VBA Programming in EXCEL: Matrices – 2-D Arrays – Range variables

By Gilberto E. Urroz, September 2013

In chapter 3 we introduced the different types of looping statements applying them to solutions of equations and of ordinary differential equations. We also introduced the idea of one-dimensional (1-D) arrays and used them for calculating statistics of a sample, and showing applications of random number generation to simulating random motions. Finally, we introduced some basic applications of string functions. In this chapter we will introduce matrices and their operations, and the use of two-dimensional (2-D) arryas for matrix operation and applications.

Matrices

A matrix is a rectangular array of numbers or expressions. The following are examples of matrices:

$$A=\begin{pmatrix}-2 & 5\\ 1 & 3\end{pmatrix} \quad B=\begin{pmatrix}-5 & 3\\ 0 & 10\\ -1 & 7\end{pmatrix} \quad C=\begin{pmatrix}5 & -2 & 3\\ 1 & 0 & 2\end{pmatrix} \quad D=\begin{pmatrix}1 & 2 & 4\\ 0 & 3 & -2\\ 5 & -8 & 6\end{pmatrix}$$

A matrix is typically identified by its numbers of rows and columns. Thus, in the examples above, matrix A has 2 rows and 2 columns, and it's referred to as a 2x2 matrix, sometimes written as $\mathbf{A}_{2x2}$. The other matrices in the examples above can be written as $\mathbf{B}_{3x2}$, $\mathbf{C}_{2x3}$, and $\mathbf{D}_{3x3}$. Matrices such as **A** and **D**, which have the same number of rows and columns, are called *square matrices*.

A generic reference to the matrix elements is given by the expression: $\boldsymbol{A}=[a_{i,j}]_{mxn}$.

This notation indicates that an arbitrary term in the matrix is $a_{i,j}$ corresponding to the element in row *i* and column *j*. The notation shown above also indicates that the matrix has *m* rows and *n* columns.

For example, consider the 2x3 matrix $\boldsymbol{A}=\begin{bmatrix}2 & 3 & -5\\ 1 & -7 & 6\end{bmatrix}$. Some of its individual elements are:

$a_{1,,2} = 3$ (first row, second column), $a_{2,3} = 6$ (second row, third column), etc.

***Matrix operations*: addition and subtraction** – Two matrices of the same size can be added or subtracted by adding or subtracting their corresponding elements. Let $\boldsymbol{A}=[a_{i,j}]_{mxn}$, $\boldsymbol{B}=[b_{i,j}]_{mxn}$, and $\boldsymbol{C}=[c_{i,j}]_{mxn}$ be two matrices of the same size, then, addition is defined as:

$$\boldsymbol{C}_{mxn}=\boldsymbol{A}_{mxn}+\boldsymbol{B}_{mxn} \text{ , with } c_{i,j}=a_{i,j}+b_{i,j} \text{ .}$$

For example, consider the 2x3 matrices $\boldsymbol{A}=\begin{bmatrix}2 & 3 & -5\\ 1 & -7 & 6\end{bmatrix}$, $\boldsymbol{B}=\begin{bmatrix}0 & -5 & 3\\ -8 & 4 & 2\end{bmatrix}$. Then,

$$\boldsymbol{C}=\boldsymbol{A}+\boldsymbol{B}=\begin{bmatrix}2 & 3 & -5\\ 1 & -7 & 6\end{bmatrix}+\begin{bmatrix}0 & -5 & 3\\ -8 & 4 & 2\end{bmatrix}=\begin{bmatrix}2+0 & 3+(-5) & -5+3\\ 1+(-8) & -7+4 & 6+2\end{bmatrix}=\begin{bmatrix}2 & -2 & -2\\ -7 & -3 & 8\end{bmatrix}.$$

Addition is commutative, i.e., $\boldsymbol{C}_{mxn}=\boldsymbol{A}_{mxn}+\boldsymbol{B}_{mxn}=\boldsymbol{B}_{mxn}+\boldsymbol{A}_{mxn}$. Thus, for the matrices **A** and **B** defined immediately above, i.e., matrices $\boldsymbol{A}=\begin{bmatrix}2 & 3 & -5\\ 1 & -7 & 6\end{bmatrix}$ and $\boldsymbol{B}=\begin{bmatrix}0 & -5 & 3\\ -8 & 4 & 2\end{bmatrix}$, we can also write:

$$\boldsymbol{C}=\boldsymbol{B}+\boldsymbol{A}=\begin{bmatrix}0 & -5 & 3\\ -8 & 4 & 2\end{bmatrix}+\begin{bmatrix}2 & 3 & -5\\ 1 & -7 & 6\end{bmatrix}=\begin{bmatrix}0+2 & -5+3 & 3+(-5)\\ -8+1 & 4+(-7) & 2+6\end{bmatrix}=\begin{bmatrix}2 & -2 & -2\\ -7 & -3 & 8\end{bmatrix}=\boldsymbol{A}+\boldsymbol{B}.$$

Subtraction is defined as: $\boldsymbol{C}_{mxn}=\boldsymbol{A}_{mxn}-\boldsymbol{B}_{mxn}$, with $c_{i,j}=a_{i,j}-b_{i,j}$. For example,

$$\boldsymbol{C}=\boldsymbol{A}-\boldsymbol{B}=\begin{bmatrix}2 & 3 & -5\\ 1 & -7 & 6\end{bmatrix}+\begin{bmatrix}0 & -5 & 3\\ -8 & 4 & 2\end{bmatrix}=\begin{bmatrix}2-0 & 3-(-5) & -5-3\\ 1-(-8) & -7-4 & 6-2\end{bmatrix}=\begin{bmatrix}2 & 8 & -8\\ 9 & -11 & 4\end{bmatrix}.$$

Multiplication by a scalar – Let *k* be a scalar quantity, then $\boldsymbol{C}_{mxn}=k\cdot\boldsymbol{A}_{mxn}$, with $c_{i,j}=k\cdot a_{i,j}$. For example, given $\boldsymbol{A}=\begin{bmatrix}2 & 3 & -5\\ 1 & -7 & 6\end{bmatrix}$ and $k = 2.5$, then

$$\boldsymbol{C}=k\cdot\boldsymbol{A}=2.5\times\begin{bmatrix}2 & 3 & -5\\ 1 & -7 & 6\end{bmatrix}=\begin{bmatrix}2.5\times2 & 2.5\times3 & 2.5\times(-5)\\ 2.5\times1 & 2.5\times(-7) & 2.5\times6\end{bmatrix}=\begin{bmatrix}5.0 & 7.5 & -12.5\\ 2.5 & -17.5 & 15.0\end{bmatrix}.$$

Linear combination of matrices - A linear combination of matrices **A** and **B** affected by constants *k* and *r*, is defined then as: $\boldsymbol{C}_{mxn}=k\cdot\boldsymbol{A}_{mxn}+r\cdot\boldsymbol{B}_{mxn}$, with $c_{i,j}=k\cdot a_{i,j}+r\cdot b_{i,j}$. For example, given the matrices $\boldsymbol{A}=\begin{bmatrix}2 & 3 & -5\\ 1 & -7 & 6\end{bmatrix}$ and $\boldsymbol{B}=\begin{bmatrix}0 & -5 & 3\\ -8 & 4 & 2\end{bmatrix}$, and the scalar values $k = 2$ and $r = 3$, we can write:

$$\boldsymbol{C}=k\cdot\boldsymbol{A}+r\cdot\boldsymbol{B}=2\times\begin{bmatrix}2 & 3 & -5\\ 1 & -7 & 6\end{bmatrix}+3\times\begin{bmatrix}0 & -5 & 3\\ -8 & 4 & 2\end{bmatrix}=\begin{bmatrix}2\times2+3\times0 & 2\times3+3\times(-5) & 2\times(-5)+3\times3\\ 2\times1+3\times(-8) & 2\times(-7)+3\times4 & 2\times6+3\times2\end{bmatrix}$$

$$\boldsymbol{C}=\begin{bmatrix}4+0 & 6+(-15) & -10+9\\ 2+(-24) & -14+12 & 12+6\end{bmatrix}=\begin{bmatrix}4 & -9 & -1\\ -22 & -2 & 18\end{bmatrix}.$$

Vectors of data as matrices - In Chapter 3 we introduced row and column vectors. A row vector can be thought of as a matrix with a single row ($m = 1$), while a column vector can be thought of as a matrix with a single column ($n = 1$). Examples:

- Row vector: $\boldsymbol{u}=\boldsymbol{u}_{1\times 4}=\begin{bmatrix}-2 & 5 & 0 & 2\end{bmatrix}$

- Column vector: $\boldsymbol{v}=\boldsymbol{v}_{4\times 1}=\begin{bmatrix}-2\\5\\0\\2\end{bmatrix}$

Multiplication of matrices – Matrix multiplication is defined only if the number of columns of the first multiplicand is equal to the number of rows of the second multiplicand. Thus, matrix multiplication requires that $C_{mxn}=A_{mxp}\cdot B_{pxn}$. The elements of the matrix multiplication are calculated as:

$$c_{i,j}=\sum_{k=1}^{p} a_{i,k}\cdot b_{k,j}=a_{i,1}\cdot b_{1,j}+a_{i,2}\cdot b_{2,j}+...+a_{i,p}\cdot b_{p,j} \quad .$$

Thus, each element in the product, $c_{i,j}$, is similar to the scalar product of a row vector corresponding to the *i*-th row of matrix $[\mathbf{A}]_i$ with a column vector corresponding to the *j*-th column of matrix $[\mathbf{B}]_j$, i.e., the scalar (dot) product of the vectors:

$$[\boldsymbol{A}]_i=\begin{bmatrix}a_{i,1} & a_{i,2} & ... & a_{i,p}\end{bmatrix} \text{ , and } [\boldsymbol{B}]_j=\begin{bmatrix}b_{1,j}\\a_{2,j}\\...\\a_{p,j}\end{bmatrix} \text{ , so that } c_{i,j}=a_{i,1}\cdot b_{1,j}+a_{i,2}\cdot b_{2,j}+...+a_{i,p}\cdot b_{p,j} \quad .$$

For example, consider the matrices $\boldsymbol{A}_{2\times 3}=\begin{bmatrix}2 & 3 & -5\\1 & -7 & 6\end{bmatrix}$ and $\boldsymbol{B}_{3\times 2}=\begin{bmatrix}0 & -5\\3 & -8\\4 & 2\end{bmatrix}$, which can be multiplied to produce:

$$\boldsymbol{C}_{2\times 2}=\boldsymbol{A}_{2\times 3}\cdot\boldsymbol{B}_{3\times 2}=\begin{bmatrix}2 & 3 & -5\\1 & -7 & 6\end{bmatrix}\cdot\begin{bmatrix}0 & -5\\3 & -8\\4 & 2\end{bmatrix}=\begin{bmatrix}2\times 0+3\times 3+(-5)\times 4 & 2\times(-5)+3\times(-8)+(-5)\times 2\\1\times 0+(-7)\times 3+6\times 4 & 1\times(-5)+(-7)\times(-8)+6\times 2\end{bmatrix}$$

$$\boldsymbol{C}_{2\times 2}=\begin{bmatrix}-11 & -44\\3 & 63\end{bmatrix} \quad .$$

Matrix multiplication is not commutative, i.e., in general, $\boldsymbol{A}\cdot\boldsymbol{B}\neq\boldsymbol{B}\cdot\boldsymbol{A}$. For example, for matrices $\boldsymbol{A}_{2\times 3}=\begin{bmatrix}2 & 3 & -5\\1 & -7 & 6\end{bmatrix}$ and $\boldsymbol{B}_{3\times 2}=\begin{bmatrix}0 & -5\\3 & -8\\4 & 2\end{bmatrix}$, we can calculate

$$\boldsymbol{D}_{3\times3}=\boldsymbol{B}_{3\times2}\cdot\boldsymbol{A}_{2\times3}=\begin{bmatrix}0&-5\\3&-8\\4&2\end{bmatrix}\cdot\begin{bmatrix}2&3&-5\\1&-7&6\end{bmatrix}=\begin{bmatrix}0\times2+(-5)\times1&0\times3+(-5)\times(-7)&0\times(-5)+(-5)\times6\\3\times2+(-8)\times1&3\times3+(-8)\times(-7)&3\times(-5)+(-8)\times6\\4\times2+2\times1&4\times3+2\times(-7)&4\times(-5)+2\times6\end{bmatrix}$$

$$\boldsymbol{D}_{3\times3}=\begin{bmatrix}-5&35&-30\\-2&65&-63\\10&-2&-8\end{bmatrix}$$

Matrix representation of systems of linear equations – Consider the following system of 3 linear equations with 3 unknowns:

$$\begin{aligned}-2\cdot x+5\cdot y+7\cdot z&=26\\3\cdot x-5\cdot y+z&=25\\x+10\cdot y-5\cdot z&=-47\end{aligned}$$

We can collect the coefficients of the unknowns into a matrix: $\boldsymbol{A}=\begin{bmatrix}-2&5&7\\3&-5&1\\1&10&-5\end{bmatrix}$, and the unknowns and the coefficients in the right-hand side of the equations as the following column vectors: $\boldsymbol{x}=\begin{bmatrix}x\\y\\z\end{bmatrix}$, and $\boldsymbol{b}=\begin{bmatrix}26\\25\\-47\end{bmatrix}$. You can easily show that the system of equations, above, can be written as: $\begin{bmatrix}-2&5&7\\3&-5&1\\1&10&-5\end{bmatrix}\cdot\begin{bmatrix}x\\y\\z\end{bmatrix}=\begin{bmatrix}26\\25\\-47\end{bmatrix}$, or, more concisely, as: $\boldsymbol{A}\cdot\boldsymbol{x}=\boldsymbol{b}$. The solution to this matrix equation can be found by using the *inverse* of matrix **A**, or $\mathbf{A}^{-1}$, so that $\boldsymbol{x}=\boldsymbol{A}^{-1}\cdot\boldsymbol{b}$.

Diagonal matrix, identity matrix and the inverse matrix – A <u>*diagonal matrix*</u> is a square matrix whose elements are all zero, except for those in the *main diagonal*. The main diagonal of a matrix consists of all elements with indices $i = j$. The following are some examples of diagonal matrices:

$$D_1=\begin{pmatrix}5&0\\0&-2\end{pmatrix}\qquad D_2=\begin{pmatrix}-5&0&0\\0&3&0\\0&0&8\end{pmatrix}\qquad D_3=\begin{pmatrix}10&0&0&0\\0&-8&0&0\\0&0&7&0\\0&0&0&11\end{pmatrix}$$

A diagonal matrix whose main diagonal elements are all equal to 1.0 is referred to as the <u>*identity matrix*</u>. Identity matrices of dimensions 2x2, 3x3, and 4x4 are shown next:

$$I_1=\begin{pmatrix}1&0\\0&1\end{pmatrix}\qquad I_2=\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\qquad I_3=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}$$

The reason why these **I** matrices are referred to as identity matrices is because they have the property that if **A** is a square matrix (of the same size as **I**), then: $A \cdot I = I \cdot A = A$. For example, given

$$A = \begin{bmatrix} 2 & -4 \\ 1 & -2 \end{bmatrix} \text{ , then } A \cdot I = \begin{bmatrix} 2 & -4 \\ 1 & -2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 \times 1 + (-4) \times 0 & 2 \times 0 + (-4) \times 1 \\ 1 \times 0 + (-2) \times 1 & 1 \times 0 + (-2) \times 1 \end{bmatrix} = \begin{bmatrix} 2 & -4 \\ 1 & -2 \end{bmatrix} = A \quad .$$

The *inverse* of a square matrix **A** is a matrix $\mathbf{A}^{-1}$ such that $A \cdot A^{-1} = A^{-1} \cdot A = I$. Calculation of an inverse matrix is not straightforward. Algorithms for such calculations will be presented later in this chapter.

Transpose of a matrix – The *transpose* $[\mathbf{A}^T]_{nxm}$ of a matrix $[\mathbf{A}]_{nxm}$ is a matrix in which the rows and columns of the original matrix have been exchanged. Formally, this means that $a^T_{i,j} = a_{j,i}$, i.e., the indices for rows and columns get switched. The following is an example of a 3x2 matrix and its transpose:

$$\mathbf{A} = \begin{pmatrix} -5 & 3 \\ 0 & 10 \\ -1 & 7 \end{pmatrix} \qquad \mathbf{A}^{\mathrm{T}} = \begin{pmatrix} -5 & 0 & -1 \\ 3 & 10 & 7 \end{pmatrix}$$

NOTE: Matrix calculations using *SMath Studio*

The mathematical software *SMath Studio* was introduced in Chapter 3 for calculating integrals, however, it can be used for many other mathematical calculations.

General information on the software is available here:
http://www.neng.usu.edu/cee/faculty/gurro/SMathStudio.html

Specific information on matrix manipulation in *SMath Studio* is available here:
http://www.neng.usu.edu/cee/faculty/gurro/Classes/ClassNotesAllClasses/CEE3510/LectureNotes/PipePumpFrictionMinorSMathStudio/UsingTheMatricesPaletteInSMathStudio.pdf

Here is an example of a matrix inverse calculated using SMath Studio:

$$\mathbf{A} := \begin{pmatrix} 1 & 2 & 4 \\ 0 & 3 & -2 \\ 5 & -8 & 6 \end{pmatrix} \qquad \mathbf{A}^{-1} = \begin{pmatrix} -0.0256 & 0.5641 & 0.2051 \\ 0.1282 & 0.1795 & -0.0256 \\ 0.1923 & -0.2308 & -0.0385 \end{pmatrix}$$

$$\mathbf{A} \cdot \mathbf{A}^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{A}^{-1} \cdot \mathbf{A} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The following example shows the solution to a system of linear equations using SMath Studio:

Set-up:

$$\begin{aligned} -2\cdot x+5\cdot y+7\cdot z&=26\\ 3\cdot x-5\cdot y+z&=25\\ x+10\cdot y-5\cdot z&=-47 \end{aligned} \qquad A:=\begin{pmatrix}-2 & 5 & 7\\ 3 & -5 & 1\\ 1 & 10 & -5\end{pmatrix} \qquad b:=\begin{pmatrix}26\\25\\-47\end{pmatrix}$$

Solution:

$$x:=A^{-1}\cdot b \qquad x=\begin{pmatrix}3\\-2\\6\end{pmatrix}$$

Check:

$$A\cdot x=\begin{pmatrix}26\\25\\-47\end{pmatrix} \quad \text{or} \quad A\cdot x-b=\begin{pmatrix}0\\0\\0\end{pmatrix}$$

In an upcoming section, we'll learn how to perform these operations using EXCEL.

The determinant of a matrix – The *determinant* of a square matrix is a scalar value (a single number) resulting from a systematic combination of multiplications and additions of the elements of the matrix. For example, for a 2x2 matrix, the determinant is calculated as:

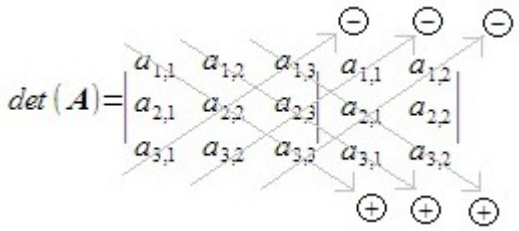
$$det(\boldsymbol{A})=\begin{vmatrix}a_{1,1} & a_{1,2}\\ a_{2,1} & a_{2,2}\end{vmatrix}=a_{1,1}\cdot a_{2,2}-a_{2,1}\cdot a_{1,2}$$

This operation can be shown schematically as follows:

$$det(\boldsymbol{A})=\begin{vmatrix}a_{1,1} & a_{1,2}\\ a_{2,1} & a_{2,2}\end{vmatrix}$$

For example, $\begin{vmatrix}2 & -3\\ 5 & 4\end{vmatrix}=2\times 3-5\times(-3)=6-(-15)=21$.

For a 3x3 matrix, the determinant can be calculated using the following schematic. Basically, the first two columns of the determinant are copied to the right of the three original columns, and products of the different diagonals of 3 elements are calculated, with those downward products being positive, and those upward products being negative. Finally, all those products are added together with their corresponding signs.



For example, to calculate $\begin{vmatrix} 3 & -2 & 5 \\ -1 & 2 & 4 \\ 3 & 2 & -5 \end{vmatrix}$ we write: $\begin{vmatrix} 3 & -2 & 5 \\ -1 & 2 & 4 \\ 3 & 2 & -5 \end{vmatrix} \begin{matrix} 3 & -2 \\ -1 & 2 \\ 3 & 2 \end{matrix}$ and calculate the diagonal products of 3 terms, positive downwards, negative upwards, i.e.,

$$3\times 2\times(-5)+(-2)\times 4\times 3+5\times(-1)\times 2-(3\times 2\times 5+2\times 4\times 3+(-5)\times(-1)\times(-2)) =$$
$$-30+(-24)+(-10)-(30+24+(-10))=-108 \quad .$$

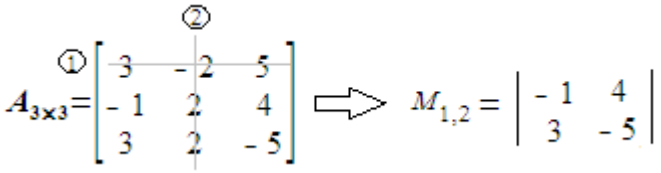
Minors - A *minor* (or *minor* determinant), $M_{i,j}$, is a determinant associated with a particular element of a matrix **A**, say $a_{i,j}$, obtained by eliminating the *i*-th row and *j*-th column of the original matrix. Thus, if matrix **A** is an *n*x*n* matrix, the minor $M_{i,j}$, will be an (*n*-1)x(*n*-1) array. For example, for matrix

$$\boldsymbol{A}_{3\times 3}=\begin{bmatrix} 3 & -2 & 5 \\ -1 & 2 & 4 \\ 3 & 2 & -5 \end{bmatrix} ,$$

associated with the first row we can form the following minor determinants:

$$M_{1,1}=\begin{vmatrix} 2 & 4 \\ 2 & -5 \end{vmatrix} , \quad M_{1,2}=\begin{vmatrix} -1 & 4 \\ 3 & -5 \end{vmatrix} , \text{ and } \quad M_{1,3}=\begin{vmatrix} -1 & 2 \\ 3 & 2 \end{vmatrix} .$$

The following figure shows how to form the minor $M_{1,2}$ for the matrix **A** shown above:



Expanding a determinant using minors – To calculate a determinant using minors, first select a row or column to start the expansion, and then add the product of each term $a_{i,j}$ in that row or column times the corresponding minor, $M_{i,j}$, multiplied by $(-1)^{i+j}$. The expanding of the minor determinants continues along these lines until reaching a number of 2x2 determinants, which can be calculated using the rule shown earlier for such determinants, i.e.,

$$\begin{vmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{vmatrix} = a_{1,1} \cdot a_{2,2} - a_{2,1} \cdot a_{1,2} \quad .$$

For a 3x3 determinant, the expansion using minors is relatively simple. For example, we expand the following 3x3 determinant using the first row:

$$\begin{vmatrix} 3 & -2 & 5 \\ -1 & 2 & 4 \\ 3 & 2 & -5 \end{vmatrix} = 3 \cdot (-1)^{1+1} \cdot \begin{vmatrix} 2 & 4 \\ 2 & -5 \end{vmatrix} + (-2) \cdot (-1)^{1+2} \cdot \begin{vmatrix} -1 & 4 \\ 3 & -5 \end{vmatrix} + 5 \cdot (-1)^{1+3} \cdot \begin{vmatrix} -1 & 2 \\ 3 & 2 \end{vmatrix} =$$

$$3 \times 1 \times (-10-8) + (-2) \times (-1) \times (5-12) + 5 \times 1 \times (-2-6) = -54 - 14 - 40 = -108 \quad .$$

For a 4x4 determinant, the first set of minors will be 3x3 determinants that can be, in turn, expanded using 2x2 minors. Thus, expansion by minors is a systematic way to calculate determinants of matrices, however, it is not the most efficient way to perform such calculation as the number of multiplications involved grows fast with the determinant size. Other, more efficient, approaches for calculating determinants will be presented in this chapter.

NOTE: Determinant calculation using SMath Studio
Function *det*( ) allows you to find the determinant of a matrix in SMath Studio as illustrated below:

$$A := \begin{pmatrix} 3 & -2 & 5 \\ -1 & 2 & 4 \\ 3 & 2 & -5 \end{pmatrix}$$

type: det(A)=

|A|=-108

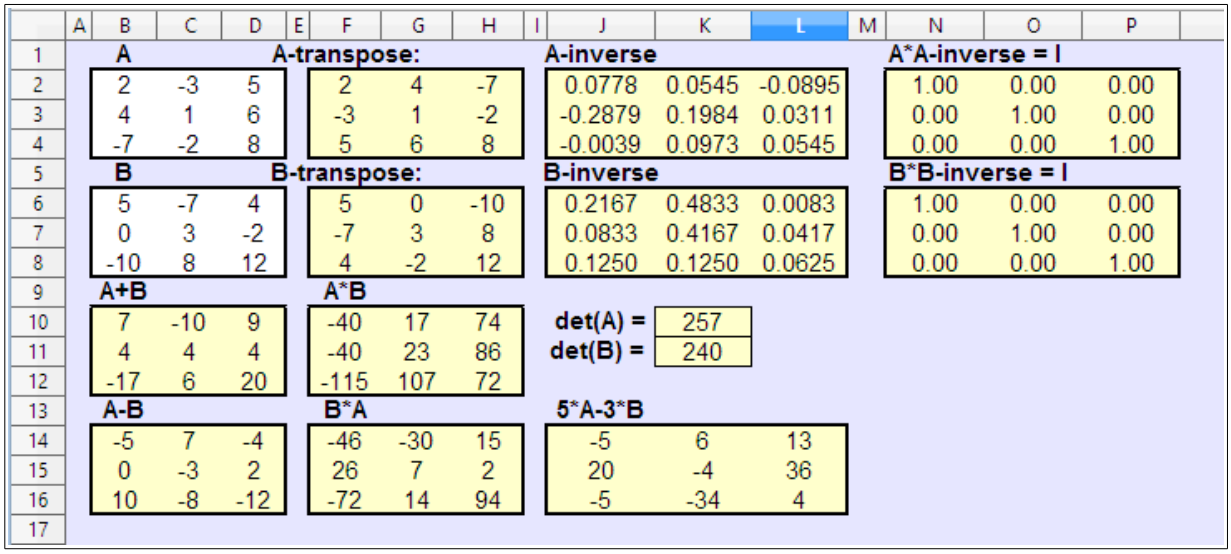
Array operations and functions in EXCEL
Rectangular ranges of data in a spreadsheet can be treated as matrices or two-dimensional arrays. Many of the operations defined above for matrices (addition, subtraction, multiplication by a scalar, linear combination, matrix multiplication, transpose, inverse, determinant) can be performed directly in EXCEL by using array operations.

In general, to perform an array operation you need to have one or more array operands, i.e., rectangular arrays of data in the worksheet, and a single cell or a rectangular collection of cells to produce the output of the array operation. (1) Select the output cell or rectangular array of cells, and (2) type the operation to be performed in the spreadsheet's input bar. Then, (3) use the keyboard combination: CNTL+SHIFT+ENTER. **NOTE**: do not simply press [ENTER] for it may hang up your EXCEL spreadsheet.

A listing of EXCEL Array Functions follows. The operation of these functions is summarized below:

- FREQUENCY: returns a frequency distribution
- GROWTH: for exponential regression
- LINEST: for linear regression
- LOGEST: for logarithmic regression
- MDETERM: determinant of a matrix
- MINVERSE: inverse of a matrix
- MMULT: matrix multiplication
- SUMPRODUCT: inner product, scalar product, or dot product of two vectors
- SUMX2MY2: sum of the differences of the squares of two arrays
- SUMX2PY2 : square sum of two arrays
- SUMXMY2: sum of squares of differences of two arrays
- TRANSPOSE: matrix transpose
- TREND: calculate points along a regression line

Herein we'll illustrate only the use of those array functions related to matrices, namely, MDETERM, MINVERSE, MMULT, and TRANSPOSE, using the examples shown in the following figure:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1		**A**				**A-transpose:**				**A-inverse**				**A*A-inverse = I**		
2		2	-3	5		2	4	-7		0.0778	0.0545	-0.0895		1.00	0.00	0.00
3		4	1	6		-3	1	-2		-0.2879	0.1984	0.0311		0.00	1.00	0.00
4		-7	-2	8		5	6	8		-0.0039	0.0973	0.0545		0.00	0.00	1.00
5		**B**				**B-transpose:**				**B-inverse**				**B*B-inverse = I**		
6		5	-7	4		5	0	-10		0.2167	0.4833	0.0083		1.00	0.00	0.00
7		0	3	-2		-7	3	8		0.0833	0.4167	0.0417		0.00	1.00	0.00
8		-10	8	12		4	-2	12		0.1250	0.1250	0.0625		0.00	0.00	1.00
9		**A+B**				**A*B**										
10		7	-10	9		-40	17	74		**det(A) =**	257					
11		4	4	4		-40	23	86		**det(B) =**	240					
12		-17	6	20		-115	107	72								
13		**A-B**				**B*A**				**5*A-3*B**						
14		-5	7	-4		-46	-30	15		-5	6	13				
15		0	-3	2		26	7	2		20	-4	36				
16		10	-8	-12		-72	14	94		-5	-34	4				
17																

Figure 1. Examples of array operations and functions in EXCEL.

The operations shown in Figure 1, above, are calculated as follows:

- Enter matrices **A** and **B** in ranges B2:D4 and B6:D8, as shown in Figure 1.
- **A+B**: select B10:D12, type "=B2:D4+B6:D8", CNTL+SHIFT+ENTER
- **A-B**: select B14:D16, type "=B2:D4-B6:D8", CNTL+SHIFT+ENTER
- **A-transpose**: select F2:H4, type "=TRANSPOSE(B2:D4)", CNTL+SHIFT+ENTER
- **B-transpose**: select F6:H8, type "=TRANSPOSE(B6:D8)", CNTL+SHIFT+ENTER
- **A*B**: select F10:H12, type "=MMULT(B2:D4,B6:D8)", CNTL+SHIFT+ENTER
- **B*A**: select F14:H16, type "=MMULT(B6:D8,B2:D4)", CNTL+SHIFT+ENTER
- **A-inverse**: select J2:L4, type "=MINVERSE(B2:D4)", CNTL+SHIFT+ENTER

- **B-inverse**: select J6:L8, type "=MINVERSE(B6:D8)", CNTL+SHIFT+ENTER
- **det(A)**: select K10, type "=MDETERM(B2:D4)", CNTL+SHIFT+ENTER
- **det(B)**: select K11, type "=MDETERM(B6:D8)", CNTL+SHIFT+ENTER
- **5*A-3*B**: select J14:L16, type "=5*B2:D4-3*B6:D8", CNTL+SHIFT+ENTER
- **A*A-inverse = I** : select N2:P4, type "=MMULT(B2:D4,J2:L4), CNTL+SHIFT+ENTER
- **B*B-inverse = I** : select N6:P8, type "=MMULT(B6:D8,J6:L8), CNTL+SHIFT+ENTER

When multiplying matrices of different sizes, e.g., $\boldsymbol{C}_{4\times3}=\boldsymbol{A}_{4\times2}\cdot\boldsymbol{B}_{2\times3}$, care should be taken to select an output range of the right dimensions (2 x 3, in this case).

Some applications of array functions in EXCEL are illustrated next.

Solving a system of linear equations using Cramer's rule
It was indicated earlier that a system of linear equations such as:

$$\begin{aligned} -2\cdot x+5\cdot y+7\cdot z&=26\\ 3\cdot x-5\cdot y+z&=25\\ x+10\cdot y-5\cdot z&=-47 \end{aligned}$$

can be written as a matrix equation **A x** = **b**, with

$$\boldsymbol{A}=\begin{bmatrix}-2 & 5 & 7\\ 3 & -5 & 1\\ 1 & 10 & -5\end{bmatrix}\ ,\quad \boldsymbol{x}=\begin{bmatrix}x\\ y\\ z\end{bmatrix}\ ,\text{ and }\quad \boldsymbol{b}=\begin{bmatrix}26\\ 25\\ -47\end{bmatrix}\ .$$

Cramer's rule is a way to solve the system of equations by using determinants. Let |**A**| be the determinant of the matrix of coefficients **A**. Let matrix $\mathbf{A}_x$ be a matrix formed by replacing vector **b** in the first column of matrix **A**. For this case, thus,

$$\boldsymbol{A}_x=\begin{bmatrix}26 & 5 & 7\\ 25 & -5 & 1\\ -47 & 10 & -5\end{bmatrix}\ .$$

The unknown *x* is thus, calculated as:

$$x=\frac{det\left(\boldsymbol{A}_x\right)}{det\left(\boldsymbol{A}\right)}=\frac{\left|\boldsymbol{A}_x\right|}{\left|\boldsymbol{A}\right|}\ .$$

In a similar fashion, we form matrix $\mathbf{A}_y$ by replacing the second column in the matrix of coefficients **A** with vector **b**, and matrix $\mathbf{A}_z$ by replacing the third column of matrix **A** with vector **b**. For the present case we'll have then,

$$A_y = \begin{bmatrix} -2 & 26 & 7 \\ 3 & 25 & 1 \\ 1 & -47 & -5 \end{bmatrix} \text{, and } A_z = \begin{bmatrix} -2 & 5 & 26 \\ 3 & -5 & 25 \\ 1 & 10 & -47 \end{bmatrix} .$$

The unknowns y and z are then calculated as follows:

$$y = \frac{det(A_y)}{det(A)} = \frac{|A_y|}{|A|} \quad , \quad z = \frac{det(A_z)}{det(A)} = \frac{|A_z|}{|A|} .$$

Using the EXCEL spreadsheet and array functions *MDETERM* and *MMULT* we find the solution for this case as illustrated below:

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2		**A**	det(A):	295		**Ax**	det(Ax):	885		**b**		**check:**
3		-2	5	7		26	5	7		26		26
4		3	-5	1		25	-5	1		25		25
5		1	10	-5		-47	10	-5		-47		-47
6		**Ay**	det(Ay):	-590		**Az**	det(Az):	1770		**x**		**A*x**
7		-2	26	7		-2	5	26	x:	3		
8		3	25	1		3	-5	25	y:	-2		
9		1	-47	-5		1	10	-47	z:	6		
10												

Figure 2. Solution of a system of 3 linear equations using Cramer's rule and array functions in EXCEL. Details of the calculations are shown next:

Enter input data as follows:

- Matrix **A** in B3:D5
- Matrix **Ax** in F3:H5
- Matrix **Ay** in B7:D9
- Matrix **Az** in F7:H9
- Vector **b** in J3:J5

Calculate determinants as follows:

- **det(A)** : Select D2, type "=MDETERM(B3:D5), CNTL+SHIFT+ENTER
- **det(Ax)**: Select H2, type "=MDETERM(F3:H5), CNTL+SHIFT+ENTER
- **det(Ay)**: Select D6, type "=MDETERM(B7:D9), CNTL+SHIFT+ENTER
- **det(Az)**: Select H6, type "=MDETERM(F7:H9), CNTL+SHIFT+ENTER

Calculate solutions as follows:

- **x**: Select J7, type "=H2/D2", ENTER
- **y**: Select J8, type "=D6/D2", ENTER
- **z**: Select J9, type "=H6/D2", ENTER

To check that the solution found does satisfy the matrix equation **A x = b**, calculate the following matrix multiplication:

- **check:** Select L3:L5, type "=MMULT(B3:D5,J7:J9)", CNTL+SHIFT+ENTER

Solving a system of linear equations using the inverse of the matrix of coefficients
The system of linear equations used before:

$$\begin{aligned} -2\cdot x+5\cdot y+7\cdot z&=26\\ 3\cdot x-5\cdot y+z&=25\\ x+10\cdot y-5\cdot z&=-47 \end{aligned}$$

resulted in the matrix equation **A x** = **b**, with

$$A=\begin{bmatrix}-2 & 5 & 7\\ 3 & -5 & 1\\ 1 & 10 & -5\end{bmatrix},\quad x=\begin{bmatrix}x\\ y\\ z\end{bmatrix},\text{ and }\quad b=\begin{bmatrix}26\\ 25\\ -47\end{bmatrix}.$$

Using the properties of the inverse matrix, namely, $\mathbf{A}^{-1}\,\mathbf{A}=\mathbf{I}$, the matrix equation **A x = b**, when multiplied by $\mathbf{A}^{-1}$, results in

$$\mathbf{A}^{-1}\,\mathbf{A}\,\mathbf{x}=\mathbf{A}^{-1}\,\mathbf{b}.$$

Also, we know that $\mathbf{A}^{-1}\,\mathbf{A}=\mathbf{I}$, so that we can write the previous equation as

$$\mathbf{I}\,\mathbf{x}=\mathbf{A}^{-1}\,\mathbf{b},$$

and since **I x = x**, we find the solution to the system to be $\mathbf{x}=\mathbf{A}^{-1}\,\mathbf{b}$. This solution is implemented in EXCEL as illustrated below.

	A	B	C	D	E	F	G	H	I	J
1		A				b		check A*x = b		
2		-2	5	7		26		26		
3		3	-5	1		25		25		
4		1	10	-5		-47		-47		
5		A^(-1)				x		check A*A^(-1) = I		
6		0.0508	0.322	0.1356		3		1.00	0.00	0.00
7		0.0542	0.0102	0.078		-2		0.00	1.00	0.00
8		0.1186	0.0847	-0.0169		6		0.00	0.00	1.00
9										

Figure 3. Solution of a system of 3 linear equations using matrix inverse array functions in EXCEL.

The solution proceeds as follows:

Enter matrix **A** and vector **b**:

- Matrix **A** in B2:D4
- Vector **b** in F2:F4

Calculate solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$:

- **A^(-1):** Select B6:D8, type "=MINVERSE(B2:D4)", CNTL+SHIFT+ENTER
- **x:** Select F6:F8, type "=MMULT(B6:D8,F2:F4)", CNTL+SHIFT+ENTER

Check solution and inverse matrix properties:

- **check A*x=b:** Select H2:H4, type "=MMULT(B2:D4,F6:F8)", CNTL+SHIFT+ENTER
- **check A*A^(-1)=I:** Select H6:J8, type "=MMULT(B2:D4,B6:D8)", CNTL+SHIFT+ENTER

In general, using the inverse matrix is more efficient than using Cramer's rule, since the amount of information required is smaller (one matrix **A** and one vector **b**, instead of several matrices **A**, $\mathbf{A}_x$, etc., for Cramer's rule). Thus, the method illustrated in this example is recommended, in general, for solving systems of linear equations.

When there is no solution – singular matrices – Let **A** be the *n* x*n* matrix of coefficients corresponding to a system of *n* linear equations with *n* unknowns. If the determinant of **A** is zero (det(**A**) = 0, or |**A**| = 0), the matrix is said to be *singular*, and no solution exists for the system of equations. This can be explained by recalling that Cramer's rule requires us to divide by det(**A**) to calculate solutions to the system. Obviously, if det(**A**) = 0, no solution exists for the system. The following figure illustrates what happens if we try to obtain the inverse of a singular matrix in the EXCEL spreadsheet.

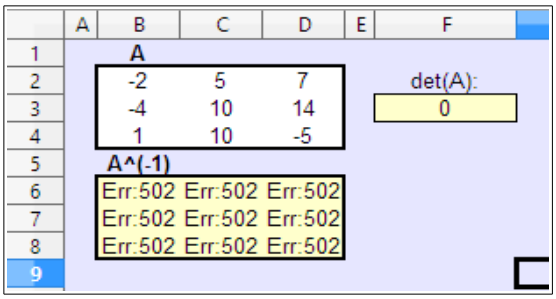


Figure 4. Determinant and inverse of a singular matrix using array functions in EXCEL.

In this case, the operations used were the following:

- Enter matrix **A** in B2:D4
- To calculate determinant, select F3, type "=MDETERM(B2:D4)", CNTL+SHIFT+ENTER
- To try calculating the inverse, select B6:D8, type = "MINVERSE(B2,D4)", CNTL+SHIFT+ENTER

Figure 3 shows that the determinant of the matrix $\boldsymbol{A} = \begin{bmatrix} -2 & 5 & 7 \\ -4 & 10 & 14 \\ 1 & 10 & -5 \end{bmatrix}$ is zero. It also shows that attempting to calculate the inverse $\mathbf{A}^{-1}$ for matrix **A** produces an error (Err:502).

Typically, a matrix that has a row or column whose elements are all zero, will produce a singular matrix. Also, if two rows (or two columns) are the same, or one multiple of the other, the matrix is singular. For example, singular matrix **A**, shown above, is such that the second row [-4 10 14] is twice the first row [-2 5 7].

Two-dimensional (2D) arrays in VBA

One dimensional arrays were introduced in Chapter 3. In this chapter we introduce the use of two-dimensional (2D) arrays in VBA. While one-dimensional arrays can be thought of row or column vectors, two-dimensional arrays in VBA can be interpreted as representing matrices.

Two-dimensional arrays in VBA can be declared using variable names with two indices, separated by commas, in a *Dim* statement. For example:

```
Dim A(1 to 10, 5 To 8) As Double
Dim ux(0 to 10, 0 to 100) As Double
Dim Temp(1 to 5, 1 To 10) As Integer
```

If the statement `Option Base 1` is active, then we only need to declare the upper limit of the indices, understanding that the first value of the indices will be 1. If we want to use zero (0) as the starting index value, use the statement `Option Base 0`. In the following examples, all indices have default starting value of 1.

```
Option Base 1
Dim A(10,3) As Double
Dim ux(11,101) As Double
Dim Temp(5,10) As Integer
```

The *Redim* statement can be used to redimension an array as a two-dimensional one. As in the case of one-dimensional arrays, a dynamic array must be declared as so by using the array name followed by empty parentheses, before redimensioning can take place. Consider the following example:

```
Option Base 1
Dim A() As Double
…
n = 3 : m = 5
ReDim A(n,m)
```

In this case, array *A* is declared as a dynamic array by using *Dim A*( ) *As Double*, and redimensioned as a 3x5 two-dimensional array by using *ReDim A*(*n*,*m*), after *n* and *m* are given values of 3 and 5, respectively.

Individual elements of an array can be assigned values through assignment statements, and used in operations, as illustrated in the following code:

```
Option Base 1
Dim A(10,10) As Double, B(5,5) As Double, C(5,5) As Double
Dim i As Integer, j As Integer
...
A(2,3) = 5.0 : B(5,4) = -10.0
C(3,5) = A(2,3) + B(5,4)
...
For i = 1 To 5
      For j = 1 To 5
            A(i,j) = i+j
            B(i,j) = i-j
            C(i,j) = A(i,j) + B(i,j)
     Next
Next
```

VBA code for selected matrix operations using two-dimensional arrays

Suppose that we have loaded a couple of two-dimensional arrays in memory, say, A and B, representing matrices **A** and **B**, respectively. If the arrays are of the same size, say, *m* x *n*, (i.e., *m* rows and *n* columns), then the operations of addition (**C = A+B**), subtraction (**D = A-B**), and linear combination (**E = *k A* + *s*B**) of matrices can be performed as illustrated in the following code:

```
Option Base 1
Dim A(10,5) As Double, B(10,5) As Double, C(10,5) As Double
Dim D(10,5) As Double, E(10,5) As Double
Dim i As Integer, j As Integer
…

'Addition
For i = 1 To 10
    For j = 1 To 5
        C(i,j) = A(i,j) + B(i,j)
    Next
Next
'Subtraction
For i = 1 To 10
    For j = 1 To 5
        C(i,j) = A(i,j) - B(i,j)
    Next
Next
'Linear combination
For i = 1 To 10
    For j = 1 To 5
        C(i,j) = k*A(i,j) + s*B(i,j)
    Next
Next
```

The multiplication of two matrices **A** and **B**, say **C = A×B**, requires that the number of columns of **A** be equal to the number of rows of **B**. This can be summarized by using this equation showing the dimensions of the three matrices: $\boldsymbol{C}_{m\times n} = \boldsymbol{A}_{m\times p} \cdot \boldsymbol{B}_{p\times n}$.

If that condition is fulfilled, then the elements of the product matrix can be calculated as:

$$c_{i,j} = \sum_{k=1}^{p} a_{i,k} \cdot b_{k,j} \quad \text{, for } i = 1, 2, \ldots, m; \text{ and } j = 1, 2, .., n$$

The code required for matrix multiplication is shown next (using `Option Base 1`):

```
Dim m As Integer, n As Integer, p As Integer, i As Integer, j As Integer
Dim A(m,p) As Double, B(p,n) As Double, C(m,n) As Double
…
For i = 1 to m
    For j = 1 To n
        C(i,j) = 0.0
        For k = 1 to p
            C(i,j) = C(i,j) + A(i,k)*B(k,j)
        Next
    Next
Next
```

The solution of a system of linear equations requires multiplying an nxn matrix, say A, with a nx1 vector, say b. If the vector **b** is defined in code as a one-dimensional vector, then the matrix multiplication of $\boldsymbol{c}_{n\times 1} = \boldsymbol{A}_{n\times n} \cdot \boldsymbol{b}_{n\times 1}$, would be defined by

$$c_i = \sum_{k=1}^{p} a_{i,k} \cdot b_k \quad \text{, for } i = 1, 2, .., n,$$

and the VBA code would look as follows (using `Option Base 1`):

```
Dim n As Integer, i As Integer
Dim A(n,n) As Double, b(n) As Double, c(n) As Double
…
For i = 1 to n
    c(i) = 0.0
    For k = 1 to n
        c(i) = c(i) + A(i,k)*b(k)
    Next
Next
```

The code for calculating the transpose matrix $\mathbf{B}_{nxm} = \mathbf{A}^T$ of a matrix $\mathbf{A}_{mxn}$ is based on the fact that $b_{ij} = a_{ji}$. The following code takes of producing the transpose:

```
For i = 1 To n
    For j = 1 To m
        B(i,j) = A(j,i)
    Next
Next
```

Interface and code for matrix addition, subtraction, and multiplication

Suppose you want to create an interface where you will enter the elements of a matrix A and of a matrix

B, whose number of rows and columns are not known beforehand. One simple way to set up the interface is as shown in the figure below.

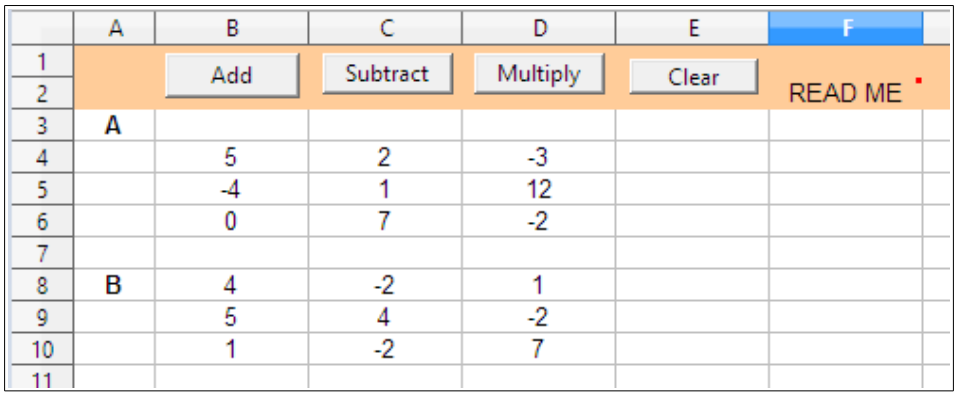


Figure 5. Example of a spreadsheet interface for matrix addition, subtraction, and multiplication.

In the interface of Figure 5, the first column is used to identify the matrices **A** and **B**. Matrix **A** is entered starting with its upper left corner in cell B4. After the last row of matrix **A** is entered, a blank line is left, and then matrix **B** is entered with its upper left corner starting (in the case of Figure 5) in cell B8. The key in entering the elements of matrices **A** and **B** in the interface, is to always start matrix **A** in cell B4, while matrix **B** is entered after leaving a blank line at the bottom of matrix **A**. As a matter of fact, for informational purposes, one could add a *comment* to a cell (e.g., cell F2, entitled "READ ME") to explain the required format for entering matrices **A** and **B**. The comment may look as follows:

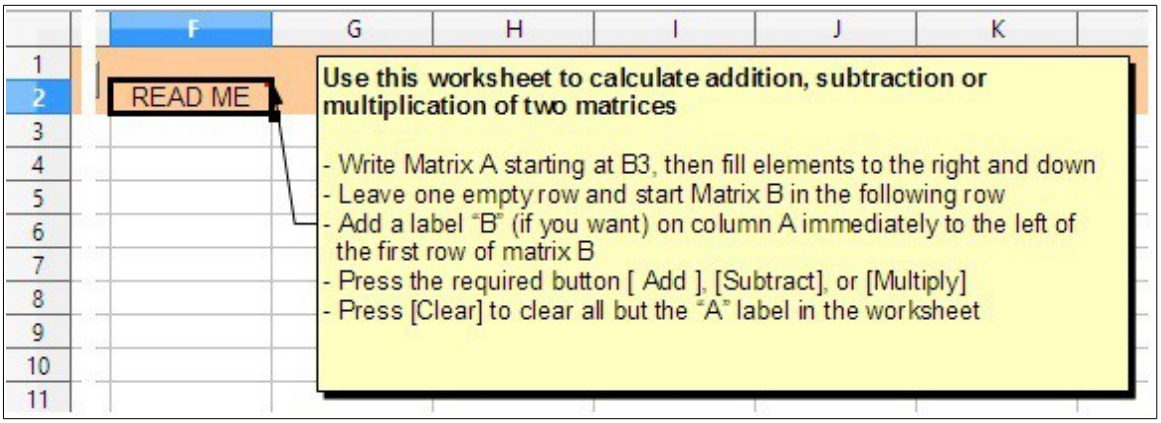


Figure 6. Comment with instructions for using the matrix operations interface of Figure 5.

The code required to enter the data in matrices **A** and **B** needs to perform the following:

- Count the number of columns in matrix **A** by identifying the first cell to the right of the first row that is empty (*mA*)
- Count the number of rows in matrix **A** by identifying the first cell below the first cell at the bottom of the first column that is empty (*nA*) – hence the need to leave one row empty between matrices **A** and **B**

- Starting at the position of the upper left corner of matrix **B**, count number of columns of matrix **B** (*mB*) using an approach similar to the one used above for matrix **A**
- Starting at the position of the upper left corner of matrix **B**, count number of rows of matrix **B** (*nB*) using an approach similar to the one used above for matrix **A**
- Check that the dimensions of matrices **A** (*mA, nA*) and **B** (*mB*,*nB*) are compatible for the operation intended, i.e.,
  - *mA* = *mB* and *nA*= *nB* for addition and subtraction
  - *nA* = *mB* for multiplication

  If the dimensions are not compatible, a message should be send to the user indicating so, and the process stopped at this point

- If the dimensions are compatible for the required operation, we proceed to enter the data from matrices **A** and **B** using the *Range* and *Cells* properties of the worksheet, as illustrated below.

The code for entering data for matrices **A** and **B** is shown below. The worksheet where the interface is located is called *Matrix01*:

```
Dim A() As Double, B() As Double, C() As Double
Dim k As Integer, i As Integer, j As Integer
Dim nA As Integer, mA As Integer '- dimensions of matrix A
Dim nB As Integer, mB As Integer '- dimensions of matrix B
Dim nC As Integer, mC As Integer '- dimensions of matrix C = A+B or C = A-
nA = 0 : mA = 0   ' Check size of matrix A(mA x nA)
For k = 1 To 100
    If Worksheets("Matrix01").Range("B3").Cells(1,k).Value <> "" Then
       nA = nA + 1
    Else
       Exit For
    End If
Next
For k = 1 To 100
    If Worksheets("Matrix01").Range("B3").Cells(k,1).Value <> "" Then
       mA = mA + 1
    Else
       Exit For
    End If
Next
nB = 0 : mB = 0    ' Check size of matrix B(mB x nB)
For k = 1 To 100
    If Worksheets("Matrix01").Range("B3").Cells(mA+2,k).Value <> "" Then
       nB = nB + 1
    Else
       Exit For
    End If
Next k
For k = 1 To 100
    If Worksheets("Matrix01").Range("B3").Cells(mA+1+k,1).Value <> "" Then
```

```
        mB = mB + 1
     Else
        Exit For
     End If
Next k
If ((mA = mB) And (nA = nB)) Then 'Check that A and B have same size
   mC = mA : nC = nA
   'Redimension matrices
   Redim A(mA, nA)
   Redim B(mB, nB)
   Redim C(mC, nC)
   For i = 1 to mA  'Read data for matrix A
       for j = 1 to nA
           A(i,j) = _
           Worksheets("Matrix01").Range("B3").Cells(i,j).Value
       Next
   Next
   'Read data for matrix B
   For i = 1 to mB
       for j = 1 to nB
           B(i,j) = _
           Worksheets("Matrix01").Range("B3").Cells(mA+1+i,j).Value
       Next
   Next
   '… addition or subtraction operations go here
Else
   '… let user know matrices are incompatible
End If
```

Detailed description of code for the buttons of the interface in Figure 5
The interface of Figure 5 includes the following buttons:

- [Add]: Matrix addition, **C = A+B**, associated with macro *AddTwoMatrices*( )
- [Subtract] : Matrix subtraction, **C = A-B**, associated with macro *SubtractTwoMatrices*( )
- [Multiply] : Matrix multiplication, **C = AB**, associated with macro *MultiplyTwoMatrices*( )
- [Clear]: Clear input and output in the interface, associated with macro *ClearEntries*( )

The codes for macros *AddTwoMatrices*, *SubtractTwoMatrices*, and *MultiplyTwoMatrices*, are shown next. The general outline of each of the codes is as follows:

- Define matrices A, B, and C as dynamic arrays
- Determine the dimensions of matrices A (*mA, nA*), B (*mB*, *nB*)
- Check if matrices have compatible dimensions, if so, continue, if not, let the user know about the incompatible dimensions and stop process
- If matrix dimensions are compatible, redimension the matrices with the proper dimensions, and read in matrices A and B
- Perform required operation **C = A+B**, **C = A-B**,, or **C = AB**, and produce output to interface. A blank line is left immediately after matrix **B** before outputting matrix **C**, and the proper label is

added to the interface

The following is the code for macro *AddTwoMatrices* associated with the [Add] button in Figure 5:

```
'================================================================================
Option Explicit
Option Base 1
'================================================================================
Sub AddTwoMatrices()
    Dim A() As Double, B() As Double, C() As Double
    Dim k As Integer, i As Integer, j As Integer
    Dim nA As Integer, mA As Integer
    Dim nB As Integer, mB As Integer
    Dim nC As Integer, mC As Integer
    ' Check size of matrix A(mA x nA)
    nA = 0 : mA = 0
    For k = 1 To 100
        If Worksheets("Matrix01").Range("B3").Cells(1,k).Value <> "" Then
           nA = nA + 1
        Else
           Exit For
        End If
    Next k
    For k = 1 To 100
        If Worksheets("Matrix01").Range("B3").Cells(k,1).Value <> "" Then
           mA = mA + 1
        Else
           Exit For
        End If
    Next k
    ' Check size of matrix B(mB x nB)
    nB = 0 : mB = 0
    For k = 1 To 100
        If Worksheets("Matrix01").Range("B3").Cells(mA+2,k).Value <> "" Then
           nB = nB + 1
        Else
           Exit For
        End If
    Next k
    For k = 1 To 100
        If Worksheets("Matrix01").Range("B3").Cells(mA+1+k,1).Value <> "" Then
           mB = mB + 1
        Else
           Exit For
        End If
    Next k
    If ((mA = mB) And (nA = nB)) Then 'Check that A and B have same size
       mC = mA : nC = nA

       'Redimension matrices
       Redim A(mA, nA)
       Redim B(mB, nB)
```



```
       Redim C(mC, nC)

       'Read data for matrix A
       For i = 1 to mA
           for j = 1 to nA
               A(i,j) = _
               Worksheets("Matrix01").Range("B3").Cells(i,j).Value
           Next
       Next
       'Read data for matrix B
       For i = 1 to mB
           for j = 1 to nB
               B(i,j) = _
               Worksheets("Matrix01").Range("B3").Cells(mA+1+i,j).Value
           Next
       Next
       'Add matrices A + B = C, show results
       Worksheets("Matrix01").Range("A3").Cells(mA+mB+3,1).Value = "C"
       For i = 1 to mC
           for j = 1 to nC
               C(i,j) = A(i,j) + B(i,j)
               Worksheets("Matrix01").Range("B3").Cells(mA+mB+2+i,j).Value = _
                    C(i,j)
           Next
       Next
       'Redo A and B labels -- not needed, but just in case
       Worksheets("Matrix01").Range("A3").Cells(1,1).Value = "A"
       Worksheets("Matrix01").Range("A3").Cells(nA+2,1).Value = "B"
   Else
       MsgBox("Matrices are of different size. " & Chr(13) & _
              "Addition cannot be performed.")
   End If
End Sub
```

Figure 7. Code for matrix addition (*AddTwoMatrices*) associated with button [Add] in the interface of Figure 5.

The following is the code for the macro *SubtractTwoMatrices* associated with the [Subtract] button in the interface of Figure 5:

```
Sub SubtractTwoMatrices()
    Dim A() As Double, B() As Double, C() As Double
    Dim k As Integer, i As Integer, j As Integer
    Dim nA As Integer, mA As Integer
    Dim nB As Integer, mB As Integer
    Dim nC As Integer, mC As Integer
    ' Check size of matrix A(mA x nA)
    nA = 0 : mA = 0
    For k = 1 To 100
        If Worksheets("Matrix01").Range("B3").Cells(1,k).Value <> "" Then
```



```
        nA = nA + 1
    Else
        Exit For
    End If
Next k
For k = 1 To 100
    If Worksheets("Matrix01").Range("B3").Cells(k,1).Value <> "" Then
        mA = mA + 1
    Else
        Exit For
    End If
Next k
' Check size of matrix B(mB x nB)
nB = 0 : mB = 0
For k = 1 To 100
    If Worksheets("Matrix01").Range("B3").Cells(mA+2,k).Value <> "" Then
        nB = nB + 1
    Else
        Exit For
    End If
Next k
For k = 1 To 100
    If Worksheets("Matrix01").Range("B3").Cells(mA+1+k,1).Value <> "" Then
        mB = mB + 1
    Else
        Exit For
    End If
Next k
If ((mA = mB) And (nA = nB)) Then 'Check that A and B have same size
   mC = mA : nC = nA
   'Redimension matrices
   Redim A(mA, nA)
   Redim B(mB, nB)
   Redim C(mC, nC)
   'Read data for matrix A
   For i = 1 to mA
       for j = 1 to nA
           A(i,j) = _
           Worksheets("Matrix01").Range("B3").Cells(i,j).Value
       Next
   Next
   'Read data for matrix B
   For i = 1 to mB
       for j = 1 to nB
           B(i,j) = _
           Worksheets("Matrix01").Range("B3").Cells(mA+1+i,j).Value
       Next
   Next
   'Subtract matrices A - B = C, show results
   Worksheets("Matrix01").Range("A3").Cells(mA+mB+3,1).Value = "C"
   For i = 1 to mC
       for j = 1 to nC
           C(i,j) = A(i,j) - B(i,j)
```

```
                Worksheets("Matrix01").Range("B3").Cells(mA+mB+2+i,j).Value = _
                          C(i,j)
            Next
        Next

        'Redo A and B labels -- not needed, but just in case
        Worksheets("Matrix01").Range("A3").Cells(1,1).Value = "A"
        Worksheets("Matrix01").Range("A3").Cells(nA+2,1).Value = "B"
    Else
        MsgBox("Matrices are of different size. " & Chr(13) & _
               "Subtraction cannot be performed.")
    End If
End Sub
```

Figure 8. Code for matrix addition (*SubtractTwoMatrices*) associated with button [Subtract] in the interface of Figure 5.

The following is the code for the macro *MultiplyTwoMatrices* associated with the [Multiply] button in the interface of Figure 5:

```
Sub MultiplyTwoMatrices()
    Dim A() As Double, B() As Double, C() As Double
    Dim k As Integer, i As Integer, j As Integer
    Dim nA As Integer, mA As Integer
    Dim nB As Integer, mB As Integer
    Dim nC As Integer, mC As Integer
    ' Check size of matrix A(mA x nA)
    nA = 0 : mA = 0
    For k = 1 To 100
        If Worksheets("Matrix01").Range("B3").Cells(1,k).Value <> "" Then
           nA = nA + 1
        Else
           Exit For
        End If
    Next k
    For k = 1 To 100
        If Worksheets("Matrix01").Range("B3").Cells(k,1).Value <> "" Then
           mA = mA + 1
        Else
           Exit For
        End If
    Next k
    ' Check size of matrix B(mB x nB)
    nB = 0 : mB = 0
    For k = 1 To 100
        If Worksheets("Matrix01").Range("B3").Cells(mA+2,k).Value <> "" Then
           nB = nB + 1
        Else
           Exit For
        End If
    Next k
```

```
    For k = 1 To 100
        If Worksheets("Matrix01").Range("B3").Cells(mA+1+k,1).Value <> "" Then
           mB = mB + 1
        Else
           Exit For
        End If
    Next k
    If nA = mB Then 'Check that A and B have same size
       mC = mA : nC = nB
       'Redimension matrices
       Redim A(mA, nA)
       Redim B(mB, nB)
       Redim C(mC, nC)
       'Read data for matrix A
       For i = 1 to mA
           for j = 1 to nA
               A(i,j) = _
               Worksheets("Matrix01").Range("B3").Cells(i,j).Value
           Next
       Next

       'Read data for matrix B
       For i = 1 to mB
           for j = 1 to nB
               B(i,j) = _
               Worksheets("Matrix01").Range("B3").Cells(mA+1+i,j).Value
           Next
       Next
       'Multiply matrices A*B = C, show results
       Worksheets("Matrix01").Range("A3").Cells(mA+mB+3,1).Value = "C"
       For i = 1 to mC
           for j = 1 to nC
               C(i,j) = 0.0
               For k = 1 to nA
                   C(i,j) = C(i,j) + A(i,k)*B(k,j)
               Next
               Worksheets("Matrix01").Range("B3").Cells(mA+mB+2+i,j).Value = _
                         C(i,j)
           Next
       Next
       'Redo A and B labels -- not needed, but just in case
       Worksheets("Matrix01").Range("A3").Cells(1,1).Value = "A"
       Worksheets("Matrix01").Range("A3").Cells(nA+2,1).Value = "B"
    Else
       MsgBox("Matrices are of incompatible sizes. " & Chr(13) & _
              "Multiplication cannot be performed.")
    End If
End Sub
```

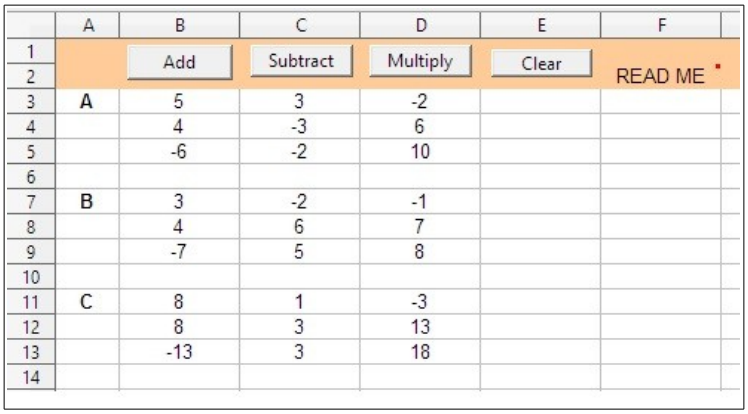
Figure 9. Code for matrix multiplication (*MultiplyTwoMatrices*) associated with button [Multiply] in the interface of Figure 5.

The following is the code for the macro *ClearEntries* associated with the [Clear] button in the interface of Figure 5. It assumes that the largest size for matrices A, B, and C is 50x50 elements:

```
Sub ClearEntries()
    Dim i As integer, j As integer
    For i = 2 To 50
        Worksheets("Matrix01").Range("A3").Cells(i,1).Value = ""
    Next
    For i = 1 to 100
        For j = 1 to 102
           Worksheets("Matrix01").Range("B3").Cells(i,j).Value = ""
        Next
    Next
End Sub
```

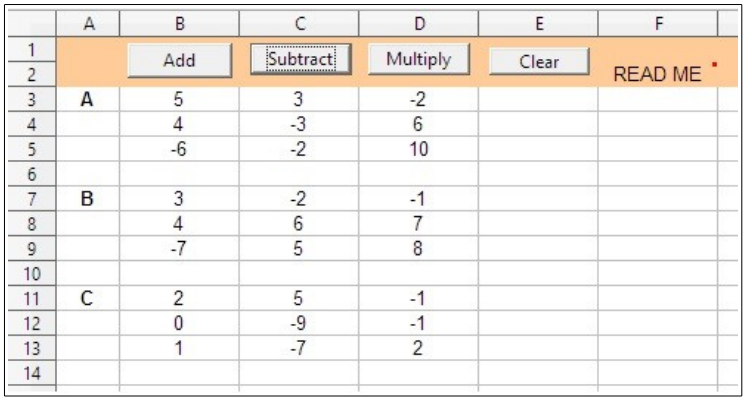
Figure 10. Code for clearing entries in the interface of Figure 5 (*ClearEntries*). This code is associated with button [Clear] in the interface.

The following figures show examples of addition, subtraction, and multiplication of matrices:



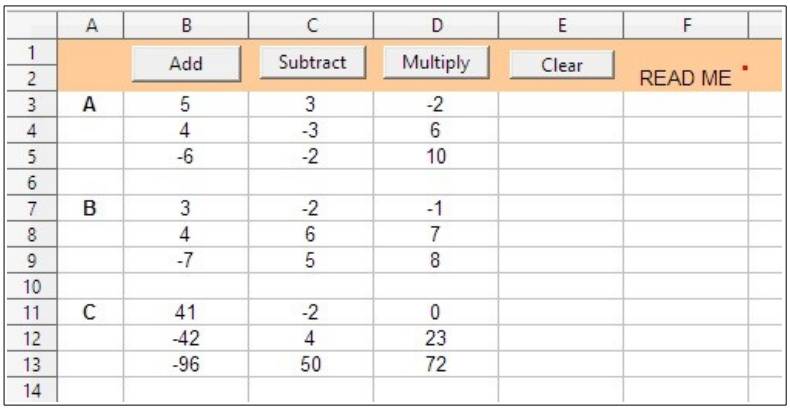
	A	B	C	D	E	F
1						
2						
3	A	5	3	-2		
4		4	-3	6		
5		-6	-2	10		
6						
7	B	3	-2	-1		
8		4	6	7		
9		-7	5	8		
10						
11	C	8	1	-3		
12		8	3	13		
13		-13	3	18		
14						

Figure 11. Example of matrix addition using the interface of Figure 5.



	A	B	C	D	E	F
1						
2						
3	A	5	3	-2		
4		4	-3	6		
5		-6	-2	10		
6						
7	B	3	-2	-1		
8		4	6	7		
9		-7	5	8		
10						
11	C	2	5	-1		
12		0	-9	-1		
13		1	-7	2		
14						

Figure 12. Example of matrix subtraction using the interface of Figure 5.



	A	B	C	D	E	F
1						
2						
3	A	5	3	-2		
4		4	-3	6		
5		-6	-2	10		
6						
7	B	3	-2	-1		
8		4	6	7		
9		-7	5	8		
10						
11	C	41	-2	0		
12		-42	4	23		
13		-96	50	72		
14						

Figure 13. Example of matrix multiplication using the interface of Figure 5.

VBA code for selected matrix operations using EXCEL array functions – *Range variables*

The examples in the previous section dealt with selected matrix operations (addition, subtraction, multiplication, and matrix transpose) using two-dimensional arrays and operating in the elements of those arrays. There are more complex matrix operations, such as calculating the determinant of a matrix, or its inverse, or solving systems of linear equations, which require more sofisticated numerical methods. A numerical procedure commonly used to solve systems of linear equations is the so-called *Gaussian elimination*. An extension of Gaussian elimination, called *Gauss-Jordan elimination*, can be used to obtain a matrix inverse, and to calculate the determinant in an efficient way. These methods are typically addressed in textbooks on numerical methods, and due to their complexity, will not be addressed in this book. However, we can take advantage of the array functions mentioned in an earlier section to calculate matrix determinants (MDETERM), matrix inverses (MINVERSE), and solutions to systems of equations (MINVERSE, MMULT) using VBA code. The use of these array functions require us, however, to introduce a new data type, the *Range* data type, into the code.

The* Range *data type and the* Variant *data type

In previous sections in this chapter, and in previous chapters, we have used the *Range*( ) property of the *Worksheets*( ) object in EXCEL for input and output of data. We have also used the *Cells*( ) property of the *Range*( ) property to address individual cells related to a particular range in the EXCEL spreadsheet. A *range* in a EXCEL worksheet can be simply a cell, specified, for example, as *Range*("B3"), or it could encompass contiguous cells in a row, e.g., *Range*("B3:B7"), or in a column, e.g., *Range*("B3:F3"), or even include a rectangular array of cells, such as in *Range*("B3:D7"), etc.

Array operations, as illustrated in an earlier section, utilize ranges as arguments, e.g.,

```
= MMULT(B6:D8,B2:D4)
= MINVERSE(B2:D4)
= MDETERM(F2:H4)
= TRANSPOSE(B12:D14)
```

Thus, if we were to utilize variable names to refer to these ranges within VBA code, we could use statements such as:

```
C = Application.MMULT(A,B)
D = Application.MINVERSE(A)
d = Application.MDETERM(A)
E = Application.TRANSPOSE(A)
```

where A and B would have been defined as *Range* data types. Calling array functions in VBA code require us to use the full reference *Application.<array_function_name>*, which emphasizes that the functions being invoked, e.g., MMULT, MINVERSE, etc., are associated with the application currently in use, namely, EXCEL.

In the examples of array operations shown above, variable *d*, which stores a single (scalar) value, would be declared as a *Double* variable. However, in VBA programs in Excel, the arrays that get stored in variables *C*, *D*, and *E* must be declared as *Variant* data types. A *Variant* data type is used to declare a variable that has not been declared as other data type (e.g., not *Integer*, nor *Double*, nor *String*, nor *Range*, etc.). A *Variant* variable is initialized with the value *Empty*. When you declare a variable as a *Variant*, you can load it with any value, and, thus, the variable would adopt the type of the value you load. For array operations, as in the calculations of *C, D,* and *E*, above, these variables are declared as *Variant*, but stored as *Range* variables.

Defining range variables

To define a variable as a *Range* data type use the *Dim* statement as in the following examples:

```
Dim A As Range, B As Range, C As Range
Dim myRange01 As Range, myRange02 As Range
Dim Range_A As Range, Range_B As Range
```

Assigning range objects to range variables

To assign a range object to a range variable you can use an assignment statement preceded by the particle *Set* (all assignment statements involving ranges in VBA code must be preceded by *Set*). Some examples of assigning range objects to range variables are presented next:

- This statement will set range variable Range_A equal to the range B5:F10 in the current worksheet:

  ```
  Set Range_A = Range("B5:F10")
  ```

- This statement sets range variable Range_B equal to the range C7:E10 in worksheet "Sheet1".

  ```
  Set Range_B = Worksheets("Sheet1").Range("C7:E10")
  ```

- This statement sets range variable Range_C equal to the range whose upper left corner starts at "C7" an encompasses its cell (1,1) through its cell (3,4).

  ```
  Set Range_C = Worksheets("Sheet2").Range("C7.Cells(1,1)", "C7.Cells(3,4)")
  ```

Input and output of matrices using range variables and arrays

By "setting" a range variable equal to a range object in VBA code (as done above), all the information available in the spreadsheet object range of interest is passed onto the VBA code. This information includes cells' colors, font properties, etc., but, more importantly, cells' values. The values contained in the elements of the range can be turned into elements of an array by simply assigning them to the array elements using, for example, two nested loops as illustrated in the following code:

```
Sub InputOutputRange3x3()
     Dim i As Integer, j As Integer
     Dim RangeA As Range
     Dim A(3,3) As Double
     Set RangeA = Worksheets("Sheet8").Range("C3:E5")
     For i = 1 To 3
        For j = 1 To 3
          A(i,j) = RangeA(i,j)
          Worksheets("Sheet8").Range("C11").Cells(i,j).Value = A(i,j)
       Next
   Next
End Sub
```

Figure 14. Macro illustrating the use of range variables and arrays for input and output.

The code of Figure 14 uses a range variable called *RangeA*, and a 3x3, floating point (double precision), two-dimensional array called *A*. The statement

```
Set RangeA = Worksheets("Sheet1").Range("C3:E5")
```

sets the range variable *RangeA* equal to the range object C3:E5 (a 3x3 range) in worksheet "Sheet8". Thus, all the values in that range are now available in code as elements of *RangeA*. To extract those elements we use the two nested *For ... Next* loops shown above. Within those nested loops, the line

```
A(i,j) = RangeA.Cells(i,j).Value
```

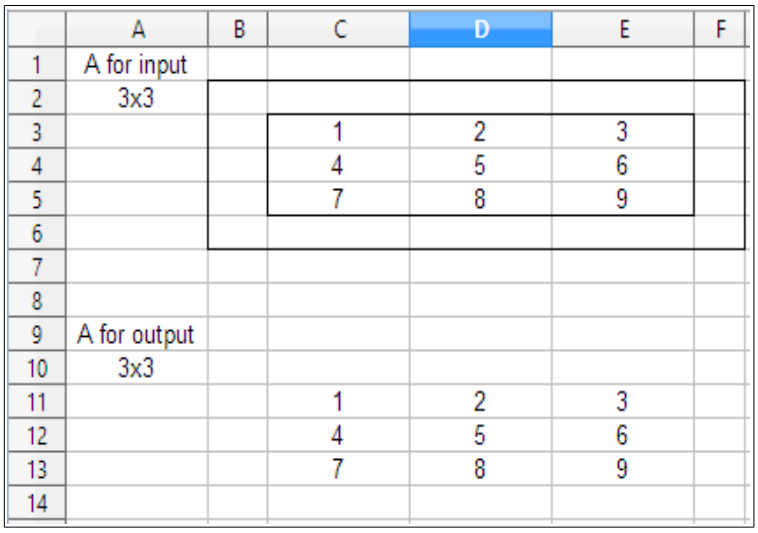
assigns the value of cell (i,j) in range *RangeA* to element A(i,j) of the double-precision array. The second line within the nested *For ... Next* loops writes out the value of element A(i,j) to cell (i,j) in the range whose upper left corner is "B11" in the spreadsheet "Sheet8":

```
Worksheets("Sheet8").Range("C11").Cells(i,j).Value = A(i,j)
```

In summary, the code of Figure 14 illustrates the use of range variables and array variables for input and output:

- Input is performed by simply assigning a range object to a range variable.
- Output, on the other hand, requires that the range elements' values be assigned to elements of an array, and then those values are moved to the spreadsheet's cells by changing the *Value* property of those cells.

The "Sheet8" spreadsheet, to which the code of Figure 14 applies, would look as shown in Figure 15, after the code is executed.



	A	B	C	D	E	F
1	A for input					
2	3x3					
3			1	2	3	
4			4	5	6	
5			7	8	9	
6						
7						
8						
9	A for output					
10	3x3					
11			1	2	3	
12			4	5	6	
13			7	8	9	
14						

Figure 15. Spreadsheet "Sheet8" after executing the code of Figure 14

***Example*: *Matrix transpose using ranges, arrays, and the array function* TRANSPOSE**

The example of Figures 14 and 15 simply read the data in range B3:D5 and copied them to range B11:D13. However, operations can be performed on the data in range variable *RangeA* in the code, before an output is performed. For example, adding a second range variable *RangeB*, defined as a *Variant* data type, we can store, and then, output to the spreadsheet, the transpose of the matrix in *RangeA*, as illustrated in the following code that applies to a worksheet called "Sheet9":

```
Sub Transpose3x3()
    Dim i As Integer, j As Integer
    Dim RangeA As Range, RangeB As Variant
    Dim AT(3, 3) As Double
    Set RangeA = Worksheets("Sheet9").Range("C3:E5")
    RangeB = Application.Transpose(RangeA)
    For i = 1 To 3
        For j = 1 To 3
           AT(i, j) = RangeB(i, j)
           Worksheets("Sheet9").Range("C11").Cells(i, j).Value = AT(i, j)
        Next
   Next
End Sub
```

Figure 16. Code for calculating the transpose of a matrix using range variables, arrays, and the TRANSPOSE array function.

The code of Figure 16 uses two range variables, *RangeA* (which is defined as a *Range* variable and is related to the worksheet range B3:D5), and *RangeB* (which is defined as a *Variant* variable and it stores

the transpose of *RangeA*). The statement

```
Set RangeA = Worksheets("Sheet9").Range("B3:D5")
```

associates range variable *RangeA* with the range B3:D5 in "Sheet9," while the statement

```
RangeB = Application.TRANSPOSE(RangeA)
```

(without the particle *Set*) calculate the transpose of *RangeA*, and stores it into *RangeB*.

NOTE: In VBA code, a range variable (such as *RangeA* in Figure 16), which is associated with a worksheet range, uses subindices (i,j), both of which start with 1. Similarly, variable *RangeB*, defined as a *Variant* data type, also has elements with subindices (i,j). Thus, extracting the components of *RangeB* in Figure 16 into the elements of the array AT(i,j) – which represent the elements of the transpose of *RangeA* – requires the following line:

```
AT(i,j) = RangeB(i,j)
```

The second line within the two nested loops assigns values of AT(i,j) to cells in the worksheet for the worksheet range whose upper left corner is located at B11:

```
Worksheets("Sheet9").Range("B11").Cells(i,j).Value = AT(i,j)
```

The following figure shows worksheet "Sheet9" after the code of Figure 16 is executed:

	A	B	C	D	E	F
1	A					
2						
3			1	2	3	
4			4	5	6	
5			7	8	9	
6						
7						
8						
9	A transpose					
10						
11			1	4	7	
12			2	5	8	
13			3	6	9	
14						

Figure 17. Worksheet "Sheet9" after the code of Figure 16 is executed.

Input of a generic matrix using range variables

The codes of Figures 14 and 16 used as inputs ranges whose sizes where known beforehand. For example, the line

```
Set RangeA = Worksheets("Sheet9").Range("B3:D5")
```

in the code of Figure 16 sets *RangeA* as a 3x3 array of cells in the worksheet "Sheet9." Thus, the size of *RangeA* is set to be a 3x3 array of cells.

Suppose that you want to read a matrix without knowing the number of rows and columns a priori. The only requirement is that the worksheet range to be entered has empty columns to the left and to the right, and empty rows at the top and the bottom, as illustrated, for a particular case, in the following Figure.

	A	B	C	D	E	F	G
1	A for input						
2							
3			1	2	3	16	
4			4	5	6	17	
5			7	8	8	18	
6			10	11	12	19	
7			13	14	15	20	
8							

Figure 18. Generic matrix to be entered using range variables in VBA code in worksheet "Sheet10".

The 3x4 matrix in Figure 18 has its upper left corner in cell C3, and it also has the required empty columns (columns B and G) to the left and right, and empty rows (rows 2 and 8) at the top and bottom. In order to read the matrix shown in Figure 18 (or any other matrix similarly defined), we will set a range variable *RangeA*, first, to the matrix's upper left corner (C3, in this case), and then use some special properties of the *Range* object to detect the number of rows and columns. This is done in code by finding the empty column to the right and row at the bottom of the range of interest. The code for inputting (and then outputting) the matrix of Figure 18 is shown next:

```
Sub InputOutputVariableSizeMatrix()
   Dim i As Integer, j As Integer, n As Integer, m As Integer
   Dim A() As Double
   Dim RangeA As Range
   Set RangeA = Worksheets("Sheet10").Range("C3")
   Set RangeA = RangeA.CurrentRegion.SpecialCells(xlCellTypeConstants, xlNumbers)
   m = RangeA.Rows.Count:  n = RangeA.Columns.Count
   MsgBox("No. of rows (m) = " & CStr(m) & _
          ", No. of columns (n) = " & CStr(n))
   ReDim A(m,n)
   For i = 1 To m
       For j = 1 To n
           A(i, j) = RangeA(i,j)
           RangeA.Cells(m+3+i,j) = A(i,j)
       Next j
   Next i
End Sub
```

Figure 19. Code for inputting (and then, outputting) a generic matrix using range variables – the size of the input range is automatically detected by using special properties of the *Range* object.

The code defines integer variables *i, j, n,* and *m*, and a dynamic double-precision array *A*( ), as well as a range variable *RangeA*. To select the range to be entered we use the following lines:

```
Set RangeA = Worksheets("Sheet10").Range("C3")
Set RangeA = RangeA.CurrentRegion.SpecialCells(xlCellTypeConstants, xlNumbers)
```

The first line above simply sets range variable *RangeA* to the upper left corner of the worksheet range of interest (`Worksheets("Sheet10").Range("C3")`). The second line uses the *CurrentRegion* and *SpecialCells* properties of the *RangeA* object to detect the extend of the range, as determined by the empty columns, left and right, and the empty rows, top and bottom, of the worksheet range of interest.

The *SpecialCells* property uses as arguments a couple of specifications, namely, *xlCellTypeConstants* and *xlNumbers*. These specifications look for constants (*xlCellTypeConstants*) or numbers (*xlNumbers*) in the cells near the specified upper corner until empty rows and columns are detected. Once those empty columns and empty rows framing the worksheet range of interest are detected, the range is redefined as extending from the specified upper left corner (C3, in this case) down to the lower right corner determined by the empty rows and columns (F7, in this case).

Associated with the redefined range variable *RangeA* are a couple of properties that determine the number of rows and columns in the range. These are stored into variables *m* and *n*, respectively, by using the line:

```
m = Range.Rows.Count :  n = RangeA.Columns.Count
```

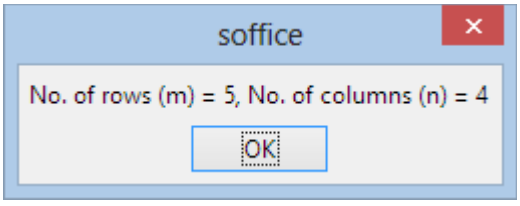
Recall that the colon (:) in VBA is used to separate two code statements that are placed in the same line. Thus, the line above, is equivalent to the two lines:

```
m = Range.Rows.Count
n = RangeA.Columns.Count
```

The statement

```
MsgBox("No. of rows (m) = " & CStr(m) & _
     ", No. of columns (n) = " & CStr(n))
```

is not strictly required for the code to perform the required input and output, but is added here to show the values of *m* and *n* thus detected. For the generic matrix of Figure 18, the following values are shown:



The values of *m* (number of rows) and *n* (number of columns) in range *RangeA* are then used to redimension the double-precision array as the two dimensional array: `Redim A(m, n)`. The values of *m* and *n* are also used as upper limits for the indices *i* and *j*, respectively, of the two nested *For ... Next*

loops that assign elements of *RangeA* to elements of the matrix *A*(i,j), and outputs the values of *A*(i,j) to the worksheet. The nested *For ... Next* loops are copied below:

```
For i = 1 To m
    For j = 1 To n
        A(i, j) = RangeA(i,j)
        RangeA.Cells(m+3+i,j).Value = A(i,j)
    Next j
Next i
```

In particular, the line `RangeA.Cells(m+3+i,j).Value = A(i,j)` takes care of outputting the value of A(i,j) at a row located *m*+3+i cells below the upper left corner of *RangeA*, which was defined as cell "C3" earlier in the code of Figure 19.

	A	B	C	D	E	F	G
1	A for input						
2							
3			1	2	3	16	
4			4	5	6	17	
5			7	8	8	18	
6			10	11	12	19	
7			13	14	15	20	
8							
9	A for output						
10							
11			1	2	3	16	
12			4	5	6	17	
13			7	8	8	18	
14			10	11	12	19	
15			13	14	15	20	
16							

Figure 20. Spreadsheet "Sheet10" after the code of Figure 19 is executed for the input matrix shown in Figure 18.

You can check the code of Figure 19 with other matrices and make sure that it works properly for those matrices.

Modifying the transpose code (in Figure 16) for a generic input matrix

The code of Figure 16, that produces the transpose of a matrix, uses a range of fixed size. The following is the modified code to include inputting a generic matrix.

```
Sub TransposeGenericMatrix()
    Dim i As Integer, j As Integer, n As Integer, m As Integer
    Dim AT() As Double
    Dim RangeA As Range, RangeB As Variant
    Set RangeA = Worksheets("Sheet11").Range("C3")
    Set RangeA = RangeA.CurrentRegion.SpecialCells(xlCellTypeConstants, xlNumbers)
    m = RangeA.Rows.Count:  n = RangeA.Columns.Count
    RangeB = Application.Transpose(RangeA)
    ReDim AT(n, m)
    For i = 1 To n
        For j = 1 To m
           AT(i, j) = RangeB(i, j)
           RangeA.Cells(m + 3 + i, j).Value = AT(i, j)
        Next
   Next
End Sub
```

Figure 21. Code for calculating the transpose of a matrix using range variables, arrays, and the TRANSPOSE array function.

Since the range *RangeA* has *m* rows and *n* columns, its transposed has *n* rows and *m* columns. This is accounted for by the statement **`Redim AT(n,m)`**. Also, the statement **`RangeA.Cells(m+3+i,j).Value = AT(i,j)`** takes care of outputting the contents of AT(i,j) at a row located *m*+3+*i* cells below the upper left corner of *RangeA*, i.e., *m*+3+*i* cells below cell C3.

The following figure shows an example of a transpose in worksheet "Sheet11."

	A	B	C	D	E	F	G	H
1	A:							
2								
3			1	2	3	16		
4			4	5	6	17		
5			7	8	8	18		
6			10	11	12	19		
7			13	14	15	20		
8								
9	A transpose:							
10								
11			1	4	7	10	13	
12			2	5	8	11	14	
13			3	6	8	12	15	
14			16	17	18	19	20	
15								

Figure 22. Example of a transpose calculated using the code of Figure 21.

VBA code for calculating the determinant of a matrix using MDETERM

The determinant of a matrix is a scalar value, therefore, a double-precision variable (say, *detA*) is typically used to store such value. The following code shows how to calculate, and output, the determinant of a matrix entered in worksheet "Sheet12," using a range variable and the MDETERM array function.

```
Sub DeterminantGenericMatrix()
    Dim m As Integer, n As Integer, detA As Double, RangeA As Range
    Set RangeA = Worksheets("Sheet12").Range("C3")
    Set RangeA = RangeA.CurrentRegion.SpecialCells(xlCellTypeConstants, xlNumbers)
    m = RangeA.Rows.Count : n = RangeA.Columns.Count
    If m = n Then
      detA = Application.MDETERM(RangeA)
      RangeA.Cells(m+3,1).Value = detA
    Else
       MsgBox("No determinant exists as the matrix is not square.")
    End If
End Sub
```

Figure 23. Code for calculating the determinant of a matrix using a range variable and the MDETERM array function.

The matrix is entered in *RangeA*, which is framed by empty rows and columns in worksheet "Sheet12." Input of matrix A, as *RangeA*, is performed as done in the codes of Figures 19 and 21. The dimensions of the input range, *RangeA*, are stored in integer variables *m* and *n*, respectively. Since a determinant is only available for square matrices, having the matrix dimensions, *m* and *n*, allows us to check if the matrix A is indeed square. If that is the case, the line `detA = Application.MDETERM(RangeA)` takes care of calculating the determinant and assigning its value to the double-precision variable *detA*. If the matrix is not square, then a message is sent to the user indicating so.

The following figure shows an example of calculating the determinant of a matrix using the code of Figure 23.

	A	B	C	D	E	F	G
1	A						
2							
3			1	2	3	4	
4			-2	6	-8	8	
5			3	4.3	11	5	
6			-5	4	-2	3	
7							
8							
9	det(A):		686.2				
10							

Figure 24. Example of determinant calculation for worksheet "Sheet12" using the code of Figure 23.

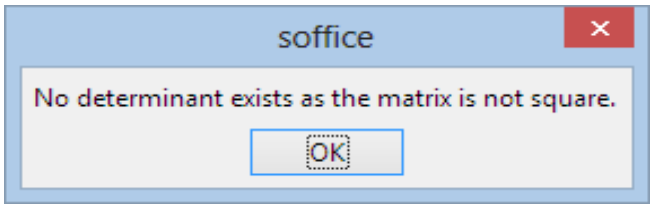
NOTE: All the labels of matrices and determinants for the examples of Figures 15, 17, 18, 20, 22, and 24, were entered by hand. No VBA code was used to enter those labels, although it should not be too difficult to enter them in code.

To test whether the code of Figure 23 is handling non-square matrices properly, try using the following input in "Sheet12":

	A	B	C	D	E	F
1	A					
2						
3			1	2	3	
4			-2	6	-8	
5			3	4.3	11	
6			-5	4	-2	
7						

Figure 25. An attempt to calculate the determinant for a non-square (4x3) matrix in "Sheet12" using the code of Figure 23.

The code of Figure 23 detects that the number of rows of the matrix in Figure 25 is not equal to the number of columns, therefore, the following message box is produced:



After pressing the [OK] button in this message box, the program ends.

VBA code for calculating the inverse of a matrix using MINVERSE

To put together the code for calculating the inverse of a matrix inputted from a worksheet range we incorporate the calculation of the determinant, since we know that a square matrix whose determinant is zero (a singular matrix) has no inverse. The following code can be used to input a matrix from the worksheet (as the range *RangeA*), determine the matrix dimensions (*m,n*), check if the matrix is square ($m = n$), calculate the determinant *detA* if the matrix is square, check if the matrix is nonsingular, and calculate the matrix inverse if that is the case.

To determine if the matrix is singular or not, we check if the absolute value of the determinant is smaller than a predetermined small value *epsilon* (ε). Since the determinant calculated in the VBA code, *detA*, is a floating point value, it could evaluate to a very small, but not necessarily zero, value. Therefore, testing whether *detA* = 0 is, most likely, not going to detect a singular matrix. Instead, using $|detA| \leq \varepsilon$ will do the job.

```
    Dim i As Integer, j As Integer, n As Integer, m As Integer
    Dim Ainv() As Double, detA As Double
    Dim RangeA As Range, RangeB As Variant
    Const epsilon = 0.00001 'value used to check for singular matrices
    Set RangeA = Worksheets("Sheet13").Range("C3")
    Set RangeA = RangeA.CurrentRegion.SpecialCells(xlCellTypeConstants, xlNumbers)
    m = RangeA.Rows.Count:  n = RangeA.Columns.Count  'matrix dimensions
    If m = n Then  'check that matrix is square
       detA = Application.MDeterm(RangeA)
       If Abs(detA) > epsilon Then   'check that matrix is nonsingular
         RangeA.Cells(m + 3, 1).Value = detA         'output determinant
          RangeB = Application.MInverse(RangeA)      'calculate inverse matrix
          ReDim Ainv(m, n)                           'array for inverse matrix
          For i = 1 To n                             'output inverse matrix
              For j = 1 To m
                  Ainv(i, j) = RangeB(i, j)
                  RangeA.Cells(m + 5 + i, j).Value = Ainv(i, j)
              Next
          Next
       Else
          MsgBox ("Determinant = " & CStr(detA) & "." & Chr(13) & _
                 "Singular matrix detected. No inverse matrix is possible.")
       End If
    Else
        MsgBox ("No inverse matrix exists as the matrix entered is not square.")
    End If
End Sub
```

Figure 26. VBA code for determinant and matrix inverse using range variables, arrays, and the MDETERM and MINVERSE array functions.

The following figure shows the calculation of the inverse matrix for a 4x4 nonsingular matrix in worksheet "Sheet12."

(see next page)

	A	B	C	D	E	F	G
1	A						
2							
3			1	2	3	4	
4			-2	6	-8	8	
5			3	4.3	11	5	
6			-5	4	-2	3	
7							
8							
9	det(A):		686.2				
10	A inverse:						
11							
12			-0.3200	0.1502	0.1545	-0.2314	
13			-0.8423	0.1997	0.3702	-0.0262	
14			0.1140	-0.0858	0.0087	0.0624	
15			0.6657	-0.0730	-0.2303	0.0242	
16							
17							
18	A*Ainverse:						
19							
20			1	5.5511E-017	-1.1102E-016	-1.388E-017	
21			0	1	-4.4409E-016	-8.327E-017	
22			-8.8818E-016	5.5511E-017	1	-1.388E-017	
23			2.2204E-016	2.7756E-017	1.1102E-016	1	

Figure 27. Example of calculating the determinant and inverse matrix using the code of Figure 26.

Figure 27 shows not only the original matrix A, its determinant and its inverse, but also the check that A*Ainverse = I, the identity matrix. The latter result, shown in B20:F23 in Figure 26, was calculated directly in the worksheet by using the MMULT array function as follows:

- Select range B20:F23
- In the input bar type "`=MMULT(C3:F6,C12:F15)`"
- Press CNTL+SHIFT+ENTER

The identity matrix in range B20:F23 should be:

	A	B	C	D	E	F	G
18	A*Ainverse:						
19							
20			1	0	0	0	
21			0	1	0	0	
22			0	0	0	1	
23			0	0	0	1	
24							

However, due to numerical error carryovers, inherent to all calculations with floating-point data, some of the "zero" values in the range B20:F23 in Figure 27 are actually non-zero. It should be noticed, however, that those nonzero values are of the order of 10^{-16} to 10^{-17}, i.e., zeros, for all practical purposes. Thus, the property A*Ainverse = I is verified with the data of Figure 27.

Entering two generic matrices using range variables – Matrix multiplication example
The code for entering two generic matrices and multiplying them using two-dimensional arrays was presented in Figure 9. In that case, determining the size of the two input matrices was done by looping through the worksheet cells until an empty row and an empty column were found. The multiplication then proceeded by using array element operations. Using the technique for reading a range out of the worksheet into a range variable in VBA code, and then, taking advantage of the MMULT array function, we can simplify the code of Figure 9, to the one shown below.

```
Sub MultiplicationRangeMMULT()
    Dim i As Integer, j As Integer
    Dim nA As Integer, mA As Integer
    Dim nB As Integer, mB As Integer
    Dim nC As Integer, mC As Integer
    Dim C() As Double
    Dim RangeA As Range, RangeB As Range, RangeC As Variant
    Set RangeA = Worksheets("Sheet14").Range("C3")
    Set RangeA = RangeA.CurrentRegion.SpecialCells(xlCellTypeConstants, xlNumbers)
    mA = RangeA.Rows.Count:  nA = RangeA.Columns.Count
    'MsgBox("mA = " & CStr(mA) & ", nA = " & CStr(nA))
    Set RangeB = RangeA.Cells(mA + 2, 1)
    Set RangeB = RangeB.CurrentRegion.SpecialCells(xlCellTypeConstants, xlNumbers)
    mB = RangeB.Rows.Count:  nB = RangeB.Columns.Count
    'MsgBox("mB = " & CStr(mB) & ", nB = " & CStr(nB))
    If (nA = mB) Then
        RangeC = Application.MMult(RangeA, RangeB)
        ReDim C(mA, nB)
        For i = 1 To mA
            For j = 1 To nB
                C(i, j) = RangeC(i, j)
                RangeB.Cells(mB + 1 + i, j).Value = C(i, j)
            Next
        Next
        Range("A3").Cells(mA + nA + 3).Value = "C(" & CStr(mA) & "x" & _
                                               CStr(nB) & ")"
    Else
       MsgBox ("Dimensions of matrices A and B in C = A*B are incompatible.")
    End If
End Sub
```

Figure 28. Code for matrix multiplication using range variables, arrays, and the MMULT array function.

An example of multiplication of two matrices in worksheet "Sheet12" is shown below:

	A	B	C	D	E	F
1						
2						
3	A (3x4)		5	3	8	4
4			-2	1	6	-8
5			3	4	5	2
6						
7	B(4x2)		3	6		
8			-5	2		
9			8	-1		
10			2	0		
11						
12	C(3x2)		72	28		
13			21	-16		
14			33	21		

Figure 29. Matrix multiplication using range variables, arrays, and the MMULT array function through the code of Figure 28.

VBA code for solving a matrix equation using array operations

We have indicated above that a system of linear equations can be written as a matrix equation of the form **A x** = **b**, where **A** is a square matrix, and **b** is a vector of right-hand side coefficients. An example given earlier refers to the system of equations:

$$\begin{aligned} -2\cdot x+5\cdot y+7\cdot z&=26\\ 3\cdot x-5\cdot y+z&=25\\ x+10\cdot y-5\cdot z&=-47 \end{aligned}$$

with $$\boldsymbol{A}=\begin{bmatrix}-2 & 5 & 7\\ 3 & -5 & 1\\ 1 & 10 & -5\end{bmatrix},\quad \boldsymbol{x}=\begin{bmatrix}x\\ y\\ z\end{bmatrix},\text{ and }\quad \boldsymbol{b}=\begin{bmatrix}26\\ 25\\ -47\end{bmatrix},$$

so that the system of equations, becomes

$$\begin{bmatrix}-2 & 5 & 7\\ 3 & -5 & 1\\ 1 & 10 & -5\end{bmatrix}\cdot\begin{bmatrix}x\\ y\\ z\end{bmatrix}=\begin{bmatrix}26\\ 25\\ -47\end{bmatrix},\text{ or: }\quad \boldsymbol{A}\cdot\boldsymbol{x}=\boldsymbol{b}\ .\ \text{The solution is }\quad \boldsymbol{x}=\boldsymbol{A}^{-1}\cdot\boldsymbol{b}\ .$$

VBA code for solving a matrix equation using range variables, arrays, and the array functions MDETERM, MINVERSE, and MMULT is shown below. The solution thus presented applies to any square matrix **A**, entered in a worksheet range that is isolated through empty columns to the right and left, and empty rows at the top and bottom, and followed immediately below by a similarly isolated vector **b**. The solution is shown immediately below vector **b**.

```
Sub SolveMatrixEquationRangeArrayFunctions()
    Dim i As Integer, j As Integer
    Dim nA As Integer, mA As Integer
    Dim nx As Integer, mx As Integer
    Dim nB As Integer, mB As Integer
    Dim x() As Double, detA As Double
    Dim RangeA As Range, RangeAinv As Variant, RangeB As Range, RangeX As Variant
    Const epsilon = 0.00001
    Set RangeA = Worksheets("Sheet15").Range("C3")
    Set RangeA = RangeA.CurrentRegion.SpecialCells(xlCellTypeConstants, xlNumbers)
    mA = RangeA.Rows.Count:  nA = RangeA.Columns.Count
    If mA = nA Then
        detA = Application.MDeterm(RangeA)
        If Abs(detA) >= epsilon Then
            RangeAinv = Application.MInverse(RangeA)
            Set RangeB = RangeA.Cells(mA + 2, 1)
            Set RangeB = RangeB.CurrentRegion.SpecialCells(xlCellTypeConstants, _
                         xlNumbers)
            mB = RangeB.Rows.Count:  nB = RangeB.Columns.Count
            If (nA = mB) Then
                RangeX = Application.MMult(RangeAinv, RangeB)
                ReDim x(mA, nB)
                For i = 1 To mA
                    For j = 1 To nB
                        x(i, j) = RangeX(i, j)
                        RangeB.Cells(mB + 1 + i, j).Value = x(i, j)
                    Next
                Next
                Range("A3").Cells(mA + nA + 3).Value = "x"
            Else
                MsgBox ("Dimensions of matrix A and vector (or matrix) b in x =
Ainv*b are incompatible.")
            End If
        Else
            MsgBox ("The matrix of coefficients is singular.  " & Chr(13) & _
                   "No solution to the linear system was found.")
        End If
     Else
        MsgBox ("The matrix of coefficient is not square.  No solution
attempted.")
     End If
End Sub
```

Figure 30. VBA code for calculating the solution to a matrix equation using range variables, arrays, and the array functions MDETERM, MINVERSE, and MMULT.

The solution to the matrix equation **A x = b** set up earlier is given in the following Figure.

	A	B	C	D	E	F
1						
2						
3	A		-2	5	7	
4			3	-5	1	
5			1	10	-5	
6						
7	b		26			
8			25			
9			-47			
10						
11	x		3			
12			-2			
13			6			
14						

Figure 31. Solution to the matrix equation **A x = b** using the code of Figure 30.

Some applications of matrix equation solutions

The code of Figure 30 nicely summarizes many of the concepts presented in this chapter, as well as in earlier chapters:

- Mathematical concepts: matrices, matrix dimensions, matrix elements, matrix inverse, matrix multiplication, singular matrices, determinants, systems of linear equations, matrix equations
- VBA programming concepts: different data types: *Integer, Double* (double precision floating-point), *Range, Variant*; programming structures: sequence, decision (*If ... Then ... Else ... End If*), and loops (*For ... Next*); assignment statements; built-in functions (*Abs*); two-dimensional arrays, dynamic arrays, redimensioning; objects and their properties and methods; use of message boxes (*MsgBox*) and strings, etc.
- Worksheet and application (EXCEL) elements: ranges, cells, application functions (*Application.MDETERM*, *Application.MINVERSE, Application.MMULT*)

The code of Figure 30 also represents the solution to a commonly found mathematical problems: a system of linear equations. Thus, the code of Figure 30 will be very useful in practical applications in science and engineering problems. Two of those applications are presented next. The basic idea is to set up the problem as a system of linear equations and then use an interface (which may be similar to that of Figure 31) together with the code of Figure 30, or some modified version of it, to solve the problem. To facilitate calculation, the code of Figure 30, or its modified version, can be associated with a button in the interface. Additionally, a button to clear data from the interface (as was done in the interface of Figures 11, 12, and 13) could be added to the worksheet.

The examples to be considered are: (1) Solving multiples systems of linear equations; and (2) Direct fitting of a polynomial

Solving multiple systems of linear equations – Dimensional Analysis

Dimensional analysis is a technique used in Fluid Mechanics to reduce the number of physical variables involved in a fluid phenomenon to a smaller number of dimensionless numbers. This technique is commonly used to design experiments by relating dimensionless numbers to each other rather than using the original variables. Consider the following problem set up: to study the behavior of the drag force, F_D, that a flowing fluid at a velocity V exerts on a sphere of diameter D, we take into consideration not only these three variables (F_D, V, D), but also the acceleration of gravity, g; the density of the fluid, ρ; and the fluid viscosity, μ. Mathematically, the phenomenon under study can be written in a functional form as:

$$f(F_D, V, D, g, \rho, \mu) = 0 \quad .$$

The dimensions that describe the 6 variables involve are then listed in terms of three basic dimensions: either MLT: mass (M), length (T), and time (T), or FLT: force (F), length (L), and time (T). Table 1, below, shows the combinations of the three basic dimensions (MLT or FLT) corresponding to different variables used in fluid mechanics.

Thus, for the problem of interest, the variables involved and their dimensions are: F_D (MLT^{-2}), V (LT^{-1}), D(L), g(LT^{-2}), ρ(ML^{-3}), and μ($ML^{-1}T^{-1}$). The number of variables involved is $n = 6$, while the number of dimensions required to describe those variables, namely, MLT, is $m = 3$. A mathematical theorem, known as *Buckingham's Pi theorem*, indicates that we can form n-$m = 3$ dimensionless terms, also known as *Pi terms*, because they're typically named using the upper-case Greek letter Pi (Π).
To form the Π terms we take (m=) 3 so-called *repeating variables* from among the 6 variables involved. These need to be selected in such a way that we get:

- one geometric variable, i.e., one involving length, area, or volume
- one kinematic variable, i.e., one involving length (area or volume) and time , e.g., velocity, acceleration, discharge
- one dynamic variable, i.e., one involving length (area or volume), time, and another property (e.g., density, viscosity, etc.)

Table 1. Dimensions for various variables used in Fluid Mechanics

	Dimensions	
Quantity	(M,L,T)	(F,L,T)
Length (L)	L	L
Time (T)	T	T
Mass (M)	M	FT^2L^{-1}
Area (A)	L^2	L^2
Volume (Vol)	L^3	L^3
Velocity (V)	LT^{-1}	LT^{-1}
Acceleration (a)	LT^{-2}	LT^{-2}
Discharge (Q)	L^3T^{-1}	L^3T^{-1}
Kinematic viscosity (ν)	L^2T^{-1}	L^2T^{-1}
Force (F)	MLT^{-2}	F
Pressure (p)	$ML^{-1}T^{-2}$	FL^{-2}
Shear stress (τ)	$ML^{-1}T^{-2}$	FL^{-2}
Density (ρ)	ML^{-3}	FT^2L^{-4}
Specific weight (ω)	$ML^{-2}T^{-2}$	FL^{-3}
Energy/Work/Heat (E)	ML^2T^{-2}	FL
Power (P)	ML^2T^{-3}	FLT^{-1}
Dynamic viscosity (μ)	$ML^{-1}T^{-1}$	FTL^{-2}

Each Π term consists of multiplying these three (repeating) variables raised to different, unknown powers, say, *x, y, z*, and then multiplying each Π term by one of the remaining variables (one per Π term), raised to the first power. Thus, for the problem under consideration we can select our repeating variables to be:

- geometric variable = D
- kinematic variable = V
- dynamic variable = ρ

Thus, we form the following three Pi terms:

- $\Pi_1 = D^{x_1} \cdot V^{y_1} \cdot \rho^{z_1} \cdot F_D$
- $\Pi_2 = D^{x_2} \cdot V^{y_2} \cdot \rho^{z_2} \cdot g$
- $\Pi_3 = D^{x_3} \cdot V^{y_3} \cdot \rho^{z_3} \cdot \mu$

Next, we replace the dimensions found earlier for each term in the Π terms, taking into account that, since the Pi terms are dimensionless, their dimensions should be $M^0 \cdot L^0 \cdot T^0$.

Thus, for Π_1: $M^0 \cdot L^0 \cdot T^0 = (L)^{x_1} \cdot (L \cdot T^{-1})^{y_1} \cdot (M \cdot L^{-3})^{z_1} \cdot (M \cdot L \cdot T^{-2})$, i.e.,

$$M^0 \cdot L^0 \cdot T^0 = M^{z_1+1} \cdot L^{x_1+y_1-3 \cdot z_1+1} \cdot T^{-y_1-2} \quad ,$$

from which we get the system of linear equations:

$$\begin{array}{lll} z_1+1=0 & & z_1=-1 \\ x_1+y_1-3 \cdot z_1+1=0 & => & x_1+y_1-3 \cdot z_1=-1 \\ -y_1-2=0 & & -y_1=2 \end{array}$$

or,

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & -3 \\ 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ 2 \end{bmatrix}$$

Also, for Π_2: $M^0 \cdot L^0 \cdot T^0 = (L)^{x_2} \cdot (L \cdot T^{-1})^{y_2} \cdot (M \cdot L^{-3})^{z_2} \cdot (L \cdot T^{-2})$, i.e.,

$$M^0 \cdot L^0 \cdot T^0 = M^{z_2} \cdot L^{x_2+y_1-3 \cdot z_2+1} \cdot T^{-y_2-2} \quad ,$$

from which we get the system of linear equations:

$$\begin{array}{lll} z_2=0 & & z_2=0 \\ x_2+y_2-3 \cdot z_2+1=0 & => & x_2+y_2-3 \cdot z_2=-1 \\ -y_1-2=0 & & -y_2=2 \end{array}$$

or,

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & -3 \\ 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 2 \end{bmatrix} \quad .$$

Finally, for Π_3: $M^0 \cdot L^0 \cdot T^0 = (L)^{x_3} \cdot (L \cdot T^{-1})^{y_3} \cdot (M \cdot L^{-3})^{z_3} \cdot (M \cdot L^{-1} \cdot T^{-1})$, i.e.,

$$M^0 \cdot L^0 \cdot T^0 = M^{z_3+1} \cdot L^{x_3+y_3-3 \cdot z_3-1} \cdot T^{-y_3-1} \quad ,$$

from which we get the system of linear equations:

$$\begin{array}{lll} z_3+1=0 & & z_3=-1 \\ x_3+y_3-3 \cdot z_3-1=0 & => & x_3+y_3-3 \cdot z_3=1 \\ -y_3-1=0 & & -y_3=1 \end{array}$$

or,

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & -3 \\ 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_3 \\ y_3 \\ z_3 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix} \quad .$$

When you look at the matrix equations corresponding to each Π term, you'll find that they all have the same matrix of coefficients, but different unknowns and right-hand side vectors. However, since the matrix of coefficients is exactly the same, we can put together the following matrix equation incorporating the three matrix equations listed above, namely:

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & -3 \\ 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{bmatrix} = \begin{bmatrix} -1 & 0 & -1 \\ -1 & -1 & 1 \\ 2 & 2 & 1 \end{bmatrix}$$

This matrix equation has the general form: $\mathbf{A\,X = B}$,

with: $\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & -3 \\ 0 & -1 & 0 \end{bmatrix}$, $\boldsymbol{X} = \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{bmatrix}$, and $\boldsymbol{B} = \begin{bmatrix} -1 & 0 & -1 \\ -1 & -1 & 1 \\ 2 & 2 & 1 \end{bmatrix}$.

Thus, the solution sought is: $\boldsymbol{X} = \boldsymbol{A}^{-1} \cdot \boldsymbol{B}$.

Using the interface of Figure 31 and the code of Figure 30, we find the solution to be as shown in the Figure below.

	A	B	C	D	E	F
1						
2						
3	A		0	0	1	
4			1	1	-3	
5			0	-1	0	
6						
7	b		-1	0	-1	
8			-1	-1	1	
9			2	2	1	
10						
11	x		-2	1	-1	
12			-2	-2	-1	
13			-1	0	-1	
14						

Figure 32. Solution to multiple systems of linear equations for the case of the 3 Pi terms for dimensional analysis of drag force on a sphere

From the result shown above it follows that $x_1=-2, y_1=-2, z_1=-1$, $x_2=1, y_2=-2, z_2=0$, and $x_3=-1, y_3=-1, z_3=-1$. With these results, the Π terms become:

- $\Pi_1=D^{x_1}\cdot V^{y_1}\cdot\rho^{z_1}\cdot F_D=D^{-2}\cdot V^{-2}\cdot\rho^{-1}\cdot F_D=\dfrac{F_D}{\rho\cdot V^2\cdot D^2}$
- $\Pi_2=D^{x_2}\cdot V^{y_2}\cdot\rho^{z_2}\cdot g=D^{1}\cdot V^{-2}\cdot\rho^{0}\, g=\dfrac{g\cdot D}{V^2}$
- $\Pi_3=D^{x_3}\cdot V^{y_3}\cdot\rho^{z_3}\cdot\mu=D^{-1}\cdot V^{-1}\cdot\rho^{-1}\cdot\mu=\dfrac{\mu}{\rho\cdot V\cdot D}$

With these 3 dimensionless Pi terms, the original relationship involving 6 variables reduces to:

$$\varphi\left(\frac{F_D}{\rho\cdot V^2\cdot D^2},\frac{g\cdot D}{V^2},\frac{\mu}{\rho\cdot V\cdot D}\right)=0 \quad .$$

Thus, the technique of forming Π terms through dimensional analysis allowed us to reduce the number of variables describing the physical phenomenon of drag force on a submerged sphere from 6 original variables to 3 dimensionless Π terms.

Setting up the matrix equation does not have to involve replacing dimensions in each Π term and expanding the exponents of M, L, and T, every time. Instead, we can simply set up the following table listing the exponents of M, L, and T, for the repeating and non repeating variables, taking care to change the sign of the exponents for the non-repeating variables. Thus, for the present case we can write:

	Repeating variables			Non-repeating (multiplied by -1)		
	D	V	ρ	F_D	g	μ
M	0	0	1	-1	0	-1
L	1	1	-3	-1	-1	1
T	0	-1	0	2	2	1
		matrix **A**			matrix **B**	

Figure 33. Setting up the matrix of coefficients and the right-hand side matrix for the multiple system of linear equations corresponding to the dimensional analysis of drag force on a sphere.

Direct fitting of (*x*,*y*) data a polynomial – Fitting a quadratic pump curve

By direct fitting of (*x*,*y*) data to a polynomial we understand that a list of data (x_i, y_i), i = 1, 2, …, n, is given and it will be used to fit a polynomial of degree (n-1), i.e., a polynomial of the form:

$$y=a_1+a_2\cdot x+a_3\cdot x^2+\ldots+a_n\cdot x^{n-1}$$

The problem consists, basically, of determining the coefficients a_1, a_2, …, a_n of the polynomial.

By replacing the individual data points, (x_i, y_i), $i = 1, 2, \ldots, n$, in the polynomial we can form the following n linear equations with n unknowns $(a_1, a_2, \ldots, a_n)$:

$$\begin{aligned} a_1 + a_2 \cdot x_1 + a_3 \cdot x_1^2 + \ldots + a_n \cdot x_1^{n-1} &= y_1 \\ a_1 + a_2 \cdot x_2 + a_3 \cdot x_2^2 + \ldots + a_n \cdot x_2^{n-1} &= y_2 \\ &\ldots \\ a_1 + a_2 \cdot x_n + a_3 \cdot x_n^2 + \ldots + a_n \cdot x_n^{n-1} &= y_n \end{aligned}$$

This system of linear equations can be written out as the matrix equation:

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^n \\ 1 & x_2 & x_2^2 & \ldots & x_2^n \\ 1 & \ldots & \ldots & \ldots & x_1^n \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \ldots \\ a_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \ldots \\ y_n \end{bmatrix} ,$$

or,

$$\mathbf{A\,x = b}$$

where

$$\boldsymbol{A} = \begin{bmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^n \\ 1 & x_2 & x_2^2 & \ldots & x_2^n \\ 1 & \ldots & \ldots & \ldots & x_1^n \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{bmatrix} , \quad \boldsymbol{x} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \ldots \\ a_n \end{bmatrix} , \text{ and } \quad \boldsymbol{b} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \ldots \\ y_n \end{bmatrix} .$$

For example, consider the case of obtaining the pump curve for a centrifugal pump. It is known that discharge Q and head h_P data for a centrifugal pump can be fitted to a quadratic equation, whose plot is referred to as the *pump curve*. For direct fitting to a quadratic equation (n=2) we need to have only 3 points. Thus, we could set up the pump of interest in a lab, and measure three discharge-head points. The following table shows data for a particular pump:

Q(cfs)	hP(ft)
0.1	15.71
4.8	12
9.8	0.81

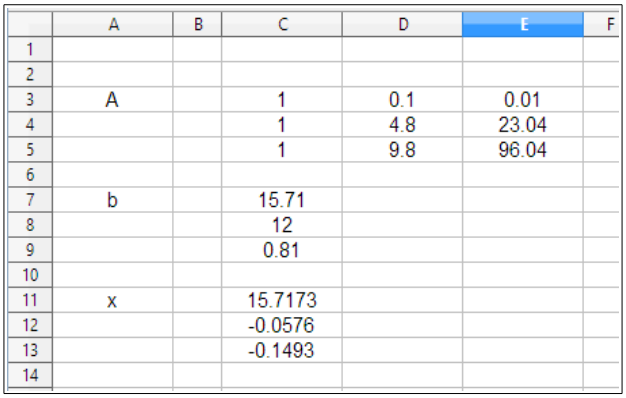
Matrix **A** for this case is:

$$\boldsymbol{A} = \begin{bmatrix} 1 & Q_1 & Q_1^2 \\ 1 & Q_2 & Q_2^2 \\ 1 & Q_3 & Q_3^2 \end{bmatrix} = \begin{bmatrix} 1 & 0.1 & 0.1^2 \\ 1 & 4.8 & 4.8^2 \\ 1 & 9.8 & 9.8^2 \end{bmatrix} = \begin{bmatrix} 1 & 0.1 & 0.01 \\ 1 & 4.8 & 23.04 \\ 1 & 9.8 & 96.04 \end{bmatrix} ,$$

while vector **b** is:

$$\boldsymbol{b}=\begin{bmatrix} hP_1 \\ hP_2 \\ hP_3 \end{bmatrix}=\begin{bmatrix} 15.71 \\ 12.00 \\ 0.81 \end{bmatrix} .$$

Thus, we can use the interface of Figure 31 and code of Figure 30 to get the coefficients of the quadratic fitting. The result is shown in the following figure:



	A	B	C	D	E	F
1						
2						
3	A		1	0.1	0.01	
4			1	4.8	23.04	
5			1	9.8	96.04	
6						
7	b		15.71			
8			12			
9			0.81			
10						
11	x		15.7173			
12			-0.0576			
13			-0.1493			
14						

Figure 34. Using the code of Figure 30 to fit pump data to a quadratic equation.

The solution shown in Figure 34 indicates that $a_1 = 15.71$, $a_2 = -0.0576$, and $a_3 = -0.1493$, i.e., $h_P = 15.7173 - 0.0576\,Q - 0.1493Q^2$, with h_P (ft) and Q(cfs).

Direct polynomial fitting – interface using (x,y) data directly

In the example shown above we used the code of Figure 30 and the interface of Figure 31 to solve the matrix equation resulting from fitting a quadratic equation. We could develop an interface specifically aimed at direct fitting of polynomial, which will require inputting two columns of data (*x* column, and *y* column), creating the matrix **A**, the vector **b**, and then incorporating parts of the code of Figure 30 to solve the matrix equation. The following interface is suggested and used to repeat the problem of fitting the quadratic equation to the pump data (x → Q, y → hP):

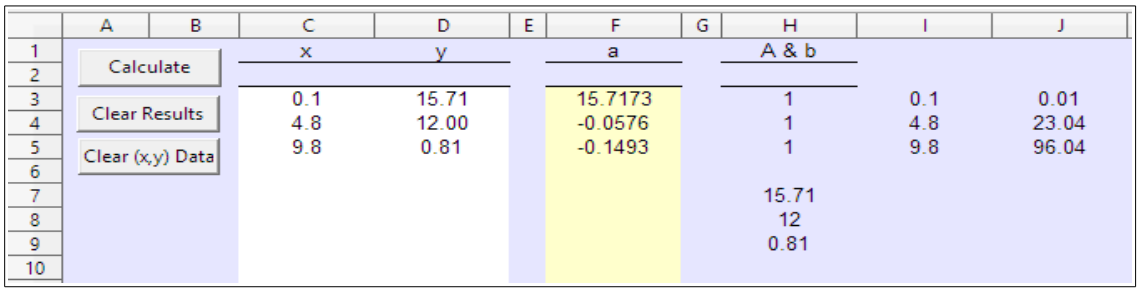


Figure 35. Interface for polynomial fitting given the data set (*x*,*y*). The coefficients *a* correspond to powers of *x* starting with 0, 1, 2, …

There are three buttons included in this interface:

- [Calculate]: Associated with macro *DirectPolyonomialFitting* – calculates polynomial fitting
- [Clear Results]: Associated with macro *ClearResultsPolynomialFitting* – clears worksheet from cell F3 to the right and below
- [Clear (x,y) Data]: Associated with macro *ClearXYDataPolynomialFitting* – clears table of (*x,y*) data

The different macros are listed below.

```
Sub DirectPolynomialFitting()
    Dim i As Integer, j As Integer, n As Integer, m As Integer
    Dim RangeTable As Range, RangeA As Range, RangeB As Range
    Dim RangeAA As Variant, RangeAinv As Variant
    Dim x() As Double, y() As Double, A() As Double, B() As Double, aa() As Double
    Dim detA As Double
    Const epsilon = 0.00001
    Set RangeTable = Worksheets("Sheet16").Range("C3") 'Input (x,y) TABLE
    Set RangeTable = RangeTable.CurrentRegion.SpecialCells(xlCellTypeConstants,
xlNumbers)
    n = RangeTable.Rows.Count: m = RangeTable.Columns.Count
    ReDim x(n): ReDim y(n): ReDim A(n, n): ReDim B(n)   'REDIM (x,y), A and B
    For i = 1 To n                                      'Separate x and y
            x(i) = RangeTable.Cells(i, 1).Value
            y(i) = RangeTable.Cells(i, 2).Value
    Next
    For i = 1 To n                                  'Create vector B and matrix A
        B(i) = y(i)
        Range("H3").Cells(i + n + 1, 1).Value = B(i)   'Write B, A to worksheet
        For j = 1 To n
            A(i, j) = x(i) ^ (j - 1)
            Range("H3").Cells(i, j).Value = A(i, j)
        Next
    Next
    Set RangeA = Worksheets("Sheet16").Range("H3")                  'Input RangeA
    Set RangeA = RangeA.CurrentRegion.SpecialCells(xlCellTypeConstants, xlNumbers)
    Set RangeB = Worksheets("Sheet16").Range("H3").Cells(i + n + 1, 1) 'In RangeB
    Set RangeB = RangeB.CurrentRegion.SpecialCells(xlCellTypeConstants, xlNumbers)
    detA = Application.MDeterm(RangeA)       'Check that matrix A is nonsingular
    If Abs(detA) >= 0 Then
        RangeAinv = Application.MInverse(RangeA)        'A^(-1) gets calculated
        RangeAA = Application.MMult(RangeAinv, RangeB)  'aa = A^(-1)*B,
coefficients
        ReDim aa(n)                                     'Vector of coefficients aa
        For i = 1 To n                                  'Extract coefficients aa
            aa(i) = RangeAA(i, 1)                       'and write components out
            Range("F3").Cells(i, 1).Value = aa(i)
        Next
    Else
        MsgBox ("The matrix of coefficients is singular.  " & Chr(13) & _
                  "No solution to the linear system was found.")
   End If
End Sub
```

Figure 36. VBA code for direct polynomial fitting for the interface in Figure 35.

```
Sub ClearResultsPolynomialFitting()
    Range("F3:ZZ103").ClearContents
End Sub

Sub ClearXYDataPolynomialFitting()
    Range("C3:D103").ClearContents
End Sub
```

Figure 37. VBA code for clearing results and input data for the interface in Figure 35.

Interpreting the code of Figure 36 should be straightforward. This code borrows many of the ideas used in the code of Figure 30. In Figure 37, on the other hand, we introduce a *method* of the *Range object*, the *ClearContents* method to clear up all cells in the selected ranges for cleaning up the interface after a direct polynomial fitting had been calculated.

Example of polynomial fitting for a river bed profile

The data shown in the table below shows the profile of the center line of a movable river bed. The *x* data represents distance in *ft* along the center line, and *y* represents elevation, also in *ft*, above a reference level (or *datum*):

x(ft)	0	0.5	1	1.2	2	2.5	3.1	3.5
y(ft)	0.25	0.8	1.25	0.4	0.1	0.66	1.2	1.5

The data is entered in the interface of Figure 35, after clearing output and data. The results of operating the [Calculate] button after entering the new data set are shown below.

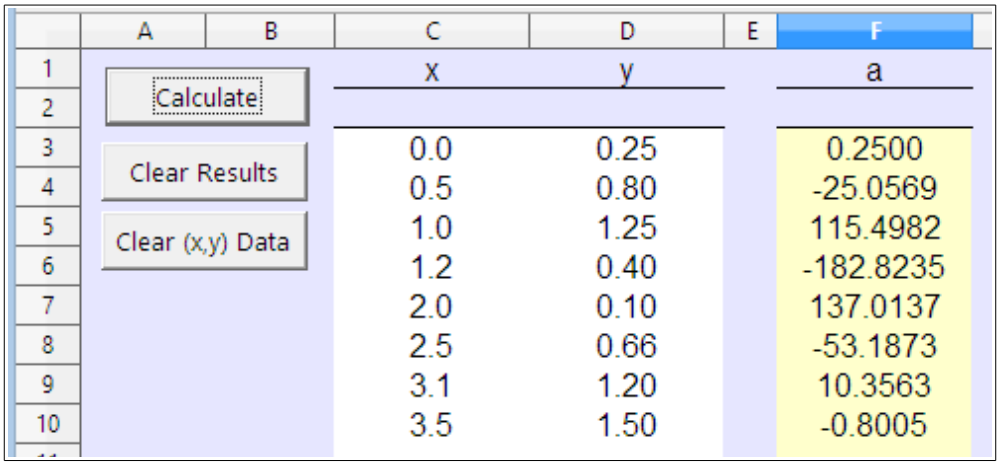


Figure 38. Polynomial data fitting for a river bed profile

The results of Figure 38 show that the polynomial fitted is:

$$y = 0.25 - 25.0569 \cdot x + 115.4982 \cdot x^2 - 182.8235 \cdot x^3 + 137.0137 \cdot x^4 - 53.1873 \cdot x^5 + 10.3563 \cdot x^6 - 0.8005 \cdot x^7 \quad .$$

A plot of the original data and the fitted data is shown next:

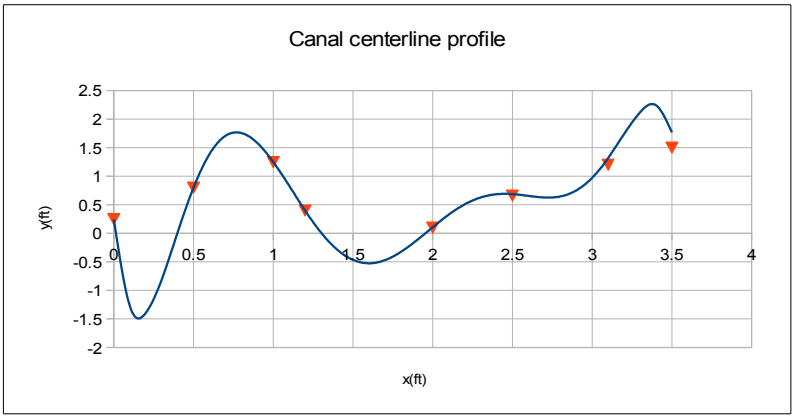


Figure 39. Original and fitted data for a canal center line profile.

FREQUENCY: EXCEL array function for finding the frequency distribution for a data sample
In an earlier section we indicated that the built-in array functions in EXCEL included the following: FREQUENCY, GROWTH, LINEST, LOGEST, MDETERM, MINVERSE, MMULT, SUMPRODUCT, SUMX2MY2, SUMX2PY2, SUMXMY2, TRANSPOSE, and TREND. Of these, we have presented in detail the use of array functions related to matrix operations, namely, MDETERM, MINVERSE, MMULT, and TRANSPOSE. In this section we introduce the use of the remaining array function FREQUENCY, which returns a frequency distribution.

Problem [29] in the exercises for Chapter 3 presents the basic concepts for producing a frequency distribution of a large data set. Some of those ideas are repeated here: Suppose that a large sample of numbers is represented by a list of size n : $\{x_1, x_2, \ldots, x_n\}$, from which we can obtain the sample's minimum and a maximum values: x_{min} and x_{max}. In producing the frequency distribution, we select a number of bins in which the data is going to be grouped. Suppose we decide that the minimum value for the bins will be Xb_{min}, while the maximum value for the bins is Xb_{max}. The data will be grouped into bins defined by the following *bin limits* or *class limits*: $[[Xb_1, Xb_2], [Xb_2, Xb_3], \ldots, [Xb_r, Xb_{max}]]$, where $Xb_1 = Xb_{min}$, and $Xb_1 < Xb_2 < \ldots < Xb_{max}$.

The values of Xb_{min} and Xb_{max} can be selected quite arbitrarily so that, for example, some values in the list $\{x_1, x_2, \ldots, x_n\}$ could fall to the left of Xb_{min}, while some could fall to the right of Xb_{max}. Thus, we could define two more bins to contains these *outliers*:

- The *lower-outlier* bin will contain values $x_k \le Xb_{min,}$. This is bin number 1.
- The *upper-outlier* bin will contain values $x_k > Xb_{max}$. This is the last bin to be used.

Let the array f_j represent the number of data points x_k that belong in the j-th bin. The values f_j are referred to as the *frequency count* for bin j. The list $\{f_1, f_2, \ldots, f_{nF}\}$ represents the frequency count list of the bins (j = 1, 2, .., nF). The number nF includes the two outlier bins. It is easy to show that:

$$n = sumf = \sum_{i=1}^{nF} f_i$$

Associated with the bin frequency count is the *bin midpoints* or *class marks* defined as:

$$XM_j = \frac{Xb_{j-1} + Xb_j}{2} \quad , \text{for } j = 2, \ldots, nF\text{-}1.$$

From the frequency count list completed above, namely, $\{f_1, f_2, \ldots, f_{nF}\}$ we can calculate the *cumulative frequency* list $\{cf_1, cf_2, \ldots, cf_{nF}\}$, by using:

- $cf_1 = f_1$
- $cf_j = cf_{j-1} + f_j \quad , \text{for } j = 2, 3, \ldots, nF$

We can also calculate the *relative frequencies* cf_j and *cumulative relative frequencies* crf_j, as follows:

- $rf_j = \frac{f_j}{sumf} \quad , \quad crf_j = \frac{cf_j}{sumf} \quad , \text{for } j = 1, 2, \ldots, r$

A table summarizing this frequency count exercise will include columns corresponding to: (i) Lower Class Limit Xb_j, (ii) Upper Class Limit Xb_{j+1}, (iii) Class marks XM_j, (iv) Frequency count f_j, (v) Cumulative frequency cf_j, (vi) Relative frequency rf_j, and, (vii) Cumulative relative frequency crf_j. In a spreadsheet, the frequency table may look as shown in the Figure below.

Class Limits		Mark	Freq.	Cfreq.	Rfreq.	CRFreq.
xLow	xHigh	xM	f	cf	rf	crf
outliers	<150	n/a	0	0	0	0
150	150	150	1	1	0.01	0.01
200	200	175	16	17	0.16	0.17
250	250	225	29	46	0.29	0.46
300	300	275	39	85	0.39	0.85
350	350	325	12	97	0.12	0.97
400	400	375	3	100	0.03	1
outliers	> 400	n/a	0	100	0	1
		Sum:	100			

Figure 40. Example of frequency distribution table

***Using array function* FREQUENCY -** Array function *FREQUENCY* can be used to obtain the frequency counts f_j given a data sample and a set of class limits in a EXCEL worksheet. The description presented here corresponds to the frequency table shown in Figure 41, below. To illustrate the use of array function FREQUENCY we generate 100 data points in cells B4:B103 (100 cells) by starting with the formula: "`=NORMINV(RAND(), 300, 50)`" in cell B4, and dragging it down to cell B103. In cells D1 and F1 we show the minimum and maximum values, respectively, of the list of numbers in B4:B013. Thus, in cell D1 we enter the formula "`=MIN(B4:B103)`", while in cell F1 we enter the formula "`=MAX(B4:B103)`". Cells D1 and F1 reveal that the minimum and maximum value for our example are *min* = 158.263 and *max* = 429.163. Using this information we select the bins listed in cells D4: D10, starting with Xb_{min} = 150, in increments of 50, until reaching Xb_{max} = 450. Function FREQUENCY requires that the last entry in the bin limits be simply the reference " > 450".

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		Min_Max:		158.263		429.163		INTERPRETATION:						
2		Data		Classes		Freq.		Class Limits		Mark	Freq.	Cfreq.	Rfreq.	CRFreq.
3								xLow	xHigh	xM	f	cf	rf	crf
4		301.26		150		0		<	150	n/a	0	0	0	0
5		294.84		200		2		150	200	175	2	2	0.02	0.02
6		302.20		250		16		200	250	225	16	18	0.16	0.18
7		269.79		300		38		250	300	275	38	56	0.38	0.56
8		347.05		350		32		300	350	325	32	88	0.32	0.88
9		279.81		400		9		350	400	375	9	97	0.09	0.97
10		241.81		450		3		400	450	425	3	100	0.03	1
11		418.66		>450		0		>	450	n/a	0	100	0	1
12		291.37								SUM:	100			
13		382.23												

Figure 41. Frequency distribution table resulting from using array function FREQUENCY in EXCEL

.

To obtain the frequency list we select cell F4 and type the formula "`=FREQUENCY(B4:B103,D4:D11)`" in that cell, finishing this operation by entering CNTL+SHIFT+ENTER. The result is the listing of frequencies in cells F4:F11. The table under the heading "INTERPRETATION" is then generated by copying class limits in columns H and I, and frequencies in column K. Class marks are calculated in column J (except for the outlier classes), while cumulative frequencies, relative frequencies, and cumulative relative frequencies are calculated in columns L, M, and N, respectively. The following Figure shows the formulas used in the worksheet of Figure 41.

	A	B	C	D	E	F
1		Min_Max:		=MIN(B4:B103)		=MAX(B4:B103)
2		Data		Classes		Freq.
3						
4		=NORMINV(RAND(),300,50)		150		{=FREQUENCY(B4:B103,D4:D11)}
5		=NORMINV(RAND(),300,50)		=D4+50		{=FREQUENCY(B4:B103,D4:D11)}
6		=NORMINV(RAND(),300,50)		=D5+50		{=FREQUENCY(B4:B103,D4:D11)}
7		=NORMINV(RAND(),300,50)		=D6+50		{=FREQUENCY(B4:B103,D4:D11)}
8		=NORMINV(RAND(),300,50)		=D7+50		{=FREQUENCY(B4:B103,D4:D11)}
9		=NORMINV(RAND(),300,50)		=D8+50		{=FREQUENCY(B4:B103,D4:D11)}
10		=NORMINV(RAND(),300,50)		=D9+50		{=FREQUENCY(B4:B103,D4:D11)}
11		=NORMINV(RAND(),300,50)		>450		{=FREQUENCY(B4:B103,D4:D11)}
12		=NORMINV(RAND(),300,50)				
13		=NORMINV(RAND(),300,50)				
14		=NORMINV(RAND(),300,50)				

	H	I	J	K	L	M	N
1	INTERPRETATION:						
2	Class Limits		Mark	Freq.	Cfreq.	Rfreq.	CRFreq.
3	xLow	xHigh	xM	f	cf	rf	crf
4	<	=D4	n/a	=F4	=K4	=K4/$K$12	=M4
5	=D4	=D5	=(H5+I5)/2	=F5	=L4+K5	=K5/$K$12	=N4+M5
6	=D5	=D6	=(H6+I6)/2	=F6	=L5+K6	=K6/$K$12	=N5+M6
7	=D6	=D7	=(H7+I7)/2	=F7	=L6+K7	=K7/$K$12	=N6+M7
8	=D7	=D8	=(H8+I8)/2	=F8	=L7+K8	=K8/$K$12	=N7+M8
9	=D8	=D9	=(H9+I9)/2	=F9	=L8+K9	=K9/$K$12	=N8+M9
10	=D9	=D10	=(H10+I10)/2	=F10	=L9+K10	=K10/$K$12	=N9+M10
11	>	=D10	n/a	=F11	=L10+K11	=K11/$K$12	=N10+M11
12			SUM:	=SUM(K4:K11)			
13							
14							

Figure 42. Formulas used to complete the table of Figure 41.

***VBA* code for producing a frequency table** – We can make the production of a frequency table along the lines of that presented in Figure 41 by adding VBA code controlled by the buttons shown in the worksheet interface of Figure 43, below.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		Min_Max:		184.745		447.06		INTERPRETATION:						
2	Calculate	Data		Classes		Freq.		Class Limits		Mark	Freq.	Cfreq.	Rfreq.	CRFreq.
3								xLow	xHigh	xM	f	cf	rf	crf
4	Clear Data	331.59		150		0		<	150	n/a	0	0	0	0
5		250.60		200		1		150	150	150	1	1	0.01	0.01
6		233.73		250		16		200	200	175	16	17	0.16	0.17
7	Clear Classes	248.04		300		29		250	250	225	29	46	0.29	0.46
8		328.11		350		39		300	300	275	39	85	0.39	0.85
9		309.89		400		12		350	350	325	12	97	0.12	0.97
10	Clear Table	333.98		450		3		400	400	375	3	100	0.03	1
11		253.13		>450		0		>	400	n/a	0	100	0	1
12		317.67								Sum:	100			
13		258.49												
14		207.60												

Figure 43. Interface for calculating frequency distribution based on the example of Figure 41.

The user needs to enter the data in column B and the class limits in column D. Before doing that, however, it is recommended to use the buttons [Clear Data], [Clear Classes], and [Clear Table] to ensure that no extraneous data values are left in the worksheet before activating the [Calculate] button. The reason for this suggestion is that the input data in columns B and D needs to be framed with empty columns to the left and right, and empty rows above and below, to be able to input these ranges into code (in a similar fashion as data ranges were entered in the codes of Figure 36, for example). Thus, just pressing button [Calculate], after populating the *Data* and *Classes* columns (columns B and D, respectively), should produce the rest of the table. The only calculation that is not controlled by code is that of the maximum and minimum values of the data (cells D1 and F1, respectively). The formulas entered in those cells are "=MIN(B4:1003)" and "=MAX(B4:1003)", respectively. This means that the worksheet is set to allow up to 1000 data points to obtain the proper minimum and maximum values in cells D1 and F1, respectively. If you want to use more than 1000 data points, you can change the range in functions MIN and MAX, accordingly. For out example, we stick to using up to 1000 data points.

The four buttons in the worksheet interface of Figure 43 are described next:

- [Calculate] – associated with macro *myFrequencyApplication* – calculates frequency table given the data in column B and the class limits in column D
- [Clear Data] – associated with macro *ClearDataFrequency* – clears data in column B
- [Clear Classes] – associated with macro *ClearClassesFrequency* – clears class limits in column D
- [Clear Table] – associated with macro *ClearFrequencyFrequency* – clears the rest of the table

The codes for those macros are listed next.

```
Option Explicit
Option Base 1
'=====================================================
Sub myFrequencyApplication()
    'DEFINE VARIABLES TO BE USED
    Dim i As Integer, j As Integer
    Dim mD As Integer, nD As Integer
    Dim mC As Integer, nC As Integer
    Dim mF As Integer, nF As Integer
    Dim RangeData As Range, RangeClass As Range, RangeFreq As Variant
    Dim Xb() As Double, xMark() As Double
    Dim f() As Double, cf() As Double, rf() As Double, crf() As Double, sumf As
Double
    'INPUT DATA VALUES
    Set RangeData = Worksheets("FREQUENCY").Range("B4") 'Input data list
    Set RangeData = RangeData.CurrentRegion.SpecialCells(xlCellTypeConstants,
xlNumbers)
    mD = RangeData.Rows.Count: nD = RangeData.Columns.Count
    'INPUT CLASS LIMITS
    Set RangeClass = Worksheets("FREQUENCY").Range("D4") 'Input data list
    Set RangeClass = RangeClass.CurrentRegion.SpecialCells(xlCellTypeConstants,
xlNumbers)
```

```
    mC = RangeClass.Rows.Count: nC = RangeClass.Columns.Count
    'REDIMENSION Xb and xMark, EXTRACT Xb VALUES from RangeClass
    ReDim Xb(mC - 1): ReDim xMark(mC - 2)
    For i = 1 To mC - 1
        Xb(i) = RangeClass(i)(0)
    Next
    'CALCULATE FREQUENCY USING THE ARRAY FUNCTION
    RangeFreq = Application.Frequency(RangeData, RangeClass)
    'f = WorksheetFunction.Frequency(RangeData, RangeClass)
    mF = mC
    'REDIMENSION FREQUENCY VECTORS
    ReDim f(mF): ReDim cf(mF): ReDim rf(mF): ReDim crf(mF)
    sumf = 0#                                   'INITIALIZE sumf AS ZERO
    'THIS For LOOP EXTRACT FREQUENCIES (f), ACCUMULATES sumf, CALCULATES cf,
OUTPUTS f
    For i = 1 To mF
        f(i) = RangeFreq(i, 1)
        sumf = sumf + f(i)
        If i = 1 Then
           cf(i) = f(i)
        Else
           cf(i) = sumf
        End If
        Range("F4").Cells(i, 1).Value = f(i)
    Next
    'THIS For LOOP CALCULATES RELATIVE FREQUENCY rf AND ITS CUMULATIVE VALUES crf
    For i = 1 To mF
        rf(i) = f(i) / sumf
        crf(i) = cf(i) / sumf
    Next
    'WRITE OUT FIRST COLUMN OF TABLE
    Range("H4").Cells(1, 1).Value = "<"
    For i = 2 To mC - 1
        Range("H4").Cells(i, 1).Value = Xb(i)
    Next
    Range("H4").Cells(i, 1).Value = ">"
    'WRITE OUT REMAINING COLUMNS OF TABLE
    For i = 1 To mC
        If i = mC Then
           Range("H4").Cells(i, 2).Value = Xb(i - 1)
        Else
           Range("H4").Cells(i, 2).Value = Xb(i)
        End If
        If i = 1 Or i = mC Then
           Range("H4").Cells(i, 3).Value = "n/a"
        Else
           Range("H4").Cells(i, 3).Value = (Xb(i - 1) + Xb(i)) / 2
        End If
        Range("H4").Cells(i, 4).Value = f(i)
        Range("H4").Cells(i, 5).Value = cf(i)
        Range("H4").Cells(i, 6).Value = rf(i)
        Range("H4").Cells(i, 7).Value = crf(i)
    Next
```

```
    'WRITE OUT SUM OF FREQUENCIES TO COMPLETE TABLE
    Range("H4").Cells(i, 3).Value = "Sum:"
    Range("H4").Cells(i, 4).Value = sumf
End Sub
```

Figure 44. VBA code associated with the [Calculate] button for producing the frequency table shown in Figure 43.

```
'=============================================================
Sub ClearDataFrequency()
    Worksheets("FREQUENCY").Range("B4:B1500").ClearContents
End Sub
'=============================================================
Sub ClearClassesFrequency()
    Worksheets("FREQUENCY").Range("D4:D100").ClearContents
End Sub
'=============================================================
Sub ClearFrequencyFrequency()
    Worksheets("FREQUENCY").Range("F4:N100").ClearContents
End Sub
'=============================================================
```

Figure 45. VBA code associated with the [Clear Data], [Clear Classes], and [Clear Table] buttons, respectively, in the interface of Figure 43.

From the data of Figure 43 we can produce a couple of graphs that summarize the data nicely, namely, the histogram of the data and the cumulative frequency ogive.

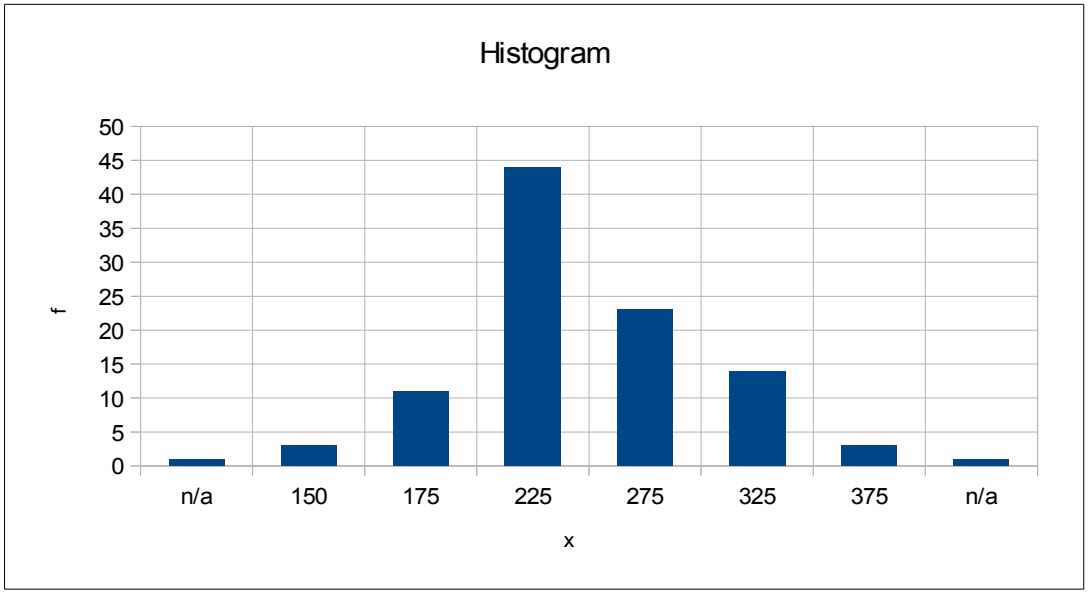


Figure 46. Histogram for the data of Figure 43.

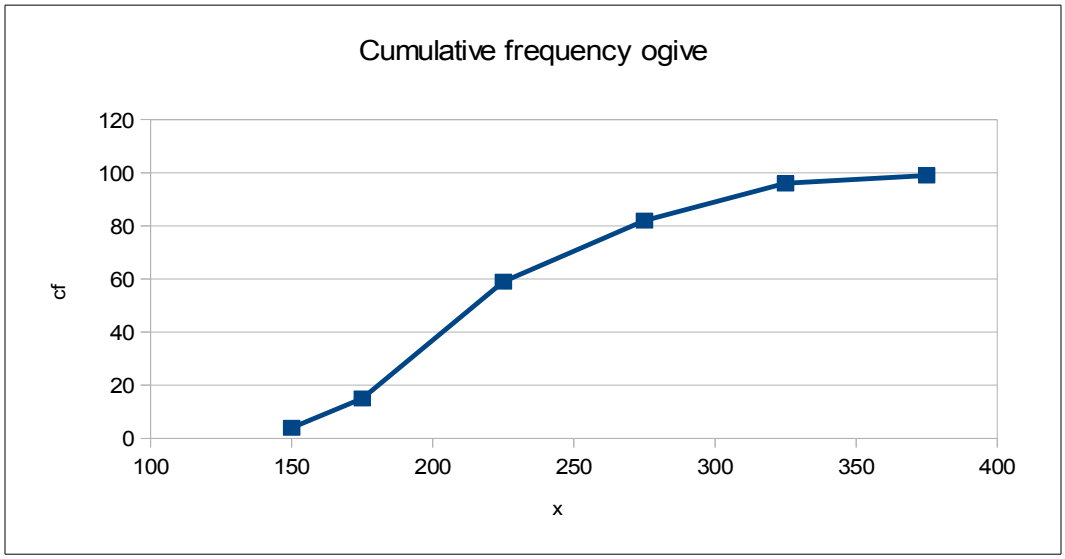


Figure 47. Cumulative frequency ogive for the data of Figure 43.

Summary

This chapter introduced concepts related to matrices, matrix operations, array operations and functions, two-dimensional arrays, and the use of *Range* and *Variant* variables. Some practical applications were also presented, e.g., solution of single and multiple systems of linear systems, frequency distributions, etc.