MATHEMATICA APPLICATIONS IN CHAPTERS 1 – 8

MATHEMATICA APPLICATIONS IN CHAPTER 1

Section S1.2.9

Suppose that we have a set **lis1=List[2,4,6,8,10]**. Assume that we want to test whether the numbers 2 and 5 are the elements of **lis1**. This can be obtained, using the command **MemberQ**, as

```
lis1=List[2,4,6,8,10]
MemberQ[lis1,2]
MemberQ[lis1,5]
True
False
```

Suppose now that we have two sets **lis2=List[9,8,7,6]** and **lis3=List[8,7,9,6]**. We now want to test whether these two sets are equal. This can be done by using the == sign as

```
lis2=List[9,8,7,6];
lis3=List[8,7,9,6];
lis2==lis3
False
```

The reason for this is that, even though the two sets are mathematically equivalent, Mathematica treats them as two different sets due to the difference in the order of their elements. Therefore, to obtain the correct, expected result we must enter and execute as

```
lis2=List[9,8,7,6];
lis4=List[9,8,7,6];
lis2==lis4
True
```

Assume that we have two other sets: **lis5=List[1,2,3]** and **lis6=List[1,2,3,4,5]**. The subsets of the set **lis6=List[1,2,3,4,5]** can be found by using the command **Subsets** as

```
lis6=List[1,2,3,4,5];
Subsets[lis6]
{{},{1},{2},{3},{4},{5},{1,2},{1,3},{1,4},{1,5},{2,3},{2,4},{2,5},{3,4},{3,5},
{4,5},{1,2,3},{1,2,4},{1,2,5},{1,3,4},{1,3,5},{1,4,5},{2,3,4},{2,3,5},{2,4,5},
{3,4,5},{1,2,3,4},{1,2,3,5},{1,2,4,5},{1,3,4,5},{2,3,4,5},{1,2,3,4,5}}
```

, which clearly shows that the set **lis5** is a subset (strictly speaking, a proper subset) of the set **lis6**. It also show that the power set contains a total of 32 elements as required by our earlier result $2^{n(lis\ 6)} = 2^5 = 32$. Since we have presented the set operations using Mathematica in Section 11.5.6, we do not repeat them here. However, the reader is encouraged to do those operations using the examples given above or using own examples.

Section S1.3.4

Let us first test whether the numbers 5 and 6 are prime numbers or not. To do this we may use the command **PrimeQ** as

```
PrimeQ[5]
PrimeQ[6]
True
False
```

Similarly, we may test whether the numbers 10, -10, 0, and 0.5 are integers or not. To do this may use the command **IntegerQ** as

```
IntegerQ[10]
IntegerQ[-10]
IntegerQ[0]
IntegerQ[0.5]
True
```

```
True
True
False
```

A similar result can be obtained if we use the command **Head[*expr*]**. **Head[*expr*]** shows whether the ***expr*** belongs to a particular set of numbers or not. Using this command with different ***expr*** as

```
Head[5/6]
Head[2.71828]
Head[0.5]
Head[0]
Head[2+2i]
Rational
Real
Real
Integer
Complex
```

Section S1.4.4

Notice that one can enter exponential expressions in Mathematica by typing the command **Exponent** or by using the concerned buttons in the palettes. The results in equations (1.4.1), (1.4.2) and (1.4.3) can be verified using the accompanying examples by entering and executing as

```
x^4×x^3
(x^4)^3
x^4/x^2
x^7
x^12
x^2
```

Section S1.5.5

One can enter logarithm in Mathematica either by typing **Log** or by entering logarithm from the Palettes menu. No matter how it is entered, the command **Log[*b*,*z*]** yields the common logarithm of ***z*** to the base ***b*** and the command **Log[*z*]** yields the natural logarithm of ***z*** (logarithm to the base ***e***) .

As an example, assume that we need to compute the common logarithms of 1, 10, and 100. These logarithms can be found as

```
N[Log[10,1]]
N[Log[10,10]]
N[Log[10,100]]
0.
1.
2.
```

Notice that these results are in accordance with our results in equation (1.5.3). The reader must have observed the letter **N** in the last three commands. If we didn't use this letter, we would have obtained the same outputs as the inputs. The letter **N** in Mathematica is the reserved notation for the command **N[expr]**. This command returns the numerical value of the expression **expr**.

As another example, suppose that we need to compute the natural logarithms of e, 1, and 10. These logarithms can be computed as

```
Log[e]
Log[1]
Log[e^10]
1
0
10
```

Notice that these are in accordance with our results in equation (1.5.4).

Let us now verify the properties in equations (1.5.5) and (1.5.6) using two numbers $x = 2$ and $y = 3$ as

```
Log[2×3]==Log[2]+Log[3]
Log[2/3]==Log[2]-Log[3]
True
True
```

Similarly, the properties in equations (1.5.11) through (1.5.13) can be verified as

```
e^Log[X]
Log[b,n]==Log[a,n]/Log[a,b]
Log[a,e]==1/Log[e,a]
x
True
True
```

Section S1.6.9

One of the important uses of Mathematica is to solve equations. Equations can be solved in Mathematica using the built-in-command **Solve**. The command **Solve[*eqns*,*vars*]** gives solution to an equation or set of equations ***eqns*** involving the variables ***vars*** and the command **Solve[*eqns*,*vars*,*elims*]** gives the solution by eliminating ***elims***.

Although one can obtain the complete information on this command from the Documentation Centre, few points must be noticed here. When the command **Solve** is used, equations must be entered in the form LHS==RHS, using the equal sign twice. Simultaneous equations can be entered either as a list or using the "and" sign (**&**) twice as **&&**. The command **Solve** returns the solution as ***x->sol*** (where ***x*** represents the variable and ***sol*** denotes the solution) in the case of single variable and returns the solution set **{*x*->s_x,*y*->s_y,...}** when there are several variables. If there is no solution, then the command returns a null set **{}** and returns **{{}}** if all variables can have all possible solutions.

As example, suppose that we need to solve the linear equation in one variable $2x + 7 = 17 - 8x$, which we solved at the beginning of Section 1.6.3. This equation can be solved using the command **Solve[*eqns*,*vars*]** as

```
Solve[2x+7==17-8x,x]
```

```
x→1
```

, which was the solution we obtained. Now consider the quadratic and cubic equations we solved in Section 1.6.3: $x^2 - 3x - 10 = 0$ and $x^3 - 2x^2 + x = 0$. The solutions of these equations are

```
Solve[x^2-3x-10==0,x]
Solve[x^3-2x^2+x==0,x]
{{x→-2},{x→5}}
{{x→0},{x→1},{x→1}}
```

, which were precisely the solutions we obtained.

Let us now consider the solution of systems of linear equations. As an example, consider the first system we solved in Section 1.6.5. The equations in this system were $y = 10 + 2x$ and $y = 5 + 4x$. This system can be solved as

```
Solve[{y==10+2x,y==5+4x},{x,y}]
{{x→5/2,y→15}}
```

As another example, consider the system in equation (1.6.14). The equations in this system were $z = 6 - 2x - y$, $z = 2 - x - 2y$, and $z = 2 - 0.5x - 0.5y$. The solution of this system is

```
Solve[{z==6-2x-y,z==2-x-2y,z==2-0.5x-0.5y},{x,y,z}]
{{x→3.,y→-1.,z→1.}}
```

We shall now attempt to solve systems of nonlinear equations. Let us first solve the first system we solved in Section 1.6.6. The equations in this system were $x^2 + x + 2 = 2y$ and $4x + 5 = y$. This system can be solved as

```
Solve[{x^2+x+2==2y,4x+5==y},{x,y}]
{{y→1,x→-1},{y→37,x→8}}
```

As the last example, consider the system in equation (1.6.16). The equations in this system were $z = x^2 - y^2$, $z = x^2 - y$, and $z = x - y^2$. As before, this system can be solved as

```
Solve[{z==x^2-y^2,z==x^2-y,z==x-y^2},{x,y,z}]
{{z→-1,x→0,y→1},{z→0,x→0,y→0},{z→0,x→1,y→1},{z→1,x→1,y→0}}
```

Section S1.7.9

Let us first see the ways of testing inequalities in Mathematica. If we wish to know whether an inequality is true or false, we may use the notations **>**, **<**, **<=** (or≤), and **>=** (or ≥).

Suppose that we need to test whether 2 is less than 1, 2 is less than or equal to 1, 2 is greater than 1, and 2 is greater than or equal to 1. If we enter these statements using above in equality signs we obtain the respective output as

```
2<1
2≤1
2>1
2≥1
False
False
True
True
```

We may also use multiple inequality signs to test inequalities. Suppose that we want to test whether inequalities $2 < 1 < 3$ and $3 > 2 > 1$ are true or false. This can be done as

```
2<1<3
3>2>1
False
True
```

Let us now see how we can solve inequalities in Mathematica. This can be done using the built-in-command **Reduce**. The command **Reduce[*expr*,*vars*]** solves the statement ***expr*** by solving equations or inequalities for ***vars***. As an example, suppose that we need to solve $3(2 - a) < 5$ and $0 < 2/a < 5$. Entering these inequalities with the command **Reduce** yield

```
Reduce[3(2-a)<5]
Reduce[0<2/a<5]
a>1/3
a>2/5
```

Absolute values can be found in Mathematica using the command **Abs**. The command **Abs[*x*]** returns the absolute value of the real or complex number *x*. As an example, the absolute values of -3 and {-2,4,-6} can be found as

```
Abs[-3]
Abs[{-2,4,-6}]
3
{2,4,6}
```

Section S1.9.8

Mathematica has a built-in-command to find the limits of most functions. This command is **Limit**. The command **Limit[*expr*,*x*->x_0]** finds the limit of the expression ***expr*** as ***x*** tends to x_0. As an example, consider finding the limit of the function $y = f(x) = (x^2 - 1)/(x - 1)$, which we evaluated in Section 1.9.1. We may find the limit of this function when $x \to 1$ as

```
Limit[(x^2-1)/(x-1),x→1]
2
```

Suppose now that we need to find the left-side and right-side limits of the same function as $x \to 1$. These limits can be found using the commands **Limit[*expr*,*x*->x_0,Direction->1]** and **Limit[*expr*,*x*->x_0,Direction->-1]**, respectively. Let us now apply these commands with our function to obtain the respective results as

```
Limit[(x^2-1)/(x-1),x→1,Direction→-1]
Limit[(x^2-1)/(x-1),x→1,Direction→1]
2
2
```

Section S1.10.9

One can generate sequences in Mathematica using the built-in-command **Table**. Since we have introduced some of the features of this command in Section 11.5.5 we do not repeat them here. However, we shall present here how we can generate sequences using this command.

Suppose that we need to generate the three arithmetic progressions we dealt with in the beginning of Section 1.10.4. These can be generated using the command **Table** as

```
Table[i+1,{i,0,40,10}]
Table[i+1,{i,0,4,1}]
Table[10-i,{i,0,3,1}]
{1,11,21,31,41}
{1,2,3,4,5}
{10,9,8,7}
```

The sums of these arithmetic progressions, or the associated arithmetic series, can be found using the command **Sum** as

```
Sum[i+1,{i,0,40,10}]
Sum[i+1,{i,0,4,1}]
Sum[10-i,{i,0,3,1}]
105
15
34
```

The command **Table** can also be applied to generate geometric progression. For example, the geometric progression we mentioned in Section 1.10.5 can be generated as

```
Table[2^k,{k,0,6,1}]
{1,2,4,8,16,32,64}
```

And the sum of this geometric progression, or the associated geometric series, can also be found using the command **Sum** as

```
Sum[2^k,{k,0,6,1}]
127
```

Factorials can be found in Mathematica using the built-in-notation **!** or the command **Factorial**. For example, 3 factorial can be found as

```
3!
Factorial[3]
```

```
6
6
```

Permutations can be found using the command **Permutations**. For example, the number of permutations of the first three letters of English alphabet is

```
Permutations[{a,b,c}]
{{a,b,c},{a,c,b},{b,a,c},{b,c,a},{c,a,b},{c,b,a}}
```

If we need the number of permutations of the first three letters of English alphabet taken two at a time, it can be obtained as

```
Permutations[{a,b,c},{2}]
{{a,b},{a,c},{b,a},{b,c},{c,a},{c,b}}
```

Section S1.12.7

Mathematica has many commands to manipulate trigonometric expressions. Before dealing with these commands, we shall first convert degrees into radians and vice versa using equation (1.12.3) in Mathematica. Suppose that we want to find the radian equivalent of 270 degrees and the degree equivalent of $\pi/4$ radians. These can be obtained (to conform to the values in Table 1.12.1) as

```
270×(π/180)
((π/4)×(180/π))
(3 π)/2
45
```

Let us now evaluate trigonometric functions of x for different values of x. Assume that we need to find $\sin x$, $\cos x$ and $\tan x$ when $x = 2.5\pi$, $x = 4\pi$, and $x = \pi$, respectively. These can be found (to conform to the results in Panels (A) through (C), respectively, of Figure 1.12.5) as

```
N[Sin[2.5π]]
N[Cos[4π]]
N[Tan[π]]
1.
1.
0.
```

MATHEMATICA APPLICATIONS IN CHAPTER 2

Section S2.2.10

We have introduced the Mathematica commands **List**, **Table** and **Array**, and their manipulations in Section 11.5. We do not repeat them here. However, notice that one can use these commands to generate vectors. Suppose that we want to generate the vectors **u'** =[1 2] and **v'**=[2 1] in Mathematica. These can be done using the command **List** as

```
lis1=List[1,2]
lis2=List[2,1]
{1,2}
{2,1}
```

Notice that one can generate vector (or matrices) clicking **Menu Bar → Palettes → Basic Math Assistant → Typesetting →** $\begin{pmatrix} \square \\ \square \end{pmatrix}$. One can also add more rows (or columns) to the last vector by clicking the coma key (or enter

key) while holding down the control key on the keyboard. We will sometimes use this method of entering vectors and matrices in Mathematica.

One can test whether **lis1** and **lis2** are vectors using the command

```
VectorQ as
VectorQ[lis1]
VectorQ[lis2]
True
True
```

One can also express **lis1** and **lis2** in vector forms using the command

```
MatrixForm or using the post fixing notation // as
MatrixForm[lis1]
lis2//MatrixForm
```

$$\begin{pmatrix}1\\2\end{pmatrix}, \qquad \begin{pmatrix}2\\1\end{pmatrix}.$$

Let us now carry out few operations on our vectors. Suppose that we need to multiply **lis1** by the scalar 0.5, add the two vectors **lis1** and **lis2**, and subtract **lis2** from **lis1**. These operations can be carried out and the associated outputs can be expressed in vector forms as

```
0.5×lis1//MatrixForm
lis1+lis2//MatrixForm
lis1-lis2//MatrixForm
```

$$\begin{pmatrix}0.5\\1.\end{pmatrix}, \qquad \begin{pmatrix}3\\3\end{pmatrix}, \qquad \text{and} \qquad \begin{pmatrix}-1\\1\end{pmatrix}.$$

The dot or inner product of the vectors in our example can be obtained using the command **Dot** . Therefore, entering **Dot[lis1,lis2]** or simply **lis1.lis2** yields

```
Dot[lis1,lis2]
lis1.lis2
4
4
```

Section S2.3.11

Mathematica has a large number of built-in-commands to deal with vectors and matrices. But, we shall present only those commands that we use normally. We shall begin with scalar multiplication. If we carry out the same scalar multiplication as the one we did in Section 2.3.2 we obtain

```
mat = ( a11  a12 )
      ( a21  a22 );
MatrixForm[s × mat]
```

$$\begin{pmatrix} s\,a_{11} & s\,a_{12} \\ s\,a_{21} & s\,a_{22} \end{pmatrix}$$

which exactly was the result we obtained. Let us now carry out some of the matrix operations using the same matrices we used in Section 2.3.3. Therefore, we have **A** + **B** and **A** – **B** as

```
A = ( 1  2 ); B = ( 5  6 ); MatrixForm[A + B]
    ( 3  4 )     ( 7  8 )
A = ( 1  2 ); B = ( 5  6 ); MatrixForm[A - B]
    ( 3  4 )     ( 7  8 )
```

$$\begin{pmatrix} 6 & 8 \\ 10 & 12 \end{pmatrix}$$

$$\begin{pmatrix} -4 & -4 \\ -4 & -4 \end{pmatrix}$$

Let us now carry out multiplication of matrices. The product of the two matrices can be obtained as

```
A = {{a11, a12, a13}, {a21, a22, a23}, {a31, a32, a33}}; B = {{b11, b12, b13}, {b21, b22, b23}, {b31, b32, b33}}; MatrixForm[A.B]
```

$$\begin{pmatrix} a_{11} b_{11} + a_{12} b_{21} + a_{13} b_{31} & a_{11} b_{12} + a_{12} b_{22} + a_{13} b_{32} & a_{11} b_{13} + a_{12} b_{23} + a_{13} b_{33} \\ a_{21} b_{11} + a_{22} b_{21} + a_{23} b_{31} & a_{21} b_{12} + a_{22} b_{22} + a_{23} b_{32} & a_{21} b_{13} + a_{22} b_{23} + a_{23} b_{33} \\ a_{31} b_{11} + a_{32} b_{21} + a_{33} b_{31} & a_{31} b_{12} + a_{32} b_{22} + a_{33} b_{32} & a_{31} b_{13} + a_{32} b_{23} + a_{33} b_{33} \end{pmatrix}$$

which was the result we obtained in Section 2.3.4. As a specific numerical example, consider the following product

```
A = {{1, 2, 2}, {1, 2, 2}}; B = {{2, 1}, {1, 2}, {2, 1}}; MatrixForm[A.B]
```

$$\begin{pmatrix} 8 & 7 \\ 8 & 7 \end{pmatrix}$$

The transpose of a matrix can be obtained using the built-in-command **`Transpose`**. The transpose of the following matrix can be obtained as

```
A = {{a11, a12, a13}, {a21, a22, a23}, {a31, a32, a33}}; MatrixForm[Transpose[A]]
```

$$\begin{pmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{pmatrix}.$$

We can also carry out row reduction in Mathematica. The command for this is **`RowReduce`**. Using the first example in Section 2.3.7 with this command yields

```
A = {{2, 1, 1, 6}, {1, 2, 1, 2}, {0.5, 0.5, 1, 2}}; MatrixForm[RowReduce[A]]
```

$$\begin{pmatrix} 1 & 0. & 0. & 3. \\ 0 & 1 & 0. & -1. \\ 0 & 0 & 1 & 1. \end{pmatrix}$$

Notice that this result was precisely the result we obtained.

Let us now attempt to solve SSLEs. There are different ways to solve a SSLEs in Mathematica. One way is to use the built-in-command **`LinearSolve`**. **`LinearSolve[`*`m`*`,`*`b`*`]`** finds an ***x*** which solves the matrix equation ***m.x==b***. Using this command with the system in equation (2.3.9)

```
m = {{2, 1}, {1, 2}}; b = {4, 2}; LinearSolve[m, b]
{2, 0}
```

yields precisely the same result as that we obtained in Section 2.3.8. As another example, consider the system (2.3.10). This system can be solved as

```
m={{2,1,1},{1,2,1},{1,1,2}};b={10,5,5}; LinearSolve[m,b]
{5,0,0}
```

Section S2.4.8

Mathematica has a built-in-command to find the determinant of a matrix: **`Det`**. **`Det[`*`m`*`]`** computes the determinant of the square matrix ***m***. Suppose that we want to compute the determinants of the three matrices in Section 2.4.1. These can be computed as

```
A = ( 1  2 ); Det[A]
    ( 3  4 )

    ( a11  a12  a13 )
B = ( a21  a22  a23 ); Det[B]
    ( a31  a32  a33 )

    ( 1  2  3 )
F = ( 4  5  6 ); Det[F]
    ( 7  8  9 )
```

$$-2$$

$$-a_{13}a_{22}a_{31} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33} + a_{11}a_{22}a_{33}$$

$$0$$

which are identical with the results we obtained in Section 2.4.1.

Minors of a square matrix ***m*** can be found by using the command **Minors[*m*]**. The minors of the two matrices we dealt with in Section 2.4.3 can be found as

```
    ( 1  2  3 )
A = ( 4  5  6 ); MatrixForm[Minors[A]]
    ( 7  8  9 )

    ( a11  a12  a13 )
B = ( a21  a22  a23 ); MatrixForm[Minors[B]]
    ( a31  a32  a33 )
```

$$\begin{pmatrix} -3 & -6 & -3 \\ -6 & -12 & -6 \\ -3 & -6 & -3 \end{pmatrix}$$

$$\begin{pmatrix} -a_{12}a_{21} + a_{11}a_{22} & -a_{13}a_{21} + a_{11}a_{23} & -a_{13}a_{22} + a_{12}a_{23} \\ -a_{12}a_{31} + a_{11}a_{32} & -a_{13}a_{31} + a_{11}a_{33} & -a_{13}a_{32} + a_{12}a_{33} \\ -a_{22}a_{31} + a_{21}a_{32} & -a_{23}a_{31} + a_{21}a_{33} & -a_{23}a_{32} + a_{22}a_{33} \end{pmatrix}$$

Section S2.5.8

The inverse of a square matrix ***m***, if it exists, can be found in Mathematica using the built-in-command **Inverse**. **Inverse[*m*]** gives the inverse of a square matrix ***m***. Suppose that we need to find the inverse of the first 3×3 matrix we used in Section 2.5.2. Using the command **Inverse** we obtain

```
    ( 1  2  1 )
A = ( 1  1  2 ); MatrixForm[Inverse[A]]
    ( 2  2  1 )
```

$$\begin{pmatrix} -1 & 0 & 1 \\ 1 & -\frac{1}{3} & -\frac{1}{3} \\ 0 & \frac{2}{3} & -\frac{1}{3} \end{pmatrix},$$

which exactly was the inverse we obtained in Section 2.5.2.

Let us now solve in Mathematica the first system we solved in Section 2.5.5. This system can be solved, as we saw in that section, using equation 2.5.2: $\mathbf{x} = \mathbf{A}^{-1}\mathbf{d}$. Then, we can obtain the same solution as

```
    ( 2    1   1 )          ( 6 )          ( x1 )
A = ( 1    2   1 ); b = ( 2 ); x = ( x2 ) = MatrixForm[Inverse[A].b]
    ( 0.5  0.5 1 )          ( 2 )          ( x3 )
```

$$\begin{pmatrix} 3. \\ -1. \\ 1. \end{pmatrix}$$

Section S2.6.4

Mathematica has a built-in-command for finding the rank of a matrix: **MatrixRank**. Using this command with the matrix in Section 2.6.1 yields

```
B = {{1, 2, 3}, {2, 4, 6}, {2, 3, 1}}; MatrixRank[B]
2,
```

which was exactly the same result as that we obtained. As another example, the rank of the following matrix can be found as

```
A = {{1, 2, 3, 4}, {5, 1, 4, 5}, {3, 2, 1, 4}}; MatrixRank[A]
3
```

Section S2.8.6

One can carry out most of the procedures we discussed so far in the present section in Mathematica. Let us begin with Jacobian matrices and Jacobians. As an example, consider the first numerical, simultaneous system of functions we dealt with in Section 2.8.1: $f^1 = 2x_1 + x_2$ and $f^2 = 4x_1^2 + 4x_1x_2 + x_2^2$. Although Mathematica does not have a built-in-command for finding Jacobian matrix, it can be found indirectly using the command **D**. The command **D[*f*,*x*]** yields the partial derivative **∂f/∂x**, which will be explained in detail shortly. Using command **D** with $f^1 = f, f^2 = g, x_1 = x, and\ x_2 = y$, we get the same result as the one we obtained in Section 2.8.1:

```
f=2×x+y;
g=4×x^2+4×x×y+y^2;
D[{f,g},{{x,y}}]//MatrixForm
{{2, 1}, {8 x + 4 y, 4 x + 2 y}}
```

The Jacobian of the above Jacobian matrix can be found by using the command for determinant as

```
Det[D[{f,g},{{x,y}}]]//MatrixForm
0
```

We shall now consider finding the discriminants and Hessians of square matrices. Suppose that we have two bivariate functions $U = f(x_1, x_2) = x_1^2 + x_2^2 - x_1 - x_2$ and $U = g(x_1, x_2) = x_1 + x_2 - x_1^2 - x_2^2$; and two trivariate functions $U = h(x_1, x_2) = x_1^2 + x_2^2 + x_3^2 - x - y - z$, and $U = j(x_1, x_2) = x + y + z - x_1^2 - x_2^2 - x_3^2$ which for convenience we convert to $U = f(x, y) = x^2 + y^2 - x - y$, $U = g(x, y) = x + y \quad -x^2 - y^2$, $U = h(x, y, z) = x^2 + y^2 + z^2 - x - y - z$, and $U = j(x, y, z) = x + y + z - x^2 - y^2 - z^2$ respectively, where $x_1 = x, x_2 = y$, and $x_3 = z$. Notice that the four numerical examples of sign definiteness of the quadratic forms we presented in Section 2.8.2 were in fact the quadratic forms of the last four functions.

We know from Section 2.8.3 that the matrix of the second partial derivatives of the function is called the Hessian matrix. As earlier, Mathematica does not have built-in-commands for finding the Hessian matrix and the Hessian. But, for these we can once again use the command **D**. Now using **D** with a specification for the second partial derivative we obtain, as in Section 2.8.2, the Hessian matrices and the Hessians associated with the last four functions' (or the discriminants associated with the last four functions') quadratic forms, respectively, as

```
f=x^2+y^2-x-y;
D[f,{{x,y},2}]//MatrixForm
Det[D[f,{{x,y},2}]]
{{2, 0}, {0, 2}}
```

```
g=x+y-x^2-y^2;
D[g,{{x,y},2}]//MatrixForm
Det[D[g,{{x,y},2}]]
{{-2, 0}, {0, -2}}
```

4. 4

```
h=x^2+y^2+z^2-x-y-z;
D[h,{{x,y,z},2}]//MatrixForm
Det[D[h,{{x,y,z},2}]]
```

$$\begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

8

```
j=x+y+z-x^2-y^2-z^2;
D[j,{{x,y,z},2}]//MatrixForm
Det[D[j,{{x,y,z},2}]]
```

$$\begin{pmatrix} -2 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & -2 \end{pmatrix}$$

-8

Notice that these are similar to the results we obtained in Section 2.8.2.

Lastly, we shall attempt to find the characteristic polynomial, eigenvalues and eigenvectors of a characteristic matrix in Mathematica. Suppose that our characteristic matrix is the 2×2 matrix in the numerical example in Section 2.8.4. Then the characteristic polynomial can be found using the built-in-command **`CharacteristicPolynomial[d,v]`**. This command yields the characteristic equation or polynomial of the characteristic matrix **`d`**, where we used lowercase to avoid conflict with the reserved Mathematica notation **`D`**. The built-in-commands **`Eigenvalues[d]`**, **`Eigenvectors[d]`**, and **`Eigensystem[d]`** yield the eigenvalues, eigenvectors, and eigensystem (eigenvalues and eigenvectors together) of the matrix **`d`**. As an example, suppose that our characteristic matrix is the 2×2 matrix in the numerical example in Section 2.8.4. Using this matrix and the above commands yield

```
d = {{2, 1}, {1, 2}};
CharacteristicPolynomial[a,v]
Eigenvalues[a]
Eigenvectors[a]//MatrixForm
Eigensystem[a]
```

$3 - 4\,v + v^2$

$\{3,1\}$

$$\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$$

$\{\{3,1\},\{\{1,1\},\{-1,1\}\}\}$

Notice that the characteristic polynomial and the eigenvalues here are the same as those we obtained in Section 2.8.4. Notice also that the eigenvectors we obtained here are different from those we obtained in that section. The reason for this difference is the fact that Mathematica chooses only the linearly independent eigenvectors in the case of eigenvectors corresponding to degenerate eigenvalues.

Section S2.9.7

We shall show in this section how to estimate the parameters of LPRF using sample data in Mathematica. Mathematica has more than one built-in-command for carrying out regression. The widely used commands are **`Fit[`*`data,funs,vars`*`]`**, **`LinearModelFit[{`*$y_1,y_2,\ldots$*`},{`*$f_1,f_2,\ldots$*`},`*x*`]`**, **`FindFit[`*`data,expr,pars, vars`*`]`**, etc. Moreover, many of the **post-regression diagnoses** can also be carry out using different options available which can be accessed at the Documentation Centre. We shall use the command **`Fit[`*`data,funs,vars`*`]`** to estimate the regression parameters in the case of the two examples in Section S2.9.6 . Using this command with the sets of data in Section S2.9.5 we obtain

```
data1={{7,28},{7.25,24},{7.5,23},{7.75,22},{8,21},{8.25,19}
      ,{8.5,16},{8.75,14},{9,11},{9.25,10}};
data2={{3,4,2},{3.1,3.9,1.9},{3,3.7,1.95},{3.25,3.75,1.85}
      ,{3.5,3.7,1.75},{3.7,3.5,1.5},{3.75,3.25,1.4},
      {3.7,3,1.45},{3.9,3.1,1.25},{4,3,1}};
Fit[data1,{1,x1},x1]
Fit[data2,{1,x1,x2},{x1,x2}]
```

$81.4364 - 7.70909\ x_1$

$2.94102 - 0.634588\ x_1 + 0.251776\ x_2$

which are identical with the estimates (except for the decimals after the second place) we obtained in Section S2.9.5.

Section S2.10.7

In this section we shall attempt to solve input-output problems in Mathematica. For this consider the problems in the illustrative examples in Section S2.10.2. Solving these problems (with lowercase notation "i" to denote identity matrix to avoid conflict with Mathematica's reserved notation I) in Mathematica we obtain the results

```
A = {{0.3, 0.5}, {0.4, 0.25}}; i = {{1, 0}, {0, 1}}; d = {{450}, {300}};
X = Inverse[i - A].d // MatrixForm
```

$$\begin{pmatrix} 1500. \\ 1200. \end{pmatrix}$$

```
A = {{0.35, 0.25, 0.15}, {0.3, 0.2, 0.2}, {0.15, 0.2, 0.35}}; i = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}}; d = {{350}, {250}, {550}};
X = Inverse[i - A].d // MatrixForm
```

$$\begin{pmatrix} 1353.55 \\ 1202.19 \\ 1528.42 \end{pmatrix}$$

Notice that these results are similar to the results we obtained in Section 2.10.2.

MATHEMATICA APPLICATIONS IN CHAPTER 3

Section S3.2.6

Although we introduced the command **D** earlier, we shall expose it a bit further here. Two other equivalent commands for finding derivatives in Mathematica are **f'** and **Derivative**. The command **D[*f*,*x*]** yields the partial derivative of the function ***f*** with respect to ***x***; **f'[x]** yields the derivative of **x** in the function **f[x_]**; and **Derivative[n_1,n_2,...][*f*]** yields the n_i^{th} derivative of the function ***f*** . We shall here apply the first two commands and the last one will be used later.

As an example, consider the profit function we dealt with in Section 3.2.1: $y = f(x) = x^2$. The derivative of this function with respect to x can be found using the first two commands above as

```
f[x_]:=x^2;
D[f[x],x]
f'[x]
2x
2x
```

Section S3.4.6

Higher-order derivatives can be found in Mathematica using the same built-in-commands which we used to find the first-order derivatives above: **D[*f*,*x*]** or **f'[x]**. Suppose that we need to find the second-derivative of the first two functions we used in Section 3.4.2. These can be found as

```
f[x_]:=2x-3x^2;
D[f[x],{x,2}]
f''[x]
-6
-6
```

```
g[x_]:=e^x;
D[g[x],{x,2}]
g''[x]
e^x
e^x
```

which are identical with the results we obtained in Section 3.4.2.

Section S3.6.7

Mathematica has a built-in-command to find the expansion of a function. This command is **Series**. The command **Series[*f*,{*x*,x_0,*n*}]** gives the power series expansion of ***f*** about the point ***x*=x_0** to order $(X - X_0)^n$. As an example, suppose that we need to find the expansions of the function $y = f(x) = 10 + 2x + 5x^2 + 4x^3$, which we used in Section 3.6.2, from degree zero to degree 3. These can be found using the command

Series[*f*,{*x*,x_0,*n*}] as

```
Series[10+2x+5x^2+4x^3,{x,0,0}]
Series[10+2x+5x^2+4x^3,{x,0,1}]
Series[10+2x+5x^2+4x^3,{x,0,2}]
Series[10+2x+5x^2+4x^3,{x,0,3}]
10+O[x]^1
10+2 x+O[x]^2
10+2 x+5 x^2+O[x]^3
10+2 x+5 x^2+4 x^3+O[x]^4
```

Notice that these results are identical with the results we obtained in Section 3.6.2. The reader must have noticed the command **Series[*f*,{*x*,x_0,*n*}]** can be used to find Maclaurin and Taylor series by using 0 and other value, respectively, for x_0.

Let us now find Euler relations in Mathematica. The command for finding Euler relations is **ExpToTrig[*expr*]**, converts the exponentials in ***expr*** to trigonometric functions . As an example, suppose that we need to find the Euler relations of the expressions e^{ix} and e^{-ix} (or that we need to convert the exponential expressions e^{ix} and e^{-ix} to equivalent trigonometric expressions). Notice that one can also carry out the reverse operations. These can be done (to yield precisely the same results as those we obtained in the last section) as

```
ExpToTrig[e^(i x)]
ExpToTrig[e^(-i x)]
Cos[x]+i Sin[x]
Cos[x]-i Sin[x]
```

```
TrigToExp[Cos[x]+i Sin[x]]
TrigToExp[Cos[x]-i Sin[x]]
e^(i x)
e^(-i x)
```

Section S3.7.7

Recall that we have used the built-in-command **D** earlier. In this section we shall use it in the context of partial derivatives. Suppose that we need to find out the first partial derivatives of the two functions we considered in Section 3.7.1: $y = f(x_1, x_2) = x_1^2 + x_2^2$ and $y = g(x_1, x_2) = x_1^2 + x_2^2 + x_1 x_2$. These partial derivatives can be found in Mathematica as

```
f=x1^2 + x2^2;
g=x1^2 + x2^2 + x1 x2;
D[f,x1]
2x1
```

```
D[f,x2]
2x2
```

```
D[g,x1]
2x1+x2
```

```
D[g,x2]
x1+2x2
```

The higher partial derivatives (the second and the third) of the above two functions can be obtained as

```
f=x1^2 + x2^2;
g=x1^2 + x2^2 + x1 x2;
D[f,{x1,2}]
2
```

```
D[f,{x1,3}]
0
```

```
D[f,{x2,2}]
2
```

```
D[f,{x2,3}]
0
```

```
D[g,{x1,2}]   D[g,{x1,3}]   D[g,{x2,2}]   D[g,{x2,3}]
2             0             2             0
```

Mixed partial derivatives can also found using the command **D**. As an example, suppose that our multivariate function is the one we used in Section 3.7.3: $z = f(x,y) = x^3y^3$. Then the associated mixed partial derivatives can be found as

```
f=x^3y^3;
D[f,x]        D[f,{x,2}]    D[f,{x,3}]    D[f,{x,4}]
3x^2y^3       6xy^3         6y^3          0
D[f,y]        D[f,{y,2}]    D[f,{y,3}]    D[f,{y,4}]
3x^3y^2       6x^3y         6x^3          0
D[f,x,y]      D[f,x,y,x]    D[f,y,x]      D[f,y,x,y]
9x^2y^2       18xy^2        9x^2y^2       18x^2y
```

Section S3.8.9

The built-in-command **Dt[*f*]** can be used in Mathematica to find the total differential ***d f*** of the function ***f***. As an example, consider the function we used in Section 3.8.1: $y = f(x) = 20 - 15x + 12x^2$. The first order total differential of this function can be obtained as

```
f=20-15x+12x^2;
a=Dt[f]
-15Dt[x]+24xDt[x],
```

where `Dt[x]` denotes dx. Notice that this was the result we obtained in Section 3.8.1. Higher differentials can be obtained by finding the successive differentials as

```
b=Dt[a]
24Dt[x]^2-15Dt[Dt[x]]+24xDt[Dt[x]]
Dt[b]
72Dt[x]Dt[Dt[x]]-15Dt[Dt[Dt[x]]]+24xDt[Dt[Dt[x]]]
```

In the case of multivariate functions such as the function $Q = f(K,L) = K^{\alpha}L^{1-\alpha}$ in example 2 in Section 3.8.7, with $\alpha = 0.5$, the partial and total differential can also be obtained using the command **Dt[*f*]** as

```
f=k^0.5 l^1-0.5;
Dt[f]
(0.5 l^0.5 Dt[k])/k^0.5 +(0.5 k^0.5 Dt[l])/l^0.5,
```

which is identical, given dk = `Dt[k]` and dl = `Dt[l]`, with the result we obtained in the solution to the problem in the said example. It is needless to mention that higher total differentials of multivariate functions can be obtained by finding the successive differentials.

Total derivatives can be obtained using the built-in-command **Dt[*f*,*t*]**. This command yields the total derivative ***df/dt*** of the function ***f*** with respect to ***t***. As an example, consider the function we used in Section 3.8.5: $y = f(x) = 10xt + 2x^2$, where $x = g(t) = t^2$. The total derivative of this function with respect to t can be found as

```
f[x_,t_]:=10xt+2x^2;
Dt[f[x,t],t]
4xDt[x,t]+10Dt[xt,t],
```

where `Dt[x,t]` denotes dx/dt. If we find dx/dt and substitute the result, along with $x = g(t) = t^2$, into the result `4xDt[x,t]+10Dt[xt,t]`, we will obtain the same result as the one we obtained in Section 3.8.5. Higher total derivatives can also be obtained as mentioned earlier.

MATHEMATICA APPLICATIONS IN CHAPTER 4

Section S4.2.10

Mathematica has a number of built-in-commands to find the optima of univariate objective functions. We shall present few of these commands below. Notice that more information on these commands can be found at the Documentation Centre.

The commands related to minimization of univariate functions include: **Minimize[*f*,*x*]** minimizes ***f*** with respect to ***x***; **NMinimize[*f*,*x*]** minimizes ***f*** numerically with respect to ***x***; **FindMinimum[*f*,*x*]** searches for local minimum in ***f*** starting from an automatically selected point; **FindMinimum[*f*,{*x*,x_0}]** searches for local minimum in ***f*** starting point $x=x_0$; **MinValue[*f*,*x*]** minimizes the value of ***f*** with respect to ***x***; **NMinValue[*f*,*x*]** minimizes numerically the value of ***f*** with respect to ***x***; **FindMinValue[*f*,*x*]** yields the value at a local minimum of ***f***; **FindMinValue [*f*,{*x*,x_0}]** gives the value at a local minimum of ***f*** found by a search starting from the point $x=x_0$; **ArgMin[*f*,*x*]** gives a position x_{min} at which ***f*** is minimized; **FindArgMin[*f*,*x*]** gives the position x_{min} of a local maximum of ***f***; and **NArgMin[*f*,*x*]** gives a position x_{min} at which ***f*** is numerically minimized.

Similarly, the commands related to maximization of univariate functions include: **Maximize[*f*,*x*]** maximizes ***f*** with respect to ***x***; **NMaximize[*f*,*x*]** maximizes ***f*** numerically with respect to ***x***; **FindMaximum[*f*,*x*]** searches for local maximum in ***f*** starting from an automatically selected point; **FindMaximum[*f*,{*x*,x_0}]** searches for local naximum in ***f*** starting point $x=x_0$; **MaxValue[*f*,*x*]** maximizes the value of ***f*** with respect to ***x***; **NMaxValue[*f*,*x*]** maximizes numerically the value of ***f*** with respect to ***x***; **FindMaxValue[*f*,*x*]** yields the value at a local naximum of ***f***; **FindMaxValue [*f*,{*x*,x_0}]** gives the value at a local maximum of ***f*** found by a search starting from the point $x=x_0$; **ArgMax[*f*,*x*]** gives a position x_{max} at which ***f*** is maximized; **FindArgMax[*f*,*x*]** gives the position x_{max} of a local maximum of ***f***;and **NArgMax[*f*,*x*]** gives a position x_{max} at which ***f*** is numerically maximized.

Although one can use any of the above commands, we use here the commands **Minimize[*f*,*x*]**, **NMinimize[*f*,*x*]**, **FindMinimum[*f*,*x*]**, and **FindMinimum[*f*,{*x*,x_0}]** as examples of univariate

minimization commands. Using these commands with the function $y = f(x) = 10 - 4x + x^2$ in our minimization example in Section 4.2.4, we obtain here the same results as those we obtained there:

```
Minimize[10-4x+x^2,x]
{6,{x→2}}
FindMinimum[10-4x+x^2,x]
{6.,{x→2.}}
NMinimize[10-4x+x^2,x]
{6.,{x→2.}}
FindMinimum[10-4x+x^2,{x,0}]
{6.,{x→2.}}
```

Similarly, we shall use here the commands **Maximize[*f*,*x*]**, **NMaximize[*f*,*x*]**, **FindMaximum[*f*,*x*]**, **FindMaximum[*f*,{*x*,x_0}]** as examples of univariate maximization commands. Using these commands with the function $y = g(x) = 10 + 4x - x^2$ in our maximization example in Section 4.2.4, we obtain here the same results as those we obtained there:

```
Maximize[10+4x-x^2,x]
{14,{x→2}}
FindMaximum[10+4x-x^2,x]
{14.,{x→2.}}
NMaximize[10+4x-x^2,x]
{14.,{x→2.}}
FindMaximum[10+4x-x^2,{x,0}]
{14.,{x→2.}}
```

Section S4.3.8

As in the case of optimization of univariate objective functions, Mathematica has a number of built-in-commands to find the extrema of multivariate objective functions. We shall present few of these commands below. Notice that more information on these commands can be found at the Documentation Centre.

The commands related to minimization of multivariate functions include: **Minimize[*f*,{*x*,*y*,...}]** minimizes ***f*** with respect to **{*x*,*y*,...}**; **NMinimize [*f*,{*x*,*y*,...}]** minimizes ***f*** numerically with respect to **{*x*,*y*,...}**; **FindMinimum [*f*,{{*x*,x_0},{*y*,y_0},...}]** searches for local minimum in ***f*** of several variables; **MinValue[*f*,{*x*,*y*,...}]** minimizes the value of ***f*** with respect to **{*x*,*y*,...}**; **NMinValue[*f*,{*x*,*y*,...}]** minimizes numerically the value of ***f*** with respect to **{*x*,*y*,...}**; **FindMinValue[*f*,{{*x*,x_0},{*y*,y_0},...}]** gives the value at a local minimum of ***f*** of several variables; **ArgMin[*f*,{*x*,*y*,...}]** gives a position **{x_{min},y_{min},...}** at which ***f*** is minimized; **FindArgMin[*f*,{{*x*,x_0},{*y*,y_0},...}]** gives the position **{x_{min},y_{min},...}** of a local maximum of ***f***; and **NArgMin[*f*,{*x*,*y*,...}]** gives a position **{x_{min},y_{min},...}** at which *f* is numerically minimized.

The commands related to maximization of multivariate functions include: **Maximize[*f*,{*x*,*y*,...}]** maximizes ***f*** with respect to **{*x*,*y*,...}**; **NMaximize [*f*,{*x*,*y*,...}]** maximizes ***f*** numerically with respect to **{*x*,*y*,...}**; **FindMaximum [*f*,{{*x*,x_0},{*y*,y_0},...}]** searches for local maximum in ***f*** of several variables; **MaxValue[*f*,{*x*,*y*,...}]** maximizes the value of ***f*** with respect to **{*x*,*y*,...}**; **NMaxValue[*f*,{*x*,*y*,...}]** maximizes numerically the value of ***f*** with respect to **{*x*,*y*,...}**; **FindMaxValue[*f*,{{*x*,x_0}, {*y*,y_0},...}]** gives the value at a local maximum of ***f*** of several variables; **ArgMax[*f*,{*x*,*y*,...}]** gives a position **{x_{min},y_{min},...}** at which ***f*** is maximized; **FindArgMax[*f*,{{*x*,x_0},{*y*,y_0},...}]** gives the position **{x_{min},y_{min},...}** of a local maximum of ***f***; and **NArgMax[*f*,{*x*,*y*,...}]** gives a position **{x_{min},y_{min},...}** at which *f* is numerically maximized.

Although one can use any of the above commands, we use here the commands **Minimize[*f*,{*x*,*y*,...}]** and **NMinimize[*f*,{*x*,*y*,...}]** as examples of multivariate minimization commands. Using these commands with the functions $U = f(x_1, x_2) = x_1^2 + x_2^2 - x_1 - x_2$ and $U = f(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2 - x_1 - x_2 - x_3$ in our minimization examples in Sections 4.3.1 and 4.3.3, respectively, we obtain here the same results as those we obtained there:

```
Minimize[x1^2 + x2^2 - x1 - x2, {x1, x2}]
NMinimize[x1^2 + x2^2 - x1 - x2, {x1, x2}]
Minimize[x1^2 + x2^2 + x3^2 - x1 - x2 - x3, {x1, x2, x3}]
NMinimize[x1^2 + x2^2 + x3^2 - x1 - x2 - x3, {x1, x2, x3}]
{-(1/2),{x1→1/2,x2→1/2}}
{-0.5,{x1→0.5,x2→0.5}}
{-(3/4),{x1→1/2,x2→1/2,x3→1/2}}
{-0.75,{x1→0.5,x2→0.5,x3→0.5}}
```

Similarly, we use here the commands **Maximize[*f*,{*x*,*y*,...}]** and **NMaximize[*f*,{*x*,*y*,...}]** as examples of multivariate minimization commands. Using these commands with the functions $U = f(x_1, x_2) = x_1 + x_2 - x_1^2 - x_2^2$ and $U = f(x_1, x_2, x_3) = x_1 + x_2 + x_3 - x_1^2 - x_2^2 - x_3^2$ in our minimization examples in Sections 4.3.1 and 4.3.3, respectively, we obtain here the same results as those we obtained there:

```
Maximize[x1 + x2 - x1^2 - x2^2, {x1, x2}]
NMaximize[x1 + x2 - x1^2 - x2^2, {x1, x2}]
Maximize[x1 + x2 + x3 - x1^2 - x2^2 - x3^2, {x1, x2, x3}]
NMaximize[x1 + x2 + x3 - x1^2 - x2^2 - x3^2, {x1, x2, x3}]
{1/2,{x1→1/2,x2→1/2}}
{0.5,{x1→0.5,x2→0.5}}
{3/4,{x1→1/2,x2→1/2,x3→1/2}}
{0.75,{x1→0.5,x2→0.5,x3→0.5}}
```

Section S4.4.13

As in the case of unconstrained optimization, Mathematica has a number of built-in-commands to solve constrained optimization problems. We shall present few of these commands below. It is needless to say that more information on these commands can be found at the Documentation Centre.

The commands related to constrained minimization include: **Minimize[{*f*,*cons*},{*x*,*y*,...}]** minimizes ***f*** subject to constrains ***cons***; **NMinimize[{*f*,*cons*},{*x*,*y*,...}]** minimizes ***f*** numerically subject to constrains ***cons***; **FindMinimum[{*f*,*cons*},{{*x*,x_0},{*y*,y_0},...}]** searches for local minimum in ***f*** of several variables subject to constrains ***cons***; **MinValue[{*f*,*cons*},{*x*,*y*,...}]** minimizes the value of ***f*** subject to constrains ***cons***; **NMinValue[{*f*,*cons*},{*x*,*y*,...}]** minimizes numerically the value of ***f*** subject to constrains ***cons***; **FindMinValue[{*f*,*cons*},{{*x*,x_0},{*y*,y_0},...}]** gives the value at a local minimum of ***f*** of several variables subject to constrains ***cons***; **ArgMin[{*f*,*cons*},{*x*,*y*,...}]** gives a position **{x_{min},y_{min},...}** at which ***f*** is minimized subject to constrains ***cons***; **FindArgMin[{*f*,*cons*},{{*x*,x_0},{*y*,y_0},...}]** gives the position **{x_{min},y_{min},...}** of a local maximum of ***f*** subject to constrains ***cons***; and **NArgMin[{*f*,*cons*},{*x*,*y*,...}]** gives a position **{x_{min},y_{min},...}** at which *f* is numerically minimized subject to constrains ***cons***.

The commands related to constrained maximization include: **Maximize[{*f*,*cons*},{*x*,*y*,...}]** maximizes ***f*** subject to constrains ***cons***; **NMaximize[{*f*,*cons*},{*x*,*y*,...}]** maximizes ***f*** numerically subject to constrains ***cons***; **FindMaximum[{*f*,*cons*},{{*x*,x_0},{*y*,y_0},...}]** searches for local maximum in ***f*** of several variables subject to constrains ***cons***; **MaxValue[{*f*,*cons*},{*x*,*y*,...}]** maximizes the value of ***f*** subject to constrains ***cons***; **NMaxValue[{*f*,*cons*},{*x*,*y*,...}]** maximizes numerically the value of ***f*** subject to constrains ***cons***; **FindMaxValue[{*f*,*cons*},{{*x*,x_0},{*y*,y_0},...}]** gives the value at a local maximum of ***f*** subject to constrains ***cons***; **ArgMax[{*f*,*cons*},{*x*,*y*,...}]** gives a position **{x_{min},y_{min},...}** at which ***f*** is maximized subject to constrains ***cons***; **FindArgMax[{*f*,*cons*},{{*x*,x_0},{*y*,y_0},...}]** gives the position **{x_{min},y_{min},...}** of a local maximum of ***f*** subject to constrains ***cons***; and **NArgMax[{*f*,*cons*},{*x*,*y*,...}]** gives a position **{x_{min},y_{min},...}** at which *f* is numerically maximized subject to constrains ***cons***.

Although one can use any of the above commands, we use here the commands **Minimize[{*f*,*cons*},{*x*,*y*,...}]** and **NMinimize[{*f*,*cons*},{*x*,*y*,...}]** as examples of constrained minimization commands. Using these commands with the minimization problems $C = f(q_1, q_2) = q_1^2 + q_2^2 - q_1 q_2$ subject to $q_1 + q_2 = 10$ and $U = f(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2 - x_1 - x_2 - x_3$ subject to $x_1 + x_2 + x_3 = 6$ we solved in Sections 4.4.4 and 4.4.6, respectively, we obtain here the same results as those we obtained there:

```
Minimize[{q1^2 + q2^2 - q1*q2, q1+q2==10},{q1,q2}]
NMinimize[{q1^2 + q2^2 - q1*q2, q1+q2==10},{q1,q2}]
Minimize[{x1^2 + x2^2 + x3^2 - x1 - x2 - x3, x1+x2+x3==6},{x1,x2,x3}]
NMinimize[{x1^2 + x2^2 + x3^2 - x1 - x2 - x3, x1+x2+x3==6},{x1,x2,x3}]
{25,{q1→5,q2→5}}
{25.,{q1→5.,q2→5.}}
{6,{x1→2,x2→2,x3→2}}
```

```
{6., {x1→2., x2→2., x3→2.}}
```

In the case of constrained maximization, we use here the commands **Maximize[{*f*,*cons*},{*x*,*y*,...}]** and **NMaximize[{*f*,*cons*},{*x*,*y*,...}]** as examples of constrained maximization commands. Using these commands with the maximization problems $U = f(x_1, x_2) = x_1^{1/2} + x_2^{1/2}$ subject to $2x_1 + 4x_2 = 40$ and $U = f(x_1, x_2, x_3) = x_1 + x_2 + x_3 - x_1^2 - x_2^2 - x_3^2$ subject to $x_1 + x_2 + x_3 = 6$ we solved in Sections 4.4.4 and 4.4.6, respectively, we obtain here the same results as those we obtained there:

```
Maximize[{x1^1/2 + x2^1/2, 2x1+4x2==40}, {x1, x2}]
NMaximize[{x1^1/2 + x2^1/2, 2x1+4x2==40}, {x1, x2}]
Maximize[{x1+x2+x3-x1^2-x2^2-x3^2, x1+x2+x3==6}, {x1, x2, x3}]
NMaximize[{x1+x2+x3-x1^2-x2^2-x3^2, x1+x2+x3==6}, {x1, x2, x3}]
{Sqrt[30], {x1→40/3, x2→10/3}}
{5.40553, {x1→10.1588, x2→4.92058}}
{-6, {x1→2, x2→2, x3→2}}
{-6., {x1→2., x2→2., x3→2.}}
```

Suppose that we need to minimize the function $U = f(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2$ subject to two constraints $g^1(x_1, x_2, x_3) = 2x_1 + x_2 + x_3 = 10 = c^1$ and $g^2(x_1, x_2, x_3) = x_1 + 2x_2 + x_3 = 20 = c^2$, and maximize the function $U = f(x_1, x_2, x_3) = -x_1^2 - x_2^2 - x_3^2$ subject to two constraints $g^1(x_1, x_2, x_3) = 2x_1 + x_2 + x_3 = 10 = c^1$ and $g^2(x_1, x_2, x_3) = x_1 + 2x_2 + x_3 = 20 = c^2$. These can be carried out, to obtain the same results as those in Section 4.4.8, in Mathematica as

```
Minimize[{x1^2 + x2^2 + x3^2, 2x1+x2+x3==10, x1+2x2+x3==20}, {x1, x2, x3}]
NMinimize[{x1^2 + x2^2 + x3^2, 2x1+x2+x3==10, x1+2x2+x3==20}, {x1, x2, x3}]
Maximize[{-x1^2-x2^2-x3^2, 2x1+x2+x3==10, x1+2x2+x3==20}, {x1, x2, x3}]
NMaximize[{-x1^2-x2^2-x3^2, 2x1+x2+x3==10, x1+2x2+x3==20}, {x1, x2, x3}]
{1000/11, {x1→-(10/11), x2→100/11, x3→30/11}}
{90.9091, {x1→-0.909091, x2→9.09091, x3→2.72727}}
{-(1000/11), {x1→-(10/11), x2→100/11, x3→30/11}}
{-90.9091, {x1→-0.909091, x2→9.09091, x3→2.72727}}
```

MATHEMATICA APPLICATIONS IN CHAPTER 5

Section S5.2.12

Recall that we applied few of the Mathematica commands earlier to find the solutions of constrained optimization problems with linear or nonlinear objective functions and constraints. It is important to notice that these commands can equally be applied to find the solutions of constrained optimization problems with linear objective functions and constraints. In other words, these commands are equally applicable to find the solutions, if they exist, of LP problems. Since interested reader can replicate those commands to LP problems, we do not repeat them here.

However, Mathematica has few dedicated commands to solve various types of LP problems in matrix form. These commands include, among others, **LinearProgramming[*c*,*m*,*b*]** that yields a vector ***x*** which minimizes the quantity ***c*.*x*** subject to the constraints ***m*.*x*≥*b*** and ***x*≥0**. If we want to maximize ***c*.*x*** subject to the constraints ***m*.*x*≥*b*** and ***x*≥0**, then we can minimize ***-c*.*x*** subject the constraints ***-m*.*x* ≤*b*** and ***x*≥0**. As a set of examples, solving the maximization problems in equations (5.2.5) and (5.2.15) with the command **LinearProgramming[*c*,*m*,*b*]** yields similar results (to those we obtained in Sections 5.2.4 and 5.2.6 respectively) as

```
LinearProgramming[{-2,-2},{{-2,-3},{-3,-2}},{-90,-120}]
LinearProgramming[{-10,-20},{{-4,-2},{-2,-2},{-3,-9}},
                  {-30,-20,-45}]
{36,6}
{6,3}
```

As another set of examples, solving the minimization problems in equations (5.2.11) and (5.2.18) with the command **LinearProgramming[*c*,*m*,*b*]** yields similar results (to those we obtained in Sections 5.2.5 and 5.2.6 respectively) as

```
LinearProgramming[{9,12},{{15,10},{10,15}},{90,90}]
LinearProgramming[{5,5},{{2,1},{1,4},{1,2}},{15,25,20}]
{18/5,18/5}
{10/3,25/3}
```

Section S5.3.9

Mathematica has another command to solve LP problems with ***mixed constraints***. This command is **LinearProgramming[*c*,*m*,{{b_1,s_1},{b_2,s_2},...}]**, which gives a vector ***x*** which minimizes ***c*.*x*** subject to ***x*≥0** and linear constraints specified by the matrix ***m*** and the pairs **{b_i,s_i}**. For each row m_i of ***m***, the corresponding constraint is m_i**.*x*≥b_i** if **s_i==1**, or m_i**.*x*==b_i** if **s_i==0**, or m_i**.*x*≤b_i** if **s_i==-1**. Mixed constraint problems are problems in which constraints appear with a combination of signs $\leq, =, or \geq$. Notice that we can solve LP problems with

mixed constraints using slack, surplus, and artificial variables in the graphic or tabular methods we exposed. As examples, suppose that we have to solve the following problems:

Maximize	$\Pi = 6x + 2y$	Minimize	$C = 6x + 4y$
subject to	$4x + 2y \geq 56$	subject to	$2x + 4y \leq 24$, $4x + 6y = 24$
	$10x + 4y \leq 128$, and $x, y \geq 0$.		$4x + 2y \geq 16$, and $x, y \geq 0$.

These problems can be solved in Mathematica using the command **`Linear Programming[c,m, {{b1,s1},{b2,s2},...}]`** to obtain the results as

```
LinearProgramming[{-6,-2},{{4,2},{-10,-4}},{{56,1},{-128,1}}]
LinearProgramming[{6,4},{{-2,-4},{4,6},{4,2}},{{-24,1},{24,0},{16,1}}]
{8,12}
{3,2}
```

Section S5.5.9

One of the easiest ways to solve transportation problem in Mathematica is to use the command **`LinearProgramming[c,m,b]`** introduced earlier. As examples, consider the problems in examples 1 through 3 in Section 5.5.5. These problems can be solved in Mathematica using the command **`LinearProgramming[c,m,b]`**, along with the commands **`Flatten`**, **`Table`**, **`Join`**, **`Select`**, and **`Thread`** to obtain (exactly the same respective results as in Section 5.5.5) as

```
Clear[s, d, c, v, x, i, j, solution, m, bs]
s = {100, 50}; d = {80, 70}; c = {5, 2, 3, 4};
v = Flatten[Table[x_{i,j}, {i, 2}, {j, 2}]];
m = {{1, 1, 0, 0}, {0, 0, 1, 1}, {1, 0, 1, 0}, {0, 1, 0, 1}}; bs = {Join[s, d], {0, 0, 0, 0}}^T;
solution = LinearProgramming[c, m, bs]
Select[Thread[v → solution], #[[2]] ≠ 0 &]
c.solution
{30, 70, 50, 0}
{x_{1,1} → 30, x_{1,2} → 70, x_{2,1} → 50}
440

Clear[s, d, c, v, x, i, j, solution, m, bs]
s = {200, 150, 25}; d = {150, 75, 150}; c = {5, 2, 2, 3, 6, 4, 0, 0, 0};
v = Flatten[Table[x_{i,j}, {i, 3}, {j, 3}]];
m = {{1, 1, 1, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 1, 1, 1, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 1, 1, 1}, {1, 0, 0, 1, 0, 0, 1, 0, 0}, {0, 1, 0, 0, 1, 0, 0, 1, 0}, {0, 0, 1, 0, 0, 1, 0, 0, 1}}; bs = {Join[s, d], {0, 0, 0, 0, 0, 0}}^T;
solution = LinearProgramming[c, m, bs]
Select[Thread[v → solution], #[[2]] ≠ 0 &]
c.solution
{0, 50, 150, 150, 0, 0, 0, 25, 0}
{x_{1,2} → 50, x_{1,3} → 150, x_{2,1} → 150, x_{3,2} → 25}
850
```

```
Clear[s, d, c, v, x, i, j, solution, m, bs]
s = {200, 150, 50}; d = {175, 175, 50}; c = {5, 2, 0, 3, 6, 0, 4, 2, 0};
v = Flatten[Table[x_{i,j}, {i, 3}, {j, 3}]];
```

$$m = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix};$$

```
bs = {Join[s, d], {0, 0, 0, 0, 0, 0}}^T;
solution = LinearProgramming[c, m, bs]
Select[Thread[v → solution], #[[2]] ≠ 0 &]
c.solution
```

```
{0, 150, 50, 150, 0, 0, 25, 25, 0}
{x_{1,2} → 150, x_{1,3} → 50, x_{2,1} → 150, x_{3,1} → 25, x_{3,2} → 25}
900
```

The Mathematica command **LinearProgramming[*c*,*m*,*b*]** can also be used to solve assignment problems. As examples, consider the two assignment problems we solved in Section 5.5.7. Let us solve these two problems using the said command to obtain the same results as those we obtained in Section 5.5.7 as

```
Clear[s, d, c, v, x, i, j, solution, m, bs]
s = {1, 1, 1}; d = {1, 1, 1}; c = {15, 12, 11, 13, 10, 14, 12, 16, 13};
v = Flatten[Table[x_{i,j}, {i, 3}, {j, 3}]];
```

$$m = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}; \quad bs = \{Join[s, d], \{0, 0, 0, 0, 0, 0\}\}^T;$$

```
solution = LinearProgramming[c, m, bs]
Select[Thread[v → solution], #[[2]] ≠ 0 &]
c.solution
```

```
{0, 0, 1, 0, 1, 0, 1, 0, 0}
{x_{1,3} → 1, x_{2,2} → 1, x_{3,1} → 1}
33
```

```
Clear[s, d, c, v, x, i, j, solution, m, bs]
s = {1, 1, 1, 1}; d = {1, 1, 1, 1}; c = {6, 8, 4, 0, 5, 3, 1, 0, 4, 2, 1, 0, 3, 4, 1, 0};
v = Flatten[Table[x_{i,j}, {i, 4}, {j, 4}]];
```

$$m = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix};$$

$$bs = \{Join[s, d], \{0, 0, 0, 0, 0, 0, 0, 0\}\}^T;$$

```
solution = LinearProgramming[c, m, bs]
Select[Thread[v → solution], #[[2]] ≠ 0 &]
c.solution
```

```
{0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0}
{x_{1,4} → 1, x_{2,3} → 1, x_{3,2} → 1, x_{4,1} → 1}
```

6

MATHEMATICA APPLICATIONS IN CHAPTER 6

Section S6.2.10

Notice that we can apply the Mathematica commands we introduced earlier to solve NLP problems. As examples, suppose that we need to solve those problems that we solved geometrically in Sections 6.2.5 and 6.2.6. Solving the problems in Section 6.2.5 using the command **NMaximize[{*f*,*cons*},{*x*,*y*,...}]** and the problems in Section 6.2.6 using the command **NMinimize[{*f*,*cons*},{*x*,*y*,...}]** we obtain the same solutions as those we obtained in Sections 6.2.5 and 6.2.6, respectively, as

```
NMaximize[{2 x1 + 4 x2, x1^2 + x2^2 ≤ 4, x1^2 + x2 ≤ 3}, {x1, x2}]
NMaximize[{50 x1 + 50 x2 - 5 x1^2 - 5 x2^2, x1 ≤ 10, x2 ≤ 4, x1 + 3 x2 ≤ 15}, {x1, x2}]
{8.94427, {x1 → 0.894427, x2 → 1.78885}}
{237.5, {x1 → 4.5, x2 → 3.5}}
```

```
NMinimize[{x1^2 - 10 x1 + x2^2 - 10 x2 + 50, x1 + x2 ≤ 6, x1 + 2 x2 ≥ 8}, {x1, x2}]
NMinimize[{x1 + x2, 0.5 x1 + x2 ≥ 3, x1 + 0.5 x2 ≥ 3, x1^2 + x2^2 ≥ 8}, {x1, x2}]
{8., {x1 → 3., x2 → 3.}}
{4., {x1 → 2., x2 → 2.}}
```

MATHEMATICA APPLICATIONS IN CHAPTER 7

These applications are in the process of generation and will be posted here soon.

MATHEMATICA APPLICATIONS IN CHAPTER 8

Section S8.3.17

As in the case of differentiation, Mathematica has a large number of built-in-commands to perform integration. One simple command is **`Integrate[f,x]`**, which gives the indefinite integral of the function ***f*** with respect to its argument ***x***. Notice that Mathematica omits the constant C, found in our examples so far, from the output. We can also use the notation $\int expr \; dvar$, instead of the command **`Integrate`**, from the **Palettes → Basic Math Assistant → Typesetting**.

As an example, suppose that we need to integrate the function in equation (8.2.1): $(1/3).t^{-2/3}$. This can carried out to obtain the same result as that we obtained in Section 8.3.2 except with the constant, as

```
Integrate[(1/3)×t^-2/3,t]
```

$$t^{1/3}$$

$$\int (1/3) \times t^{-2/3} \, dt$$

$$t^{1/3}$$

We can also carry out partial integration using the command **`Integrate`**. As demonstrations, consider the functions we used in the examples in Section 8.3.13: $\int F_{x_1} dx_1 = \int f(x_1, x_2) dx_1 = \int (2x_1 x_2 + x_2) dx_1$; $\int F_{x_1} dx_2 = \int f(x_1, x_2) dx_2 = \int (2x_1 x_2 + x_2) dx_2$; and $\int F_{x_1} dx_1 = \int f(x_1, x_2, x_3) dx_1 = \int (x_1 x_2 + x_2 x_3 + x_1 x_3) dx_1$. These can be carried out, to obtain the same results (excepts for the constants) as those we obtained in Section 8.3.13, as

```
Integrate[2x1 x2+x2,x1]
```

$$x_1 x_2 + x_1^2 x_2$$

```
Integrate[2x1 x2+x2,x2]
```

$$\frac{x_2^2}{2} + x_1 x_2^2$$

```
Integrate[x1 x2+x2 x3+x1 x3,x1]
```

$$\frac{1}{2} x_1^2 x_2 + \frac{1}{2} x_1^2 x_3 + x_1 x_2 x_3$$

Multiple integrals can be found by using the command **`Integrate`** successively. As a demonstration, consider the function in the first example in Section 8.3.14: $F_{x_1} = f(x_1, x_2) = 2x_1 x_2 + x_2$. Suppose that we want to find $\int\int \left(F_{x_1}\right) dx_1 \, dx_2$. This can found in Mathematica, to obtain the same results as those we obtained there, as

```
firstintegral=Integrate[2x1 x2+x2,x1]
secondintegral=Integrate[firstintegral,x2]
```

$$x_1 x_2 + x_1^2 x_2$$

$$\frac{1}{2} x_1 x_2^2 + \frac{1}{2} x_1^2 x_2^2$$

Section S8.4.14

In order to evaluate a definite integral in Mathematica we can use the command **Integrate** by specifying the range of values that the variable of the integrand can take. The command **Integrate[*f*,{*x*,*x*$_{min}$,*x*$_{max}$}]** integrates the function ***f*** when the argument of the function ***x*** varies from ***x*$_{min}$** to ***x*$_{max}$**. As a demonstration, consider the function we integrated in Section 8.4.2: $y = f(x) = 2x$. This function can be integrated when *x* varies from 0 to 1 using the command **Integrate[*f*,{*x*,*x*$_{min}$,*x*$_{max}$}]** to obtain the same result as that we obtained there:

```
Integrate[2x,{x,0,1}]
1
```

$$\int_0^1 2x\,dx$$
$$1$$

Definite partial integrals can be obtained using the same command **Integrate[*f*,{*x*,*x*$_{min}$,*x*$_{max}$},{*y*,*y*$_{min}$,*y*$_{max}$}]**. This command evaluates the integral when the integrand is a multivariate function of arguments ***x*** and ***y***, and when ***x*** and ***y*** vary from ***x*$_{min}$** to ***x*$_{max}$** and from ***y*$_{min}$** to ***y*$_{max}$**, respectively. As demonstrations, consider the integrands we evaluated in Section 8.4.10: $\int_0^1 (2x_1x_2 + x_2)\,dx_1$; $\int_0^1 (2x_1x_2 + x_2)dx_2$ and $\int_0^2 (x_1 + x_2 + x_3 + x_1x_2 + x_1x_3)\,dx_1$. Evaluating these integrals we obtain, the same results as those we obtained there, as

```
Integrate[2×x1×x2+x2,{x1,0,1}]
2x2
```

```
Integrate[2×x1×x2+x2,{x2,0,1}]
1/2+x1
```

$$\int_0^2 (x_1 + x_2 + x_3 + x_1x_2 + x_1x_3)\,dx_1$$
$$2+4x_2+4x_3$$

We shall now use the command **Integrate** to evaluate few definite multiple integrals. As demonstrations, we shall use the same integrands as those we used in Section 8.4.11. Then using the command **Integrate**, we have

```
Integrate[Integrate[2×x×y+y,{x,0,1}],{y,0,1}]
1
```

```
Integrate[Integrate[Integrate[x+y+z+x×y+x×z,{x,0,1}],{y,0,1}],{z,0,1}]
2
```