Summary Report: MATLAB Problem Set Solutions

Parisa Mohammadi[1]
Department of Electrical Engineering, University of Tehran

This report provides a conceptual summary of solutions to a MATLAB problem set covering algebraic computations, programming logic, and graphical representation. It discusses common issues such as floating-point limitations, the importance of vectorization for efficiency, and the application of core programming constructs like loops and conditional statements for solving complex problems.

Keywords. MATLAB, Programming, Vectorization, Floating-Point, Plotting, Symbolic Math

1 Introduction

This report provides a conceptual breakdown of the solutions for the MATLAB programming and mathematics problem set. It is divided into two main parts, mirroring the structure of the original assignment.

2 Part 1: Algebraic Computations and Graphics

This section focused on MATLAB's capabilities for numerical calculation, handling precision, and creating plots.

2.1 Calculations

This part explored the nuances of MATLAB's arithmetic, especially the limitations and behaviors of computer-based math.

Approximating $\sqrt{7}$

The best fractional approximation was found using two methods. The first was a simple visual comparison after using the `format long` command to display sufficient decimal places. The second, more programmatic method involved calculating the absolute difference between $\sqrt{7}$ and each fraction, then using the `min` function to identify which fraction produced the smallest difference.

Calculating 330!

This problem highlighted the difference between standard and high-precision math. The standard `factorial(330)` calculation resulted in `Inf` (Infinity). This is not an error but the correct outcome, as the true value of 330! is an enormous number that far exceeds the maximum value representable by standard double-precision numbers (a phenomenon known as numerical overflow). To get the exact integer value, it was necessary to use MATLAB's symbolic math capabilities. By treating '330' as a symbolic object, the factorial could be computed exactly, bypassing the limits of floating-point arithmetic.

[1] parisa.mohammadi7@ut.ac.ir

Floating-Point Inaccuracy

The expressions $20/3 - 20 \times (1/3)$ and $10^{16} + 1 - 10^{16}$ demonstrated common floating-point issues. The first expression resulted in a tiny non-zero number due to representation error, as fractions like 1/3 cannot be represented perfectly in binary. The second expression resulted in 0 instead of 1 due to a loss of significance. A standard double-precision number only has about 15-16 digits of precision; the number $10^{16} + 1$ requires 17 digits, so the '+ 1' is lost during rounding.

2.2 Calculating Values to 15 Digits

This task was accomplished by using the `format long` command to set MATLAB's display output to 15 decimal places. The required values were then calculated using the standard built-in functions (`cosh`, `log` for the natural logarithm, and `atan` for the arctangent), which compute results in double-precision by default.

2.3 Plotting Functions

This section compared different methods for visualizing functions.

Plotting $\sin(1/x^2)$

This demonstrated the importance of choosing the right plotting tool for a highly oscillatory function. The `fplot` and `ezplot` commands produced an accurate graph because they use an adaptive step size, calculating more points in areas where the function changes rapidly. In contrast, the standard `plot` command, which uses a fixed step size, gave a poor representation.

Plotting the Butterfly Curve

This parametric plot was generated by first creating a vector for the parameter t. Then, the x and y coordinate vectors were calculated based on t. A critical technique here was the use of element-wise operators (`.*`, `.∧`), which are necessary to perform calculations on every element of the t vector simultaneously.

3 Part 2: Mathematics and Programming

This section focused on programming logic, including logical operations, loops, and function creation.

3.1 Evaluating Logical Expressions

The key principle for these problems is that MATLAB represents 'true' as '1' and 'false' as '0' in calculations. A crucial behavior is that MATLAB evaluates chained inequalities from left to right. For instance, $-7 < -5 < -2$ is evaluated as '(true) < -2', which becomes '1 < -2', resulting in 'false'. The expressions were evaluated based on standard operator precedence.

3.2 Pascal's Triangle

The matrix representing Pascal's triangle was generated algorithmically. The process involved initializing a matrix of zeros and then using nested `for` loops to iterate through each cell. An `if` statement was used to set the edges of the triangle to 1. All other interior cells were calculated using Pascal's rule: each number is the sum of the two numbers directly above it.

3.3 Finding the Largest Eigenvalue

This task required finding the largest eigenvalue of a 500x500 Hilbert matrix. To avoid the extreme slowness of nested loops, an efficient, vectorized approach was used. The `meshgrid` command generated two matrices containing the row and column indices, allowing the entire Hilbert matrix to be constructed in a single line of code. The built-in `eig` function was then used to find all eigenvalues, and the `max` function identified the largest among them.

3.4 Random Vector Manipulation

This was a multi-step programming task. A vector of random integers was created with `randi`. The core of the problem was to replace all odd numbers until none remained. This was achieved using a `while` loop that continued as long as the `any` function detected at least one odd number. Inside the loop, logical indexing provided a loop-free way to replace only the odd elements.

3.5 Custom `mylcm` Function

A function named `mylcm.m` was designed to find the least common multiple (LCM) of any quantity of numbers. It used `varargin` to handle a flexible number of inputs. The function first validated the inputs, using the `error` command to halt if any input was not a positive integer. The logic was based on an iterative calculation, where the built-in two-argument `lcm` function was repeatedly applied in a `for` loop to find the final LCM of all numbers.

4 Final Remarks

This report summarized the conceptual approaches to solving a variety of computational problems in MATLAB. Key takeaways include the importance of understanding floating-point limitations, leveraging vectorization for performance, and applying fundamental programming structures to build robust solutions.