Homework 1

Adam Fasulo

August 25, 2025

Problem 1: Vector Manipulation

Question

Suppose $z = [10, 40, 70, 90, 20, 30, 50, 60]$. What does this vector look like after each of these commands?

1. `z(1:3:7) = zeros(1,3)`
2. `z([3 4 1]) = []`

Solution

My starting vector is $z = [10, 40, 70, 90, 20, 30, 50, 60]$.

Command 1: `z(1:3:7) = zeros(1,3)`

This command takes the 1st, 4th, and 7th elements and makes them zero. The vector is now: $z = [0, 40, 70, 0, 20, 30, 0, 60]$

Command 2: `z([3 4 1]) = []`

This command deletes the 3rd, 4th, and 1st elements from the new vector. The final vector is: $z = [40, 20, 30, 0, 60]$

MATLAB Output:

```
% After command 1:
     0    40    70     0    20    30     0    60

% After command 2 (Final z):
    40    20    30     0    60
```

Problem 2: Vector Creation

Question

1. Use the `linspace` function to create vectors identical to the following created with colon notation:
   - `t = 1:4:25`
   - `x = -11:1`
2. Use colon notation to create vectors identical to the following created with the `linspace` function:
   - `v = linspace(-10,-8,6)`
   - `r = linspace(0,1,5)`

Solution

Part 1: 'linspace' from Colon Notation

To use linspace, I just needed to count how many numbers were in each vector. For `t`, it was 7 points, and for `x` it was 13 points.

```
t = linspace(1, 25, 7);
x = linspace(-11, 1, 13);
```

Part 2: Colon Notation from 'linspace'

To go the other way, I had to find the step size. I used the formula (end - start) / (num_points - 1). For `v`, the step is $(-8-(-10))/(6-1) = 0.4$. For `r`, the step is $(1-0)/(5-1) = 0.25$.

```
v = -10:0.4:-8;
r = 0:0.25:1;
```

Problem 3: Single-Line Summation

Question

Given that $t = 0 : 0.1 : 1$ and $y = \sin(\pi * t)$, write a single-line MATLAB code that returns the following summations:

1. $\sum_{k=1}^{N} t_k$
2. $\sum_{k=1}^{N} t_k y_k$
3. $\sum_{k=1}^{N} t_k^2$

Solution

```
% Given vectors
t = 0:0.1:1;
y = sin(pi*t);

% 1. Sum of t
sum_t = sum(t);

% 2. Sum of t times y
sum_ty = sum(t .* y);

% 3. Sum of t squared
sum_t_squared = sum(t.^2);
```

Problem 4: Plotting

Question

Write a MATLAB script to plot e^x, x^2, and x^3 over the interval $0 \leq x \leq 1$ using `plot`, `semilogy`, `semilogx`, and `loglog`.

MATLAB Script

```
% I had to start x at a small number instead of 0
% because log(0) gives an error.
x = linspace(0.001, 1, 200);
y1 = exp(x);
y2 = x.^2;
y3 = x.^3;

% 1. Standard Linear Plot
figure;
plot(x, y1, 'r', x, y2, 'b--', x, y3, 'g:', 'LineWidth', 2);
title('Linear Plot'); xlabel('x'); ylabel('y');
legend('e^x', 'x^2', 'x^3', 'Location', 'northwest');
grid on;

% 2. Semilog Y-axis Plot
figure;
semilogy(x, y1, 'r', x, y2, 'b--', x, y3, 'g:', 'LineWidth', 2);
title('Semilog Y-axis Plot'); xlabel('x'); ylabel('y (log scale)');
legend('e^x', 'x^2', 'x^3', 'Location', 'northwest');
grid on;

% 3. Semilog X-axis Plot
figure;
semilogx(x, y1, 'r', x, y2, 'b--', x, y3, 'g:', 'LineWidth', 2);
title('Semilog X-axis Plot'); xlabel('x (log scale)'); ylabel('y');
legend('e^x', 'x^2', 'x^3', 'Location', 'southeast');
grid on;

% 4. Log-Log Plot
figure;
loglog(x, y1, 'r', x, y2, 'b--', x, y3, 'g:', 'LineWidth', 2);
title('Log-Log Plot'); xlabel('x (log scale)'); ylabel('y (log scale)');
legend('e^x', 'x^2', 'x^3', 'Location', 'southeast');
grid on;
```

Discussion

The normal `plot` command shows the basic shapes of the functions. The `semilogy` plot helps see the y-values better since they are so different. The `loglog` plot was interesting because it turned the x^2 and x^3 curves into straight lines. The line for x^3 is steeper.

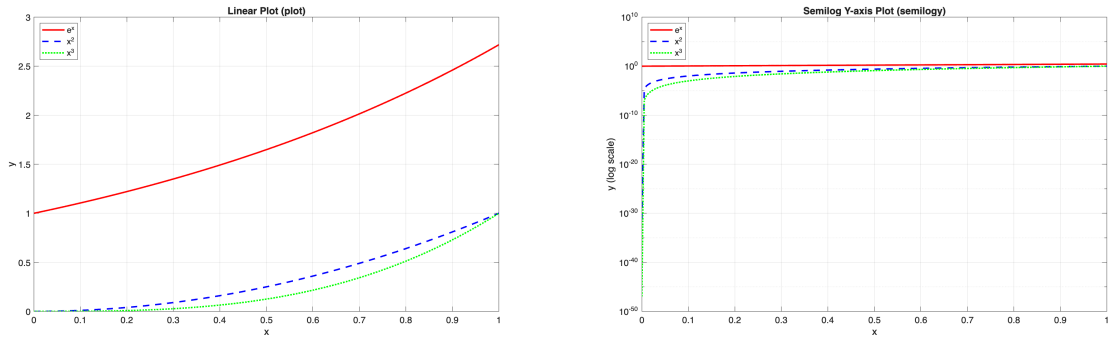

Figure 1: Left: Linear plot. Right: Semilog-Y plot.

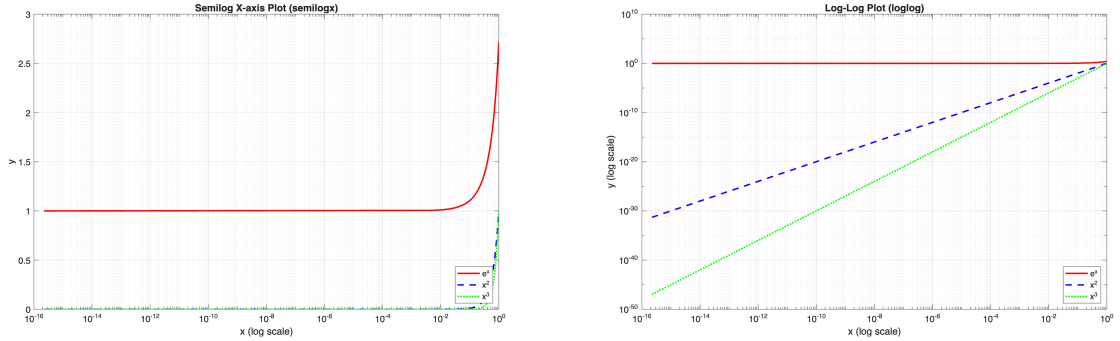

Figure 2: Left: Semilog-X plot. Right: Log-Log plot.

Problem 5: Function and Integration

Question

1. Write a MATLAB function `my_mean` which approximates $\int_a^b f(x)dx$. 2. Write a MATLAB function `my_fun` which returns the value of xe^x. 3. Use these functions to compute approximations for $M = \int_0^1 xe^x dx$ for various N. 4. Plot the absolute error vs. N and print a table of the results.

Solution

Function: `my_fun.m`

```
function y = my_fun(x)
    % Calculates the value of x*exp(x) for a vector x
    y = x .* exp(x);
end
```

Function: `my_mean.m`

```
function approx_integral = my_mean(f, a, b, N)
    % Approximates the integral using the midpoint rule.
    dx = (b - a) / N;
    midpoints = a + dx/2 : dx : b - dx/2;
    y_values = f(midpoints);
```

```
    approx_integral = sum(y_values) * dx;
end
```

Results

The exact value of the integral is $M = 1$. I made a plot and a table to show the absolute error, which is $|1 - \text{approximation}|$, for different numbers of intervals (N). You can see from the plot that the error goes down in a straight line on the log-log scale when N gets bigger. This means my approximation is getting more accurate.

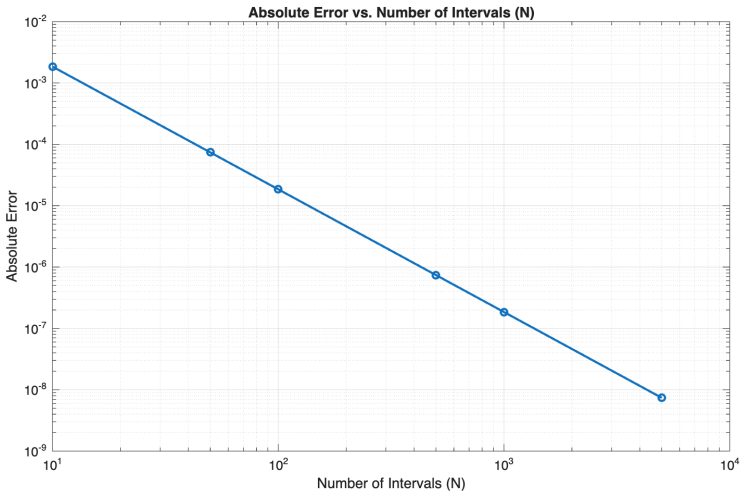

Figure 3: Absolute error vs. number of intervals (N).

Table 1: Approximation results and absolute error for $\int_0^1 xe^x dx$.

N	Approximation	Absolute Error
10	0.99965315	3.4685×10^{-4}
50	0.99998612	1.3879×10^{-5}
100	0.99999653	3.4697×10^{-6}
500	0.99999986	1.3879×10^{-7}
1000	0.99999997	3.4697×10^{-8}
5000	1.00000000	1.3879×10^{-9}