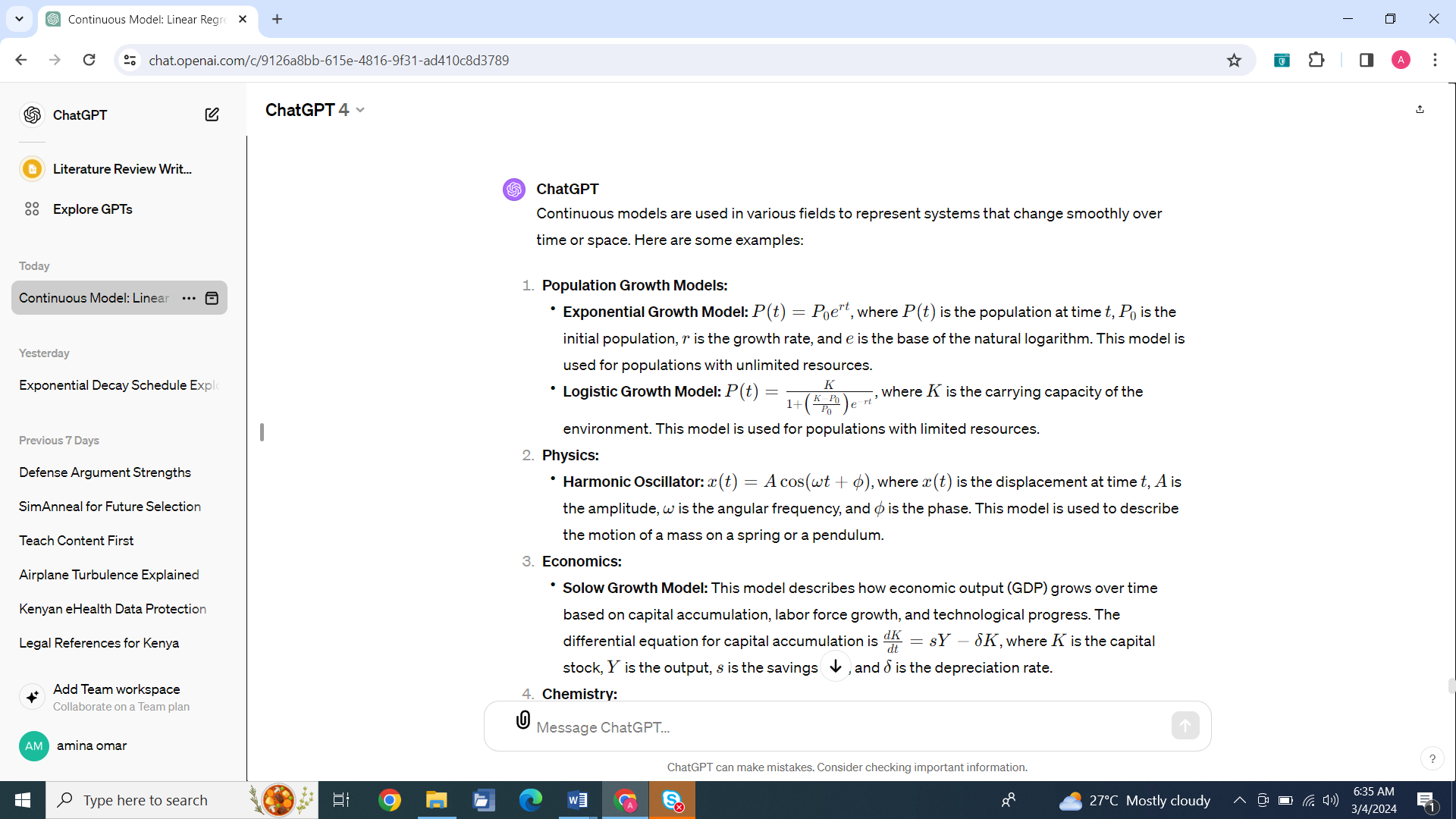
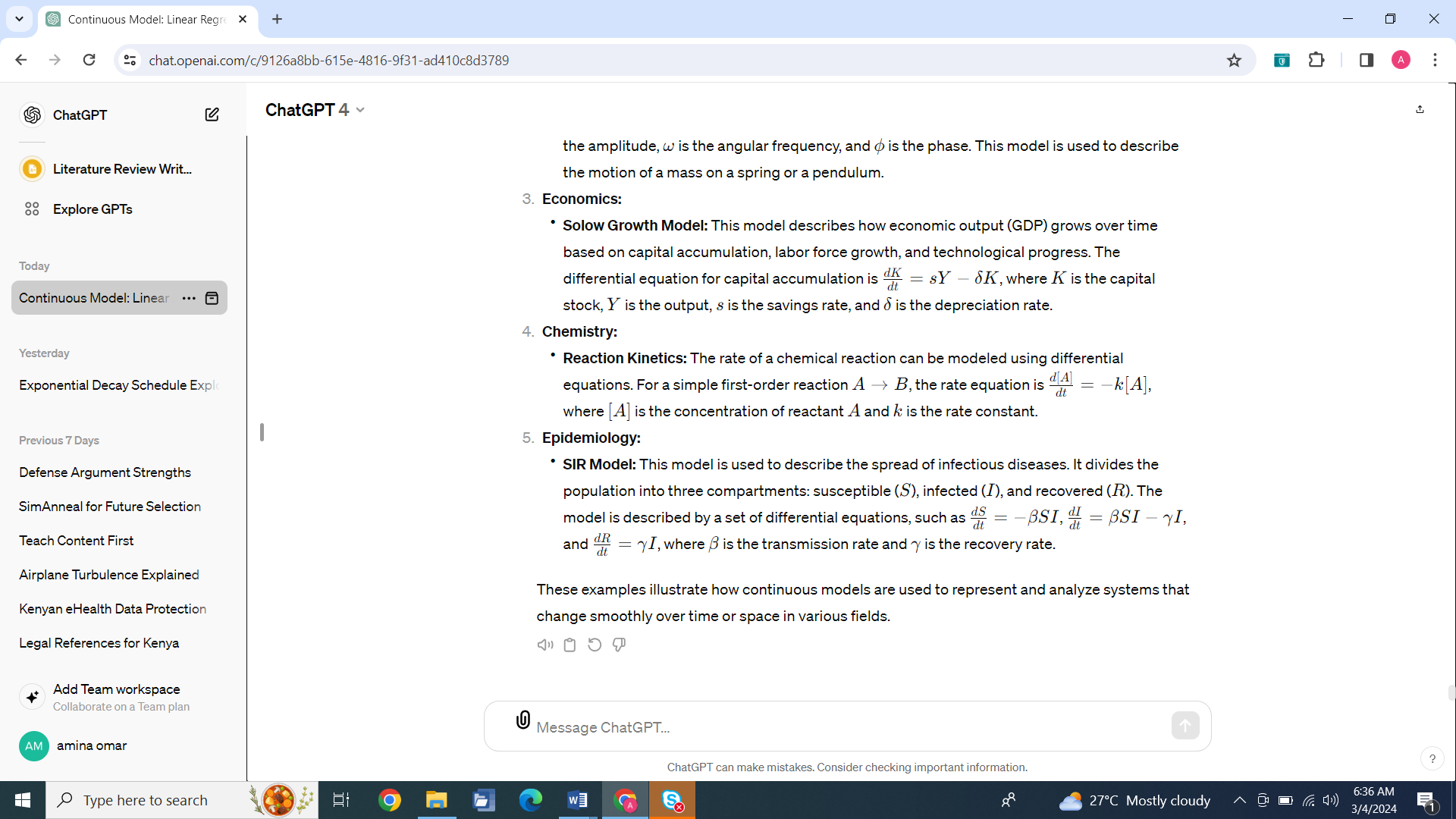
**Continuous Event Models**

Continuous event modeling refers to the mathematical or computational representation of events or processes that occur continuously over time. This type of modeling is often used in fields such as physics, engineering, finance, and environmental science to study and predict the behavior of dynamic systems. Continuous event models are typically represented using **differential equations or stochastic processes.**

Here are a few examples of continuous event modeling:

Continuous models are used in various fields to represent systems that change smoothly over time or space. Here are some examples:





**SYSTEM CLASS**

We now consider some classes of continuous systems, according to the mathematical methods, and therefore the numerical techniques, associated with modeling their members. Table 4.2 presents a few common examples. Very often, simulation or evaluation of continuous physical systems boils down to the integration of a first - or second-order differential equation (ordinary or partial). A very large fraction of continuous systems is governed by second-order differential equations, which is due to something of a coincidence, that several different essential equations in physics are of second-order. For this chapter, we will focus on the first three items in the table, which all involve ordinary differential equations of first or second order.

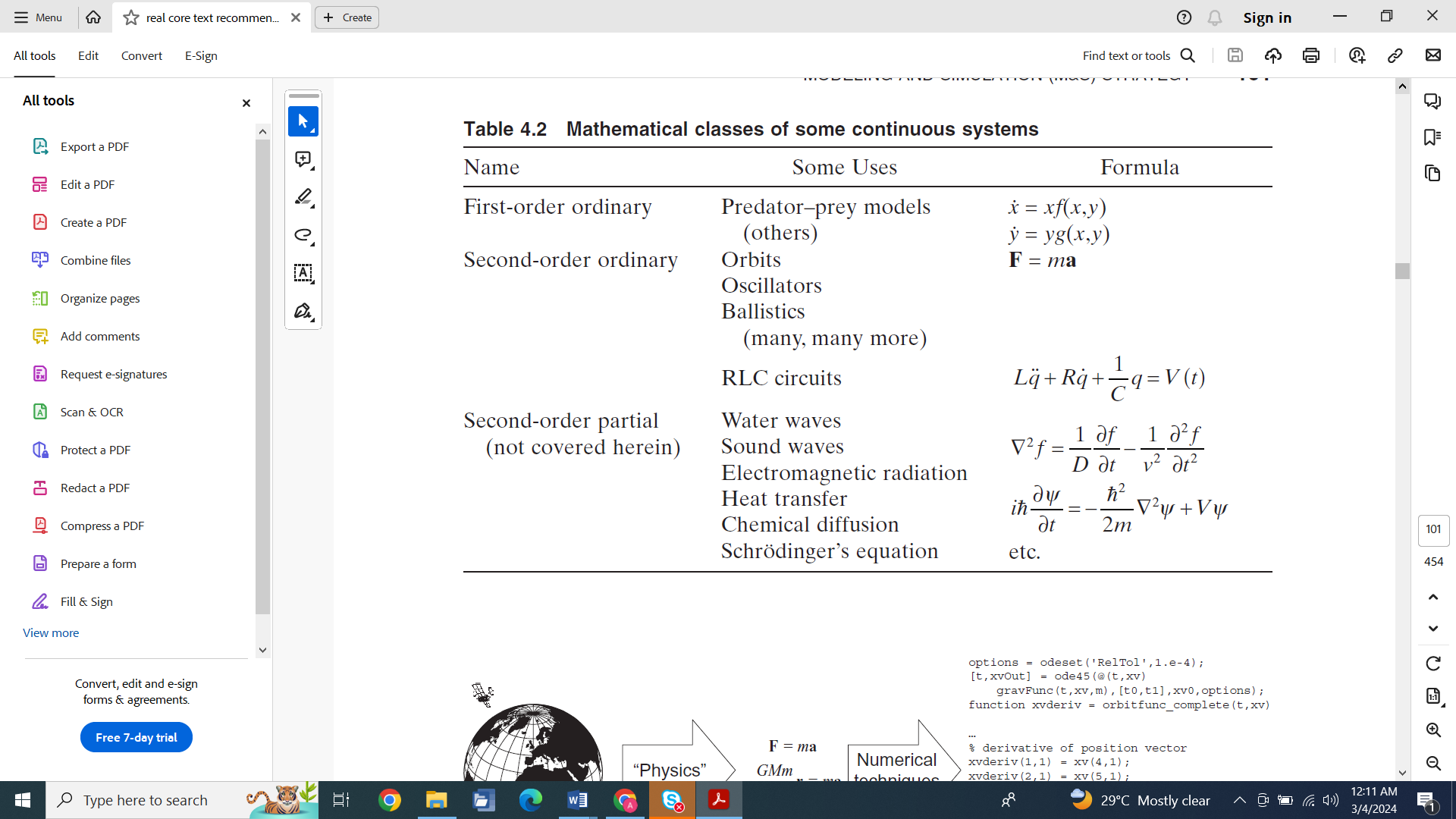
In practice, exact analytical solutions to differential equations are not always possible, especially for complex systems. Instead, numerical methods are used to approximate the solutions. These methods involve **discretizing the time domain** and iteratively solving the equations to obtain an approximate solution at each time step. Common numerical methods for solving differential equations include Euler's method, Runge-Kutta method, and finite difference methods.

To discretize in the context of mathematics and numerical analysis means to convert continuous functions, models, or equations into discrete counterparts. This process involves breaking up continuous intervals or ranges into finite, countable sets of points or intervals.

For example, when discretizing time for numerical methods like Euler's Method, you would:

* Divide a continuous time interval into a finite number of small, discrete time steps.
* Approximate continuous change over each small-time step with a simple, discrete change.

Discretization allows for the numerical approximation of solutions to continuous problems using algorithms that operate on discrete sets of data points. It is an essential step in numerical simulation because computers can only handle discrete quantities.



Digital computers cannot, in general, represent continuous (real) numbers, so how can we possibly use computers to simulate continuous systems? Mathematicians define the degree of continuity by the ability to take a derivative, but when we look at Leibniz’s formula (Eq. (4.1)) for computing the derivative,

**MODELING AND SIMULATION (M & S) STRATEGY**

The basic M & S strategy for modeling continuous systems is dictated by two key steps: understanding the physics of the system and identifying the appropriate numerical implementation for solving the resulting equations (see Fig. 4.1). We use the term “physics ” very broadly, to mean translation of a real-world system into a set of quantitative mathematical equations, a process that is the very essence of physics, in particular. Of course, this is not to say that we may not use equations from biology, chemistry, economics, sociology, or other sciences, as applicable to the system at hand.

**MODELING APPROACH**

The “physics” part of the problem is the modeling phase. There is no way around developing a great enough understanding of the science of the problem to pose it as a set of mathematical equations. This is a fundamental and repeating challenge to the modeler over a career — on project after project, the modeler is presented with new problems for which he or she must provide credible models with perhaps only cursory (or even no) training in the relevant scientific fields. This is why successful modelers often call themselves “generalists.” Remember, the modeler is not trying to make breakthrough discoveries in biochemistry or economics but is trying to use the relevant knowledge base from such fields to provide accurate models in a particular context. Fortunately for modelers, reliable sources of such knowledge are more accessible than ever.

**State Variables**

The usual assumption in continuous modeling, and for that matter, much of science, is that a system can be defined, at any particular instant, by its state. This state is, in turn, defined by a suite of state variables. We regard that the state variables of the real-world system evolve in time continuously (hence the name of the chapter), and we further regard that almost always, the state variables themselves are continuous, meaning real or complex rather than integer-valued.

Common examples of state variables are position, velocity, mass, rate, angle, fraction, current, and voltage. All of these quantities are real (or possibly complex) and evolve continuously with time. Perhaps most important in the selection of state variables is the notion of completeness. The selected state variables must contain sufficient information to elicit all the relevant phenomena associated with the system with adequate accuracy and precision, as set forth during the requirements definition. Identifying all of the correct (i.e., necessary and sufficient) state variables is critical.

Just as a simple example, consider the problem of how long it takes an object on earth’s surface to drop 10 ft through the air (under “normal” conditions) after starting at rest. If the object is an anvil, the only two state variables necessary to obtain an excellent result are position and speed. If, however, the object is a big-leaf maple seed (“whirligig” type seed), which generates lift by spinning, we must know about its rotation rate to estimate its lift. Leaving out rotation would give completely invalid results.

**State Variable Equations**

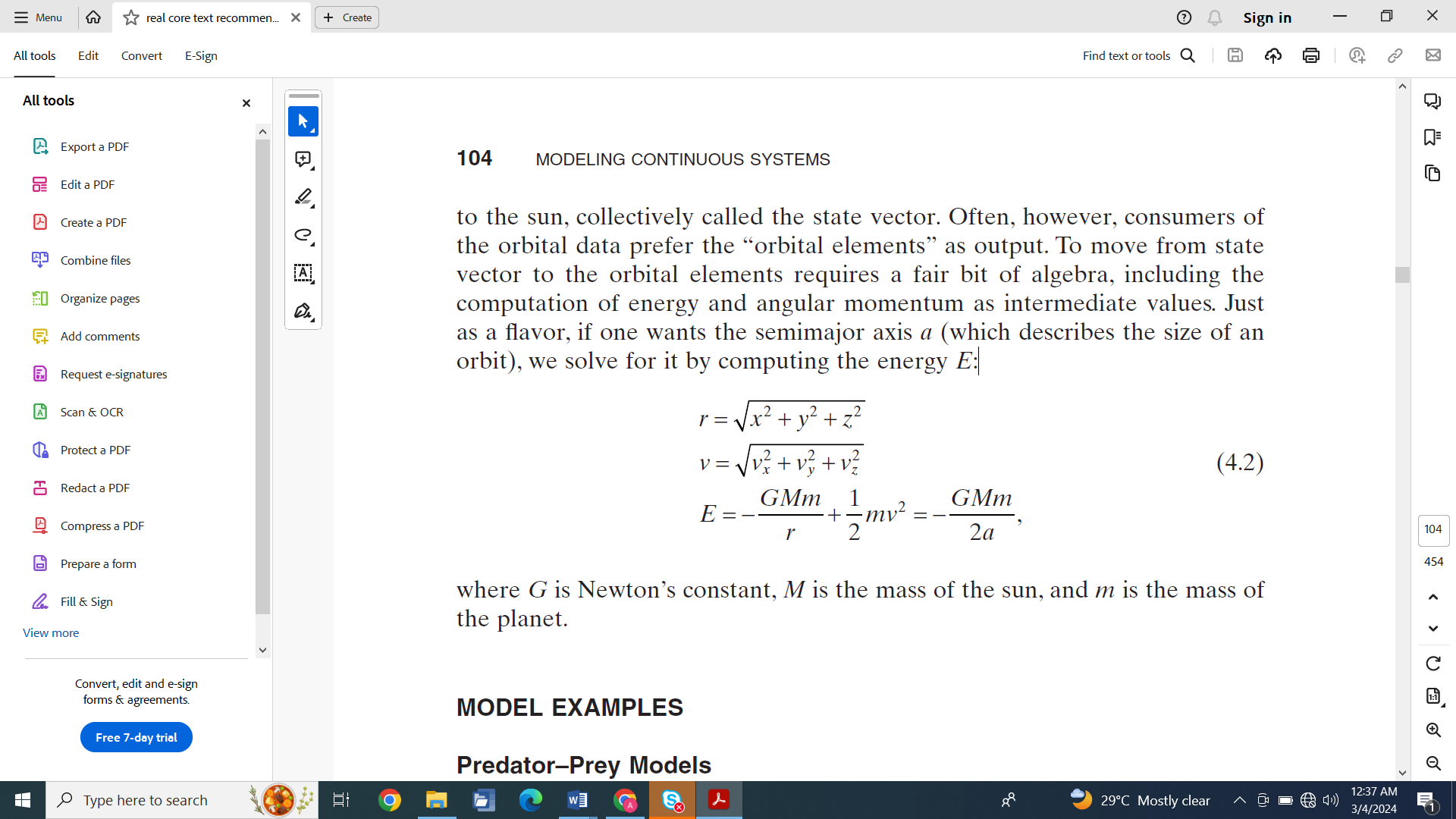
A task closely interrelated with identification of the state variables is identification of the equations that govern them. These equations dictate how the state of the system changes with respect to the independent variable (for the purposes of this chapter, time). These equations may be akin to any of those seen in Table 4.1, or may be completely different, depending on the model selected. There is no fundamental limit to the form the state variable equations may take, but fortunately, as we have previously discussed, most real - world systems obey first - or second - order differential equations one way or the other. Quite often, auxiliary variables must be computed during the computation of the state variable equations, and these variables, with their associated equations, must be treated with equal care to the state variables and state variable equations themselves.

**Output Equations**

Finally, we must consider what the relevant outputs are: what quantities does the consumer of our simulation want or need? These issues are usually addressed during the requirements definition phase of development. In the simple examples given in this chapter, the output variables we report are state variables themselves, and sometimes that is the case with other models. However, much of the time, the desired outputs must be computed from the state variables using a new set of equations, called output equations.

Output equations often require the same level of expertise and rigor as the state variable equations and, as such, should always be treated with appropriate care.

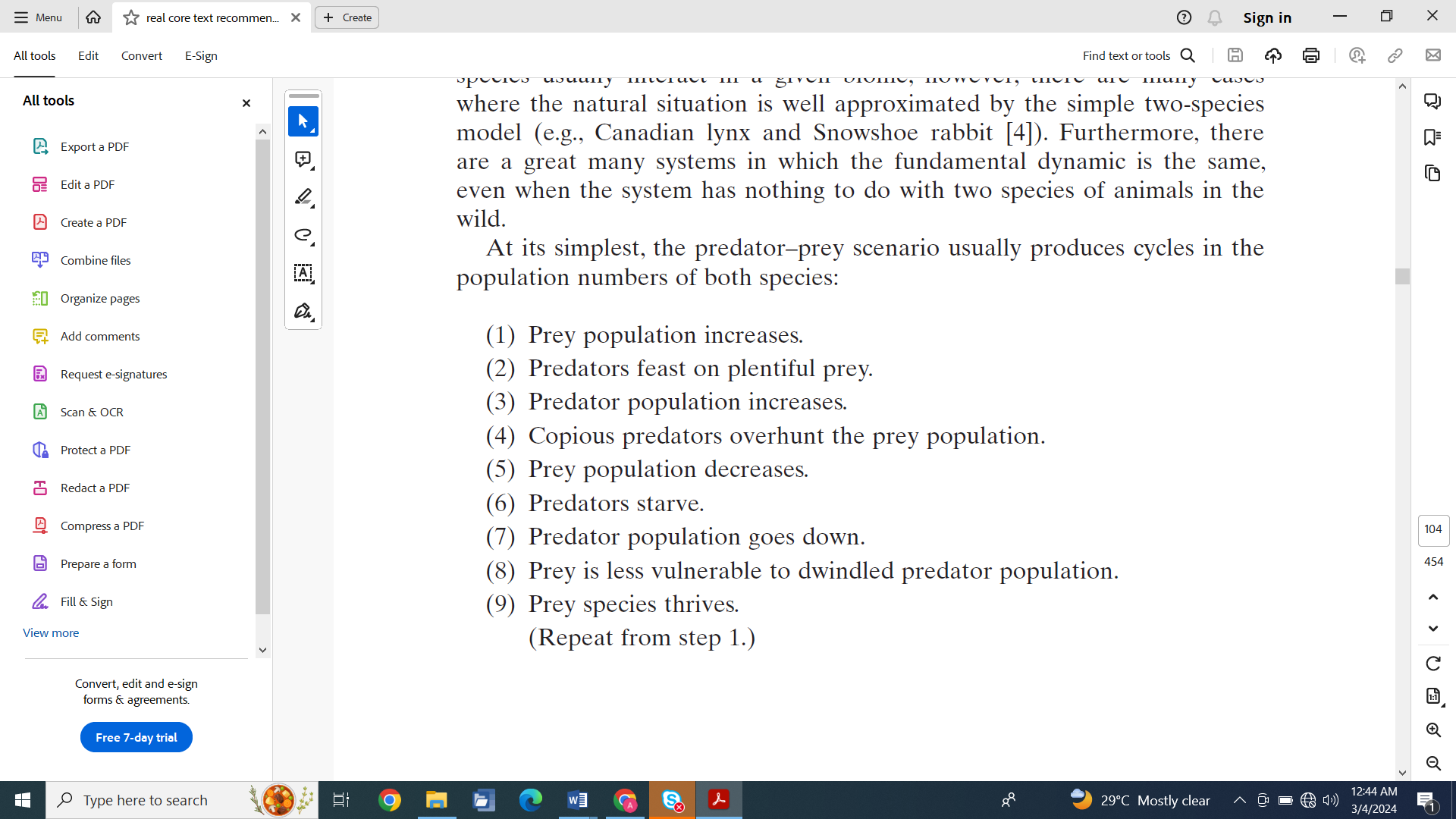
As an example, let us consider planetary orbits. The state variables at hand are simply the (three - dimensional) position and velocity of the planet relative to the sun, collectively called the state vector. Often, however, consumers of the orbital data prefer the “orbital elements” as output. To move from state vector to the orbital elements requires a fair bit of algebra, including the computation of energy and angular momentum as intermediate values. Just as a flavor, if one wants the semimajor axis a (which describes the size of an orbit), we solve for it by computing the energy E :

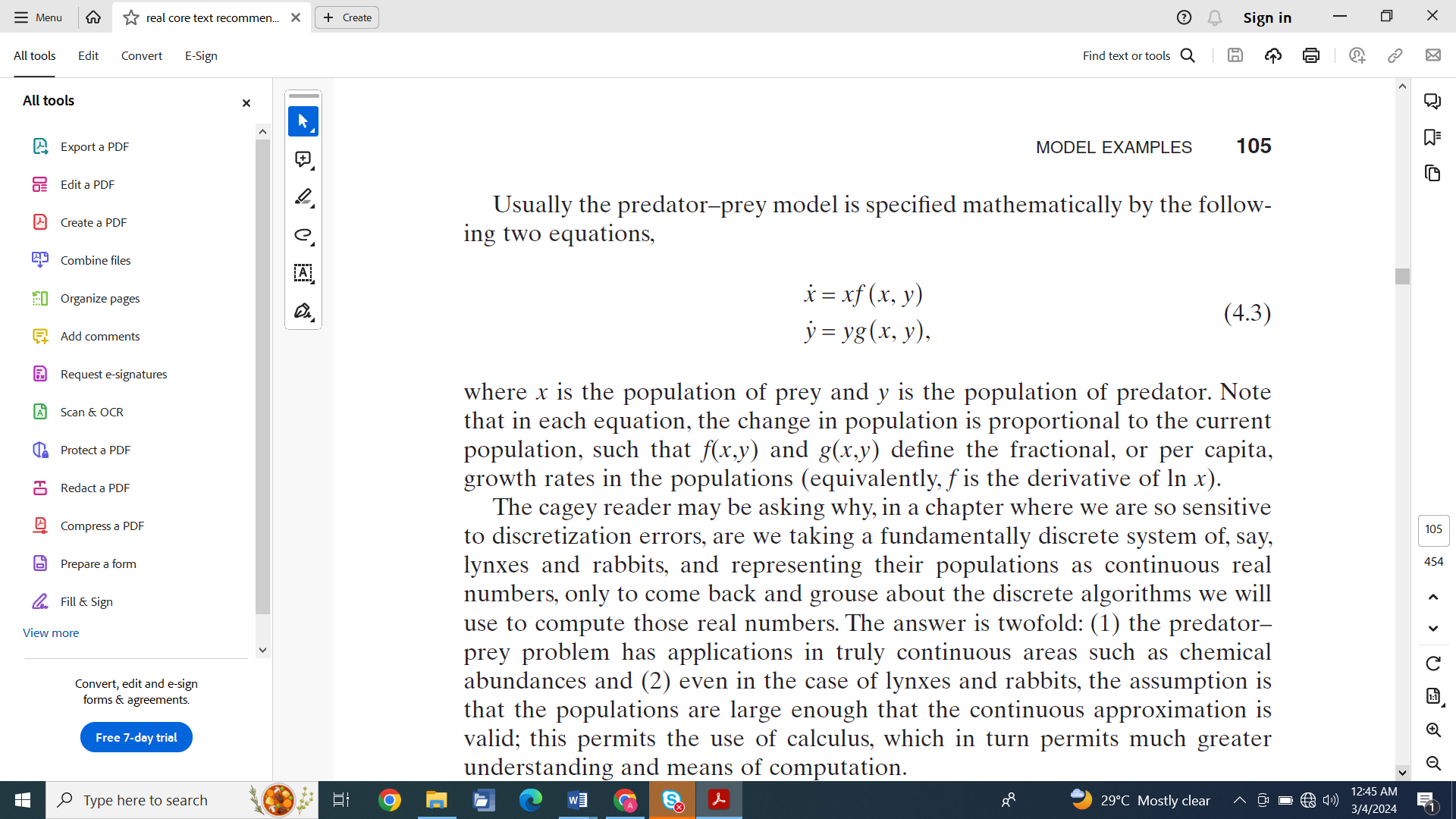


**MODEL EXAMPLES**

**Predator-Prey Models**

Predator–prey models are intended to model two species whose populations are very closely dependent on each other, much in the way we might expect “real” predator–prey systems to work in the wild. Perfectly isolated two - species predator–prey situations are fairly rare in the wild since so many species usually interact in a given biome; however, there are many cases where the natural situation is well approximated by the simple two–species model (e.g., Canadian lynx and Snowshoe rabbit [4]). Furthermore, there are a great many systems in which the fundamental dynamic is the same, even when the system has nothing to do with two species of animals in the wild. At its simplest, the predator-prey scenario usually produces cycles in the population numbers of both species:



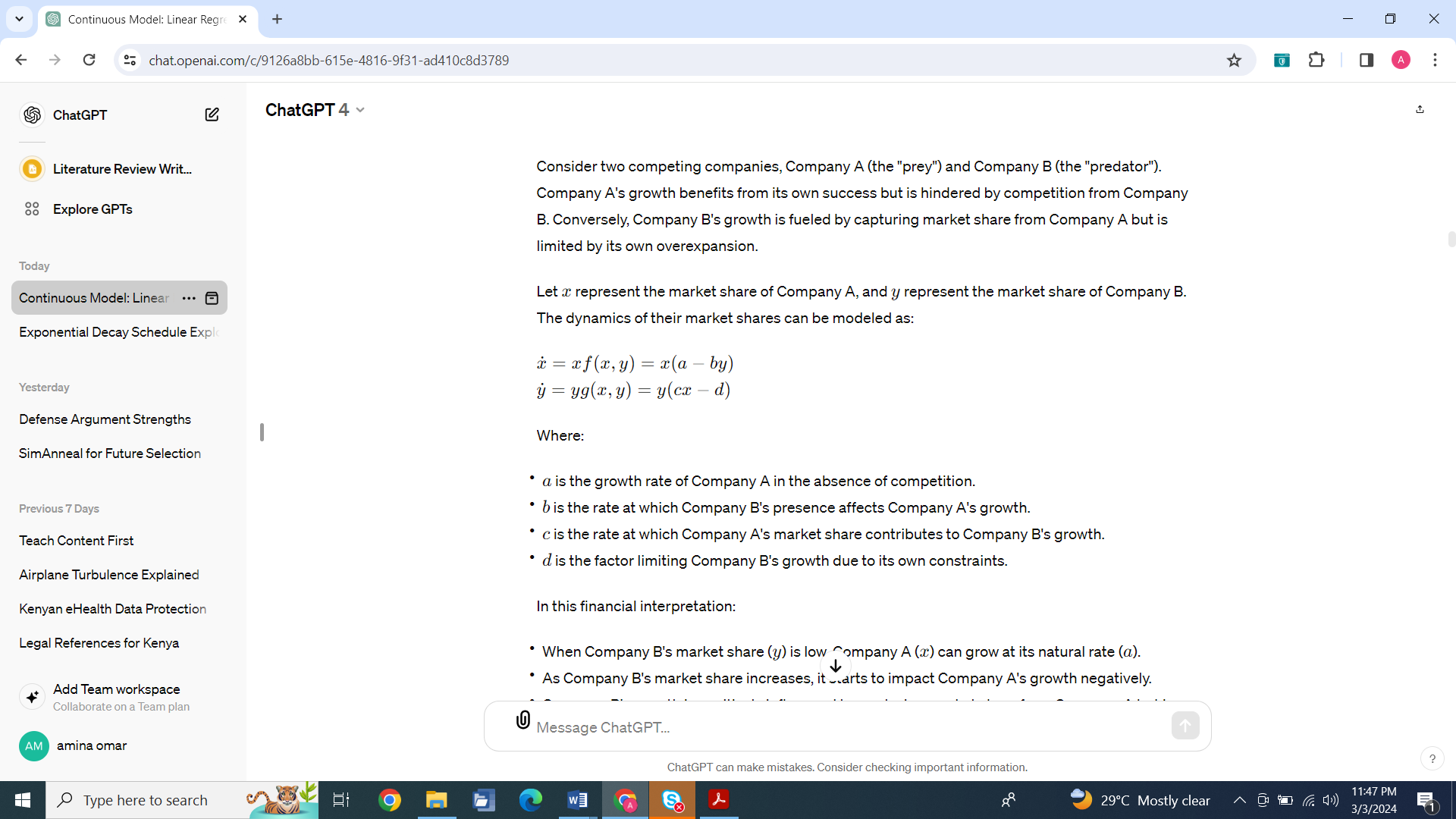


**The Predator-Prey model:**

The Predator-Prey model, also known as the Lotka-Volterra model, is primarily used in ecology to describe the dynamics between predator and prey populations. However, in finance, we can draw an analogy to this model to describe the interaction between two competing financial entities, such as two companies in the same industry or two trading strategies in the market.

Here's a simplified example using the Predator-Prey model in finance:

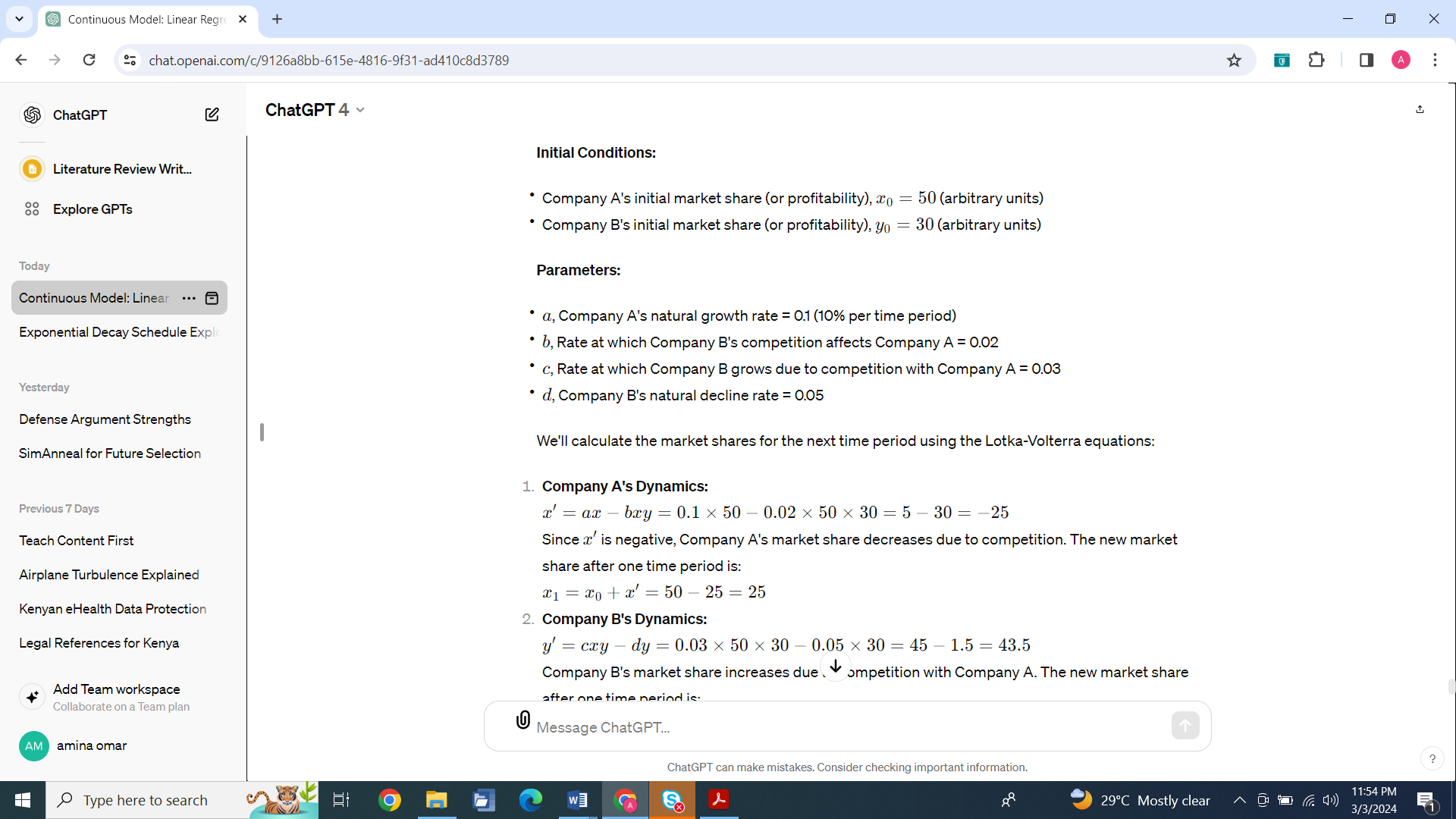
Consider two competing companies, Company A (the "prey") and Company B (the "predator"). Company A's growth benefits from its own success but is hindered by competition from Company B. Conversely, Company B's growth is fueled by capturing market share from Company A but is limited by its own overexpansion.

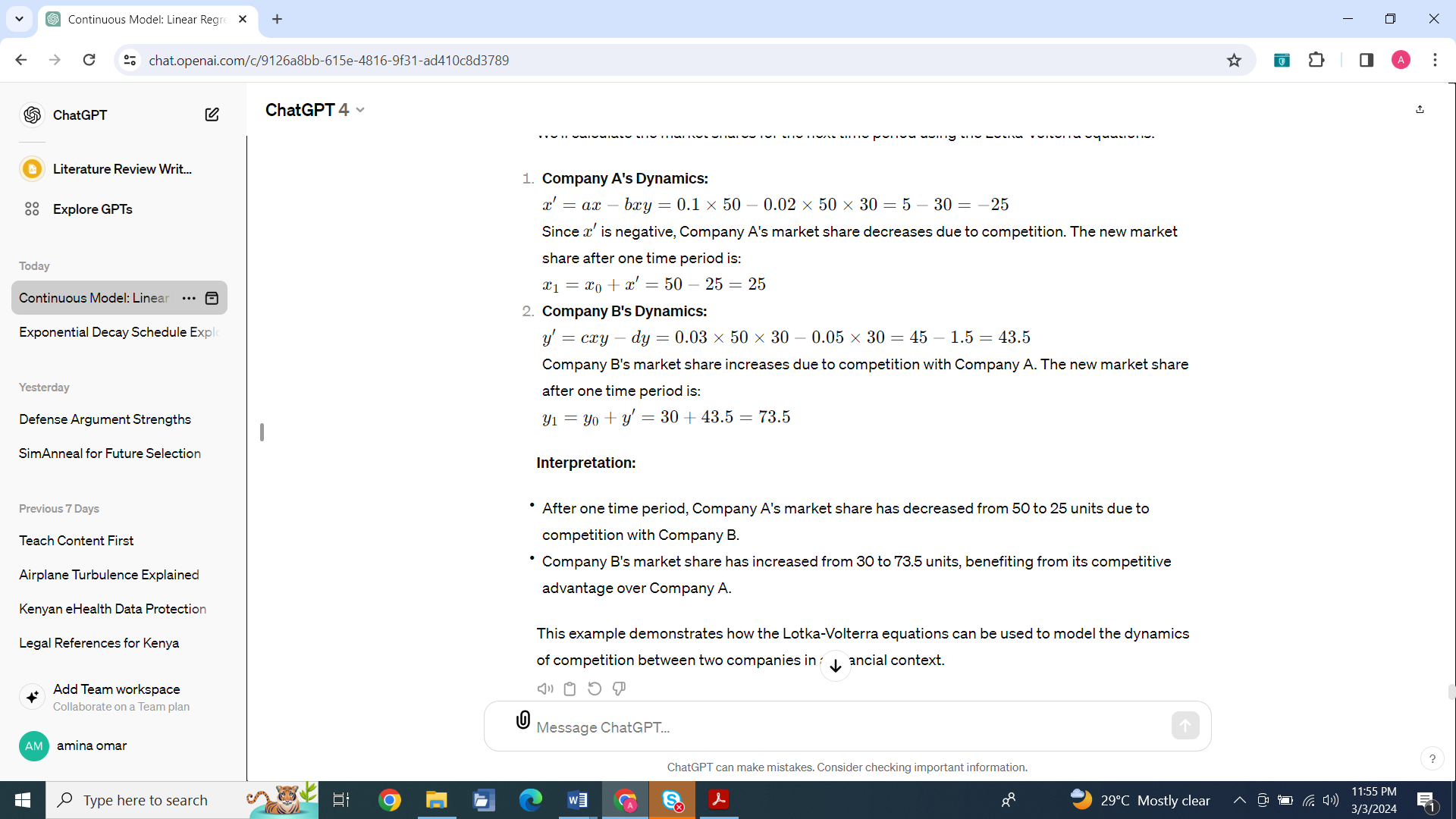


In this financial interpretation:

* When Company B's market share (*y*) is low, Company A (*x*) can grow at its natural rate (*a*).
* As Company B's market share increases, it starts to impact Company A's growth negatively.
* Company B's growth is positively influenced by capturing market share from Company A but is eventually limited by its own constraints (*d*).

This model can be used to simulate and analyze the competitive dynamics between the two companies and to study the impact of different strategies on their market shares over time.





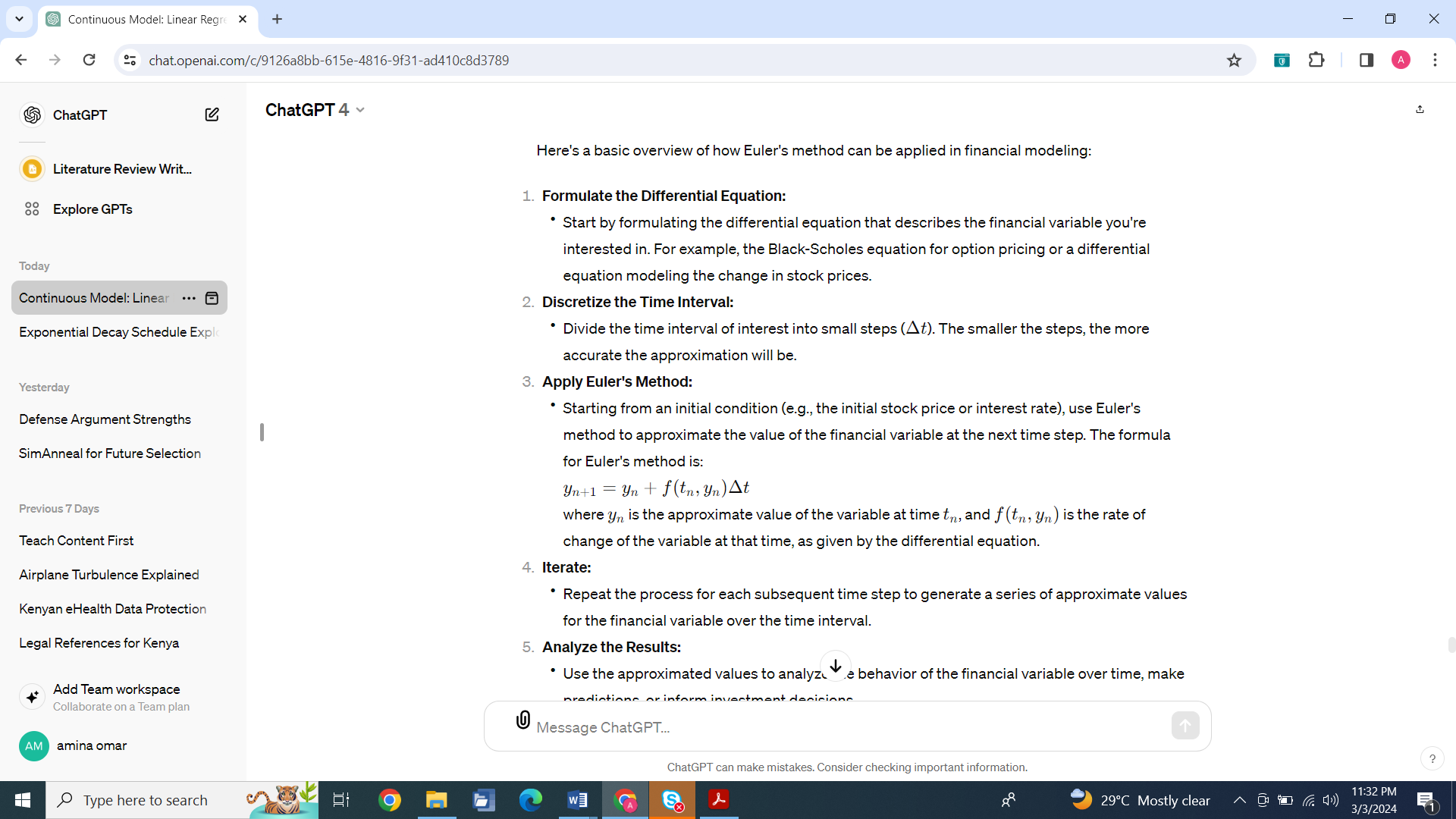
**Modeling stock prices**

Euler's method is a numerical technique used to solve ordinary differential equations (ODEs), which can be applied in financial modeling to simulate the behavior of financial variables over time. In finance, differential equations are often used to model the dynamics of interest rates, stock prices, option pricing, and other financial instruments.

In practice, exact analytical solutions to differential equations are not always possible, especially for complex systems. Instead, numerical methods are used to approximate the solutions. These methods involve discretizing the time domain and iteratively solving the equations to obtain an approximate solution at each time step. Common numerical methods for solving differential equations include Euler's method, Runge-Kutta method, and finite difference methods.

Here's a basic overview of how Euler's method can be applied in financial modeling:

1. **Formulate the Differential Equation:**
   * Start by formulating the differential equation that describes the financial variable you're interested in. For example, the Black-Scholes equation for option pricing or a differential equation modeling the change in stock prices.
2. **Discretize the Time Interval:**
   * Divide the time interval of interest into small steps (Δ*t*). The smaller the steps, the more accurate the approximation will be.
3. **Apply Euler's Method:**



1. **Iterate:**
   * Repeat the process for each subsequent time step to generate a series of approximate values for the financial variable over the time interval.
2. **Analyze the Results:**
   * Use the approximated values to analyze the behavior of the financial variable over time, make predictions, or inform investment decisions.

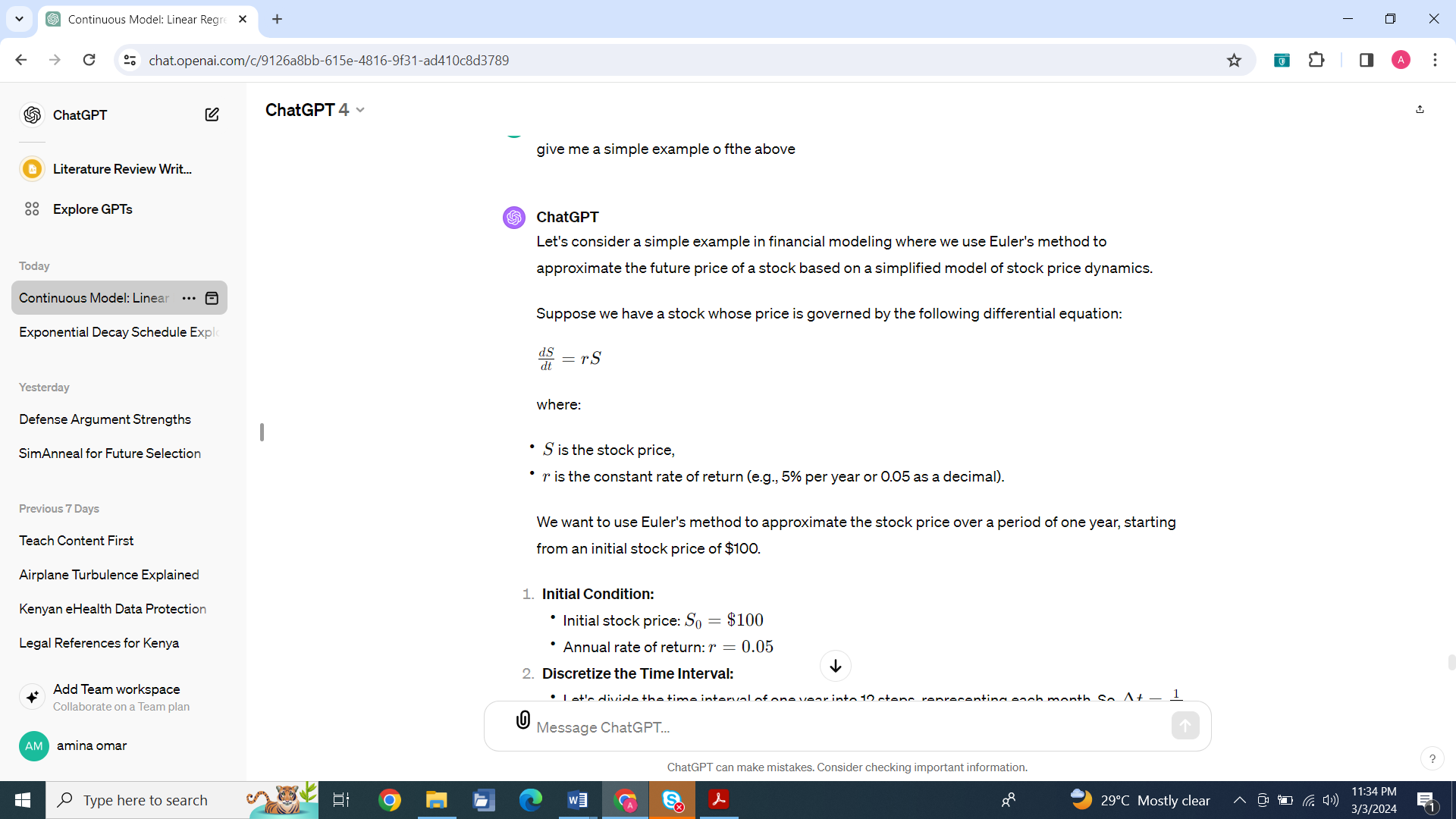
For example, in option pricing, Euler's method can be used to numerically solve the Black-Scholes partial differential equation to approximate the price of an option over time. Similarly, in interest rate modeling, Euler's method can be used to simulate the path of interest rates based on stochastic differential equations.

It's important to note that while Euler's method is simple and easy to implement, it is a first-order method and may not be as accurate as higher-order numerical methods, especially for problems with rapidly changing solutions or stiff equations. In financial

modeling, where accuracy is crucial, it's often necessary to use more sophisticated numerical methods or to refine the discretization to achieve the desired level of precision.

Let's consider a simple example in financial modeling where we use Euler's method to approximate the future price of a stock based on a simplified model of stock price dynamics.

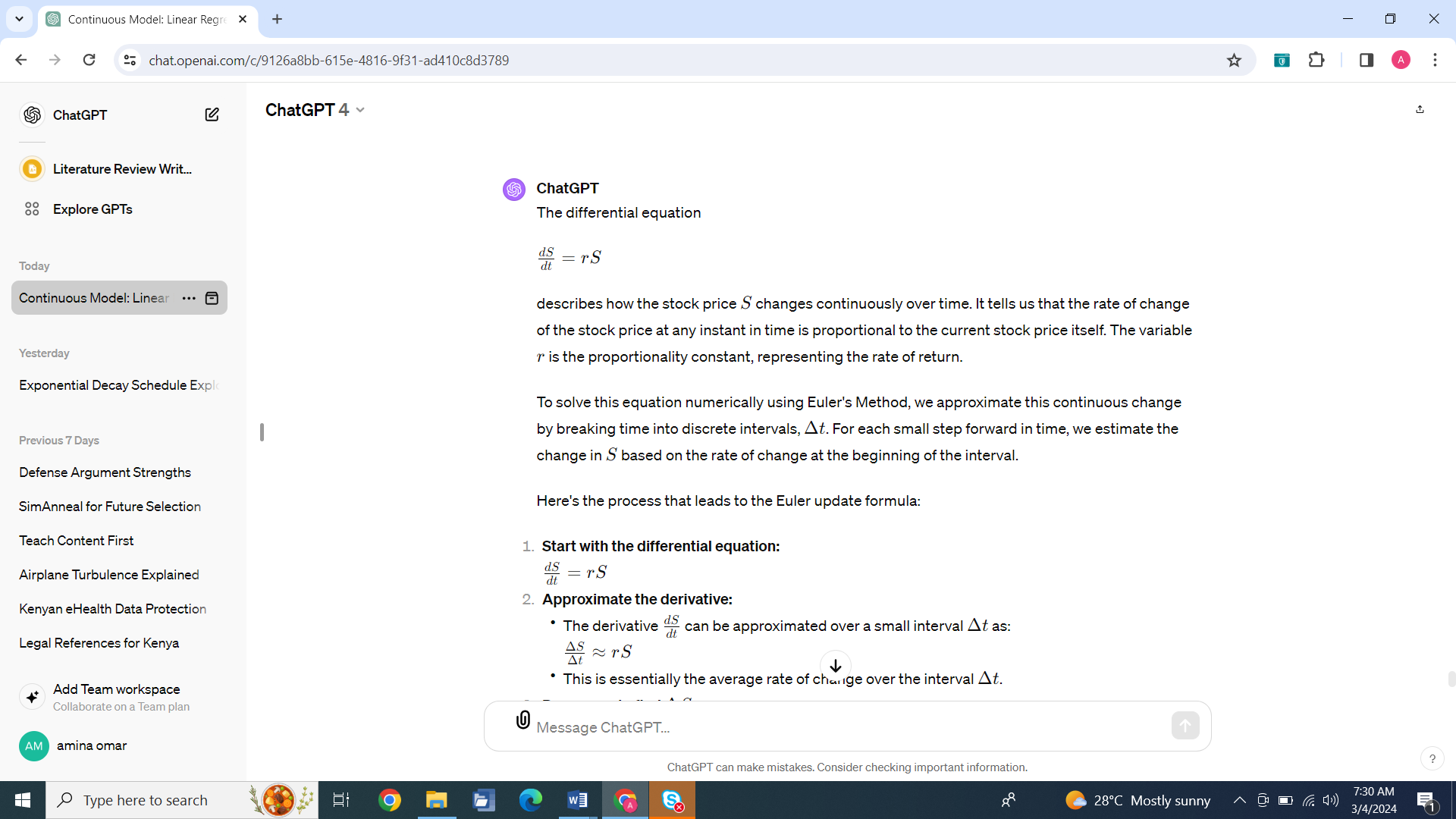
Suppose we have a stock whose price is governed by the following differential equation:

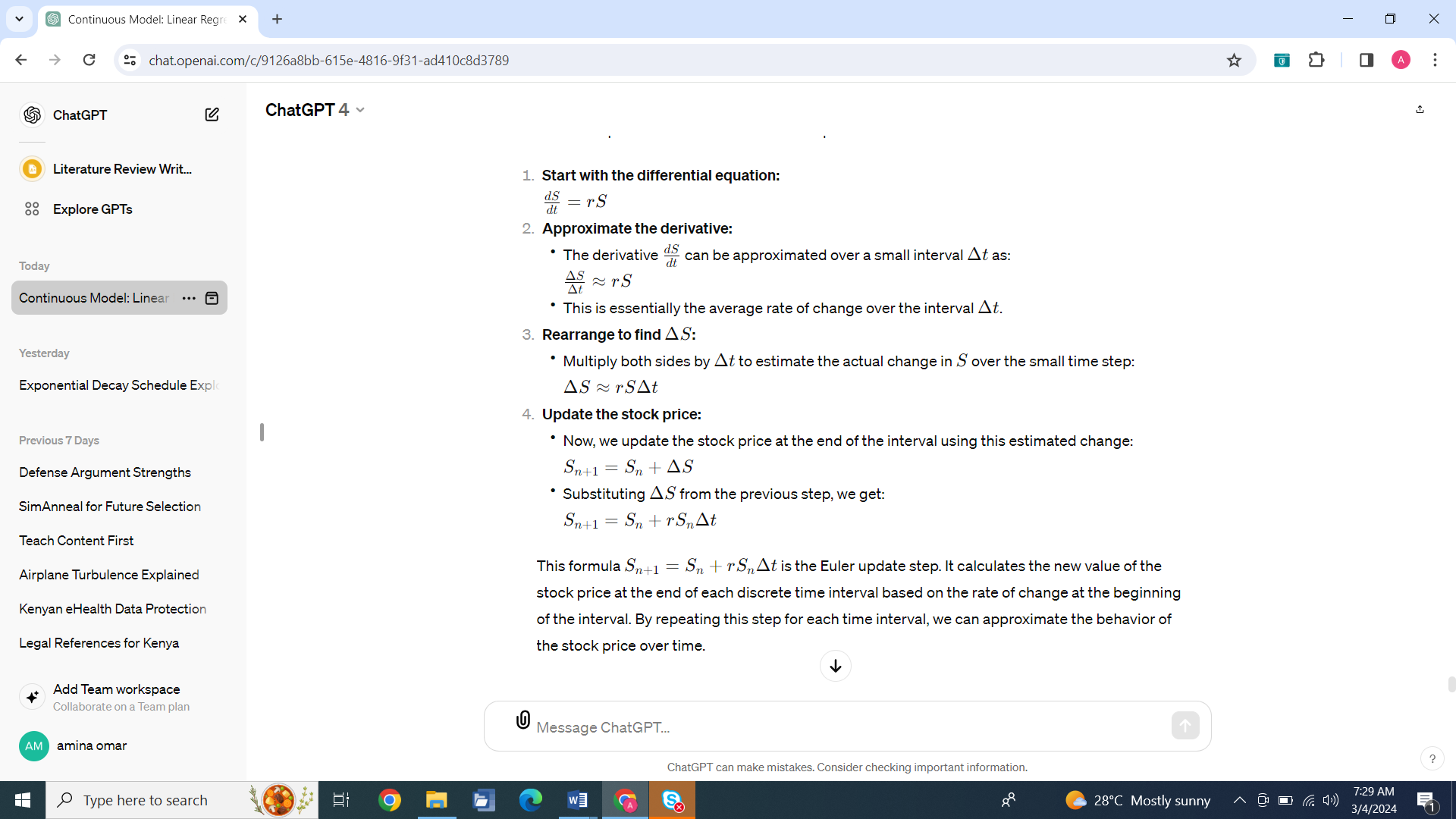


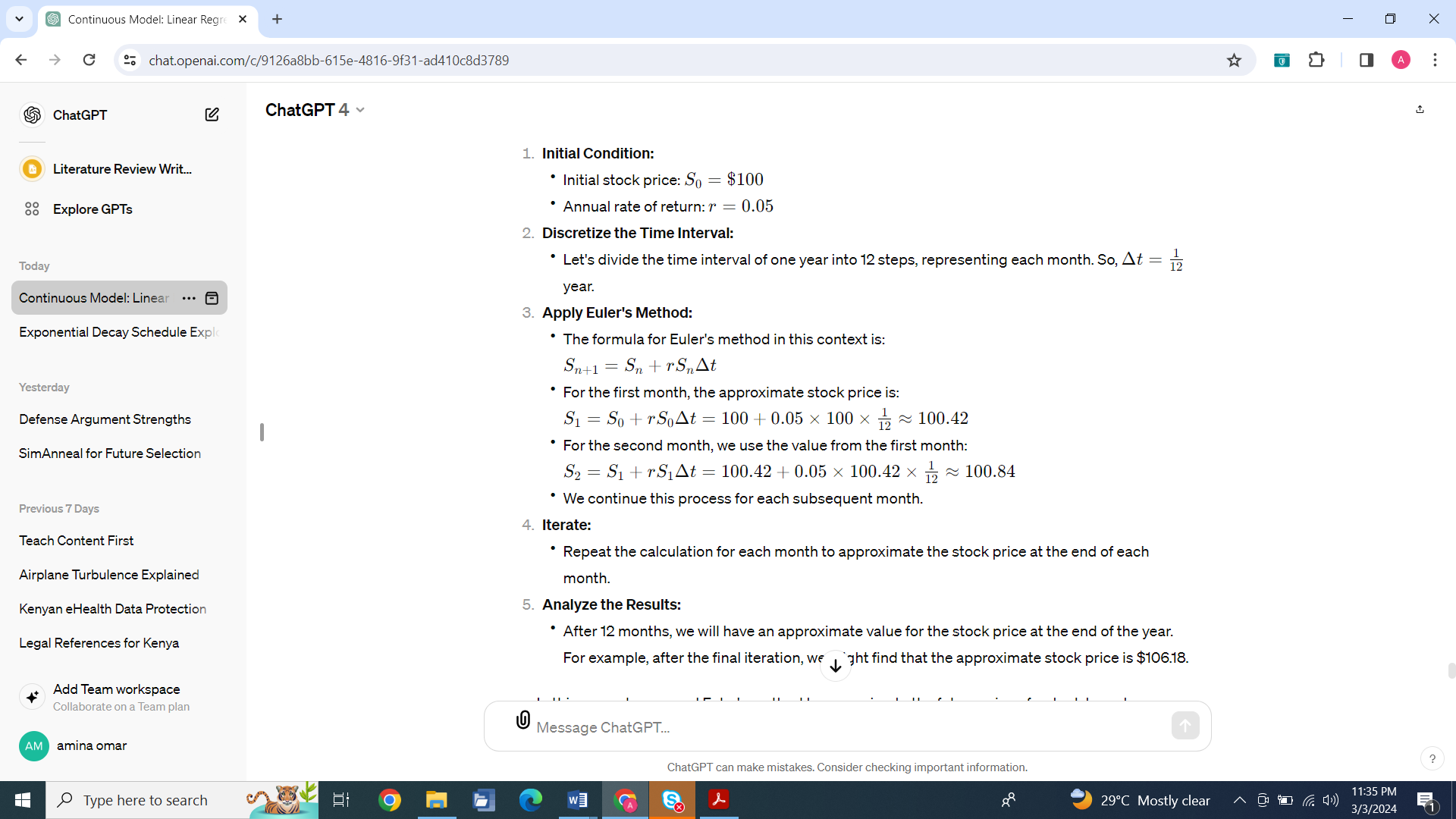
where:

* *S* is the stock price,
* *r* is the constant rate of return (e.g., 5% per year or 0.05 as a decimal).

We want to use Euler's method to approximate the stock price over a period of one year, starting from an initial stock price of $100.







In this example, we used Euler's method to approximate the future price of a stock based on a constant rate of return. This is a simplified example, and in real-world financial modeling, the dynamics of stock prices are often more complex and may involve stochastic processes and other factors. Nonetheless, Euler's method provides a basic numerical tool for approximating solutions to differential equations in finance.

To implement Euler's method in MATLAB for the scenario described in the image, where we approximate the future stock price based on a constant rate of return, you would write a script similar to the following:

% Initial condition

S0 = 100; % Initial stock price

r = 0.05; % Annual rate of return

% Discretize the time interval

N = 12; % Number of steps (one for each month)

delta\_t = 1 / N; % Size of each time step in years

% Initialize the stock price array

S = zeros(1, N+1); % Pre-allocate space for efficiency

S(1) = S0; % Set initial stock price

% Apply Euler's method

for n = 1:N

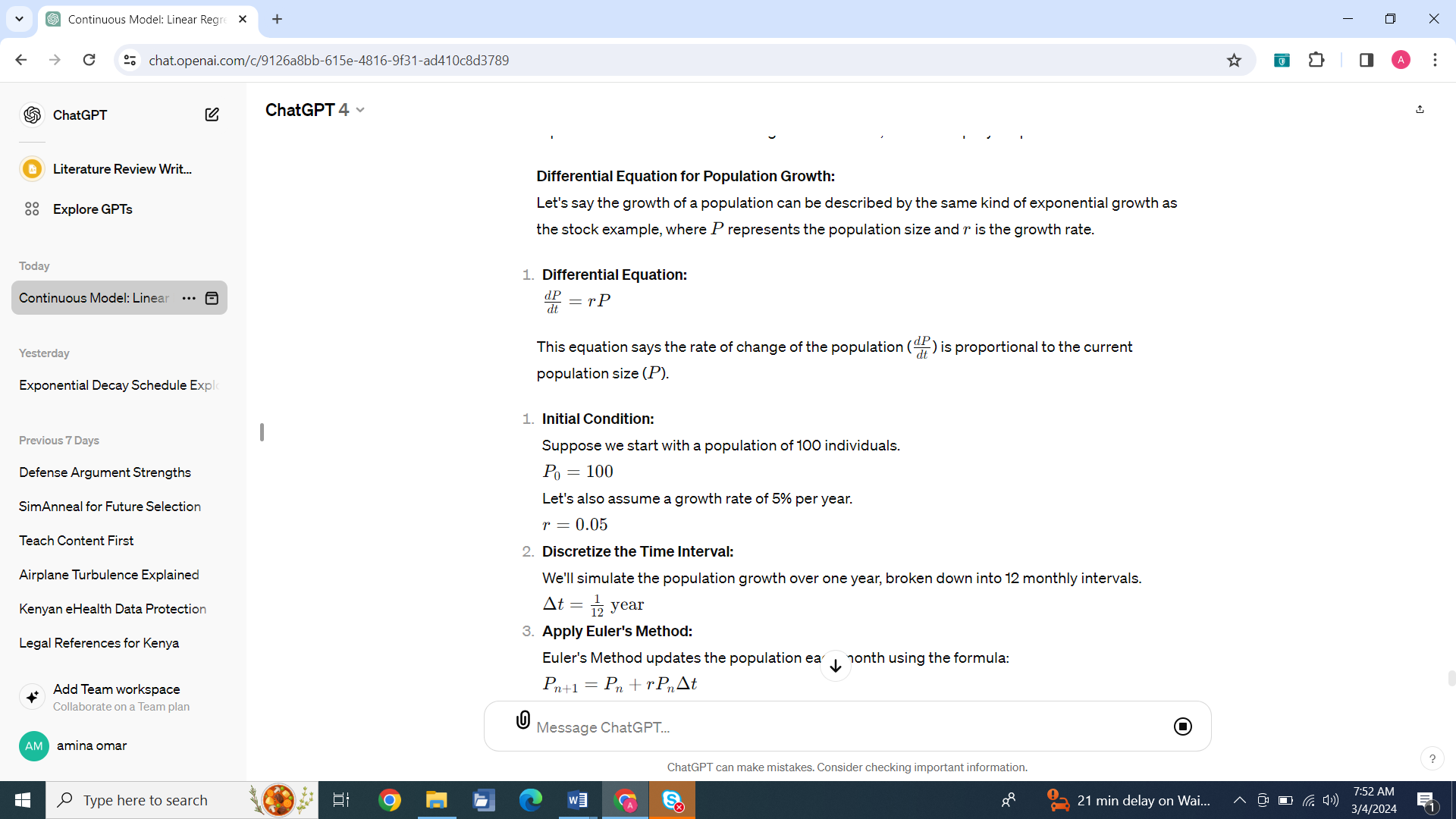
S(n+1) = S(n) + r \* S(n) \* delta\_t;

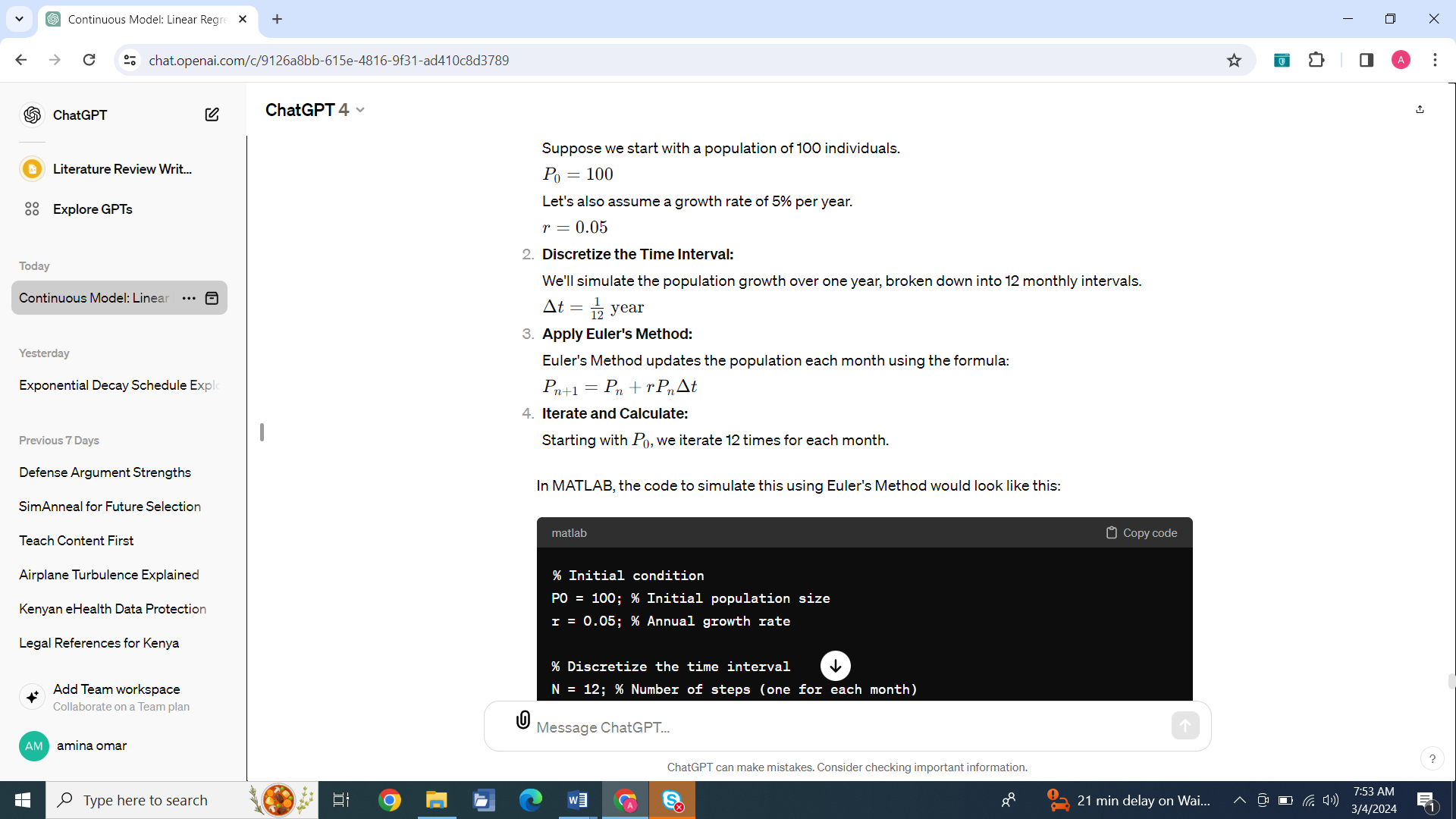
end

% Output the final stock price

fprintf('The approximate stock price after one year is $%.2f\n', S(end));

**Population growth**





**In MATLAB, the code to simulate this using Euler's Method would look like this:**

% Initial condition

P0 = 100; % Initial population size

r = 0.05; % Annual growth rate

% Discretize the time interval

N = 12; % Number of steps (one for each month)

delta\_t = 1 / N; % Size of each time step in years

% Initialize the population array

P = zeros (1, N+1); % Pre-allocate space for efficiency

P(1) = P0; % Set initial population size

% Apply Euler's method

for n = 1:N

P(n+1) = P(n) + r \* P(n) \* delta\_t;

end

% Output the final population size

fprintf('The approximate population size after one year is %d\n', P(end));