Enhancing Development Image Analysis Tools with GPT-4

Cayden Chang, Jason Junge, Scott E. Fraser, Francesco Cutrale
Translational Imaging Center, Bridge Institute, University of Southern California, Los Angeles, CA , USA

Bridge UnderGrad Science (BUGS) ***Summer Research Program***

Abstract

In Fluorescent Lifetime Imaging (FLIM), pixels contain information about the fluorescent lifetimes of the imaged fluorophores, rather than the typical qualities of intensity or color.
Because of this difference, imaging analysis tools are used very frequently in FLIM, as they are necessary for extracting quantitative data and creating useful visualizations. Exploiting these imaging analysis tools furthers discovery and our understanding of novel biological processes.
The focus of this project is centered around image analysis tools, specifically FLIM filtering and segmentation techniques developed at the Translational Imaging Center at USC. Despite its excellent functionality, the existing software tool is significantly limited by its choice of programming language: Matlab. Several reasons including high costs, a lack of community support, and OS and Version dependencies have prompted an effort to translate this code to Python for its ease of use and superior functionalities.
However, translating code to a new programming language is a tedious process, as significant time is invested in searching for equivalent functions between the old and new language. Generally, porting code between languages requires proper understanding of both the topic and the code itself, as well as both programming languages. In this project, we harness the power of GPT-4 in order to speed up the process of translating code significantly, using it to expedite learning of both the software and foundational knowledge, achieving faster deliverables.

Objectives

- Test the capabilities of GPT-4.0 for the task of translating image analysis code from Matlab to Python.
- Test GPT-4's ability to handle prompts that are vaguer and in layman terms, originating from someone who lacks experience in imaging and programming languages.
- Create a proof of concept for GPT-4's potential to save significant amounts of time in coding tasks.

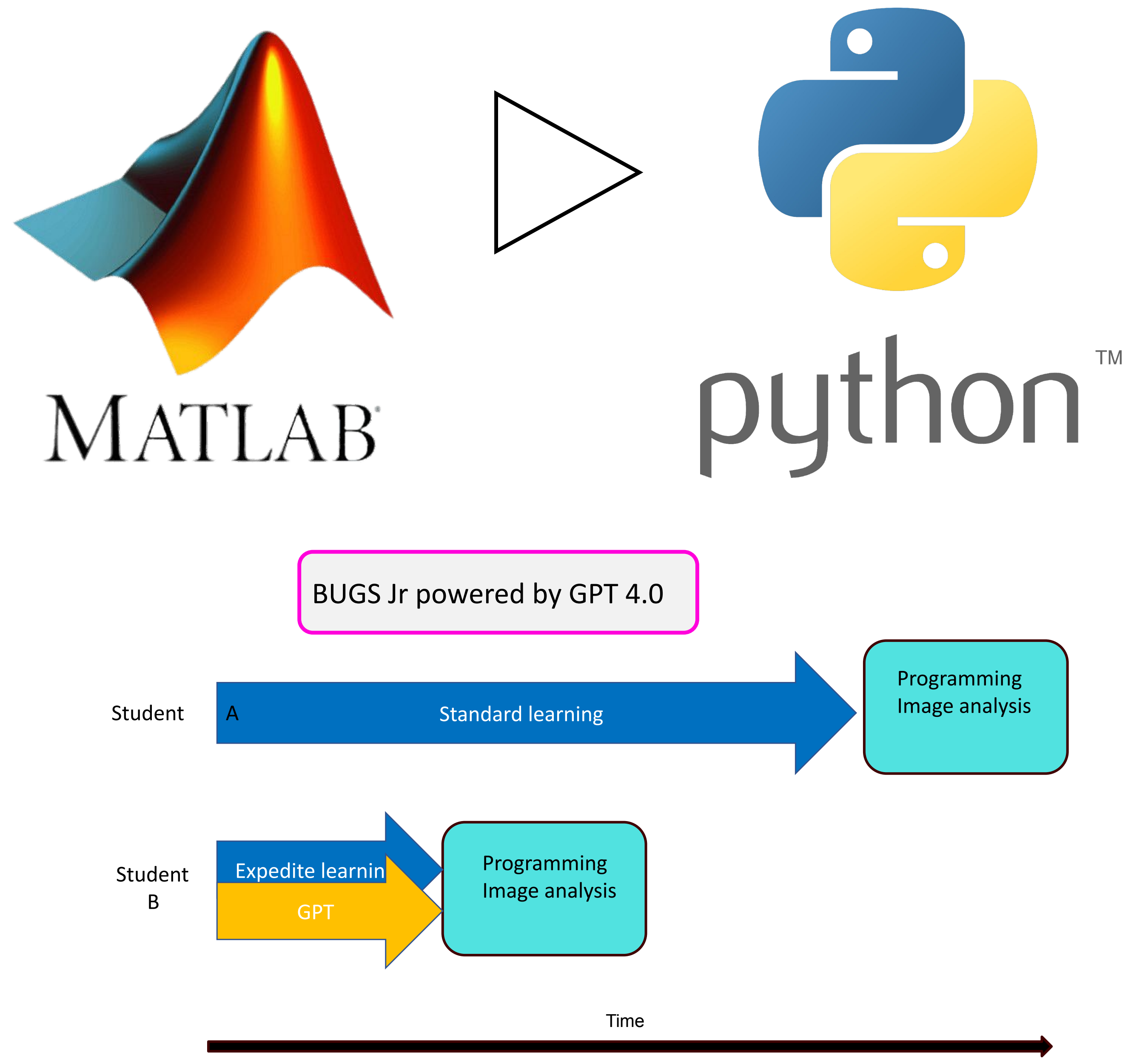


Figure 1 : Project Overview. *Top*, diagram illustrating the software conversion scope of this project from Matlab to Python. *Bottom*, illustration of the rationale behind a GPT-4 approach. A code conversion performed by conventional means requires in depth learning of both programming languages, resulting in an extended time to completion. By contrast, a GPT-4 powered approach has the potential to shorten the process and facilitate learning of the coding language.

Methods

Step 1: Copy

As GPT-4 cannot directly access the user's terminal, users must handle independently the creation of folders for the translation. Thus, a corresponding file structure was created in PyCharm to simplify the code evaluation and progress visualization. The main file, `analysistemplate.m`, contains the analysis algorithm, calling `Functions`, a folder with 26 different files. After recreating the file organization of the Matlab code in a PyCharm virtual environment, all the .m files are copied into GPT-4.

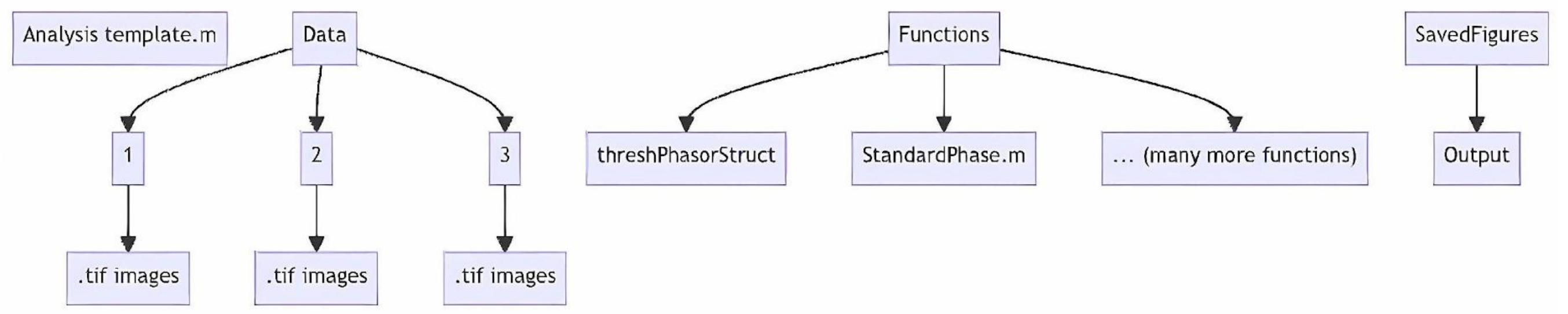


Figure 2: The file organization of the code

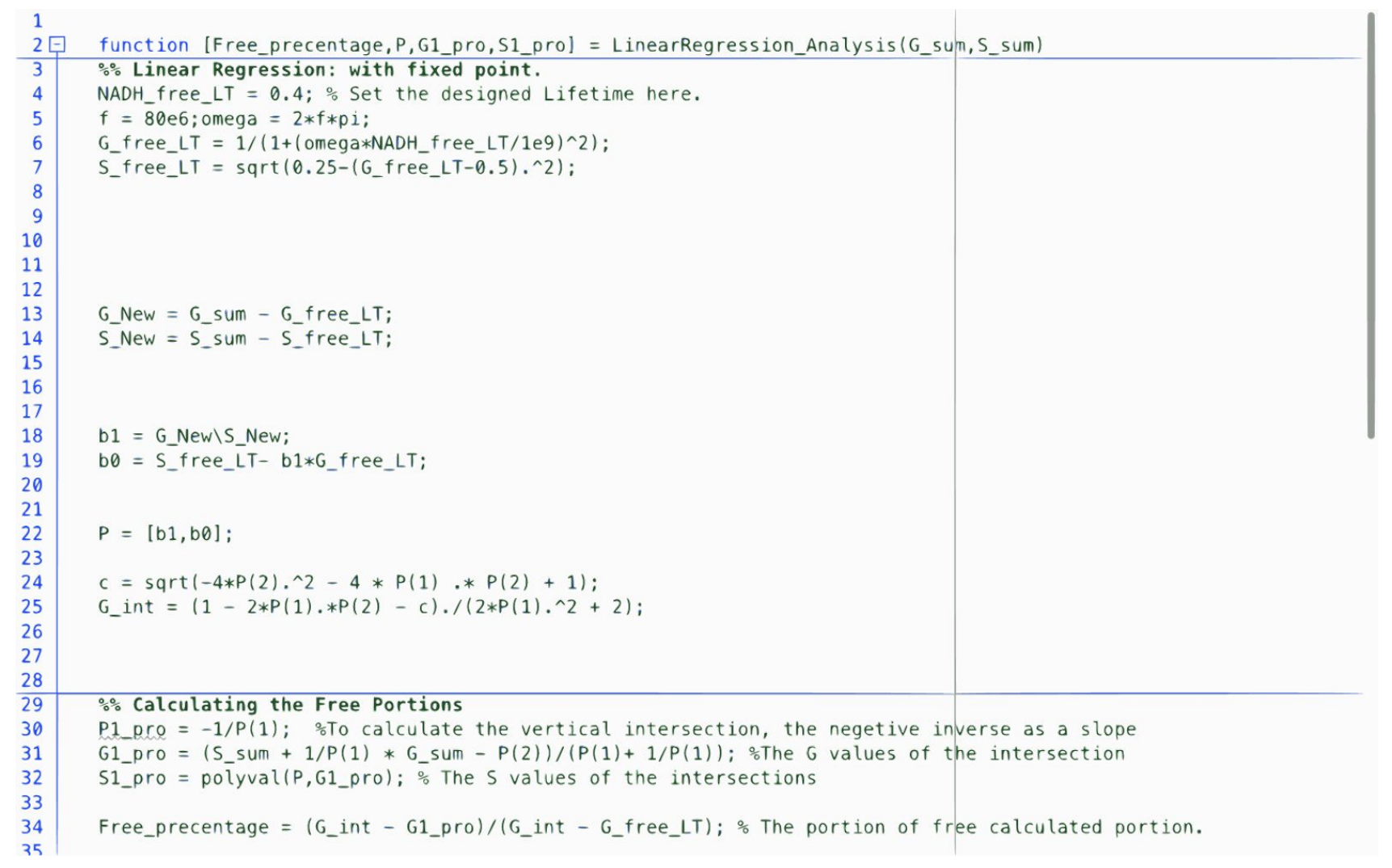
Step 2: Prompt

Prompts are instructions that users provide to GPT-4 to respond to. Proper prompt-engineering is essential achieve proper translation of programming code. For the initial prompt, GPT-4 was asked to assume a persona of an expert in software porting. Sample inputs and expected outputs were additionally told when available. The increased specificity of instructions and inputs greatly increased GPT-4's success rate. However, porting of long portions of code (e.g. >200 lines), necessitated an approach by-parts, as it overcame local boundaries of GPT-4 messages or inadequately translated the code. A successful approach in this scenario was to a unique prompt that ordered it translate batches line separately, recomposing the whole code at the end.

Step 3: Debugging

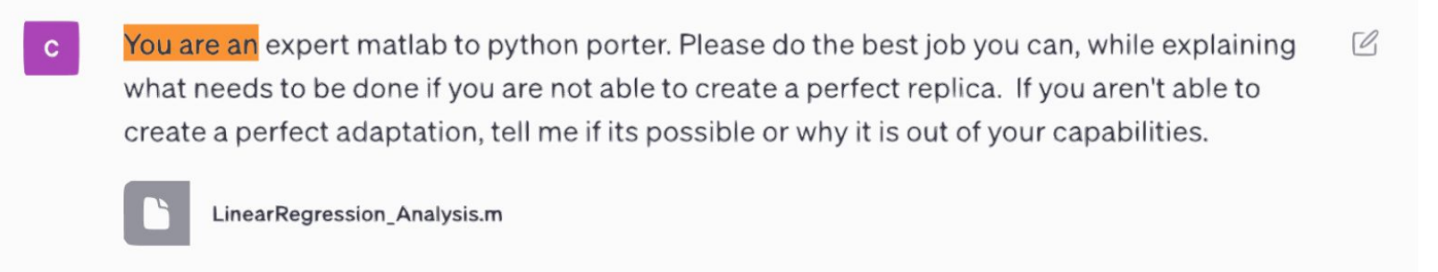
Debugging with GPT-4 was performed by of copy and pasting error codes and asking it to provide a revised version of the code. On some occasions, manual intervention was required to further correct GPT-4 conversion.

Experimental Process



```
function [Free_precentage,P,G1_pro,S1_pro] = LinearRegression_Analysis(G_sum,S_sum)
%% Linear Regression: with fixed point.
NADH_free_LT = 0.4; % Set the designed Lifetime here.
f = 80e6;omega = 2*f*pi;
G_free_LT = 1/(1+(omega*NADH_free_LT/1e9)^2);
S_free_LT = sqrt(0.25-(G_free_LT-0.5).^2);





G_New = G_sum - G_free_LT;
S_New = S_sum - S_free_LT;



b1 = G_New\S_New;
b0 = S_free_LT- b1*G_free_LT;


P = [b1,b0];

c = sqrt(-4*P(2).^2 - 4 * P(1) .* P(2) + 1);
G_int = (1 - 2*P(1).*P(2) - c)./(2*P(1).^2 + 2);



%% Calculating the Free Portions
P1_pro = -1/P(1);  %To calculate the vertical intersection, the negetive inverse as a slope
G1_pro = (S_sum + 1/P(1) * G_sum - P(2))/(P(1)+ 1/P(1)); %The G values of the intersection
S1_pro = polyval(P,G1_pro); % The S values of the intersections

Free_precentage = (G_int - G1_pro)/(G_int - G_free_LT); % The portion of free calculated portion.
```

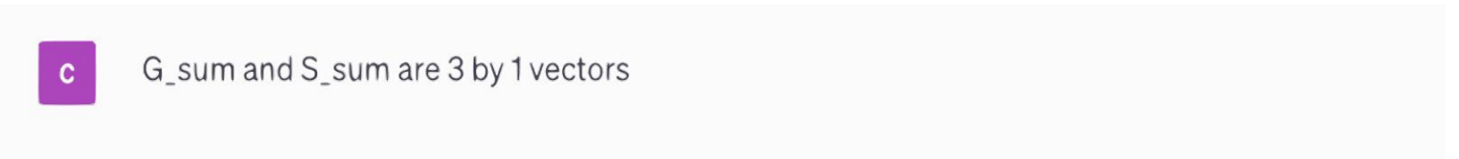
Figure 3: Screenshot of code for the function, `LinearRegressionAnalysis.m`



C You are an expert matlab to python porter. Please do the best job you can, while explaining what needs to be done if you are not able to create a perfect replica. If you aren't able to create a perfect adaptation, tell me if its possible or why it is out of your capabilities.

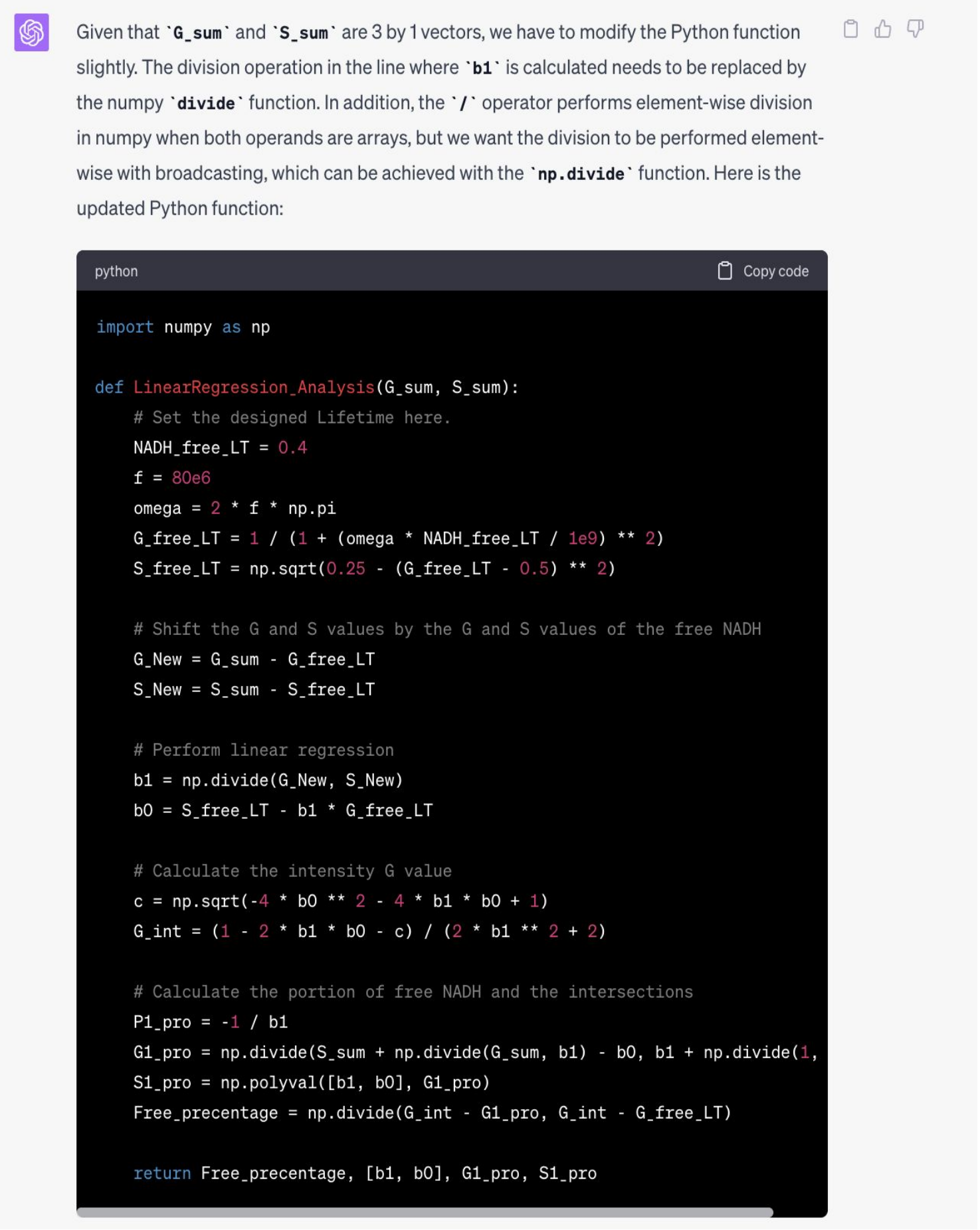
LinearRegression_Analysis.m

Figure 4: Screenshot of initial prompt used for roughly translating every <200 lines .m file. Components of the prompt include persona, and instructions to report difficulties or limitations.



C G_sum and S_sum are 3 by 1 vectors

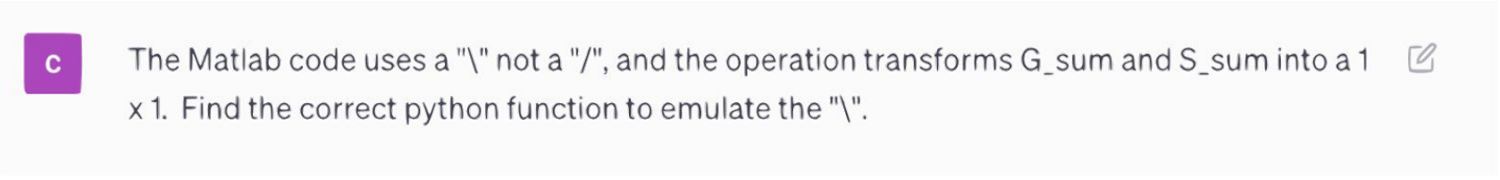
Figure 5 : Screenshot of additional prompt instructions provide more specific information about the Inputs to GPT-4. This prompt is specific to `LinearRegressionAnalysis.m`



Given that `G_sum` and `S_sum` are 3 by 1 vectors, we have to modify the Python function slightly. The division operation in the line where `b1` is calculated needs to be replaced by the numpy `divide` function. In addition, the `/` operator performs element-wise division in numpy when both operands are arrays, but we want the division to be performed element-wise with broadcasting, which can be achieved with the `np.divide` function. Here is the updated Python function:

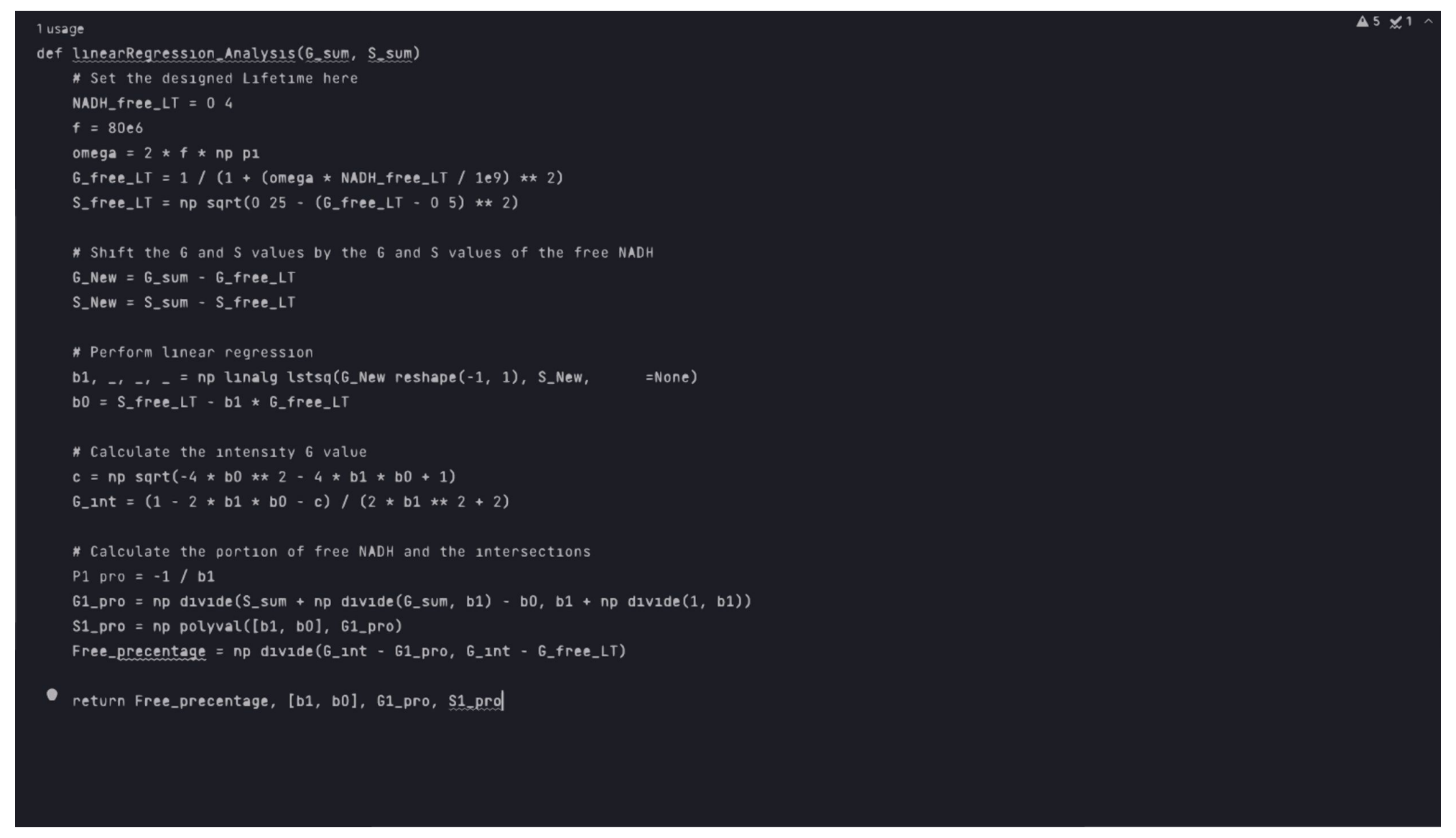
```python
import numpy as np

def LinearRegression_Analysis(G_sum, S_sum):
    # Set the designed Lifetime here.
    NADH_free_LT = 0.4
    f = 80e6
    omega = 2 * f * np.pi
    G_free_LT = 1 / (1 + (omega * NADH_free_LT / 1e9) ** 2)
    S_free_LT = np.sqrt(0.25 - (G_free_LT - 0.5) ** 2)

    # Shift the G and S values by the G and S values of the free NADH
    G_New = G_sum - G_free_LT
    S_New = S_sum - S_free_LT

    # Perform linear regression
    b1 = np.divide(G_New, S_New)
    b0 = S_free_LT - b1 * G_free_LT

    # Calculate the intensity G value
    c = np.sqrt(-4 * b0 ** 2 - 4 * b1 * b0 + 1)
    G_int = (1 - 2 * b1 * b0 - c) / (2 * b1 ** 2 + 2)

    # Calculate the portion of free NADH and the intersections
    P1_pro = -1 / b1
    G1_pro = np.divide(S_sum + np.divide(G_sum, b1) - b0, b1 + np.divide(1,
    S1_pro = np.polyval([b1, b0], G1_pro)
    Free_precentage = np.divide(G_int - G1_pro, G_int - G_free_LT)

    return Free_precentage, [b1, b0], G1_pro, S1_pro
```

Figure 6: Screenshot of an example erroneous output. When the GPT4 generated program is run, it returns a different value than the Matlab function. Note that in the line where b1 is defined, it mistakenly confuses a backslash, used in Matlab for matrix left division, for a forward slash prompting the use numPy "divide" function.



C The Matlab code uses a "\" not a "/", and the operation transforms G_sum and S_sum into a 1 x 1. Find the correct python function to emulate the "\".

Figure 7: Screenshot of specific prompting utilized to inform GPT-4 of its erroneous assumption. Note that specifying its error in the prompt is crucial, as GPT-4 would typically experience "confident incorrectness", continually producing outputs that would not carry out the desired function.

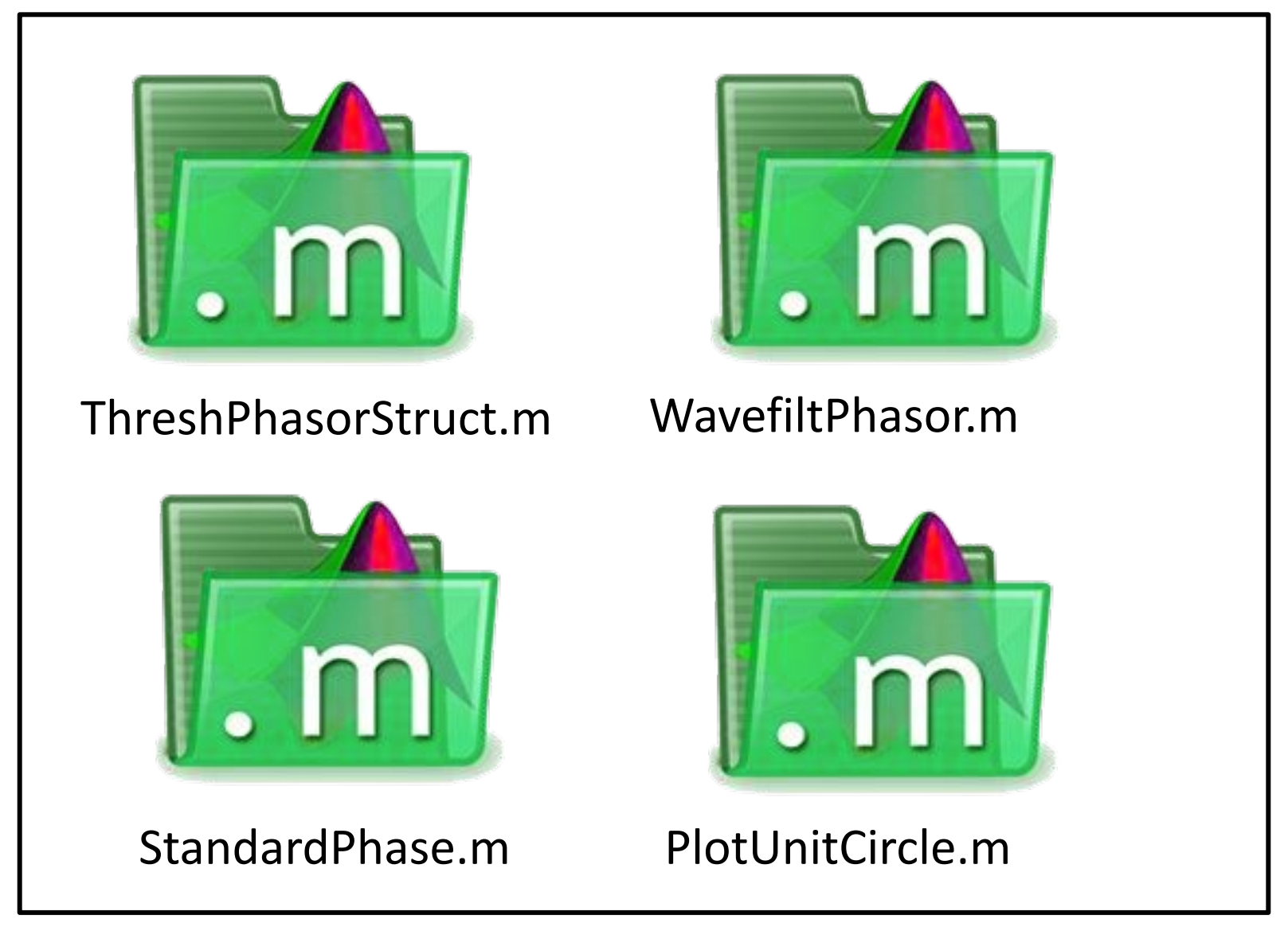


```python
def linearRegression_Analysis(G_sum, S_sum)
    # Set the designed Lifetime here
    NADH_free_LT = 0 4
    f = 80e6
    omega = 2 * f * np pi
    G_free_LT = 1 / (1 + (omega * NADH_free_LT / 1e9) ** 2)
    S_free_LT = np sqrt(0 25 - (G_free_LT - 0 5) ** 2)

    # Shift the G and S values by the G and S values of the free NADH
    G_New = G_sum - G_free_LT
    S_New = S_sum - S_free_LT

    # Perform linear regression
    b1, _, _, _ = np linalg lstsq(G_New reshape(-1, 1), S_New,      =None)
    b0 = S_free_LT - b1 * G_free_LT

    # Calculate the intensity G value
    c = np sqrt(-4 * b0 ** 2 - 4 * b1 * b0 + 1)
    G_int = (1 - 2 * b1 * b0 - c) / (2 * b1 ** 2 + 2)

    # Calculate the portion of free NADH and the intersections
    P1 pro = -1 / b1
    G1_pro = np divide(S_sum + np divide(G_sum, b1) - b0, b1 + np divide(1, b1))
    S1_pro = np polyval([b1, b0], G1_pro)
    Free_precentage = np divide(G_int - G1_pro, G_int - G_free_LT)

    return Free_precentage, [b1, b0], G1_pro, S1_pro
```

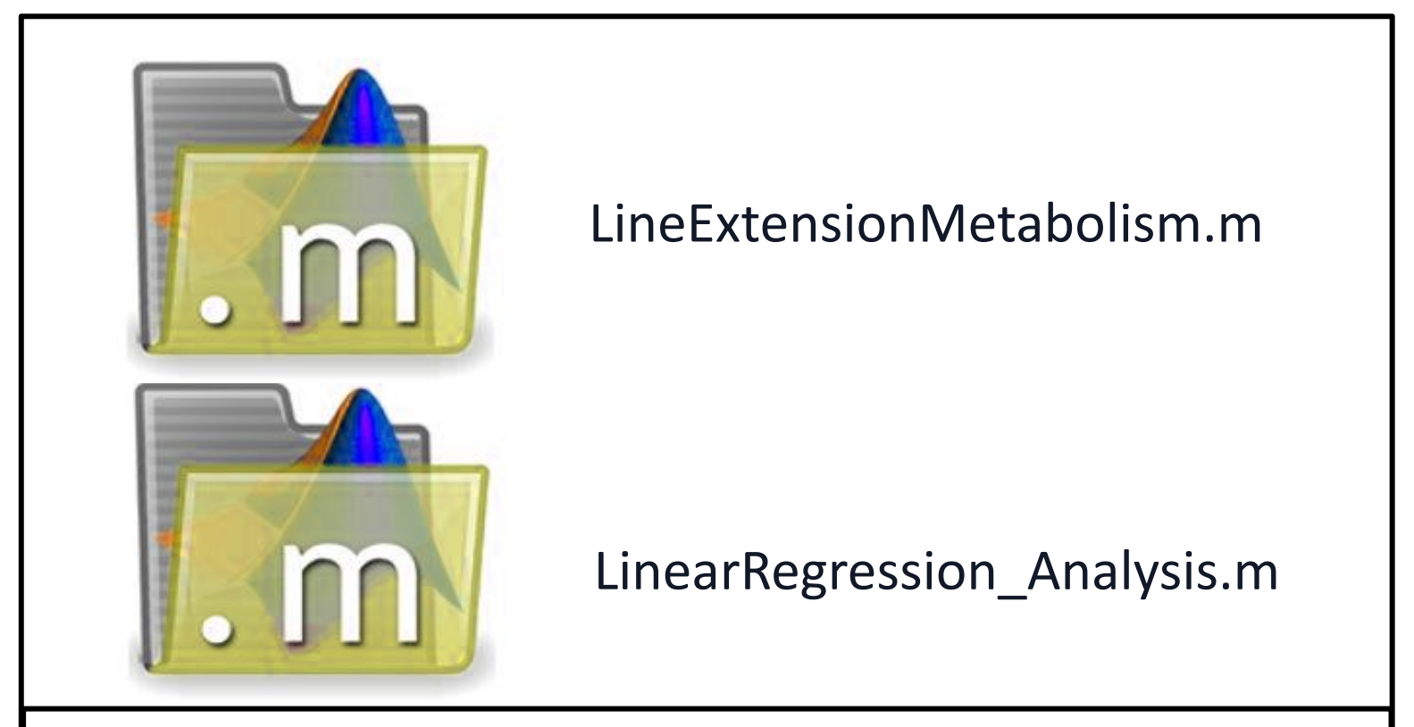
Figure 8: Screenshot of GPT-4's final working output in PyCharm.

Results

Results for code porting of the 10 functions called in the Matlab code `analysistemplate.m` to Python. The ported code has been segmented into three divisions in accordance with their distinct functions. However, certain directories have not been tested, primarily due to the ongoing troubleshooting of the main file, delaying the some of the testing.



Converted successfully



Required troubleshooting but eventually functional

MaskPhasorStruct.m
MedfiltPhasor.m
NImfiltphasor.m
PlotPhasorFast.m

Untested

Summary

- GPT-4 can quickly generate rough translations of code from one language to another, significantly reducing the time spent on manual translation. Its strengths are in converting shorter pieces of code, as it usually handles them excellently.
- GPT-4 can debug code if error messages are reported or if logical mistakes are specifically pointed out, reducing the time spent on troubleshooting and debugging.
- GPT-4 can handle vague prompts, making it a useful tool for individuals who lack experience in programming languages or computer science, saving them the time of learning the intricacies of a new programming language
- Prompt-engineering is critical in the success of the task requested to the AI.

CONTACT US

bridge.usc.edu/bugs **Caaaayden@gmail.com;** **Cutrale@usc.edu**