MM 220 Autumn 2020

Tutorial 3: Solving Differential Equations using MATLAB

Anirban Patra, Amrita Bhattacharya

Department of Metallurgical Engineering and Materials Science, IIT Bombay

We will learn how to solve differential equations using MATLAB in this tutorial. Until now, most of the functions we have learnt in Tutorial 1 and 2 can be found in the open-source MATLAB alternative Octave as well. Today we will make use of the Symbolic Math Toolbox, which is specific to MATLAB only!

Lesson 1: *Differentiating equations using MATLAB*

First we learn how to create symbolic variables and functions in MATLAB and how to differentiate them. At the Command Window, type:

```
>> clear all;
>> syms t;
>> f = sin(t);
>> dfdt = diff(f,t)
dfdt =
cos(t)
```

Here we create a symbolic variable `t` in MATLAB using `syms`. And then define our function `f = sin(t)`. The function `diff` is used to calculate the derivative of `f` with respect to `t`. Now if we want to calculate the second derivative of `f` with respect to `t`, we simply use:

```
>> d2fdt2 = diff(f,t,2)
d2fdt2 =
-sin(t)
```

The function `diff` can also be used to calculate partial derivatives if we have multiple variables in our equation. See the example below.

```
>> clear all;
>> syms x t;
>> f = exp(t)*sin(x*t);
>> dfdt = diff(f,t)
dfdt =
exp(t)*sin(t*x) + x*exp(t)*cos(t*x)
```

```
>> dfdx = diff(f,x)
dfdx =
t*exp(t)*cos(t*x)
```

Here `f = exp(t)*sin(x*t)` is a function of `x` and `t`. We store the derivative of `f` with respect to `t` in `dfdt` and that with respect to `x` in `dfdx`.

Now, we will learn how to integrate differential equations in MATLAB in two different ways: explicitly and numerically.

Lesson 2: *Explicit integration*

Let us say we want to solve the ODE: $dy/dt = 2yt^2$. This is an ordinary differential equation (ODE), i.e., a differential equation which is a function of only one independent variable.

In order to explicitly solve this ODE, we use the function `dsolve` in MATLAB. We first create a symbolic variable `y` which is a function of the independent variable `t`. This is done using `syms`. Then we store our ODE in `eqn1`. To solve this, we simply use:

```
>> clear all;
>> syms t;
>> eqn1 = 'Dy = 2*y*(t^2)';
>> y1 = dsolve(eqn1)
y1 =
C2*exp((2*t^3)/3)
```

Here we define a symbolic equation in `eqn1`. `Dy` is used to denote dy/dt. If we have a second order term (d^2y/dt^2), we use `D2y`, i.e.,

```
>> eqn2 = 'D2y = sin(t)';
>> y2 = dsolve(eqn2)
y2 =
C11 - sin(t) + C10*t
```

MATLAB assumes names for the integration constants such as `C2`, `C11`, `C10`, etc. If we provide initial conditions to MATLAB, it can evaluate the values of these integration constants as well.

```
>> ic = 'y(0) = 5'
>> y1 = dsolve(eqn1,ic)
```

```
y1 =
5*exp((2*t^3)/3)
```

Here the value of `C2` is computed as `5`.

Once we have an expression for the integrand, we might want to perform different matrix operations on it. These symbolic equations are stored in MATLAB as strings. We first need to convert our expression from a string to an interpretable expression and calculate the corresponding values. For this we use the function `subs`.

```
>> t = [0:0.1:1];
>> y_explicit = subs(y1);
>> plot(t,y_explicit,'Color','r');
>> hold on;
>> xlabel('t');
>> ylabel('y');
```

Lesson 3: *Numerical integration*

It is possible to explicitly integrate a differential equation if the equation is simple enough. However, for more complex equations we need to use numerical integration. In this lesson, we will integrate the same equation using the `ode45` function in MATLAB. MATLAB implements a combination of 4th and 5th order Runge-Kutta (R-K) formulations for the `ode45` function.

For a given differential equation, $dy/dt = f(y,t)$, the R-K formulation of order s has the following form:

$$y_{t+1} = y_t + h\sum_{i=1}^{s} a_i k_i ,$$

where,

$$k_1 = f(y_n, t_n),$$

$$k_2 = f(y_n + h(a_{21}k_1), t_n + c_2 h),$$

$$\ldots$$

$$k_s = f(y_n + h(a_{s1}k_1 + \ldots \; + a_{s,s-1}k_{s-1}), t_n + c_s h)$$

The coefficients a_{ij} and c_j are given *a priori* based on the particular method.

In order to use ode45, first we define our function dydt using the function handle @(t,y). ode45 also takes as input the range tspan over which the equation is to be integrated, and the initial value y0 at $t = 0$ of the integrand.

```
>> dydt = @(t,y) 2*y.*(t.^2);
>> tspan = [0 1];
>> y0 = 5;
>> [t,y_numerical] = ode45(dydt,tspan,y0);
```

Note that this method does not use symbolic variables. Instead it uses variables in the more conventional MATLAB (matrix) way! There are other ODE solvers in MATLAB such as ode23 and ode113 that use R-K methods of different orders.

Now we plot the numerical solution and compare it with the explicit solution.

```
>> plot(t,y_numerical,'Color','g');
>> legend('Explicit soln','Numerical soln');
>> hold off;
>> print('plot1','-dpng');
```

Assignment

Create a MATLAB function called MM220A3 with the file name MM220A3.m. Write all your code in this file.

The Avrami equation, also known as the Johnson-Mehl-Avrami-Kolmogorov (JMAK) equation, is generally used to model the transformation kinetics of a solid from one phase to another. The rate of transformation is given by the equation:

$$df/dt = \left(nt^{n-1}/\tau^{n}\right) exp\left(-(t/\tau)^{n}\right)$$

where, f is the fraction of phase transformed, t is the time taken, and τ, n are material constants. For an Fe-Cr alloy, the material constants for the transformation from the σ phase to the α phase at 1113 K are: $\tau = 1236\ s$, and $n = 4.2$.

(a) Integrate the above expression explicitly using the dsolve function and plot the fraction of phase transformed as a function of time from 0 s to 1800 s. Plot this data using a green line.

(b) Now integrate the above equation numerically using the `ode45` function to obtain the fraction of phase transformed as a function of time from 0 s to 1800 s. Plot this data using a red line.

(c) An experiment was conducted to measure the phase transformation kinetics of this alloy. The following are the experimental measurements of the fraction transformed as a function of time (data courtesy: Alice Mikikits-Leitner, Masters thesis, University of Vienna, 2009):

Time (s)	*Fraction transformed (f)*
300	0.039
540	0.050
660	0.065
780	0.108
960	0.285
1080	0.458
1260	0.661
1440	0.857
1620	0.935
1800	0.967

On the same graph, plot the experimental data using the '`*`' symbol.

Please label your axes properly and also give the legend for the three plots.

If you have done everything correctly, there should be a good correlation between the model predictions and the experimental data!

Please use the following variable names in the code as these will be used for evaluating the test cases: **f_explicit**: for values obtained using explicit integration; **f_numerical**: for values obtained using numerical integration.