Supplementary Document

Generative AI and Simulation Modeling: How Should You (Not) Use Large Language Models Like ChatGPT

Table of Content:

S1. Python codes in online repository

We present the codes used in this paper in the following public repository:

https://github.com/ali-akhavan89/generative-AI-and-simulation-modeling

The online repository includes these five code files:

- Python code for estimating the parameters of the example model
- Python code for the example model, revised based on ChatGPT's feedback
- Python code for estimating the parameters of the example model
- Python code used for sensitivity analysis by changing EFT and PAT
- Python code provided by ChatGPT to plot a heatmap for sensitivity analysis

S2. Details of optimization iterations provided in a ChatGPT prompt

Iteration	Total nfev	Cost	Cost reduction	Step norm	Optimality
0	1	5.4874e+08			1.96e+09
1	2	8.3329e+06	5.40e+08	8.01e+02	6.03e+07
2	3	5.3994e+06	2.93e+06	1.35e+03	2.83e+07
3	4	5.2599e+06	1.40e+05	4.10e+02	9.64e+07
4	5	3.6457e+06	1.61e+06	2.88e+02	4.16e+06
5	6	3.4853e+06	1.60e+05	2.98e+02	2.73e+06
6	7	3.3795e+06	1.06e+05	2.10e+02	2.45e+06
7	8	3.2289e+06	1.51e+05	3.45e+02	1.18e+06
8	9	2.9876e+06	2.41e+05	6.58e+02	1.28e+06
9	10	2.9562e+06	3.14e+04	2.40e+01	2.08e+06
10	11	2.9313e+06	2.50e+04	3.76e+01	1.92e+06
11	13	2.9306e+06	6.12e+02	2.10e-01	5.53e+05
12	14	2.9303e+06	3.91e+02	2.73e-02	3.83e+05
13	15	2.9302e+06	1.51e+01	3.07e-02	3.90e+05
14	16	2.9302e+06	4.82e+01	4.12e-03	2.47e+05
15	17	2.9302e+06	2.03e+01	3.43e-03	4.41e+05
16	19	2.9302e+06	1.56e+00	3.85e-04	1.94e+05
17	27	2.9302e+06	2.47e-01	1.40e-08	1.66e+05
18	34	2.9302e+06	1.31e-01	7.36e-12	1.91e+05

S3. Supplementary figure

Here is a step-by-step guide to creating a Flask application with a slider and a graph based on your Python code:

1. **Install Flask:** If you haven't already, you'll need to install Flask. You can do this using pip:

bash Copy code

```bash
pip install flask
```

2. **Create a Flask Application:** Write a Python script to create a Flask application. Here's a template that integrates your ODE model:

python Copy code

```python
from flask import Flask, render_template, request
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import io
import base64
```

3. **Create an HTML Template:** You'll need an HTML template named `index.html` in a `templates` directory. Here's a basic one with a slider and an image to display the graph:

html Copy code

```html
<!doctype html>
<html>
<head>
    <title>ODE Model Interactive Graph</title>
</head>
<body>
    <h1>ODE Model Interactive Graph</h1>
    <form method="post">
        <label for="MPC">Marginal Propensity to Consume:</label>
        <input type="range" id="MPC" name="MPC" min="0" max="1" step="0.01" value="{
        <input type="submit" value="Update Graph">
    </form>
    <img src="data:image/png;base64,{{ data }}">
</body>
</html>
```

4. **Run Your Flask Application:** Execute the Python script to start the Flask server and visit `http://127.0.0.1:5000` in your web browser to interact with the application.
5. **Deployment:** For real-world usage, you'll want to deploy your application to a web server. You can use services like Heroku, AWS, or Google Cloud Platform for deployment.

Figure S1. Step-by-step instructions for setting up the Flask application provided by ChatGPT

S4. Model documentation provided in a ChatGPT prompt

Introduction

The purpose of this model is to replicate the impact of COVID-19 shock on US GDP growth and to estimate the change in consumer behavior. This model employs a system of ordinary differential equations (ODEs) to simulate the dynamic relationship between expected income, consumption, and the Gross Domestic Product (GDP) of the United States. By integrating key economic indicators and parameters such as the Marginal Propensity to Consume (MPC), Expectation Formation Time, and Production Adjustment Time, the model provides insights into how these factors interact over time, particularly in response to significant external shocks like the COVID-19 pandemic.

The model accounts for the temporal shifts in consumer behavior resulting from the pandemic, particularly through the modification of the MPC during the COVID-19 period. This is achieved by adjusting the Reference MPC based on a parameter (alpha) during the initial phase of the pandemic, reflecting a decrease in consumer spending. The integration of real-world economic data, including investment, government expenditure, and net exports, further enhances the model's applicability.

Utilizing data from 2013 to 2023, the model offers a view of the pre-pandemic economic conditions, the immediate impact of the pandemic, and the subsequent recovery phase. This time frame allows for a detailed analysis of the pandemic's effects on the U.S. economy and aids in understanding the resilience and adaptability of consumer behavior in the face of such unprecedented challenges.

Model Parameters and Variables

The model includes several key parameters and variables, each with a specific economic interpretation:

- **Marginal Propensity to Consume (MPC)**: This parameter represents the fraction of changes in income that the population will spend on consuming goods and services, as opposed to saving. It is central to understanding consumer behavior in the model.
- **Expectation Formation Time**: This parameter indicates the time it takes for people to adjust their expectations regarding future income, reflecting the dynamic nature of the economy.
- **Production Adjustment Time**: It signifies the time required for production levels to adapt to the prevailing level of demand and other economic indicators.
- **Investment, Government Expenditure, Net Exports**: These are key components of the Aggregate Demand in the model. They are treated as external inputs and are fed into the model through interpolation of historical data.

Code Description

Libraries Used

- **NumPy**: This library is used for numerical computing in Python. In this model, NumPy is primarily utilized for array manipulation and mathematical operations.
- **SciPy**: The **integrate** and **interpolate** modules from SciPy are employed. The **integrate** module's **odeint** function is crucial for solving the system of ordinary differential equations (ODEs) that form the core of the model. The **interpolate** module's **interp1d** function is used to create a continuous function from discrete data points, allowing for the interpolation of economic data over time.
- **Matplotlib**: This plotting library is used for creating static, interactive, and animated visualizations in Python. Matplotlib's **pyplot** interface is used to plot the simulated GDP against historical data, providing a visual representation of the model's output.
- **Pandas**: A library offering data structures and data analysis tools. It is used for reading and manipulating the input data stored in an Excel file, making data handling efficient and straightforward.

Importing Data

The model imports economic data from an Excel file using Pandas. This data includes time series for investment, government expenditure, net exports, and GDP. The data is collected from Federal Reserve Economic Data. The **interp1d** function from SciPy's **interpolate** module is used to create continuous functions from these discrete data sets, enabling the model to utilize real-world economic data dynamically over the simulation period.

Code Structure

The core of the model is a function named **model** that defines a system of ODEs representing expected income and GDP. This function takes the current state of the system (expected income and GDP), time, and model parameters as inputs and returns the derivatives of expected income and GDP. The system of ODEs is solved using the **odeint** function from SciPy over a specified time range, and the results are plotted against historical data for comparison and analysis.

Model Definition

This economic model is constructed around a system of Ordinary Differential Equations (ODEs), which are utilized to simulate the dynamics of expected income and Gross Domestic Product (GDP) in response to economic variables and external shocks, such as the COVID-19 pandemic.

System of ODEs

The model uses two main ODEs that describe the evolution of Expected Income and GDP over time. These equations are central to capturing the dynamic behavior of the economy under various conditions.

1. **Change in Expected Income**: This is modeled as the difference between GDP and Expected Income, divided by the Expectation Formation Time (EFT). This equation reflects how expectations about future income adjust over time based on the current state of the economy.
2. **Change in GDP**: It is derived from the difference between Aggregate Demand and GDP, adjusted by the Production Adjustment Time (PAT). Aggregate Demand itself is calculated as the sum of Consumption (dependent on MPC and Expected Income), Investment (I), Government Expenditure (G), and Net Exports (NX). This equation embodies the response of GDP to fluctuations in overall economic demand.

Model Function

The **model** function encapsulates these ODEs. It is designed to be used with the **odeint** solver from SciPy, which numerically integrates these equations over a specified time period.

- **Inputs**: The function accepts the current state vector (comprising Expected Income and GDP), the current time point, and a tuple of parameters (Reference MPC, EFT, PAT, alpha).
- **Outputs**: It returns the derivatives of Expected Income and GDP as an array, representing the rate of change of these variables at each time point.

COVID-19 Shock Adjustment

A feature of the model is its ability to adjust for the economic impact of the COVID-19 pandemic. This is achieved through a temporal modification of the MPC during the initial phase of the pandemic (2020 to 2020.25). The MPC is adjusted by a factor of **alpha**, representing the reduction in consumer spending due to the pandemic. This adjustment captures the immediate and pronounced shift in consumer behavior during the early stages of the crisis.

Numerical Solution and Analysis

The model's ODEs are solved numerically using the **odeint** function over a predefined range of quarter-year intervals from 2013 to 2023. This approach allows for an analysis of the model's behavior over time, offering insights into the interplay between economic variables and their collective impact on GDP and expected income, especially during the period of the COVID-19 shock.

Data Sources

Federal Reserve Economic Data (FRED)

The data utilized in this economic model is sourced from the Federal Reserve Economic Data (FRED), a database maintained by the Federal Reserve Bank of St. Louis. FRED offers a wide array of economic data, making it a useful resource for empirical economic analysis and modeling.

Data Selection and Relevance

For this model, we have selected specific time series from FRED that are crucial for simulating the U.S. economy:

- **Investment Data**: Reflects the total amount of investment in the economy over the time period. This data is integral to understanding the role of investment in driving economic growth.
- **Government Expenditure Data**: Represents the spending by the government, a key component of the overall economic demand.
- **Net Exports Data**: The difference between the country's total value of exports and total value of imports, indicating the contribution of trade to the economy.
- **GDP Data**: Gross Domestic Product figures provide the baseline against which the model's output (simulated GDP) is compared.

Data Handling and Processing

The data is imported into the model using Pandas, a data manipulation library in Python. This allows for efficient handling and transformation of the time series data. The data is initially stored in an Excel file (Data.xls), which is read into the model.

Interpolation of Data

Given that the model operates on a continuous time frame, while the economic data is available in discrete time points, interpolation is essential. We used the **interp1d** function from SciPy's **interpolate** module to create continuous functions from the discrete data points. This method ensures that the model can estimate economic variables at any given time within the simulation period.

The interpolation is set to "extrapolate" to allow the model to estimate values beyond the range of the available data points.

Data Reliability and Credibility

The choice of FRED as the data source adds reliability to the model, given FRED's reputation for providing accurate and up-to-date economic data.

Simulation Execution

Setting Up the Simulation Environment

The simulation of the economic model is executed in a Python environment, requiring the installation of specific libraries: NumPy, SciPy, Matplotlib, and Pandas. These libraries are essential for numerical calculations, solving differential equations, data handling, and visualization.

Defining Simulation Parameters

The simulation is driven by a set of pre-defined parameters that are estimated later. However, in the version of the model without calibration, we used randomly assigned values:

- **Expectation Formation Time (EFT)**: Set at 2 years, this parameter governs the speed at which expected income adjusts to the GDP.
- **Production Adjustment Time (PAT)**: Set at 1 year, it dictates the responsiveness of GDP to changes in aggregate demand.
- **Net Exports**: A fixed value of 10, representing the net export level in the model.

- **Reference Marginal Propensity to Consume (MPC)**: Initially set at 0.8, indicating the proportion of income spent on consumption.
- **Alpha**: Set at 0.4, this parameter represents the reduction in MPC during the COVID-19 shock period (2020 to 2020.25).

Initializing Model Variables
The model begins with specific initial conditions for the state variables:

- **Expected Income**: Initially set at 250 (in billions of dollars), representing the starting value for expected consumer income.
- **GDP**: The initial value is set at 16648.189 (in billions of dollars), aligning with historical GDP data at the start of the simulation period.

Running the Simulation
The simulation covers a period from 2013 to 2023, divided into quarter-year intervals. This time frame is selected to encapsulate the pre-pandemic economic conditions, the immediate impacts of the pandemic, and the beginning of the recovery phase.
The core of the simulation involves solving the system of ODEs using the **odeint** function from SciPy. This numerical solver integrates the model equations over the specified time points, considering the initial conditions and parameters. The resulting output provides a time series of Expected Income and GDP values.

Post-Processing and Analysis
After the simulation, the results are unpacked and analyzed:

- The expected income and GDP values are extracted from the results for further analysis.
- These values are compared against the historical data to evaluate the model's accuracy and to understand the economic impact of the COVID-19 shock.

Visualization
The final step involves visualizing the simulated GDP alongside the actual GDP data using Matplotlib. This graphical representation allows for an intuitive comparison of the model's output with real-world data, highlighting the model's effectiveness in simulating economic trends and shocks.

Results and Visualization

Analysis of Simulation Results
Key findings include:

1. **Expected Income Trends**: The model traces the trajectory of expected income, highlighting how consumer expectations evolve in response to changing economic conditions, including the impact of the COVID-19 pandemic.
2. **GDP Dynamics**: The simulation illustrates the fluctuations in GDP, showing the immediate effects of external shocks and the gradual adaptation of the economy over time. This is particularly evident during the COVID-19 period, where the model captures the sharp contraction and subsequent recovery phases.
3. **Impact of COVID-19**: The results demonstrate the significant impact of the COVID-19 shock on both expected income and GDP. The adjustment of MPC during the pandemic period models the reduction in consumer spending and its broader economic implications.
4. **Comparative Analysis**: By juxtaposing the simulated data against actual historical data, the model's ability in replicating real-world economic patterns is assessed.

Visualization of Results
Key components of the visualization include:

- **Simulated GDP vs. Historical Data**: A line graph is generated using Matplotlib, where the simulated GDP is plotted alongside the historical GDP data. This visualization aids in directly comparing the model's predictions with actual economic outcomes.
- **Graphical Layout**: The graph is designed for clarity and ease of understanding.

Replication Steps

Environment Setup

To replicate the results of this economic model simulation, the following setup is required:

1. **Python Installation**: Ensure Python (version recommended: 3.8 or newer) is installed.
2. **Library Installation**: Install the required Python libraries: NumPy, SciPy, Matplotlib, and Pandas. This can be done using pip, Python's package installer, with the command:
   ```
   pip install numpy scipy matplotlib pandas
   ```
3. **Data Files**: Obtain the necessary data files from Federal Reserve Economic Data (FRED). Specifically, gather time-series data for Investment, Government Expenditure, Net Exports, and GDP. Save this data in an Excel file named 'Data.xls'.

Running the Model

Follow these steps to execute the simulation:

1. **Code Preparation**: Copy the provided Python code into a Python script (**.py** file) or a Jupyter notebook (**.ipynb** file).
2. **Data File Placement**: Place the 'Data.xls' file in an accessible directory. Ensure the file path in the code matches the location of this file on your system.
3. **Parameter Adjustment (Optional)**: Adjust the model parameters (EFT, PAT, MPC, etc.) as needed to explore different scenarios or to align the model with updated economic data.
4. **Execute the Script**: Run the script or notebook. If using a command line interface, navigate to the directory containing the script and execute it using:
   ```
   python script_name.py
   ```

For Jupyter notebooks, run each cell sequentially.

Analyzing the Output

Upon execution, the model will generate a plot comparing the simulated GDP with the historical data. Examine this plot to assess the model's performance and to understand the economic dynamics during the simulation period.

Troubleshooting Common Issues

- **Data Import Errors**: Ensure the 'Data.xls' file is correctly formatted and located in the specified path. Double-check the column names and data types.
- **Library Compatibility**: If there are errors related to library functions, verify that you are using the recommended versions of Python and the libraries.
- **Numerical Instabilities**: If the model produces unrealistic or unstable results, consider adjusting the initial conditions or the parameters, and ensure that the ODE solver is configured correctly.

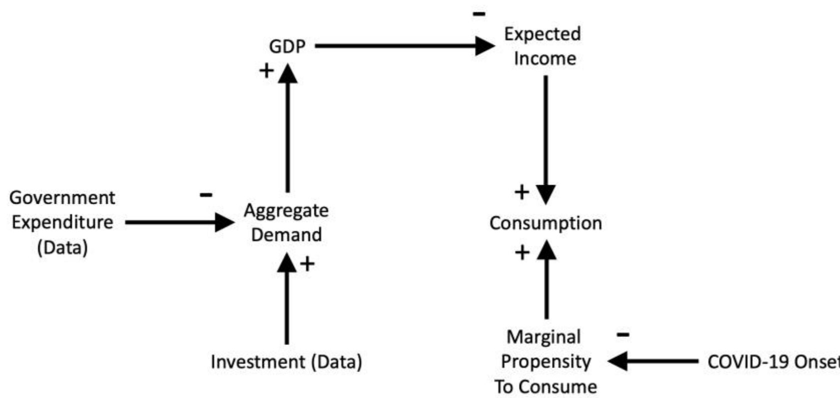
Further Exploration

- **Model Extension**: Researchers can extend the model by incorporating additional economic factors or by adjusting it to simulate different economic scenarios.
- **Sensitivity Analysis**: Conduct sensitivity analyses by varying key parameters and observing the impact on the model's output, providing deeper insights into the dynamics of the economic system.

S5. Example of full ChatGPT's response

Prompt

I want your feedback on my attempt to identify variables and develop a conceptual model for including COVID-19 shock in a simplified macroeconomic model of the consumption multiplier. I have put together the key relationships for my model. Can you check the attached photo and give me your feedback? Be critical.



Initial model of consumption multiplier, provided in a ChatGPT prompt

ChatGPT's Full Response

Based on your diagram, you seem to be incorporating the COVID-19 shock into a macroeconomic model by adjusting several variables that influence aggregate demand, specifically through changes in government expenditure, investment, and the marginal propensity to consume. Here are some critical points to consider:

1. **Government Expenditure**: You indicate that government expenditure negatively affects aggregate demand, which is unconventional. Typically, increased government expenditure is a fiscal stimulus that increases aggregate demand. If you mean to show that COVID-19 has led to a decrease in other forms of government expenditure, it would be more accurate to show this as a positive relationship with a note that COVID-19 has caused a shift in the composition of expenditure.
2. **Investment**: You've marked investment with a positive relationship to aggregate demand, which is correct. However, it's important to specify the nature of the investment. During COVID-19, certain types of investment may have fallen (e.g., new commercial real estate), while others may have risen (e.g., healthcare infrastructure).
3. **Expected Income**: The link between GDP and expected income is not entirely clear in the model. Normally, GDP would positively influence expected income, not negatively. If your intention is to show that the GDP has decreased due to COVID-19, thereby lowering expected income, then the direction of the arrow is correct, but it could be made clearer.
4. **Consumption**: The relationship between expected income and consumption is correct; as expected income decreases, consumption typically does as well. However, the addition of the marginal propensity to consume (MPC) is crucial, and it appears you've depicted this relationship correctly.
5. **Marginal Propensity to Consume**: The negative relationship between COVID-19 onset and the marginal propensity to consume is appropriate, assuming that the shock of the pandemic has made consumers more likely to save rather than spend.

6. **Clarity**: While the relationships are mostly indicated, it may help to clarify the time frame and conditions under which these relationships hold. For instance, initial government responses to COVID-19 often involved increased spending, which would generally boost aggregate demand.
7. **Feedback Mechanisms**: Your model currently lacks feedback loops. For instance, consumption affects GDP, and the shock to one part of the model may cycle through and affect other parts multiple times.
8. **Exogenous and Endogenous Variables**: It's important to distinguish between what your model treats as exogenous (e.g., COVID-19 shock) and what it treats as endogenous. Make sure this is clearly stated.
9. **Causality**: Ensure that the causality in your model reflects real-world dynamics and is supported by empirical evidence.
10. **Complexity and Simplifications**: Every model simplifies reality. Be explicit about the simplifications you're making and consider whether they might affect the validity of your model's predictions.
11. **Assumptions**: State the assumptions that underpin your model. This includes assumptions about behavior, markets, and the specific mechanisms of the COVID-19 shock.
12. **Data and Calibration**: If this is more than a theoretical exercise, consider how you will calibrate your model. What data will you use? How will you account for uncertainty and potential changes in relationships due to the pandemic?
13. **Policy Implications**: Think about the implications of your model for policy. What does it suggest about effective responses to the COVID-19 economic shock?

In constructing such models, it is crucial to base your relationships on established economic theories and empirical data, and to be ready to adjust them as more information becomes available, especially in the context of an ongoing and unprecedented event like the COVID-19 pandemic.

S6. Example of iterative prompt refinement (three iterations)

Iteration 1

We started by asking a general question about developing an interface, presented below.

Prompt

Can you help me develop a user interface for the code I developed?

ChatGPT's Full Response

To develop a user interface (UI) for your code, we have several options, including creating a simple graphical user interface (GUI) using Python libraries like Tkinter, PyQt, or building a web-based interface using frameworks like Flask or Streamlit. The choice depends on your preference and the intended users of the application.

Option 1: Tkinter GUI

Tkinter is a standard GUI toolkit for Python. It is lightweight and relatively easy to use for simple applications.

- **Advantages**: Simple, no need for web servers or internet access.
- **Disadvantages**: Limited in terms of styling and advanced features.

Option 2: PyQt GUI

PyQt is another GUI framework which offers more sophisticated widgets and styling options compared to Tkinter.

- **Advantages**: More features and better styling options.
- **Disadvantages**: More complex than Tkinter and requires understanding of the Qt framework.

Option 3: Flask Web Interface

Flask is a micro web framework for Python. It is great for creating web-based interfaces.

- **Advantages**: Accessible from any device with a web browser, more modern interface possibilities.
- **Disadvantages**: Requires a web server setup for remote access.

Option 4: Streamlit

Streamlit is a newer framework specifically designed for quickly building data science and machine learning apps.

- **Advantages**: Very easy to set up, great for data visualization.
- **Disadvantages**: Less flexibility for non-standard UI elements or complex interactions.

Example of a Simple Tkinter UI

Here is a basic example of what a Tkinter UI could look like for your application:

- **Input fields** for parameters like Reference_MPC, alpha, EFT, PAT, Expected_Income0, and GDP0.
- **Buttons** to run the simulation and to load data.
- **Graph area** to display the plot generated by Matplotlib.
- **Status messages** to inform the user of the current state of the application (e.g., "Running simulation...", "Simulation completed").

Would you like to proceed with one of these options, and if so, which one? Also, do you have any specific features or elements you want to include in the UI?

Iteration 2

From the previous response, we realized that we should have considered the inputs to our model that can be changed by a user. To revise the prompt, we specified the inputs, which are the model parameters we are interested in.

Prompt

I want to create an interface with the parameters Reference Marginal Propensity to Consume (Reference MPC), Production Adjustment Time (PAT), Expectation Formation Time (EFT), and alpha as inputs for the code I developed.

ChatGPT's Full Response

Creating an interface for your code involves a few steps. Since you want to input parameters like Reference Marginal Propensity to Consume (Reference MPC), Production Adjustment Time (PAT), Expectation Formation Time (EFT), and alpha, I recommend using a Python library like tkinter for a GUI-based approach or ipywidgets if you're working within a Jupyter Notebook. Here's a basic outline for both approaches:

Using tkinter **for a Standalone Application**

- **Install tkinter**: If not already installed, you can install it using pip.
- **Create a GUI Window**: Initialize the tkinter window.
- **Add Input Fields**: Create input fields for Reference MPC, PAT, EFT, and alpha.
- **Add a Button**: To run the simulation.
- **Define the Simulation Function**: Include your existing code, modified to take inputs from the GUI.
- **Display Results**: Possibly in a new window or as a download option.

Here's a basic skeleton code:

```
import tkinter as tk
from tkinter import filedialog
import numpy as np
from scipy.integrate import odeint
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt
import pandas as pd

def run_simulation(Reference_MPC, EFT, PAT, alpha):
    # Your existing simulation code here
    # Modify to use the parameters from the GUI

def on_run_clicked():
    Reference_MPC = float(entry_ref_mpc.get())
    EFT = float(entry_eft.get())
    PAT = float(entry_pat.get())
    alpha = float(entry_alpha.get())
    run_simulation(Reference_MPC, EFT, PAT, alpha)

root = tk.Tk()
root.title("Economic Model Simulation")

tk.Label(root, text="Reference MPC").pack()
entry_ref_mpc = tk.Entry(root)
entry_ref_mpc.pack()

tk.Label(root, text="EFT").pack()
entry_eft = tk.Entry(root)
entry_eft.pack()

tk.Label(root, text="PAT").pack()
entry_pat = tk.Entry(root)
entry_pat.pack()
```

```
tk.Label(root, text="Alpha").pack()
entry_alpha = tk.Entry(root)
entry_alpha.pack()

run_button = tk.Button(root, text="Run Simulation", command=on_run_clicked)
run_button.pack()

root.mainloop()
```

Using ipywidgets **for a Jupyter Notebook**

- **Install ipywidgets**: Use '!pip install ipywidgets in a Jupyter cell.
- **Create Widgets**: For input fields and a button.
- **Display Widgets**: Arrange them using layout options.
- **Link Function**: Link your simulation function to the button click event.

Here's a basic example:

```
import ipywidgets as widgets
from IPython.display import display
import numpy as np
from scipy.integrate import odeint
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt
import pandas as pd

def run_simulation(Reference_MPC, EFT, PAT, alpha):
    # Your existing simulation code here
    # Modify to use the parameters from the GUI

ref_mpc_widget = widgets.FloatText(description='Reference MPC')
eft_widget = widgets.FloatText(description='EFT')
pat_widget = widgets.FloatText(description='PAT')
alpha_widget = widgets.FloatText(description='Alpha')
run_button = widgets.Button(description='Run Simulation')

def on_run_clicked(b):
    run_simulation(ref_mpc_widget.value, eft_widget.value, pat_widget.value, alpha_widget.value)

run_button.on_click(on_run_clicked)

display(ref_mpc_widget, eft_widget, pat_widget, alpha_widget, run_button)
```

Remember, this is just a basic framework. You can expand upon this by adding more features like error handling, dynamic updating of graphs, etc. Also, make sure to adapt the simulation function to work with the GUI components.

Iteration 3

We realized that we could have specified the changes in the model to be considered as sliders where the user can change to see the updates in the model. We also included the details of the graph that we wanted to include in the interface.

Prompt

I'm interested in having a simple interactive interface for my model. I want to have four sliders to change the values for Reference Marginal Propensity to Consume (Reference MPC), Production Adjustment Time (PAT), Expectation Formation Time (EFT), and alpha, and the graph that be updated when I change the sliders. The graph should show Simulated GDP and Historical GDP. Of course, when I change the sliders, the Simulated GDP will be updated accordingly. Can you guide me on how to create this interface?

ChatGPT's Full Response

Creating an interactive interface with sliders to adjust parameters in your model can be achieved using Jupyter widgets. These widgets allow interactive controls in a Jupyter notebook. Here's a step-by-step guide on how to create this interface:

1. **Import Necessary Libraries**: You need to import **ipywidgets** for creating sliders and **IPython.display** for dynamic display.
2. **Define a Function to Update the Graph**: Create a function that takes the slider values as inputs, runs the model with these parameters, and plots the graph.
3. **Create Sliders**: Use **ipywidgets** to create sliders for each parameter you want to control (Reference MPC, PAT, EFT, and alpha).
4. **Display the Sliders and Graph**: Use **ipywidgets.interactive** to link the sliders with the graph updating function.
5. **Optional Styling**: You can adjust the layout and styling of the sliders and graph display according to your preference.

Here is an example code to illustrate these steps:

```
import ipywidgets as widgets
from IPython.display import display, clear_output
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint
import pandas as pd
# Other imports already in your code...

# Assuming your model and other code components are correctly defined as above...

# Step 2: Define a function to update the graph
def update_graph(Reference_MPC, EFT, PAT, alpha):
    results = odeint(model, y0, t, args=(Reference_MPC, EFT, PAT, alpha))
    Expected_Income_results, GDP_results = results.T

    # Plotting
    plt.figure(figsize=(10, 6))
    plt.plot(t, GDP_results, label="Simulated GDP")
    plt.plot(t, GDP_Data(t), label="GDP Data", linestyle="--")
    plt.xticks(np.arange(2013, 2024, 1), rotation=45)
    plt.title("U.S. Simulated GDP vs. Historical Data")
    plt.xlabel("Year")
    plt.ylabel("Billions of Dollars")
    plt.legend()
    plt.grid(True)
    plt.show()

# Step 3: Create sliders
slider_Reference_MPC = widgets.FloatSlider(value=0.84, min=0, max=1, step=0.01,
description='Reference MPC:')
slider_PAT = widgets.FloatSlider(value=0.27, min=0, max=1, step=0.01, description='PAT:')
slider_EFT = widgets.FloatSlider(value=4.39, min=0, max=10, step=0.01, description='EFT:')
slider_alpha = widgets.FloatSlider(value=0.28, min=0, max=1, step=0.01, description='Alpha:')

# Step 4: Display the Sliders and Graph
ui = widgets.VBox([slider_Reference_MPC, slider_PAT, slider_EFT, slider_alpha])
out = widgets.interactive_output(update_graph, {'Reference_MPC': slider_Reference_MPC, 'EFT':
slider_EFT, 'PAT': slider_PAT, 'alpha': slider_alpha})

display(ui, out)
```

Note: This code assumes you're running it in a Jupyter notebook environment. The sliders will allow you to dynamically change the parameters and the graph will update accordingly. Remember to import your data and define the model function as in your original code.