CHAPTER 1

OVERVIEW OF DISCRETE DYNAMICAL MODELING AND MATLAB®

1.1. INTRODUCTION TO MODELING AND DIFFERENCE EQUATIONS

In this section we introduce dynamical systems, discuss discrete dynamical systems vs. continuous dynamical systems and informally define a mathematical model.

1.1.1. Model 1.1: Population Dynamics, A Discrete Dynamical System

Consider the population of a city with a constant growth rate per year. The population is counted at the end of each year. For simplicity, assume that there is no immigration to or emigration from the city.

i. Model the population dynamic and predict the long-term behavior of the system.
ii. In 2010, the city's population was 100,000. The natural annual growth rate of the population is 1% per year. Predict the city's population in 2020. Estimate the population over the next 30 years and graph it. What is the long-term behavior of the population?

Explorations of Mathematical Models in Biology with MATLAB®, First Edition. Mazen Shahin.

Discussion

i. We will measure the population at discrete time intervals in one-year units. Let

p_n = population size at the end of the time period (year), n.
p_0 = the initial population size, 0.
r = the constant growth rate per period (year).

The relationship between the current population, p_n, and the next population, p_{n+1}, is

$$p_{n+1} = p_n + rp_n.$$
$$p_{n+1} = (1+r)p_n. \tag{1.1}$$

Therefore, the population dynamics can be modeled by equation 1.1.

Equation 1.1 is a **difference equation** (or recurrence equation). The system 1.1 and the initial value, p_0, represent the population dynamics. Because the population changes over time, this system is a **dynamical system**. Because this dynamical system changes over discrete time intervals, the system is called a **discrete dynamical system**. We say that the population dynamics is **modeled** by the discrete dynamical system (or the difference equation 1.1).

To find p_k, use p_0 in equation 1.1 to find p_1, then use p_1 to find p_2 and so on until p_k. This process is called **iteration** of the difference equation 1.1, and the sequence 1.2,

$$p_0, p_1, p_2, \ldots, p_k, \tag{1.2}$$

for any value of k (positive integer) is called a **solution** or **numerical solution** of the given difference equation 1.1.

From equation 1.1, if the current value of p_n is known, the next value, p_{n+1}, can be calculated. For example, if we have p_5 we can calculate p_6. However, if we have p_0, equation 1.1 does not allow us to calculate, for example, p_6 in one step. Therefore, we are in need of a closed form to calculate p_n in one step if we know the values of p_0 and n. It can be easily proven that

$$p_n = (1+r)^n p_0. \tag{1.3}$$

Equation 1.3 is called the **analytical solution** of the difference equation 1.1.

Equation 1.3 is an exponential function and will grow or decay exponentially, depending on the value of r. If $r>0$, then $(1+r)>1$, and therefore population size, p_n, grows unbounded when n is very large. If $r<0$, then $(1+r)<1$, and consequently, the population size approaches zero when n is very large.

ii. Let's apply the model just discussed to the given information, where $r = 0.01$ and $p_0 = 100{,}000$. The city's population is modeled by the system

$$p_{n+1} = 1.01p_n, \quad p_0 = 100{,}000. \tag{1.4}$$

To find the population in 2020 (10 years from 2010), we use equation 1.3 with $n = 10$ and $p_0 = 100{,}000$. We are looking for p_{10}. We have

$$p_{10} = (1.01)^{10}100{,}000 = 110{,}462.$$

One way to find the values of $p_1, p_2, \ldots, p_{30}$ is to iterate equation 1.4. Then graph the ordered pairs (n, p_n). To illustrate how the iteration works, let's see how to calculate, for example, p_3. Set $n = 0$ in equation 1.4 to get p_1,

$$p_1 = 1.01p_0 = 1.01(100{,}000) = 101{,}000.$$

Then set $n = 1$ in equation 1.4 to get p_2,

$$p_2 = 1.01p_1 = 1.01(101{,}000) = 102{,}010.$$

Finally, setting $n = 2$ in equation 1.4 gives p_3,

$$p_3 = 1.01p_2 = 1.01(102{,}010) = 103{,}030.$$

In particular, to find P_{10}, the answer to the original question, we can use the difference equation 1.4 and the initial condition to find the sequence $p_0, p_1, p_2, \ldots, p_{10}$ which is 100,000; 101,000; 102,010; 103,030; 104,060; 105,101; 106,152; 107,213; 108,285; 109,368; 110,462. Thus $p_{10} = 110{,}462$.

Usually we use MATLAB to iterate a difference equation. We will introduce MATLAB in Section 1.3. The graph of p_n, $n = 0, 1, 2, \ldots, 30$ vs. n is shown in Figure 1.1.

The analytical solution and numerical solution of equation 1.4 show that the city population slowly grows unbounded as n becomes very large.

1.1.2. Model 1.2: Population Dynamics, A Continuous Dynamical System

Consider the following situation. There are some bacteria in a tube with a nutritive solution. As time progresses, the bacteria reproduce by splitting and dying. Assuming that there is enough food and space for the bacteria, model the dynamics of the bacteria. Investigate the long-term behavior of the model.

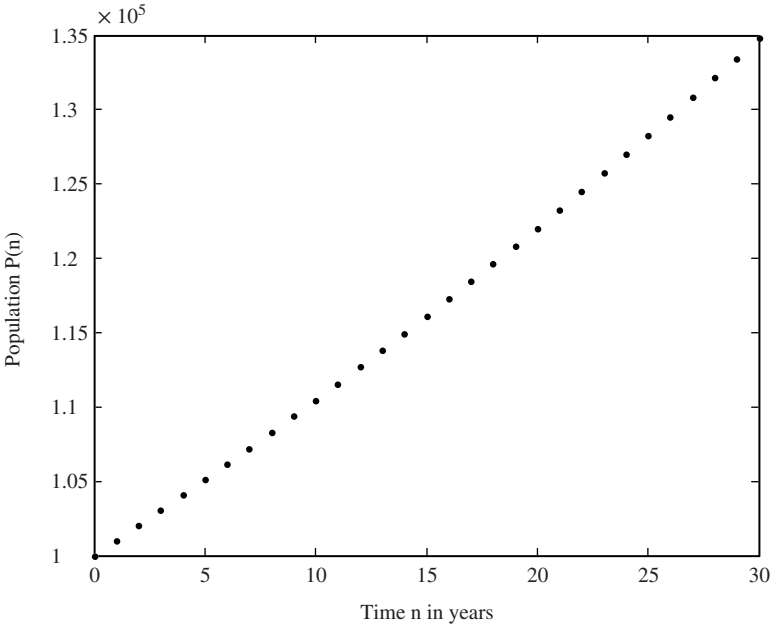

FIGURE 1.1. Graph of a city's population after n years, p_n, vs. time, n, in years. The population is modeled by the difference equation $p_{n+1} = 1.01p_n$, with the initial population $p_0 = 100{,}000$ and $n = 0, 1, \ldots, 30$.

Discussion

Let $p(t)$ be the bacteria's population size (number of bacteria) at time t and $p(0) = p_0$. Assume that the growth rate $r = b - d$, where b is the birth rate and d is death rate. The assumption that there is enough food and space means there is no restriction on the increasing number of bacteria. Therefore, the rate of change of bacteria's population size $\frac{dp}{dt}$ is proportional to the bacteria's population p. Consequently, the dynamic of the bacteria is modeled by the dynamical systems 1.5 and 1.6:

$$\frac{dp}{dt} = (b - d)p = rp. \tag{1.5}$$

$$p(0) = p_0. \tag{1.6}$$

Equation 1.5 is a **first-order ordinary differential equation** and equation 1.6 is called the initial value (condition). From the basics of differential calculus or differential equations, the solution of systems 1.5 and 1.6 is

$$p(t) = e^{rt}p_0. \tag{1.7}$$

Knowing the values of r and p_0, the population, $p(t)$, can be evaluated at any time t. For $r > 0$, equation 1.7 implies that the population size, $p(t)$, increases and grows

unbounded as $t \to \infty$, while $r < 0$ implies that the population size decreases and approaches zero as $t \to \infty$.

Because the change in the system 1.5 is continuous, this system is called a **continuous dynamical system**. Usually continuous dynamical systems are represented by one or more ordinary or partial differential equations. Note that the models discussed in this text are restricted to discrete dynamical systems. In other words, the text discusses only those models represented by difference equations.

1.1.3. Why Modeling with Difference Equations is Adopted

There are compelling reasons to restrict models in this text to difference equations, including

1. Modeling with difference equations is a very powerful, yet simple tool for modeling dynamical systems in biology, ecology, the environment, and chemistry.
2. Modeling with difference equations requires knowledge of algebra but does not require knowledge of differential calculus; modeling with differential equations requires a course in differential equations. This text targets freshman and sophomores majoring in the life sciences and mathematics, many of whom have not had a differential equations course.

1.1.4. What is a Mathematical Model?

The following is a possible informal definition of a mathematical model:

> A mathematical model is a translation of a real-world problem into mathematical notation by forming a mathematics problem corresponding to the real-world problem. Then mathematics tools, ideas, concepts, and techniques are used to solve the mathematics problem. The obtained solution is translated back into the real-world problem context.

1.1.5. Basic Terminology of Difference Equations

The **order** of a difference equation is equal to the difference between the largest and the smallest indices in the difference equation. For example, in the difference equations 1.1 and 1.4, the difference between the largest subscript, $n+1$, and the smallest subscript, n, is 1. Therefore each of equations 1.1 and 1.4 is a **first-order difference equation**. The difference equation 1.8 is a **second-order difference equation**,

$$x_{n+2} = x_{n+1} + x_n. \tag{1.8}$$

Examples of first-order difference equations are

$$y_n = 0.8y_{n-1} + 20. \tag{1.9}$$

$$p_n = 1.02p_{n-1}. \tag{1.10}$$

$$x_{n+1} = 1.2x_n - 0.15n. \tag{1.11}$$

$$z_n = n^2 z_{n-1} + 2^n. \tag{1.12}$$

$$B_{n+2} = \frac{4}{n}B_{n+1} - n^{1/2}. \tag{1.13}$$

$$y_{n+1} = 3y_n^2 + 3n. \tag{1.14}$$

$$x_n = \sqrt{x_{n-1}} - 23. \tag{1.15}$$

$$y_{n+1} = y_{n+1}y_n + 6. \tag{1.16}$$

A difference equation is called **linear** if its terms are not raised to a power other than 1 and if the terms are not multiplied together. Otherwise the difference equation is called **nonlinear**. For example, the difference equations 1.1, and 1.8–1.13 are linear. The difference equation 1.14 is nonlinear because y_n is raised to the power of 2. Equation 1.15 is nonlinear because x_{n-1} is raised to the of power $\frac{1}{2}$. In equation 1.16, two terms, y_{n+1} and y_n, are multiplied together; therefore it is nonlinear.

A **first-order linear difference equation** can be represented by the equation

$$y_{n+1} = a_n y_n + b_n, \tag{1.17}$$

where a_n and b_n are two known sequences, and a_n is a coefficient. Note that each of the sequences a_n and b_n can be a constant sequence or can depend on n. When the sequence a_n is a constant, such as a, the difference equation

$$y_{n+1} = ay_n + b_n \tag{1.18}$$

is called a **first-order linear difference equation with constant coefficients**. When $b_n = 0$, the difference equation is called **homogenous** and equation 1.17 becomes

$$y_{n+1} = a_n y_n. \tag{1.19}$$

Equation 1.19 is called a **first-order linear homogenous difference equation**. If $a_n = a$, where a is a constant, equation 1.19 becomes

$$y_{n+1} = ay_n \tag{1.20}$$

and is called a **first-order linear homogenous difference equation with constant coefficients**. For example, the difference equations 1.1, 1.4 and 1.10 are first-order linear homogenous difference equations with constant coefficients, and equations 1.9 and 1.11 are first-order linear difference equations with constant coefficients.

1.1.6. Exercises 1.1

1. Suppose that the initial population of a species is 10,000 and the growth rate is 5% per year.

A. Model this situation by a difference equation.

B. Find the first five populations.

C. Find the population size after 20 years.

2. Assume that the population of a country is 100 million and the natural growth rate is 2% per year. Assume that 20,000 immigrants are allowed into the country every year.

A. Model this situation by a difference equation.

B. Find the population for the next 4 years.

C. Find the population after 10 years.

D. Can you predict the long-term behavior of the system?

3. Assume that the kidneys remove 25% of an anesthetic substance from the body every hour. A patient is injected with 600 mg of the substance before a minor surgery on her tooth.

A. Model this situation by a difference equation.

B. Determine the amount of the substance in the patient's body after the first 4 hr.

C. What is the long-term behavior of the system?

4. In 2000, a lake contained 800 lb of contamination. In the same year, a new plant started to dump 120 lb of the contaminant into the lake every year. Assume that 15% of the contaminant in the lake is naturally removed from the lake every year. Assume that this trend continues.

A. Model the amount of contaminant in the lake at any year after 2000.

B. Determine the amount of the contaminant in the lake in years 2001–2006.

C. Can you determine the long-term behavior of the system?

5. Identify the following difference equations as linear or nonlinear.

A. $x_{n+1} = 3x_n + 2$

B. $y_n = 2y_{n-1} - 4n$

C. $y_{n+1} = 4\sqrt{y_n} + 5$

D. $x_{n+1} = x_n x_{n-1} + 6$

E. $z_{n+2} = 2z_{n+1} - 3$

F. $x_{n+1} = 3x_n^2 + 4$

6. Determine the order of the following difference equations.

A. $x_{n+1} = 2x_n + 6n$

B. $x_{n+1} = 4x_n - 2x_{n-1}$

C. $y_{n+2} = 4\sqrt{y_n} + 5$

D. $z_{n+2} = 2nz_{n+1} - 4z_n$

E. $z_{n+2} = 2z_{n-1} - 3$

F. $y_n = y_{n-2}^2 + 4$

7. Determine whether the following difference equations are homogenous or nonhomogenous

A. $x_{n+1} = 3x_n + 6n$

B. $y_{n+2} = 4y_{n+1} - 2y_n$

C. $x_{n+1} = 2x_n + n^2 + 5n + 2$

D. $y_{k+3} = 5y_{k+1} + 6$

E. $z_n = n^2 z_{n-1} - nz_{n-2}$

F. $z_{n+2} = n^3 z_{n+1} + 4z_n + 2n$

8. Identify the order, linearity, and homogeneity of each of the following difference equations.

A. $x_{n+3} = 2x_{n+1} + 3x_n - n^2$

B. $y_{n+2} = 4x_{n+1}x_n - 4$

C. $z_{n+1} = 3z_n + 4n$

D. $P_{n+2} = 2nP_n$

E. $P_{n+1} = P_n + aP_n - bP_n^2$, where a and b are constants.

F. $x_{n+1} = rx_n(1 - x_n)$, where r is a constant

1.2. THE MODELING PROCESS

It is useful to view mathematical modeling as a process, as illustrated in Figure 1.2 The modeling process is represented by a loop, where the starting point is Step 1, located in the upper-left-hand corner of the figure.

1.2.1. Step 1: Formulate the Real-World Problem

The first step is to get information pertaining to the system under consideration and identify the question(s) to be answered. The question should be neither too general nor too narrow. If the formulated question is too general, it is difficult to manage the problem; if the question is too narrow, the problem might become trivial. Because the question will be translated into mathematical notation, it should be stated in precise mathematical terms.

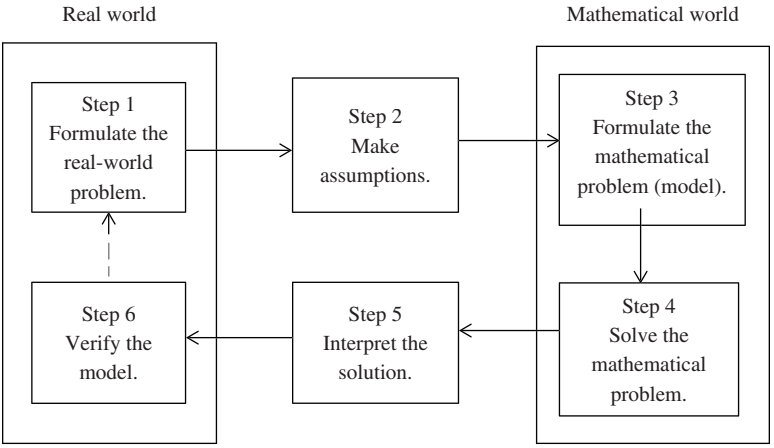

FIGURE 1.2. The mathematical modeling process.

As an example we consider two species in a forest, rabbits and foxes, where foxes eat rabbits and there is enough food for rabbits. We are interested in:

1. Determining whether the rabbits and foxes could co-exist in this environment.
2. Finding the equilibrium values of the system and determining if they are stable, unstable, or semistable.
3. Modeling the dynamics of the interaction between the predator foxes and the prey rabbits, so we can predict the populations of rabbits and foxes at any year.
4. Investigating the long-term behavior of the two species.

1.2.2. Step 2: Make Assumptions

It is very important to state the assumptions clearly. The construction of a mathematical model greatly depends on the assumptions. It is clear that a change in the assumptions would result in a different model. It is advisable to simplify assumptions, at least at the beginning, to make the model manageable. Therefore, some assumptions are made to simplify the model.

We will use the predator–prey example to illustrate step 2. Because the forest is a complex ecosystem, we need to make the following assumptions to simplify the predator–prey model:

i. There is enough food for the rabbits and the population of rabbits increases by a constant rate. That is, the rabbits' population increases exponentially.
ii. The population of rabbits decreases as a result of the interactions between rabbits and foxes.

iii. The rabbits are the only source of food for foxes. Therefore, in the absence of rabbits the population of decreases by a constant rate and dies out. That is, the foxes' population decreases exponentially.

iv. The population of foxes increases as a result of the interactions between rabbits and foxes.

v. The rabbits and foxes live in a closed environment. That is, that there is no interaction between these two species and other species, there is no emigration from or immigration to the forest, and there is no harvesting or hunting.

1.2.3. Step 3: Formulate the Mathematical Problem

In step 3 we enter the mathematics world. In the first part of this step we need to choose mathematical symbols for the variables and parameters. Recall that variables are quantities that change within the problem, whereas the parameters are constant within a problem.

For example, in the predator–prey example we might choose the following variables:

R_n = the population of the prey rabbits at the time period n.

F_n = the population of predator foxes at time period n.

n = the time periods in years, $n = 0, 1, 2, \ldots, k$.

For the parameters we might choose:

a = the natural growth rate of rabbits in the absence of foxes, and $a > 0$.

b = the death rate of rabbits as a result of the presence of foxes, and $b > 0$.

c = the natural decay rate of foxes in the absence of rabbits, and $c > 0$.

d = the growth factor of foxes due to the presence of rabbits, and $d > 0$.

In the second part of step 3 we use the assumptions made in step 2 and the variables and parameters defined in the first part of step 3 to formulate the problem in mathematical notation. As a result of the formulation, the problem might be represented by a single algebraic, difference, differential, or matrix equation or by a system of algebraic, difference, or differential equations. The problem might be represented by an algorithm, and so on.

For example, the predator–prey model may be represented by a system of the following linear difference equations

$$R_{n+1} = R_n + aR_n - bF_n. \tag{1.21}$$

$$F_{n+1} = F_n - cF_n + dR_n. \tag{1.22}$$

or the following system of two nonlinear difference equations

$$R_{n+1} = R_n + aR_n - bR_nF_n. \tag{1.23}$$

$$F_{n+1} = F_n - cF_n + dR_nF_n. \tag{1.24}$$

It is necessary to use the selected representation to answer the posed questions for the real-world problem.

1.2.4. Step 4: Solve the Mathematical Problem (Model)

In step 4 we use appropriate available mathematical, computational, or graphical tools and techniques to solve the mathematical problem (model). The solution might be an analytical solution (a closed-form mathematical expression), a numerical solution, or a graph. It might also be the implementation of an algorithm or the running/testing of a simulation.

For example, there is an analytical solution of the predator–prey model represented by linear equations 1.21 and 1.22 in the form

$$X_n = T^n X_0,$$

where $T = \begin{bmatrix} 1+a & -b \\ d & 1-c \end{bmatrix}$, $X_n = \begin{bmatrix} R_n \\ F_n \end{bmatrix}$, and $X_0 = \begin{bmatrix} R_0 \\ F_0 \end{bmatrix}$ is the initial distribution vector.

The matrix algebra allows us to investigate the solution fully. However, the modeler will realize that the linear representations 1.21 and 1.22 of the predator–prey model are unrealistic.

The predator–prey model is a realistic one if it is represented by the nonlinear difference equations 1.23 and 1.24. Therefore, we will focus our attention on answering the questions of the real-world problems using the nonlinear representation. However, for nonlinear equations there is no analytical solution.

It can be easily shown that this system has two equilibrium values, say R_e and F_e, where

$$R_e = \frac{c}{d} \quad \text{and} \quad F_e = \frac{a}{b}.$$

Using MATLAB it can be concluded that these equilibrium values are unstable. With MATLAB we create two types of graphs: time series graphs (R_n and F_n vs. n) and phase plane graphs (F_n vs. R_n or R_n vs. F_n), for different values of the parameters a, b, c, and d and the initial values R_0 and F_0. Answers to the posed questions and the system's long-term behavior can be extracted from these graphs.

1.2.5. Step 5: Interpret the Solution

In step 5, the answers to the mathematical problem need to be interpreted in terms of the context of the real-world problem. The modeler must check the answers to ensure that the model answered the original real-world problem within the assumptions made in step 2 and the initial conditions.

In our predator–prey example the following might be interpretations to solutions obtained in step 4:

- The rabbits and foxes may co-exist together without extinction of one of the species. At certain values of the rabbits and foxes, there is no change in the species' populations. This interpretation is a conclusion of the existence of equilibrium values of rabbits and foxes.
- The equilibrium values of the rabbits and foxes are sensitive to small change. In other words, a small change to the equilibrium values makes the rabbits and foxes populations diverge from the equilibrium values. This interpretation is a conclusion from the instability of the equilibrium values established in step 4.

1.2.6. Step 6: Verify the Model

Next, it is necessary to verify the validity of the model. A common verification method is to compare the model's predicted results with known real-world data or with data obtained from an experiment designed to test the model. Be sure that all the necessary variables were used and all the assumptions were incorporated.

If the outcome of the verification is unsatisfactory, the modeler needs to refine the model, which is improving it. To refine the model you need to reexamine the modeling process starting with step 1. Be sure that you did not omit any necessary assumptions and variables. Check that the mathematical problem accurately represents the real-world problem. Check for the correctness of solving the mathematical problem.

If the verification of the model in step 6 is satisfactory, the modeler writes a report on the model, if he or she is required to submit one. The report should follow the requested format.

1.2.7. Exercises 1.2

In the following problems, real-world situations are briefly stated without specifics. For each situation do the following:

A. Formulate the real-world problem.
B. Make assumptions. If you made assumptions to simplify the problem, state them at the beginning.
C. Choose mathematical symbols for the variables and parameters.
D. Formulate the mathematical problem. Note that you are not asked to solve the problem.

1. The dynamics of the population of the United States of America.
2. The dynamics of the population of a single species.
3. The dynamics of the population of a single species, such as deer, with hunting.
4. The dynamics of the interaction of two species competing for the same food, such as foxes and wolves who compete for rabbits.
5. The dynamics of two interacting species, predator and prey, such as falcons and rats.
6. Obtaining the maximum sustainable yield of a natural renewable resource, such as a colony of whales.
7. Administering a drug, such as antibiotic, for a patient.
8. The dynamics of the spread of a contagious disease, such as flu, among the students of a college.

1.3. GETTING STARTED WITH MATLAB

In this section we introduce the basics for getting started with MATLAB. This introduction is by no means a reference or manual for the program. It is simply a quick start guide to writing simple commands, iterating difference equations, and producing graphs.

To start MATLAB, double click on the MATLAB icon on computer. You should see a screen broken into five parts (panels). The main part in the middle is the **Command Window**. On the left-hand side are two panels, **Current Folder** and **Details**. On the right-hand side are two panels, **Workspace** and **Command History**.

In the Command Window you enter the commands and get answers or error messages.

The MATLAB prompt is >>

Three main ways to do computations are

- Enter (type) the commands in the Command Window
- Use commands saved in a **script file**.
- Use a **function**, where the function code is saved in an **M-flie**.

1.3.1. Simple Arithmetic and Definition of Variables

The basic arithmetic operators are

+	addition
–	subtraction
*	multiplication
/	division
^	power operator

Examples

To set the variable x to be 3 type x = 3 and hit enter key.

```
>> x=3
x=
   3
>> y=4;                    % The ; suppresses the output
>> x+y
ans=
   7
>> z=sqrt(x^2 + y^2)       % sqrt is the square root function
z=
   5
```

1.3.2. Vectors and Matrices

Vectors can be defined in two ways. The first method is to enter the elements of a vector in square brackets separated by spaces or by a comma (,):

```
>> v=[3  8  6  1]          % To enter the vector v = (3, 8, 6, 1)
v=
   3   8   6   1
>> u=[10, 12, 16]
u=
   10   12   16
```

The second method for defining vectors uses elements going in even steps from one value to another value

```
>> a=4:9                   % The vector a goes from 4 to 9 in increments of 1
a=
   4   5   6   7   8   9
>> b=5:3:22                % The vector b goes from 5 to 22 in increments of 3
b=
   5   8   11   14   17   20
>> c=1:0.5:4               % The vector c goes from 1 to 4 in increments of 0.5
c=
   1.0000 1.5000 2.0000 2.5000 3.0000 3.5000 4.000

>> v(1)                    % The first element of the vector v
ans=
   3
>> v(4)                    % The fourth element of the vector v
```

```
ans =
   1
>> v(5) = 9                    % Additional elements can be added to a vector
v =
   3   8   6   1   9
>> b(2:5)                      % The subvector of b from the 2nd to 5th
                               % element
ans =
   8   11   14   17
>> x=[5 8 6]; y=[20 33];
>> z=[x y]                     % A new vector can be defined by previously
z =                            % defined vectors
   5   8   6   20   33
>> length(z)
ans =
   5
>> m=[3; 6; 9]                 % To define column vector m = [3; 6; 9]

m =
   3
   6
   9
```

Matrices are defined by entering the rows separated by semicolons (;), and the entries of a row are separated by spaces or by a comma (,). The matrix entries are enclosed in square brackets.

For example matrices $A = \begin{bmatrix} 2 & 6 & 7 \\ 4 & 8 & 10 \end{bmatrix}$ and $B = \begin{bmatrix} -2 & 3 & 1 \\ 2 & -4 & -3 \end{bmatrix}$ may be entered as:

```
>> A=[2 6 7; 4 8 10]           % The elements of each row are separated
>>                             % by spaces
A =
   2 6  7
   4 8 10
>> B=[-2, 3, 1; 2, -4, -3]     % The elements of rows are separated by ,
B =
   -2   3   1
    2  -4  -3
>> A(2, 3)                     % The element in row 2 and column 3 of matrix A
ans =
   10
>> A(2, :)                     % To extract the 2nd row of matrix A
```

```
ans =
   4  8  10
>> A(:, 2)                    % To extract the 2nd column of matrix A
ans =
   6
   8
>> A=[A; 1 3 5]               % To add the row [1 3 5] at the end of matrix A
A =
   2 6  7
   4 8  10
   1 3  5
>> B=[B [6; 8]]               % To add the column [6; 8] to matrix B
B =
  -2  3  1 6
   2 -4 -3 8
>>
```

1.3.3. Iteration

Let us consider the following situation. The population of a species increases every year by 10%. Letting y_n be the population at the end of n years and $y_1 = 100$ be the initial population, this situation is modeled by the recurrence relation (or difference equation)

$$y_{n+1} = 1.1y_n, \qquad y_1 = 100, \quad n = 1, 2, \ldots \tag{1.25}$$

To obtain the sequence $y_1, y_2, y_3, \ldots, y_7$, we use y_1 to get y_2, use y_2 to get $y_3, \ldots,$ use y_6 to get y_7. To obtain the first 6 values of y_n after the initial value y_1 we use the difference equation (recurrence relation) 1.25 recursively. We have

$$\text{For } n = 1, \quad y_2 = 1.1y_1 = 1.1(100) = 110.$$
$$\text{For } n = 2, \quad y_3 = 1.1y_2 = 1.1(110) = 121.$$
$$\text{For } n = 3, \quad y_4 = 1.1y_3 = 1.1(121) = 133.10.$$
$$\text{For } n = 4, \quad y_5 = 1.1y_4 = 1.1(133.10) = 146.41.$$
$$\text{For } n = 5, \quad y_6 = 1.1y_5 = 1.1(146.41) = 161.051.$$
$$\text{For } n = 6, \quad y_7 = 1.1y_6 = 1.1(161.051) = 177.1561.$$

This process is called *iteration* and can be obtained by a **for** loop in MATLAB.

One way to write code to iterate difference equation 1.25 is to create an array that contains the values of the initial value and the calculated values. Let us call the array y. It is a good practice to create the array with zeros—that is, an array with the required length and store zeros in it.

Here is one possible code (list of commands) for this task:

```
>> y=zeros(1, 7);          % Define y as an array of zeros with length 7 which
>>                         % will be used to store 7 terms of the sequence
>> y(1) = 100;             % Store the initial value in the first slot of y,
>>                         % y(1)=100
>> for n=1:6               % This "for loop" starts at n=1 and ends
>>                         % at n=6
     y(n+1)=1.1*y(n);      % This is equation 1.25 and is the body of loop
   end;
>> y
y=
   100.0000 110.0000 121.0000 133.1000 146.4100 161.0510
   177.1561
```

Note that MATLAB considers anything written after a percent symbol (%) and to the end of the line as a comment and ignores it. You will need to write explanations and documentation for your programs in MATLAB.

Note that equation 1.25 can be written in the following form:

$$y_n = 1.1y_{n-1}, \qquad y_1 = 100, \quad n = 2, 3, \ldots \tag{1.26}$$

To obtain the first 6 values of y_n after the initial value y_1 we use the difference equation 1.26 recursively. We have

$$\text{For } n = 2, \quad y_2 = 1.1y_1 = 1.1(100) = 110.$$
$$\text{For } n = 3, \quad y_3 = 1.1y_2 = 1.1(110) = 121.$$
$$\text{For } n = 4, \quad y_4 = 1.1y_3 = 1.1(121) = 133.10.$$
$$\text{For } n = 5, \quad y_5 = 1.1y_4 = 1.1(133.10) = 146.41.$$
$$\text{For } n = 6, \quad y_6 = 1.1y_5 = 1.1(146.41) = 161.051.$$
$$\text{For } n = 7, \quad y_7 = 1.1y_6 = 1.1(161.051) = 177.1561.$$

Here is a code to do these calculations. Note the difference between the start and the end of the "for" loop in this and the previous code. Note that you obtain the same results.

```
>> y=zeros(1, 7);          % Define y as an array of zeros with length 7,
>>                         % which will be used to store 7 terms of the
>>                         % sequence
>> y(1) = 100;             % Store the initial value in the first slot of y,
>>                         % y(1)=100
```

```
>> for n=2:7                % This "for loop" starts at n=2 and ends at
>>                          % n=7
     y(n)=1.1*y(n-1);       % This is equation 1.26 and is the body of loop
   end;
>> y
y=
   100.0000 110.0000 121.0000 133.1000 146.4100 161.0510
   177.1561
```

1.3.4. Plotting

If y is an array of numbers of length k, the command **plot(y)** will plot $y(1), y(2), \ldots, y(k)$ verses x values: $1, 2, 3, \ldots, k$. For example,

```
>> x=1:10
x=
    1   2   3   4   5   6   7   8   9   10
>> y=[2 6 8 15 10 38 46 50 80 100]
y=
   2   6   8   15   10   38   46   50   80   100
>> plot (y, 'k*')
```

The graph from the command **plot(y, 'k*')** is shown in Figure 1.3A. MATLAB plots the discrete points (1, 2), (2, 6), (3, 8), (4, 15), … , (8, 50), (9, 80), (10, 100) in black stars (*).

Note that the command **plot(x, y, 'k*')** would produce the same graph as shown in Figure 1.3A.

The command **>> plot(x, y, 'k')** would produce the graph in Figure 1.3B.

The graph from the command **>> plot (x, y, 'k*', x, y, 'k')** is shown in Figure 1.3C, where the discrete points are in black stars (*), and these points are connected with segment lines in black.

Now let us assume that we have two data arrays x and y of the same length. The plot commands are as follows:

plot (x, y)	plots y vs. x and connects the dots
plot (x, y, 'b.')	plots discrete points of y vs. x in blue points (.), and the points are not connected
plot (x, y, 'r*', x, y, 'b')	plots discrete points of y vs. x in red stars (*), and the points are connected with segment lines in blue

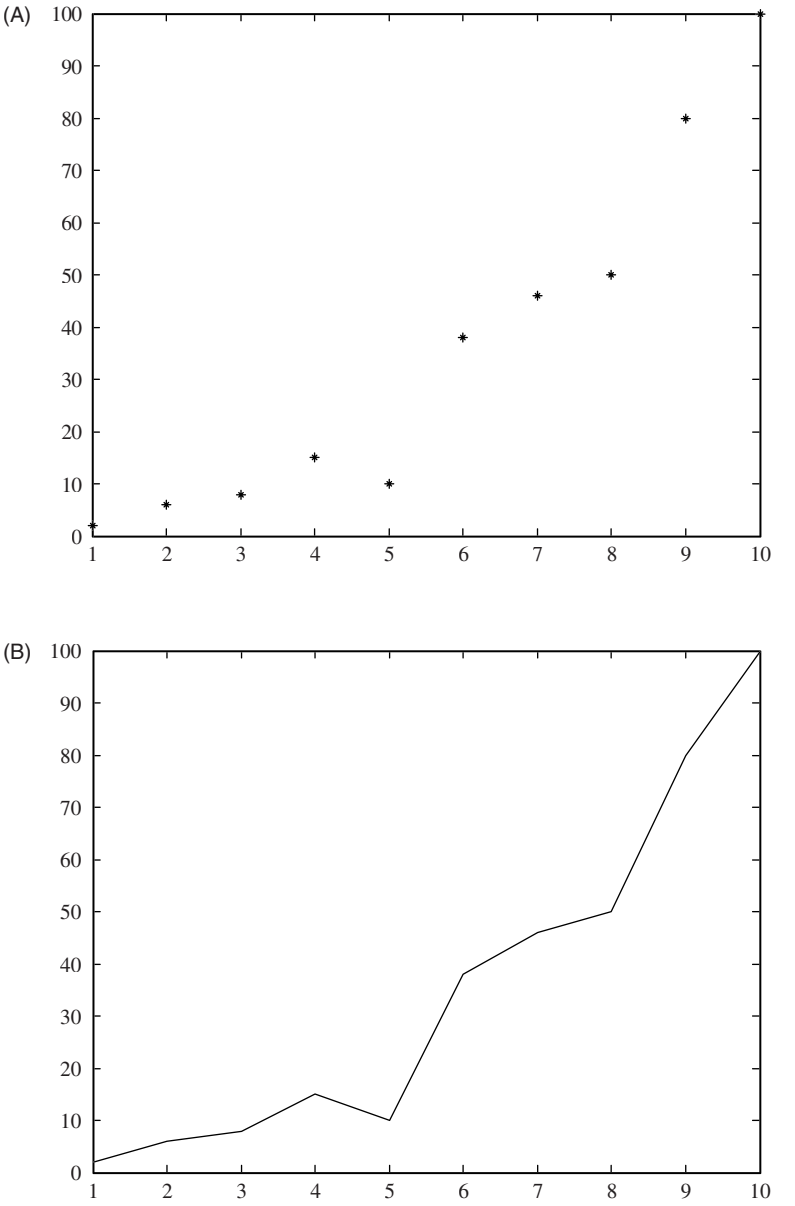

FIGURE 1.3. **A**. Graph from MATLAB command **`plot(y, 'k*')`** where **y** is an array of numbers, y=[2, 6, 8, 15, 10, 38, 46, 50, 80, 100]. **B**. Graph from MATLAB command **`plot(x, y, 'k')`** where x and y are arrays of numbers of the same length, with x=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10] and y=[2, 6, 8, 15, 10, 38, 46, 50, 80, 100]. **C**. Graph from MATLAB command **`plot(x, y, 'k*', x, y, 'k')`** where x and y are arrays of numbers of the same length, with x=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10] and y=[2, 6, 8, 15, 10, 38, 46, 50, 80, 100].

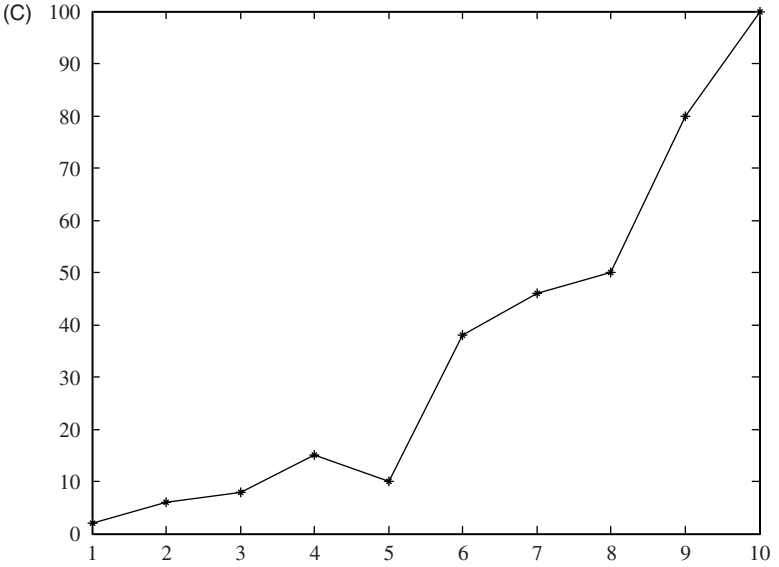

FIGURE 1.3. (*Continued*)

Example 1.1

Enter the following commands

```
>> x=-6:7
x=
  -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7
>> y1=x.^2
y1=
  36 25 16 9 4 1 4 9 16 25 36 49
>> y2=x.*2+8
y2=-4 -2 0 2 4 6 8 10 12 14 18 20 22
>> plot(x, y1, 'k.', x, y1, 'k')
```

The graph from the above plot command is shown in Figure 1.4A

If we want two graphs on one coordinate system, use the command **`hold on`**, which holds the current graph and the next graph will be plotted in the same window. For example, adding the following two commands to the earlier code will produce the graph in Figure 1.4B.

```
>> hold on
>> plot(x, y2, 'ko', x, y2, 'k')
```

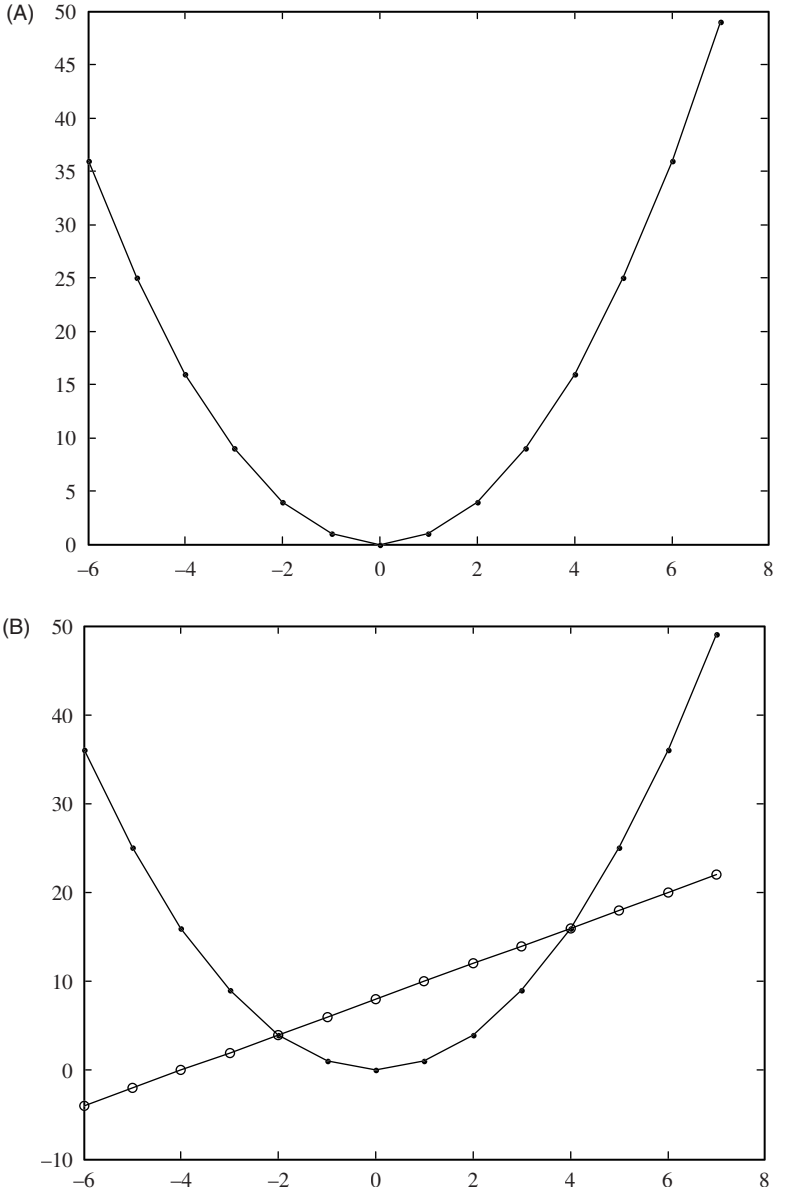

FIGURE 1.4. **A**. Graph from MATLAB command **`plot(x, y1, 'k.', x, y1, 'k')`** where x and $y1$ are arrays of numbers of the same length, with x = [–6, –5, … , 6, 7] = [–6, –5, –4, –3, –2, –1, 0, 1, 2, 3, 4, 6, 7] and $y1$ = [36, 25, 16, 9, 4, 1, 4, 9, 16, 25, 36, 49]. **B**. Two graphs on one coordinate system from MATLAB commands:

```
plot(x, y1, 'k.', x, y1, 'k')
hold on
plot(x, y2, 'ko', x, y2, 'k')
```

where x, $y1$, and $y2$ are arrays of numbers of the same length, with x = [–6, –5, –4, –3, –2, –1, 0, 1, 2, 3, 4, 6, 7], $y1$ = [36, 25, 16, 9, 4, 1, 4, 9, 16, 25, 36, 49], and $y2$ = [–4, –2, 0, 2, 4, 6, 8, 10, 12, 14, 18, 20, 22].

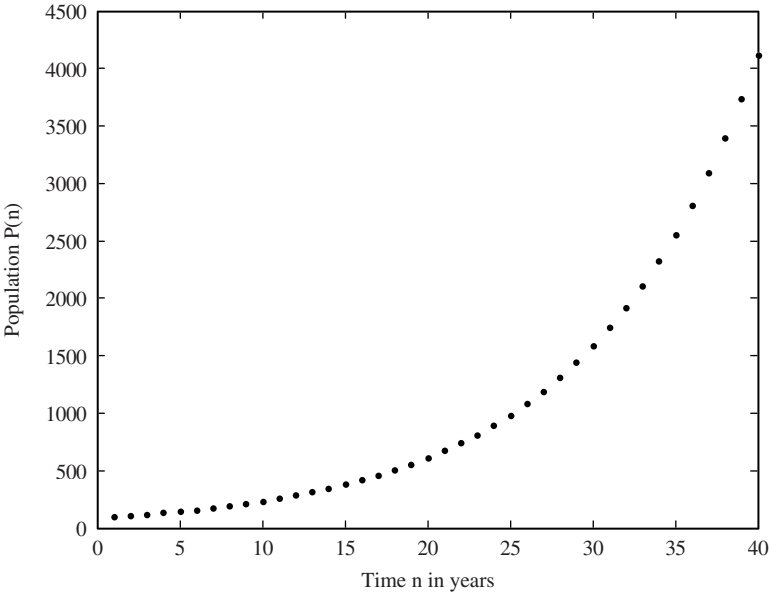

FIGURE 1.5. Graph of a species population, P_n, vs. the time, n, in years, with the labeling of the two axes, where $P_n = 1.1P_{n-1}$, $P_1 = 100$, and $n = 1, 2, \ldots, 40$.

Note that any other plots will be in the same window with other graphs as long as the command **`hold on`** is in effect. The command **`hold off`** will cancel **`hold on`** and plots a new graph in a new window.

Example 1.2

Consider the population of a species represented by the difference equation

$$P_n = 1.1P_{n-1}, \qquad P_1 = 100, \quad n = 2, 3, \ldots, 40,$$

Here is a possible code for iterating the difference equation, graph the population, p_n, vs. time, n, and label the two axes. The graph is shown in Figure 1.5.

```
>> T=1:40;                    % Define the time as an array
>> P=zeros(1, 40);            % Define P as an array of zeros with length 40
>> P(1)=100;                  % Store the initial value in the first slot of P
>> for n=2:40                 % This "for loop" starts at n=2 and
>>                            % ends at n=40
     P(n)=1.1*P(n-1);         % This is the iterated difference equation
   end;
>> plot(T, P, 'k.');          % Plots the population P(n) vs. time n in
>>                            % black points
```

```
>> xlabel('Time n in years');   % Labels the x axis – time n
>> ylabel('Population P(n)');  % Labels the y axis – population P(n)
```

MATLAB uses the following symbols for color:

r	red
y	yellow
b	blue
c	cyan
k	black
m	magenta
g	green
w	white

MATLAB uses the following symbols for line styles

.	point
-	solid
*	star
:	dotted
o	circle
--	dashed
+	plus
-.	dash-dot
x	x-mark

For example, if $x1$ and $y1$ are data arrays of the same length, the command

```
>> plot(x1, y1, 'r*')
```

plots $y1$ vs. $x1$ in discrete red stars; and the command

```
>> plot(x1, y1, 'b--')
```

plots $y1$ vs. $x1$ in a discrete blue dashed line.

1.3.5. M-files

So far, we were working with commands entered into the Command Window. It is more efficient and convenient to enter a sequence of commands into a file, save it, and run the file. MATLAB allows one to create such files, which are called **M-files**.

An M-file is a collection of MATLAB commands saved in a file that has the extension ".m". An M-file is an ordinary text file and may be created with the MATLAB editor or with an ordinary text editor. To execute the commands in an M-file, type the name of the file (without the extension ".m") as a command in the MATLAB prompt (>>). The commands in the file will be executed one by one.

There are two categories of M-files: **script files** and **function files**. A script file is a collection of MATLAB commands. A function file is a user-defined function.

Script Files

We are interested in creating, saving, and executing an M-file. To illustrate the procedure of creating an M-file using the MATLAB Editor and executing the M-file, we will use Example 1.2 (p. 22). Recall that for the example, we entered the commands into the Command Window. The task is to iterate the difference equation that represents the population of a species

$$P_n = 1.1P_{n-1}, \qquad P_1 = 100, \quad n = 2, 3, \ldots, 40$$

and graph the population, P_n, vs. time, n, labeling the two axes.

1. On the menu bar at the top of the MATLAB screen click on **File**, select **New**, select **Script**. You have a blank screen (note that at the top of the screen you see **Editor – Untitled**)
2. Enter the following commands

```
T=1:40;                          % Define the time as an array of 40
                                 % slots
P=zeros(1, 40);                  % Define P as an array of zeros
                                 % with length 40
P(1)=100;                        % Store the initial value in the first
                                 % slot of P
for k=2:40                       % This "for loop" starts at
                                 % k=2 and ends at k=40
    P(k)=1.1*P(k - 1);           % This is the iterated difference
                                 % equation
end;
plot(T, P, 'k.');                % Plots P(n) vs. n in discrete black
                                 % dots
xlabel('Time n in years');       % Labels the x axis – time n
ylabel('Population P(n)');       % Labels the y axis – population
                                 % P(n)
```

3. To save the file click on **File**, select **Save As** … . In the drive and folder of your choice enter the name of the file, say Population, without an extension. Assume that you created the folder MyMATLAB_MFiles on your C drive and you saved the file Population (without an extension) in that folder. You will see on the top screen

   ```
   Editor - C:\MyMATLAB_MFiles\Population.m
   ```

   Note that the MATLAB Editor has automatically extended the file name with .m.
4. To execute the file, click on the green icon on the top menu bar or in the Command Window and enter the following command

   ```
   >> Population
   ```

   If there are no mistakes in the contents of the script file, you will see a graph with the title Figure 1 on the MATLAB Graphing Window (this is the same graph as shown in Figure 1.5). However, if there are mistakes in the file, you will get error messages in the Command Window. You will then need to correct the errors and resave the file before executing it again.

Remarks on Script Files

Note that the variables in a script file are global. This means they are accessed in the workspace (Command Window). For example, the variables P and T in the script file Population can be accessed in the Command Window—for example, the population P_{10} is obtained by entering the following command

```
>> P(10)
ans =
    235.7948
```

It is informative to start an M-file with a comment that contains the file name plus a description of the script

```
% File name followed by description of the script file.
```

For example, let us add the following three lines to the code of the script file Population and save the file under the name **Population.m**:

```
% Population – A script file to iterate the difference equation
% representing the population of certain species, P(k) = 1.1P(k-1),
% P(1) = 100, n = 2, 3, ... , 40. The script file graphs P(n) vs. n.
```

```
T=1:40;                          % Define the time as an array of 40
                                 % slots
P=zeros(1, 40);                  % Define P as an array of zeros with
                                 % length 40
P(1)=100;                        % Store the initial value in the first slot
                                 % of P
for k=2:40                       % This "for loop" that starts at k=2
                                 % and ends at k=40
    P(k)=1.1*P(k - 1);           % This is the iterated difference equation
end;
plot(T, P, 'k.');                % Plots P(n) vs. time n in discrete black
                                 % dots
xlabel('Time n in years');       % Labels the x axis – time n
ylabel('Population P(n)');       % Labels the y axis – population P(n)
```

If you enter the command **help Population** in the Command Window, you will get the description of the script file Population you entered in first comment lines after the file name. Here is the command and MATLAB output:

```
>> help Population
  Population - A script file to iterate the difference
  equation representing the population of certain species,
  P(k)=1.1P(k-1), P(1)=100, n=2,3,...,40. The script file
  graphs P(n) vs. n
```

Function Files

A function file is an M-file containing a user-defined function.

For example, let us write a function to calculate the volume, v, of rectangular box of length l, width w, and height h. We know that $v = l \cdot w \cdot h$. We will give a name to this function, say RecVolume. You could write the following commands in the MATLAB Editor:

```
function v=RecVolume(l, w, h)
% RecVolume is a function that calculates the volume of a rectangular box.
% The function input: l, w, and h, where l=length, w=width, and h=height.
% The function output is the volume v
v=l*w*h;
```

Save this code in a file under the name RecVolume.m. Note that if you use MATLAB Editor to write the function code and save the file, the extension .m will be automatically extended to the file name.

To call this function for specific values of l, w, and h, for example $l=10$, $w=20$, and $h=5$, we write the command

```
>> v=RecVolume(10, 20, 5)
v=
  1000
```

If $l=2\pi^2$, $w=\sqrt{124.26}$, and $h=3l$, the function RecVolume can be called as

```
>> RecVolume(2*pi^2, sqrt(124.26), 3*l)
ans=
  4.1476e+03

>> help RecVolume
  RecVolume is a function that calculates the volume of
  a rectangular box.
  The function input: l, w, and h, where l=length, w=width,
  and h=height.
  The function output is the volume v
```

Remarks on Function Files

1. The first line in the code of a function must be the keyword **function** followed by the output variable (or output variables) enclosed in square brackets, followed by the equals sign, followed by the input variables enclosed in parentheses and separated by comma. If the first line of a function file does not start with the keyword *function*, you will not be able to call the function and the file will be a script file.
2. The function descriptions in the comment lines are important. If the user enters the command **help** followed by function name, MATLAB will display the function description.
3. Variables used inside a function are local to the function (in other words, these variables cannot be called globally, in the command window).
4. The list of output variables of a function may be omitted if you are not interested in their values. For example if you write a function that iterates a difference equation and graphs the ordered pairs, but you are not interested in values of the ordered pairs, you may omit an output variable.