MECH 222 Computer Lab 1—Point Clouds

Read and understand the briefing notes and lab instructions before your lab period. Each lab activity corresponds to one of the main ideas in the briefing notes.

Learning Objectives Doing this lab should be fun. The activities are designed to sharpen your skills and understanding in three important areas.

- Matlab: You will practice writing Matlab loops and functions. The modular approach makes your work easier to debug and reuse. You will meet the useful `load` function (but not its partner, `save`). You will get a chance to learn and deploy a favourite trick of Matlab power users: "vectorization." And you will start playing with Matlab's system of handle-graphics.
- Mathematics: You will have a real-life encounter with the abstract-sounding idea of splitting a complicated problem into many small pieces, doing something simple with each piece, and then reassembling the results to solve the original problem. You will see real physical meaning emerge from dot products involving vectors having over 10,000 components!
- Engineering: The context for all this is the familiar world of fundamental mechanical and kinematic quantities like total mass and centre of mass.

PRE-LAB ASSIGNMENT Hand in the requested items at the beginning of the lab, but keep a copy for your reference during the session.

PL0. Read the briefing notes. All of them. *Before the lab starts.*

PL1. Three point masses are given, with coordinates in meters and masses in kilograms shown below:

k	x_k	y_k	z_k	m_k
1	1.1	1.2	2.2	3.0
2	0.0	0.0	0.0	5.0
3	−2.3	4.5	0.8	2.2

Find the centre of mass for this three-mass system.

PL2. *Do this by hand, without relying on the computer!* Write down the Matlab vectors `u = [1;2;3;4]` and `w = [2,1,2,1]` in matrix form. (The point: `u` and `w` have different "shapes".) Then calculate both the matrix-matrix products that Matlab would denote by `u*w` and `w*u`. (The point: both products are well-defined operations with predictable results, but those results are dramatically different.)

DAILY SPECIALS The downloadable files `pointcloud_DDD.mat` and `wirenodes_DDD.mat` used in the main lab activity are different each day. Replace `DDD` with the 3-letter code for your assigned lab day as you read the instructions below.

ACTIVITY 1: *Calculate the mass and centre of mass for a given cloud of points.*

1.0 Copy the file `pointcloud_DDD.mat` from the web into your current working directory. Then, in Matlab, enter the command `load pointcloud_DDD` at the prompt. This will create four vectors `x`, `y`, `z`, `m` like the ones illustrated in Prelab item 1. Assume that the coordinates are given in meters, and the masses are given in kg. Check that the vectors have been loaded correctly by looking in Matlab's workspace browser or giving the command `whos`.

1.1 Enter the command `plot3(x,y,z,'r.')` to produce a 3D plot with a red dot at each point mass location. Also say `axis equal`, to display the cloud with accurate proportions. Add axis labels and a title to your plot; include your name in the title. Use the tools in the plot window to inspect the cloud from all sides and to choose a view that looks good in a snapshot. [Health Advisory: *If you think you see meaningful shapes in the cloud, it may be a sign that you have been working extremely hard lately. If this is a problem, talk to a trusted friend or—in extreme cases—to a member of the instructional team. But finish your lab first.*]

1.2 Find N, the number of points in the cloud. (Hint: `help length`.)

1.3 Find the total mass of the points in the cloud, namely,

$$m_{\mathrm{TOT}} = \sum_{k=1}^{N} m_k.$$

(The briefing notes outline two approaches, one using a for-loop, one using a dot product. Either choice is acceptable, provided it works!)

1.4 Find the centre of mass for the cloud, namely, $\mathbf{r}_{\mathrm{CM}} = (x_{\mathrm{CM}}, y_{\mathrm{CM}}, z_{\mathrm{CM}})$, where

$$x_{\mathrm{CM}} = \frac{\sum_{k=1}^{N} x_k m_k}{m_{\mathrm{TOT}}}, \qquad y_{\mathrm{CM}} = \frac{\sum_{k=1}^{N} y_k m_k}{m_{\mathrm{TOT}}}, \qquad z_{\mathrm{CM}} = \frac{\sum_{k=1}^{N} z_k m_k}{m_{\mathrm{TOT}}}.$$

1.5 Repackage your work into a function named `mcm_cloud`, so that the following three commands will quickly calculate all the elements requested above:

```
clear;
load pointcloud.mat
[m_tot,x_cm,y_cm,z_cm] = mcm_cloud(x,y,z,m);
```

(Remember the difference between a *function* and a *script*: it's a *function* we need here.) For full credit, do not use the Matlab functions `sum` and `dot`. For extra respect and possible bonus points, don't use a `for`-loop either.

Hand-in Checklist:

☐ A printed copy of your function `mcm_cloud`.

☐ The picture of your point cloud from item 1.2.
WARNING: Various naive approaches to this job lead to disaster. *Do not* simply select "File → Print" in the Figure window. *Do not* try to Copy/Paste the figure into a Word document. The figure contains such a huge number of points that

these simple-sounding operations take nearly forever. *Do this instead*: activate the Figure window and choose "File → Save As ..." from the menus. Save your figure as a file of type "Portable Network Graphics file", suffix `png`. Then use some program other than Matlab to manipulate and print your `png` file. (There are many choices. Even Internet Explorer can do it.) This works because the `png` file ignores thousands and thousands of points that make no difference to the visual appearance of the image.

□ The computed values of N, m_{TOT} and $\mathbf{r}_{\mathrm{CM}}$.

ACTIVITY 2: *Find the mass and centre of mass of a wire with variable density.*

2.0 Copy the file `wirenodes_DDD.mat` from the web into your current working directory. Then, in Matlab, enter the command `load wirenodes_DDD` at the prompt. This will create four vectors `x`, `y`, `z`, `rho`, as described in the Briefing Notes.

2.1 Immediately throw away most of the data in these four vectors to produce much smaller datasets for testing and development. To do so, enter these commands:

```
N   = length(x),   skip = round(N/10)
x   = x(1:skip:N); y = y(1:skip:N); z = z(1:skip:N);
rho = rho(1:skip:N);
N   = length(x)
```

These steps overwrite the vectors loaded in step 2.0 with new ones having about 10 elements. Use these in stages 2.2–2.5 below.

2.2 Say `plot3(x,y,z)` to produce a 3D plot showing the approximate wire. Say `axis equal` to set the proportions, then `hold on` to allow you to overlay additional elements later. Label the axes on the plot and add a title that includes your name.

2.3 Figure out how to approximate the test-wire with a cloud of point masses, using one point for each segment that joins two nodes. Choose the mass and location of the point as discussed in the briefing notes. Represent the desired cloud in the usual way—that is, in three column vectors containing the coordinates and one column vector containing the masses. (Be smart: invent new variable names to hold the new information!)

2.4 Overlay the plot in item 2.2 with the cloud of point masses just calculated. Use a distinctive and highly visible symbol for the points. Hand in a copy of the resulting figure.

2.5 Repackage your work from step 2.3 to define a new Matlab function named `mcm_wire`. This function should return the wire's total mass and CM coordinates when you say

```
[m_tot,x_cm,y_cm,z_cm] = mcm_wire(x,y,z,rho);
```

Report the output of this function on the low-res test copy of the wire. (Hint: Step 1.5 provides a function that can find the mass and CM coordinates for any given cloud of points. So your new `mcm_wire` can outsource its final calculations

to that existing function. All it has to do is generate the cloud of CM points and segment masses from given wire nodes and densities, then give these to `mcm_cloud` for processing.)

2.6 Clear all variables from your workspace and repeat step 2.0 to reload the full-length vectors in `wirenodes_DDD.mat`. Repeat Step 2.3 to show a much more detailed graphic of the wire. Then apply your function from Step 2.5 to determine the wire's mass and centre of mass.

2.7 Even without the given density information, the list of points (x_k, y_k, z_k) is enough to describe a curve in space. Invent an efficient way to find the centroid coordinates and total length of a such a curve *using work you have already done*. Package it as a function named `curve_facts`, which works when you say

```
clear;
load wirenodes_DDD
[len,xbar,ybar,zbar] = curve_facts(x,y,z)
```

(*Hint*: Get `mcm_wire` to do most of the work.) Find and submit the curve facts for the high-resolution wire loaded in step 2.7.

Hand-in Checklist:

- ☐ Printed copies of your functions `mcm_wire` and `curve_facts`.
- ☐ A picture of the low-res wire overlaid with the computed point cloud.
- ☐ The computed values of m_{TOT} and $\mathbf{r}_{\mathrm{CM}}$ for the low-res wire.
- ☐ A picture of the high-res wire (no overlay required).
- ☐ The computed values of m_{TOT} and $\mathbf{r}_{\mathrm{CM}}$ for the hi-res wire.
- ☐ The computed length and centroid coordinates for the hi-res wire.

ACTIVITY 3: *Automate an artist at work.*

3.0 Copy the file `artist3.m` from the web into your current working directory. This file contains the first few lines of the function you will build in this activity. The steps below illustrate "incremental program development."

3.1 Clear your Matlab workspace and reload `wirenodes_DDD` as outlined in Steps 2.0–2.1 above. Modify the instructions in Step 2.1 to extract subvectors of `x`, `y`, and `z` having length around 720.

3.2 As given, the function `artist3` creates a classic 3D viewport automatically scaled to contain the daily wire image. Run it. Then, working at the command prompt, figure out how to draw a big dot of your favourite colour at the point `(x(1),y(1),z(1))`. Just drawing a dot is easy:

```
dothandle = plot3(x(1),y(1),z(1),'.');
```

This assignment gives you access to the "handle" of the dot as a graphic object. You can see a list of all your dot's properties using the command `get(dothandle)`. The properties of particular interest are named `MarkerSize` and `Color`. Use commands of the form `set(dothandle,...)` to adjust these to your liking. Once

you have this working, add commands to the file `artist3.m` so that the function builds the dot you want.

3.3 Extend `artist3.m` further by adding a `for`-loop that moves your dot from its initial point to the next node, then the following node, and eventually all the way along the wire. The key actions in the loop will be updating the dot's location by modifying the properties named `XData`, `YData`, and `ZData`, and then refreshing the picture on the screen by giving the command `drawnow`.

The speed of your computer will influence how fast it takes to issue several hundred `drawnow` commands. If your computer is slow, extract subvectors with fewer elements in Step 3.1. If the picture goes by too fast to see anything, either use more elements instead or insert a command like `pause(0.03)` into the body of your `for`-loop. Choose the pause duration that gives a pleasant viewing experience.

3.4 Extend `artist3.m` further by drawing a line segment from the dot's starting point to its next point in every iteration of the `for`-loop from Step 3. Now the command-line input

```
close all; artist3(x,y,z);
```

should produce a little movie in which the dot represents the tip of an artist's paintbrush as it draws a single line that grows into a pretty picture.

To get this marked, print a copy of your completed file `artist3.m` and ask the Lab TA to watch your movie and initial your printout as confirmation that the movie works.

Hand-in Checklist:

- ☐ A printed copy of your function `artist3`.
- ☐ Your TA's signoff that your movie displays correctly.

OPTIONAL BONUS ACTIVITY 1: *Vectorize your functions.*

Rewrite each of the functions in Activities 1 and 2 so that no `for`-loops appear. Functions `mcm_cloud` and `curve_facts` are easy; `mcm_wire` will be a challenge. If you succeed, just swap the high-value versions of your functions into the hand-in package recording your work on the original activities.

OPTIONAL BONUS ACTIVITY 2: *Make a roller-coaster.*

Taking inspiration from `artist3`, write a function `coaster3` so that the command `coaster3(x,y,z)` will show a curve in light colour and animate the progress of a 10-car train made of dots moving along the track from beginning to end. Play around having other good ideas. Share these with the Lab TA or the instructor responsible for the Computer Labs.