CHAPTER 6

Numerical Computing Formalism

6.1 Introduction

Numerical computation enables you to compute solutions to numerical problems, provided you can frame them into a proper format. This requires certain considerations. For example, if you digitize continuous functions, then you are going to introduce certain errors due to the sampling at a finite frequency. Hence, a very accurate result would require very a fast sampling rate. When a large data set needs to be computed, it becomes a computationally intensive and time consuming task. Also you must understand that the numerical solutions are an approximation at best, compared to analytical solutions. The onus of finding their physical meaning and significance lies on you. The art of discarding solutions that do not have meaning in real world scenarios is something that a scientist/engineer develops over the years. Also, a computational device is only as intelligent as its operator. The law of GIGO (garbage-in-garbage-out) is followed very strictly in this domain.

This chapter attempts to explain some of the important steps you must consider in order to solve a physical problem using numerical computations. Defining a problem in the proper terms is just the first step. Making the right model and then using the right method to solve (solver) the issue is the difference between a naive and an experienced scientist/engineer.

S. Nagar, *Introduction to MATLAB for Engineers and Scientists*,
https://doi.org/10.1007/978-1-4842-3189-0_6

6.2 Physical Problems

Everything in our physical world is governed by physical laws. Owing to men and women of science who toiled under difficult circumstances and came up with fine solutions to the things happening around us, we obtained mathematical theories for physical laws. To test these mathematical formalisms of physical laws, we use numerical computations. If it yields the same results as that of a real experiment, they validate each other. Numerical simulations can remove the need to do an experiment altogether, provided you have a well tested mathematical formalism. For example, nuclear powers of our times need not test nuclear bombs for real any more. The data related to nuclear explosion, which was obtained during real nuclear explosions, enables scientists to model these physical systems quite accurately, thus eliminating the need to do real testing.

Apart from applications like simulating a real experiment, modeling physical problems are good educational exercises. While modeling, hands-on exercises enable students to explore the subject in depth and give proper meaning to the topic under study. Solving numerical problems and visualizing results makes the learning permanent and also elucidates any flaws in the mathematical theory, which ultimately leads to new discoveries.

6.3 Defining a Model

Modeling means writing equations for a physical system. As the name suggests, an equation is about equating two sides. An equation is written using an equals (=) sign, where terms on the left side are equal to terms on the right side. The terms on either side of an equation can be numbers or expressions. For example:

$$3x + 4y + 9z = 10$$

This equation has the term $3x + 4y + 9z$ on the left hand side (LHS) and the term 10 on the right hand side (RHS). Note that whereas LHS is an algebraic term, RHS is a number.

Expressions are written using functions, which is simply a relationship between two domains. Like $f(x) = y$ is a relationship from y to x using the rules of algebra. Mathematics has a rich library of functions, which you can use to make expressions.

Choosing the proper functions depends on the problem. Some functions describe some situations best. For example, the oscillatory behavior can be described in a reasonable manner using trigonometric functions like $sin(x)$, $cos(x)$, etc. Objects moving in straight lines can be described well using linear equations like $y = mx + c$, where x is the present position, m is the constant rate of change of x, and c is the offset position. Objects moving in a curved fashion can be described by various non-linear functions (where the power of the dependent variable is not 1).

In real life, you can have situations that are a mixture of these scenarios. An object can oscillate and move in a curved fashion at the same time. In that case, you write an expression using a mixture of functions or find new functions that can explain the behavior of the object. Verifying the functions is done by finding solutions to equations describing the behavior and matching it with observations of the object. If they match perfectly, you have a perfect solution. In most cases, an exact solution might be difficult to obtain. In these cases, you get an "approximate" solution. If the errors involved while obtaining an approximate solution are within tolerable limits, the models can be acceptable.

As discussed, physical situations can be analytically solved by writing mathematical expressions in terms of functions involving dependent variables. The simplest problems have simple functions between dependent variables with a single equation. There can be situations where multiple equations are needed to explain a physical behavior. In case of multiple equations being solved, the theory of the matrix comes in handy.

Suppose the following equations define the physical behavior of a system:

$$-x+3y=4 \qquad \text{(Equation 6-1)}$$

$$2x-4y=-3 \qquad \text{(Equation 6-2)}$$

Then this system of two equations can be represented by a matrix equation, as follows:

$$\begin{bmatrix} -1 & 3 \\ 2 & -4 \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$

Now using matrix algebra, values of variables *x* and *y* can be found such that they satisfy the equations. Those values are called *roots* of these equations. These roots are the point in 2D space (because there are two dependent variables) where the system will find stability for that physical problem. In this way, you can predict the behavior of system without actually doing an experiment.

Mathematical concepts of differentiation and integration become very important when you need to work with dynamic systems. When the system is constantly changing the values of its dependent variables to produce a scenario, it's important to know the rate of change of these variables. When these variables are independent of each other, you can use simple derivatives to define their rate of change. When they are not independent of each other, you must use partial derivatives for the same.

For example, Newton's second law of motion says that the rate of change of velocity of an object is directly proportional to the force applied on it. Mathematically:

$$F \alpha \frac{dy}{dx} \qquad \text{(Equation 6-3)}$$

The proportionality is turned into equality by substituting for a constant of multiplication `m` such that:

$$F = m \times \frac{dy}{dx} \qquad \text{(Equation 6-4)}$$

If you know values or expressions for `F`, this equation can be solved analytically and solutions can be found to this equation. But in some cases, the analytical solution may be too difficult to obtain. In those cases, you can digitize the system and find a numerical solution.

There are many methods to digitize and numerically solve a given function. Programs used to implement a particular method to solve a function numerically are called *solvers*. A lot of solvers exist to solve a function. The choice of solver is critical to successfully obtain a solution. For example, Equation 6-4 is a differential equation. It is a first order ordinary differential equation. A number of solvers exist to solve such problems, like Euler, Runge-Kutta, etc. The choice of the particular solver depends on the accuracy of its solution, the time taken for obtaining a solution, and the amount of memory used during the process. The last point is especially important when memory is not an freely expendable commodity, such as when you're using micro-computers with limited memory storage.

The advantage of using MATLAB to perform numerical computations lies in the fact that it has a very rich library of functions to perform the various tasks required. The predefined functions have been optimized for speed and accuracy (in some cases, accuracy can be predefined). This enables you to rapidly prototype the problem instead of concentrating on writing functions to do basic tasks and optimizing them for speed, accuracy, and memory usage.

6.4 Example: Polynomials

The coefficients of a vector are defined as elements of a vector. In this manner, a coefficient is defined for numerical computing. For example, consider defining two arrays, `p1` and `p2`, as shown:

```
>> p1 = [1 0 3 2]

p1 =

1     0     3     2

>> p2 =[3 4 0 5]

p2 =

3     4     0     5
```

The corresponding polynomial for `p1` is $p_1(s) = s^3 + 3s - 2 = 0$ and for `p2`, it's $p_2(s) = 3s^3 + 4s^2 - 5 = 0$. See Figure 6-1.

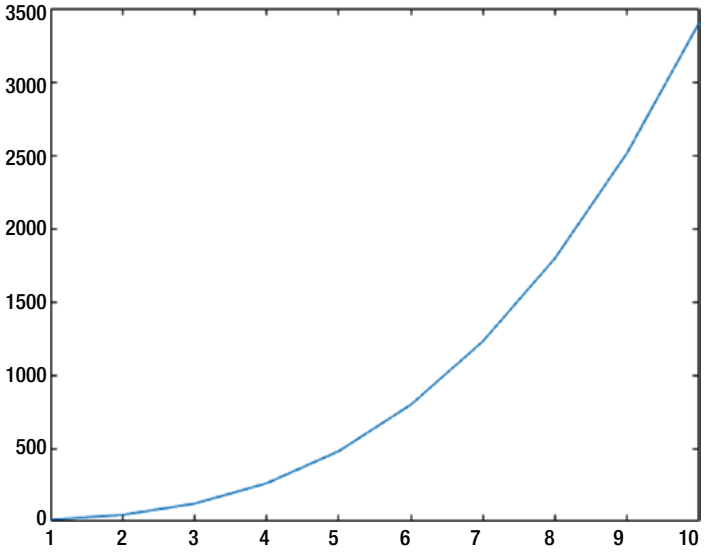

Figure 6-1. *Plot for equation* $p_2(s) = 3s^3 + 4s^2 - 5 = 0$

6.4.1 polyval()

Polynomials can be evaluated for a single value or multiple values using the `polyval()` function. Consider the polynomials defined in `p1` and `p2`. Let's calculate the values for $p_1(5)$ and p_{2}(-2).

```
>> s = 5

s =

5

>> polyval(p1,s)

```

```
 9  ans =
10
11  142
12
13  >> s=2
14
15  s =
16
17  2
18
19  >> polyval(p2,s)
20
21  ans =
22
23  45
```

If a polynomial needs to be calculated on multiple values, say from 1 to 10 for p_1, then an array `1:10` can be fed to the `s` variable and this can be used in the `polyval()` function.

```
1  >> s = 1:10;
2  >> polyval(p2,s)
3
4  ans =
5
6  12     45      122       261       480       797
   1230   1797    2516      3405
```

This facility can be used to plot polynomials easily. The `plot()` command can be fed `s` and `polyval()` output as the x and y axes to visualize a plot.

6.4.2 roots()

The *roots* of a polynomial are the numerical values where the evaluated polynomial is valued at zero. Roots can be found easily using the `roots()` function. Here's an example using the previously defined polynomials, `p1` and `p2`.

```
>> p1

p1 =

1     0     3     2

>> p2

p2 =

3     4     0     5

>> roots(p1)

ans =

0.2980 + 1.8073i
0.2980 - 1.8073i
-0.5961 + 0.0000i

>> roots(p2)

ans =

```

```
25  −1.8307 + 0.0000i
26  0.2487 + 0.9212i
27  0.2487 − 0.9212i
28
29  >> polyval(p1,roots(p1))
30
31  ans =
32
33  1.0e−14*
34
35  −0.1776 − 0.2720i
36  −0.1776 + 0.2720i
37  0.0888 + 0.0000i
38
39  >> polyval(p2,roots(p2))
40
41  ans =
42
43  1.0e−13*
44
45  −0.2753 + 0.0000i
46  −0.0089 − 0.0111i
47  −0.0089 + 0.0111i
```

As per the definition of a root, the polynomial should be valued at zero at its roots, but the value for `roots(p1,roots(p1))` is not zero. Instead, it's a very small number in the order of 10^{-14}. This is due to errors introduced in the numerical approximations for calculating the roots.

6.4.3 Addition and Subtraction of Polynomials

Two polynomials are added by adding their coefficients. Since they are defined as arrays in MATLAB, polynomial addition and subtraction is simply an element-wise operation.

```
>> p1+p2

ans =

4     4     3     7

>> p1-p2

ans =

-2    -4     3    -3
```

This effectively means that:

$$p_1(s) = s^3 + 3s - 2 = 0 \quad \text{(Equation 6-5)}$$

$$p_2(s) = 3s^3 + 4s^2 - 5 = 0 \quad \text{(Equation 6-6)}$$

$$p_1(s) + p_2(s) = 4s^3 + 4s^2 + 3s + 7 \quad \text{(Equation 6-7)}$$

$$p_1(s) - p_2(s) = -2s^3 - 4s^2 + 3 - 3 \quad \text{(Equation 6-8)}$$

6.4.4 Polynomial Multiplication

The product of two polynomials can be found using a convolution operation, which is provided using the `conv()` function in MATLAB.

```
>> p1

p1 =

1     0     3     2

>> p2

p2 =

3     4     0     5

>> conv(p1,p2)

ans =

3     4     9     23     8     15     10
```

$$p_1(s) = s^3 + 3s - 2 = 0 \quad \text{(Equation 6-9)}$$

$$p_2(s) = 3s^3 + 4s^2 - 5 = 0 \quad \text{(Equation 6-10)}$$

$$P_1(a) \times p_2(a) = 3s^6 + 4s^5 + 9s^4 + 23s^3 + 8s^2 + 15x + 10 = 0 \quad \text{(Equation 6-11)}$$

6.4.5 Polynomial Division

Polynomial division is performed by using deconvolving operations, which are provided by the `deconv()` function. It gives two outputs—a quotient and a remainder.

```
>> p1

p1 =

1     0     3     2

>> p2

p2 =

3     4     0     5

>> [q,r] = deconv(p1,p2)

q =

0.3333


r =

0     −1.3333     3.0000     0.3333

>> [q,r] = deconv(p2,p1)

q =

3
```

```
31  r =
32
33  0     4     −9     −1
```

This means that if:

$$p_1(s) = s^3 + 3s - 2 = 0 \quad \text{(Equation 6-12)}$$

$$p_2(s) = 3s^3 + 4s^2 - 5 = 0 \quad \text{(Equation 6-13)}$$

Then:

$$\frac{p_1}{p_2} \rightarrow q = 0.333, r = -1.3333s^2 + 3s + 0.3333 \quad \text{(Equation 6-14)}$$

$$\frac{p_2}{p_1} \rightarrow q = 3, r = -4s^2 - 9s - 1 = 0 \quad \text{(Equation 6-15)}$$

6.4.6 Polynomial Differentiation

Polynomial differentiation can be accomplished using the `polyder()` function. For example, say you have a polynomial $y(x) = x^3 - 2x^2 + 4x - 5 = 0$. That means:

$$\frac{dy}{dx} = 3x^2 - 4x + 4 = 0$$

This can be calculated by MATLAB as follows.

```
1  >> y = [1 −2 4 −5]
2
3  y =
4
5  1     −2     4     −5
6
```

```
>> dydx = polyder(y)

dydx =

3     −4     4
```

6.4.7 Polynomial Integration

Just as with differentiation, you can define integration of polynomials using the `polyint()` function. For example, say you have a polynomial $y(x) = x^3 - 2x^2 + 4x - 5 = 0$. Then:

$$\int y(x)dx = 0.25x^4 - 0.6667x^3 + 2x^2 - 5x = 0$$

```
>>>> y = [1 −2 4 −5]

y =

1     −2     4     −5

>> integration =vpolyint(y)

integrationv=

0.2500     −0.6667     2.0000     −5.0000     0
```

6.4.8 Polynomial Curve Fitting

Suppose you are given some data and need to find a polynomial that fits the data. This task can be performed using the `polyfit()` function. For example, suppose you want to fit the data given here:

x	1	2	3	4	5	6
y	10	11	21	2	3	7

```
>> x = [1,2,3,4,5,6]

x =

1     2     3     4     5     6

>> y = [10,11,21,2,3,7]

y =

10     11     21     2     3     7

>> polyfit(x,y,2)

ans =

−0.3750     0.9679     11.3000

>> polyfit(x,y,3)

ans =

1.0833     −11.7500     35.3095     −16.0000
```

Second and third degree polynomials that fit the data are $-0.375x^2 + 0.9679x + 11.3 = 0$ and $1.0833x^3 - 11.75x^2 + 35.3095x - 16 = 0$, respectively.

6.5 Summary

Almost all branches of science and engineering require you to perform numerical computations. MATLAB is one of the alternatives for doing so. MATLAB has a library of optimized functions for general computation. It also has a variety of packages that perform specialized jobs. This makes it an ideal choice for prototyping a numerical computation problem efficiently. This chapter summarized various issues related to errors generated during numerical computation and various methods to obtain their value or order of magnitude. These quantities are important to measure, since in real life, you will need these values to define the accuracy of the final product.