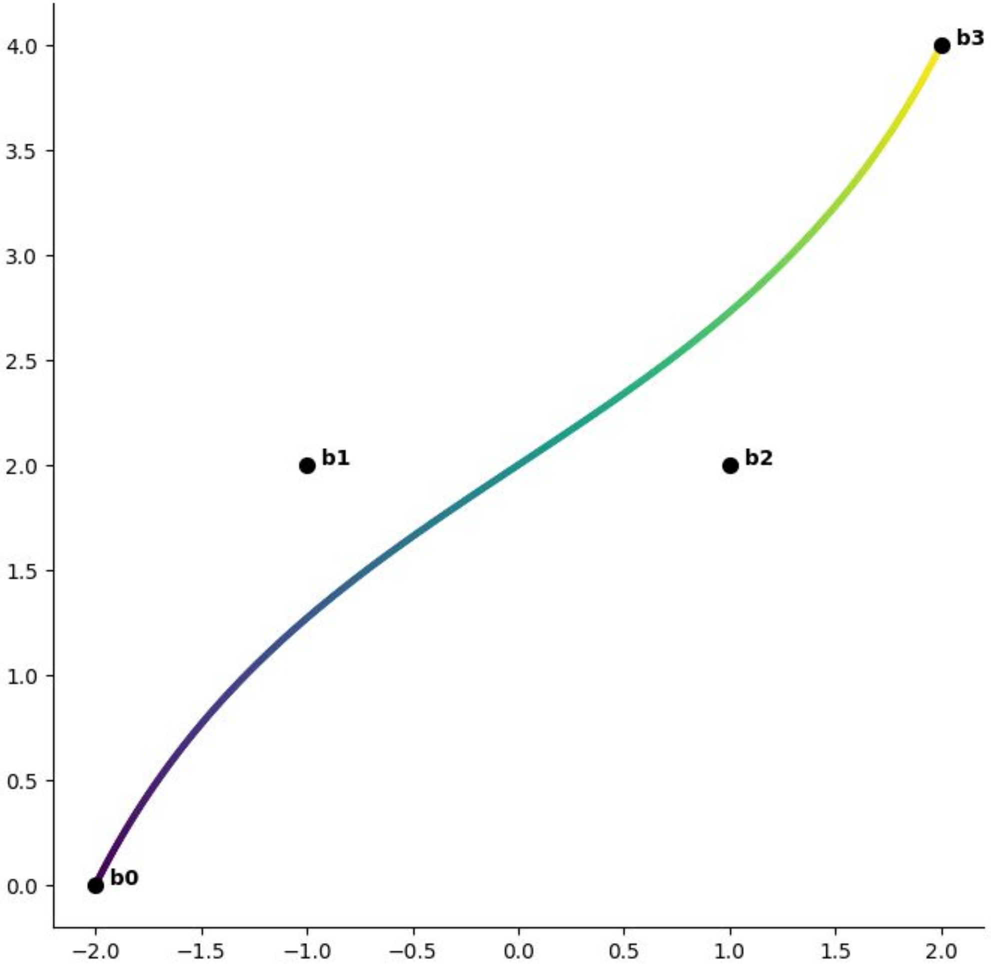
```
# A2.1a in Python

def cubic_bezier(P0, P1, P2, P3, t):
    return (
        (((1 - t) ** 3) * P0)
        + (3 * t * (((1 - t) ** 2) * P1))
        + (3 * (t ** 2) * ((1 - t) * P2))
        + ((t ** 3) * P3)
    )

x = [-4, -1, 1, 4]
y = [0, 4, 4, 0]

import matplotlib.pyplot as plt

fig = plt.gcf()
fig.set_size_inches(8, 8)
ax = fig.add_subplot(111)

for i in range(len(x)):
    ax.text(x[i], y[i], "  b" + str(i), weight="bold")
ax.scatter(x, y, s=50, c="k", zorder=10)

interp_x = []
interp_y = []

for i in range(0, 1000):
    t = i / 1000
    interp_x.append(cubic_bezier(x[0], x[1], x[2], x[3], t))
    interp_y.append(cubic_bezier(y[0], y[1], y[2], y[3], t))

plt.scatter(
    interp_x,
    interp_y,
    s=5,
    c=range(len(interp_x)),
    cmap="viridis",
)

ax.spines["top"].set_visible(False)
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_visible(True)
ax.spines["left"].set_visible(True)

plt.show()
```



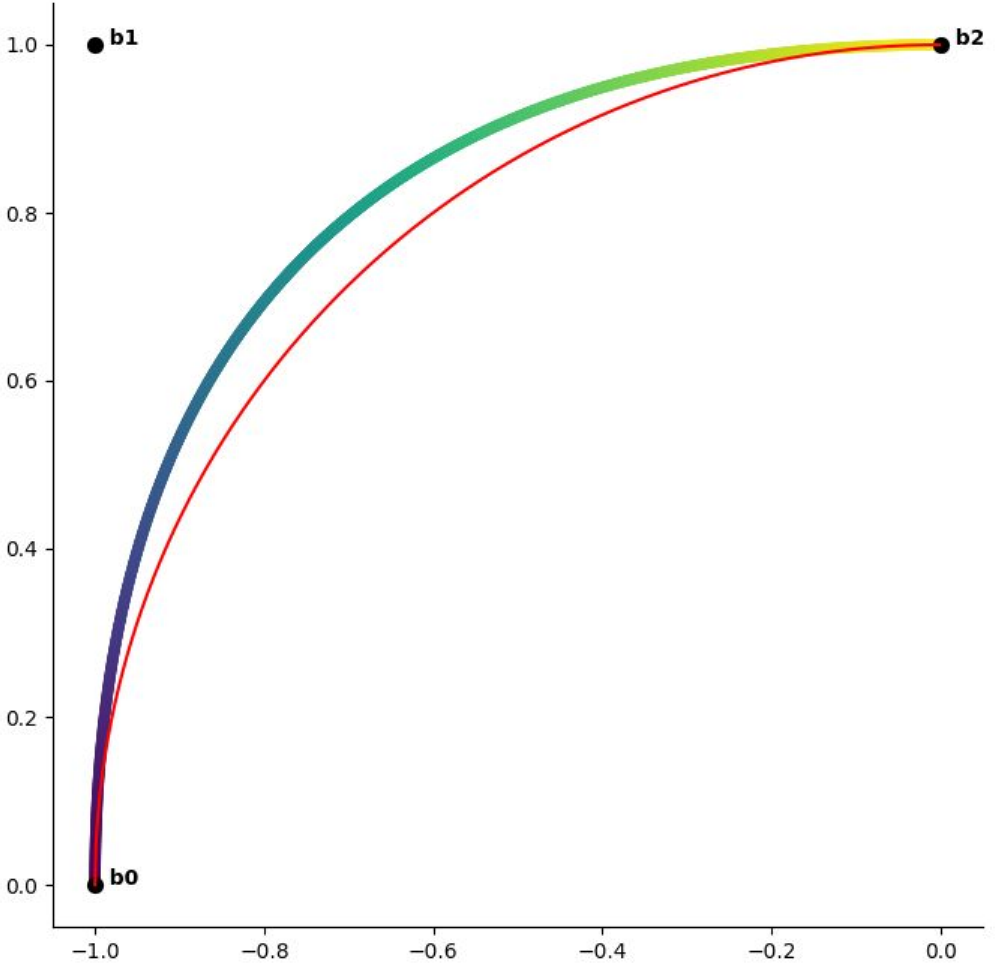
```
# A2.1b in Python

def cubic_bezier(P0, P1, P2, P3, t):
    return (
        (((1 - t) ** 3) * P0)
        + (3 * t * (((1 - t) ** 2) * P1))
        + (3 * (t ** 2) * ((1 - t) * P2))
        + ((t ** 3) * P3)
    )

x = [-2, 1, -1, 2]
y = [0, 4, 4, 0]

import matplotlib.pyplot as plt

fig = plt.gcf()
fig.set_size_inches(8, 8)
ax = fig.add_subplot(111)

for i in range(len(x)):
    ax.text(x[i], y[i], "  b" + str(i), weight="bold")
ax.scatter(x, y, s=50, c="k", zorder=10)

interp_x = []
interp_y = []

for i in range(0, 1000):
    t = i / 1000
    interp_x.append(cubic_bezier(x[0], x[1], x[2], x[3], t))
    interp_y.append(cubic_bezier(y[0], y[1], y[2], y[3], t))

plt.scatter(
    interp_x,
    interp_y,
    s=5,
    c=range(len(interp_x)),
    cmap="viridis",
)

ax.spines["top"].set_visible(False)
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_visible(True)
ax.spines["left"].set_visible(True)

plt.show()
```



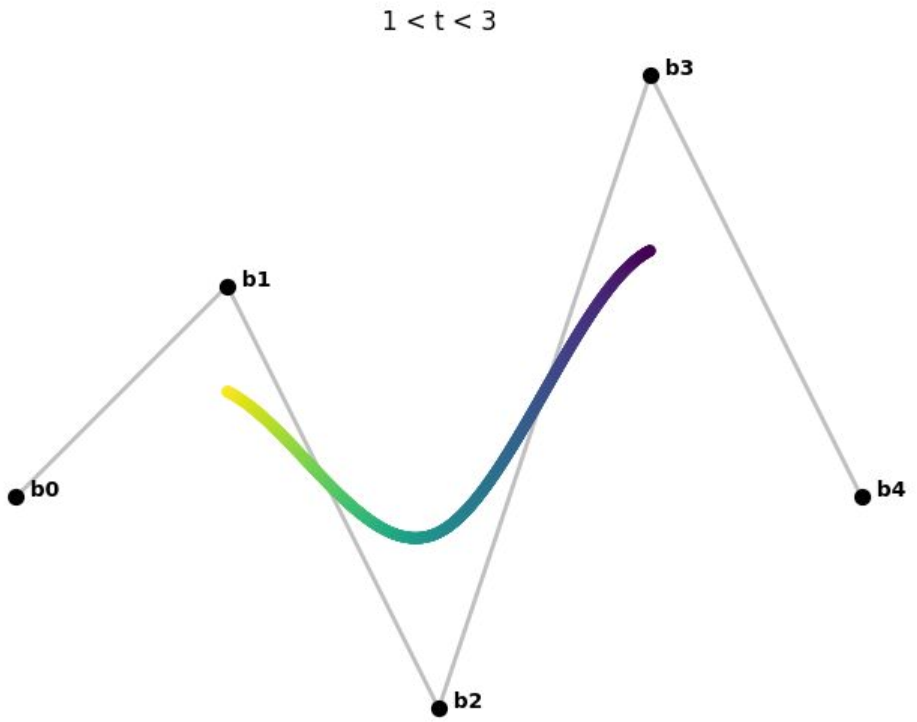
```
# A2.1c in Python

def cubic_bezier(P0, P1, P2, P3, t):
    return (
        (((1 - t) ** 3) * P0)
        + (3 * t * (((1 - t) ** 2) * P1))
        + (3 * (t ** 2) * ((1 - t) * P2))
        + ((t ** 3) * P3)
    )

x = [-2, -1, 1, 2]
y = [0, 2, 2, 4]

import matplotlib.pyplot as plt

fig = plt.gcf()
fig.set_size_inches(8, 8)
ax = fig.add_subplot(111)

for i in range(len(x)):
    ax.text(x[i], y[i], "  b" + str(i), weight="bold")
ax.scatter(x, y, s=50, c="k", zorder=10)

interp_x = []
interp_y = []

for i in range(0, 1000):
    t = i / 1000
    interp_x.append(cubic_bezier(x[0], x[1], x[2], x[3], t))
    interp_y.append(cubic_bezier(y[0], y[1], y[2], y[3], t))

plt.scatter(
    interp_x,
    interp_y,
    s=5,
    c=range(len(interp_x)),
    cmap="viridis",
)

ax.spines["top"].set_visible(False)
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_visible(True)
ax.spines["left"].set_visible(True)

plt.show()
```



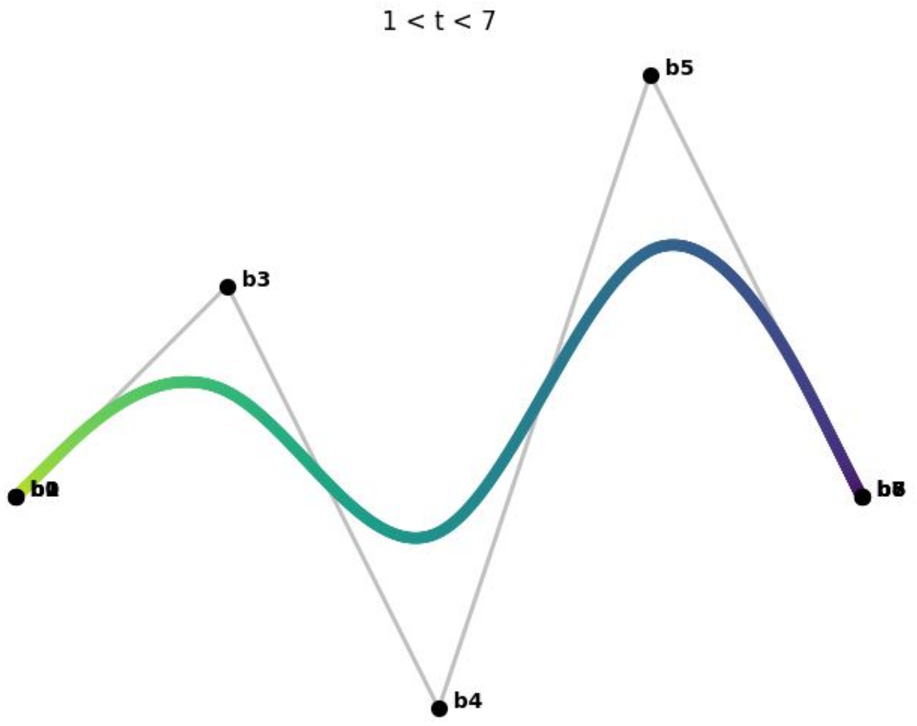
```
# A2.2b in Python

def quadratic_bezier(P0, P1, P2, t):
    return (((1 - t) ** 2) * P0) + ((2 * t * (1 - t)) * P1) + ((t ** 2) * P2)

def circle(t):
    return (1 - (t ** 2)) ** (1 / 2)

x = [-1, -1, 0]
y = [0, 1, 1]

import matplotlib.pyplot as plt

fig = plt.gcf()
fig.set_size_inches(8, 8)
ax = fig.add_subplot(111)

for i in range(len(x)):
    ax.text(x[i], y[i], "  b" + str(i), weight="bold")
ax.scatter(x, y, s=50, c="k", zorder=10)

circle_x = []
circle_y = []
for i in range(-1000, 0):
    t = i / 1000
    circle_x.append(t)
    circle_y.append(circle(-t))
plt.plot(circle_x, circle_y, c="r", zorder=20)

interp_x = []
interp_y = []
for i in range(0, 1000):
    t = i / 1000
    interp_x.append(quadratic_bezier(x[0], x[1], x[2], t))
    interp_y.append(quadratic_bezier(y[0], y[1], y[2], t))
plt.scatter(
    interp_x,
    interp_y,
    s=20,
    c=range(len(interp_x)),
    cmap="viridis",
)

ax.spines["top"].set_visible(False)
ax.spines["right"].set_visible(False)
ax.spines["bottom"].set_visible(True)
ax.spines["left"].set_visible(True)
plt.show()
```



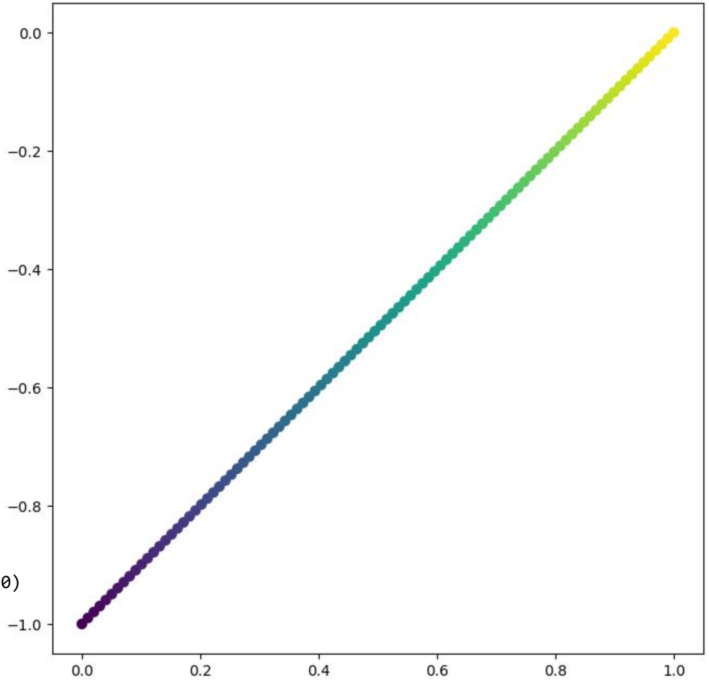
```
# A5.1a

import numpy as np
import scipy.interpolate as si
import matplotlib.pyplot as plt

points = [[0, 0], [1, 1], [2, -1], [3, 2], [4, 0]]

cp_x = []
cp_y = []
for pair in points:
    cp_x.append(pair[0])
    cp_y.append(pair[1])

degree = 3

points = np.array(points)
x = points[:, 0]
y = points[:, 1]

ipl_t = np.linspace(1, len(cp_x) - 2, 1000)

t = range(len(x))

x_list = list(si.splrep(t, x, k=degree, per=1))
x_list[1] = [0.0] + x.tolist() + [0.0, 0.0, 0.0, 0.0]
x_i = si.splev(ipl_t, x_list)

y_list = list(si.splrep(t, y, k=degree, per=1))
y_list[1] = [0.0] + y.tolist() + [0.0, 0.0, 0.0, 0.0]
y_i = si.splev(ipl_t, y_list)

fig = plt.gcf()
fig.set_size_inches(8, 8)
ax = fig.add_subplot(111)

for i in range(len(cp_x)):
    text = ax.text(cp_x[i], cp_y[i], "  b" + str(i), weight="bold")
ax.plot(cp_x, cp_y, c="silver", linewidth=2, zorder=-10)
ax.scatter(cp_x, cp_y, s=50, c="k", zorder=10)

plt.scatter(
    x_i,
    y_i,
    s=20,
    c=range(len(x_i)),
    cmap="viridis_r",
)
plt.title("1 < t < " + str(len(cp_x) - 2))
ax.set_aspect("equal")
plt.axis("off")
plt.show()
```

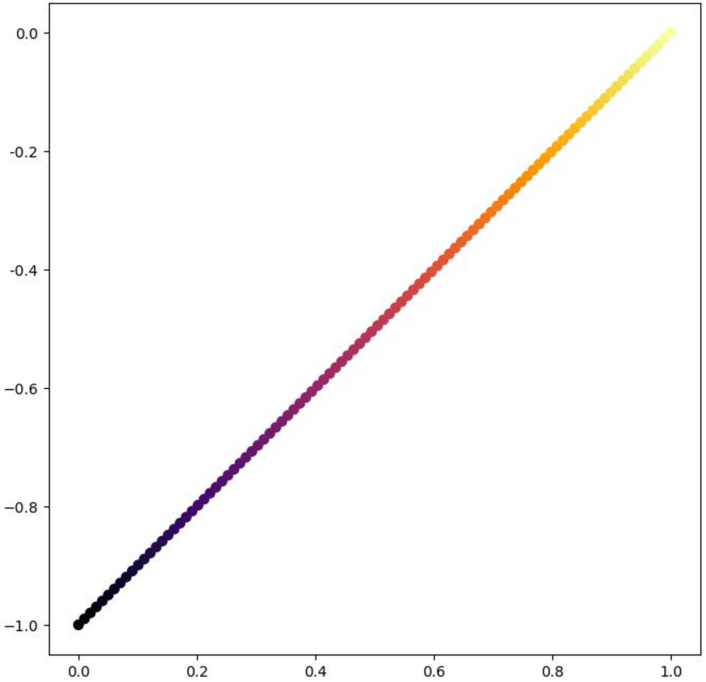


```
# A5.1b

import numpy as np
import scipy.interpolate as si
import matplotlib.pyplot as plt

points = [[0, 0], [0, 0], [0, 0],  # triplicate first point
          [1, 1], [2,-1], [3, 2],
          [4, 0], [4, 0], [4, 0]]  # triplicate last point

cp_x = []
cp_y = []
for pair in points:
    cp_x.append(pair[0])
    cp_y.append(pair[1])

degree = 3

points = np.array(points)
x = points[:, 0]
y = points[:, 1]

ipl_t = np.linspace(1, len(cp_x) - 2, 1000)

t = range(len(x))

x_list = list(si.splrep(t, x, k=degree, per=1))
x_list[1] = [0.0] + x.tolist() + [0.0, 0.0, 0.0, 0.0]
x_i = si.splev(ipl_t, x_list)

y_list = list(si.splrep(t, y, k=degree, per=1))
y_list[1] = [0.0] + y.tolist() + [0.0, 0.0, 0.0, 0.0]
y_i = si.splev(ipl_t, y_list)

fig = plt.gcf()
fig.set_size_inches(8, 8)
ax = fig.add_subplot(111)

for i in range(len(cp_x)):
    text = ax.text(cp_x[i], cp_y[i], "  b" + str(i), weight="bold")
ax.plot(cp_x, cp_y, c="silver", linewidth=2, zorder=-10)
ax.scatter(cp_x, cp_y, s=50, c="k", zorder=10)

plt.scatter(
    x_i,
    y_i,
    s=20,
    c=range(len(x_i)),
    cmap="viridis_r",
)
plt.title("1 < t < " + str(len(cp_x) - 2))
ax.set_aspect("equal")
plt.axis("off")
plt.show()
```

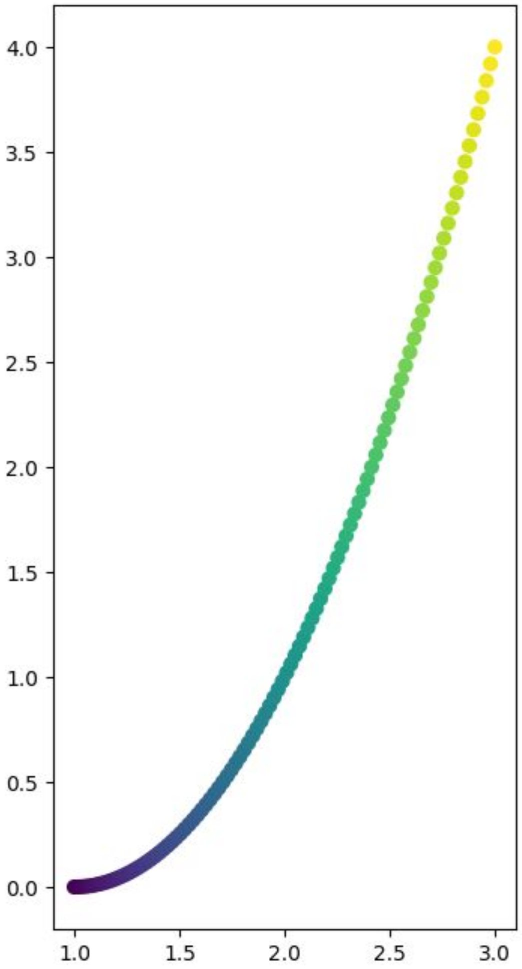


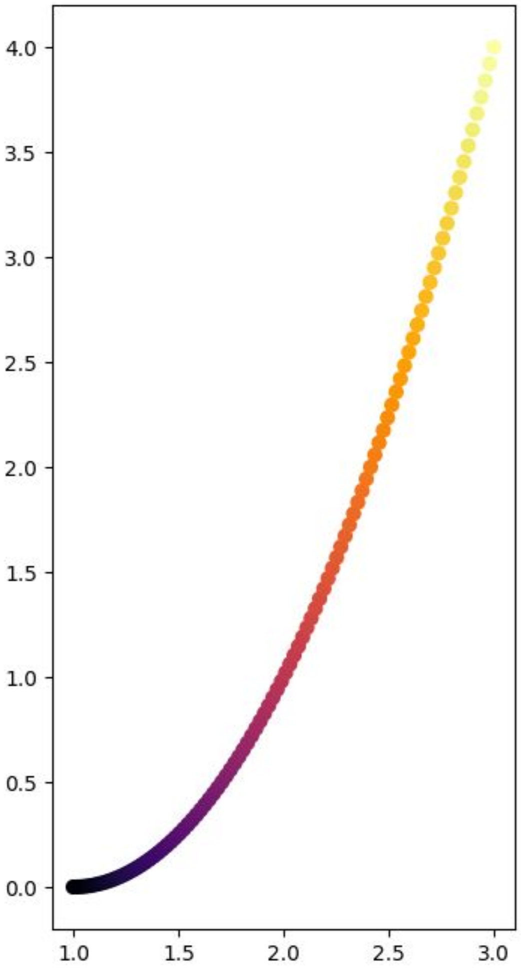
```
# A5.2a

import numpy as np
import matplotlib.pyplot as plt

interp_t = np.linspace(0, 1, 100)
interp_x = []
interp_y = []

for t in interp_t:
    interp_x.append(t)
    interp_y.append(t - 1)

fig = plt.gcf()
fig.set_size_inches(8, 8)
ax = fig.add_subplot(111)
plt.scatter(
    interp_x,
    interp_y,
    s=40,
    c=range(len(interp_x)),
    cmap="viridis",
)
ax.set_aspect("equal")
plt.show()

interp_t = np.linspace((0 / 2), (1 / 2), 100)
interp_x = []
interp_y = []

for t in interp_t:
    interp_x.append((t * 2))
    interp_y.append((t * 2) - 1)

fig = plt.gcf()
fig.set_size_inches(8, 8)
ax = fig.add_subplot(111)
plt.scatter(
    interp_x,
    interp_y,
    s=40,
    c=range(len(interp_x)),
    cmap="inferno",
)
ax.set_aspect("equal")
plt.show()
```



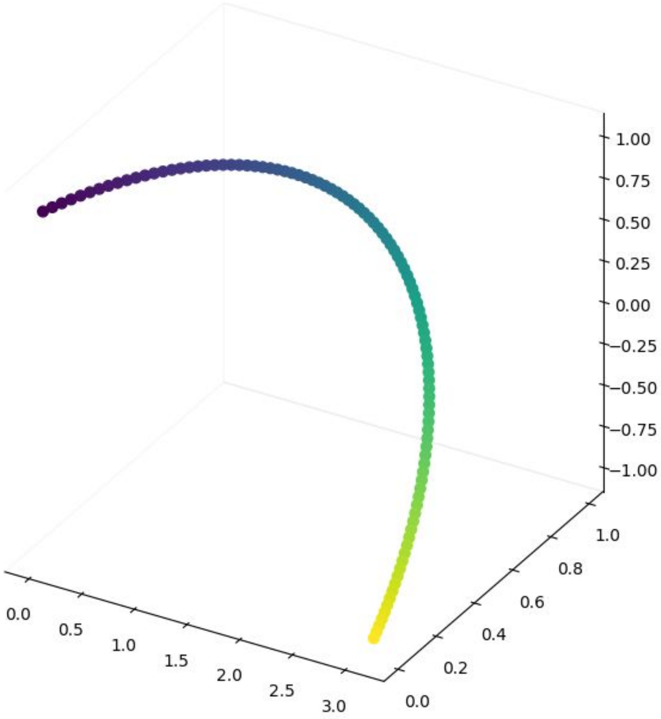


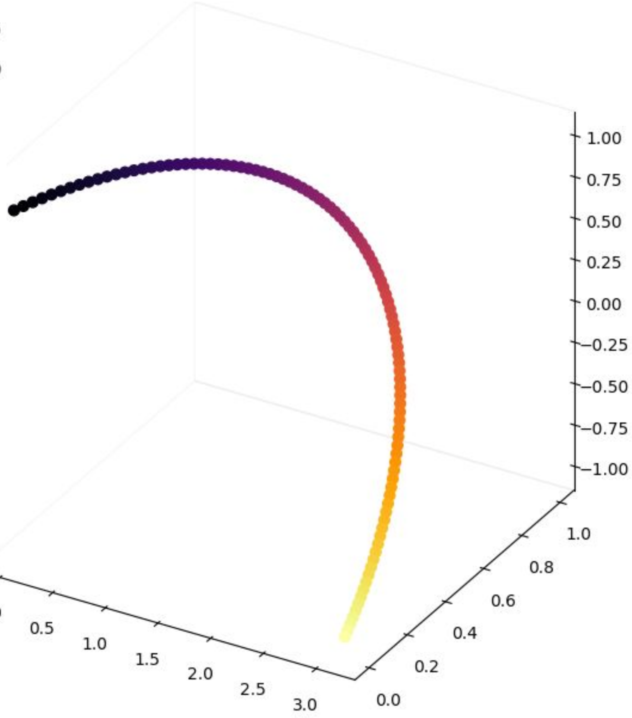
```
# A5.2b

import numpy as np
import matplotlib.pyplot as plt

interp_t = np.linspace(0, 2, 100)
interp_x = []
interp_y = []

for t in interp_t:
    interp_x.append(t + 1)
    interp_y.append(t ** 2)

fig = plt.gcf()
fig.set_size_inches(8, 8)
ax = fig.add_subplot(111)
plt.scatter(
    interp_x,
    interp_y,
    s=40,
    c=range(len(interp_x)),
    cmap="viridis",
)
ax.set_aspect("equal")
plt.show()

interp_t = np.linspace((0 + 1), (2 + 1), 100)
interp_x = []
interp_y = []

for t in interp_t:
    interp_x.append((t - 1) + 1)
    interp_y.append((t - 1) ** 2)

fig = plt.gcf()
fig.set_size_inches(8, 8)
ax = fig.add_subplot(111)
plt.scatter(
    interp_x,
    interp_y,
    s=40,
    c=range(len(interp_x)),
    cmap="inferno",
)
ax.set_aspect("equal")
plt.show()
```



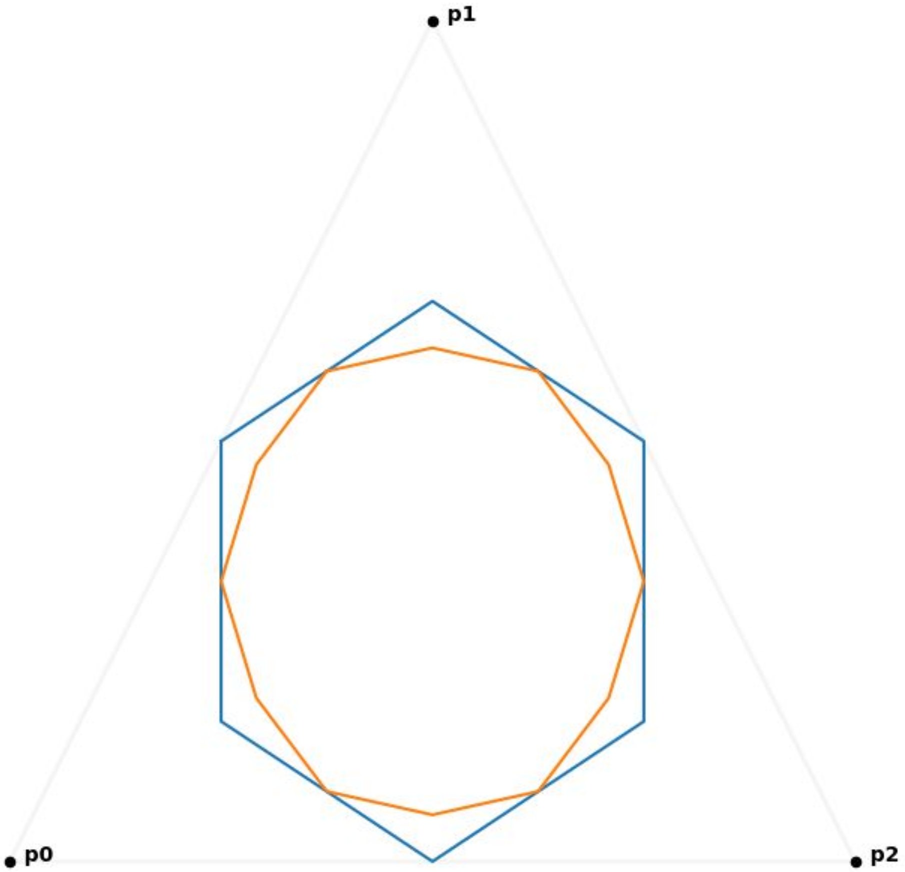


```
# A5.2c

import numpy as np
import matplotlib.pyplot as plt

interp_t = np.linspace(0, np.pi, 100)
interp_x = []
interp_y = []
interp_z = []

for t in interp_t:
    interp_x.append(t)
    interp_y.append(np.sin(t))
    interp_z.append(np.cos(t))

fig = plt.gcf()
fig.set_size_inches(8, 8)
ax = fig.add_subplot(111, projection="3d", proj_type="ortho")

ax.scatter(
    interp_x,
    interp_y,
    interp_z,
    s=40,
    c=plt.cm.viridis(np.linspace(0, 1, len(interp_x))),
    depthshade=False,
)

ax.w_xaxis.pane.fill = False
ax.w_yaxis.pane.fill = False
ax.w_zaxis.pane.fill = False
ax.set_box_aspect((1, 1, 1))
ax.grid(False)
plt.show()

interp_t = np.linspace((0 / np.pi), (np.pi / np.pi), 100)
interp_x = []
interp_y = []
interp_z = []

for t in interp_t:
    interp_x.append((t * np.pi))
    interp_y.append(np.sin((t * np.pi)))
    interp_z.append(np.cos((t * np.pi)))

fig = plt.gcf()
fig.set_size_inches(8, 8)
ax = fig.add_subplot(111, projection="3d", proj_type="ortho")

ax.scatter(
    interp_x,
    interp_y,
    interp_z,
    s=40,
    c=plt.cm.inferno(np.linspace(0, 1, len(interp_x))),
    depthshade=False,
)

ax.w_xaxis.pane.fill = False
ax.w_yaxis.pane.fill = False
ax.w_zaxis.pane.fill = False
ax.set_box_aspect((1, 1, 1))
ax.grid(False)
plt.show()
```

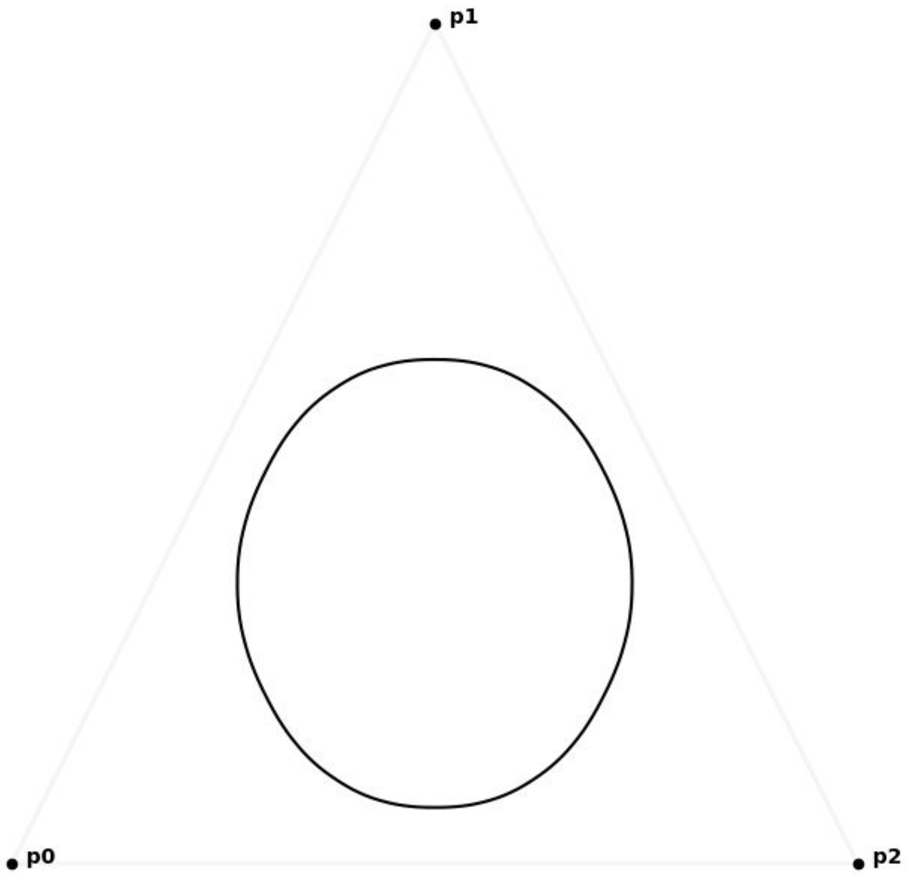




```
# A3.1a Two subdivisions of the triangle with given scheme

import numpy as np
import matplotlib.pyplot as plt

def plot_start_polygon(x, y):
    for i in range(len(x)):
        ax.text(x[i], y[i], "  p" + str(i), weight="bold")
    ax.scatter(x, y, s=20, c="k", zorder=10)
    poly_x = []
    poly_y = []
    for i in range(len(x)):
        poly_x.append(x[i])
        poly_y.append(y[i])
    poly_x.append(x[0])
    poly_y.append(y[0])
    ax.plot(poly_x, poly_y, c="whitesmoke", linewidth=2, zorder=-10)

def plot_polygon(x, y):
    x.append(x[0])
    y.append(y[0])
    ax.plot(x, y)

def triplets(array):
    triplet_list = []
    for i in range(len(array)):
        template = []
        for num in array:
            template.append(num)
        template.append(template[0])
        template.append(template[1])
        triplet_list.append([template[i], template[i + 1], template[i + 2]])
    return triplet_list

def run_subdivision(x, y):
    triplet_x = triplets(x)
    triplet_y = triplets(y)
    subdivision_x = []
    subdivision_y = []
    for i in range(len(triplet_x)):
        x_next = np.dot(subdivision_matrix, triplet_x[i])
        y_next = np.dot(subdivision_matrix, triplet_y[i])
        for j in range(2):
            subdivision_x.append(x_next[j])
            subdivision_y.append(y_next[j])
    return subdivision_x, subdivision_y

subdivision_matrix = np.array([[1/2, 1/2,   0],
                               [1/6, 4/6, 1/6],
                               [  0, 1/2, 1/2]])

gen1_x, gen1_y = run_subdivision(np.array([0, 1, 2]), np.array([0, 1, 0]))
gen2_x, gen2_y = run_subdivision(gen1_x, gen1_y)

fig = plt.gcf()
fig.set_size_inches(8, 8)
ax = fig.add_subplot(111)
plot_start_polygon(np.array([0, 1, 2]), np.array([0, 1, 0]))
plot_polygon(gen1_x, gen1_y)
plot_polygon(gen2_x, gen2_y)

plt.show()
```



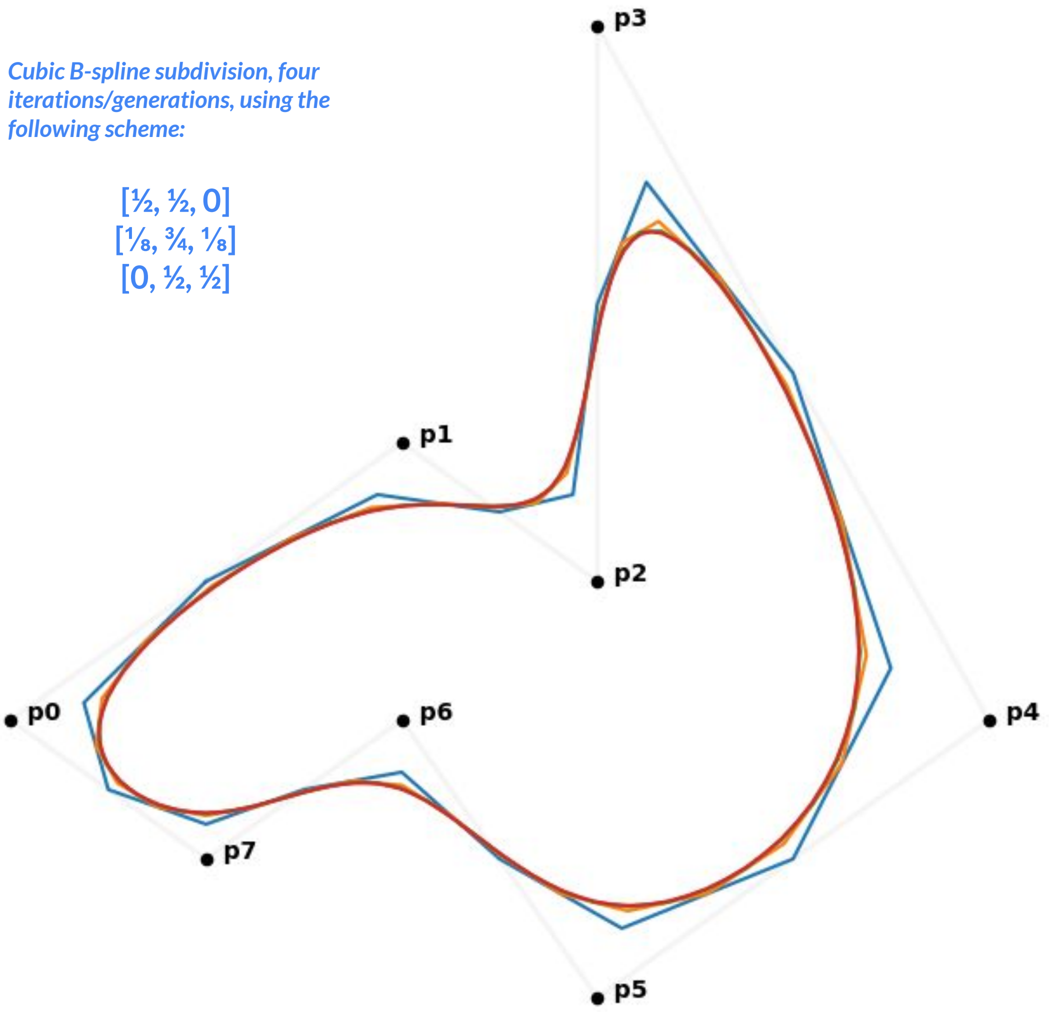
```
# A3.1d Ten subdivisions of the triangle with given scheme

import numpy as np
import matplotlib.pyplot as plt

def plot_start_polygon(x, y):
    for i in range(len(x)):
        ax.text(x[i], y[i], "  p" + str(i), weight="bold")
    ax.scatter(x, y, s=20, c="k", zorder=10)
    poly_x = []
    poly_y = []
    for i in range(len(x)):
        poly_x.append(x[i])
        poly_y.append(y[i])
    poly_x.append(x[0])
    poly_y.append(y[0])
    ax.plot(poly_x, poly_y, c="whitesmoke", linewidth=2, zorder=-10)

def plot_polygon(x, y):
    x.append(x[0])
    y.append(y[0])
    ax.plot(x, y, c="k")

def triplets(array):
    triplet_list = []
    for i in range(len(array)):
        template = []
        for num in array:
            template.append(num)
        template.append(template[0])
        template.append(template[1])
        triplet_list.append([template[i], template[i + 1], template[i + 2]])
    return triplet_list

def run_subdivision(x, y):
    triplet_x = triplets(x)
    triplet_y = triplets(y)
    subdivision_x = []
    subdivision_y = []
    for i in range(len(triplet_x)):
        x_next = np.dot(subdivision_matrix, triplet_x[i])
        y_next = np.dot(subdivision_matrix, triplet_y[i])
        for j in range(2):
            subdivision_x.append(x_next[j])
            subdivision_y.append(y_next[j])
    return subdivision_x, subdivision_y

gen0_x = np.array([0, 1, 2])
gen0_y = np.array([0, 1, 0])

subdivision_matrix = np.array([[1/2, 1/2,   0],
                               [1/6, 4/6, 1/6],
                               [  0, 1/2, 1/2]])
generations_count = 10

fig = plt.gcf()
fig.set_size_inches(8, 8)
ax = fig.add_subplot(111)

subdivision_gens_x = []
subdivision_gens_y = []
last_gen_x = gen0_x
last_gen_y = gen0_y
for gen in range(generations_count):
    last_gen_x, last_gen_y = run_subdivision(last_gen_x, last_gen_y)
    subdivision_gens_x.append(last_gen_x)
    subdivision_gens_y.append(last_gen_y)

plot_start_polygon(gen0_x, gen0_y)
plot_polygon(subdivision_gens_x[-1], subdivision_gens_y[-1])

plt.show()
```



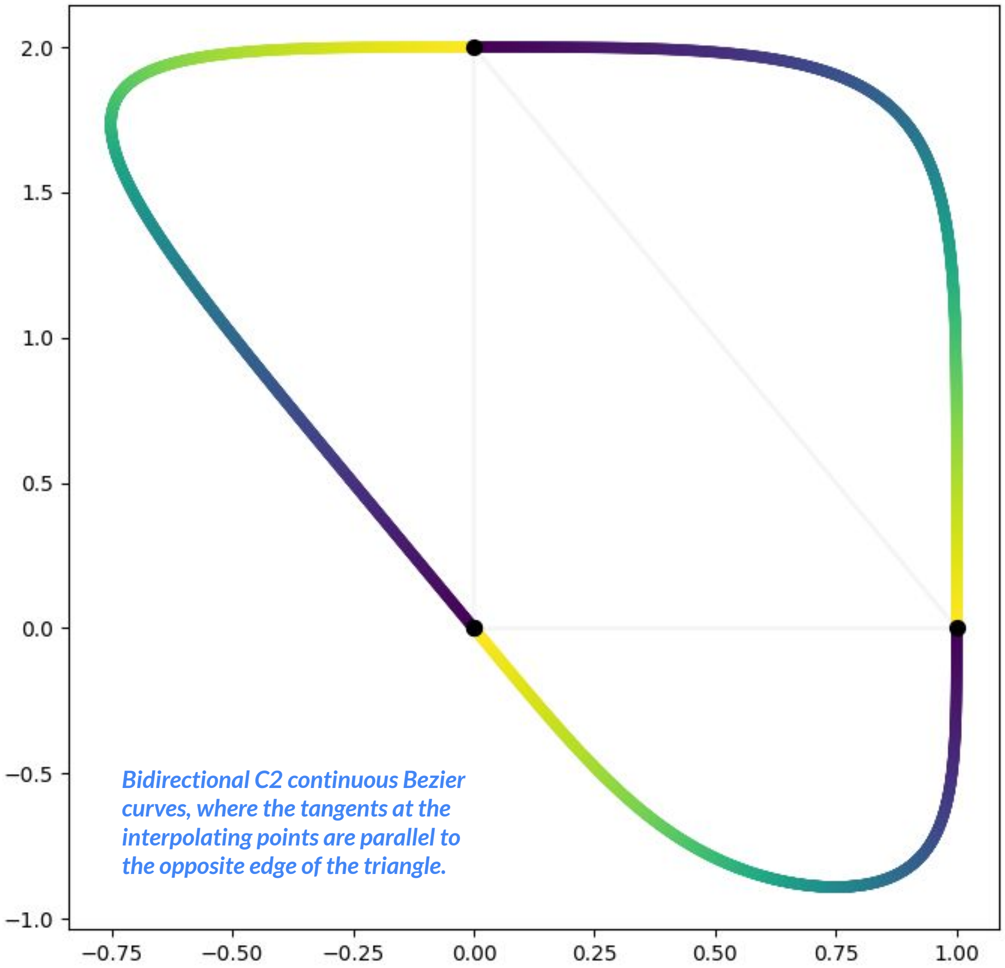
```
# [EXTRA] Cubic B-Spline subdivision for arbitrary polygons, schemes, and generations.

import numpy as np
import matplotlib.pyplot as plt

def plot_start_polygon(x, y):
    for i in range(len(x)):
        ax.text(x[i], y[i], "  p" + str(i), weight="bold")
    ax.scatter(x, y, s=20, c="k", zorder=10)
    poly_x = []
    poly_y = []
    for i in range(len(x)):
        poly_x.append(x[i])
        poly_y.append(y[i])
    poly_x.append(x[0])
    poly_y.append(y[0])
    ax.plot(poly_x, poly_y, c="whitesmoke", linewidth=2, zorder=-10)

def plot_polygon(x, y):
    x.append(x[0])
    y.append(y[0])
    ax.plot(x, y)

def triplets(array):
    triplet_list = []
    for i in range(len(array)):
        template = []
        for num in array:
            template.append(num)
        template.append(template[0])
        template.append(template[1])
        triplet_list.append([template[i], template[i + 1], template[i + 2]])
    return triplet_list

def run_subdivision(x, y):
    triplet_x = triplets(x)
    triplet_y = triplets(y)
    subdivision_x = []
    subdivision_y = []
    for i in range(len(triplet_x)):
        x_next = np.dot(subdivision_matrix, triplet_x[i])
        y_next = np.dot(subdivision_matrix, triplet_y[i])
        for j in range(2):
            subdivision_x.append(x_next[j])
            subdivision_y.append(y_next[j])
    return subdivision_x, subdivision_y

gen0_x = np.array([0, 2, 3, 3, 5, 3, 2, 1])
gen0_y = np.array([0, 2, 1, 5, 0, -2, 0, -1])

subdivision_matrix = np.array([[1/2, 1/2,   0],
                               [1/8, 6/8, 1/8],
                               [0,   1/2, 1/2]])
generations_count = 4

fig = plt.gcf()
fig.set_size_inches(8, 8)
ax = fig.add_subplot(111)
subdivision_gens_x = []
subdivision_gens_y = []
last_gen_x = gen0_x
last_gen_y = gen0_y
for gen in range(generations_count):
    last_gen_x, last_gen_y = run_subdivision(last_gen_x, last_gen_y)
    subdivision_gens_x.append(last_gen_x)
    subdivision_gens_y.append(last_gen_y)

plot_start_polygon(gen0_x, gen0_y)
for i in range(len(subdivision_gens_x)):
    plot_polygon(subdivision_gens_x[i], subdivision_gens_y[i])

plt.axis("off")
plt.show()
```

Cubic B-spline subdivision, four iterations/generations, using the following scheme:

[½, ½, 0]
[⅛, ¾, ⅛]
[0, ½, ½]

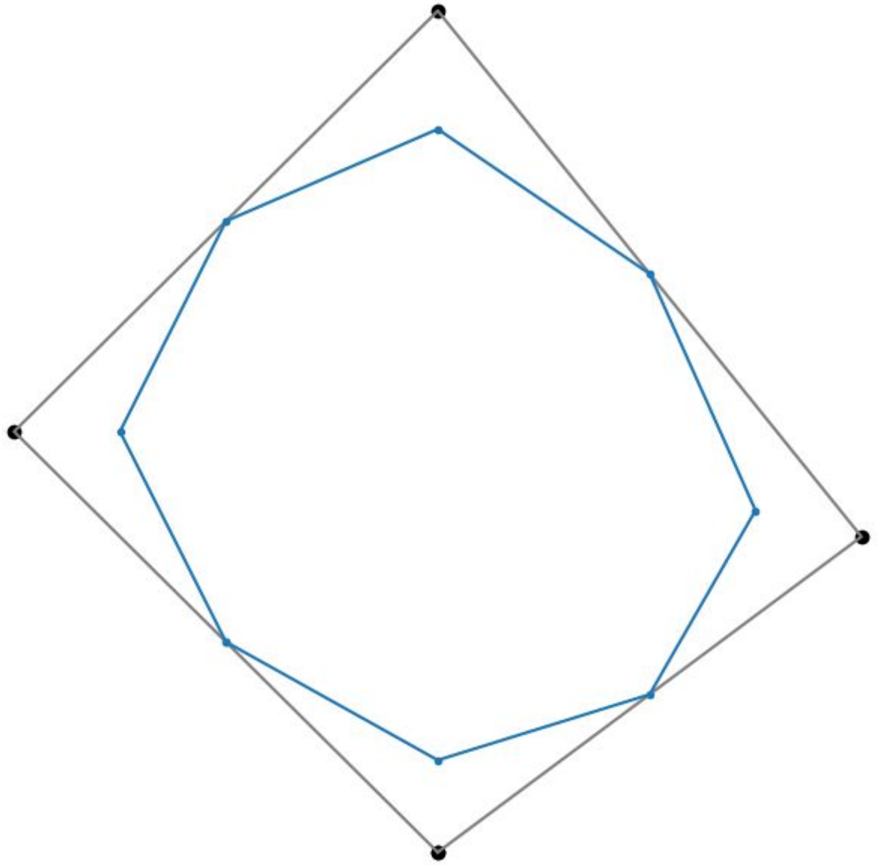


```
# A4.3 - Bonus

import numpy as np
import matplotlib.pyplot as plt

def De_Casteljau(points, t):
    point_array = [point for point in points]
    for i in range(1, len(point_array)):
        for m in range(len(point_array) - i):
            point_array[m] = point_array[m]*(1 - t) + point_array[m + 1]*(t)
    return point_array[0]
def plot_Bezier(x, y):
    interp_x = []
    interp_y = []
    for i in range(0, 1000):
        t = i / 1000
        interp_x.append(De_Casteljau(x, t))
        interp_y.append(De_Casteljau(y, t))
    plt.scatter(
        interp_x,
        interp_y,
        s=20,
        c=range(len(interp_x)),
        cmap="viridis",
    )
def point_on_vector(Pxy_start, Pxy_end, distance):
    v = np.array(Pxy_start, dtype=float)
    u = np.array(Pxy_end, dtype=float)
    n = v - u
    n /= np.linalg.norm(n, 2)
    point = v - distance * n
    return tuple(point)

upper_CP = point_on_vector((0, 0), (-1, 2), 1)
lower_CP = point_on_vector((0, 0), (1, -2), 1)

fig = plt.gcf()
fig.set_size_inches(8, 8)
ax = fig.add_subplot(111)

ax.plot([0, 0, 1, 0], [0, 2, 0, 0], c="whitesmoke", linewidth=2, zorder=-20)
ax.scatter([0, 0, 1, 0], [0, 2, 0, 0], s=50, c="k", zorder=10)

x = [0, 1, 1, 1, 1, 1]
y = [2, 2, 2, 1, 1, 0]
plot_Bezier(x, y)

x = [1, 1, 1, lower_CP[0], lower_CP[0], 0]
y = [0, -1, -1, lower_CP[1], lower_CP[1], 0]
plot_Bezier(x, y)

x = [0, upper_CP[0], upper_CP[0], -1, -1, 0]
y = [0, upper_CP[1], upper_CP[1], 2, 2, 2]
plot_Bezier(x, y)

plt.show()
```

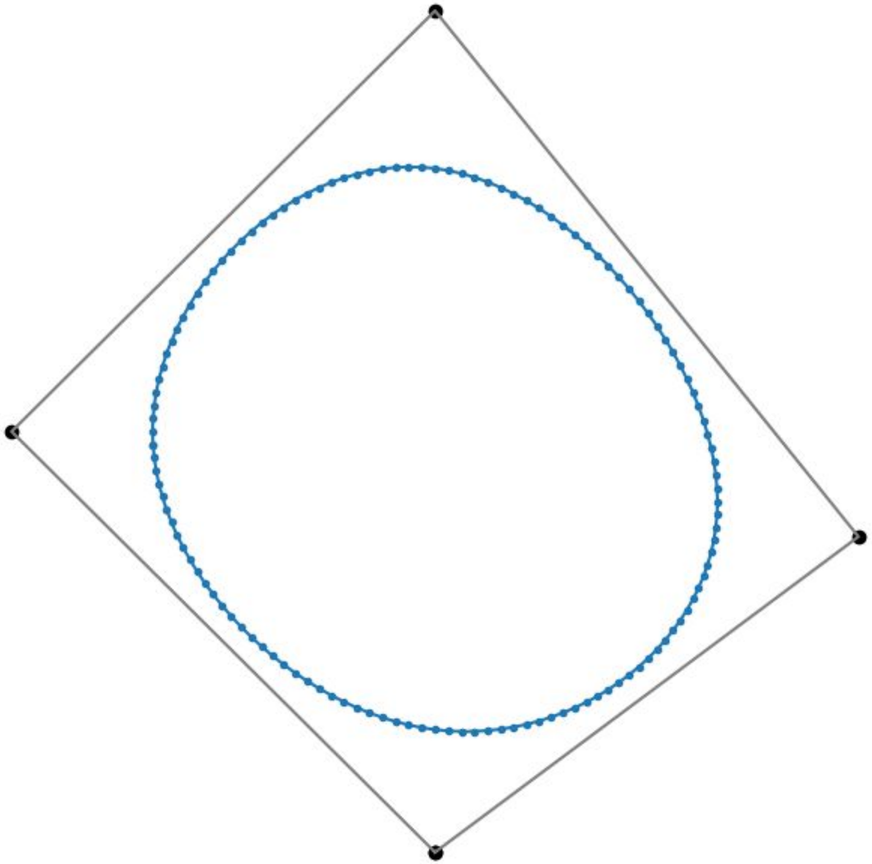


Bidirectional C2 continuous Bezier curves, where the tangents at the interpolating points are parallel to the opposite edge of the triangle.

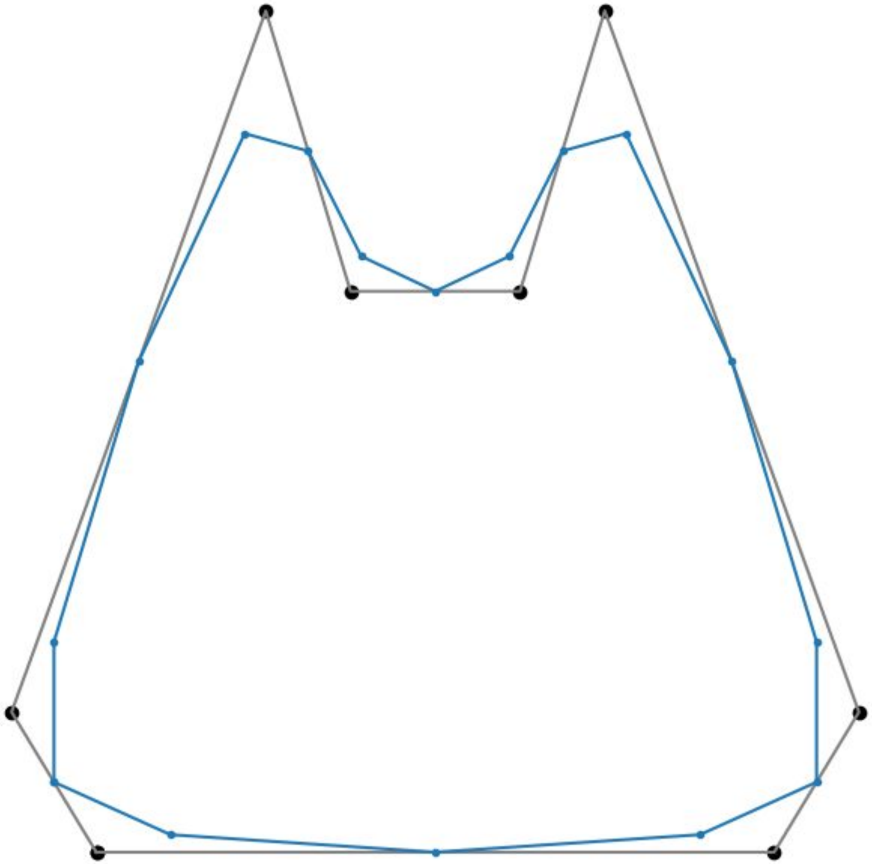
```
# Testing in Python

# A6.1  polygon=1  k=1  scheme=1

import numpy as np
import matplotlib.pyplot as plt

def triplets(array):
    triplet_list = []
    for i in range(len(array)):
        template = []
        for num in array:
            template.append(num)
        template.append(template[0])
        template.append(template[1])
        triplet_list.append([template[i], template[i + 1], template[i + 2]])
    return triplet_list

def run_next(x, y, scheme):
    triplet_x = triplets(x)
    triplet_y = triplets(y)
    subdivision_x = []
    subdivision_y = []
    for i in range(len(triplet_x)):
        x_next = np.dot(scheme, triplet_x[i])
        y_next = np.dot(scheme, triplet_y[i])
        for j in range(2):
            subdivision_x.append(x_next[j])
            subdivision_y.append(y_next[j])
    return subdivision_x, subdivision_y

def subdivide(gen0_x, gen0_y, generations, scheme_choice):
    if scheme_choice == 1:
        scheme = np.array([[1 / 2, 1 / 2, 0], [1 / 8, 3 / 4, 1 / 8], [0, 1 / 2, 1 / 2]])
    elif scheme_choice == 2:
        scheme = np.array([[1 / 2, 1 / 2, 0], [1 / 9, 7 / 9, 1 / 9], [0, 1 / 2, 1 / 2]])
    last_gen_x = gen0_x.tolist()
    last_gen_y = gen0_y.tolist()
    if generations > 0:
        subdivision_gens_x = []
        subdivision_gens_y = []
        for _ in range(int(generations)):
            last_gen_x, last_gen_y = run_next(last_gen_x, last_gen_y, scheme)
            subdivision_gens_x.append(last_gen_x)
            subdivision_gens_y.append(last_gen_y)
        return subdivision_gens_x[-1], subdivision_gens_y[-1]
    else:
        return last_gen_x, last_gen_y

def plot_polygon(start_x, start_y, x, y):
    polygon_x = start_x.tolist()
    polygon_y = start_y.tolist()
    polygon_x.append(polygon_x[0])
    polygon_y.append(polygon_y[0])
    x.append(x[0])
    y.append(y[0])
    fig = plt.gcf()
    fig.set_size_inches(8, 8)
    ax = fig.add_subplot(111)
    ax.plot(polygon_x, polygon_y, "gray")
    ax.scatter(polygon_x, polygon_y, c="k")
    ax.plot(x, y, ".-")
    plt.axis("off")
    plt.show()

x1 = np.array([0, -2, 0, 2])
y1 = np.array([0, 2, 4, 1.5])

x, y = subdivide(x1, y1, 1, 1)
plot_polygon(x1, y1, x, y)
```



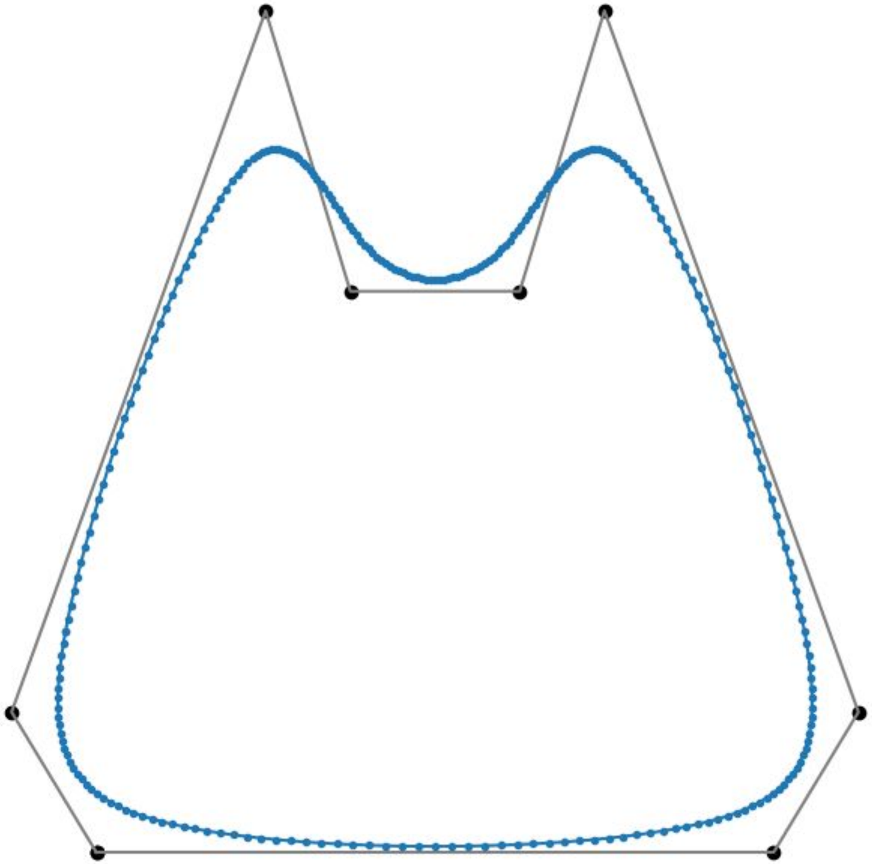
```
# Testing in Python

# A6.1  polygon=1  k=5  scheme=1

import numpy as np
import matplotlib.pyplot as plt

def triplets(array):
    triplet_list = []
    for i in range(len(array)):
        template = []
        for num in array:
            template.append(num)
        template.append(template[0])
        template.append(template[1])
        triplet_list.append([template[i], template[i + 1], template[i + 2]])
    return triplet_list

def run_next(x, y, scheme):
    triplet_x = triplets(x)
    triplet_y = triplets(y)
    subdivision_x = []
    subdivision_y = []
    for i in range(len(triplet_x)):
        x_next = np.dot(scheme, triplet_x[i])
        y_next = np.dot(scheme, triplet_y[i])
        for j in range(2):
            subdivision_x.append(x_next[j])
            subdivision_y.append(y_next[j])
    return subdivision_x, subdivision_y

def subdivide(gen0_x, gen0_y, generations, scheme_choice):
    if scheme_choice == 1:
        scheme = np.array([[1 / 2, 1 / 2, 0], [1 / 8, 3 / 4, 1 / 8], [0, 1 / 2, 1 / 2]])
    elif scheme_choice == 2:
        scheme = np.array([[1 / 2, 1 / 2, 0], [1 / 9, 7 / 9, 1 / 9], [0, 1 / 2, 1 / 2]])
    last_gen_x = gen0_x.tolist()
    last_gen_y = gen0_y.tolist()
    if generations > 0:
        subdivision_gens_x = []
        subdivision_gens_y = []
        for _ in range(int(generations)):
            last_gen_x, last_gen_y = run_next(last_gen_x, last_gen_y, scheme)
            subdivision_gens_x.append(last_gen_x)
            subdivision_gens_y.append(last_gen_y)
        return subdivision_gens_x[-1], subdivision_gens_y[-1]
    else:
        return last_gen_x, last_gen_y

def plot_polygon(start_x, start_y, x, y):
    polygon_x = start_x.tolist()
    polygon_y = start_y.tolist()
    polygon_x.append(polygon_x[0])
    polygon_y.append(polygon_y[0])
    x.append(x[0])
    y.append(y[0])
    fig = plt.gcf()
    fig.set_size_inches(8, 8)
    ax = fig.add_subplot(111)
    ax.plot(polygon_x, polygon_y, "gray")
    ax.scatter(polygon_x, polygon_y, c="k")
    ax.plot(x, y, ".-")
    plt.axis("off")
    plt.show()

x1 = np.array([0, -2, 0, 2])
y1 = np.array([0, 2, 4, 1.5])

x, y = subdivide(x1, y1, 5, 1)
plot_polygon(x1, y1, x, y)
```



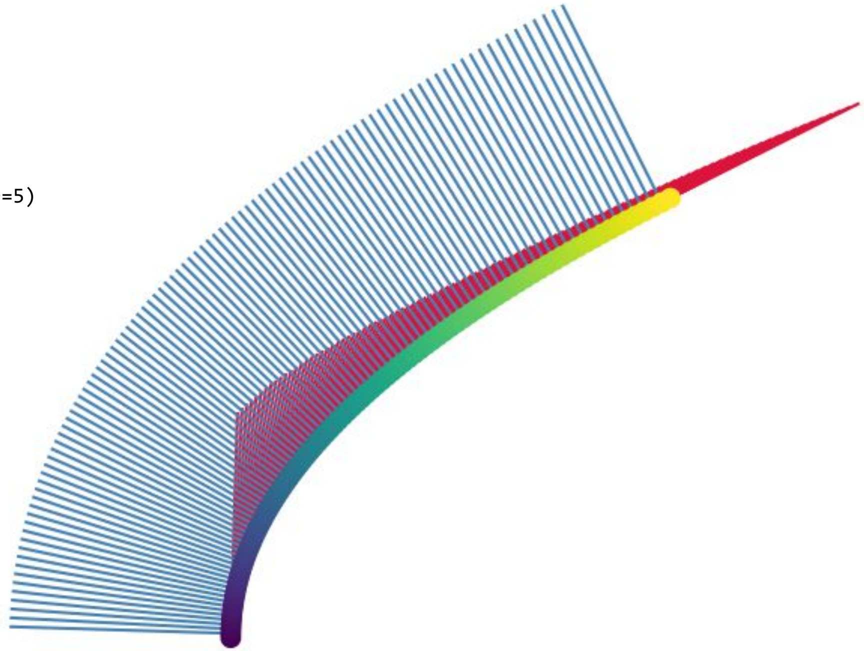
```
# Testing in Python

# A6.1  polygon=2  k=1  scheme=1

import numpy as np
import matplotlib.pyplot as plt

def triplets(array):
    triplet_list = []
    for i in range(len(array)):
        template = []
        for num in array:
            template.append(num)
        template.append(template[0])
        template.append(template[1])
        triplet_list.append([template[i], template[i + 1], template[i + 2]])
    return triplet_list

def run_next(x, y, scheme):
    triplet_x = triplets(x)
    triplet_y = triplets(y)
    subdivision_x = []
    subdivision_y = []
    for i in range(len(triplet_x)):
        x_next = np.dot(scheme, triplet_x[i])
        y_next = np.dot(scheme, triplet_y[i])
        for j in range(2):
            subdivision_x.append(x_next[j])
            subdivision_y.append(y_next[j])
    return subdivision_x, subdivision_y

def subdivide(gen0_x, gen0_y, generations, scheme_choice):
    if scheme_choice == 1:
        scheme = np.array([[1 / 2, 1 / 2, 0], [1 / 8, 3 / 4, 1 / 8], [0, 1 / 2, 1 / 2]])
    elif scheme_choice == 2:
        scheme = np.array([[1 / 2, 1 / 2, 0], [1 / 9, 7 / 9, 1 / 9], [0, 1 / 2, 1 / 2]])
    last_gen_x = gen0_x.tolist()
    last_gen_y = gen0_y.tolist()
    if generations > 0:
        subdivision_gens_x = []
        subdivision_gens_y = []
        for _ in range(int(generations)):
            last_gen_x, last_gen_y = run_next(last_gen_x, last_gen_y, scheme)
            subdivision_gens_x.append(last_gen_x)
            subdivision_gens_y.append(last_gen_y)
        return subdivision_gens_x[-1], subdivision_gens_y[-1]
    else:
        return last_gen_x, last_gen_y

def plot_polygon(start_x, start_y, x, y):
    polygon_x = start_x.tolist()
    polygon_y = start_y.tolist()
    polygon_x.append(polygon_x[0])
    polygon_y.append(polygon_y[0])
    x.append(x[0])
    y.append(y[0])
    fig = plt.gcf()
    fig.set_size_inches(8, 8)
    ax = fig.add_subplot(111)
    ax.plot(polygon_x, polygon_y, "gray")
    ax.scatter(polygon_x, polygon_y, c="k")
    ax.plot(x, y, ".-")
    plt.axis("off")
    plt.show()

x2 = np.array([-2, -2.5, -1, -0.5, 0.5, 1, 2.5, 2])
y2 = np.array([0, 0.5, 3, 2, 2, 3, 0.5, 0])

x, y = subdivide(x2, y2, 1, 1)
plot_polygon(x2, y2, x, y)
```



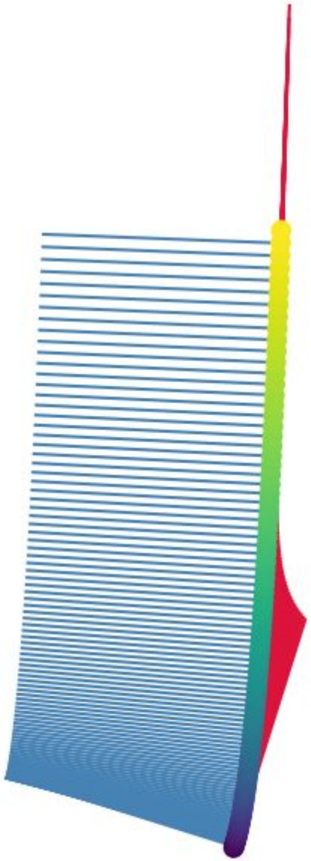
```
# Testing in Python

# A6.1  polygon=2  k=5  scheme=1

import numpy as np
import matplotlib.pyplot as plt

def triplets(array):
    triplet_list = []
    for i in range(len(array)):
        template = []
        for num in array:
            template.append(num)
        template.append(template[0])
        template.append(template[1])
        triplet_list.append([template[i], template[i + 1], template[i + 2]])
    return triplet_list

def run_next(x, y, scheme):
    triplet_x = triplets(x)
    triplet_y = triplets(y)
    subdivision_x = []
    subdivision_y = []
    for i in range(len(triplet_x)):
        x_next = np.dot(scheme, triplet_x[i])
        y_next = np.dot(scheme, triplet_y[i])
        for j in range(2):
            subdivision_x.append(x_next[j])
            subdivision_y.append(y_next[j])
    return subdivision_x, subdivision_y

def subdivide(gen0_x, gen0_y, generations, scheme_choice):
    if scheme_choice == 1:
        scheme = np.array([[1 / 2, 1 / 2, 0], [1 / 8, 3 / 4, 1 / 8], [0, 1 / 2, 1 / 2]])
    elif scheme_choice == 2:
        scheme = np.array([[1 / 2, 1 / 2, 0], [1 / 9, 7 / 9, 1 / 9], [0, 1 / 2, 1 / 2]])
    last_gen_x = gen0_x.tolist()
    last_gen_y = gen0_y.tolist()
    if generations > 0:
        subdivision_gens_x = []
        subdivision_gens_y = []
        for _ in range(int(generations)):
            last_gen_x, last_gen_y = run_next(last_gen_x, last_gen_y, scheme)
            subdivision_gens_x.append(last_gen_x)
            subdivision_gens_y.append(last_gen_y)
        return subdivision_gens_x[-1], subdivision_gens_y[-1]
    else:
        return last_gen_x, last_gen_y

def plot_polygon(start_x, start_y, x, y):
    polygon_x = start_x.tolist()
    polygon_y = start_y.tolist()
    polygon_x.append(polygon_x[0])
    polygon_y.append(polygon_y[0])
    x.append(x[0])
    y.append(y[0])
    fig = plt.gcf()
    fig.set_size_inches(8, 8)
    ax = fig.add_subplot(111)
    ax.plot(polygon_x, polygon_y, "gray")
    ax.scatter(polygon_x, polygon_y, c="k")
    ax.plot(x, y, ".-")
    plt.axis("off")
    plt.show()

x2 = np.array([-2, -2.5, -1, -0.5, 0.5, 1, 2.5, 2])
y2 = np.array([0, 0.5, 3, 2, 2, 3, 0.5, 0])

x, y = subdivide(x2, y2, 5, 1)
plot_polygon(x2, y2, x, y)
```



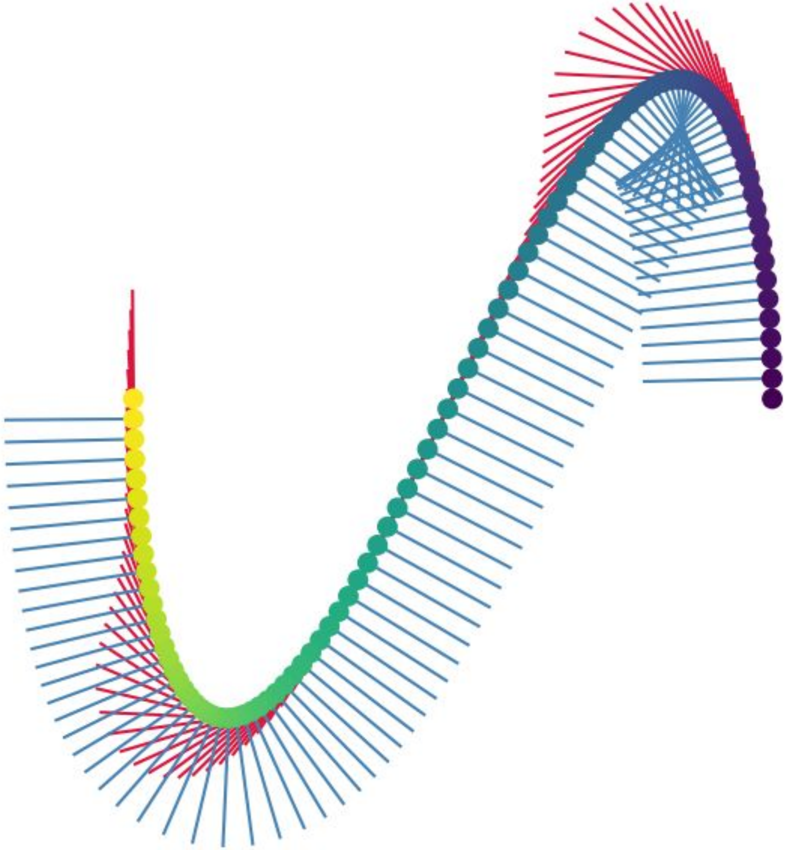
```
# A6.2a - Testing in Python first to check against matlabFunction symbolic approach

import numpy as np
import matplotlib.pyplot as plt

def plot_vectors(x, y, length):
    for idx in range(1, len(x) - 1):
        x0, y0 = x[idx], y[idx]
        x1, y1 = x[idx + 1], y[idx + 1]
        dx = x1 - x0
        dy = y1 - y0
        normalization = np.hypot(dx, dy) / length
        dx /= normalization
        dy /= normalization
        tangent_x, tangent_y = (x0, (x0 + dx)), (y0, (y0 + dy))
        normal_x, normal_y = (x0, (x0 - dy)), (y0, (y0 + dx))
        ax.plot(tangent_x, tangent_y, c="crimson")
        ax.plot(normal_x, normal_y, c="steelblue")

def plot_curve(x, y):
    plt.scatter(x, y, s=80, c=range(len(interp_x)), cmap="viridis", zorder=5)

interp_t = np.linspace(0, 2, 100)

interp_x = []
interp_y = []

for t in interp_t:
    interp_x.append(t ** 2)
    interp_y.append(t * 2)

fig = plt.gcf()
fig.set_size_inches(8, 8)
ax = fig.add_subplot(111)
ax.set_aspect("equal")

plot_curve(interp_x, interp_y)
plot_vectors(interp_x, interp_y, 2)

plt.axis("off")
plt.show()
```

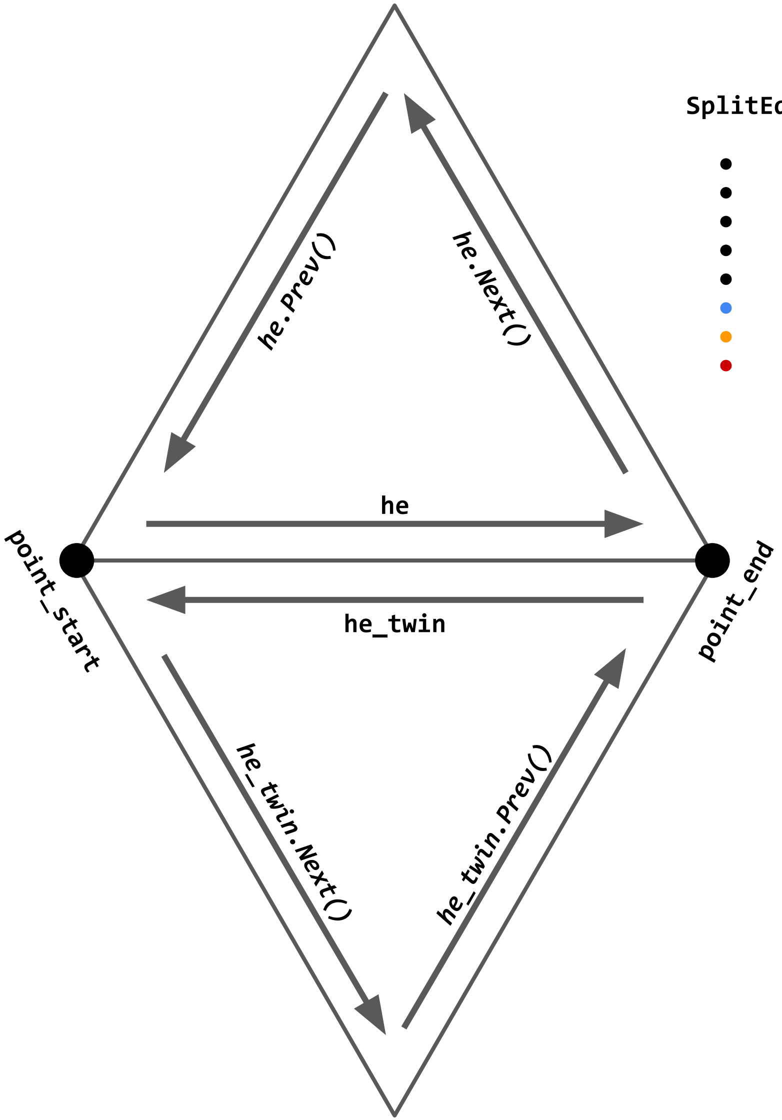


```
# A6.2b - Testing in Python first to check against matlabFunction symbolic approach

import numpy as np
import matplotlib.pyplot as plt

def plot_vectors(x, y, length):
    for idx in range(1, len(x) - 1):
        x0, y0 = x[idx], y[idx]
        x1, y1 = x[idx + 1], y[idx + 1]
        dx = x1 - x0
        dy = y1 - y0
        normalization = np.hypot(dx, dy) / length
        dx /= normalization
        dy /= normalization
        tangent_x, tangent_y = (x0, (x0 + dx)), (y0, (y0 + dy))
        normal_x, normal_y = (x0, (x0 - dy)), (y0, (y0 + dx))
        ax.plot(tangent_x, tangent_y, c="crimson")
        ax.plot(normal_x, normal_y, c="steelblue")

def plot_curve(x, y):
    plt.scatter(x, y, s=80, c=range(len(interp_x)), cmap="viridis", zorder=5)

interp_t = np.linspace(0, 2, 100)

interp_x = []
interp_y = []

for t in interp_t:
    interp_x.append(t - 1)
    interp_y.append((t + 1) ** 3)

fig = plt.gcf()
fig.set_size_inches(8, 8)
ax = fig.add_subplot(111)
ax.set_aspect("equal")

plot_curve(interp_x, interp_y)
plot_vectors(interp_x, interp_y, 2)

plt.axis("off")
plt.show()
```



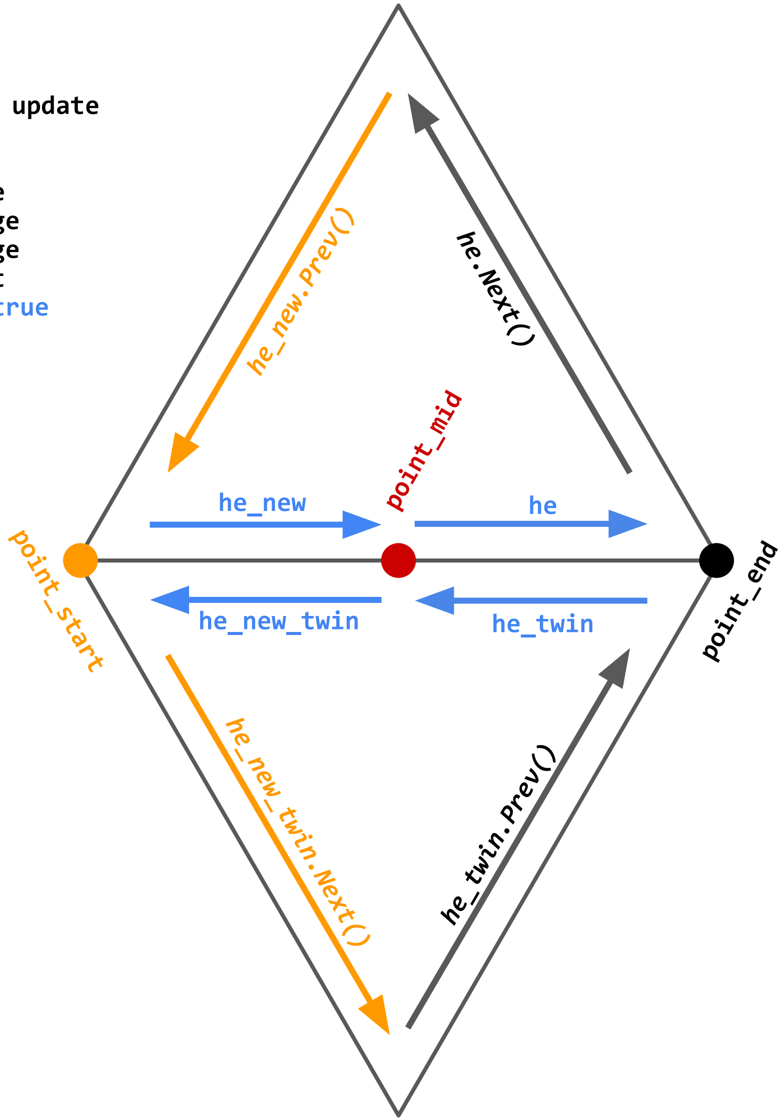
```python
# A6.2c - Testing in Python first to check against matlabFunction symbolic approach

import numpy as np
import matplotlib.pyplot as plt

def plot_vectors(x, y, length):
    for idx in range(1, len(x) - 1):
        x0, y0 = x[idx], y[idx]
        x1, y1 = x[idx + 1], y[idx + 1]
        dx = x1 - x0
        dy = y1 - y0
        normalization = np.hypot(dx, dy) / length
        dx /= normalization
        dy /= normalization
        tangent_x, tangent_y = (x0, (x0 + dx)), (y0, (y0 + dy))
        normal_x, normal_y = (x0, (x0 - dy)), (y0, (y0 + dx))
        ax.plot(tangent_x, tangent_y, c="crimson")
        ax.plot(normal_x, normal_y, c="steelblue")

def plot_curve(x, y):
    plt.scatter(x, y, s=80, c=range(len(interp_x)), cmap="viridis", zorder=5)

interp_t = np.linspace(0, 2, 100)

interp_x = []
interp_y = []

for t in interp_t:
    interp_x.append(np.cos(t * (np.pi / 2)))
    interp_y.append(np.sin(t * np.pi))

fig = plt.gcf()
fig.set_size_inches(8, 8)
ax = fig.add_subplot(111)
ax.set_aspect("equal")

plot_curve(interp_x, interp_y)
plot_vectors(interp_x, interp_y, 2)

plt.axis("off")
plt.show()
```

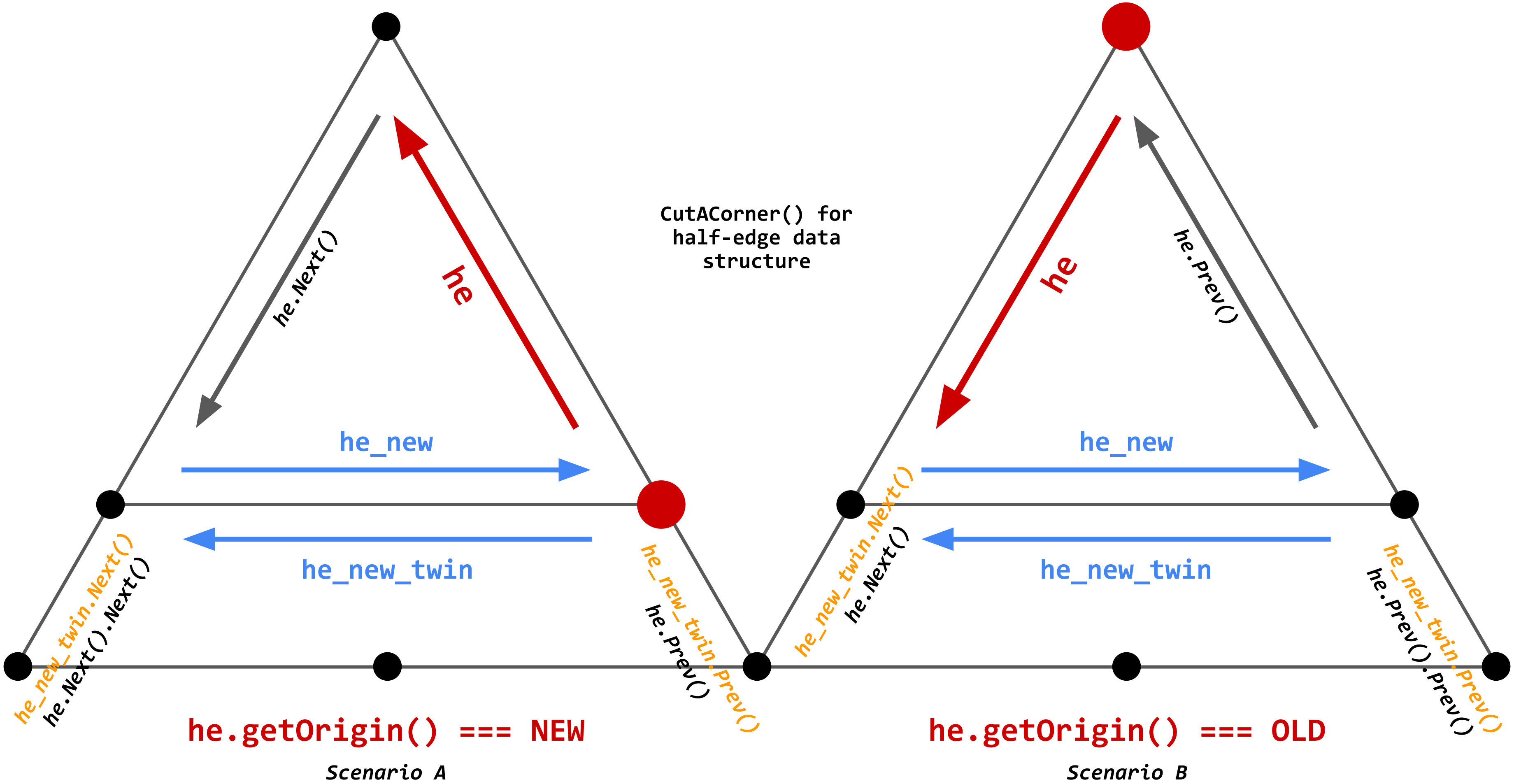




SplitEdge() has to update

- ● Face
- ● Twin halfege
- ● Next halfedge
- ● Prev halfedge
- ● Origin point
- ● Split flag true
- ● Updated
- ● New





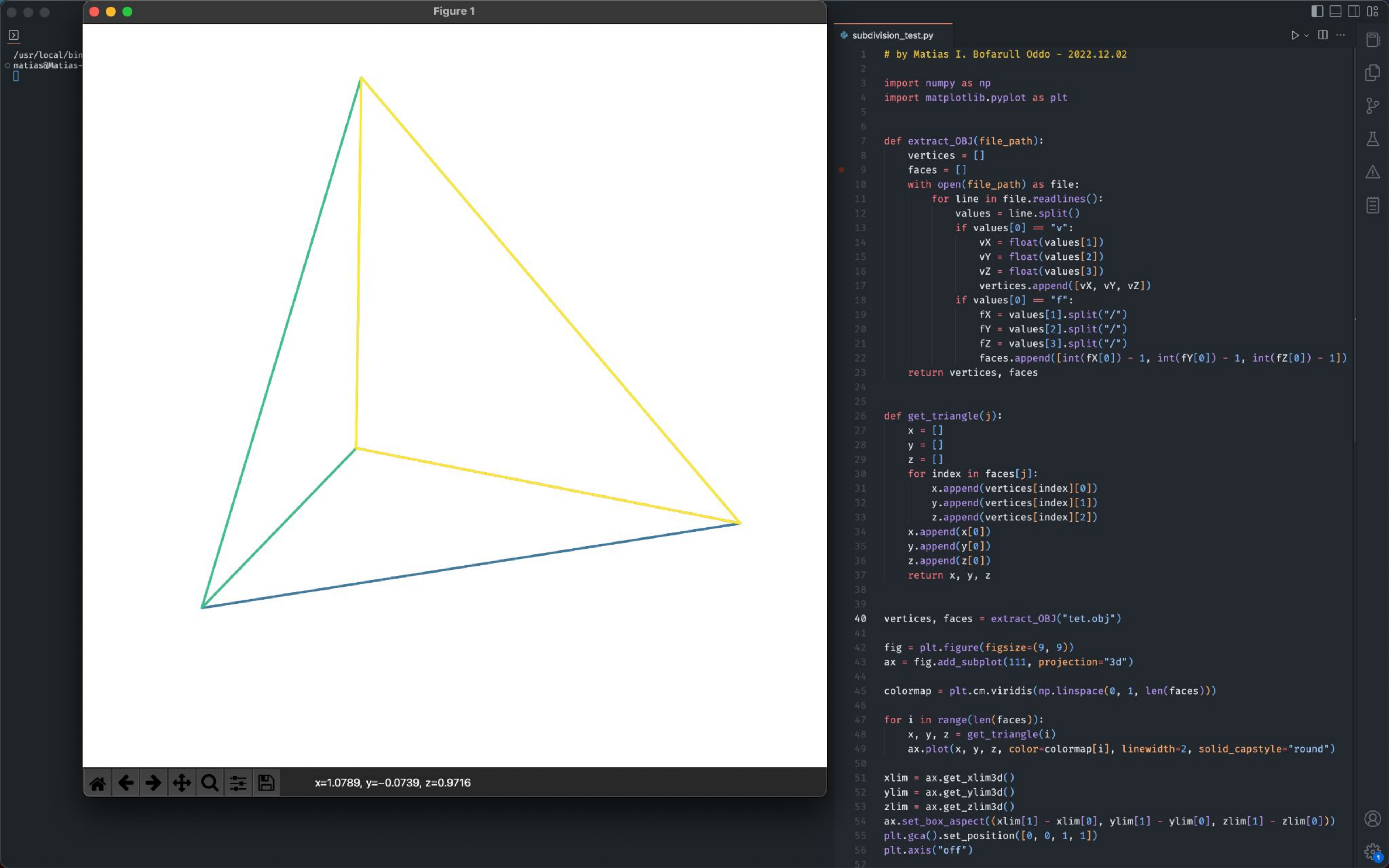
CutACorner() for
half-edge data
structure

he.getOrigin() === NEW

Scenario A

he.getOrigin() === OLD

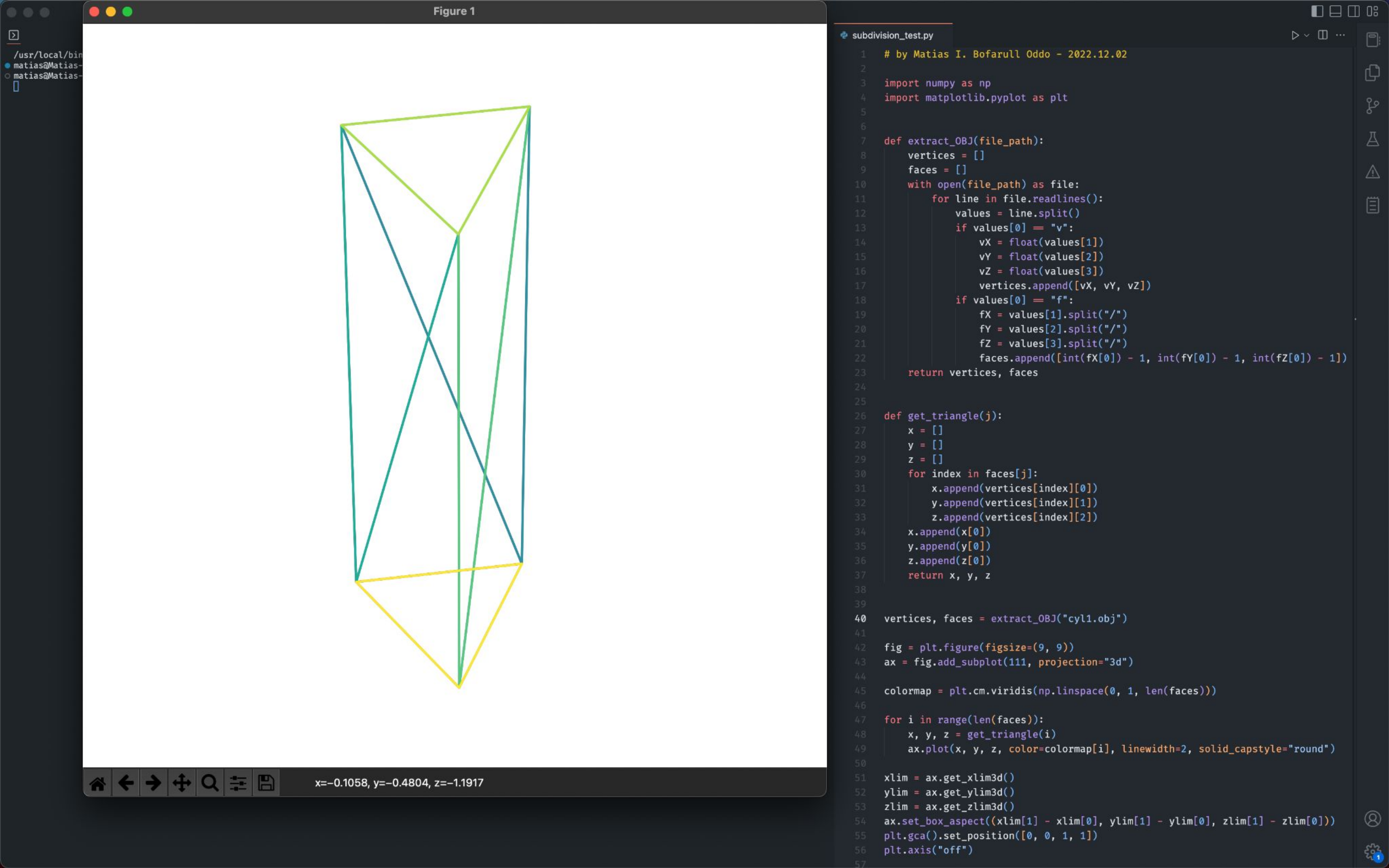
Scenario B



```
/usr/local/bin
matias@Matias-
```

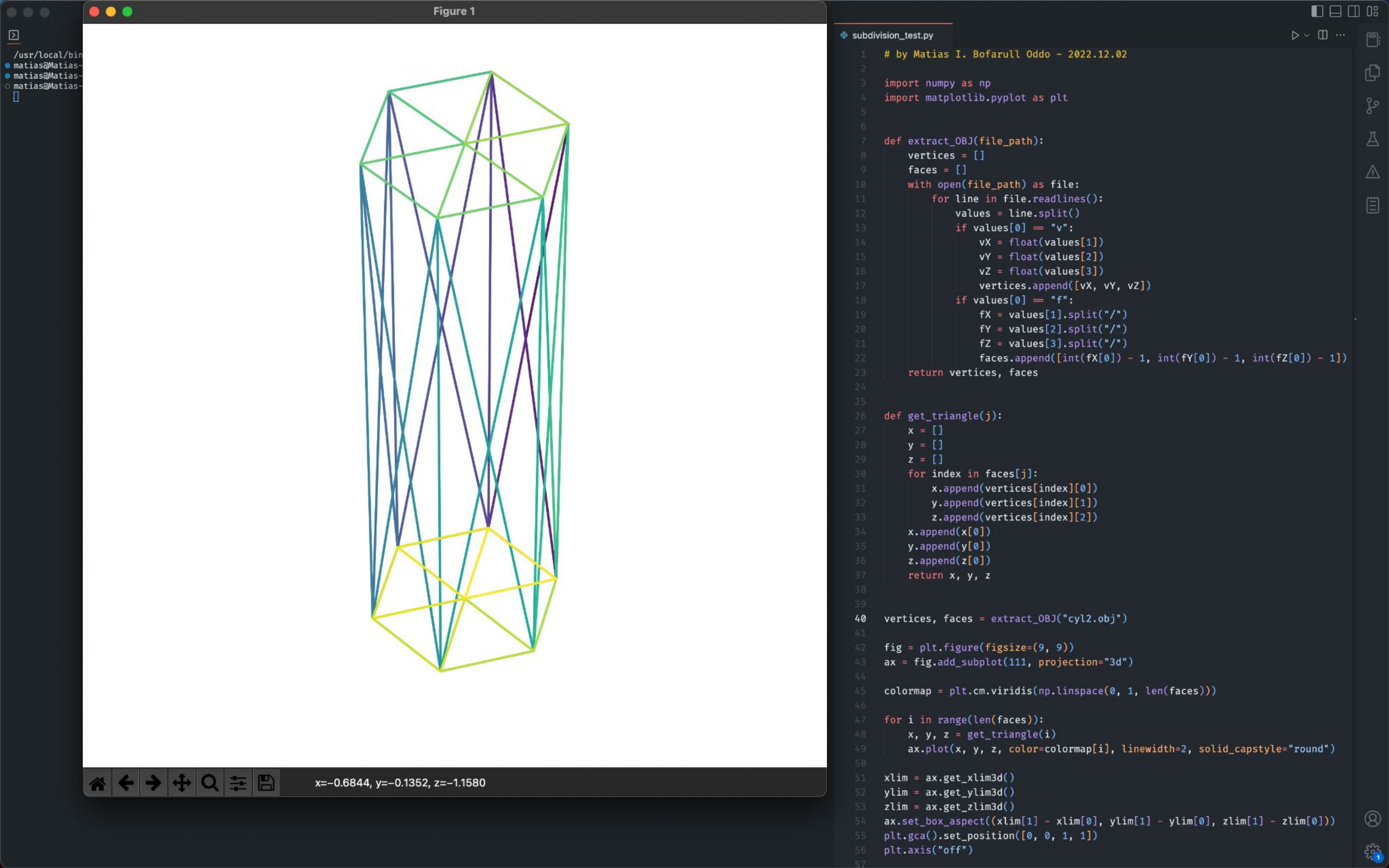
x=1.0789, y=-0.0739, z=0.9716

```python
# by Matias I. Bofarull Oddo - 2022.12.02

import numpy as np
import matplotlib.pyplot as plt


def extract_OBJ(file_path):
    vertices = []
    faces = []
    with open(file_path) as file:
        for line in file.readlines():
            values = line.split()
            if values[0] == "v":
                vX = float(values[1])
                vY = float(values[2])
                vZ = float(values[3])
                vertices.append([vX, vY, vZ])
            if values[0] == "f":
                fX = values[1].split("/")
                fY = values[2].split("/")
                fZ = values[3].split("/")
                faces.append([int(fX[0]) - 1, int(fY[0]) - 1, int(fZ[0]) - 1])
    return vertices, faces


def get_triangle(j):
    x = []
    y = []
    z = []
    for index in faces[j]:
        x.append(vertices[index][0])
        y.append(vertices[index][1])
        z.append(vertices[index][2])
    x.append(x[0])
    y.append(y[0])
    z.append(z[0])
    return x, y, z


vertices, faces = extract_OBJ("tet.obj")

fig = plt.figure(figsize=(9, 9))
ax = fig.add_subplot(111, projection="3d")

colormap = plt.cm.viridis(np.linspace(0, 1, len(faces)))

for i in range(len(faces)):
    x, y, z = get_triangle(i)
    ax.plot(x, y, z, color=colormap[i], linewidth=2, solid_capstyle="round")

xlim = ax.get_xlim3d()
ylim = ax.get_ylim3d()
zlim = ax.get_zlim3d()
ax.set_box_aspect((xlim[1] - xlim[0], ylim[1] - ylim[0], zlim[1] - zlim[0]))
plt.gca().set_position([0, 0, 1, 1])
plt.axis("off")
```



x=−0.1058, y=−0.4804, z=−1.1917

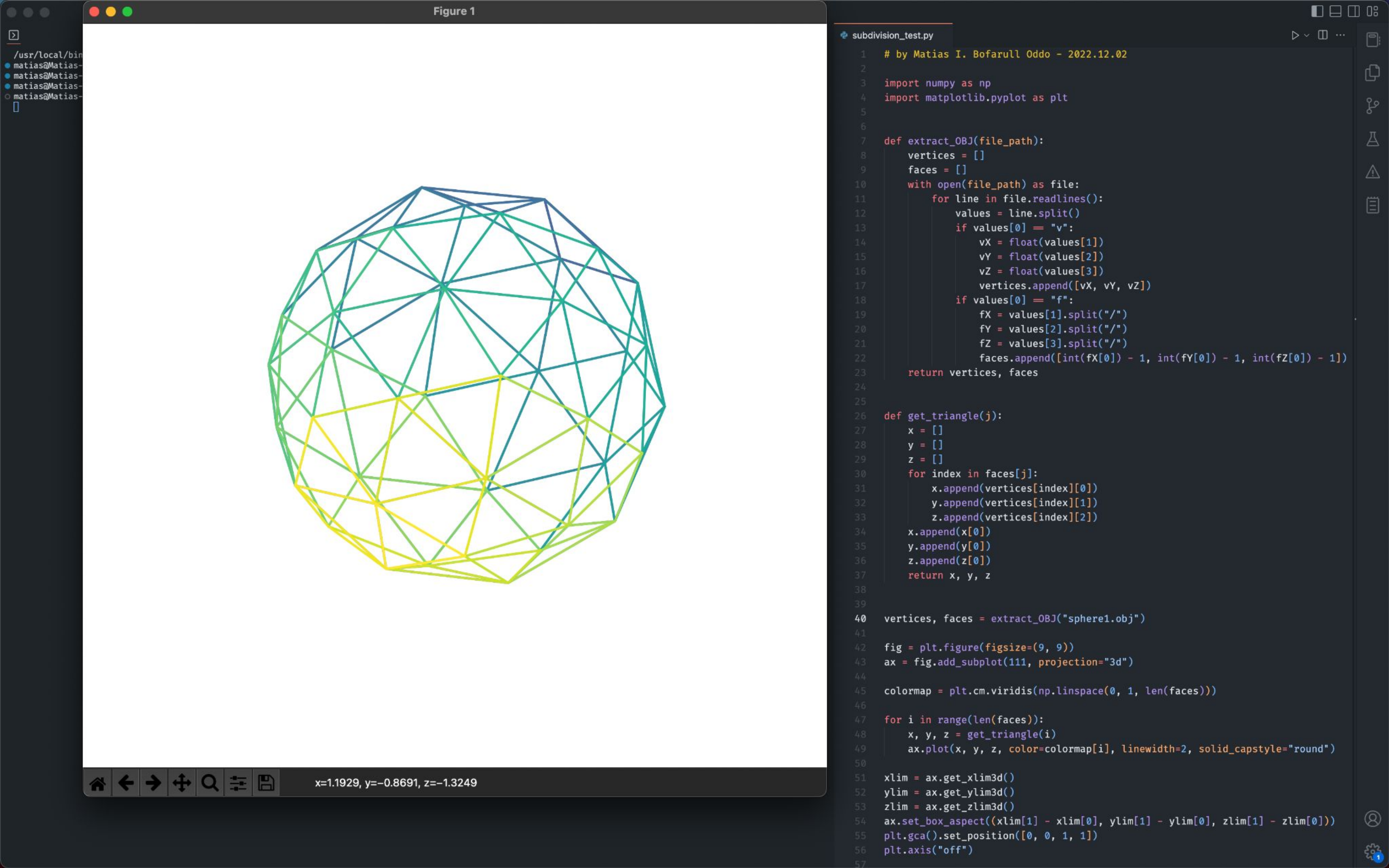
```python
# by Matias I. Bofarull Oddo - 2022.12.02

import numpy as np
import matplotlib.pyplot as plt


def extract_OBJ(file_path):
    vertices = []
    faces = []
    with open(file_path) as file:
        for line in file.readlines():
            values = line.split()
            if values[0] == "v":
                vX = float(values[1])
                vY = float(values[2])
                vZ = float(values[3])
                vertices.append([vX, vY, vZ])
            if values[0] == "f":
                fX = values[1].split("/")
                fY = values[2].split("/")
                fZ = values[3].split("/")
                faces.append([int(fX[0]) - 1, int(fY[0]) - 1, int(fZ[0]) - 1])
    return vertices, faces


def get_triangle(j):
    x = []
    y = []
    z = []
    for index in faces[j]:
        x.append(vertices[index][0])
        y.append(vertices[index][1])
        z.append(vertices[index][2])
    x.append(x[0])
    y.append(y[0])
    z.append(z[0])
    return x, y, z


vertices, faces = extract_OBJ("cyl1.obj")

fig = plt.figure(figsize=(9, 9))
ax = fig.add_subplot(111, projection="3d")

colormap = plt.cm.viridis(np.linspace(0, 1, len(faces)))

for i in range(len(faces)):
    x, y, z = get_triangle(i)
    ax.plot(x, y, z, color=colormap[i], linewidth=2, solid_capstyle="round")

xlim = ax.get_xlim3d()
ylim = ax.get_ylim3d()
zlim = ax.get_zlim3d()
ax.set_box_aspect((xlim[1] - xlim[0], ylim[1] - ylim[0], zlim[1] - zlim[0]))
plt.gca().set_position([0, 0, 1, 1])
plt.axis("off")
```



```
/usr/local/bin
matias@Matias-
matias@Matias-
matias@Matias-
```

x=−0.6844, y=−0.1352, z=−1.1580

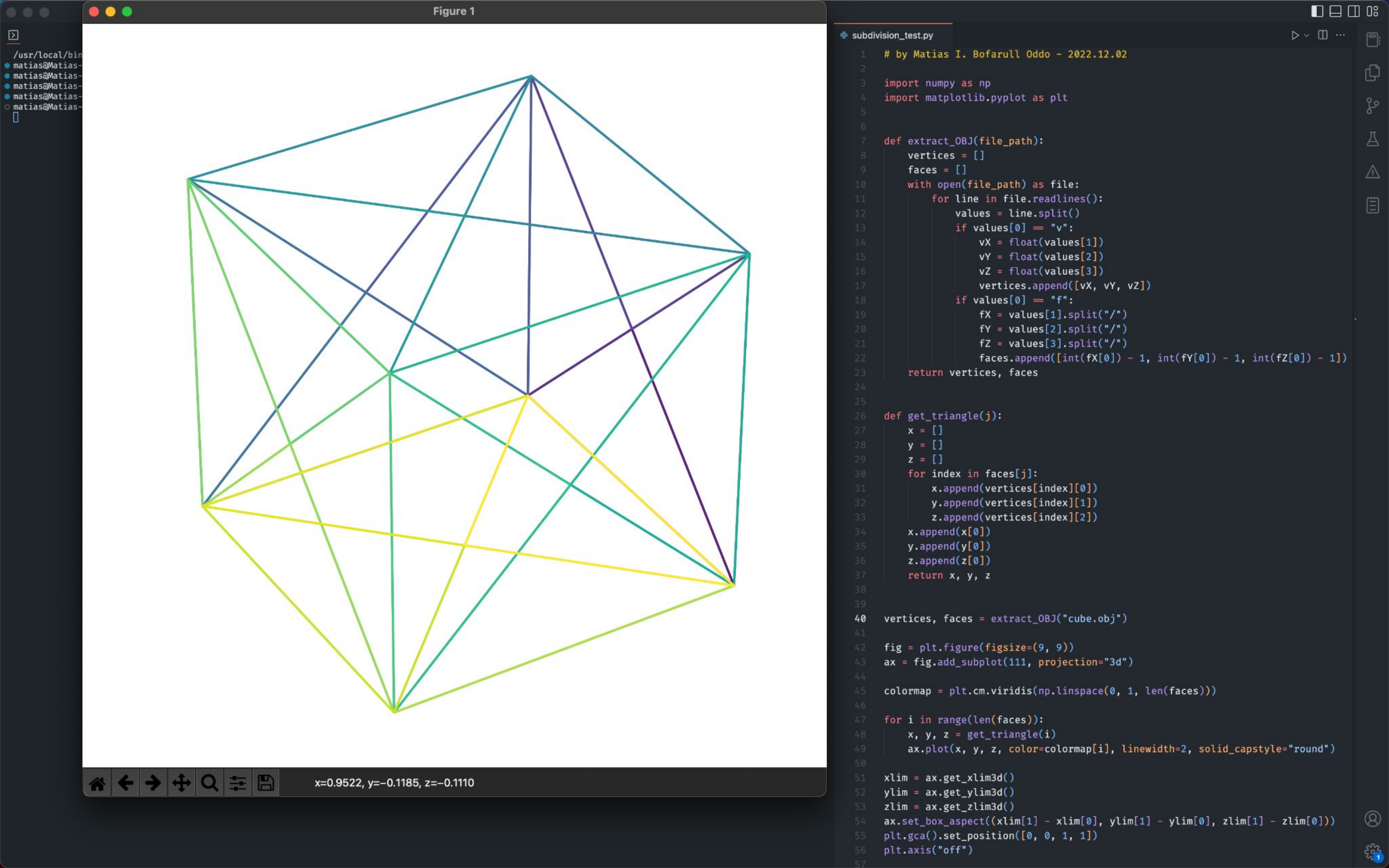
```python
# by Matias I. Bofarull Oddo - 2022.12.02

import numpy as np
import matplotlib.pyplot as plt


def extract_OBJ(file_path):
    vertices = []
    faces = []
    with open(file_path) as file:
        for line in file.readlines():
            values = line.split()
            if values[0] == "v":
                vX = float(values[1])
                vY = float(values[2])
                vZ = float(values[3])
                vertices.append([vX, vY, vZ])
            if values[0] == "f":
                fX = values[1].split("/")
                fY = values[2].split("/")
                fZ = values[3].split("/")
                faces.append([int(fX[0]) - 1, int(fY[0]) - 1, int(fZ[0]) - 1])
    return vertices, faces


def get_triangle(j):
    x = []
    y = []
    z = []
    for index in faces[j]:
        x.append(vertices[index][0])
        y.append(vertices[index][1])
        z.append(vertices[index][2])
    x.append(x[0])
    y.append(y[0])
    z.append(z[0])
    return x, y, z


vertices, faces = extract_OBJ("cyl2.obj")

fig = plt.figure(figsize=(9, 9))
ax = fig.add_subplot(111, projection="3d")

colormap = plt.cm.viridis(np.linspace(0, 1, len(faces)))

for i in range(len(faces)):
    x, y, z = get_triangle(i)
    ax.plot(x, y, z, color=colormap[i], linewidth=2, solid_capstyle="round")

xlim = ax.get_xlim3d()
ylim = ax.get_ylim3d()
zlim = ax.get_zlim3d()
ax.set_box_aspect((xlim[1] - xlim[0], ylim[1] - ylim[0], zlim[1] - zlim[0]))
plt.gca().set_position([0, 0, 1, 1])
plt.axis("off")
```



```
/usr/local/bin
matias@Matias-
matias@Matias-
matias@Matias-
matias@Matias-
```

x=1.1929, y=−0.8691, z=−1.3249

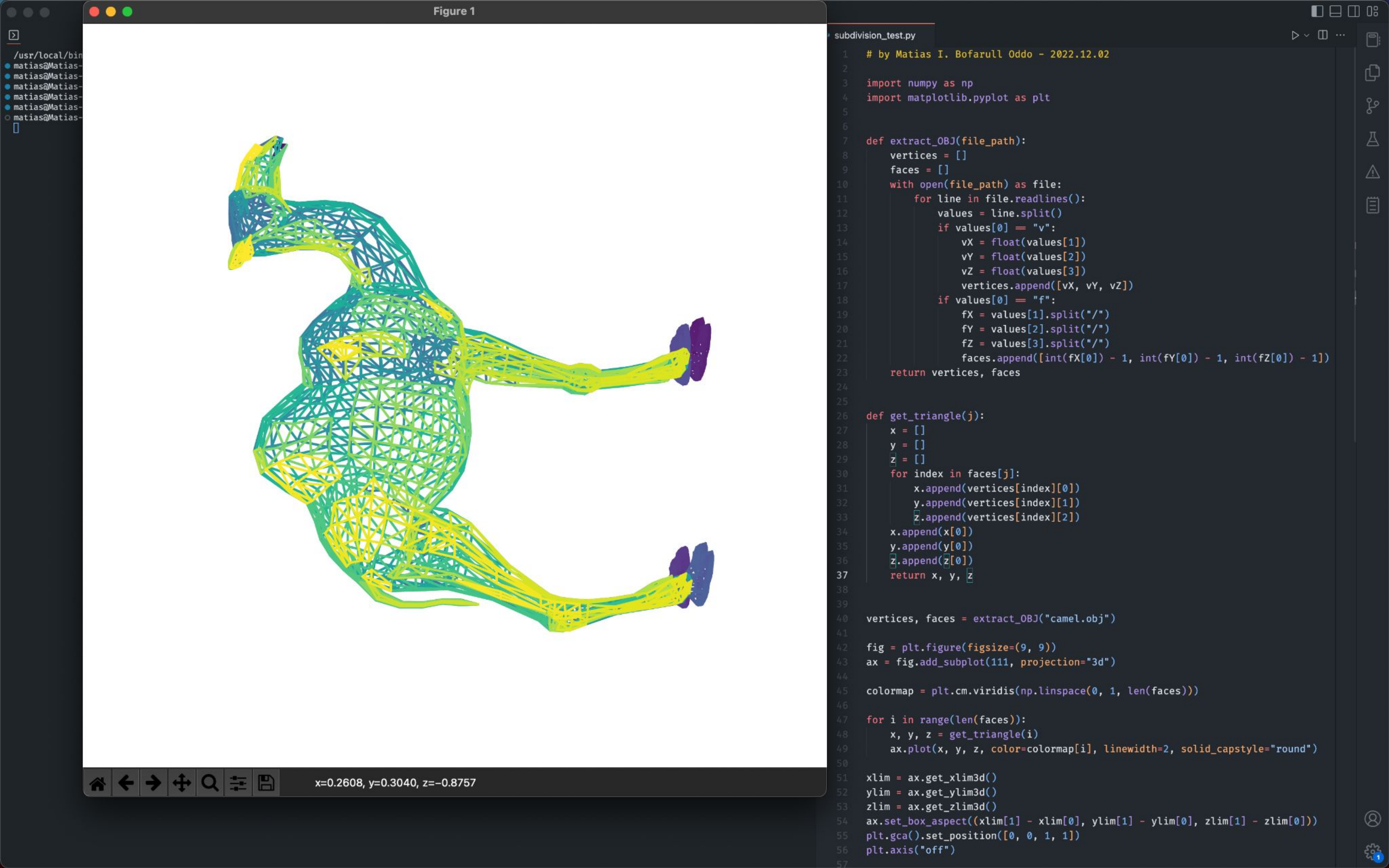
```python
# by Matias I. Bofarull Oddo - 2022.12.02

import numpy as np
import matplotlib.pyplot as plt


def extract_OBJ(file_path):
    vertices = []
    faces = []
    with open(file_path) as file:
        for line in file.readlines():
            values = line.split()
            if values[0] == "v":
                vX = float(values[1])
                vY = float(values[2])
                vZ = float(values[3])
                vertices.append([vX, vY, vZ])
            if values[0] == "f":
                fX = values[1].split("/")
                fY = values[2].split("/")
                fZ = values[3].split("/")
                faces.append([int(fX[0]) - 1, int(fY[0]) - 1, int(fZ[0]) - 1])
    return vertices, faces


def get_triangle(j):
    x = []
    y = []
    z = []
    for index in faces[j]:
        x.append(vertices[index][0])
        y.append(vertices[index][1])
        z.append(vertices[index][2])
    x.append(x[0])
    y.append(y[0])
    z.append(z[0])
    return x, y, z


vertices, faces = extract_OBJ("sphere1.obj")

fig = plt.figure(figsize=(9, 9))
ax = fig.add_subplot(111, projection="3d")

colormap = plt.cm.viridis(np.linspace(0, 1, len(faces)))

for i in range(len(faces)):
    x, y, z = get_triangle(i)
    ax.plot(x, y, z, color=colormap[i], linewidth=2, solid_capstyle="round")

xlim = ax.get_xlim3d()
ylim = ax.get_ylim3d()
zlim = ax.get_zlim3d()
ax.set_box_aspect((xlim[1] - xlim[0], ylim[1] - ylim[0], zlim[1] - zlim[0]))
plt.gca().set_position([0, 0, 1, 1])
plt.axis("off")
```



```
/usr/local/bin
matias@Matias-
matias@Matias-
matias@Matias-
matias@Matias-
matias@Matias-
```

x=0.9522, y=−0.1185, z=−0.1110

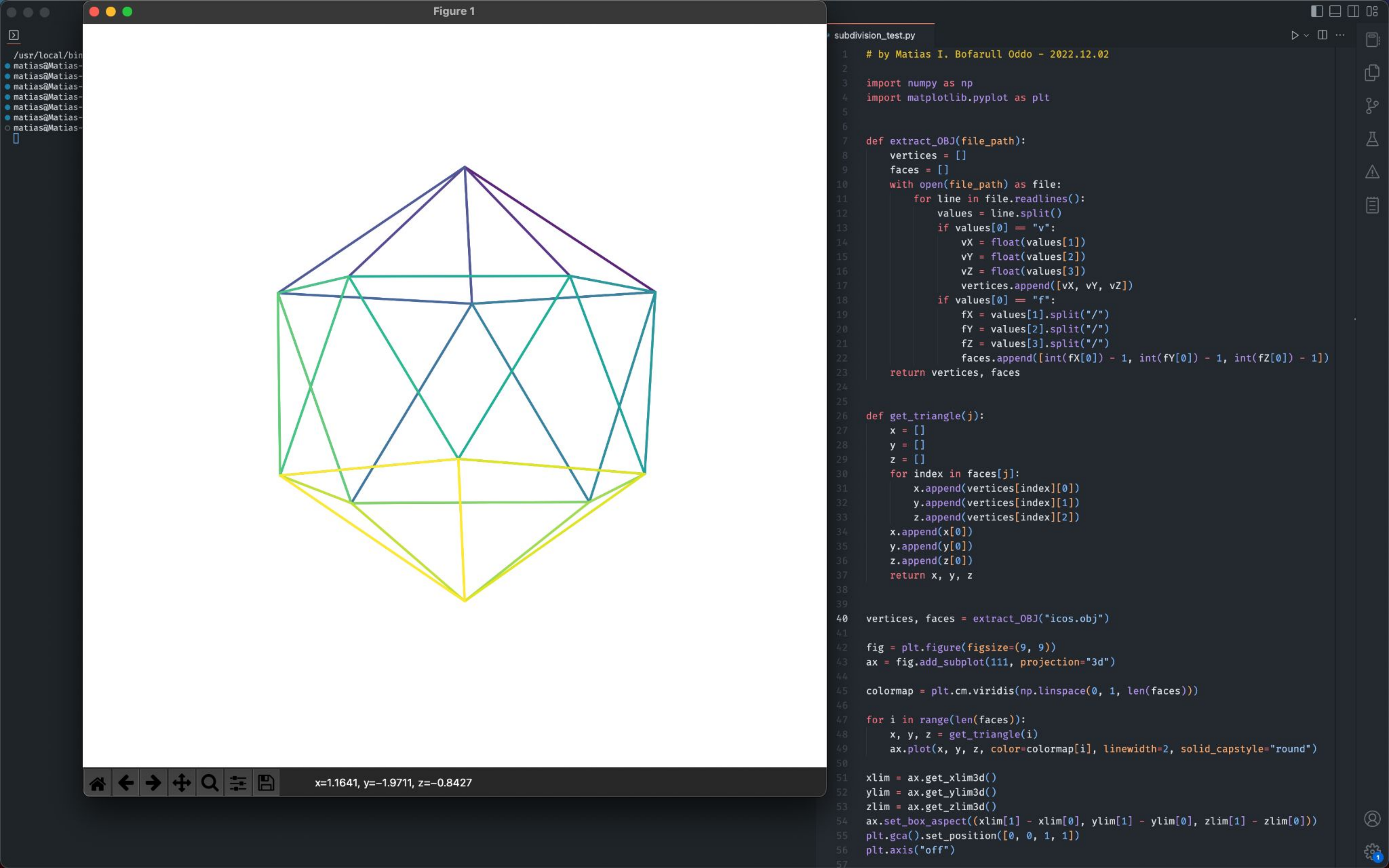
```python
# by Matias I. Bofarull Oddo - 2022.12.02

import numpy as np
import matplotlib.pyplot as plt


def extract_OBJ(file_path):
    vertices = []
    faces = []
    with open(file_path) as file:
        for line in file.readlines():
            values = line.split()
            if values[0] == "v":
                vX = float(values[1])
                vY = float(values[2])
                vZ = float(values[3])
                vertices.append([vX, vY, vZ])
            if values[0] == "f":
                fX = values[1].split("/")
                fY = values[2].split("/")
                fZ = values[3].split("/")
                faces.append([int(fX[0]) - 1, int(fY[0]) - 1, int(fZ[0]) - 1])
    return vertices, faces


def get_triangle(j):
    x = []
    y = []
    z = []
    for index in faces[j]:
        x.append(vertices[index][0])
        y.append(vertices[index][1])
        z.append(vertices[index][2])
    x.append(x[0])
    y.append(y[0])
    z.append(z[0])
    return x, y, z


vertices, faces = extract_OBJ("cube.obj")

fig = plt.figure(figsize=(9, 9))
ax = fig.add_subplot(111, projection="3d")

colormap = plt.cm.viridis(np.linspace(0, 1, len(faces)))

for i in range(len(faces)):
    x, y, z = get_triangle(i)
    ax.plot(x, y, z, color=colormap[i], linewidth=2, solid_capstyle="round")

xlim = ax.get_xlim3d()
ylim = ax.get_ylim3d()
zlim = ax.get_zlim3d()
ax.set_box_aspect((xlim[1] - xlim[0], ylim[1] - ylim[0], zlim[1] - zlim[0]))
plt.gca().set_position([0, 0, 1, 1])
plt.axis("off")
```



```
/usr/local/bin
matias@Matias-
matias@Matias-
matias@Matias-
matias@Matias-
matias@Matias-
matias@Matias-
```

x=0.2608, y=0.3040, z=−0.8757

```python
# by Matias I. Bofarull Oddo - 2022.12.02

import numpy as np
import matplotlib.pyplot as plt


def extract_OBJ(file_path):
    vertices = []
    faces = []
    with open(file_path) as file:
        for line in file.readlines():
            values = line.split()
            if values[0] == "v":
                vX = float(values[1])
                vY = float(values[2])
                vZ = float(values[3])
                vertices.append([vX, vY, vZ])
            if values[0] == "f":
                fX = values[1].split("/")
                fY = values[2].split("/")
                fZ = values[3].split("/")
                faces.append([int(fX[0]) - 1, int(fY[0]) - 1, int(fZ[0]) - 1])
    return vertices, faces


def get_triangle(j):
    x = []
    y = []
    z = []
    for index in faces[j]:
        x.append(vertices[index][0])
        y.append(vertices[index][1])
        z.append(vertices[index][2])
    x.append(x[0])
    y.append(y[0])
    z.append(z[0])
    return x, y, z


vertices, faces = extract_OBJ("camel.obj")

fig = plt.figure(figsize=(9, 9))
ax = fig.add_subplot(111, projection="3d")

colormap = plt.cm.viridis(np.linspace(0, 1, len(faces)))

for i in range(len(faces)):
    x, y, z = get_triangle(i)
    ax.plot(x, y, z, color=colormap[i], linewidth=2, solid_capstyle="round")

xlim = ax.get_xlim3d()
ylim = ax.get_ylim3d()
zlim = ax.get_zlim3d()
ax.set_box_aspect((xlim[1] - xlim[0], ylim[1] - ylim[0], zlim[1] - zlim[0]))
plt.gca().set_position([0, 0, 1, 1])
plt.axis("off")
```



```
/usr/local/bin
matias@Matias-
matias@Matias-
matias@Matias-
matias@Matias-
matias@Matias-
matias@Matias-
matias@Matias-
```

x=1.1641, y=−1.9711, z=−0.8427

```python
# by Matias I. Bofarull Oddo - 2022.12.02

import numpy as np
import matplotlib.pyplot as plt


def extract_OBJ(file_path):
    vertices = []
    faces = []
    with open(file_path) as file:
        for line in file.readlines():
            values = line.split()
            if values[0] == "v":
                vX = float(values[1])
                vY = float(values[2])
                vZ = float(values[3])
                vertices.append([vX, vY, vZ])
            if values[0] == "f":
                fX = values[1].split("/")
                fY = values[2].split("/")
                fZ = values[3].split("/")
                faces.append([int(fX[0]) - 1, int(fY[0]) - 1, int(fZ[0]) - 1])
    return vertices, faces


def get_triangle(j):
    x = []
    y = []
    z = []
    for index in faces[j]:
        x.append(vertices[index][0])
        y.append(vertices[index][1])
        z.append(vertices[index][2])
    x.append(x[0])
    y.append(y[0])
    z.append(z[0])
    return x, y, z


vertices, faces = extract_OBJ("icos.obj")

fig = plt.figure(figsize=(9, 9))
ax = fig.add_subplot(111, projection="3d")

colormap = plt.cm.viridis(np.linspace(0, 1, len(faces)))

for i in range(len(faces)):
    x, y, z = get_triangle(i)
    ax.plot(x, y, z, color=colormap[i], linewidth=2, solid_capstyle="round")

xlim = ax.get_xlim3d()
ylim = ax.get_ylim3d()
zlim = ax.get_zlim3d()
ax.set_box_aspect((xlim[1] - xlim[0], ylim[1] - ylim[0], zlim[1] - zlim[0]))
plt.gca().set_position([0, 0, 1, 1])
plt.axis("off")
```