CSPB 4830: Special Topics - Coding Calculus

Description

Python based course exploring the original numerical motivations for calculus though code. In this course we coded and visualized: differential equations, infinite sequences, derivatives, integrals and model real world dynamical systems like pandemics, predator prey models and even wine making.

Course text

Calculus in Context, Callahan, ISBN 0- 71672-630-0

Technology used

Python, NumPy, Matplotlib, Desmos

Skills acquired

- Numerical analysis
- Mathematical modeling
- Data visualization
- Differential equation modeling
- Calculus fundamentals

Selected Work

The work I have selected to display here is broken out into three different categories based on the arc of the course. Numerical analysis and computational modeling were layered in throughout this course and provided a visual and conceptual framework upon which the fundamental components of calculus (the derivative and integral) were built.

The most exemplary work is contained in the diffeq.ipynb, in which I model various differential equations and utilize the methods learned in this course to build my own meat defrosting model (the results of which I use regularly). I would start here then explore the rest of the content as you see fit.

Modeling

Explored population growth, pred-prey systems, and pandemic models using Python.

Derivatives

Visualized and coded numerical and symbolic methods for calculating derivatives, connecting conceptual theory to application.

Integrals

Focused on numerical integration techniques with special focus on Riemann sums.

Discussion Contributions

Charlie Bailey 3 months ago
Ok this may be a dumb question, but are we supposed to solve this question using the constant value for S (-470) at each time step, or use the S' equation from page 10 to factor in the dynamic rate change?

My assumption is we use the constant -470 and simply continue filling out the table on page 5, but just wanted to check.
helpful! 0

Tammy Metz 3 months ago
For number 7 I assumed always using S = 20000 – 470t as the starting point (the 470 is S', not S, but I think that's what you meant?).
undo helpful 1

Actions

Charlie Bailey 3 months ago
Ok great sorry yeah I meant $S' = -470$ and used that same equation. Thanks for your help!
helpful! 0

Charlie Bailey 3 months ago
During office hours one of the key clarifiers for me was the difference between **t** (the total number of days we are looking at for the illness) vs. the **timestep**—which is the number of times we update our values within that timeframe. I.e., if t = 30 and numsteps = 100, this means that we will essentially divide the 30 day time period into 30/100 "segments" (delta t), and increment t, S, I, and R for each segment.

I think this distinction becomes clearer in this chapter, but I had been confused between the two up until now.

~ An instructor (Elisabeth stade) thinks this is a good comment ~

helpful! 4

Reply to this followup discussion

Resolved Unresolved @14_f5

Actions

Charlie Bailey 3 months ago
One question I have is how everyone is interpreting the second question regarding SIRPLOT:

"How does this relate to the length of time we want to look at?"

Following my post above, does "length of time" refer to the length of the time step or the length of the time window for the illness?
helpful! 0

Ross Hastings 3 months ago
Think of the time steps as the resolution of the plot. The number of steps will govern the number of 'snippets' of the SIR model that you will see as your loop runs. The code provided uses the number of steps as a range for iteration. Increasing the number will increase the smoothness of the plot but will take longer to run (more steps, more iterations). The time window t is the number of days that will be shown of the SIR model.

The time frame of the SIR model (t, t initial, and t final) are basically the window that you want to look at for the infection. If you were to put 90 days in for tfinal (assuming 0 for tinitial), with a recovery period of 10 days, the graph would have the infection right at the start, and then all plots either approaching 100 or 0 based on what they're tracking (S, I, R) for the remainder of the graph. Adjusting tinitial and tfinal allows us to take a look at specific periods of the infection.
helpful! 0

Lukas Talaga 3 months ago
I was a little confused by this question as well. I didn't feel like I understood until I went in and changed my length of time variable in an edge case plot of SIR. Basically, I used values to try and get a very narrow peak in my "I" curve. I immediately noticed that the number of steps I used in previous examples to make smooth curves no longer worked. Additionally, I saw that changing the total length of time on the graph compresses the curves in the horizontal direction which makes changes in S, I, or R appear more drastic. This then in turn required smaller timesteps to make up for the horizontal compression.

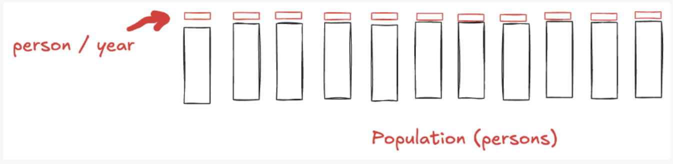
I also realized that changing the timesteps only matters when they get long enough to remove the smooth appearance of the curves. If they are any smaller, the already smooth curves just get smoother, and we can't tell a difference the way we can if they get more jagged as with longer steps.
helpful! 0

Harper Chen 2 months ago
I had a similar realization. The key difference between t and the timestep became much clearer for me. If t = 30 and numsteps = 100, we're essentially dividing the 30-day period into 100 smaller intervals (with each timestep being 30/100 days), and we update S, I, and R for each of those segments. about SIRPLOT and how this relates to the "length of time" we want to look at, I interpret "length of time" as referring to the total duration we're analyzing for the illness. So, if t = 30, it's the full 30-day window. The timestep is just how finely we slice that window, but the "length of time" is the total time period we're examining, not the step size.

helpful! 2

Nandini Bhat 2 months ago
Thanks, Harper! I noticed the same thing while changing tfinal, but wasn't entirely sure how to explain what was going on there. If tfinal = 3, the number of steps that resulted in a smooth graph was 15 (with a resulting timestep of 0.2). When I changed tfinal to 30, I could still see graph segments. Changing the number of steps to 150 gave me a smooth curve, and resulted in the same timestep of 0.2. If you change the period of time your graph covers, I assume you have to change the number of steps accordingly? Like 3 * 10 = 30 and 15 * 10 = 150 in this case.
helpful! 0

Charlie Bailey 3 months ago
I've been struggling as well to wrap my head about this section/set of problems. For problem 24, I'm thinking about it visually and very literally. In the diagram below, we can think about the population being split out into individual persons (the black rectangles) and the growth rate being the red rectangle on top. In this way, it makes sense when the units for k are persons per year/persons—in effect, while it might not make literal sense, we are essentially saying that each person in the population of 100,000 is growing by 0.015 persons each year. At least that's the way I'm understanding it.



~ An instructor (Elisabeth stade) thinks this is a good comment ~

helpful! 1

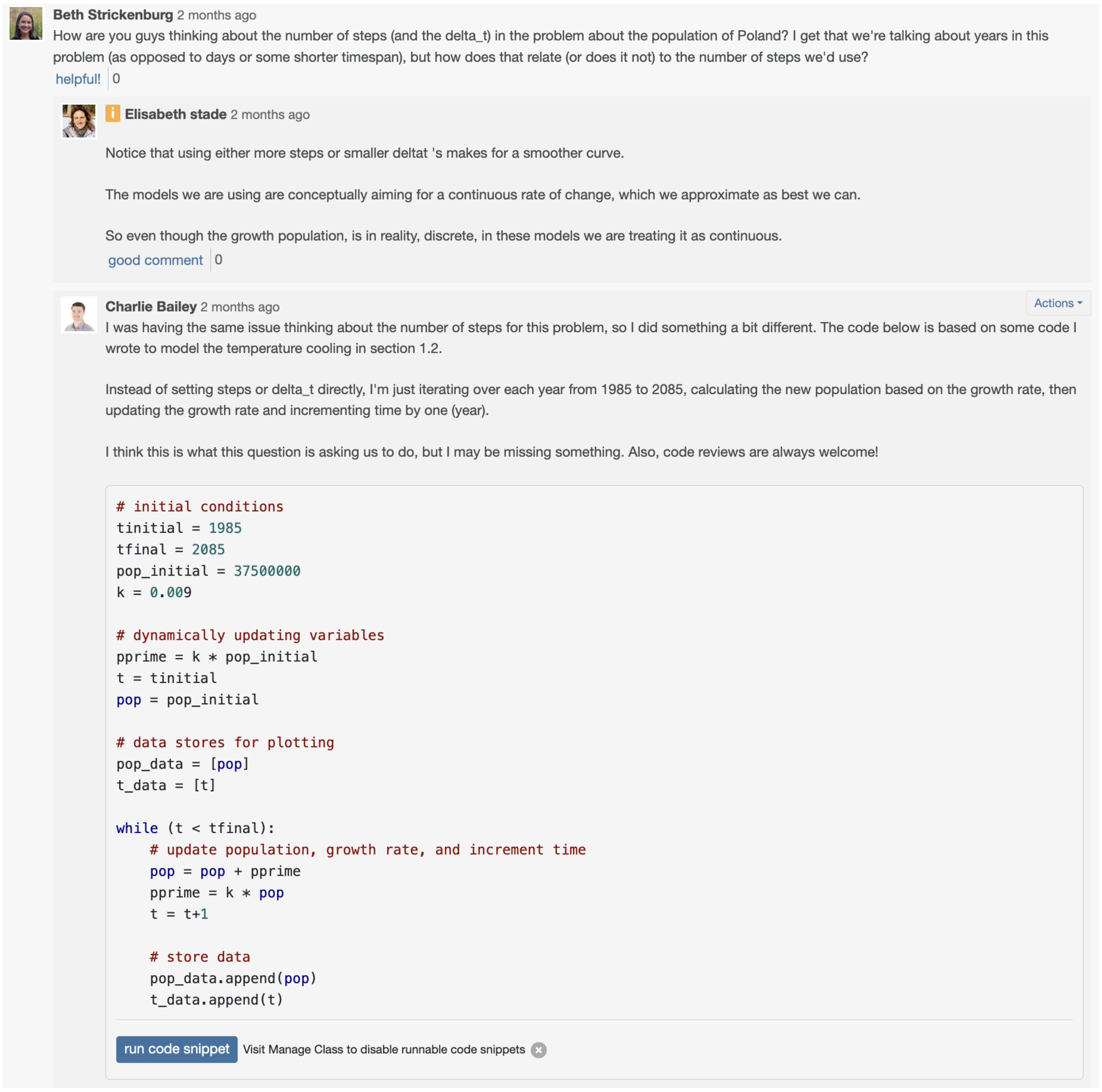
Travis Williams 2 months ago
The visual representation is definitely helpful, and I came to the same conclusion. Each person in the population is growing by the delta 0.015 person each year. It doesn't make any sense literally - we would be a world of giants at that rate! I struggled with this one too, because of that intuition.
helpful! 0

Ross Hastings 2 months ago
They way I went about solving this one was to do the same math for the larger city. I know it sounds intuitive already and should go without saying, but the population growth rate k is also 0.015 for the larger city.

$P' = 3000 = k P = k \times 200000$ so $k = 3000/200000 = .015$.

I think the important thing to remember is that P' is an over simplification of the growth rate of cities. It's safe to assume that no one is entering or leaving the city (sounds like a dystopia), and the city's growth rate is self contained.
helpful! 0

Stephen Enriquez 2 months ago
I appreciate the visual interpretation here and found this helpful. I think the idea of persons per year per person is unintuitive at first glance, since we think of persons as discrete units--but concepts like "temperature" that are more on a gradient are easier to digest as variable over time.
helpful! 0



Beth Strickenburg 2 months ago
How are you guys thinking about the number of steps (and the delta_t) in the problem about the population of Poland? I get that we're talking about years in this problem (as opposed to days or some shorter timespan), but how does that relate (or does it not) to the number of steps we'd use?

Elisabeth stade 2 months ago

Notice that using either more steps or smaller deltat 's makes for a smoother curve.

The models we are using are conceptually aiming for a continuous rate of change, which we approximate as best we can.

So even though the growth population, is in reality, discrete, in these models we are treating it as continuous.

Charlie Bailey 2 months ago

I was having the same issue thinking about the number of steps for this problem, so I did something a bit different. The code below is based on some code I wrote to model the temperature cooling in section 1.2.

Instead of setting steps or delta_t directly, I'm just iterating over each year from 1985 to 2085, calculating the new population based on the growth rate, then updating the growth rate and incrementing time by one (year).

I think this is what this question is asking us to do, but I may be missing something. Also, code reviews are always welcome!

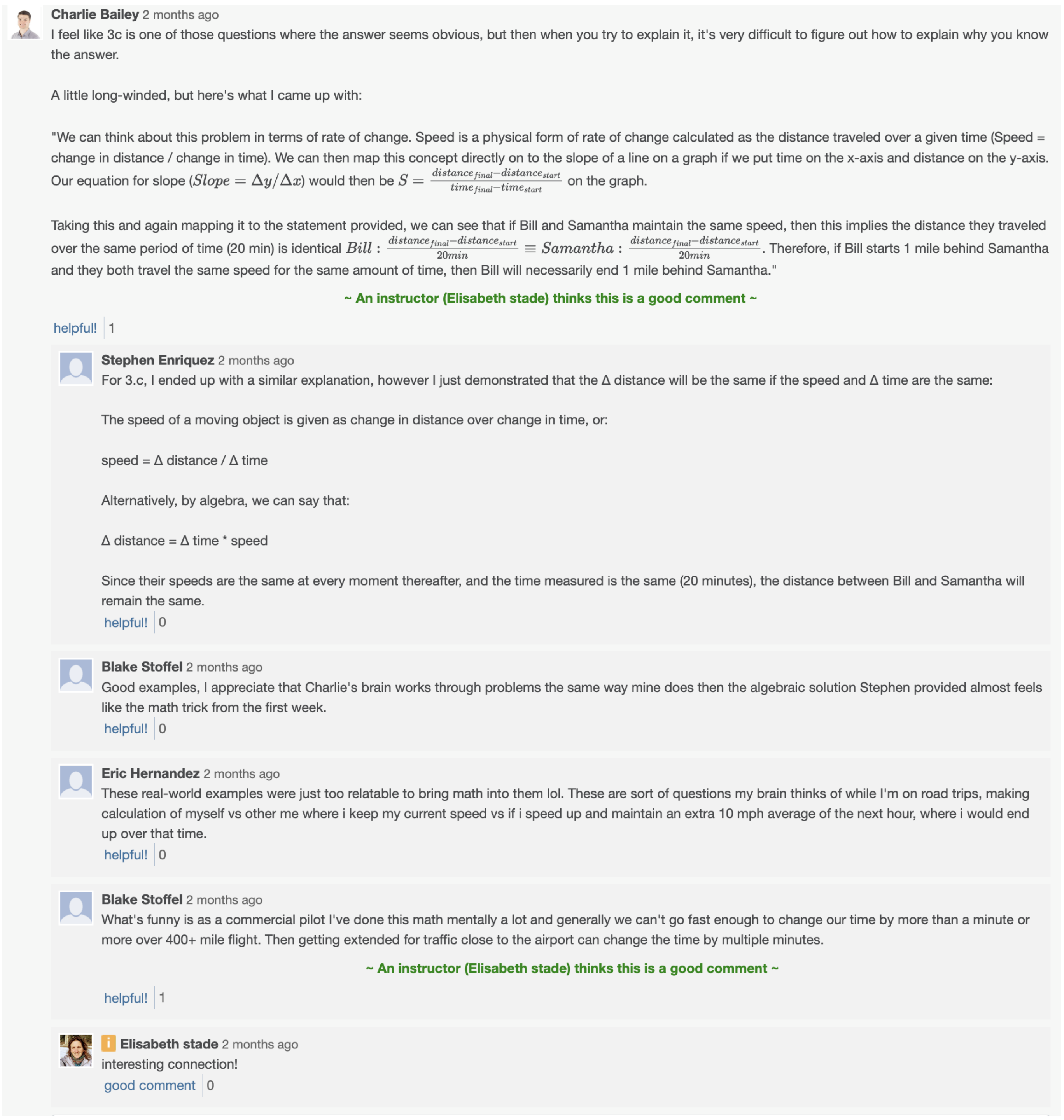
```
# initial conditions
tinitial = 1985
tfinal = 2085
pop_initial = 37500000
k = 0.009

# dynamically updating variables
pprime = k * pop_initial
t = tinitial
pop = pop_initial

# data stores for plotting
pop_data = [pop]
t_data = [t]

while (t < tfinal):
    # update population, growth rate, and increment time
    pop = pop + pprime
    pprime = k * pop
    t = t+1

    # store data
    pop_data.append(pop)
    t_data.append(t)
```



Charlie Bailey 2 months ago
I feel like 3c is one of those questions where the answer seems obvious, but then when you try to explain it, it's very difficult to figure out how to explain why you know the answer.

A little long-winded, but here's what I came up with:

"We can think about this problem in terms of rate of change. Speed is a physical form of rate of change calculated as the distance traveled over a given time (Speed = change in distance / change in time). We can then map this concept directly on to the slope of a line on a graph if we put time on the x-axis and distance on the y-axis. Our equation for slope ($Slope = \Delta y/\Delta x$) would then be $S = \frac{distance_{final}-distance_{start}}{time_{final}-time_{start}}$ on the graph.

Taking this and again mapping it to the statement provided, we can see that if Bill and Samantha maintain the same speed, then this implies the distance they traveled over the same period of time (20 min) is identical $Bill : \frac{distance_{final}-distance_{start}}{20min} \equiv Samantha : \frac{distance_{final}-distance_{start}}{20min}$. Therefore, if Bill starts 1 mile behind Samantha and they both travel the same speed for the same amount of time, then Bill will necessarily end 1 mile behind Samantha."

~ An instructor (Elisabeth stade) thinks this is a good comment ~

helpful! 1

Stephen Enriquez 2 months ago
For 3.c, I ended up with a similar explanation, however I just demonstrated that the Δ distance will be the same if the speed and Δ time are the same:

The speed of a moving object is given as change in distance over change in time, or:

speed = Δ distance / Δ time

Alternatively, by algebra, we can say that:

Δ distance = Δ time * speed

Since their speeds are the same at every moment thereafter, and the time measured is the same (20 minutes), the distance between Bill and Samantha will remain the same.
helpful! 0

Blake Stoffel 2 months ago
Good examples, I appreciate that Charlie's brain works through problems the same way mine does then the algebraic solution Stephen provided almost feels like the math trick from the first week.
helpful! 0

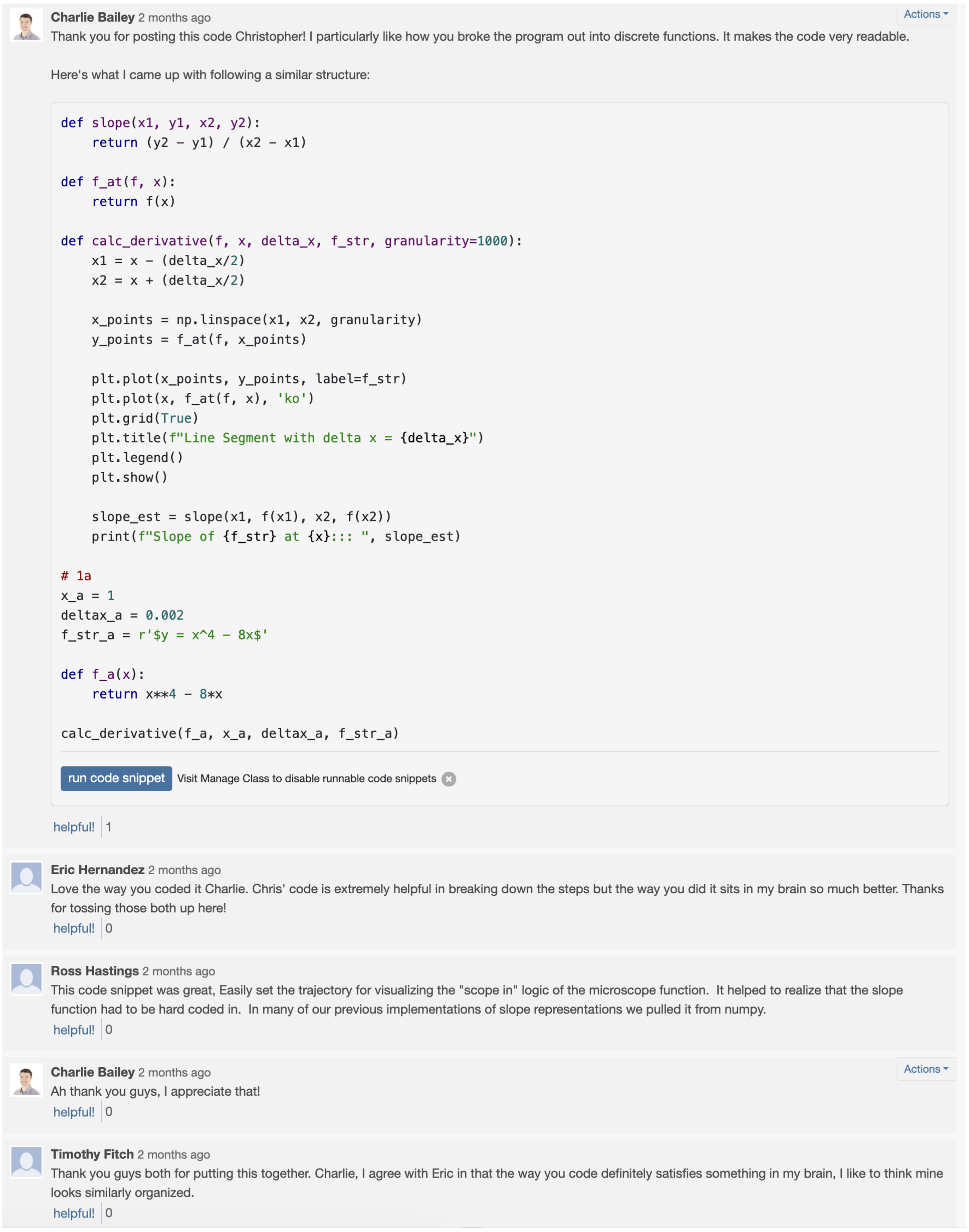
Eric Hernandez 2 months ago
These real-world examples were just too relatable to bring math into them lol. These are sort of questions my brain thinks of while I'm on road trips, making calculation of myself vs other me where i keep my current speed vs if i speed up and maintain an extra 10 mph average of the next hour, where i would end up over that time.
helpful! 0

Blake Stoffel 2 months ago
What's funny is as a commercial pilot I've done this math mentally a lot and generally we can't go fast enough to change our time by more than a minute or more over 400+ mile flight. Then getting extended for traffic close to the airport can change the time by multiple minutes.

~ An instructor (Elisabeth stade) thinks this is a good comment ~

helpful! 1

i **Elisabeth stade** 2 months ago
interesting connection!
good comment 0



Charlie Bailey 2 months ago

Thank you for posting this code Christopher! I particularly like how you broke the program out into discrete functions. It makes the code very readable.

Here's what I came up with following a similar structure:

```python
def slope(x1, y1, x2, y2):
    return (y2 - y1) / (x2 - x1)

def f_at(f, x):
    return f(x)

def calc_derivative(f, x, delta_x, f_str, granularity=1000):
    x1 = x - (delta_x/2)
    x2 = x + (delta_x/2)

    x_points = np.linspace(x1, x2, granularity)
    y_points = f_at(f, x_points)

    plt.plot(x_points, y_points, label=f_str)
    plt.plot(x, f_at(f, x), 'ko')
    plt.grid(True)
    plt.title(f"Line Segment with delta x = {delta_x}")
    plt.legend()
    plt.show()

    slope_est = slope(x1, f(x1), x2, f(x2))
    print(f"Slope of {f_str} at {x}::: ", slope_est)

# 1a
x_a = 1
deltax_a = 0.002
f_str_a = r'$y = x^4 - 8x$'

def f_a(x):
    return x**4 - 8*x

calc_derivative(f_a, x_a, deltax_a, f_str_a)
```

Eric Hernandez 2 months ago

Love the way you coded it Charlie. Chris' code is extremely helpful in breaking down the steps but the way you did it sits in my brain so much better. Thanks for tossing those both up here!

Ross Hastings 2 months ago

This code snippet was great, Easily set the trajectory for visualizing the "scope in" logic of the microscope function. It helped to realize that the slope function had to be hard coded in. In many of our previous implementations of slope representations we pulled it from numpy.

Charlie Bailey 2 months ago

Ah thank you guys, I appreciate that!

Timothy Fitch 2 months ago

Thank you guys both for putting this together. Charlie, I agree with Eric in that the way you code definitely satisfies something in my brain, I like to think mine looks similarly organized.

Charlie Bailey 2 months ago
Building on the numerical_point_derivative we can plot the full derivative function:

```
def numerical_function_derivative(f, f_str, xl=1, xr=3, num_points=20):
    x_points = np.linspace(xl, xr, num_points)
    y_points = np.array([numerical_point_derivative(f, x) for x in x_points])

    plt.plot(x_points, y_points)
    plt.title(f"Numerically calculated derivative of {f_str}")
    plt.grid(True)
    plt.show()

numerical_function_derivative(lambda x: 1/x, r'1/x')
```

run code snippet Visit Manage Class to disable runnable code snippets

Charlie Bailey 2 months ago
Keeping it simple to start. The nice thing about this simplicity is I can now call this function in a loop in my numerical_function_derivative to "build" the graph of the derivative point by point.

```
def numerical_point_derivative(f, a, h=0.0001):
    return (f(a + h) - f(a - h)) / (2*h)

derivative_1 = numerical_point_derivative(lambda x: 1/x, 2)
print(derivative_1) # -0.25000000062558314
```

run code snippet Visit Manage Class to disable runnable code snippets

helpful! 0

Stephen Enriquez 2 months ago
I have been keeping it very simple by defining 'func' as another python function and simply calling it in my derivative function. I've also found that parameterizing 'a', h' and 'func' allowed easier calls within the Jupyer notebook format (and also for use in later functions).

```
def Numerical_Point_Derivative(a, h, func):
    difference_quotient = (func(a + h) - func(a - h)) / (2 * h)
    return difference_quotient

def inverse(x):
    y = 1 / x
    return y

print(Numerical_Point_Derivative(2, 0.0002, inverse))
# -0.25000000249991716
```

run code snippet Visit Manage Class to disable runnable code snippets

But I like how you went with lambda which really simplifies things. I also see that people went with eval() which I've never seen before. Love these threads for learning small neat Python tricks.

helpful! 0

Lukas Talaga 2 months ago
I did something similar to Stephen last week, though it's something I struggle to remember. Passing function names into functions as arguments is still a bit strange to me, but it makes the code extremely readable, especially when you can name it something relevant so we don't have to keep jumping around the code to understand. That being said, the use of Charlie's lambda function in this case might make the code a little quicker to adapt to different functions, since less would need to be modified for each iteration/use of the code.

helpful! 0

Liam Keyek 2 months ago
I agree with both of you - I've been trying to pass functions as arguments in my solutions. Modularizing generally makes things easier to deal with in my experience!

helpful! 0

Charlie Bailey 2 months ago Actions
Yeah, the modularization has been extremely helpful in coding these activities. I've been able to reuse a lot of the functions I end up writing for subsequent problems. I've also started using the lambda functions just to cut down on attempting to name each function before passing it into a given calculator/plotting function. I was getting overwhelmed with a lot of f_{number} function names!

helpful! 0

Charlie Bailey 1 month ago
Really nice code Chris—thanks for sharing!

One thing I did on the last modeling activity that could be helpful here is using subplots within a single fig to plot multiple graphs side-by-side for easy comparison. In your implementation, you could throw all of the functions into a list and simultaneously iterate over the functions and the subplots to call PLOT on each function and plot it within a corresponding subplot. Something like this:

```
funcs = [A, B, C, D, E, F]

fig, axes = plt.subplots(3, 2, figsize=(15, 25))
axes = axes.flatten()

for i, func in enumerate(funcs):
    ax = axes[i]
    PLOT(0, 30, 50, func, ax)

...
```

run code snippet Visit Manage Class to disable runnable code snippets

You can add an ax param to your PLOT function, use that instead of plt and it should display all the graphs on a single figure!
helpful! | 0

Lukas Talaga 1 month ago
Before coding this out and trying it out, I was having a tough time recognizing how changing the number of steps would affect the accumulated value. After doing some testing, I've come to realize that similar to previous functions we worked on, increasing the number of steps will change the accumulated value, where more steps should make the final value more accurate. I noticed that the value eventually converges, though there can be some significant variation between small step values until the number of steps starts to capture most of the variation between datapoints.
helpful! | 0

Travis Williams 1 month ago
This is a great point! Changing the number of steps does seem to result in a more accurate accumulated value. I ended up rounding the accumulated value and delta y value to 4 digitsin order to match the table in the book. I wanted to ensure that I interpreted the code correctly, and needed to see that my table matched with the book.
helpful! | 0

Charlie Bailey 1 month ago Actions
For me, number of steps continues to be one of those things that I have to stop and really think about what is going on every time it comes up in the code.

One of the metaphors I'm using now is to think about it like the frame rate in a movie. If I increase the frame rate, I'm able to get super slo-mo video—capturing every minute detail. Conversely, decreasing the frame rate makes the movie seem to skip, almost like a picture book being flipped.

I *think* this metaphor somewhat captures what you're saying about increasing the number of steps creates more accurate final values—more steps means we capture more frames thus leading to a better overall picture of the change.
helpful! | 0