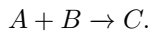


Chemical Kinetics and Enzyme Dynamics

Cells are the basic units of life. A cell consists of a concentrated aqueous solution of molecules contained in a membrane, called **plasma membrane**. A cell is capable of replicating itself by growing and dividing. Cells that have a nucleus are called **eukaryotes**, and cells that do not have a nucleus are called **prokaryotes**. Bacteria are prokaryotes, while yeast and amoebas, as well as most cells in our body, are eukaryotes. The **deoxyribonucleic acid (DNA)** is a very long polymeric molecule, consisting of two strands of chains, having double helix configuration, with repeated nucleotide units A, C, G, and T. The DNA is packed in chromosomes, within the nucleus in eukaryotes. In humans, the number of chromosomes is 46, except in sperm and egg cells where the number is 23.

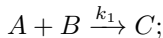
The DNA is the genetic code of the cell; it codes for proteins. Proteins lie mostly in the cytoplasm of the cells, that is, outside the nucleus; some proteins are attached to the plasma membrane, while some can be found in the nucleus. Proteins are polymers of amino acids whose number typically ranges from hundreds to thousands; there are 20 different amino acids from which all proteins are made. Each protein assumes three-dimensional configuration, called **conformation**. Proteins perform specific tasks by changing their conformation.

Two proteins, A and B , may combine to form a new protein C . We express this process by writing



Biological processes within a cell involve many such reactions. Some of these reactions are very slow, others are very fast, and in some cases the reaction rate may start slow, then speed up until it reaches a maximal level. In this chapter we consider the question: how to determine the speed of biochemical reactions among proteins. In order to address this question we shall develop some mathematical models.

We begin with a simple case. Suppose we have two proteins, A and B , or more generally two molecules A and B . We assume that A and B , when coming in contact, undergo a reaction, at some rate k_1 , that makes them form a new molecule C . We express this reaction by writing



k_1 is called the **rate coefficient**. The respective concentrations of three molecules are denoted by $[A]$, $[B]$, and $[C]$. The **law of mass action** states that the **reaction rate** $\frac{d[C]}{dt}$, or v_1 , of the above reaction is given by

$$v_1 = k_1[A][B],$$

that is,

$$\frac{d[C]}{dt} = k_1[A][B].$$

Then also

$$\frac{d[A]}{dt} = -v_1 = -k_1[A][B], \quad \frac{d[B]}{dt} = -v_1 = -k_1[A][B].$$

Suppose one molecule of A and two molecules of B react to form a new molecule C :



Since this reaction can be viewed as $A + B + B \rightarrow C$, the law of mass action states that

$$\frac{d[C]}{dt} = v_1, \quad \text{where} \quad v_1 = k_1[A][B]^2,$$

and then

$$\frac{d[A]}{dt} = -v_1, \quad \frac{d[B]}{dt} = -2v_1.$$

The **stoichiometric coefficients** of A , B , C in this reaction are 1, 2, 1. For the reversible reaction



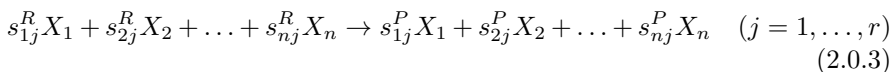
the reaction rate $\frac{d[C]}{dt}$ is $-v_2$ where $v_2 = k_2[C]$, and under both reactions, (2.0.1) and (2.0.2),

$$\begin{aligned} \frac{d[A]}{dt} &= -v_1 + v_2, \\ \frac{d[B]}{dt} &= -2v_1 + 2v_2, \\ \frac{d[C]}{dt} &= v_1 - v_2, \end{aligned}$$

or, in a vector form,

$$\frac{d}{dt} \begin{pmatrix} [A] \\ [B] \\ [C] \end{pmatrix} = \begin{pmatrix} -1 & 1 \\ -2 & 2 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}.$$

Consider n chemical species whose concentrations are $[X_1], [X_2], \dots, [X_n]$. Let there be r chemical reactions, with each reaction being symbolized by



where s_{ij}^R , s_{ij}^P are stoichiometric coefficients of species i on the reactant side and product side, respectively, of reaction j . Let \mathbf{X} be the concentration vector $[[X_1], \dots, [X_n]]$, \mathbf{v} the reaction velocity vector $[v_1, \dots, v_r]$ where v_j is the rate of reaction j and S the so-called **stoichiometric matrix** whose element s_{ij} is equal to $(s_{ij}^P - s_{ij}^R)$. The general set of dynamics equations for chemical reactions systems can be written succinctly as $\frac{d[X_l]}{dt} = \sum_{j=1}^r s_{lj} v_j$, or

$$\dot{\mathbf{X}} = \mathbf{S}\mathbf{v}, \quad (2.0.4)$$

where $\dot{\mathbf{X}}$ means $d\mathbf{X}/dt \equiv [d[X_1]/dt, \dots, d[X_n]/dt]$. Such a system of ODEs is called a **stoichiometric dynamics system**.

A reaction j is said to have **mass-action kinetics** if its rate v_j has the form

$$v_j = k_j \Pi_{i=1}^n [X_i]^{s_{ij}^R},$$

and, if this holds for all j , then (2.0.4) can be written in the form

$$\frac{d[X_l]}{dt} = \sum_{j=1}^r s_{lj} k_j \Pi_{i=1}^n [X_i]^{s_{ij}^R} \quad (1 \leq l \leq n). \quad (2.0.5)$$

In the next chapter we shall review the basic theory of ordinary differential equations. By solving the system of differential equations (2.0.5) we can find how each concentration $[X_l]$ will evolve as a function of time. This is illustrated in Problems 2.1 and 2.2.

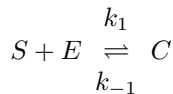
Metabolism in a cell is the sum of physical and chemical processes by which material substances are produced, maintained, or destroyed and by which energy is made available. **Enzymes** are proteins that act as catalysts in speeding up chemical reactions within a cell. They play critical roles in many metabolic processes within the cell. An enzyme, say E , can take a molecule S and convert it to a molecule P in one millionth of a second. The original molecule S is referred to as the **substrate**, and P is called the **product**. The enzyme-catalyzed conversion of a substrate S into a product P is written in the form



The profile $[S] \rightarrow [P]$ can take different forms, depending on the underlying biology. Two typical profiles are shown in Fig. 2.1.

Figure 2.1a, b have been shown to hold in different experiments, but it would be useful to derive them by mathematical analysis based on known properties of enzymes. We begin with the derivation of a formula that yields the profile of Fig. 2.1a.

In what follows we show how such a profile can be derived from the law of mass action. We write, schematically,



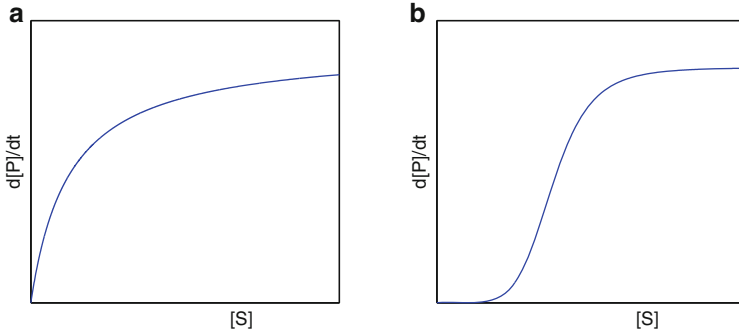
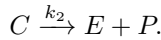


FIGURE 2.1. Two different profiles of the enzymatic conversion of $S \rightarrow P$

where C is the complex SE ,



By the law of mass action

$$\frac{d[C]}{dt} = k_1[S][E] - (k_{-1} + k_2)[C], \quad (2.0.7)$$

$$\frac{d[E]}{dt} = -k_1[S][E] + (k_{-1} + k_2)[C], \quad (2.0.8)$$

$$\frac{d[S]}{dt} = -k_1[S][E] + k_{-1}[C], \quad (2.0.9)$$

$$\frac{d[P]}{dt} = k_2[C]. \quad (2.0.10)$$

Notice that

$$\frac{d}{dt} ([E] + [C]) = 0$$

so that $[E] + [C] = \text{const} = e_0$; e_0 is the total concentration of the enzyme in both E and the complex C . Note also that $\frac{d[C]}{dt} + \frac{d[S]}{dt} + \frac{d[P]}{dt} = 0$; hence Eq. (2.0.9) follows from Eqs. (2.0.7) and (2.0.10) and may therefore be dropped.

We focus on Eq. (2.0.7) and note that the enzymatic process is very fast, that is, the rates of formation and breakdown of the complex C are very fast. We assume that these rates are essentially the same, so that $\frac{d[C]}{dt}$ is approximately zero. Hence, approximately,

$$k_1[S][E] - (k_{-1} + k_2)[C] = 0. \quad (2.0.11)$$

Substituting $[E] = e_0 - [C]$ we get

$$k_1[S](e_0 - [C]) - (k_{-1} + k_2)[C] = 0$$

or

$$[C] = \frac{k_1 e_0 [S]}{(k_{-1} + k_2) + k_1 [S]} = \frac{e_0 [S]}{K_M + [S]}$$

where $K_M = \frac{k_{-1} + k_2}{k_1}$.

Recalling (2.0.10) we have thus derived the **Michaelis–Menten formula**

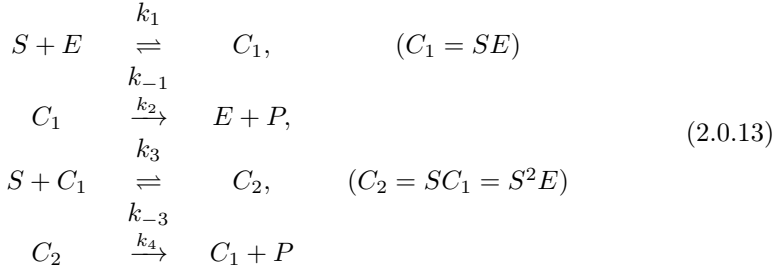
$$\frac{d[P]}{dt} = \frac{V_{max}[S]}{K_M + [S]} \quad (2.0.12)$$

where $V_{max} = k_2 e_0$ and K_M are constants; note that

$$\frac{d[P]}{dt} \rightarrow V_{max} \quad \text{as} \quad [S] \rightarrow \infty.$$

The assumption we made in the derivation of (2.0.12) that $d[C]/dt$ is very small is quite reasonable and, indeed, the Michaelis–Menten formula is widely used in describing enzymatic processes.

But what about Fig. 2.1b? Such a profile is based on a different enzymatic process, for example, when an enzyme E can bound first with one substrate S and then with another substrate S . Furthermore, in such a case, as is well established experimentally, the speed by which the enzyme bounds with the second substrate is much faster. We model such processes as follows:



so that

$$\frac{d[P]}{dt} = k_2[C_1] + k_4[C_2].$$

Note that $[E] + [C_1] + [C_2] = \text{const.} = e_0$. Assuming the steady state approximations

$$\frac{d[C_1]}{dt} = \frac{d[C_2]}{dt} = 0$$

one can show (see Problem 2.3) that

$$\frac{d[P]}{dt} = \frac{(k_2 K_2 + k_4 [S]) e_0 [S]}{K_1 K_2 + K_2 [S] + [S]^2}, \quad (2.0.14)$$

where

$$K_1 = \frac{k_{-1} + k_2}{k_1}, \quad K_2 = \frac{k_{-3} + k_4}{k_3}.$$

Steps 1 and 3 in Eq. (2.0.13) represent sequential binding of two substrate molecules to the enzyme. We assume that previously enzyme-bound substrate molecule significantly increases the rate of bounding of a second substrate molecule, so that $k_3 \gg k_1$. In the extreme case of $k_1 \rightarrow 0$, $k_3 \rightarrow \infty$ with $k_1 k_3$ a finite positive constant, we get $K_1 \rightarrow \infty$, $K_2 \rightarrow 0$, $K_1 K_2 \rightarrow K_H > 0$, so that

$$\frac{d[P]}{dt} = \frac{V_{max}[S]^2}{K_H + [S]^2} \quad (2.0.15)$$

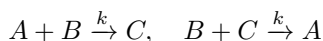
where V_{max} and K_H are constants. Formula (2.0.15) is called the **Hill kinetics**; it displays a profile similar to Fig. 2.1b.

Some enzymes can bound with three or more substrates. In this case it is often the case that when enzyme has already bounded with m substrates S , it has a greater affinity to bound with the next substrate S . Under this biological assumption, one can derive the Hill kinetics of order n :

$$\frac{d[P]}{dt} = \frac{V_{max}[S]^n}{K_H + [S]^n}. \quad (2.0.16)$$

Problem 2.4 which will be mentioned in the following suggests how to mathematically prove formula (2.0.16). The profile of $[S] \rightarrow [P]$ in Hill kinetics for large n makes a very fast transition from very slow speed to saturated (or limit) speed. The Hill kinetics of order $n \geq 3$ is sometimes used in modeling biochemical reactions within a cell.

PROBLEM 2.1. Consider the chemical reactions



with $[A] + [C] = 3$ at time $t = 0$. Show that $y = [B]$ satisfies:

$$y(t) = y_0 e^{-3kt}.$$

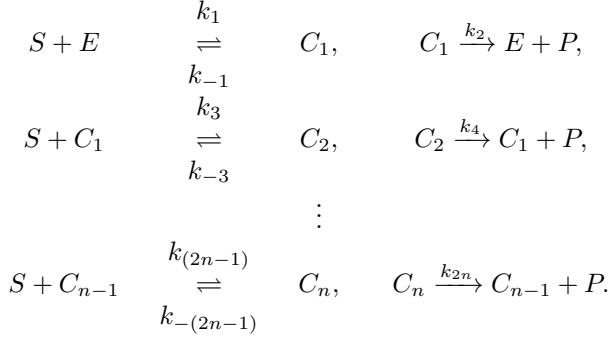
PROBLEM 2.2. Consider the chemical reactions (2.0.1), (2.0.2) and set $[A] + [C] = m$, $2[A] - [B] = n$ at time $t = 0$. Show that the function $y = [B]$ satisfies a differential equation

$$\frac{dy}{dt} = ay^3 + by^2 + cy + d,$$

and compute a, b, c, d in terms of m, n .

PROBLEM 2.3. Derive the relation (2.0.14) under the steady-state approximations $d[C_1]/dt = 0$, $d[C_2]/dt = 0$.

PROBLEM 2.4. Consider the enzymatic process

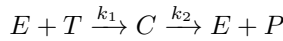


Under the assumptions that $k_1 < k_3 < \dots < k_{2n-1}$ and $k_1 \rightarrow 0$, $k_{2n-1} \rightarrow \infty$, and $d[C_j]/dt = 0$ for $1 \leq j \leq n$, show how to derive the Hill kinetics

$$\frac{d[P]}{dt} = \frac{V_{max}[S]^n}{K_H + [S]^n}.$$

[Hint: $k_{2i-1}[S][C_{i-1}] - (k_{-(2i-1)} + k_{2i})[C_i] = 0$ with $[C_0] = [E]$; hence $[C_j] = \prod_{i=j+1}^n K_i [S]^{-(n-j)} [C_n]$, where $K_i = \frac{k_{-(2i-1)} + k_{2i}}{k_{2i-1}}$. Note that $[E] + \sum_{j=1}^n [C_j] = e_0$, $K_1 \rightarrow \infty$, $K_n \rightarrow 0$ and assume that the K_j are bounded if $2 \leq j \leq n-1$ and $K_1 K_2 \dots K_n \rightarrow K_H$.]

The law of mass action and the Michaelis–Menten law can be applied also to any large population of species, for instance, to cells. Consider two populations of cells: tumor cells with number density T and effector cells of the immune system (such as cytotoxic $CD8^+$ T cells) with number density E . When an effector cell senses a tumor cell in its vicinity, it secretes toxic molecules which kill the tumor cells. We can describe this process schematically by



where C denotes the number density of tumor cells “surrounded” by toxic-secreting effector cells and P is the number density of dead tumor cells.

Applying the law of mass action we can write

$$\begin{aligned}
 \frac{d[E]}{dt} &= -k_1[E][T] + k_2[C], \\
 \frac{d[C]}{dt} &= k_1[E][T] - k_2[C]
 \end{aligned}$$

so that $[E] + [C] = \text{const.} = e_0$. Assuming that the killing of the tumor cells is an enzymatic process, we approximately have $d[C]/dt = 0$, so that

$$k_1[E][T] = k_2[C] = k_2(e_0 - [E]),$$

or

$$[E] = \frac{k_2 e_0}{k_2 + k_1[T]}.$$

It follows that

$$\frac{d[P]}{dt} = k_2[C] = \frac{(k_1 k_2 e_0) [T]}{k_2 + k_1 [T]}$$

which is the Michaelis–Menten law.

2.1. Introduction to MATLAB

It is well known that closed-form or analytic solutions may or may not exist when one tries to solve algebraic equations or differential equations. Even if the closed-form solutions do exist, the derivation may be quite cumbersome. Due to the invention of computers, modern computation techniques and algorithms enable us to seek for solutions in an efficient and robust way. When the closed-form solutions exist, one can perform **symbolic** calculation on computers to obtain the expression of solutions. When the closed-form solutions do not exist, one can look for numerical approaches that provide approximations of the exact solutions. Since numerical solutions are not exact, it is important to understand how accurate numerical solutions are and how robust the numerical approaches are. A numerical approach can be implemented in many different programming languages, such as *C*, Fortran, Java, MATLAB, Maple, and Mathematica. We will use MATLAB, due to its simplicity and flexibility to code, and provide detail instructions on solving biological models introduced in this book.

MATLAB (MATrix LABoratory) is a high-level numerical computing environment which is developed by MathWorks. MATLAB contains a lot of built-in functions which allow matrix manipulations, algebraic and differential equations solving, data analysis, visualization, etc. We will first describe some basic manipulations for scalars, vectors, and matrices.

To launch MATLAB which is already installed and ready for usage, click on a MATLAB icon and wait for the default user interface to appear. Take the version MATLAB R2012a [17] as an example. The layout of the interface contains several subwindows, including a current directory address on the top to inform you which directory you are currently working on, a current folder on the left which lists the files in the current folder, a command window in the center which allows you to perform calculations, a workspace on the top right to indicate variables you use and their values, and a command history which keeps track of all command lines you type in the command window. If you are a beginner in MATLAB, click on the rightmost item “Help” in the manual bar on top of the directory address. Select “Product Help,” double click on “MATLAB,” double click on “Getting Started,” and then double click on “Quick Start.” Here you can learn about the topics such as “Matrices and Arrays,” “Functions,” and “Plots” to perform some basic calculations and visualizations in MATLAB.

2.1.1. Basic Arithmetic Operations, Creating Variables, and Calling Functions. To perform the calculation, we need to type the commands into the command window where you see the prompt sign “>.”

You can perform the basic arithmetic operations such as addition, subtraction, multiplication, and division by using “+,” “-,” “*,” and “/”. For example, you can calculate $2.3 \times 5 + 6 \div 3 - 2$ by entering

```
>> 2.3*5+6/3-2
```

Algorithm 1 BasicArithmicOperation.m

```
function [ f, g ] = BasicArithmicOperation(a,b,c,d,e)
% This function has five inputs a, b, c, d, and e and two outputs f, and g.
% It perform two calculations: f = a*b+c/d-e and g = c-b/e.
f = a*b+c/d-e;
g = c-b/e;
end
```

After you hit the enter (return) key, the solution `ans = 11.5000` will show up and another prompt sign will appear to wait for your next command. Notice that “ans” will be added to the workspace as a variable.

You can also enter numbers in scientific notation, e.g., 1.1×10^{-9} , by entering

```
>> 1.1*10^-9
```

The solution `ans = 1.1000e-09` will show up in the command window. You can also enter `1.1e-9` or `1.1E-9`. Notice that “ans” variable in the workspace is replaced by `1.1000e-09`. This implies the current value of “ans” has overwritten the previous value. The default format to represent a number in MATLAB is “format short” which is the scaled fixed-point format with five digits. One can switch to the scaled fixed-point format with 15 digits for double precision and seven digits for single precision by entering “format long.”

Sometimes it is more convenient to perform the calculations by using variables or calling a function, especially when the same variables or calculations need to be used multiple times. For example, we can create variables “a,” “b,” “c,” “d,” and “e” and assign the values as 2.3, 5, 6, 3, and 2 by entering

```
>> a = 2.3, b = 5, c = 6, d = 3, e=2
```

To compute $2.3 \times 5 + 6 / 3 - 2$ and assign it as the variable “f,” we can then enter

```
>> f = a*b+c/d-e
```

The solution `f = 11.5000` will show up in the common window.

Since the variables from “a” to “f” have been stored in the workspace, we can perform other calculations, e.g.,

```
>> g = c-b/e
```

which will return `g = 3.5000`.

To create a function (subroutine) to perform both calculations $f = a*b+c/d-e$ and $g = c-b/e$, click “File,” move your mouse to “NEW,” and then click on “Function.” A new window will pop out and allow you to write a function and save it as a script M-file. Type in exactly the same commands shown in Algorithm 1 and save it as “BasicArithmicOperation.m.” In MATLAB script M-file, all characters from the % to the end of the line

are treated as a comment and will not be executed. Now you can call this function in the command window by entering

```
>> [ f, g] = BasicArithmeticOperation(2,3,5,6,3,2)
```

MATLAB function	Description
abs(x)	Absolute value of x
sqrt(x)	Square root of x
sin(x)	Sine of x in radians
sind(x)	Sine of x in degrees
asin(x)	Inverse sine of x in radians
asind(x)	Inverse sine of x in degrees
sinh(x)	Hyperbolic sine of x in radians
asinh(x)	Inverse hyperbolic sine of x in radians
cos(x)	Cosine of x in radians
cosd(x)	Cosine of x in degrees
tan(x)	Tan of x in radians
cot(x)	Cotangent of x in radians
sec(x)	Secant of x in radians
csc(x)	Cosecant of x in radians
exp(x)	Exponential of x
log(x)	Natural logarithm of x
log2(x)	Base 2 logarithm of x
log10(x)	Base 10 logarithm of x
floor(x)	Round x toward minus infinity
ceil(x)	Round toward plus infinity
round(x)	Round toward nearest integer
fix(x)	Round toward zero

TABLE 2.1. Some built-in MATLAB functions

MATLAB will return $f = 11.5000$ and $g = 3.5000$. If you want to perform this calculation with $a = 1$, $b = 2$, $c = 3$, $d = 4$, and $e = 5$, you can simply perform that by entering

```
>> [ f, g] = BasicArithmeticOperation(1,2,3,4,5)
```

MATLAB will return $f = -2.2500$ and $g = 2.6000$.

MATLAB contains a lot of built-in functions which can execute in the command window or in the script M-file you would like to generate. In Table 2.1, some commonly used functions are listed. For example, in the command window, you can enter

```
>> log2(16)
```

which will return $\text{ans} = 4$.

2.1.2. Vector and Matrix Calculations. Since most of the biological models are formulated as system of equations, we need to know how to

perform vector and matrix calculations on MATLAB. To create a row vector with three elements 1, 2.5, and 4, separate the elements with either a comma or a space. Enter either

```
>> a = [1, 2.5, 4]
```

or

```
>> a = [1 2.5 4]
```

which will return

```
a =
1.0000 2.5000 4.0000
```

To create a column vector with the same three elements, separate the elements with semicolons:

```
>> a = [1; 2.5; 4]
```

which will return

```
a =
1.0000
2.5000
4.0000
```

To create a matrix that has multiple rows, separate the rows with semicolons. For example, to create a 3-by-3 matrix with 1 to 9 integers, enter

```
>>a = [1 2 3; 4 5 6; 7 8 9]
```

which will return

```
a =
1 2 3
4 5 6
7 8 9
```

MATLAB contains several built-in functions to create fundamental matrices such as ones, zeros, eye, rand, randi, or randn. For example, entering

```
>>eye(5)
```

will create a 5-by-5 identity matrix.

To perform basic matrix arithmetic operations such as addition, subtraction, and multiplication, simply use “+,” “-,” and “*”. For example, enter

```
>> A = [1 2; 3 4]; B = [5 6 ;7 8];
```

to create two 2-by-2 matrices A and B. If we want to compute $C = AB$, enter

```
>> C = A*B
```

which will return

```
C =
19 22
43 50
```

Notice that the matrix product is not commutative. If we compute $D = BA$, enter

```
>> D = B*A
```

which will return

```
D =
23 34
31 46
```

To perform element-wise multiplication rather than matrix multiplication, use the `.*` operator

```
>> E = A.*B
```

MATLAB script	Description
<code>A'</code> or <code>ctranspose(A)</code>	Complex conjugate transpose of A
<code>A.'</code> or <code>transpose(A)</code>	Non-conjugate transpose of A
<code>inv(A)</code>	Matrix inverse of the square matrix A
<code>norm(A)</code> or <code>norm(A,2)</code>	2-norm of A
<code>norm(A,1)</code>	1-norm of A
<code>norm(A,Inf)</code>	Infinity norm of A
<code>norm(A,'fro')</code>	Frobenius norm of A
<code>rank(A)</code>	An estimate of the number of linearly independent rows or columns of A
<code>size(A)</code>	Size of A
<code>numel(A)</code>	Total number of elements in a vector or matrix

TABLE 2.2. Some manipulations of a given matrix A

which will return

```
E =
5 12
21 32
```

Some other important matrix manipulations are listed in Table 2.2.

MATLAB also provides various ways to solve linear system of equations [19]. `X = linsolve(A,B)` solves the linear system $AX = B$ using LU factorization with partial pivoting when A is square and QR factorization with column pivoting otherwise. Warning is given if A is ill conditioned for square matrices and rank deficient for rectangular matrices. For example,

```
>> A = [1 2 ;3 4]; B = [4;10]; X = linsolve(A,B)
```

which will return

```
X =
2.0000
1.0000
```

One can also use backslash ("`\`") to solve $AX = B$ by entering

```
>> X = A\B
```

which will also return

```
X =
2.0000
1.0000
```

The backslash computes the solution in a different way. To check out the detail description, enter

```
>> help \
```

which will provide the usage of backslash (“\”) and the method for numerical implementation.

2.1.3. Symbolic Calculation. Symbolic Math Toolbox [18] provides functions for solving and manipulating symbolic math expressions and performing variable-precision arithmetic. However, it requires a different license from the main MATLAB package. If it is installed on your computer, it allows you to analytically perform differentiation, integration, simplification, transformation, and equation solving.

To declare variables x and y as symbolic objects, use the `syms` command:

```
>> syms x y
```

You can then perform mathematical calculation symbolically. For example, $x + y + 4y$ can be computed by entering

```
>> x+y+4*y
```

which will return

```
ans =  
x + 5*y
```

Symbolic Math Toolbox also enables you to convert numbers to symbolic objects. To create a symbolic number, use the `sym` command. For example, when we want to compute $\sqrt{2}$ by using double precision, enter

```
>> a = sqrt(2)
```

which will return

```
ans =  
1.4142
```

However, when we want to perform the calculation symbolically, enter

```
>> a = sqrt(sym(2))
```

which will return

```
a = 2^(1/2)
```

To solve an algebraic equation, one can use the built-in function “`solve`”. For example, to solve $ax^2 + bx + c = 0$, one can enter

```
>> syms a b c x; solve(a*x^2 + b*x + c)
```

which will return

```
ans =  
-(b + (b^2 - 4*a*c)^(1/2))/(2*a)  
-(b - (b^2 - 4*a*c)^(1/2))/(2*a)
```

To solve a system of algebraic equations, one can use “,” to separate each algebraic equation. For example, to solve

$$\begin{cases} x^2 + xy + y &= 7, \\ x^2 - 4x - y &= -5, \end{cases}$$

and assign the solution to variables Sx and Sy , one can enter

```
>> syms x y; [Sx,Sy] = solve(x^2 + x*y + y == 7,x^2 - 4*x - y == -5)
```

which will return

Sx =

2

-i

i

Sy =

1

4 + 4*i

4 - 4*i

where $i = \sqrt{-1}$ is the imaginary unit.

To solve a differential equation [2], one can use the built-in function “dsolve”. For example, to solve $\frac{dx(t)}{dt} = ax(t)$ without specified initial condition, enter

```
>>syms x(t) a; dsolve(diff(x) == a*x)
```

which will return

```
ans =
```

```
C1*exp(a*t)
```

where C1 may be replaced by another constant. To solve $\frac{dx(t)}{dt} = ax(t)$ with the initial condition $x(0) = 1$ and assign it to Sx, enter

```
>> syms x(t) a; Sx = dsolve(diff(x) == a*x,x(0)==1)
```

which will return

```
Sx =
```

```
exp(a*t)
```

To solve a system of differential equations

$$\begin{cases} \frac{df(t)}{dt} = f(t) + 2g(t), \\ \frac{dg(t)}{dt} = -f(t) + 2g(t), \end{cases} \quad (2.1.1)$$

and assign the solution to S, enter

```
>>syms f(t) g(t); S = dsolve(diff(f) == f + 2*g, diff(g) == -f + 2*g)
```

which will return

```
S =
```

```
g: [1x1 sym]
```

```
f: [1x1 sym]
```

To return the values of $g(t)$, enter the command:

```
>>g(t) = S.g
```

which will return

```
g(t) = C2*exp((3*t)/2)*cos((7^(1/2)*t)/2) - C3*exp((3*t)/2)*
      sin((7^(1/2)*t)/2)
```

where C2 and C3 may be replaced by other constants.

To solve (2.1.1) with initial conditions $f(0) = 1$ and $g(0) = 2$, enter

```
>>syms f(t) g(t); S = dsolve(diff(f) == f + 2*g, diff(g) == -f + 2*g,
f(0)==1, g(0)==2);
```

```
>>g(t) = S.g
```

```
g(t) =
```

```
2*cos((7^(1/2)*t)/2)*exp(t)^(3/2)
```

PROBLEM. 2.5. Write scripts in MATLAB to solve Problems [2.1](#) and [2.2](#) symbolically.

To exit MATLAB, you can either click on the red button at the top left corner or simply type quit in the command window.

Mathematical Modeling of Biological Processes

Friedman, A.; Kao, C.-Y.

2014, VI, 154 p. 32 illus., 17 illus. in color., Softcover

ISBN: 978-3-319-08313-1