LAB 4: DISCONTINUOUS FORCING, SIGNALS, AND NUMERICAL METHODS, PART A

MATLAB commands we use in this lab include the following.

1. **disp.** Displays text to the command window. For example,
   ```matlab
   >> disp('This is text sent to the command window')
   ```

2. **eulermethod_w19.** This isn't a native MATLAB command; download it from the labs page. The command eulermethod_w19 approximates the solution to a differential equation or system using Euler's method. It takes as arguments the same arguments as we use with ode45\(^1\), plus a step size to use:
   ```matlab
   >> eulermethod_w19(f_handle, [tmin tmax], init_cond, h);
   ```
   For example,
   ```matlab
   >> eulermethod_w19(@(t,x) [x(2); -x(1)], [0 10], [0;1], 0.1);  
   ```

3. **ode45.** Finds a numerical approximation to a differential equation or system of equations:
   ```matlab
   >> [tsol,xsol] = ode45(f_handle, [tmin tmax], init_cond);
   ```
   For example,
   ```matlab
   >> [tsol,xsol] = ode45(@(t,x) [x(2); -x(1)], [0 10], [0;1]);
   ```
   It is possible to set options that determine how ode45 behaves; for example, we can set the maximum step size it is allowed to try by setting up an options object and passing that to ode45:
   ```matlab
   >> options = odeset('MaxStep', 1);
   >> [tsol,xsol] = ode45(@(t,x) [x(2); -x(1)], [0 10], options);
   ```
   (the elipses, ..., are just to break the line here). Of course, in most instances this isn't necessary.

4. **ode15s.** This is another numerical solver for differential equations, and takes exactly the same arguments as ode45. It deals well with stiff systems, for which solutions have regions that change very much faster than they do in others.

5. **plot.** Plot one vector against another; e.g., to plot component plots from output from eulermethod_w19 and ode45,
   ```matlab
   >> plot( tesol,xesol(:,1),'-k', t45sol,x45sol(:,1),'--k' );
   ```

---

\(^1\)Except that it doesn't support the addition of options.
1.6. tic. This starts MATLAB’s internal timer, so that you can see how long a command runs for; see toc.

1.7. toc. This stops MATLAB’s internal timer, so that you can see how long a command runs for. For example, to see how long a call to ode45 takes:

```matlab
>> disp('timing for ode45');
>> tic
>> [t,x] = ode45(@(t,x) [1000*x(2); -1000*x(1)], [0 100],...[-2;5]);
>> toc
```

(obviously, it makes sense to use tic and toc in a script, where the only thing you are measuring is the time for the intervening commands, rather than how long it takes you to type in commands).

2. Background

In this lab we consider a circuit model, which is a second-order, linear, constant-coefficient differential equation. In the prelab we found this to be

$$y'' + 2\gamma y' + \omega_0^2 y = F(t).$$

The characteristic polynomial of the associated homogeneous equation is $$\lambda^2 + 2\gamma \lambda + \omega_0^2$$, with roots $$\lambda = -\gamma \pm \sqrt{\gamma^2 - \omega_0^2}$$. Thus, if $$\gamma$$ is a small (relative to $$\omega_0$$) positive number, then the system is underdamped and the solution can be written in the form $$y_c(t) = Re^{-\gamma t} \cos((\sqrt{\omega_0^2 - \gamma^2} t - \phi_0)$$ for some $$R$$ and $$\phi_0$$.

Previously we considered forcing functions $$F(t)$$ of the form $$F(t) = A \cos(\omega t)$$. In this lab, we instead consider forcing functions, such as the step function $$u_c(t)$$, which are discontinuous.

To solve an equation such as (1) numerically (e.g., with ode45), we rewrite it as a system and the numerical method then uses known data (e.g., the initial conditions and system of equations) to predict the values of the variables $$y$$ and $$y'$$ at a later time. For Euler’s method, which is very simple (and not very accurate), we approximate a solution to $$x' = f(t, x)$$ (or, for a system, $$x' = f(t, x)$$) by

$$x(t_{k+1}) \approx x_{k+1} = x_k + hf(t_k, x_k).$$

Then $$x_{k+1}$$ is an approximation of $$x(t_{k+1})$$ for all $$k \geq 0$$, and so $$x(t_f) \approx x_n$$. When $$f(t, x)$$ changes very rapidly (e.g., discontinuously), an approximation such as this may have difficulty resolving the solution accurately.

3. Part A

Exercises in this section are to be completed by pairs. At the end of Workday 1, pairs should present their work to each other. Note that material from Part A appears in one of your written homework problems and will be relevant for Part B.
Unless otherwise stated, let $I(t) = \frac{1}{a}(u_c(t) - u_{c+a}(t))$. This function is defined by the file `Impulse.m`; you will need to download that from the course website, then define the impulse you want with
\[
>> a = \text{value};
\]
\[
>> c = \text{value};
\]
\[
>> I = @(t) \text{Impulse}(t,c,a);
\]
Note that the $I$ that this defines uses the values $a$ and $c$ have when $I$ is defined; if you want to change those values, you will need to redefine $I(t)$, or create a new function handle $I2$ that uses the different values.

For Euler’s method, download the file `eulermethod_w19.m`. It is described in the MATLAB section, above, and takes the same arguments as `ode45` and `ode15s`, plus a stepsize $h$.

4. Pair 1 Exercises

**Pair 1 Exercise 1.** If $a = c = 1$, what will the graph if $I(t)$ look like? Plot this impulse using `Impulse` to confirm your expectation. How will the graph be different if $a = 0.5$ and $c = 1$? If $a = 1$ and $c = 0.5$? Add both of these to your graph to see.

**Pair 1 Exercise 2.** Let $a = c = 1$. Find numerical solutions to the initial value problem $y'' + y' + 40y = I(t), y(0) = y'(0) = 0$, using `ode45`, `ode15s`, and `eulermethod_w19`. (For the last, use $h = 0.05$.) Plot the solutions (it may be easiest to do this on different graphs) so that you can see the steps that the different methods are taking. How are these different (for `ode45` and `ode15s`, look carefully at the step sizes)? Be sure you can explain why you might see the differences that you do.

Then plot the Euler’s method solution with the solution from `ode15s`. What is the difference? Why?

**Pair 1 Exercise 3.** Next take $c = 1.01$ and $a = 0.5$. Find numerical solutions to the initial value problem with `ode45` and `ode15s` and a maximum $t$ value of at least 3. Then try $c = 1.01$ and $a = 0.1$. Explain what is going on in the latter case.

**Pair 1 Exercise 4.** Continue using $c = 1.01$ and $a = 0.1$. Add an options object (see the MATLAB section, above) to your function calls to `ode45` and `ode15s`, decreasing the maximum step size. How small does the maximum step size have to be for the numerical solutions to be valid?

After finding a maximum step size that works for both methods, add the MATLAB command `tic` and `toc` before and after the calls to `ode15s` and `ode45`. Which is faster?

Show your work for the last three three exercises to pair 2 to come up with answers to the questions: How accurate is Euler’s method? How do the accuracies of and solutions generated by `ode45` and `ode15s` compare? How efficient are they (in terms of the time to generate a solution)?
5. Pair 2 Exercises

**Pair 2 Exercise 1.** If $a = c = 1$, what will the graph if $I(t)$ look like? Plot this impulse using `Impulse` to confirm your expectation. How will the graph be different if $a = 0.5$ and $c = 1$? If $a = 1$ and $c = 0.5$? Add both of these to your graph to see.

**Pair 2 Exercise 2.** Let $a = c = 1$. Find numerical solutions to the initial value problem $y'' + 21y' + y = I(t)$, $y(0) = y'(0) = 0$ using `ode45`, `ode15s`, and `eulermethod.w19`. (For the last, use $h = 0.05$.) Plot the solutions (it may be easiest to do this on different graphs) so that you can see the steps that the different methods are taking. How are these different (for `ode45` and `ode15s`, look carefully at the step sizes)? Be sure you can explain why you might see the differences that you do.

Then plot the Euler’s method solution with the solution from `ode15s`. What is the difference? Why?

**Pair 2 Exercise 3.** Next let $a = 1$. Define impulses $I_1(t), I_2(t), I_3(t)$ and $I_4(t)$ that have, respectively, $c = 1$, $c = 1.5$, $c = 2$, $c = 2.5$ and $c = 3$. (For example,
```matlab
>> a = 1; c1 = 1;
>> I1 = @(t) Impulse(t,c1,a);
>> c1 = 2;
>> I2 = @(t) Impulse(t,c2,a);
and so on—see the note in the MATLAB section above about defining different impulses.) Then find numerical solutions the initial value problems $y'' + 21y' + y = I_j(t)$, $y(0) = y'(0) = 0$, for each of these functions $I_j$ (generate these out to at least $t = 10$). Plot all on the same axes. Then generate, for the same functions $I_j$, solutions with `ode15s`, and make a second plot showing those solutions. How are the two graphs different (look carefully at the maximum values on each graph)? The same? Add the `tic` and `toc` commands before and after the series of calls to `ode45` and `ode15s`. Which is faster?

Explain how the graphs are different, and why that might be. Also explain how the results you obtain make sense given the changing value of $c$ that you are using.

**Pair 2 Exercise 4.** Finally, consider impulses with $a = 0.5$, $a = 0.25$, $a = 0.1$ and $a = 0.01$. Solve $y'' + 21y' + y = I(t)$ for these impulses using `ode15s`; for each, use `odeset` to define options setting the maximum step size to be the width of the impulse. For example,
```matlab
>> a1 = 0.5;
>> opt1 = odeset( 'MaxStep', a1 );
>> I1 = @(t) Impulse( t, c, a1 );
>> a2 = 0.25;
>> opt2 = odeset( 'MaxStep', a2 );
>> I2 = @(t) Impulse( t, c, a2 );
```
and so on. Plot the solutions together on a single graph. What happens as \( a \) gets smaller? Does it appear that there is a limit?

Show your work for the last three exercises to pair 1 to come up with answers to the questions: How accurate is Euler’s method? How do the accuracies of and solutions generated by \texttt{ode45} and \texttt{ode15s} compare? How efficient are they (in terms of the time to generate a solution)? What happens as \( a \to 0 \) in the impulse?

**References**