INDEX

1. Simulate a Random Walk Problem.

One of the important application of random numbers is a drunkard walk. A drunkard is trying to go in a direction (say y-axis in xy plane). But sometimes he moves in forward direction and sometimes left, right or backward direction. Random walks has many applications in the field of physics. Brownian motion of molecules is like random walk. Probabilities of drunkard's steps are given as follows:

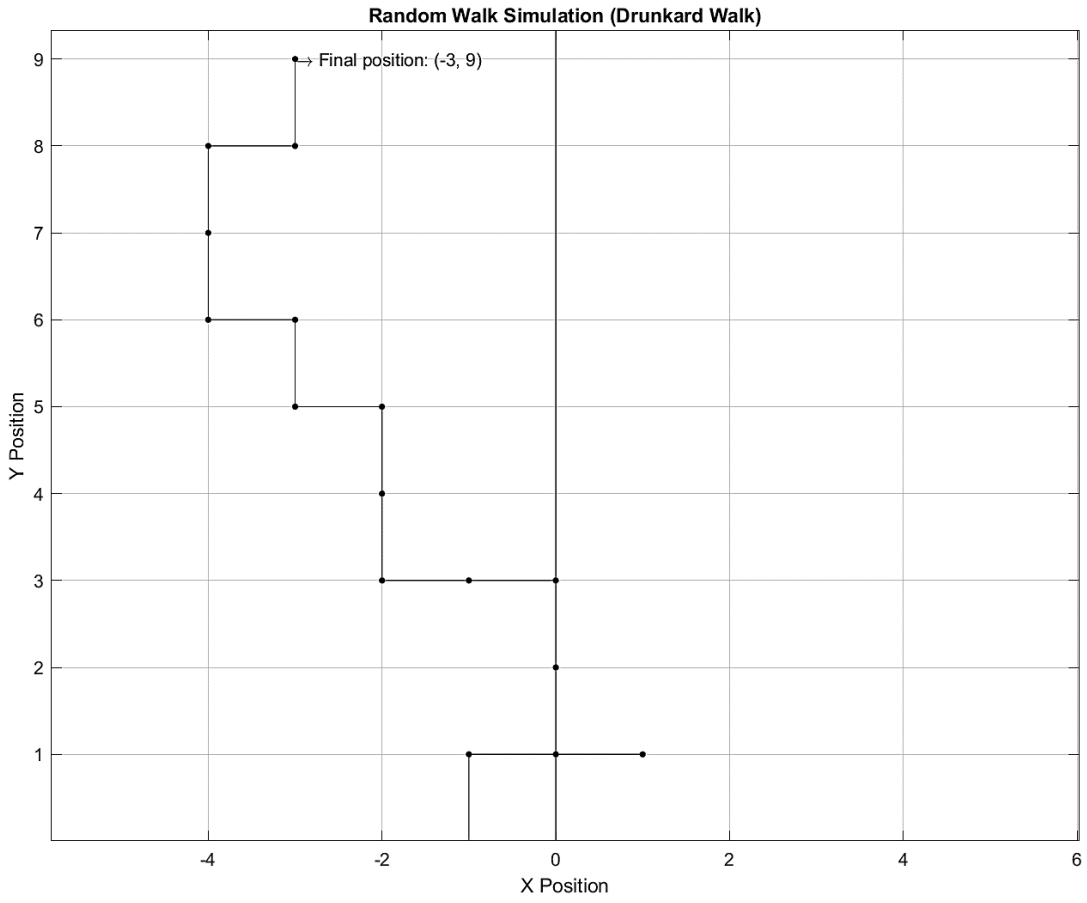
Probability of moving forward $= 0.5$	Probability of moving backward $= 0.1$
Probability of moving left $= 0.2$	Probability of moving right $= 0.2$

Code:

```
clc;
clear all;

% Parameters
num_steps = 20;             % Number of steps
start_pos = [0, 0];         % Starting position (x, y)
curr_position = start_pos;  % Initialize current position

% Mapping of random numbers to directions:
% Forward (F): 0, 1, 2, 3, 4
% Backward (B): 5
% Left (L): 6, 7
% Right (R): 8, 9
directions_map = containers.Map({0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, {'F', 'F', 'F', 'F',
'F', 'B', 'L', 'L', 'R', 'R'});

% Store positions to plot later
x_pos = zeros(1, num_steps + 1);
y_pos = zeros(1, num_steps + 1);
x_pos(1) = start_pos(1);
y_pos(1) = start_pos(2);

% Random walk simulation
for i = 1:num_steps
    % Generate a random number between 0 and 9
    rand_num = randi([0, 9]);

    % Determine direction based on random number
    direction = directions_map(rand_num);

    % Update position based on direction
    switch direction
        case 'F'
            curr_position(2) = curr_position(2) + 1;  % Move forward (Y-axis +1)
        case 'B'
            curr_position(2) = curr_position(2) - 1;  % Move backward (Y-axis -1)
        case 'L'
            curr_position(1) = curr_position(1) - 1;  % Move left (X-axis -1)
        case 'R'
            curr_position(1) = curr_position(1) + 1;  % Move right (X-axis +1)
    end

    % Store new position
    x_pos(i + 1) = curr_position(1);
    y_pos(i + 1) = curr_position(2);
end
```

```
% Plot the random walk
figure;
plot(x_pos, y_pos, '-ko', 'MarkerSize', 3, 'MarkerFaceColor', 'k', 'MarkerEdgeColor',
'k');
hold on;

% Display the final position
text(curr_position(1), curr_position(2), sprintf('\\rightarrow Final position: (%d,
%d)', curr_position(1), curr_position(2)));

% Draw X and Y axes
line([-max(abs(x_pos))-1, max(abs(x_pos))+1], [0, 0], 'Color', 'black', 'LineWidth',
0.5);  % X-axis
line([0, 0], [-max(abs(y_pos))-1, max(abs(y_pos))+1], 'Color', 'black', 'LineWidth',
0.5);  % Y-axis

xlabel('X Position');
ylabel('Y Position');
title('Random Walk Simulation (Drunkard Walk)');
grid on;
axis equal;
```

Output:



2. Simulate the following:

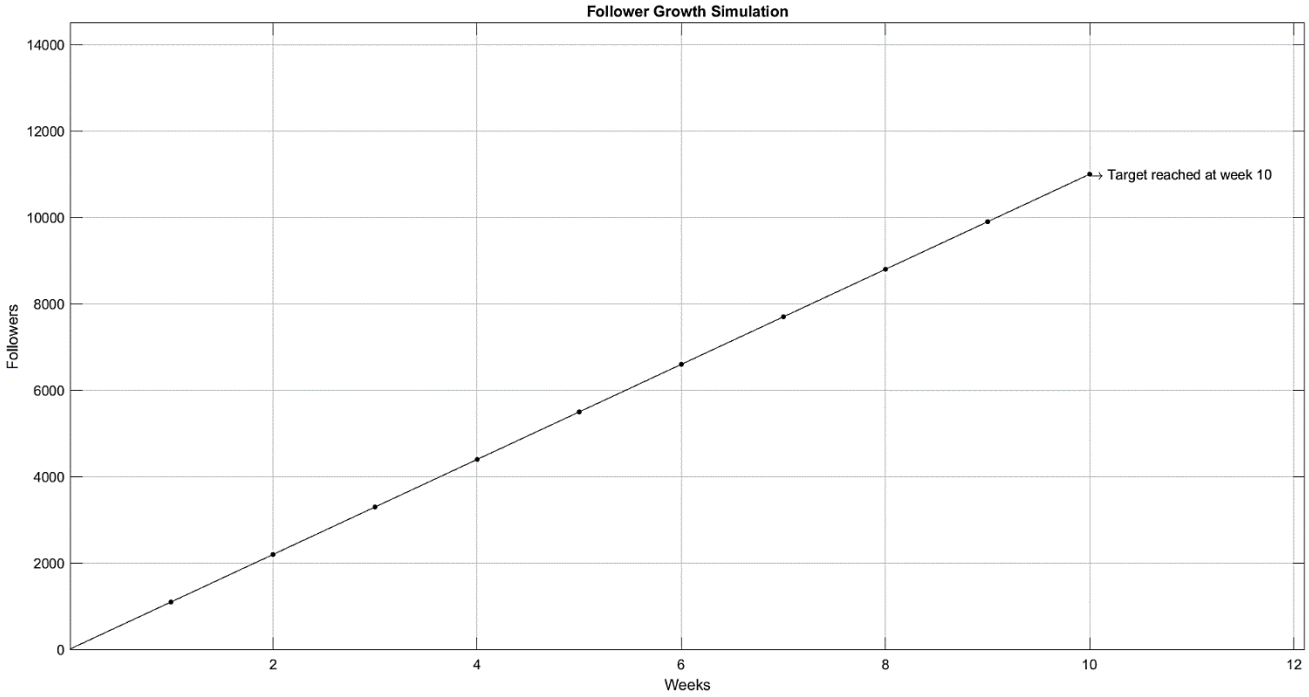
A social media influencer decides to open a new page and her target is to reach 10k followers in 10 weeks. Given her past experience, she assumes that each week she will get 1.5k new followers that had never followed the page and of her current followers she believes 500 will stop following the page each week. However, 100 of those that left the page in the past will join again each week. Will she reach her target?

Code:

```
clc;
clear all;

% Simulation parameters
total_weeks = 10;                 % Total weeks
target_followers = 10000;         % Target followers (10k)
new_followers_per_week = 1500;    % New followers per week
leaving_followers_per_week = 500; % Followers who leave each week
rejoin_followers_per_week = 100;  % Followers who rejoin each week

% Store followers and weeks to plot later
followers = zeros(1, total_weeks + 1);
weeks = 0:10;

% Initial current followers
curr_followers = 0;

% Iterate over each week
for week = 1:total_weeks
    % Update the number of followers for the current week
    curr_followers = curr_followers + new_followers_per_week -
leaving_followers_per_week + rejoin_followers_per_week;
    followers(week + 1) = curr_followers;

    % Check if the target has been reached
    if curr_followers >= target_followers
        reached_week = week;
    end
end

% Plot the growth graph
plot(weeks, followers, '-ko', 'MarkerSize', 3, 'MarkerFaceColor', 'k',
'MarkerEdgeColor', 'k');
xlabel('Weeks');
ylabel('Followers');
title('Follower Growth Simulation');
grid on;

% Check if the target has been reached
if curr_followers >= target_followers
    fprintf('Target reached\n');
    text(reached_week, followers(reached_week + 1), sprintf('\\rightarrow Target reached
at week %d ', reached_week));
else
    fprintf('Target not reached.\n');
end
```

Output:

```
Target reached.
```



3. Simulation of pursuit-evasion problem:

The bomber continues to fly along a predetermined path and the fighter has to change its direction to keep pointer towards the bomber. Let us consider, while the bomber and fighter or pursuit fly in the same horizontal plane. The Fighter speed S is constant at 20 km/min, while the bomber part is specified as a function of time as below:

$Time(t)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$xb(t)$	100	110	120	129	140	149	158	168	179	188	198	209	219	226	234	240
$yb(t)$	0	3	6	10	15	20	26	32	37	34	30	27	23	19	16	14

The fighter is at position xf, $yf(0,50)$ when it sights the bomber that is at time $t = 0$, the time at which pursuit begins. The fighter corrects his direction after a fixed interval of one minute, so as to point towards the bomber. If the distance between fighter and bomber is 10 km or less, then the fighter shoots the bomber. If the distance between fighter and bomber is not 10 km or less than 10 km in 15 minutes (One example we use 15 min) then the bomber is escaped.

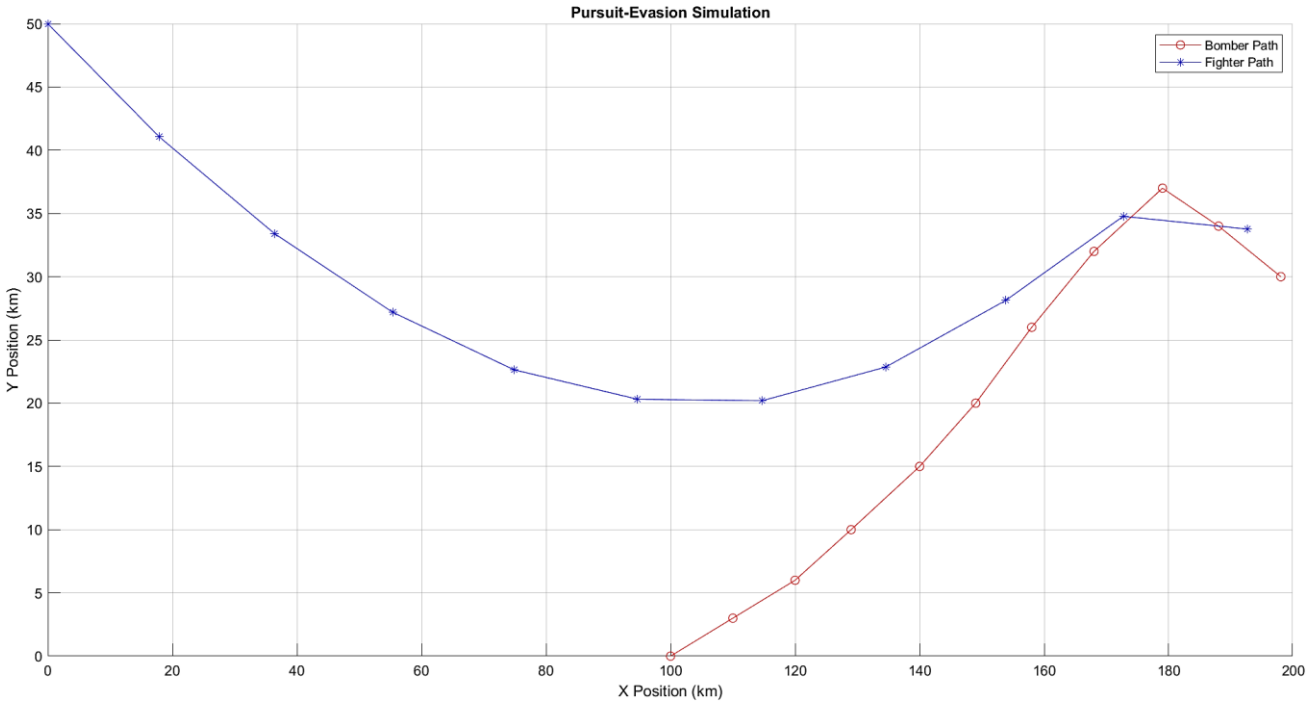
Code:

```
clc;
clear;

% Bomber path data as per time
time = 0:15; % Time in minutes
xb = [100 110 120 129 140 149 158 168 179 188 198 209 219 226 234 240]; % Bomber X position
yb = [0 3 6 10 15 20 26 32 37 34 30 27 23 19 16 14]; % Bomber Y position

% Initialize array to store Fighter's positions
xf = zeros(1, length(time));
yf = zeros([1, length(time)]);

%  Fighter's Parameters
xf(1) = 0;  % Fighter's X position (Initial)
yf(1) = 50; % Fighter's Y position (Initial)
S = 20;  % Fighter speed (S) in km/min
min_distance = 10; % Minimum distance to shoot the bomber

% Flag to check if bomber is intercepted
intercepted = false;

% Main loop for pursuit simulation
for t = 1:length(time)
    % Calculate the distance between fighter and bomber at time t - 1
    distance = sqrt((yb(t) - yf(t))^2 + (xb(t) - xf(t))^2); % Euclidean distance

    % Display current fighter position and distance
    fprintf('Time: %d min, Fighter Position: (%.2f, %.2f), Distance: %.2f km\n', t-1,
xf(t), yf(t), distance);

    % Check if fighter is close enough to shoot the bomber
    if distance <= min_distance
        fprintf('Pursuit end. Bomber intercepted at time t = %d minutes, distance = %.2f
km\n', t-1, distance);
        intercepted = true;
        break;
    end
```

```
    % Update fighter's position (move towards the bomber)
    xf(t + 1) = xf(t) + S * (xb(t) - xf(t)) / distance;
    yf(t + 1) = yf(t) + S * (yb(t) - yf(t)) / distance;
end

% Check if bomber escaped
if ~intercepted
    fprintf('Bomber escaped\n');
end

% Plot the bomber and fighter paths
figure;
hold on;
plot(xb(1:t), yb(1:t), '-ro', 'DisplayName', 'Bomber Path');
plot(xf(1:t), yf(1:t), '-b*', 'DisplayName', 'Fighter Path');
legend show;
xlabel('X Position (km)');
ylabel('Y Position (km)');
title('Pursuit-Evasion Simulation');
grid on;
```

Output:

```
Time: 0 min, Fighter Position: (0.00, 50.00), Distance: 111.80 km
Time: 1 min, Fighter Position: (17.89, 41.06), Distance: 99.66 km
Time: 2 min, Fighter Position: (36.37, 33.42), Distance: 88.01 km
Time: 3 min, Fighter Position: (55.38, 27.19), Distance: 75.60 km
Time: 4 min, Fighter Position: (74.85, 22.64), Distance: 65.59 km
Time: 5 min, Fighter Position: (94.72, 20.31), Distance: 54.28 km
Time: 6 min, Fighter Position: (114.72, 20.20), Distance: 43.67 km
Time: 7 min, Fighter Position: (134.54, 22.85), Distance: 34.69 km
Time: 8 min, Fighter Position: (153.83, 28.13), Distance: 26.69 km
Time: 9 min, Fighter Position: (172.69, 34.78), Distance: 15.33 km
Time: 10 min, Fighter Position: (192.67, 33.76), Distance: 6.53 km
Pursuit end. Bomber intercepted at time t = 10 minutes, distance = 6.53 km
```



4. Simulation of queuing system:

The distribution of inter arrival times in a single server model is:

T:	1	2	3
f(t):	1/4	1/2	1/4

The distribution of service time is:

S:	1	2	3
f(s):	1/2	1/4	1/4

Complete the following table, using two digits random numbers 12, 40, 48, 93, 61, 17, 55, 21, 85, 88 to generate arrival times and 54, 90, 18, 38, 16, 87, 91, 41, 54, 11 to generate the corresponding service times.

Arrival number	Arrival Time (AT + T)	Time service begins (SB)	Time service ends (SB + S = SE)	Waiting time in Queue

Code:

```
clc;
clear;

% Given Data
T = [1, 2, 3]; % Possible inter-arrival times
f_t = [0.25, 0.50, 0.25]; % Probability distribution of inter-arrival times
S = [1, 2, 3]; % Possible service times
f_s = [0.50, 0.25, 0.25]; % Probability distribution of service times

% Random numbers for inter-arrival times and service times
rand_inter_arrival = [12, 40, 48, 93, 61, 17, 55, 21, 85, 88];
rand_service = [54, 90, 18, 38, 16, 87, 91, 41, 54, 11];

% Map random numbers to inter-arrival times
inter_arrival_time = zeros(1, length(rand_inter_arrival));
for i = 1:length(rand_inter_arrival)
    if rand_inter_arrival(i) <= 24
        inter_arrival_time(i) = 1;
    elseif rand_inter_arrival(i) <= 74
        inter_arrival_time(i) = 2;
    else
        inter_arrival_time(i) = 3;
    end
end

% Map random numbers to service times
service_time = zeros(1, length(rand_service));
for i = 1:length(rand_service)
    if rand_service(i) <= 49
        service_time(i) = 1;
    elseif rand_service(i) <= 74
        service_time(i) = 2;
    else
        service_time(i) = 3;
    end
end

% Initializing arrays for calculations
arrival_time = zeros(1, length(rand_inter_arrival)); % Arrival times (AT)
service_begin = zeros(1, length(rand_inter_arrival)); % Service begin (SB)
```

```
service_end = zeros(1, length(rand_inter_arrival)); % Service end (SE)
waiting_time = zeros(1, length(rand_inter_arrival)); % Waiting time (WT)

% First customer
arrival_time(1) = inter_arrival_time(1);
service_begin(1) = arrival_time(1);
service_end(1) = service_begin(1) + service_time(1);
waiting_time(1) = 0;

% Loop for subsequent customers
for i = 2:length(rand_inter_arrival)
    arrival_time(i) = arrival_time(i-1) + inter_arrival_time(i);
    service_begin(i) = max(arrival_time(i), service_end(i-1));
    service_end(i) = service_begin(i) + service_time(i);
    waiting_time(i) = service_begin(i) - arrival_time(i);
end

% Display the results
disp('Arrival number   Arrival Time (AT+T)   Time Service Begin (SB)   Time Service End
(SE=SB+S)   Waiting time in Queue');
disp('--------------   -------------------   ----------------------   -----------------
---------   --------------------')
for i = 1:length(rand_inter_arrival)
    fprintf('%8d %16d %25d %25d %26d\n', i, arrival_time(i), service_begin(i),
service_end(i), waiting_time(i));
end
```

Output:

Arrival number	Arrival Time (AT+T)	Time Service Begin (SB)	Time Service End (SE=SB+S)	Waiting time in Queue
1	1	1	3	0
2	3	3	6	0
3	5	6	7	1
4	8	8	9	0
5	10	10	11	0
6	11	11	14	0
7	13	14	17	1
8	14	17	18	3
9	17	18	20	1
10	20	20	21	0

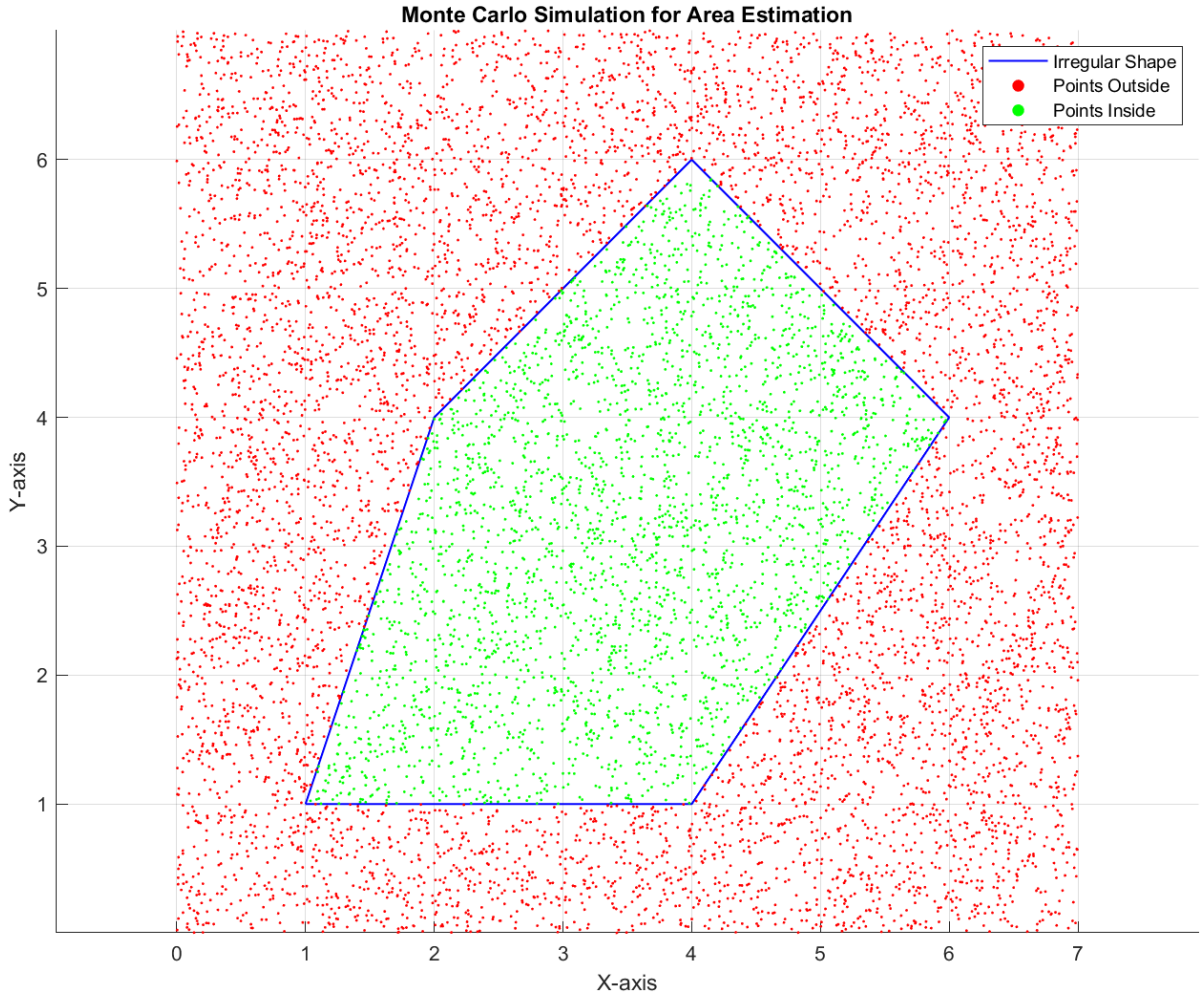
5. Write a program to determine i) Area of an irregular figure ii) Value of an integration iii) The value of "pi" by Monte-Carlo method.

i) Code:

```
clc;
clear all;

% Define the vertices of the irregular figure
x = [1, 4, 6, 4, 2, 1]; % X-coordinates of the irregular shape
y = [1, 1, 4, 6, 4, 1]; % Y-coordinates of the irregular shape

% Number of random points to generate
num_points = 10000;

% Define the bounding box around the irregular figure
min_x = min(x) - 1;
max_x = max(x) + 1;
min_y = min(y) - 1;
max_y = max(y) + 1;

% Generate random points within the bounding box
random_x = min_x + (max_x - min_x) * rand(1, num_points);
random_y = min_y + (max_y - min_y) * rand(1, num_points);

% Check if the random points are inside the irregular figure
in_polygon = inpolygon(random_x, random_y, x, y);

% Estimate the area
% Calculate the area of the bounding box
bounding_box_area = (max_x - min_x) * (max_y - min_y);
% Calculate the ratio of points inside the figure to total points
area_ratio = (sum(in_polygon) / num_points);
area_estimate = area_ratio * bounding_box_area;

% Display the calculated area
fprintf('The estimated area of the irregular figure is %.4f square units.\n',
area_estimate);

% Plot the figure and the random points
figure;
hold on;
plot(x, y, 'b-', 'LineWidth', 1); % Plot the boundary of the irregular figure
% Plot points outside the figure
scatter(random_x(~in_polygon), random_y(~in_polygon), 2, 'red', 'filled');
% Plot points inside the figure
scatter(random_x(in_polygon), random_y(in_polygon), 2, 'green', 'filled');
title('Monte Carlo Simulation for Area Estimation');
xlabel('X-axis');
ylabel('Y-axis');
grid on;
axis equal;
legend('Irregular Shape', 'Points Outside', 'Points Inside');
hold off;
```

Output:

```
The estimated area of the irregular figure is 14.3913 square units.
```

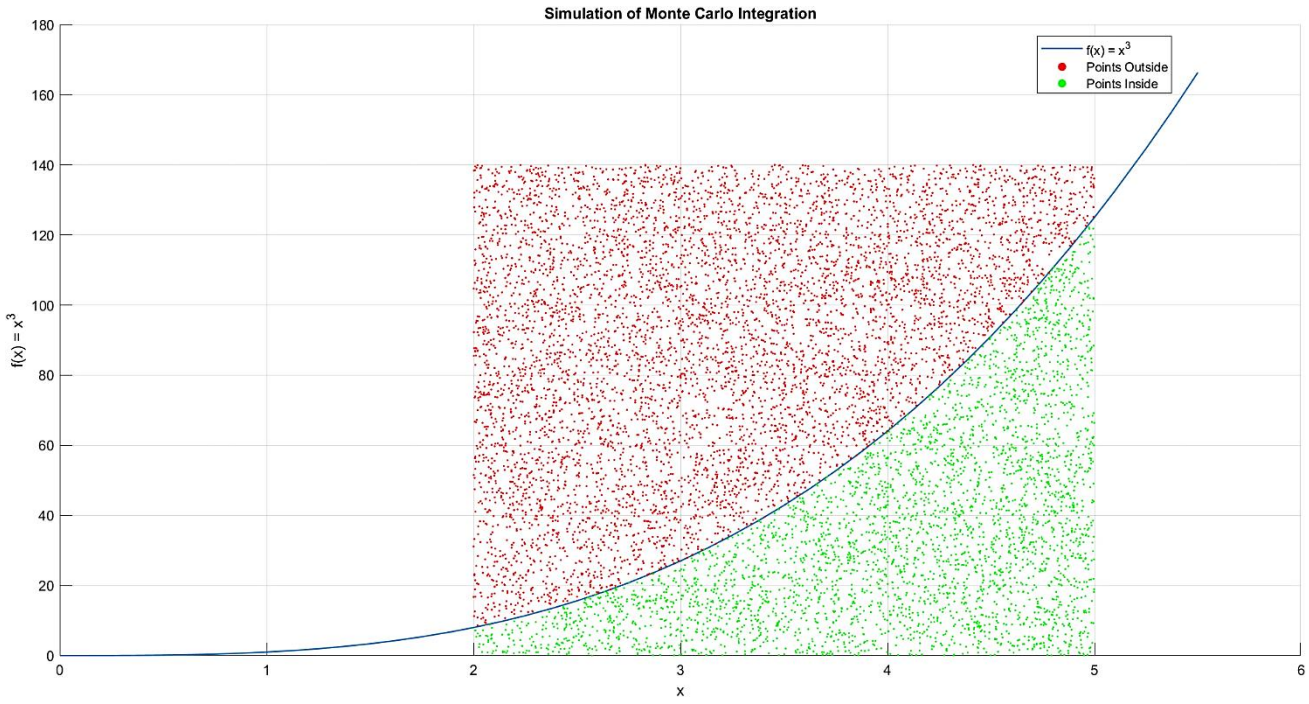


ii) Code:

```
clc;
clear all;

% Define the vertices of the irregular figure
x = [1, 4, 6, 4, 2, 1]; % X-coordinates of the irregular shape
y = [1, 1, 4, 6, 4, 1]; % Y-coordinates of the irregular shape

% Number of random points to generate
num_points = 10000;

% Define the bounding box around the irregular figure
min_x = min(x) - 1;
max_x = max(x) + 1;
min_y = min(y) - 1;
max_y = max(y) + 1;

% Generate random points within the bounding box
random_x = min_x + (max_x - min_x) * rand(1, num_points);
random_y = min_y + (max_y - min_y) * rand(1, num_points);

% Check if the random points are inside the irregular figure
in_polygon = inpolygon(random_x, random_y, x, y);

% Estimate the area
% Calculate the area of the bounding box
bounding_box_area = (max_x - min_x) * (max_y - min_y);
% Calculate the ratio of points inside the figure to total points
area_ratio = (sum(in_polygon) / num_points);
area_estimate = area_ratio * bounding_box_area;

% Display the calculated area
fprintf('The estimated area of the irregular figure is %.4f square units.\n',
area_estimate);

% Plot the figure and the random points
figure;
hold on;
plot(x, y, 'b-', 'LineWidth', 1); % Plot the boundary of the irregular figure
% Plot points outside the figure
scatter(random_x(~in_polygon), random_y(~in_polygon), 2, 'red', 'filled');
% Plot points inside the figure
scatter(random_x(in_polygon), random_y(in_polygon), 2, 'green', 'filled');
title('Monte Carlo Simulation for Area Estimation');
xlabel('X-axis');
ylabel('Y-axis');
grid on;
axis equal;
legend('Irregular Shape', 'Points Outside', 'Points Inside');
hold off;
```

Output:

```
Estimated value of the integral using Monte Carlo is 153.2580
```

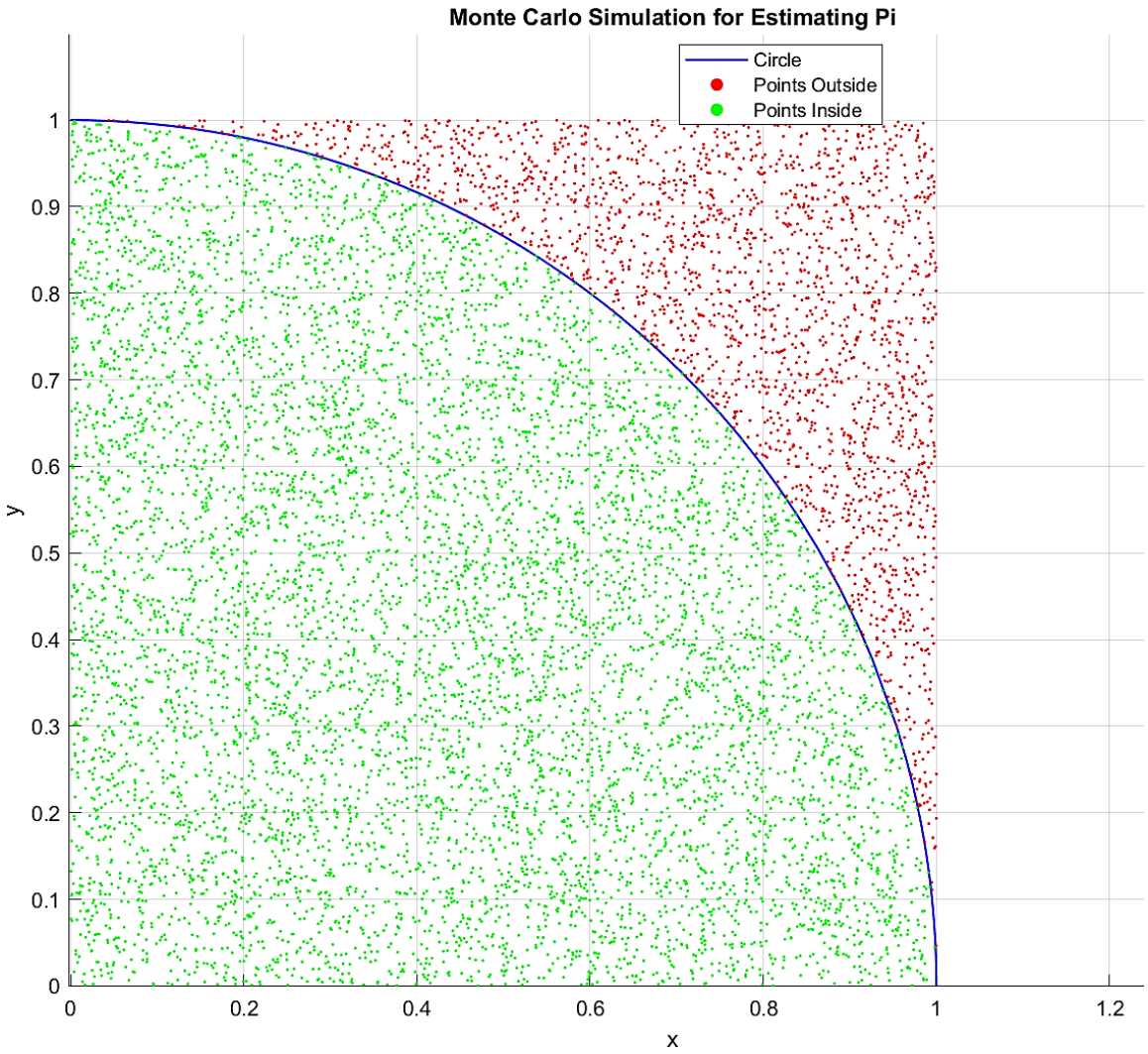


iii) Code:

```
clc;
clear all;

% Define the function of circle
f = @(x) sqrt(1 - x.^2);  % x^2 + y^2 = 1 or, y = sqrt(1 - x^2)

% Number of random points to generate
num_points = 10000;

% Generate random points between 0 and 1
random_x = rand(1, num_points);
random_y = rand(1, num_points);

% Initialize inside_circle
inside_circle = false(1, num_points);

% Determine if the random points are inside the 1st quadrant of circle
for i = 1:num_points
    x = random_x(i);
    y = random_y(i);

    % Check if the point (x, y) is inside the 1st quadran of circle
    if x^2 + y^2 <= 1
        inside_circle(i) = true;
    end
end

% Estimate the value of Pi
pi_estimate = (sum(inside_circle) / num_points) * 4;

% Display the result
fprintf('Estimated value of Pi using Monte Carlo is %.5f\n', pi_estimate);

% Plotting the simulation
figure;
hold on;
fplot(f, [0 1], 'b-', 'LineWidth', 1);  % Plot the 1st quadrant of circle
% Plot points outside the circle
scatter(random_x(~inside_circle), random_y(~inside_circle), 2, 'red', 'filled');
% Plot points inside the circle
scatter(random_x(inside_circle), random_y(inside_circle), 2, 'green', 'filled');
title('Monte Carlo Simulation for Estimating Pi');
xlabel('x');
ylabel('y');
legend('Circle', 'Points Outside', 'Points Inside');
grid on;
axis equal;
hold off;
```

Output:

```
Estimated value of Pi using Monte Carlo is 3.13520
```



6. Write a program to generate random number using i) Mid-square method and ii) Linear Congruence Method

i) Code:

```
clc;
clear all;

% Parameters
seed = 5673;     % Initial seed value
digits = 4;      % Number of digits in the seed value
n = input('Enter number of random numbers to be generated: ');

% Pre-allocate array to store random numbers
random_numbers = zeros(1, n);

% Initialize the current value with the seed
current_value = seed;

% Loop to generate random numbers
for i = 1:n
    % Square the current value
    squared_value = current_value^2;

    % Convert the squared value to a string
    squared_str = int2str(squared_value);

    % Extract the middle digits
    mid_start = floor((length(squared_str) - digits) / 2) + 1;
    middle_digits = squared_str(mid_start:(mid_start + digits - 1));

    % Convert the middle digits back to a number
    current_value = str2num(middle_digits);

    % store the current_value in the array
    random_numbers(i) = current_value;
end

 % Display the generated random numbers
 disp('Generated Random Numbers:');
 disp(random_numbers);
```

Output:

```
Enter number of random numbers to be generated: 10
Generated Random Numbers:
1829    3452    9163    9605    2560    5536    6472    8867    6236    8876
```

ii) Code:

```
clc;
clear all;

% Parameters of the Linear Congruential Generator (LCM)
a = input('Enter the value of multiplier (a): ');
b = input('Enter the value of increment (b): ');
m = input('Enter the value of modulus (m): ');
seed = input('Enter the value of seed: ');
n = input('Enter number of random numbers to be generated: ');

% Initialize array to store random numbers
r = zeros(1, n);

% Generate 1st random number using the LCM formula
r(1) = mod(a * seed + b, m);

% Generate other random numbers using the LCM formula
for i = 2:n
    r(i) = mod(a * r(i - 1) + b, m); % Linear Congruential Formula
end

% Display the generated random numbers
disp('Generated Random Numbers:');
disp(r);
```

Output:

```
Enter the value of multiplier (a): 5
Enter the value of increment (b): 3
Enter the value of modulus (m): 16
Enter the value of seed: 7
Enter number of random numbers to be generated: 10
Generated Random Numbers:
     6     1     8    11    10     5    12    15    14     9
```