Solving Differential Equations with Matlab

Introduction

In today's recitation, we will take a quick look at the MATLAB library for solving ordinary differential equations (ODEs). MATLAB provides a variety of functions, and uses a variety of algorithms to solve a broad class of equations. While these algorithms are not geared specifically toward solving Newton's equations of motion and computing molecular dynamics trajectories, they are general enough to be used toward that end, and therefore are of interest to us.

We will focus on the MATLAB function ode23(), which is a low-order accuracy differential equation solver. There are several other functions in the ODE library, but the only other one we will mention here is ode45(), which provides medium-order accuracy solutions at the cost of taking up more processor time. Both functions are completely identical in syntax and usage.

Both ode23() and ode45() numerically approximate the function $x(t)$ by solving differential equations of form $dx/dt = f(t, x)$, where t is a scalar time parameter. In order to solve the differential equation, ode23() needs to know the function to integrate, f, along with the initial conditions x_0 and the time interval over which to integrate. A call to ode23() might look like this:

```
>> [T,X] = ODE23('F',TSPAN,X0)
```

Here, 'F' is the name of the file containing the Matlab function that evaluates $f(t, x)$. TSPAN is a row vector, containing the starting and ending times of the time interval (for instance, [0 10]). X0 is a row vector containing the initial conditions (for instance, the position and velocity of a particle at the beginning of the interval TSPAN).

Of the above three parameters, the first is the most interesting. 'F' is the MATLAB function we must write to evaluate the mathematical function $f(t, x)$. While it is executing, ode23() will make repeated calls to 'F', in order to evaluate $f(t, x)$ for various t's and x's. Hence, ode23() expects 'F' to obey a certain format. The basic format for the file 'F' is as follows:

```
function Xprime = F(t, X)

% X should be a column vector, and the returned value Xprime should
% also be a column vector, with the same number of elements as X.

Xprime = zeros(size(X,1), 1);
Xprime = <<<  Some function of X and t  >>>;
```

Note that after you have written 'F', there is no need to ever call it directly from your code; this will be done by ode23() itself. Also, note that while we pass X0 to ode23() as a row vector, ode23() will pass every X to 'F' as a column vector, and it expects 'F' to return Xprime as a column vector as well. This is a convention of the ODE library, albeit not a very intuitive one.

Finally, let us look at the values returned by ode23() itself. By default, ode23() returns two column vectors: T and X. T is the array of all points in time that the equation solver stepped through during the interval TSPAN, and X contains the values of $x(t)$ for each time t in T. Thus, T and X define the trajectory of the particle during the time interval TSPAN.

Example 1: A Simple Differential Equation

Before we proceed to use the ODE library to solve Newton's equations of motion, we will first use it to solve a much simpler differential equation. Consider the following equation:

$$\frac{dx}{dt} = -2tx$$

First, note that we can solve this equation analytically, and thus obtain the exact value for $x(t)$ as follows:

$$\begin{aligned} & \frac{dx}{dt} = -2tx \quad \Rightarrow \quad \frac{1}{x}\ dx = -2t\ dt \quad \Rightarrow \quad \int \frac{1}{x}\ dx = \int -2t\ dt \\ \Rightarrow \quad & \ln x = -t^2 + C \quad \Rightarrow \quad x(t) = C\mathrm{e}^{-t^2}. \end{aligned}$$

Here, C is a constant depending on the value of x at time $t = 0$. If $x(0) = x_0$, then $x_0 = x(0) = C\mathrm{e}^0 = C$. Hence, $x(t) = x_0 \exp(-t^2)$.

Let us now solve this differential equation using Matlab's ODE solver, and compare the solver's answer against the analytical result. First, we need to write the MATLAB function that computes $-2tx$.

```
function Xprime = ODEex1(t, X)

% Function to be use by ODE to solve the equation dx/dt = -2tx
% Compute and return -2tx

Xprime = -2*t*X;
```

Now, let us solve the differential equation, and plot the trajectory of x through time on the interval $tin[0, 5]$ starting at position $x_0 = 1$. Then, we will plot the analytical trajectory of $x(t)$ on the same time interval, and compare the two results.

```
>> TSPAN = [0 5];
```

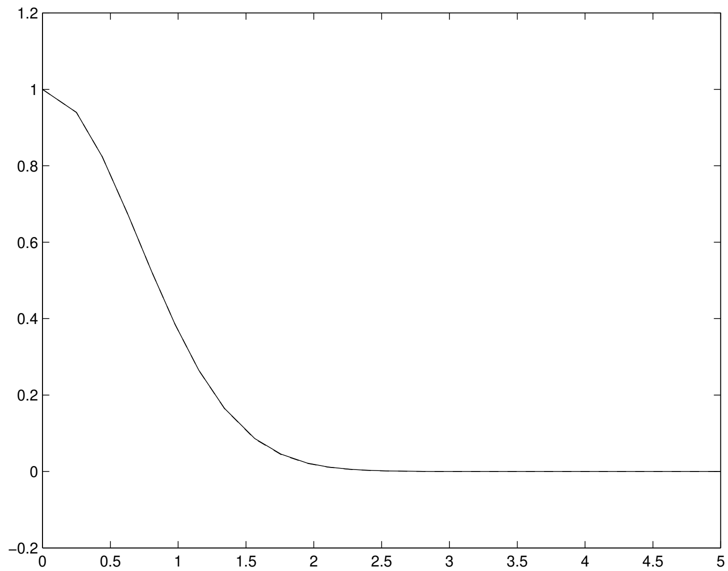

Figure 1: *Plots of numerical and analytical trajectories for $dx/dt = -2tx$. The two coincide completely, at least to the naked eye.*

```
>> X0 = 1;
>> [T, Xbar] = ode23('ODEex1', TSPAN, X0);
>> X = X0 * exp(-T.^2);
>> plot(T,X,'b',T,Xbar,'r--')
>> plot(T, (Xbar-X)/X);      % Plot the relative errors
```

So far, so good! As figure 1 indicates, the numerical and analytical solution coincide completely. In fact, if we examine the relative errors in the numerical trajectory as shown in Figure 2 on the next page, we see that they start off extremely small, and quickly tend to zero as t increases. Therefore, in this case, solving the differential equation numerically worked quite well.

Example 2: Motion of a Simple Harmonic Oscillator

We will now solve a differential equation describing the motion of a simple harmonic oscillator, such as a spring. This is a simple case of a Newton's motion equation, where the analytical solution for the trajectory is known. This will once again allow us to compare the numerical solution to the analytical one.

The motion equation for a simple harmonic oscillator is given by:

$$m\frac{d^2x}{dt^2} = -kx \tag{1}$$

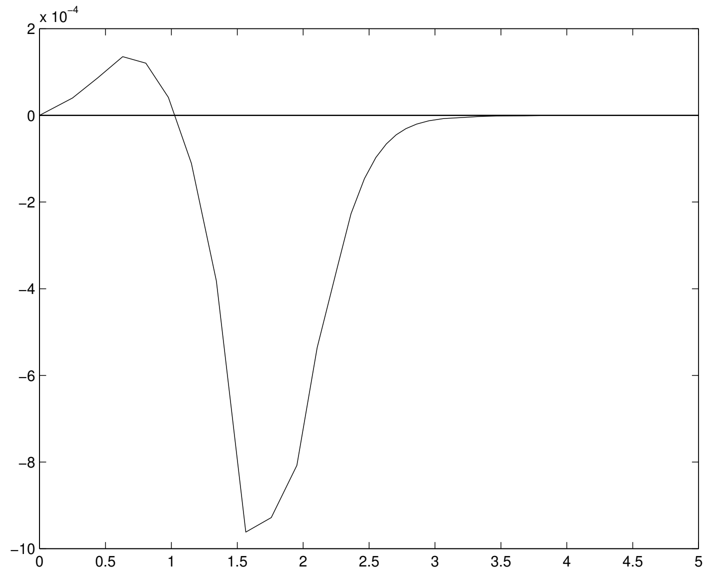

Figure 2: *Relative error in the trajectory as a function of* t*. The vertical axis is scaled by the factor of* 10^{-4}*.*

where m is the mass of the particle, and k is called the spring constant. Without proof, we will state the analytical equation for $x(t)$.

$$x(t) = A\cos(\omega t + \phi) \tag{2}$$

where ω is the frequency of the oscillation, A is its amplitude, and ϕ is its phase. The frequency, ω, depends solely on the mass and the the spring constant: $\omega = \sqrt{k/m}$. The amplitude and phase, on the other hand are parameters that depend solely on the initial position and velocity of the oscillating particle. If at time $t = 0$ the particle was at rest at position x_0 (that is, $x(0) = x_0$ and $v(0) = x'(0) = 0$), then the trajectory can be described by the equation:

$$x(t) = x_0\cos(\omega t) \tag{3}$$

Before we attempt to solve the motion equation numerically using ODE, we have to get past one apparent stumbling block. The ODE functions in MATLAB only solve first-order differential equations: that is, equations of form $dx/dt = f(t,x)$. Our equation, however, is second-order, since it contains a d^2x/dt^2 term. Before we can use an ODE solver on this equation, we must convert to first-order form. We do this by breaking up the equation into a system of two first-order differential equations, as follows: Let $v(t) = x'(t)$. Then,

$$\begin{aligned} dx/dt &= v \\ dv/dt &= -(k/m)x \end{aligned}$$

It is straightforward to encode this system of equations into a MATLAB function. Let $\mathsf{Y} = [\mathsf{x}\ \mathsf{v}]'$. Then, according to our equations,

$$\mathsf{Yprime} = \begin{bmatrix} dx/dt \\ dv/dt \end{bmatrix} = \begin{bmatrix} v \\ -(k/m) * x \end{bmatrix} = \begin{bmatrix} \mathsf{Y}(2) \\ -(k/m) * \mathsf{Y}(1) \end{bmatrix}$$

The function oscillate() representing the motion equation for the simple harmonic oscillator is given below. ode23() can now call this function in order to solve the motion equation. Note that since so far we have not discusses how to get ode23() to pass additional parameters to the functions it calls, we cannot pass the values of m and k to oscillate(), and hence have to hard-code them inside the file.

```
function Yprime = oscillate(t, Y)

% This function evaluates the motion equation for a simple
% harmonic oscillator.

% Hard-code the values of m and k, for now.
m = 1;
k = 10/(2*pi);

Yprime = zeros(2,1);           % dy must be a column vector
Yprime(1) = Y(2);              % Corresponds to dx/dt = v (velocity)
Yprime(2) = -(k/m) * Y(1);     % Corresponds to m(dv/dt) = -kx
```

Let us once again call ode23() to find the trajectory of the harmonic oscillator. We will start oscillator off from rest at time $t = 0$ and location $x_0 = 1$, so that we can use Equation 3 on the previous page to describe its analytical trajectory. We will calculate the trajectory on the time interval [0,50]. We will then compare the numerical trajectory against the analytical, and plot the relative error.

```
>> X0 = 1;             % Initial position of oscillator
>> V0 = 0;             % Initial velocity of oscillator (at rest)
>> Y0 = [X0 V0];       % Initial condition vector
>> TSPAN = [0 50];     % Time interval
>> [T23, Y23] = ode23('oscillate', TSPAN, Y0);
>> Xbar23 = Y23(:,1);  % The first column contains x(t), the second v(t)
>>
>> m  = 1;             % Hard-code k and m
>> k  = 10/pi;
>> X23 = X0 * cos(sqrt(k/m)*T23);    % Compute the analytical trajectory
>> plot(T23, (Xbar23-X23)/X23);      % Plot the relative error
```

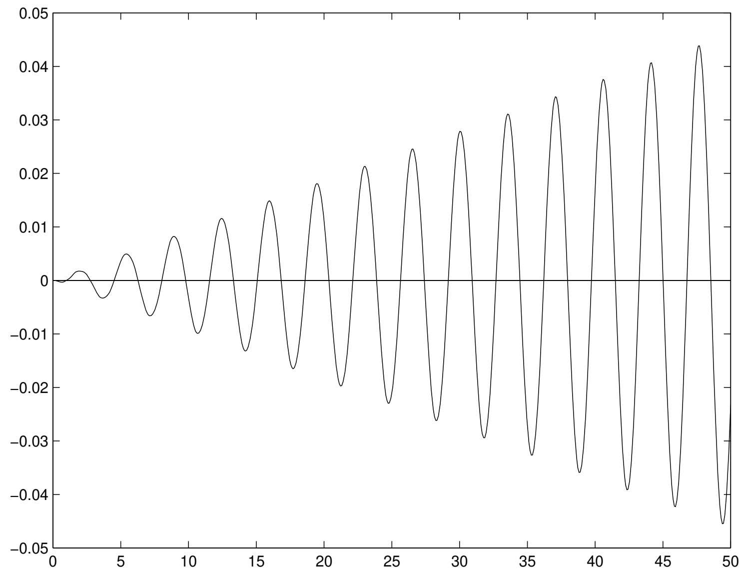

Figure 3: *Relative error in the trajectory of the oscillator as a function of t*

Here, a glance at the plot of relative error makes it apparent that the numerical approximation only works for short time intervals. The amplitude of the relative errors increase linearly with t. When $t = 50$, the relative error is only about $\pm 5\%$, which is still fairly reliable. However, by the time $t = 500$, relative error can be as large as $\pm 40\%$. Thus, it is not viable to use the MATLAB ODE solver to compute the trajectory of the oscillator over long time intervals.