LAB 4: DISCONTINUOUS FORCING, SIGNALS, AND NUMERICAL METHODS

1. Objectives and Instructions

1.1. **Model.** In this lab, we will study differential equations of the form

$$y'' + 2\gamma y' + \omega_0^2 y = F(t) \tag{1}$$

where $F(t)$ is a discontinuous forcing function. We first encounter equations of this type with continuous $F(t)$ in Chapter 4 of Brannan & Boyce, when we consider applications to mechanical systems. In this lab, we see how they are related to electrical signals and filters.

Recall that the characteristic polynomial of the associated homogeneous equation is $\lambda^2 + 2\gamma\lambda + \omega_0^2$, with roots $\lambda = -\gamma \pm \sqrt{\gamma^2 - \omega_0^2}$. If γ is a small (relative to ω_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 ϕ_0. We specifically considered forcing functions $F(t)$ of the form $F(t) = A\cos(\omega t)$. In this lab, we will instead consider forcing functions, such as the step function $u_c(t)$, which are discontinuous.

What's $u_c(t)$? It's just a function that is zero until $t = c$ and one afterwards: $u_c(t) = \begin{cases} 0, & t < c \\ 1, & t \geq c \end{cases}$. We will work with it in [BB, §§5.5–6], but for now this definition is what you need to know.

1.2. **Objectives.** Throughout this semester we have used *MATLAB*'s ode45 function to generate numerical approximations to solutions of differential equations. Our first goal in this lab is to get a sense of how numerical solvers like this work, the types of errors that may occur, and how they may be addressed. In particular, we will

- see how numerical methods such as *MATLAB*'s ode45 work, by considering a much simpler method, Euler's method, for generating an approximate solution to a differential equation;
- explore how the manner in which these numerical approximations are generated may result in instances where the approximate solution is unreliable; and
- see how Laplace transforms may allow us to deal reliably with some of these cases involving discontinuous or impulsive forcing functions in the differential equation.

2. Pre-Lab

Laplace transform methods are particularly useful when there is a discontinuous function in our differential equation. In the real-world, of course, there are relatively few instances in which we see truly discontinuous functions. One application where it is particularly reasonable to consider such functions is in

circuits with an applied voltage; in these it is possible to have very sudden changes in imposed voltages. Motivated by this, we consider a circuit model in this lab, to see how numerical methods work in this context.

2.1. **RLC Circuits.** In [BB, p.213], (1) is derived from Kirchhoff's voltage law: the forcing voltage $e(t)$ in a closed circuit is equal to the sum of the voltages across the elements in the circuit. For a circuit with a resistor with resistance R, capacitor with capacitance C, and inductor with inductance L, this leads to an equation for the current i in the circuit and charge q on the capacitor, $L\frac{di}{dt} + Ri + \frac{1}{C}q = e(t)$. We can rewrite this by noting that $i = \frac{dq}{dt}$, so that $Lq'' + Rq' + \frac{1}{C}q = e(t)$. Dividing through by L and letting $y = q$, $\gamma = R/(2L)$, $\omega_0^2 = 1/(LC)$ and $F(t) = e(t)/L$, we obtain (1).

Note that we could similarly obtain a second order equation for i: if we differentiate the equation $L\frac{di}{dt} + Ri + \frac{1}{C}q = e(t)$ and use the fact that $dq/dt = i$, we get an equation for i. Will it have the same characteristic equation?

Exercise 1: Rewrite the system $L\frac{di}{dt} + Ri + \frac{1}{C}q = e(t)$, $\frac{dq}{dt} = i$ as a matrix equation in the vector $\mathbf{x} = \begin{pmatrix} q \\ i \end{pmatrix}$. Show that the eigenvalues of the system match those given for (1) in the introduction of the model at the beginning of the lab.

If we think of a circuit such as this as a signal[1] processor, we are thinking of the input voltage $e(t)$ as the input signal, and the voltage across the capacitor, q/C, as the processed output signal. The amplitude of the output q depends on the values of R, L and C we pick, and by picking those the output may have a form that we specifically want. Conversely, if we don't know R, L and C, we may be able to determine something about them by picking an input $e(t)$ and seeing what the output q is.

2.2. **Numerical Solvers.** Throughout this semester, we have relied on *MATLAB*'s `ode45` function to plot (approximate) solutions to differential equations and systems of differential equations. To do this, `ode45` uses a method to approximate points on a solution trajectory by using known information about the preceding points and the differential equation. To illustrate this idea we will consider a much simpler (and far less accurate) numerical method, Euler's method. Euler's method is easy to visualize and implement, so we will briefly describe it to get a sense of what numerical solvers do.

2.2.1. *Euler's Method.* We illustrate Euler's method first with a specific example, and then state it more generally. Suppose that we are solving $x' = t\cos(x)$, with $x(0) = 0$, for $0 \le t \le 1$ (more generally, this is $x' = f(t, x)$, with $x(t_0) = x_0$, on $[t_0, t_f]$). We will approximate the solution to this initial value problem on the interval by taking $n = 2$ steps to get from $t_0 = 0$ to $t_f = 1$. That is, we start with $t = 0$, where we know $x(0) = 0$ and $x' = t\cos(x)$: thus, at $(0, 0)$, $x' = 0 \cdot \cos(0) = 0 \cdot 1 = 0$. We use this to predict what x will be a short distance $h = \frac{t_f - t_0}{n} = \frac{1}{2}$ from the initial condition. Euler's method uses a

Of course, for any real application, we would use far more steps than $n = 2$!

[1]For example, a radio signal sent to an electronic device will be received by an antenna, and then translated into a voltage in the circuit.

tangent line approximation, and estimates that $x(\frac{1}{2}) \approx x_1 = x_0 + hf(t_0, x_0) = 0 + \frac{1}{2}(0) = 0$. Thus, we approximate $x(\frac{1}{2})$ with $x_1 = 0$. We then repeat the process, using x_1 at the time $t_1 = \frac{1}{2}$ and the differential equation to find $x_2 = x_1 + hf(t_1, x_1) = 0 + \frac{1}{2}(\frac{1}{2}\cos(0)) = \frac{1}{4}$. Thus, Euler's method has given us an approximation for two points on the solution curve: $(t_1, x_1) = (\frac{1}{2}, 0)$ and $(t_2, x_2) = (1, \frac{1}{4})$. Formally, Euler's method is an algorithm for generating each of the approximate values x_k at the t-values t_k: it takes (for $0 \le k \le n$)

$$\begin{aligned} h &= \tfrac{t_f - t_0}{n} \\ t_{k+1} &= t_k + h \\ x(t_{k+1}) \approx x_{k+1} &= x_k + hf(t_k, x_k). \end{aligned} \tag{2}$$

Then x_{k+1} is an approximation of $x(t_{k+1})$ $(0 \le k \le n)$, and so $x(t_f) \approx x_n$.

If we're solving a system of equations we can do the same thing as in (2), but our dependent variable is a vector: we're approximating the vector $\mathbf{x}_k$ at each step.

Example 1*: Find the Euler's method approximations* $\mathbf{x}_1$ *and* $\mathbf{x}_2$ *if* $\mathbf{x}' = \begin{pmatrix} 0 & 1 \\ -1 & -2 \end{pmatrix}\mathbf{x}$*,* $\mathbf{x}(0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ *and* $h = 0.2$*.*

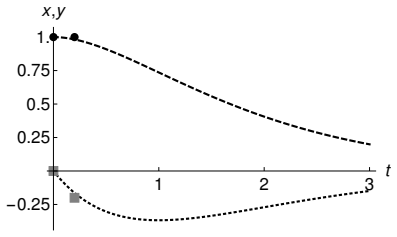
Solution: Let $\mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix}$. Then the differential equation gives $x' = f(t, x, y) = y$ and $y' = g(t, x, y) = -x - 2y$. At $t_0 = 0$ we have $x_0 = 1$, $y_0 = 0$. Then at $t_1 = t_0 + h = 0.2$, we approximate

$$x_1 = x_0 + h\, f(t_0, x_0, y_0) = 1 + 0.2(0) = 1$$

and

$$y_1 = y_0 + h\, g(t_0, x_0, y_0) = 0 + 0.2(-1) = -0.2.$$

Thus, we have so far generated the following approximation (the dashed and dotted curves are the exact solution to this problem; the first point and square are $x_0 = 1$ and $y_0 = 0$, and the second point and square our approximations x_1 and y_1).



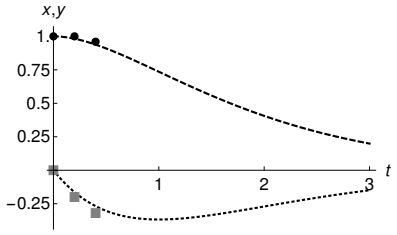
Then, at $t_2 = t_1 + h = 0.4$,

$$x_2 = x_1 + h\, f(t_1, x_1, y_1) = 1 + 0.2(-0.2) = 0.96,$$

and

$$y_2 = y_1 + h\, g(t_1, x_1, y_1) = -0.2 + 0.2(-1 + 0.4) = -0.32.$$

Thus $\mathbf{x}_1 = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} 1 \\ -0.2 \end{pmatrix}$, and $\mathbf{x}_2 = \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} 0.96 \\ -0.32 \end{pmatrix}$. Adding these values to our graph, we have the figure below.



Exercise 2: Suppose $x' = 2\sqrt{x}$. Let $t_0 = 1$ and $x_0 = 1$. The exact solution to this initial value problem is $x(t) = t^2$.

(1) Calculate x_1 with $h = 1$, 0.1, 0.01, and 0.001. How does $|x(t_1) - x_1|$ change with h?
(2) Approximate $x(2)$ using Euler's method with $h = 1/2$. Calculate $|x(1.5) - x_1|$ and $|x(2) - x_2|$.

Exercise 3: Suppose we are considering the system $x' = x - xy$, $y' = -y + xy$, $x(0) = 2$, $y(0) = 1$. Approximate x_1, y_1, x_2, and y_2 with Euler's method and a step size of $h = 0.25$.

2.2.2. *Step Size.* Fixing a uniform step size may not be the most effective way to implement a numerical method. Instead, we can modify the algorithm by taking any sequence $t_0 < t_1 < t_2 < \cdots$, where $t_1 - t_0$ does not have to be the same as $t_2 - t_1$, and so on. For Euler's method, we modify (2) to approximate $x(t_{n+1})$ by

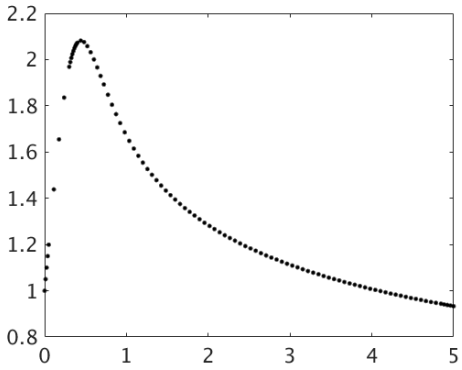
$$x_{n+1} = x_n + f(t_n, x_n)(t_{n+1} - t_n).$$

Notice that this is just relaxing our initial use of a constant step size $h = \frac{t_f - t_0}{n}$! In particular, we can make our algorithm more efficient by increasing h when the error is small, and decreasing h when the error exceeds some chosen tolerance level.

How would we know if this were the case? Ideally we would compare the approximate solution for x with the actual value—except that if we knew the actual value, we wouldn't in general be using a numerical solver. To get around this, most solvers at each step make a second calculation with a smaller step size and see if the difference between the two is small. If the difference between the two calculated values is small enough, the solver will continue with the existing (or a larger) step size, and if it is too big it decreases the step size. This is why a solver like `ode45` gives its approximation to a solution `xsol` at a set of points `tsol` that may not be evenly spaced.

Example 2: Approximate the solution to $x' = 4 - t\,x^3$, $x(0) = 1$ *using* `ode45` *to see how the step size varies.*
Solution: In *MATLAB*, we have

```
>> [ts, xs] = ode45( @(t,x) 4 - t*x^3, [0 5], 1 );
>> plot( ts, xs, '.k', 'MarkerSize', 10 );
```



References

[BB] Brannan, James R, and William E Boyce. *Differential Equations: an Introduction to Modern Methods And Applications*. Third edition. Hoboken, NJ: Wiley, 2015.