An Introduction to *Mathematica*

Mathematica is a powerful tool for performing both symbolic and numerical calculations. It is available on all major computing platforms, including Macintosh, Windows and UNIX systems. The *notebook* interface on all systems is very similar. A user normally enters *commands* in *cells* in a notebook window and *executes* these commands to see the results appear in the same notebook. Starting with version 3.0 of *Mathematica,* the notebook interface has become quite sophisticated. In addition to *Mathematica* input and output, the notebooks can now contain graphics and typeset text, including all mathematics symbols and complicated equations. The *style* of a cell determines whether it contains live input and output or just text. Similar to a word processor, several other specialized cell styles are available to create section headings, sub-headings, etc. A large number of pull-down menus are available to manipulate information in different notebook cells.

Mathematica is a complex system and requires considerable practice to become proficient in its use. However getting started with it and using it as an advanced calculator requires understanding of only a few basic concepts and knowledge of few commands. This brief introduction is intended as a quick start guide for the users of this text. Serious users should consult many excellent books on *Mathematica* listed at the end of this chapter. *Mathematica* has excellent on-line help system as well. This help system has usage instructions and examples of all *Mathematica* commands. The entire *Mathematica* book is available on-line as well and is accessible through the help browser.

1. Basic manipulations in *Mathematica*

Mathematica performs a wide variety of numeric, graphic, and symbolic operations. Expressions are entered using a syntax similar to that of common programming languages such as BASIC, FORTRAN, or Pascal. Some of the operators available are: +[add], -[subtract], *[multiply], /[divide], ^[exponentiate], and Sqrt[ ] [square root]. The multiplication operator is optional between two variables.

A command can be split into any number of lines. The **Enter key** (or Option-Return) on the keyboard is used to actually execute the command line containing the insertion point (blinking vertical line). Note on some keyboards the usual Return key (the one used to move to the next line) is labelled as Enter. As far as *Mathematica* is concerned this is still a Return key. A semicolon [;] is used at the end of a line of *Mathematica* input if no output is desired. This feature should be used freely to suppress printout of long intermediate expressions that no one cares to look at anyway.

```
1 + 5 + 32 * 43 / 34^3
```

```
29650
─────
 4913
```

As seen from this example, Mathematica normally does rational arithmetic. To get numerical answer the function 'N' can be used.

```
N[1 + 5 + 32 * 43 / 34^3]
```

```
6.03501
```

The N function (or any other function requiring a single argument) can also be applied using the // operator as follows.

```
1 + 5 + 32 * 43 / 34^3 // N
```

```
6.03501
```

Caution on *Mathematica* Input Syntax

It is important to remember that Mathematica commands are case-sensitive. The built-in functions all start with an uppercase letter. Also you must be very careful in using brackets. Different brackets have different meaning. Arguments to functions must be enclosed in square brackets ([]). Curly braces ({}) are used to define lists and matrices. Parentheses '()' do not have any built-in function and therefore can be used to group terms in expressions, if desired.

To indicate multiplication operation one can use either an asterisk (*) or simply one or more blank space between the two variables. When a number precedes a variable even a blank is not necessary. Sometimes this convention can lead to unexpected problems as demonstrated in the following examples.

Suppose one wants product of variables *a* and *b*, a space is necessary between the two variables. Without space *Mathematica* simply defines a new symbol "ab" and does not give the intended product.

```
a = 10; b = 3;
2 ab
```

```
2 ab
```

No space is needed between 2 and *a* but there must be a space or * between *a* and *b* to get the product $2\,a\,b$.

```
2 a b
```

```
60
```

If we enter 2a^3b without any spaces it is interpreted as $2\,a^3\,b$ and not as $2\,a^{3\,b}$.

```
2 a^3 b
```

```
6000
```

In situations like these it is much better to use parentheses to make the association clear. For example there is no ambiguity in the following form.

```
(2 a)^(3 b)
```

```
512000000000
```

Using built-in functions

Mathematica has a huge library of built-in functions. Few common ones are introduced here. The functions 'Simplify' and 'Expand' are used to get expressions in simpler forms. Note the use of Clear to remove any previously defined values for *a*.

```
Clear[a];
Simplify[a^2 + 3 a + (a + b) a]
```

```
a (3 + 2 a + b)
```

```
Expand[a^2 + 3 a + (a + b) a]
```

```
3 a + 2 a^2 + a b
```

Just like a programming language we can assign names to expressions so that they can be referred to later. For example we define an expression e1 as follows.

```
e1 = x + Sin[x] Cos[x^3] / Exp[x]
```

```
x + E^-x Cos[x^3] Sin[x]
```

With this definition the expression on the right-hand side of the equal sign can be manipulated easily. For example it can be differentiated with respect to *x* as follows. For later reference the derivative expression is denoted by de1.

```
de1 = D[e1, x]
```

$1 + E^{-x} \text{Cos}[x]\,\text{Cos}[x^3] - E^{-x} \text{Cos}[x^3]\,\text{Sin}[x] - 3 E^{-x} x^2 \text{Sin}[x]\,\text{Sin}[x^3]$

We can integrate de1 as follows.

```
Integrate[de1, x]
```

$\frac{1}{2} (2 x + E^{-x} \text{Sin}[x - x^3] + E^{-x} \text{Sin}[x + x^3])$

This does not look like e1. But the following expansion, using the trigonometric identities, shows that it is indeed equal to e1. Note the use of '%' symbol to refer to the result of previous evaluation as a short cut.

```
Expand[%, Trig -> True]
```

$\frac{1}{2} (2 x + 2 E^{-x} \text{Cos}[x^3]\,\text{Sin}[x])$

Note that % refers to result of the previous evaluation and not the result appearing in the cell just above the current cell. In a notebook with many input cells one can execute any cell simply by highlighting it (or placing the insertion point anywhere in the cell). The % will always refer to the most recent evaluation. In fact the evaluation does not have to be in the current notebook and can be in any other notebook open at the same time. Therefore the use of % as a shortcut is strongly discouraged. It is better to assign a variable to refer to an evaluation in other places.

Mathematica can also compute definite integrals involving complicated expressions using numerical integration.

```
NIntegrate[e1, {x, 0, 1}]
```

```
0.724231
```

Substitution rule

The symbol '/.' (slash-period) is a useful operator and is used to substitute different variables or numerical values into existing expressions. For example expression e1 can be evaluated at $x = 1.5$ as follows.

```
e1 /. x->1.5
```

```
1.28346
```

The arrow symbol is a combination of hyphen(-) and greater than sign (>) and is known as 'rule' in *Mathematica*. Of course more complicated substitutions are possible. The following example illustrates defining a new expression called e2 by substituting $x = a + \text{Sin}[b]$ in expression e1.

```
e2 = e1 /. {x->a + Sin[b]}
```

$$10 + \text{Sin}[3] + E^{-10-\text{Sin}[3]}\,\text{Cos}[(10 + \text{Sin}[3])^3]\,\text{Sin}[10 + \text{Sin}[3]]$$

Substitute $a = 1$ and $b = 2$ in expression e2.

```
e2 /. {a->1, b->2}
```

$$1 + \text{Sin}[2] + E^{-10-\text{Sin}[2]}\,\text{Cos}[(1 + \text{Sin}[2])^2]\,\text{Sin}[1 + \text{Sin}[2]]$$

To get numerical result after substituting $a = 1$ and $b = 2$ in expression e2 we can use N function

```
N[e2 /. {a->1, b->2}]
```

```
1.90928
```

Note that the same result can also be obtained as follows.

```
a = 1;
b = 2;
N[e2, 10]
```

```
10.14109419
```

However the substitution form is much more desirable. This second form defines the symbols *a* and *b* to have the indicated numerical values. *Mathematica* will automatically substitute these values into any subsequent evaluation that involves *a* and *b*.

```
a + b x
```

```
1 + 2 x
```

The only way to later get symbolic expressions involving *a* and *b* would be to explicitly clear these variables as follows.

```
Clear[a, b];
a + b x
```

```
a + b x
```

2. Lists and Matrices

List are quantities enclosed in curly brackets ({}). They are one of the basic data structures used by *Mathematica*. Many *Mathematica* functions expect arguments in the form of lists and return results as lists. When working with these functions, it is important to pay close attention to the nesting of braces. Two expressions, that otherwise are identical, except that one is enclosed in single set of braces and the other in double braces means two entirely different things to *Mathematica*. Beginners often don't pay close attention to braces and get frustrated when *Mathematica* does not do what they think it should be doing.

Lists

Here is an example of a one dimensional list with 6 elements.

```
s = {1, 3, Sin[bb], 2 * x, 5, (3 * y) / x^3}
```

$\left\{1,\ 3,\ \mathrm{Sin}[bb],\ 2\,x,\ 5,\ \frac{3\,y}{x^3}\right\}$

The Length function gives the number of elements in a list.

```
Length[s]
```

```
6
```

We can extract elements of a list by enclosing the element indices in double square brackets. For example to get elements 3, 4, and 6 from s we use

```
ss = s[[{3, 4, 6}]]
```

$\left\{\mathrm{Sin}[bb],\ 2\,x,\ \frac{3\,y}{x^3}\right\}$

Most standard algebraic operations can be applied to all elements of a list. For example to differentiate all elements of a list by x, we simply need to use D function on the list.

```
D[ss, x]
```

$\left\{0,\ 2,\ -\frac{9\,y}{x^4}\right\}$

Multidimensional lists can be defined in a similar way. Here is an example of a two dimensional list with first *row* having one element, second row three elements and the third row as the *s* list defined above.

```
a = {{1}, {2, 3, 4}, s}
```

$$\left\{\{1\},\ \{2,\ 3,\ 4\},\ \left\{1,\ 3,\ \text{Sin[bb]},\ 2\,x,\ 5,\ \frac{3\,y}{x^3}\right\}\right\}$$

```
Length[a]
```

```
3
```

Elements of a lists can be extracted by specifying row number and the element number within each row in pairs of square brackets. For example, to get the second row of list a we use the following structure.

```
a[[2]]
```

```
{2, 3, 4}
```

To get the sixth element of the third row of list *a* we use the following structure.

```
a[[3, 6]]
```

$$\frac{3\,y}{x^3}$$

Obviously *Mathematica* will generate an error message if one tries to get an element that is not defined in a list. For example trying to get the fourth element from the second row will produce the following result.

```
a[[2, 4]]
```

```
Part::partw : Part 4 of {2, 3, 4} does not exist.
```

$$\left\{\{1\},\ \{2,\ 3,\ 4\},\ \left\{1,\ 3,\ \text{Sin[bb]},\ 2\,x,\ 5,\ \frac{3\,y}{x^3}\right\}\right\}[\![2,\ 4]\!]$$

Sometimes it is necessary to change the structure of a list. **`Flatten`** and **`Partition`** functions are simple ways to achieve this. Flatten removes all internal braces and converts any multidimensional list into a single *flat* list, keeping the order same as that in the original list. For example we can define a new one dimensional list *b* by flattening list *a* as follows.

```
b = Flatten[a]
```

$\{1, 2, 3, 4, 1, 3, \text{Sin[bb]}, 2x, 5, \frac{3y}{x^3}\}$

Using Partition we can create partitions (row structure) in a given list. The second argument of Partition specifies number of elements in each row. Extra elements that do not define a complete row are ignored.

```
Partition[b, 3]

{{1, 2, 3}, {4, 1, 3}, {Sin[bb], 2 x, 5}}
```

```
Partition[b, 4]

{{1, 2, 3, 4}, {1, 3, Sin[bb], 2 x}}
```

Matrices

Matrices are special two dimensional lists in which each row has exactly the same number of elements. Here we define a 3x4 matrix m.

```
m = {{1,2,3,4},{5,6,7,8},{9,10,11,12}}

{{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}}
```

To see the result displayed in a conventional matrix form we use MatrixForm.

```
MatrixForm[m]
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$$

TableForm is another way of displaying multi-dimensional lists.

```
TableForm[m]

1       2       3       4
5       6       7       8
9       10      11      12
```

The following two dimensional list is not a valid matrix because number of elements in each row is not the same. However there is no error message produced because it is still a valid list in *Mathematica*.

```
a = {{1}, {2, 3}, {4, 5, 6}}
```

```
{{1}, {2, 3}, {4, 5, 6}}
```

```
MatrixForm[a]
```

$$\begin{pmatrix} \{1\} \\ \{2, 3\} \\ \{4, 5, 6\} \end{pmatrix}$$

Once a matrix is defined, standard matrix operations can be performed on it. For example we can define a new matrix mt as Transpose of m.

```
mt = Transpose[m]; MatrixForm[mt]
```

$$\begin{pmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{pmatrix}$$

Matrices can be multiplied by using . (period) between their names.

```
MatrixForm[mt.m]
```

$$\begin{pmatrix} 107 & 122 & 137 & 152 \\ 122 & 140 & 158 & 176 \\ 137 & 158 & 179 & 200 \\ 152 & 176 & 200 & 224 \end{pmatrix}$$

Using * or blank will produce an error or simply element by element multiplication.

```
mt * m
```

```
Thread::tdlen : Objects of unequal length
   in {{1, 5, 9}, {2, 6, 10}, {3, 7, 11}, {4, 8, 12}}
    {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}} cannot be combined.
```

```
{{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}}
 {{1, 5, 9}, {2, 6, 10}, {3, 7, 11}, {4, 8, 12}}
```

Here we get an element by element product and not the matrix product.

```
mt * mt

{{1, 25, 81}, {4, 36, 100}, {9, 49, 121}, {16, 64, 144}}
```

When using MatrixForm, it is very important to note that the MatrixForm is used for display purposes only. If a matrix is defined with the MatrixForm in it, that matrix definition cannot be used in any subsequent calculation. For example, consider the following definition of matrix 'm' used previously.

```
m = MatrixForm[{{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}}]
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$$

The output looks exactly like it did in the earlier case. However we cannot perform any operations on matrix m in this form. For example using Transpose function on m simply returns the initial matrix m wrapped in Transpose.

```
Transpose[m]
```

$$\text{Transpose}\left[\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}\right]$$

The solution obviously is not to use the MatrixForm in the definition of matrices. After the definition we can use the MatrixForm to get a nice looking display.

```
m = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}}; MatrixForm[m]
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$$

Elements of matrices can be extracted using the double square brackets. For example, the second row of m can be extracted as follows.

```
m2 = m[[2]]

{5, 6, 7, 8}
```

The element in the 2nd row and the 4th column is

```
m[[2, 4]]
```

```
8
```

Extracting a column is little more difficult. The simplest way is to transpose the matrix and then take its desired row (which obviously corresponds to the original column). For example the second column of matrix m is extracted as follows.

```
c2 = Transpose[m][[2]]
```

```
{2, 6, 10}
```

Matrices can also be defined in terms of symbolic expressions. For example here is a 2x2 matrix called m.

```
m = {{y, 2*x}, {(3*y)/x^3, 4*y}}; MatrixForm[m]
```

$$\begin{pmatrix} y & 2\,x \\ \frac{3\,y}{x^3} & 4\,y \end{pmatrix}$$

Determinant of m

```
Det[m]
```

$$-\frac{6\,y}{x^2} + 4\,y^2$$

Inverse of m

```
mi = Inverse[m]; MatrixForm[mi]
```

$$\begin{pmatrix} \frac{4\,y}{-\frac{6\,y}{x^2}+4\,y^2} & -\frac{2\,x}{-\frac{6\,y}{x^2}+4\,y^2} \\ -\frac{3\,y}{x^3\,\left(-\frac{6\,y}{x^2}+4\,y^2\right)} & \frac{y}{-\frac{6\,y}{x^2}+4\,y^2} \end{pmatrix}$$

Check the inverse

```
MatrixForm[Simplify[mi.m]]
```

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Operations, such as differentiation and integration, are performed on each element of a matrix.

```
D[m,y]//MatrixForm
```

$$\begin{pmatrix} 1 & 0 \\ \frac{3}{x^3} & 4 \end{pmatrix}$$

```
Integrate[m,x]//MatrixForm
```

$$\begin{pmatrix} x\,y & x^2 \\ -\frac{3\,y}{2\,x^2} & 4\,x\,y \end{pmatrix}$$

Generating lists with Table function

The Table function is a handy tool to generate lists or matrices with a systematic pattern. For example to generate a 3x5 matrix with all entries as zero we use the following form.

```
m = Table[0, {3}, {5}]; MatrixForm[m]
```

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

The first argument to Table function can be any expression. Here is a 3x4 matrix whose entries are equal to the sum of row and column indices divided by 3^(column index).

```
z = Table[(i + j) / 3^j, {i, 1, 3}, {j, 1, 4}]; MatrixForm[z]
```

$$\begin{pmatrix} \frac{2}{3} & \frac{1}{3} & \frac{4}{27} & \frac{5}{81} \\ 1 & \frac{4}{9} & \frac{5}{27} & \frac{2}{27} \\ \frac{4}{3} & \frac{5}{9} & \frac{2}{9} & \frac{7}{81} \end{pmatrix}$$

Using the If command we can construct complicated matrices. For example here is a 4x4 upper triangular matrix.

```
z = Table[If[i > j, 0, (i + j) / 3^j], {i, 1, 4}, {j, 1, 4}]; MatrixForm[z]
```

$$\begin{pmatrix} \frac{2}{3} & \frac{1}{3} & \frac{4}{27} & \frac{5}{81} \\ 0 & \frac{4}{9} & \frac{5}{27} & \frac{2}{27} \\ 0 & 0 & \frac{2}{9} & \frac{7}{81} \\ 0 & 0 & 0 & \frac{8}{81} \end{pmatrix}$$

Note the syntax of If command is as follows.

```
If[test, Statements to be executed if test is True, Statements to be executed if test is False]
```

Caution when dealing with row and column vectors

Since *Mathematica* treats matrices as essentially lists, it does not distinguish between a column or a row vector. The actual form is determined from syntax in which it is used. For example define two vectors a and b as follows.

```
a = {1, 2, 3}; b = {4, 5, 6};
```

Matrix inner product (Transpose[a] . b) is evaluated simply as a . b, resulting in a scalar. Explicitly evaluating Transpose[a]. b will produce an error.

```
a . b
```

```
32
```

```
Transpose[a] . b
```

```
Transpose::nmtx : The first two levels of the
   one-dimensional list {1, 2, 3} cannot be transposed.

Transpose[{1, 2, 3}].{4, 5, 6}
```

If we want to treat *a* as a column vector (3 x 1) and *b* as a row vector (1 x 3) to get a 3x3 matrix from the product, we need to use Outer function of *Mathematica* as follows.

```
ab = Outer[Times, a, b]; MatrixForm[ab]
```

$$\begin{pmatrix} 4 & 5 & 6 \\ 8 & 10 & 12 \\ 12 & 15 & 18 \end{pmatrix}$$

Another way to explicitly define a 3x1 column vector is to enter it as a two dimensional list. Here each entry defines a row of a matrix with one column.

```
a = {{1}, {2}, {3}}; MatrixForm[a]
```

$$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

Now Transpose[a] . b makes sense. However the result is 1x1 matrix and not a scalar.

```
Transpose[a] . b

{32}
```

Obviously now a . b will produce an error because the dimensions do not match.

```
a . b

Dot::dotsh :
 Tensors {{1}, {2}, {3}} and {4, 5, 6} have incompatible shapes.

{{1}, {2}, {3}}.{4, 5, 6}
```

To get a 3x3 matrix we need to define b as a two dimensional matrix with only one row as follows.

```
b = {{4, 5, 6}}

{{4, 5, 6}}
```

Now a (3x1) . b (1x3) can be evaluated directly, as one would expect.

```
MatrixForm[a . b]
```

$$\begin{pmatrix} 4 & 5 & 6 \\ 8 & 10 & 12 \\ 12 & 15 & 18 \end{pmatrix}$$

3. Solving equations

An equation in *Mathematica* is defined as an expression with two parts separated by two equal signs (==, with out any spaces between the two signs). Thus the following is an equation.

```
x^3 - 3 == 0
```

$-3 + x^3 == 0$

Note you get the following strange looking error message if you try to define an equation with a single equal sign. (The error message makes sense if you know more details of how *Mathematica* handles expressions internally.)

```
x^3 - 3 = 0
```

```
Set::write : Tag Plus in -3 + x^3 is Protected.

0
```

Equations can also be assigned to variables for later reference. For example we can call the above equation as eq so that we can refer to it in other computations.

```
eq = x^3 - 3 == 0
```

```
-3 + x^3 == 0
```

A single equation, or a systems of equations, can be solved using the Solve command.

```
sol = Solve[eq, x]
```

```
{{x → -(-3)^(1/3)}, {x → 3^(1/3)}, {x → (-1)^(2/3) 3^(1/3)}}
```

Note the solution is returned in the form of a two dimensional list of substitution 'rules'. Because of this form we can substitute any solution into another expression if desired. For example we can substitute the second solution back into the equation to verify that the solution is correct, as follows.

```
eq /. sol[[2]]
```

```
True
```

The 'Solve' command tries to find all possible solutions using symbolic manipulations. If a numerical solution is desired the 'NSolve' command is usually much faster.

```
NSolve[x^3 - 3 == 0, x]
```

```
{{x → -0.721125 + 1.24902 I}, {x → -0.721125 - 1.24902 I}, {x → 1.44225}}
```

Both Solve and NSolve commands can be used for systems of linear or nonlinear equations.

```
eqn1 = x + 2y + 3z == 41;
eqn2 = 5x + 5y + 4z == 20;
eqn3 = 3y + 4z == 125;
sol = Solve[{eqn1, eqn2, eqn3}, {x, y, z}]
```

$$\left\{\left\{x \to -\frac{527}{13},\ y \to \frac{635}{13},\ z \to -\frac{70}{13}\right\}\right\}$$

Notice that even when there is only one solution, the result is a two dimensional list. If we want to evaluate a given function at this solution point, we still need to extract the first element of this list. For example

```
a = N[Sin[x y] Cos[y^z] /. sol[[1]]]

-0.810238
```

The following substitution produces the same result but in the form of a list whose structure must be kept in mind if it is to be used in other expressions.

```
b = N[Sin[x y] Cos[y^z] /. sol]

{-0.810238}
```

For example, since *a* is a scalar as defined by the first form, it can be used in any matrix operation. However *b* can only used with appropriate sized matrices.

```
a {a1, a2, a3}

{-0.810238 a1, -0.810238 a2, -0.810238 a3}


b {a1, a2, a3}

Thread::tdlen : Objects of unequal
   length in {-0.810238} {a1, a2, a3} cannot be combined.

{-0.810238} {a1, a2, a3}
```

A linear system of equations, written in matrix form as K d = R, can be solved very efficiently using the LinearSolve[K, R] command as follows.

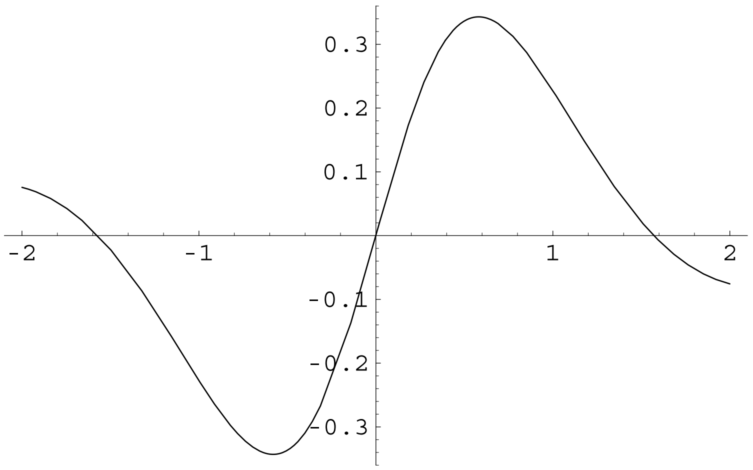
```
K = {{2, 4, 6}, {7, 9, 2}, {1, 2, 13}};
R = {1, 2, 3};
LinearSolve[K, R]
```

$$\left\{\frac{21}{20}, -\frac{13}{20}, \frac{1}{4}\right\}$$

4. Plotting in *Mathematica*

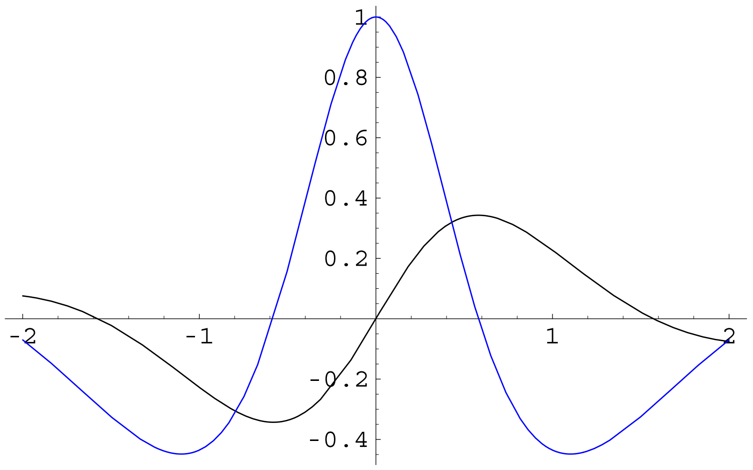
Mathematica supports both 2-dimensional and 3-dimensional graphics. The Plot command provides support for 2-dimensional graphs of one or more functions specified as expressions, or functions. The following shows plot of an expression in the interval from -2 to 2.

```
p1 = Plot[Sin[x] Cos[x]/(1+x^2),{x,-2,2}];
```



Multiple plots can be shown on the same graph using the 'Show' command. The following example plots expression e1 and its first derivative. Each plot command automatically displays resulting graph. For multiple plots, usually the final plot showing all graphs superimposed, is of interest. Using `DisplayFunction` as `Identity`, suppresses display of intermediate plots.

```
de1 = D[Sin[x] Cos[x]/(1+x^2),x];
p2 = Plot[de1,{x,-2,2}, PlotStyle->{RGBColor[0,0,1]},
    DisplayFunction -> Identity];
Show[{p1,p2}];
```



The RGBColor function returns a combination color by mixing specified amounts of Red, Green and Blue colors. In the example the Blue component is set to 1 and others to 0 to get a blue color. Use Mathematica on-line help to explore many other plot options.

Note before generating the plot for the derivative we defined a new expression and then used it in the plot function. If we had actually used the derivative operation inside the argument of the Plot function, we would get a series of strange error messages as follows.

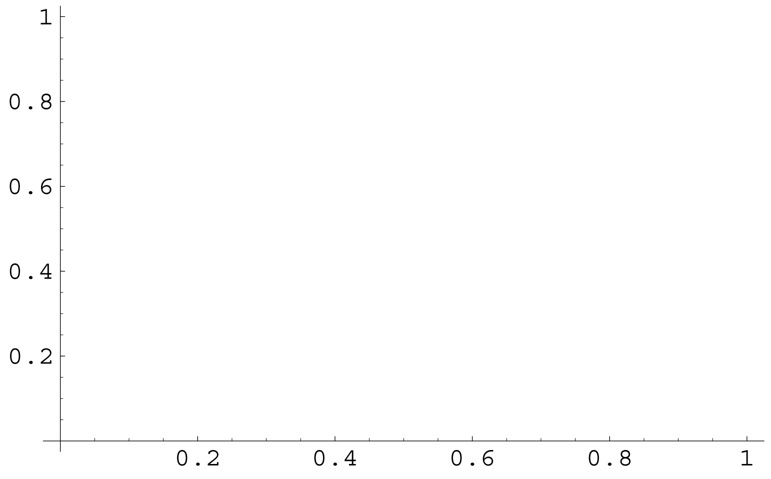
```
Plot[D[Sin[x] Cos[x]/(1+x^2),x],{x,-2,2}];

General::ivar : -2. is not a valid variable.

General::ivar : -2. is not a valid variable.

General::ivar : -2. is not a valid variable.

General::stop : Further output of
   General::ivar will be suppressed during this calculation.
```

Plot::plnr :
$\partial_x \frac{\text{Sin[x] Cos[x]}}{1 + x^2}$ is not a machine-size real number at x = -2..

Plot::plnr :
$\partial_x \frac{\text{Sin[x] Cos[x]}}{1 + x^2}$ is not a machine-size real number at x = -1.83773.

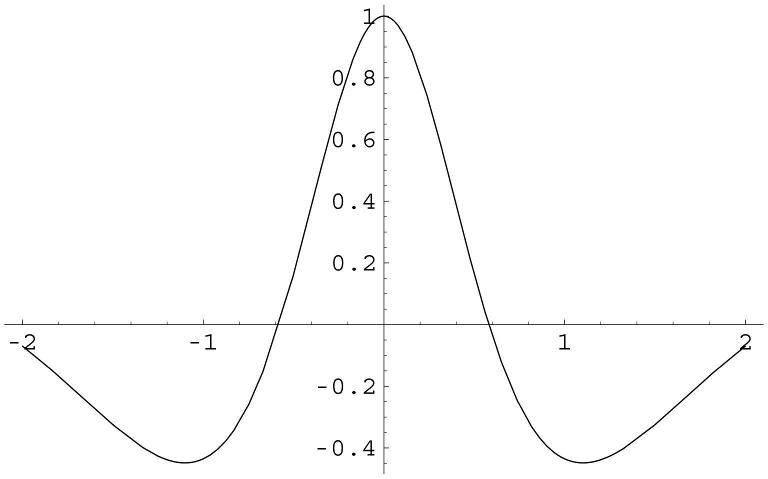
Plot::plnr :
$\partial_x \frac{\text{Sin[x] Cos[x]}}{1 + x^2}$ is not a machine-size real number at x = -1.66076.

```
General::stop : Further output of
   Plot::plnr will be suppressed during this calculation.
```



The reason for the error is that Plot and several other built-in functions (for valid reasons but difficult to explain here) do not evaluate their arguments. We can force them to evaluate the arguments by enclosing the computation inside Evaluate as follows.

```
Plot[Evaluate[D[Sin[x]  Cos[x]/(1+x^2),x]],{x,-2,2}];
```

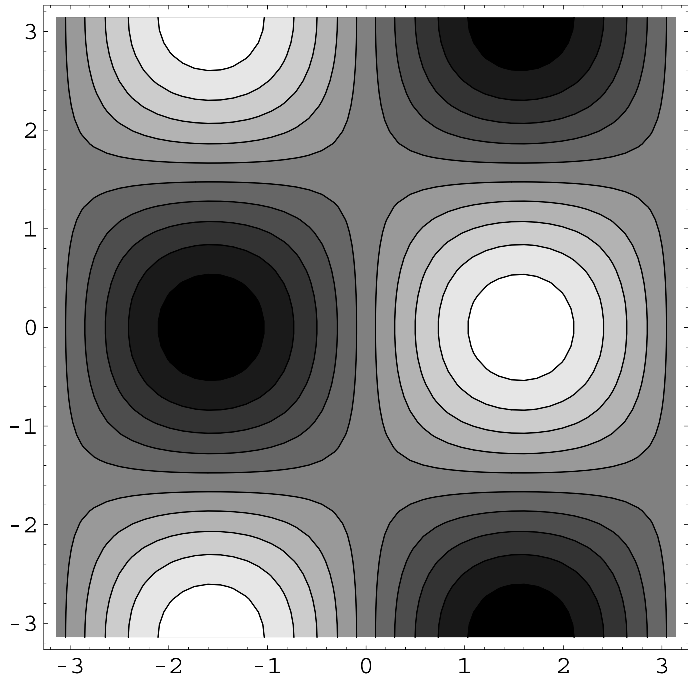


Functions of two variables can be shown either as contour plots or as three dimensional surface plots.

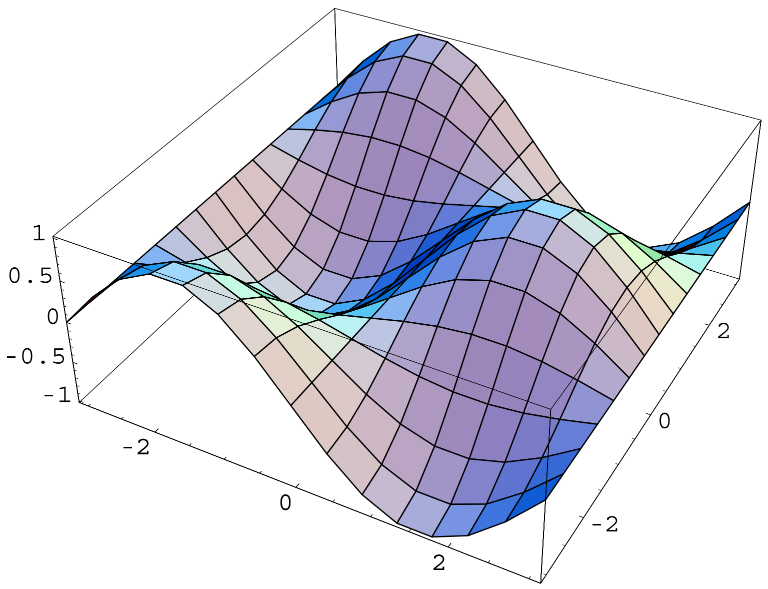
```
f = Sin[x] Cos[y]
```

```
Cos[y] Sin[x]
```

```
ContourPlot[f, {x, -π, π}, {y, -π, π}, PlotPoints -> 50];
```



```
Plot3D[f, {x, -π, π}, {y, -π, π}];
```



5. Differential Equations

A variety of ordinary differential equations can be solved using DSolve command. If successful this function returns an analytical expression for solution of the differential equation.The function also can solve a limited number of boundary value problems. The function an solve a single equation or a system of ordinary differential equations. Initial and boundary conditions can be specified as part of these equations.

```
?DSolve
```

```
DSolve[eqn, y, x] solves a differential equation for the function
  y, with independent variable x. DSolve[{eqn1, eqn2, ...  }, {y1,
  y2, ...  }, x] solves a list of differential equations. DSolve[
  eqn, y, {x1, x2, ... }] solves a partial differential equation.
```

As an example we evaluate solution of the following boundary value problem.

$$\frac{d^2 u}{\mathrm{dx}^2} + u + x = 0 \quad 1 < x < 3$$

$$u(1) = 2 \qquad \frac{\mathrm{du}}{\mathrm{dx}}(3) = 0$$

```
sol = DSolve[{D[u[x], {x, 2}] + u[x] + x == 0,
   u[1] == 2, (D[u[x], x] /. x -> 3) == 0}, u[x], x]
```

```
{{u[x] → -x + Cos[x] Sec[2] (3 Cos[3] - Sin[1]) +
     Sec[2] (Cos[1] + 3 Sin[3]) Sin[x]}}
```

We can verify the solution by substituting the solution into the differential equation as follows.

```
ux = u[x] /. sol[[1]]
```

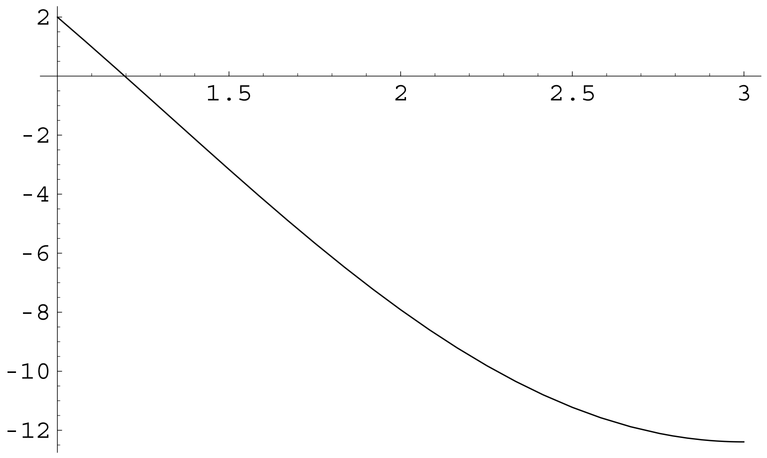
```
-x + Cos[x] Sec[2] (3 Cos[3] - Sin[1]) + Sec[2] (Cos[1] + 3 Sin[3]) Sin[x]
```

```
D[ux, {x, 2}] + ux + x
```

```
0
```

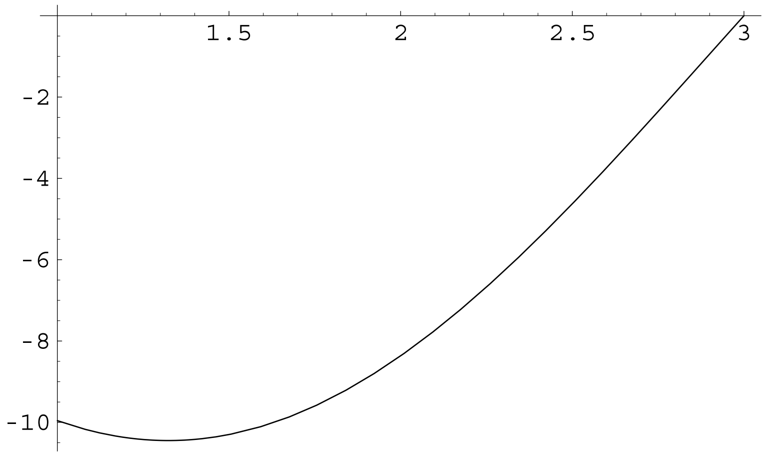
The solution can also be plotted as follows.

```
Plot[ux, {x, 1, 3}];
```



The first derivative of the solution can also be plotted as follows.

```
Plot[Evaluate[D[ux, x]], {x, 1, 3}];
```



For more complicated differential equation, an analytical solution may not be possible. In these situation NDSolve can be used to give an approximate numerical solution. This function returns solution in the form an cubic interpolation function. This solution can be plotted using Plot, differentiated using D, and integrated using Integrate, just like a ordinary function.

```
? NDSolve
```

```
NDSolve[eqns, y, {x, xmin, xmax}] finds a numerical solution to
  the ordinary differential equations eqns for the function
  y with the independent variable x in the range xmin to
  xmax. NDSolve[eqns, y, {x, xmin, xmax}, {t, tmin, tmax}]
  finds a numerical solution to the partial differential
  equations eqns. NDSolve[eqns, {y1, y2, ... }, {x, xmin,
  xmax}] finds numerical solutions for the functions yi.
```

As an example we evaluate solution of the following boundary value problem.

$$x \frac{d^2 u}{\mathrm{dx}^2} + u + x = 0 \quad u(1) = 2 \quad \frac{\mathrm{du}}{\mathrm{dx}}(3) = 0$$

As seen in the following, the DSolve cannot find the solution to this problem.

```
DSolve[{x D[u[x], {x, 2}] + u[x] + x == 0,
  u[1] == 2, (D[u[x], x] /. x -> 3) == 0}, u[x], x]
```

```
DSolve[{x + u[x] + x u''[x] == 0, u[1] == 2, u'[3] == 0}, u[x], x]
```

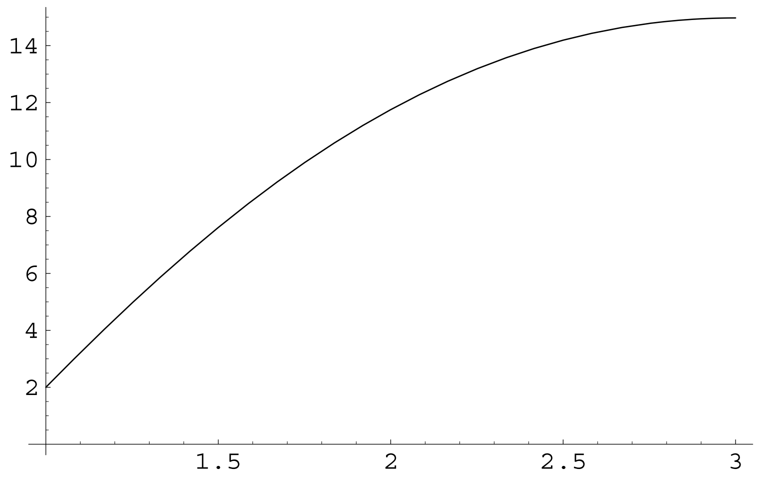
Using NDSolve we can obtain a numerical solution as follows.

```
sol = NDSolve[{x D[u[x], {x, 2}] + u[x] + x == 0,
   u[1] == 2, (D[u[x], x] /. x -> 3) == 0}, u[x], {x, 1, 3}]
```

```
{{u[x] → InterpolatingFunction[{{1., 3.}}, <>][x]}}
```

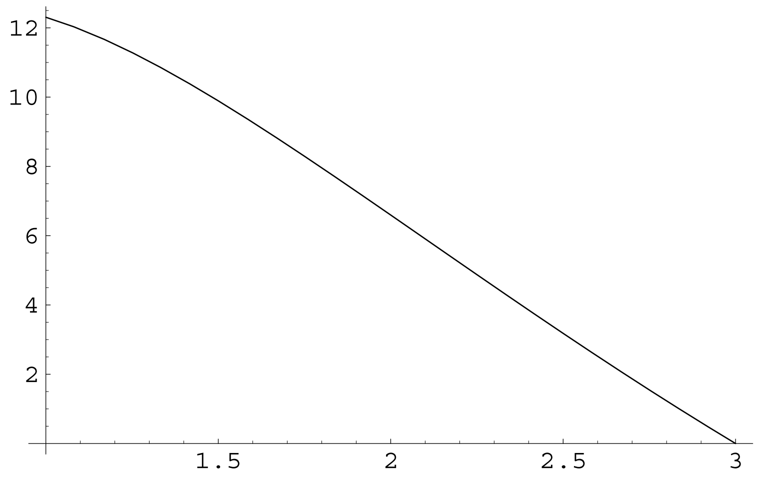
The solution can also be plotted as follows.

```
Plot[u[x] /. sol[[1]], {x, 1, 3}];
```



The first derivative of the solution can also be plotted as follows.

```
Plot[Evaluate[D[(u[x] /. sol[[1]]), x]], {x, 1, 3}];
```



6. Programming in *Mathematica*

Defining new functions

For calculations that involve many steps using interactive approach may become tedious. *Mathematica* offers a rich and sophisticated programming environment to create functions that perform complex series of computations.

The general syntax of the function definition is as follows.

```
newFunctionName[var1_, var2_, ...] := Module[
  {localVar1, localVar2, …},
        statement 1;
        statement 2;
        ⋮
        last statement
    ]
```

The newFunctionName can be any name that the user wants. Since all built-in functions start with an upper-case letter, it may be a good idea to start your functions with a lower case letter to avoid any conflict with built-in functions. All needed variables for the function are enclosed in square brackets. For reasons beyond the scope of this introductory tutorial, an underscore must be appended to all names in the variable list. The definition of the function starts with := followed by the word 'Module' and opening square bracket. The first line in the definition is a list of local variables that are to be used only inside the body of the function. Outside of the function these variables do not exist. The list of local variables ends with a comma. The remainder of the function body can contain as many statements as needed to achieve the goal of the function. Each statement ends with a semi-colon. The last statement does not have a semi-colon and is the one that is returned by the function when it is used. The end of the function is indicated by the closing square bracket.

Simple one line functions, that do not need any local variables, can be defined simply as follows.

```
oneLineFcn[var1_, var2_, ...] := expression involving vars
```

An example of a one line function is presented in the next section. As an example of Module we define the following function to return stresses in thick-walled cylinders. The tangential and radial stresses in an open-ended thick cylinder are given by the following formulas.

$$\sigma_t = \frac{p_i\, r_i^2}{r_0^2 - r_i^2}\left(1 + \frac{r_0^2}{r^2}\right) \qquad \sigma_r = \frac{p_i\, r_i^2}{r_0^2 - r_i^2}\left(1 - \frac{r_0^2}{r^2}\right)$$

where p_i is internal pressure on the cylinder, r_i and r_0 are inner and outer radii and r is radius of the point where the stress is desired.

```
thickCylinderStresses[pi_, ri_, r0_, r_] :=
 Module[{c1, c2, σt, σr},
    c1=pi ri^2/(r0^2 - ri^2);
    c2=r0^2/r^2;
    σt=c1(1+c2);
    σr=c1(1-c2);
    {σt, σr}
 ]
```

After entering the function definition, the input must be executed to actually tell *Mathematica* about the new function. Unless there is an error, the execution of the function definition line does not produce any output. After this the function can be used as any other *Mathematica* function. For example we can compute stresses in a cylinder with pi = 20, ri = 5, r0 = 15 and r = 10 as follows.

```
thickCylinderStresses[20, 5, 15, 10]
```

$$\left\{\frac{65}{8}, -\frac{25}{8}\right\}$$

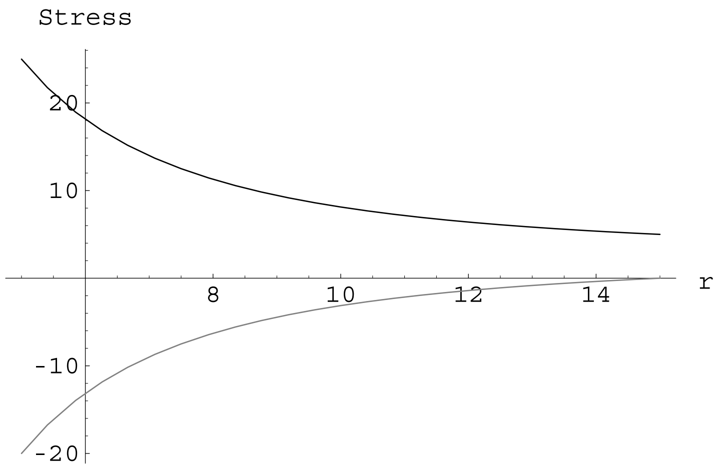
Leaving some variables in symbolic form we can get symbolic results.

```
{st, sr} = thickCylinderStresses[20, 5, 15, r]
```

$$\left\{\frac{5}{2}\left(1+\frac{225}{r^2}\right), \frac{5}{2}\left(1-\frac{225}{r^2}\right)\right\}$$

These expression can be plotted to see the stress distribution through the cylinder wall.

```
Plot[{st, sr}, {r, 5, 15}, AxesLabel -> {"r", "Stress"},
  PlotStyle -> {{GrayLevel[0]}, {GrayLevel[0.5]}}];
```



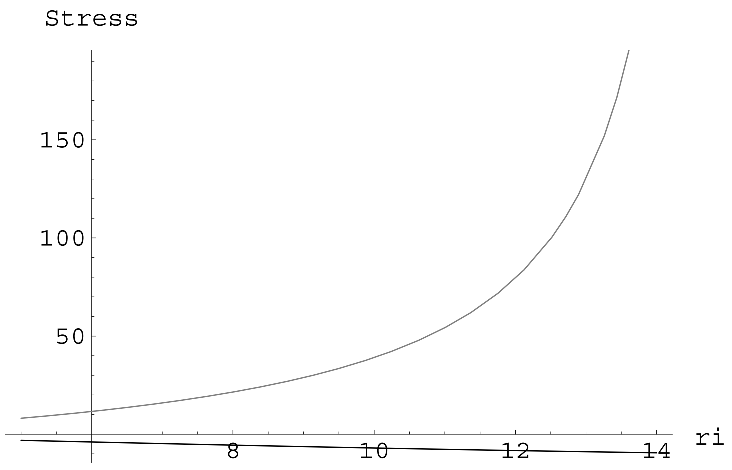
A natural question might be to see what happens to the stresses as the cylinder becomes "thin"? This question can be answered by evaluating the stresses at center of wall thickness as a function of inner radius as follows.

```
{st, sr} = thickCylinderStresses[20, ri, 15, (ri + 15) / 2]
```

$$\left\{\frac{20\,ri^2\left(1+\frac{900}{(15+ri)^2}\right)}{225-ri^2}, \frac{20\,ri^2\left(1-\frac{900}{(15+ri)^2}\right)}{225-ri^2}\right\}$$

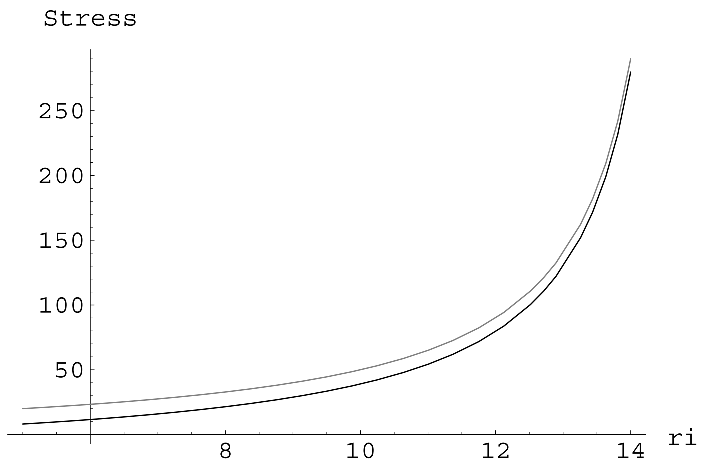
A plot of these expressions is as follows.

```
Plot[{st, sr}, {ri, 5, 14}, AxesLabel -> {"ri", "Stress"},
  PlotStyle -> {{GrayLevel[0.5]}, {GrayLevel[0]}}];
```



We can see that σ_t becomes predominant stress as the wall becomes thin. The usual approximation for thin walled cylinders is $\sigma_t = \frac{p_i\, r}{t}$. We can graphically see how this approximation compares with the thick cylinder solution as follows.

```
Plot[{st, (20 (ri + 15) / 2) / (15 - ri)},
  {ri, 5, 14}, AxesLabel -> {"ri", "Stress"},
  PlotStyle -> {{GrayLevel[0]}, {GrayLevel[0.5]}}];
```



Use of Map, Apply, and Thread

Frequently we need to perform operations on a list of functions or variables. Many built-in functions are designed to operate on lists directly. However there are still many situations where it is necessary to perform some operation on a list of quantities. In traditional programming language, we typically perform these operations using a Do loop or an equivalent. *Mathematica* has a Do loop that can be used in a similar way. However it also offers many other elegant and more convenient ways of doing the same thing. **`Map`**, **`Thread`** and **`Apply`** are three of these handy functions. These functions have been used in several examples in this text.

Suppose we are given the following data that we have entered as lists in *Mathematica*.

```
data1 = {18.24, 12.12, 15.23, 5.26};
data2 = {12.24, 19.16, 35.07, 23.46, -10.62, -7.43, 2.62, 10.42};
data3 = {8.23, 8.96, 8.35, 9.16, 8.05};
data4 = {8.12, 8.26, 8.34, 9.01, 9.11, 8.95, 7.29};
```

We want to compute average of data. The computation is fairly simple. We simply need to add all entries in a data list and divide them by the number of entries. Computation of sum or multiplication of all elements of a list is done very conveniently by using Apply. It does not matter how long the list is or whether it consists of numerical or symbolic quantities.

```
Apply[Plus, data1]
```

```
50.85
```

The first argument of Apply is the function to be applied and the second is the list of items. Multiplication of all elements can be computed in exactly the same way.

```
Apply[Times, data1]
```

```
17709.8
```

Obviously the function to be applied must expect a list as its argument. For example applying **`Sin`** function to a list will not produce any useful result.

```
Apply[Sin, data1]
```

```
Sin::argx : Sin called with 4 arguments; 1 argument is expected.

Sin[18.24, 12.12, 15.23, 5.26]
```

The Map function is similar to Apply, but it applies a function to each element of a list. For example if we want to compute Sin of each term in data1 list, we can do it as follows.

```
Map[Sin, data1]

{-0.572503, -0.431695, 0.459972, -0.853771}
```

To compute the average all we need to now is to divide the sum by the number of entries in the list. The Length function, described earlier, does exactly this. Thus the following one line program can compute average of any list.

```
average[n_] := Apply[Plus, n] / Length[n]

average[data1]

12.7125
```

To compute average of all different data lists, we can use the average function repeatedly on the other lists. However it is much more convenient to define a new list of all data and simply Map average function to each element of this list. We first define a new list and then use the average function defined above.

```
allData = {data1, data2, data3, data4}

{{18.24, 12.12, 15.23, 5.26},
 {12.24, 19.16, 35.07, 23.46, -10.62, -7.43, 2.62, 10.42},
 {8.23, 8.96, 8.35, 9.16, 8.05},
 {8.12, 8.26, 8.34, 9.01, 9.11, 8.95, 7.29}}


averages = Map[average, allData]

{12.7125, 10.615, 8.55, 8.44}
```

The elegance of these constructs is that we never need to know how long the lists really are. We can keep adding or deleting elements into any of the lists and the process will keep working. Map and Apply functions are used so frequently in *Mathematica* programming that several short cuts have been designed to make their use even more efficient. One such useful technique is combining Map with the function definition itself. In the example of computation of averages, we had to define a function (called average) and then apply it to elements of the list using Map. We can do exactly the same thing, without explicitly defining the function, as follows.

```
averageReturns = Map[Apply[Plus, #] / Length[#] &, allData]

{12.7125, 10.615, 8.55, 8.44}
```

We can see that the first argument is exactly the definition of the function. The # sign stands for the function argument. The ampersand (&) at the end is very important. It essentially tells *Mathematica* that we are defining a function as the first argument of Map.

The Thread function is similar to Map as it threads a function over its arguments. The most common use of this function in the text has been to define rules for substitution into an expression. Suppose we have a function of 4 variables that we would like to evaluate at a given point.

```
f = x1 x2 + Sin[x3] Cos[x4];
pt = {1.1, 2.23, 3.2, 4.556};
vars = {x1, x2, x3, x4};
```

A tedious way to evaluate f at the given point is as follows.

```
f /. {x1 -> 1.1, x2 -> 2.23, x3 -> 3.2, x4 -> 4.556}

2.46209
```

A more convenient way is to use Thread to define substitution rule.

```
Thread[vars -> pt]

{x1 → 1.1, x2 → 2.23, x3 → 3.2, x4 → 4.556}


f /. Thread[vars -> pt]

2.46209
```

Again the advantage of last form is clear. We don't have to change anything if the number of variables is increased or decreased.

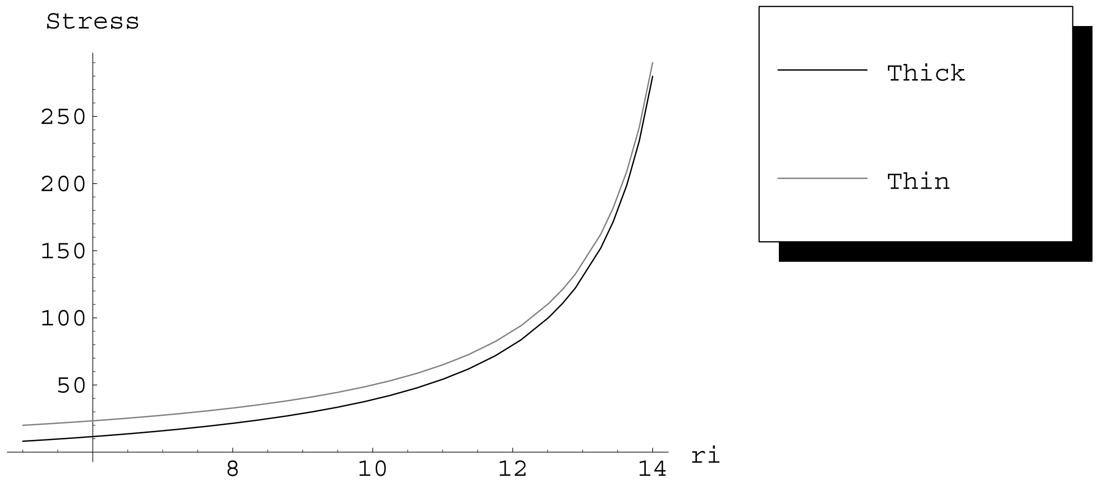
7. Packages in *Mathematica*

Mathematica comes with a wide variety of special packages. Among these are the Linear Algebra and Graphics Packages. This packages provide additional commands for manipulating matrices and plots. Loading the matrix manipulation commands from the LinearAlgebra Package and Graphics: Legend packages is accomplished as follows:

```
Needs["LinearAlgebra`MatrixManipulation`"];
Needs["Graphics`Legend`"];
```

With Graphics`Legend we can show labels for different graphs on the same plot. For example

```
Plot[{st, (20 (ri + 15) / 2) / (15 - ri)},
  {ri, 5, 14}, AxesLabel -> {"ri", "Stress"},
  PlotStyle -> {{GrayLevel[0]}, {GrayLevel[0.5]}},
    PlotLegend -> {"Thick", "Thin"},
    LegendPosition -> {1, 0}];
```



The large of functions created for this text are included in the MinTools package. Specific instructions for loading and using this package are included on the accompanying CD.

8. On-Line Help

Mathematica contains a complete on-line help system. The Help menu provides access to all *Mathematica* features. In fact the entire *Mathematica* 3.0 book is on-line. In addition you can obtain information about any command by typing '?' followed by the command name. This form also supports use of wild-cards. For example to get a listing of all commands that start with letter G type

```
?G*
```

```
Gamma                          Generic
GraphicsSpacing
GammaRegularized               Get                            GrayLevel
GaussianIntegers               GetContext                     Greater
GaussKronrod                   GetLinebreakInformationPacket
GreaterEqual
GaussPoints                    GoldenRatio
GridBaseline
GCD                            Goto                           GridBox
Gear                           Gradient
GridBoxOptions
GegenbauerC                    Graphics                       GridFrame
General                        Graphics3D                     GridLines
GenerateBitmapCaches           GraphicsArray
GroebnerBasis
GenerateConditions             GraphicsData
GroupPageBreakWithin
GeneratedCell
```

Detailed instructions about a specific command can be obtained by typing ? followed by the command name. For example

```
?GaussPoints
```

```
GaussPoints is an option for NIntegrate. With
  GaussPoints -> n, the Gaussian part of Gauss-Kronrod
  quadrature uses n points. With GaussPoints -> Automatic,
  an internal algorithm chooses the number of points.
```

Bibliography

1. Wolfram, S., The *Mathematica* Book, 3rd Edition, Wolfram Media/Cambridge University Press, Boston, MA, 1996.

2. Wickham-Jones, T., *Mathematica* Graphics: Techniques and Applications, TELOS (Springer-Verlag), Santa Clara, CA, 1994.

3. Maeder, R., Programming in *Mathematica*, 3rd Edition, Addison-Wesley, Redwood City, CA, 1997.

4. Shaw, W.T. and Tigg, J., Applied *Mathematica*: Getting Started, Getting It Done, Addison-Wesley, Reading, MA, 1994.

5. Gray, J.W., Mastering *Mathematica,* AP Professional, Cambridge, MA, 1994.

6. Wagner, D.B., Power Programming with *Mathematica*, McGraw-Hill, New York, NY, 1996.

7. Abel, M.L. and Braselton, J.P., *Mathematica* By Example, Academic Press, San Diego, CA, 1997.

8. Gray, Theodore and Glynn, Jerry, A Beginners Guide to *Mathematica*, Version 2, Addison-Wesley, Reading, MA, 1992.

9. Bahder, T., *Mathematica* for Scientists and Engineers, Addison-Wesley, Reading, MA, 1995.

Introduction to *Mathematica*

Mathematica Exercises

Prepare a *Methematica* notebook with solutions to the following problems. Put each solution in a separate section. Add your comments/description in text cells if needed. Display the matrices in the matrix form.

1.1 Given a function $f(x) = \dfrac{1}{x + ax^2}$, compute $\dfrac{df}{dx}$ at x=1.5 and $\int_1^2 f dx$.

1.2 Given a function $f(x, y, z) = \dfrac{1}{x^2 + y^2 + z^2}$, compute its gradient vector and the Hessian matrix. Evaluate these quantities at x=1, y=2 and z=3. Note that for a functional of n variables, the gradient and Hessian are defined as

$$\text{Gradient vector } \nabla f(\mathbf{x}) = \begin{pmatrix} \dfrac{\partial f}{\partial x_1} \\ \vdots \\ \dfrac{\partial f}{\partial x_n} \end{pmatrix}$$

$$\text{Hessian matrix } \nabla^2 f(\mathbf{x}) = \begin{pmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1 \partial x_n} \\ \dfrac{\partial^2 f}{\partial x_2 \partial x_1} & \dfrac{\partial^2 f}{\partial x_2^2} & \cdots & \dfrac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \dfrac{\partial^2 f}{\partial x_n \partial x_1} & \dfrac{\partial^2 f}{\partial x_n \partial x_2} & & \dfrac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

1.3 Given a 3-by-1 vector $\mathbf{a} = (x, \sin x, \cos x)^T$, obtain the 3-by-3 matrix resulting from $\int \mathbf{a}\mathbf{a}^T dx$.

1.4 Consider the following system of homogeneous linear equations

$$\begin{aligned} x_1 - 2x_2 + x_3 + 3x_4 &= 0 \\ 2x_1 + 2x_2 - x_3 + x_4 &= 0 \\ -x_1 - x_2 + 3x_3 + 2x_4 &= 0 \\ x_1 - 8x_2 - x_3 + 3x_4 &= 0 \end{aligned}$$

Show that the system of equations is singular by computing the determinant of the coefficient matrix.

1.5 For the system of equations given in problem 1.4, if $x_4 = 1$, then the first three equations are nonsingular. Compute the solution.