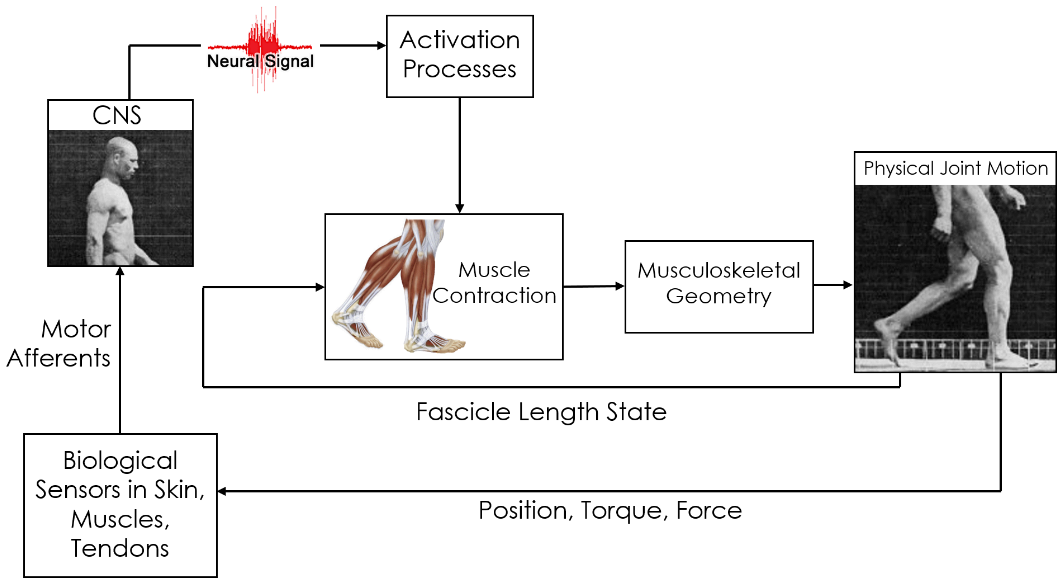
MATLAB Problem 2

A quick note before you start – you will answer all questions for this problem within this Word document. When finished, please save the document as a .pdf, which you will upload directly to Gradescope.

The field of neuromusculoskeletal biomechanics uses mathematical models to understand how neural commands are converted into movement and locomotion. These models allow us to predict, with impressive fidelity, muscle forces, joint moments, and joint angles from neural excitations and external loads. There are several steps to these models, as laid out in the following diagram:



In this problem, we're going to focus on the box called "Activation Processes", which uses differential equations to describe how neural excitation in the form of action potentials translates to muscle activation and force production. If you're interested in the underlying physiology or the full model, check out the manuscript PDF included with the problem set on CCLE. I'm admittedly biased, but my mind was blown open when I first read about these models in their entirety.

2a: Build a mathematical model and analytically solve the resulting differential equation.

Activation dynamics describe the chemical processes of calcium release and uptake within muscle tissue, as well as the physiological formation of actin-myosin cross bridges that are responsible for muscle force production. These processes can be described by the following differential equation:

$$\frac{da}{dt} = \frac{u(t)-a(t)}{t_{act}(0.5+1.5a(t))}$$

Where $a(t)$ is muscle activation, $u(t)$ is neural excitation, and t_{act} is the activation time constant.

Let's consider an admittedly contrived scenario where neural excitation is some function of muscle activation – the physiological implications of this would be a reflex-type scenario, where a currently-activated muscle is given more neural excitation in a positive feedback loop, with some sinusoidal modulator on top (don't worry if that sounds like jargon, I'm just defending the formulation of the problem for anyone who knows muscle physiology). We can model this scenario as:

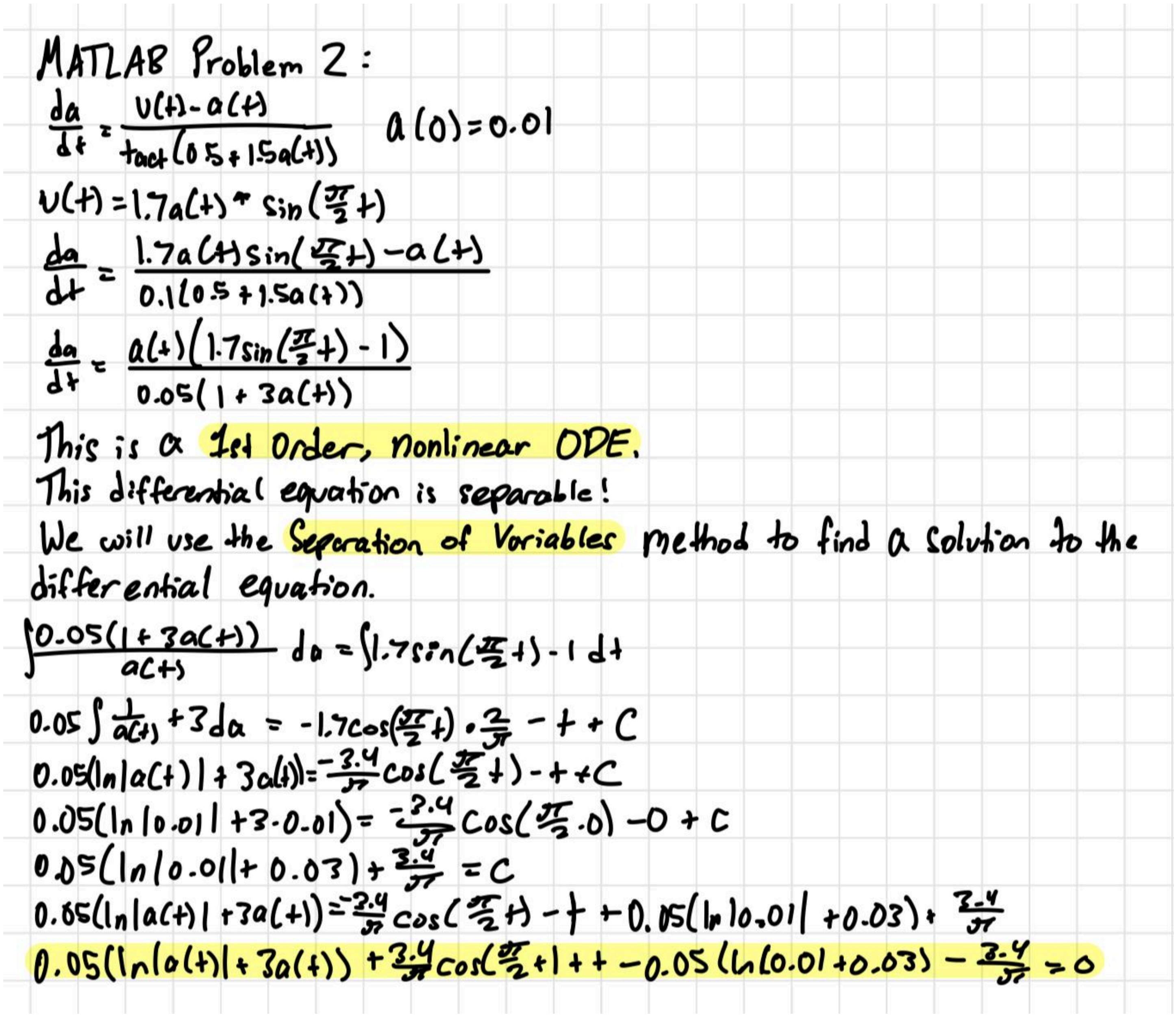
$$u(t) = 1.7a(t) * sin(\tfrac{\pi}{2}t)$$

We will also define $t_{act} = 0.1\ s$ and set the initial conditions $a(0) = 0.01$. Important note: don't worry about units here - both activation and excitation are normalized to be unitless.

Your first task is to **classify this differential equation, identify an analytical solution method, and find a specific solution analytically. You can do this by hand, then take a picture and insert it into this**

document. Show your classification and your work, and make sure that your work is readable. Important note – if you've done this correctly, you will find that your specific solution is better left in implicit form. Please do not try to solve this algebraically!

Tip: It's really important that you get the correct value here for your constant of integration. It is probably worth checking with a friend or one of the TAs to avoid a calculator mistake that would make the rest of the problem very difficult.



MATLAB Problem 2:

$$\frac{da}{dt} = \frac{v(t) - a(t)}{tact(0.5 + 1.5a(t))} \qquad a(0) = 0.01$$

$$v(t) = 1.7a(t) * \sin(\tfrac{\pi}{2}t)$$

$$\frac{da}{dt} = \frac{1.7a(t)\sin(\frac{\pi}{2}t) - a(t)}{0.1(0.5 + 1.5a(t))}$$

$$\frac{da}{dt} = \frac{a(t)(1.7\sin(\frac{\pi}{2}t) - 1)}{0.05(1 + 3a(t))}$$

This is a 1st Order, nonlinear ODE.
This differential equation is separable!
We will use the Separation of Variables method to find a solution to the differential equation.

$$\int \frac{0.05(1 + 3a(t))}{a(t)}\, da = \int 1.7\sin(\tfrac{\pi}{2}t) - 1\, dt$$

$$0.05 \int \frac{1}{a(t)} + 3\, da = -1.7\cos(\tfrac{\pi}{2}t) \cdot \frac{2}{\pi} - t + C$$

$$0.05(\ln|a(t)| + 3a(t)) = \frac{-3.4}{\pi}\cos(\tfrac{\pi}{2}t) - t + C$$

$$0.05(\ln|0.01| + 3 \cdot 0.01) = \frac{-3.4}{\pi}\cos(\tfrac{\pi}{2} \cdot 0) - 0 + C$$

$$0.05(\ln|0.01| + 0.03) + \frac{3.4}{\pi} = C$$

$$0.05(\ln|a(t)| + 3a(t)) = \frac{-3.4}{\pi}\cos(\tfrac{\pi}{2}t) - t + 0.05(\ln|0.01| + 0.03) + \frac{3.4}{\pi}$$

$$0.05(\ln|a(t)| + 3a(t)) + \frac{3.4}{\pi}\cos(\tfrac{\pi}{2}t) + t - 0.05(\ln(0.01 + 0.03) - \frac{3.4}{\pi} = 0$$

2b: Use numerical methods to solve your implicit equation for $a(t)$.
Download Homework2_MatlabProbs.m, muscleActODEfun.m, and eulerSolver.m. Without changing any of the file names for these three files, put all three files into a single folder. Navigate to that folder in Matlab's "current folder" pane.

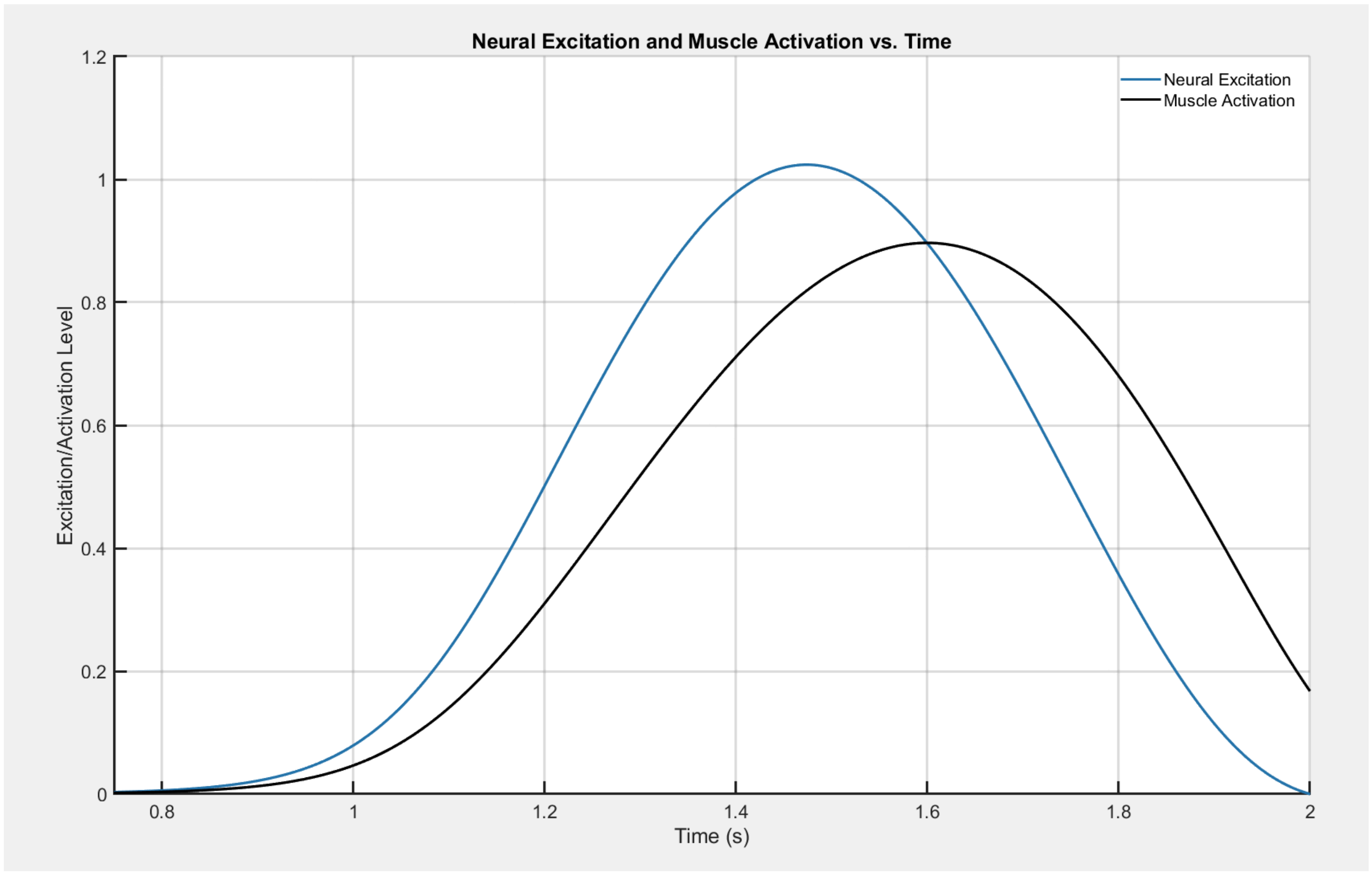
Open all three files in MATLAB. In problem 2b, you're going to use this code to solve the implicit equation you found in 2a for $a(t)$. Throughout the code, you'll see double asterisks ("**") – this indicates places where you need to fill in missing code snippets. **Note:** if you run this right away, you will get errors, because the code is not complete. That's where you come in!

The first step in this process is to prepare our implicit equation for entry into MATLAB. To do this, rearrange your implicit solution in part 2a to take the form:

$$F_{imp}(t, a(t)) = 0$$

This will allow us to use a built-in function in MATLAB called fzero() to find solutions to this equation at different values of t. To make this happen, follow along in the code for problem 2b.

Save the resultant figure as a .png of .jpg (don't just screenshot it), and insert the image into this document.



What do the two curves in this plot represent? How do they relate to each other?

The UCLA blue curve represents the neural excitation vs. time and the black curve represents the muscle activation vs. time. The muscle activation curve is a function of the neural excitation, so the muscle activation depends on the neural excitation. When neural excitation is greater than muscle activation, the muscle activation will increase. When neural excitation is less than muscle activation, the muscle activation will decrease.

In this problem, we used analytical tools to solve the differential equation, and then numerical tools for just the algebra. **Compared to a fully numerical approach like Euler's method, how accurate do you think this mixed approach is? Explain your answer.**

I believe that the mixed approach is more accurate than a fully numerical approach like Euler's method, but less accurate than a fully analytical approach. With a mixed approach, approximations are still made, meaning the curve will not be completely accurate, but close enough to make good estimations.

2c. Use Euler's method to solve your differential equation.

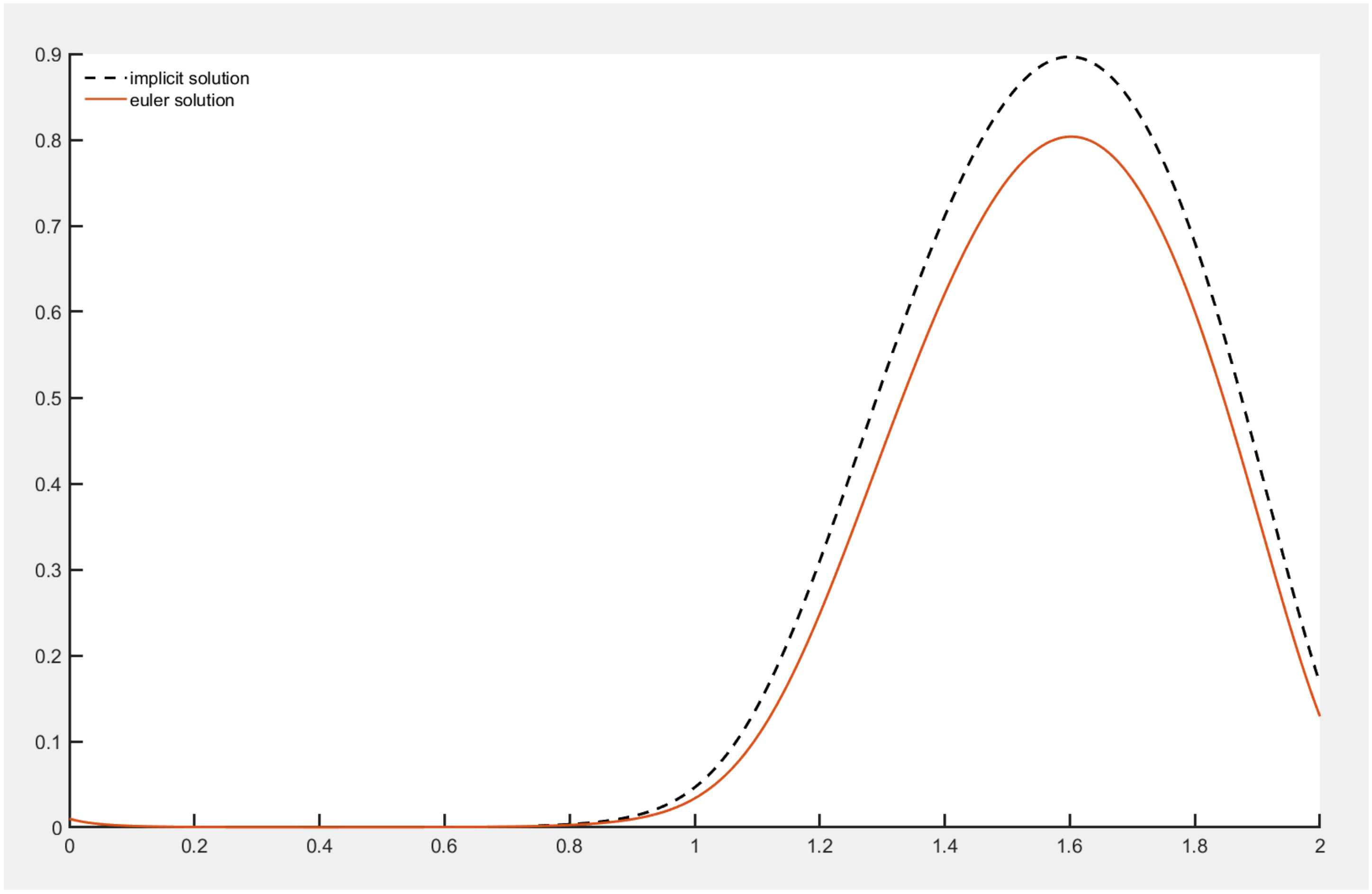
We are now going to build our first numerical solver (WOOHOO!) and use it to solve this differential equation numerically. Follow along in the code for problem 2c as we set this up.

First, fill in values for the initial conditions, time range of interest, and step size. For this first pass, we will use the same initial conditions as above, with a time range of 0 to 2s, and a step size of 0.005 seconds.

Next, go to muscleActODEfun.m and code up your differential equation. You should all be experts at this by now, from Pset 1.

When that's done, we need to build our solver. Open eulerSolver.m and fill in the missing code.

Now back to Homework2_MatlabProbs.m. Split your code after figure 2 finishes plotting (around line 86). Run the top half, and drop your figure .png here:



How does our Euler solution compare to our implicit solution? Does it look the same?

Our Euler estimation seems to be an underestimate of the implicit solution. Both graphs follow the same shape, but the Euler solution graph is lower than the implicit solution graph. At the peak of both graphs, the implicit solution is greater than the Euler solution by about 0.1.

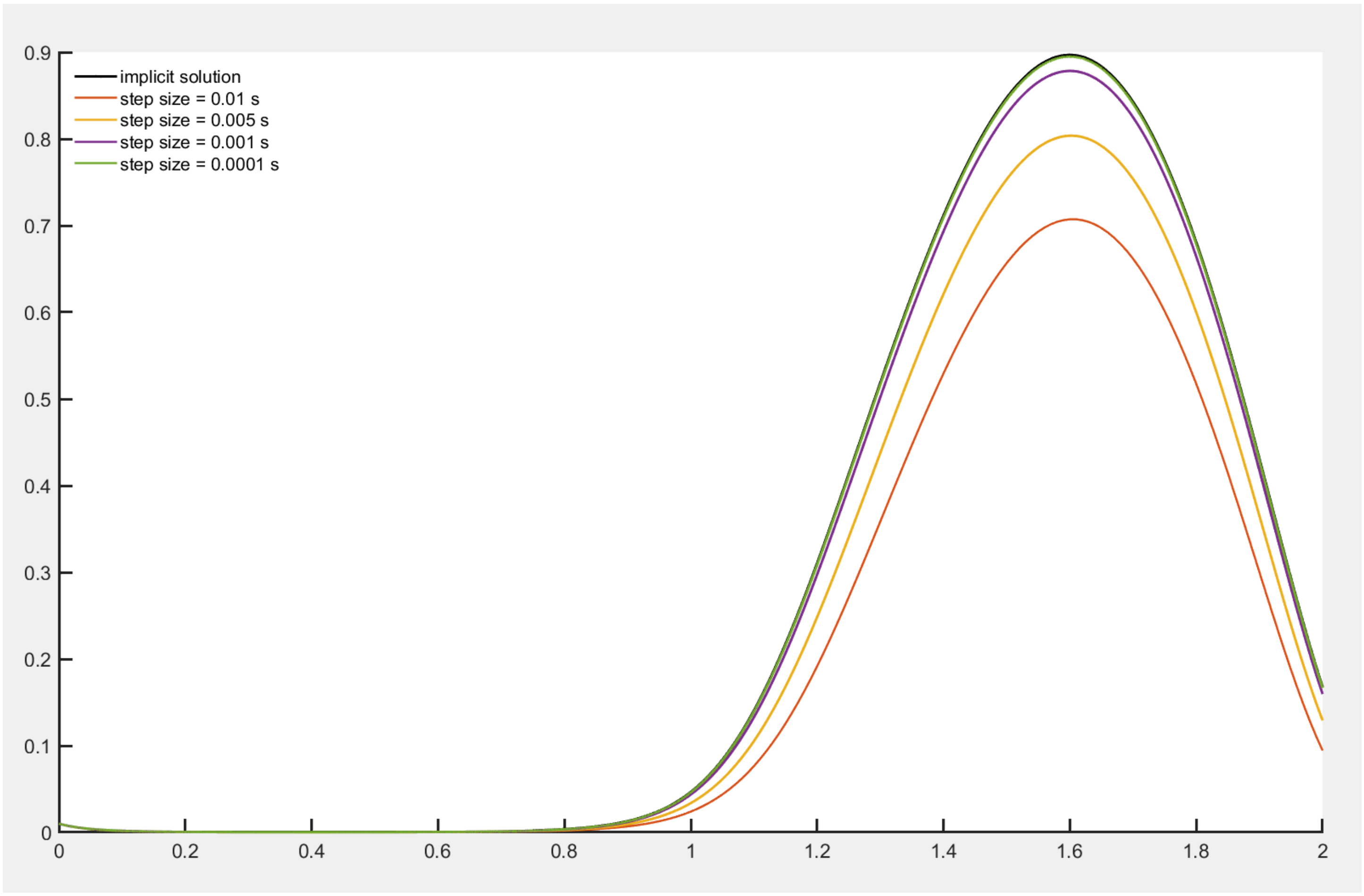
If they're different, which one gives you more confidence?

The implicit solution gives me more confidence, because we solved it using a mixed method between analytical and numerical. The Euler method is solely numerical. Numerical solutions tend to be less accurate than analytical solutions.

What are some reasons we might be seeing this outcome?

The Euler solution is less than the implicit solution because the Euler solution was solved using numerical methods, which are less accurate than analytical methods because numerical methods use approximations.

Let's play with the step size, and see if that helps us. Fill in the rest of the code in section 2c, and drop the resulting figure here:



What happens to our Euler solution as the step size decreases?

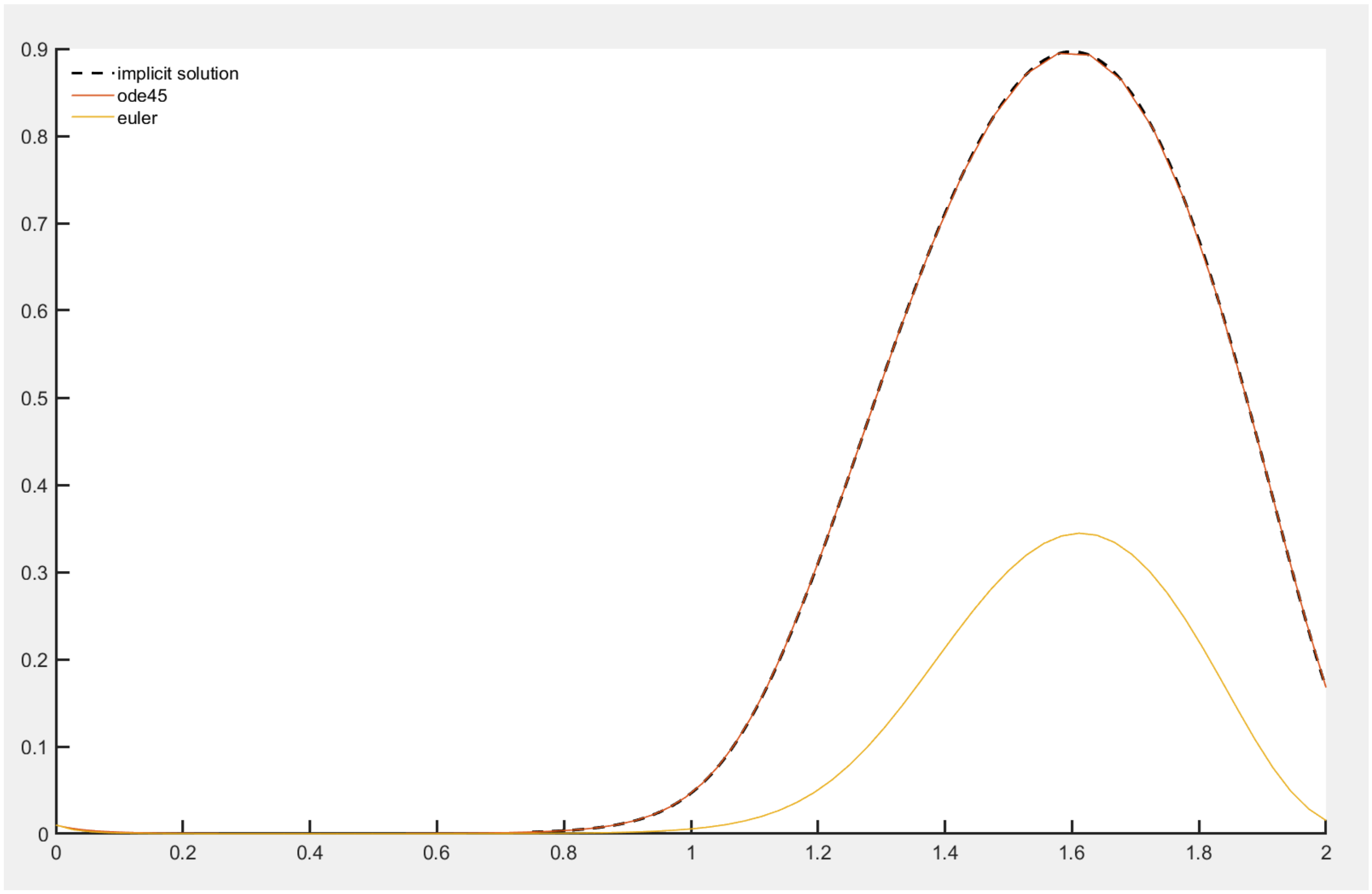
As the step size decreases, the Euler solution graph gets closer to the implicit solution graph. This is because a smaller step size reduces error.

What are some tradeoffs associated with decreasing step size? Why can't we take infinitely small steps?

We cannot take infinitely small steps because it would require a lot of computer processing time and power, and there are limitations with the computers we currently have today. Using larger step sizes, however, can result in a less accurate Euler solution compared to the implicit solution.

2d. Compare to ODE 45

Now that you're all experts at working with ODE45, let's check to see how our Euler solver stacks up. Head to problem 2d in the code, fill it in, and run it. Put your resulting figure here:



How does ODE45's solution compare to your implicit result?

The ODE45 solution follows the implicit solution super closely compared to the Euler solution.

What is the average step size for ODE45? How does this compare to the step sizes we tried in 2c?

The average step size for ODE45 is 0.027777777777778 which is larger than the biggest step size we tried in 2c.

When we run the Euler with the average step from ODE45, what do we see? What does this tell us about ODE45?

It makes sense that the Euler solution here dips lower than the Euler solution with the 0.01 step size. The larger the step size, the less accurate the graph will be. But since the ODE45 solution doesn't dip as low as the Euler solution, ODE45 does something different. ODE45 uses varying step sizes on different parts of the graph.

Look at MATLAB's documentation for ODE45. **What are some ways other than step size that ODE45 might achieve the performance we see in Figure 4?**

ODE45 is based on the Runge-Kutta (4,5) formula, the Dormand-Prince pair, which is a solver that computes y(t_n) in a single step, only needing the solution at the immediately preceding time point, y(t_n-1). This is the reason it achieves high performance in Figure 4.