PROJCET REPORT OF NUMERICAL COMPUTING
PROJECT NAME: CENTER OF MASS

WHAT IS CENTER OF MASS?

- The ***center of mass*** is a position defined relative to an object or system of objects. It is the ***average position*** of all the parts of the system, weighted according to their **masses**.

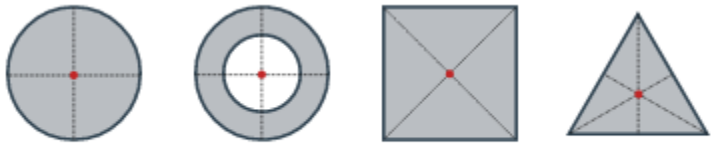


Figure 1: Center of mass for some simple geometric shapes (red dots).

GENERAL METHAMETICAL DEFINITION OF CENTER OF MASS:

"It is the unique position at which the weighted position vectors of all the parts of a system sum up to zero."

DIFFERENCE BETWEEN CENTER OF MASS AND CENTER OF GRAVITY:

Center of mass is the point at which the distribution of mass is equal in all directions, and does not depend on gravitational field.
Center of gravity is the point at which distribution of weight is equal in all directions, and does depend on gravitational field.
The center of mass and the center of gravity of an object are in the same position if the gravitational field in which the object exists is uniform.

WHY CENTER OF MASS IS USEFUL?

- For simple rigid objects with uniform density, the center of mass is located at the **centroid**.

FOR EXAMPLE:

- ✓ The center of mass of a uniform disc shape would be at its center. Sometimes the center of mass doesn't fall anywhere on the object.

OR

- ✓ The center of mass of a ring for example is located at its center, where there isn't any material.

CENTER OF MASS IS USEFUL FOR SOLVING MECHANICS PROBLEMS:

- ➢ The interesting thing about the center of mass of an object or system is that it is the point where any ***uniform force*** on the object acts. This is useful because it makes it easy to solve ***mechanics problems*** where we have to describe the **motion of oddly-shaped objects and complicated systems**.
- ➢ For the purposes of calculation, we can treat an oddly-shaped object as if all its mass is concentrated in a tiny object located at the center of mass. We sometimes call this imaginary object a ***point mass***.

HOW CAN WE FIND THE CENTER OF MASS OF ANY OBJECT OR SYSTEM?

- ➢ In general the center of mass can be found by **vector addition** of the weighted position vectors which point to the center of mass of each object in a system. One quick technique which lets us avoid the use of

vector arithmetic is finding the center of mass separately for components along each axis. i.e:

- For object positions along the x axis:

$$\mathrm{COM}_x = \frac{m_1 \cdot x_1 + m_2 \cdot x_2 + m_3 \cdot x_3 + \ldots}{m_1 + m_2 + m_3 + \ldots}$$

And similarly for the y axis:

$$\mathrm{COM}_y = \frac{m_1 \cdot y_1 + m_2 \cdot y_2 + m_3 \cdot y_3 + \ldots}{m_1 + m_2 + m_3 + \ldots}$$

IRRGULAR SHAPES:

CODE OF CENTRE OF MASS OF REGULAR AND IRREGULAR SHAPES

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import pylab
import pylab as plt
from skimage.color import label2rgb
import imageio as iio
from skimage import filters
from skimage.color import rgb2gray  # only needed for incorrectly saved images
from skimage.measure import regionprops
from matplotlib.figure import Figure
from mpl_toolkits.mplot3d import Axes3D
from math import *
```

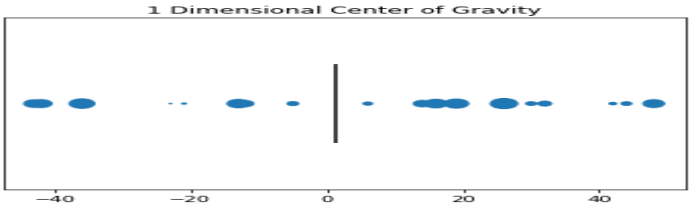
```
%matplotlib inline
n=20
x = np.random.randint(-50, 50, n)
m = np.random.randint(1,200, n)
y = np.random.randint(0,200,n)
z = np.random.randint(0,30,n)
y1 = np.zeros(len(m))
```

Centre of Mass of 1D Object

$$\bar{x} = \frac{\sum_{i=0}^{n} x_i * m_i}{\sum_{i=0}^{n} m_i}$$

```
cgx = np.sum(x*m)/np.sum(m)
print('The center of mass in x is %f' % cgx)
plt.scatter(x,y1,s=m);
plt.scatter(cgx, 0, color='k', marker='|', s=1e4);
plt.gca().set_yticks([])  ;
plt.title('1 Dimensional Center of Gravity');
```

```
The center of mass in x is 1.224888
```



Center Of Mass Of 2D Object

$$\bar{x} = \frac{\sum_{i=0}^{n} x_i * m_i}{\sum_{i=0}^{n} m_i}$$
$$\bar{y} = \frac{\sum_{i=0}^{n} y_i * m_i}{\sum_{i=0}^{n} m_i}$$

```
cgx = np.sum(x*m)/np.sum(m)
cgy = np.sum(y*m)/np.sum(m)
print('The center of mass in x is %f' % cgx)
print('The center of mass in y is %f' % cgy)
plt.plot(x,y, color='green', linestyle='dashed', linewidth = 3,marker='o', markerfacecolor='blue', markersize=12)
plt.xlabel('x - axis')
plt.ylabel('y - axis')
plt.title('Centre of Mass')
plt.show()
plt.plot(cgx, cgy, color='green', linestyle='dashed', linewidth = 3,marker='o', markerfacecolor='red', markersize=12)
plt.xlabel('x - axis')
plt.ylabel('y - axis')
plt.title('Centre of Mass')
plt.show()
```

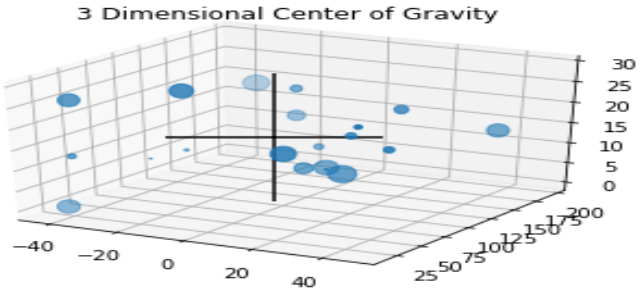
```
The center of mass in x is 1.224888
The center of mass in y is 86.679910
```

CENTRE OF MASS OF 3D OBJECT

```
z = np.random.randint(0,30,n)
cgz = np.sum(z*m)/np.sum(m)
print('The center of mass is %f' % cgz)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x, y, z, s=m);

ax.scatter(cgx, cgy, cgz, color='k', marker='+', s=1e4);
plt.title('3 Dimensional Center of Gravity');
```

```
The center of mass is 16.123688
```



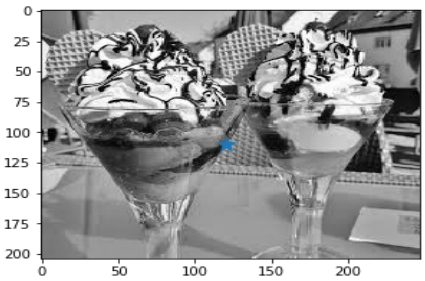
CENTRE OF MASS OF IMAGE

CONSIDER IMAGE AS RECTANGLE

```
image = rgb2gray(iio.imread('download.jpg'))
threshold_value = filters.threshold_otsu(image)
labeled_foreground = (image > threshold_value).astype(int)
properties = regionprops(labeled_foreground, image)
center_of_mass = properties[0].centroid
weighted_center_of_mass = properties[0].weighted_centroid

print(center_of_mass)
colorized = label2rgb(labeled_foreground, image, colors=['black', 'yellow'], alpha=0.0)
fig, ax = plt.subplots()
ax.imshow(colorized)
# Note the inverted coordinates because plt uses (x, y) while NumPy uses (row, column)
ax.scatter(center_of_mass[1], center_of_mass[0], s=160, c='C0', marker='*')
plt.show()
```

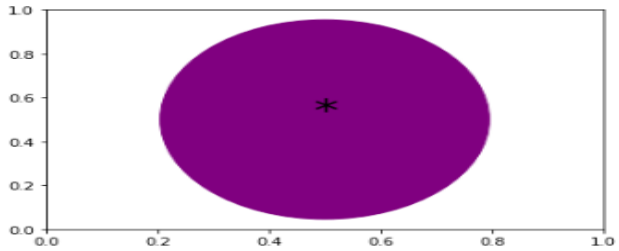
```
(109.75668269768482, 120.37772620512247)
```



Centre of Mass of a Circle

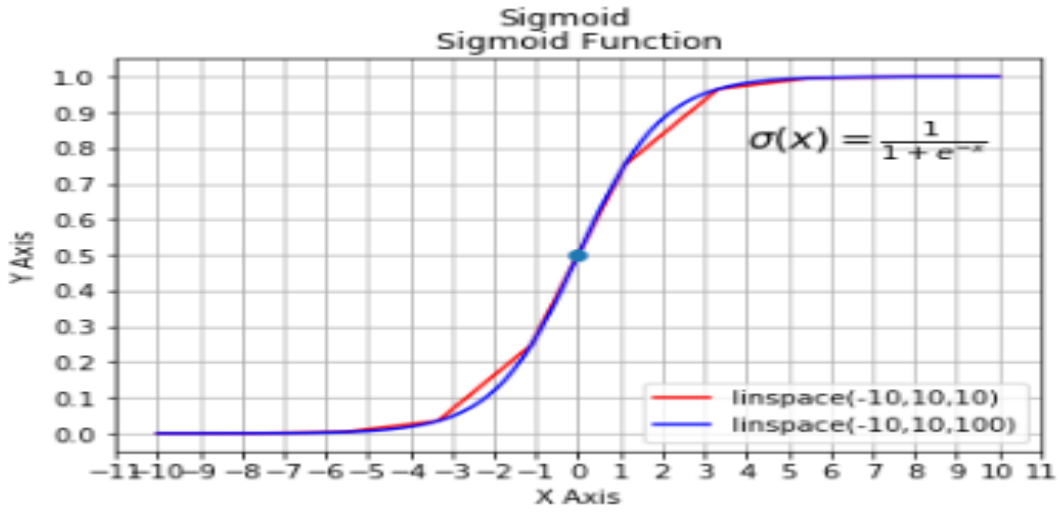
```
def Circle_equation(x1, y1, r):
    a = -2 * x1;
    b = -2 * y1;
    c = (r * r) - (x1 * x1) - (y1 * y1);
    # Printing result
    print("x^2 + (", a, "x) + ", end = "");
    print("y^2 + (", b, "y) = ", end = "");
    print(c, ".");
x1 = 2;
y1 = -3;
r = 8;
Circle_equation(x1, y1, r);
fig, ax = plt.subplots()
ax = fig.add_subplot(111)
circle = plt.Circle((x1, y1), r,color="purple")
ax.add_patch(circle)
label = ax.annotate("*", xy=(x1, y1),fontsize=30, ha="center")
ax.axis('off')
ax.set_aspect('equal')
ax.autoscale_view()
plt.show()
```

```
x^2 + ( -4 x) + y^2 + ( 6 y) = 51 .
```



CENTRE OF MASS OF SIGMOID

```
#sigmoid = lambda x: 1 / (1 + np.exp(-x))
def sigmoid(x):
    return (1 / (1 + np.exp(-x)))
mySamples = []
mySigmoid = []
x = plt.linspace(-10,10,10)
y = plt.linspace(-10,10,100)
# prepare the plot, associate the color r(ed) or b(lue) and the label
plt.plot(x, sigmoid(x), 'r', label='linspace(-10,10,10)')
plt.plot(y, sigmoid(y), 'b', label='linspace(-10,10,100)')
cgx=0
cgy=0.5
#cgx=np.sum(x)
#cgy=np.sum(y)
#center=cgx+cgy/4
#plt.plot(center,marker="o")
plt.plot(cgx,cgy,marker="o")
# Draw the grid line in background.
plt.grid()
# Title & Subtitle
plt.title('Sigmoid Function')
plt.suptitle('Sigmoid')
# place the legen boc in bottom right of the graph
plt.legend(loc='lower right')
# write the Sigmoid formula
plt.text(4, 0.8, r'$\sigma(x)=\frac{1}{1+e^{-x}}$', fontsize=15)
#resize the X and Y axes
plt.gca().xaxis.set_major_locator(plt.MultipleLocator(1))
plt.gca().yaxis.set_major_locator(plt.MultipleLocator(0.1))
# plt.plot(x)
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
# create the graph
plt.show()
```



CENTRE OF MASS OF BOB SHAPE

```
]: def sph_car(point):
    if len(point) == 2:
        point.append(1.0)
    rlon = radians(float(point[0]))
    rlat = radians(float(point[1]))
    x = cos(rlat) * cos(rlon) * point[2]
    y = cos(rlat) * sin(rlon) * point[2]
    z = sin(rlat) * point[2]
    return [x, y, z]
def xprod(v1, v2):
    x = v1[1] * v2[2] - v1[2] * v2[1]
    y = v1[2] * v2[0] - v1[0] * v2[2]
    z = v1[0] * v2[1] - v1[1] * v2[0]
    return [x, y, z]
def dprod(v1, v2):
    dot = 0
    for i in range(3):
        dot += v1[i] * v2[i]
    return dot
def plot(poly_xyz, g_xyz):
    fig = mpl.pyplot.figure()
    ax = fig.add_subplot(111, projection='3d')
    # plot the unit sphere
    u = numpy.linspace(0, 2 * numpy.pi, 100)
    v = numpy.linspace(-1 * numpy.pi / 2, numpy.pi / 2, 100)
    x = numpy.outer(numpy.cos(u), numpy.sin(v))
    y = numpy.outer(numpy.sin(u), numpy.sin(v))
    z = numpy.outer(numpy.ones(numpy.size(u)), numpy.cos(v))
    ax.plot_surface(x, y, z, rstride=4, cstride=4, color='w', linewidth=0,
                    alpha=0.3)
    # plot 3d and flattened polygon
    x, y, z = zip(*poly_xyz)
    ax.plot(x, y, z)
    ax.plot(x, y, zs=0)
    # plot the alleged 3d and flattened centroid
    x, y, z = g_xyz
    ax.scatter(x, y, z, c='r')
    ax.scatter(x, y, 0, c='r')
    # display
    ax.set_xlim3d(-1, 1)
    ax.set_ylim3d(-1, 1)
    ax.set_zlim3d(0, 1)
    mpl.pyplot.show()
lons, lats, v = list(), list(), list()
# put the two-column data at the bottom of the question into a file called
# example.txt in the same directory as this script
with open('longitude and latitude.txt') as f:
```

```
with open('longitude and latitude.txt') as f:
    for line in f.readlines():
        sep = line.split()
        lons.append(float(sep[0]))
        lats.append(float(sep[1]))
# convert spherical coordinates to cartesian
for lon, lat in zip(lons, lats):
    v.append(sph_car([lon, lat, 1.0]))
# z unit vector/pole ('north pole'). This is an arbitrary point selected to act as one
#(fixed) vertex of the summed spherical triangles. The other two vertices of any
#triangle are composed of neighboring vertices from the polygon boundary.
np = [0.0, 0.0, 1.0]
# Gx,Gy,Gz are the cartesian coordinates of the calculated centroid
Gx, Gy, Gz = 0.0, 0.0, 0.0
for i in range(-1, len(v) - 1):
    # cycle through the boundary vertices of the polygon, from 0 to n
    if all((v[i][0] != v[i+1][0],
            v[i][1] != v[i+1][1],
            v[i][2] != v[i+1][2])):
        # this just ignores redundant points which are common in  input files
        # A,B,C are the internal angles in the triangle: 'np-v[i]-v[i+1]-np'
        A = asin(sqrt((dprod(np, xprod(v[i], v[i+1])))**2
                      / ((1 - (dprod(v[i+1], np))**2) * (1 - (dprod(np, v[i]))**2))))
        B = asin(sqrt((dprod(v[i], xprod(v[i+1], np)))**2
                      / ((1 - (dprod(np , v[i]))**2) * (1 - (dprod(v[i], v[i+1]))**2))))
        C = asin(sqrt((dprod(v[i + 1], xprod(np, v[i])))**2
                      / ((1 - (dprod(v[i], v[i+1]))**2) * (1 - (dprod(v[i+1], np))**2))))
        # A/B/Cbar are the vertex angles, such that if 'O' is the sphere center, Abar
        # is the angle (v[i]-O-v[i+1])
        Abar = acos(dprod(v[i], v[i+1]))
        Bbar = acos(dprod(v[i+1], np))
        Cbar = acos(dprod(np, v[i]))
        # e is the 'spherical excess'
        e = A + B + C - pi
        # mag1/2/3 are the magnitudes of vectors np,v[i] and v[i+1].
        mag1 = 1.0
        mag2 = float(sqrt(v[i][0]**2 + v[i][1]**2 + v[i][2]**2))
        mag3 = float(sqrt(v[i+1][0]**2 + v[i+1][1]**2 + v[i+1][2]**2))
        # vec1/2/3 are cross products
        vec1 = xprod(np, v[i])
        vec2 = xprod(v[i], v[i+1])
        vec3 = xprod(v[i+1], np)
        # multiplying vec1/2/3 by e and respective internal angles
        for x in range(3):
            vec1[x] *= Cbar / (2 * e * mag1 * mag2
                               * sqrt(1 - (dprod(np, v[i]))**2))
            vec2[x] *= Abar / (2 * e * mag2 * mag3
                               * sqrt(1 - (dprod(v[i], v[i+1]))**2))
            vec3[x] *= Bbar / (2 * e * mag3 * mag1
                               * sqrt(1 - (dprod(v[i+1], np))**2))
```

```
        # vec1/2/3 are cross products
        vec1 = xprod(np, v[i])
        vec2 = xprod(v[i], v[i+1])
        vec3 = xprod(v[i+1], np)
        # multiplying vec1/2/3 by e and respective internal angles
        for x in range(3):
            vec1[x] *= Cbar / (2 * e * mag1 * mag2
                               * sqrt(1 - (dprod(np, v[i])**2)))
            vec2[x] *= Abar / (2 * e * mag2 * mag3
                               * sqrt(1 - (dprod(v[i], v[i+1])**2)))
            vec3[x] *= Bbar / (2 * e * mag3 * mag1
                               * sqrt(1 - (dprod(v[i+1], np)**2)))
        Gx += vec1[0] + vec2[0] + vec3[0]
        Gy += vec1[1] + vec2[1] + vec3[1]
        Gz += vec1[2] + vec2[2] + vec3[2]

approx_expected_Gxyz = (0.78, -0.56, 0.27)
print('Approximate Expected Gxyz: {0}\n'
      '               Actual Gxyz: {1}'
      ''.format(approx_expected_Gxyz, (Gx, Gy, Gz)))
if plotting_enabled:
    plot(v, (Gx, Gy, Gz))
```

```
Approximate Expected Gxyz: (0.78, -0.56, 0.27)
              Actual Gxyz: (0.690076955898216, -0.6108369363223827, 0.777844765369879)
```