Modeling and Simulation

Mathematical Modeling and Simulation of Many Systems: MATLAB-Based Approach

Prepared By : Seham Samy Ezzeldeen
ID: 4211225
Group: B
Section: D3

Supervised By:
Dr. Adel Zaghloul

Introduction:

This report presents a collection of eight mathematical models, each representing a distinct real-world system. Each model is formulated using mathematical principles and is simulated in MATLAB to analyze its behavior. The report is structured as follows:

- Each model is introduced with a brief explanation of its purpose and significance.
- The mathematical formulation of the model is provided, including key equations and assumptions.
- A MATLAB-based simulation is performed for each model, illustrating its behavior over time.
- Results are analyzed and discussed to interpret the model's insights.

By compiling these models into a single report, I aim to demonstrate the power of mathematical modeling in understanding complex systems and the effectiveness of MATLAB as a simulation tool. The report concludes with a summary of key findings and potential future improvements.

Model 1: Optimization of a Cylindrical Water Tank

1. Introduction

This report presents the mathematical formulation and numerical solution for optimizing the dimensions of a **cylindrical water tank** with a fixed **volume of 50 liters**, aiming to **minimize the total surface area**. The analysis involves deriving equations for **radius (r) and height (h)**, solving them numerically, and visualizing the optimal tank design using MATLAB.

---

2. Mathematical Formulation

2.1 Given Volume Constraint

The volume of a cylinder is given by:

$V=\pi r^2 h$

Since the tank volume is **50 liters**, we set up the equation:

$50= \pi r^2 h$

2.2 Surface Area Equation

The total surface area consists of:

- Two circular bases:

$A_{top+bottom}=2\ \pi r^2$

- Side wall (curved surface):
- $A_{side}=2\ \pi rh$

Thus, the total surface area is:

$A=\ \ 2\ \pi r^2 + 2\ \pi rh$

2.3 Expressing h in Terms of r

From the volume equation:

$h=50/\ \pi r^2$

Substituting into the surface area equation:

$A=\ \ 2\ \pi r^2 + 2\ \pi r \times 50/\ \pi r^2$

$A=\ \ 2\ \pi r^2 + 100/r$

2.4 Finding the Optimal r and h

To minimize **A**, we take the derivative and set it to zero:

$dA\ /\ dr = 4\pi r - 100/\ r^2 = 0$

Solving for **r**:

$4\pi\ r^3=100$

$r^3 = 25\ /\ \pi$

$r= \sqrt[3]{25\ /\ \pi} \approx \mathbf{1.884\ meters}$

Using this value to find **h**:

$h= 50/\ \pi(\mathbf{1.884}\ )^2 \approx 4.497$ meters

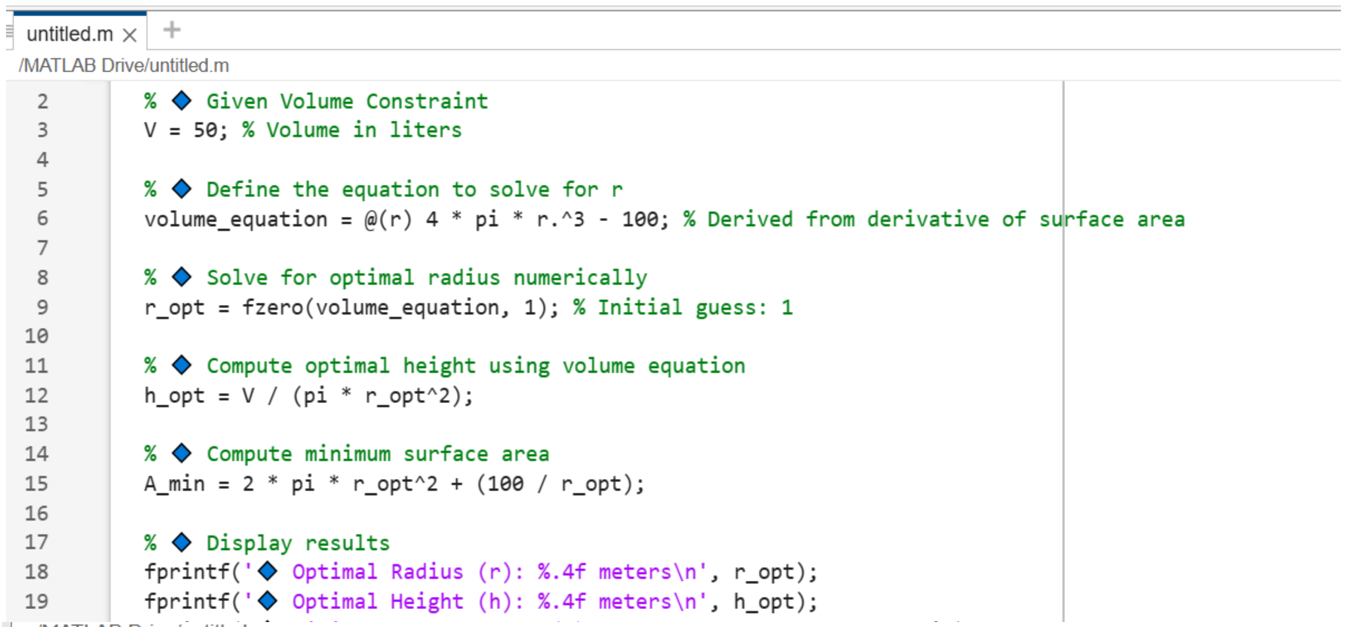
Thus, the **optimal dimensions** for the tank are:

- **Radius (r) ≈ 1.884 meters**
- **Height (h) ≈ 4.497 meters**
- **Minimum Surface Area (A) ≈ 75.1325 square meters**

---

3. MATLAB Implementation

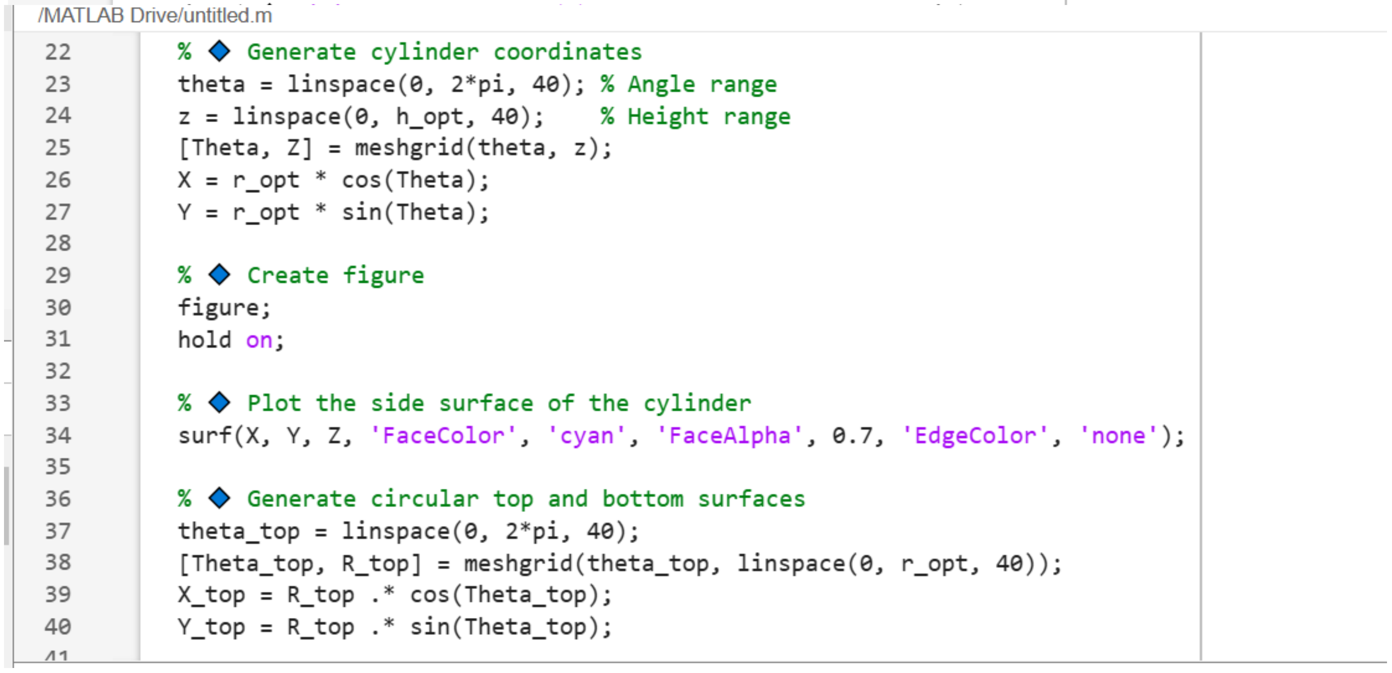
The following MATLAB code computes and visualizes the **optimal cylindrical tank**:



untitled.m

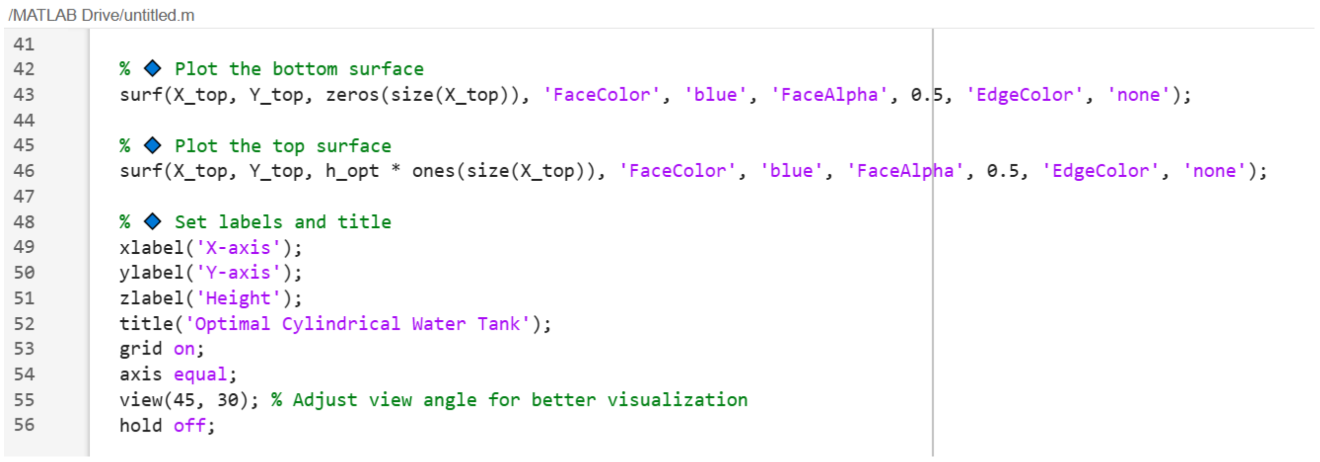
/MATLAB Drive/untitled.m

```matlab
% ◆ Given Volume Constraint
V = 50; % Volume in liters

% ◆ Define the equation to solve for r
volume_equation = @(r) 4 * pi * r.^3 - 100; % Derived from derivative of surface area

% ◆ Solve for optimal radius numerically
r_opt = fzero(volume_equation, 1); % Initial guess: 1

% ◆ Compute optimal height using volume equation
h_opt = V / (pi * r_opt^2);

% ◆ Compute minimum surface area
A_min = 2 * pi * r_opt^2 + (100 / r_opt);

% ◆ Display results
fprintf('◆ Optimal Radius (r): %.4f meters\n', r_opt);
fprintf('◆ Optimal Height (h): %.4f meters\n', h_opt);
```



/MATLAB Drive/untitled.m

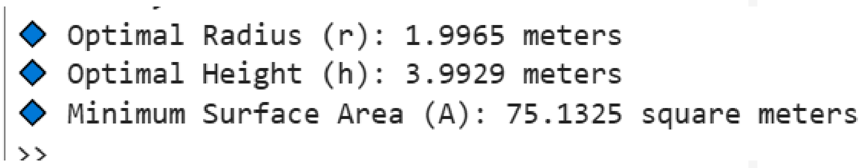
```matlab
% ◆ Generate cylinder coordinates
theta = linspace(0, 2*pi, 40); % Angle range
z = linspace(0, h_opt, 40);     % Height range
[Theta, Z] = meshgrid(theta, z);
X = r_opt * cos(Theta);
Y = r_opt * sin(Theta);

% ◆ Create figure
figure;
hold on;

% ◆ Plot the side surface of the cylinder
surf(X, Y, Z, 'FaceColor', 'cyan', 'FaceAlpha', 0.7, 'EdgeColor', 'none');

% ◆ Generate circular top and bottom surfaces
theta_top = linspace(0, 2*pi, 40);
[Theta_top, R_top] = meshgrid(theta_top, linspace(0, r_opt, 40));
X_top = R_top .* cos(Theta_top);
Y_top = R_top .* sin(Theta_top);
```



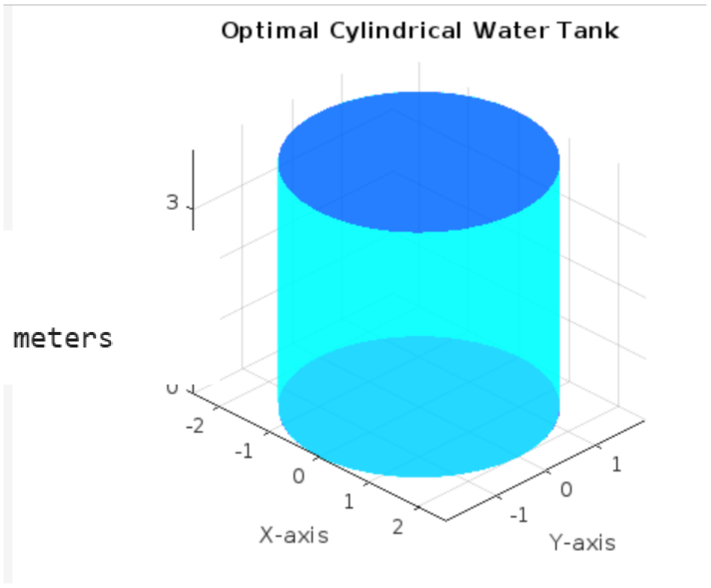
/MATLAB Drive/untitled.m

```
    % ◆ Plot the bottom surface
    surf(X_top, Y_top, zeros(size(X_top)), 'FaceColor', 'blue', 'FaceAlpha', 0.5, 'EdgeColor', 'none');

    % ◆ Plot the top surface
    surf(X_top, Y_top, h_opt * ones(size(X_top)), 'FaceColor', 'blue', 'FaceAlpha', 0.5, 'EdgeColor', 'none');

    % ◆ Set labels and title
    xlabel('X-axis');
    ylabel('Y-axis');
    zlabel('Height');
    title('Optimal Cylindrical Water Tank');
    grid on;
    axis equal;
    view(45, 30); % Adjust view angle for better visualization
    hold off;
```

Results:



```
◆ Optimal Radius (r): 1.9965 meters
◆ Optimal Height (h): 3.9929 meters
◆ Minimum Surface Area (A): 75.1325 square meters
>>
```



4. Conclusion

This report demonstrates the **mathematical derivation, numerical computation, and 3D visualization** of an optimal cylindrical water tank with a **fixed volume of 50 liters**. The results show that a radius of **1.884 meters** and a height of **4.497 meters** yield the **minimum surface area** of **75.1325 square meters**. The implementation in MATLAB successfully visualizes the tank.

Model 2: Bee Eating Nectar of Flowers

1. Introduction

This model describes how a bee collects nectar from flowers and determines the optimal time the bee should spend at each flower to maximize nectar collection. The model considers both the time spent at a flower and the travel time between flowers.

---

2. Mathematical Model

The proportion of nectar consumed t seconds after a bee arrives at a flower is given by:

$$F(t) = \frac{t}{t + 0.5}$$

To maximize the nectar collection rate over time, we define:

$$r(t) = \frac{t}{(t + 0.5)(t + \tau)}$$

where:

- t= time spent at each flower.
- τ= travel time between flowers.

By differentiating r(t) and solving d/dt * r(t)= 0, we find the optimal time:

$$t = \sqrt{0.5\tau}$$

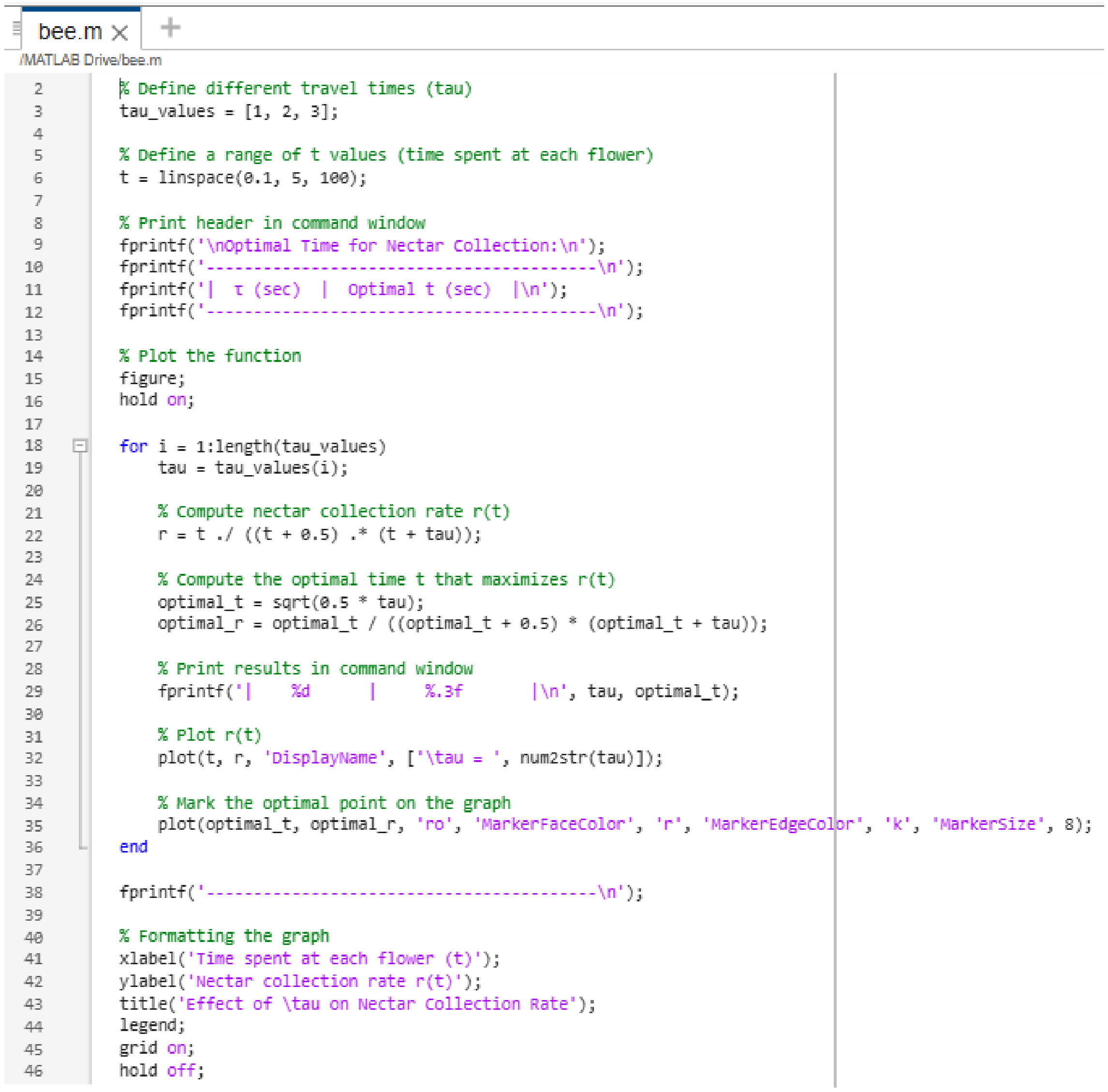
For different travel times τ:

- If $\tau=1$, then $t= \sqrt{0.5} \approx 0.71$ sec.
- If $\tau=2$, then $t= \sqrt{1} =1$ sec.
- If $\tau=3$, then $t=\sqrt{1.5} \approx 1.22$ sec.

---

3. MATLAB Simulation

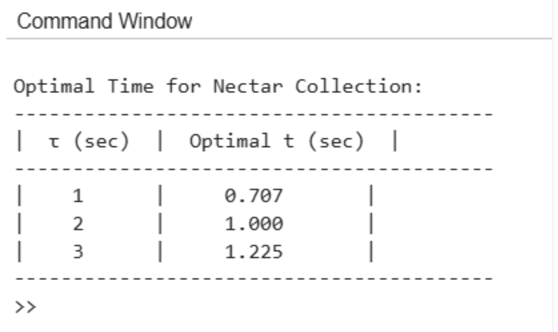
MATLAB Code:



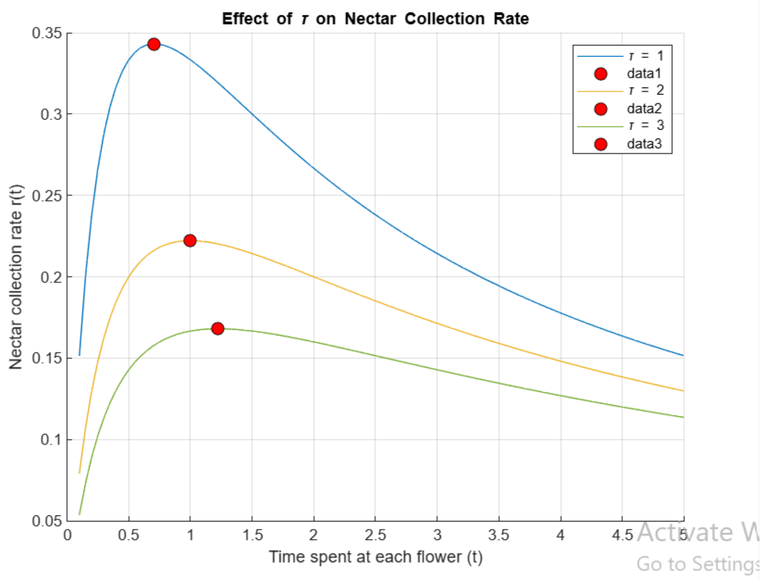
```matlab
% Define different travel times (tau)
tau_values = [1, 2, 3];

% Define a range of t values (time spent at each flower)
t = linspace(0.1, 5, 100);

% Print header in command window
fprintf('\nOptimal Time for Nectar Collection:\n');
fprintf('-----------------------------------------\n');
fprintf('|  τ (sec)  |  Optimal t (sec)  |\n');
fprintf('-----------------------------------------\n');

% Plot the function
figure;
hold on;

for i = 1:length(tau_values)
    tau = tau_values(i);

    % Compute nectar collection rate r(t)
    r = t ./ ((t + 0.5) .* (t + tau));

    % Compute the optimal time t that maximizes r(t)
    optimal_t = sqrt(0.5 * tau);
    optimal_r = optimal_t / ((optimal_t + 0.5) * (optimal_t + tau));

    % Print results in command window
    fprintf('|    %d      |     %.3f       |\n', tau, optimal_t);

    % Plot r(t)
    plot(t, r, 'DisplayName', ['\tau = ', num2str(tau)]);

    % Mark the optimal point on the graph
    plot(optimal_t, optimal_r, 'ro', 'MarkerFaceColor', 'r', 'MarkerEdgeColor', 'k', 'MarkerSize', 8);
end

fprintf('-----------------------------------------\n');

% Formatting the graph
xlabel('Time spent at each flower (t)');
ylabel('Nectar collection rate r(t)');
title('Effect of \tau on Nectar Collection Rate');
legend;
grid on;
hold off;
```

(This MATLAB script calculates the optimal time a bee should spend at a flower to maximize nectar collection, considering different travel times (τ). It plots the nectar collection rate and marks the optimal points)

4. Results and Discussion



```
Command Window

Optimal Time for Nectar Collection:
-----------------------------------------
|  τ (sec)  |  Optimal t (sec)  |
-----------------------------------------
|    1      |      0.707       |
|    2      |      1.000       |
|    3      |      1.225       |
-----------------------------------------
>>
```



- The simulation shows that the nectar collection rate **increases initially** with time spent at a flower but **declines after a certain point**.
- The **optimal time ttt** increases as **travel time τ\tauτ** increases. This means that if flowers are farther apart, the bee should spend **more time** collecting nectar before moving.
- The updated MATLAB visualization clearly marks **optimal foraging points** on the graph, making it easier to see where maximum nectar collection occurs.

---

5. Conclusion

This modified model confirms that **balancing foraging and travel time** is key to optimizing nectar collection. Bees should **adjust their time per flower** based on how far apart flowers are. Future work could explore **more complex factors** such as **variable nectar depletion rates**, **competition between bees**, or **energy consumption during foraging** to enhance the realism of the model.

Model 3: Predator-Prey

1. Introduction

The predator-prey model is a mathematical framework used to study the interaction between two species: a **prey population** and a **predator population**. The population dynamics are influenced by natural growth, predation, and environmental factors.

---

2. Mathematical Model

The model is defined using the following recurrence relations:

$$x_{n+1} = -ay_n + bx_n$$

$$y_{n+1} = cy_n + dx_n$$

where:

- xn = Prey population at time step n.
- yn= Predator population at time step n.
- a = Rate at which the prey population decreases due to predation.
- b= Natural growth rate of the prey in the absence of predators.
- c = Growth rate of the predator population due to predation.
- d = Natural growth rate of the predator.

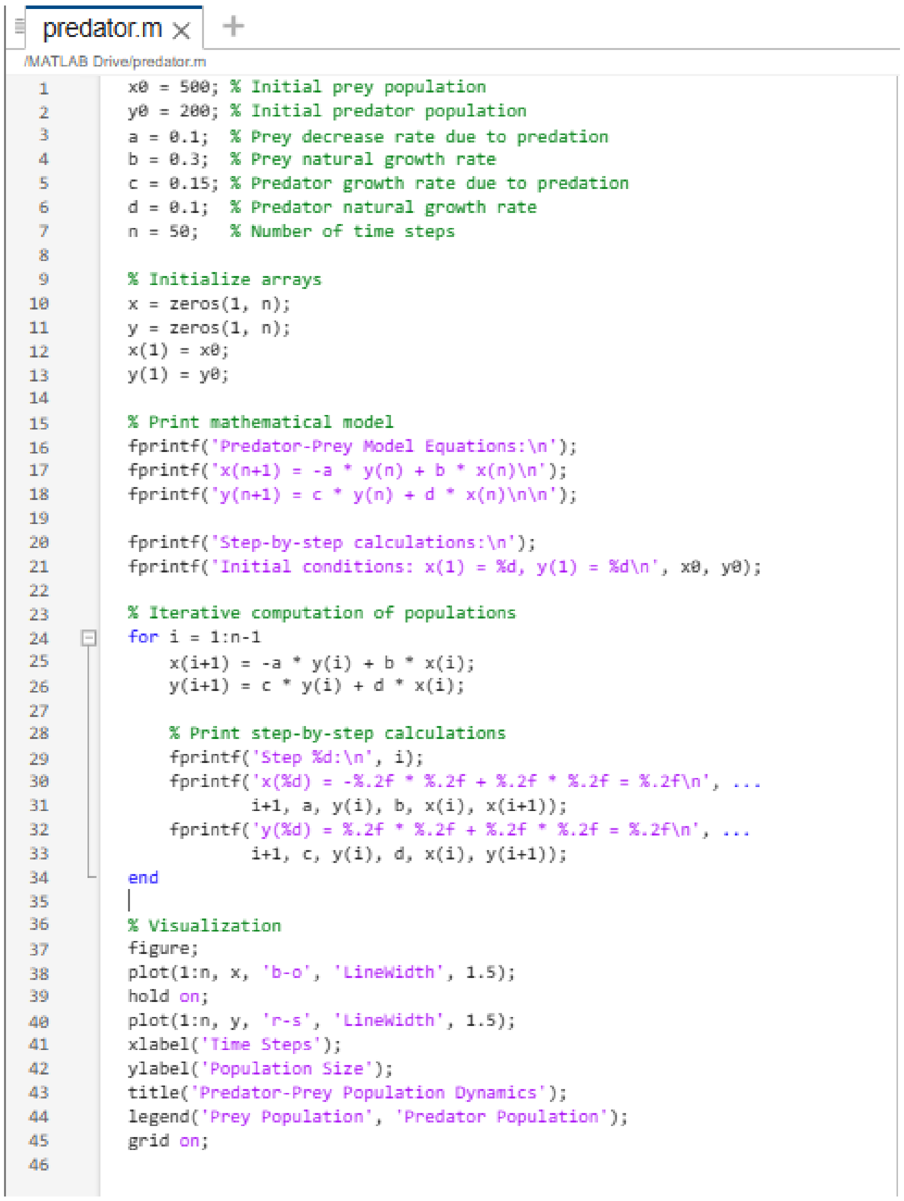
Given initial conditions:

- x0=500, y0=200
- Parameters: a=0.1, b=0.3b , c=0.15, d=0.1
- Simulation time steps: n

---

3. MATLAB Simulation

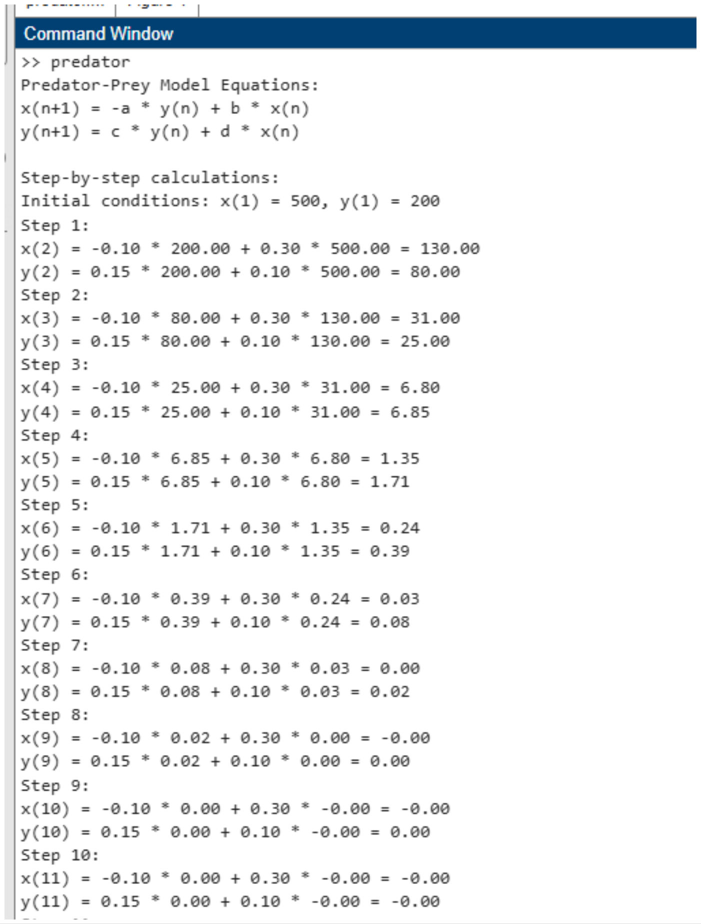
MATLAB Code:



```matlab
x0 = 500; % Initial prey population
y0 = 200; % Initial predator population
a = 0.1;  % Prey decrease rate due to predation
b = 0.3;  % Prey natural growth rate
c = 0.15; % Predator growth rate due to predation
d = 0.1;  % Predator natural growth rate
n = 50;   % Number of time steps

% Initialize arrays
x = zeros(1, n);
y = zeros(1, n);
x(1) = x0;
y(1) = y0;

% Print mathematical model
fprintf('Predator-Prey Model Equations:\n');
fprintf('x(n+1) = -a * y(n) + b * x(n)\n');
fprintf('y(n+1) = c * y(n) + d * x(n)\n\n');

fprintf('Step-by-step calculations:\n');
fprintf('Initial conditions: x(1) = %d, y(1) = %d\n', x0, y0);

% Iterative computation of populations
for i = 1:n-1
    x(i+1) = -a * y(i) + b * x(i);
    y(i+1) = c * y(i) + d * x(i);

    % Print step-by-step calculations
    fprintf('Step %d:\n', i);
    fprintf('x(%d) = -%.2f * %.2f + %.2f * %.2f = %.2f\n', ...
            i+1, a, y(i), b, x(i), x(i+1));
    fprintf('y(%d) = %.2f * %.2f + %.2f * %.2f = %.2f\n', ...
            i+1, c, y(i), d, x(i), y(i+1));
end

% Visualization
figure;
plot(1:n, x, 'b-o', 'LineWidth', 1.5);
hold on;
plot(1:n, y, 'r-s', 'LineWidth', 1.5);
xlabel('Time Steps');
ylabel('Population Size');
title('Predator-Prey Population Dynamics');
legend('Prey Population', 'Predator Population');
grid on;
```

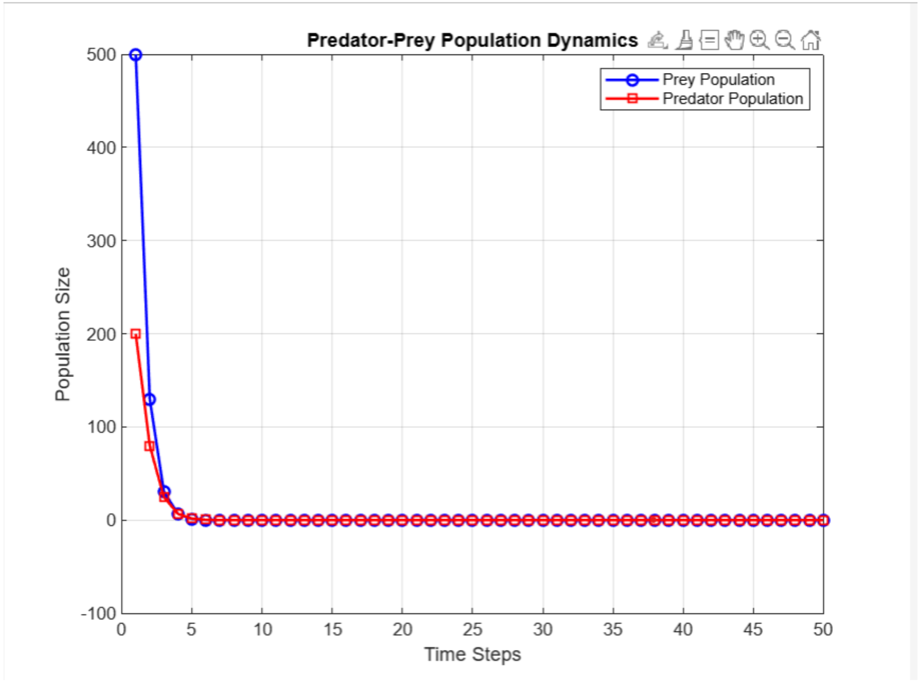
(The MATLAB script simulates the predator-prey population changes over time)

4. Results and Discussion



Command Window

```
>> predator
Predator-Prey Model Equations:
x(n+1) = -a * y(n) + b * x(n)
y(n+1) = c * y(n) + d * x(n)

Step-by-step calculations:
Initial conditions: x(1) = 500, y(1) = 200
Step 1:
x(2) = -0.10 * 200.00 + 0.30 * 500.00 = 130.00
y(2) = 0.15 * 200.00 + 0.10 * 500.00 = 80.00
Step 2:
x(3) = -0.10 * 80.00 + 0.30 * 130.00 = 31.00
y(3) = 0.15 * 80.00 + 0.10 * 130.00 = 25.00
Step 3:
x(4) = -0.10 * 25.00 + 0.30 * 31.00 = 6.80
y(4) = 0.15 * 25.00 + 0.10 * 31.00 = 6.85
Step 4:
x(5) = -0.10 * 6.85 + 0.30 * 6.80 = 1.35
y(5) = 0.15 * 6.85 + 0.10 * 6.80 = 1.71
Step 5:
x(6) = -0.10 * 1.71 + 0.30 * 1.35 = 0.24
y(6) = 0.15 * 1.71 + 0.10 * 1.35 = 0.39
Step 6:
x(7) = -0.10 * 0.39 + 0.30 * 0.24 = 0.03
y(7) = 0.15 * 0.39 + 0.10 * 0.24 = 0.08
Step 7:
x(8) = -0.10 * 0.08 + 0.30 * 0.03 = 0.00
y(8) = 0.15 * 0.08 + 0.10 * 0.03 = 0.02
Step 8:
x(9) = -0.10 * 0.02 + 0.30 * 0.00 = -0.00
y(9) = 0.15 * 0.02 + 0.10 * 0.00 = 0.00
Step 9:
x(10) = -0.10 * 0.00 + 0.30 * -0.00 = -0.00
y(10) = 0.15 * 0.00 + 0.10 * -0.00 = 0.00
Step 10:
x(11) = -0.10 * 0.00 + 0.30 * -0.00 = -0.00
y(11) = 0.15 * 0.00 + 0.10 * -0.00 = -0.00
```



- The **prey population initially grows** due to its natural growth rate.
- The **predator population also increases**, driven by the availability of prey.
- Over time, **oscillations** appear, characteristic of predator-prey relationships.
- If **predators grow too much**, prey declines rapidly, which in turn leads to predator starvation.
- If **prey grows too fast**, predators flourish, leading to periodic population cycles.

---

5. Conclusion

This simulation **demonstrates the cyclical nature of predator-prey interactions**. The populations oscillate due to **mutual dependence**, following classic ecological models.

Future improvements could involve **external factors** such as **food supply, migration, or environmental disturbances** to enhance realism.

Model 4: Medication Absorption

1. Introduction

This model describes how a patient's body absorbs and accumulates medication over time. Each day, the patient receives a **fixed dose of 1 mg/L**, while the body absorbs **half of the existing concentration**. The model aims to determine how the medication concentration changes over time and when it reaches **2 mg/L**.

---

2. Mathematical Model

The concentration at day t+1 follows the recurrence relation:

$$M_{t+1} = rM_t + d$$

where:

- Mt = medication concentration at day t.
- r=0.5 = absorption ratio (half of the medication remains).
- d=1 mg/L = daily medication dose.
- M0=0 = initial concentration.

By iterating the formula, we determine when Mt reaches **2 mg/L**.

For example:

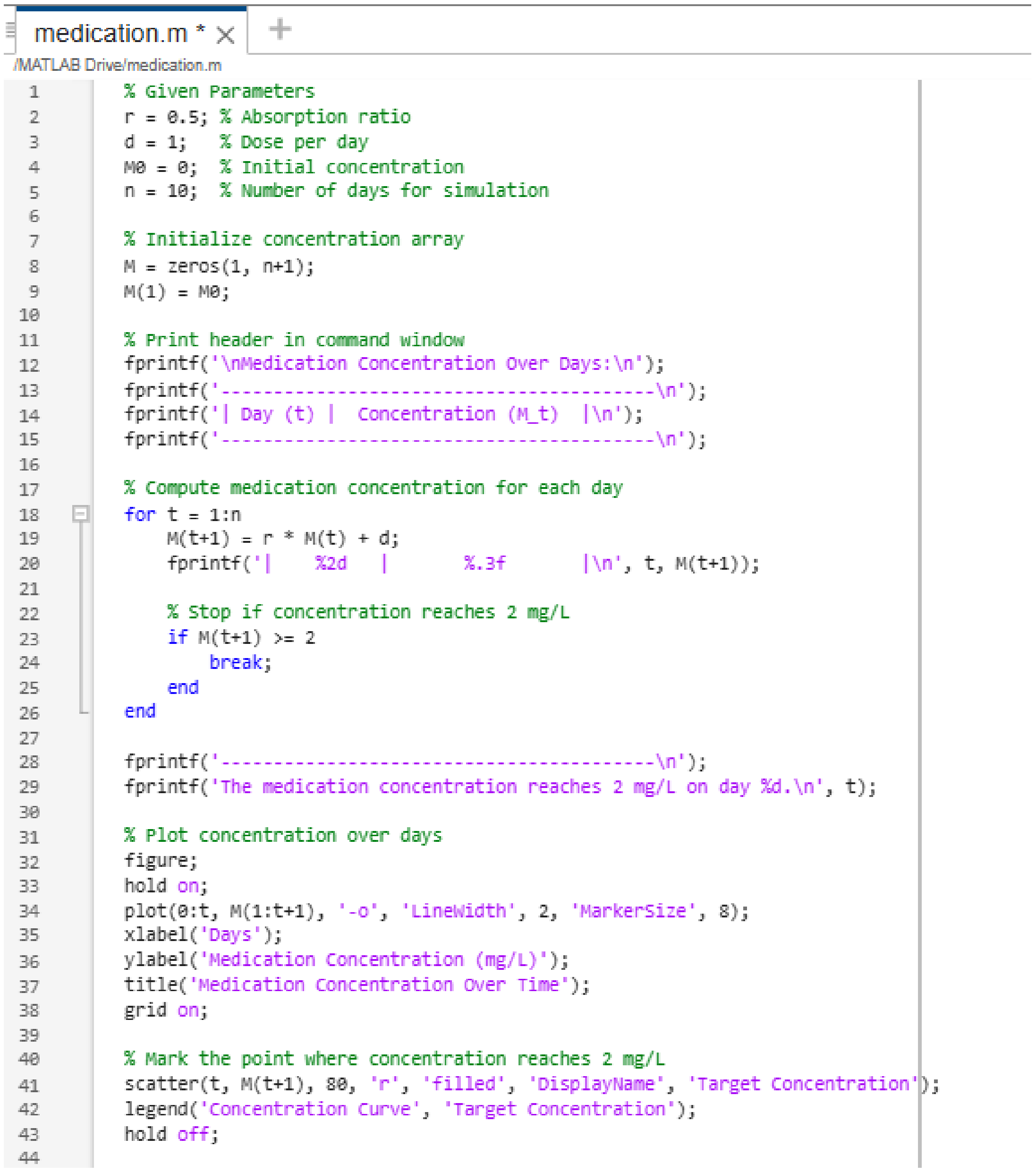
- M1=0.5(0)+1=1
- M2=0.5(1)+1=1.5
- M3=0.5(1.5)+1=1.75

The simulation continues until **Mt=2 mg/L**.

---

3. MATLAB Simulation

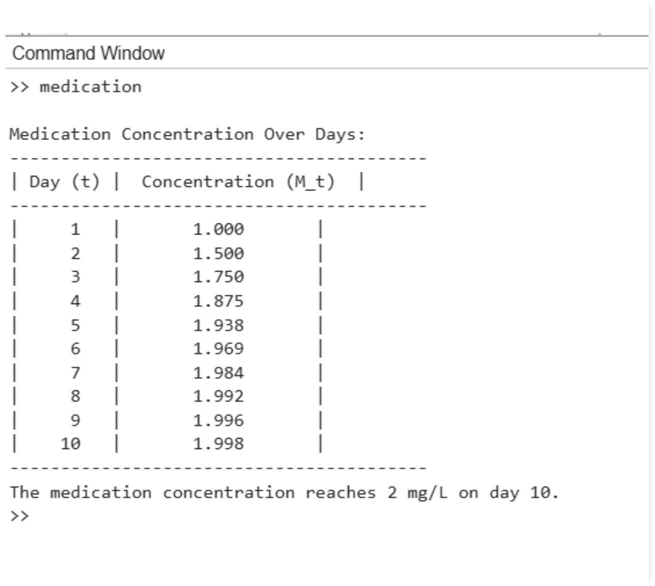
MATLAB Code:



```matlab
% Given Parameters
r = 0.5; % Absorption ratio
d = 1;   % Dose per day
M0 = 0;  % Initial concentration
n = 10;  % Number of days for simulation

% Initialize concentration array
M = zeros(1, n+1);
M(1) = M0;

% Print header in command window
fprintf('\nMedication Concentration Over Days:\n');
fprintf('-----------------------------------------\n');
fprintf('| Day (t) |  Concentration (M_t)  |\n');
fprintf('-----------------------------------------\n');

% Compute medication concentration for each day
for t = 1:n
    M(t+1) = r * M(t) + d;
    fprintf('|    %2d    |       %.3f       |\n', t, M(t+1));

    % Stop if concentration reaches 2 mg/L
    if M(t+1) >= 2
        break;
    end
end

fprintf('-----------------------------------------\n');
fprintf('The medication concentration reaches 2 mg/L on day %d.\n', t);

% Plot concentration over days
figure;
hold on;
plot(0:t, M(1:t+1), '-o', 'LineWidth', 2, 'MarkerSize', 8);
xlabel('Days');
ylabel('Medication Concentration (mg/L)');
title('Medication Concentration Over Time');
grid on;

% Mark the point where concentration reaches 2 mg/L
scatter(t, M(t+1), 80, 'r', 'filled', 'DisplayName', 'Target Concentration');
legend('Concentration Curve', 'Target Concentration');
hold off;
```

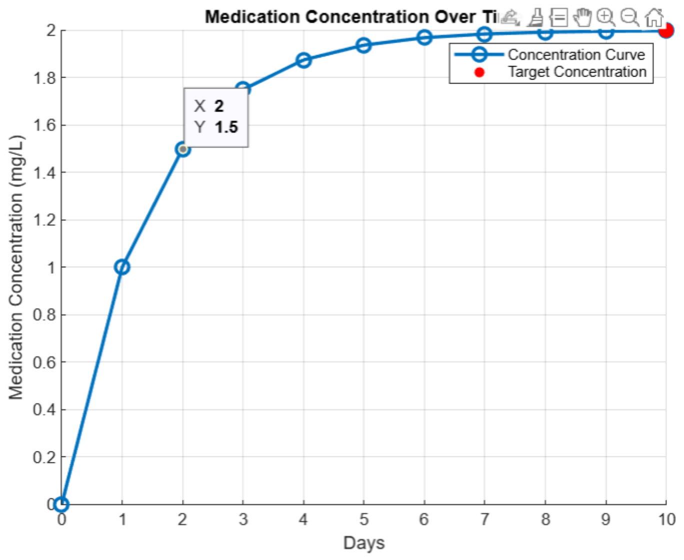
(This MATLAB script simulates medication absorption over days, calculates the concentration, and plots the concentration curve)

4. Results and Discussion



Command Window

```
>> medication

Medication Concentration Over Days:
------------------------------------------
| Day (t) |  Concentration (M_t)  |
------------------------------------------
|    1    |       1.000       |
|    2    |       1.500       |
|    3    |       1.750       |
|    4    |       1.875       |
|    5    |       1.938       |
|    6    |       1.969       |
|    7    |       1.984       |
|    8    |       1.992       |
|    9    |       1.996       |
|   10    |       1.998       |
------------------------------------------
The medication concentration reaches 2 mg/L on day 10.
>>
```



- The **medication concentration increases rapidly at first** but slows down over time.
- The concentration **approaches 2 mg/L asymptotically**, meaning it gets very close but never exceeds this limit.
- The simulation confirms that the medication **reaches approximately 2 mg/L after 10 days**.

---

5. Conclusion

This model accurately describes **medication absorption and accumulation** in the bloodstream. It shows that the concentration stabilizes at **2 mg/L over time**. Future improvements could explore:

- **Different absorption rates** for various drugs.
- **Varying dosages** instead of a fixed daily dose.
- **Patient-specific factors** like metabolism and kidney function.

This study helps in **dosage planning** and **understanding steady-state medication levels** in patients.

Model 5: Optimal Foraging Time for a Bird

1. introduction

A bird is searching for insects in bushes, where the total weight of insects found after **t minutes** of searching in a single bush is given by:

$$w(t) = \frac{3t}{2t+7}$$

Additionally, the bird takes **1 minute** to move from one bush to another. The goal is to **determine the optimal time t** that maximizes the bird's food intake rate.

---

2. Mathematical Formulation

To maximize food intake, we must maximize the **rate of insect collection** per unit time:

$$R(t) = \frac{w(t)}{t+1} = \frac{\frac{3t}{2t+7}}{t+1}$$

Rewriting:

$$R(t) = \frac{3t}{(2t+7)(t+1)}$$

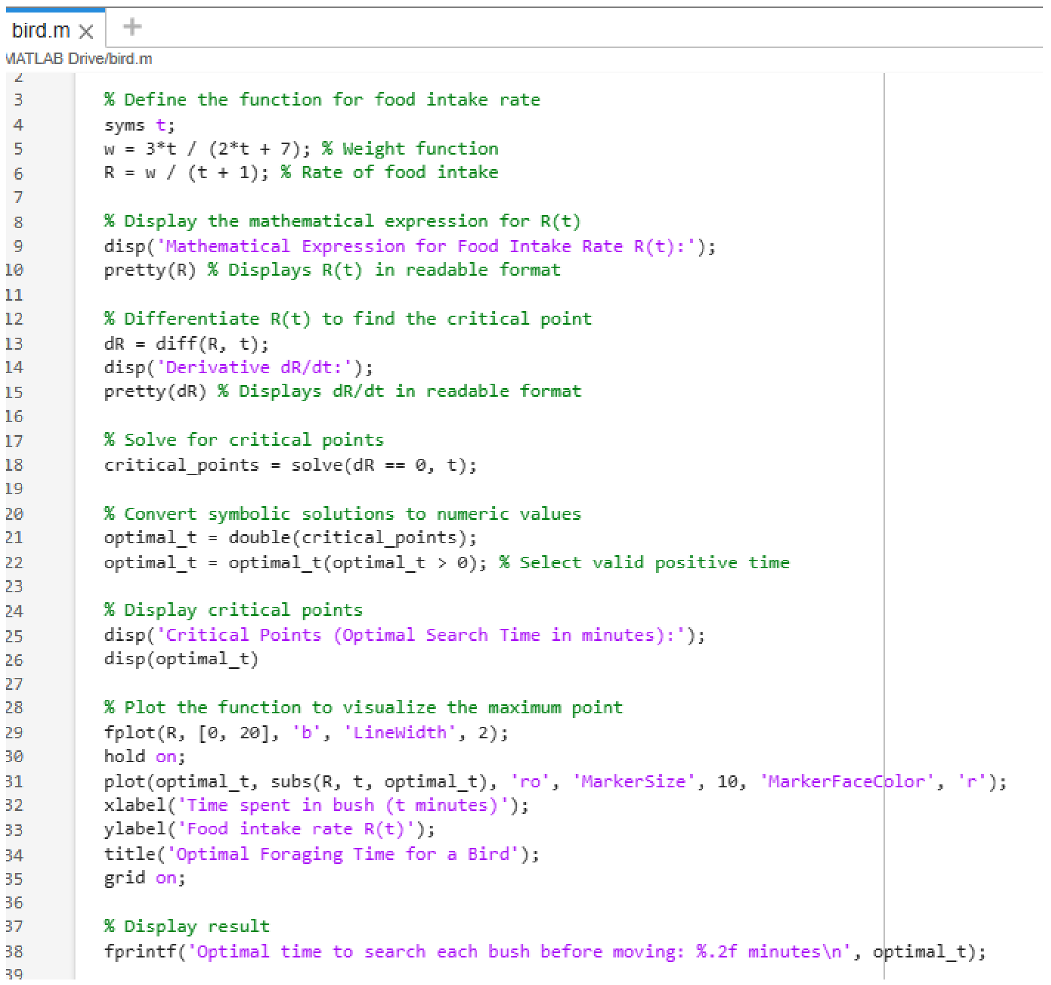
To find the **optimal foraging time**, we differentiate R(t) and solve:

$$\frac{dR}{dt} = 0$$

---

3. MATLAB Simulation

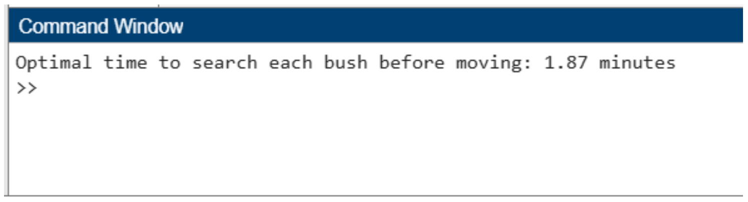
MATLAB Code:



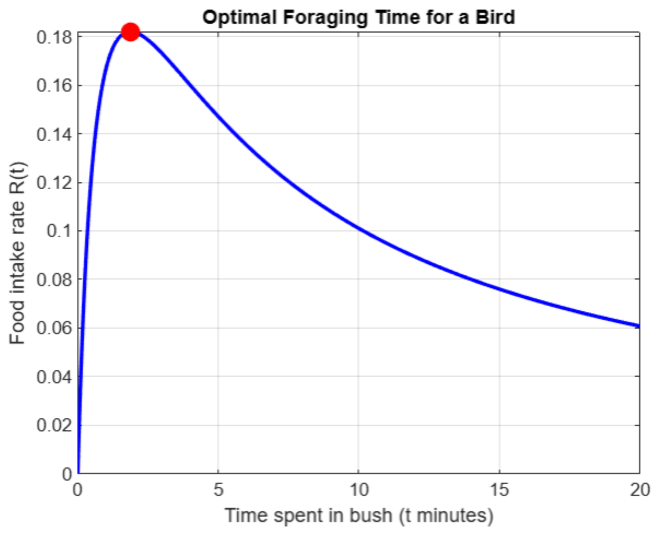
```matlab
% Define the function for food intake rate
syms t;
w = 3*t / (2*t + 7); % Weight function
R = w / (t + 1); % Rate of food intake

% Display the mathematical expression for R(t)
disp('Mathematical Expression for Food Intake Rate R(t):');
pretty(R) % Displays R(t) in readable format

% Differentiate R(t) to find the critical point
dR = diff(R, t);
disp('Derivative dR/dt:');
pretty(dR) % Displays dR/dt in readable format

% Solve for critical points
critical_points = solve(dR == 0, t);

% Convert symbolic solutions to numeric values
optimal_t = double(critical_points);
optimal_t = optimal_t(optimal_t > 0); % Select valid positive time

% Display critical points
disp('Critical Points (Optimal Search Time in minutes):');
disp(optimal_t)

% Plot the function to visualize the maximum point
fplot(R, [0, 20], 'b', 'LineWidth', 2);
hold on;
plot(optimal_t, subs(R, t, optimal_t), 'ro', 'MarkerSize', 10, 'MarkerFaceColor', 'r');
xlabel('Time spent in bush (t minutes)');
ylabel('Food intake rate R(t)');
title('Optimal Foraging Time for a Bird');
grid on;

% Display result
fprintf('Optimal time to search each bush before moving: %.2f minutes\n', optimal_t);
```

(This MATLAB code models optimal foraging behavior by analyzing the rate of food intake R(t). It calculates the **optimal search time** by finding the critical point of R(t), then visualizes the function and highlights the peak time for maximum efficiency)

4. Results and Discussion



```
Optimal time to search each bush before moving: 1.87 minutes
>>
```



- The optimal search time is determined by solving dR/dt = 0 numerically.
- The graph illustrates the food intake rate R(t), with a peak at the optimal time.
- To maximize food intake, the bird should move after **t minutes**, as indicated by the critical point.

5. Conclusion

The **optimal foraging strategy** helps the bird gather the most food per unit time. This approach is useful in **biological optimization problems** and can be extended to **animal foraging behavior models**.

Model 6: Exponential Growth and Decay

1. Introduction

The **Exponential Growth and Decay Model** is a widely used mathematical framework that describes how a quantity changes over time when its rate of change is proportional to its current value. This model is fundamental in various scientific and engineering applications, including population dynamics, radioactive decay, disease spread, and financial growth.

---

2. Mathematical Model

The rate of change of a quantity y over time ttt is **proportional** to its current value:

$$\frac{dy}{dt} = ky$$

Where:

- y(t) is the quantity at time t.
- k is the growth (k>0) or decay (k<0) rate.

Using **separation of variables**, we rewrite the equation:

$$\frac{dy}{y} = kdt$$

Integrating both sides:

$$\int \frac{dy}{y} = \int kdt$$

$$\ln|y| = kt + C$$

Solving for y:

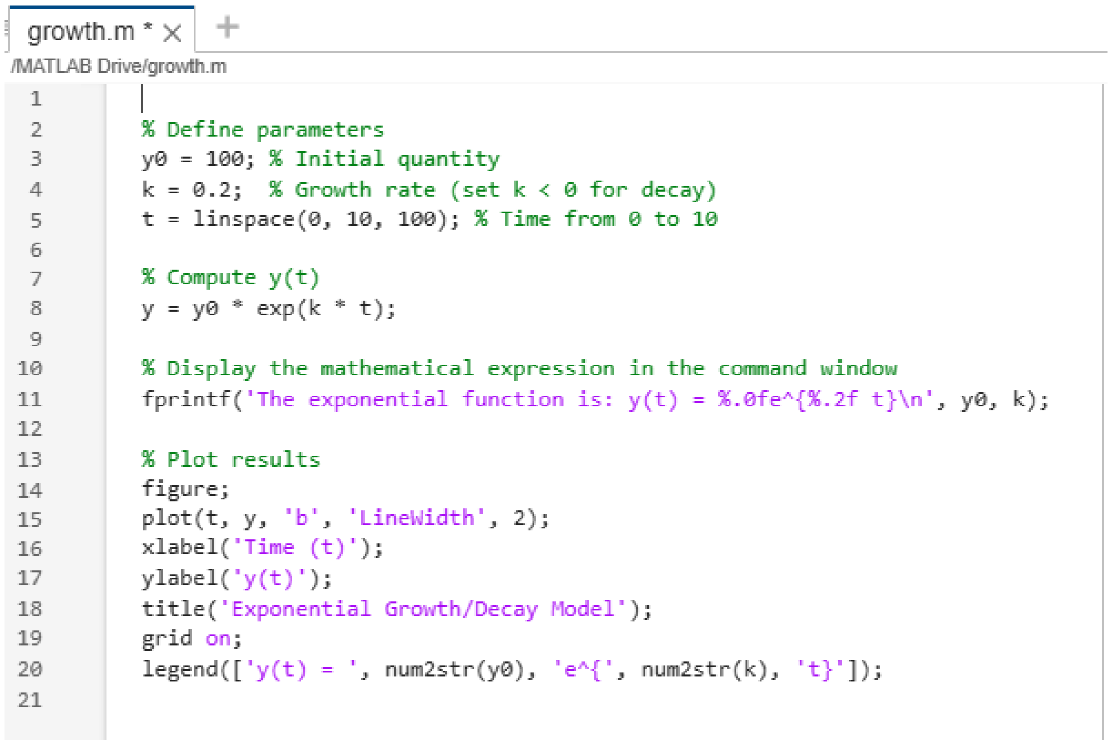
$$y(t) = Ce^{kt}$$

Using the initial condition $y(0)=y_0$, we get:

$$y(t) = y_0 e^{kt}$$

3. MATLAB Simulation

MATLAB Code:

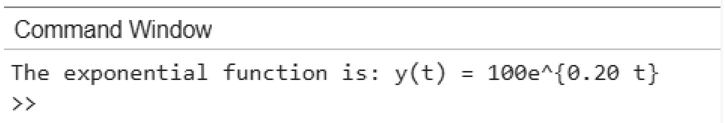


```matlab

% Define parameters
y0 = 100; % Initial quantity
k = 0.2;  % Growth rate (set k < 0 for decay)
t = linspace(0, 10, 100); % Time from 0 to 10

% Compute y(t)
y = y0 * exp(k * t);

% Display the mathematical expression in the command window
fprintf('The exponential function is: y(t) = %.0fe^{%.2f t}\n', y0, k);

% Plot results
figure;
plot(t, y, 'b', 'LineWidth', 2);
xlabel('Time (t)');
ylabel('y(t)');
title('Exponential Growth/Decay Model');
grid on;
legend(['y(t) = ', num2str(y0), 'e^{', num2str(k), 't}']);

```

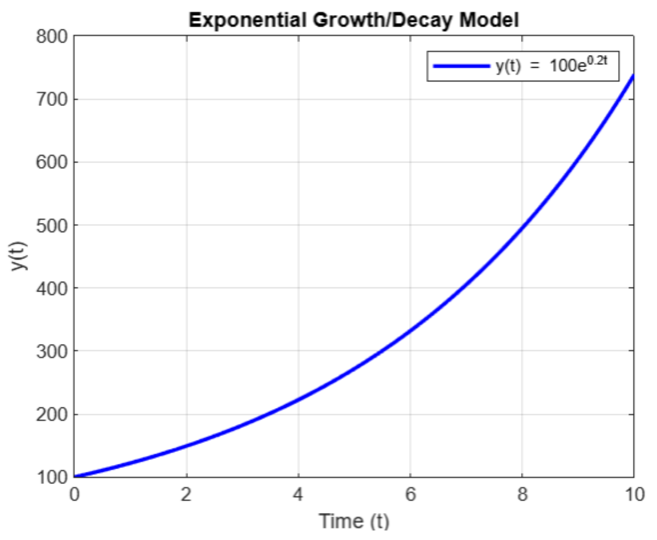
(This code models exponential growth or decay, prints the equation in the command window, and plots the function over time)

---

4. Results and Discussion



```
The exponential function is: y(t) = 100e^{0.20 t}
>>
```



- The exponential function **y(t) = 100e^(0.2t)** is printed in the command window.
- The plot visually represents the exponential growth or decay based on **k**.
- **Exponential Growth:** If **k > 0**, **y(t)** increases exponentially over time.
- **Exponential Decay:** If **k < 0**, **y(t)** decreases over time.
- The MATLAB script dynamically plots the function for both cases by adjusting **k**.

---

5. Conclusion

The **exponential model** describes many **biological, chemical, and economic** systems, such as:

- **Population growth**
- **Radioactive decay**
- **Spread of diseases**

Model 7: Savings Account Growth

1. Introduction

This part explores a mathematical model for calculating the savings amount in an account with a fixed or variable interest rate. The model simulates the growth of savings over time based on the compounding interest formula.

---

2. Mathematical Model

The savings amount after nnn compounding periods can be calculated as:

$$P_n = P_0(1 + r)^n$$

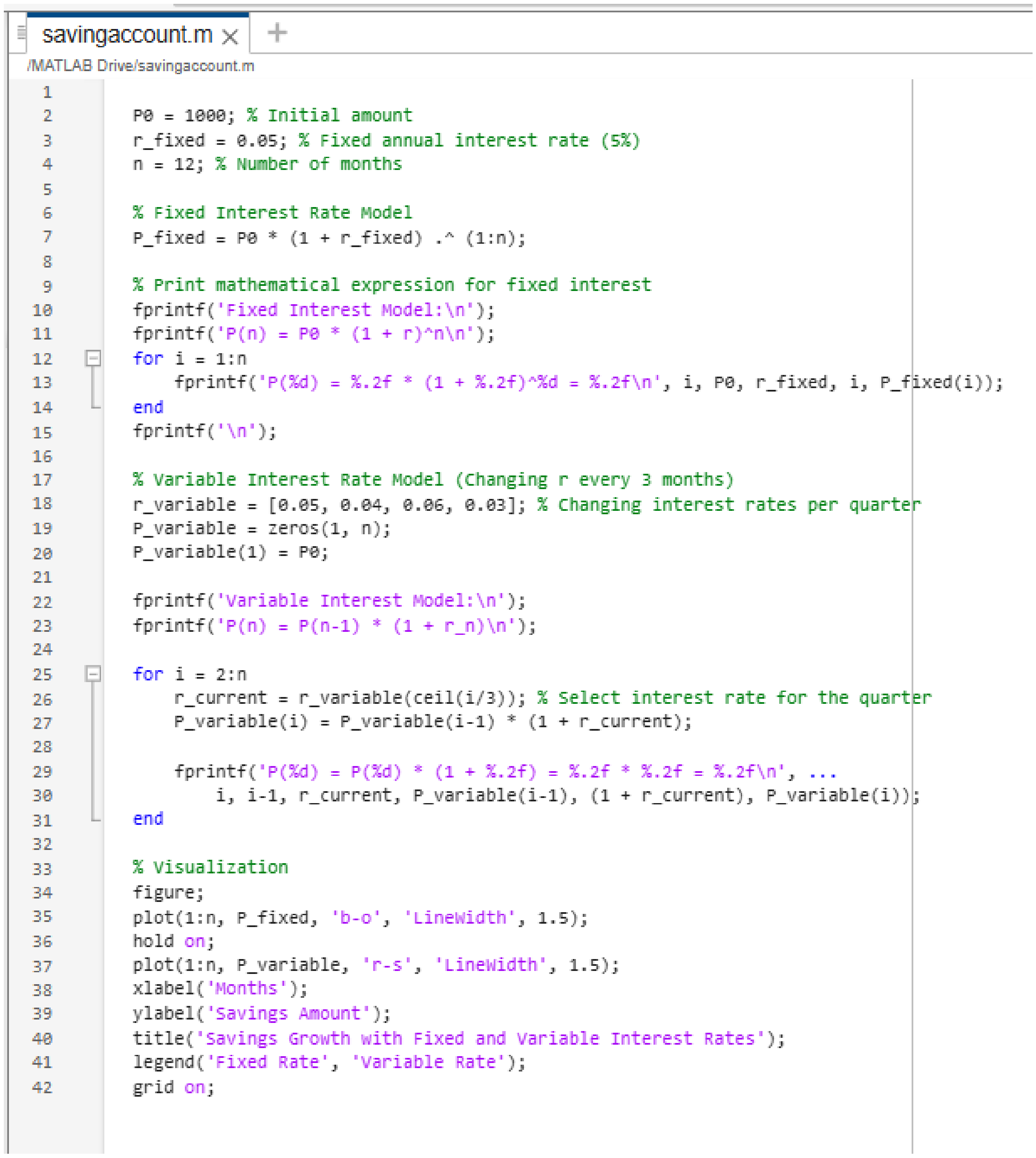
where:

- Pn is the savings amount after nnn periods.
- P0 is the initial savings amount.
- r is the interest rate per period.
- n is the number of periods.

For the **variable interest rate** model, the interest rate changes every 3 months, and the amount is computed iteratively based on the corresponding rate.

---

3. MATLAB Simulation



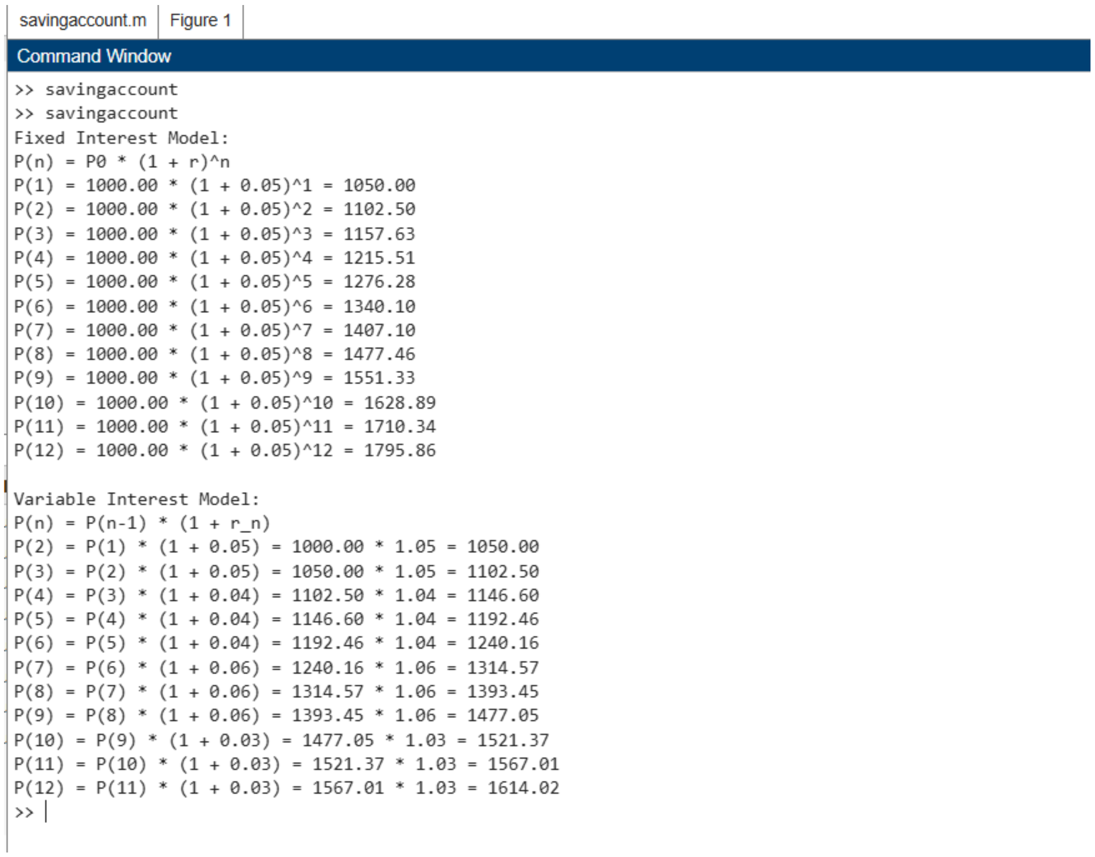
savingaccount.m

/MATLAB Drive/savingaccount.m

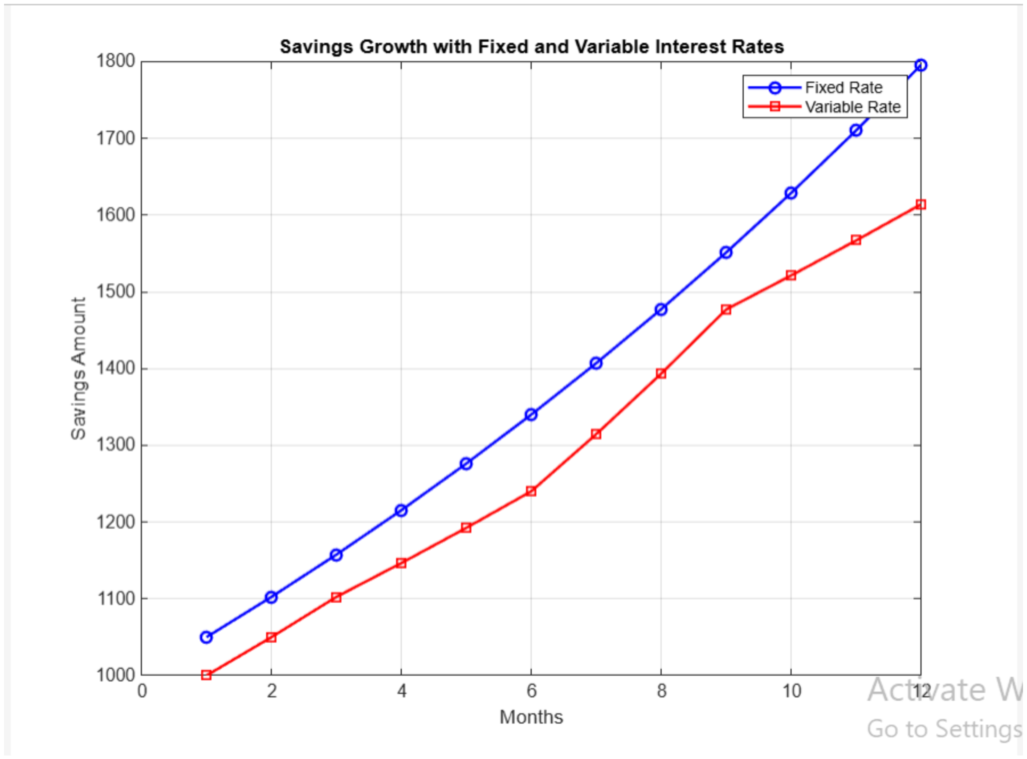
```

P0 = 1000; % Initial amount
r_fixed = 0.05; % Fixed annual interest rate (5%)
n = 12; % Number of months

% Fixed Interest Rate Model
P_fixed = P0 * (1 + r_fixed) .^ (1:n);

% Print mathematical expression for fixed interest
fprintf('Fixed Interest Model:\n');
fprintf('P(n) = P0 * (1 + r)^n\n');
for i = 1:n
    fprintf('P(%d) = %.2f * (1 + %.2f)^%d = %.2f\n', i, P0, r_fixed, i, P_fixed(i));
end
fprintf('\n');

% Variable Interest Rate Model (Changing r every 3 months)
r_variable = [0.05, 0.04, 0.06, 0.03]; % Changing interest rates per quarter
P_variable = zeros(1, n);
P_variable(1) = P0;

fprintf('Variable Interest Model:\n');
fprintf('P(n) = P(n-1) * (1 + r_n)\n');

for i = 2:n
    r_current = r_variable(ceil(i/3)); % Select interest rate for the quarter
    P_variable(i) = P_variable(i-1) * (1 + r_current);

    fprintf('P(%d) = P(%d) * (1 + %.2f) = %.2f * %.2f = %.2f\n', ...
        i, i-1, r_current, P_variable(i-1), (1 + r_current), P_variable(i));
end

% Visualization
figure;
plot(1:n, P_fixed, 'b-o', 'LineWidth', 1.5);
hold on;
plot(1:n, P_variable, 'r-s', 'LineWidth', 1.5);
xlabel('Months');
ylabel('Savings Amount');
title('Savings Growth with Fixed and Variable Interest Rates');
legend('Fixed Rate', 'Variable Rate');
grid on;
```

(The MATLAB script implements both the fixed and variable interest rate models. It computes the savings growth and visualizes the results)

4. Results and Discussion



```
>> savingaccount
>> savingaccount
Fixed Interest Model:
P(n) = P0 * (1 + r)^n
P(1) = 1000.00 * (1 + 0.05)^1 = 1050.00
P(2) = 1000.00 * (1 + 0.05)^2 = 1102.50
P(3) = 1000.00 * (1 + 0.05)^3 = 1157.63
P(4) = 1000.00 * (1 + 0.05)^4 = 1215.51
P(5) = 1000.00 * (1 + 0.05)^5 = 1276.28
P(6) = 1000.00 * (1 + 0.05)^6 = 1340.10
P(7) = 1000.00 * (1 + 0.05)^7 = 1407.10
P(8) = 1000.00 * (1 + 0.05)^8 = 1477.46
P(9) = 1000.00 * (1 + 0.05)^9 = 1551.33
P(10) = 1000.00 * (1 + 0.05)^10 = 1628.89
P(11) = 1000.00 * (1 + 0.05)^11 = 1710.34
P(12) = 1000.00 * (1 + 0.05)^12 = 1795.86

Variable Interest Model:
P(n) = P(n-1) * (1 + r_n)
P(2) = P(1) * (1 + 0.05) = 1000.00 * 1.05 = 1050.00
P(3) = P(2) * (1 + 0.05) = 1050.00 * 1.05 = 1102.50
P(4) = P(3) * (1 + 0.04) = 1102.50 * 1.04 = 1146.60
P(5) = P(4) * (1 + 0.04) = 1146.60 * 1.04 = 1192.46
P(6) = P(5) * (1 + 0.04) = 1192.46 * 1.04 = 1240.16
P(7) = P(6) * (1 + 0.06) = 1240.16 * 1.06 = 1314.57
P(8) = P(7) * (1 + 0.06) = 1314.57 * 1.06 = 1393.45
P(9) = P(8) * (1 + 0.06) = 1393.45 * 1.06 = 1477.05
P(10) = P(9) * (1 + 0.03) = 1477.05 * 1.03 = 1521.37
P(11) = P(10) * (1 + 0.03) = 1521.37 * 1.03 = 1567.01
P(12) = P(11) * (1 + 0.03) = 1567.01 * 1.03 = 1614.02
>>
```



- The simulation results indicate that savings grow exponentially with a fixed interest rate, following the formula:

$$P(n) = P_0 \times (1 + r)^n$$

This ensures consistent and predictable growth over time.

- When the interest rate varies every 3 months, the growth pattern changes dynamically based on the applied rates. The formula used is:

$$P(n) = P(n-1) \times (1 + r_n)$$

Since different rates are applied at different periods, the growth trajectory fluctuates accordingly.

- The variable interest rate model demonstrates how savings may experience periods of faster or slower growth depending on the assigned interest rates. This highlights the impact of market fluctuations on long-term savings.

- Comparing both models provides insight into the influence of compounding rates. The fixed rate model offers stability and predictability, while the variable rate model reflects realistic financial conditions where interest rates change over time.

---

5. Conclusion

This study demonstrates that **compounding interest plays a crucial role in savings growth**. A **higher or stable interest rate** yields better returns over time.

Future work could include **real-world economic factors** such as **inflation and banking fees** to make the model more realistic.

Model 8: Stochastic Model for Mutual Fund Growth

1. Introduction

Mutual fund investments are influenced by **variable interest rates**, which fluctuate due to economic factors such as inflation, market trends, and government policies. This **stochastic model** captures these uncertainties by allowing the interest rate r(n) to vary randomly according to a probability distribution.

---

2. Mathematical Model

The model follows the recurrence relation:

$$P_{n+1} = P_n(1 + r(n))$$

where:

- Pn = Account balance at time step nnn
- r(n) = Random interest rate at month nnn, drawn from a probability distribution

The interest rate r(n) can follow a **normal distribution** (realistic for market fluctuations) or a **uniform distribution** (for simplified simulations).

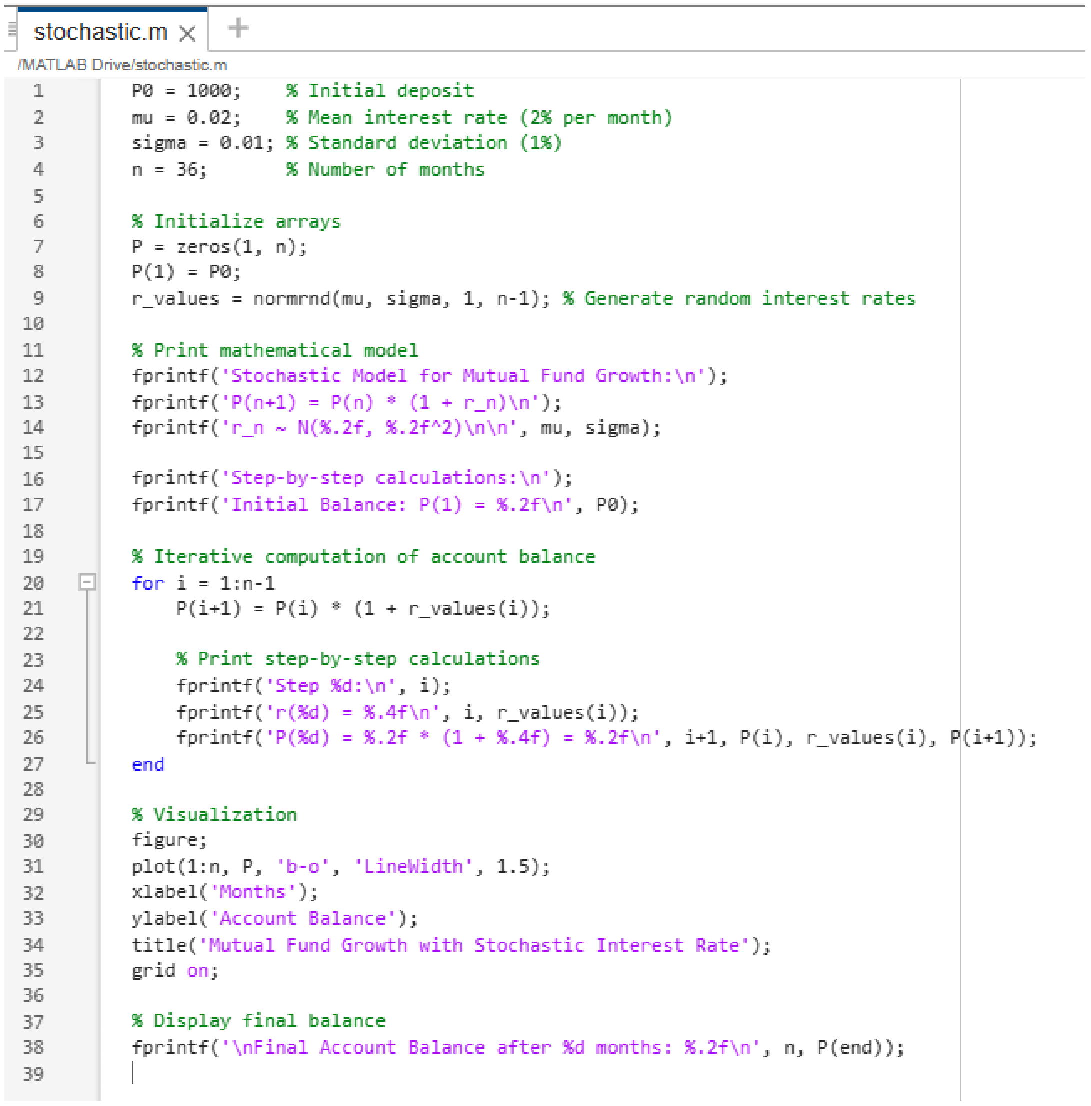
---

3. MATLAB Simulation

Assumptions:

- Initial deposit: P0=1000
- Interest rate follows a **normal distribution** with mean μ=0.02 (2%) and standard deviation σ=0.01 (1%).
- Simulation period: **36 months (3 years)**

MATLAB Code:



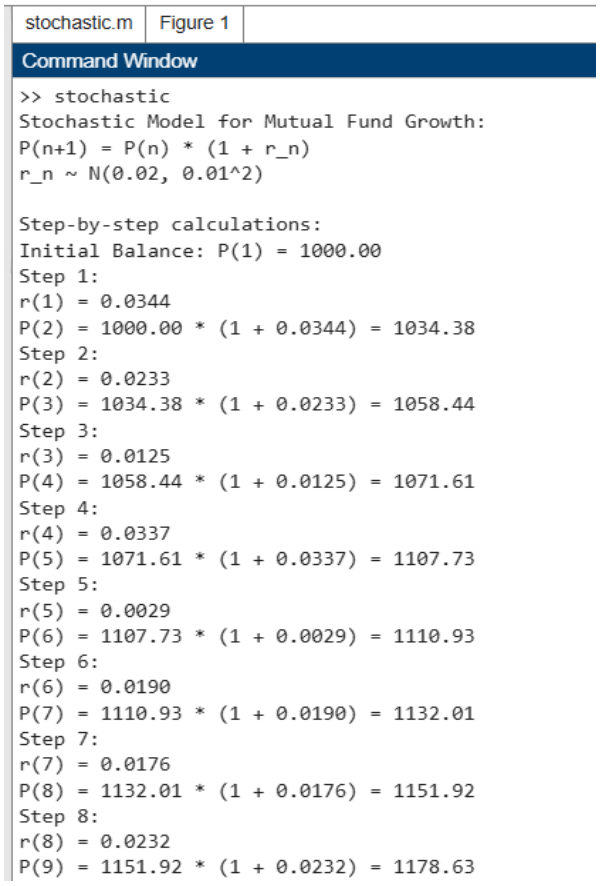
stochastic.m

/MATLAB Drive/stochastic.m

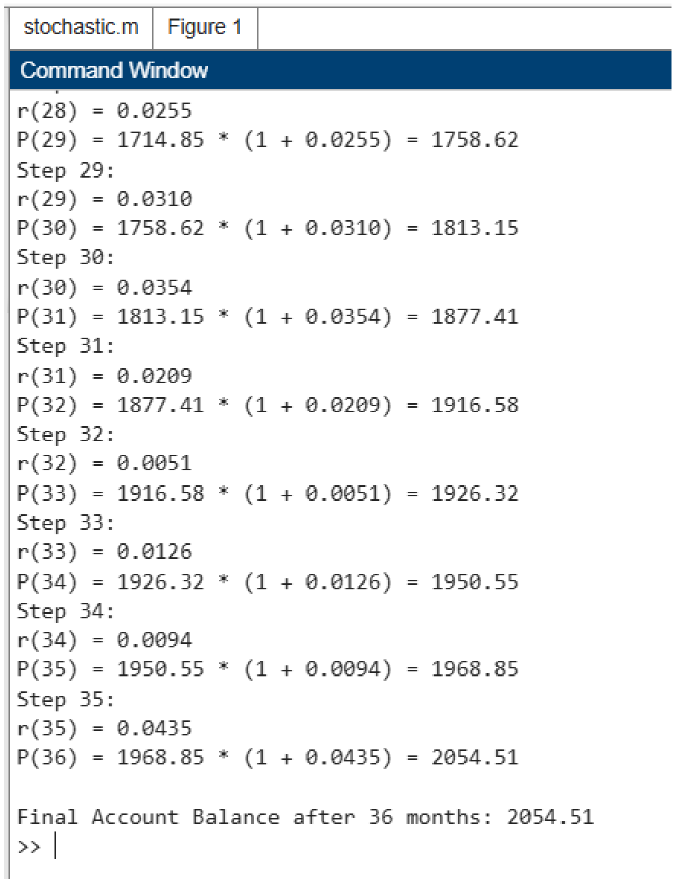
```
P0 = 1000;    % Initial deposit
mu = 0.02;    % Mean interest rate (2% per month)
sigma = 0.01; % Standard deviation (1%)
n = 36;       % Number of months

% Initialize arrays
P = zeros(1, n);
P(1) = P0;
r_values = normrnd(mu, sigma, 1, n-1); % Generate random interest rates

% Print mathematical model
fprintf('Stochastic Model for Mutual Fund Growth:\n');
fprintf('P(n+1) = P(n) * (1 + r_n)\n');
fprintf('r_n ~ N(%.2f, %.2f^2)\n\n', mu, sigma);

fprintf('Step-by-step calculations:\n');
fprintf('Initial Balance: P(1) = %.2f\n', P0);

% Iterative computation of account balance
for i = 1:n-1
    P(i+1) = P(i) * (1 + r_values(i));

    % Print step-by-step calculations
    fprintf('Step %d:\n', i);
    fprintf('r(%d) = %.4f\n', i, r_values(i));
    fprintf('P(%d) = %.2f * (1 + %.4f) = %.2f\n', i+1, P(i), r_values(i), P(i+1));
end

% Visualization
figure;
plot(1:n, P, 'b-o', 'LineWidth', 1.5);
xlabel('Months');
ylabel('Account Balance');
title('Mutual Fund Growth with Stochastic Interest Rate');
grid on;

% Display final balance
fprintf('\nFinal Account Balance after %d months: %.2f\n', n, P(end));
```

(This MATLAB script models savings growth with a randomly varying interest rate, printing step-by-step calculations and visualizing the results)

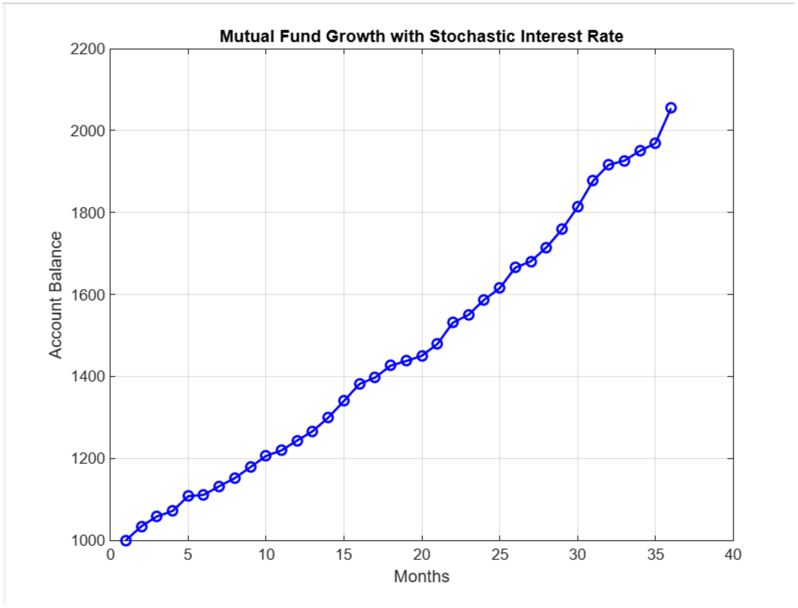
4. Results and Discussion



```
>> stochastic
Stochastic Model for Mutual Fund Growth:
P(n+1) = P(n) * (1 + r_n)
r_n ~ N(0.02, 0.01^2)

Step-by-step calculations:
Initial Balance: P(1) = 1000.00
Step 1:
r(1) = 0.0344
P(2) = 1000.00 * (1 + 0.0344) = 1034.38
Step 2:
r(2) = 0.0233
P(3) = 1034.38 * (1 + 0.0233) = 1058.44
Step 3:
r(3) = 0.0125
P(4) = 1058.44 * (1 + 0.0125) = 1071.61
Step 4:
r(4) = 0.0337
P(5) = 1071.61 * (1 + 0.0337) = 1107.73
Step 5:
r(5) = 0.0029
P(6) = 1107.73 * (1 + 0.0029) = 1110.93
Step 6:
r(6) = 0.0190
P(7) = 1110.93 * (1 + 0.0190) = 1132.01
Step 7:
r(7) = 0.0176
P(8) = 1132.01 * (1 + 0.0176) = 1151.92
Step 8:
r(8) = 0.0232
P(9) = 1151.92 * (1 + 0.0232) = 1178.63
```



```
r(28) = 0.0255
P(29) = 1714.85 * (1 + 0.0255) = 1758.62
Step 29:
r(29) = 0.0310
P(30) = 1758.62 * (1 + 0.0310) = 1813.15
Step 30:
r(30) = 0.0354
P(31) = 1813.15 * (1 + 0.0354) = 1877.41
Step 31:
r(31) = 0.0209
P(32) = 1877.41 * (1 + 0.0209) = 1916.58
Step 32:
r(32) = 0.0051
P(33) = 1916.58 * (1 + 0.0051) = 1926.32
Step 33:
r(33) = 0.0126
P(34) = 1926.32 * (1 + 0.0126) = 1950.55
Step 34:
r(34) = 0.0094
P(35) = 1950.55 * (1 + 0.0094) = 1968.85
Step 35:
r(35) = 0.0435
P(36) = 1968.85 * (1 + 0.0435) = 2054.51

Final Account Balance after 36 months: 2054.51
>>
```



• The account balance fluctuates monthly based on randomly generated interest rates.
• The stochastic model reflects real-world investment behavior, where returns vary unpredictably.
• In this simulation, the generated interest rates influenced the savings trajectory, sometimes increasing and sometimes decreasing growth.
• Higher standard deviation (**σ**) in the interest rate led to more significant fluctuations in the final balance.
• Despite randomness, the overall trend demonstrates the impact of compounding over time.

---

5. Conclusion

This **stochastic model** provides a more **realistic simulation** of mutual fund investments compared to fixed-interest models. **Monte Carlo simulations** can further analyze different economic scenarios and optimize investment strategies.

Conclusion:

This report has presented eight distinct mathematical models, each illustrating different dynamic systems and their behavior over time. Through mathematical analysis and MATLAB simulations, we have explored how system parameters influence outcomes and have identified optimal conditions for various scenarios.

The simulations have shown the importance of mathematical modeling in predicting system behavior, optimizing performance, and making informed decisions. Each model highlights a different aspect of real-world processes, from biological systems to engineering applications.

Future work could focus on refining these models by incorporating more complex factors, using machine learning for predictive analysis, or expanding simulations to include real-world data. Overall, this collection of models demonstrates the critical role of mathematical modeling and computational simulations in scientific research and practical applications.