San Francisco State University

Engineering 315

Laboratory #2 – Differential Equations

Outline

1. Purpose

In this unit, you will simulate the time-domain response of linear systems to step and impulse inputs. We'll use Matlab to check that the accuracy of your theoretical solution of differential equations.

Background reading includes:

- Lathi, Chapter2
- Holton notes, Unit 2

2. Background

Exact solution of differential equations using Matlab

The general form of an N^{th}-order linear constant-coefficient differential is

$$\sum_{n=0}^{N} a_n \frac{d^n y(t)}{dt^n} = \sum_{m=0}^{M} b_m \frac{d^m x(t)}{dt^m} . \tag{L2.1}$$

The general form of the solution of this equation to an impulse, $x(t) = \delta(t)$, is the sum of exponentials

$$y(t) = \sum_{n=1}^{N} (A_n + B_n t^n) e^{s_n t} u(t) , \tag{L2.2}$$

where the A_n and B_n are possibly complex constants and s_n are the time constants, which are the roots of the characteristic equation

$$\sum_{n=0}^{N} a_n s_n = 0 . \tag{L2.3}$$

For any real-world problem we are likely to encounter, the coefficients a_n and b_n of the differential equation, Equation (L2.1), are purely real; hence, the roots of the characteristic equation can include only *real roots* and/or roots that occur in *complex conjugate* pairs. If all the real roots are distinct (i.e. non-repeating), then the B_n in Equation (L2.2) are all zero. If there are repeated roots, then at least some of both the A_n and B_n will be non-zero.

The notes present two general methods of solving differential equations of arbitrary order in response to impulses and steps – the algebraic substitution and singularity matching methods. In either of these methods, we need to "guess" the general solution of the differential equation correctly. Matlab can help.

Example 1: Real distinct roots

For example, consider the third order equation

$$\frac{d^3y(t)}{dt^3}+10\frac{d^2y(t)}{dt^2}+29\frac{dy(t)}{dt}+20y(t)=20\delta(t), \tag{L2.4}$$

which has characteristic equation

$$s^3+10s^2+29s+20=0. \tag{L2.5}$$

Equation (L2.5) has three distinct real roots, s_1, s_2, and s_3, which can be found using the Matlab `roots` function

```
» s = roots([1 10 29 20])
s =
   -5.0000
   -4.0000
   -1.0000
```

Because these roots are all distinct and real, we know the general form of the solution to an impulse is

$$y(t)=\left(A_1e^{s_1t}+A_2e^{s_2t}+A_3e^{s_3t}\right)u(t), \tag{L2.6}$$

Given an impulse input and a set of initial conditions on $y''(0^-)$, $y'(0^-)$ and $y(0^-)$, we can use the impulse matching method to generate a series of simultaneous equations that allow us to determine $y''(0^+)$, $y'(0^+)$ and $y(0^+)$, from which we can determine the unknown constants A_1, A_2, and A_3 in Equation (L2.6) . The impulse matching method says that if there is an impulse of area 20 on the right-hand side of the equation, the highest order derivative on the left-hand side, y''', must contain the impulse, hence y'' contains a step discontinuity of height 20 at *t=0* and y' and y are both continuous at *t=0*. In the notes, we schematize this as follows:

$$\overbrace{y'''(t)}^{\uparrow 20}+10\overbrace{y''(t)}^{20|}+29\overbrace{y'(t)}^{20/}+20\overbrace{y(t)}^{20)}=20\overbrace{x(t)}^{\uparrow 20}$$

Hence,

$$\begin{aligned} y(0^+) &= y(0^-) \\ y'(0^+) &= y'(0^-) \\ y''(0^+) &= y''(0^-)+20 \end{aligned} \tag{L2.7}$$

If we assume zero initial conditions, we immediately have values of $y''(0^-)$, $y'(0^-)$ and $y(0^-)$:

$$y(0^-)=y'(0^-)=y''(0^-)=0 \tag{L2.8}$$

In addition, we can evaluate Equation (L2.6) at $t=0^+$ to get values of $y''(0^+)$, $y'(0^+)$ and $y(0^+)$

$$\begin{aligned} y(0^+) &= A_1 + A_2 + A_3 \\ y'(0^+) &= A_1 s_1 + A_2 s_2 + A_3 s_3 \\ y''(0^+) &= A_1 s_1^2 + A_2 s_2^2 + A_3 s_3^2 \end{aligned} \tag{L2.9}$$

which we combine with Equations (L2.7)-(L2.9) to get a set of simultaneous linear equations

$$\begin{aligned} A_1 + A_2 + A_3 &= 0 \\ A_1 s_1 + A_2 s_2 + A_3 s_3 &= 0 \\ A_1 s_1^2 + A_2 s_2^2 + A_3 s_3^2 &= 20 \end{aligned} \tag{L2.10}$$

Since we have obtained s_1, s_2, and s_3 from the solution of Equation (L2.5) , we can use Cramer's rule, Gaussian elimination or a number of other standard techniques to solve Equation (L2.9) for the unknown constants A_1, A_2, and A_3. For example, expressing the equations in matrix form

$$\mathbf{Ma} = \mathbf{b}\,, \tag{L2.11}$$

where

$$\mathbf{M} = \begin{bmatrix} 1 & 1 & 1 \\ s_1 & s_2 & s_3 \\ s_1^2 & s_2^2 & s_3^2 \end{bmatrix}, \quad \mathbf{a} = \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ 20 \end{bmatrix}. \tag{L2.12}$$

Taking the inverse of Equation (L2.11) yields

$$\mathbf{a} = \mathbf{M}^{-1}\mathbf{b}\,, \tag{L2.13}$$

which gives us A_1, A_2, and A_3. Matlab can do this easily:

```
» M = [s.^0 s s.^2].';
» b = [0 0 20].';
» a = M \ b
a =
    5.0000
   -6.6667
    1.6667
```

The complete solution of Equation (L2.6) is therefore

$$y(t) = \tfrac{1}{3}\left(15e^{-5t} - 20e^{-4t} + 5e^{-t}\right)u(t)\,, \tag{L2.14}$$

which we plot with Matlab like this

```
» t = 0:0.05:4;
» plot(t, real(exp((s*t).') * a));
```

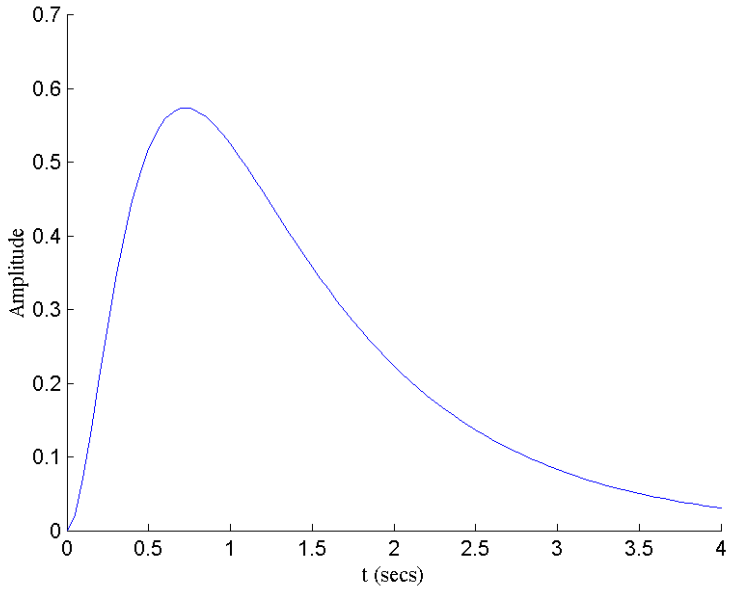


Figure 1

Plotting the real part isn't strictly necessary in this case, but it is useful in cases where the evaluation of the solution for $y(t)$ may have a trace imaginary part due to computational errors, such as the next example.

The same solution procedure applies to calculating the step or impulse response of a differential equation of any order, whether the roots are real or complex as long as the roots are distinct (non-repeating) and the right-hand side of the equation is only of the form $Kx(t)$, where K is a constant. These are the only type of differential equation we will consider in this lab.

Example 2: Complex roots

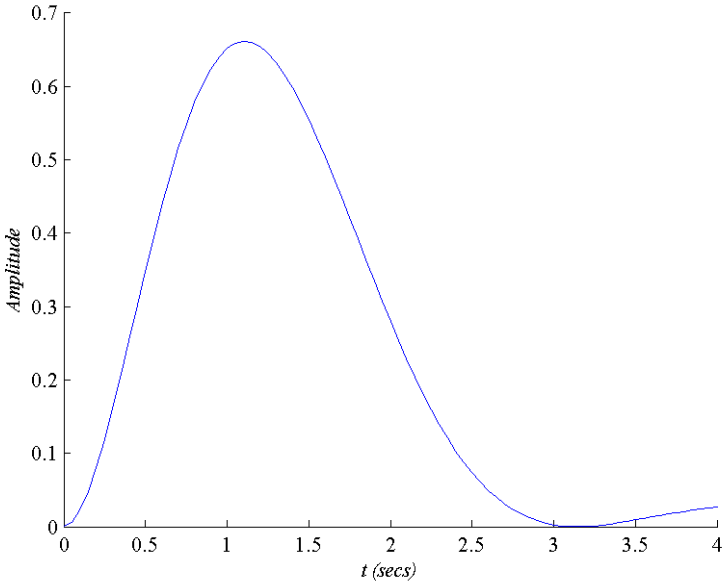
The differential equation

$$\frac{d^3y(t)}{dt^3}+3\frac{d^2y(t)}{dt^2}+7\frac{dy(t)}{dt}+5y(t)=5\delta(t) \tag{L2.15}$$

has a solution which is the same general form as Equation (L2.6), but two of the time constants derived from the solution of the characteristic equation are complex conjugates: $s_1=-1+2j$, $s_2=-1-2j$ and $s_3=-1$. Given zero initial conditions, we can solve Equation (L2.13) with the appropriate values for **M** and **b** to give coefficients $A_1=-5/8$, $A_2=-5/8$ and $A_3=5/4$. Therefore the complete solution is

$$\begin{aligned} y(t) &= \left(A_1e^{s_1t}+A_2e^{s_2t}+A_3e^{s_3t}\right)u(t) \\ &= \tfrac{1}{8}\left(-5e^{(-1+2j)t}-5e^{(-1-2j)t}+10e^{-t}\right)u(t) \\ &= \tfrac{1}{8}e^{-t}\left(10-5(e^{2jt}+e^{-2jt})\right)u(t) \\ &= \tfrac{5}{4}e^{-t}\left(1-\cos 2t\right)u(t) \end{aligned} \tag{L2.16}$$

Note that Matlab doesn't care whether the time constants, s_1 and s_2, are real or complex. The equations for **M** and **b** are the same as those presented in Example 1, and the eventual result is a real expression for *y(t)*, which contains a oscillatory term, cos*t*. Here's the plot produced from code identical to that we used in the previous example:



Example 3: Repeated roots

The differential equation

$$\frac{d^3y(t)}{dt^3}+4\frac{d^2y(t)}{dt^2}+5\frac{dy(t)}{dt}+2y(t)=2\delta(t) \tag{L2.17}$$

has three real roots, of which two repeat: $s_1=-1$, $s_2=-1$ and $s_3=-2$. If there are repeating roots, the general form of the solution is not the same as in the previous two examples. It will include terms of the form $B_n t^n e^{s_n t}u(t)$ as well as terms of the form $A_n e^{s_n t}u(t)$. In this case,

$$y(t)=(A_1e^{-t}+B_1te^{-t}+A_2e^{-2t})u(t) \tag{L2.18}$$

Given zero initial conditions, you should be able to show that the matrices **M**, **b** and **a** in Equation (L2.12) are

$$\mathbf{M}=\begin{bmatrix}1 & 0 & 1\\ -1 & 1 & -2\\ 1 & -2 & 4\end{bmatrix},\quad \mathbf{a}=\begin{bmatrix}A_1\\ B_1\\ A_2\end{bmatrix},\quad \mathbf{b}=\begin{bmatrix}0\\ 0\\ 2\end{bmatrix}. \tag{L2.19}$$

which yields

$$y(t)=\left(-2e^{-t}+2te^{-t}+2e^{-2t}\right)u(t). \tag{L2.20}$$

We're not going to worry about the case of repeated roots in this lab.

Exact solution of step response of differential equations using Matlab

We could repeat the preceding treatment to find the solution of the differential equation when the input to the system is a step, $x(t)=u(t)$, but there is a very simple way of computing the step response that leverages the method we've just discussed of finding the impulse response. Consider again the differential equation (L2.4), but now with a step input instead of an impulse on the right-hand side:

$$\frac{d^3y(t)}{dt^3}+10\frac{d^2y(t)}{dt^2}+29\frac{dy(t)}{dt}+20y(t)=20u(t)\,. \tag{L2.21}$$

Taking the derivative of both sides, we get

$$\frac{d^4y(t)}{dt^4}+10\frac{d^3y(t)}{dt^3}+29\frac{d^2y(t)}{dt^2}+20\frac{dy(t)}{dt}=20\delta(t) \tag{L2.22}$$

So, to find the step response of Equation (L2.21), we just have to compute the impulse response of Equation (L2.22). This has characteristic equation

$$s^4+10s^3+29s^2+20s+0=0\,. \tag{L2.23}$$

Note that there is no term for $y(t)$.

Numerical solution of differential equations using Matlab

Given a system described by an N^{th}-order linear constant-coefficient differential equation of Equation (L2.1), we can equivalently completely describe the system by specifying the coefficients a_n, b_m, N and M. For example, given the system described by Equation (L2.4), we have N =3, M =1, a_3 =1, a_2 =10, a_1 =29, a_0 =20 and b_0 =20.

The Control Systems Toolbox in Matlab offers several ways of simulating the response of continuous-time systems described by differential equations by numerical simulation methods. `step` simulates the response to a step, `impulse` simulates the response to an impulse and `lsim` simulates the zero-state response of a system to an arbitrary input. `initial` shows the zero-input response of a system. To use any of these Matlab functions, we do the following:

1) Describe the differential equation in terms of two arrays `a` and `b` that correspond to the coefficients a_n and b_m.. For example, in the preceding system

```
a = [1 10 29 20];
b = 20;
```

2) Create a Matlab 'transfer function' object, `h`, using the `tf` command with the two arrays `a` and `b` as arguments.

```
h = tf(b, a);
```

Note the order of the arrays; `b` comes first. At this point, don't worry about what a transfer function is. We'll deal with this in detail in a later unit.

3) Plot the impulse response

```
impulse(h);
```

Of course, you can do the whole thing in one step

```
impulse(tf(20, [1 10 29 20]));
```

and in fact, you can shorten the whole thing further (though this doesn't appear to be in the documentation of the function) as follows:

```
impulse(20, [1 10 29 20]);
```

Here's the resultant plot given the parameters above:

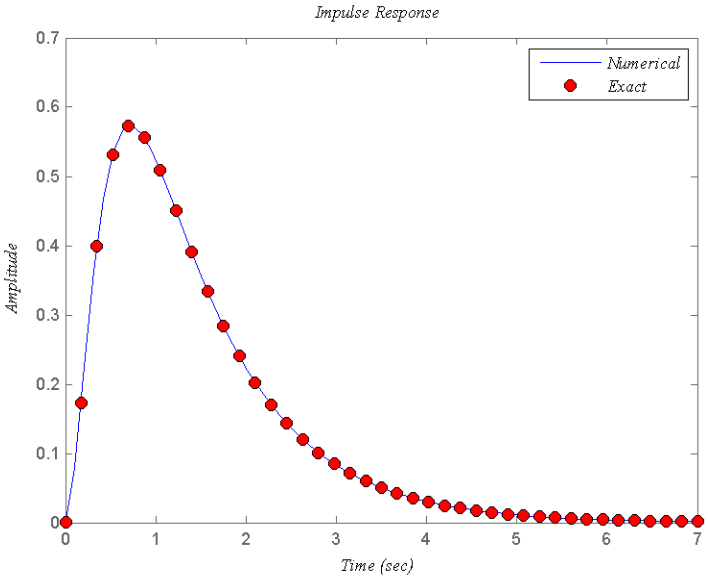


Figure 2: Numerical solution of differential equation

In order to generate the numerical approximation, Matlab uses numerical methods to approximate the continuous-time derivative. Unless told to do otherwise, it automatically does the best it can to select the appropriate quantization interval and the maximum time extent of the plot, in this case seven secs. You can override the choice of quantization interval or the maximum time extent using optional arguments to the `step` function. For example, to increase the time limit to 10 seconds, you'd call

```
step(h, 10);
```

In Figure 2, I've superimposed the exact solution, which we got from our matrix solution, plotted with red dots. It is identical to the numerical solution.

3. Assignment

In this lab you will create a function, `impuls`, that calculates the exact solution of differential equations to an impulse.

```
function y = impuls(b, a, t)
%  IMPULS   Impulse response of system
%
%  y = impul(b, a, t)
```

The function accepts the *a* and *b* arrays, as well as an array of time points, *t*. The function then calculates, using the methods described above, the output array that corresponds to the exact solution, $y(t)$. Your `impuls` function should produce no output to the command window.

In addition to the impulse function, you will create two other 'helper' functions just to plot the impulse response and step response and compare them to Matlab's numerical solutions.

```
function y = plot_impulse(b, a)
%  PLOT_IMPULSE   Plot the impulse response of system
%
%  y = plot_impulse(b, a)
```

This function will first call Matlab's `impulse` function with parameters `b` and `a` and obtain a plot of the numeric solution. You will use Matlab's `xlim` function to determine the timescale that Matlab chose for the plot of the numerical

solution. Inside this function, you will create an array of 41 equally spaced time points, *t*, that span the time interval $x_{\min} \le t \le x_{\max}$. Then you will call your `impuls` function to obtain the exact solution and superimpose this plot on top of the numerical solution, as shown in Figure 2.

You will also create a small function to plot the step response and compare it to Matlab's numerical solution.

```
function y = plot_step(b, a)
%  PLOT_STEP   Plot the step response of system
%
%  y = plot_step(b, a)
```

This function will first call Matlab's `step` function with parameters `b` and `a` and obtain a plot of the numeric solution. You will use Matlab's `xlim` function as before to determine the timescale that Matlab chose for the plot of the numerical solution and create an array of 41 equally spaced time points, *t*, that span that time interval. If you are clever, you don't actually have to write a lot of code. Your `plot_step` function will just call your `impuls` function with appropriate values to obtain the exact solution of the step response and superimpose this plot on top of the numerical solution. I suggest that all you have to do to plot the step response is to call your `impuls` function with a slightly different value of the parameter `a`. For example, look at the difference between the values of `a` corresponding to Equations (L2.21) and (L2.22).

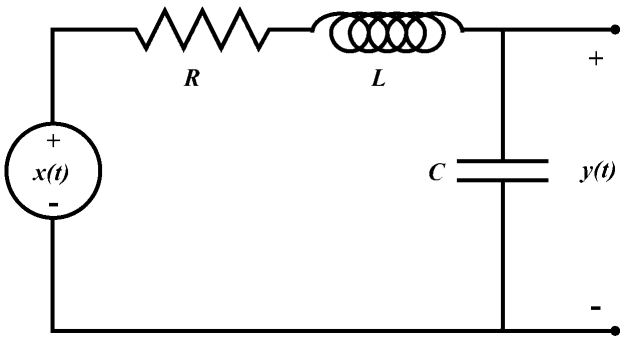
1. In order to test your function, please provide plots of the impulse response and step response for the following differential equations.

a) $y' + 5y = 5x(t)$
b) $y'' + 4y' + 3y = 6x(t)$
c) $y'' + 2y' + 26y = 13x(t)$
d) $y'' + y = x(t)$
e) $y''' + 10y'' + 29y' + 20y = 20x(t)$
f) $y''' + 3y'' + 7y' + 5y = 5x(t)$
g) $y''' + 6y'' + 11y' + 6y = 3x(t)$
h) $y''' + 4y'' + 21y' + 34y = 34x(t)$

I've provided a little function, `lab2_1.m`, that you can download that will plot all the functions in this part. Put it in the same directory as your two functions. You can use Matlab's `publish` command, as we did in the last lab. Note also, that you can publish to a .doc file as well as to .html. Just

```
publish('lab2_1', 'doc')
```

2. Second order systems are among the most important ones we will deal with, both in electrical and mechanical systems. Here is an example of a second order electrical system:



The relation between the source voltage, $v_s(t)$, and the output voltage, $v_c(t)$ is given by the second order equation

$$y''(t) + \tfrac{R}{L} y'(t) + \tfrac{1}{LC} y(t) = \tfrac{1}{LC} x(t). \qquad \text{(L2.24)}$$

Performing a change of variables

$$\omega_0^{\,2} = \frac{1}{LC} \Rightarrow \omega_0 = \frac{1}{\sqrt{LC}}$$
$$2\zeta\omega_0 = \frac{R}{L} \Rightarrow \zeta = \frac{R}{2}\sqrt{\frac{C}{L}},$$

the differential equation becomes

$$y''(t) + 2\zeta\omega_0 y'(t) + \omega_0^{\,2} y(t) = \omega_0^{\,2} x(t).$$

ω_0 is termed the *natural frequency* of the response, which depends on both *L* and *C*. ζ is termed the *damping factor*. For a fixed choice of *L* and *C*, the damping factor depends only on *R*, and varies over the range $0 \le \zeta < \infty$. There are four distinct solutions to this problem, depending on the value of ζ.

- If $\zeta > 1$, the system is said to be *overdamped*.
- If $\zeta < 1$, the system is *underdamped.*
- If $\zeta = 1$, the system is *critically damped*.
- If $\zeta = 0$, the system is *undamped*.

We will investigate the impulse response of this system to different values of ζ and a maximum time of 5 seconds.

For the remainder of the lab, you will write your own function to plot stuff. If you wish, you can download and modify my `lab2_2` function and `publish` it.

Let $\omega_0 = 10$. Plot on the same axis the impulse response for values of ζ=0.1, 0.2, 0.5, 0.8, 1, 2, 5, 8 and 10. Note that as ζ decreases below 1.0, the response becomes more and more oscillatory. Even with ζ=0.8, you can see that the response dips below the line *y*=0. As ζ increases above 1, the response becomes "slower"; that is, dominated by its longest time constant.

Finally, we want to look at the step response of the second order system as we vary ζ, again using the exact solution.

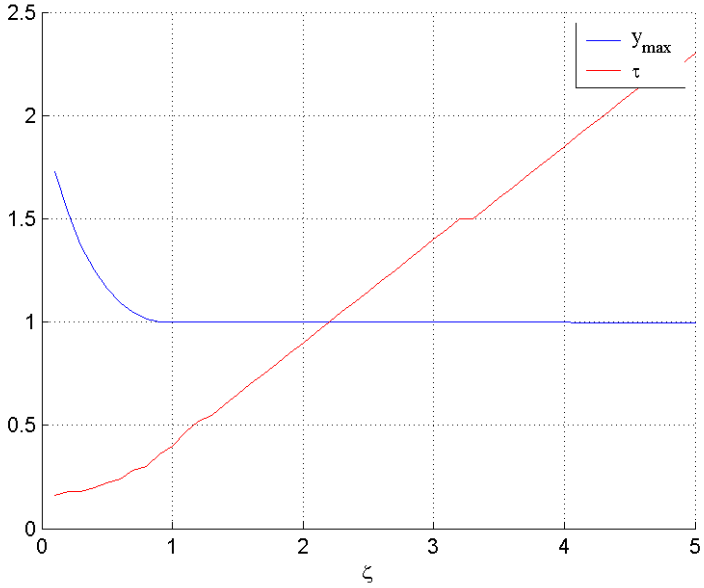
a) Repeat Problem 2, except plot the step response (using `step`) on the same axis for values of ζ=0.1, 0.2, 0.5, 0.8, 1, 2, 5, 8 and 10. Again, let $\omega_0 = 10$. Note that as ζ decreases below 1.0, the response becomes more and more oscillatory and exceeds the value of 1. As ζ increases above 1, the response becomes "slower" and never exceeds 1.

b) We can define the *rise-time* of a system as the time it takes the system to reach 90% of its final value. In this part, we want to plot the rise-time and the maximum output value of the system as a function of ζ. Write a script or a function, `plottc`, that will do this. Here's what you need to do:

- Let ζ range from 0 to 5 in increments of 0.1. Fix $\omega_0 = 10$.
- For each value of ζ, compute the arrays `a` and `b`, and use these to compute the step response over a 5 second interval using this syntax.

```
[y, t] =step(tf(b, a), 5);
```

  You must use `tf` or it won't work. This variation of the step command produces arrays for the output response, *y*, and the time vector, *t*, that corresponds to these values of *y*. If you were to plot *y* vs. *t*, you would see the step response for the given value of ζ.
- For each value of ζ, find the rise-time, τ, which we will define as the time that it takes for the response to rise to 90% of its theoretical steady-state value (which is 1). You can use the Matlab `find` command to find the largest value of y that doesn't exceed 90% of the steady-state value.
- For each value of ζ, find the maximum value, y_{max}. You can use the Matlab `max` command to help you here.
- Finally, plot both y_{max} and τ against ζ on the same graph. You should see something like this:



c) You'll want to note a few things from the plot.

1) The time constant increases monotonically as you increase the damping, ζ;
2) The maximum value decreases to the steady state value as ζ increases;
3) The critically damped system is special. When $\zeta = 1$, the rise-time is as low as it can be while the maximum value of the response does not exceed 1.

That's all folks!

- As always, you should upload copies of your Matlab code and printouts of your plots
- Your code should be neat, easy to read and commented where appropriate.
- You are encouraged to help each other, but YOU ARE EACH RESPONSIBLE FOR DESIGNING AND CREATING YOUR OWN WORK.
