Buffon Needles and Noodles

Methum Kasthuriarachchi

March 17, 2021

1 Introduction

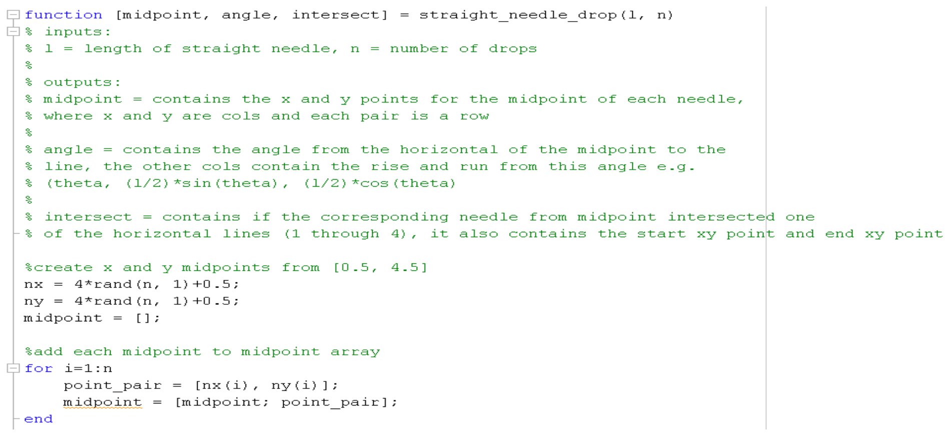
For this project, I wrote Matlab code to simulate the Buffon's Needle problem and its variant, Buffon's Noodle.

2 Buffon's Needles and Noodles Code

For all cases of the simulation, the grid was set up to be from 0 to 5 for both the x and y axis. The entire length of the needle/noodle for all simulations was 1.

2.1 Simulating dropping straight needles

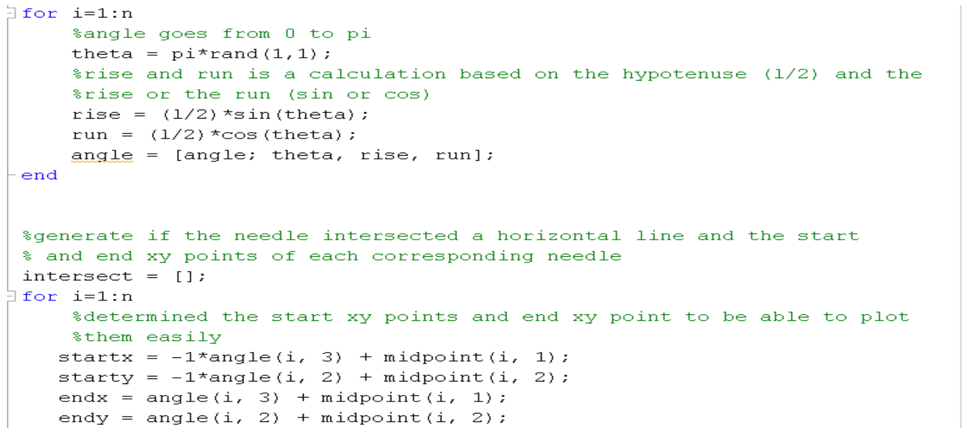
The function straight_needle_drop, shown in figure 1, generates the needles that will be animated in later as the points are connected. To create the points, I randomly picked the middle of the line to be from 0.5 to 4.5. I did this because I wanted to make sure that the line would still be able to hit the horizontal axis if it's midpoint happened to be at the lowest or possible y values.



```
function [midpoint, angle, intersect] = straight_needle_drop(l, n)
% inputs:
% l = length of straight needle, n = number of drops
%
% outputs:
% midpoint = contains the x and y points for the midpoint of each needle,
% where x and y are cols and each pair is a row
%
% angle = contains the angle from the horizontal of the midpoint to the
% line, the other cols contain the rise and run from this angle e.g.
% (theta, (l/2)*sin(theta), (l/2)*cos(theta)
%
% intersect = contains if the corresponding needle from midpoint intersected one
% of the horizontal lines (1 through 4), it also contains the start xy point and end xy point

%create x and y midpoints from [0.5, 4.5]
nx = 4*rand(n, 1)+0.5;
ny = 4*rand(n, 1)+0.5;
midpoint = [];

%add each midpoint to midpoint array
for i=1:n
    point_pair = [nx(i), ny(i)];
    midpoint = [midpoint; point_pair];
end
```

Figure 1: Create midpoints for straight needles

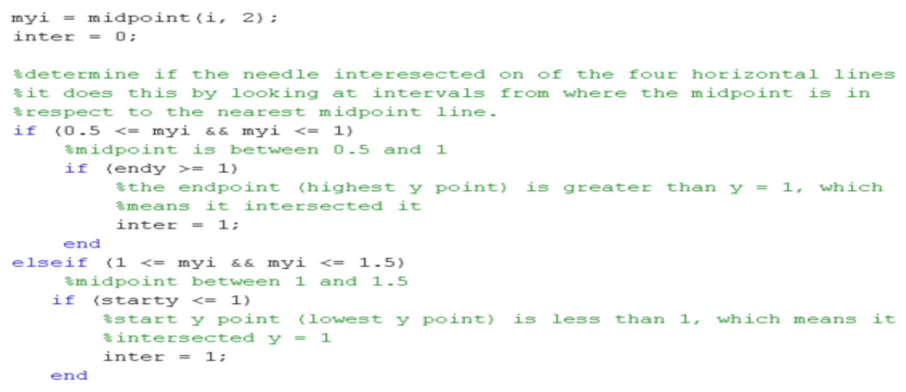
I would then generate an angle to orient the line. Through this angle, and because the lenght of the line is know, I can generate the x and y endpoints for each line. These endpoints are stored in the array intersect. I then check to see if the lines are intersected based on the middle point. A variable that contains whether the line was intersected is also stored in the intersect array. This can be seen in figure 2.



```
for i=1:n
    %angle goes from 0 to pi
    theta = pi*rand(1,1);
    %rise and run is a calculation based on the hypotenuse (l/2) and the
    %rise or the run (sin or cos)
    rise = (l/2)*sin(theta);
    run = (l/2)*cos(theta);
    angle = [angle; theta, rise, run];
end


%generate if the needle intersected a horizontal line and the start
% and end xy points of each corresponding needle
intersect = [];
for i=1:n
    %determined the start xy points and end xy point to be able to plot
    %them easily
   startx = -1*angle(i, 3) + midpoint(i, 1);
   starty = -1*angle(i, 2) + midpoint(i, 2);
   endx = angle(i, 3) + midpoint(i, 1);
   endy = angle(i, 2) + midpoint(i, 2);
```

Figure 2: Find rise and run through the angles, then determine x and y position through the midpoint.

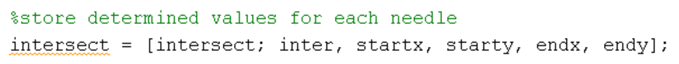
After that, I can determine the intersection points of the lines by checking the midpoint and the endpoints, as seen in figure 3 for each of the horizontal lines (y = 1, 2, 3, and 4).



```
myi = midpoint(i, 2);
inter = 0;

%determine if the needle interesected on of the four horizontal lines
%it does this by looking at intervals from where the midpoint is in
%respect to the nearest midpoint line.
if (0.5 <= myi && myi <= 1)
    %midpoint is between 0.5 and 1
    if (endy >= 1)
        %the endpoint (highest y point) is greater than y = 1, which
        %means it intersected it
        inter = 1;
    end
elseif (1 <= myi && myi <= 1.5)
    %midpoint between 1 and 1.5
   if (starty <= 1)
       %start y point (lowest y point) is less than 1, which means it
       %intersected y = 1
       inter = 1;
   end
```

Figure 3: Find rise and run through the angles, then determine x and y position through the midpoint.

Both start and end point's x and y values, as well as if the line intersected is then stored in the intersect array, as show in figure 4.



```
%store determined values for each needle
intersect = [intersect; inter, startx, starty, endx, endy];
```

Figure 4: Find rise and run through the angles, then determine x and y position through the midpoint.

These points can then be combined using the plot command in Matlab to create the needles, seen in figure 2.1. I decided to make the needles blue if they intersected the red horizontal lines, and black if they didn't intersect.

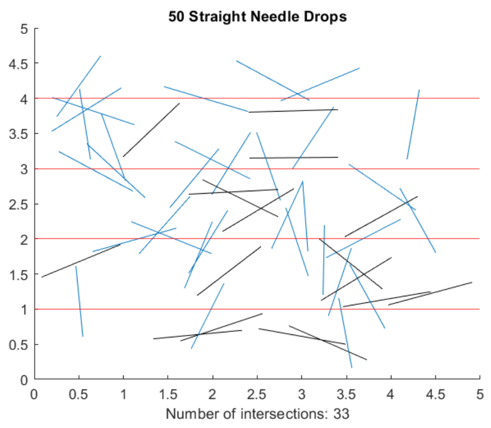


Figure 5: Dropping 50 straight needles. Blue needles means they intersected; black lines means no intersection.

2.2 Simulating dropping needles bent once

The function bent_needle_drop, shown in figure 6, generates a needle bent once. This can both create needles with an angle between the legs to be a constant chosen by the user, or a random angle from $\frac{\pi}{6}$ to $\frac{5\pi}{6}$, as seen in 7 which contains the bent needle with a constant angle and 8 which shows the bent needle with an inbetween angle that is random. To create these needles I first started out by choosing the midpoints, which is the same as what I did in figure 1.

```
function [bent_intersect] = bent_needle_drop(l, bend, n)
% inputs:
% l = length of entire bent needle,
% bend = sets acute angle between needle to a constant (if bend = 0, acute angle random from pi/6 to 5pi/6),
% n = # of bent needles
%
% output:
% bent_intersect = first column contains if the needle intersected the
% horizontal line (1 - yes, 0 - no), the 2 next columns contain the x and
% y pos for the left most side, the next 2 columns contain the middle x
% and y pos, and the next 2 columns contain the x and y pos for the right
% most side
```

Figure 6: The function that drops the bent needles. In the comments includes the inputs, and the outputs of the function.

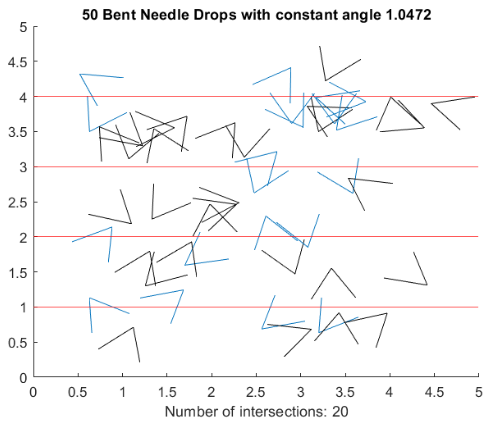


Figure 7: Dropping bent needle with an inbetween angle that is constant.

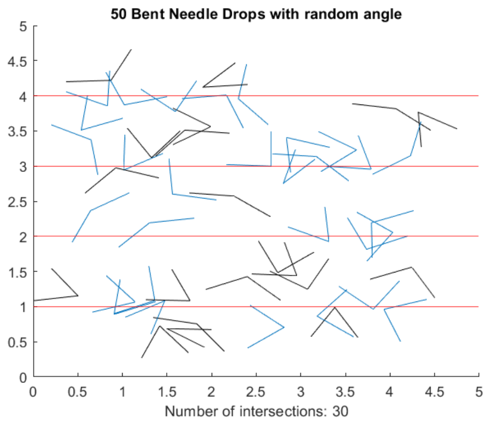
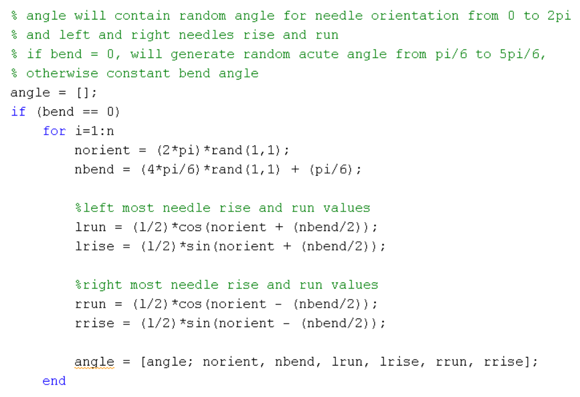


Figure 8: Dropping bent needle with an inbetween angle that is random.

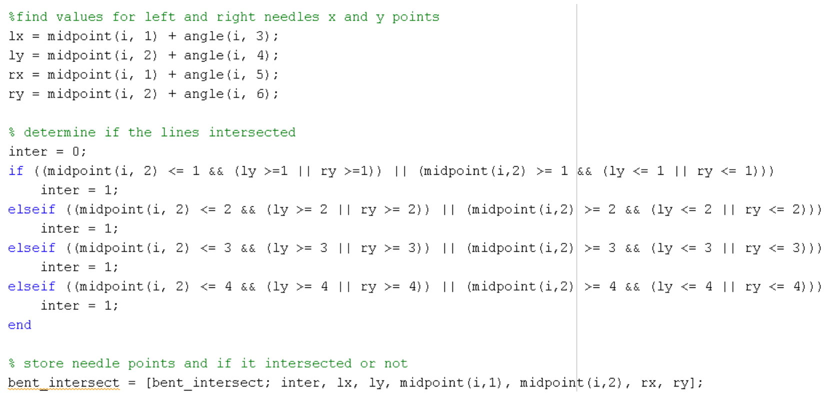
After generating the midpoint (the vertex) of the needle, I needed to create two other points. To do this I generated two angles, one to orient it as if there was a line through the middle, similar to figure 2, but also another angle that determines the random inbetween angle, as seen in figure 9.



```
% angle will contain random angle for needle orientation from 0 to 2pi
% and left and right needles rise and run
% if bend = 0, will generate random acute angle from pi/6 to 5pi/6,
% otherwise constant bend angle
angle = [];
if (bend == 0)
    for i=1:n
        norient = (2*pi)*rand(1,1);
        nbend = (4*pi/6)*rand(1,1) + (pi/6);

        %left most needle rise and run values
        lrun = (1/2)*cos(norient + (nbend/2));
        lrise = (1/2)*sin(norient + (nbend/2));

        %right most needle rise and run values
        rrun = (1/2)*cos(norient - (nbend/2));
        rrise = (1/2)*sin(norient - (nbend/2));

        angle = [angle; norient, nbend, lrun, lrise, rrun, rrise];
    end
```

Figure 9: Generate random angles and rise and run based on it for the two other points for the bent needle.

After that I can determine the points for those two lines, and whether they intersect as seen in figure 10.



```
%find values for left and right needles x and y points
lx = midpoint(i, 1) + angle(i, 3);
ly = midpoint(i, 2) + angle(i, 4);
rx = midpoint(i, 1) + angle(i, 5);
ry = midpoint(i, 2) + angle(i, 6);

% determine if the lines intersected
inter = 0;
if ((midpoint(i, 2) <= 1 && (ly >=1 || ry >=1)) || (midpoint(i,2) >= 1 && (ly <= 1 || ry <= 1)))
    inter = 1;
elseif ((midpoint(i, 2) <= 2 && (ly >= 2 || ry >= 2)) || (midpoint(i,2) >= 2 && (ly <= 2 || ry <= 2)))
    inter = 1;
elseif ((midpoint(i, 2) <= 3 && (ly >= 3 || ry >= 3)) || (midpoint(i,2) >= 3 && (ly <= 3 || ry <= 3)))
    inter = 1;
elseif ((midpoint(i, 2) <= 4 && (ly >= 4 || ry >= 4)) || (midpoint(i,2) >= 4 && (ly <= 4 || ry <= 4)))
    inter = 1;
end

% store needle points and if it intersected or not
bent_intersect = [bent_intersect; inter, lx, ly, midpoint(i,1), midpoint(i,2), rx, ry];
```

Figure 10: Determine the x and y points for the other two points to create a bent needle, and determine whether it intersects one of the horizontal lines or not. Then store those values to be used in array bent_intersect.

2.3 Simulating dropping needles bent twice

The function drop_noodle, generates the points and drops needles bent twice randomly and counts the number of intersections, but also if the intersection is positive or negative based on the preferred end (seen in figure 11).

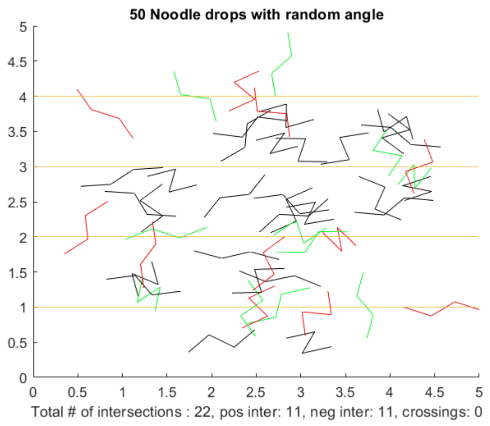
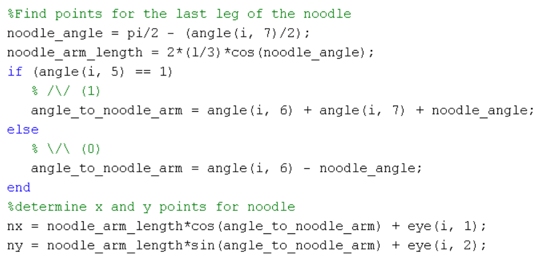


Figure 11: Drop 50 random noodles. Green noodles account for the positive intersections, red noodles count for the negative intersections, blue intersections are meant to count when the preferred end lands on a horizontal line exactly, and black lines represent no intersections. The crossing values is created by summing the positive and negative intersections.

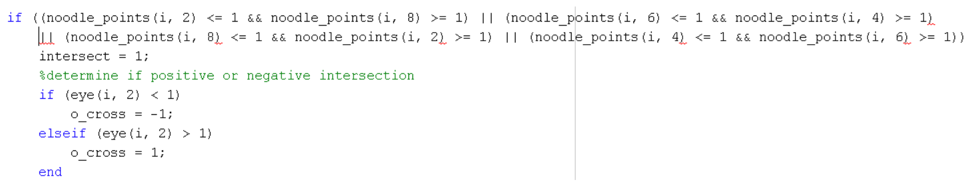
To create this, I started out using the same methods to make the bent noodle, seen in section 2.2. To create the third line I had to determine the angle based on the orientation and in between angle, as seen in figure 12.



```
%Find points for the last leg of the noodle
noodle_angle = pi/2 - (angle(i, 7)/2);
noodle_arm_length = 2*(1/3)*cos(noodle_angle);
if (angle(i, 5) == 1)
   % /\/ (1)
   angle_to_noodle_arm = angle(i, 6) + angle(i, 7) + noodle_angle;
else
   % \/\ (0)
   angle_to_noodle_arm = angle(i, 6) - noodle_angle;
end
%determine x and y points for noodle
nx = noodle_arm_length*cos(angle_to_noodle_arm) + eye(i, 1);
ny = noodle_arm_length*sin(angle_to_noodle_arm) + eye(i, 2);
```

Figure 12: Determine the x and y points for the third line that will be connected to the bent needle.

I then have to check for the intersection based on the end points of the noodle and whether there is a positive or negative intersection based on the preferred end, as seen in figure 13.



```
if ((noodle_points(i, 2) <= 1 && noodle_points(i, 8) >= 1) || (noodle_points(i, 6) <= 1 && noodle_points(i, 4) >= 1)
    || (noodle_points(i, 8) <= 1 && noodle_points(i, 2) >= 1) || (noodle_points(i, 4) <= 1 && noodle_points(i, 6) >= 1))
    intersect = 1;
    %determine if positive or negative intersection
    if (eye(i, 2) < 1)
        o_cross = -1;
    elseif (eye(i, 2) > 1)
        o_cross = 1;
    end
```

Figure 13: Determine if the endpoints of the noodle intersected any of the horizontal lines, and then check if the intersection is positive or negative.

2.4 Conjecture for average crossing number for needles bent twice

The average crossing number for the noodles (Number of positive intersections - Number of negative intersections) should be 0. Given that a random amount of noodles intersect certain horizontal lines. Because the intersections are random, there should be roughly half that have the preferred end over the top of the line and half that are under the line. Giving us an equal number of positive and negative intersections, meaning that the average crossings for the noodles should equal 0, which can be seen in figure 11.