Solving Differential Equations Using MATLAB

Abraham Asfaw
aasfaw.student@manhattan.edu

November 28, 2011

1 Introduction

In this lecture, we will follow up on lecture 2 with a discussion of solutions to Ordinary Differential Equations (ODEs) using MATLAB. We will discuss standard methods of solution, and an additional method based on simulation of linear systems. Throughout the lecture, examples will be used to demonstrate these methods. Interested readers should refer to the various references available online on related topics, such as numerical methods and the solutions of partial differential equations. In order to encourage a trial-and-error method of learning the methods contained herein, no MATLAB outputs are shown.

2 Solving ODEs

Let us begin with a simple example.

$$\frac{dy}{dx} = y(x)$$

The MATLAB code to solve this ODE with no initial conditions is shown below.

```
ODE1 = 'Dy = y'
ODE1solved = dsolve(ODE1, 'x')
```

We can specify initial conditions for the ODE as follows.

```
initConds = 'y(0) = 5'
ODE1solved = dsolve(ODE1, initConds, 'x')
```

As shown in the previous lectures, MATLAB makes plotting functions easy. We can plot our solved function as follows.

```
x = -5:0.01:5;
y_values = eval(vectorize(ODE1solved));
plot(x,y_values)
```

The same ideas apply to higher ODEs. Here, we solve a second-order ODE with initial values at $y(0)$ and $y'(0)$. We then plot the function in the range [-5,5].

```
ODE2 = '3*D2y - Dy + 6*y = 6*sin(t) + 2*cos(t)'
initConds = 'y(0) = 1, Dy(0)=2'
ODE2solved = simplify(dsolve(ODE2, initConds));
pretty(ODE2solved)
t = -5:0.01:5;
y_values = eval(vectorize(ODE2solved));
plot(t,y_values)
```

3 Solving Systems of ODEs

Systems of ODEs can be solved in a similar manner. One simply defines each equation as before. The only thing that chances is the return value of the *dsolve* function, which is now an array containing the explicit solutions of each of the functions in the system.

```
sysODE1 = 'Dx = 2*x + 3*z'
sysODE2 = 'Dy = 6*z - y'
sysODE3 = 'Dz = 3*y - 12*x'
initConds = 'x(1) = 5, y(2) = 3, z(9) = 0'
[x,y,z] = dsolve(sysODE1,sysODE2, sysODE3, initConds)
```

4 Laplace Transforms

Laplace transforms are handy in solutions of differential equations when the transforms of the forcing functions are known and can easily be inverted with slight modification. Typically, these include sinusoidal forcing functions, making this method ideal in the study of linear systems. Here is an example where we solve the differential equation below with zero initial conditions.

$$3\frac{d^2y}{dt^2} - \frac{dy}{dt} + 6y = 6sin(t) + 2cos(t)$$

```
syms s; syms t;
lhs = 3*s^2 - s + 6;
rhs = laplace(6*sin(t)) + laplace(2*cos(t));
soln = simplify(ilaplace(rhs*1/lhs))
```

5 Solving ODEs Using Simulation

This will be demonstrated in the lectures. The basic theory is as follows. Assume that we would like to simulate the system characterized by the following differential equation.

There are several ways of doing so. One such way is to take the Laplace Transform of both sides, thereby acquiring a transfer function. In `MATLAB`, particularly in `Simulink`, there is a `transfer function` block which will take the coefficients of the numerator and denominator to generate the transfer function. That block can be fed several types of inputs, and conclusions can be made about the outputs.

In this lecture, we discuss another interesting method of solution, namely the use of block diagrams in representing (and solving) differential equations. Consider the equation

$$6\frac{d^2y}{dt^2} + 5\frac{dy}{dt} + 9y = 2u(t)$$

with some initial conditions $y(0)$ and $y'(0)$. Note that we are dealing with a causal system in which the forcing function begins at zero. We can rewrite the above ODE as

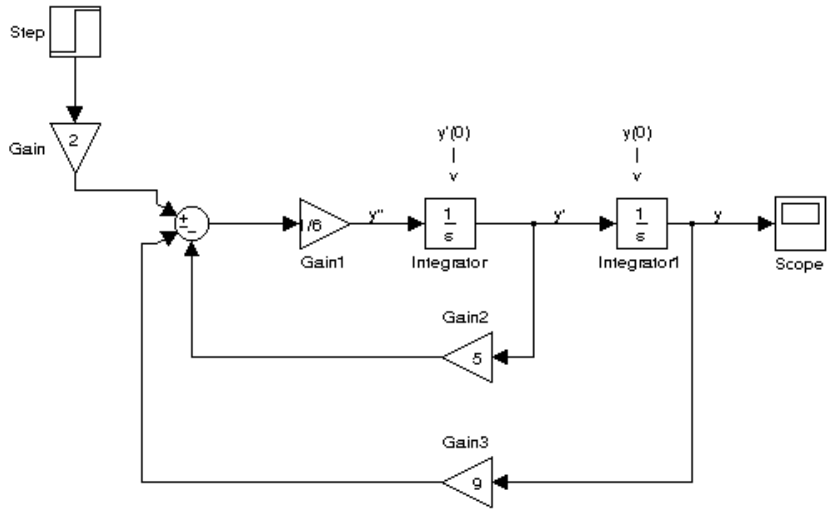
$$\frac{d}{dt}\left(\frac{dy}{dt}\right) = \frac{1}{6}\left(-5\frac{dy}{dt} - 9y + 2u(t)\right)$$

Let us integrate both sides of the differential equation twice ignoring the initial conditions for the moment.

$$y(t) = \frac{1}{6}\left[-5\left(\int y(t)\right) - 9\left(\int\int y(t)\right) + 2\left(\int\int u(t)\right)\right]$$

In `Simulink`, such a description is easily represented as shown below. Note that the initial conditions are fed to the integrators.

Figure 1: `Simulink` Model of an ODE



We can add a `To Workspace` block to the block diagram to observe the output in a normal `MATLAB` plot instead of the scope in `Simulink`. Assuming that the output variable of such a block is `simout`, the following code will plot the simulation outputs. Note the use of the dots to navigate through the `simout` struct.

```
plot(tout, simout.signals.values)
```

6 Conclusion

MATLAB presents several tools for modeling linear systems. These tools can be used to solve differential equations arising in such models, and to visualize the input-output relations. Further reading on this topic would cover material such as system modeling using commands such as `sys` and `tf`, advanced usage of `Simulink` in laboratory measurement, and modeling tied systems using numerical methods. The interested reader is referred to the internet, where a simple search for `Modeling Dynamical Systems Using Simulink` points to several interesting introductory articles on MATLAB's true power in the process of modeling linear systems.