Center of Mass

This notebook is about computing the center of mass of a set of objects in 1D, 2D, or 3D. It introduces some of the MATLAB tools you will need for the boat project, and a framework for doing this kind of computation.

Center of mass in 1D

Let's start with a 30 cm ruler that weighs 25 grams. We'll use a "structure" to represent it. A MATLAB structure is similar to a System object in the ModSimPy framework.

```
ruler.length = 30;   % cm
ruler.weight = 25;   % gram
ruler
```

```
ruler = struct with fields:
    length: 30
    weight: 25
```

To represent the coordinate system, we'll use a structure called **mesh** that contains a matrix of equally-spaced points, called **xs**, and the space between points, **dx**.

```
mesh.dx = 0.1;                          % cm
mesh.xs = 0:mesh.dx:ruler.length;   % coordinates in cm
mesh
```

```
mesh = struct with fields:
    dx: 0.1000
    xs: [1×301 double]
```

Now we want a matrix to represent the mass at each point along the ruler. We start by computing the density of the ruler in terms of weight per cell in the mesh.

```
ruler.density = ruler.weight / numel(mesh.xs)    % grams / cell
```

```
ruler = struct with fields:
     length: 12
     weight: 34
    density: 0.2810
```

Then we create a matrix with the same size as the mesh, but where the value at every location is the density.

```
masses = ones(size(mesh.xs)) * ruler.density
```

```
masses =
    0.2810    0.2810    0.2810    0.2810    0.2810    0.2810    0.2810    0.2810 ···
```

We have a function to compute the total mass in **masses**. You can see how it works below.

```
M = matrixSum(masses)
```

```
M = 34.0000
```

Now we get to the good stuff. Read the definition of **centerOfMass** below and see if you can figure out how it works. Can you see a relationship between this code and the definition of COM in one dimension?

$$COM = \frac{1}{M}\sum_i m_i x_i$$

Is the computed center of mass where you expected?

```
COM = centerOfMass(masses,mesh)
```

```
COM = 6.0000
```

Now let's place a 100 gram mass at the point 3 cm from the left end of the ruler. Read **addMass,** below, to see how it works.

```
mass = 100   % gram
```

```
mass = 100
```

```
position = 3  % cm
```

```
position = 3
```

```
masses = addMass(mass,position,masses,mesh)
```

```
masses =
    0.2810    0.2810    0.2810    0.2810    0.2810    0.2810    0.2810    0.2810 ···
```

Compute the center of mass for this scenario by hand. Then run the code and see if you get the same result.

```
M = matrixSum(masses)
```

```
M = 134.0000
```

```
COM = centerOfMass(masses,mesh)
```

```
COM = 3.7612
```

Exercise 1: Check your notes from the last class to see how you arranged weights on your ruler. Modify the code above to represent the scenario you created. Adjust the length and weight of the ruler if nessesary. Run the code and see if the answer is consistent with what you calculated yesterday.

```
ruler.length = 12;
ruler.weight = 34;
mesh.dx = 0.1;
mesh.xs = 0:mesh.dx:ruler.length;
ruler.density = ruler.weight / numel(mesh.xs);
```

```
masses = ones(size(mesh.xs)) * ruler.density;
COM = centerOfMass(masses,mesh)
```

```
COM = 6.0000
```

```
% Add masses 20 grams and 50 grams on each end
masses = addMass(20,0,masses,mesh);
masses = addMass(50,ruler.length,masses,mesh);
```

```
COM = centerOfMass(masses,mesh)
```

```
COM = 7.7308
```

Center of mass in two dimensions

Now let's consider the other scenario you worked on in class: masses arranged on a plate.

Here a MATLAB structure to represent the plate (based on the size and weight of the acrylic slabs we used in class).

```
plate.length = 14;   % cm
plate.width = 7;     % cm
plate.weight = 77;   % gram
```

Now we need a 2D mesh. We'll divide the place into small rectangles, each **dx** long (in the direction of **length**) and **dy** wide (in the direction of **width**).

xs and **ys** are matrices with only one column.

We pass **xs** and **ys** to **meshgrid** to get **xgrid** and **ygrid**, which are matrices with one column for each x location and one row for each y location.

```
clear mesh
mesh.dx = 0.1;            % cm
mesh.dy = 0.1;            % cm
mesh.xs = 0:mesh.dx:plate.length;   % coordinates in cm
mesh.ys = 0:mesh.dy:plate.width;    % coordinates in cm
[mesh.xgrid,mesh.ygrid] = meshgrid(mesh.xs, mesh.ys);
mesh
```

```
mesh = struct with fields:
       dx: 0.1000
       dy: 0.1000
       xs: [1×141 double]
       ys: [1×71 double]
    xgrid: [71×141 double]
    ygrid: [71×141 double]
```

Again, we define **density** to be the mass of each cell in the mesh.

```
plate.density = plate.weight / numel(mesh.xgrid);    % grams / cell
plate
```

```
plate = struct with fields:
     length: 14
      width: 7
     weight: 77
    density: 0.0077
```

Next we create a matrix that's the same size as the mesh, where the value at each location is the density.

```
masses = ones(size(mesh.xgrid)) * plate.density
```

```
masses =
    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077 ⋯
    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077
    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077
    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077
    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077
    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077
    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077
    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077
    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077
    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077
      ⋮
```

-

The matrixSum function still works (in fact, it works with any number of dimensions). We can use it to confirm that the total mass is correct.

```
M = matrixSum(masses)
```

```
M = 77.0000
```

Computing the center of mass in two dimensions takes a little more work. See **centerOfMass2** below and make sure you understand it.

```
COM = centerOfMass2(masses,mesh)
```

```
COM =
    7.0000    3.5000
```

-

Does the result make sense?

Now let's put a 100 gram mass at the position x=0.3 cm, y=0.3 cm, measured from a corner of the plate. To represent a point in 2D, I could use a vector like this:

position = [0.3, 0.3]; % cm

Or a structure with fields **x** and **y**, as shown below.

```
mass = 100;   % gram

clear position
position.x = 0.3;   % cm
position.y = 0.3;   % cm
```

To add the mass to the plate, we need a new version of **addMass2**.

Exercise 2: Fill in the body of **addMass2**, below, so it does what it is supposed to.

```
masses = addMass2(mass,position,masses,mesh)
```

```
index_x = 4
index_y = 4
masses =
    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077 ···
    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077
    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077
    0.0077    0.0077    0.0077  100.0077    0.0077    0.0077    0.0077    0.0077
    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077
    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077
    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077
    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077
    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077    0.0077
     ⋮
```

Once you have **addMass2** working, compute the COM again and see if it shifted as expected. Check the result by hand.

```
M = matrixSum(masses)
```

```
M = 177.0000
```

```
% The result is correct
COM = centerOfMass2(masses,mesh)
```

```
COM =
    3.2147    1.6921
```

Exercise 3: Check your notes from the last class to see how you arranged weights on your plate. Modify the code above to represent the scenario you created. Adjust the parameters of the plate if nessesary. Run the code and see if the answer is consistent with what you calculated yesterday.

```
plate.length = 6;   % cm
plate.width = 3;     % cm
plate.weight = 73;
```

```
clear mesh
mesh.dx = 0.01;            % cm
mesh.dy = 0.01;            % cm
mesh.xs = 0:mesh.dx:plate.length;   % coordinates in cm
mesh.ys = 0:mesh.dy:plate.width;    % coordinates in cm
[mesh.xgrid,mesh.ygrid] = meshgrid(mesh.xs, mesh.ys);
mesh
```

```
mesh = struct with fields:
```

```
        dx: 0.0100
        dy: 0.0100
        xs: [1×601 double]
        ys: [1×301 double]
     xgrid: [301×601 double]
     ygrid: [301×601 double]
```

```
plate.density = plate.weight / numel(mesh.xgrid);    % grams / cell
plate
```

```
plate = struct with fields:
     length: 6
      width: 3
     weight: 73
    density: 4.0354e-04
```

```
masses = ones(size(mesh.xgrid)) * plate.density;
```

```
COM = centerOfMass2(masses,mesh)
```

```
COM =
    3.0000    1.5000
```

```
% Add masses 20 grams at (1,1)
clear position
position.x = 1;   % cm
position.y = 1;
masses = addMass2(20,position,masses,mesh);
```

```
index_x = 101
index_y = 101
```

```
% Add masses 50 grams at (3.8,1.7)
clear position
position.x = 3.8;   % cm
position.y = 1.7;
masses = addMass2(50,position,masses,mesh);
```

```
index_x = 381
index_y = 171
```

```
% The result is correct. The COM stays the same.
COM = centerOfMass2(masses,mesh)
```

```
COM =
    3.0000    1.5000
```

Center of mass in three dimensions

This is an optional exercise if you finish the previous sections and have some additional time. It is a good opportunity to practice the MATLAB features we have learned so far and continue to develop your progamming skills.

Exercise 4: Now that we have two dimensions, three dimensions is easy!

1. Create a structure called **cube** that contains **length** = 10 cm, **width** = 10 cm, **height** = 10 cm, and **weight** = 1000 grams.
2. Create a mesh with **dx**, **dy**, and **dz** = 0.2.
3. Compute **cube.density** and create a matrix called **masses** with the same size as the mesh. Use matrixSum to confirm that the total mass is right.
4. Write a function called **centerOfMass3** that computes the center of mass in three dimensions. Test your function and confirm that the center of mass is where you expect it to be.
5. Suppose we replace one cell of the cube, in one corner, with a super dense material, so that the mass of that cell is 1000 grams. Where do you expect the new center of mass to be? Compute it and see if you got it right.

```
cube.length = 10;
cube.width = 10;
cube.height = 10;
cube.weight = 1000;
```

```
clear mesh
mesh.dx = 0.1;            % cm
mesh.dy = 0.1;            % cm
mesh.dz = 0.1;            % cm
mesh.xs = 0:mesh.dx:cube.length;   % coordinates in cm
mesh.ys = 0:mesh.dy:cube.width;    % coordinates in cm
mesh.zs = 0:mesh.dz:cube.height;    % coordinates in cm
[mesh.xgrid,mesh.ygrid,mesh.zgrid] = meshgrid(mesh.xs, mesh.ys, mesh.zs);
mesh
```

```
mesh = struct with fields:
       dx: 0.1000
       dy: 0.1000
       dz: 0.1000
       xs: [1×101 double]
       ys: [1×101 double]
       zs: [1×101 double]
    xgrid: [101×101×101 double]
    ygrid: [101×101×101 double]
    zgrid: [101×101×101 double]
```

```
cube.density = cube.weight / numel(mesh.xgrid);    % grams / cell
cube
```

```
cube = struct with fields:
     length: 10
      width: 10
     height: 10
     weight: 1000
```

```
    density: 9.7059e-04
```

```
masses = ones(size(mesh.xgrid)) * cube.density;
COM = centerOfMass3(masses,mesh)
```

```
COM =
    5.0000    5.0000    5.0000
```

•

```
% Add masses 1000 grams at (0,0,0)
clear position
position.x = 0;   % cm
position.y = 0;
position.z = 0;
masses = addMass3(1000,position,masses,mesh);
```

```
index_x = 1
index_y = 1
index_z = 1
```

```
COM = centerOfMass3(masses,mesh)
```

```
COM =
    2.5000    2.5000    2.5000
```

•

Function definitions

```
function M = matrixSum(masses)
    % matrixSum: returns total of all elements in the matrix

    % normally sum(m) computes the sums of the columns
    % selecting m(:) flattens the matrix and computes the sum of all elements
    % see https://stackoverflow.com/questions/1721987/what-are-the-ways-to-sum-matrix-elements
    M = sum(masses(:));
end

function COM = centerOfMass(masses,mesh)
    % centerOfMass: computes center of mass in 1D
    % masses: matrix of masses
    % mesh: structure containing xs
    % returns: scalar
    M = matrixSum(masses);
    COM = matrixSum(masses .* mesh.xs) / M;
end

function masses = addMass(mass,position,masses,mesh)
    % addMass: adds mass at a given position in 1D
    % mass: scalar, in grams
    % position: scalar, in cm
    % masses: matrix of masses
    % mesh: structure containing xs
```

```
    % returns: the updated matrix of masses

    % find the index of the location in the mesh closest to position
    index = closestIndex(position,mesh.xs);

    % update the matrix
    masses(index) = masses(index) + mass;
end

function index = closestIndex(coord,grid)
    % closestIndex: finds the index of the grid point closest to coord
    % coord: scalar coordinate
    % grid: matrix of coordinates
    % returns: integer index

    % see: https://www.mathworks.com/matlabcentral/answers/152301-find-closest-value-in-array
    [c index] = min(abs(coord-grid));
end

function COM = centerOfMass2(masses,mesh)
    % centerOfMass2: computes center of mass in 2D
    % masses: matrix of masses
    % mesh: structure containing xgrid and xgrid
    % returns: Vector [xcom,ycom]

    M = matrixSum(masses);
    xcom = matrixSum(masses .* mesh.xgrid) / M;
    ycom = matrixSum(masses .* mesh.ygrid) / M;
    COM = [xcom,ycom];
end

function COM = centerOfMass3(masses,mesh)
    % centerOfMass2: computes center of mass in 2D
    % masses: matrix of masses
    % mesh: structure containing xgrid and xgrid
    % returns: Vector [xcom,ycom]

    M = matrixSum(masses);
    xcom = matrixSum(masses .* mesh.xgrid) / M;
    ycom = matrixSum(masses .* mesh.ygrid) / M;
    zcom = matrixSum(masses .* mesh.zgrid) / M;
    COM = [xcom,ycom,zcom];
end

function masses = addMass2(mass,position,masses,mesh)
    % addMass2: adds mass at a given position in 2D
    % mass: scalar, in grams
    % position: scalar, in cm
    % masses: matrix of masses
    % mesh: structure containing xs
    % returns: the updated matrix of masses

    % TODO: FIX THIS SO IT PUTS THE MASS IN THE RIGHT PLACE
    % find the index of the location in the mesh closest to position
    index_x = closestIndex(position.x,mesh.xs)
    index_y = closestIndex(position.y,mesh.ys)

    % update the matrix
    masses(index_y,index_x) = masses(index_y,index_x) + mass;
end
```

```
function masses = addMass3(mass,position,masses,mesh)
    % addMass2: adds mass at a given position in 2D
    % mass: scalar, in grams
    % position: scalar, in cm
    % masses: matrix of masses
    % mesh: structure containing xs
    % returns: the updated matrix of masses

    % TODO: FIX THIS SO IT PUTS THE MASS IN THE RIGHT PLACE
    % find the index of the location in the mesh closest to position
    index_x = closestIndex(position.x,mesh.xs)
    index_y = closestIndex(position.y,mesh.ys)
    index_z = closestIndex(position.z,mesh.zs)

    % update the matrix
    masses(index_z,index_y,index_x) = masses(index_z,index_y,index_x) + mass;
end
```