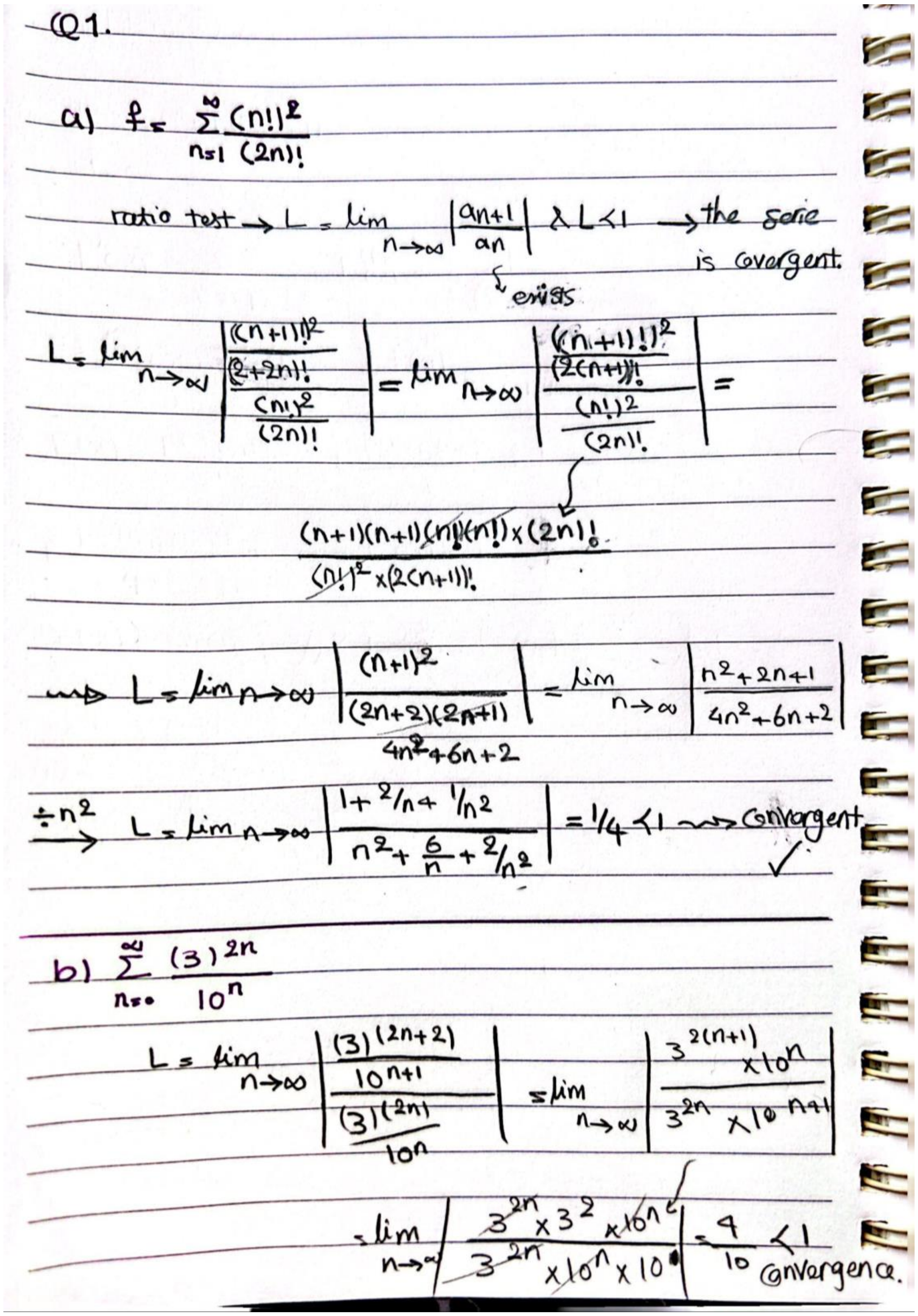
Question 1, Analytical Part:

I conducted a ratio test to ascertain the convergence of the series "a, b, and c". The results revealed that all three series are indeed convergent.



Q1.

a) $f = \sum_{n=1}^{\infty} \frac{(n!)^2}{(2n)!}$

ratio test $\rightarrow L = \lim_{n\to\infty} \left|\frac{a_{n+1}}{a_n}\right|$ & $L<1 \rightarrow$ the serie is covergent.

exists

$$L = \lim_{n\to\infty} \left| \frac{\frac{((n+1)!)^2}{(2+2n)!}}{\frac{(n!)^2}{(2n)!}} \right| = \lim_{n\to\infty} \left| \frac{\frac{((n+1)!)^2}{(2(n+1))!}}{\frac{(n!)^2}{(2n)!}} \right| =$$

$$\frac{(n+1)(n+1)\cancel{(n!)}(n!)\times(2n)!}{\cancel{(n!)^2}\times(2(n+1))!}$$

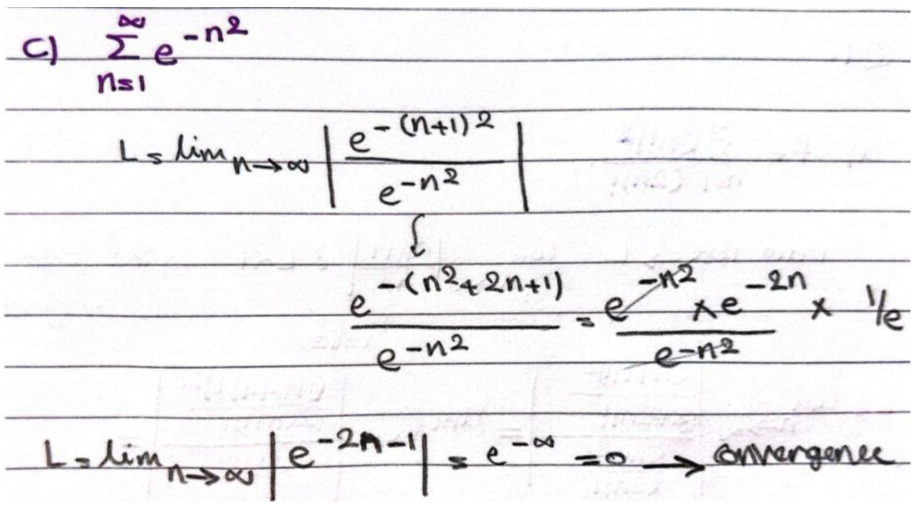
$$\Rightarrow L = \lim_{n\to\infty} \left|\frac{(n+1)^2}{\underbrace{(2n+2)(2n+1)}_{4n^2+6n+2}}\right| = \lim_{n\to\infty} \left|\frac{n^2+2n+1}{4n^2+6n+2}\right|$$

$$\xrightarrow{\div n^2} L = \lim_{n\to\infty} \left|\frac{1+2/n+1/n^2}{n^2+\frac{6}{n}+2/n^2}\right| = 1/4 < 1 \rightsquigarrow \text{Convergent } \checkmark$$

b) $\sum_{n=0}^{\infty} \frac{(3)^{2n}}{10^n}$

$$L = \lim_{n\to\infty} \left| \frac{\frac{(3)^{(2n+2)}}{10^{n+1}}}{\frac{(3)^{(2n)}}{10^n}} \right| = \lim_{n\to\infty} \left| \frac{3^{2(n+1)}\times 10^n}{3^{2n}\times 10^{n+1}} \right|$$

$$= \lim_{n\to\infty} \left| \frac{\cancel{3^{2n}}\times 3^2 \times \cancel{10^n}}{\cancel{3^{2n}}\times \cancel{10^n}\times 10} \right| = \frac{9}{10} < 1$$ convergence.



c) $\sum_{n=1}^{\infty} e^{-n^2}$

$$L = \lim_{n\to\infty} \left| \frac{e^{-(n+1)^2}}{e^{-n^2}} \right|$$

$$\frac{e^{-(n^2+2n+1)}}{e^{-n^2}} = \frac{\cancel{e^{-n^2}} \times e^{-2n} \times 1/e}{\cancel{e^{-n^2}}}$$

$$L = \lim_{n\to\infty} \left| e^{-2n-1} \right| = e^{-\infty} = 0 \rightarrow \text{convergence}$$

Question 1, Computer Part:

In this question, first I define the number of terms (N) which is set to 50, that indicated that I want to calculate and plot the first 50 partial sums of the series.

Second, I initialize the two arrays, “partialSum” and “nValues” in order to store the partial sums and corresponding values of n. The partialSum array keep the calculated partial sums, and the nValues array keep the values of n from 1 to 50.

Third, my program enters a loop that runs from n = 1 to N. In each iteration, it calculates the term of the series for the current value of n using the “a, b, and c” formula. Then, the calculated term is added to the partialSum array, representing the partial sum up to the current value of n.

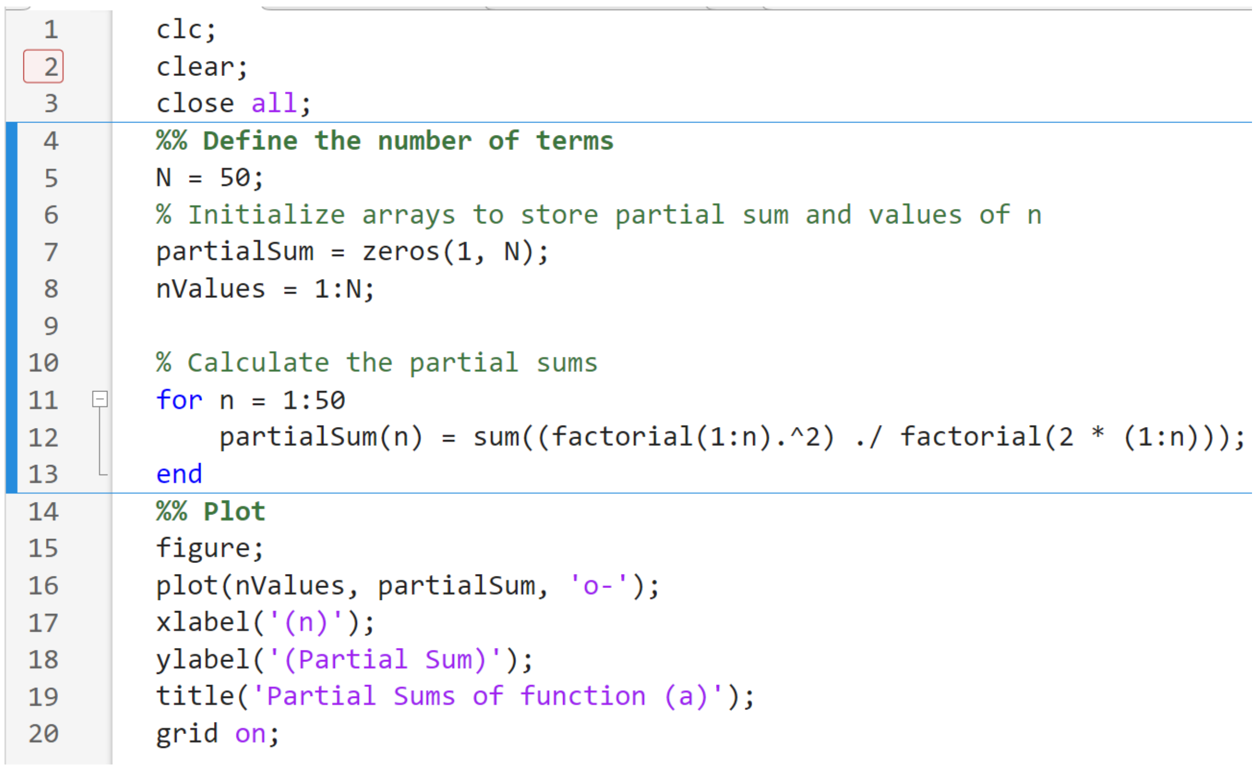
After calculating all partial sums, the program creates a plot of the partialSum array against the nValues array. It uses the plot function to generate the plot and specifies that data points should be marked as circles ('o-').

By observing the plot for series “a, b, and c”, you can see how the partial sum changes as more terms are added. Initially, it might fluctuate significantly, but as 'n' increases, it tends to stabilize or converge to a particular value.

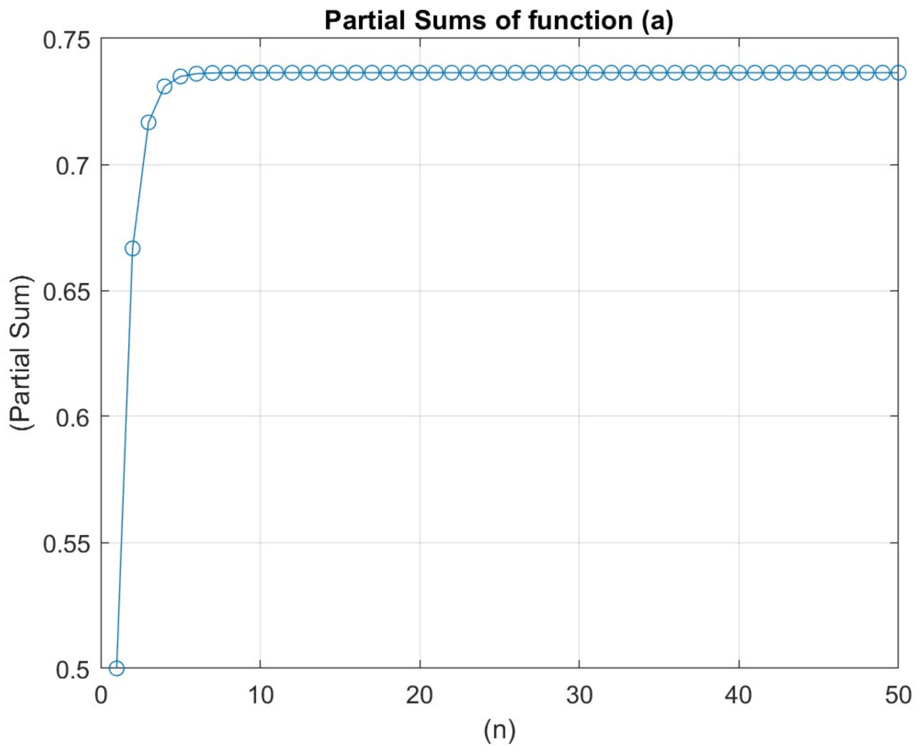
In the plots of “a, b, and c”, you can observe whether the series converges to a finite value or diverges as 'n' increases. If the partial sums eventually reach a constant value, it suggests convergence. If they keep growing without bound or oscillating, it indicates divergence. In all of these three series, we see that they are convergence because they reach to a consent value. The results are the same as using a ratio test.

(In my last coding, instead of doing first 50 partial sums, and then plot it, I only plot the value of them and did not do the partial sums correctly. Now, I have the correct code and the correct results.)

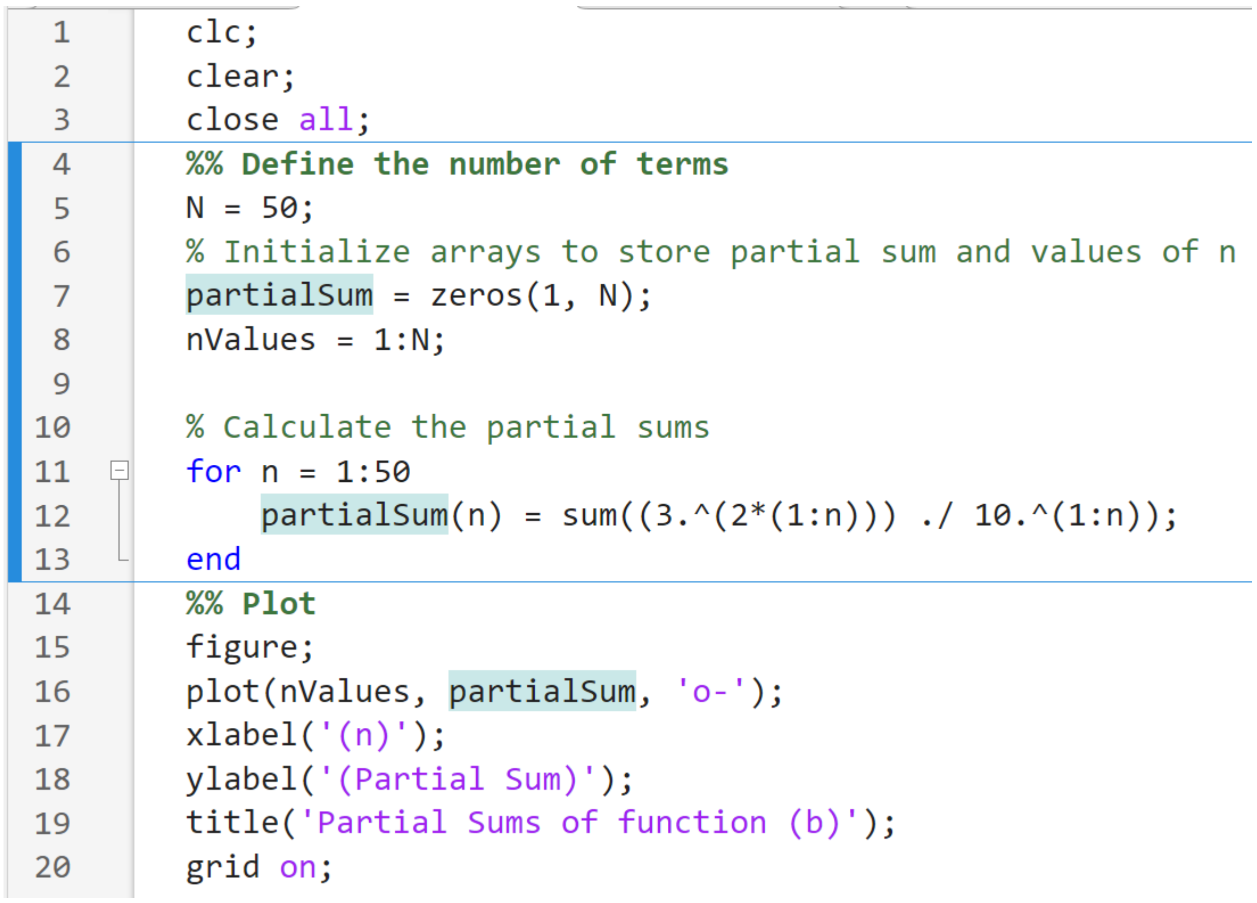
a) The code and the result:



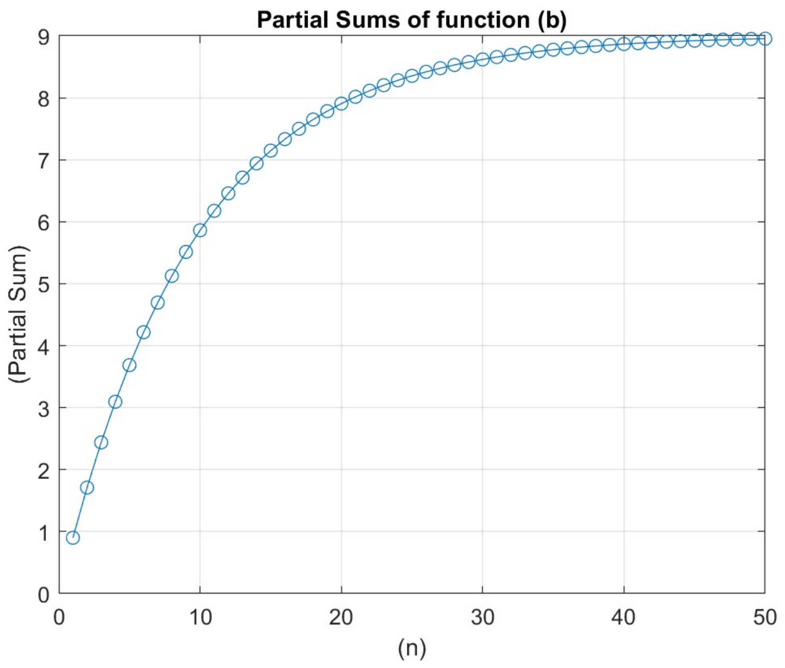
```
clc;
clear;
close all;
%% Define the number of terms
N = 50;
% Initialize arrays to store partial sum and values of n
partialSum = zeros(1, N);
nValues = 1:N;

% Calculate the partial sums
for n = 1:50
    partialSum(n) = sum((factorial(1:n).^2) ./ factorial(2 * (1:n)));
end
%% Plot
figure;
plot(nValues, partialSum, 'o-');
xlabel('(n)');
ylabel('(Partial Sum)');
title('Partial Sums of function (a)');
grid on;
```



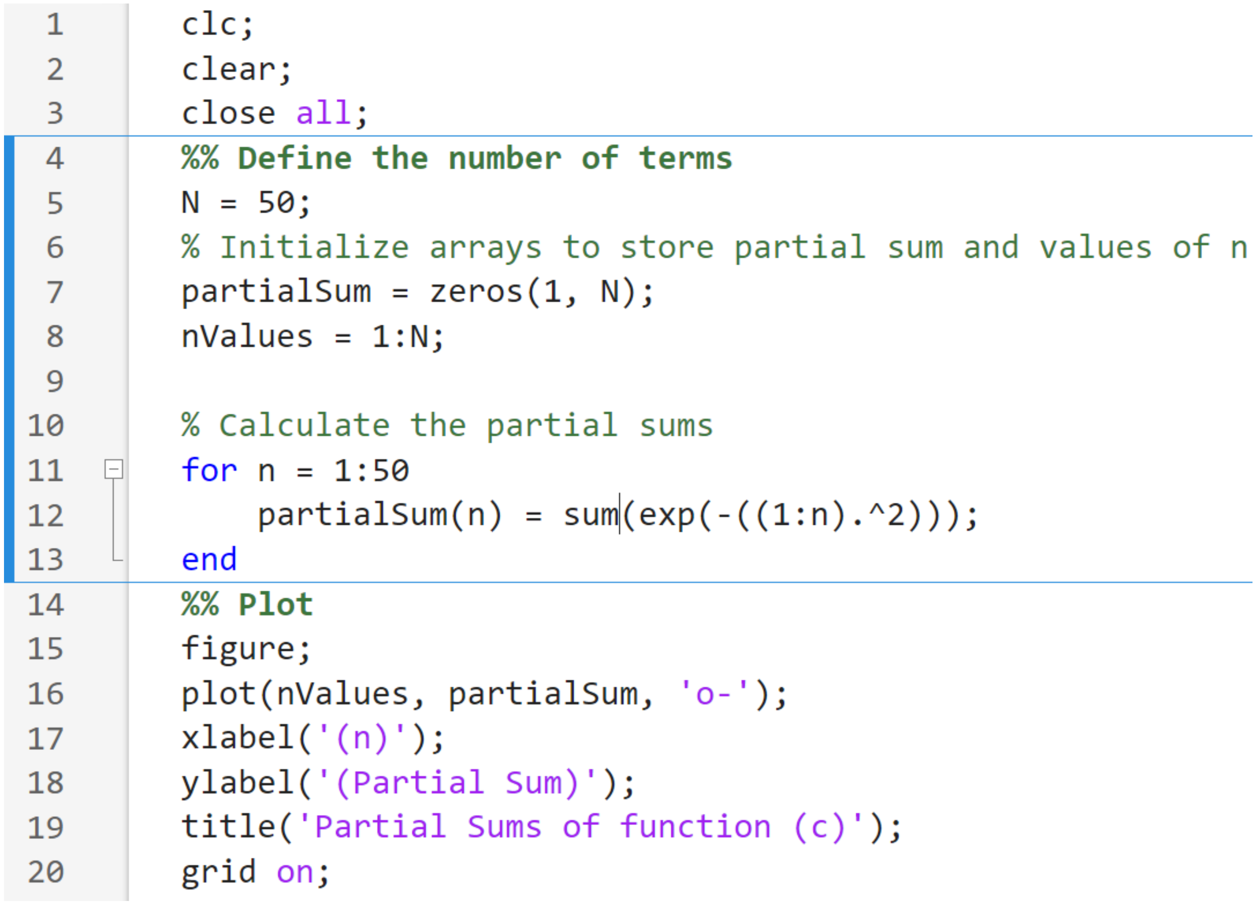
b) The code and the result:



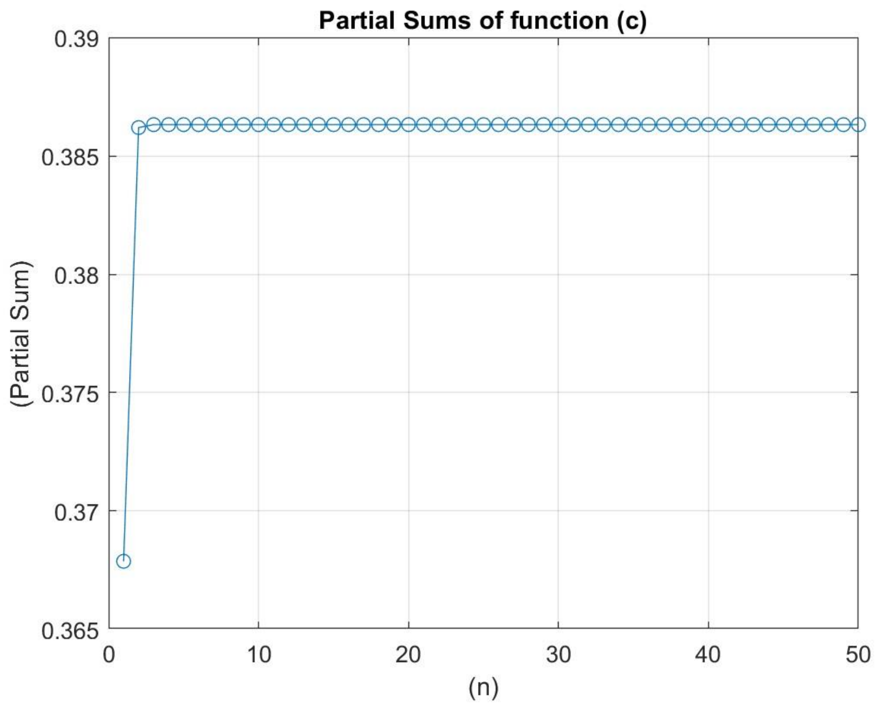
```
clc;
clear;
close all;
%% Define the number of terms
N = 50;
% Initialize arrays to store partial sum and values of n
partialSum = zeros(1, N);
nValues = 1:N;

% Calculate the partial sums
for n = 1:50
    partialSum(n) = sum((3.^(2*(1:n))) ./ 10.^(1:n));
end
%% Plot
figure;
plot(nValues, partialSum, 'o-');
xlabel('(n)');
ylabel('(Partial Sum)');
title('Partial Sums of function (b)');
grid on;
```



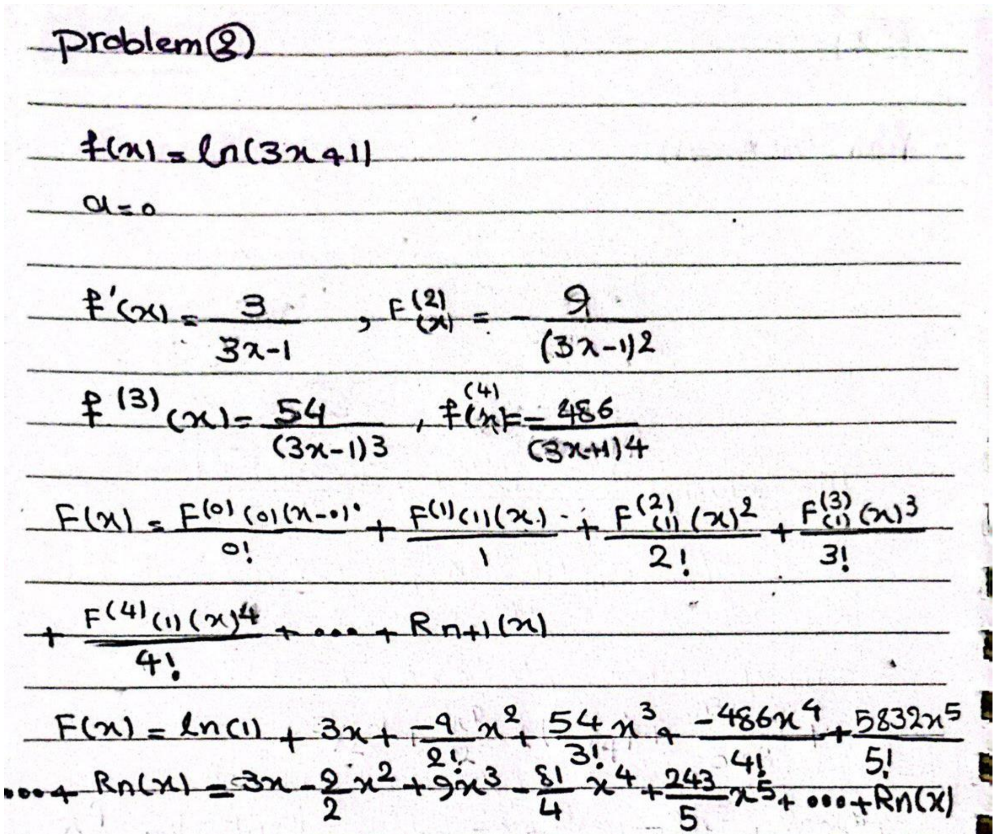
c) The code and the result:



```
clc;
clear;
close all;
%% Define the number of terms
N = 50;
% Initialize arrays to store partial sum and values of n
partialSum = zeros(1, N);
nValues = 1:N;

% Calculate the partial sums
for n = 1:50
    partialSum(n) = sum(exp(-((1:n).^2)));
end
%% Plot
figure;
plot(nValues, partialSum, 'o-');
xlabel('(n)');
ylabel('(Partial Sum)');
title('Partial Sums of function (c)');
grid on;
```



Question 2, Analytical Part:



problem(2)

$f(x) = \ln(3x+1)$

$a = 0$

$f'(x) = \frac{3}{3x-1}$, $F^{(2)}(x) = -\frac{9}{(3x-1)^2}$

$f^{(3)}(x) = \frac{54}{(3x-1)^3}$, $f^{(4)}(x) = -\frac{486}{(3x+1)^4}$

$F(x) = \frac{F^{(0)}(0)(x-0)^0}{0!} + \frac{F^{(1)}(1)(x)}{1} + \frac{F^{(2)}_{(1)}(x)^2}{2!} + \frac{F^{(3)}_{(1)}(x)^3}{3!}$

$+ \frac{F^{(4)}(1)(x)^4}{4!} + \dots + R_{n+1}(x)$

$F(x) = \ln(1) + 3x + \frac{-9}{2!}x^2 + \frac{54}{3!}x^3 + \frac{-486x^4}{4!} + \frac{5832x^5}{5!}$

$\dots + R_n(x) = 3x - \frac{9}{2}x^2 + 9x^3 - \frac{81}{4}x^4 + \frac{243}{5}x^5 + \dots + R_n(x)$

Question 2, Computer Part:

a) The MATLAB code computes and plots the exact function f(x) = ln(3x+1) with its Taylor series approximations using 2 and 5 terms over the interval x∈[0,0.5] with Δx=0.001.

The blue line in the plot represents the exact function of f(x). This is the real function we want to approximate. The green dashed line represents the Taylor series approximation of the function using 2 terms. The Taylor series is a polynomial expansion that approximates the original function around a specific point (in this case, around 0). With only 2 terms, this approximation might not capture the function's behavior accurately, especially for larger values of x. The red dashed line represents the Taylor series approximation using 5 terms. This approximation is more accurate than the one with 2 terms. With more terms, the Taylor series can better approximate the function over a wider range of x values.

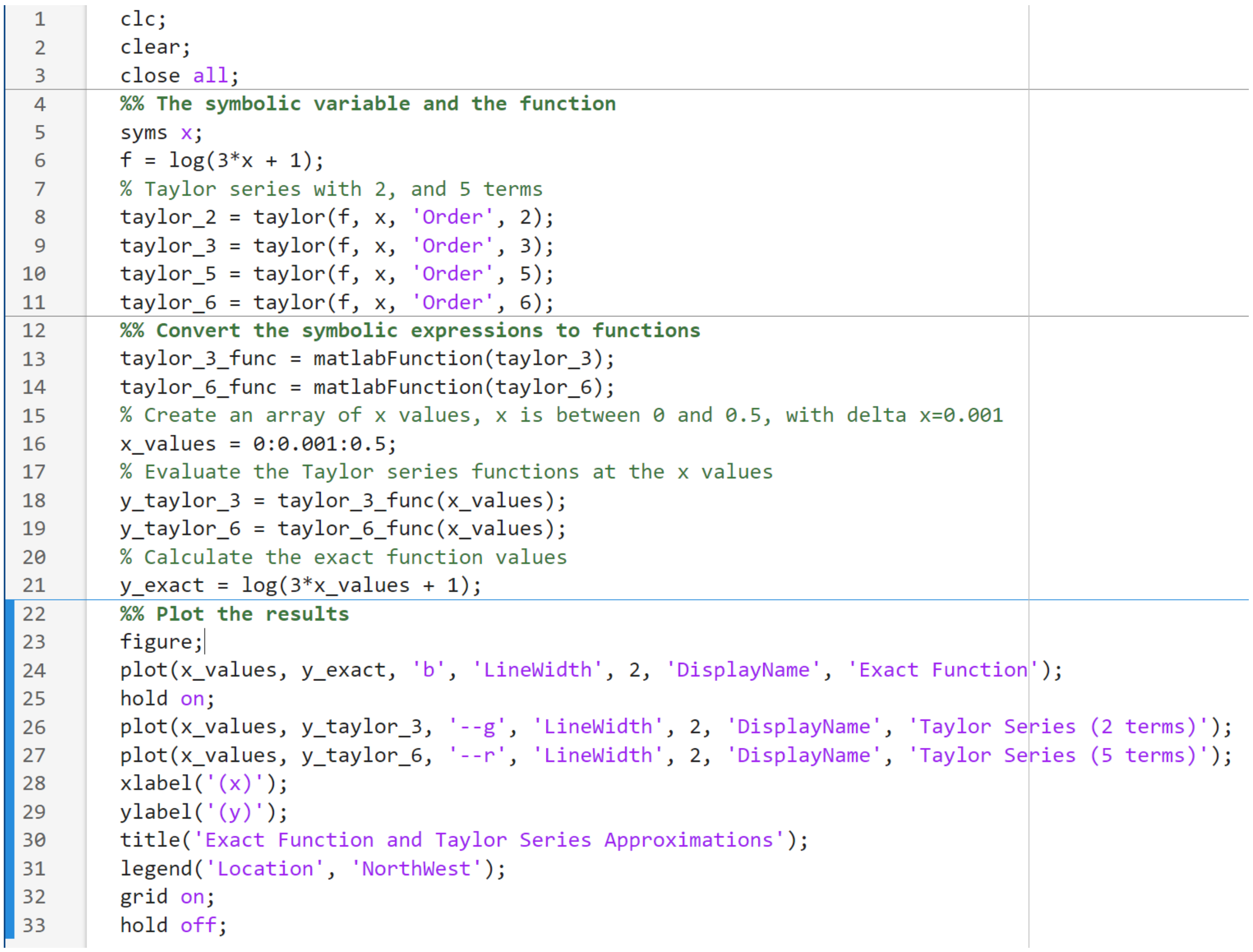
For small values of x, all three lines (exact, 2-term Taylor, and 5-term Taylor) overlap closely because the Taylor series is generally more accurate near the point of expansion (0).

As x increases, we can see a small deviation between the 2-term Taylor series (green dashed line) and the exact function (blue line). The 2-term approximation starts to deviate from the exact function because it cannot capture the curvature of the logarithmic function well.

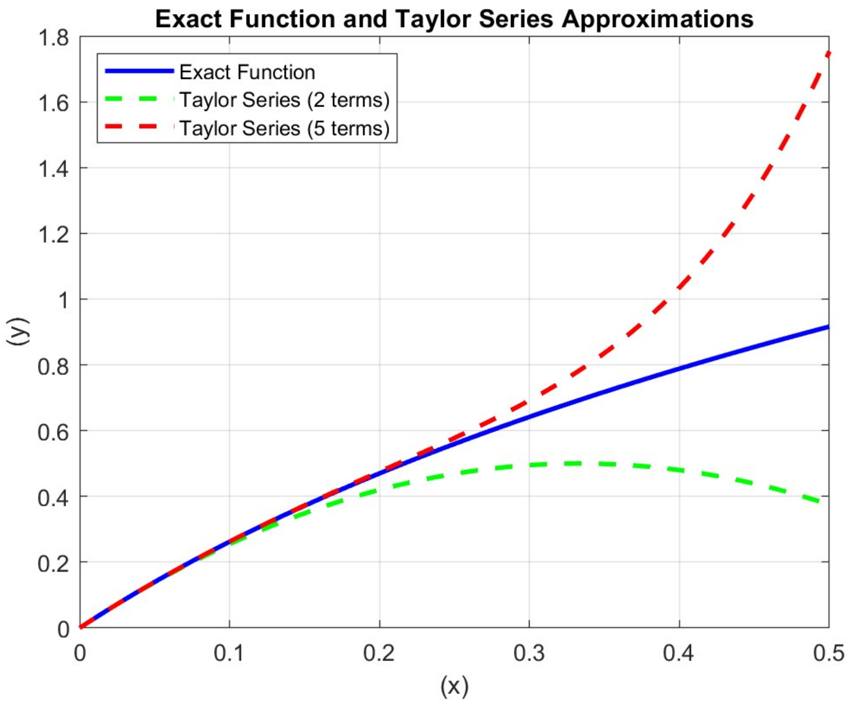
The 5-term Taylor series (red dashed line) provides a more accurate approximation for a larger range of x values compared to the 2-term approximation. However, even the 5-term Taylor series eventually deviates from the exact function for large x because the Taylor series is inherently a local approximation and may not be accurate over a wide range.

In summary, the results show how increasing the number of terms in the Taylor series leads to a more accurate approximation of the function, especially around the point of expansion. However, the accuracy of the Taylor series still diminishes as you move further from the expansion point, highlighting the trade-off between accuracy and the range of validity for the approximation.

(In my last coding, I had a mistake that you told me, and I really appreciate it. Since, I wanted to use Taylor series MATLAB function, I had to use taylor_3 and taylor_6 in order to get Taylor series approximation using 2 and 5 terms. Now when I call taylor_3, it gives me 2 terms and for taylor_6 I get 5 terms.)



```
clc;
clear;
close all;
%% The symbolic variable and the function
syms x;
f = log(3*x + 1);
% Taylor series with 2, and 5 terms
taylor_2 = taylor(f, x, 'Order', 2);
taylor_3 = taylor(f, x, 'Order', 3);
taylor_5 = taylor(f, x, 'Order', 5);
taylor_6 = taylor(f, x, 'Order', 6);
%% Convert the symbolic expressions to functions
taylor_3_func = matlabFunction(taylor_3);
taylor_6_func = matlabFunction(taylor_6);
% Create an array of x values, x is between 0 and 0.5, with delta x=0.001
x_values = 0:0.001:0.5;
% Evaluate the Taylor series functions at the x values
y_taylor_3 = taylor_3_func(x_values);
y_taylor_6 = taylor_6_func(x_values);
% Calculate the exact function values
y_exact = log(3*x_values + 1);
%% Plot the results
figure;
plot(x_values, y_exact, 'b', 'LineWidth', 2, 'DisplayName', 'Exact Function');
hold on;
plot(x_values, y_taylor_3, '--g', 'LineWidth', 2, 'DisplayName', 'Taylor Series (2 terms)');
plot(x_values, y_taylor_6, '--r', 'LineWidth', 2, 'DisplayName', 'Taylor Series (5 terms)');
xlabel('(x)');
ylabel('(y)');
title('Exact Function and Taylor Series Approximations');
legend('Location', 'NorthWest');
grid on;
hold off;
```



b) This code computes the L2-norm of the error vector for both the 2-term and 5-term Taylor series approximations over the interval [0,0.5]. It evaluates the error at multiple points within the interval and then calculates the norm of the error vector.

```
L2-Norm of Error (2 Terms): 4.930001
L2-Norm of Error (5 Terms): 5.449882
```

The results were interesting for me and I did not anticipate this result. Here's what these results mean:

1. L2-Norm of Error (2 Terms): 4.930001: This value represents the L2-norm of the error vector for the 2-term Taylor series approximation. It quantifies how different the 2-term Taylor series approximation is from the exact function values over the specified range. A smaller L2-norm indicates that the approximation is closer to the exact values.
2. L2-Norm of Error (5 Terms): 5.449882: This value represents the L2-norm of the error vector for the 5-term Taylor series approximation. It quantifies the error between the 5-term approximation and the exact function values. In this case, the L2-norm is slightly larger than that of the 2-term approximation, indicating that the 5-term approximation introduces slightly more error over the specified range.

In summary, these L2-norm values provide a measure of how well the Taylor series approximations capture the behavior of the logarithmic function $f(x)=\ln(3x+1)$ over the given range. While the 2-term approximation has a slightly smaller L2-norm, both approximations introduce some error, especially as you move further away from the expansion point ($a \approx 0$).
The L2-norm being smaller for the 2-term Taylor series approximation compared to the 5-term Taylor series approximation indicates that, over the specified range [0, 0.5], the 2-term approximation is closer to the exact function values on average.

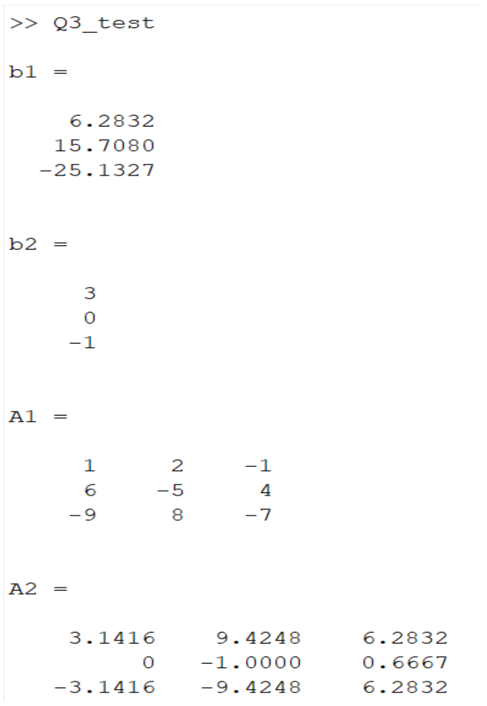
It's important to note that while the 2-term approximation might be closer to the exact function values within the specified range, it may become less accurate as you move further away from a specific point. The 5-term approximation, on the other hand, may capture the behavior of the function more accurately in a broader range but introduces slightly more error within the specified range.

Question 3:

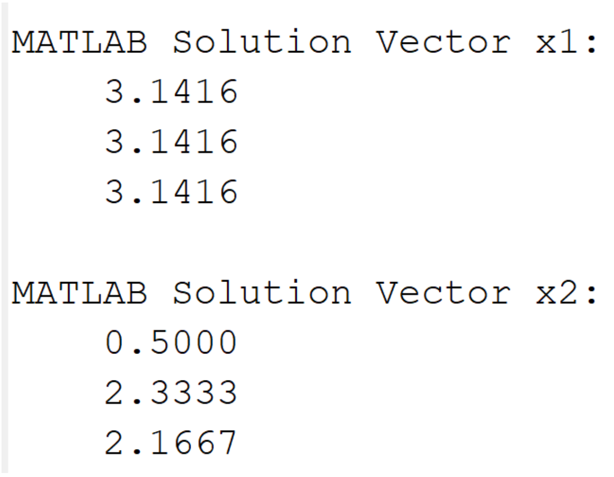
For these two algorithms (Algorithm #1: luDecomposition, Algorithm #2: solveLinearEquation), I have written a MATLAB code. The source code is in the file, and the results of the test are written below:

Solution vector x1, is for A1, b1. Solution vector x2, is for A2, b2.

(For this question, I only was able to add your display command to my code, the results are below.)



```
>> Q3_test

b1 =

    6.2832
   15.7080
  -25.1327


b2 =

     3
     0
    -1


A1 =

     1     2    -1
     6    -5     4
    -9     8    -7


A2 =

    3.1416    9.4248    6.2832
         0   -1.0000    0.6667
   -3.1416   -9.4248    6.2832
```



```
MATLAB Solution Vector x1:
    3.1416
    3.1416
    3.1416

MATLAB Solution Vector x2:
    0.5000
    2.3333
    2.1667
```