Assignment IX

Mythreyi Ramesh MM15B022
Computer Modelling and Simulation MA5790

May 17, 2019

Question 1. Monte Carlo simulation of an assembly line.

Solution. The geometric criterion for fitting can be broken down as follows:

- Consider one of the circles to be centred at the origin, with radius r_1. Let the other circle (radius r_2) be centred at $(r, 0)$ because the distance between the two centres is given to be r. The diameter circle than can be constructed to touch the two circles internally is the maximum diameter of the bolt that can fit. (Figure 1 is an exaggerated representation of the problem.)

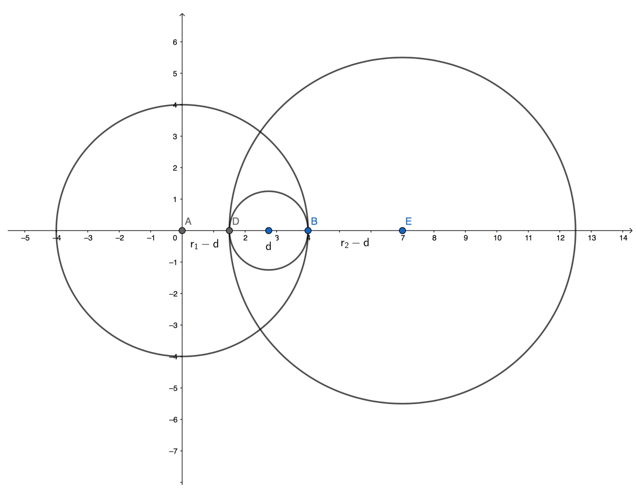


Figure 1: The three circles.

- There are three cases to be considered:

  1. $r > r_1 + r_2$: In this case, the bolt can never fit because the circles can't overlap.
  2. $r < |r_1 + r_2|$: In this case, the bolt will only fit if it is smaller than the smaller of the holes because the one of the circles lies inside the other.
  3. $|r_1 + r_2| < r < r_1 + r_2$: The maximum diameter of the bolt that can fit in horizontally is $r_1 + r_2 - r$. This is evident from Figure 1.

4. When $|r_1 + r_2| < r < r_1 + r_2$: The maximum diameter of the bolt that can fit in vertically is given by the length of the common vertical chord: $\dfrac{4 \times \text{area}(r_p, r_b, r)}{r}$.

The following MATLAB script was written to perform the simulation:

```

% Top plate X and Y, Base plat X and Y, Top Plate Radius, Base Plate
% Radius, Bolt Radius. distribution parameters.

t_X_mean = 100; t_Y_mean = 100;
b_X_mean = 100; b_Y_mean = 100;
r_p_mean = 25.15; r_b_mean = 25.25;
d_mean = 24.95;

t_X_std = 0.2/3; t_Y_std = 0.2/3;
b_X_std = 0.2/3; b_Y_std = 0.2/3;
r_p_std = 0.05/3; r_b_std = 0.05/3;
d_std = 0.21/3;

N_sim = 1000000;

% Generating the random numbers
t_X = t_X_std.*randn(N_sim,1)+t_X_mean;
t_Y = t_Y_std.*randn(N_sim,1)+t_Y_mean;
b_X = b_X_std.*randn(N_sim,1)+b_X_mean;
b_Y = b_Y_std.*randn(N_sim,1)+b_Y_mean;
d_bolt = d_std.*randn(N_sim,1)+d_mean;
r_p = r_p_std.*randn(N_sim,1)+r_p_mean;
r_b = r_b_std.*randn(N_sim,1)+r_b_mean;
r_diff = sqrt((t_X-b_X).^2 + (t_Y-b_Y).^2);

misfits = 0;

% The criteria for fitting
for i=1:N_sim
    if r_diff(i) > r_p(i)+r_b(i)
        misfits = misfits + 1;
    elseif r_diff(i) < abs(r_p(i)-r_b(i))
        if d_bolt(i) > min(r_p(i),r_b(i))
            misfits = misfits+1;
        end
    else
        semi_p = (r_p(i)+r_b(i)+r_diff(i))/2;
        tri_area = sqrt(semi_p*(semi_p-r_p(i))*(semi_p-r_b(i))*(semi_p-r_diff(i)));
        vert = 4*tri_area/r_diff(i);
        if d_bolt(i) > min((r_p(i)+r_b(i)-r_diff(i)),vert)
           misfits = misfits+1;
```

```
        end
    end
end

disp(misfits/N_sim)
```

The output comes out to be roughly 88500 defective parts per 1 million trials. This is adequately low because the tolerance levels are already acceptable. We can decrease it further by reducing the tolerance. □

Question 2. Markov Chains.

Solution. The following MATLAB script was written to obtain the steady state distribution theoretically:

```
clc; clear; close all;

p = 0.3;

N_sim = 1E6;

states = zeros(N_sim,1);
counts = zeros(4,1);

% starts at state 1
states(1) = 1;
counts(1) = 1;

for i=2:N_sim
   et = rand(); % this is a uniformly distributed random number
   switch states(i-1)
        case 1
            if (et < p)
               states(i) = 2;
               counts(2) = counts(2)+1;
            else
               states(i) = 1;
               counts(1) = counts(1)+1;
            end
        case 2
            if (et < p)
                states(i) = 1;
                counts(1) = counts(1)+1;
            elseif (et < 2*p)
                states(i) = 3;
                counts(3) = counts(3)+1;
            else
                states(i) = 2;
                counts(2) = counts(2)+1;
```

```
            end
        case 3
            if (et < p)
                states(i) = 2;
                counts(2) = counts(2)+1;
            elseif (et < 2*p)
                states(i) = 4;
                counts(4) = counts(4)+1;
            else
                states(i) = 3;
                counts(3) = counts(3)+1;
            end
        case 4
            if (et < p)
               states(i) = 3;
               counts(3) = counts(3)+1;
            else
               states(i) = 4;
               counts(4) = counts(4)+1;
            end
    end
end

disp(counts/N_sim)
```

The output generated is:

```
0.2492
0.2505
0.2508
0.2495
```

This can also be calculated by solving set of linear equations as follows:

$$P = \begin{bmatrix} 1-p & p & 0 & 0 \\ p & 1-2p & p & 0 \\ 0 & p & 1-2p & p \\ 0 & 0 & p & 1-p \end{bmatrix}$$

Let $\Pi = \begin{bmatrix} \pi_1 & \pi_2 & \pi_3 & \pi_4 \end{bmatrix}$ be the steady state values.

$$\begin{bmatrix} \pi_1 & \pi_2 & \pi_3 & \pi_4 \end{bmatrix} \times \begin{bmatrix} 1-p & p & 0 & 0 \\ p & 1-2p & p & 0 \\ 0 & p & 1-2p & p \\ 0 & 0 & p & 1-p \end{bmatrix} = \begin{bmatrix} \pi_1 & \pi_2 & \pi_3 & \pi_4 \end{bmatrix}$$

$$
\begin{aligned}
(1-p)\pi_1 + p\pi_2 &= \pi_1 & &\implies \pi_1 = \pi_2 \\
p\pi_1 + (1-2p)\pi_2 + p\pi_3 &= \pi_2 & &\implies \pi_2 = \pi_3 \\
p\pi_2 + (1-2p)\pi_3 + p\pi_4 &= \pi_3 & &\implies \pi_3 = \pi_4 \\
p\pi_3 + (1-p)\pi_4 &= \pi_4 & & \\
\text{Also, } \pi_1 + \pi_2 + \pi_3 + \pi_4 = 1 & & &
\end{aligned}
$$

Hence, $\pi_1 = \pi_2 = \pi_3 = \pi_4 = 0.25$. This value is independent of p.

□

Question 3. Monte Carlo evaluation of integrals.

Solution. Since the function has both positive and negative values, to implement method 2, we need to ensure that the two cases are handled separately. Figure 2 demonstrates the three categories the points can lie in. They could lie in the green region, in which case they add to the integral, or in the red region, in which case, they subtract from the integral or in the blue region, in which case, they are not added neither subtracted from the integral. The

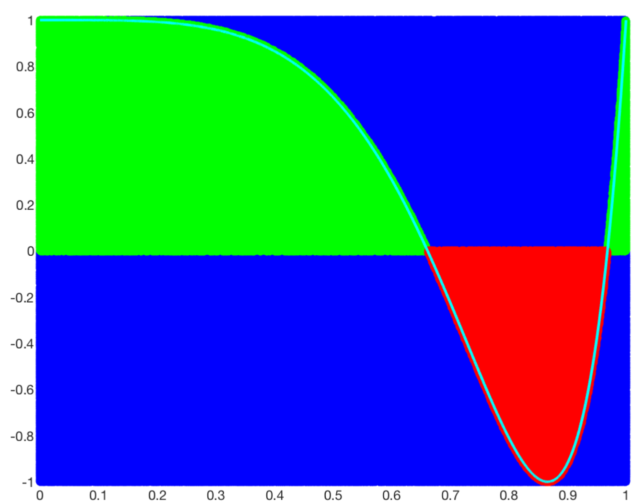


Figure 2: The three categories points could fall into.

following MATLAB script was written for the calculations:

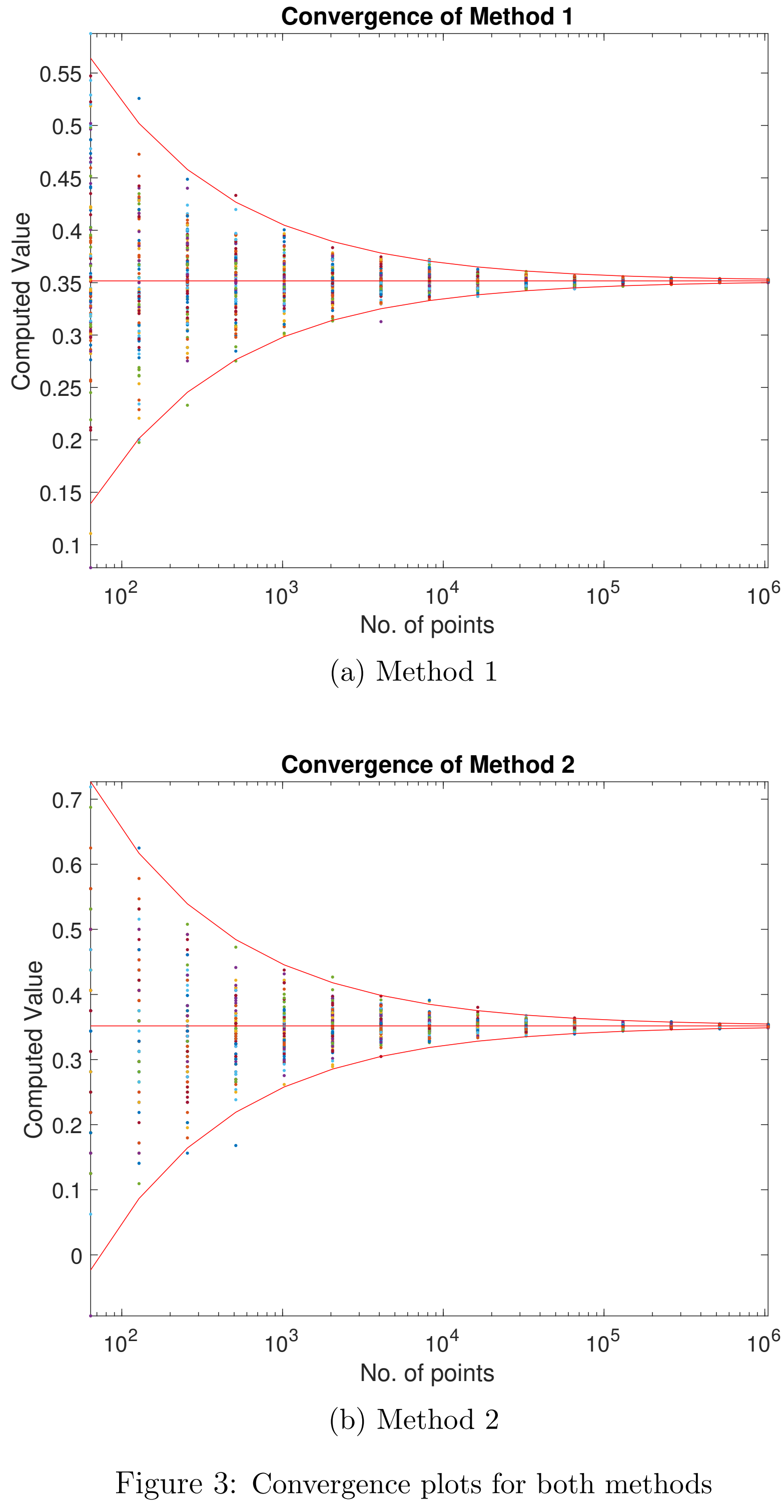
```
clc; clear; close all;
% Computing the integral of cos((2*pi)*sqrt(1-x_plot.^2)) from 0 to 1
N_sim = 100; % how many dots should be spread

% number of sample points in each simulation
min_index = 6;
max_index = 20;
K_index = min_index:max_index;
N = 2.^K_index;

```

```
fun = @(x) cos((2*pi)*sqrt(1-x.^2));
I_exact = quadgk(fun,0,1);

n_index = max_index-min_index+1;

N_matrix = zeros(n_index,N_sim); % for plotting
I_computed_1 = zeros(n_index,N_sim);  % for storing

for k =min_index:max_index
    N_matrix(k-min_index+1,:) = N(k-min_index+1);
    % multiple simulations for each run
    for j=1:N_sim
        y = rand(N(k-min_index+1),1);
        I_computed_1(k-min_index+1,j) = mean(cos((2*pi)*sqrt(1-y.^2)));
    end
end
figure(1)
semilogx(N_matrix,I_computed_1,'.')
hold on
c = 1.7;
semilogx(N,I_exact+c./sqrt(N),'r-')
semilogx(N,I_exact*ones(n_index,1),'r-')
semilogx(N,I_exact-c./sqrt(N),'r-')
title("Convergence of Method 1")
xlabel("No. of points")
ylabel("Computed Value")
ax = gca;
ax.FontSize = 14;
axis tight


% Computing integral by counting the number of points inside a [0,1]*[-1,1] rectangle that lie below the integral

I_computed_2 = zeros(n_index,N_sim); % for storing

for k =min_index:max_index
    % multiple simuations for each N
    for j=1:N_sim
        % generate x and y_guess within the rectangle.
        x = rand(N(k-min_index+1),1);
        y_guess = 2*rand(N(k-min_index+1),1)-1;
        y_actual = cos((2*pi)*sqrt(1-x.^2));
        m = 0;
        for i = 1:N(k-min_index+1)
            if y_actual(i) >=0
                if (y_actual(i) > y_guess(i)) && (y_guess(i) >= 0)
```

```
                    m = m+1;
                end
            else
                if (y_actual(i) < y_guess(i)) && (y_guess(i) < 0)
                    m = m-1;
                end
            end
        end
        I_computed_2(k-min_index+1,j) = 2*m/(N(k-min_index+1));
    end
end

figure(2)
semilogx(N_matrix,I_computed_2,'.')
hold on
c = 3;
semilogx(N,I_exact+c./sqrt(N),'r-')
semilogx(N,I_exact*ones(n_index,1),'r-')
semilogx(N,I_exact-c./sqrt(N),'r-')
title("Convergence of Method 2")
xlabel("No. of points")
ylabel("Computed Value")
ax = gca;
ax.FontSize = 14;
axis tight
```

The resulting convergence plots for the two methods are in Figure 3. From the way the red curves (which go as $\mathcal{O}(1/\sqrt{N})$) act as bounds for the spread, we can see that the scaling of intervals goes down in the same fashion.

$\square$



(a) Method 1

(b) Method 2

Figure 3: Convergence plots for both methods