MATLAB® Assignment Report written in LaTeX

Joseph Pym, SiD: 8404110

Due 14 December, 2018

Contents

1 Task 1: Collision-less Brownian Motion

1.1 Task a) Plotting the Walls

To generate a random amount of particles, I used[1] the code referenced 1. To generate the polar coordinates for the x and y values of each particle (compared code with standard maths), I used the code referenced 2.

```
Na     = randi([100 500],1,1);
Theta  = 360 * rand(Na,1,1);
Ra     = 100 * rand(Na,1,1);
```
(1)

Maths Formulae:	**In my code:**
$x = r\cos\theta$	`xa=ra.*cosd(theta);`
$y = r\sin\theta$	`ya=ra.*sind(theta);`

(2)

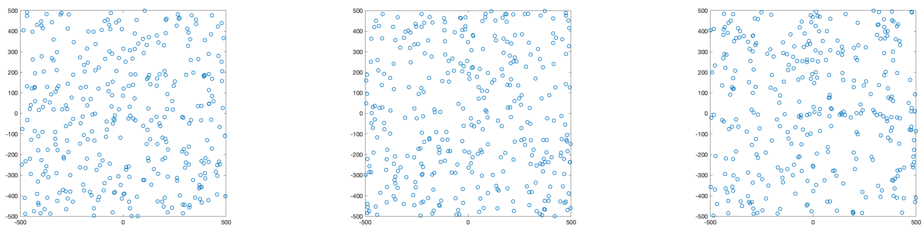
My plot is created in lines 10 to 13 in Appendix 4.1. Lines 15 to 17 in this appendix gives each individual particle a random velocity[2]. The variables that are defined in this task are given in Table 1.

Variable	Job of Variable
`Na`	Random number of particles generated ($100 \leq$ `Na` ≤ 500)
`theta` (θ)	A randomly generated angle the particles are sent at ($0^o \leq \theta \leq 360^o$)
`ra`	The distance from the centre the particle is generated at ($0 \leq$ `ra` ≤ 100)
`xa`, `ya`	The polar coordinates for x and y respectively
`ha`	The plot
`va`	The randomly generated velocity ($10 \leq$ `va` ≤ 50)
`vxa`, `vya`	This calculates the horiztonal and vertical components of the velocity
`ta`	Time passed
`dta`	The time step

Table 1: *Variables used in task 1a, and their jobs*

1.2 Task b) Updating the Particles Positions

After the code from Section 1.1, I used a while loop. I implimented the equation $v = v_0 + at$, and used `h.XData` and `h.YData` to update the positions on the plot. This while loop is between lines 25 and 30 in Appendix 4.1.



(a) Screenshot 10 (b) Screenshot 15 (c) Screenshot 25

Figure 1: *Some of the screenshots from code ref. 3*

This code sends the particles off in a random direction at a random velocity, however it does not 'bounce' off the wall. In order to solve this, I used the `find` command (lines 32-35 in Appendix 4.1). To take the screenshots

[1] I used `Na` instead of just `N` to distinguish it from any other variables I would wish to call `N` in future tasks.
[2] Between 10 and 50 units

required, I used modulo artithmetic take screenshots every time `t` is a multiple of 10. The code I used for is referenced 3. Some screenshots created by this are shown in figure 1.

```
if    mod(ta,10)==0 ta<=250
      filename = ['screenshot_taskb_' num2str(ta/10) '.png];
      saveas(gcf,filename)
end
```
(3)

1.3 Task c) Adding a Trace behind Particles

The first part is similar to Task 1a). I started off by changing:

- any variable ending with `a` to `c` (i.e. `xa` ⇒ `xc`)
- the line `figure(1)` ⇒ `figure(2)`
- the line `Na = randi([100 500],1,1);` ⇒ `Nc = randi([20 40],1,1);` (as only a few particles are needed).
- the time limit: the `while` loop is now until `t < = 100` .
- the number of screenshots: there is now only one when `tc == 99`

With the help of an M-file Alex shared (Pedcenko, 2018 [2]), I added serveral features to my code, referenced 4. The output of this is shown in figure 2.

With the previous variable definitions:

```
Xc = [xc];
Yc = [yc];
for i = 1:Nc
      trc(i) = plot(Xc(:,i),Yc(:,i),'-k');
end
```

Within the loop creating particle motion:

```
Xc = [Xc; xc];
Yc = [Yc; yc];
hc.XData = xc;
hc.YData = yc;
for i = 1:Nc
      trc(i).XData=Xc(:,i);
      trc(i).YData=Yc(:,i);
end
```
(4)

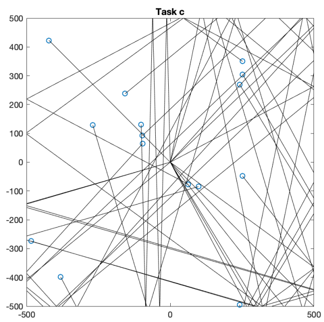


Figure 2: *The figure resulting from the code ref. 4*

1.4 Task d) Action of Gravity

Defining `g=9.8` , I added an updated calculation for `vyd` : $v = v_0 + gt$, by `vyd = vyd-g*dt` . The only major change from the previous code is shown in Appendix 4.3.

1.5 Task e) Loss of Energy on Collision

For this task, I used a lot of the code from Task a). I added two variables to the ones in Table 1, defined in Table 2. When a particle hits the wall, its `en` value loses 10%. Its velocity is then multipled by the new energy, so the particle slows down after each bounce. The code ref. 5 is used for this (x values. For y values, `x` $\Rightarrow$ `y`).

Name	Job of Variable	Defined as
`En`	Array containing the energy of each individual particle	`= ones(Ne,1);`
`Toten`	Plot of the time passed again the total energy in the system	`= plot(te,sum(en),'c+:');`

Table 2: Variables added to those in Tab. 1 for task 1e).

```
en(wallxe)     = 0.9 * en(wallxe);
vxe(wallxe)    = vxe(wallxe)*-1.*en(wallxe);
```
(5)

Some improvements I would make with this task include:

- making the particles move smoother in the figure,
- not having the particles leave the boundary at all, and
- starting the particles at random points in task d) instead of the origin.

1.6 Task f) Plot of Total System Energy

To plot the sum of the energy of the particles from previous tasks, I added the variable `en`i (i is the task), and then added a new figure to them. Prior to the motion creating while loop, I added the code referenced 6. In the while loop causing the particle motion, I added the second part of the code in Appendix 4.4. However, I didn't manage to get the line to stay. When I added similar code for tasks c) and d)[3], I saw that in:

- Tasks c) and d), the total energy is constant throughout.
- Task e), the total energy starts off constant, but begins decaying as soon as there are collisions.

```
figure(z)                              z ∈ ℝ
eni = ones(Ni,1)                       i is the task
toteni = plot (ti,sum(eni),'c+:');
hold on
axis manual
axis([0 u 0 (Ni+5)])                   u is the time limit
```
(6)

1.7 Task g) Approach for Particle Motion with Collisions between Particles

If I were to create a code for this to happen, I would use the `find` function, similar to how I generated the 'bounce'. To show what I would do, I have used a pseudocode for MATLAB®, which is in appendix 4.5.

2 Task 2: Symbolic Algebra

My cubic equation to complete the second task is:

$$y(x) = -3x^3 + 12x^2 - 9x - 9 \tag{7}$$

To help with this task I used the Week 5 worksheet [3]. I had to firstly define $y(x)$. In order to do this, I used `syms x`, which introduces the symbolic variable x. I defined $y(x)$ by `y = -3 * x^3 + 12 * x^2 - 9 * x - 9`.

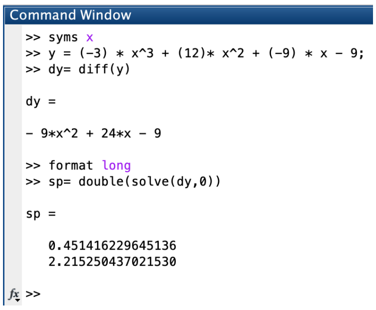
[3] I removed the code from task c) because it interefered with figure 2

2.1 Task a) Stationary Points of $y(x)$

To find the stationary points, I differentiated, using the command `dy = diff(y)`. My new variable, `dy` is equal to $\frac{d}{dx}\left[y(x)\right] = -9x^2 + 24x - 9$. To solve for when $\frac{dy}{dx} = 0$, I used the code referenced 8. This gives the x coordinates for the stationary points, as an array. In order to find the corresponding y coordinates, I used the code referenced 9.

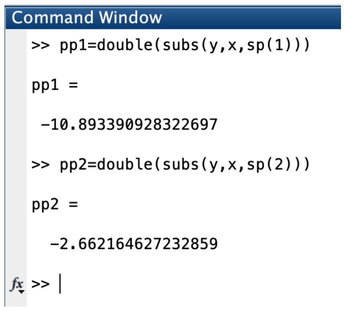
```
dy=diff y
format long
sp=double(solve(dy,0))
```
(8)

```
pp1=double(subs(y,x,sp(1)))
pp2=double(subs(y,x,sp(2)))
```
(9)



```
>> syms x
>> y = (-3) * x^3 + (12)* x^2 + (-9) * x - 9;
>> dy= diff(y)

dy =

- 9*x^2 + 24*x - 9

>> format long
>> sp= double(solve(dy,0))

sp =

   0.451416229645136
   2.215250437021530

>>
```

(a) *Code Ref. (8)*



```
>> pp1=double(subs(y,x,sp(1)))

pp1 =

 -10.893390928322697

>> pp2=double(subs(y,x,sp(2)))

pp2 =

  -2.662164627232859

>>
```

(b) *Code Ref. (9)*

Once this was done, I used `pp = [pp1 pp2]`, to create an array with the y co-ordinates of the stationary values in. The coordinates to my stationary points (to six significant figures) are:

$$\begin{aligned} (x_1,\ y_1) &= (0.451416, -10.8934) \\ (x_2,\ y_2) &= (2.21525, -2.66216) \end{aligned} \tag{10}$$

2.2 Task b) Second Derivatives of $y(x)$

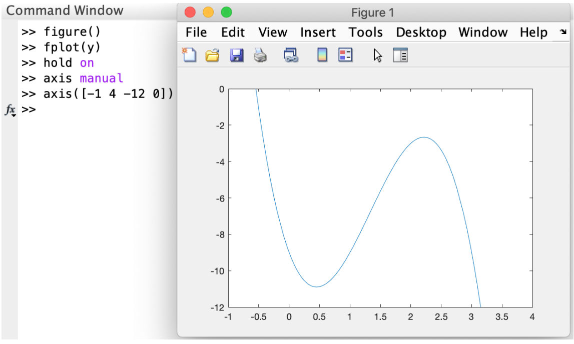
```
d2=diff dy;
cl=double(subs(d2,x,sp))
```
(11)

In order to assign the second derivitive test to the corresponding stationary point, I used the `find` function, which is between lines 7 and 11 in Appendix 4.6. From this code, we see that (x_1, y_1) is a **local minima** and (x_2, y_2) is a **local maxima**.

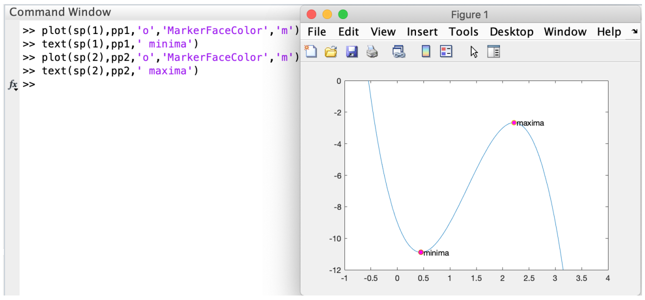
2.3 Task c) Plotting a Graph of a Function

To plot $y(x)$, I used the `figure()` command to create a blank figure, on which $y(x)$ can be plotted. The full code for this is between lines 12 and 16 in Appendix 4.6. This will plot the cubic equation referenced 7 between -1 and 4 in the x direction and -12 and 0 in the y direction. I chose this as both stationary points lie in this region. To add the markers if the maxima and minima (Ref. 10) I used the code referenced 12, which, in the command window, gives figure 4b.

```
plot(sp(1),pp1,'o','MarkerFaceColor','m')
text(sp(1),pp1,' minima')
plot(sp(2),pp2,'o','MarkerFaceColor','m')
text(sp(2),pp2,' maxima')
```
(12)



(a) *Plot of $y(x)$*



(b) *Plot of $y(x)$ with Labels*

Figure 4: Figures produced by code throughout Section 2.3

2.4 Task d) Integration of $y(x)$

$y(x)$ has one real root (Fig. 4a), so I integrated between the real root and the first stationary point (x_1, y_1). I used `root = double(solve(y==0))` to find the root, and `integral = abs(double(int(y,root(1),sp(1)))` to integrate[4]. This gave figure 5.

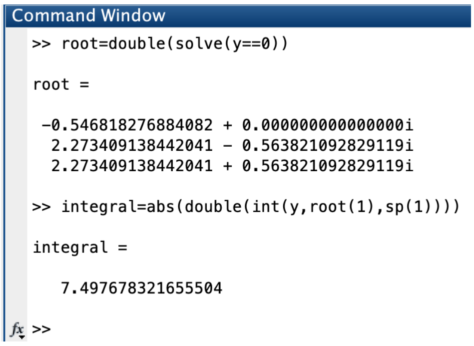


Figure 5: *Integral of $y(x)$ between the $\mathbb{R}$ root and (x_1, y_1)*

We can see, if r is the point where $y(x) = 0,\ x \in \mathbb{R}$, and using (x_1, y_1) (Ref. (10), then, to seven sig. fig.:

$$\int_r^{x_1} y(x)dx = 7.497678$$

3 Conclusion

Prior to this semester, I had never used MATLAB® or LaTeXbefore, and I am happy with how it has gone, however there are areas of improvement:

- The plot for task 1f) does not stay,
- The particles leave the boundaries for a split second, and so go out of the figure, and
- I didn't use any mathematical equations for energy, I just did a percentage.

[4] `abs` ensures the inetgral is an absolute value–Area can not be negative.

4 Appendices: Matlab Codes

4.1 Code for Task 1a) and 1b)

```
figure(1)                       % Opens a new plot

Na    = randi([100 500],1,1); % Generates particles
theta= 360*rand(Na,1,1);        % Random angle between 0 and 360 deg
ra    = 100*rand(Na,1,1);       % Random radius less than 100 units
xa    = ra.*cosd(theta);
ya    = ra.*sind(theta);        % Polar coordinates

ha    = plot(xa,ya,'o');        % Plotting the graph
axis manual
axis([-500 500 -500 500])
daspect([1 1 1])
title('Task 1a')                % Plot title

va    = 40*rand(Na,1,1)+10;     % Random veloicty 10 ? v ? 50
vxa   = va.*cosd(theta);        % Horizontal velocity component
vya   = va.*sind(theta);        % Vertical velocity component

ta    = 0;                      % Resets the time
dta   = 2;                      % Time 'step'

while ta<=250
    ta = ta+dta;                % Updates time

    xa = xa+vxa.*dta;           % Updates x coordinate
    ya = ya+vya.*dta;           % Updates y coordinate
    ha.XData = xa;              % Plots the updated x coordinate
    ha.YData = ya;              % Plots the updated y coordinate
    drawnow
    pause(0.005)                % Leaves a 0.05 second gap before continuing

    wallxa = find(abs(xa) >= 500); % Finds any x value outside the bounds
    wallya = find(abs(ya) >= 500); % Finds any y value outside the bounds
    vxa(wallxa) = vxa(wallxa)*-1;  % Flips vxa of x value outside bound
    vya(wallya) = vya(wallya)*-1;  % Flips vya of y value outside bound

    if mod(ta,10)==0 && ta<=250
        filename=['screenshot_taskb_' num2str(ta/10) '.png'];
        saveas(gcf,filename)
    end
end
```

4.2 Code for Task 1c)

Replacing lines 7-20 in Section 4.1:

```
xc=zeros(1,Nc);                 % Creates an array of all zeroes, size 1 x Nc
yc=zeros(1,Nc);
hc=plot(xc,yc,'o');
hold on
axis([-500 500 -500 500])
axis manual
daspect([1 1 1])
Xc=[xc];                        % Adds the current xc value to a vector 'Xc'
Yc=[yc];                        % Adds the current yc value to a vector 'Yc'
for i=1:Nc                      % For 'Nc' times:
    trc(i)=plot(Xc(:,i),Yc(:,i),'-k'); % Plots the trace
end
tc=0;   % Resets the timer
dtc=1;  % Time step
```

After line 27-40 in Section 4.1:

```
    Xc=[Xc; xc];    % Adds new value of xc to Xc vector
    Yc=[Yc; yc];    % Adds new value of yc to Yc vector
    hc.XData=xc;
    hc.YData=yc;
    for i=1:Nc      % For Nc times
        trc(i).XData=Xc(:,i);   % This is the x data
        trc(i).YData=Yc(:,i);   % This is the y data
    end
    drawnow;
    if tc==99       % Takes a screenshot at the end of the loop.
        filenamec=['screenshot_taskc.png'];
        saveas(gcf,filenamec)
    end
end
```

4.3 Code for Task 1d)

Replacing lines 23 and 24 in Section 4.1:

```
    td = td+dtd;                % Time ticks
    vyd=vyd-g*dtd;              % Gravity now has an effect on vertical
```

4.4 Code for Task 1e) and 1f)

After line 21 in Section 4.1:

```
te    = 0;
dte   = 1;
figure(5)                          % New figure for task f)
en    = ones(Ne,1);                % Energy of all particles = 1 (100%)
toten= plot(te,sum(en),'c+:');    % Plots the time against sum of energies
hold on
axis manual
title('Total Energy Task e')
```

Replacing all after line 29 in Section 4.1:

```
    toten.XData = te;              % x data of task f) plot is time
    toten.YData = sum(en);         % y data of task f) plot is sum of en
    hold on
    drawnow
    pause(0.01)
    wallxe = find(abs(xe) >= 500);  % Finds particles outside walls
    wallye = find(abs(ye) >= 500);  % Finds particles outside top/bottom
    en(wallxe)  = 0.9 * en(wallxe); % Decreases particle energy by 10%
    en(wallye)  = 0.9 * en(wallye); % Decreases particle energy by 10%
    vxe(wallxe) = vxe(wallxe)*-1.*en(wallxe);  % The energy now has an
    vye(wallye) = vye(wallye)*-1.*en(wallye);  % effect on velocity
    sum(en)     % Prints the sum of the energy. I used this as a check.
end
```

4.5 Pseudocode for Task 1g)

```
if x(i)==x(j) && i not= j; % Finds particles in the same location

    % The masses are the same, and the collisions are elastic
    % Some temporary variables
    vx(i)    = vxi;
    vy(i)    = vyi;
    theta(i) = theti;
    vx(j)    = vxj;
    vy(j)    = vyj;
    theta(j) = thetj;

    % Swapping each velocity component of i and j, and the angles
    vx(i)    = vxj;
    vy(i)    = vyj;
    theta(i) = thetj;
    vx(j)    = vxi;
    vy(j)    = vyi;
    theta(j) = theti;
end
```

4.6 Code for Task 2)

```
syms x
y  = (-3) * x^3 + (12)* x^2 + (-9) * x - 9   % My function , y(x)
dy = diff(y)                         % Differentiates y
format long                          % Shows to 15 digits
sp = double(solve(dy,0))             % Finds Stationary Points
d2 = diff(dy)                        % Second Derivative of y
cl = double(subs(d2,x,sp));          % Stationary Pt -> Second Derivative
mn = find(cl > 0);                   % Finds when 2nd Deriv is positive
minimum_points = cl(mn)              % This is a minimum point.
mx = find(cl < 0);                   % Finds when 2nd Deriv is negative
maximum_points = cl(mx)              % This is a maximum point.
figure()                             % Opens an empty figure
fplot(y)                             % Plots y(x)
hold on                              % Holds the current plot
axis manual                          % Freezes any scaling
axis([-1 4 -12 0])                   % Sets axis limits
pp1 = subs(y,x,sp(1));               % Finds the y value of the first SP
pp2 = subs(y,x,sp(2));               % Finds the y value of the second SP
plot(sp(1),pp1,'o','MarkerFaceColor','m')
text(sp(1),pp1,'␣␣minima')  % Plots 'minima' at the first SP
plot(sp(2),pp2,'o','MarkerFaceColor','m')
text(sp(2),pp2,'␣␣maxima')  % Plots 'maxima' at the second SP
% Because I have one real root, I will use the
% first stationary point and the real root
root = double(solve(y==0))  % Finds the roots of y(x)
integral = abs(double(int(y,root(1),sp(1))))
                                     % Finds int of y between the root & first SP.
```

References

[1] MathWorks (2018)*MATLAB®Documentation (Online Help)]* [online], avaliable from: < https://uk.mathworks.com/help/matlab/index.html > [5 December 2018]

[2] A Pedcenko (2018)*An example of a project with the trace behind M-file* [online], available from: < https://cumoodle.coventry.ac.uk/pluginfile.php/2467358/mod_ resource/content/6/N_projectiles.m > [7 December 2018]

[3] *Matlab PC Coursework for week 5* [online], avaliable from: < https://cumoodle.coventry.ac.uk/pluginfile.php/2451903/mod_resource/content/2/Lab%2005%20Worksheet%20–%20Symbolic%20Math.pdf >

Used to help compose the LaTeX report:

[4] R Low (2007)*LaTeX: A cursory intoduction*, unpublished.

[5] JH Silverman (2007)*AMS-LaTeX Reference Card* [online], availiable from <http://www.math.brown.edu/ jhs/ReferenceCards/LaTeXRefCard.v2.0.pdf> [11 December 2018]