MATLAB® Programming for Engineers

Second Edition

Updated Series

Stephen J. Chapman

Bookware Companion Series

Section 1

Engineering Problem Solving

Engineering often involves applying a consistent, structured approach to the solving of problems.

A general problem-solving approach and method can be defined, although variations will be required for specific problems.

Problems must be approached methodically, applying an *algorithm*, or step-by-step procedure by which one arrives at a solution.

1.1 Problem-Solving Process

The **problem-solving process** for a computational problem can be outlined as follows:

1. Define the problem.
2. Create a mathematical model.
3. Develop a computational method for solving the problem.
4. Implement the computational method.
5. Test and assess the solution.

The boundaries between these steps can be blurred and for specific problems one or two of the steps may be more important than others. Nonetheless, having this approach and strategy in mind will help to focus our efforts as we solve problems

1. Problem Definition:

The first steps in problem solving include:

- Recognize and define the problem precisely by exploring it thoroughly (may be the most difficult step).

- Determine what question is to be answered and what output or results are to be produced.
- Determine what theoretical and experimental knowledge can be applied.
- Determine what input information or data is available

Many academic problems that you will be asked to solve have this step completed by the instructor. For example, if your instructor ask you to solve a quadratic algebraic equation and provides you with all of the coefficients, the problem has been completely defined before it is given to you and little doubt remains about what the problem is.

If the problem is not well defined, considerable effort must be expended at the beginning in studying the problem, eliminating the things that are unimportant, and focusing on the root problem. Effort at this step pays great dividends by eliminating or reducing false trials, thereby shortening the time taken to complete later steps.

After defining the problem:

- Collect all data and information about the problem.
- Verify the accuracy of this data and information.
- Determine what information you must find: intermediate results or data may need to be found before the required answer or results can be found.

2. Mathematical Model:

To create a mathematical model of the problem to be solved:

- Determine what fundamental principles are applicable.
- Draw sketches or block diagrams to better understand the problem.
- Define necessary variables and assign notation.
- Reduce the problem as originally stated into one expressed in purely mathematical terms.
- Apply mathematical expertise to extract the essentials from the underlying physical description of the problem.
- Simplify the problem only enough to allow the required information and results to be obtained.
- Identify and justify the assumptions and constraints inherent in this model.

3. Computational Method:

A computational method for solving the problem is to be developed, based on the mathematical model.

- Derive a set of equations that allow the calculation of the desired parameters and variables.

- Develop an algorithm, or step-by-step method of evaluating the equations involved in the solution.
- Describe the algorithm in mathematical terms and then implement as a computer program.
- Carefully review the proposed solution, with thought given to alternative approaches.

4. Implementation of Computational Method:

Once a computational method has been identified, the next step is to carry out the method with a computer, whether human or silicon.

Some things to consider in this implementation:

- Assess the computational power needed, as an acceptable implementation may be hand calculation with a pocket calculator.
- If a computer program is required, a variety of programming languages, each with different properties, are available.
- A variety of computers, ranging from the most basic home computers to the fastest parallel supercomputers, are available.
- The ability to choose the proper combination of programming language and computer, and use them to create and execute a correct and efficient implementation of the method, requires both knowledge and experience. In your engineering degree program, you will be exposed to several programming languages and computers, providing you with some exposure to this issue.

The mathematical algorithm developed in the previous step must be translated into a computational algorithm and then implemented as a computer program.

The steps in the algorithm should first be outlined and then decomposed into smaller steps that can be translated into programming commands.

One of the strengths of MATLAB is that its commands match very closely to the steps that are used to solve engineering problems; thus the process of determining the steps to solve the problem also determines the MATLAB commands. Furthermore, MATLAB includes an extensive toolbox of numerical analysis algorithms, so the programming effort often involves implementing the mathematical model, characterizing the input data, and applying the available numerical algorithms.

5. Test and Assess the Solution:

The final step is to test and assess the solution. In many aspects, assessment is the most open-ended and difficult of the five steps involved in solving computational problems.

The numerical solution must be checked carefully:

- A simple version of the problem should be hand checked.

- The program should be executed on obtained or computed test data for which the answer or solution is either known or which can be obtained by independent means, such as hand or calculator computation.
- Intermediate values should be compared with expected results and estimated variations. When values deviate from expected results more than was estimated, the source of the deviation should be determined and the program modified as needed.
- A "reality check" should be performed on the solution to determine if it makes sense.
- The assumptions made in creating the mathematical model of the problem should be checked against the solution.

1.2 Problem Solving Example

As an example of the problem solving process, consider the following problem:

> A small object is launched into flight from the ground at a speed of 50 miles/hour at 30 degrees above the horizontal over level ground. Determine the time of flight and the distance traveled when the ball returns to the ground.

1. **Problem Definition:**

   As stated above, this problem is well defined. The following items could be considered in further defining the problem.

   - Additional information needed:
     - Properties of the object and the flight medium could affect the flight trajectory. For example, if the object is light in weight, has a relatively large surface area, and travels in air, the air resistance would affect its flight. If the air were moving (by wind, for example), the flight of the object would also be affected. If this information is not available, it will be necessary to assume that the medium has no affect on the trajectory of flight.
     - Acceleration of gravity also affects the flight. If no further information is available, it would be reasonable to assume that the gravitational acceleration at the surface of the Earth would apply.
     - The accuracy of the initial speed and angle of the object is needed to determine the necessary accuracy of the quantities to be computed. The problem is one for which we have some physical insight, as it could represent the throwing of a baseball, although it is thrown from the ground instead of shoulder level. From this interpretation, the initial speed and angle seem reasonable.
     - Unit conversions needed:
       1 mile = 5280 feet
       1 hour = 60 minutes = 3600 seconds
       360 degrees = 2π radians

- Output or results to be produced: It is clear from the problem statement that the results to be produced are the time of flight and the distance traveled. The units for these quantities haven't been specified, but from our knowledge of throwing a baseball, computing time in seconds and distance in feet would be reasonable.
- Theoretical and experimental knowledge to be applied: The theory to be applied is that of ballistic motion in two dimensions.
- Input information or data: This includes the object initial velocity of 50 miles per hour at an angle 30 degree above horizontal.

2. **Mathematical Model:**

To pose this problem in terms of a mathematical model, we first need to define the notation:

- Time: t (s), with $t = 0$ when the object is launched.
- Initial velocity magnitude: $v = 50$ miles/hour.
- Initial angle: $\theta = 30°$.
- Horizontal position of ball: $x(t)$ (ft).
- Vertical position of ball: $y(t)$ (ft).
- Acceleration of gravity: $g = 32.2$ ft/s^2, directed in negative y direction.

The key step in developing a mathematical model is to divide the trajectory into its horizontal and vertical components. The initial velocity can be divided in this way, as shown in Figure 1.1. From basic trigonometry, we know that

$$v_h = v\cos\theta$$

$$v_v = v\sin\theta$$

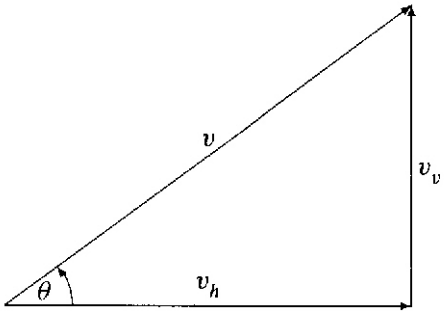

Figure 1.1: Initial velocity (v) divided into horizonal (v_h) and vertical (v_v) components.

Given the horizontal and vertical components of the initial velocity, the horizontal and vertical positions can be determined as functions of time. Since there is no external force acting to retard the horizontal motion, the object will move at a constant speed of v_h in the horizontal direction

$$x(t) = vt\cos\theta$$

In the vertical direction, the object motion is retarded by gravity and its position is

$$y(t) = vt\sin\theta - \frac{1}{2}gt^2$$

3. **Computational Method:**

   Using the model developed above, expressions for the desired results can be obtained. The object will hit the ground when its vertical position is zero

$$y(t) = vt\sin\theta - \frac{1}{2}gt^2 = 0$$

   which can be solved to yield two values of time

$$t = 0, \quad \frac{2v\sin\theta}{g}$$

   The second of the two solutions indicates that the object will return to the ground at the time

$$t_g = \frac{2v\sin\theta}{g}$$

   The horizontal position (distance of travel) at this time is

$$x(t_g) = vt_g\cos\theta$$

4. **Computational Implementation:**

   The equations defined in the computational method can be readily implemented using MATLAB. The commands used in the following solution will be discussed in detail later in the course, but observe that the MATLAB steps match closely to the solution steps from the computational method.

```
% Flight trajectory computation
%
% Initial values
g = 32.2;                          % gravity, ft/s^2
v = 50 * 5280/3600;                % launch velocity, ft/s
theta = 30 * pi/180;               % launch angle, radians

% Compute and display results
disp('time of flight (s):')        % label for time of flight
tg = 2 * v * sin(theta)/g          % time to return to ground, s
disp('distance traveled (ft):')    % label for distance
xg = v * cos(theta) * tg          % distance traveled

% Compute and plot flight trajectory
t = linspace(0,tg,256);
x = v * cos(theta) * t;
y = v * sin(theta) * t - g/2 * t.^2;
plot(x,y), axis equal, axis([ 0 150 0 30 ]), grid, ...
   xlabel('Distance (ft)'), ylabel('Height (ft)'), title('Flight Trajectory')
```

   Comments:

- Words following percent signs (%) are comments to help in reading the MATLAB statements.
- A word on the left of an equals sign is known as a **variable name** and it will be assigned to the value or values on the right of the equals sign. Commands having this form are known as **assignment statements**.
- If a MATLAB command assigns or computes a value, it will display the value on the screen if the statement does not end with a semicolon (;). Thus, the values of `v`, `g`, and `theta` will not be displayed. The values of `tg` and `xg` will be computed and displayed, because the statements that computes these values does not end with a semicolon.
- The three dots (...) at the end of a line mean that the statement continues on the next line.
- More than one statement can be entered on the same line if the statements are separated by commas.
- The statement `t = linspace(0,tg,256);` creates a vector of length 256.
- The expression `t.^2` squares *each element* in `t`, making another vector.
- In addition to the required results, the flight trajectory has also been computed and plotted. This will be used below in assessing the solution.
- The `plot` statement generates the plot of height against distance, complete with a title and labels on the x and y axes.

5. **Testing and Assessing the Solution:**

   This problem is sufficiently simple that the results can be computed by hand with the use of a calculator. There is no need to generate test data to check the results. One could even question the need to use the power of MATLAB to solve this problem, since it is readily solved using a calculator. Of course, our objective here is to demonstrate the application of MATLAB to a problem with which you are familiar.

   Executing the statements above provides the following displayed results:

```
time of flight (s):
tg =
    2.2774
distance traveled (ft):
xg =
  144.6364
```

   Computing these quantities with a calculator can be shown to produce the same results.

   The flight trajectory plot that is generated is shown in Figure 1.2. This demonstrates another strength of MATLAB, as it has taken care of the details of the plot generation, including axis scaling, label placement, etc. This result can be used to assess the solution. Our physical intuition tell us that this result appears to be correct, as we know that we could throw a baseball moderately hard and have it travel a distance of nearly 150 feet and that maximum height of about 20 feet also seems reasonable. The shape of the trajectory also fits our observations.

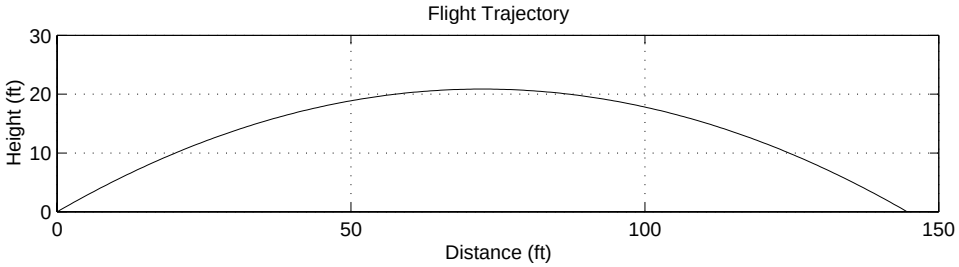

Figure 1.2: MATLAB generated plot of flight trajectory

1.2.1 Programming Style

Programs that are not documented internally, while they may do what is request, can be difficult to understand when read a few months later, in order to correct or update them. Thus, it is extremely important to develop the art of writing programs that are well structured, with all of the logic clearly described. This is known as *programming style*. Elements of good programming style include:

- Use comments liberally, both at the beginning of a script, to describe briefly what it does and any special methods that may have been used, and also throughout the script to introduce different logical sections.
- Describe each variable briefly in a comment when it is initialized.
- Separate sections of code with blank lines.
- Indent multiple line structures to make them stand out as a logical unit.
- Use spaces in expressions to make them more readable (for example, on either side of operators and equal signs).

1.3 Computing Software

Before discussing MATLAB in more detail, a brief discussion on computing software is useful.

Computer software contains the instructions or commands that the computer is to execute. Categories include:

- System software
- Languages
- Tools

System Software

Computer system software provides:

- An interface between the user and the hardware
- An environment for developing, selecting, and executing software.

The **operating system** is the most important part of the systems software, managing the computer resources, including:

- The individual user sessions on *multiuser* computer systems.
- The division of CPU time across various tasks.
- The *random access memory*, or RAM, where both instructions and data are stored.
- The secondary storage systems, such as disk drives, CD-ROM drives, tape drives, or any other device in which information is stored in binary form on some medium. Information in secondary storage is organized into units called *files*.

When a computer is turned on, it loads the operating system into RAM (usually from secondary storage) before a user can execute a program.

The system software may also include one or more **command shells**:

- Programs that direct the interaction with users. In most cases, when a you type a command, you are interfacing with the shell.
- Early shells provided a "text" only interface, but modern computers (particularly personal computers) have *graphical user interfaces* (GUIs) that allow users to express what they want to do by using a pointing device and selecting among choices displayed on the screen.

System software also includes:

- Language compilers
- Text editors
- Utilities for file management
- Utilities for printing

Computer Languages

Computer languages are used to develop programs to be executed. The can be described in terms of levels.

Machine language:

- Lowest-level language
- Binary-encoded instructions that are directly executed by the hardware.

- Language is closely tied to the hardware design.
- Machine specific: machine language for one computer is different from that of a computer having a different hardware design.

Assembly language:

- Also unique to a specific computer design.
- Commands or instructions are written in text statements instead of numbers.
- Assembly language programming requires good knowledge of the specific computer hardware. This is a tedious process, but it results in programs that are very fast, because they take advantage of the specific computer hardware.
- System software called an **assembler** translates the assembly language program to a machine language program for execution on the hardware.

High-level languages:

- Have English-like command syntax.
- Include languages such as Basic, Fortran, COBOL, Pascal, Ada, C, C++, and Java.
- Supported on many computer designs and knowledge of the specific hardware is not as critical in writing a program.
- System software called a **compiler** translates the program into machine language.

Compilation Process:

- Original program is called the **source program**
- Translated machine language program is called the **object program**.
- Errors detected by the compiler (called **compile errors** or **syntax errors**) must be corrected and compilation performed again.
- The process of compiling, correcting errors (or **debugging**) must often be repeated several times before the program compiles without compile errors.

Execution Process:

- To prepare the object program for **execution**, system software is applied to **link** other machine language statements to the object program and then **load** the program into memory.
- The program is executed and new errors, called execution errors, run-time errors or **logic errors** (also called **bugs**) may be identified.

- These program errors are caused when the programmer errs in determining the correct steps to be taken in the problem solution, such as an attempt to divide by zero.
- These errors must be corrected in the source program, which must again be compiled and link-loaded for execution.
- The process of program compilation, linking, and execution is shown in Figure 1.3.

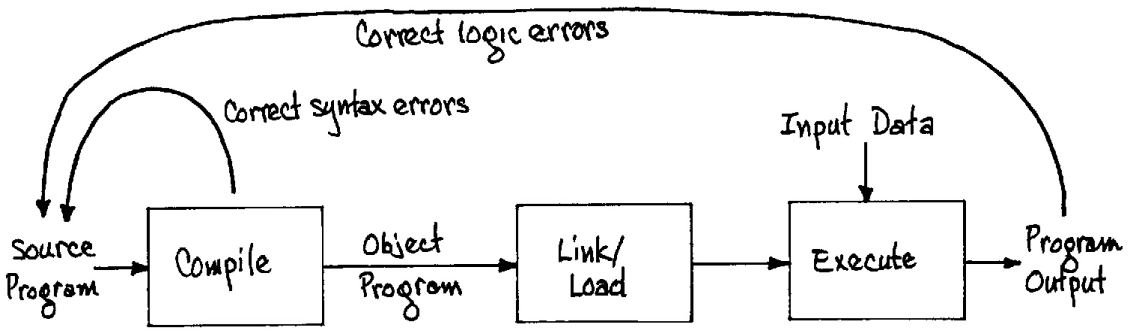

Figure 1.3: High-level language program development

Even when a program executes without an error message, the results must be checked carefully to be sure that they are correct. The computer performs the steps precisely as specified in the source; if the wrong steps are specified, the computer will execute these wrong (but syntactically correct) steps and produce a result that is incorrect.

MATLAB Mathematical Computation Tool

Matlab and other mathematical computation tools are computer programs that combine computation and visualization power that make them particularly useful tools for engineers. MATLAB is both a computer programming language and a software environment for using that language effectively.

The name MATLAB stands for **Mat**rix **lab**oratory, because the system was designed to make matrix computations particularly easy. Don't worry if don't know what a matrix is, as this will be explained later. The MATLAB environment allows the user to manage variables, import and export data, perform calculations, generate plots, and develop and manage files for use with MATLAB.

The MATLAB environment is an **interactive** environment:

- Single-line commands can be entered and executed, the results displayed and observed, and then a second command can be executed that interacts with results from the first command that remain in memory. This means that you can type commands at the MATLAB prompt and get answers immediately, which is very useful for simple problems.
- MATLAB is a executable program, developed in a high-level language, which *interprets* user commands.
- Portions of the MATLAB program execute in response to the user input, results are displayed, and the program waits for additional user input.

- When a command is entered that doesn't meet the command rules, an error message is displayed. The corrected command can then be entered.
- Use of this environment doesn't require the compile-link/load-execution process described above for high-level languages.

While this interactive, line-by-line execution of MATLAB commands is convenient for simple computational tasks, a process of preparation and execution of programs called **scripts** is employed for more complicated computational tasks:

- A **script** is list of MATLAB commands, prepared with a text editor.
- MATLAB executes a script by reading a command from the script file, executing it, and then repeating the process on the next command in the script file.
- Errors in the syntax of a command are detected when MATLAB attempts to execute the command. A syntax error message is displayed and execution of the script is halted.
- When syntax errors are encountered, the user must edit the script file to correct the error and then direct MATLAB to execute the script again.
- The script may execute without syntax errors, but produce incorrect results when a logic error has been made in writing the script, which also requires that the script be editted and execution re-initiated.
- Script preparation and debugging is thus similar to the compile-link/load-execution process required for in the development of programs in a high-level language.

1.4 Computing Terminology

Definitions of computing terms:

Command: A user-written statement in a computer language that provides instructions to the computer.

Variable: The name given to a quantity that can assume a value,.

Default: The action taken or value chosen if none has been specified.

Toggle: To change the value of a variable that can have one of two states or values. For example, if a variable may be "on" or "off" and the current value is "on," to toggle would change the value to "off."

Arguments: The values provided as inputs to a command.

Returns: The results provided by the computer in response to a command.

Execute: To run a program or carry out the instructions specified in a command.

Display: Provide a listing of text information on the computer monitor or screen.

Echo: To display commands or other input typed by the user.

Print: To output information on a computer printer (often confused with "display" in the textbook).