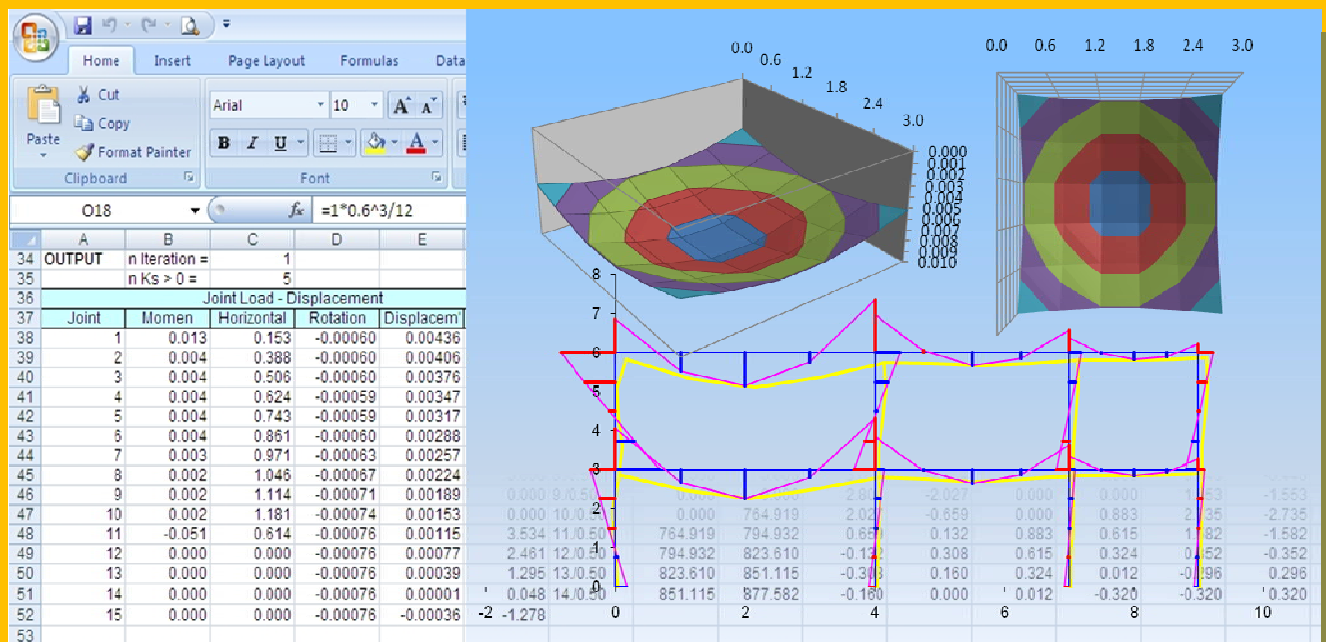


An Introduction to EXCEL for Civil Engineers

From engineering theory
to Excel practice



e-Book

Gunthar Pangaribuan

- This book is intended to introduce a beginner level of using Microsoft Excel in civil engineering practices
- A direct translation

Copyright © Gunthar Pangaribuan 2016. No part of the book may be translated, reproduced or transmitted by any form or by any means without permission in writing from the author. Unless for private use, sharing, distributing and modifying the associated files for any purpose are prohibited.

PREFACE

Microsoft Excel learning is perceived as more attractive from time to time and it is probably the most widely software-learning topic written into books, websites, courses, tutorial videos, groups, etc. Favored by many people because Excel is relatively easy to operate and giving "completely" results by showing spreadsheet form (rows and columns), images, text, tables, charts, and so on. Talking about Excel for applied engineering calculation cannot be separated from the discussion on Visual Basic for Application (VBA) macro, which is the programming language of Microsoft Visual Basic for the automation of certain tasks. This is due to macro like any other programming language capable of doing iterative calculation or repeating the calculation process with ease. There is a unique combination between worksheet as user-interface and VBA, which turned out to be a lot makes it easy for users to create a program.

This book also discussed the depiction in AutoCAD software. Why? Because the drawing creation process can be done through Excel formulas or macros, and this will enhance a series of producing program. The advantage of an AutoCAD drawing creation is no doubt that relies on high image accuracy with a myriad of features it will certainly be a challenge to create drawings. With the ease of working with Excel, coupled with a lot of given examples in this book, it is expected to increase the interest of the reader to create new original application programs. Thus, each model or even a specific model of calculation will be an exciting challenge for a programming job is already enjoyable.

Happy Excel programming!

Jakarta, November, 2015

Gunthar Pangaribuan

CONTENTS

PREFACE	ii
Chapter 1: BASICS OF EXCEL.....	1
1.1 Worksheet and Workbook	1
1.2 Data Type	2
1.3 Formula.....	5
1.4 Built-In Function.....	7
1.5 Array Formula.....	9
1.6 Data Formatting.....	11
1.7 Error Message	11
1.8 Printing	12
1.9 Making Charts	13
1.10 Engineering Drawing.....	15
1.11 Visual Basic for Application	24
1.11.1 Creating Macro	25
1.11.2 Recording Macro	27
1.11.3 Procedure	29
1.11.4 Running Macro	31
1.11.5 VBA Dictionary	32
Chapter 2: EXCEL FUNCTIONS.....	33
2.1 Math and Trigonometry Functions.....	33
2.2 Logical Functions.....	36
2.3 Lookup Functions.....	38
2.4 Text Functions	40
2.5 Data Analysis Functions	43
2.5.1 Linear Regression	43
2.5.2 Polynomial Regression	52
2.5.3 Interpolation	53
2.5.4 Statistical Data.....	59
2.5.5 Circular Reference	65

Chapter 3: CREATING MACRO	69
3.1 Function Procedure.....	69
3.2 Sub Procedure.....	74
3.3 Control Structures.....	76
3.3.1 Looping.....	76
3.3.2 Branching.....	79
3.4 User Defined Function Problems.....	83
3.5 Structure of Program.....	97
3.5.1 Input Output Form.....	97
3.5.2 Work With Modules	98
3.5.3 Tips.....	100
3.6 Chart Macro.....	102
3.7 Manipulation on Program Steps	108
Chapter 4: MATRIX PROGRAM.....	112
4.1 Matrix Definition.....	112
4.1.1 Types of Matrix.....	112
4.1.2 Matrix Operation	115
4.2 Program for Matrix Operations.....	124
4.3 Matrix Method for Structural Analysis.....	132
4.3.1 Upper Structure	132
4.3.2 Sub Structure	134
Chapter 5: NUMERICAL METHOD	135
5.1 Numerical Integration.....	135
5.2 Numerical Differentiation.....	138
Chapter 6: PROGRAM FOR 2D FRAME STRUCTURE ANALYSIS.....	144
6.1 Case Example.....	144
6.2 Sign Convention for Diagram	160
6.3 Application	162
Chapter 7: PROGRAM FOR 2D TRUSS STRUCTURE ANALYSIS.....	163
7.1 Case Example.....	164
7.2 Application	176

Chapter 8: BEAM ON ELASTIC FOUNDATION.....	180
8.1 Case Example.....	182
8.1 Application	187
Chapter 9: Laterally Loaded Structure.....	189
9.1 Case Example.....	189
9.2 Application	196
Chapter 10: One Dimensional Consolidation	199
10.1 Application 1	205
10.2 Application 2	207
Chapter 11: AutoCAD Script File.....	210
11.1 Creating Scripts in Worksheet.....	210
11.2 Creating Scripts in VBA.....	231
REFERENCES	240
Attachment: Program Code.....	241

Gunthar Pangaribuan

Graduated from Indonesia Institute of Technology and earned a bachelor degree in Civil Engineering. Getting started with a career in geotechnical engineering services and became his major work which he has spent over 10 years. During the time, he has created numerous computer programs especially for completion of geotechnical problems using Excel-VBA-AutoCAD - the magic trio he relies on. Some of the programs are presented in this book. In 2012, he joined the oil and gas company as a facility engineer. The current activities and interests include, traveling, social media, tea, music, band, and rock guitar solo.

CHAPTER 1

BASICS OF EXCEL

The work performed by Excel is basically the job of entering data which is then processed to obtain the desired results. It is, in principle, the same as entering data into an electronic calculator. However, the data entered here consists of various types and coupled with existing Excel facilities makes it possible to present the appearance of numbers, text, associated tables, graphs and a database. This makes Excel becomes well integrated to create a reporting text and the problem analysis as well.

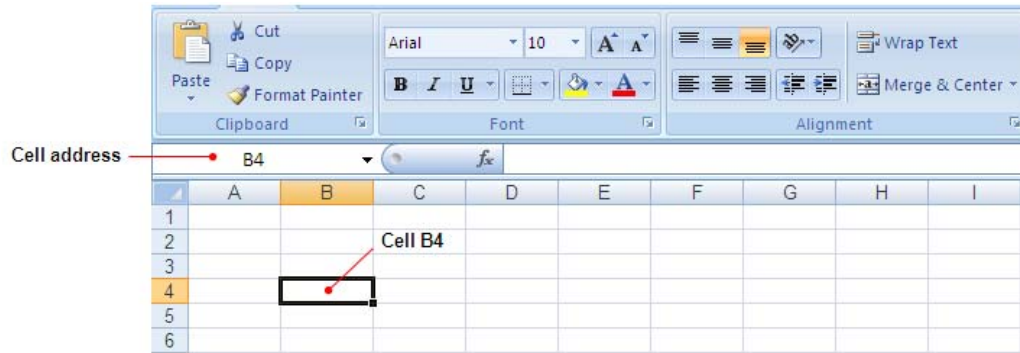
Each version of Excel to be developed to always make changes and additions of new facilities, while is still maintaining compatibility with previous versions. But the changes do not alter the basic features of this software as a worksheet for computing applications. In this book, we will be working with Microsoft Excel 2007.

1.1 WORKSHEET AND WORKBOOK

When opening Excel, by default Book1 is the name of the first workbook. This workbook consists of 3 worksheets named Sheet1, Sheet2 and Sheet3. Excel worksheet is also referred to as a spreadsheet that is the sheet for processing text and numbers.

Figure 1.1 shows the elements of a workbook. At the top of the page there is a title bar displays the workbook name. Underneath, there is a **Ribbon**, new interface introduced in Excel 2007, which is a navigation tool replaces the menu and tool bars in earlier Excel version as a tool of access to Excel commands. All commands are grouped and placed into tabs for particular purposes, thus a tab contains groups of commands.

Worksheet is divided into rows and columns. In Excel 2007, the number of columns and rows have been improved from previous versions. Columns are from A to Z, then AA, AB to XFD (16,384 columns), while the rows start from 1 to 1,048,576. The intersection point of column and row forms a cell as a place to fill data. Each cell has an address referred to by a column and a row, for example cell B4 is a cell in column B and 4th row. The address of selected cell can be seen in the Name Box below the Ribbon.



A collection of cells is called a cell range, forming an array extending horizontally or vertically. For example range A1: A5 is a collection of cells from A1 to A5, forming an array of 5 x 1 or the range A1: C5 form a 5 x 3 array, and so on.

The name of the worksheet can be replaced by other names, by clicking twice on the sheet tab, press **Delete** to clear the name, then write the new name; or by right-clicking on the tab and then on the shortcut menu click **Rename** to change its name.

1.2 DATA TYPE

The input data in the spreadsheet can be divided into some types that are:

- a. Text: alphabet characters and text: A, B, -Z, AB, A2, Computers ...
- b. Numbers: numerical data: 1,2,3,0, -1, -2,4,5.85...
- c. Date: the date data typically refers to the setting of a computer calendar or formulated in the calculation.
- d. Hours: Data of hours generally are referred to computer time setting or formulated in the calculation.
- e. Formula: mathematical expressions that calculate two or more values produce a new value.
- f. Function: a function that is used for various applications such as calculations, finance, mathematics and trigonometry, statistics, database, logic and others.

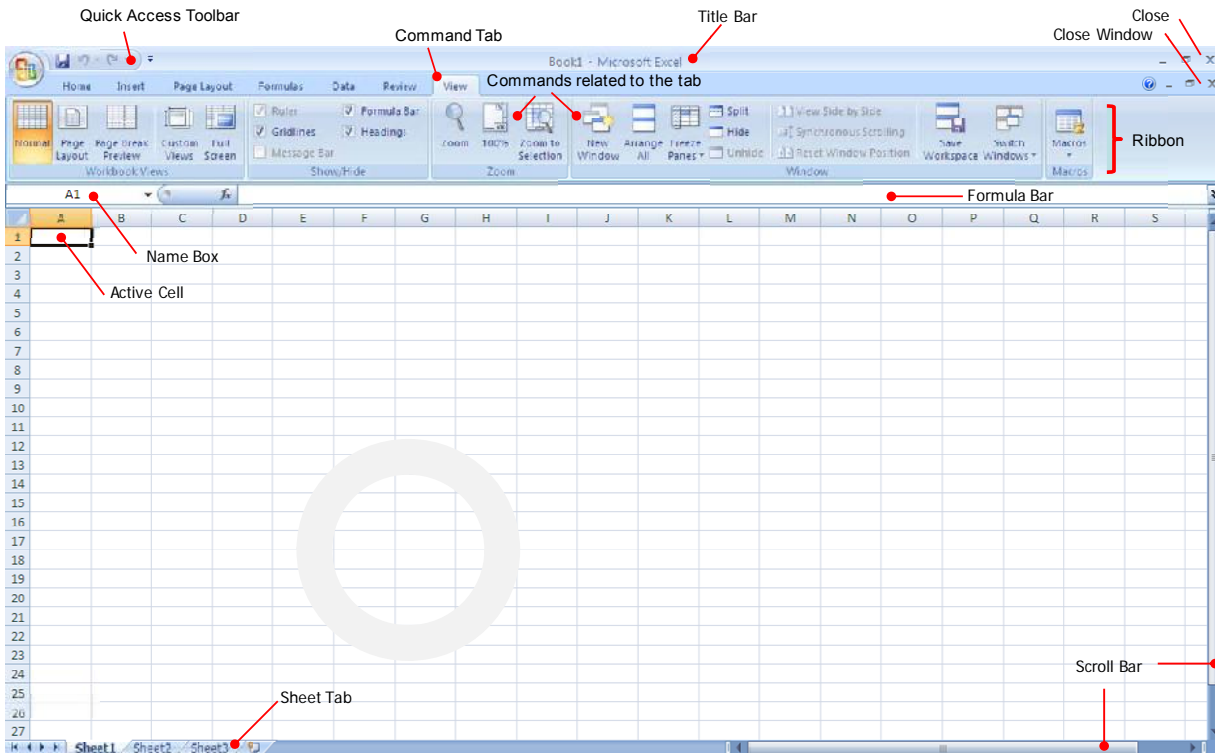








Figure 1.1: Excel 2007 Worksheet

Table 1.1: Excel Basic Commands

Name	Shortcut or Icon	Description
Office Button		Icon to basic commands below:
Save		Save a file with the name of the workbook that is being done
Save As	Alt+FA	Store the file with another name into a folder and directory, or to a disc. By default file extension of .xlsx is given (free macros). If the file contains macro (VBA) and needs to activate each time it is opened, an extension of .xlsm extension must be given.
New		Open a new workbook
Open		Open an existing workbook
Print		Print a worksheet through the printer device
Close		Close the workbook (Close Window icon)







Exit		Close the program and exit Excel (icon Close)
Quick Access Toolbar:		
Undo		Cancel the last job
Redo		Repeat the last job
Tab HOME		
Copy		Copy the data of a cell or range of cells to another location. To select the data to be copied or cut, then click on the cell or range of cells where the data is then press Ctrl+C
Cut		Move the data to another location. Commands such as Cut, Copy and Paste can be found in the pop-up menu by clicking the right mouse button on the highlighted cells or by pressing Ctrl+X
Paste		Put the data into a new cell or range of cells. Paste command is done after preceding Cut or Copy. To place the data, click the new location, then press Enter or by pressing Ctrl+V
Insert > Cells	Alt+IE	Display Insert dialog box. Used to insert a new cell, row or column at the pointer position (highlighted cell)
Insert > Sheet Rows	Alt+IR	Insert a new row at the pointer position
Insert > Sheet Columns	Alt+IC	Insert a new column at the pointer position
Format .> Row Height	Alt+HO (Home > Format)	Format row, consists of the row height setting, to hide and to unhide row
Format .> Column Width	Alt+HO (Home > Format)	Format column, consists of the column width setting, automatic width (autofit selection), hide and show columns, and showing information of the standard column width
Format > Cell	Alt+HO (Home > Format)	Format cell such as numbers, text, font used, borders, colors and worksheet protection

Table 1.2 Operators and mathematical relationships

Operator	Description
+	Summation
-	Subtraction
*	Multiplication
/	Division
%	Percent
^	Exponentiation
Relationships	
=	Equal to
>	Greater than
<	Less than
<>	Not equal to
>=	Greater then or equal to
<=	Less than or equal to

1.3 FORMULA

By definition, a formula is a mathematical expression to calculate the results of two or more values. Formula can consist of numbers, mathematical operators, functions, reference cell or range of cells. Cells and cell range are often given a name, for example "A" to B2, or "B" for a range of D2: D6. Naming cells will be discussed later in this section. All formulas begin with an equal sign (=). For example, in cell B4 is written the following formula:

	A	B	C	D
1				
2		4		
3		5		
4		=B2+B3		
5				

=B2+B3 summing the data in cell B2 to cell B3

=B4*B5 multiplying the data in cell B4 to cell B5

=SUM(D2:D4) summing the number of cells D2, D3 and D4

When a formula to be copied, it needs to be considered what is the type of cell in the formula. Relative cell is the cell that will adjust to its new place when copied. Cell C1 that contain formula =A1+B1 will be =C2+D2 in Cell E2. The result is as shown in Figure 1.2.

Absolute cell is a cell that does not change the address if copied to another place. The notation is to add \$ before the name of a column or row number, for example: \$A\$4. Adding \$ in the name of any column or row number will only change one address. Column A in \$A4 will remain when copied, but the row number will adjust to its new location. And vice versa with cell A\$4. Such cells are called semi-absolute cell.

	A	B	C	D	E	F
1	20	30	=A1+B1			
2					=C2+D2	
3			=\$A\$1+B1			
4					=\$A\$1+D2	
5			=\$A1+B1			
6					=\$A2+D2	
7			=A\$1+B7			
8					=C\$1+D8	
9						
10						

Figure 1.2: Copied formula and the results in spreadsheet

Formula stating cell relationships or range of cells, for example, =B4+C30 or =SUM (A4: C20) are more difficult to read than the mathematical relationship with the more practical variable name, for example, =x+n or SUM(B). Name of cell or cell range is created by clicking **Formula > Define Name**. This name applies to all sheets in a workbook. If it only represents the specific sheet, for example, Sheet1, the writing is Sheet1!x, i.e. to the variable x in Sheet1. The use of variables is highly recommended because it will simplify a formula. The names used and the locations can be seen in **Name Manager** and you can also create, edit and delete a name.

	A	B	C	D
1				
2			4 — x	
3			5 — n	
4		=x*n		
5				

1.4 BUILT-IN FUNCTION

Excel has many built-in functions to build complex formulas, some of them are shown in Table 1.3. Functions such as mathematic and trigonometry or statistic, for example, are the most common functions used in engineering practices e.g. to produce calculation data sheet in laboratory soil testing. Before using a function, it is advised to know well the function and its arguments. The reference could be found the Help menu or press F1.

Table 1.3 Excel Built-in Functions

A. Math and Trigonometry	
ABS(num)	Returns the absolute value of a number
ACOS(num)	Returns the arccosine of a number
ACOSH(num)	Returns the inverse hyperbolic cosine of a number
ATAN(num)	Returns the arctangent of a number
EXP(num)	Returns e raised to the power of a given number
FACT(num)	Returns the factorial of a number
INT(num)	Rounds a number down to the nearest integer
LOG(num, base)	Returns the logarithm of a number to a specified base
LOG10(num)	Returns the base-10 logarithm of a number
MDETERM(array)	Returns the matrix determinant of an array
MINVERS(array)	Returns the matrix inverse of an array
MMULT(array1,array2)	multiplying of 2 arrays
PI()	Value of pi = 3.141592654
RAND()	Random value between antara 0 dan 1
SIGN(num)	Sign of number. Sign 1 or 0 or -1 if the number is positive, zero or negative, respectively.
SIN(num)	Returns the sine of the given angle

SINH(num)	Returns the hyperbolic sine of a number
SQRT(num)	Returns square root of a number
SUM (num1,num2,...)	Add the numbers
SUMPRODUCT(array1,array2)	Multiplies corresponding components in the given arrays
TAN(num)	Returns the tangent of a number
TANH(num)	Returns the hyperbolic tangent of a number
B. IS Function	
ISBLANK(value)	TRUE if value is empty
ISLOGICAL(value)	TRUE if value is logical value (TRUE or FALSE)
ISNUMBER(value)	TRUE if value is a number
ISTEXT(value)	TRUE if value is a text
C. Statistic	
AVERAGE(num1,num2,...)	Average value of the numbers
COUNT(value1,value2,...)	Counts the number of cells that contain numbers within the list of arguments
COUNTA(value1,value2,...)	Counts the number of cells that are not empty within the list of arguments
LINEST(y's,x's,const,stats)	Returns the parameters of a linear trend. Const and stats are logical values (see in Excel Help)
MAX(num1,num2,...)	Maximum value in a list of arguments
MIN(num1,num2,...)	Minimum value in a list of arguments
SLOPE(y's,x's)	Returns the slope of linear regression line
INTERCEPT(y's,x's)	Returns the intercept of the linear regression line
TREND(y's,x's, new x's,const)	Returns values along a linear trend. Const is a logical value specifying whether to set the constant $b = 0$ in $y = mx + b$ relationship
D. Lookup and Reference	
COLUMNS(reference)	Returns the column number of the given reference
INDEX(array,row_num,column_num)	To choose a value from a reference or array
ROWS(reference)	Returns the row number of the given reference

TRANSPOSE(array)	Returns the transpose of an array
HLOOKUP(value,table,row_in)	Search for a value based on the row index in the table data arranged horizontally
VLOOKUP(value,table,col_in)	Search for a value based on the column index in the table data arranged vertically
E. Logical	
AND(logical1,logical2,..)	Returns TRUE if all of its arguments are TRUE
IF(log_value,value_if_true,value_if_false)	Specifies a logical test to perform
NOT(logical)	Reverses the logic of its argument: NOT(TRUE) = FALSE
OR(logical1,logical2)	TRUE if one of its argument is TRUE
F. Text	
CHAR(num)	Returns the character specified by the code number
EXACT(text1,text2)	Checks between two text strings and returns TRUE if they are exactly the same, otherwise returns FALSE
FIND(text1,text2,start_num)	Finds one text value within another with start number
LEFT(text, num_character)	Returns the first character based on the specified number of characters
LEN(text)	Returns the number of characters in a text string
RIGHT(text, num_character)	Returns the last character or characters in a text string, based on the specified number of characters
TRIM(text)	Removes spaces from text except for single spaces between words
UPPER(text)	Converts text to uppercase

1.5 ARRAY FORMULA

By Excel definition, array formula is a formula that can perform multiple calculations and then return either a single result or multiple results. Array formulas act on two or more sets of values known as array arguments. Each array argument must have the same number of rows and columns. For an example, the built-in **LINEST** returns two results

which are Slope and Y-Intercept. This function gives linear regression lines that fit with the known x and y-values, as shown in Figure 1.3.

	A	B
1	x	y
2	-1.0	-5.0
3	-0.5	0.0
4	1.0	5.0
5	2.0	4.0
6	3.0	0.5
7	4.0	-5.0
8	5.0	-12.0
9	Slope	Y-intercept
10	-1.204	0.537

9	Slope	Y-intercept
10	= {LINEST(B2:B8,A2:A8)}	{=LINEST(B2:B8,A2:A8)}

Figure 1.3: Array formula for linear regression and the results

Array formula is entered by pressing **Ctrl+Shift+Enter** and Excel will automatically insert the formula in brackets ({}). To display values returned by **LINEST** function, make **LINEST** formula in cell A10 as shown in Figure 1.3, and then select the range A10: B10 > press **F2** > **Ctrl+Shift+ Enter**.

Here are some other examples of array formula:

{=SUM(A2: B2*A3:B3)} is a single result

{=TREND(A2:A6,B2:B6)} is a separately 5 results

Alternatively, the INDEX function can be used to return the values, as shown below. **LINEST** has two results indexed sequentially that are Slope and Y-intercept.

9	Slope	Y-intercept
10	-1.204	0.537

9	Slope	Y-intercept
10	=INDEX(LINEST(B2:B8,A2:A8),1)	=INDEX(LINEST(B2:B8,A2:A8),2)

1.6 DATA FORMATTING

Data and cells formatting can be done through the **Home** tab > **Format** > **Cells**. In the **Format Cells** dialog box, there are several sheet tabs which are **Number**, **Alignment**, **Font**, **Border**, **Fill** and **Protection**. Number formatting in **Number** tab is divided into several categories such as general, number, date and so on. For example, for custom category or user preference is shown in Table 1.4. **Alignment** is the setting for layout of text in a cell such as vertical and horizontal alignment, control and orientation of text in a cell. **Fonts** are the settings that relate to the character, such as the font face used (Arial, Times New Roman, Tahoma, ...), style (italic, bold ...), size, color and so on. **Border** is to make the margins, line types (straight, dotted, thin, thick), and color. **Fill** to create a displayed cell with shading (horizontal, diagonal, dot, ...), shading color and the color of the cell background. **Protection** is used to select the condition (locked or not) of data and formulas (hidden or not) in a cell when worksheet protection is enabled. To protect a worksheet click **Review** tab > in **Changes** group click **Protect Sheet**.

Table 1.4: Custom format for numbers

Format	Displayed
0.00	2343.00
0.00E+00	2.34E+03
##0.0+0	2.3E+3
#,##0.00	2,343.00
#,##0.00%	234,300.00%
0.0 "m"	2343.0 m

Column width and row height can be adjusted by: first way, by dragging the mouse (hold the left mouse button while moving) when the pointer position at the boundaries of the column and row headings (there is a sign "→" or "↓"), or second way is by right-clicking the mouse on the column or row heading and from the shortcut menu click **Row Height** or **Column Width**, then enter desired value in the textbox. The other way is through **Home** > **Format**.

1.7 ERROR MESSAGE



Error message will be appeared if a formula is not working as it should be, for examples, arguments in a formula are not complete, the data type does not match, or wrong in developing the logic. Another case, for example is division by 0 (zero) or the width of the

column is not wide enough to display a result. The message started with the # sign followed by the type of error. Table 1.5 shows some error messages that may appear.

Table 1.5 Error Messages

Messages	Remarks
#####	Cell contains number, date or time that is wider than the cell width or minus result in the date and time format
#N/A	(Not Available) when a value is not available to a function or formula.
#DIV/0!	Error due to division by zero (0).
#NAME?	Excel does not recognize text in a formula
#REF!	When a reference cell is not valid
#VALUE!	The types of arguments used in the function are wrong. The argument can be a numerical value, text, name, label, reference cell, cell range and function.

1.8 PRINTING

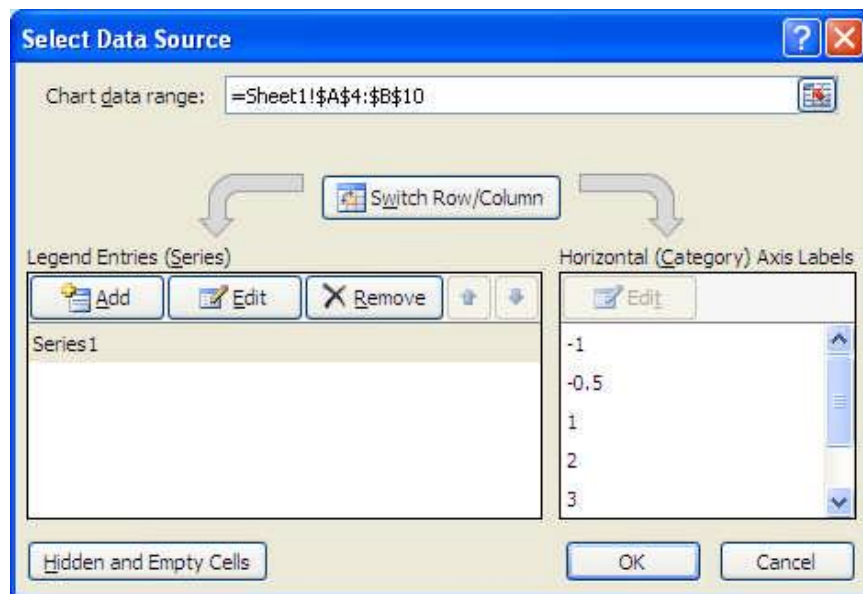
Before printing the worksheet to paper, you can set the area by blocking the area that will be printed (press **Shift** + "**→**" or "**↓**" or drag the mouse). Afterward, click the **Page Layout** tab > **Print Area** > **Set Print Area**. The print area can be seen in  **Office Button** > **Print** > **Print Preview** or to press the shortcut key **Ctrl+F2**. Printing then performed through  > **Print** > **OK** or pressing **Ctrl+P**.

The limits of printing page can also be specified through the **View** tab > **Page Break Preview** instead current **Normal** view. This will show all the data in worksheet to be printed with a page number. You can also adjust the boundaries of the printing page by dragging the mouse on the blue boundary line.

Page configuration is then set through the **Page Layout** tab. Click the **Print Titles** in the **Page Setup** group, and a **Page Setup** dialog box will appear with several sheets tabs which are **Page**, **Margins**, **Header/ Footer**, and **Sheet**. **Page** is to select the orientation and size of paper, and the scale of worksheet on paper; **Margins** is to set the boundaries of the text in the paper; **Header/Footer** is to make custom header and footer; and **Sheet** is to set print area, showing grids, row and column headings, and the print order.

1.9 MAKING CHARTS

A set of data can be well delivered and communicated through a chart image shown the correlation between the data. In Excel there are many types of chart depictions such as Bar, Column, Line, Pie, Area and so on, plus a 3-dimensional view. Chart is created through the **Insert** tab > **Chart**, and there will be **Charts** group as shown in Figure 1.4. For this example, we will use the data from Figure 1.3. Select **Scatter** > **Scatter with Smooth Lines and Markers** because smooth line between points to be made. Click **Select Data** on the Ribbon interface to display the **Select Data Source** dialog box as shown below.



Enter x and y-values data in the **Chart Data Range** by clicking a button at the right side of the input box and then point the pointer to cell A2 and drag the mouse from cell A2 to cell B8. Click **OK** and a chart will be created as shown in Figure 1.5. In the chart, the regression line is also shown with the associated equation. Regression line is created by clicking mouse on the data in the chart (at point or a line), then right click > **Add Trendline** > **Linear** > checked the **Display Equation on Chart**. Furthermore, if you want to work with the chart thoroughly, click the graph area to display **Chart Tools** tab where commands are grouped in accordance with their task name, which are **Type**, **Data**, **Chart Layouts**, **Chart Styles** and **Move Chart**.



Figure 1.4: Charts group displays chart type options

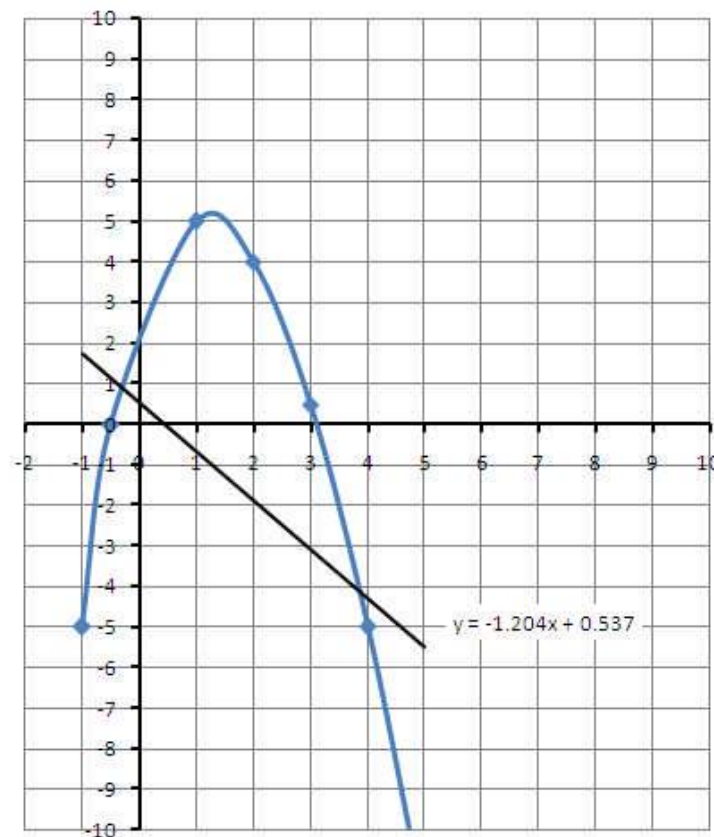


Figure 1.5: Example of Scatter with smooth lines chart type

Note:

As designed for office applications, Excel offers various types of charts. However, we will not discuss many different types of charts, and but only focused on **Scatter chart type with straight lines** that frequently used in this book.

1.10 ENGINEERING DRAWING

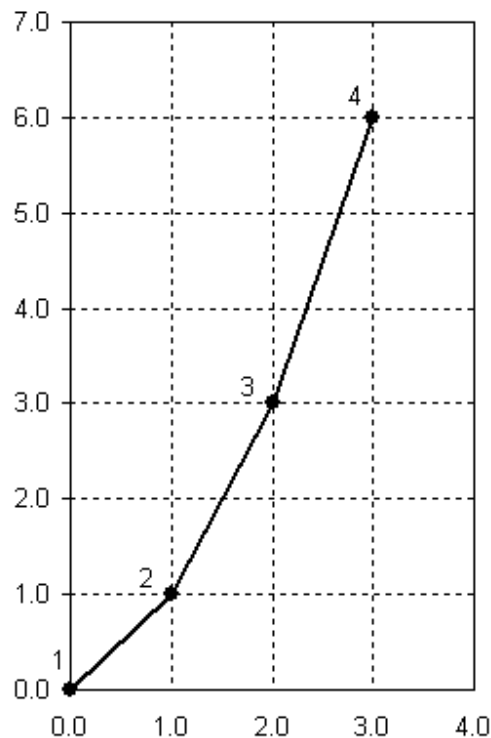
Chart type of **XY Scatter** is suitable for use in Civil engineering practices in giving a drawing presentation that forms lines or elements of structure. The reason is that every line can stand alone, so that easily modified and formulated for the intended drawing.

A straight line of **XY Scatter** chart type is determined by the given coordinates of both ends required for its input data. If both ends of the line called joint, then a line segment is formed by the coordinates of two joints. Thus, there will be a series of joint data put into a worksheet table to create straight lines.

To make it easier to understand the intent and purpose of this discussion, below is given examples and the followed steps.

Example 1

DDRCreat 3 continuous lines drawing through 4 points (1 to 4), as shown in figure below.



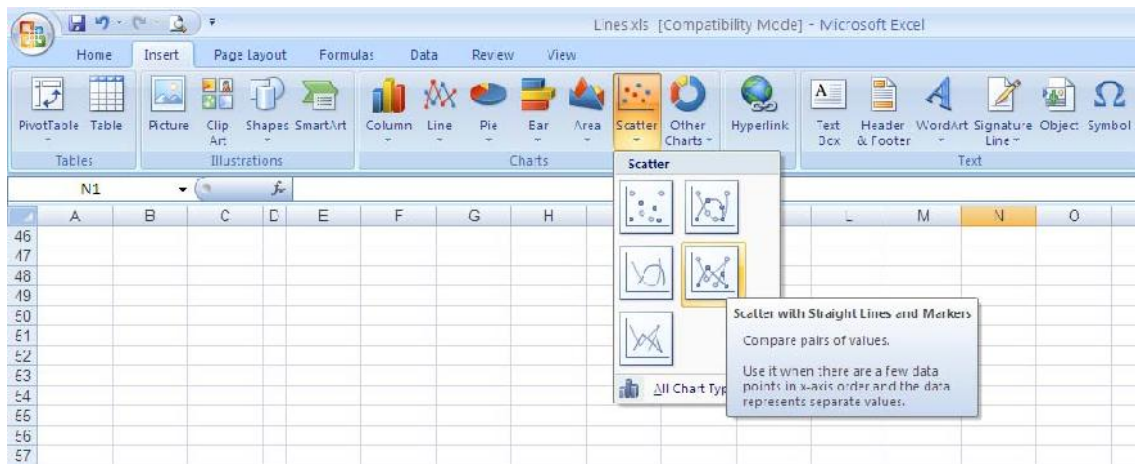
The joint coordinates are:

	x	y
Joint 1	0	0

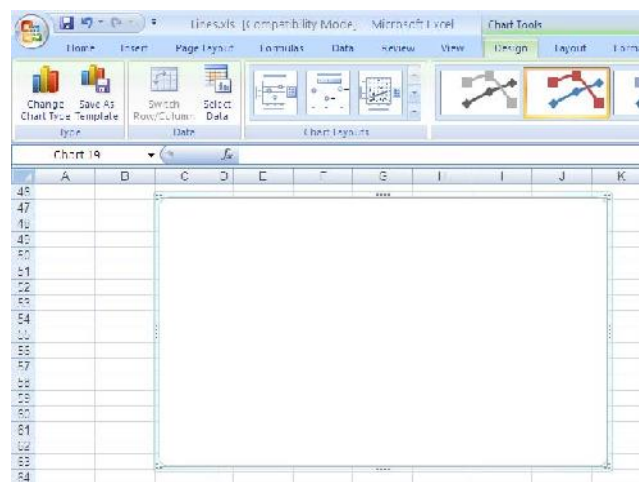
Joint 2	1	1
Joint 3	2	3
Joint 4	3	4

To create continuous lines as the figure above, it takes the following steps:

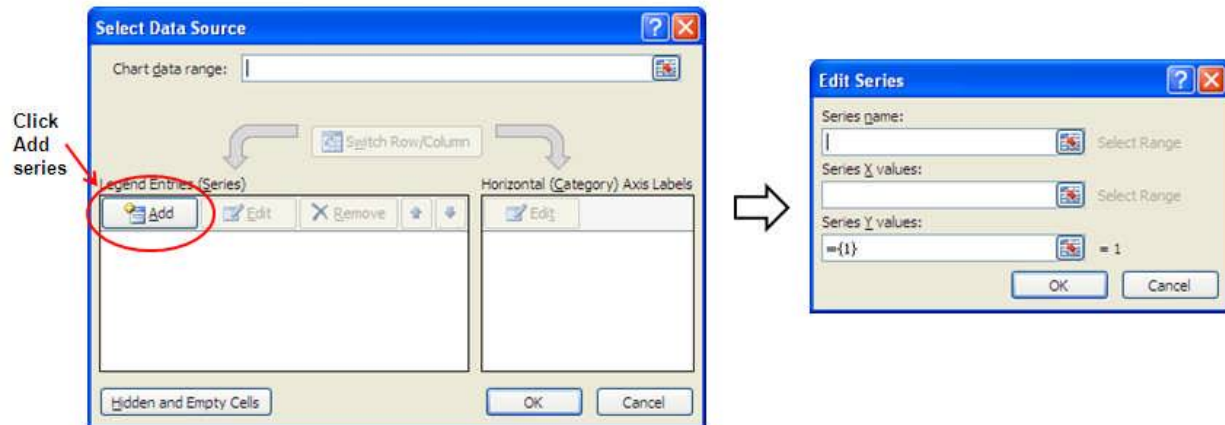
1. First step: click on the **Insert** tab > **Scatter** > **Scatter with Straight Lines and Markers**.



2. **Chart Area** displays nothing because no data on it as shown below. Click **Select Data** on the Ribbon interface to display the **Select Data Source** dialog box.



3. A **Select Data Source** dialog box is displayed as shown in the figure below (left side). Click **Add** to display the **Edit Series** dialog box (right side). Each series that added (clicking Add) represents a **line** that requires a **pair** of x and y values as inputs. Thus, a line series shall consist of the coordinates of the both ends, the one to be (x1, y1) and the other is (x2, y2).



4. Following Step 3 is to enter the line coordinate values i.e. series x and series y-values. When needed, give a name for the series in the **Series name** input box.
5. Click OK to end.

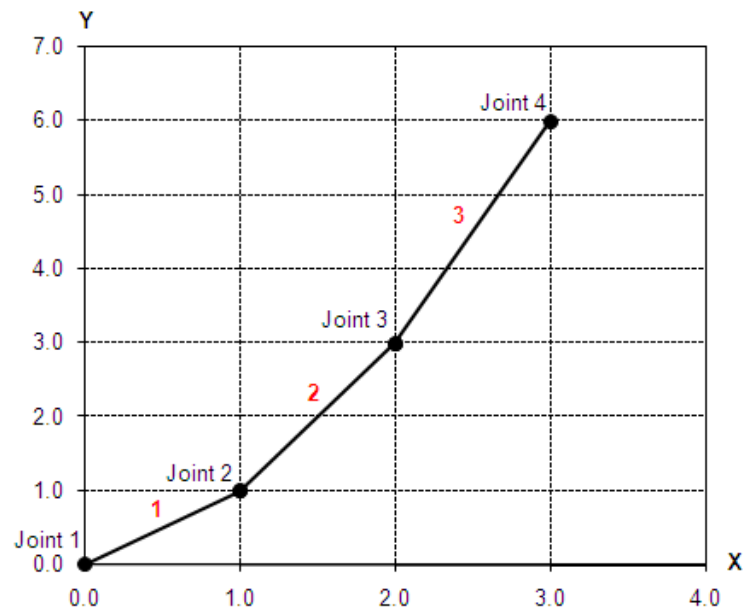
The intended joint coordinate and lines for chart depiction can be summarized as below:

Line	Joint		Coordinate			
			x1	y1	x2	y2
1	1	2	0.0	0.0	1.0	1.0
2	2	3	1.0	1.0	2.0	3.0
3	3	4	2.0	3.0	3.0	6.0

Thus it can be stated that the first line (no.1) is a line created by given joint 1 and 2 coordinates (refers to the figure).

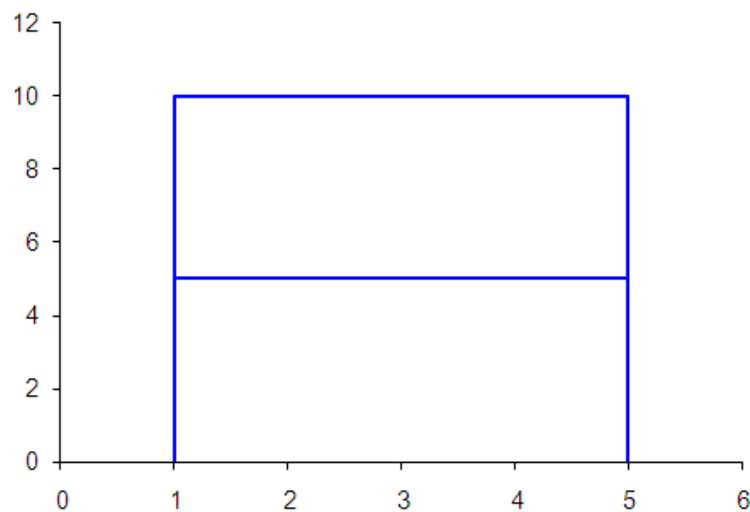
In step 4, what we do is to enter x1 and x2 range of values (colored yellow) into the **Series X values** input box, while the **Series Y values** is filled by column y1 and y2 range of values (colored green). To enter x-coordinates, press **Ctrl+left mouse button click** at x1 column and repeat **left mouse button click** at x2 column to get the range of cells. Do the same way to enter y-coordinates.

Repeat step 3 to 5 to make 2nd and 3rd line. The result will be shown as below:

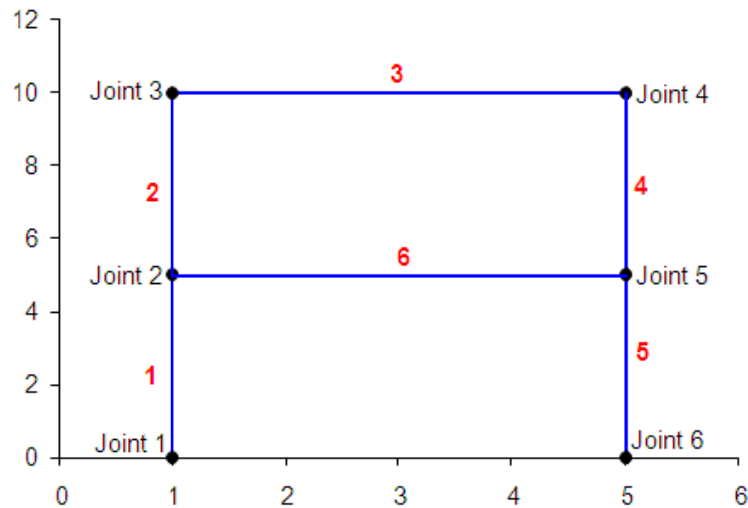


Example 2

Draw a simple picture of one floor building as below:



As in Example 1, to simplify the portrayal of the drawing it needs to produce joints and lines coordinate tables. The building is composed of 6 joints and 6 straight lines that can be built up with the following numbering system:



Based on the picture above, the following tables can be formed:

[illegible]

The joint coordinates for assigning lines on the right table is obviously the repetition of inputting task of the previous joint coordinates on the left table. To avoid repeated data entry manually, **VLOOKUP** function can be used by adopting the x, y joint coordinates data of the left table as a reference table to obtain x1, y1 and x2, y2 coordinates of the right table. The formulas in the worksheet are shown as follows:

Joint reference (no.)

Table of coordinates

Joint X-value

Joint Y-value

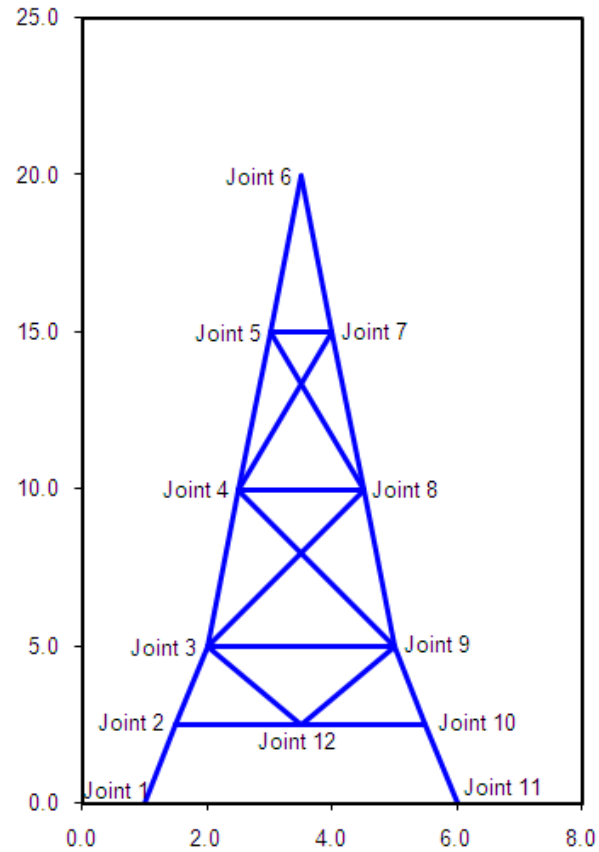
Line	Joint	x1	y1	x2	y2
1	1	=VLOOKUP(F5,SASS,SC\$10,2)	=VLOOKUP(F5,SASS,SC\$10,3)	=VLOOKUP(G5,SASS,SC\$10,2)	=VLOOKUP(G5,SASS,SC\$10,3)
2	2				
3	3				
4	4				
5	5				

copy formula

To get the drawing, do the sequence steps as in Example 1.

Example 3

Draw the following truss structure that consists of 21 lines and 12 joints. The joint coordinates are tabulated as below:



Joint	x	y
1	1.0	0.0
2	1.5	2.5
3	2.0	5.0
4	2.5	10.0
5	3.0	15.0
6	3.5	20.0
7	4.0	15.0
8	4.5	10.0
9	5.0	5.0
10	5.5	2.5

11	6.0	0.0
12	3.5	2.5

Before creating the drawing of truss, it is convenient to form lines coordinates as required for **Series X** and **Series Y** input data. The coordinates of truss lines are tabulated as the following:

Line	Joint		x1	y1	x2	y2
1	1	2	1.0	0.0	1.5	2.5
2	2	3	1.5	2.5	2.0	5.0
3	3	4	2.0	5.0	2.5	10.0
4	4	5	2.5	10.0	3.0	15.0
5	5	6	3.0	15.0	3.5	20.0
6	6	7	3.5	20.0	4.0	15.0
7	7	8	4.0	15.0	4.5	10.0
8	8	9	4.5	10.0	5.0	5.0
9	9	10	5.0	5.0	5.5	2.5
10	10	11	5.5	2.5	6.0	0.0
11	2	12	1.5	2.5	3.5	2.5
12	12	10	3.5	2.5	5.5	2.5
13	3	12	2.0	5.0	3.5	2.5
14	12	9	3.5	2.5	5.0	5.0
15	3	8	2.0	5.0	4.5	10.0
16	3	9	2.0	5.0	5.0	5.0
17	4	9	2.5	10.0	5.0	5.0
18	4	8	2.5	10.0	4.5	10.0
19	4	7	2.5	10.0	4.0	15.0
20	5	8	3.0	15.0	4.5	10.0
21	5	7	3.0	15.0	4.0	15.0

As the previous examples, the line coordinates data entry used **VLOOKUP** function. The formulas writing is also to follow the same manner as the previous example. To obtain the truss drawing, do the sequence steps as in Example 1.

Next example is to draw truss structure subjected to horizontal load so that each joint will be shifted as far as n unit. Suppose that the displacement at the joints will be from 0 at supports to maximum +0.5 unit at joint 6 (at the top), so it will be in a range of $0 < n < 0.5$ to the right direction. The table of joints coordinates is then modified and labeled for two conditions, before and after loading, where n is used in the calculation. It will show as follow:

	A	B	C	D	E
4					
5		Coord Before		Coord After	
6	Joint	x	y	x	y
7	1	1.0	0.0	1.0	0.0
8	2	1.5	2.5	=B8+0.1	2.5
9	3	2.0	5.0	=B9+0.2	5.0
10	4	2.5	10.0	=B10+0.3	10.0
11	5	3.0	15.0	=B11+0.4	15.0
12	6	3.5	20.0	=B12+0.5	20.0
13	7	4.0	15.0	=B13+0.4	15.0
14	8	4.5	10.0	=B14+0.3	10.0
15	9	5.0	5.0	=B15+0.2	5.0
16	10	5.5	2.5	=B16+0.1	2.5
17	11	6.0	0.0	6.0	0.0
18	12	3.5	2.5	=B18+0.2	2.5
19					

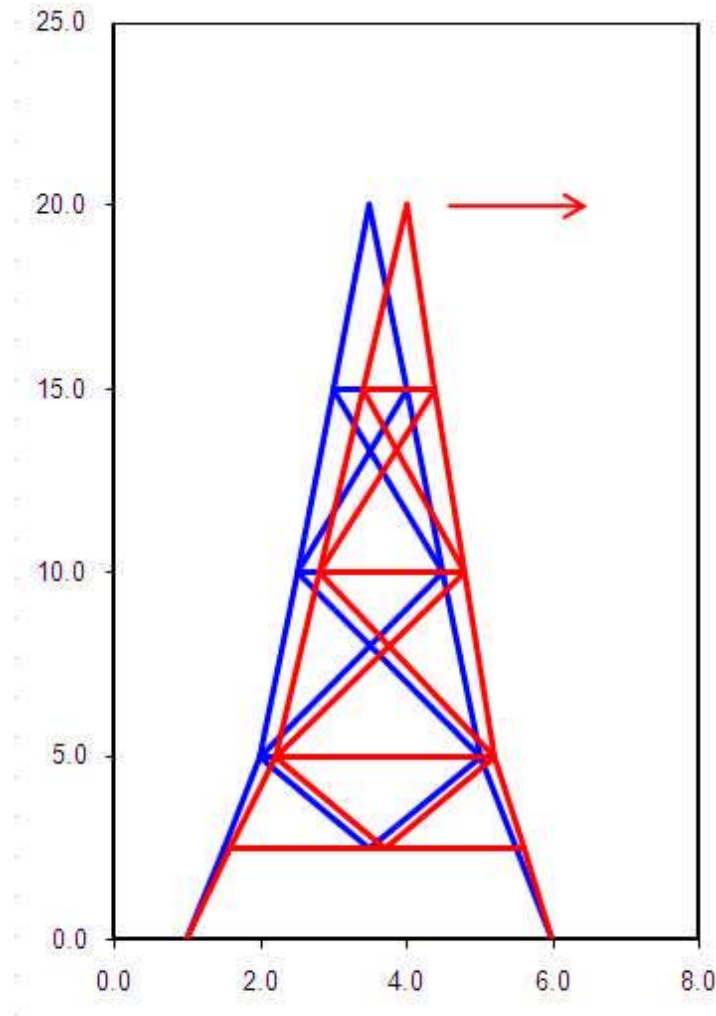
It is noticed that only the x-values will be changed by adding the horizontal displacement value of n , from 0 at Joint 1 to a maximum of 0.5 units at the Joint 6. Joint 1 and 11 are fixed supports so their positions remain in place before and after loading.

After forming the joints coordinates table, next is to create lines coordinates table of **Series X** and **Y**. There are now 42 **Series X** and **Y** to be created for the condition of before and after loading. Table below shows lines coordinates before and after loading:

	F	G	H	I	J	K	L	M	N	O	P	Q	R
4													
5					Coord Before					Coord After			
6		Line	Joint		x1	y1	x2	y2		x1	y1	x2	y2
7		1	1	2	1.0	0.0	1.5	2.5		1.0	0.0	1.6	2.5
8		2	2	3	1.5	2.5	2.0	5.0		1.6	2.5	2.2	5.0
9		3	3	4	2.0	5.0	2.5	10.0		2.2	5.0	2.8	10.0
10		4	4	5	2.5	10.0	3.0	15.0		2.8	10.0	3.4	15.0
11		5	5	6	3.0	15.0	3.5	20.0		3.4	15.0	4.0	20.0
12		6	6	7	3.5	20.0	4.0	15.0		4.0	20.0	4.4	15.0
13		7	7	8	4.0	15.0	4.5	10.0		4.4	15.0	4.8	10.0
14		8	8	9	4.5	10.0	5.0	5.0		4.8	10.0	5.2	5.0
15		9	9	10	5.0	5.0	5.5	2.5		5.2	5.0	5.6	2.5
16		10	10	11	5.5	2.5	6.0	0.0		5.6	2.5	6.0	0.0
17		11	2	12	1.5	2.5	3.5	2.5		1.6	2.5	3.7	2.5
18		12	12	10	3.5	2.5	5.5	2.5		3.7	2.5	5.6	2.5
19		13	3	12	2.0	5.0	3.5	2.5		2.2	5.0	3.7	2.5
20		14	12	9	3.5	2.5	5.0	5.0		3.7	2.5	5.2	5.0
21		15	3	8	2.0	5.0	4.5	10.0		2.2	5.0	4.8	10.0
22		16	3	9	2.0	5.0	5.0	5.0		2.2	5.0	5.2	5.0
23		17	4	9	2.5	10.0	5.0	5.0		2.8	10.0	5.2	5.0
24		18	4	8	2.5	10.0	4.5	10.0		2.8	10.0	4.8	10.0
25		19	4	7	2.5	10.0	4.0	15.0		2.8	10.0	4.4	15.0
26		20	5	8	3.0	15.0	4.5	10.0		3.4	15.0	4.8	10.0
27		21	5	7	3.0	15.0	4.0	15.0		3.4	15.0	4.4	15.0
28													

The reference range of data table in **VLOOKUP** function is now range A7 to E18. Thus, the coordinates of the lines before and after loading are respectively referred to column BC and DE or column index of 2,3, and 4,5.

The drawing of truss lines before and after the horizontal loading are shown in chart below, which are in blue and red, respectively.




1.11 VISUAL BASIC FOR APPLICATION

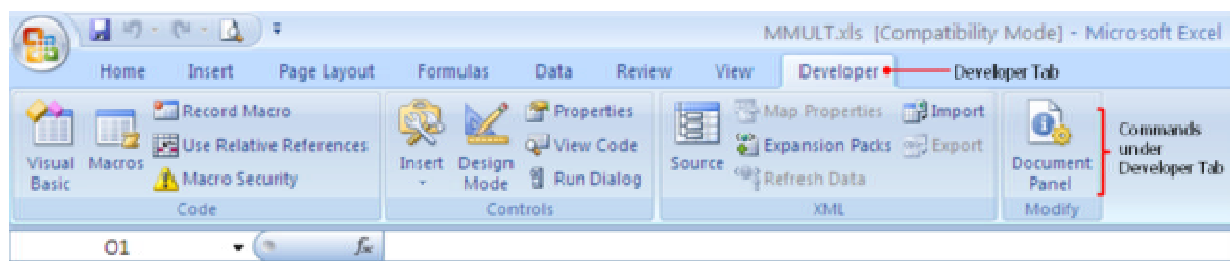
One of the strong points of Excel is its macro capability of using macro programming language to shorten and simplify the repetitive works (automation task). It can not be separated, however, from the presence of Microsoft Visual Basic application which is by default embedded in as an Excel command menu. The Visual Basic application for the repetitive works or macros in an application program (e.g. Excel or Word) is called **Visual Basic for Applications (VBA)**. In addition to Microsoft Office, many of non-Microsoft software also use VBA, such as AutoCAD, CorelDraw, Visio and Norton.

VBA can also serve as a programming language to solve many problems in science and technology field, for instance to solve a complex iterations and the analysis of civil structures that are not easily or can not be done rely on the built-in functions and spreadsheet standard commands. In this respect, the analogy of using macro is therefore a repetitive calculation in an effort to get new output when new input is entered.

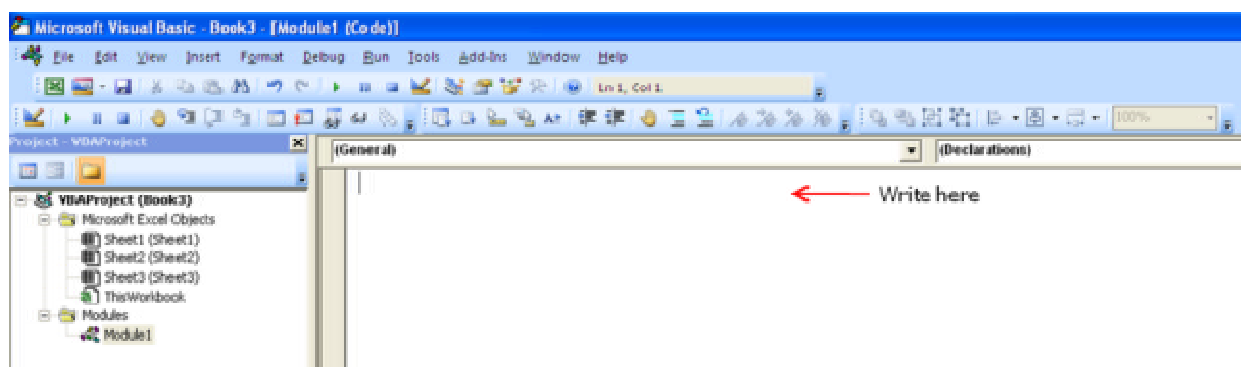
Microsoft Visual Basic as well as Excel VBA is an Object Oriented Programming, which reads and analyzes objects exposed by Excel through object library. The reference to these objects is stored in Microsoft Excel Object Library (EXCEL12.OLB). In a programming structure for a calculation application, Excel exposes the property of an object which is the value of input data that to be processed by VBA and then returns the result to a worksheet. So, in writing inputs what we have actually done is storing object value into a variable, or the value of the variable = value of cell property of the worksheet (= Range object, that will be mentioned next).

1.11.1 CREATING MACRO

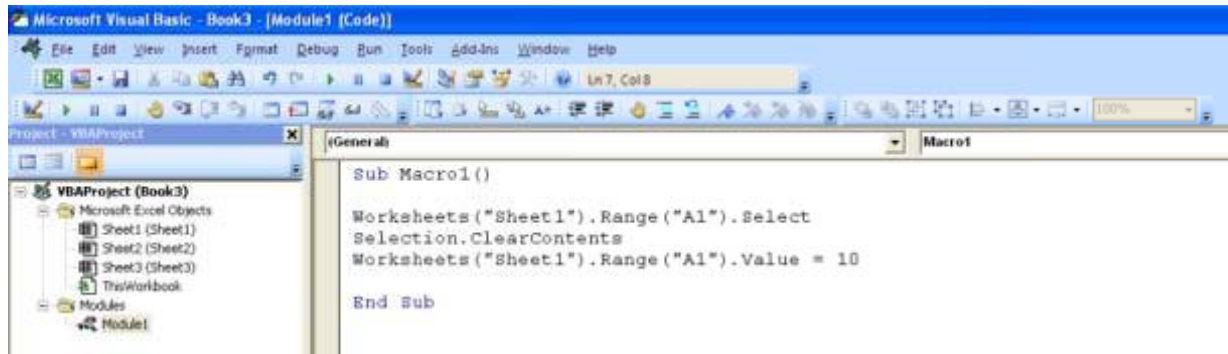
To create macros, activate the Developers tab through  > **Excel Options** > **Popular** > **Show Developer tab in the Ribbon**, so it looks like this:



To go to the window where the VBA programming language is created, click **Visual Basic** to open **Visual Basic Editor (VBE)** window > click the **Insert** menu > **Module** to show the following window:




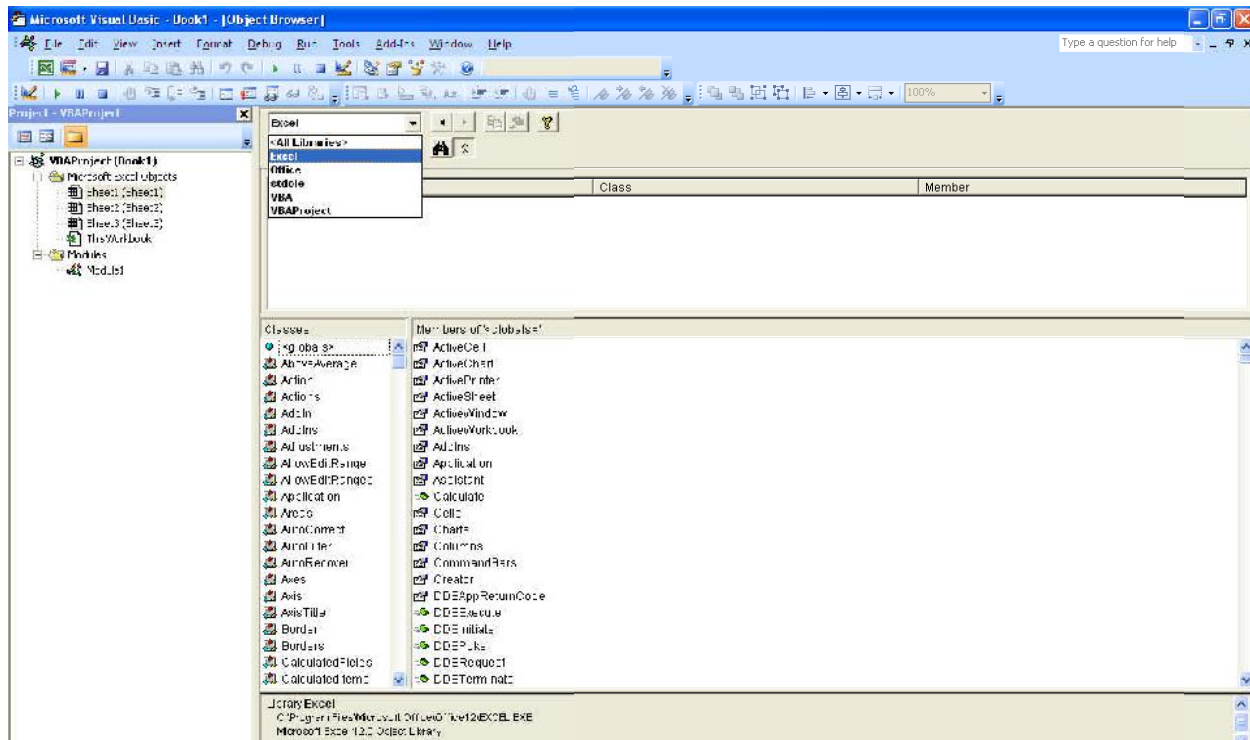
VBA programming language is written in a module that has been added before by **Insert** > **Module**. Below is an example of VBA programming language (program code) in a procedure in a module:



VBA reads, analyzes and manipulates Excel's objects through their properties and methods. A VBA code for an object is orderly composed of the object name, a period, followed by the property or method. The property is an attribute of an object and is always accompanied by an equal sign (=), while the method is the action carried out with the object.

In the code created above, Select and ClearContents are called **method**, whereas Selection in the second row is an **object** refers to Range object in the first row. Value on the third row is a **property** of the Range object with the value of 10. The code above is intended to remove the existing value in the range A1 on Sheet1 and replace it with a new value of 10. The Worksheets ("Sheet1") code refers to active sheet name, in this case is Sheet1.

To see the available objects in Excel, you can access through the **Object Browser** by clicking the icon  on VBE toolbar or press **F2**. Select the Excel libraries in the pull down menu as shown below:



Objects are classified into **Classes** (on Class window) and their **Members** (on Members window) that could be a **property**, a **function** or a **constant**. Names, Cells, FormulaArray for instance, are members of the Range class. Excel's objects are arranged in a hierarchy; objects contain other objects from top level to down level. Code below shows an example of an object hierarchy from the Application object to the Range object:

```
Application.Workbooks("Book1.xls").Worksheets("Sheet1").Range("A1")
```

In the code, Range is the property of the Worksheet object that returns a Range object. It uses **Range(arg)** syntax where *arg* is name of the range (A1) to return a Range object represents cell A1. Moreover, Worksheet is the property of the Workbook object that returns a Worksheet object, Workbook is the property of the Application object that returns a Workbook object, and the top level of this hierarchy is the Application object that is Excel.

1.11.2 RECORDING MACRO

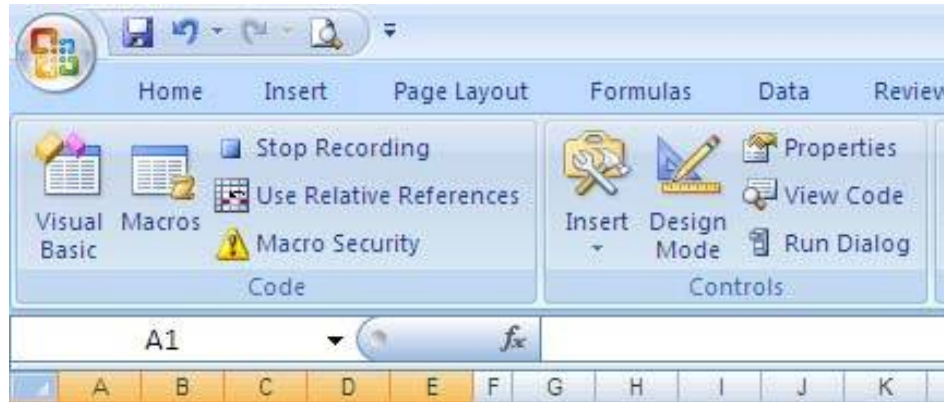
The best way to figure out how VBA communicates with Excel's objects is to record a macro. This tool gives a quick introduction of all Excel's object as well as how to write code in VBA. For instance, we would like to record a macro to delete data on a range of cells, say, in a range of A1 to E10. We need to create an automation steps to clean the existing data as the program produces a new one. The steps for recording macro are as follows:

1. Click the **Developer** tab > **Record Macro** to display the **Record Macro** dialog box with the default name of Macro1 for macro name as shown below:



2. Change macro name to *ClearOutput*

3. Move the pointer to cell A1 and drag the mouse to the cell E10 to assign the range data that will be removed.
4. Press Delete to delete the contents of range A1 to E10 and then place the pointer in cell A1.
5. End the process by clicking **Stop Recording**.



The recorded macro is stored in the workbook, and to see the code you have to open the **Visual Basic Editor** window through **Developer** tab > **Visual Basic**. The code is created as the following:.

```
Sub ClearOutput ()
'
'Macro ClearOutput
'
    Range("A1:E10").Select
    Selection.ClearContents
End Sub
```

In the recorded macro above there are VBA statements such as Range (), Select, Selection and ClearContents. You may be a bit familiar with these statements that have been briefly explained in the previous section.

The recording macro tool is very useful for writing complex code or you may indeed never imagined before, like how to manipulate a chart object such as to copy, to create a title, to change the colors, making marker, automation for creating lines, and so on. This can all be done without the need to understand more about the Excel's object hierarchy that such "long" and may be a bit confused.

1.11.3 PROCEDURE

VBA Macro is a programming language written in a procedure, for instance, *ClearOutput* macro that is written in a *Sub* procedure. Procedure in VBA is divided into three types as follows:

- *Sub* procedure, procedure that begins with *Sub*, its name and ended with *End Sub*
- *Function* procedure, begins with *Function*, its name and ended with *End Function*
- *Event* procedure, works when there is a certain event, for example, open a workbook, click the button, activate the worksheet, and so on.

The syntaxes of above types of procedure are as follows:

1. Sub Procedure

Syntax:

[Private | Public | Friend] [Static] Sub *name* [(*arguments*)]

[*statements*]

[Exit Sub]

[*statements*]

End Sub

Remarks

Public Optional. Indicates that the Sub procedure is accessible to all other procedures in all modules. If not defined, the procedure by default is Public

Private Optional. Indicates that the Sub procedure is accessible only to other procedures in the module where it is declared

Friend Optional. Used only in a class or object module. Can be accessed from other procedure in all modules in a workbook

Static Optional. Indicates that the Sub procedure's local variables are preserved between calls.

name Required. Name of the Sub; follows standard variable naming conventions.

<i>arglist</i>	Optional. List of variables representing arguments that are passed to the Sub procedure when it is called. Multiple variables are separated by commas.
<i>statements</i>	Optional. Any group of statements to be executed within the Sub procedure.

2. Function Procedure

Syntax:

[Private | Public | [Static] Function *name* [(arguments)][As type]

[*statements*]

End Function

3. Event Procedure

Syntax:

Sub *expression.eventname* (*parameters*)

Remarks:

expression variabel that refers to object

eventname name of event

parameters argumens used in *Event* procedure

The most frequently used of Event procedure is clicking a button (command button) on the worksheet. Here is the example code:

```
Private Sub CommandButton1_Click()  
    'call other procedure  
    Call Pro_Sub1  
End Sub
```

The steps and examples of the above procedures will be shown in next chapters.

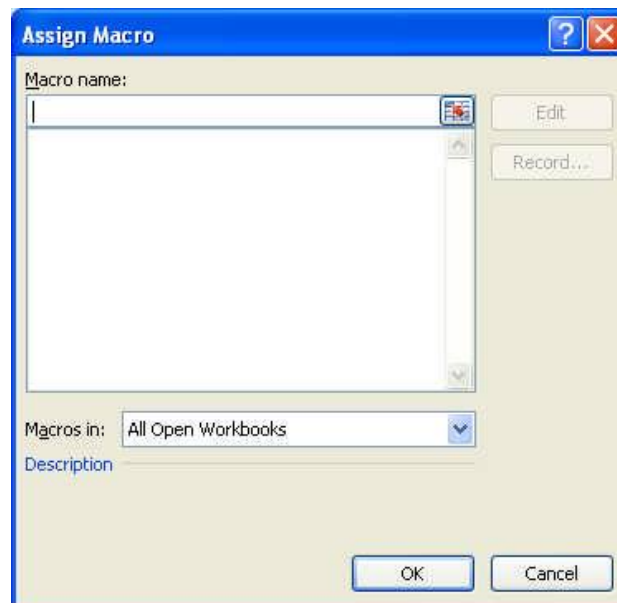
1.11.4 RUNNING MACRO

Macros can be executed in the two following ways:

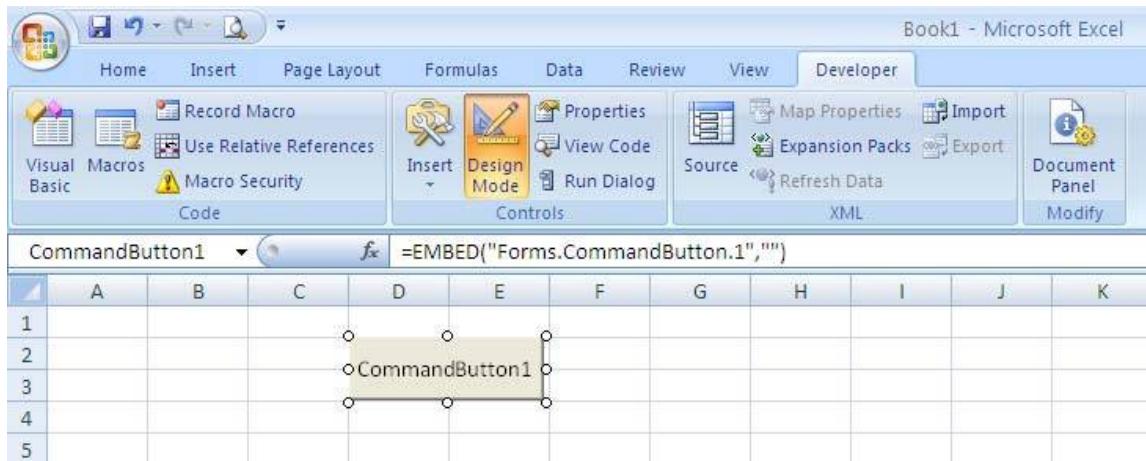
1. Through the **Developer** tab > **Macros** > select the macro name > **Run**. Function procedure, Private Sub procedure and Sub procedure written with argument, can not be run by this way.
2. Clicking **Button** (Form Control) or the **Command Button** (Active X control) on the worksheet.

The second way is commonly used because it is done by only one step. However, it can be done in two ways, which are as follows:

- Create a button through the **Developer** tab > **Insert** > **Form Controls** > **Button** > there is a sign "+" on worksheet, and place it on worksheet by dragging the mouse, the same way to determine its size as well. The **Assign Macro** dialog box will be displayed as below. Enter the name of a macro that to be executed and then click **OK**.



- Macro is executed by clicking a command button on the worksheet which is associated with an event procedure. To create command button click the **Developer** tab> **Insert** > **Active X Controls** > **Command Button** > there is sign "+" at the pointer, and place it on the worksheet by dragging the mouse, the same way to determine its size as well. The results is as shown below:



Double-click the command button to write code in the following procedure:

```
Private Sub CommandButton1_Click()  
  
End Sub
```

When a command button is created, the code program is inactive or when the Design Mode is enabled. Before run macro, click the Design Mode to disable (until it looks unselected).

1.11.5 VBA DICTIONARY

Statements or words that are used in Excel-VBA are too many because the capability VBA to support Excel for diverse goals. Some statements are similar to those used by Excel, for examples, operator and mathematical relationships, some strings functions and math functions. In this book, all statements related to this book will not be placed in a special section, but it will be discussed respectively, on each topic in its section.

CHAPTER 2

EXCEL FUNCTIONS

Almost identical to a function definition in mathematics, a function in Excel works based on the given arguments and written as follows:

$$\begin{array}{ccc} \text{Result} & \text{Name} & \text{Argument/s} \\ \swarrow & \swarrow & \swarrow \\ \mathbf{y} = \mathbf{f} & (\mathbf{a,b,c,...}) \end{array}$$

Arguments are written after the function name, between parentheses, which are the values used to perform the operation. The argument can be a numerical value, text, references or range of cell, name, label or a function.

Excel functions are listed by category such as, math and trigonometry, statistics, finance, logic, and so on. In addition, Excel also provides macrosheet and Visual Basic for Applications (VBA) to create a function that is defined by the user (user-defined function).

2.1 MATH AND TRIGONOMETRY FUNCTIONS

ABS

Returns the absolute value of a number.

Syntax:

ABS(number)

Example:

ABS(4) = 4

ABS(-4) = 4

SQRT(ABS(-81)) = 9

INT

Rounds a number down to the nearest integer

Syntax:

`INT(number)`

Example:

`INT(8.9) = 8`

`INT(-8.9) = -9`

TRUNC

Makes a number to an integer by removing the fractional part of the number. To specify the precision of the truncation, enter a specified number of digits after the number.

Syntax:

`TRUNC (number,number_digits)`

Example:

`TRUNC(8.9) = 8`

`TRUNC(-8.9) = -8`

`TRUNC(PI ()) = 3`

`TRUNC(PI (), 3) = 3,141`

ROUND

Rounds a number to a specified number of digits.

Syntax:

`ROUND(number,num_digits)`

Example:

`ROUND(3.25,1) = 3.3`

$\text{ROUND}(3.247,1) = 3.2$

$\text{ROUND}(3.247,0) = 3$

$\text{ROUND}(32.47,-1) = 30$

ROUNDDOWN

Rounds a number down to a specified number of digits.

Syntax:

$\text{ROUNDDOWN}(\text{number}, \text{number_digits})$

Example:

$\text{ROUNDDOWN}(3.3,0) = 3$

$\text{ROUNDDOWN}(66.8,0) = 66$

$\text{ROUNDDOWN}(3.14159,3) = 3,141$

$\text{ROUNDDOWN}(-5.24621,2) = -5.24$

$\text{ROUNDDOWN}(1550.24621,-2) = 1500$

ROUNDUP

Rounds a number up to the desired number of digits.

Syntax:

$\text{ROUNDUP}(\text{number}, \text{num_digits})$

Example:

$\text{ROUNDUP}(3.3, 0) = 4$

$\text{ROUNDUP}(66.8, 0) = 67$

$\text{ROUNDUP}(3.14159, 3) = 3,142$

$\text{ROUNDUP}(-5.24621, 2) = -5.25$

$\text{ROUNDUP}(1550.24621,-2) = 1600$

SUMPRODUCT

The number resulted by multiplying corresponding components of arrays

Syntax:

SUMPRODUCT(array1,array2,...)

Example

SUMPRODUCT(1,2,3) = 1 x 2 x 3 = 6

SUMPRODUCT({1,2,3},{4,5,6}) = 1 x 4 + 2 x 5 + 3 x 6 = 36

SUMPRODUCT({1,2,3,"hello"},{4,5,6,5}) = 36

2.2 LOGICAL FUNCTIONS

IF

This function has two values of results, which are TRUE and FALSE, with logical test, if met will do the TRUE value. If the result returns a text, then the text must be between quotes ("text").

Syntax:

IF(logical_test,value_ if true,value_if_ false)

Example 1:

In a class, students which have exam scores below or equal to (\leq) 55 will FAIL the exam, and above the value will PASS the exam.

	A	B	C	D	E	F
1						
2	Name	Score	Remark			
3	V. Magda	51	FAIL	=IF(B3<=55,"FAIL","PASS")		
4	Y. Artist	93	PASS			
5	L. Money	90	PASS			
6	M. House	70	PASS			
7	H. Aunt	37	FAIL			
8						
9						

Result of copying cell from C3 to C4:C7

Example 2: Nested IFs with two logical tests

Students who FAIL with exam score > 45 have chance to take a supplementary exam to increase their scores. Use the function to find out the name of student:

	A	B	C	D	E	F	G	H
1								
2	Name	Score	Remark					
3	V. Magda	51	RE-EXAM	=IF(B3<=45,"FAIL",IF(B3<=55,"RE-EXAM","PASS"))				
4	Y. Artist	93	PASS	} Result of copying cell from C3 to C4:C7				
5	L. Money	90	PASS					
6	M. House	70	PASS					
7	H. Aunt	37	FAIL					
8								
9								

AND

Returns TRUE if all of its arguments are TRUE, and returns FALSE if one its argument is FALSE. Generally nested with IF function.

Syntax:

AND(logical_1, logical_2, ...)

Example 1:

AND(2*2=4,2+2=4) Returns TRUE

AND(2<100,4<100,102<100) Returns FALSE

Example 2:

Now, with the same class and the same students in Example 1, but with different exam subjects: a course score can help the student that receives exam score > 40 and <= 55. The given formula is:

$(2 \times \text{Exam score} + 1 \times \text{Course score}) / 3$.

	A	B	C	D	E	F	G	H	I
1									
2	Name	Exam	Course	Final Score					
3	V. Magda	60	56	60	=IF(AND(B3>40,B3<=55),ROUND((2*B3+C3)/3,0),B3)				
4	Y. Artist	55	72	61	} Result of copying cell from C3 to C4:C7				
5	L. Money	60	65	60					
6	M. House	40	60	40					
7	H. Aunt	46	64	52					
8									
9									

COUNTIF

Counting the number of cells in a range of cells according to given criteria.

Syntax:

COUNTIF(range,criteria)

Example 1

Looking for the number of students with specified value for Example 2 above:

COUNTIF(D2: D6,60) = 2

COUNTIF(D2: D6,> 60) = 1

OR

The result is TRUE if one of its arguments is TRUE. Generally nested the IF function.

Syntax:

OR(logical1,logical2, ...)

Example:

OR(2*2 = 4,2+2=4) Returns TRUE

OR(2<100,4<100,102<100) Returns TRUE

2.3 LOOKUP FUNCTIONS

VLOOKUP

Search for a value based on the column index in the table data arranged vertically

Syntax:

VLOOKUP(value,table,col_index,range_lookup)

Example

The experimental result of laboratory test is affected by the size and weight of the used tools. This example shows how to use VLOOKUP function to read the size and weight of the ring based on the **Ring Calibration** table. Suppose it is intended to search the diameter and

weight of ring no. 3 and 7. The table data refers to range A4 to F11, ignoring the column heading.

	A	B	C	D	E	F
1	RING CALIBRATION					
2						
3	Ring No.	Height (cm)	Diameter (cm)	Weight (gram)	Area (cm ²)	Volume (cm ³)
4	1	1.950	5.000	32.850	19.635	38.288
5	2	2.000	5.000	32.000	19.635	39.270
6	3	2.000	5.000	33.440	19.635	39.270
7	4	2.000	5.000	34.980	19.635	39.270
8	5	1.950	5.000	33.700	19.635	38.288
9	6	1.925	4.985	32.000	19.517	37.571
10	7	2.000	5.000	33.000	19.635	39.270
11	8	1.975	5.125	34.000	20.629	39.711

VLOOKUP result:

	A	B	C	D	E	F
14	Ring No.	Diameter (cm)	Weight (gram)			
15	3	5.000	33.440			
16	7	5.000	33.000			

Formula in cell B15 to C16:

	A	B	C
14	Ring No.	Diameter (cm)	Weight (gram)
15	3	=VLOOKUP(A15,\$A\$4:\$F\$11,3)	=VLOOKUP(A15,\$A\$4:\$F\$11,4)
16	7	=VLOOKUP(A16,\$A\$4:\$F\$11,3)	=VLOOKUP(A16,\$A\$4:\$F\$11,4)

Range_lookup is a logical value that specifies whether VLOOKUP finds an exact match or an approximate match. If TRUE (or omitted), the values in the first column of table_array must be in ascending sort order (-2, -1,0,1,2, ..) because VLOOKUP may return incorrect. If FALSE, the table_array values do not need to be sorted. For TRUE condition, if the lookup_value does not match with table_array, VLOOKUP returns the next largest value that is less than lookup_value. Otherwise, if FALSE, VLOOKUP will find an exact match, if it is not found, VLOOKUP returns #N/A (error).

HLOOKUP

Search for a value based on the row index in the table data arranged horizontally. The application of HLOOKUP are almost identical to VLOOKUP, in exception that row index is needed to lookup the value.

Syntax:

HLOOKUP(value,table,row_index,range_lookup)

Example

Transpose (**Copy > Paste Special > Transpose**) the Ring Calibration table to arrange table data as shown below. After that, use HLOOKUP the same way as VLOOKUP example above to search the diameter and weight of ring no. 3 and 7.

Ring No.	1	2	3	4	5	6	7	8
Height (cm)	1.95	2.00	2.00	2.00	1.95	1.93	2.00	1.98
Diameter (cm)	5.00	5.00	5.00	5.00	5.00	4.99	5.00	5.13
Weight (gram)	32.85	32.00	33.44	34.98	33.70	32.00	33.00	34.00
Area (cm ²)	19.64	19.64	19.64	19.64	19.64	19.52	19.64	20.63
Volume (cm ³)	38.29	39.27	39.27	39.27	38.29	37.57	39.27	39.71

2.4 TEXT FUNCTIONS

LEFT, RIGHT, MID

Returns characters in a text string based on the specified number of characters

=LEFT(text,num_chars)

Returns the first character

=**RIGHT**(text,num_chars)

Returns the last character

=**MID**(text,start_num,num_chars)

Returns characters starting at specified position

Example

	A	B	
1	ENGINEERING	ENG	=LEFT(A1,3)
2	745.56	RING	=RIGHT(A1,4)
3		ENGINEER	=MID(A1,1,8)
4		745.5	=LEFT(A2,5)

CONCATENATE

Combines two or more strings from different cells into one string

Example

	A	B	C	D
1				
2	Civil	Engineer		Civil Engineer
3				
4	Point			Coordinate
5	x	y	z	
6	2.0	4.0	0.0	2,4,0
7	2.0	4.0	-1.5	2,4,-1.5
8	2.0	4.0	-1.5	(2,4,-1.5)

	D	
1		
2	Civil Engineer	=CONCATENATE(A2," ",B2)
3		
4	Coordinate	
5		
6	2,4,0	=CONCATENATE(A6,"",B6,"",C6)
7	2,4,-1.5	=CONCATENATE(A7,"",B7,"",C7)
8	(2,4,-1.5)	=CONCATENATE("("&A8,"",B8,"",C8,"")")

Another way is to use the way of writing as the following:

numeric & "text" & numeric

Example

	A	B	C	D
1				
2	Civil	Engineer		Civil Engineer
3				
4	Point			Coordinate
5	x	y	z	
6	2.0	4.0	0.0	2,4,0
7	2.0	4.0	-1.5	2,4,-1.5
8	2.0	4.0	-1.5	(2,4,-1.5)

	D	
1		
2	Civil Engineer	=A2&" "&B2
3		

4	Coordinate	
5		
6	2,4,0	=A6&","&B6&","&C6
7	2,4,-1.5	=A7&" "&B7&","&C7
8	(2,4,-1.5)	=(" "&A8&","&B8&","&C8&")"

2.5 DATA ANALYSIS FUNCTIONS

Excel has many built-in functions that are used to analyze data obtained from experimental result. The objective of the analysis is to obtain theoretical parameters that give the best relationship between theory and experimental result. Three methods will be discussed here which are: linear regression, polynomial regression and interpolation.

2.5.1 LINEAR REGRESSION

Linear regression is to determine a straight line that fits or the most closely fits to a number of points data, providing a linear relationship between two variables. The method used to obtain the line is called the least squares method. Thus, linear regression consists of a series of points that fit to a number (n) of points data (x_i, y_i) written into a straight line equation:

$$y = Ax + B \quad (2.1)$$

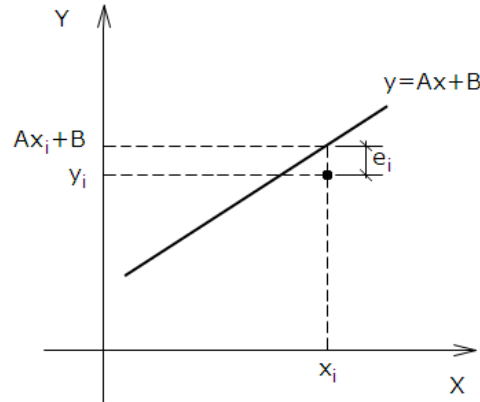
where:

A = slope of the line,

B = the intersection of the straight line to the Y-axis

The accuracy of the straight line over a number of points data is evaluated by a total deviation E , which is the sum of squares of the distance e between the points data and the fitted points:

$$E = \sum_{i=1}^n e_i^2 = \sum_{i=1}^n (Ax_i + B - y_i)^2 \quad (2.2)$$



By making E to minimum, A and B values therefore can be determined to obtain the equation 2.1. Beside E , R^2 value is also used for the accuracy measurement by a relationship below:

$$R^2 = 1 - \frac{\sum_{i=1}^n [y_i - f(x_i)]^2}{\sum_{i=1}^n y_i^2 - \frac{1}{n} \left(\sum_{i=1}^n y_i \right)^2} \quad (2.3)$$

R^2 value varies from 0 to 1. $R^2 = 1$ is when the regression line coincides with the data. In polynomial regression, the higher-order of polynomial, the closer R^2 value to = 1.

Regression in Excel can be obtained using the **TREND** function, **SLOPE**, **INTERCEPT** and **LINEST** or with **Trendline**, a regression line from data relations in an XY coordinate system. Trendline is created by the following steps: right-clicking the mouse when the pointer is on one of the points data in the graph to display a box menu > click **Add Trendline** to display the **Format Trendline** dialog box > select **Linear** for linear regression.

Example 1: Linear Regression

The soil shear strength determination from a laboratory test obtains the result as shown in Figure 2.1. In general, the depiction of data generated from the test is rarely a straight line to show failure envelope, therefore, it needs a fitted straight line to represent the data.

Regression is used to obtain shear strength parameters, which are cohesion (c) and shear angle in (ϕ) of the soil mass.

In direct shear test, the failure envelope is a straight line that is expressed into equation:

$$\tau = c + \sigma_n \tan \phi$$

Where,

- τ shear strength
- c cohesion intercept
- σ_n normal stress
- ϕ shear angle or slope of failure envelope

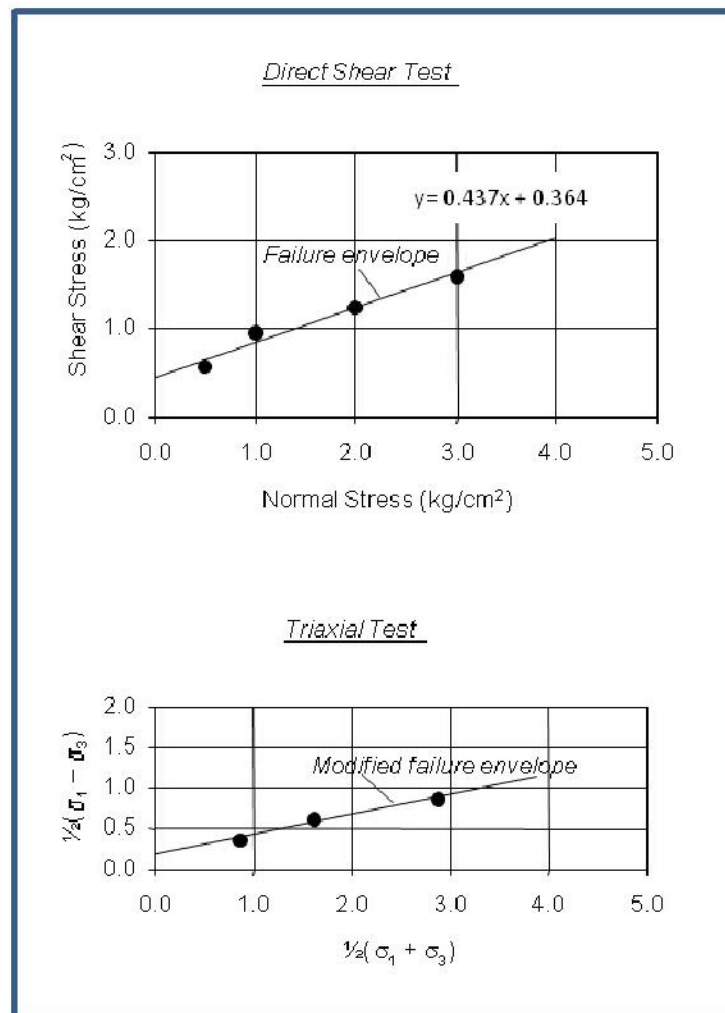
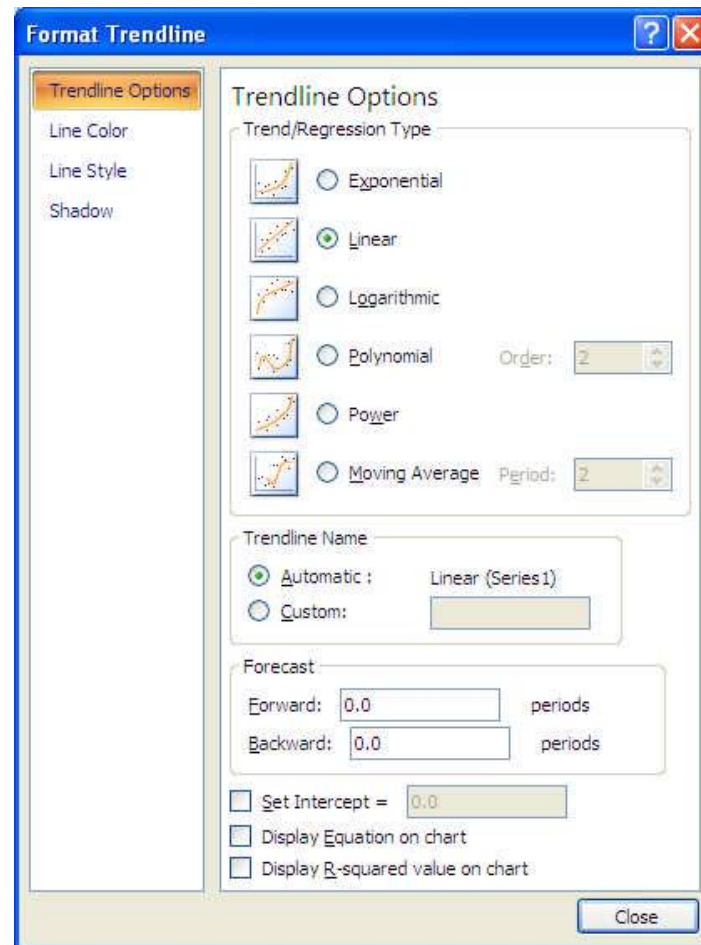


Figure 2.1: Regression on Excel Chart

Shear strength in this model is a linear function over the normal stress, where c and $\tan \phi$ respectively expresses Y-axis intercept and the slope of the line.

In Chart, a regression can be done with Trendline by the following steps: right clicking mouse when the pointer is on one of the points data on Chart to display a box menu. From the box menu select **Add Trendline** to display **Format Trendline** dialog box > **Linear** > **Backward** to intercept Y-axis > click **Display Equation on Chart**. The result is as shown in Figure 2.1.



The values of A and B that are shown in line equation, $y = Ax + B$ in direct shear test are equal to those calculated with the following functions:

$A = \text{slope of the line} = \tan \phi = \text{SLOPE}(Y,X)$

$B = Y\text{-intercept} = \text{INTERCEPT}(Y,X)$

Shear angle ϕ therefore can be calculated with ATAN function (in degree):

$$= \text{ATAN}(\text{SLOPE}(Y,X)) * 180 / \text{PI}()$$

LINEST and TREND function used and written as follows:

=INDEX(LINEST(Y,X),1) to obtain Slope,

=INDEX(LINEST(Y,X),2) to obtain Y-intercept

In practices, TREND function is entered in a series of worksheet formulas to obtain fitted points represents a series of data (array Y, array X). The formula can be written as follows:

	A	B	C	D	E
1	Direct Shear Test				
2		Normal	Shear	Fitted	
3		Stress	Stress	Shear	
4		0.50	0.50	0.583	=TREND(\$C\$4:\$C\$7,\$B\$4:\$B\$7,B4)
5		0.90	1.00	0.802	Result of copying cell from D4 to D5:D8
6		1.25	2.00	1.239	
7		1.65	3.00	1.676	
8	Y-Intercept =		0.00	0.364	
9					

Shear strength can also be expressed in major σ_1 and minor σ_3 principal stresses in the triaxial test by plotting $\frac{1}{2}(\sigma_1 - \sigma_3)$ against $\frac{1}{2}(\sigma_1 + \sigma_3)$. The stress condition that fit to a number of points data is referred to as modified failure envelope, which expressed into equation:

$$\frac{1}{2}(\sigma_1 - \sigma_3) = a + \frac{1}{2}(\sigma_1 + \sigma_3) \tan \alpha$$

where a and α are the modified shear strength parameters. The parameters c and ϕ are then given through relations below:

$$\phi = \sin^{-1}(\tan \alpha)$$

$$c = \frac{a}{\cos \phi}$$

Example 2: Logarithmic Regression

It is often found that the data relationship from a test is presented where the X-axis is made in log scale as shown figure 2.2. The equation 2.1 now becomes:

$$y = A \ln(x) + B \quad (2.4)$$

where A and B are constants

The example data here is adopted from laboratory liquid limit (LL) test, i.e. to determine soil moisture content at 25 blows. Below are the related informations:

- Moisture content (W_n) = $[(\text{weight of wet soil} - \text{weight of dry soil}) / (\text{weight of dry soil})] \times 100\%$.
- Water content' (W_n') = fitted W_n in the semi-log relationship.
- N = blows number

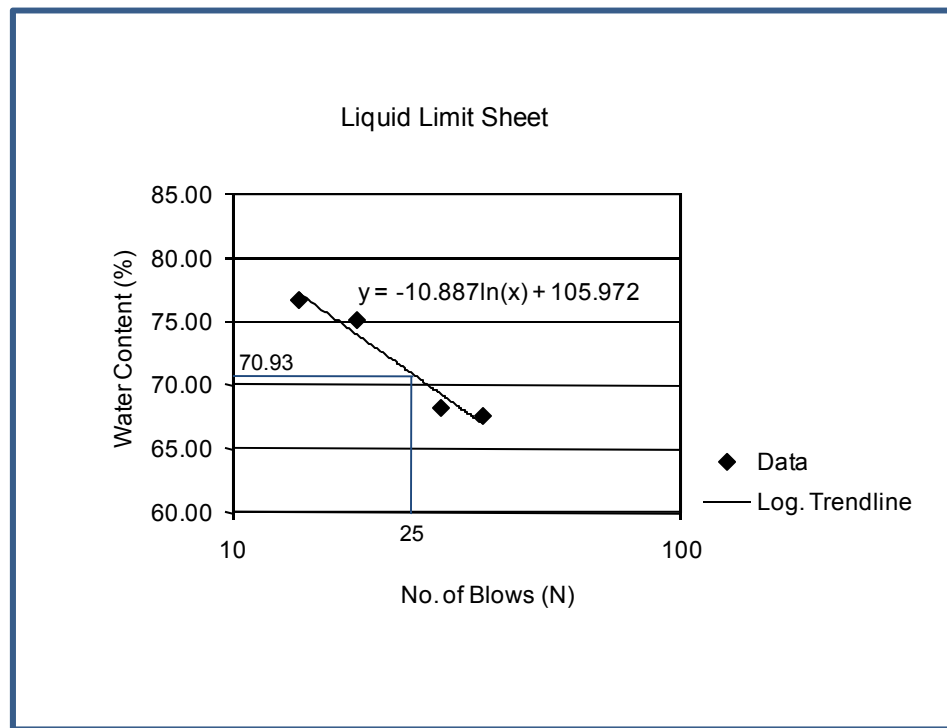


Figure 2.2: Regression on Semi-log Liquid Limit Test

Figure 2.2 shows the LL test result and a regression line created through **Add Trendline** command. The fitted W_n value (W_n') that represents a number laboratory W_n for each N value can be obtained through the equation below:

$$y = -10.887\ln(x) + 105.972$$

Using the above equation, at $N = 25$ blows, the related fitted $W_n = 70.93\%$.

The calculation sheet of LL test in the worksheet can be made as follows:

	A	B	C	D	E	F	G
1	LIQUID LIMIT TEST						
2							
3	No. Set	Weight of	Can +	Can +	Blows N	Water	Fitted
4		Can (gr)	Soil (gr)	Dry Soil (gr)		Content (%)	Wn (%)
5		Wc	Ws	Wd		Wn	Wn'
6	1	8.3	52.46	33.29	14	76.71	77.24
7	2	2.8	55.61	32.96	19	75.10	73.92
8	3	2.8	57.14	35.12	29	68.13	69.31
9	4	7.8	52.72	34.62	36	67.49	66.96

Liquid Limit (%) = 70.93

3	Water	Fitted
4	Content (%)	Wn (%)
5	Wn	Wn'
6	=100*(C6-D6)/(D6-B6)	=TREND(Wn,LOG10(N),LOG10(E6))
7	=100*(C7-D7)/(D7-B7)	=TREND(Wn,LOG10(N),LOG10(E7))

Liquid Limit (%) =TREND(Wn,LOG10(N),LOG10(25))

The relationship of Wn 'and N in sheet above is a regression line as pictured in Figure 2.2.

Example 3: Log-log Regression

In the third example, we will examine on the data that is formed in log-log scale in the XY coordinate system. The given data is the standard sieve size (US) for the particle size distribution of the granular material as shown in Figure 2.3.

	A	B	C	D
1	Sieve	Sieve	Fitted	Formula :
2	No.	Size	Sieve Size	
3	(US)	(mm)	(mm)	
4	N	S		
5	4	4.750	5.142	=10^TREND(LOG10(S),LOG10(N),LOG10(A5))
6	5	4.000	4.031	=10^TREND(LOG10(S),LOG10(N),LOG10(A6))
7	6	3.350	3.305	=10^TREND(LOG10(S),LOG10(N),LOG10(A7))
8	7	2.800	2.794	Copy the formula down...
9	8	2.360	2.415	
10	10	2.000	1.894	
11	12	1.700	1.552	
12	14	1.400	1.312	
13	16	1.180	1.134	
14	18	1.000	0.998	
15	20	0.850	0.889	
16	25	0.710	0.697	
17	30	0.600	0.572	
18	35	0.500	0.483	
19	40	0.425	0.418	
20	45	0.355	0.367	
21	50	0.300	0.328	
22	60	0.250	0.268	
23	70	0.212	0.227	
24	80	0.180	0.196	
25	100	0.150	0.154	
26	120	0.125	0.126	
27	140	0.106	0.107	
28	170	0.090	0.086	
29	200	0.075	0.072	
30	230	0.063	0.062	
31	270	0.053	0.052	
32	325	0.045	0.043	
33	400	0.038	0.034	
34	500	0.025	0.027	

Figure 2.3: Formula for Log-log data relationship

Regression in log-log data relationship can be obtained using formulas as shown in Figure 2.3, or by adding Trendline by the following steps: click the **Add Trendline > Power > Display equation on chart and the R-squared value on the graph**, the results will be shown as in Figure 2.4. The general equation of Power Trendline is:

$$y = 10^c X^a, \text{ c and a = constant} \quad (2.5)$$

Slope and Y-intercept are respectively the constants a and c in equation 2.5, where the formulas for this series are, respectively:

$$=\text{SLOPE}(\text{LOG (S)}, \text{LOG (N)}) = -1.09022$$

$$=\text{INTERCEPT}(\text{LOG (S)}, \text{LOG (N)}) = 1.36749$$

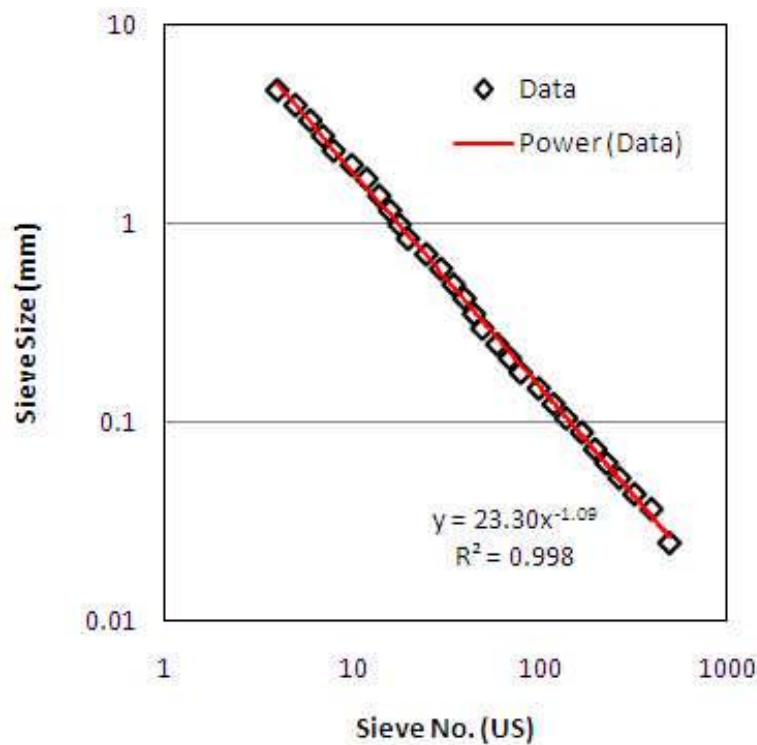


Figure 2.4: Chart from Figure 2.2 data series where TREND is used to obtained the fitted points

2.5.2 POLYNOMIAL REGRESSION

Polynomial regression is used to obtain fitted points of a number of points data which having non-linear (curve) trend in XY coordinate system. Thus, polynomial regression consists of fitted points of a number (n) of points data (x_i, y_i) to be a polynomial equation form:

$$f(x) = a_m x^m + a_{m-1} x^{m-1} + \dots + a_1 x + a_0 \quad (2.6)$$

In Equation 2.6, m = a positive integer and the equation is called a polynomial function of order m. Linear regression is a special form of polynomial regression with m = 1, A = ai and B = ao.

	A	B
	Temp	Unit Mass
1	(°C)	(gr/cm ³)
2	X	Y
3	4	1.00000
4	16	0.99897
5	17	0.99880
6	18	0.99862
7	19	0.99844
8	20	0.99823
9	21	0.99802
10	22	0.99780
11	23	0.99757
12	24	0.99733
13	25	0.99708
14	26	0.99682
15	27	0.99655
16	28	0.99627
17	29	0.99598
18	30	0.99568

Figure 2.5: Example of two data series for polynomial regression

Example

Figure 2.5 shows two data series from the relationship between water density and temperature, where X-coordinate is the temperature (°C) and the Y-coordinate is the water

density (g/cm³). By plotting Y-values against X-values, it is apparently visible that the result has a curve trend.

Figure 2.6 shows chart of the plotted data from Figure 2.5 and regression lines created using **Trendline** that consist of Linear and Polynomial of order 2 and 3. It is apparently seen that the polynomial regression is much better than linear regression for this data series. The fitted points produced from order 2 and 3 regression seemed like really coincides with the data and are only distinguished by the value of R², where order 3 (red curve) the value of R² = 1. Excel 2007 provides the type of polynomial regression up to order 6.

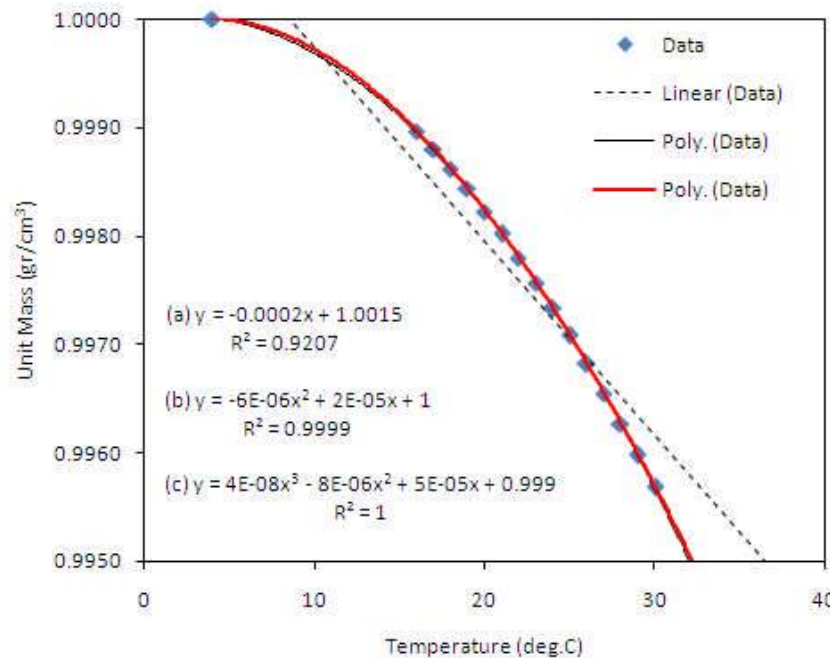


Figure 2.6: Regression analysis using Trendline for data series in Figure 2.5

2.5.3 INTERPOLATION

In an engineering calculation, sometimes it only needs a data interpolation without resolve it into regression analysis. Here is to find y-value that correspond to known x-value a line segment, based on a number data (x_i, y_i) tabulation, where x_i may continuously be increased or decreased. Linear interpolation of y-value corresponds to x-value is:

$$y = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i) \quad (2.7)$$

where $(x - x_i)(x - x_{i+1}) \leq 0$ and $x_i \neq x_{i+1}$

If y and x-axis has linear-log scale, respectively (semi-log), equation 2.7 becomes:

$$\ln(y) = \ln(y_i) + \frac{\ln(y_{i+1}) - \ln(y_i)}{x_{i+1} - x_i} (x - x_i) \quad (2.8)$$

where $(x - x_i)(x - x_{i+1}) \leq 0$ and $x_i \neq x_{i+1}$

Equation 2.8 can also be written:

$$\ln \frac{y}{y_i} = \frac{x - x_i}{x_{i+1} - x_i} \ln \frac{y_{i+1}}{y_i} \quad (2.9)$$

Or,

$$y = y_i \left(\frac{y_{i+1}}{y_i} \right)^{(x - x_i) / (x_{i+1} - x_i)} \quad (2.10)$$

In Excel, the interpolation value can be obtained by **TREND** function. Data series for this function is a line segment formed by x_i, y_i and x_{i+1}, y_{i+1} .

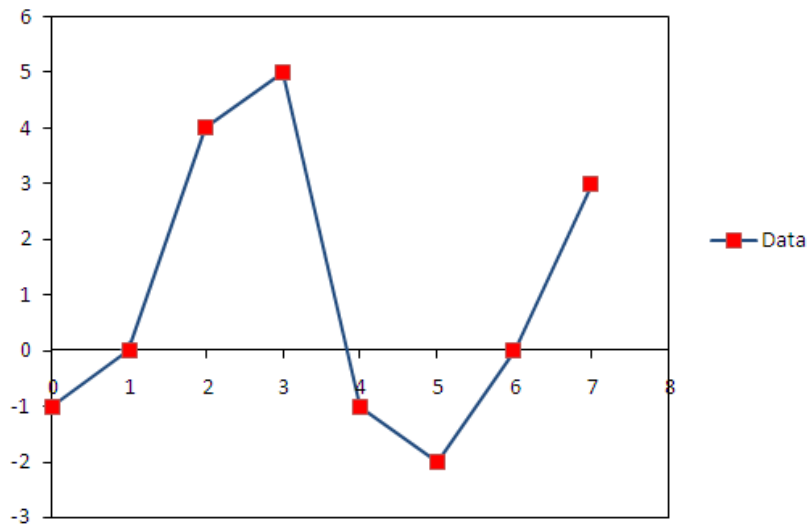
Example 1: Linear Interpolation

Given below is two data series of X and Y-array for interpolation:

Table 2.1: Two Data Series for Linear Interpolation

Data X	Data Y
0	-1
1	0
2	4
3	5
4	-1
5	-2
6	0
7	3

Plotted in chart:



Worksheets below show formulas to obtain the interpolation of the data in Table 2.1, using TREND function and Equation 2.7, respectively.

Using TREND function:

	A	B	C	D	E
1	Linear Interpolation				
2	Data X	Data Y	New X	Y Value =	
3	0	-1	0	-1	=TREND(B3:B4,A3:A4,C3)
4	1	0	1	0	} Result of copying cell from D3 to D4:D9
5	2	4	2	4	
6	3	5	3	5	
7	4	-1	4	-1	
8	5	-2	5	-2	
9	6	0	6	0	
10	7	3	7	-	
11					
12			New Y	X Value =	
13			-1	0	=TREND(A3:A4,B3:B4,C13)
14			0	1	} Result of copying cell from D13 to D14:D19
15			4	2	
16			5	3	
17			-1	4	
18			-2	5	
19			0	6	
20			3	-	
21					

Using Equation 2.7:

	A	B	C	D	E	F
1	Linear Interpolation					
2	Data X	Data Y	New X	Y Value =		
3	0	-1	0	-1	=B3+((B4-B3)*(C3-A3)/(A4-A3))	(Eq. 2.7)
4	1	0	1	0	Result of copying cell from D3 to D4:D10	
5	2	4	2	4		
6	3	5	3	5		
7	4	-1	4	-1		
8	5	-2	5	-2		
9	6	0	6	0		
10	7	3	7	3		
11						
12			New Y	X Value =		
13			-1	0	=A3+((A4-A3)*(C13-B3)/(B4-B3))	
14			0	1	Result of copying cell from D13 to D14:D20	
15			4	2		
16			5	3		
17			-1	4		
18			-2	5		
19			0	6		
20			3	7		
21						

The interpolation values are in column D, which correspond to new values entered in column C. The y-value that correspond to a x-value that lies between 0 - 1 in cell C3 is shown in cell D3; the y-value that correspond to a x-value that lies between 1 - 2 in cell C4 is shown in cell D4, and so forth.

TREND function and Equation 2.7 can also be used to interpolate x-values from known y-values, by replacing the variable x with y and vice versa in Equation 2.7, or by switching x and y-argument in TREND function. The formulas are shown in the worksheet above at row 12, columns C and D.

Example 2: Logarithmic Interpolation

A sands soil sample of 1,000 grams weight is taken and placed in the sieving machine for a sieve analysis test. The results are presented in Table 2.2, showing sieve size used and percent finer. The percent finer (or percent passing) is calculated based on cumulative weight of soil retained on each sieve. Chart of log grain sizes versus percent finer of soil is presented in Figure 2.7.

Table 2.2: Sieve Analysis Results

	A	B
1	Sieve Size	Finer
2	(mm)	%
3	9.50	100.00
4	4.75	97.80
5	2.36	95.48
6	1.18	88.50
7	0.60	70.60
8	0.30	24.10
9	0.15	2.40
10		0.00

The next step is to obtain grain size distribution in percentage of the weight of the sample, for an example to obtain grain size value corresponds to 60 percent finer or D60. The corresponding value can be obtained using Equation 2.10. Formulas for semilog interpolation are shown in worksheet as follows:

	A	B	C	D	E	F	G
1	Grain Size Distribution						
2	x's	y's					
3	Grain Size	Finer					
4	(mm)	(%)					
5	9.50	100.00		100.00	9.50	=A5*(A6/A5)^((D5-B5)/(B6-B5))	(Eq. 2.10)
6	4.75	97.80		97.80	4.75	} Result of copying cell from E5 to E6:E10	
7	2.36	95.48		95.48	2.36		
8	1.18	88.50		88.50	1.18		
9	0.60	70.60		70.60	0.60		
10	0.30	24.10		24.10	0.30		
11	0.15	2.40		2.40	-		
12							
13			D10 =	10	0.191	=A10*(A11/A10)^((D13-B10)/(B11-B10))	
14			D30 =	30	0.328	=A9*(A10/A9)^((D14-B9)/(B10-B9))	
15			D60 =	60	0.512	=A9*(A10/A9)^((D15-B9)/(B10-B9))	
16							

Plotted in chart:

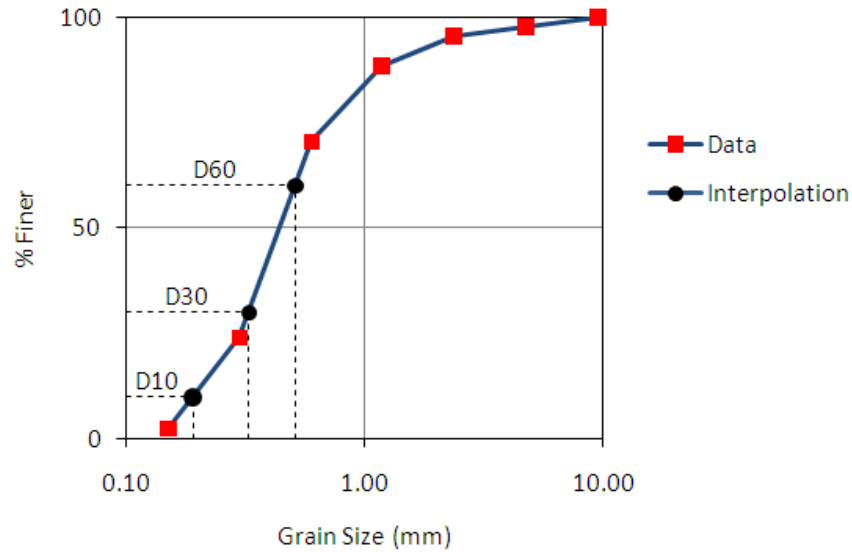


Figure 2.7: Semilog interpolation for the data of Table 2.2

Notes:

- There is however more-flexible way to obtain data interpolation which to resolve them with a macro (custom function), using Equation 2.7 or 2.10. When entered into macro, formulas can be shortened and does not depend on a line segment where a known value is placed. Discussion on the custom function macro will be in the following chapter in the topic of user defined function.
- The interpretation on sieve analysis result includes to determine how well the gradation of sands soil. Gradation of sands or grained particles (silt to gravel) can be measured by the coefficient of uniformity (C_U) and the curvature coefficient (C_C) as defined below.

$$C_U = \frac{D_{60}}{D_{10}}, \quad C_C = \frac{D_{30}^2}{D_{60} D_{10}}$$

The greater the value of C_U , the greater grain distribution range, where well-graded particles usually has C_C value between 1 and 3.

2.5.4 STATISTICAL DATA

Histogram and Cumulative Distribution

In a statistical data presentation, it is convenient to make the data from observation into histogram to see the data distribution. In histogram, the observed data plotted against its frequency distribution and thus, we can get a visual summary and give quick impression on the observed data.

Example

The following is concrete compressive strength data selected randomly from 40 samples obtained from the test of characteristics compressive strength of concrete:

Concrete compressive strength test result (kg/cm ²)			
376.28	399.30	375.41	370.41
394.43	409.47	393.53	371.43
387.19	374.34	386.30	372.44
390.35	392.29	389.46	390.30
401.27	386.78	390.37	384.82
379.95	361.23	379.08	362.28
380.97	402.90	380.10	400.86
389.33	368.22	388.44	366.35
384.64	399.20	383.76	407.46
396.34	365.36	398.83	363.51

To simplify data distribution, the result data divided into several classes within interval (taken) = 5 kg/cm², and then tabulated in Table 2.3. The scores from each class of data is called class frequency, which is represented by height of the bar. To ease data distribution process into classes can be done by using a function macro or custom function. The discussion on custom function will be in the next chapter in the topic of user defined function.

Table 2.3: Data distribution to create a histogram

Class Interval kg/cm ²	Mid Point kg/cm ²	Frequency N	Percent (%)	Cumulative (%)
360-365	362.5	3	7.5	7.5
365-370	367.5	3	7.5	15.0
370-375	372.5	4	10.0	25.0
375-380	377.5	4	10.0	35.0
380-385	382.5	5	12.5	47.5
385-390	387.5	6	15.0	62.5
390-395	392.5	6	15.0	77.5
395-400	397.5	4	10.0	87.5
400-405	402.5	3	7.5	95.0
405-410	407.5	2	5.0	100.0
		40	100.0	

Histogram chart of Figure 2.8 is made based on Table 2.3 data. To create a histogram take the following steps: click **Insert** tab > **Column** > **Column clustered** > **Select Data** > **Add** > **Series Values**: select range "Percentage" > **OK**. Click **Edit for Horizontal Axis Labels** > insert range "Mid Point".

To create cumulative frequency chart as Figure 2.9, select the type of **XY scatter chart with smoothlines and markers** > **Select Data** > **Add** > **Series X values**: select range "Mid Point" > **Y-values**: select range "Cumulative".

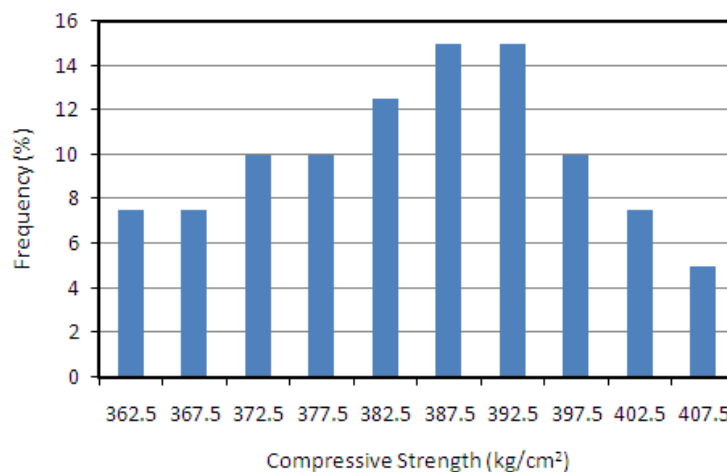


Figure 2.8: Histogram of concrete compressive strength

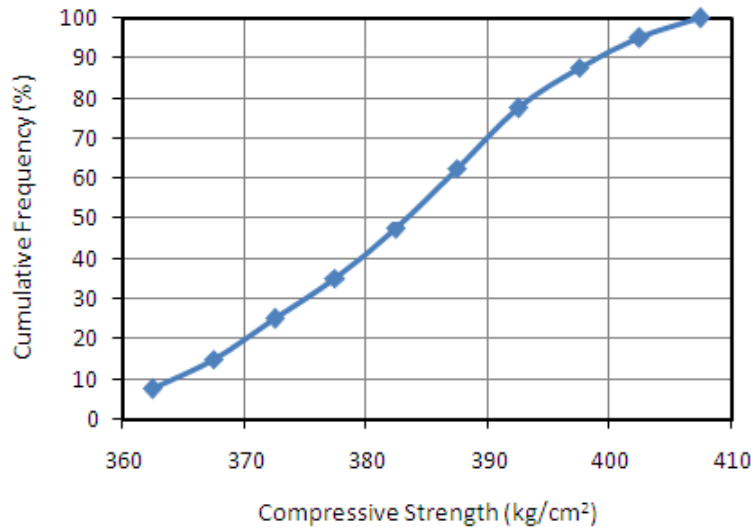


Figure 2.9: Cumulative frequency of concrete compressive strength Range, Mean dan Standard Deviation

The classification or data distribution is generally defined into three statistical parameters, which are, range, mean and standard deviation. Range is the difference between the lowest and the highest values of data. From the concrete compressive strength test result, there is a range of 48.24, from the lowest 361.23 to the highest 409.47 in kg/cm² units. The mean or average is the sum of the data values divided by the total number of data (n). From the test, the mean is obtained = 384.87 kg/cm². If a set of data values are divided into M classes, the mean formula is written as follows:

$$\bar{x} = \frac{\sum_{i=1}^M m_i x_i}{n} \quad (2.11)$$

Where,

m_i = frequency of class interval i

x_i = mid point of class interval i

Mean is a value that commonly used to represent values of a set of data. In consistent with this, several measurements are made to state deviation, and one of them is the formulation of mean square deviation (MSD). The MSD is also called the variance, and the formula is:

$$\frac{\sum_{i=1}^M m_i (x_i - \bar{x})^2}{n} \quad (2.12)$$

Another measurement to be used is the standard deviation (S), which is the square root of MSD:

$$S = \sqrt{\frac{\sum_{i=1}^M m_i (x_i - \bar{x})^2}{n}} \quad (2.13)$$

Through Equation 2:13, the standard deviation from the test result as in Table 2.3 is calculated = 12:49 kg/cm², and with divisor of (n - 1), calculated from the sample, MSD and S is respectively calculated 394.74 and 16.12 kg/cm². The standard deviation is a measure of dispersion of a set of data values from its mean; where a big or small of the value indicates the quality of job execution.

Normal Distribution

From the histogram and measurement of statistical parameters above, it can now be drawn a normal distribution curve. The normal distribution function is,

$$f(x, S, \bar{x}) = \frac{1}{S\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\bar{x}}{S}\right)^2} \quad (2.14)$$

For log-normal distribution, it can be obtained by replacing x with ln(x) in Equation 2.14.

Figure 2.10 and 2:11 show normal distribution curve that fits with the percent frequency data in Table 2.3. Figure 2.12 shows the formulas used in the presentation of the test result.

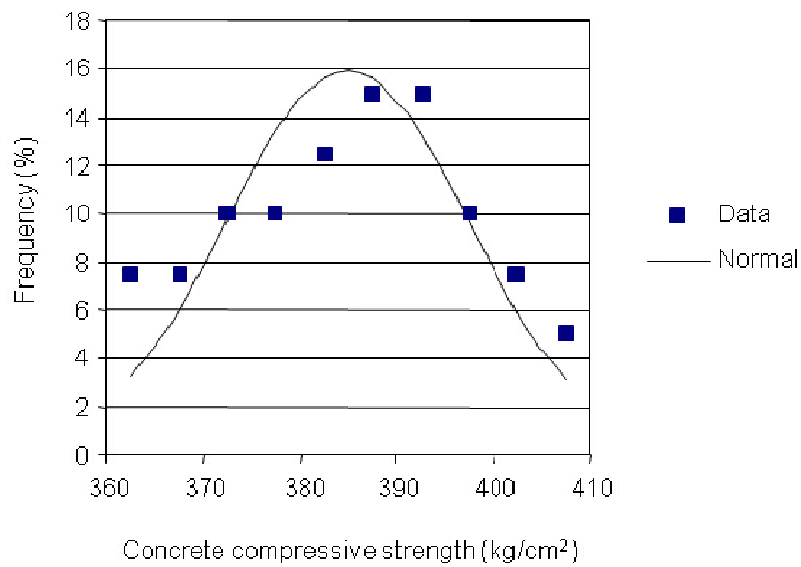


Figure 2.10: Normal distribution curve that fits to the data of Table 2.3

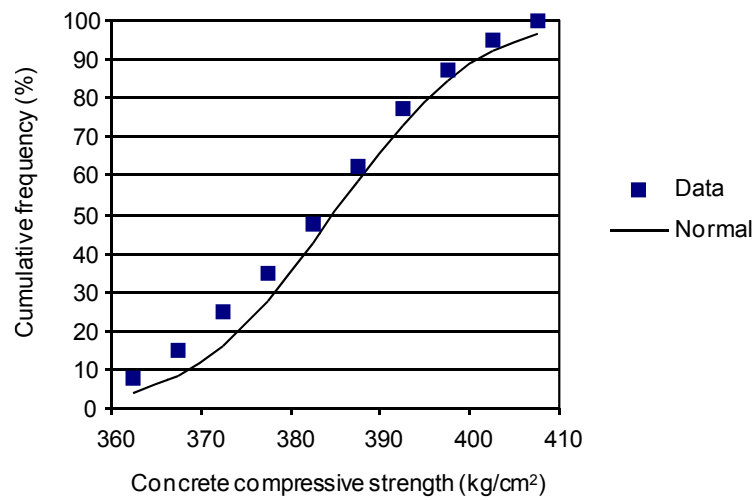


Figure 2.11: Cumulative normal distribution curve that fits to data of Table 2.3

	A	B	C	D	E	F	G
1	Class Interval	Mid Point	Frequency	Percent	Cumulative	Normal	Normal Cum.
2	kg/cm²	Kg/cm²		(%)	(%)		
3		k	N	f			
4	360-365	362.5	3	7.5	7.5	3.2	3.7
5	365-370	367.5	3	7.5	15.0	6.1	8.2
6	370-375	372.5	4	10.0	25.0	9.8	16.1
7	375-380	377.5	4	10.0	35.0	13.4	27.8
8	380-385	382.5	5	12.5	47.5	15.7	42.5
9	385-390	387.5	6	15.0	62.5	15.6	58.3
10	390-395	392.5	6	15.0	77.5	13.2	72.9
11	395-400	397.5	4	10.0	87.5	9.6	84.4
12	400-405	402.5	3	7.5	95.0	5.9	92.1
13	405-410	407.5	2	5.0	100.0	3.1	96.5
14			40	100.0			
15							
16	Mean =				384.87	kg/cm²	
17	S =				12.49	kg/cm²	
	F				G		
1	Normal				Normal		
2					Cumulative		
3							
4	=500/(\$E\$17*SQRT(2*PI()))*EXP(-0.5*((k-\$E\$16)/\$E\$17)^2)				=NORMDIST(k,\$E\$16,\$E\$17,TRUE)*100		
5	=500/(\$E\$17*SQRT(2*PI()))*EXP(-0.5*((k-\$E\$16)/\$E\$17)^2)				=NORMDIST(k,\$E\$16,\$E\$17,TRUE)*100		

	D	E	F
15			
16	Mean =	=SUMPRODUCT(k,n)/SUM(n)	kg/cm²
17	S =	=SQRT(SUMPRODUCT(n,(k-E16)^2)/SUM(n))	kg/cm²

Figure 2.12: Data presentation from the test result to obtain normal distribution and the formula used in spreadsheet

Formulas used as follows:

Mean : =SUMPRODUCT(k,n)/SUM(n)

$S = \text{SQRT}(\text{SUMPRODUCT}(n, (k - \text{Mean})^2) / \text{SUM}(n))$

Normal : $=500 / (S * \text{SQRT}(2 * \text{PI}())) * \text{EXP}(-0.5 * ((k - \text{Mean}) / S)^2)$

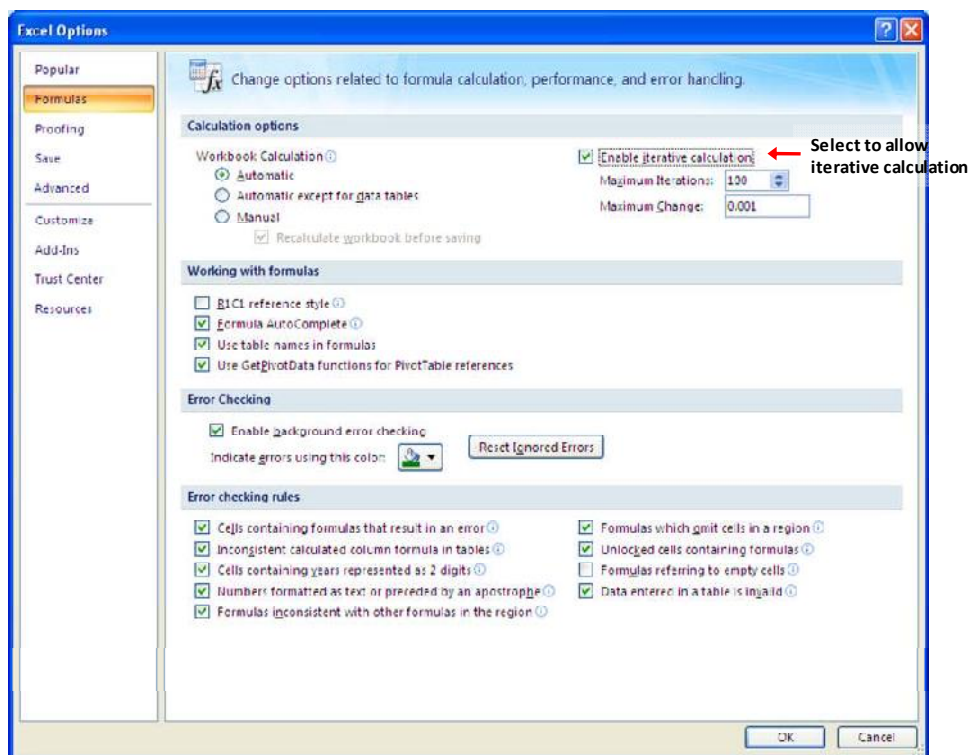
Normal Cumulative : $=\text{NORMDIST}(k, \text{Mean}, S, \text{TRUE}) * 100$

Mean and standard deviation are respectively, calculated using Equation 2:11 and 2:13. The actual scale for normal function (Equation 2.14) is determined from the interval used, in this case were taken 5 kg/cm². Multiply then by 100 to get the depiction in percent (%).

Excel has also the other ways to make distributions, such as Exponential, Weibull, Gamma and Poisson.

2.5.5 CIRCULAR REFERENCE

When formulas are connected with their own cell, either directly or indirectly, then this condition is called a circular reference. Excel by default can not process a formula that contains a circular reference until we select **Enable Iterative calculation** by the following steps: **Office Button > Excel Options > Formulas**. Iteration is the repetition of numerical calculation until a condition is met. The process is executed based on iterations number entered in the textbox **Maximum iterations** and **Maximum Change** for the maximum change resulting from the last two iterations. Excel will stop counting when one of the conditions is met, or whichever comes first.



Cell with a circular reference in a spreadsheet as shown below:

Example 1

	A	B
1	4	
2	=B2	=(A1+A2)/3

Result : 1 x iteration

	A	B
1	4	
2	0	1.33333

5 x iterations

	A	B
1	4	
2	1.97531	1.99177

100 x iterations

	A	B
1	4	
2	2	2

In short, it can be explained herein about the iteration method - to describe the way of Excel executes Example 1. Formulas in cell A2 and B2 are interrelated and have the form: $x = f(x)$ and calculated by the following equation:

$$x_{n+1} = f(x_n),$$

where: x_n = predicted value of x at iteration n^{th}

$$x_{n+1} = \text{predicted new value of } x \text{ at iteration } n^{\text{th}} + 1$$

n = number of iterations

Formula in cell sel A2 and B2 can be written: $x_{n+1} = (4 + x_n)/3$.

At first iteration ($n=1$), x_1 is taken = 0 $\rightarrow x_2 = (4 + 0)/3 = 1.33333$

At $n = 2$, x_2 is taken = 1.33333 $\rightarrow x_3 = 1.77778$

At $n = 5$, $x_5 = 1.97531 \rightarrow x_6 = 1.99177$

And so on.

Excel will do this process based on the number of iterations (n) and the maximum change $|x_{n+1} - x_n|$ that you specify, and will stop if one is met. This analogy is found in the process of Newton Raphson Method function in Chapter 3, but of course, the method is different.

Application of Excel iteration can also be applied to solve system of linear equations, to obtain unknown variables of equations. For an example, to obtain values of x, y which satisfy the equations below:

$$2x + 3y = 7$$

$$3x - 2y = 4$$

The solution is firstly to re-write each of equation to form x and y:

$$2x + 3y = 7 \rightarrow x = (7 - 3y)/2; \quad y = (7 - 2x)/3$$

$$3x - 2y = 4 \rightarrow x = (4 + 2y)/3; \quad y = (4 - 3x)/2$$

Next, convert the formed x and y into formulas and placed them in cell B2 until C3 as shown in the worksheet below. In order to perform iteration, the cell must be linked which is in this operation the formula in cell B3 and C2 are to be interrelated.

	A	B	C	D
1		Eq.1	Eq.2	
2	x =	$= (7 - 3*B3)/2$	$= (4 + 2*B3)/3$	
3	y =	$= (7 - 2*C2)/3$	$= (4 - 3*C2)/2$	
4				

10 x iterations

	A	B	C	D
1		Eq.1	Eq.2	
2	x =	1.999	2.000	
3	y =	1.000	1.001	
4				

11 x iterations

	A	B	C	D
1		Eq.1	Eq.2	
2	x =	2.000	2.000	
3	y =	1.000	1.000	
4				

Values of x and y that satisfy both equations are obtained at iteration 11, which are: $x = 2$ and $y = 1$.

Example 2

	A	B	C
1	0	10	10
2	10	$=(B1+B3+A2+C2)/4$	$=B2$
3	10	$=(B2+B4+A3+C3)/4$	$=B3$
4	0	0	0

1 x iteration

	A	B	C
1	0	10	10
2	10	5	5
3	10	3.75	3.75
4	0	0	0

5 x iterations

	A	B	C
1	0	10	10
2	10	8.64128	8.64128
3	10	6.18032	6.18032
4	0	0	0

100 x iterations

	A	B	C
1	0	10	10
2	10	8.75	8.75
3	10	6.25	6.25
4	0	0	0

In the cell B2 formula, the initial value for B3 and C2 are taken = 0. The value of B2 will be used as the initial value in the cell B3 formula, where C3 = 0. The values obtained are then used as a predicted new value in the next iteration.

In its application, for examples, the Excel iterative calculation or circular reference can be used in the solution of linear simultaneous equations, slope stability analysis using Bishop method (where the safety factor is on both sides of the equation), and the solution for seepage below dam using finite difference method.

CHAPTER 3

CREATING MACRO

3.1 FUNCTION PROCEDURE

In addition to the functions that already exist in Excel, it is often to create a function to perform calculations which is determined by the user. Such function is called user-defined (UD) function. This function will be added by Excel into a collection of functions and can be used as well as a built-in function.

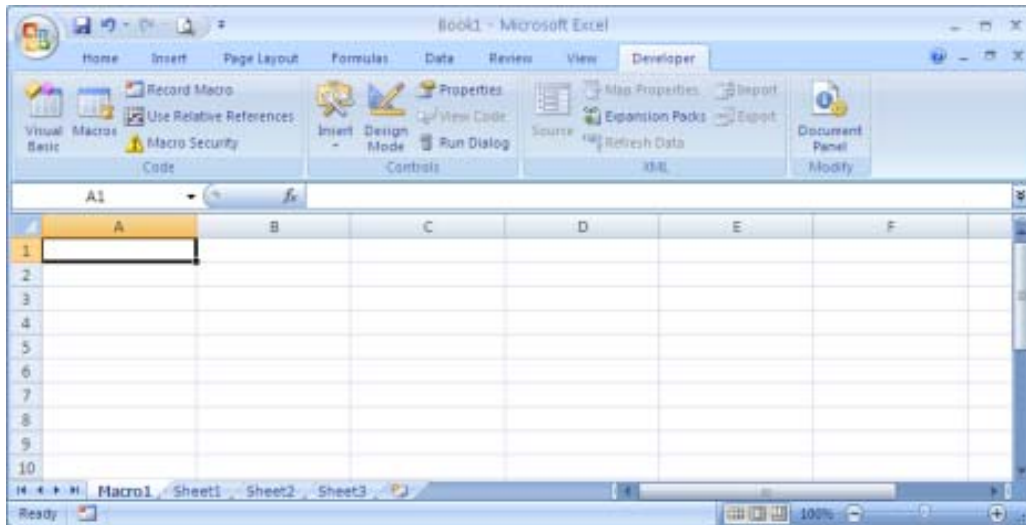
Some UD functions are made by combining formulas and built-in functions. UD function is made in macrosheet or using VBA. Macrosheet is a sheet where macro is created by XLM language, the original Excel macro language that is still maintained, at least until the Excel 2007. Figure 3.1 shows the UD functions to obtain cosine of an angle in macrosheet and VBA.

	A	
1	COSD	Function Cosd(x)
2	=RESULT(1)	Cons Pi = 3.141592655
3	=ARGUMENT("X",1)	Cosd = Cos(Pi * x/180)
4	=RETURN(COS(PI()*X/180))	End Function

Figure 3.1: Function in Macrosheet (left) and VBA (right)

Creating Function in Macrosheet

To insert a macrosheet in worksheet, right-click on the sheet tab to display a command menu, click **Insert > MS. Excel 4.0 Macro > OK**, to insert a macrosheet named **Macro1** as shown below:



This sheet is seemingly similar to a worksheet, but there are columns that are wide enough to write a function. The first row is to write the name of function, for example, COSD in macro example in Figure 3.1. The second row is to define the type of output refers to Table 3.1, for example, number, text or an array, using **RESULT** statement and followed by the data type. Next, is to define variables or input data using **ARGUMENT** statement, followed by the name and data type.

User has to determine formulas used in the operation of function of the input variables, which is defined by ARGUMENT. A variable can refer a name or cell reference of worksheet, such as "x" of COSD for a cell in the worksheet. A formula can be given a name to simplify writing the next formulas. The process in writing a function is ended by **RETURN** statement, which returns the function value. Below is an example of macrosheet macro to calculate volume of a cube:

	A	B	C
1		MVOL	
2		=RESULT(1)	
3		=ARGUMENT("a",1)	
4		=ARGUMENT("b",1)	
5		=ARGUMENT("h",1)	
6	Area	=a*b	Cell named "Area"
7		=RETURN(Area*h)	
8			

MVOL function is used in the worksheet as follows:

	A	B	C	D	E	F	G
1	length	width	height	Volume			
2	3	2	10	60	=MVOL(A2,B2,C2)		
3							

Table 3.1: Data Types in Macrosheet

Value	Data Type
1	Number
2	Text
4	Logical
8	Reference
16	Error
64	Array

Creating Function in Visual Basic for Application (VBA)

UD function is made in a module that inserted through **Insert** tab > **Module** in **Visual Basic Editor** which has been shown previously in Chapter 1. Function procedure is begin with word function, followed by its name and ends with the word End Function. The type of a function can be Single, Double or all data types of Variant, depending on what it would be produced, a number or an array. Data types in VBA are presented in Table 3.2.

The selection on these data types in addition to define a function and arguments, or variables, it is also related to the allocation of computer memory which depends on the range of the data type (the greater the value, the greater memory allocation). If an argument is not defined, it will be classified as a Variant and consumes memory more than any other data type. Defining data type in a programming is highly recommended because it saves computer memory consumption and accelerates the process of VBA in recognizing the data type as well, or in the other word to accelerate VBA to execute a macro.

Table 3.1: Data Types in VBA

Data Type	Range
Integer	-32,768 to 32,767
Long (long integer)	-2,147,483,648 to 2,147,483,647

Single (single precision)	-3.402823E38 to -1.401298E-45 (- values) 1.401298E-45 to 3.402823E38 (+ values)
Double (double precision)	-1.79769313486231E308 to -4.94065645841247E-324 (-) 4.94065645841247E-324 to 1.79769313486232E308 (+)
Currency	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
String (variable length)	0 to approx.2 billion characters
String (fixed length)	1 to approx. 65,400 characters
Byte	0 to 255
Boolean (logika)	TRUE or FALSE
Date	1 Januari 100 to 31 Desember 9999
Object	Any object reference
Variant (with numbers)	Any numeric value up to the range of Double
Variant (with characters)	0 to approx.2 billion characters

Such as Excel, VBA also has many built-in functions that have names similar to Excel built in functions, say, Cos function in Figure 3.1 or Abs to find the absolute value of a number. Some Excel built-in functions can also be used in VBA by using the prefix Application. Figure 3.2 is an example where Excel built-in functions are used in UD functions MLIN to obtain linear regression parameters. MLIN returns an array result (multiple result) which is the slope of the line and the intersection with Y-axis (Y-intercept).

```
Function MLIN(Y, X) As Variant
    Slope = Application.Slope(Y, X)
    Intercept = Application.Intercept(Y, X)
    MLIN = Array(Slope, Intercept)
End Function
```

Figure 3.1: Excel built-in functions in VBA

Notes:

- Module with Option Explicit requires all variables are to be declared. If not, VBA will show an error message "Variable not defined". To declare a variable use Dim, ReDim, Private, Public and Static statement.

Example

```

Option Explicit
Sub OptEx1()
Dim myVar as Integer
    MyInt = 5 'error: variable not defined
    myVar = 10
End Sub

```

- Dim is to declare a size of variable that already set at the beginning of the program, while ReDim is to change the size of variable that could be changed during the execution of program. Variables declared by Dim or Redim at procedure level (between Sub – End Sub statements) are available only to the procedure where the variables are declared.

Example

```

Sub OptEx2()
Dim myVar as Integer
Dim myString(1 to 3) as String
Dim n As Integer
n = 3 'an input

```

```

ReDim MyInt(n) As Integer
    MyInt(1) = 1
    MyInt(2) = 3
    MyInt(3) = 5
    myVar = 10
    myString(3) = "Hello"
End Sub

```

- Use **Type..End Type** with the example below to define a user-defined data type.

```

Type FamilyTree
    Name As String
    Address As String
    Phone As Long
    Numsiblings As Byte
End Type

Sub GetRecord()
    Dim TheFirst As FamilyTree
    TheFirst.Name = "Johan"

```

```

TheFirst.Address = "Street 123"
TheFirst.Phone = 1234
TheFirst.Numsiblings = 3
End Sub

```

3.2 SUB PROCEDURE

Macro that previously used to create a function can also be used to make an engineering calculation program to solve more complex problem, such as structural analysis. In VBA, such program is created in the Sub procedure, that is procedure with syntax as briefly discussed in Chapter 1. In general, the process of input - output data from a Sub procedure is:

1. Read input data from the worksheet
2. To execute data input by VBA
3. Print the results to the worksheet

In Excel-VBA programming, the value of input data is taken from the value of the cell or range of cells on worksheet or a Range object with value as shown by its Value property. So typing in the value in the cell in the worksheet with numbers, text, and so on actually is to set the value on Value property of these cells. Cells property is commonly used to express the Range object with the syntax Cells (row_index, column_index), i.e. cell (1,1) to return a Range object of cell A1.

Here is an example to set a number to a cell in VBA language, which sets the value of cell A1 = 10:

```
Worksheets("Sheet1").Cells(1,1).Value = 10
```

Value is the default property of the Range object, therefore negligible or means no need to be written again. If no object qualifier or an objects before period (Worksheets () above), then a range of cells are on the active sheet, in this case from which the macro runs, for example via a command button on the worksheet. Use an object qualifier before the range of cells if the data is not located on the active sheet with the writing of Worksheet ("sheet name").

To complete this topic of discussion, the following is a simple example of Sub procedure named Pro_Sub1:

```

Sub Pro_Sub1()
Dim Length As Double

```

```

Dim Width As Double
Dim Area As Double
    Length = Cells(2, 1)
    Width = Cells(3, 1)
'Determine Area
    Area = Length * Width
    Cells(4, 1) = Area
End Sub

```

The above procedure is to calculate the area of a rectangle which is the multiplication of the length and width of its sides. The variables are declared using the Dim statement and the data type is a number with double precision (Double). Cells property is created without an object qualifier, therefore, the range of cells is on the active sheet, which the active sheet (opened) when the code program is executed.

Sign apostrophe (') is a sign where program ignores the comments in it. This sign is used to insert an explanation. The default color of the text is green.

The process of input-output of Pro_Sub1 procedure can be explained as below:

	A	B	C	D	E	F
1			Sub Pro_Sub1()			
2	5					
3	6		Dim Length As Double			
4			Dim Width As Double			
5			Dim Area As Double			
6			Length = Cells(2, 1)			
7			Width = Cells(3, 1)			
8			'Count area			
9			Area = Length * Width			
10			Cells(4, 1) = Area			
11						
12			End Sub			

If the procedure in the above figure is executed, value of cell A4 or Cells (4,1) = 30. To run the macro, perform a sequence of steps as described in Chapter 1.11.

In creating large scale application program, normally, the program is divided into several sub-programs or procedures Sub and/or with some functions that each handles a specific task then they called with Call statement or the name of function. The advantages of this

division are to make a programming become more focused, more readable and if an error occurred it can be quickly addressed. Chapters 6 to 10 show the examples of such computer programming.

3.3 CONTROL STRUCTURES

A program is created to be able to handle any changes either from the data input or to that occurred during program execution. Therefore, there are necessary control structures in order to make program remains on its specified steps. Such controls are looping and branching or a combination of both.

3.3.1 LOOPING

Do While...Loop or Do...Loop While

Repeats a set of statements while a condition is TRUE. If condition becomes FALSE, the program will exit the looping and move to the next code.

Syntax:

Do While condition

[statements]

[Exit Do]

[statements]

Loop

Example

```
Sub exLoop1()  
    i = 1  
    Do While i <= 5  
        Cells(1 + i, 1) = i  
        i = i + 1  
    Loop  
End Sub
```

Do Until...Loop or Do...Loop Until

Repeats a set of statements until a condition becomes TRUE. If a condition becomes FALSE, the looping will be discontinued.

Syntax:

Do Until condition

[Statements]

[Exit Do]

[Statements]

Loop

Example

```
Sub exLoop2()  
    i = 1  
    Do Until i >= 5  
        Cells(1 + i, 1) = i  
        i = i + 1  
    Loop  
End Sub
```

For...Next

Repeats a set of statements a given number of times

Syntax:

For counter = start **To** end [**Step** step]

[statements]

[Exit For]

[statements]

Next [counter]

Example


```

Sub exLoop3()
j = 1
    For i = 1 To 5
        Cells(1 + i, 1) = j
        j = j + 1
    Next i
End Sub

```

For Each...Next

Repeats a set of statements a number of elements in an array. It helps if the number is unknown. For example, to be used in a function below:

Syntax:

For Each element **In** array

[statements]

[Exit For]

[statements]

Next [element]

Example

```

Function exLoop4(Yarray, Xarray) As Variant
Dim n As Integer, x As Single, y As Single
    n = 0
    x = 0
    y = 0
    For Each c In Xarray
        n = n + 1
        x = x + Xarray(n)
        y = y + Yarray(n)
    Next
    a = x * x
    b = y * x
    exLoop4 = Array(a, b)
End Function

```

3.3.2 BRANCHING

Select Case

Executes several set of statements depends on the value of an expression

Syntax:

Select Case testexpression

[**Case** expressionlist-n

[statements-n]]

[**Case Else**

[elstatements]]

End Select

Example

```
Sub exLoop5()  
Dim Number  
Number = Cells(1, 1) 'a number  
Select Case Number  
Case 1 To 5  
    Cells(2, 1) = "Between 1 and 5"  
Case 6, 7, 8  
    Cells(2, 1) = "Between 6 and 8"  
Case Else  
    Cells(2, 1) = "Greater than 8"  
End Select
```

If...Then...Else

Executes a set of statements because of certain condition, when it is not appropriate then move to another state of condition.

Syntax:

If condition (1) **Then**

statement(1)

Elseif condition(2) **Then**

statement(2)

Else

statement(3)

End If

Branching is usually nested in a looping. If the condition is met, the program will exit the looping with Exit command. If...Then...Else branching can also be combined with And, Or, or Not statement to state the expression or the relationship between two expressions.

Syntax:

For...or Do Until...

statement(1)

If statement(1) **And** or **Or** expression(2) **Then**

statement(2)

Exit For or **Exit Do** or **Exit Function** or **Exit Sub**

End If

Next...or Loop

Goto...Gosub...Return

This branching is used in a procedure consisting of one or several blocks of statements (subroutines) that is made for particular task or calculation. The block is then called upon to do the job with GOSUB statements and returned with RETURN statement. GoTo statement will branch to a code line, subroutines, or line label, and then continue to the next code.

Example

```
Sub exBranch()  
Dim myNum As Double  
myNum = Cells(2, 1)
```

```

    If myNum >= 0 Then
        GoSub MyRoutine
        GoTo 10
    Else 'means myNum < 0
        GoTo 20
    End If
MyRoutine:
    myNum = Sqr(myNum) 'determine square root of number >=0
    Return
10 Cells(3, 1) = myNum
    Exit Sub
20 Cells(3, 1) = "#NUM!" 'error message
End Sub

```

Call...Sub

Calls a procedure from another procedure

Syntax

[Call] name [argument]

Example 1: Calling Sub procedure from function procedure

```

Sub Area(x, y, L)
    L = x * y
End Sub

Function Volume(a, b, t) As Single
    Call Area(a, b, L)
    Volume = L * t
End Function

```

Example 2: Calling Sub procedure from another Sub procedure.

```

Sub Area(x, y, L)
    L = x * y
End Sub

```

```

Sub Pro_Vol()
Dim a As Single, b As Single, t As Single
Dim volume As Single

a = Cells(2, 2)
b = Cells(3, 2)
t = Cells(4, 2)

Call Area(a, b, L)
    volume = (L * t)
    Cells(5, 2) = volume
End Sub

```

In addition to looping and branching defined above, there is also useful control structure that is **With** statement.

With

Execute a series of statements on the same object

Syntax:

With object

[*statements*]

End With

Example:

```

With MyObject
    .HorizontalAlignment = xlCenter
    .VerticalAlignment = xlCenter
    .Position = xlLabelPositionRight
    .Orientation = xlHorizontal
    With .Font
        .Name = "Arial"
        .FontStyle = "Regular"
        .Size = 8
    End With
End With

```

3.4 USER DEFINED FUNCTION PROBLEMS

Problem 1

Create a macro to determine rectangular area where $\text{area} = \text{length} \times \text{width}$.

Solution

If a and b express length and width respectively, area in macro can be expressed as a function of $(a, b) = a \times b$. If the name of the function is Area, the expression is:

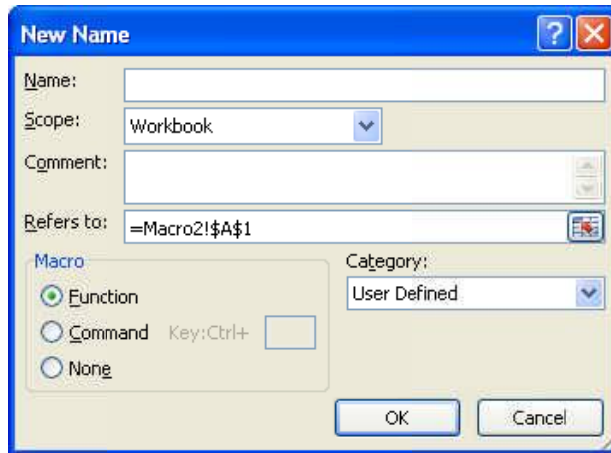
$\text{Area}(a, b) = a \times b$.

To solve this problem, we will use macrosheet language. Insert a macrosheet by right-clicking on the worksheet **Sheet** tab > **Insert** > **MS. Excel 4.0 Macro** > **OK**. Type the macro as follows:

	A	B
1	AREA	
2	=RESULT(1)	
3	=ARGUMENT("a",1)	
4	=ARGUMENT("b",1)	
5	=RETURN(a*b)	
6		

Afterward, we have to define the function name so that it can be recognized by Excel and incorporated it into Excel functions. The steps are as follows:

1. Click cell A1 where the name of the function (Area) is placed.
2. Click the **Formula** tab > **Define Name** to show the **New Name** dialog box as below. Enter a macro name in the name text box. Macro name will automatically be referred to cell A1 with step 1 above.



3. Select **Function** in the option button, so the function (by default) is categorized as **User Defined** or you can choose a category for your function.
4. Click **OK**.

Now back to the worksheet. **Area** function is written as follows:

	A	B	C	D	E
1	a (length)	b (width)	Area		
2	3	2	6 ← =Area(a,b)		
3					
4					

Cells A2 and B2 are respectively named a and b which carried out by follow step No. 1 and 2 above through active sheet. In this step, there is no option (option button) for Macro category in **New Name** dialog box.

Problem 2

Find the area between a straight line $y = mx + c$ and the X-axis, between $x = a$ and $x = b$ ($c \geq 0$). This will be solved using integral with formula:

$$\text{Area} = \int_a^b (mx + c) dx$$

In macro, area can be expressed as:

$$\text{function}(m,c,a,b) = (\frac{1}{2}.mb^2 + cb) - (\frac{1}{2}ma^2 + ca)$$

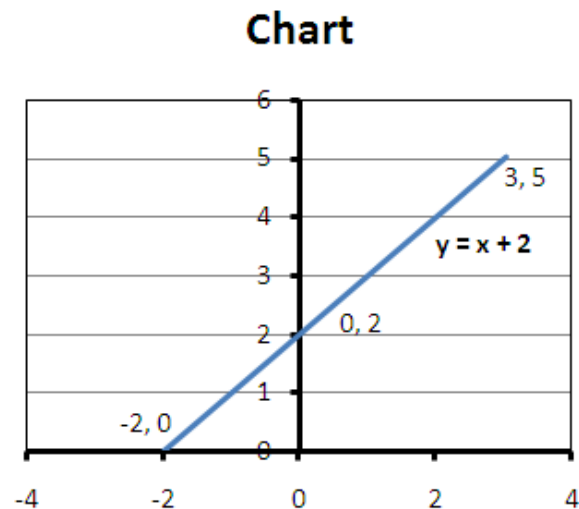
Now, based on the area formulation above, find the area between $y = x + 2$, X-axis, between $a = -1$, $b = 3$. The name of function is INAREA.

Solution

The macro in macrosheet looks like below:

	A	B
1	INAREA	
2	=RESULT(1)	
3	=ARGUMENT("m",1)	
4	=ARGUMENT("c_",1)	
5	=ARGUMENT("a",1)	
6	=ARGUMENT("b",1)	
7	=-c_/m	
8	=IF(m<0,IF(OR(a>A7,b>A7),RETURN(#VALUE!)))	
9	=IF(m>0,IF(OR(a<A7,b<A7),RETURN(#VALUE!)))	
10	=1/2*m*a^2+(c_*a)	
11	=1/2*m*b^2+(c_*b)	
12	=RETURN(A11-A10)	

Problem 2 in chart:



Before applying the function, perform step no.1 to 4 as in Problem 1. The use of function in worksheet as follows:

	A	B	C	D	E	F
1	m	c	a	b	Area	
2	1	2	-1	3	12	=INAREA(A2,B2,C2,D2)
3	1	2	-3	3	#VALUE!	=INAREA(A3,B3,C3,D3)
4	-0.5	2	1	5	#VALUE!	=INAREA(A4,B4,C4,D4)
5						

Problem 3

Create a VBA macro to assign letter grade for a course with score range 0 - 100 as follows, A: representing scores from 80 - 100, B: 65 - 79, C: 51 - 64, D: 40 - 50 and F: for scores below 40. Macro will return literal grade with sign + and - at given numerical value.

Solution

```
Function VBGrade(g, a, b, c, d) As String
```

```
    Select Case g
```

```
        Case Is >= 2 * (100 - a) / 3 + a
```

```
            VBGrade = "A+"
```

```
        Case Is >= (100 - a) / 3 + a
```

```
            VBGrade = "A"
```

```
        Case Is >= a
```

```
            VBGrade = "A-"
```

```
        Case Is >= 2 * (a - b) / 3 + b
```

```
            VBGrade = "B+"
```

```
        Case Is >= (a - b) / 3 + b
```

```
            VBGrade = "B"
```

```
        Case Is >= b
```

```
            VBGrade = "B-"
```

```
        Case Is >= 2 * (b - c) / 3 + c
```

```
            VBGrade = "C+"
```

```
        Case Is >= (b - c) / 3 + c
```

```
            VBGrade = "C"
```

```
        Case Is >= c
```

```
            VBGrade = "C-"
```

```
        Case Is >= 2 * (c - d) / 3 + d
```

```
            VBGrade = "D+"
```

```
        Case Is >= (c - d) / 3 + d
```

```
            VBGrade = "D"
```

```

Case Is >= d
    VBGrade = "D-"
Case Else
    VBGrade = "F"
End Select
End Function

```

Functions that created in VBA will automatically be entered into Excel functions collection. The use of the function in worksheet is as follows:

	A	B	C	D	E
1	Name	Score	Letter		
2	V. Magda	64	C+	=VBGrade(B2,80,65,51,40)	
3	Y. Artist	90	A	Result of copying cell from C2 to C3:C6	
4	L. Money	79	B+		
5	M. House	66	B-		
6	H. Aunt	39	F		
7					

If the teacher wants to apply a different standard for course scoring, it is simply done by changing value of the function arguments.

Problem 4

Create a VBA macro to find vertical stress distribution below a foundation with elastic equation from Boussinesq - Newmark (1935). The equation to get the vertical stress (σ) below a corner of a rectangular area $b \times l$ at depth y as shown in Figure 3.3 is:

$$\sigma_y = \sigma_0 \frac{1}{4\pi} \left[\frac{2MN\sqrt{V}}{V + V_1} \cdot \frac{V + 1}{V} + \tan^{-1} \left(\frac{2MN\sqrt{V}}{V - V_1} \right) \right]$$

where,

$$M = b/y$$

$$N = l/y$$

$$V = M^2 + N^2 + 1$$

$$V_1 = (MN)^2$$

b dan l is length of rectangular sides

If $V_1 > V$, then the arctan in the equation needs to be added with π .

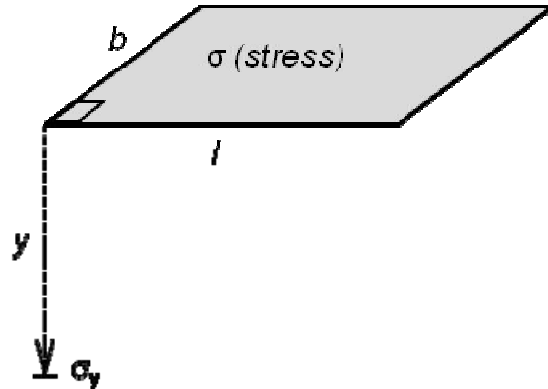


Figure 3.3: Vertical stress below a corner of a rectangular area

Solution

```
Function Bous(q, b, l, y) As Double
```

```
Const pi = 3.141593
```

```
Dim m, n, v, vs, rad, at
```

```
    m = b / y
```

```
    n = l / y
```

```
    v = m ^ 2 + n ^ 2 + 1
```

```
    vs = (m * n) ^ 2
```

```
    rad = (2 * m * n * Sqr(v)) / (v - vs)
```

```
    If vs > v Then
```

```
        at = Atn(rad) + pi
```

```
    Else
```

```
        at = Atn(rad)
```

```
    End If
```

```
    Bous = q / (4 * pi) * (((2 * m * n * Sqr(v) / (v + vs)) * ((v + 1) / v)) + at)
```

```
End Function
```

The advantage to convert an equation or formula into a function is that it greatly simplifies the calculation. The above equation with a condition $V_1 > V$ certainly would be very lengthy and requires a lot attention if built into worksheet formula. It is more easily solved by typing it into VBA code.

Now, given a foundation with a size of 3 x 3 m is subjected to uniform pressure of 10 ton/m². The vertical stress distribution of each interval depth of 0.5 meters below the **center** of the foundation is shown in the worksheet as below:

	A	B	C
1	Uniform Pressure on Rectangular Area		
2	q =	10 ton/m ²	
3	b =	1.5 m	
4	l =	1.5 m	
5			
6	Vertical Stress Distribution		
7	y(m)	py (t/m ²)	
8	0.00	10.00	=4*Bous(\$B\$2,\$B\$3,\$B\$4,A8)
9	0.50	9.76	<div>Result of copying cell from B8 to B9:B20</div>
10	1.00	8.63	
11	1.50	7.01	
12	2.00	5.49	
13	2.50	4.28	
14	3.00	3.36	
15	3.50	2.68	
16	4.00	2.17	
17	4.50	1.79	
18	5.00	1.49	
19	5.50	1.26	
20	6.00	1.08	
21			

- Boussinesq - Newmark equation is intended to find vertical stress below a corner of a rectangular area, therefore, below a center a foundation of 3 x 3 m is obtained 4 area with the same size, with 1.5 x 1.5 m dimension. Furthermore, for the contribution of four corners it shall be calculated = 4 σ_y .
- Division by 0 is not allowed in the calculation, therefore the value entered in cell A8 shall be an approach = 0 (take, for example, y = 0.0001).

Chart of stress versus depth is shown as below:

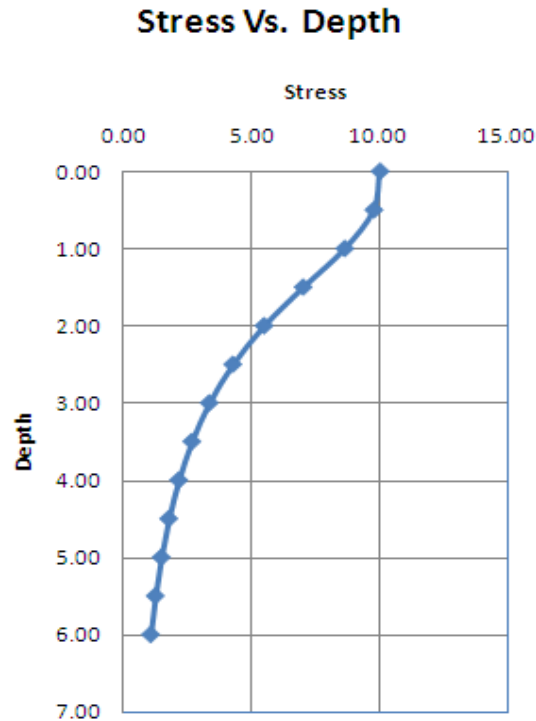


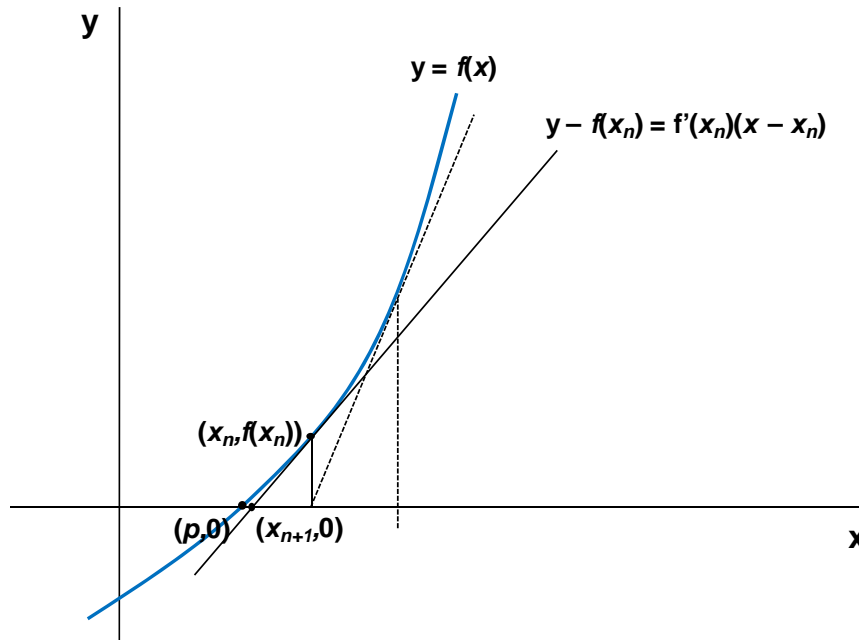
Chart is created through **Insert** tab > **Scatters with smoothlines and markers** > **Select Data** > **Add** and then select x and y-data. To modify the chart click once on chart area then click **Chart Tools** tab > **Chart Layout** > **Layout 1**, and then add chart titles by clicking once on the title, select **Edit Text** to type. The depth orientation is made to increase downward by right clicking the Y-axis > **Format Axis** > **Axis Option** > **Values in reverse order**.

Problem 5

Find root x in equation: $ax^4 + bx^3 + cx^2 + dx + c = 0$

Solution

This problem will be solved numerically by Newton-Raphson method. The process of calculation is done in an iterative technique, which root x is gradually approached. Function of equation depicted in chart in Figure 3.4 and exemplified intercepts the X-axis at the point p and $f(x) = 0$ or $(p, 0)$. In the first step, it is taken x_1 -value to get tangent line at $y = f(x)$ which intercepts the X-axis at the point $(x_2, 0)$, where x_2 is new approach value to p. The next approaching process is done using x_2 -value to get x_3 -value and so on until $x_n \rightarrow p$.



Thus, by induction it obtains tangent line: $y - f(x_n) = f'(x_n) (x - x_n)$ which intercepts X-axis at point $(x_{n+1}, 0)$, where:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Macro

```
Function NRM(a, b, c, d, e) As Double
Dim funct, deriv, x, xn
Dim n As Integer, i As Integer
n = 1000
i = 1
x = 100
Do
    funct = (a * x ^ 4) + (b * x ^ 3) + (c * x ^ 2) _
    + (d * x) + e
    deriv = (4 * a * x ^ 3) + (3 * b * x ^ 2) + _
    (2 * c * x) + d
    xn = x - (funct / deriv)
    If funct = 0 Or Abs(xn - x) < 0.00001 Then
        NRM = xn
    End If
    x = xn
    i = i + 1
Loop Until i = n
```

```

Exit Function
End If
i = i + 1
x = xn
Loop Until i = n
NRM = "Divergen!"
End Function

```

Function NRM (stands for Newton-Raphson Method) is now used to find the value of x in the equation below:

$$-2.79x^3 - 2.11x^2 + 24.51x + 36.68 = 0.$$

This equation form can be found in the geotechnical engineering to find the embedment depth of retaining wall with Free-Earth Support Method. The use of the function in worksheet is as below:

	A	B	C	D	E	F	G	H	I
1	Root of Equation								
2									
3	ax^4	+ bx^3	+ cx^2	+ dx	+ e	= 0			
4	0	-2.79	-12.11	24.51	36.68	x =	2.2325	=NRM(A4,B4,C4,D4,E4)	
5									

Problem 6

Table X is a number of random numeric data from 0 - 100. Create macro to calculate the amount of data in Table X with a range of values from Y to Z.

	A	B	C	D	E	F	G
1	Tabel X						
2	77.58	73.27	22.21	70.40	81.40	52.57	98.34
3	68.58	21.86	62.37	37.61	51.24	12.41	18.55
4	60.92	22.26	35.01	23.76	63.34	32.81	83.16
5	98.27	63.85	30.97	34.98	64.44	70.49	17.96
6	71.23	79.63	74.21	100.00	37.95	3.94	95.27
7	10.62	20.69	5.82	74.81	95.09	37.48	0.74

Solution

Macro

```
Function DNUM(Xarray, Y, Z) As Integer
Dim n As Integer, Num As Integer
Dim c
n = 0
    For Each c In Xarray
        n = n + 1
        If Xarray(n) >= Y And Xarray(n) <= Z Then Num = Num + 1
    Next
DNUM = Num
End Function
```

The use of the function in worksheet is as below:

	I	J	K	Formula at Column K
2	0.0	50.0	19	=DNUM(X,I2,J2)
3	50.0	100.0	23	=DNUM(X,I3,J3)
4	0.0	100.0	42	=DNUM(X,I4,J4)

Argument X is an array argument, in this example is the name that has been defined for the table data in a range A2 to G7. Argument array can be also written in the following manner, **range(A2:G7)** or a certain range, for example **range(B2:C7)**, **range(A7:G7)** and so on. One of the simplicities of using function in Excel because it is well supported by the spreadsheet to create variable names, writing function and put the result at once.

Problem 7

Create a macro for linear interpolation example in Chapter 2.5.3. Use Equation 2.7 to solve.

Solution

Macro

```
Function Linter(Xrange, Yrange, nval) As Double
    Dim i As Integer
    Dim c, x
    x = Xrange(1)
    i = 1
    For Each c In Xrange
        If nval = c Then
            Linter = Yrange(i)
        End If
        i = i + 1
    Next
End Function
```



```

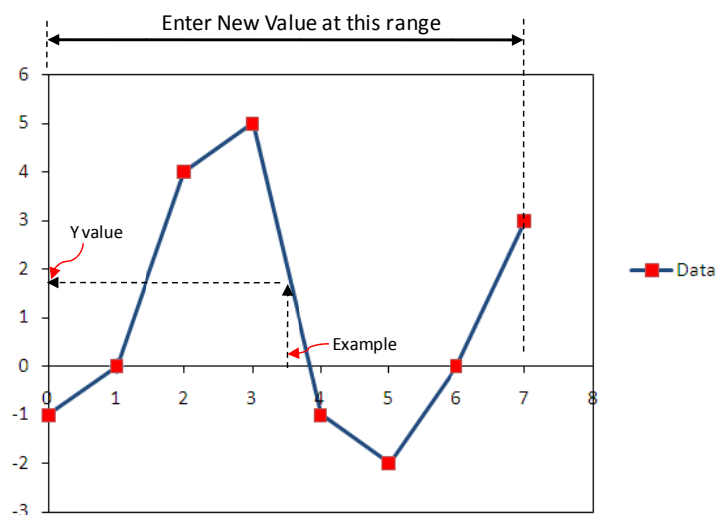
Exit Function
'find a new value satisfying the given condition
ElseIf (nval - c) * (nval - x) <= 0 And c <> x Then
    'Eq. 2.7:
    Linter = Yrange(i - 1) + (Yrange(i) - Yrange(i - 1)) _
    * (nval - x) / (c - x)
Exit Function
End If
x = c
i = i + 1
Next
End Function

```

The use of the function in worksheet is as below:

	A	B	C	D	E
1	Linear Interpolation				
2	Data X	Data Y	New X	Y Value =	
3	0	-1	3.54	1.76	=linter(A3:A10,B3:B10,C3)
4	1	0			
5	2	4			
6	3	5			
7	4	-1			
8	5	-2			
9	6	0			
10	7	3			
11					

Interpolation in Chart:



Problem 8

Create a macro for semilog interpolation example in Chapter 2.5.3. Use Equation 2.10 to solve.

Solution

Macro

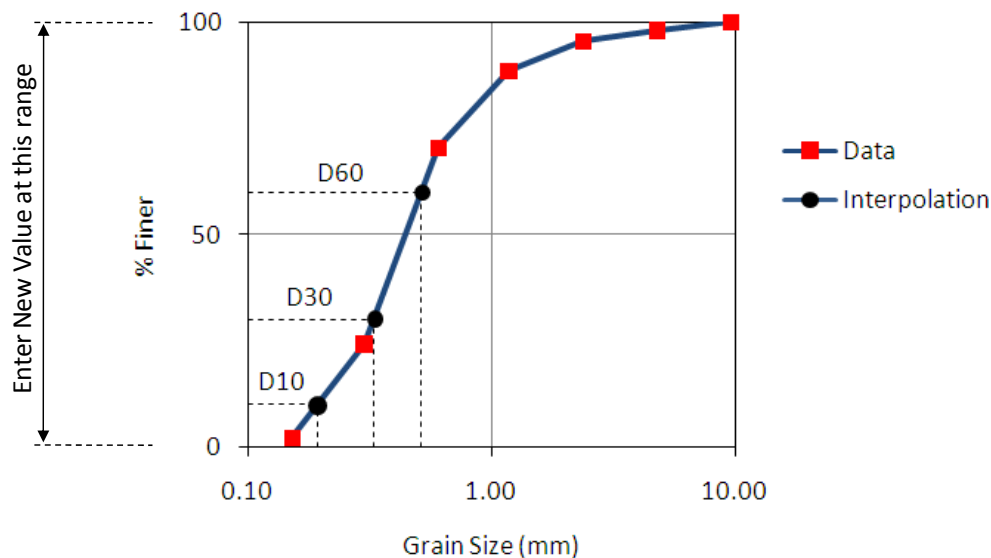
```
Function Loginter(Xrange, Yrange, nval) As Double
    Dim i As Integer
    Dim c, x
    x = Xrange(1)
    i = 1

    For Each c In Xrange
        If nval = c Then
            Loginter = Yrange(i)
            Exit Function
        'find a new value satisfying the given condition
        ElseIf (nval - c) * (nval - x) <= 0 And c <> x Then
            'Eq 2.10:
            Loginter = Yrange(i - 1) * (Yrange(i) / Yrange(i - 1)) _
                ^ ((nval - x) / (c - x))
            Exit Function
        End If
        x = c
        i = i + 1
    Next
End Function
```

The use of the function in worksheet is as below:

	A	B	C	D	E	F
1	Grain Size Distribution					
2	x's	y's				
3	Grain Size	Finer				
4	(mm)	(%)				
5	9.50	100.00		100.00	9.50	=loginter(B5:B11,A5:A11,D5)
6	4.75	97.80		97.80	4.75	Result of copying cell from E5 to E6:E11
7	2.36	95.48		95.48	2.36	
8	1.18	88.50		88.50	1.18	
9	0.60	70.60		70.60	0.60	
10	0.30	24.10		24.10	0.30	
11	0.15	2.40		2.40	0.15	
12						
13			D10 =	10	0.191	=loginter(\$B\$5:\$B\$11,\$A\$5:\$A\$11,D13)
14			D30 =	30	0.328	=loginter(\$B\$5:\$B\$11,\$A\$5:\$A\$11,D14)
15			D60 =	60	0.512	=loginter(\$B\$5:\$B\$11,\$A\$5:\$A\$11,D15)

Interpolation in Chart:



Notes:

- Character "c" and "r" can not be used for variable names in macrosheet, because C and R stand for column and row, respectively. However, you can define a variable names such as "c_" or "r_", "cx" or "rx" and so on.

3.5 STRUCTURE OF PROGRAM

Data processing in Excel-VBA is a sequence of step; **read – execution – print**, as described previously in Section 3.2, which is (rewritten for your you convenience):

1. Read input data from the worksheet
2. To execute data input by VBA
3. Print the results to the worksheet

In large data processing either from the input data or from program execution, a computer programming (coding) must be planned and managed properly. It is intended in order to make the program flow becomes simple (not complex), saving coding, and it is easy to solve if an error is occurred. In this section, the three of the above steps will be discussed that is how to create Input Form, Output Form and writing code in VBA module. In this topic, we are going to create a civil structural analysis program.

3.5.1 INPUT OUTPUT FORM

In Excel-VBA programming structure, input-output form is worksheet itself and can be made in the same sheet, e.g. on Sheet 1. For large input data, for an example for structural analysis program as shown in Figure 3.4, it would be better if the input data extends to the right, for an example, the data entered here is consist of number of joint, coordinates and properties of structure elements. By creating form widens on the right side, the location (read: rows) of data output is therefore can be specified in the program.

INPUT							
Joint Coordinates:							
Joint No.	1	2	3	4	5	6	7
X_coord	0.0	2.0	4.0	6.0	8.0	11.0	14.0
Y_coord	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Element Information:							
El No.	1	2	3	4	5	6	
J1st	1	2	3	4	5	6	
J2nd	2	3	4	5	6	7	
Density	0.00	0.00	0.00	0.00	0.00	0.00	
A_Sec	0.090	0.090	0.090	0.090	0.114	0.114	
E	2.10E+06	2.10E+06	2.10E+06	2.10E+06	2.10E+06	2.10E+06	
inertia IZ	0.000675	0.000675	0.000675	0.000675	0.001013	0.001013	
Support Condition:							
El No.	1	5	7				
R_x	0	0	1				
R_y	1	1	1				
R_z	0	0	1				
Joint Load:							
Joint	3						
Px	0.00						
Py	-10.00						
Mz	0.00						
Uniform Load:							
Wemns	5	6					
Load*	4.00	4.00					
Note: *Perpendicular to member axis (unit FL^-1)							
OUTPUT							
Member End Forces							
Joint	Px	Py	Mz	Dx	Dy	Dz	Member End Forces
							1-Axial 2-Shear 3-Mom 4-Axial 5-Shear 6-Mom
1	0.000	0.000	0.000	0.00000	0.00000	-0.01488	1/2.0 0.000 1.227 0.000 0.000 -3.227 6.455
2	0.000	0.000	0.000	0.00000	-0.02672	-0.01033	2/2.0 0.000 1.227 -8.455 0.000 -3.227 12.910

Figure 3.4: Input-Output Example Form

Figure 3.4 shows that how much the input data entered in the form does not change the row numbers for data output. It thus obtains a fixed input-output form and does not depend on the size of the input data, so that code of read - print steps can be specified in the program.

Contrary to the input data, it is possible to extend the output data downward because end of the row is not a boundary that has to be identified for program input. Therefore, the data presented here following the common way in presenting the results of the structure analysis extending downward to improve readability.

3.5.2 WORK WITH MODULES

A large data processing such as for an application program generally consists of several smaller sub-programs united into one unit for a goal that is well managed. What is meant by sub-program is VBA Sub procedure or function procedure given the scope of their respective task and then placed into a module.

For the convenience, Author distinguishes each procedure into three modules according to their tasks as follows:

Module 1: Main Module

Module 1 consists of data provision from input-output form (see Figure 3.3), which is mainly from: geometry data, materials data, determination of node support or fixities and load data. Module 1 is the main module, which controls the process of read - execution - print.

Module 2: Calculation Process (Analysis)

Procedures in module 2 are associated with the theory of the structure and the analytical methods used, e.g. building stiffness matrix, generate load matrix and compute nodal displacement. Module 2 works based on data provided by Module 1.

Module 3: Creating Charts

Procedures in module 3 are associated with the presentation of data in graphical form, e.g. for 1). Input data: geometry of the structure and loads, and 2). Data output: displacement, moment and shear distribution. Module 3 works based on data provided by Module 1 and Module 2.

Figure 3.5 shows in general how Excel-VBA programming structure is depicted in a chart. Figure 3.6 shows a chart of programming structure in real world; consist of main procedures (abbreviated to facilitate in writing) that exist in TRUSS2D, program for truss analysis as presented in Chapter 7.

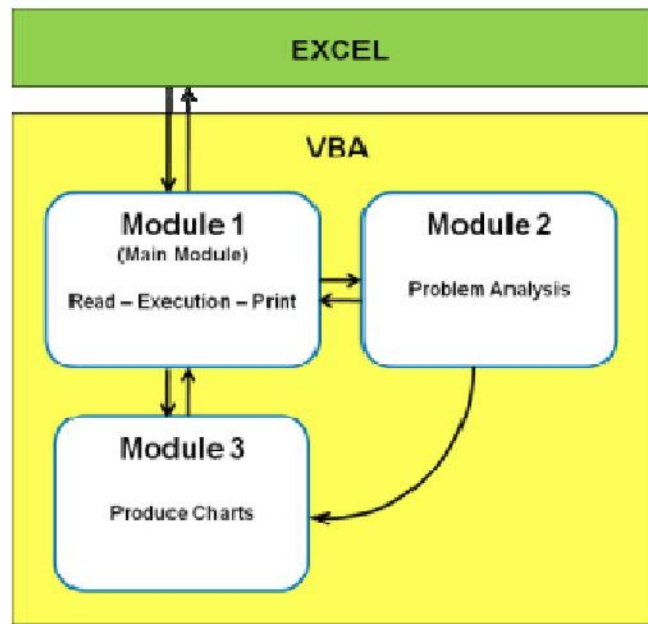


Figure 3.5: Excel-VBA programming structure

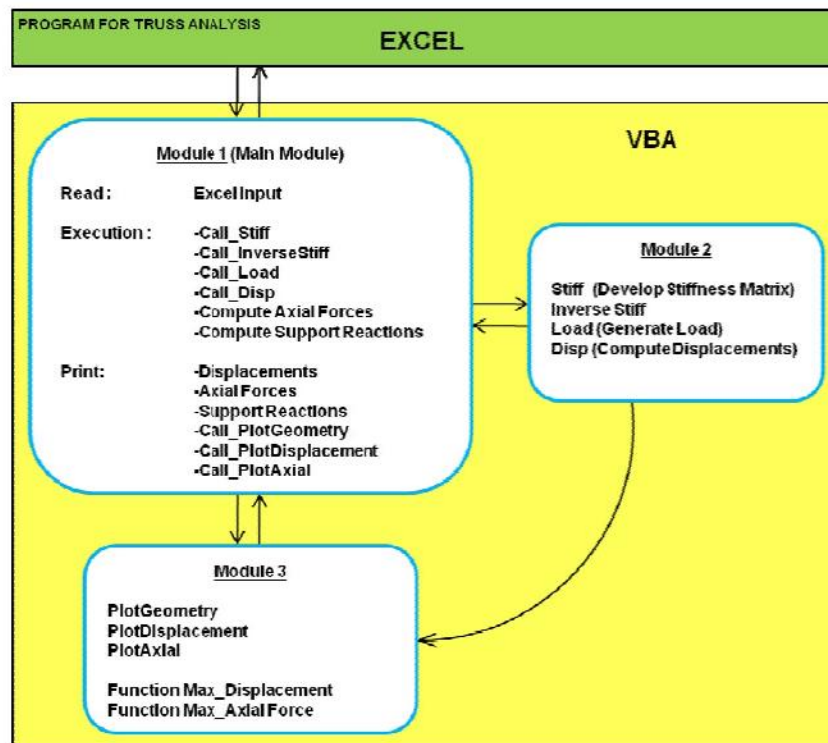


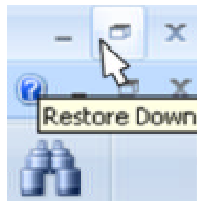
Figure 3.6: Excel-VBA programming structure for truss analysis (TRUSS2D)

3.5.3 TIPS

In designing macros, the movement between worksheet and Visual Basic Editor (VBE) window is often done because both windows are interconnected. To open VBE, click on **Visual Basic** icon **Developer** tab and from VBE click **Microsoft Excel** icon to go back to worksheet, or by pressing shortcut **Alt+F11** on both windows. Actually, to accelerate the process of displaying window is to place together in the monitor screen as shown in Figure 3.7. Thus, if a VBA program code is executed, its interaction to the worksheet can be viewed simultaneously.

Here are the steps to put a worksheet and VBE in one screen and simple techniques to execute line by line:

1. In the worksheet window click **Restore Down** icon in the top right corner, then drag the sheet to the left, place it approximately half size of the monitor screen. Drag the mouse pointer on the worksheet window ends to adjust its size.



2. Next, go to the VBE window, in the same way place it on the right side of the monitor screen. There are now already both these windows simultaneously.
3. Now, go to the VBE by clicking its window. Click **Close VBA Project** icon to close VBA Project window on the left so only VBA module is now visible (to display back, click icon **Project Explorer** or press **Ctrl+R**). To start from the first line press **F8** and press **F8** again to continue to the next line. If you want to jump to the desired line press **Ctrl+F8** at the cursor. Each line to be executed is marked in yellow.
4. Press **F9** to select program line at which execution will stop (**breakpoint**) when you run macro by clicking **Run Macro (F5)**. The other way is to click the left bar at program line that you want to set a breakpoint until red dot appears. Click on the red dot to remove.
5. The variable value computed during program execution can be seen its value by attaching a pointer on a variable, as shown in Figure 3.8. This variable reads the value of the cell of the worksheet. Hold on a second the pointer until the value is displayed, which is the cell value of the worksheet on the left.

Through step 1 and 2, the movement between worksheet - VBE is simply done by clicking one window without losing another window. This kind of method is very helpful if you want to run the program and see the results at once (through step 3 and 4).

The above steps are very helpful in developing flow of program and solve errors as well. Running step 3 and 4 is actually the process of tracing errors (debugging) by using commands in the Debug menu and it is also similar to the process of program verification (testing) by running code line by line. Testing and debugging is a routine activity to build a program that requires a number of control structures (branching and looping). A condition where a program is paused for the purpose of developing the code is called **break mode**.

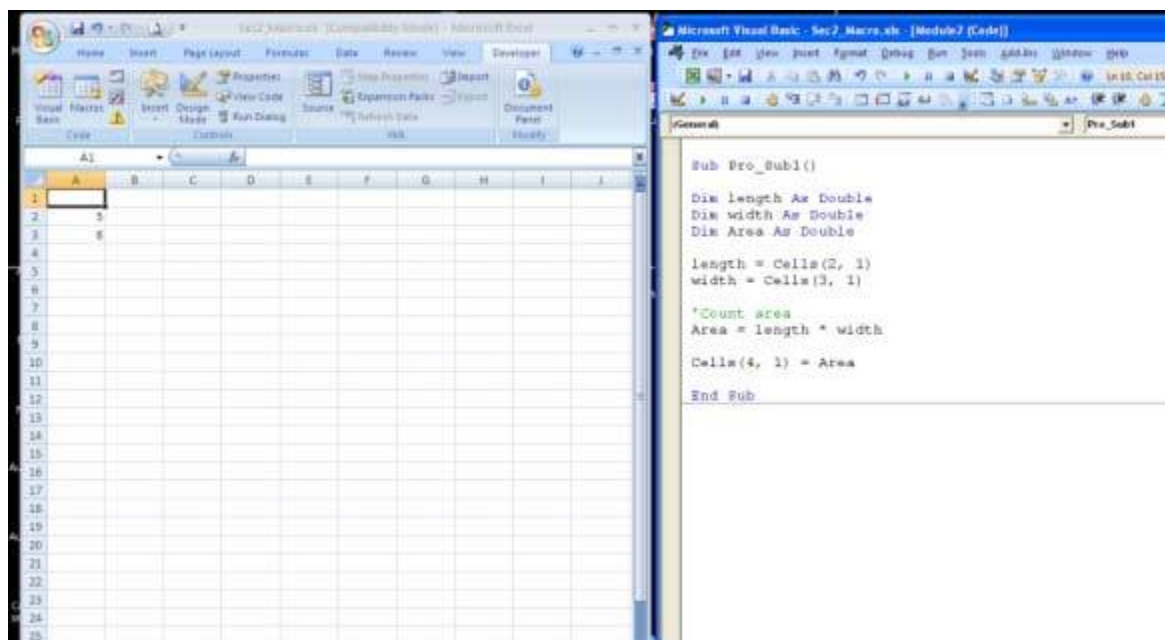


Figure 3.7: VBA – Worksheet window is placed in one screen

One of error type that may occur is when element of an array smaller or larger than the size that has been assigned to, so when program is running it will show an error message box "subscript out of range". For example in a looping where the number of process exceeds the subscript of an element. To get to the location where an error occurs clicks **Debug**, and **Reset** to stop the program.

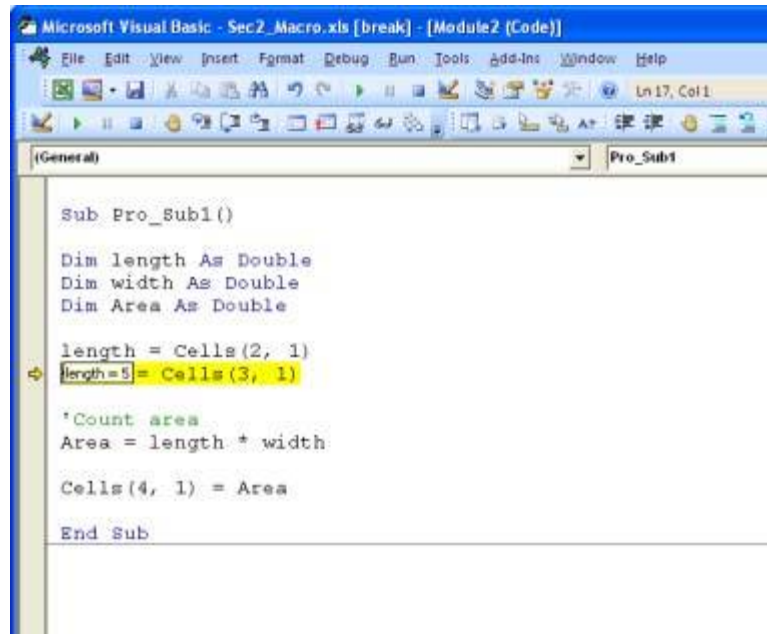


Figure 3.8: To see a variable value by attaching pointer

3.6 CHART MACRO

Excel chart is an object that can be manipulated by VBA through its methods and properties, for example, in an aim of adding lines (**Add** series), changing the line color, border color, creating line styles and setting the thickness (**Weight**). By creating macros, the job becomes very easy, fast, and advances to the most difficult level, which would not be done manually.

In Chapter 1 we have discussed how to make lines in the worksheet with formulas and tables. In this section, it will be created using macro instead of what has done manually. For a case, Example 3 of Chapter 1.10 will be used here - which is to create truss structure drawing with joint and coordinate tables as shown below:

Table of joints

Joint	x	y
1	1.0	0.0
2	1.5	2.5
3	2.0	5.0
4	2.5	10.0
5	3.0	15.0
6	3.5	20.0

7	4.0	15.0
8	4.5	10.0
9	5.0	5.0
10	5.5	2.5
11	6.0	0.0
12	3.5	2.5

Table of Coordinates

Line	Joint		x1	y1	x2	y2
1	1	2	Data is programmed			
2	2	3				
3	3	4				
4	4	5				
5	5	6				
6	6	7				
7	7	8				
8	8	9				
9	9	10				
10	10	11				
11	2	12				
12	12	10				
13	3	12				
14	12	9				
15	3	8				
16	3	9				
17	4	9				
18	4	8				
19	4	7				
20	5	8				
21	5	7				

By creating macro, the data of x1, y1, x2, y2 of above table do not need to be formulated because it will automatically be created by the program. The Macro is shown as follows:

```
Sub MChart1()
Dim npt, nln, i
npt = 12 'no. of joint
nln = 21 'no. of line
ReDim x(npt), y(npt)
```

Now, we focus on three blocks of statements enclosed by a red line, as below. The blocks description are given thereafter.

```

Sub MChart1()

Dim npt, nln, i

npt = 12 'no. of joint
nln = 21 'no. of line

ReDim x(npt), y(npt)
ReDim x1(nln), y1(nln), x2(nln), y2(nln)

'Read joint coordinates
For i = 1 To npt
    x(i) = Cells(4 + i, 2)
    y(i) = Cells(4 + i, 3)
Next i

'Read line coordinates
For i = 1 To nln
    x1(i) = x(Cells(4 + i, 6))
    y1(i) = y(Cells(4 + i, 6))
    x2(i) = x(Cells(4 + i, 7))
    y2(i) = y(Cells(4 + i, 7))
Next i

ActiveSheet.ChartObjects("Chart 1").Activate
ActiveChart.ChartArea.Select
Selection.ClearContents

'creating lines
For i = 1 To nln
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = Array(x1(i), x2(i))
        .SeriesCollection(i).Values = Array(y1(i), y2(i))
        .SeriesCollection(i).Name = "L" & i
    End With
    ActiveChart.SeriesCollection(i).Select
Next i

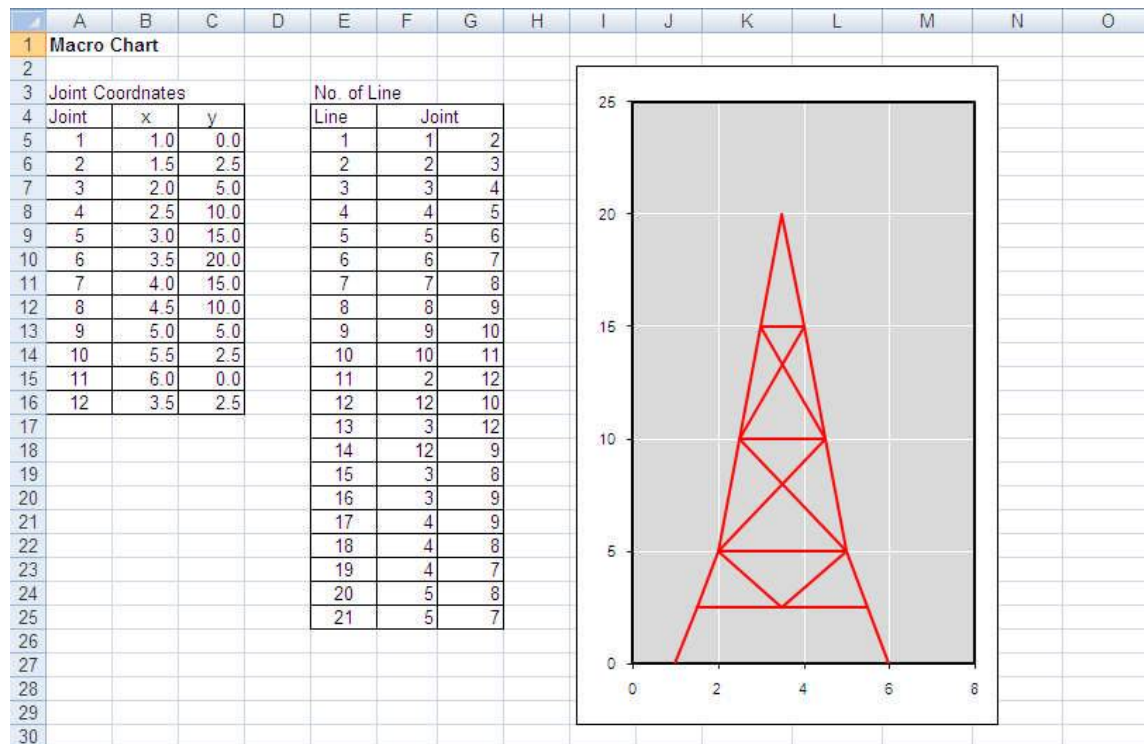
End Sub

```

Description:

- Block 1 is the same to that done using VLOOKUP formula in Chapter 1. This is the emptied line coordinates in the sheet table above because it is now processed by VBA. Figure out this, formulas as many as 21 rows x 4 columns (84 formulas) for creating 21 lines be is simply replaced with only 6 VBA statements without limit to any a number of lines that will be created, 100 or 200, 1000, etc.!
- Block 2 is code to activate a Chart that to be manipulated, in this case is Chart1. Note that, we will never (ever) create a new Chart (**Charts.Add**) because it will make code longer. Instead, we will proceed manually (through **Insert > Charts > Scatter**) and format the chart as desired. To identify a chart name right-click on the **Chart Area > Assign Macro**, and then see the name in the **Macro Name** text box.
- Block 3 is steps that we do with **Add Series** in the **Select Data Source** dialog box for data entry (see Chapter 1). This VBA looping replaces job for line coordinates data entry which takes a long time when done manually.

Excel input form for the chart macro above as below:



If you want to go further in creating chart it can be continued by setting its properties, such as line color, marker style and color, or line style (continuous, dashed). The following is code to set the Chart properties (code enclosed by red lines)

```

Sub MChart1()
Dim npt, nln, i
npt = 12 'no. of joint
nln = 21 'no. of line
ReDim x(npt), y(npt)
ReDim x1(nln), y1(nln), x2(nln), y2(nln)

'Read joint coordinates
For i = 1 To npt
    x(i) = Cells(4 + i, 2)
    y(i) = Cells(4 + i, 3)
Next i

```

```

'Read line coordinates
For i = 1 To nln
    x1(i) = x(Cells(4 + i, 6))
    y1(i) = y(Cells(4 + i, 6))
    x2(i) = x(Cells(4 + i, 7))
    y2(i) = y(Cells(4 + i, 7))
Next i

    ActiveSheet.ChartObjects("Chart 1").Activate
    ActiveChart.ChartArea.Select
    Selection.ClearContents

'creating lines
For i = 1 To nln
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = Array(x1(i), x2(i))
        .SeriesCollection(i).Values = Array(y1(i), y2(i))
        .SeriesCollection(i).Name = ""
    End With

    ActiveChart.SeriesCollection(i).Select
    With Selection
        .MarkerBackgroundColorIndex = xlNone
        .MarkerForegroundColorIndex = xlNone
        .MarkerStyle = xlNone
        .Smooth = False
    End With
    With Selection.Border
        .ColorIndex = 3 'red
        .Weight = xlMedium
        .LineStyle = xlContinuous
    End With
Next i

End Sub

```

The above examples is about setting the type and color of lines, which is respectively, continuous (xlcontinuous) and red (color index = 3). This code uses Border property of the Selection object to return Border object with its properties that are ColorIndex, Weight and LineStyle. Selection represents SeriesCollection(index) object of ActiveChart object that is an active chart named Chart 1.

3.7 MANIPULATION ON PROGRAM STEPS

An application program (for specific application) built through a sequence of steps that took quite long process, start from defining problems, to find solutions, build algorithms, to write and debug the code. An algorithm is defined as a sequence of logical steps in problem solving.

Algorithm is closely related to the use of the control structures in program that are branching and looping that have been discussed earlier. In array data basis such as structural analysis, it is necessary to have little bit of knowledge how to manipulate of program steps or flow of array data using control structures. For examples, is to read and print data, changing the composition of elements of an array and create integer variables for storing data.

Example 1

Given a simple example; look at an array data in the worksheet as shown below. Build elements of array on the left side to be as on the right side, where only the main diagonal value will still remain while the other = 0.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1													
2	INPUT							OUTPUT					
3	1	7	13	19	25	31		1	0	0	0	0	0
4	2	8	14	20	26	32		0	8	0	0	0	0
5	3	9	15	21	27	33		0	0	15	0	0	0
6	4	10	16	22	28	34		0	0	0	22	0	0
7	5	11	17	23	29	35		0	0	0	0	29	0
8	6	12	18	24	30	36		0	0	0	0	0	36
9													
10													

Here is a code to solve Example 1, starts from reading the data, manipulating the data (code in red line) and print the results:

```

Sub ExSTEP1()
Dim i, j, n
ReDim myArray(6, 6) As Integer

```

```

'Read input data
For i = 1 To 6
    For j = 1 To 6
        myArray(i, j) = Cells(2 + i, j)
    Next j
Next i

```

```

'Manipulate data
For i = 1 To 6
n = myArray(i, i)
    For j = 1 To 6
        myArray(i, j) = 0
    Next j
myArray(i, i) = n
Next i

```

```

'Print ouput data
For i = 1 To 6
    For j = 1 To 6
        Cells(2 + i, 7 + j) = myArray(i, j)
    Next j
Next i
End Sub

```

The code (outlined in red line) is the Author's version. Every programmer must has own steps to proceed, based on his/her interpretation in the same problem. Similarly, if you are a programmer or whose accustomed to use Excel-VBA.

Example 2

Create an elements of an array in Example 1 in sequence from bottom to top or from the largest subscript (myArray (6,6)) to the smallest (myArray (1,1)).

	A	B	C	D	E	F	G	H	I	J	K	L	M
1													
2	INPUT							OUTPUT					
3	1	7	13	19	25	31		36	30	24	18	12	6
4	2	8	14	20	26	32		35	29	23	17	11	5
5	3	9	15	21	27	33		34	28	22	16	10	4
6	4	10	16	22	28	34		33	27	21	15	9	3
7	5	11	17	23	29	35		32	26	20	14	8	2
8	6	12	18	24	30	36		31	25	19	13	7	1
9													
10													

The code (The Author's version):

```

Sub ExSTEP2()
Dim i, j, m, n
ReDim myArray(6, 6) As Integer
ReDim rvArray(6, 6) As Integer

'Read input data
For i = 1 To 6
    For j = 1 To 6
        myArray(i, j) = Cells(2 + i, j)
    Next j
Next i

'Manipulate data
m = 0
n = 0
For i = 6 To 1 Step -1
    n = n + 1
    For j = 6 To 1 Step -1
        m = m + 1
        rvArray(j, i) = myArray(m, n)
    Next j
m = 0
Next i

'Print data
For i = 1 To 6
    For j = 1 To 6
        Cells(2 + i, 7 + j) = rvArray(i, j)
    
```

```

Next j
Next i
End Sub

```

Example 3

Assemble the upper triangular array on the left side into the arrangement shown on the right side, as below.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2	INPUT							OUTPUT						
3	1	7	13	19	25	31		1	7	13	19	25	31	
4	2	8	14	20	26	32		8	14	20	26	32		
5	3	9	15	21	27	33		15	21	27	33			
6	4	10	16	22	28	34		22	28	34				
7	5	11	17	23	29	35		29	35					
8	6	12	18	24	30	36		36						
9														
10														

This example is made as an exercise for readers to make their own code. As previous examples, the data manipulation in this example is done and completed within two looping. Use always the **Debug** menu to verify the program steps. Good luck and hopefully these examples are helpful to develop your Excel-VBA skill.

Note:

An algorithm or in the (Author's) other words is a manipulation on program steps (flow) is basic procedure to tell computer how the program will be executed step by step. Although it may formally a guideline to be followed but everything is gained from experiences and also exercises. So, the coding more depends on instinct or interpretation of an author on the problems faced. However, it was still agreed that the fewer steps needed to solve a problem, the better the program built, meaning the faster execution time by the computer. Thus, it is not because of the fewer program lines created.

A Note for Beginners:

If you are a beginner in program algorithm, it is fine because your lesson has just started here. Run the program code of Example 1 and 2 **step by step** using the **Debug** menu (for example by pressing **F8**) and be focused on the code in the red lines. In **break mode**, see how the algorithm for both examples and try to understand the idea behind those code writing. After you have understood, try to solve Example 3 or any problem you define by yourself, for example that are developed from Example 1 and 2.

CHAPTER 4

MATRIX PROGRAM

The calculation using matrix operation is very suitable and easy to be performed by a computer. Thus, in line with rapidly growth of computer usage, matrix method is becoming very popular and widely used instead of the analysis with manual method. It is widely used in mathematical sciences or in civil engineering field.

4.1 MATRIX DEFINITION

A matrix is a rectangular array which involves a series of numbers of constituent components or elements.

Example:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & & & \\ a_{31} & & & \\ - & & & \\ - & & & \\ a_{m1} & a_{m2} & & a_{mn} \end{bmatrix}$$

The notation for a matrix is usually used $[]$ and $()$ or $\{\}$ for row or column matrix. A matrix can be written $[A] = [a_{ij}]$, where a_{ij} is element of the matrix, $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$. Subscript m indicates the number of rows and n for the number of column, or called has order $(m \times n)$. If $m = n$, then called a square matrix.

4.1.1 TYPES OF MATRIX

1. Row matrix or row vector, where $m = 1$.

Example: $\{1 \ 2 \ 3 \ 4\}$

2. Column matrix or column vector, where $n = 1$.

Example: $\begin{Bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{Bmatrix}$

3. Square Matrix

Diagonal matrix

All elements = 0, except in the main diagonal.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

Upper Triangular Matrix

All elements under the main diagonal = 0.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \dots & a_{1n} \\ 0 & a_{22} & a_{23} & a_{24} & \dots & a_{2n} \\ 0 & 0 & a_{33} & a_{34} & \dots & a_{3n} \\ 0 & 0 & 0 & a_{44} & \dots & a_{4n} \\ - & - & - & - & \dots & - \\ 0 & 0 & 0 & 0 & \dots & a_{nn} \end{bmatrix}$$

Lower Triangular Matrix

All elements above the main diagonal = 0.

$$\begin{bmatrix} a_{11} & 0 & 0 & 0 & \dots & 0 \\ a_{21} & a_{22} & 0 & 0 & \dots & 0 \\ a_{31} & a_{32} & a_{33} & 0 & \dots & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} & \dots & 0 \\ - & - & - & - & \dots & 0 \\ a_{m1} & a_{m2} & a_{m3} & a_{m4} & \dots & a_{mn} \end{bmatrix}$$

Scalar matrix

Scalar matrix is a diagonal matrix where the diagonal elements are the same number.

$$\begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

Unit matrix

Unit matrix is a scalar matrix in which the value of elements = 1. This matrix is also called identity matrix and generally denoted by [I]. Multiplying by [I] will return the origin matrix.

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

Band Matrix

Band matrix is a square matrix (n x n) where the non-zero elements are grouped and form a diagonal elements band.

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & \dots & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & a_{33} & a_{34} & \dots & 0 & 0 \\ 0 & 0 & a_{43} & a_{44} & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & 0 & 0 & \dots & a_{n,n-1} & a_{n,n} \end{bmatrix}$$

Symmetric matrix

A matrix [A] is announced symmetric if,

$$[A]^T = [A] \text{ or } a_{ij} = a_{ji}$$

Example:

$$\begin{bmatrix} 1 & 4 & -5 \\ 4 & 2 & 6 \\ -5 & 6 & 3 \end{bmatrix}$$

4.1.2 MATRIX OPERATION

Matrix Addition and Subtraction

If [A] and [B] are two matrices that have equal dimension, then they can be added.

If [A] = [a_{ij}] and [B] = [b_{ij}] returns [C] = [C_{ij}], it is written:

$$[C] = [A] + [B]$$

$$c_{ij} = a_{ij} + b_{ij} \text{ for each } i \text{ dan } j.$$

Example

$$[A] = \begin{bmatrix} 2 & 4 & 3 \\ 1 & 5 & 4 \end{bmatrix}, [B] = \begin{bmatrix} 0 & -3 & 2 \\ 4 & 2 & -3 \end{bmatrix}$$

$$[A] + [B] = \begin{bmatrix} 2+0 & 4-3 & 3+2 \\ 1+4 & 5+2 & 4-3 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 5 \\ 5 & 7 & 1 \end{bmatrix}$$

Subtract [B] from [A] is equal to add [A] with [-B] or written: [A] - [B] = [A] + [-B].

Matrix Multiplication

Multiplication of two matrices, [A] (m x n) by [B] (n x p) will return [C] of order (m x p). When do the matrix multiplication the number of columns [A] must be equal to the number of rows [B].

If [A] = [a_{ij}], [B] = [b_{ij}] and [C] = [C_{ij}] then,

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \dots + a_{im}b_{mj}$$

or written as,

$$c_{ij} = \sum_{k=1}^m a_{ik}b_{kj}$$

Example 1

$$[A] = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}, [B] = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$\begin{aligned} [C] = [A] \times [B] &= \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \\ &= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \\ a_{31}b_{11} + a_{32}b_{21} & a_{31}b_{12} + a_{32}b_{22} \end{bmatrix} \end{aligned}$$

Example 2

$$[A] = \begin{bmatrix} 2 & 3 \\ 4 & 1 \\ 7 & 4 \end{bmatrix}, [B] = \begin{bmatrix} 3 & 3 \\ -1 & 5 \end{bmatrix}$$

$$\begin{aligned} [C] = [A] \times [B] &= \begin{bmatrix} 2 & 3 \\ 4 & 1 \\ 7 & 4 \end{bmatrix} \begin{bmatrix} 3 & 3 \\ -1 & 5 \end{bmatrix} \\ &= \begin{bmatrix} 2.3 + 3.(-1) & 2.3 + 3.5 \\ 4.3 + 1.(-1) & 4.3 + 1.5 \\ 7.3 + 4.(-1) & 7.3 + 4.5 \end{bmatrix} \\ &= \begin{bmatrix} 3 & 21 \\ 11 & 17 \\ 17 & 41 \end{bmatrix} \end{aligned}$$

Matrix Inverse

Division is not defined in a matrix operation, for it used the inverse of a matrix. Matrix inversion is only done on square matrix and written with the notation $[]^{-1}$. Multiplication by matrix inverse is a division by the matrix. For example: $[A] \times [B] = [C]$ where $[A]$ is a square matrix, the equation can be written into a division operation: $[B] = [A]^{-1} [C]$. $[A]^{-1}$ is called the inverse of $[A]$.

Matrix Transpose

If $[A]$ is a matrix of order $(m \times n)$, then the transpose of $[A]$ is a matrix of order $(n \times m)$ or written as $[A]^T$. Rows and columns of $[A]$ become the columns and rows of $[A]^T$.

$$[B] = [A]^T$$

$$b_{ij} = a_{ji}$$

Example

$$[A] = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$$[A]^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

Some properties associated with matrix transpose:

$$([A]^T)^T = [A]$$

$$([A] + [B])^T = [A]^T + [B]^T$$

$$([A][B])^T = [B]^T [A]^T$$

System of Linear Equations

Many problems in civil engineering are represented by systems of "n" linear equations with unknown number "n" and this can be solved using matrix methods. It is often, the number of equations equal to number of unknowns.

Example: two systems of equations

$$3x + 2y = 9$$

$$5x - y = 2$$

When expressed in matrix form it becomes:

$$\begin{bmatrix} 3 & 2 \\ 5 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 9 \\ 2 \end{bmatrix}$$

In general form, "n" linear equations with "n" unknown number can be expressed as:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix} \quad (4.1)$$

and simplified into:

$$[A] \cdot \{X\} = \{B\} \quad (4.2)$$

where,

[A] square matrix represents coefficients of Equation 4.2

{X} unknowns column vector

{B} constants column vector

Simply to say that the solution of systems of "n" linear equations in matrix form as expressed in Equations 4.2 is finding {X}, where [A] and [B] are known values. There are many methods used out there to solve Equation 4.2, some will briefly be discussed in examples below.

1. Elimination – Back Substitution

Example

$$2x + 3y = 7 \quad \dots(1)$$

$$3x - 2y = 4 \quad \dots(2)$$

This is an example of two linear equations with 2 unknowns, x and y.

Elimination

Eliminate x from both equations by subtracting Equation 2 from Equation 1 that has been given a multiplying factor (Eq. B1):

$$2x + 3y = 7 \dots (1) \times \frac{3}{2} \quad \rightarrow \quad 3x + \frac{9}{2}y = \frac{21}{2} \quad \dots(B1)$$

$$3x - 2y = 4 \quad \dots(2)$$

$$\begin{array}{r} 3x + \frac{9}{2}y = \frac{21}{2} \\ \underline{3x - 2y = 4} \quad - \\ \frac{13}{2}y = \frac{13}{2} \end{array}$$

$$y = 1$$

Back Substitution

Substituting y value back into Equation 1, then we have x = 2.

$$\text{Thus, solution for the unknowns } \{X\}: \begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} 2 \\ 1 \end{Bmatrix}$$

2. Directly get the inverse of [A] or [A]⁻¹

From Equation 4.2:

$$[A]\{X\} = \{B\}$$

Multiply both matrices by [A]⁻¹:

$$[A]^{-1}[A]\{X\} = [A]^{-1}\{B\}, \text{ it is known } [A]^{-1}[A] = [I]$$

$$[I]\{X\} = [A]^{-1}\{B\}$$

$$\{X\} = [A]^{-1}\{B\} \quad (4.3)$$

Thus, if [A]⁻¹ is known, Equation 4.3 can be solved.

Example

Given two a system of two linear equations:

$$2x + 3y = 7$$

$$3x - 2y = 4$$

In this example the matrix inverse will be found using by Gauss-Jordan method. The solution is to put the unit matrix [I] next to coefficient matrix [A] such that when the process is finished [A] become [I] and the previous [I] become matrix inverse [A]⁻¹. The overall process can be just expressed as:

$$[A : I] \rightarrow [I : [A]^{-1}]$$

The process is done through elementary row operation on [A] and [I] as well to reduce [A] to [I]. Thus, it is a sequence of row operation to turn elements of [A] = 1 on the main diagonal and zero elsewhere. For detail see the following example:

Derive matrix coefficient:

$$[A] = \begin{bmatrix} 2 & 3 \\ 3 & -2 \end{bmatrix}$$

Put [I] next to [A]:

$$\begin{bmatrix} 2 & 3 \\ 3 & -2 \end{bmatrix} \left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array} \right]$$

Step 1: From above matrix, divide row 1 by 2

$$\begin{bmatrix} 1 & \frac{3}{2} \\ 3 & -2 \end{bmatrix} \left[\begin{array}{cc} \frac{1}{2} & 0 \\ 0 & 1 \end{array} \right]$$

Step 2: Add row 2 to row 1 times -3

$$\begin{bmatrix} 1 & \frac{6}{2} \\ 0 & -\frac{13}{2} \end{bmatrix} \left[\begin{array}{cc} \frac{1}{2} & 0 \\ -\frac{3}{2} & 1 \end{array} \right]$$

Step 3: Multiply row 2 by $-\frac{2}{13}$

$$\begin{bmatrix} 1 & \frac{3}{2} \\ 0 & 1 \end{bmatrix} \left[\begin{array}{cc} \frac{1}{2} & 0 \\ \frac{6}{26} & -\frac{2}{13} \end{array} \right]$$

Step 4: Add row 1 to row 2 times $-\frac{3}{2}$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \left[\begin{array}{cc} \frac{8}{52} & \frac{6}{26} \\ \frac{6}{26} & -\frac{2}{13} \end{array} \right]$$

Thus,

$$[A]^{-1} = \begin{bmatrix} \frac{8}{52} & \frac{6}{26} \\ \frac{6}{26} & -\frac{2}{13} \end{bmatrix}$$

Enter $[A]^{-1}$ into Equation 4.3

$$\{X\} = [A]^{-1} \cdot \{B\}$$

$$\begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{bmatrix} \frac{8}{52} & \frac{6}{26} \\ \frac{6}{26} & -\frac{2}{13} \end{bmatrix} \begin{Bmatrix} 7 \\ 4 \end{Bmatrix}$$

$$\begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} \frac{104}{52} \\ \frac{26}{26} \end{Bmatrix}$$

So, the solution to this system,

$$\begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} 2 \\ 1 \end{Bmatrix}$$

3. Iterative Method

This method is actually an approach and done by taking initial value of the unknown then corrected or refined to enter the next value. We have examined the iterative method on the example above in the previous discussion in Chapter 3. Here is now an example from a system of three linear equations:

$$5x + 4y + 3z = 12 \quad (4.4)$$

$$4x + 7y + 4z = 15 \quad (4.5)$$

$$3x + 4y + 4z = 11 \quad (4.6)$$

The iterative process is quite long, so in this example we will be working with help of a computer through Excel iteration.

Re-write Equation 4.4, 4.5 and 4.6 into forms below:

Equation 4.4: $x = 12/5 - 4y/5 - 3z/5 = (12 - 4y - 3z)/5$

Equation 4.5: $y = 15/7 - 4x/7 - 4z/7 = (15 - 4x - 4z)/7$

Equation 4.6: $z = 11/4 - 3x/4 - 4y/4 = (11 - 3x - 4y)/4$

Next, convert the equations into worksheet formulas and enter in cell B2, C3 and D4. However, until this stage no iteration to be done because the formulas are still standing on their own equations.

	A	B	C	D	E	F
1		Eq.1	Eq.2	Eq.3		
2	x	$= (12 - 4*B3 - 3*B4)/5$				
3	y		$= (15 - 4*C2 - 4*C4)/7$			
4	z			$= (11 - 3*D2 - 4*D3)/4$		
5						
6						

The iterative process will be carried out after these connections:

The value of y, z of Equation 4.4 = value y, z of Equation 4.5

The value of x, z of Equation 4.5 = value x of Equation 4.4 and z of Equation 4.6

The value of x, y of Equation 4.6 = value x, y of Equation 4.5

In the worksheet, the equations are done as follows:

	A	B	C	D	E
1		Eq.1	Eq.2	Eq.3	
2	x	$= (12 - 4*B3 - 3*B4)/5$	=B2	=C2	
3	y	=C3	$= (15 - 4*C2 - 4*C4)/7$	=C3	
4	z	=C4	=D4	$= (11 - 3*D2 - 4*D3)/4$	
5					
6					

The iteration steps shown in the worksheet below:

1 x iteration:

	A	B	C	D	E
1		Eq.1	Eq.2	Eq.3	
2	x	2.40000	2.40000	2.40000	
3	y	0.00000	0.77143	0.77143	
4	z	0.00000	0.00000	0.17857	
5					

5 x iterations:

	A	B	C	D	E
1		Eq.1	Eq.2	Eq.3	
2	x	1.29890	1.29890	1.29890	
3	y	1.07655	1.16637	1.16637	
4	z	0.40995	0.56703	0.60945	
5					

10 x iterations:

	A	B	C	D	E
1		Eq.1	Eq.2	Eq.3	
2	x	1.01854	1.01854	1.01854	
3	y	1.03905	1.02231	1.02231	
4	z	0.94242	0.95086	0.96379	
5					

20 x iterations:

	A	B	C	D	E
1		Eq.1	Eq.2	Eq.3	
2	x	1.00597	1.00597	1.00597	
3	y	1.01594	1.00960	1.00960	
4	z	0.97723	0.98348	0.98592	
5					

50 x iterations:

	A	B	C	D	E
1		Eq.1	Eq.2	Eq.3	
2	x	1.00046	1.00046	1.00046	
3	y	1.00172	1.00097	1.00097	
4	z	0.99784	0.99848	0.99868	
5					

The iteration process of finding {X} of Equation 4.4 is actually to make x, y, z to the same value on the worksheet columns B, C and D in satisfying the given equations. It appears that they are very close at 50th iteration and by rounding obtained:

$$\begin{Bmatrix} x \\ y \\ z \end{Bmatrix} = \begin{Bmatrix} 1 \\ 1 \\ 1 \end{Bmatrix}$$

4.2 PROGRAM FOR MATRIX OPERATIONS

Multiplication and inversion of matrix are two main operations in matrix equation found in most of engineering calculations using matrix method. In this discussion, it will demonstrate how to create program code for multiplication and inversion of matrix.

Matrix Multiplication

In the following example is given a program to calculate the multiplication of two matrices. The program is named MMULT1, input data and the results will be done on Sheet1.

Example

Find the multiplication of two matrices in the following equation:

$$[C] = [A], [B]$$

where,

$$[A] = \begin{bmatrix} 2 & 3 \\ 4 & 1 \\ 7 & 4 \end{bmatrix}, [B] = \begin{bmatrix} 3 & 3 \\ -1 & 5 \end{bmatrix}$$

We know that the multiplication of a matrix is the multiplication of element row in one matrix by element column of other matrix. If done manually as in the example of Section 4.1.2, code to get [C] is:

```
Sub MMULT1()  
Dim MA(3, 2) As Double  
Dim MB(2, 2) As Double  
Dim MC(1 To 3, 1 To 2) As Double  
  
'=====br/>'Read Input Data  
'=====br/>'[A] or MA  
MA(1, 1) = Cells(6, 2)  
MA(2, 1) = Cells(7, 2)  
MA(3, 1) = Cells(8, 2)  
MA(1, 2) = Cells(6, 3)  
MA(2, 2) = Cells(7, 3)  
MA(3, 2) = Cells(8, 3)
```

```

'[B] or MB
MB(1, 1) = Cells(6, 5)
MB(2, 1) = Cells(7, 5)
MB(1, 2) = Cells(6, 6)
MB(2, 2) = Cells(7, 6)

'=====
'Multiplying
'[C] = [A].[B]
'=====

MC(1, 1) = MA(1, 1) * MB(1, 1) + MA(1, 2) * MB(2, 1)
MC(2, 1) = MA(2, 1) * MB(1, 1) + MA(2, 2) * MB(2, 1)
MC(3, 1) = MA(3, 1) * MB(1, 1) + MA(3, 2) * MB(2, 1)
MC(1, 2) = MA(1, 1) * MB(1, 2) + MA(1, 2) * MB(2, 2)
MC(2, 2) = MA(2, 1) * MB(1, 2) + MA(2, 2) * MB(2, 2)
MC(3, 2) = MA(3, 1) * MB(1, 2) + MA(3, 2) * MB(2, 2)

'=====
'Print [C]:
'=====

Range(Cells(12, 2), Cells(14, 3)).FormulaArray = MC
End Sub

```

Worksheet form that fits to the above program code is as below:

	A	B	C	D	E	F	G
1	MULTIPLY TWO MATRICES 1						
2	[A]= 3x2						
3	[B]= 2x2						
4	[A][B]= 3x2						
5							
6	[A]=	2	3	[B]=	3	3	
7		4	1		-1	5	
8		7	4				
9							
10	[C]=[A][B]						
11							
12	[C]=						
13							
14							
15							

'=====

'Read Input Data

'=====

'[A] or MA

MA(1, 1) = Cells(6, 2)

MA(2, 1) = Cells(7, 2)

MA(3, 1) = Cells(8, 2)

MA(1, 2) = Cells(6, 3)

MA(2, 2) = Cells(7, 3)

MA(3, 2) = Cells(8, 3)

'[B] or MB

MB(1, 1) = Cells(6, 5)

MB(2, 1) = Cells(7, 5)

MB(1, 2) = Cells(6, 6)

MB(2, 2) = Cells(7, 6)

In this example, [A] has order 3 x 2 and [B] has order 2 x 2 to return [C] of order 3 x 2. Input data of elemen [A] is made on a range of cells (6,2) to cell (8,3), while element [B] on a cell range (6.5) to cell (7.6). To run the program, click the command button that has been made

in the worksheet. The results are printed on a range of cells (12, 2) to cell (14, 3), using statement **FormulaArray** = [C], that shown below:

	A	B	C	D	E	F	G	H
1	MULTIPLY TWO MATRICES 1							
2	[A]= 3x2				MULTIPLY			
3	[B]= 2x2							
4	[A][B]= 3x2							
5								
6		[A]=	2	3		[B]=	3	3
7			4	1			-1	5
8			7	4				
9								
10	[C]=[A][B]							
11								
12		[C]=	3	21				
13			11	17				
14			17	41				
15								

```
'=====
'Multiplying
'[C] = [A].[B]
'=====
MC(1, 1) = MA(1, 1) * MB(1, 1) + MA(1, 2) * MB(2, 1)
MC(2, 1) = MA(2, 1) * MB(1, 1) + MA(2, 2) * MB(2, 1)
MC(3, 1) = MA(3, 1) * MB(1, 1) + MA(3, 2) * MB(2, 1)
MC(1, 2) = MA(1, 1) * MB(1, 2) + MA(1, 2) * MB(2, 2)
MC(2, 2) = MA(2, 1) * MB(1, 2) + MA(2, 2) * MB(2, 2)
MC(3, 2) = MA(3, 1) * MB(1, 2) + MA(3, 2) * MB(2, 2)
```

For a large-sized matrix, the given MMULT1 example is inefficient in term of wasting time and coding to write code for all matrix (row) operations one by one. Instead, use **For-Next** looping for reading input data, to process and print the results. This way saves a lot of code writing.

The code program of above matrix multiplication can be modified by giving the row and column indices (i, j and k) to [A], [B] and [C], which is done in three looping below:

Row
[A]

Col
[B]

Row
[A]

Col
[B]

```

'=====
'Multiplying
'[C] = [A].[B]
'=====
MC(1, 1) = MA(1, 1) * MB(1, 1) + MA(1, 2) * MB(2, 1)
MC(2, 1) = MA(2, 1) * MB(1, 1) + MA(2, 2) * MB(2, 1)
MC(3, 1) = MA(3, 1) * MB(1, 1) + MA(3, 2) * MB(2, 1)
MC(1, 2) = MA(1, 1) * MB(1, 2) + MA(1, 2) * MB(2, 2)
MC(2, 2) = MA(2, 1) * MB(1, 2) + MA(2, 2) * MB(2, 2)
MC(3, 2) = MA(3, 1) * MB(1, 2) + MA(3, 2) * MB(2, 2)

```

↓

```

'Multiplying
'[A] (mxn) x [B] (nxp) returns
'[C] (mxp)

For i = 1 To m
  For j = 1 To p
    MC(i, j) = 0
    For k = 1 To n
      MC(i, j) = MC(i, j) + MA(i, k) * MB(k, j)
    Next k
  Next j
Next i

```

In a looping above, m and n represents number of rows and columns of [A], respectively, while n and p represents number of rows and columns of [B]. In the following example, we will use the modified program code to multiply a 6x6 matrix by a 6x3 matrix. The program is named MMULT2.

```

Sub MMULT2()
Dim m, n, p As Integer
m = Cells(2, 5)
n = Cells(3, 5)
p = Cells(4, 5)
ReDim MA(1 To m, 1 To n) As Double
ReDim MB(1 To n, 1 To p) As Double
ReDim MC(1 To m, 1 To p) As Double

'Input data [A]
For i = 1 To m
    For j = 1 To n
        MA(i, j) = Cells(5 + i, 1 + j)
    Next j
Next i

'Input data [B]
For i = 1 To n
    For j = 1 To p
        MB(i, j) = Cells(5 + i, 9 + j)
    Next j
Next i

'Multiplying
'[A](mxn) x [B](n xp) returns
'[C](mxp)

For i = 1 To m
    For j = 1 To p
        MC(i, j) = 0
        For k = 1 To n
            MC(i, j) = MC(i, j) + MA(i, k) * MB(k, j)
        Next k
    Next j
Next i

'Print result

```

```

For i = 1 To m
  For j = 1 To p
    Cells(15 + i, 1 + j) = MC(i, j)
  Next j
Next i
End Sub

```

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2	[A] =	mxn		m =	6							
3	[B] =	nxp		n =	6							
4	[C] =	mxp		p =	3							
5												
6	[A] =	20	5	0	0	0	0		[B] =	0.25	0	0
7		5	20	0	0	0	0			0.25	1	0
8		0	0	30	10	0	0			-0.50	1	0
9		0	0	10	30	0	0			-0.50	0	1
10		0	0	0	0	12	6			0.25	0	1
11		0	0	0	0	6	12			0.25	0	0
12												
13												
14	[C] =[A].[B]											
15												
16	[C] =	6.25	5	0								
17		6.25	20	0								
18		-20	30	10								
19		-20	10	30								
20		4.5	0	12								
21		4.5	0	6								
22												

In this program, the printing of the results that used Array Formula has been replaced by two orders of For-Next looping. The first and second looping is respectively for printing rows and columns of [C]. This way is much better in term of automation, because we do not need to calculate the range of the array that will be generated as it has been known = m x p.

Matrix multiplication can also be done in a worksheet using Excel built-in function **MMULT** (array1, array2). For MMULT2 procedure example, a formula can be written as follows:

= MMULT(B6:G11,J6:L11).

To print the result use **INDEX** function or you can enter the formula above as an array formula. See about array formula in Chapter 1.5.

Matrix Inversion

In finding an inverse of matrix, Gauss-Jordan method is among the most popular method to be used and programmed, with a process that has been described in the previous section. The steps are an effort to turn a matrix to be searched its inverse (e.g. [A]) becomes an identity matrix [I] and at the same time, [I] to [A]⁻¹. That is done by elementary row operation as ever shown in the previous example.

In the program, this can be done by taking one matrix e.g. [C], such that at the end of the process [C] turns into [A]⁻¹. In the beginning is to divide the first row by element **a₁₁** of [A], in an effort to turn a₁₁ = 1 so that c₁₁ = 1 / a₁₁ and this is analogous to that performed in [I]. If [A] is n x n matrix, then in the **first step** the first column of [C] is analogous of that performed in [I], while the second column to column n is analogous of that performed in [A]. At the end of step 1, the value of [C] of order 3x3 can be written as follows:

$$[C] = \begin{bmatrix} \frac{1}{a_{11}} & \frac{a_{12}}{a_{11}} & \frac{a_{13}}{a_{11}} \\ -\frac{a_{21}}{a_{11}} & a_{22} + \frac{a_{12}}{a_{11}} \cdot -a_{21} & a_{23} + \frac{a_{13}}{a_{11}} \cdot -a_{21} \\ -\frac{a_{31}}{a_{11}} & a_{32} + \frac{a_{12}}{a_{11}} \cdot -a_{31} & a_{33} + \frac{a_{13}}{a_{11}} \cdot -a_{31} \end{bmatrix}$$

In the **second step**, the first and second column of [C] is analogous of that performed in [I], while the third column to column n is analogous of that performed in [A], etc.

Program for finding inverse is created in VBA and provided here. It notes that matrix inversion requires square size (say, m x m) matrix and the determinant of matrix must not be zero. For the purpose to create worksheet form, the maximum element of matrix is limited to m = 6, but it can certainly be made more than 6.

Example 1

Find the inverse of $[A] = [5]$.

Solution for Example 1:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	INVERSE MATRIX															
2																
3	m =	1	Matrix size m x m = 1 x 1													
4																
5		5	Input data													
6																
7																
8																
9																
10																
11																

Example 2

Find the invers of,

$$[A] = \begin{bmatrix} 2 & 3 \\ 3 & -2 \end{bmatrix}$$

Solution for Example 2:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	INVERSE MATRIX															
2																
3	m =	2	Matrix size m x m = 2 x 2													
4																
5		2	3													
6		3	-2													
7																
8																
9																
10																
11																

Example 3

Find the invers of,

$$[A] = \begin{bmatrix} 1 & 3 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 4 \end{bmatrix}$$

Solution for Example 3:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	INVERSE MATRIX															
2																
3	m =	3														
4																
5		1	3	3												
6		1	4	3												
7		1	3	4												
8																
9																
10																
11																

To find inverse of matrix in worksheet can also be used the built-in function **MINVERSE**. The way to use it is the same as **MMULT**.

VBA code:

```

Sub Inverse()
On Error Resume Next
m = Cells(3, 2)
ReDim MA(1 To m, 1 To m) As Double
Range("k5:p10").ClearContents

'Read input data
For i = 1 To m
    For j = 1 To m
        MA(i, j) = Cells(4 + i, j)
    Next j
Next i

'Find inverse []
For i = 1 To m
    For j = 1 To m
        If j <> i Then MA(i, j) = MA(i, j) / MA(i, i)
    Next j
    For n = 1 To m
        If n = i Then GoTo 10
        For j = 1 To m
            If j <> i Then MA(n, j) = MA(n, j) - MA(i, j) * MA(n, i)
        Next j
10    Next n
    For n = 1 To m
        If n <> i Then MA(n, i) = -MA(n, i) / MA(i, i)
    Next n

```

```

MA(i, i) = 1 / MA(i, i)
For ii = 1 To m
    For ij = 1 To m
        Cells(4 + ii, 10 + ij) = MA(ii, ij)
    Next ij
Next ii
Next i

'Print output data
For i = 1 To m
    For j = 1 To m
        Cells(4 + i, 10 + j) = MA(i, j)
    Next j
Next i

End Sub

```

4.3 MATRIX METHOD FOR STRUCTURAL ANALYSIS

Structure can be analyzed as a series of structural elements which are connected to each other at endpoint called node. Solution of this system of arrangement of elements can be expressed by simultaneous linear equations that are made in a matrix form. In matrix form, then the calculation of the structure becomes easier to be solved with the help of a computer.

4.3.1 UPPER STRUCTURE

The method that will be used for structural analysis is **direct stiffness method**. A brief introduction to the basics of this method will be given here; readers are encouraged to read other references to supplement the discussion on this topic. In short, the aim using of this method of analysis is to obtain **force - displacement relationship** expressed as:

$$\{P\} = [K]\{X\} \quad (4.7)$$

where, $\{P\}$ vector of external force acting on node
 $[K]$ structure stiffness matrix
 $\{X\}$ vector of nodal displacements

By deriving [K], thus it can be obtained force - displacement relationship as expressed by Equation 4.7. One of the processes here is to derive stiffness matrix of an element, then applied for other elements and put together to form stiffness matrix of the whole structure. To meet deformation continuity and compatibility between the elements is by superposing corresponding force - displacement components.

Furthermore, to simplify the calculation we can partition element of the stiffness matrix at the nodes that are free to displace (free nodes) and fixed (zero displacement at the supports). If X_b represents displacements vector at the support, X_f represents displacements vector at free nodes, P_f and P_b are the external forces vector which correspond to their displacements, Equation 4.7 can be written as follows:

$$\begin{Bmatrix} P_f \\ P_b \end{Bmatrix} = \begin{bmatrix} K_{ff} & K_{fb} \\ K_{bf} & K_{bb} \end{bmatrix} \begin{Bmatrix} X_f \\ X_b \end{Bmatrix} \quad (4.8)$$

If node displacements vector $\{X_b\} = 0$, then Equation 4.8 can be written:

$$\{P_f\} = [K_{ff}]\{X_f\} \quad (4.9)$$

And

$$\{P_b\} = [K_{bf}]\{X_f\} \quad (4.10)$$

The displacements vector at free nodes is then obtained using Equation 4.9 that becomes $\{X_f\} = [K_{ff}]^{-1}\{P_f\}$. If $\{X_f\}$ is known then the internal forces of the element can be obtained. In this relation, the internal force of element i is expressed in local coordinate and obtained by the following equation:

$$\{P\}_i = \{P_o\} + [K]_i\{X\}_i \quad (4.11)$$

where,

$\{P_o\}$ fixed-end forces: reactions at end of element i due to external load directly acting on element i

$[K]_i$ stiffness matrix of element i in local coordinate system

$\{X\}_i$ deformations matrix = $[T]_i\{X_f\}$

$[T]_i$ transformation matrix of element i

To find the support reactions, it can be directly obtained in the global coordinate of the structure using Equation 4.10 becomes:

$$\{R_b\} + \{P_b\} = [K_{bf}]\{X_f\}$$

or,

$$\{R_b\} = [K_{bf}]\{X_f\} - \{P_b\} \quad (4.12)$$

4.3.2 SUB STRUCTURE

The structure analysis in this subject is to obtain a force - displacement relationship of soil - structure interaction which is modeled by the means of **beam on elastic foundation**. Generally it is solved with the finite element method where the system is built by a series of finite elements connected at endpoint called node.

Some equations are made in order to find force - displacement relationship at the nodes, and begins by writing the relationship between external forces at the nodes $\{P\}$ and internal forces of element $\{F\}$ in principle of equilibrium.

The relationship between $\{P\}$ and $\{F\}$ is expressed as:

$$\{P\} = [A]\{F\} \quad (4.13)$$

The relationship between node displacements $\{X\}$ and deformations of element $\{d\}$:

$\{d\} = [B]\{X\}$, and from reciprocal theorem it can be shown

$$\{d\} = [A]^T\{X\} \quad (4.14)$$

From elastic behavior of element, the relation between $\{F\}$ and $\{d\}$:

$$\{F\} = [S]\{d\} \quad (4.15)$$

Substituting Equation 4.14 into 4.15 to get,

$$\{F\} = [S]\{d\} = [S][A]^T\{X\} \quad (4.16)$$

Substituting Equation 4.16 into 4.13 to get.

$$\{P\} = [A]\{F\} = [A][S][A]^T\{X\} \quad (4.17)$$

$\{X\}$ is unknown displacements to be determined, thus

$$\{X\} = ([A][S][A]^T)^{-1}\{P\} \quad (4.18)$$

Substituting $\{X\}$ back to Equation 4.16 to find internal forces $\{F\}$.

Equations 4.16, 4.17 and 4.18 are the fundamental equations in the analysis using finite element method (J.E. Bowles, 1982).

CHAPTER 5

NUMERICAL METHOD

Many problems in science and technology involve the use of integral and differential equations. If the mathematical equation model is quite simple, then it can be solved by means of analytical method. However, many of those equations are difficult or may be impossible to be solved analytically. Instead, it must be solved numerically or using numerical methods.

The result of the numerical method is an approach to its exact value by means of analytically. Thus, it is just an approximate calculation with estimated accuracy. Numerical methods will provide many benefits if done by a computer.

5.1 NUMERICAL INTEGRATION

The integral of a function $f(x)$ from a to b is written:

$$y = \int_a^b f(x) dx$$

has a broad sense as the area bounded by the curve $y = f(x)$ and the X-axis, the between $x = a$ and $x = b$.

Analytical solution to integral or exact solution is:

$$y = \int_a^b f(x) dx = \left[F(x) \right]_a^b = F(a) - F(b) \quad (5.1)$$

where, $F(x)$ is an anti-derivative of $f(x)$ such that $F'(x) = f(x)$.

Numerical integration to calculate area under the curve uses the concept of approach whereby the area is divided into smaller pieces of area in such that, if combined the total area of the pieces approaching the exact result. In this example we will use trapezoidal method where the area under the curve is divided into small pieces of area with sides $f(x_n)$ and $f(x_{n-1})$ that resembles a trapezoid as shown in Figure 5.1.

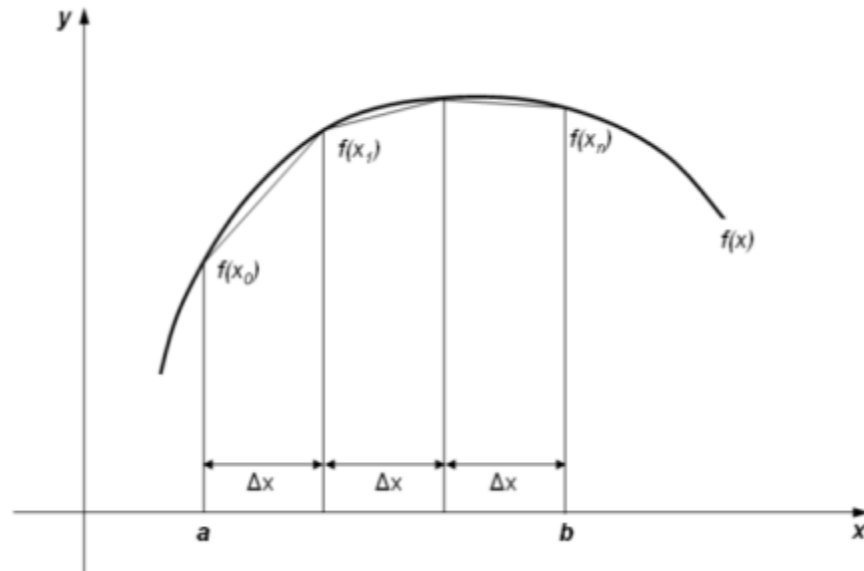


Figure 5.1: Trapezoidal method for approximating area

Trapezoids are having the same width for Δx , if there are n intervals (or n trapezoids) from a to b , then the $\Delta x = (b - a) / n$ as shown Figure 5.1. Figure 5.2 shows the comparison between the area obtained from the total area of the trapezoids and the exact results obtained using Equation 5.1. The formula to find total area (L) is given by:

$$L_{total} = \frac{\Delta x}{2} ((f(x_0) + f(x_1)) + (f(x_1) + f(x_2)) + \dots + (f(x_{n-1}) + f(x_n))) \quad (5.2)$$

$$= \frac{\Delta x}{2} (f(x_0) + 2f(x_1) + 2f(x_2) + 2f(x_3) + \dots + 2f(x_{n-1}) + f(x_n)) \quad (5.3)$$

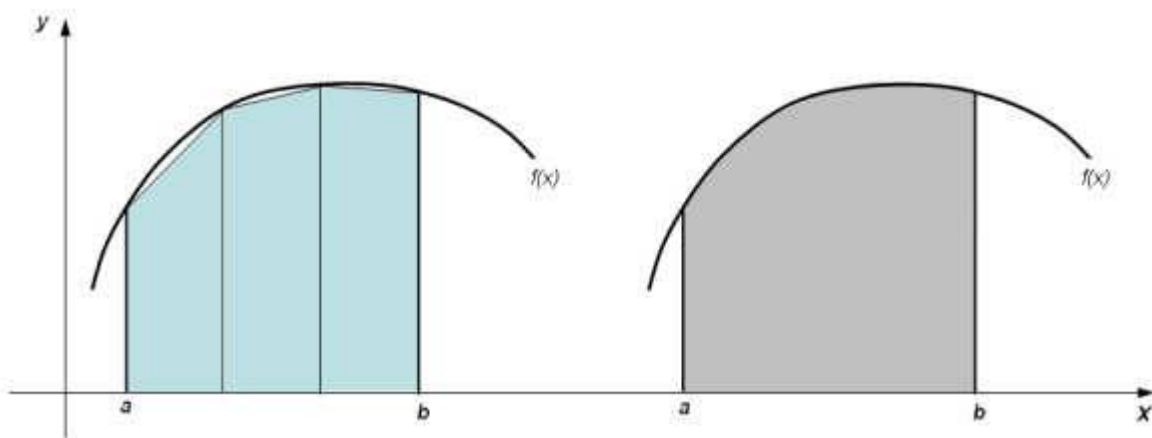


Figure 5.2: The comparison of area resulted from trapezoids (left) and the exact one (right)

Figure 5.1 and 5.2 show, the calculation by means of trapezoidal method is getting closer to the exact result when taking more trapezoids into account, or by making smaller Δx .

Furthermore, to complement this discussion we will examine the given example below via VBA.

Example

Find the area bounded by the parabola $y = 4x - x^2$, the X-axis, from $x = a$ and $x = b$. Find also the exact value.

Solution

VBA code

```
Function Trapez(a, b, n)
a = Cells(3, 2)
b = Cells(4, 2)
n = Cells(5, 2)

Dx = (b - a) / n
x = a
    For i = 2 To n - 1
        x = x + Dx
        Sum = Sum + 2 * f(x)
    Next i

Sum = Sum + f(a) + f(b)
Trapez = Sum * Dx / 2
End Function

Function f(x)
    f = 4 * x - x ^ 2
End Function

Function integf(x)
    integf = 2 * x ^ 2 - 1 / 3 * x ^ 3
End Function
```

In this experiment, we take $a = 0$ and $b = 4$ respectively, for $n = 10$ and 100 . The values of a , b and n are then used as function arguments of **Integf** and **Trapez** in the above program. The use of function in the worksheet and the results are shown below:

USING FUNCTION:

	A	B	C	D	E
1	Trapezoidal Method				
2					
3	a =	0			
4	b =	4			
5	n =	10			
6					
7	A num =	=trapez(B3,B4,B5)			
8	A exact =	=integf(B4)-integf(B3)			
9					

RESULT:

	A	B	C	D
1	Trapezoidal Method			
2				
3	a =	0		
4	b =	4		
5	n =	10		
6				
7	A num =	9.984		
8	A exact =	10.66667		
9				

	A	B	C	D
1	Trapezoidal Method			
2				
3	a =	0		
4	b =	4		
5	n =	100		
6				
7	A num =	10.65926		
8	A exact =	10.66667		
9				

In the worksheet, the area calculated using trapezoidal method and its exact value is respectively expressed as A_{num} and A_{exact} . This experiment shows that the smaller width of trapezoids taken with $n = 100$, the closer A_{num} to A_{exact} , compared with $n = 10$. However, the more thorough results requires a longer calculation process and it is certainly suitable done with the help of computer.

The other methods for approaching area, for examples are by the Simpson method and Gauss quadrature.

5.2 NUMERICAL DIFFERENTIATION

Differential equation is any equation that contains derivatives of function.

Examples:

$$F = m.a = m \frac{dv}{dt} = m \frac{d^2x}{dt^2} \quad (5.4)$$

$$\frac{d^2 v}{dx^2} = \frac{M}{EI} \quad (5.5)$$

$$\frac{\partial u}{\partial t} = c_v \frac{\partial^2 u}{\partial z^2} \quad (5.6)$$

Description:

- Equation 5.4 is Newton's second law of motion that states force (F) acting on an object = mass (m) of the object multiplied by its acceleration (a) where, acceleration (a) is the first derivative of velocity (v) with respect to time (t) and the second derivative of position (x).
- Equation 5.5 shows the relationships between bending moment (M) at any point on the X-axis and transverse deflection (v) of the beam. EI is flexural rigidity of the beam.
- Equation 5.6 is the partial differential equation of one-dimensional consolidation, where u, t, z are respectively, excess pore-water pressure, time and the depth of clay layer. Cv is a coefficient of consolidation where the value is assumed to be constant.

Numerical differentiation solution is an approach to continuous differential form into discrete or finite model. In this discussion we adopt Taylor approach for it is among the most popular and widely used in introducing numerical approximation. Taylor theorem for approximating a function is graphically shown in Figure 5.3.

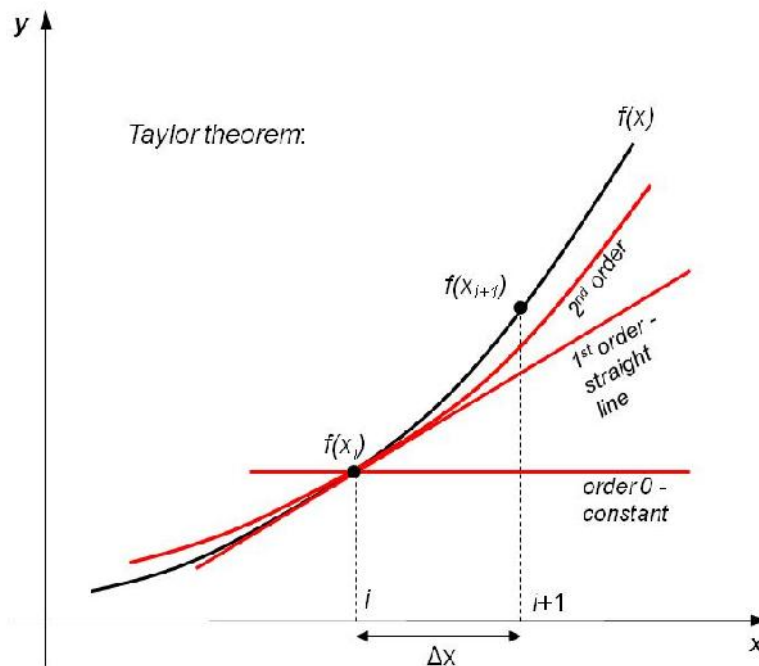


Figure 5.3: Taylor theorem for approximating a function

The value of the function at point x_{i+1} located at a distance Δx from point x_i can be approximated by Taylor series below:

$$f(x_{i+1}) = f(x_i) + f'(x_i) \frac{\Delta x}{1!} + f''(x_i) \frac{\Delta x^2}{2!} + f'''(x_i) \frac{\Delta x^3}{3!} + \dots + f^{(n)}(x_i) \frac{\Delta x^n}{n!} + R_n \quad (5.7)$$

where,

- All derivatives of function through the point $(x_i, f(x_i))$.
- R_n : truncation error when the series counted up to n -order.

Taylor series will give an estimation of the function correctly if all derivatives in the series are taken into account, where $n = \text{infinite number}$. However, in practice, the use of Taylor series is done by simply taking the first few terms of the series.

Equation 5.7 of the Taylor series can be derived into the discrete form (finite) as shown in Figure 5.4. The estimation of first derivative of a function $f'(x)$ is approached through several slope of the line through the value $f(x)$ at $x = x_{i-1}$ (point A), $x = x_i$ (point B) and $x = x_{i+1}$ (point C) as in Figure 5.4. From Equation 5.7, with respect to first derivative:

$$f(x_{i+1}) = f(x_i) + f'(x_i) \Delta x + R_n$$

or,

$$\frac{\partial f}{\partial x} = f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{\Delta x} - R_n \quad (5.8)$$

Equation 5.8 is called forward difference formula.

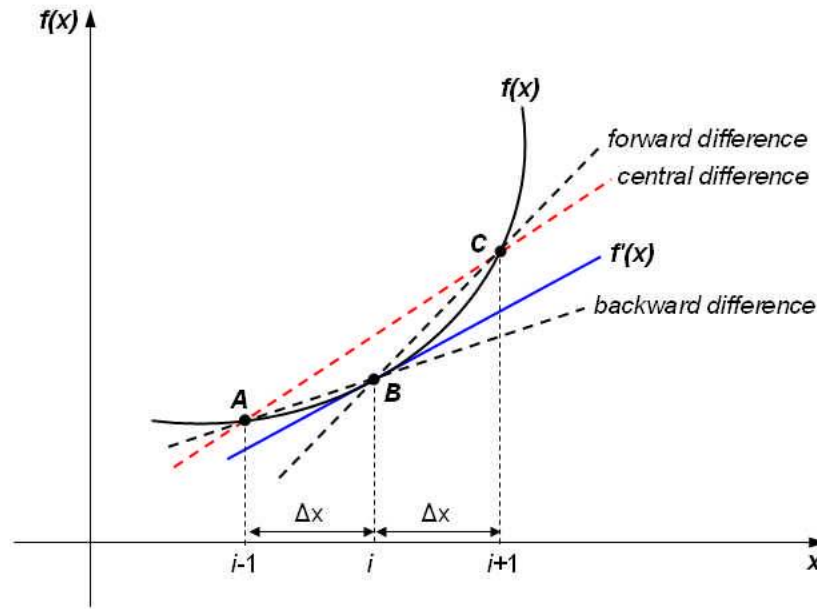


Figure 5.4: Estimation of first derivative of a function

Moreover, backward difference formula is given by:

$$\frac{\partial f}{\partial x} = f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{\Delta x} + R_n \quad (5.9)$$

Central difference formula:

$$\frac{\partial f}{\partial x} = f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1}))}{2\Delta x} + R_n \quad (5.10)$$

If the function contains two independent variables such as $f(x, y)$, then the derivatives in the Taylor series will have the following form:

$$f(x_{i+1}, y_{j+1}) = f(x_i, y_j) + \frac{\partial f}{\partial x} \frac{\Delta x}{1!} + \frac{\partial f}{\partial y} \frac{\Delta y}{1!} + \frac{\partial^2 f}{\partial x^2} \frac{\Delta x^2}{2!} + \frac{\partial^2 f}{\partial y^2} \frac{\Delta y^2}{2!} + \dots + \frac{\partial^n f}{\partial y^n} \frac{\Delta y^n}{n!} \quad (5.11)$$

Equation 5.11 can be written into the following:

Forward difference formula of first derivative:

$$\frac{\partial f}{\partial x} \approx \frac{f_{i+1,j} - f_{i,j}}{\Delta x}$$

$$\frac{\partial f}{\partial y} \approx \frac{f_{i,j+1} - f_{i,j}}{\Delta y}$$

Central difference formula of first derivative:

$$\frac{\partial f}{\partial x} \approx \frac{f_{i+1,j} - f_{i-1,j}}{2\Delta x}$$

$$\frac{\partial f}{\partial y} \approx \frac{f_{i,j+1} - f_{i,j-1}}{2\Delta y}$$

Central difference formula of **second** derivative:

$$\frac{\partial^2 f}{\partial x^2} \approx \frac{f_{i-1,j} - 2f_{i,j} + f_{i+1,j}}{\Delta x^2}$$

$$\frac{\partial^2 f}{\partial y^2} \approx \frac{f_{i,j-1} - 2f_{i,j} + f_{i,j+1}}{\Delta y^2}$$

Subscript i, j on the function f(i, j) is a simplification of f(xi, yj) form. A grid points of function values in the xy coordinate system is shown in Figure 5.5, it is used to estimate derivative of partial differential with respect to x and y.

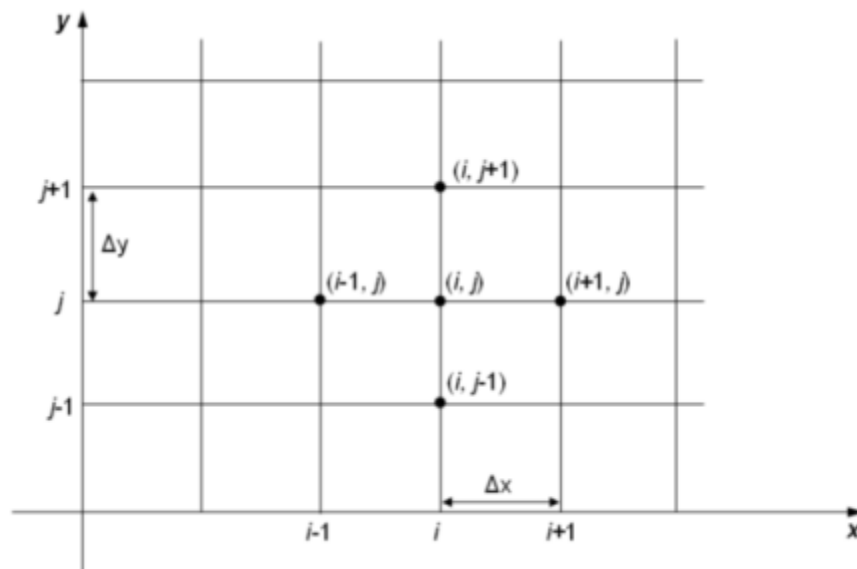


Figure 5.5: Two dimensional grid points of a function

Taylor's discretization method whereby derivative of a function is approached as the difference value of a function between the given independent variable (e.g. x_1) and a small increment (e.g. $x_1 + \Delta x$), is called the **Finite Difference Method** (Gilberto E.U, 2004). An approach using finite difference method implies that the smaller Δx is used the better the result closer to the exact value as shown from an approach of slope of the line in Figure 5.4.

In Chapter 10, we will build a program using finite difference method for the one-dimensional consolidation based on Equation 5.6. In the process, it is necessary to put a multiplier into input data to divide interval of the independent variables (e.g. Δx) to be smaller so that the solution is expected close to exact result. Program is created in VBA and of course, with the advantages of making chart automation to present the result.

CHAPTER 6

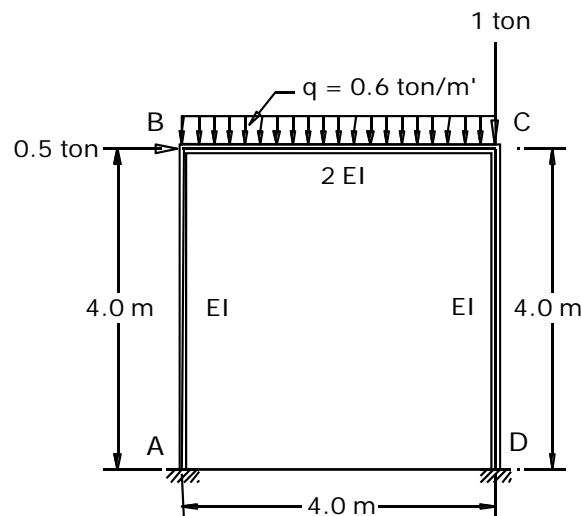
PROGRAM FOR 2D FRAME STRUCTURE ANALYSIS

The matrix program as described in Chapter 4 is now implemented for analysis of structure in this chapter and the next few chapters. It is expected that these series of references for Excel-VBA programming will be complementary, as basic knowledge for readers to make an application program.

Please differentiate between the purpose of programming in Excel and the aim to build an executable program (using Visual Basic or C++ programming language). Using Excel means the program as a whole to be convenient and practical for it is created case by case. Each case is packed in such a way to utilize the facilities on the spreadsheet and then presented as in creating a technical report. This is consistent with the nature of working with Excel.

6.1 CASE EXAMPLE

Structure of plane frame is composed of 3 members with 4 m long each. The stiffness of member BC is 2 times of the stiffness of members AB or CD. Point B, and C are the free nodes, each has 3 displacement components.



All members of the frame are made of the same linear elastic material with modulus of elasticity, $E = 2.1 \times 10^6 \text{ ton/m}^2$. Dimension of member $AB = BC = CD = 30 \text{ cm}$.

The given load as the following:

- Known external load:

H at node B = 0.5 ton

V at node C = 1 ton

Uniform load, q at member BC = 0.6 ton/m'

- Uniform load, q is converted to equivalent nodal loads at B and C

(see sign convention in Fig 6.1)

$$P_{BC(M)} = -P_{CB(M)} = 1/12 \cdot q \cdot L^2 = -0.8 \text{ ton.m.}$$

$$P_{B(V)} = P_{C(V)} = -1/2 \cdot q \cdot L = -1.2 \text{ ton.}$$

Sign convention represents the direction of the forces - displacements with reference to either the global coordinate or the local coordinate. It is set as follows: axial and shear forces follow the right-hand rule and the moments follow the right-hand screw rule, or in the easy way it is evident from Figure 6.1, i.e. positive when in the direction of the positive axis.

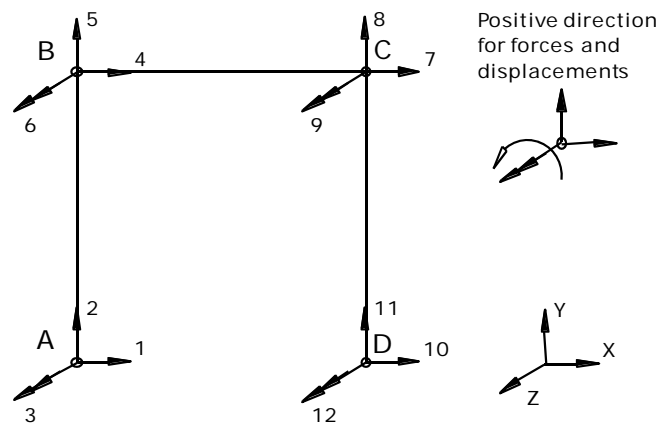


Figure 6.1: Force and displacement components at nodes in global coordinate system

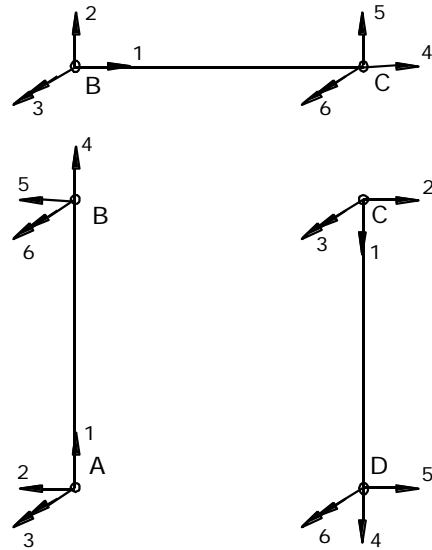


Figure 6.2: Members force and displacement components in local coordinate system

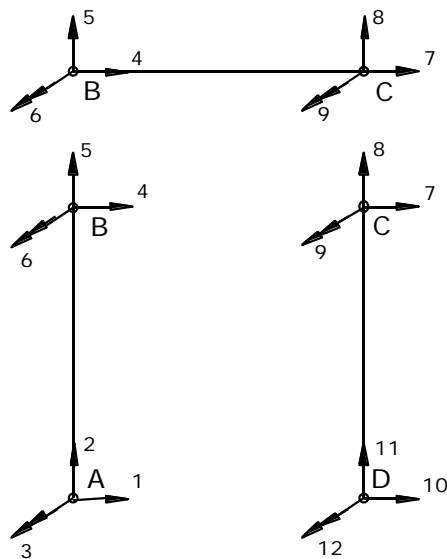


Figure 6.3: Members force and displacement components that conform to global coordinate system

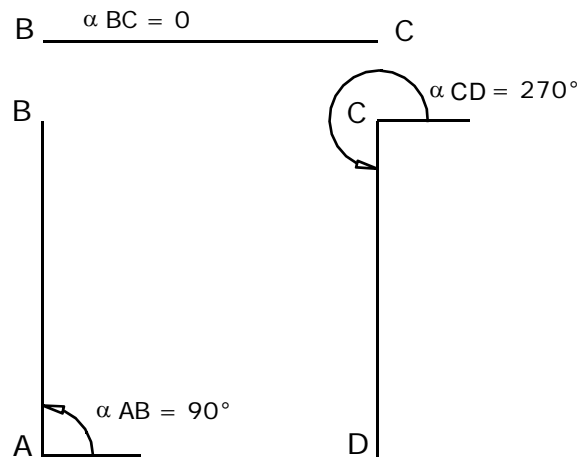
The process of frame analysis consists of the following steps:

1. Stiffness matrix for plane frame member is derived as the following:

$$[K]_i = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & \frac{12EI}{L^3} & \frac{6EI}{L^2} & 0 & -\frac{12EI}{L^3} & \frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{4EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{2EI}{L} \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & -\frac{12EI}{L^3} & -\frac{6EI}{L^2} & 0 & \frac{12EI}{L^3} & -\frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{2EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{4EI}{L} \end{bmatrix} \quad (6.1)$$

2. Transformation matrix to convert vector components from local coordinate to global coordinate is as the following:

For 6 force - displacement vector components of each member,



$$[T]_i = \begin{bmatrix} \cos \alpha & \sin \alpha & 0 & 0 & 0 & 0 \\ -\sin \alpha & \cos \alpha & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos \alpha & \sin \alpha & 0 \\ 0 & 0 & 0 & -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.2)$$

3. Member stiffness matrix in the global coordinate system:

$$[K]_s = [T]_i^T [K]_i [T]_i \quad (6.3)$$

Member stiffness matrix is then superposed with other member, which altogether form global stiffness matrix or called $[K]$ structure. This process is written:

$$\text{Superpose } [K]_s \rightarrow [K] \text{ structure} \quad (6.4)$$

4. Displacements at the free nodes to be determined by Equation 4.9 that becomes:

$$\{X_f\} = [K_{ff}]^{-1} \{P_f\} \quad (6.5)$$

5. Internal forces on each member can be found by firstly transforming displacements from global coordinate back to local coordinate,

$$\{X\}_i = [T]_i \{X_f\}$$

After the displacements of member $\{X\}_i$ are obtained, then internal forces on each member can be found using Equation 4.11:

$$\{P\}_i = \{P_o\} + [K]_i \{X\}_i \quad (6.6)$$

6. Support reactions are determined using Equation 4.12:

$$\{R_b\} = [K_{bf}] \{X_f\} - \{P_b\} \quad (6.7)$$

From above sequence of process then we can create a simple program for frame structure analysis named FRAME2D.

Programming:

In designing matrix program, it is convenient to mastering design-build in matrix-based data. In this relation, is how to make matrix variables of Equation 6.3 to 6.7. There are a few things in coding that actually require a little imagination of a programmer.

For instance is how to assembly $[K]$ structure. For that purpose, it requires to manipulate program steps, so that $[K]$ members can be placed on their position in $[K]$ structure. The idea is simply creating an integer variable, namely **dindex**, to equalize subscripts of $[K]$ member. **Dindex** reads index (numbering) of members joint displacements in the global coordinate system and uses the subscripts to store $[K]$ members in $[K]$ structure and do the superposition at once.

The program code is as follows:

```

'Member end-displacements index:
For i = 1 To NM
    With Member(i)
        dindex(1, i) = 3 * .J1 - 2
        dindex(2, i) = 3 * .J1 - 1
        dindex(3, i) = 3 * .J1
        dindex(4, i) = 3 * .J2 - 2
        dindex(5, i) = 3 * .J2 - 1
        dindex(6, i) = 3 * .J2
    End With
Next i

'Storing members and superposition
For i = 1 to NM 'NM= no. of member
    For j = 1 To 6
        For n = 1 To 6
            S(Idm(j, i), Idm(n, i)) = S(Idm(j, i), Idm(n, i)) + M(j, n, i)
        Next n
    Next j
Next i

```

Notes:

- **Idm** variable is the same displacement index as stated by **dindex** variable, but declared at a different level. **Idm** is at public-level, while **dindex** is at procedure-level. For more clearly understand how these variables are proceeded you can see in FRAME2D code provided in the Attachment.
- The above code is for completion of the description (6.4).

The other thing is to partition (separation) of the structure stiffness matrix to form $[K_{ff}]$ and $[K_{bf}]$ as written in Equation 4.8. To do this, it uses two integer variables namely **Ifr** to form $[K_{ff}]$ and **Ifx** to form $[K_{bf}]$.

The sequence of steps to partition the structure stiffness matrix is as follows:

1. Creating displacement index at free nodes and supports

```

'Joint free displacement index, IFr and restraint index, IFx:
n = 1

```



```

j = 1
For i = 1 To NP
    If Rs(i) = 0 Then IFr(n) = i: n = n + 1
Next i

n = 1
For i = 1 To NS
    n = Cells(22, 1 + i)
    IFx(3 * i - 2) = 3 * n - 2
    IFx(3 * i - 1) = 3 * n - 1
    IFx(3 * i) = 3 * n
Next i

```

2. Sub-matrix $[K_{ff}]$ is obtained as follows:

```

'submatrix Stiffness, KFF
For i = 1 To DOF
    For j = 1 To DOF
        SD(i, j) = GS(Idj(i), Idj(j))
    Next j
Next i

```

3 Sub-matrix $[K_{bf}]$ is obtained as follows:

```

'submatrix Stiffness, KBF
For i = 1 To DOF
    For j = 1 To 3 * NS
        SR(j, i) = GS(Irj(j), Idj(i))
    Next j
Next i

```

Notes:

- Rs represents joint displacement condition at supports, 1 = fixed, 0 = free.
- NS = no. of supports
- Variable $Idj = Ifr$, $Irj = Ifx$, but they are declared at different level which is at *public* and *procedure level*, respectively. See FRAME2D code provided in the Attachment.

The input and output data in form of FRAME2D program are presented in Figure 6.4. Below are the results of each step of the analysis.

Member AB

[K] (member stiffness matrix)

47250.00	0.00	0.00	-47250.00	0.00	0.00
0.00	265.78	531.56	0.00	-265.78	531.56
0.00	531.56	1417.50	0.00	-531.56	708.75
-47250.00	0.00	0.00	47250.00	0.00	0.00
0.00	-265.78	-531.56	0.00	265.78	-531.56
0.00	531.56	708.75	0.00	-531.56	1417.50

[T] (member transformation matrix from local to global coordinate system)

0.00	1.00	0.00	0.00	0.00	0.00
-1.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	1.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	1.00	0.00
0.00	0.00	0.00	-1.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	1.00

$[K]_s = [T]^T [K]_l [T]$ (member stiffness matrix in global coordinate)

265.78	0.00	-531.56	-265.78	0.00	-531.56
0.00	47250.00	0.00	0.00	-47250.00	0.00
-531.56	0.00	1417.50	531.56	0.00	708.75
-265.78	0.00	531.56	265.78	0.00	531.56
0.00	-47250.00	0.00	0.00	47250.00	0.00
-531.56	0.00	708.75	531.56	0.00	1417.50

Member BC

[K]

59850.00	0.00	0.00	-59850.00	0.00	0.00
0.00	540.15	1080.29	0.00	-540.15	1080.29
0.00	1080.29	2880.78	0.00	-1080.29	1440.39
-59850.00	0.00	0.00	59850.00	0.00	0.00
0.00	-540.15	-1080.29	0.00	540.15	-1080.29
0.00	1080.29	1440.39	0.00	-1080.29	2880.78

[T]

1.00	0.00	0.00	0.00	0.00	0.00
0.00	1.00	0.00	0.00	0.00	0.00
0.00	0.00	1.00	0.00	0.00	0.00
0.00	0.00	0.00	1.00	0.00	0.00
0.00	0.00	0.00	0.00	1.00	0.00
0.00	0.00	0.00	0.00	0.00	1.00

[K]s

59850.00	0.00	0.00	-59850.00	0.00	0.00
0.00	540.15	1080.29	0.00	-540.15	1080.29
0.00	1080.29	2880.78	0.00	-1080.29	1440.39
-59850.00	0.00	0.00	59850.00	0.00	0.00
0.00	-540.15	-1080.29	0.00	540.15	-1080.29
0.00	1080.29	1440.39	0.00	-1080.29	2880.78

Member CD

[K]

47250.00	0.00	0.00	-47250.00	0.00	0.00
0.00	265.78	531.56	0.00	-265.78	531.56
0.00	531.56	1417.50	0.00	-531.56	708.75
-47250.00	0.00	0.00	47250.00	0.00	0.00
0.00	-265.78	-531.56	0.00	265.78	-531.56
0.00	531.56	708.75	0.00	-531.56	1417.50

[T]

0.00	-1.00	0.00	0.00	0.00	0.00
1.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	1.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	-1.00	0.00
0.00	0.00	0.00	1.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	1.00

[K]s

265.78	0.00	531.56	-265.78	0.00	531.56
0.00	47250.00	0.00	0.00	-47250.00	0.00
531.56	0.00	1417.50	-531.56	0.00	708.75
-265.78	0.00	-531.56	265.78	0.00	-531.56
0.00	-47250.00	0.00	0.00	47250.00	0.00
531.56	0.00	708.75	-531.56	0.00	1417.50

[K] Structure

	[K]AB						[K]BC			[K]CD		
	1	2	3	4	5	6	7	8	9	10	11	12
1	265.78	0.00	-531.56	-265.78	0.00	-531.56	0.00	0.00	0.00	0.00	0.00	0.00
2	0.00	47250.00	0.00	0.00	-47250.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	-531.56	0.00	1417.50	531.56	0.00	708.75	0.00	0.00	0.00	0.00	0.00	0.00
4	-265.78	0.00	531.56	60115.78	0.00	531.56	-59850.00	0.00	0.00	0.00	0.00	0.00
5	0.00	-47250.00	0.00	0.00	47790.15	1080.29	0.00	-540.15	1080.29	0.00	0.00	0.00
6	-531.56	0.00	708.75	531.56	1080.29	4298.28	0.00	-1080.29	1440.39	0.00	0.00	0.00
7	0.00	0.00	0.00	-59850.00	0.00	0.00	60115.78	0.00	531.56	-265.78	0.00	531.56
8	0.00	0.00	0.00	0.00	-540.15	-1080.29	0.00	47790.15	-1080.29	0.00	-47250.00	0.00
9	0.00	0.00	0.00	0.00	1080.29	1440.39	531.56	-1080.29	4298.28	-531.56	0.00	708.75
10	0.00	0.00	0.00	0.00	0.00	0.00	-265.78	0.00	-531.56	265.78	0.00	-531.56
11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	47250.00	0.00	0.00	47250.00	0.00
12	0.00	0.00	0.00	0.00	0.00	0.00	531.56	0.00	708.75	-531.56	0.00	1417.50

Partition at free nodes $\rightarrow [K_{ff}]$

	4	5	6	7	8	9
4	60115.78	0.00	531.56	-59850.00	0.00	0.00
5	0.00	47790.15	1080.29	0.00	-540.15	1080.29
6	531.56	1080.29	4298.28	0.00	-1080.29	1440.39
7	-59850.00	0.00	0.00	60115.78	0.00	531.56
8	0.00	-540.15	-1080.29	0.00	47790.15	-1080.29
9	0.00	1080.29	1440.39	531.56	-1080.29	4298.28

Partition at support nodes $\rightarrow [K_{bf}]$

	4	5	6	7	8	9
1	-265.78	0.00	-531.56	0.00	0.00	0.00
2	0.00	-47250.00	0.00	0.00	0.00	0.00
3	531.56	0.00	708.75	0.00	0.00	0.00
10	0.00	0.00	0.00	-265.78	0.00	-531.56
11	0.00	0.00	0.00	0.00	-47250.00	0.00
12	0.00	0.00	0.00	531.56	0.00	708.75

Joint displacements at free nodes : $\{X_f\} = \begin{Bmatrix} 0.00000 \\ 0.00000 \\ 0.00000 \\ 0.00117 \\ -0.00002 \\ -0.00039 \\ 0.00117 \\ -0.00005 \\ 0.00017 \\ 0.00000 \\ 0.00000 \\ 0.00000 \end{Bmatrix}$

After getting $\{X_f\}$ then member deformations are calculated by using relationship:

$$\{X\}_i = [T]_i \{X_f\}$$

Member internal forces are calculated by using Equation 6.6:

$$\{P\}_i = \{P_o\} + [K]_i \{X\}_i$$

Support reactions are calculated by using Equation 6.7:

$$\{R_b\} = [K_{bf}] \{X_f\} - \{P_b\}$$

The result of the analysis is presented in FRAME2D input-output form as shown in Figure 6.4 and 6.5 below:

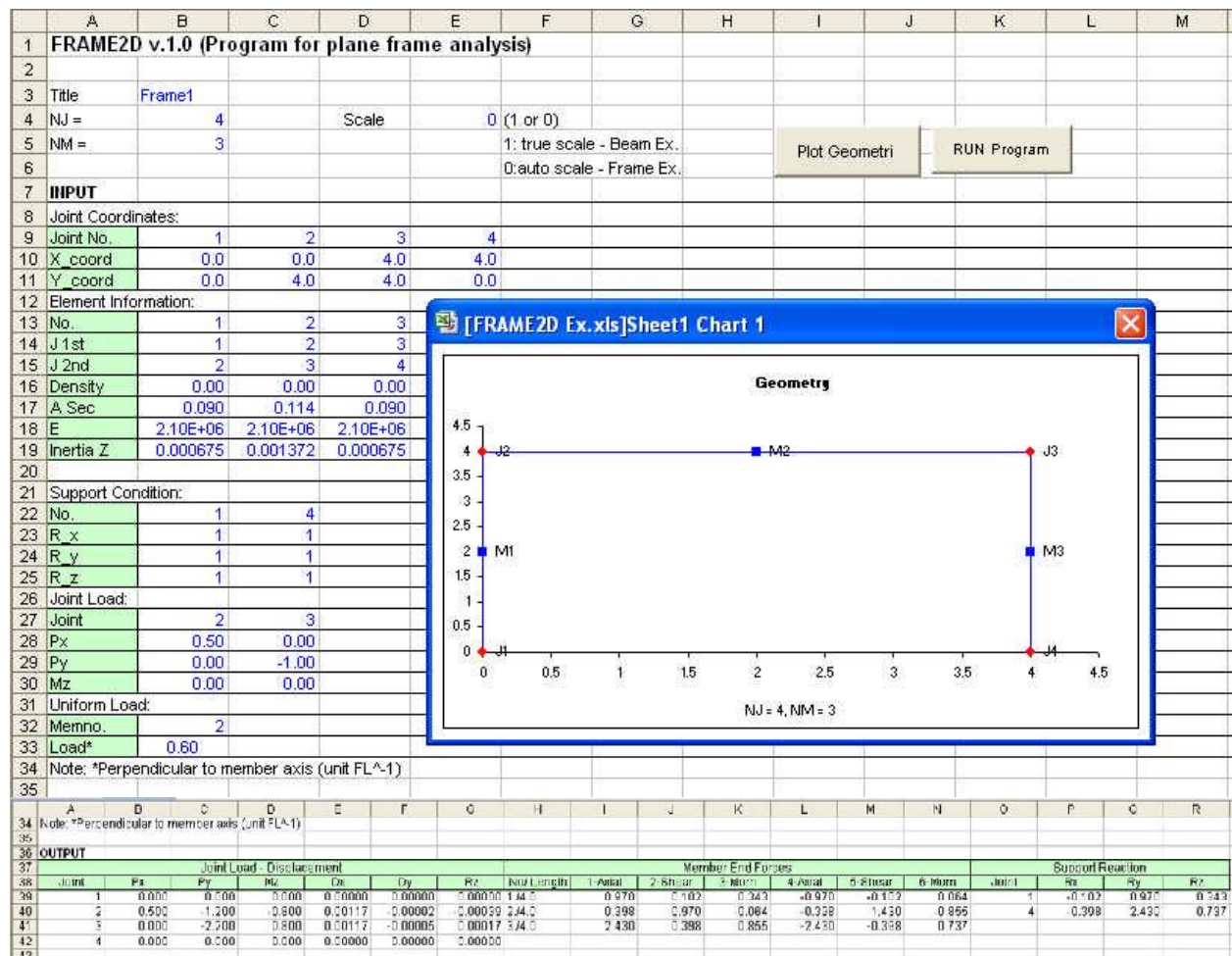


Figure 6.4: FRAME2D input-output form

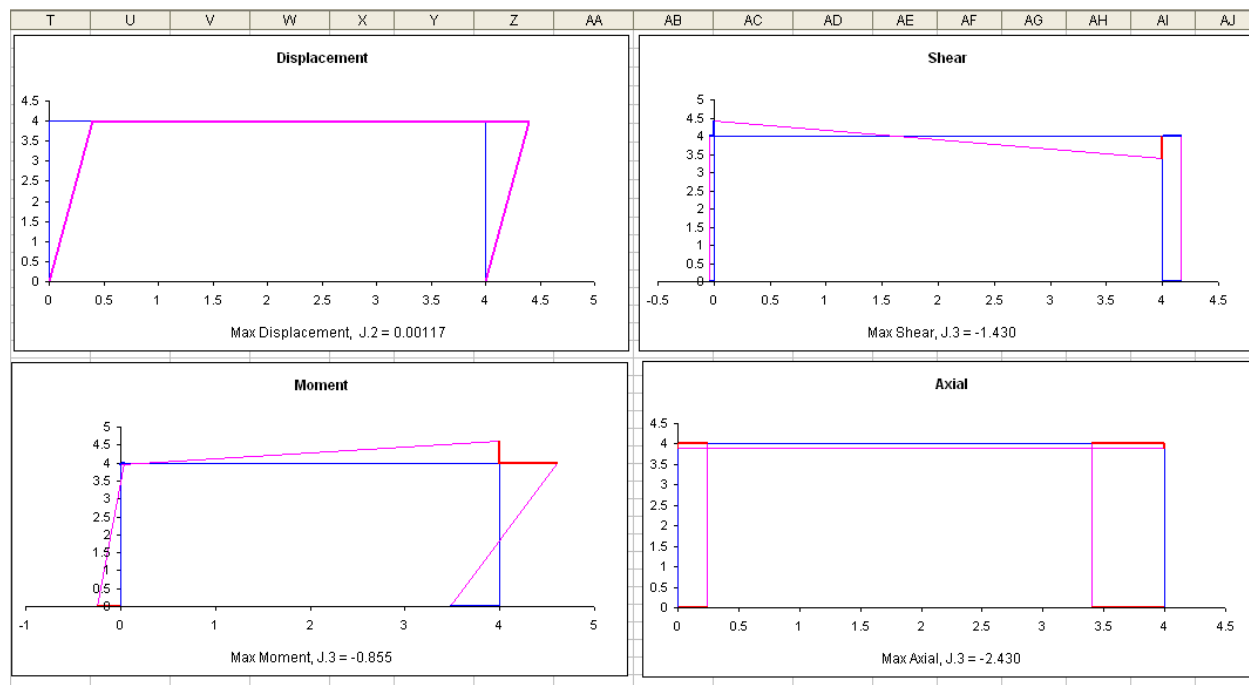


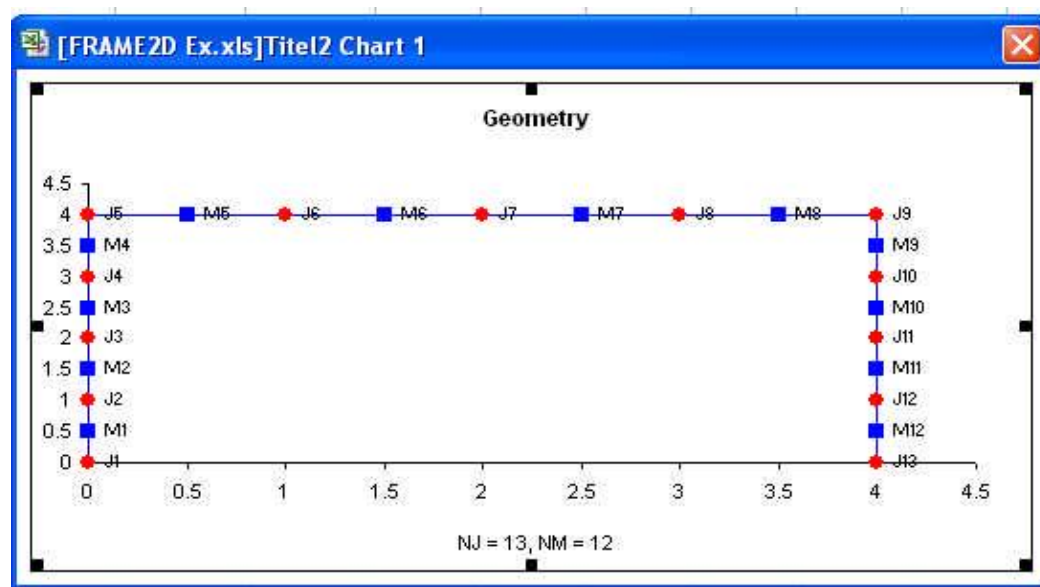
Figure 6.5: Charts of displacement, diagrams of bending moment, shear force and axial force

Notes for FRAME2D input and output form:

- Data input is blue text.
- Scale for presenting output data is divided into 2 cases: **true** scale for beam members (continuous beam) and **auto** scale for frame structure.
- NM: number of structural members
- NJ: number of joints or nodes
- FRAME2D distinguishes 3 types of loading: **selfweights** by inputting material density (row 16), **nodal loads** (row 27 to 30), and **uniform loads** (row 32 to 33).
- When **Plot Geometry** button is clicked it shows Chart Window of the geometry of structure to give geometry verification to that inputted before the program is executed by clicking **RUN Program** button. It is expected to minimize kind of errors that caused by numbering system of the members and joints.
- Chart Window command used in Plot Geometry is only compatible with **Excel 2003** and earlier, which is a version where FRAME2D was created (also the next few programs). When running it in Excel 2007 version, Chart Window (as shown in Figure 6.4) will not be displayed. However, readers can see it placed before displacement chart in the input-output form.

- Figure 6.5 shows chart of displacement as well as diagrams of moment, shear and axial forces that are drawn automatically within the program. It is also included maximum values occurred on nodes. Sign convention used for presenting diagrams of the moments and internal forces is described in Chapter 6.2.
- The above calculation does not show the optimum result as the displacement of the second member looks like a straight line, contrary to the fact that the member is subjected to uniform load of 0.6 t/m'. Therefore, it is necessary to expose the result thoroughly by dividing all members into smaller members.
- With the intention of the above point, the frame members is now re-inputted into smaller members, so that it becomes $NM = 12$, and $NJ = 13$. The results are then shown in Figure 6.6 and 6.7.

The displayed geometry in chart window (only available in Excel 2003) from re-inputted data is shown below:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	FRAME2D v.1.0 (Program for plane frame analysis)													
2														
3	Title =	Frame2												
4	NJ =	13		Scale	0	(1 or 0)								
5	NM =	12				1: true scale - Beam Ex.								
6						0:auto scale - Frame Ex.								
7	INPUT													
8	Joint Coordinates:													
9	Joint No.	1	2	3	4	5	6	7	8	9	10	11	12	13
10	X_coord	0.0	0.0	0.0	0.0	0.0	1.0	2.0	3.0	4.0	4.0	4.0	4.0	4.0
11	Y_coord	0.0	1.0	2.0	3.0	4.0	4.0	4.0	4.0	4.0	3.0	2.0	1.0	0.0
12	Element Information:													
13	No.	1	2	3	4	5	6	7	8	9	10	11	12	
14	J 1st	1	2	3	4	5	6	7	8	9	10	11	12	
15	J 2nd	2	3	4	5	6	7	8	9	10	11	12	13	
16	Density	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
17	A Sec	0.090	0.090	0.090	0.090	0.114	0.114	0.114	0.114	0.090	0.090	0.090	0.090	
18	E	2.10E+06	2.10E+06	2.10E+06	2.10E+06	2.10E+06	2.10E+06	2.10E+06	2.10E+06	2.10E+06	2.10E+06	2.10E+06	2.10E+06	
19	Inertia Z	0.000675	0.000675	0.000675	0.000675	0.001372	0.001372	0.001372	0.001372	0.000675	0.000675	0.000675	0.000675	
20														
21	Support Condition:													
22	No.	1	13											
23	R_x	1	1											
24	R_y	1	1											
25	R_z	1	1											
26	Joint Load:													
27	Joint	5	9											
28	Px	0.50	0.00											
29	Py	0.00	-1.00											
30	Mz	0.00	0.00											
31	Uniform Load:													
32	Memno.	5	6	7	8									
33	Load*	0.60	0.60	0.60	0.60									
34	Note: *Perpendicular to member axis													
35														
36	OUTPUT													
37														
38		Joint Load - Displacement						Member End Forces						Support Reaction
39	Joint	Px	Py	Mz	Dx	Dy	Rz	No/Length	1/Axial	2/Shear	3/Mom	4/Axial	5/Shear	6/Mom
40	1	0.000	0.000	0.000	0.00000	0.00000	0.00000	1.71.0	0.970	0.102	0.543	-0.970	-0.102	-0.342
41	2	0.000	0.000	0.000	0.00011	-0.00001	-0.00021	2.71.0	0.970	0.102	0.242	-0.970	-0.102	-0.140
42	3	0.000	0.000	0.000	0.00039	-0.00001	-0.00034	3.71.0	0.970	0.102	0.140	-0.970	-0.102	-0.038
43	4	0.000	0.000	0.000	0.00072	-0.00002	-0.00040	4.71.0	0.970	0.102	0.038	-0.970	-0.102	0.064
44	5	0.500	-0.300	0.000	0.00117	-0.00002	-0.00030	5.71.0	0.308	0.270	-0.064	-0.308	0.270	0.734
45	6	0.000	-0.600	0.000	0.00117	-0.00036	-0.00024	6.71.0	0.308	0.270	-0.734	-0.308	0.230	0.804
46	7	0.000	-0.600	0.000	0.00117	-0.00046	-0.00005	7.71.0	0.200	-0.220	-0.504	-0.200	0.020	0.274
47	8	0.000	-0.600	0.000	0.00117	-0.00029	0.00025	8.71.0	0.308	-0.530	-0.274	-0.308	1.430	-0.856
48	9	0.000	-1.300	0.000	0.00117	-0.00005	0.00017	9.71.0	2.430	0.398	0.555	-2.430	-0.398	-0.457
49	10	0.000	0.000	0.000	0.00108	-0.00004	-0.00030	10.71.0	2.430	0.308	0.457	-2.430	-0.308	-0.060
50	11	0.000	0.000	0.000	0.00067	-0.00003	-0.00040	11.71.0	2.430	0.200	0.259	-2.430	-0.200	0.339
51	12	0.000	0.000	0.000	0.00027	-0.00001	-0.00039	12.71.0	2.430	0.200	-0.239	-2.430	-0.200	0.737
52	13	0.000	0.000	0.000	0.00000	0.00000	0.00000							

Figure 6.6: FRAME2D input-output form – Re-inputted

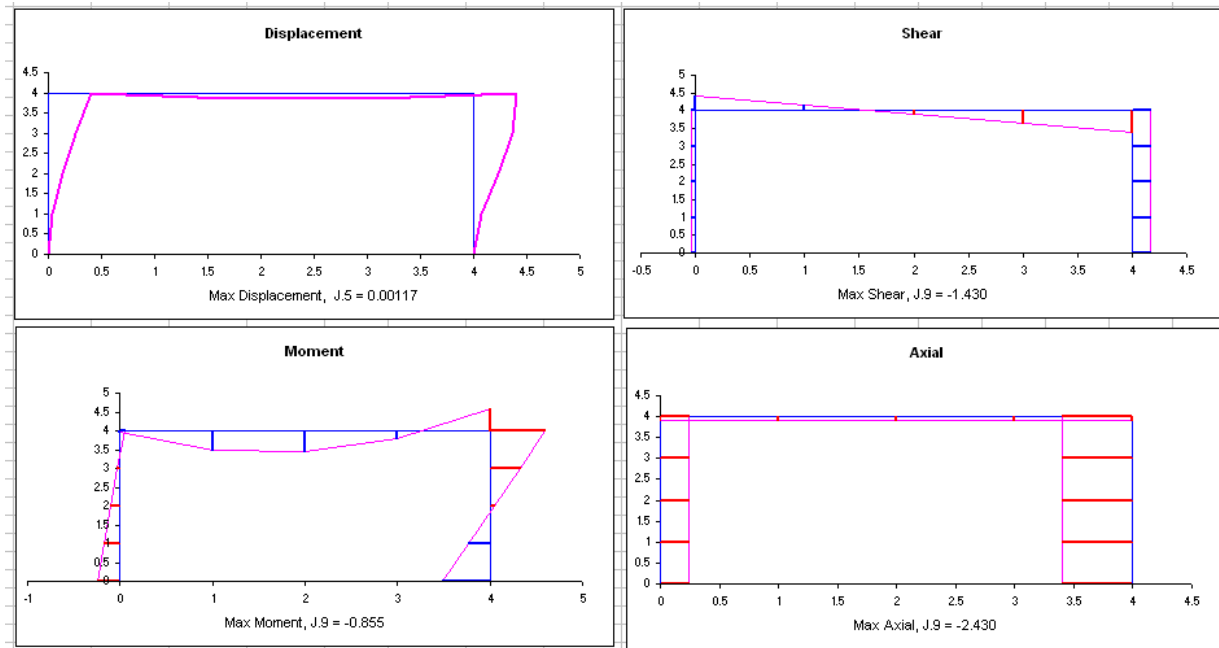
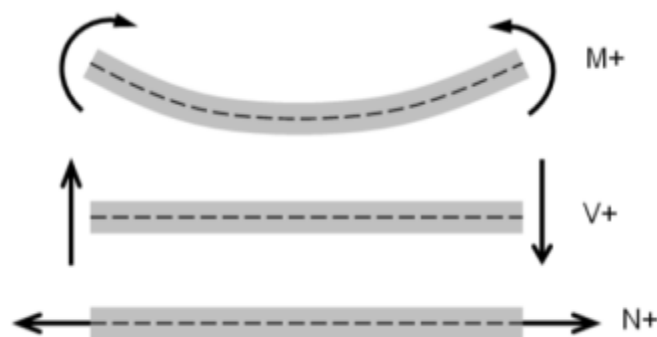


Figure 6.7: Charts of displacement, diagrams of bending moment, shear force and axial force – Re-inputted

The results shown in Figure 6.7 and 6.8 are much better than in Figure 6.5 and 6.6. Thus, it is convenient to divide into smaller members and for this example at least 4 smaller members with the same length for each member. Avoid odd divisors (e.g. 5, 7, 9 etc.) so that the results in the middle of span can be exposed.

6.2 SIGN CONVENTION FOR DIAGRAM

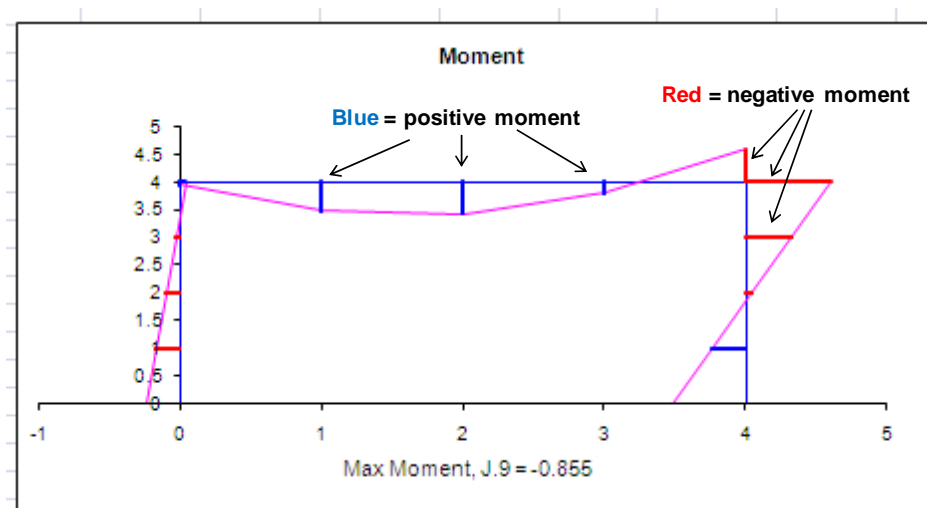
To depict diagrams of moment, shear force and axial force in members, program refers to the sign convention as follows:



Positive direction is as shown in the above picture and ruled as follows:

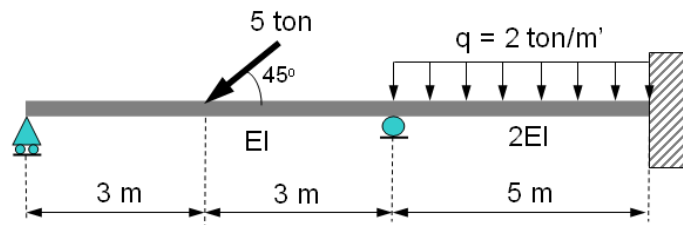
- Bending moment (M) is given a positive sign if causing the upper fibers of members is compressed or to curve upward, and is given a negative sign if to curve downward.
- Shear force (V) is given a positive sign if rotates the member clockwise and is given a negative sign if rotates the member counter-clockwise.
- Axial force (N) is given a positive sign if pulls the member apart, and is given a negative sign if pushes the member.

This convention is adopted in order to have same interpretation on the chart depiction of the application programs in the next few chapters, thus it is more easily interpreted and also design-oriented. The program automatically differentiates the moments and forces that occur in members section, which is **blue line** inside the curve for the **positive** direction and **red** for the **negative** direction. The example below shows the coloring for bending moment result:



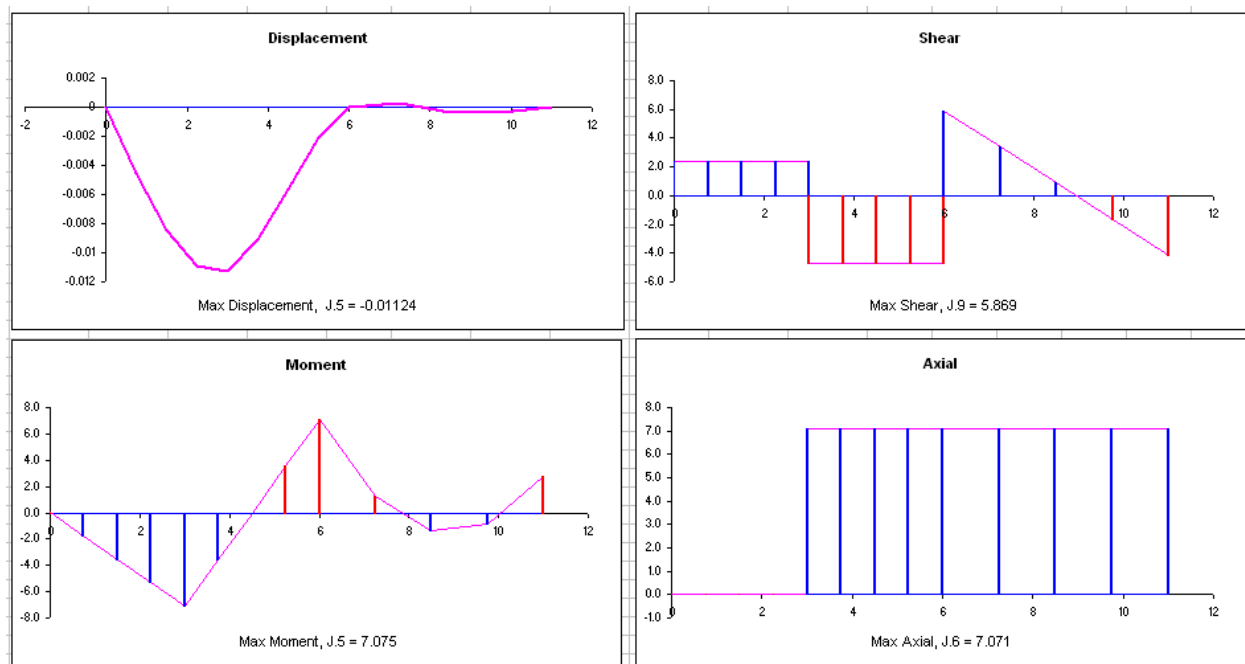
6.3 APPLICATION

FRAME2D is now applied to the following beam problem:



The beam can be seen to have three main members that are, roll-load segment, load-roll segment, roll-fixed node, have lengths 3, 3 and 5 meters, respectively. In practice however it needs to divide the beam into smaller members for exposing the result more accurately.

FRAME2D is then inputted from which each of main members is divided into 4 equal parts. So, there are 12 members and 13 joints. When the program has run, the charts will be shown as below.



CHAPTER 7

PROGRAM FOR 2D TRUSS STRUCTURE ANALYSIS

Member of plane truss is analyzed with consideration that carries only axial force, thus, there are two associated displacement components at both ends with respect to axial force. Truss member is generally made of steel or aluminum connected one to another as pin connection.

Stiffness matrix for plane truss member is derived as the following:

$$[K]_i = \begin{bmatrix} \frac{AE}{L} & 0 & -\frac{AE}{L} & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{AE}{L} & 0 & \frac{AE}{L} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Transformation matrix to convert vector components from local coordinate to global coordinate is as the following:

$$[T]_i = \begin{bmatrix} \cos \alpha & \sin \alpha & 0 & 0 \\ -\sin \alpha & \cos \alpha & 0 & 0 \\ 0 & 0 & \cos \alpha & \sin \alpha \\ 0 & 0 & -\sin \alpha & \cos \alpha \end{bmatrix}$$

The equation of member stiffness matrix in global coordinate system:

$$[K]_{s\ i} = [T]_i^T [K]_i [T]_i \quad (7.1)$$

$[K]_{s\ i}$ of the member is then superposed on the appropriate force-deflection components to get global stiffness matrix or called $[K]$ structure. The internal forces and support reactions can be determined by firstly partitioning $[K]$ structure at free nodes and at supports according to Equation 4.8.

7.1 CASE EXAMPLE

The following example includes a sequence of steps in the analysis for plane trusses based on matrix method as described in Chapter 4. It also gives some notes for writing code in Excel-VBA. This section is complement explanation for previous FRAME2D example.

Given a truss structure is shown in Figure 7.1, includes numbering system in Figure 7.2 to 7.4. Each of member are made of the same linear elastic material, with $E = 2,100,000 \text{ kg/cm}^2$ and cross-sectional area $A = 68.4 \text{ cm}^2$.

Truss as a whole has only two degrees of freedom (DOF) which is at vector components 3 and 4, while both supports are pinned-pinned thus will have no any translation.

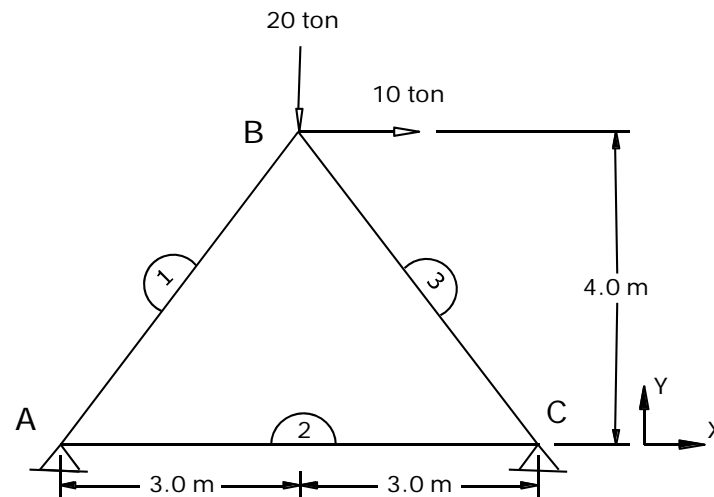


Figure 7.1: Geometry and loading condition on plane truss

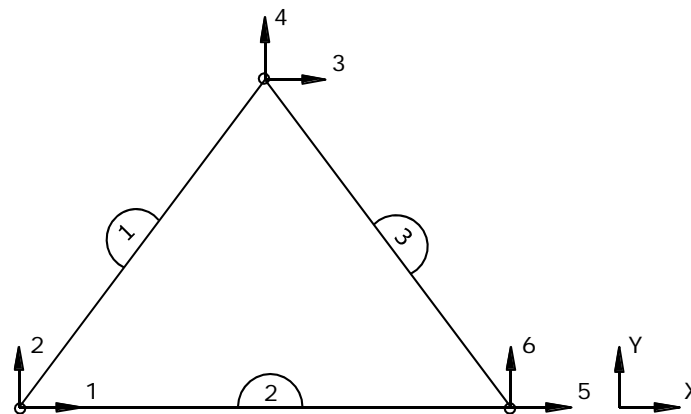


Figure 7.2: Force and displacement components at nodes in global coordinate system

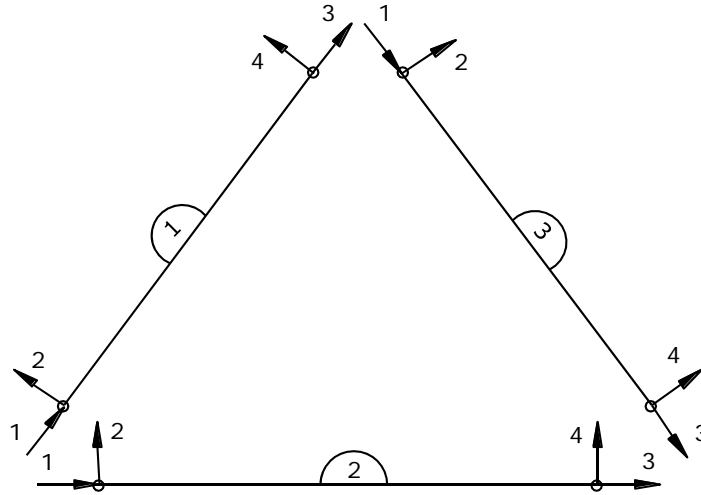


Figure 7.3: Members force and displacement components in local coordinate system

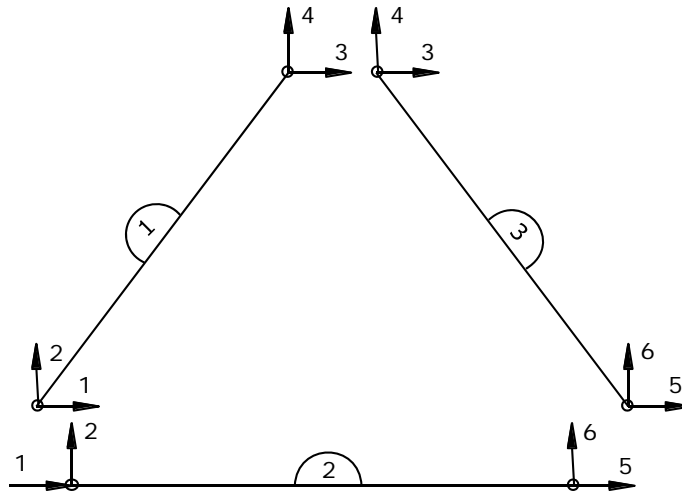


Figure 7.4: Members force and displacement components that conform to global coordinate system

The first step is to build member stiffness matrix with reference to either local or global coordinates. In the program, it is done within one looping through the member number, and made in a separate procedure in Module 2.

Program code:

```
Sub Stiff_Mtx(MK() As Double, S() As Double)
Dim sms As Double
```



```

ReDim M(4, 4, NM) As Double

'Building global stiffness matrix
For i = 1 To NM
    With Member(i)
        sms = .Ax * .E / .Lh
        'Member stiffness
        MK(1, 1, i) = sms: MK(1, 3, i) = -sms
        MK(3, 1, i) = -sms: MK(3, 3, i) = sms

        'Member global stiffness
        M(1, 1, i) = sms * .Cx * .Cx: M(1, 2, i) = sms * .Cx * .Cy: M(1, 3, i) =
-M(1, 1, i): M(1, 4, i) = -M(1, 2, i)
        M(2, 1, i) = M(1, 2, i): M(2, 2, i) = sms * .Cy * .Cy: M(2, 3, i) = M(1,
4, i): M(2, 4, i) = -M(2, 2, i)
        M(3, 1, i) = M(1, 3, i): M(3, 2, i) = M(2, 3, i): M(3, 3, i) = M(1, 1,
i): M(3, 4, i) = M(1, 2, i)
        M(4, 1, i) = M(1, 4, i): M(4, 2, i) = M(2, 4, i): M(4, 3, i) = M(3, 4,
i): M(4, 4, i) = M(2, 2, i)

        'Storing members and superposition
        For j = 1 To 4
            For n = 1 To 4
                S(Idm(j, i), Idm(n, i)) = S(Idm(j, i), Idm(n, i)) + M(j, n, i)
            Next n
        Next j
    End With
Next i

End Sub

```

Notes:

- The above code is about to build the members global stiffness matrix as shown in Equation 7.1, and then store them to form [K] structure and do the superposition at once.
- Each member matrix (variable M) has three indexes in it; the first and second states rows and columns subscripts of the matrix, respectively, while the third index states matrix of member i.

- Index Idm is an integer variable that reads displacement indexes of ends members, the same as used in FRAME2D.

The program for truss analysis is modified from FRAME2D and named TRUSS2D. Below are the results of each step of the analysis.

Member Stiffness Matrix

[K]AB

28728.00	0.00	-28728.00	0.00
0.00	0.00	0.00	0.00
-28728.00	0.00	28728.00	0.00
0.00	0.00	0.00	0.00

[K]AC

23940.00	0.00	-23940.00	0.00
0.00	0.00	0.00	0.00
-23940.00	0.00	23940.00	0.00
0.00	0.00	0.00	0.00

[K]BC

28728.00	0.00	-28728.00	0.00
0.00	0.00	0.00	0.00
-28728.00	0.00	28728.00	0.00
0.00	0.00	0.00	0.00

Transformation Matrix

[T]AB

0.60	0.80	0.00	0.00
-0.80	0.60	0.00	0.00
0.00	0.00	0.60	0.80
0.00	0.00	-0.80	0.60

[T]AC

1.00	0.00	0.00	0.00
0.00	1.00	0.00	0.00
0.00	0.00	1.00	0.00
0.00	0.00	0.00	1.00

[T]BC

0.60	-0.80	0.00	0.00
0.80	0.60	0.00	0.00
0.00	0.00	0.60	-0.80
0.00	0.00	0.80	0.60

Member Stiffness Matrix in Global Coordinate System

$$[K]_s = [T]_i^T [K]_i [T]_i$$

[K]_s AB

10342.08	13789.44	-10342.08	-13789.44
13789.44	18385.92	-13789.44	-18385.92
-10342.08	-13789.44	10342.08	13789.44
-13789.44	-18385.92	13789.44	18385.92

[K]_s AC

23940.00	0.00	-23940.00	0.00
0.00	0.00	0.00	0.00
-23940.00	0.00	23940.00	0.00
0.00	0.00	0.00	0.00

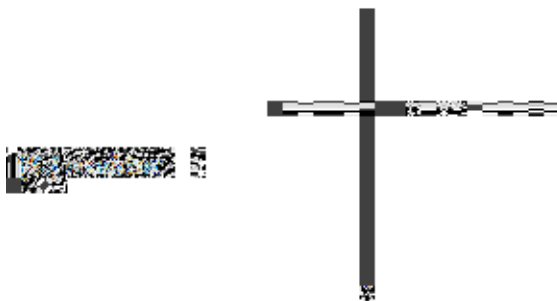
[K]s BC

10342.08	-13789.44	-10342.08	13789.44
-13789.44	18385.92	13789.44	-18385.92
-10342.08	13789.44	10342.08	-13789.44
13789.44	-18385.92	-13789.44	18385.92

Structure Stiffness Matrix

	1	2	3	4	5	6
1	34282.08	13789.44	-10342.08	-13789.44	-23940.00	0.00
2	13789.44	18385.92	-13789.44	-18385.92	0.00	0.00
3	-10342.08	-13789.44	20684.16	0.00	-10342.08	13789.44
4	-13789.44	-18385.92	0.00	36771.84	13789.44	-18385.92
5	-23940.00	0.00	-10342.08	13789.44	34282.08	-13789.44
6	0.00	0.00	13789.44	-18385.92	-13789.44	18385.92

The next step is to partition [K] structure at free nodes and supports. Manipulation on program steps to establish [K_{ff}] has been discussed earlier in FRAME2D example.



$$[K_{ff}] = \begin{bmatrix} 20684.16 & 0.00 \\ 0.00 & 36771.84 \end{bmatrix}$$

From Equation 4.9,

$$\{P_f\} = [K_{ff}]\{X_f\}$$

$$\{X_f\} = [K_{ff}]^{-1}\{P_f\}$$

$$\{X_f\} = \begin{Bmatrix} 0.00048 \\ -0.00054 \end{Bmatrix}$$

After displacements of free nodes are known, the internal forces of each member can be determined using equation:

$$\{P\}_i = \{P_o\} + [K]_i([T]_i\{X_f\})$$

$[T]_i\{X_f\}$ is displacements vector transformed from local to global coordinate system.

In this example, no external load directly acting on each member thus $\{P_o\} = 0$. The equation to determine the internal forces of member becomes:

$$\{P\}_i = [K]_i([T]_i\{X_f\})$$

The values of internal forces of each member are shown below:

$\{P\}_{AB} :$

4.167
0.00
-4.167
0.00

$\{P\}_{AC} :$

0.00
0.00
0.00
0.00

$\{P\}_{BC} :$

20.833
0.00
-20.833
0.00

The magnitude of support reactions can be found using Equation 4.12:

$$\{R_b\} = [K_{bf}]\{X_f\} - \{P_b\}$$

There is no external load directly acting on supports thus $\{P_b\} = 0$. The equation to determine the support reaction becomes:

$$\{R_b\} = [K_{bf}]\{X_f\}$$

In this stage, we have to establish $[K_{bf}]$ first. It uses the same way as to establish $[K_{ff}]$, which to partition using integer variables. The completed code is similar to that in FRAME2D:

```
'Indexing for matrix subscript
Sub Mindex(index() As Integer, IFr() As Integer, IFx() As Integer)
'Member end-displacements index:
```

```

For i = 1 To NM
    With Member(i)
        dindex(1, i) = 2 * .J1 - 1
        dindex(2, i) = 2 * .J1
        dindex(3, i) = 2 * .J2 - 1
        dindex(4, i) = 2 * .J2
    End With
Next i

'Joint free displacement index, IFr and
'restraint (support) index, IFx
n = 1
j = 1
For i = 1 To NP
    If Rs(i) = 0 Then IFr(n) = i: n = n + 1
Next i

n = 1
For i = 1 To NS
    n = Cells(22, 1 + i)
    IFx(2 * i - 1) = 2 * n - 1
    IFx(2 * i) = 2 * n
Next i
End Sub

Sub MINvers(SR() As Double, SD() As Double)
...
'submatrix Stiffness, KBF
For i = 1 To DOF
    For j = 1 To 3 * NS
        SR(j, i) = GS(Irj(j), Idj(i))
    Next j
Next i
...
End Sub

```

We have known that variable $Idj = Ifr$, $Irj = Ifx$, but they are declared at different level which is at *public* and *procedure level*, respectively. With the above code, the subscript i,j of $[K_{bf}]_{i,j}$ associated with this example are:

$$[K_{bf}] = \begin{bmatrix} 1,3 & 1,4 \\ 2,3 & 2,4 \\ 5,3 & 5,4 \\ 6,3 & 6,4 \end{bmatrix}$$

And the values of its elements are:

$$[K_{bf}] = \begin{bmatrix} -10342.08 & -13789.44 \\ -13789.44 & -18385.92 \\ -10342.08 & 13789.44 \\ 13789.44 & -18385.92 \end{bmatrix}$$

The magnitudes of support reactions are determined:

$$\{R_b\} = \begin{Bmatrix} 2.500 \\ 3.333 \\ -12.500 \\ 16.667 \end{Bmatrix} \text{ ton}$$

The results of the internal forces of members and supports reactions are then plotted on the truss chart as shown in Figure 7.5. Negative sign is used to indicate compression state along the member, and positive sign for tension state.

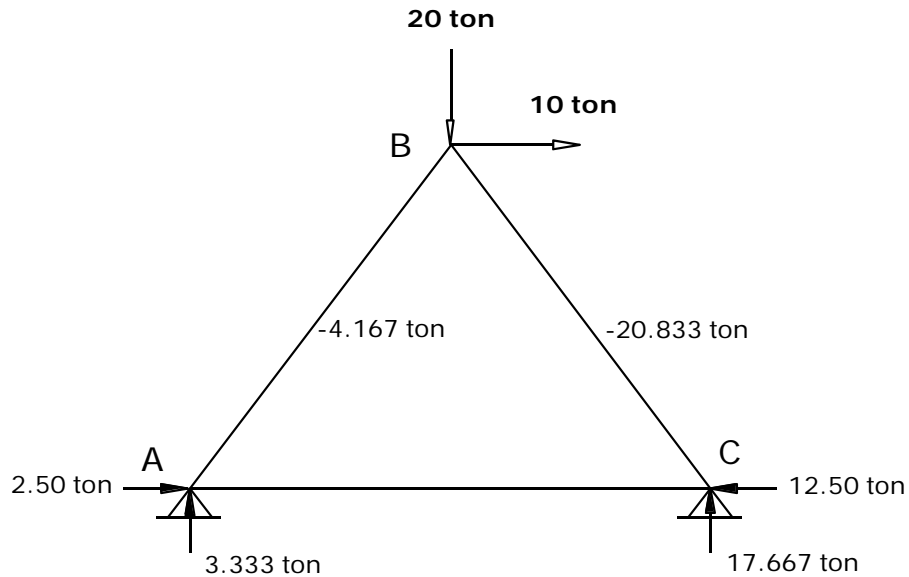


Figure 7.5: Depiction of result from 2D truss analysis

Check balance of forces at point B:

$$\Sigma F_H = 10 + 4.167 \times 3/5 - 20.833 \times 3/5 = 0$$

$$\Sigma F_V = 20 - 4.167 \times 4/5 - 20.833 \times 4/5 = 0 \dots \text{OK!}$$

Check balance of forces of a whole structure:

$$\Sigma F_H = 10 + 2.50 - 12.50 = 0$$

$$\Sigma F_V = 20 - 3.333 - 17.667 = 0 \dots \text{OK!}$$

The input and output form of TRUSS2D for given truss example is as follows:

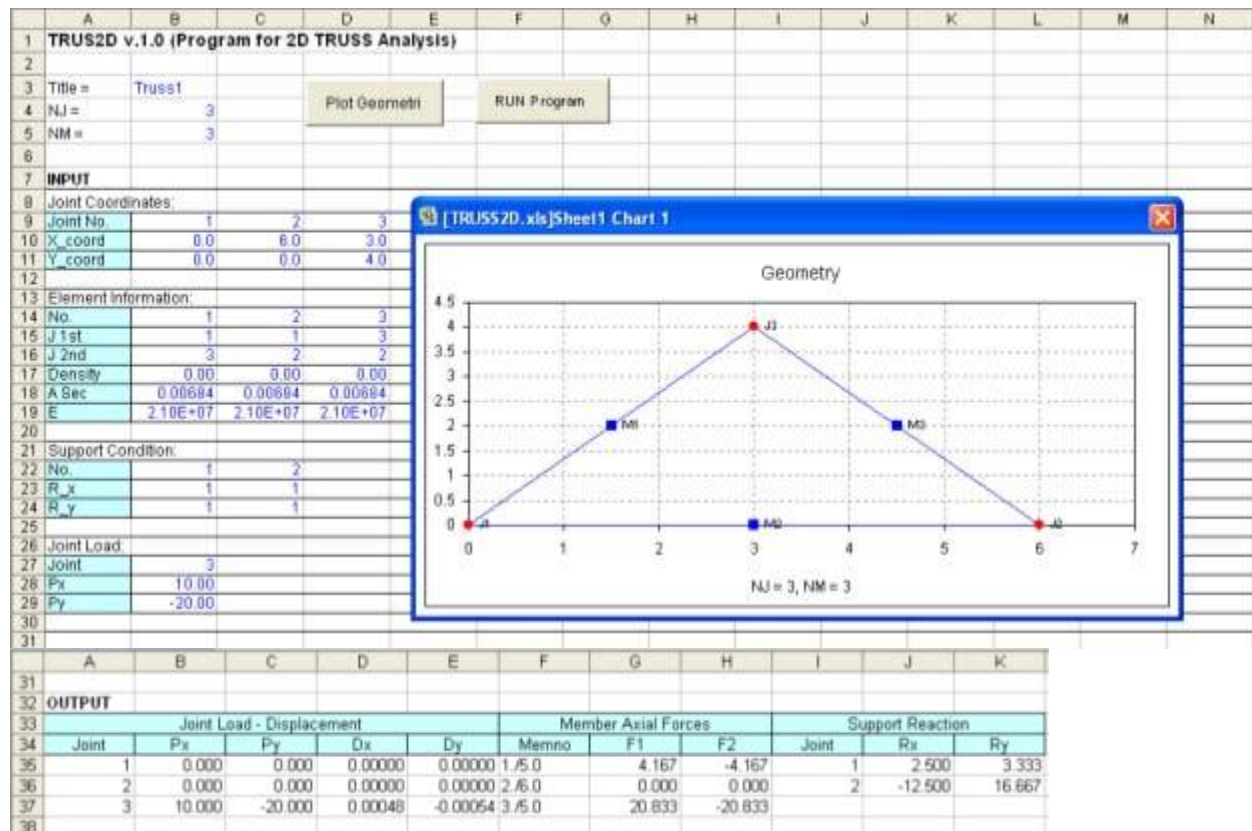


Figure 7.6: Input-output form of TRUSS2D

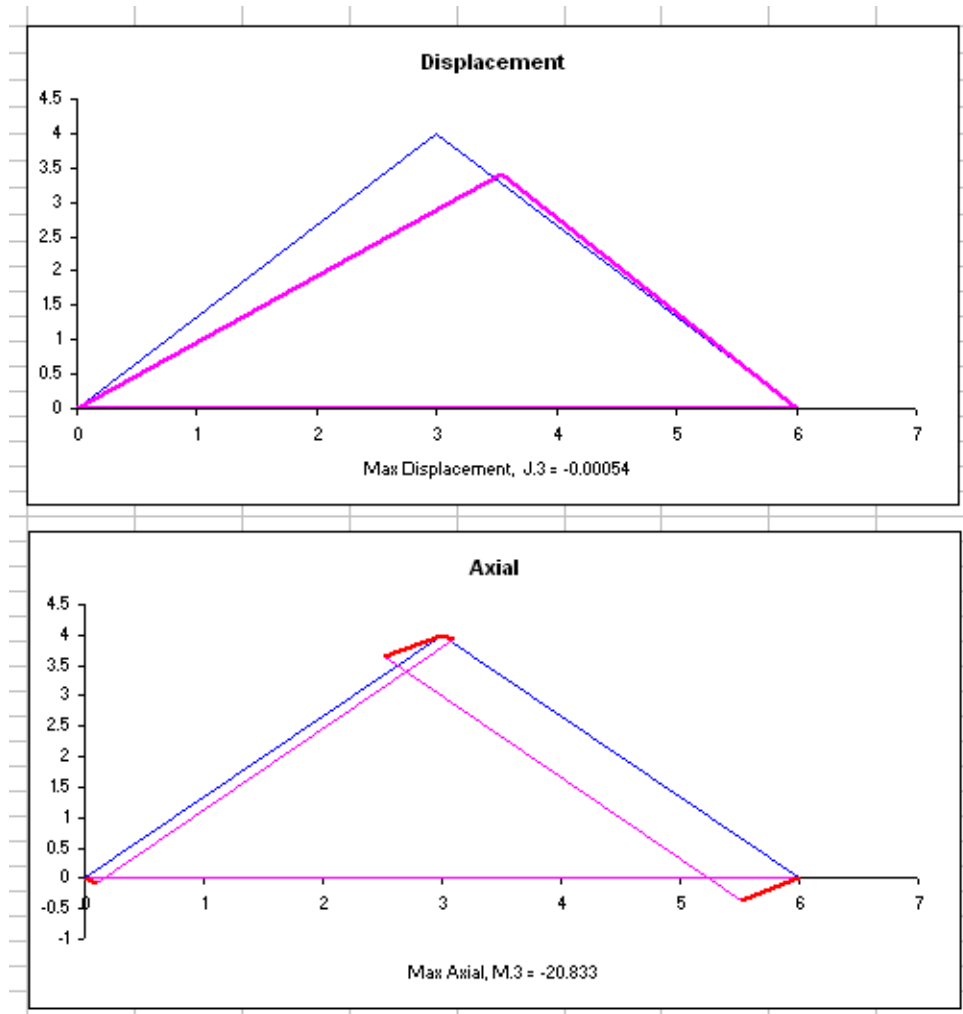


Figure 7.7: Charts of displacements and diagram of axial forces

Note:

- Color coding for axial forces is the same as used in FRAME2D. Lines - which are lines that perpendicular to long section - of member 1 and 3 are red; it means negative axial forces occurred on members (compression).

7.2 APPLICATION

Now, TRUSS2D will be used to analyze a truss bridge with geometry as shown in Figure 7.8. Given number of bridge joints = 8, number of members = 12, which is supported by a pin and a roll support. Each member of truss is made of the same linear elastic material with $E = 2,100,000 \text{ kg/cm}^2$ and cross-sectional area $A = 132 \text{ cm}^2$. The loading data is given as follows:

- Vertical load = 40 tons and horizontal load = 2 tons, at the joint as shown in Figure 7.8.
- Selfweight of all the members.

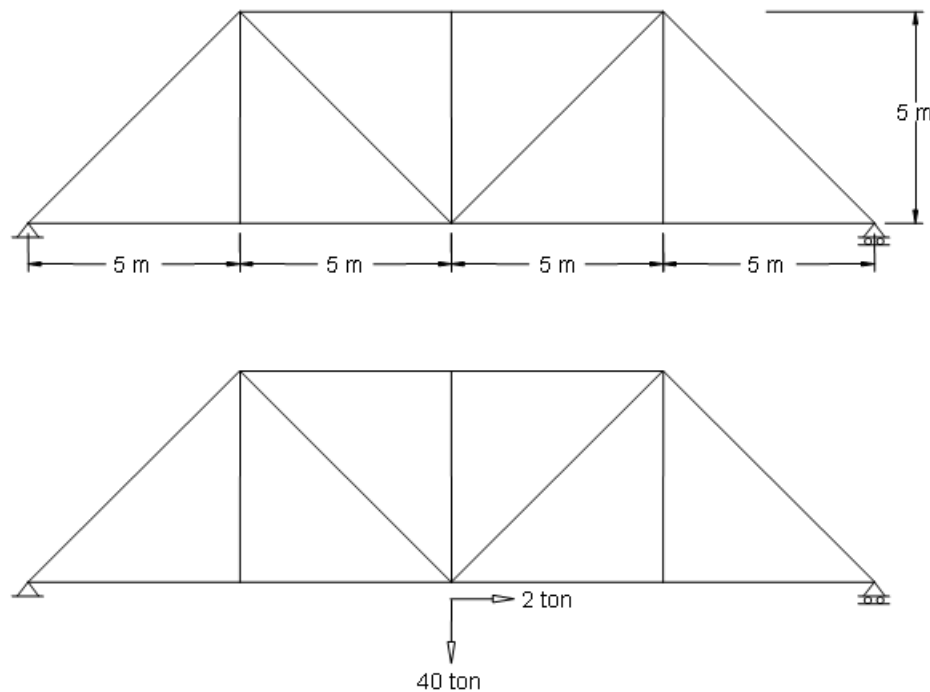
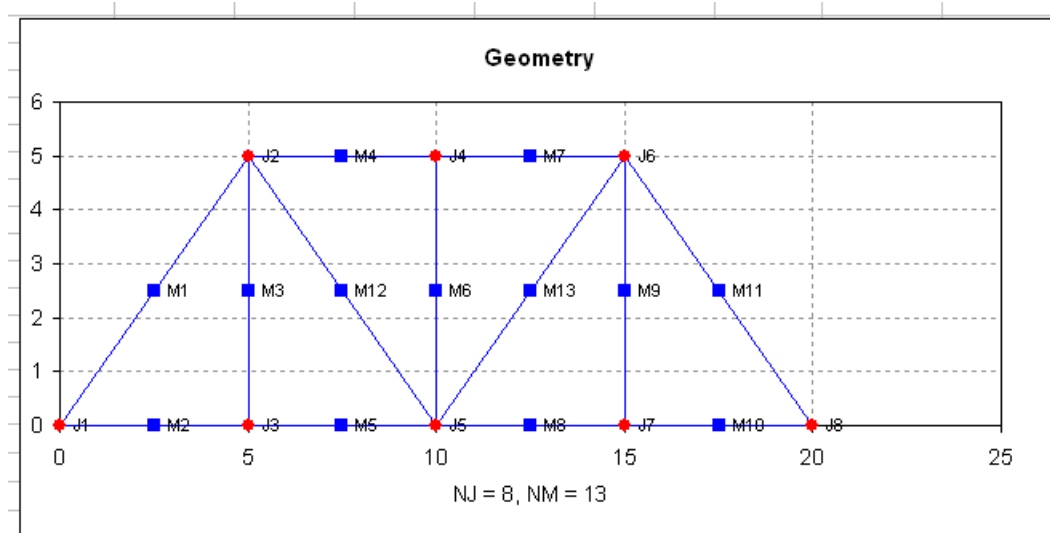


Figure 7.8: A truss bridge structure

The data of geometry, material and loading are then inputted in the input-output form of TRUSS2D as shown in Figure 7.9. Numbering system of a given truss bridge can be seen in the chart window (only available in Excel 2003) or in the worksheet shown as below:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	TRUSS2D v.1.0 (Program for 2D TRUSS Analysis)													
2														
3	Title =	Bridge1												
4	NJ =	8												
5	NM =	13												
6														
7	INPUT													
8	Joint Coordinates:													
9	Joint No.	1	2	3	4	5	6	7	8					
10	X_coord	0.0	5.0	5.0	10.0	10.0	15.0	15.0	20.0					
11	Y_coord	0.0	5.0	0.0	5.0	0.0	5.0	0.0	0.0					
12														
13	Element Information:													
14	No.	1	2	3	4	5	6	7	8	9	10	11	12	13
15	J 1st	1	1	2	2	3	4	4	5	6	7	6	2	5
16	J 2nd	2	3	3	4	5	5	6	7	7	8	8	5	6
17	Density	7.83	7.83	7.83	7.83	7.83	7.83	7.83	7.83	7.83	7.83	7.83	7.83	7.83
18	A Sec	0.0132	0.0132	0.0132	0.0132	0.0132	0.0132	0.0132	0.0132	0.0132	0.0132	0.0132	0.0132	0.0132
19	E	2.10E+07	2.10E+07	2.10E+07	2.10E+07	2.10E+07	2.10E+07	2.10E+07	2.10E+07	2.10E+07	2.10E+07	2.10E+07	2.10E+07	2.10E+07
20														
21	Support Condition:													
22	No.	1	8											
23	R_x	1	0											
24	R_y	1	1											
25														
26	Joint Load													
27	Joint	5												
28	Px	2.00												
29	Py	-40.00												
30														
31														
32	OUTPUT													
33														
34	Joint Load - Displacement					Member Forces		Support Reaction						
35	Joint	Px	Py	Dx	Dy	Memo	F Axial	Joint	Rx	Ry				
36	1	0.000	-0.624	0.00000	0.00000	1.7 / 1	33.016	1	-2.000	23.767				
37	2	0.000	-1.248	0.00169	-0.00287	2.5 / 0	-25.163	8	0.000	23.767				
38	3	0.000	-0.775	0.00045	-0.00286	3.5 / 0	-1.034							
39	4	0.000	-0.775	0.00089	-0.00474	4.5 / 0	44.304							
40	5	2.000	-41.506	0.00091	-0.00473	5.5 / 0	-25.163							
41	6	0.000	-1.248	0.00009	-0.00283	6.5 / 0	0.517							
42	7	0.000	-0.775	0.00133	-0.00266	7.5 / 0	44.304							
43	8	0.000	-0.624	0.00174	0.00000	8.5 / 0	-23.163							
44							9.5 / 0	-1.034						
45							10.5 / 0	-23.163						
46							11.7 / 1	32.500						
47							12.7 / 1	-30.156						
48							13.7 / 1	-29.639						
49														

Figure 7.9: Input-output form of TRUSS2D for given truss bridge

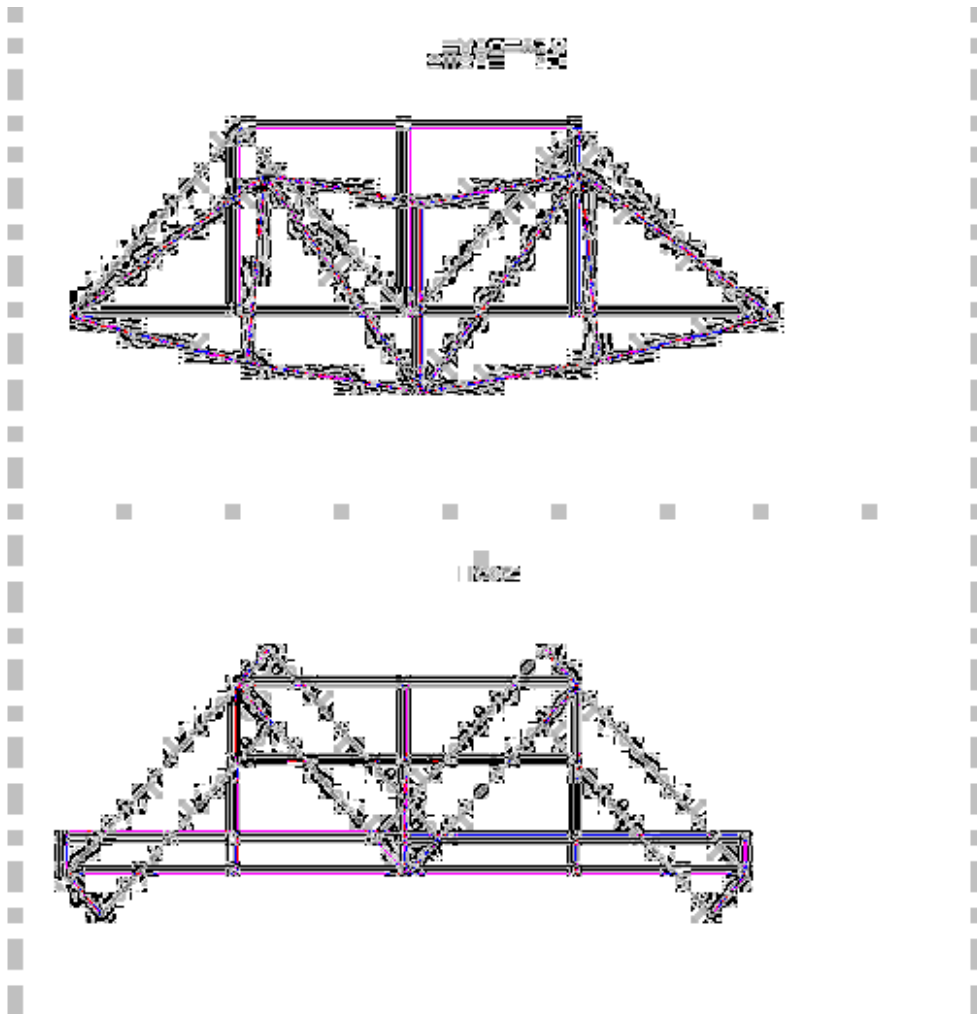
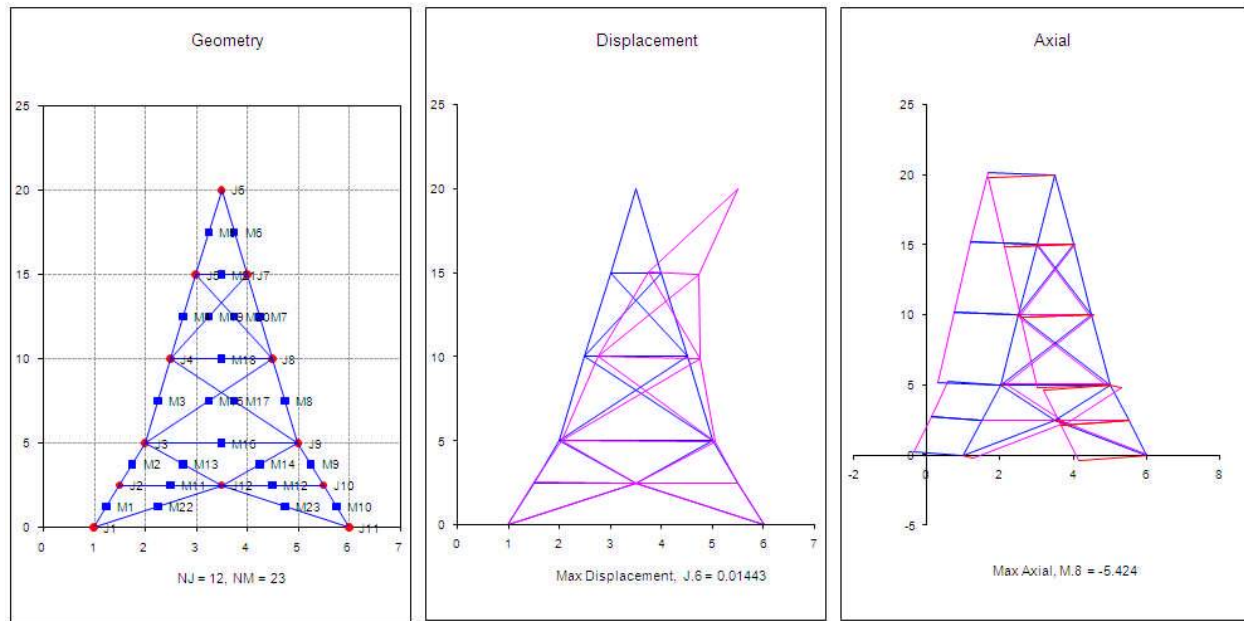


Figure 7.10: Charts of displacements and diagram of axial forces for given truss bridge

For a kind of truss, for instance, a truss tower that requires extended upward lengthwise chart, you can change a chart aspect ratio by resizing it. To do this, take the following steps:

- Click mouse on chart > move pointer on chart border > drag the sizing handles that located in the border to the size, e.g. like a chart size as shown below.
- The other way is to click mouse on chart > **Format** tab > in the **Size** group > enter the value in **Shape Height** and **Shape Width**.
- If font sizes change during chart resizing, click on the chart or at any text and then set the font size through **Font Size** setting on the **Ribbon**.



CHAPTER 8

BEAM ON ELASTIC FOUNDATION

Beam loaded on elastic foundation (BOF) is a theory that is often used in force - displacement prediction for the structural analysis that interacts directly with the ground. Elastic foundation is an approach to soil behavior as a collection of linear elastic springs, independent and one with another unrelated (Winkler soil). This theory is more solved by numerical methods, especially by finite element method (FEM).

The fundamental of FEM for BOF used herein are Equation 4.13 until 4.18, and the notations are used to express the relationship of external force-displacement components to those at nodes and internal force-deformation components on elements of finite element in Figure 8.1. Elastic foundation is represented by a series of spring constant at element nodes.

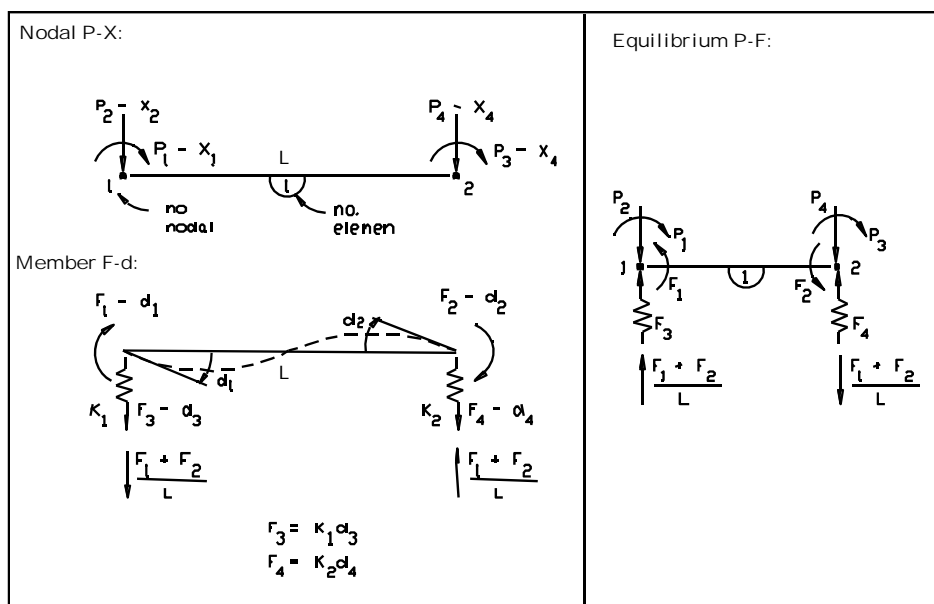


Figure 8.1: Force-displacement components at nodes and internal force-deformation components on elements of finite element

The relationship between load $\{P\}$ and internal forces $\{F\}$ is expressed as $\{P\} = [A] \{F\}$, where:

$$[A] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{L} & \frac{1}{L} & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{1}{L} & -\frac{1}{L} & 0 & 1 \end{bmatrix}$$

The relationship between internal forces {F} and element deformations {d} is expressed as {F} = [S] {d}, and from elastic behaviour of the beam we may write:



Notes:

- E modulus of elasticity of element (beam)
- I moment of inertia of element
- K_1, K_2 soil spring constant = $(L/2).B.K_s$
- B width of beam
- K_s modulus of subgrade reaction.

Foundation displacements {X} and internal forces {F} can be obtained using Equation 4.16 and 4.18:

$$\begin{Bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{Bmatrix} = \{[A][S][A]^T\}^{-1} \begin{Bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{Bmatrix} \begin{matrix} M \\ V \\ M \\ V \end{matrix}$$

$$\begin{Bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{Bmatrix} \begin{matrix} M \\ M \\ V \\ V \end{matrix} = [S][A]^T \begin{Bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{Bmatrix}$$

Notes:

- $\{[A][S][A]^T\}^{-1}$ inverse of global stiffness matrix
- $[S][A]^T$ element matrix
- M moment component
- V vertical force component
- F_1, F_2 magnitude of moment at element ends

Furthermore, in developing the program it is convenient to give boundary condition that restricts displacement by means of giving a fixed support at selected node. The magnitude of displacement at this node is = 0.

The advantage of FEM over classical solution of BOF is that properties from element and soils that are not constant such as weights, dimensions, flexural rigidities of beam and soil K_s along the beam can be easily incorporated into the calculation.

8.1 CASE EXAMPLE

An application program for beam on elastic foundation as described above is named BOF, which is used the same programming structure as previous TRUSS2D or FRAME2D. This program is now used for a footing analysis above ground with soil modulus K_s . The footing geometry and loading condition are as shown in Figure 8.2.

Given example:

Modulus of elasticity of the footing, $E = 217000 \text{ ton/m}^2$

Height x width x length of the footing = $0.3 \times 2 \times 2 \text{ m}$

$K_s = 1800 \text{ ton/m}^3$

Vertical load = 20 tons and moment load = 4 ton.m, acting in the column axis, right in the center of the footing.

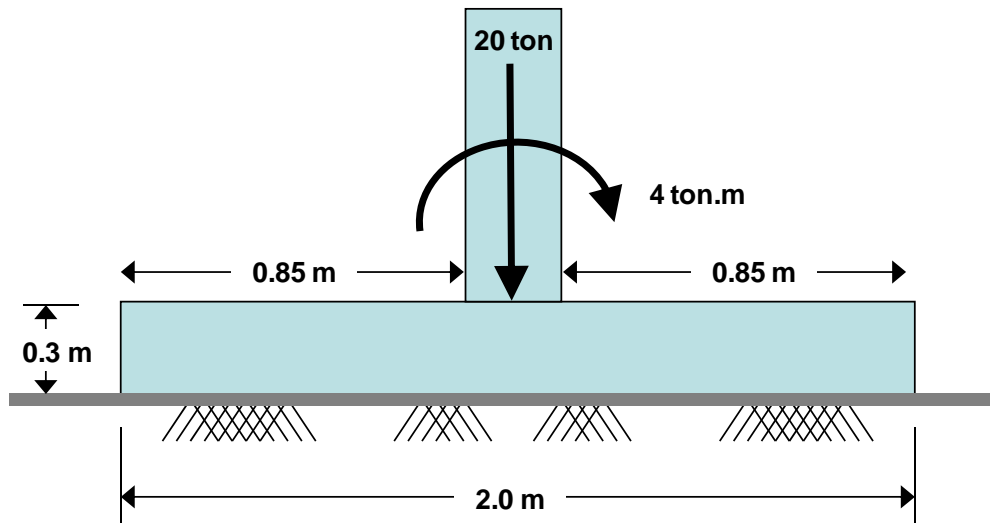
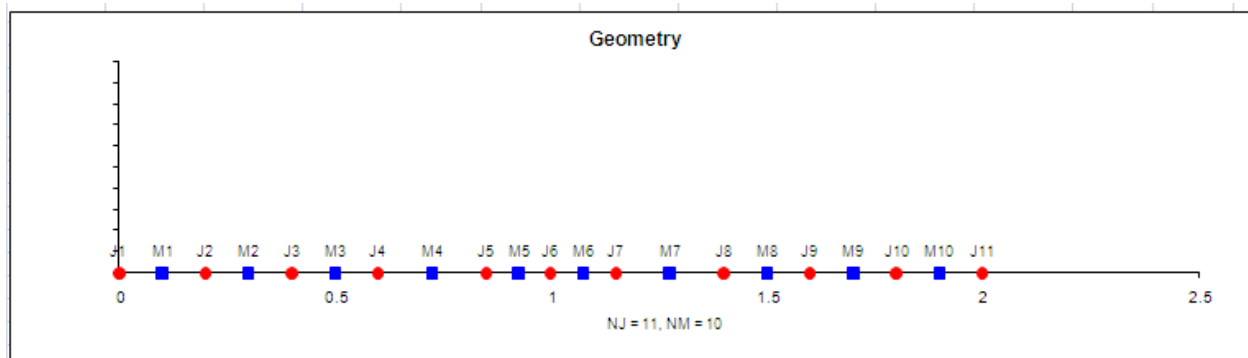


Figure 8.2: Footing on elastic foundation analysis

The input and output data of the given example is presented in the input-output form of BOF in Figure 8.3. The footing structure consists of 10 members, with a continuous numbering system as shown in the following worksheet chart:



	A	B	C	D	E	F	G	H	I	J	K	L			
1	BOF v.1.0 (Program for Beam on Elastic Foundation)														
2															
3	Title =	Footing1													
4	NJ =	11													
5	NM =	10													
6	E =	2.17E+06													
7	INPUT														
8	Joint Data:														
9	Joint No.	1	2	3	4	5	6	7	8	9	10	11			
10	X_coord	0.00	0.20	0.40	0.60	0.85	1.00	1.15	1.40	1.60	1.80	2.00			
11	Y_coord	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00			
12	Ks*	1,800.0	1,800.0	1,800.0	1,800.0	1,800.0	1,800.0	1,800.0	1,800.0	1,800.0	1,800.0	1,800.0			
13															
14	Element Data:														
15	No.	1	2	3	4	5	6	7	8	9	10				
16	J 1st	1	2	3	4	5	6	7	8	9	10				
17	J 2nd	2	3	4	5	6	7	8	9	10	11				
18	t=	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30				
19	B =	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00				
20	Pressure**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00				
21															
22	Support Condition:														
23	No.														
24	Rotation														
25	Translation														
26															
27	Joint Load:														
28	Joint	6													
29	Moment	2.00													
30	Vertical	20.00													
31	Note:	* Modulus of subgrade reaction (unit FL^3)													
32		**Perpendicular to elemen axis (unit FL^2)													
33															
34	OUTPUT	n Iteration	0												
35		n K's > 0	11												
36															
37	Joint Load - Displacement - Pressure														
38	Joint	Mj	Pj	Rotation	Displacement	Soil Pres	No./Length	K1	K2	Moment (F L)	Soil/Spring Reaction (F)	Shear (F)			
39	1	0.000	0.000	0.00082	0.00182	2.272	1.0/20	720.000	360.000	0.000	-0.262	1.309	0.713	-1.309	1.309
39	2	0.000	0.000	0.00081	0.00198	3.566	2.0/20	360.000	360.000	0.262	-0.809	0.713	0.771	-2.735	2.735
40	3	0.000	0.000	0.00080	0.00214	3.857	3.0/20	360.000	360.000	0.809	-1.664	0.771	0.928	-4.278	4.278
41	4	0.000	0.000	0.00078	0.00230	4.142	4.0/25	450.000	450.000	1.664	-3.200	1.036	1.120	-6.142	6.142
42	5	0.000	0.000	0.00072	0.00249	4.480	5.0/15	270.000	270.000	3.200	-4.390	0.672	0.700	-7.934	7.934
43	6	2.000	20.000	0.00066	0.00259	4.665	6.0/15	270.000	270.000	6.390	-4.790	0.700	0.725	-10.667	10.667
44	7	0.000	0.000	0.00057	0.00268	4.830	7.0/25	450.000	450.000	4.790	-2.606	1.208	1.266	-8.734	8.734
45	8	0.000	0.000	0.00048	0.00281	5.064	8.0/20	360.000	360.000	2.606	-1.315	1.013	1.046	-6.456	6.456
46	9	0.000	0.000	0.00044	0.00290	5.228	9.0/20	360.000	360.000	1.315	-0.442	1.046	1.076	-4.365	4.365
47	10	0.000	0.000	0.00042	0.00299	5.381	10.0/20	360.000	720.000	0.442	0.000	1.076	2.212	-2.212	2.212
48	11	0.000	0.000	0.00041	0.00307	5.530									
49															
50															

Figure 8.3: Input-output form of BOF for footing example

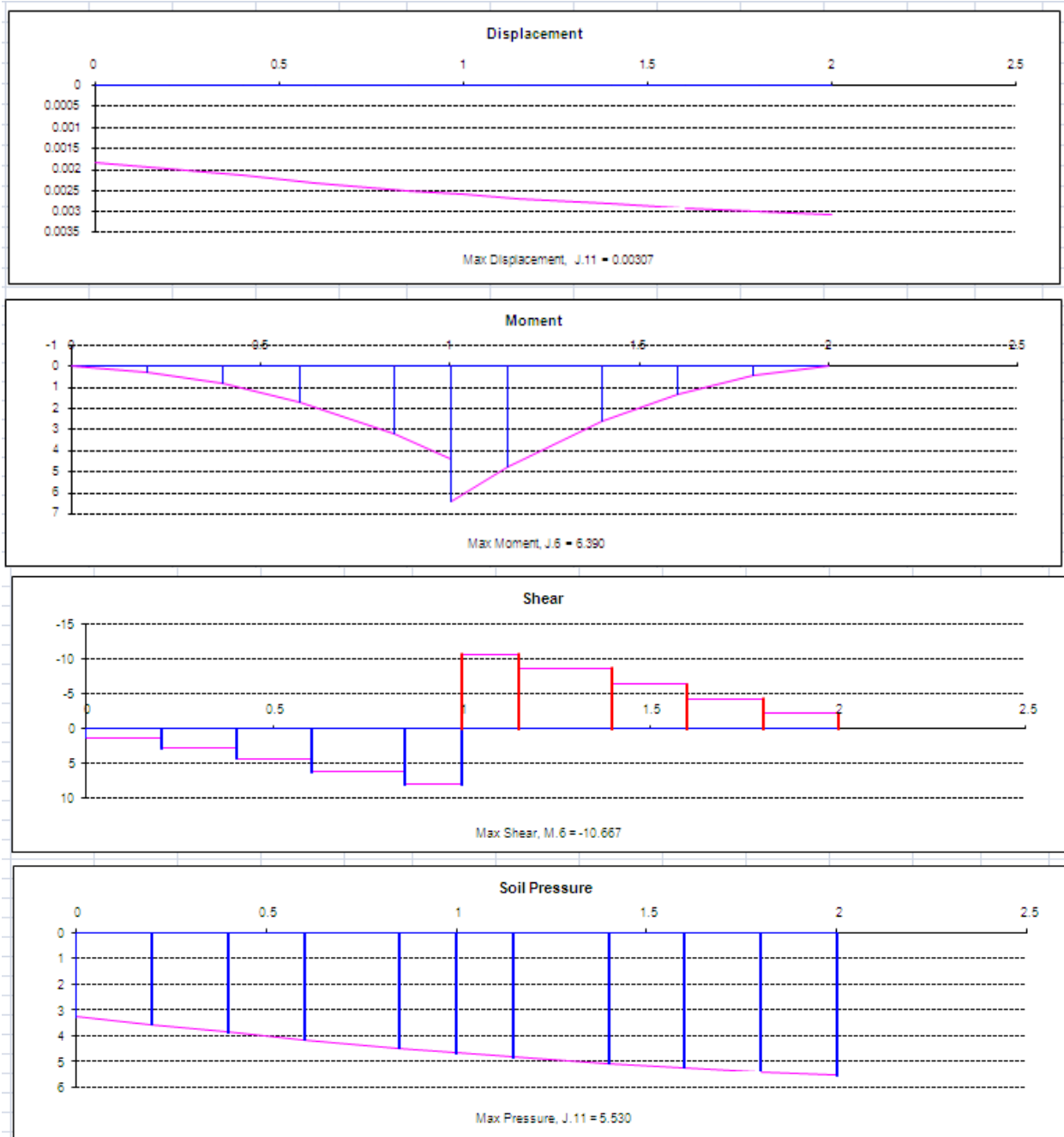


Figure 8.4: Charts of displacements and diagram of moment shear and soil pressure

Notes:

- In the input-output program, in Join - Displacement - Pressure column contains output data of soil pressures. In practice, soil pressures result is to be checked whether the given loading on the foundation does not exceed the allowable bearing capacity of the soil (q_a). Thus the maximum value should not exceed q_a .
- Unlike FRAME2D and TRUSS2D, there is iterative process in BOF to check whether there are nodes that tend to separate from the soil ($x < 0$), because the soil spring only accept pressure. Therefore ASAT matrix needs to be rebuilt with $K_s = 0$ at those nodes. Here, foundation selfweight shall be incorporated into the calculation. The result is then checked and if necessary, repeats this step until convergence is achieved, where,

Convergence \rightarrow number $As(n) = \text{number } As(n-1)$, at iteration n^{th} .

As is active spring at X positive.

- For a case of retaining wall or pile foundation where the soil spring is modeled on both sides of the structure, the soil spring would always work where the one will be active and the other will be non-active. Thus, it is convenient to give an option to select limits on nodes where iteration will be performed. The intended limits can be found in Chapter 8, where BOF is modified for vertical structures.
- Sign convention of moments and shear forces for graphic depiction is to follow the sign convention described in Chapter 6.2, which is based on the changes of structure shape after loading. Also, blue line in the curve is for positive direction and red line for a negative direction.

8.1 APPLICATION

The following example is just a program verification on a beam problem with uniform load as seen in Figure 8.5, and the results are then compared with the output of FRAME2D for the same problem. Both programs should provide the same result because they use the same analysis method and theory of structure as well.

Here, no soil property is involved so that $K_s = 0$. There are 3 supports of the beam, which is fixed at point A ($R = 1$, $T = 1$), pinned at point B ($R = 0$, $T = 1$) and fixed at point C ($R = 1$, $T = 1$).

The given data as below:

Beam:

Length = 18 m

Width = 2 m

Height = 0.5 m

Load:

Uniform load = $0.6 \text{ ton/m'}/\text{m}$ width of the beam

The output of BOF is presented in Figure 8.6.

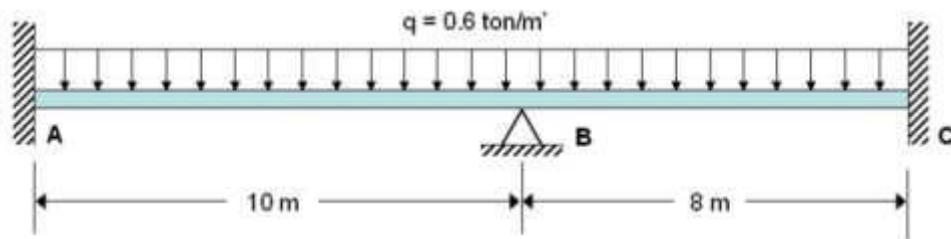


Figure 8.5: Beam with uniform load of q

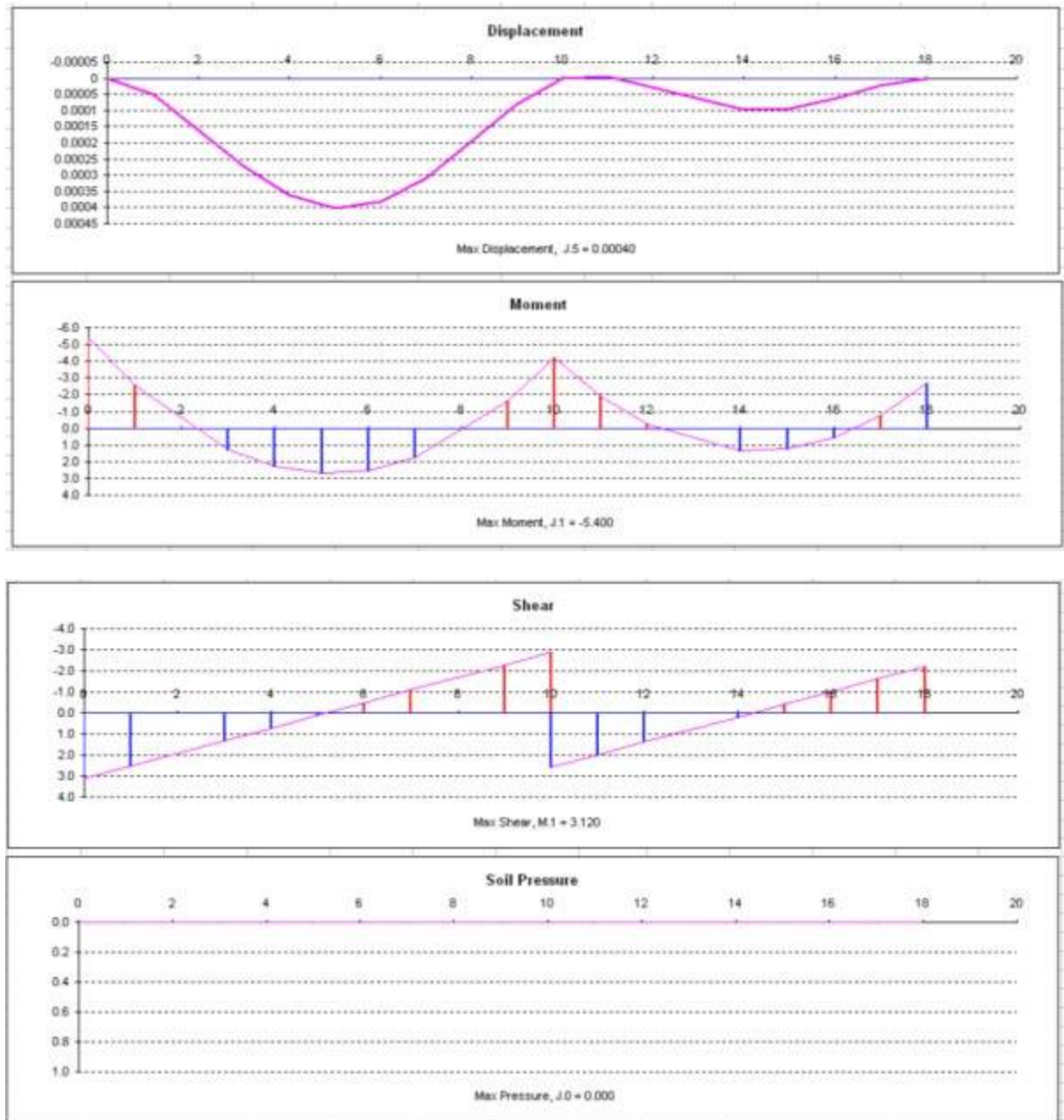


Figure 8.5: BOF solution for beam example

CHAPTER 9

LATERALLY LOADED STRUCTURE

Theory of beam on elastic foundation can also be applied for the analysis of structures subjected to lateral loads. Here, BOF program was slightly modified in which the orientation of the beam is now perpendicular to X-axis, and the external forces - displacement can be set whether lead to positive or negative axis. Load on nodes profile also to be displayed for a quick verification on inputted pressure magnitudes at both elements ends. The program for the laterally loaded structure is named XLAT.

9.1 CASE EXAMPLE

Given retaining wall example with geometry model and soil, pile and anchor data shown in Figure 9.1. The figure also shows finite element (FE) model and numbering system.

Find:

- Magnitude of displacements, distribution of moments and shear forces along the wall for design purpose.
- Forces and stress acting on anchor for design purpose.

After the FE model (Figure 9.1) is established, now is to find lateral pressure behind the wall acting from ground surface to the dredge line using active pressure coefficient from Coulomb or Rankine. You can either input the lateral pressure to 1 node below the dredge line.

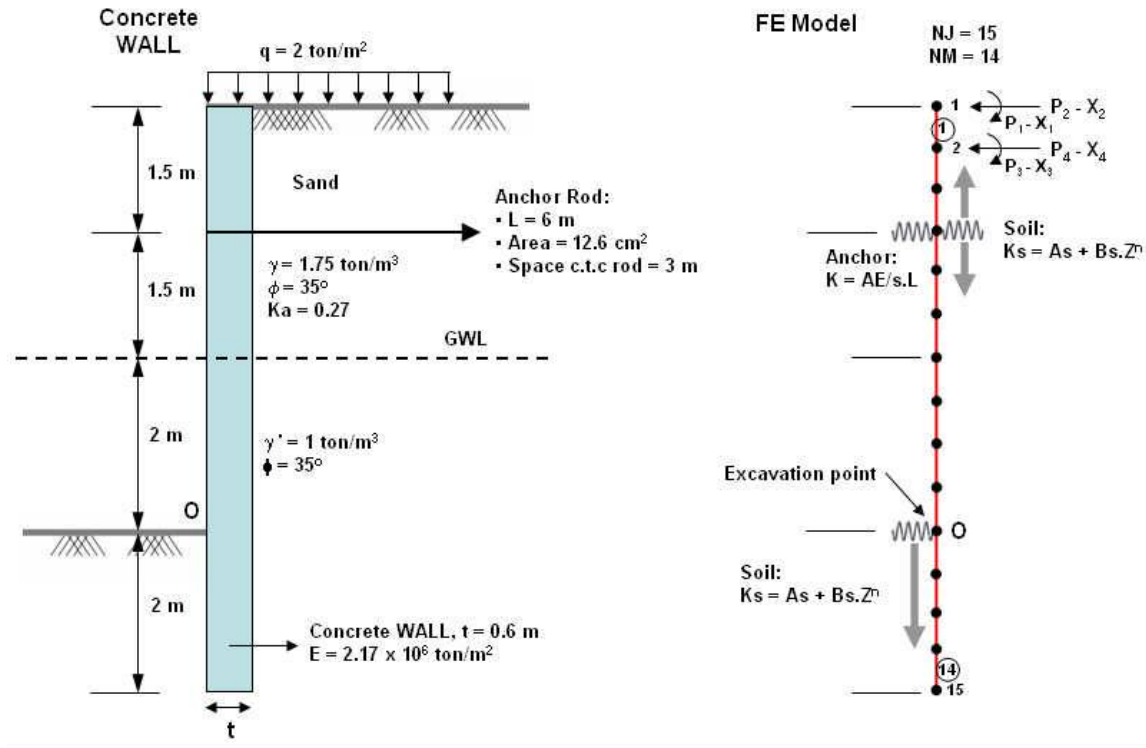
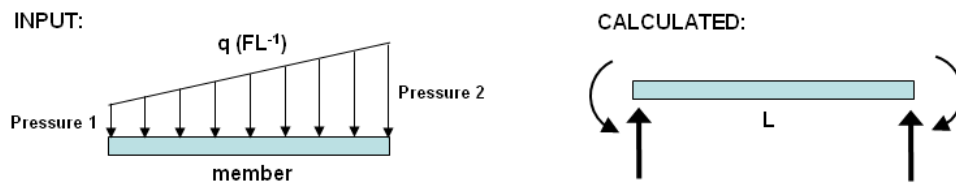


Figure 9.1: Model of retaining wall

Slight modifications were made in calculating fixed-end forces of element to take pressure profile that increases linearly with depth (triangular distribution) into account. See the description on this picture:



The code is as follows:

```
'member fixed-end forces due to pressure
For i = 1 To NM
```

```

With Member(i)
d = Abs(.Pres1 - .Pres2)
sq = Application.Min(.Pres1, .Pres2)
'rectangular part
w = sq * .bm
Pa(Idm(1, i), i) = -w * .Lh ^ 2 / 12 'moment
Pa(Idm(2, i), i) = -w * .Lh / 2 'horizontal
Pa(Idm(3, i), i) = w * .Lh ^ 2 / 12 'moment
Pa(Idm(4, i), i) = -w * .Lh / 2 'horizontal
'triangular part & summing
w = d * .bm
Pa(Idm(1, i), i) = Pa(Idm(1, i), i) + (-w * .Lh ^ 2 / 30) 'moment
Pa(Idm(2, i), i) = Pa(Idm(2, i), i) + (-w * .Lh * 3 / 20) 'horizontal
Pa(Idm(3, i), i) = Pa(Idm(3, i), i) + (w * .Lh ^ 2 / 20) 'moment
Pa(Idm(4, i), i) = Pa(Idm(4, i), i) + (-w * .Lh * 7 / 20) 'horizontal
End With
Next i

```

Equivalent load at node due to pressure on element:

```

'equivalent joint load due to pressure
For i = 1 To NM
  For j = 1 To 4
    For n = 1 To 4
      Peq(Idm(j, i), i) = -Pa(Idm(j, i), i)
    Next n
  Next j
Next i

'sum load = joint load + eq.load
For i = 1 To NP
  Ps(i) = Pj(i)
Next i

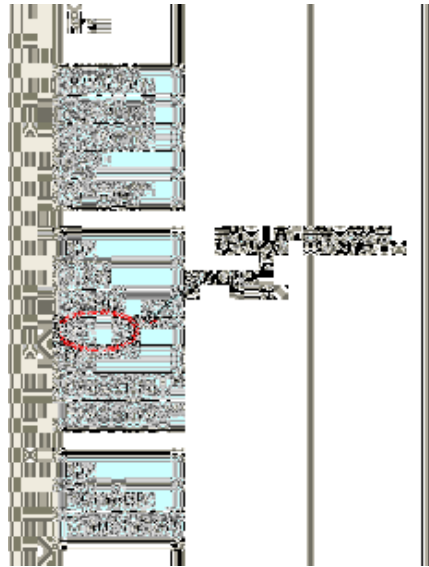
'load superposition
For i = 1 To NM
  For j = 1 To 4
    Ps(Idm(j, i)) = Ps(Idm(j, i)) + Peq(Idm(j, i), i)
  Next j
Next i

```

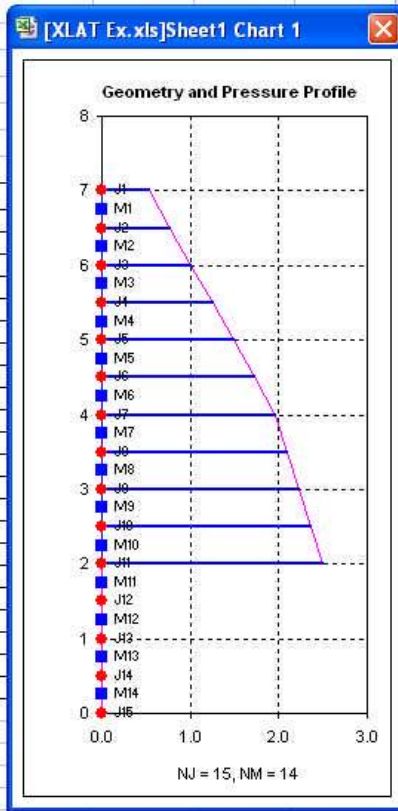
Notes:

- The above code is the same as used in the previous programs to find the equivalent load on nodes.
- Variable Idm states displacement index of elements ends. It has been described earlier in the previous chapters.

To generalize program where variation in the physical characteristics of the structure can be incorporated into calculation, thus the input of height of beam, t , in BOF is replaced by moment of inertia, I , in XLAT:



The following chart window (Excel 2003) will appear by clicking on Plot Geometry button. It shows finite element geometry and pressure profile that have been inputted into the program refer to Figure 9.1.



Modulus of subgrade reaction or spring constant of the soil (K_s) of sand layer in the front of the wall is assumed to increase with depth by the approach taken below:

$$\begin{aligned}
 k_s &= A_s + B_s \cdot Z^n \text{ ton/m}^3, \\
 &= 1000 + 2000 \cdot Z^n \rightarrow \text{above GWL} \\
 &= 600 + 1100 \cdot Z^n \rightarrow \text{below GWL}
 \end{aligned}$$

where,

$$A_s = 40(c \cdot N_c + 0.5 \gamma \cdot B \cdot N_\gamma)$$

$$B_s = 40(\gamma \cdot N_q)$$

z = depth from ground surface

N_c , N_γ , N_q = bearing capacity factors

$n = 0.5$, adopted exponent for non-linearity of K_s

Anchor rods are made of steel, with the specification shown in Figure 9.1. The value of rod anchor spring per meter width of the wall is obtained by the formula:

$$K = \frac{AE}{sL}$$

$$= \frac{12.6(2.0 \times 10^7)}{3(6)10^4} = 1400 \text{ ton/m.}$$

Input-output form of XLAT is shown in Figure 9.2 below:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	XLAT v.1.0 (Program for Laterally Loaded Structure)															
2																
3	Title = Wall															
4	NJ = 15 J excav = 11															
5	NW = 14															
6	E = 2.17E+06															
7	INPUT															
8	Joint Data															
9	Joint No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
10	X coord	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
11	Y coord	7.00	6.50	6.00	5.50	5.00	4.50	4.00	3.50	3.00	2.50	2.00	1.50	1.00	0.50	0.00
12	Ks*	1,000.0	2,414.2	3,000.0	3,449.5	3,828.4	4,162.3	4,464.1	2,657.9	2,800.0	2,933.5	3,059.7	3,179.7	3,294.4	3,404.5	3,510.3
13	Spring**				1,400.0											
14	Element Data															
15	No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	J 1st	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
17	J 2nd	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
18	I =	0.01800	0.01800	0.01800	0.01800	0.01800	0.01800	0.01800	0.01800	0.01800	0.01800	0.01800	0.01800	0.01800	0.01800	0.01800
19	B =	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
20	Pressure1*	0.54	0.78	1.01	1.25	1.49	1.72	1.96	2.09	2.23	2.36	2.50				
21	Pressure2*	0.78	1.01	1.25	1.49	1.72	1.96	2.09	2.23	2.36	2.50					
22	Support Condition:															
23	No.															
24	Rotation															
25	Translation															
26																
27	Joint Load															
28	Joint															
29	Moment															
30	Horizontal															
31	Note: * Modulus of subgrade reaction (unit FL^-3) *** Near end - far end. Perpendicular to elemen axis (unit FL^-2)															
32	** Joint spring, in X-axis direction (unit FL^-1)															
33																
34	OUTPUT															
35	n iteration = 1															
36	n kb = 0 = 5															
37	Joint Load - Displacement															
38	Joint	Moment	Horizontal	Rotation	Displacement	Soil Pres	No. Length	K1	K2	Moment (F.L)	Soil Spring Reaction (F)	Shear (F)	Support Reaction			
39	1	0.013	0.153	-0.00060	0.00436	0.000	1.0.50	0.000	0.000	0.000	0.077	0.000	0.000	0.153	-0.153	
40	2	0.004	0.389	-0.00060	0.00406	0.000	2.0.50	0.000	0.000	-0.077	0.349	0.000	0.000	0.541	-0.541	
41	3	0.004	0.506	-0.00060	0.00376	0.000	3.0.50	0.000	0.000	-0.349	0.873	0.000	0.000	1.047	-1.047	
42	4	0.004	0.624	-0.00059	0.00347	0.000	4.0.50	1400.000	0.000	-0.873	-0.716	4.852	0.000	-3.181	3.181	
43	5	0.004	0.743	-0.00059	0.00317	0.000	5.0.50	0.000	0.000	0.716	-1.934	0.000	0.000	-2.438	2.438	
44	6	0.004	0.861	-0.00060	0.00288	0.000	6.0.50	0.000	0.000	1.934	-2.722	0.000	0.000	-1.579	1.579	
45	7	0.003	0.971	-0.00063	0.00257	0.000	7.0.50	0.000	0.000	2.722	-3.025	0.000	0.000	-0.607	0.607	
46	8	0.002	1.046	-0.00067	0.00224	0.000	8.0.50	0.000	0.000	3.025	-2.804	0.000	0.000	0.440	-0.440	
47	9	0.002	1.114	-0.00071	0.00189	0.000	9.0.50	0.000	0.000	2.804	-2.027	0.000	0.000	1.553	-1.553	
48	10	0.002	1.181	-0.00074	0.00153	0.000	10.0.50	0.000	784.919	2.027	-0.659	0.000	0.883	2.735	-2.735	
49	11	-0.051	0.614	-0.00076	0.00115	3.534	11.0.50	784.919	794.832	0.659	0.132	0.883	0.615	1.582	-1.582	
50	12	0.000	0.000	-0.00076	0.00077	2.481	12.0.50	794.932	823.610	-0.132	0.308	0.615	0.324	0.352	-0.352	
51	13	0.000	0.000	-0.00076	0.00039	1.295	13.0.50	823.610	851.115	-0.308	0.160	0.324	0.012	-0.296	0.296	
52	14	0.000	0.000	-0.00076	0.00001	0.048	14.0.50	851.115	877.582	-0.160	0.000	0.012	-0.320	-0.320	0.320	
53	15	0.000	0.000	-0.00076	-0.00036	-1.278										

Figure 9.2: Input-output form of XLAT for retaining wall

Notes:

- Unit used in this example is ton - meter to equalize the values of calculation parameters used in this example.
- In the form, there is an input for joint excavation (point O) that is **J excavation**, as input, in which iterative process done from the top of the wall up to this point. Note that, iteration is not done below point O because the presence of soil spring on both sides of the wall. Definition of the iteration and its purpose are described in Chapter 8.

- To perform iterative process (to check the soil spring direction) up to joint n, enter **J excavation** = n + 1. In relation with this, iteration is not performed if **J excavation** = 1.
- To design the anchor rod, please check the reaction force in anchor and calculated the stress with steps below:

**Anchor
spring
reaction**

Member Output					
Moment (F.L)		Soil/Spring Reaction (F)		Shear (F)	
0.000	0.077	0.000	0.000	0.153	-0.153
-0.077	0.349	0.000	0.000	0.541	-0.541
-0.349	0.873	0.000	0.000	1.047	-1.047
-0.873	-0.716	4.852	0.000	-3.181	3.181
0.716	-1.934	0.000	0.000	-2.438	2.438
1.934	-2.722	0.000	0.000	-1.578	1.578
2.722	-3.025	0.000	0.000	-0.607	0.607
3.025	-2.804	0.000	0.000	0.440	-0.440
2.804	-2.027	0.000	0.000	1.553	-1.553
2.027	-0.659	0.000	0.883	2.735	-2.735
0.659	0.132	0.883	0.615	1.582	-1.582
-0.132	0.308	0.615	0.324	0.352	-0.352
-0.308	0.160	0.324	0.012	-0.296	0.296
-0.160	0.000	0.012	-0.320	-0.320	0.320

Check:

$$P_{\text{axial}} = 3 \times 4.852 = 14.557 \text{ ton (c.t.c = 3 m)}$$

$$f_s = \frac{P_{\text{axial}}}{A} = \frac{14.557}{12.6} = 1.155 \text{ ton/cm}^2$$

So, the specification chosen for the anchor rod must be adequate to resist the above force and stress.

The result of moments and shear forces are then made into diagram shown in Figure 9.3. For the wall design purposes, it can be taken maximum positive bending moment (blue line) working below anchorage, while negative moments (red line) is located around the anchor and around the wall tip.

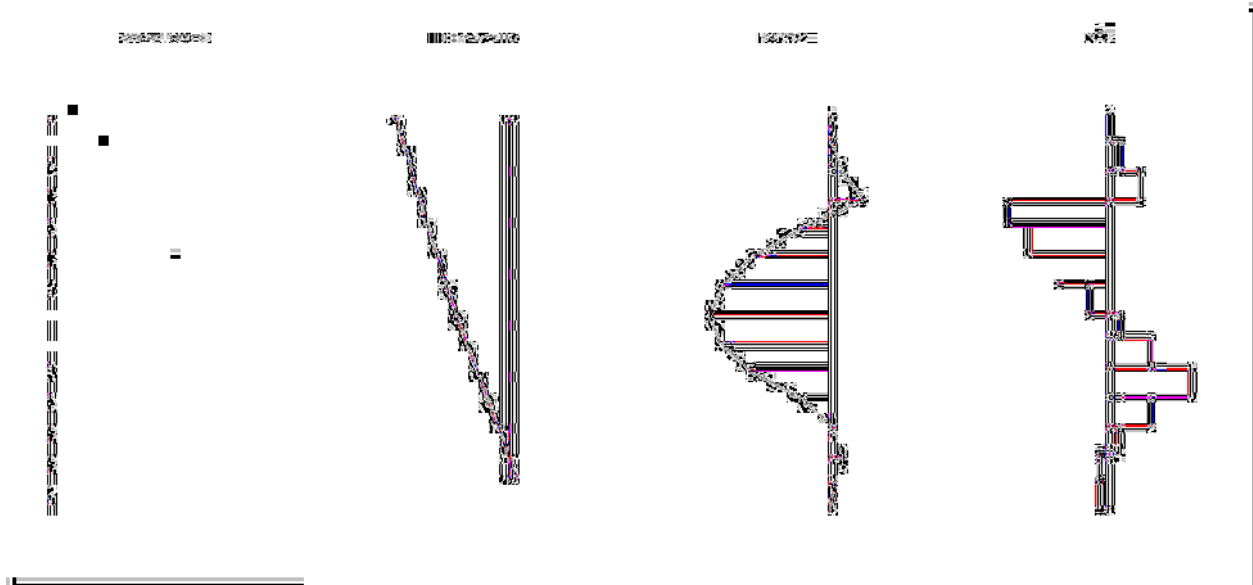


Figure 9.3: XLAT output chart

9.2 APPLICATION

Given pile foundation example with a diameter of 0.406 m embedded in sand layer to a depth of 16 meters. Pile and soil data are as shown in Figure 9.4. Pile head is assumed as fixed connection against rotation but free to translate.

Find:

- Maximum load applied on the pile head for its limited magnitude of displacement of the pile head.

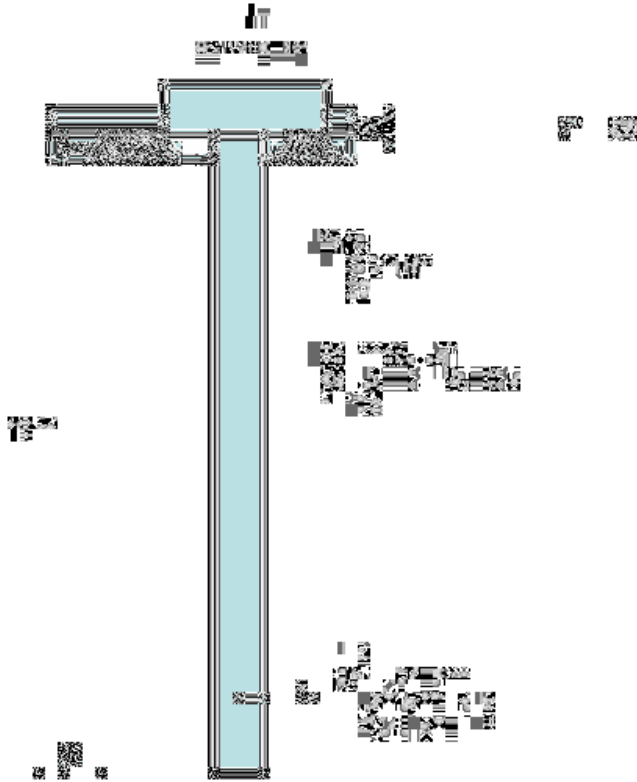


Figure 9.4: Pile foundation example

XLAT results of this example are shown in charts in Figure 9.5, where horizontal load, H , is adopted = 100 kPa. Displacement that occurs in the pile head is 0.0054 m (at joint 1) as shown in the Figure. From load and displacement linear relationship, any increase of load of 1 kPa will produce pile head displacement of 0.054 mm.

Thus, for a case where pile head displacement is limited to 5 mm, the allowed maximum load is then = $5/0.054$ kPa = 92.6 kPa.

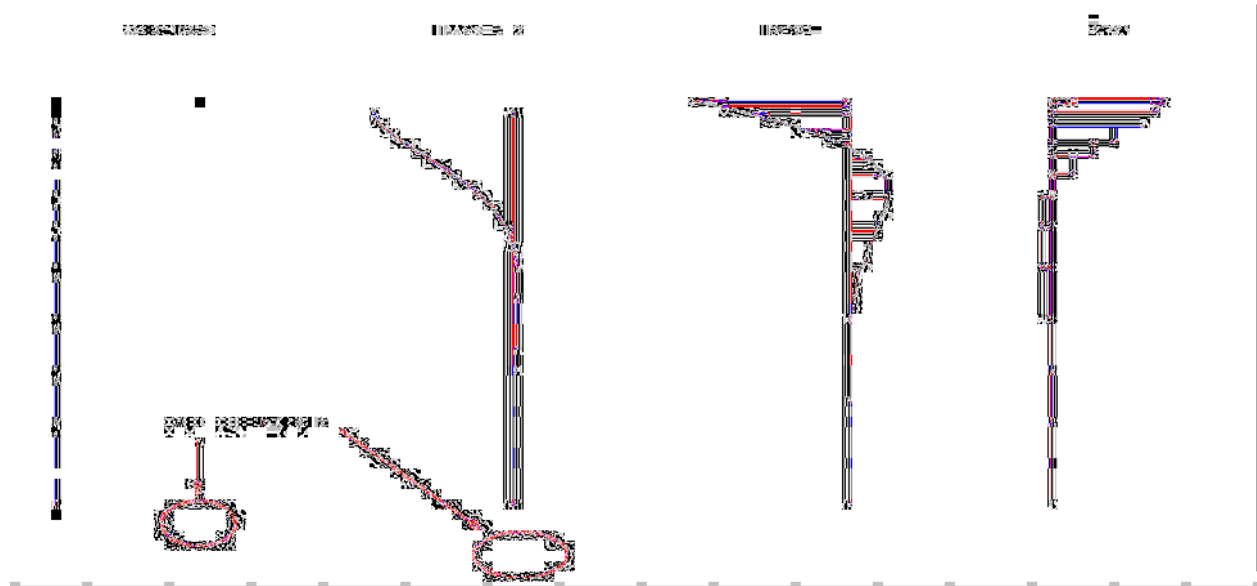


Figure 9.5: XLAT chart for pile foundation example

CHAPTER 10

ONE DIMENSIONAL CONSOLIDATION

In this chapter we will apply numerical solution using finite difference (FD) method for Terzaghi's theory of one dimensional consolidation. This method is based on a grid of time, t , versus depth, h , as shown in Figure 10.1. The depth of soil layer is divided into m equal parts of thickness Δz , while the period of time is divided to n equal parts of interval Δt .

The equation of one-dimensional consolidation according to Terzaghi's theory is:

$$\frac{\partial u}{\partial t} = c_v \frac{\partial^2 u}{\partial z^2} \quad (10.1)$$

Finite difference forms of Taylor series for the first and second order derivatives of functions have been shown in Chapter 5, and for both sides of Equation 10.1 can be written:

$$\frac{\partial u}{\partial t} = \frac{u_{i+1,j} - u_{i,j}}{\Delta t}$$

$$\frac{\partial^2 u}{\partial z^2} = \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta z^2}$$

Substituting these values into Equation 10.1 obtains finite difference approach to one-dimensional consolidation equation as follows:

$$u_{i,j+1} = u_{i,j} + \beta (u_{i-1,j} + u_{i+1,j} - 2u_{i,j}) \quad (10.2)$$

where,

$$\beta = \frac{c_v \Delta t}{(\Delta z)^2}$$

c_v coefficient of consolidation

$u_{i,j}$ excess pore pressure at depth i and time j .

β is referred to as a operator of Equation 10.2, and to make solution converge this value should not exceed $\frac{1}{2}$. The value lies between $\frac{1}{6}$ and $\frac{1}{2}$ (R.F. Craig, 1987).

For a period of time t in an open layer:

$$T_v = \frac{c_v(n\Delta t)}{(\frac{1}{2}m\Delta z)^2} = 4\beta n/m^2 \quad (10.3)$$

In case of half-closed layer, the denominator becomes $(m\Delta z)^2$:

$$T_v = \beta n/m^2 \quad (10.4)$$

T_v is a time factor for consolidation

In case of impermeable boundary, where no flow across the boundary:

$$\frac{\partial u}{\partial z} = 0, \text{ or in FD form: } \frac{u_{i-1,j} - u_{i+1,j}}{2\Delta z} = 0$$

For an impermeable boundary, Equation 10.2 becomes:

$$u_{i,j+1} = u_{i,j} + \beta(2u_{i-1,j} - 2u_{i,j}) \quad (10.5)$$

The degree of consolidation (U) at time t can then be obtained by finding area under initial isochrone (from initial excess pore pressure distribution) and area under isochrone at time t , given by the following equations:

half closed layer:

$$U = 1 - \frac{\int_0^d u \, dz}{\int_0^d u_i \, dz} \quad (10.6)$$

open layer:

$$U = 1 - \frac{\int_0^{2d} u \, dz}{\int_0^{2d} u_i \, dz} \quad (10.7)$$

where d is the thickness of clay layer. Isochrone is a curve of excess pore water pressure, u_e plotted against depth for certain t as shown in Figure 10.2. The blue line represents a variation of the initial u while the magenta line represents the pressure at time t .

Both Equation 10.2 and Equation 10.5 can be easily programmed, because the analysis is based on an equilaterally grid. This looks like to build an array of function values (u) with size of, for example, $m \times n$ (see Figure 10.1). The solution is simply taking a looping through grid points in an array; thus, to find of excess pore water pressure at a specified period of time (n) can be determined by giving specified value in a looping.

If m and n are referred to FD grid points as shown in Figure 10.1, thus, the code for half-closed layer can be written:

'Finite difference approximation:

```

For j = 0 To n
  For i = 1 To m - 1
    u(i, j + 1) = u(i, j) + Beta * (u(i - 1, j) + u(i + 1, j) - 2 * u(i, j))
  Next i
  'on impermeabel boundary (at m points):
  i = m: u(i, j + 1) = u(i, j) + Beta * (2 * u(i - 1, j) - 2 * u(i, j))
Next j

```

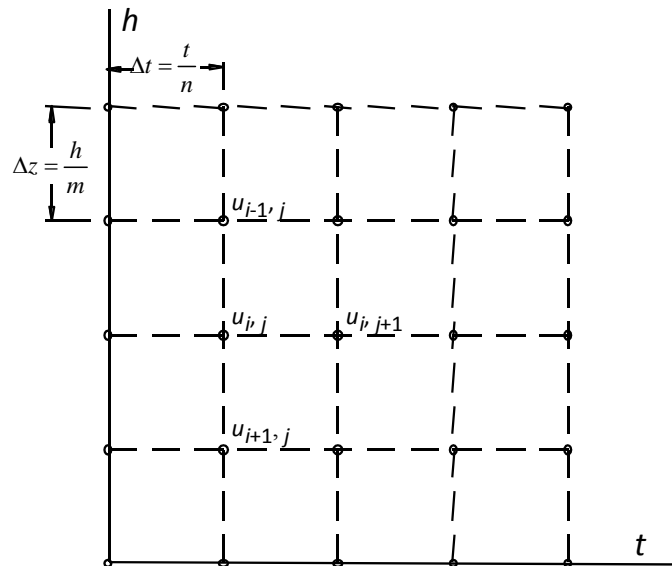


Figure 10.1: Grid of time versus depth

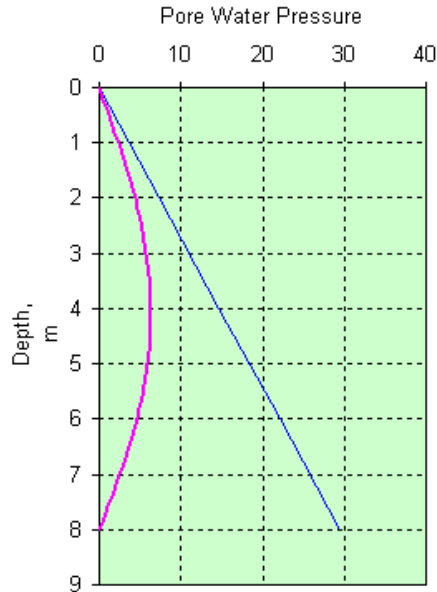


Figure 10.2: Isochrone example at specified time

In practice, the use of Taylor series adopts the first few terms omitting the higher order derivatives and the error due to truncate the series can be reduced to minimum when $\beta = 1/6$. Seeing this condition, n and m can be then directly determined to satisfy a fixed $\beta = 1/6$ for both layer conditions. If m is an input data, then Equation 10.3 and 10.4 can be re-written to determine n as follows:

```
im = Cells(4, 2) 'no. of layer
m = 10 * im
h = Cells(3, 2) 'height of layer
t = Cells(5, 5) 'time
cv = Cells(5, 2) 'coef. of consolidation
FDCCase = Cells(3, 5) 'Case of calculation
```

'open layered

```
Tv = cv * t / (h / 2) ^ 2
n = 1.5 * m ^ 2 * Tv
```

'half closed layered

```
Tv = cv * t / h ^ 2
n = 6 * m ^ 2 * Tv
```

In the process of finding U, the input data of depth and the initial u are automatically re-inputted by the program where is each multiplied by 10 (default by the Author) or $m = 10 \times im$ in the above code. For doing so, the interpolation technique is done to get the adjacent values. The discussion of numerical methods in Chapter 5 demonstrates that dividing interval of the independent variables into smaller intervals provides the result closer to the exact value, and it is similar to what we will be doing here in the process of finding U (numerical integration) as well as final u (numerical differential).

Manipulation for getting smaller FD grid interval is as follows:

- Code to get initial depth input, dz, and initial u, IP:

```
For i = 0 To im
    dZ(i) = i * h / im
    IP(i) = Cells(9 + i, 2)
Next i
```

- Re-inputted with a divisor 10 and perform interpolation to get adjacent values using rsdZ and RSIP variables:

```
n = 1
rsdZ(0) = dZ(0)
rsIP(0) = IP(0)
For i = 0 To im - 1
    For j = 1 To 10
        rsIP(j) = j * (IP(i + 1) - IP(i)) / 10
        rsdZ(j) = j * (dZ(i + 1) - dZ(i)) / 10
        rsIP(n) = rsIP(j) + IP(i)
        rsdZ(n) = rsdZ(j) + dZ(i)
        n = n + 1
    Next j
Next i
```

Running the code will result:

Initial data:

Depth (H/m)	Initial Pressure
0.00	0.00
2.00	7.50
4.00	15.00
6.00	22.50
8.00	30.00

Re-inputted:

Depth (H/m)	Initial Pressure
0.00	0.00
0.20	0.75
0.40	1.50
0.60	2.25
0.80	3.00
1.00	3.75
1.20	4.50
1.40	5.25
1.60	6.00
1.80	6.75
2.00	7.50
2.20	8.25
2.40	9.00
2.60	9.75
2.80	10.50
3.00	11.25
3.20	12.00
3.40	12.75
3.60	13.50
3.80	14.25
4.00	15.00
4.20	15.75

4.40	16.50
4.60	17.25
4.80	18.00
5.00	18.75
5.20	19.50
5.40	20.25
5.60	21.00
5.80	21.75
6.00	22.50
6.20	23.25
6.40	24.00
6.60	24.75
6.80	25.50
7.00	26.25
7.20	27.00
7.40	27.75
7.60	28.50
7.80	29.25
8.00	30.00

Figure 10.2 shows example of open layer isochrone at time t with initial u increases linearly with depth. The area under isochrone curve between $y = 0$ and $y = 8$ is then sought. For doing so, FDC uses trapezoidal method (in h/m unit) in which the total area is cumulative sum of h/m segment areas done in a looping over values of re-input dZ and IP . A detailed code can be found in FDC program in the Attachment.

10.1 APPLICATION 1

The profile of excess pore water pressure (u) over the depth in saturated clay layer from impact load is equal to the vertical stress distribution below foundation as in Problem 4 of Chapter 3, as shown in table below:

Depth (m)	u (ton/m ²)
0.0	10.00
0.5	9.76
1.0	8.63
1.5	7.01
2.0	5.49
2.5	4.28
3.0	3.36
3.5	2.68
4.0	2.17
4.5	1.79
5.0	1.49
5.5	1.26
6.0	1.08

Suppose the clay layer is half-closed layer, where water flows across the upper limit. Given $c_v = 7.19 \text{ m}^2/\text{year}$ and the thickness of layer is 6 m. Find the distribution of u and average degree of consolidation (U) at 6 months.

Answer

FDC result is shown as the following:

Depth (H/m)	Initial Pressure	After (t) Pressure	ave U
0.00	10.00	0.00	46%
0.50	9.76	0.55	
1.00	8.63	1.06	
1.50	7.01	1.52	
2.00	5.49	1.89	
2.50	4.28	2.17	
3.00	3.36	2.35	
3.50	2.68	2.46	

4.00	2.17	2.51	
4.50	1.79	2.51	
5.00	1.49	2.50	
5.50	1.26	2.48	
6.00	1.08	2.47	

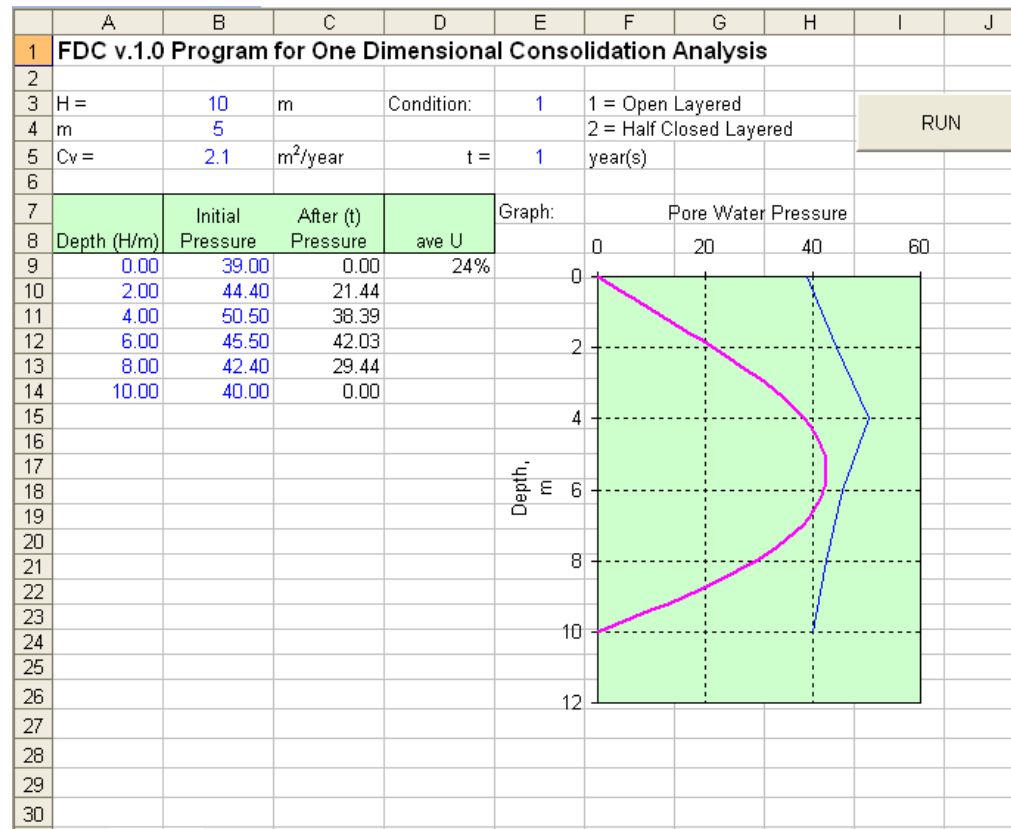
10.2 APPLICATION 2

From field piezometers measurement on the clay layer beneath embankment, the distribution of excess pore water pressure, u (after subtracted by its theoretical pressure) is shown in the table below:

Depth (m)	u (kN/m ²)
0.0	39.00
2.0	44.40
4.0	50.50
6.0	45.50
8.0	42.40
10.0	40.00

Answer

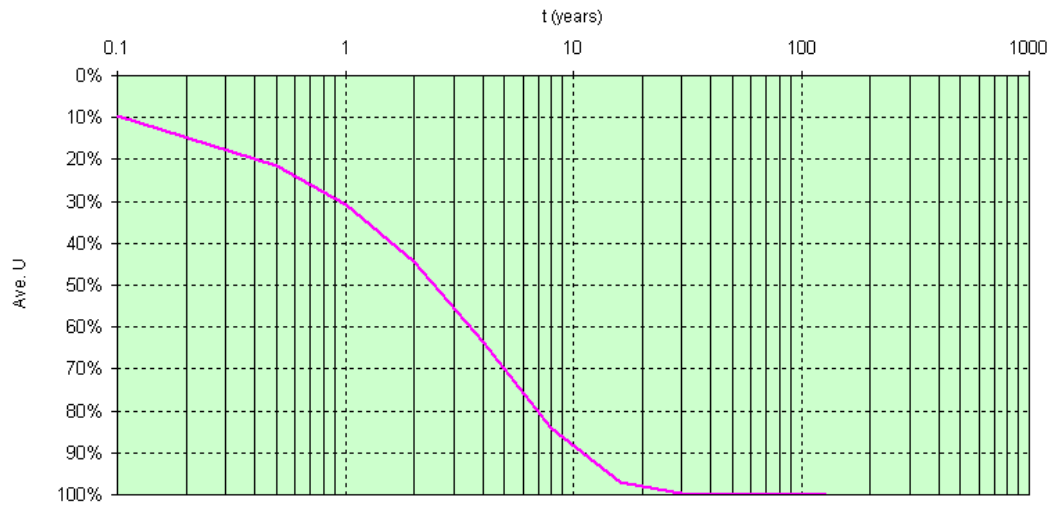
The input-output form FDC is shown below:



Notes:

- The advantage of finite difference method is that any pattern of excess pore water pressure can be incorporated into the calculation. The input data can be based on the initial field condition or load that works gradually, for examples, in the completion of the consolidation problem below dam or embankment of soil.
- The purpose of FDC programming for one-dimensional consolidation based on numerical methods has actually been achieved, that is at the stage where the degree of consolidation based on areas under isochrone at time t can be calculated. However, if you wish to go further with the solution, a curve of t versus U as shown below can also be presented. This curve is created automatically by the program to include 10 different t values into a looping, starts with $t = 0.1$ (years), then $t = 0.5, 1, 2$, and so on, and plotted on a semi-log chart.

	A	B	C	D	E	F	G	H	I	J	K	L	M
15		t (years)	0.1	0.5	1.0	2.0	4.0	8.0	16.0	32.0	64.0	128.0	
16		Ave. U	10%	21%	31%	44%	63%	84%	97%	100%	100%	100%	



CHAPTER 11

AUTOCAD SCRIPT FILE

To graphically depict outputs of a program, alternatively, you can plot the outputs in AutoCAD software instead of using Excel chart. Here, you can work further with charts drawing with either a 2D or 3D model. The main advantage of AutoCAD is because this software is specifically built for composing drawings.

Script file meant here is a text file that contains a sequence of commands executed by AutoCAD. The sequence of commands can be made on a worksheet or VBA to speed up and simplify the process of creating drawings. The script file is made by a word processor such as Win. Notepad or Microsoft Word in ASCII format and given with a .scr extension.

In this discussion, we will be working with AutoCAD 2007 version.

11.1 CREATING SCRIPTS IN WORKSHEET

Before creating a script, you should get familiar with AutoCAD **commands** and associated sequence of **entries** through **command prompt** (not remotely via menu button). For example, a command to create a line is **Line**, and followed by entries from coordinates of the first point, second and so on and ended by pressing **Enter**.

Example 1

The following is an example of steps sequence in AutoCAD to create a drawing of 3 lines segments using **Line** command through command prompt:

Command: line

Specify first point: 0,0

Specify next point or [Undo]: 1,1

Specify next point or [Undo]: 2,3

Specify next point or [Close/Undo]: 3,3

[Enter]

From the above steps, it can therefore be made the following text (script) replaces what was done above:

Line

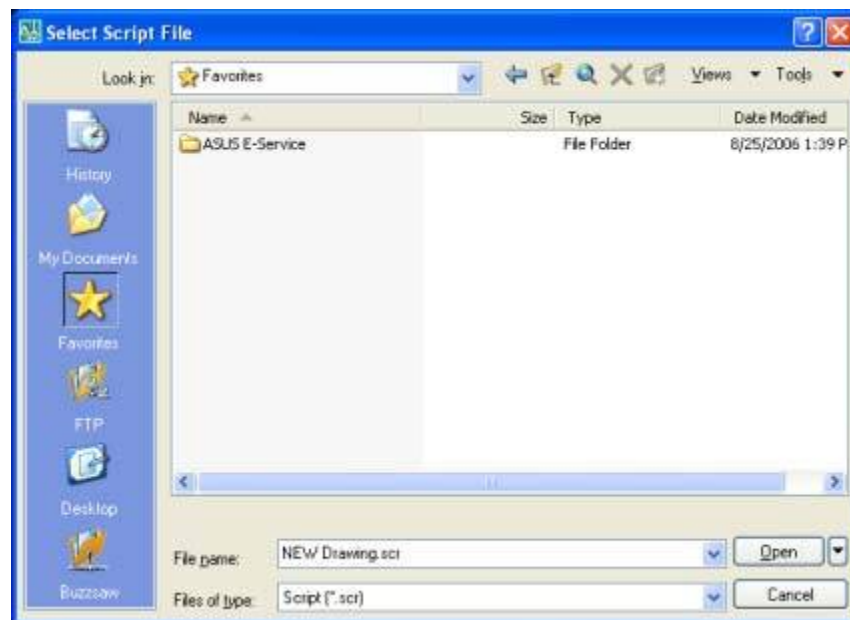
0,0

1,1

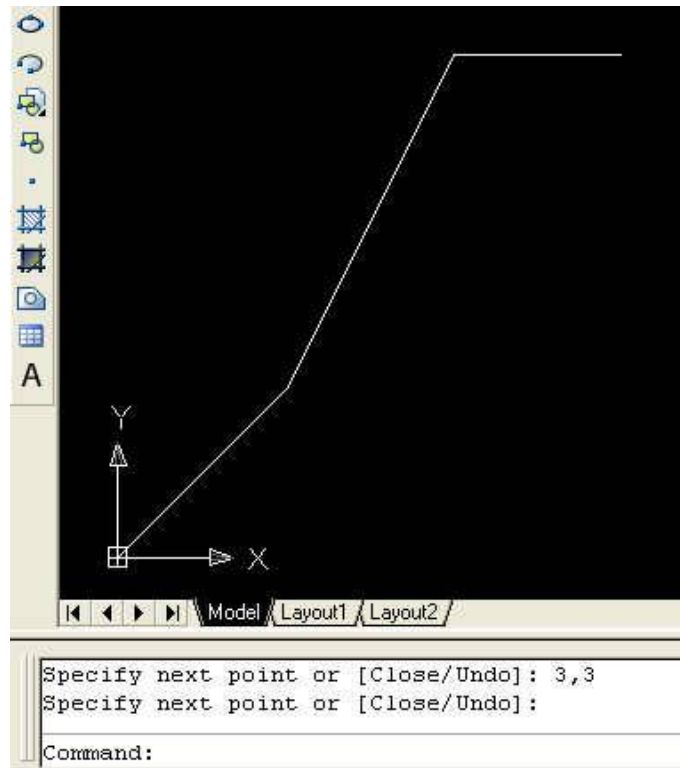
2,3

3,3 [type 2 blank spaces for an **Enter**, to end **Line** command]

Open Notepad program, type the script above, and do not forget to type 2 blank spaces after the coordinates of the last point to end the **Line** command. Afterward, save it with .scr extension, for example named **Line.scr**. Now, open your AutoCAD, type **scr** at the command prompt to display the **Select Script File** dialog box as below:



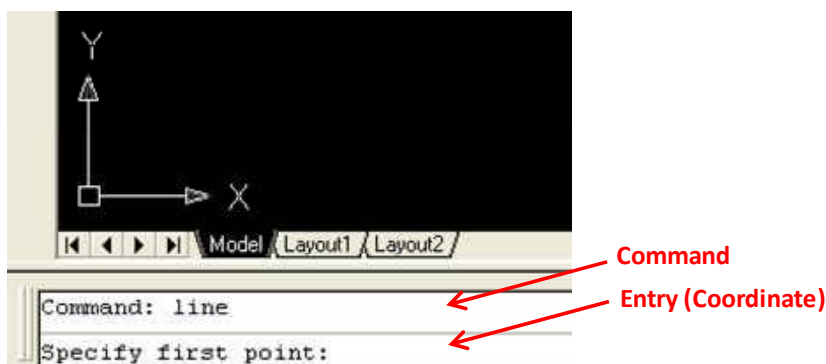
Select **Line.scr** file at the location you have placed before and click **Open**, then AutoCAD will execute all commands in script file. The result is shown as below:



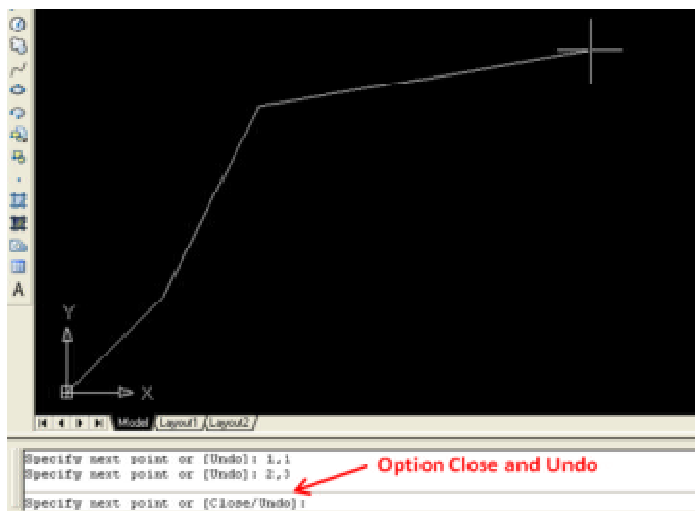
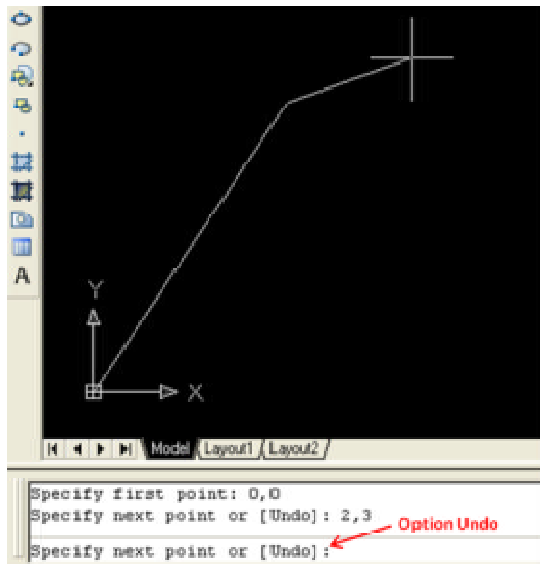
Notes:

- AutoCAD commands consist of prompt entries that include some options that must be entered in by the user. These options can also be 1). A command related to the main command and, 2). Attributes to an object, for example, properties of a line, a circle or a rectangle.
- The best way to understand a sequence of a command (e.g. **Line**) is firstly to trying out it in AutoCAD, then noted down all entries, including the requested options associated with the main command. This becomes the basis used for preparing the script later.

The following is **Line** command and the required prompt entries:



Options:

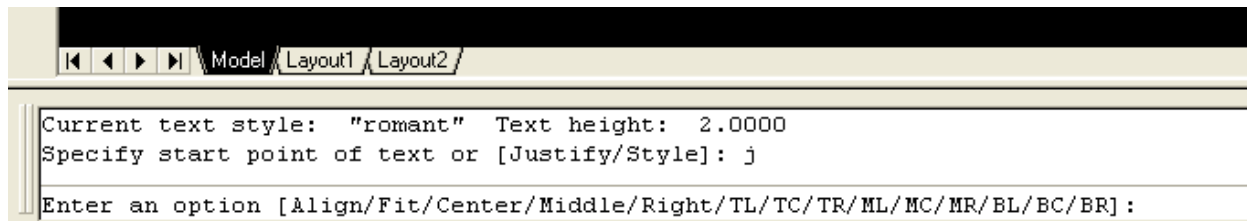


- Pressing **Enter** at the last line is to end the command, so lines are drawn up to point coordinates of the previous line. Pressing u is to do **Undo** command; undo the previous line segment, while c is to do **Close** command; close the last segment by connecting it to the first segment.
- Pressing **Enter** on keyboard is to end a command. In script, it is done by giving one blank space or create a blank line.

- Some AutoCAD commands display a dialog box to enter its data. To display prompt entries entered from keyboard (i.e. line by line through the command line), use a "-" before the command, such as: **-Style**, **-Block** or **-Layer**.
- To join two or more data between cells on a worksheet, use hyphen "&" (in quotation marks). It is the same as used in VBA to combine string and numerical data.
- Script can be extended vertically or horizontally. In the extended horizontally script, any data entered to next data must be separated by an empty space, including an empty space of pressing **Enter**.
- Every time you run a script file, object snap must be inactive or in off mode, so that it gives the actual results. Click the **Osnap** button to make the mode on or off.

The other examples of AutoCAD commands and prompt entries:

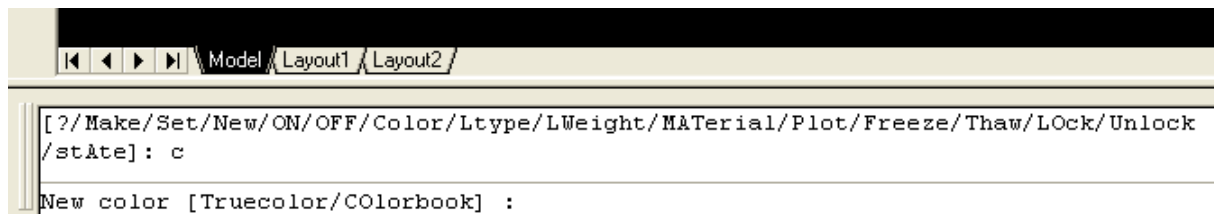
- Below are options in **Text** command. The letter "j" stands for **Justify** command, taken from the first letter.



- Options in **Layer** command.



- Prompt entry for **Color** properties (**Layer > Color**).



Example 1 shows how to draw a line using script written manually. For line depiction which comprises many segments (tens or hundreds), the manual method is rarely used. Creating AutoCAD script on a worksheet is a way to simplify and shorten the job of creating AutoCAD drawings, by relying on the Excel menu such as Copy, Paste, and its built-in functions.

Example 2

Example 1 is now made on the worksheet as follows:

	A	B	C	D
1				a space
2	X	Y		
3	0	0	line 0,0	= "line "&A3&" "&B3&" "
4	1	1	1,1	=A4&" "&B4&" "
5	2	3	2,3	} Result of copying cell from C4 to C5:C6
6	3	3	3,3	
7				

Prior to the first coordinates entry of (0,0) after the **Line** command it must be given 1 blank space to execute the command, the same is done by pressing **Enter** in AutoCAD. The second coordinates of (1,1) and so on, should be placed in different line that is on cell C4, C5, and so on, so as to be able to write the next script by copying the formula on cell C4. The last coordinates of (3,3) on cell C6, need to be added a blank space at the end of the line to end the **Line** command:

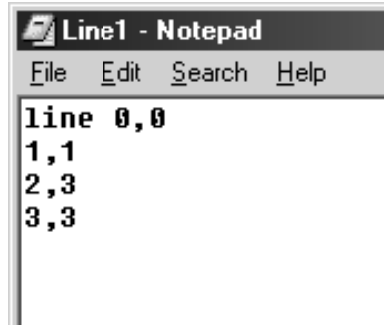
=A6&" "&B6&" "

Next is to create a script file with the following steps:

1. Select a range C3 to C6, then **Copy** by clicking the Copy icon on the toolbar or pressing Ctrl+C.

	A	B	C	D
1				
2	X	Y		
3	0	0	line 0,0	
4	1	1	1,1	
5	2	3	2,3	
6	3	3	3,3	
7				

2. Open Notepad program, and then **Paste** (Ctrl + V) the range.



3. Save the file with the .scr extension, for example, Line1.scr and place it, for example, in the directory C.
4. Open AutoCAD. Type **script** or **scr** at the Command prompt, or via **Tools > Run Script**.
5. From Directory C, select Line1.scr file, and click **Open**.
6. To fit the drawing on the screen, then type **Zoom > Extents**.

The result is appeared as below:



Example 3

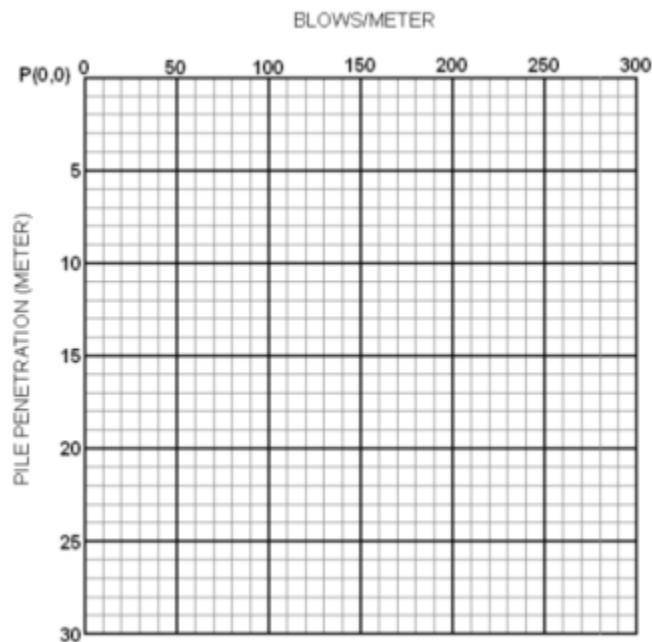
In a pile foundation work, the number of blows from driving hammer to penetrate piles is noted until it reaches a depth of 28 meters. The obtained data is presented in Table 10.1.

Table 10.1: Number of Blows versus Pile Penetration

Pile Penetration (m)	Number of blows/m'	Cumulative Blows
1	18	18
2	12	30
3	9	39
4	11	50
5	15	65
6	23	88
7	29	117
8	30	147
9	28	175
10	38	213
11	34	247
12	40	287
13	47	334
14	48	382
15	73	455
16	90	545
17	81	626
18	78	704
19	78	782
20	74	856
21	82	938
22	91	1029
23	101	1130
24	108	1238

25	120	1358
26	152	1510
27	163	1673
28	212	1885

The data in Table 10.1 will be plotted in Cartesian coordinate system, where X-coordinate is the number of blows/meter and the Y-coordinate is the penetration of the pile in meter. Thus, it is required a graph form, where the data have to fit with the form created. The penetration depth on the Y-axis then is made to increase downward to state depth below ground level, while the number of blows on the X-axis increases to the right. It looks like below.



The position of the form is placed at the bottom right of the origin $P(0,0)$ in AutoCAD drawing area. The scale made is:

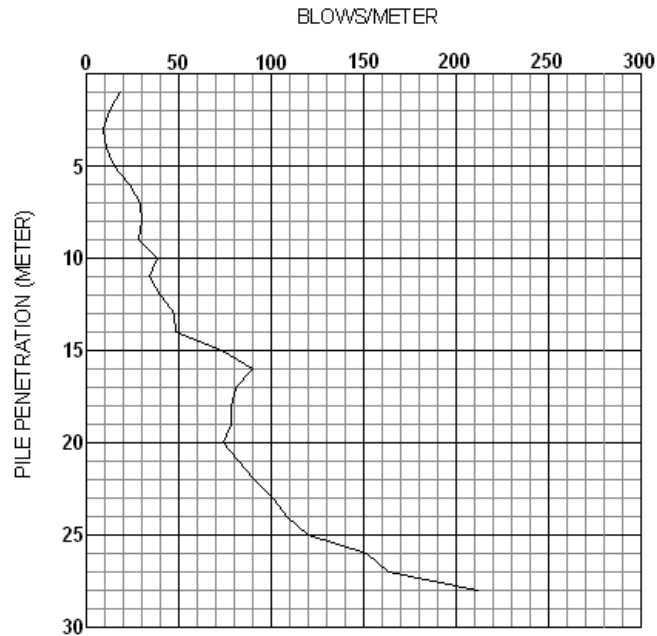
- X-coordinate: 1 unit = 10 blows
- Y-coordinate: 1 unit = 1 meter

With such scale, script for the data on the X-axis should be divided by 10, while the data on the Y-axis is divided by -1 for it has a negative direction to the origin $P(0,0)$. Script on the worksheet is created as follows:

	A	B	C	D	E	F
1	Pile Penetration Data					
2						
3	Pile No.	Penetration	Blow Count	Accumulation	Script	
4		(m)	Blows/m'	Blows/m'		
5	1	1	18	18	line 1,8,-1	= "line "&C5/10&","&B5/-1
6		2	12	30	1,2,-2	=C6/10&","&B6/-1
7		3	9	39	0,9,-3	
8		4	11	50	1,1,-4	
9		5	15	65	1,5,-5	
10		6	23	88	2,3,-6	
11		7	29	117	2,9,-7	
12		8	30	147	3,-8	
13		9	28	175	2,8,-9	
14		10	38	213	3,8,-10	
15		11	34	247	3,4,-11	
16		12	40	287	4,-12	
17		13	47	334	4,7,-13	
18		14	48	382	4,8,-14	
19		15	73	455	7,3,-15	
20		16	90	545	9,-16	
21		17	81	626	8,1,-17	
22		18	78	704	7,8,-18	
23		19	78	782	7,8,-19	
24		20	74	856	7,4,-20	
25		21	82	938	8,2,-21	
26		22	91	1029	9,1,-22	
27		23	101	1130	10,1,-23	
28		24	108	1238	10,8,-24	
29		25	120	1358	12,-25	
30		26	152	1510	15,2,-26	
31		27	163	1673	16,3,-27	
32		28	212	1885	21,2,-28	=C32/10&","&B32/-1&" "
33						

Result of copying cell
from E6 to E7:E31

Furthermore, the script will be then saved into a file with the .scr extension. When the file runs, the following graph is shown in AutoCAD:



Example 4

In this example a script will be created, where the cumulative blows will also be plotted on the graph. Thus, both of number of blows/meter and its accumulation are related to the same Y-axis (against pile penetration). But by seeing the cumulative blows has a range of values up to thousands, then a scale of 1 unit = 100 blows is given so that the values can conveniently fit in the graph form.

The result of cumulative blows is drawn using **Pline** with line thickness 0.1 unit. To create an AutoCAD script, a new column is then created named Script 2 (next to Script), contains formulas as shown below:

	A	B	C	D	E	F	G
1	Pile Penetration Data						
2							
3	Pile No.	Penetration (m)	Blow Count Blows/m'	Accumulation Blows/m'	Script	Script 2	
4							
5	1	1	18	18	line 1.8,-1	pline 0.18,-1 w 0.1	= "pline "&D5/100&" "&B5/-1&" w 0.1 "
6		2	12	30	1.2,-2	0.3,-2	=D6/100&" "&B6/-1
7		3	9	39	0.9,-3	0.39,-3	
8		4	11	50	1.1,-4	0.5,-4	
9		5	15	65	1.5,-5	0.65,-5	
10		6	23	88	2.3,-6	0.88,-6	
11		7	29	117	2.9,-7	1.17,-7	
12		8	30	147	3,-8	1.47,-8	
13		9	28	175	2.8,-9	1.75,-9	
14		10	38	213	3.8,-10	2.13,-10	
15		11	34	247	3.4,-11	2.47,-11	
16		12	40	287	4,-12	2.87,-12	
17		13	47	334	4.7,-13	3.34,-13	
18		14	48	382	4.8,-14	3.82,-14	
19		15	73	455	7.3,-15	4.55,-15	
20		16	90	545	9,-16	5.45,-16	
21		17	81	626	8.1,-17	6.26,-17	
22		18	78	704	7.8,-18	7.04,-18	
23		19	78	782	7.8,-19	7.82,-19	
24		20	74	856	7.4,-20	8.56,-20	
25		21	82	938	8.2,-21	9.38,-21	
26		22	91	1029	9.1,-22	10.29,-22	
27		23	101	1130	10.1,-23	11.3,-23	
28		24	108	1238	10.8,-24	12.38,-24	
29		25	120	1358	12,-25	13.58,-25	
30		26	152	1510	15.2,-26	15.1,-26	
31		27	163	1673	16.3,-27	16.73,-27	
32		28	212	1885	21.2,-28	18.85,-28	=D32/100&" "&B32/-1&" "
33							
34							

Result of copying cell
from F6 to F7:F31

Next step is to combine both two scripts from blows count and its accumulation into a single file. At first you have to select a range E5 to E32, then copy and paste it into Notepad and then followed by range F5 to F32. The contents of the script file will look as below:

line 1.8,-1

1.2,-2

0.9,-3

1.1,-4

1.5,-5

2.3,-6

2.9,-7

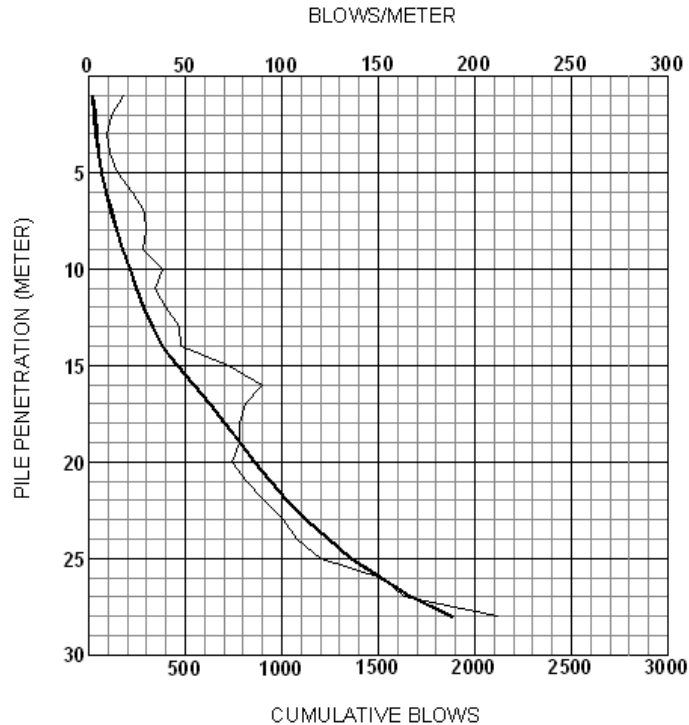
3,-8

2.8,-9

3.8,-10
3.4,-11
4,-12
4.7,-13
4.8,-14
7.3,-15
9,-16
8.1,-17
7.8,-18
7.8,-19
7.4,-20
8.2,-21
9.1,-22
10.1,-23
10.8,-24
12,-25
15.2,-26
16.3,-27
21.2,-28
pline 0.18,-1 w 0.2
0.3,-2
0.39,-3
0.5,-4
0.65,-5
0.88,-6

1.17,-7
1.47,-8
1.75,-9
2.13,-10
2.47,-11
2.87,-12
3.34,-13
3.82,-14
4.55,-15
5.45,-16
6.26,-17
7.04,-18
7.82,-19
8.56,-20
9.38,-21
10.29,-22
11.3,-23
12.38,-24
13.58,-25
15.1,-26
16.73,-27
18.85,-28

When the file runs, the following graph is shown in AutoCAD:



Example 5

In this example, we will create a **Layer** in AutoCAD for each pile data. For doing so, now is given 2 piles driving data, with location name PT-1 and PT-2, where each pile driving data will be put into **Layer** by its location name. You, who are accustomed to using AutoCAD, of course, will understand the importance of working with **Layer**.

To create a layer click the menus **Layer > Make**, and then define the layer name. If combined with the driving data the script is as follows:

```
"Layer make PT-1 line 1.8,-1"
```

The letter "m" can stand for **Make**, thus shortened to:

```
"Layer m PT-1 line 1.8,-1"
```

To customize each layer, it will be given a unique color. For example, line in layer PT-1 is blue, while in PT-2 is magenta. Now, the script becomes:

```
"layer m PT-1 c blue line 1.8,-1"
```

Script created above is only written in the first line (row). The next line will only be the coordinates of points entered into a layer. It is shown as below:

	A	B	C	D	E	F	G
1	Pile Penetration Data						
2							
3	Pile No.	Penetration	Blow Count	Accumulation	Script	Script 2	
4		(m)	Blows/m'	Blows/m'			
5	1	1	18	18	layer m PT-1 c blue line 1.8,-1	pline 0.18,-1 w 0.1	Cell E5 = "layer m PT-1 c blue line "&C5/10&";"&B5/-1
6		2	12	30	1.2,-2	0.3,-2	
7		3	9	39	0.9,-3	0.39,-3	
8		4	11	50	1.1,-4	0.5,-4	
9		5	15	65	1.5,-5	0.65,-5	
10		6	23	88	2.3,-6	0.88,-6	
11		7	29	117	2.9,-7	1.17,-7	
12		8	30	147	3,-8	1.47,-8	
13		9	28	175	2.8,-9	1.75,-9	
14		10	38	213	3.8,-10	2.13,-10	
15		11	34	247	3.4,-11	2.47,-11	
16		12	40	287	4,-12	2.87,-12	
17		13	47	334	4.7,-13	3.34,-13	
18		14	48	382	4.8,-14	3.82,-14	
19		15	73	455	7.3,-15	4.55,-15	
20		16	90	545	9,-16	5.45,-16	
21		17	81	626	8.1,-17	6.26,-17	
22		18	78	704	7.8,-18	7.04,-18	
23		19	78	782	7.8,-19	7.82,-19	
24		20	74	856	7.4,-20	8.56,-20	
25		21	82	938	8.2,-21	9.38,-21	
26		22	91	1029	9.1,-22	10.29,-22	
27		23	101	1130	10.1,-23	11.3,-23	
28		24	108	1238	10.8,-24	12.38,-24	
29		25	120	1358	12,-25	13.58,-25	
30		26	152	1510	15.2,-26	15.1,-26	
31		27	163	1673	16.3,-27	16.73,-27	
32		28	212	1885	21.2,-28	18.85,-28	
33	2	1	16	16	layer m PT-1 c magenta line 1.6,-1	pline 0.16,-1 w 0.1	Cell E33 = "layer m PT-1 c magenta line "&C33/10&";"&B33/-1
34		2	8	24	0.8,-2	0.24,-2	
35		3	6	30	0.6,-3	0.3,-3	

To select the script, take these steps, firstly you have to copy the value of cells E5 to E32 and then cells D5 to D32, secondly, select cells E33 to E61 and D33 to D61. By doing so, the script of number of blows and the cumulative blows of each pile are put into the layer that corresponds to the location number.

The script in Notepad is shown below:

layer m PT-1 c blue line 1.8,-1

1.2,-2

0.9,-3

1.1,-4

1.5,-5

2.3,-6

2.9,-7

3,-8

2.8,-9
3.8,-10
3.4,-11
4,-12
4.7,-13
4.8,-14
7.3,-15
9,-16
8.1,-17
7.8,-18
7.8,-19
7.4,-20
8.2,-21
9.1,-22
10.1,-23
10.8,-24
12,-25
15.2,-26
16.3,-27
21.2,-28
pline 0.18,-1 w 0.1
0.3,-2
0.39,-3
0.5,-4
0.65,-5

0.88,-6

1.17,-7

1.47,-8

1.75,-9

2.13,-10

2.47,-11

2.87,-12

3.34,-13

3.82,-14

4.55,-15

5.45,-16

6.26,-17

7.04,-18

7.82,-19

8.56,-20

9.38,-21

10.29,-22

11.3,-23

12.38,-24

13.58,-25

15.1,-26

16.73,-27

18.85,-28

layer m PT-2 c magenta line 1.6,-1

0.8,-2

0.6,-3
1.3,-4
1.2,-5
1.6,-6
2.5,-7
2.9,-8
3.2,-9
3.6,-10
3.9,-11
4.1,-12
4.9,-13
4.7,-14
5.7,-15
7.4,-16
5.9,-17
5.6,-18
5.7,-19
5.9,-20
6.4,-21
6.9,-22
7.1,-23
7.7,-24
8.4,-25
9.3,-26
10.3,-27

11.4,-28
12.5,-29
pline 0.16,-1 w 0.1
0.24,-2
0.3,-3
0.43,-4
0.55,-5
0.71,-6
0.96,-7
1.25,-8
1.57,-9
1.93,-10
2.32,-11
2.73,-12
3.22,-13
3.69,-14
4.26,-15
5,-16
5.59,-17
6.15,-18
6.72,-19
7.31,-20
7.95,-21
8.64,-22
9.35,-23

10.12,-24

10.96,-25

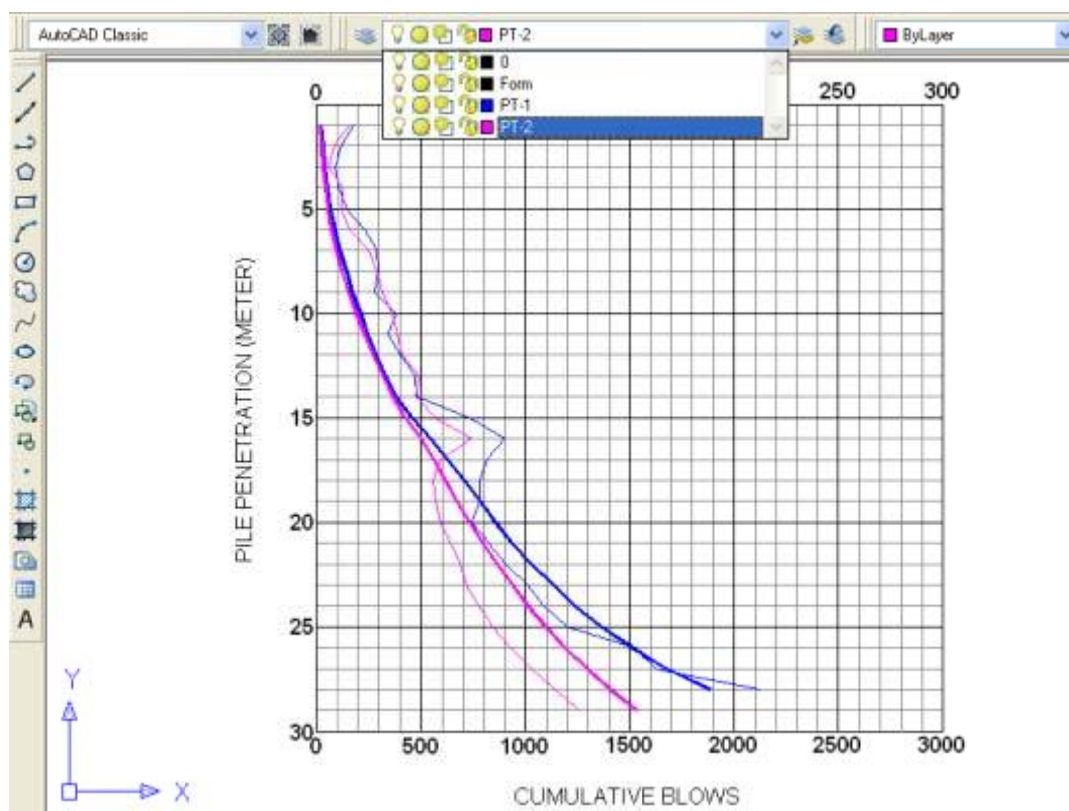
11.89,-26


12.92,-27

14.06,-28

15.31,-29

The result in AutoCAD:



Layers created in AutoCAD can be seen in a combo-box for layers, at the top-middle of the AutoCAD drawing area. Each layer can be displayed individually or together with other layer. To disable or enable a layer, click the mouse on the light icon  in the combo-box.

Note:

- Discussion on script of AutoCAD will take very long time for there are such many drawing features in AutoCAD. The detail is beyond the scope of this book. It is

expected that readers can get the ideas from the all examples above. The next step could be started by making some drawings as **Block**, which is then inserted using the **Insert** command. You can insert block such as markers, a shading object, an arrow or a form into a drawing area as many as required.

11.2 CREATING SCRIPTS IN VBA

The advantage of VBA is its ability to create a file with ASCII format that contains text. File is created using the CreateTextFile method with the following syntax:

object.**CreateTextFile**(*filename*[, *overwrite*[, *unicode*]])

Description:

object	Name of FileSystemObject or Folder object
filename	String, name of a file to create
overwrite	Optional. Boolean value, if TRUE indicates that file can be overwritten, or vice versa. If omitted, the value is FALSE.
unicode	Optional. Boolean value, indicates whether that file is created as Unicode file (TRUE) or ASCII (FALSE). If omitted the value is FALSE.

Example

```
Sub Createfile()  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    Set myfile = fso.CreateTextFile("c:\testfile.txt", True)  
    myfile.WriteLine ("Write this text")  
    myfile.Close  
End Sub
```

For accessing a file that has been created, use the **Open** command. The **Open** syntax itself consists of several parts with the descriptions that are quite detailed, however it will not be fully written here. For the purpose of creating a script file, use the following syntax:

```
Open "c:\testfile.scr" For Output As #1  
Print #1, "text"  
Write #1, variable  
Print #1, numeric & "text " & numeric  
Close #1
```

Descriptions:

- **Open** command is used for accessing a file named "testfile", to write script into a file. At the end of code, is to close the open file using **Close** command. With the above program the file will then be automatically named "filename" and stored in the directory C. Extension scr (*.scr) is to make a script file as an AutoCAD file.
- **Print** and **Write** statement is used to write data to a file. Sign "&" after the Print statement is to merge text and numerical characters. Statement of Write #1, var x, y var, var z is where each variable separated by a comma.
- If a file written in the directory does not exist, then a file will be created. By this default, the **Open** and **Close** command is more practical to use than the CreateTextFile method, because it can create a file and accessing any file at the same time.
- If the existing file is in a folder, it must be fully written the following folder and file name as well (for example, c:\MyFolder\testfile.scr). If you want to create a new folder, use **MkDir** statement.

Example 1

Creating a file and accessing to write text.

```
Open "c:\testfile" For Output As #1 'open file
Print #1, "write this text" 'write text to file
Close #1
```

With the above code, thus the testfile file contains text "write this test" can be accessed in directory C. When opened with Notepad program, it looks like this:



Example 2

Now, we will use VBA for creating script. Example 2 of Section 11.1 will be adopted for this example, and for your convenience is re-shown below:

	A	B	C	D
1				
2	X	Y		
3	0	0	line 0,0	←="line"&A3&","&B3&""
4	1	1	1,1	=A4&","&B4&""
5	2	3	2,3	} Result of copying cell from C4 to C5:C6
6	3	3	3,3	
7				

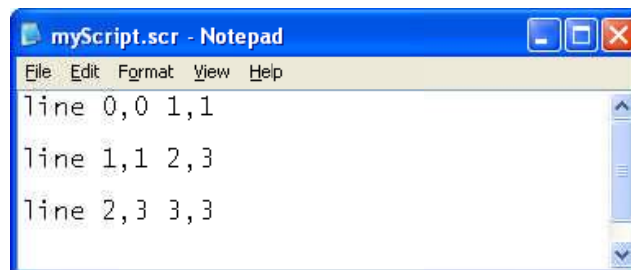
By using VBA, formulas in column C are then no longer needed. However, the worksheet form is now changed by adding a file name and a command button.

	A	B	C	D	E
1					
2	X	Y		File Name =	c:\myScript.scr
3	0	0			
4	1	1		Produce Script	
5	2	3			
6	3	3			
7					

The code can be written as follows:

```
Private Sub CommandButton1_Click()
Dim myFile As String
myFile = Cells(2, 5)
Open myFile For Output As #1
For i = 1 To 3
    Print #1, "line " & Cells(2 + i, 1) & "," & Cells(2 + i, 2) _
    & " " & Cells(3 + i, 1) & "," & Cells(3 + i, 2)
    Write #1, 'an empty line for pressing Enter
Next i
Close #1
End Sub
```

To run the above program is by clicking the command button named **Produce Script**. The resulting script file contains the following text



To run the script above, firstly, open your AutoCAD, then type scr in command prompt or via **Tools > Run Script**. From the **Select Script File** dialog box, select myScript.scr, then the line segments with the coordinates as in the script file will be created in AutoCAD.

Example 3

In creating line, it is highly recommended to create a table of joints so creating lines to be more convenient, especially when it comprises many segments and not all of them are continuous. See the previous examples in Chapter 1. Next is an example of worksheet data for creating lines. Now, it has an input to add a new folder to make your files storage tidy.

	A	B	C	D	E	F	G	H	I	J	K
1	Create Lines				Produce Script				Folder Name =	c:\myScriptFolder	
2									File Name =	myScript.scr	
3	Joint	Coordinate			Line		Joint				
4		x	y								
5	1	1.0	1.0		1	1	2		Joint No. =	4	
6	2	2.0	3.0		2	2	3		Line No. =	3	
7	3	5.0	4.0		3	2	4				
8	4	3.0	1.0								
9											
10											

The code:

```

Dim npt, nln, i
Dim myFolder, myFile
myFolder = Cells(1, 10)
myFile = Cells(2, 10)
npt = Cells(5, 10)
nln = Cells(6, 10)
ReDim x(npt), y(npt)
ReDim x1(nln), y1(nln), x2(nln), y2(nln)

'Read joint coord
For i = 1 To npt
    x(i) = Cells(4 + i, 2)
    y(i) = Cells(4 + i, 3)
Next i

'Read lines data
For i = 1 To nln

```

```

    x1(i) = x(Cells(4 + i, 6))
    y1(i) = y(Cells(4 + i, 6))
    x2(i) = x(Cells(4 + i, 7))
    y2(i) = y(Cells(4 + i, 7))
Next i

Set fso = CreateObject("Scripting.FileSystemObject")
If fso.FolderExists(myFolder) = True Then
    Filename = myFolder & "\" & myFile 'file with directory and folder
Else
    Mkdir (myFolder)
    Filename = myFolder & "\" & myFile
End If

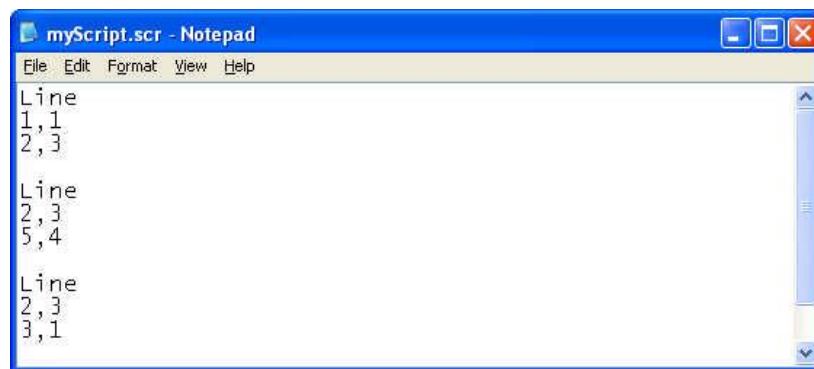
Open Filename For Output As #1

For i = 1 To nln
Print #1, "Line"
Write #1, x1(i), y1(i)
Write #1, x2(i), y2(i)
Write #1, 'an empty line for pressing Enter
Next i

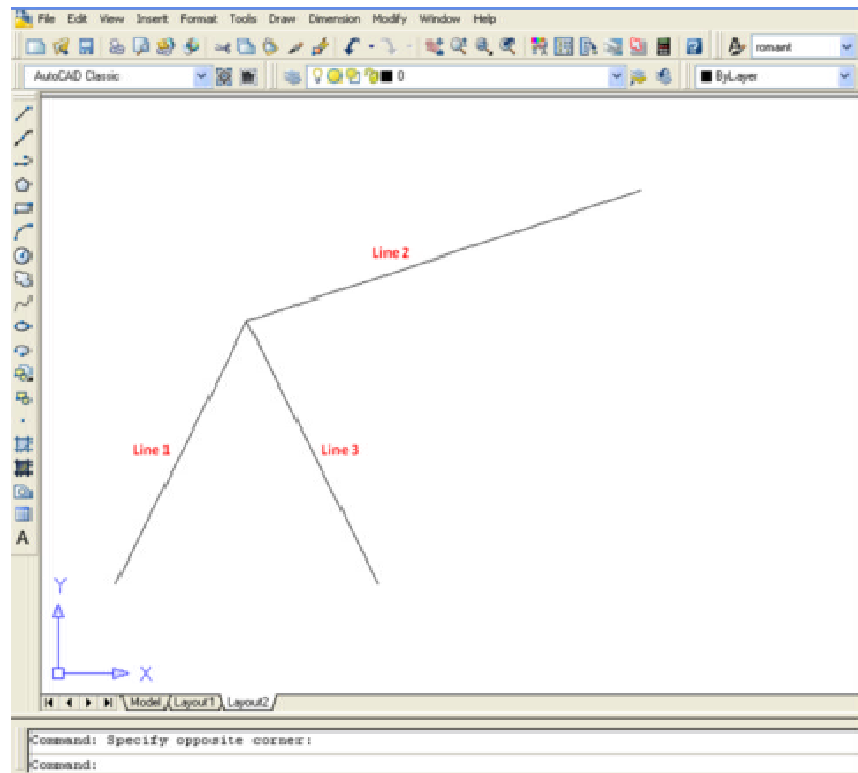
Close #1
End Sub

```

The content of script file:



When the script has run in AutoCAD, the result looks as follows:



The use of AutoCAD script in civil engineering field is quite popular because both Excel and AutoCAD could be found easily (who does not have?) and became a must standard softwares for civil engineers. Example 4 to 7 in the next pages are AutoCAD drawings produced from script files contained large amount of data. They are created for their respective interests.

Example 4 (next page):

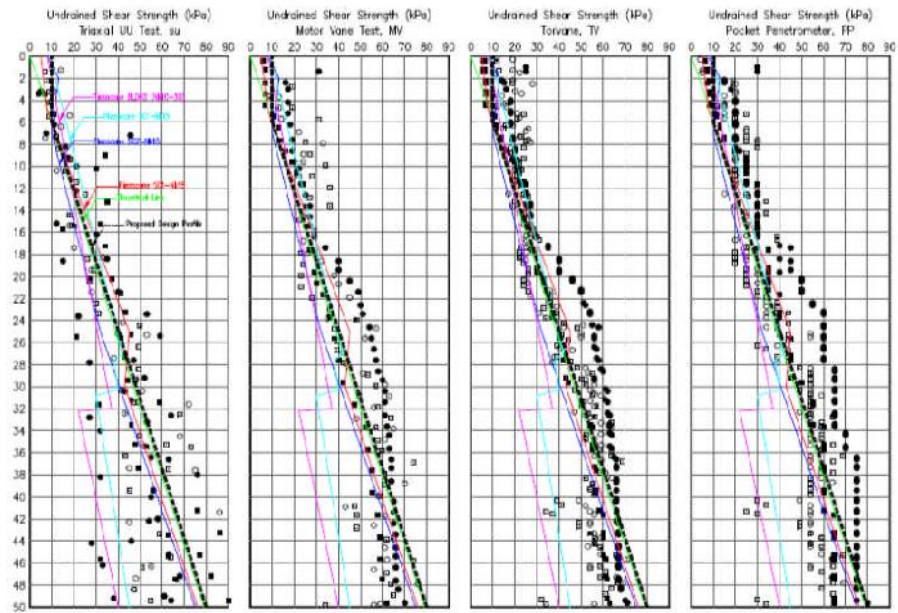


Figure 11.1: Profile of undrained shear strength of the soil versus depth from various test results

Example 5:

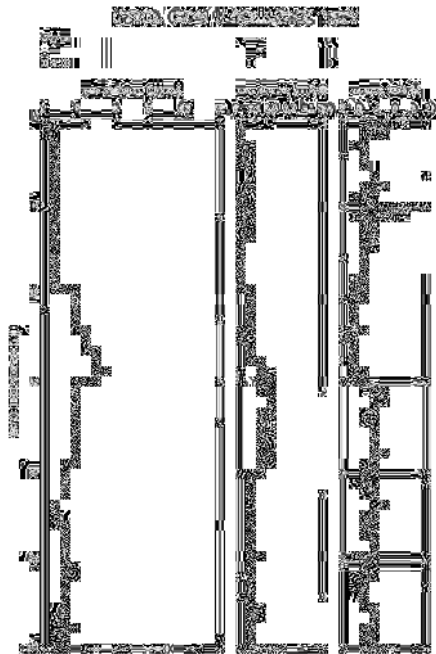


Figure 11.2: Result of Dutch Cone Penetration Test

Example 6

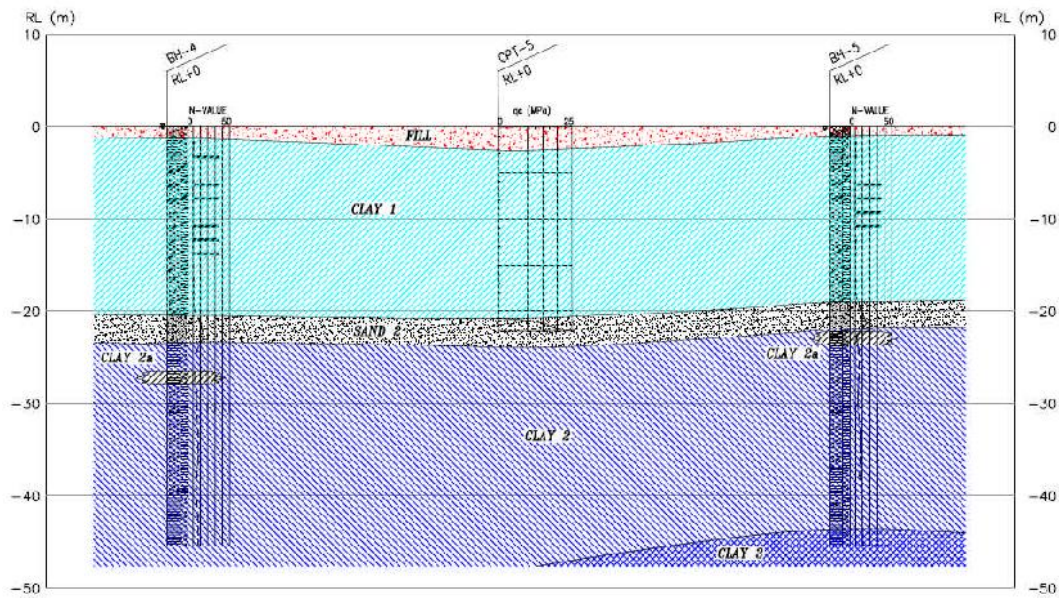


Figure 11.3: Long section of the soil profile

Example 7

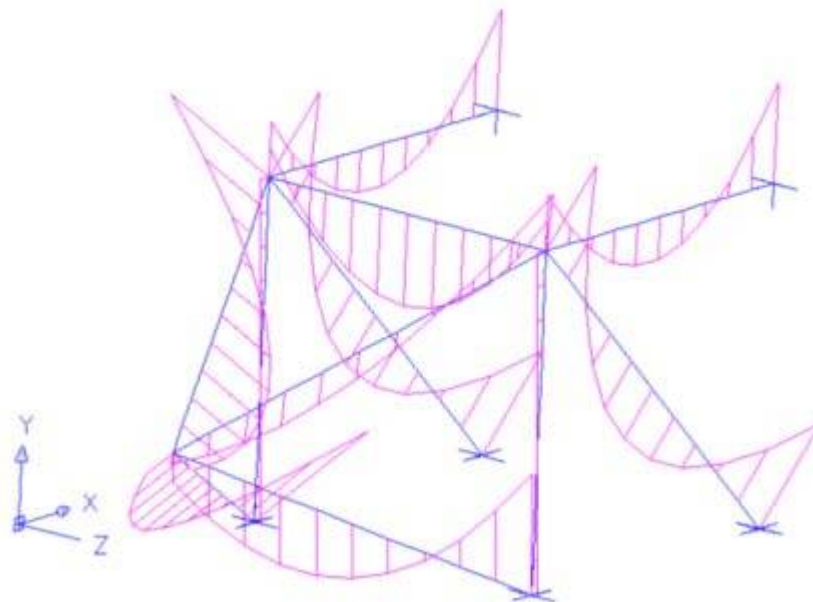


Figure 11.4: Bending moment-Z diagram from 3D frame analysis result

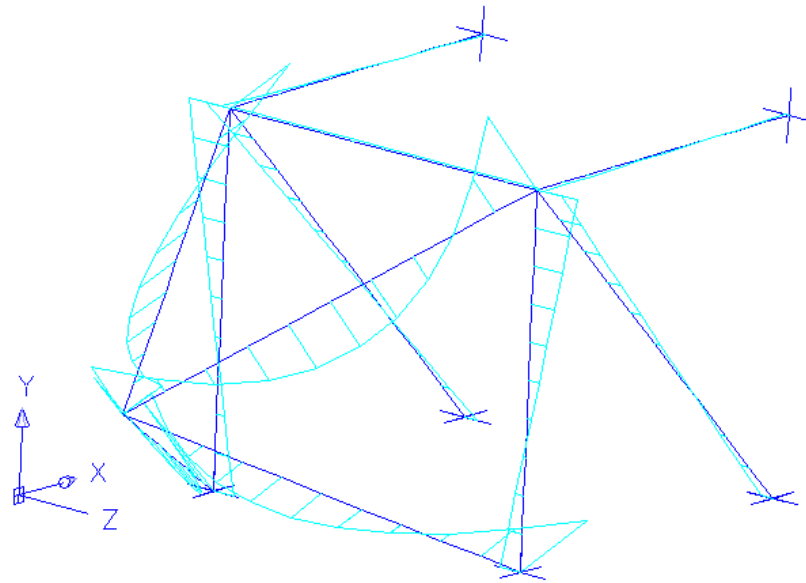


Figure 11.5: Bending moment-Y diagram from 3D frame analysis result

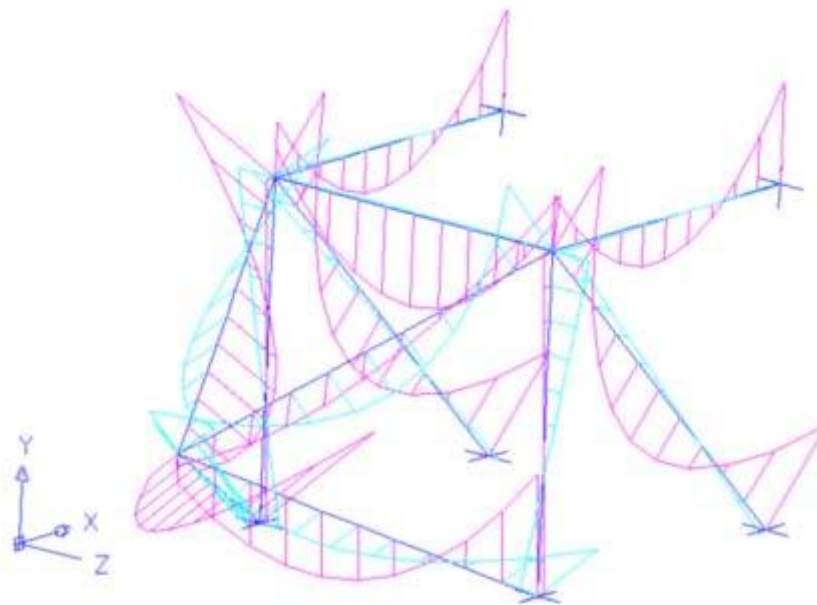


Figure 11.6: Diagram of bending moment-Z and bending moment-Y are displayed together

REFERENCES

- Bambang Triatmodjo, *Metode Numerik*, Beta Offset, Yogyakarta, 2002.
- Gunthar Pangaribuan, *Aplikasi Excel untuk Rekayasa Teknik Sipil*, Penerbit PT. Elex Media Komputindo, Jakarta, 2004.
- Gunthar Pangaribuan, *Penggunaan VBA-Excel untuk Program Perhitungan*, Penerbit PT. Elex Media Komputindo, Jakarta, 2005.
- Jean-Pierre Bardet, *Experimental Soil Mechanics*, Prentice-Hall Inc, 1997.
- Joseph E. Bowles, *Foundation Analysis and Design*, Third Edition, International Student Edition, 1982.
- R.F. Craig, *Soil Mechanics*, Fourth Edition, Spon Press, London, 1987.
- Supartono, F.X Ir. dan Ir. Teddy Boen, *Analisa Struktur dengan Metode Matrix*, Penerbit Universitas Indonesia, 1984.
- William Weaver, Jr., James M. Gere, *Matrix Analysis of Framed Structures*, Second Edition, Van Nostrand Reinhold Company, New York, 1980.
- Wiryanto Dewobroto, *Aplikasi Sain dan Teknik dengan Visual Basic 6.0*, Penerbit PT. Elex Media Komputindo, Jakarta, 2003.

ATTACHMENT: PROGRAM CODE

Module1 (FRAME2D)

```
'=====
'FRAME2D v.1.0 Program for 2D Frame Analysis, 2004
'by: Gunthar Pangaribuan - Refer to the book:
'An Introduction to: Excel for Civil Engineers
'=====

Option Explicit
Option Base 1
Public i As Integer, j As Integer, n As Integer
Type Joint_Data
    x As Double
    y As Double
End Type

Type Member_Data
    J1 As Integer
    J2 As Integer
    Dens As Double
    Ax As Double
    E As Double
    Iz As Double
    EI As Double
    lx As Double
    ly As Double
    Lh As Double
    Cx As Double
    Cy As Double
End Type

Public ChartWindow As Boolean
Public NP As Integer
Public DOF As Integer
Public NR As Integer
Public NJ As Integer
Public NM As Integer
```

```

Public Member() As Member_Data
Public Joint() As Joint_Data
Public NS As Integer 'no. of support
Public NUNI As Integer
Public UNI() As Double
Public GS() As Double 'global stiffness
Public Fm() As Double
Public Rs() As Integer
Public DFR() As Double
Public Idm() As Integer
Public Idj() As Integer
Public Irj() As Integer
Public Psum() As Double
Public Pmm() As Double
Public Pj() As Double
Public Xs() As Double
Public Xm() As Double
Public uscale As Integer
Public XAx_Wd As Double
Public YAx_Wd As Double

Sub FRAME2D()
On Error GoTo ErrMsg
Const NDJ = 3
Dim NLJ As Integer 'no of joint load
NJ = Cells(4, 2)
NM = Cells(5, 2)
uscale = Cells(4, 5)
'X-range:
XAx_Wd = Application.Max(Rows("10:10")) - Application.Min(Rows("10:10"))
'Y-range:
YAx_Wd = Application.Max(Rows("11:11")) - Application.Min(Rows("11:11"))
NS = Application.Count(Rows("22:22"))
NLJ = Application.Count(Rows("27:27"))
NUNI = Application.Count(Rows("32:32"))

```

```

'Total number of joint displacement, NP:
NP = NDJ * NJ

ReDim Joint(NJ) As Joint_Data
ReDim Member(NM) As Member_Data
ReDim MS(6, 6, NM) As Double 'member stiffness matrix
ReDim spn(NS) As Integer

ReDim UNI(NM)
ReDim GS(NP, NP) 'global stiffness matrix
ReDim Idm(6, NM) 'displacement index
ReDim Pj(NP) 'joint load vector
ReDim Pmm(NP, NM) 'fixed-end forces
ReDim Psum(NP) 'sum load matrix vector
ReDim Xs(NP) 'joint deformation vector
ReDim Xm(6, NM) 'member deformation
ReDim Fm(6, NM) 'member forces
ReDim Rs(NP)

'=====
'Read Input Data
'=====
'Read joint coordinates
For i = 1 To NJ
    With Joint(i)
        .x = Cells(10, 1 + i)
        .y = Cells(11, 1 + i)
    End With
Next i

'Read member data
For i = 1 To NM
    With Member(i)
        .J1 = Cells(14, 1 + i)
        .J2 = Cells(15, 1 + i)
        .Dens = Cells(16, 1 + i)
        .Ax = Cells(17, 1 + i)
        .E = Cells(18, 1 + i)
    End With
Next i

```

```

        .Iz = Cells(19, 1 + i)
        .EI = .E * .Iz
        .lx = Joint(.J2).x - Joint(.J1).x
        .ly = Joint(.J2).y - Joint(.J1).y
        .Lh = Sqr(.lx ^ 2 + .ly ^ 2)
        .Cx = .lx / .Lh
        .Cy = .ly / .Lh
    End With
Next i

If ChartWindow = True Then Call PlotGeometry: Exit Sub
Application.ScreenUpdating = False

'Read joint load
For i = 1 To NLJ
    n = Cells(27, 1 + i)
    Pj(3 * n - 2) = Cells(28, 1 + i)
    Pj(3 * n - 1) = Cells(29, 1 + i)
    Pj(3 * n) = Cells(30, 1 + i)
Next i

'Read uniform load
For i = 1 To NUNI
    n = Cells(32, 1 + i)
    UNI(n) = Cells(33, 1 + i)
Next i

'restrain list and support (Rs) index
For i = 1 To NS
    spn(i) = Cells(22, 1 + i)
    Rs(3 * spn(i) - 2) = Cells(23, 1 + i)
    Rs(3 * spn(i) - 1) = Cells(24, 1 + i)
    Rs(3 * spn(i)) = Cells(25, 1 + i)
Next i

DOF = 0
For i = 1 To NP

```



```

        If Rs(i) = 0 Then DOF = DOF + 1
Next i

'#restrained joint
NR = NP - DOF

ReDim DFX(NDJ * NS, DOF) As Double '[Kbf]according to Eq. 4.12
ReDim DFR(DOF, DOF) As Double '[Kff]^(-1) according to Eq. 4.9
ReDim RJ(NDJ * NS) As Double 'support reactions
ReDim Idj(DOF) 'free displacement index
ReDim Irj(NDJ * NS) 'support displacement index

'=====
'Structure Analysis
'=====
Call Mindex(Idm, Idj, Irj)
Call Stiff_Mtx(MS, GS)
Call Minvers(DFX, DFR)

'generate load
Call Genload(Pmm, Psum)
'determine displacements
Call DISP(Xs, Xm)
'=====
'Print Result
'=====
'clear previous results
Dim LastRow
LastRow = ActiveSheet.UsedRange.Rows.Count
Range(Cells(39, 1), Cells(LastRow, 18)).ClearContents
'1.Print Load
For i = 1 To NJ
    Cells(38 + i, 1).NumberFormat = "0"
    Cells(38 + i, 1) = i
    Cells(38 + i, 2).NumberFormat = "0.000"
    Cells(38 + i, 2) = Psum(3 * i - 2)
    Cells(38 + i, 3).NumberFormat = "0.000"

```

```

Cells(38 + i, 3) = Psum(3 * i - 1)
Cells(38 + i, 4).NumberFormat = "0.000"
Cells(38 + i, 4) = Psum(3 * i)
Next i

'2. Print Joint displacement
For i = 1 To NJ
    Cells(38 + i, 5).NumberFormat = "0.00000"
    Cells(38 + i, 5) = Xs(3 * i - 2)
    Cells(38 + i, 6).NumberFormat = "0.00000"
    Cells(38 + i, 6) = Xs(3 * i - 1)
    Cells(38 + i, 7).NumberFormat = "0.00000"
    Cells(38 + i, 7) = Xs(3 * i)
Next i

'-----
'Member Forces
'-----
'{F}m=[K]m.{X}m

For i = 1 To NM
    For j = 1 To 6
        Fm(j, i) = 0
        For n = 1 To 6
            Fm(j, i) = Fm(j, i) + MS(j, n, i) * Xm(n, i)
        Next n
    Next j
Next i

'Print member foces
For i = 1 To NM
    With Member(i)
        Cells(38 + i, 8).NumberFormat = "0"
        Cells(38 + i, 8) = i & "./" & Format(.Lh, "0.0")
        Cells(38 + i, 9).NumberFormat = "0.000"
        Cells(38 + i, 9) = Pmm(Idm(1, i), i) + Fm(1, i)
        Cells(38 + i, 10).NumberFormat = "0.000"
        Cells(38 + i, 10) = Pmm(Idm(2, i), i) + Fm(2, i)
    End With
Next i

```

```

Cells(38 + i, 11).NumberFormat = "0.000"
Cells(38 + i, 11) = Pmm(Idm(3, i), i) + Fm(3, i)
Cells(38 + i, 12).NumberFormat = "0.000"
Cells(38 + i, 12) = Pmm(Idm(4, i), i) + Fm(4, i)
Cells(38 + i, 13).NumberFormat = "0.000"
Cells(38 + i, 13) = Pmm(Idm(5, i), i) + Fm(5, i)
Cells(38 + i, 14).NumberFormat = "0.000"
Cells(38 + i, 14) = Pmm(Idm(6, i), i) + Fm(6, i)
End With
Next i

'-----
'Support Reactions
'-----
For i = 1 To 3 * NS
    For j = 1 To DOF
        RJ(i) = RJ(i) + DFX(i, j) * Xs(Idj(j))
    Next j
Next i

For i = 1 To NS
    Cells(38 + i, 15).NumberFormat = "0"
    Cells(38 + i, 15) = spn(i)
    Cells(38 + i, 16).NumberFormat = "0.000"
    Cells(38 + i, 16) = RJ(3 * i - 2) - Psum(Irj(3 * i - 2))
    Cells(38 + i, 17).NumberFormat = "0.000"
    Cells(38 + i, 17) = RJ(3 * i - 1) - Psum(Irj(3 * i - 1))
    Cells(38 + i, 18).NumberFormat = "0.000"
    Cells(38 + i, 18) = RJ(3 * i) - Psum(Irj(3 * i))
Next i

'=====
'Draw diagrams
'=====
Call PlotGeometry
Call PlotDisplacement
Call PlotMoment

```

```
Call PlotShear
Call PlotAxial
```

```
Range("A1").Select
Application.ScreenUpdating = True
Exit Sub
```

```
ErrMsg: MsgBox "Error, please check input ...", vbOKOnly + vbExclamation,
"FRAME2D"
```

```
Range("A1").Select
Application.ScreenUpdating = True
End Sub
```

Module2 (FRAME2D)

```
Option Explicit
Option Base 1
Public T() As Double 'transformation matrix,[T]
Public TT() As Double 'transpose [T], used in Module 3
'indexing for matrix subscript
Sub Mindex(dindex() As Integer, IFr() As Integer, IFx() As Integer)

'Member end-displacements index:
For i = 1 To NM
    With Member(i)
        dindex(1, i) = 3 * .J1 - 2
        dindex(2, i) = 3 * .J1 - 1
        dindex(3, i) = 3 * .J1
        dindex(4, i) = 3 * .J2 - 2
        dindex(5, i) = 3 * .J2 - 1
        dindex(6, i) = 3 * .J2
    End With
Next i

'Joint free displacement index, IFr and support index, IFx:
```

```

n = 1
j = 1
For i = 1 To NP
    If Rs(i) = 0 Then IFr(n) = i: n = n + 1
Next i

n = 1
For i = 1 To NS
    n = Cells(22, 1 + i)
    IFx(3 * i - 2) = 3 * n - 2
    IFx(3 * i - 1) = 3 * n - 1
    IFx(3 * i) = 3 * n
Next i
End Sub

Sub Stiff_Mtx(MK() As Double, S() As Double)
Dim sm1 As Double, sm2 As Double, sm3 As Double, sm4 As Double, sm5 As Double
ReDim M(6, 6, NM) As Double

'Building stiffness matrix
For i = 1 To NM
    With Member(i)
        sm1 = .Ax * .E / .Lh
        sm2 = 12 * .E * .Iz / (.Lh ^ 3)
        sm3 = 6 * .E * .Iz / (.Lh ^ 2)
        sm4 = 4 * .E * .Iz / .Lh
        sm5 = 2 * .E * .Iz / .Lh
        'Member stiffness
        MK(1, 1, i) = sm1: MK(1, 4, i) = -sm1
        MK(2, 2, i) = sm2: MK(2, 3, i) = sm3: MK(2, 5, i) = -sm2: MK(2, 6, i) =
sm3
        MK(3, 2, i) = sm3: MK(3, 3, i) = sm4: MK(3, 5, i) = -sm3: MK(3, 6, i) =
sm5
        MK(4, 1, i) = -sm1: MK(4, 4, i) = sm1
        MK(5, 2, i) = -sm2: MK(5, 3, i) = -sm3: MK(5, 5, i) = sm2: MK(5, 6, i) =
-sm3
        MK(6, 2, i) = sm3: MK(6, 3, i) = sm5: MK(6, 5, i) = -sm3: MK(6, 6, i) =
sm4
    End With
    'Member global stiffness

```

```

    M(1, 1, i) = sm1 * .Cx ^ 2 + sm2 * .Cy ^ 2: M(1, 2, i) = (sm1 - sm2) *
.Cx * .Cy: M(1, 3, i) = -sm3 * .Cy
    M(1, 4, i) = -M(1, 1, i): M(1, 5, i) = -(sm1 - sm2) * .Cx * .Cy: M(1, 6,
i) = M(1, 3, i)
    M(2, 1, i) = M(1, 2, i): M(2, 2, i) = sm1 * .Cy ^ 2 + sm2 * .Cx ^ 2: M(2,
3, i) = sm3 * .Cx
    M(2, 4, i) = M(1, 5, i): M(2, 5, i) = -M(2, 2, i): M(2, 6, i) = M(2, 3,
i)
    M(3, 1, i) = M(1, 3, i): M(3, 2, i) = M(2, 3, i): M(3, 3, i) = sm4
    M(3, 4, i) = -M(1, 3, i): M(3, 5, i) = -M(2, 3, i): M(3, 6, i) = sm5
    M(4, 1, i) = M(1, 4, i): M(4, 2, i) = M(1, 5, i): M(4, 3, i) = -M(1, 3,
i)
    M(4, 4, i) = M(1, 1, i): M(4, 5, i) = M(1, 2, i): M(4, 6, i) = -M(1, 3,
i)
    M(5, 1, i) = M(1, 5, i): M(5, 2, i) = -M(2, 2, i): M(5, 3, i) = -M(2, 3,
i)
    M(5, 4, i) = M(1, 2, i): M(5, 5, i) = M(2, 2, i): M(5, 6, i) = -M(2, 3,
i)
    M(6, 1, i) = M(1, 3, i): M(6, 2, i) = M(2, 3, i): M(6, 3, i) = M(3, 6, i)
    M(6, 4, i) = -M(1, 3, i): M(6, 5, i) = -M(2, 3, i): M(6, 6, i) = M(3, 3,
i)

'=====
'Structure stiffness matrix
'=====
'Storing members and superposition
For j = 1 To 6
    For n = 1 To 6
        S(Idm(j, i), Idm(n, i)) = S(Idm(j, i), Idm(n, i)) + M(j, n, i)
    Next n
Next j
End With
Next i

End Sub

'Global matrix
Sub MInvers(SR() As Double, SD() As Double)
Dim n As Integer

```

```

'submatrix Stiffness, KBF
For i = 1 To DOF
    For j = 1 To 3 * NS
        SR(j, i) = GS(Irj(j), Idj(i))
    Next j
Next i

'submatrix Stiffness, KFF
For i = 1 To DOF
    For j = 1 To DOF
        SD(i, j) = GS(Idj(i), Idj(j))
    Next j
Next i

'Inverse []
For i = 1 To DOF
    For j = 1 To DOF
        If j <> i Then SD(i, j) = SD(i, j) / SD(i, i)
    Next j
    For n = 1 To DOF
        If n = i Then GoTo 10
        For j = 1 To DOF
            If j <> i Then SD(n, j) = SD(n, j) - SD(i, j) * SD(n, i)
        Next j
10    Next n
    For n = 1 To DOF
        If n <> i Then SD(n, i) = -SD(n, i) / SD(i, i)
    Next n
    SD(i, i) = 1 / SD(i, i)
Next i

End Sub

Sub Genload(Pa() As Double, Ps() As Double)
ReDim TT(6, 6, NM) As Double
ReDim Peq(NP, NM) As Double 'equivalent load due to selfweight
Dim w As Double

```

```

'member fixed-end forces due to selfweight + uniform load
For i = 1 To NM
    With Member(i)
        w = .Dens * .Ax
        Pa(Idm(1, i), i) = .Cy * w * .Lh / 2
        Pa(Idm(2, i), i) = .Cx * w * .Lh / 2 + (UNI(i) * .Lh / 2)
        Pa(Idm(3, i), i) = (.Cx * w * .Lh ^ 2 / 12) + (UNI(i) * .Lh ^ 2 / 12)
        Pa(Idm(4, i), i) = .Cy * w * .Lh / 2
        Pa(Idm(5, i), i) = .Cx * w * .Lh / 2 + (UNI(i) * .Lh / 2)
        Pa(Idm(6, i), i) = -(.Cx * w * .Lh ^ 2 / 12) - (UNI(i) * .Lh ^ 2 / 12)
    End With
Next i

'Transformation matrix (transpose)
For i = 1 To NM
    With Member(i)
        TT(1, 1, i) = .Cx: TT(1, 2, i) = -.Cy
        TT(2, 1, i) = .Cy: TT(2, 2, i) = .Cx
        TT(3, 3, i) = 1
        TT(4, 4, i) = .Cx: TT(4, 5, i) = -.Cy
        TT(5, 4, i) = .Cy: TT(5, 5, i) = .Cx
        TT(6, 6, i) = 1
    End With
Next i

'equivalent joint load due to selfweight + UNI in global system
(superposition)
For i = 1 To NM
    For j = 1 To 6
        For n = 1 To 6
            Peq(Idm(j, i), i) = Peq(Idm(j, i), i) + TT(j, n, i) * -Pa(Idm(n,
i), i)
        Next n
    Next j
Next i

'sum load = joint load + eq.load

```



```

For i = 1 To NP
    Ps(i) = Pj(i)
Next i

'load superposition
For i = 1 To NM
    For j = 1 To 6
        Ps(Idm(j, i)) = Ps(Idm(j, i)) + Peq(Idm(j, i), i)
    Next j
Next i

End Sub

Sub DISP(x() As Double, X2() As Double)
    ReDim X2(6, NM) As Double
    ReDim Xi(6, NM) As Double
    ReDim T(6, 6, NM) As Double

    'Joint Displacement {X} = [SD]^-1.{P}
    For i = 1 To DOF
        'x(Idj(i)) = 0
        For j = 1 To DOF
            x(Idj(i)) = x(Idj(i)) + DFR(i, j) * Psum(Idj(j))
        Next j
    Next i

    'Numbering of global {X}(NP) to global {Xi} elemen
    For i = 1 To NM
        For j = 1 To 6
            Xi(j, i) = x(Idm(j, i))
        Next j
    Next i

    'Transforming {Xi} to local coordinates
    'Transformation matrix:
    For i = 1 To NM
        With Member(i)

```

```

    T(1, 1, i) = .Cx: T(1, 2, i) = .Cy
    T(2, 1, i) = -.Cy: T(2, 2, i) = .Cx
    T(3, 3, i) = 1
    T(4, 4, i) = .Cx: T(4, 5, i) = .Cy
    T(5, 4, i) = -.Cy: T(5, 5, i) = .Cx
    T(6, 6, i) = 1
    End With
Next i

'member deformation, tranformed{X} = [T].{Xi}
For i = 1 To NM
    For j = 1 To 6
        X2(j, i) = 0
        For n = 1 To 6
            X2(j, i) = X2(j, i) + T(j, n, i) * Xi(n, i)
        Next n
    Next j
Next i

End Sub

```

Module3 (FRAME2D)

```

'This module consist of only code to create charts
Option Explicit
Option Base 1

Sub PlotGeometry()
On Error Resume Next
Dim Cx1, Cx2, Cy1, Cy2
ReDim mxVal(NM) As Variant
ReDim myVal(NM) As Variant
ReDim tagxVal(NM) As Variant
ReDim tagyVal(NM) As Variant
Dim am As Integer

```

```

'member coordinates
For i = 1 To NM
    With Member(i)
        Cx1 = Joint(.J1).x
        Cy1 = Joint(.J1).y
        Cx2 = Joint(.J2).x
        Cy2 = Joint(.J2).y
    End With
    mxVal(i) = Array(Cx1, Cx2)
    myVal(i) = Array(Cy1, Cy2)
    tagxVal(i) = Array((Cx1 + Cx2) / 2)
    tagyVal(i) = Array((Cy1 + Cy2) / 2)
Next i

ActiveSheet.ChartObjects("Chart 1").Activate
ActiveChart.ChartArea.Select
Selection.ClearContents

'creating joints
For i = 1 To NJ
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = Joint(i).x
        .SeriesCollection(i).Values = Joint(i).y
        .SeriesCollection(i).Name = "J" & i
    End With
    'node marker
    ActiveChart.PlotArea.Select
    ActiveChart.SeriesCollection(i).Select
    With Selection
        .MarkerBackgroundColorIndex = 3 'red
        .MarkerForegroundColorIndex = 3
        .MarkerStyle = xlCircle
        .MarkerSize = 6
        .Shadow = False
        .ApplyDataLabels AutoText:=True, LegendKey:= _
            False, ShowSeriesName:=True, ShowCategoryName:=False,
            ShowValue:=False, _

```



```

End With
With Selection.Border
    .ColorIndex = 5
    .Weight = xlThin
    .LineStyle = xlContinuous
End With
am = am + 1
Next i

'creating member name tags
am = 1
For i = n + 1 To n + NM
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = tagxVal(am)
        .SeriesCollection(i).Values = tagyVal(am)
        .SeriesCollection(i).Name = "M" & am
    End With
    ActiveChart.PlotArea.Select
    ActiveChart.SeriesCollection(i).Select
    With Selection
        .MarkerBackgroundColorIndex = 5
        .MarkerForegroundColorIndex = 5
        .MarkerStyle = xlSquare
        .Smooth = False
        .MarkerSize = 6
        .Shadow = False
        .ApplyDataLabels AutoText:=True, LegendKey:= _
            False, ShowSeriesName:=True, ShowCategoryName:=False,
ShowValue:=False, _
            ShowPercentage:=False, ShowBubbleSize:=False
    End With
    ActiveChart.SeriesCollection(i).DataLabels.Select
    With Selection.Font
        .Name = "Arial"
        .FontStyle = "Regular"
        .Size = 8
    End With

```

```

am = am + 1
Next i

    ActiveChart.ChartArea.Select
    With ActiveChart
        .Axes(xlCategory, xlPrimary).HasTitle = True
        .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = _
            "NJ = " & Format(NJ, "0") & ", NM = " & Format(NM, "0")
    End With

ActiveChart.ShowWindow = True

End Sub

Sub PlotDisplacement()
On Error Resume Next

Dim Cx1, Cx2, Cy1, Cy2
ReDim mxValbf(NM) As Variant, myValbf(NM) As Variant
ReDim mxValaf(NM) As Variant, myValaf(NM) As Variant

Dim am As Integer, scale_Y As Single, scale_X As Single, scx As Single
Dim Xmax As Double, at_joint As Integer

    ActiveSheet.ChartObjects("Chart 2").Activate
    ActiveChart.ChartArea.Select
    Selection.ClearContents

'=====
'scale for displacement
'=====
'user scale: max displacement are drawn up to 10% of the axis range
'or follows formula: X-range or Y-range * 10% = Xmax

'determine max. joint-X and Y displacement
'(user function UXMax written at the end of this module):
Xmax = Application.index(UXMax(Xs, NP), 1)

```

```

at_joint = Application.index(UXMax(Xs, NP), 2)
scx = (Application.Max(XAx_Wd, YAx_Wd) * 10 / 100) / Abs(Xmax)
If uscale = 1 Then
    scale_X = 1
    scale_Y = 1
ElseIf uscale = 0 Then
    scale_X = scx
    scale_Y = scx
Else
    scale_X = 0
    scale_Y = 0
End If

'member coordinates: before & after loading
For i = 1 To NM
    With Member(i)
        Cx1 = Joint(.J1).x
        Cy1 = Joint(.J1).y
        Cx2 = Joint(.J2).x
        Cy2 = Joint(.J2).y
        mxValbf(i) = Array(Cx1, Cx2)
        myValbf(i) = Array(Cy1, Cy2)
        Cx1 = Joint(.J1).x + scale_X * Xs(3 * .J1 - 2)
        Cy1 = Joint(.J1).y + scale_Y * Xs(3 * .J1 - 1)
        Cx2 = Joint(.J2).x + scale_X * Xs(3 * .J2 - 2)
        Cy2 = Joint(.J2).y + scale_Y * Xs(3 * .J2 - 1)
        mxValaf(i) = Array(Cx1, Cx2)
        myValaf(i) = Array(Cy1, Cy2)
    End With
Next i

am = 1
'creating member lines: before loading
For i = 1 To NM
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = mxValbf(am)
    End With
Next i

```



```

Next i

ActiveChart.ChartArea.Select
With ActiveChart
    .Axes(xlCategory, xlPrimary).HasTitle = True
    .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = _
        "Max Displacement, J." & at_joint & " = " & Format(Xmax, "0.00000")
End With

End Sub

Sub PlotMoment()
On Error Resume Next
Dim Cx1, Cx2, Cy1, Cy2
Dim Cmx1, Cmx2, Cmy1, Cmy2
ReDim mxVal(NM) As Variant
ReDim myVal(NM) As Variant
ReDim moxVal(NM) As Variant
ReDim moyVal(NM) As Variant
ReDim tagxVal(NM) As Variant
ReDim tagyVal(NM) As Variant

ReDim mjsxVal(NM + NM) As Variant
ReDim mjyVal(NM + NM) As Variant
ReDim mcolor(NM + NM) As Variant
ReDim Pm_LM(3, NM) As Double
ReDim Pm_RM(3, NM) As Double
ReDim Pm_LT(3, NM) As Double
ReDim Pm_RT(3, NM) As Double

Dim am As Integer, scm As Single, scale_Y As Single, scale_X As Single
Dim Mmax As Double, at_joint As Integer, L_I As Integer, L_II As Integer

ActiveSheet.ChartObjects("Chart 3").Activate
ActiveChart.ChartArea.Select
Selection.ClearContents

```

```

'analogous to plot displacement for scaling
Mmax = Application.index(UMMax(Pmm, Fm, NM), 1)
at_joint = Application.index(UMMax(Pmm, Fm, NM), 2)
scm = (Application.Max(XAx_Wd, YAx_Wd) * 15 / 100) / Abs(Mmax)

If uscale = 1 Then
    scale_X = 1
    scale_Y = 1
ElseIf uscale = 0 Then
    scale_X = scm
    scale_Y = scm
Else
    scale_X = 0
    scale_Y = 0
End If

'member moment, consider only 3rd and 6th vectors
For i = 1 To NM
    Pm_LM(2, i) = Pmm(Idm(3, i), i) + Fm(3, i)
    Pm_RM(2, i) = -(Pmm(Idm(6, i), i) + Fm(6, i))
Next i

'transform member moment to global axis by using [TT]
'the drawing should be in global coordinates
For i = 1 To NM
    For j = 1 To 3
        For n = 1 To 3
            Pm_LT(j, i) = Pm_LT(j, i) + TT(j, n, i) * Pm_LM(n, i)
            Pm_RT(j, i) = Pm_RT(j, i) + TT(j, n, i) * Pm_RM(n, i)
        Next n
    Next j
Next i

For i = 1 To NM
    With Member(i)
        Cx1 = Joint(.J1).x
        Cy1 = Joint(.J1).y
    End With

```



```

For i = L_I + 1 To L_II
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = mJxVal(am)
        .SeriesCollection(i).Values = mJyVal(am)
        .SeriesCollection(i).Name = " " " " " "
    End With
    ActiveChart.SeriesCollection(i).Select
    With Selection
        .MarkerBackgroundColorIndex = xlNone
        .MarkerForegroundColorIndex = xlNone
        .MarkerStyle = xlNone
        .Smooth = False
    End With
    With Selection.Border
        If mcolor(am) < 0 Then
            .ColorIndex = 3
        Else
            .ColorIndex = 5
        End If
        .Weight = xlMedium
        .LineStyle = xlContinuous
    End With
    am = am + 1
Next i

ActiveChart.ChartArea.Select
With ActiveChart
    .Axes(xlCategory, xlPrimary).HasTitle = True
    .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = _
        "Max Moment, J." & at_joint & " = " & Format(Mmax, "0.000")
End With

End Sub

```

```

Sub PlotShear()

On Error Resume Next
Dim Cx1, Cx2, Cy1, Cy2
Dim Cmx1, Cmx2, Cmy1, Cmy2
ReDim mxVal(NM) As Variant
ReDim myVal(NM) As Variant
ReDim shxVal(NM) As Variant
ReDim shyVal(NM) As Variant
ReDim tagxVal(NM) As Variant
ReDim tagyVal(NM) As Variant

ReDim mJxVal(NM + NM) As Variant
ReDim mJyVal(NM + NM) As Variant
ReDim sColor(NM + NM) As Variant
ReDim Ps_LM(3, NM) As Double
ReDim Ps_RM(3, NM) As Double
ReDim Ps_LT(3, NM) As Double
ReDim Ps_RT(3, NM) As Double

Dim am As Integer, scs As Single, scale_Y As Single, scale_X As Single
Dim Smax As Double, at_joint As Integer, L_I As Integer, L_II As Integer

    ActiveSheet.ChartObjects("Chart 4").Activate
    ActiveChart.ChartArea.Select
    Selection.ClearContents

'analogous to with plot displacement for scaling
Smax = Application.index(USMax(Pmm, Fm, NM), 1)
at_joint = Application.index(USMax(Pmm, Fm, NM), 2)
scs = (Application.Max(XAx_Wd, YAx_Wd) * 15 / 100) / Abs(Smax)

If uscale = 1 Then
    scale_X = 1
    scale_Y = 1
ElseIf uscale = 0 Then
    scale_X = scs

```

```

        scale_Y = scs
Else
    scale_X = 0
    scale_Y = 0
End If

'member moment, consider only 2nd and 5th vectors
For i = 1 To NM
    Ps_LM(2, i) = (Pmm(Idm(2, i), i) + Fm(2, i))
    Ps_RM(2, i) = -(Pmm(Idm(5, i), i) + Fm(5, i))
Next i

'transform member shear to global axis by using [TT]
'the drawing should be in global coordinates
For i = 1 To NM
    For j = 1 To 3
        For n = 1 To 3
            Ps_LT(j, i) = Ps_LT(j, i) + TT(j, n, i) * Ps_LM(n, i)
            Ps_RT(j, i) = Ps_RT(j, i) + TT(j, n, i) * Ps_RM(n, i)
        Next n
    Next j
Next i

For i = 1 To NM
    With Member(i)
        Cx1 = Joint(.J1).x
        Cy1 = Joint(.J1).y
        Cx2 = Joint(.J2).x
        Cy2 = Joint(.J2).y
        'assign force coordinates
        Cmx1 = Cx1 + scale_X * Ps_LT(1, i) 'x direction
        Cmy1 = Cy1 + scale_Y * Ps_LT(2, i) 'y direction
        Cmx2 = Cx2 + scale_X * Ps_RT(1, i)
        Cmy2 = Cy2 + scale_Y * Ps_RT(2, i)
    End With
    'member coordinates
    mxVal(i) = Array(Cx1, Cx2)

```

```

        myVal(i) = Array(Cy1, Cy2)
'shear coordinates
        shxVal(i) = Array(Cmx1, Cmx2)
        shyVal(i) = Array(Cmy1, Cmy2)
'member-joint coordinates
        mjsxVal(2 * i - 1) = Array(Cx1, Cmx1)
        mjyVal(2 * i - 1) = Array(Cy1, Cmy1)
        scolor(2 * i - 1) = (Pmm(Idm(2, i), i) + Fm(2, i))
        mjsxVal(2 * i) = Array(Cx2, Cmx2)
        mjyVal(2 * i) = Array(Cy2, Cmy2)
        scolor(2 * i) = -(Pmm(Idm(5, i), i) + Fm(5, i))
Next i

'creating member lines
For i = 1 To NM
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = mxVal(i)
        .SeriesCollection(i).Values = myVal(i)
        .SeriesCollection(i).Name = ""
    End With
    ActiveChart.SeriesCollection(i).Select
    With Selection
        .MarkerBackgroundColorIndex = xlNone
        .MarkerForegroundColorIndex = xlNone
        .MarkerStyle = xlNone
        .Smooth = False
    End With
    With Selection.Border
        .ColorIndex = 5
        .Weight = xlThin
        .LineStyle = xlContinuous
    End With
Next i

L_I = NM + NM
am = 1

```



```

        .MarkerStyle = xlNone
        .Smooth = False
    End With
    With Selection.Border
        If scolor(am) < 0 Then
            .ColorIndex = 3
        Else
            .ColorIndex = 5
        End If
        .Weight = xlMedium
        .LineStyle = xlContinuous
    End With
    am = am + 1

Next i

ActiveChart.ChartArea.Select
With ActiveChart
    .Axes(xlCategory, xlPrimary).HasTitle = True
    .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = _
        "Max Shear, J." & at_joint & " = " & Format(Smax, "0.000")
End With

End Sub

Sub PlotAxial()

On Error Resume Next

Dim Cx1, Cx2, Cy1, Cy2
Dim Cmx1, Cmx2, Cmy1, Cmy2
ReDim mxVal(NM) As Variant
ReDim myVal(NM) As Variant
ReDim saxVal(NM) As Variant
ReDim sayVal(NM) As Variant
ReDim tagxVal(NM) As Variant
ReDim tagyVal(NM) As Variant

```

```

ReDim mJxVal(NM + NM) As Variant
ReDim mJyVal(NM + NM) As Variant
ReDim aColor(NM + NM) As Variant
ReDim Pa_LM(3, NM) As Double
ReDim Pa_RM(3, NM) As Double
ReDim Pa_LT(3, NM) As Double
ReDim Pa_RT(3, NM) As Double

Dim am As Integer, sca As Single, scale_Y As Single, scale_X As Single
Dim Amax As Double, at_joint As Integer, L_I As Integer, L_II As Integer

    ActiveSheet.ChartObjects("Chart 5").Activate
    ActiveChart.ChartArea.Select
    Selection.ClearContents

'analogueous to plot displacement for scaling
Amax = Application.index(UAMax(Pmm, Fm, NM), 1)
at_joint = Application.index(UAMax(Pmm, Fm, NM), 2)
sca = (Application.Max(XAx_Wd, YAx_Wd) * 15 / 100) / Abs(Amax)

If uscale = 1 Then
    scale_X = 1
    scale_Y = 1
ElseIf uscale = 0 Then
    scale_X = sca
    scale_Y = sca
Else
    scale_X = 0
    scale_Y = 0
End If

'member axial, consider only 1st and 4th vectors
For i = 1 To NM
    Pa_LM(2, i) = -(Pmm(Idm(1, i), i) + Fm(1, i))
    Pa_RM(2, i) = (Pmm(Idm(4, i), i) + Fm(4, i))
Next i

```

```

'transform member axial to global axis by using [TT]
'the drawing should be in global coordinates
For i = 1 To NM
    For j = 1 To 3
        For n = 1 To 3
            Pa_LT(j, i) = Pa_LT(j, i) + TT(j, n, i) * Pa_LM(n, i)
            Pa_RT(j, i) = Pa_RT(j, i) + TT(j, n, i) * Pa_RM(n, i)
        Next n
    Next j
Next i

For i = 1 To NM
    With Member(i)
        Cx1 = Joint(.J1).x
        Cy1 = Joint(.J1).y
        Cx2 = Joint(.J2).x
        Cy2 = Joint(.J2).y
        'assign force coordinates
        Cmx1 = Cx1 + scale_X * Pa_LT(1, i)
        Cmy1 = Cy1 + scale_Y * Pa_LT(2, i)
        Cmx2 = Cx2 + scale_X * Pa_RT(1, i)
        Cmy2 = Cy2 + scale_Y * Pa_RT(2, i)
    End With
    'member coordinates
    mxVal(i) = Array(Cx1, Cx2)
    myVal(i) = Array(Cy1, Cy2)
    'axial coordinates
    saxVal(i) = Array(Cmx1, Cmx2)
    sayVal(i) = Array(Cmy1, Cmy2)
    'member-joint coordinates
    mjsxVal(2 * i - 1) = Array(Cx1, Cmx1)
    mjyVal(2 * i - 1) = Array(Cy1, Cmy1)
    acolor(2 * i - 1) = -(Pmm(Idm(1, i), i) + Fm(1, i))
    mjsxVal(2 * i) = Array(Cx2, Cmx2)
    mjyVal(2 * i) = Array(Cy2, Cmy2)
    acolor(2 * i) = Pmm(Idm(4, i), i) + Fm(4, i)
Next i

```



```

Next i
    ActiveChart.ChartArea.Select
    With ActiveChart
        .Axes(xlCategory, xlPrimary).HasTitle = True
        .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = _
            "Max Axial, J." & at_joint & " = " & Format(Amax, "0.000")
    End With
End Sub

'user function to determine maximum value of
'X1 and X2 (rotation X3 is not included)

Function UXMax(Xs, NP) As Variant

    Dim Abs_FXMax As Double
    Dim FXMax As Double

    Abs_FXMax = Abs(Xs(1))
    FXMax = 0
    n = 1
    j = 3
    For i = 2 To NP
        Select Case i
            Case j
                j = j + 3
                n = n + 1
            Case Else
                If Abs(Xs(i)) > Abs_FXMax Then
                    Abs_FXMax = Abs(Xs(i))
                    FXMax = Xs(i)
                    UXMax = Array(FXMax, n)
                    'UXMax keeps origin value(+ or - value)
                End If
            End Select
        Next i
    End Function

```

```

'user function to determine maximum value of
'moment, F3 (F1 and F2 are not included)

Function UMMax(Pmm, Fm, NM) As Variant
Dim Abs_FMMMax As Double
Dim FMMMax As Double

Abs_FMMMax = 0
n = 0
For i = 1 To NM
    If Abs(Pmm(Idm(3, i), i) + Fm(3, i)) > Abs_FMMMax Then
        Abs_FMMMax = Abs(Pmm(Idm(3, i), i) + Fm(3, i))
        FMMMax = -(Pmm(Idm(3, i), i) + Fm(3, i))
        n = Member(i).J1
        UMMMax = Array(FMMMax, n)
    End If
    If Abs(Pmm(Idm(6, i), i) + Fm(6, i)) > Abs_FMMMax Then
        Abs_FMMMax = Abs(Pmm(Idm(6, i), i) + Fm(6, i))
        FMMMax = Pmm(Idm(6, i), i) + Fm(6, i)
        n = Member(i).J2
        UMMMax = Array(FMMMax, n)
    End If
Next i
End Function

```

```

'user function to determine maximum value of
'shear force, F2 (F1 and F3 are not included)
Function USMax(Pmm, Fm, NM) As Variant

Dim Abs_FSMMax As Double
Dim FSMMax As Double

Abs_FSMMax = 0
n = 0
For i = 1 To NM
    If Abs(Pmm(Idm(2, i), i) + Fm(2, i)) > Abs_FSMMax Then
        Abs_FSMMax = Abs(Pmm(Idm(2, i), i) + Fm(2, i))

```



```

        FSMax = (Pmm(Idm(2, i), i) + Fm(2, i))
        n = Member(i).J1
        USMax = Array(FSMax, n)
    End If
    If Abs(Pmm(Idm(5, i), i) + Fm(5, i)) > Abs_FSMax Then
        Abs_FSMax = Abs(Pmm(Idm(5, i), i) + Fm(5, i))
        FSMax = -(Pmm(Idm(5, i), i) + Fm(5, i))
        n = Member(i).J2
        USMax = Array(FSMax, n)
    End If
Next i
End Function

'user function to determine maximum value of
'axial force, F1 (F2 and F3 are not included)
Function UAMax(Pmm, Fm, NM) As Variant

Dim Abs_FAMax As Double
Dim FAMax As Double

Abs_FAMax = 0
n = 0
For i = 1 To NM
    If Abs(Pmm(Idm(1, i), i) + Fm(1, i)) > Abs_FAMax Then
        Abs_FAMax = Abs(Pmm(Idm(1, i), i) + Fm(1, i))
        FAMax = -(Pmm(Idm(1, i), i) + Fm(1, i))
        n = Member(i).J1
        UAMax = Array(FAMax, n)
    End If
    If Abs(Pmm(Idm(4, i), i) + Fm(4, i)) > Abs_FAMax Then
        Abs_FAMax = Abs(Pmm(Idm(4, i), i) + Fm(4, i))
        FAMax = Pmm(Idm(4, i), i) + Fm(4, i)
        n = Member(i).J2
        UAMax = Array(FAMax, n)
    End If
Next i
End Function

```

Command Button (FRAME2D)

```
Private Sub CommandButton1_Click()  
ChartWindow = True  
    Call FRAME2D  
End Sub
```

```
Private Sub CommandButton2_Click()  
ChartWindow = False  
    Call FRAME2D  
End Sub
```

Module1 (TRUSS2D)

```
'=====
```

'TRUSS2D v.1.0 Program for 2D TRUSS analysis, 2004

'by: Gunthar Pangaribuan - Refer to the book:

'An Introduction to: Excel for Civil Engineers

```
'=====
```

```
Option Explicit  
Option Base 1
```

```
Public i As Integer, j As Integer, n As Integer
```

```
Type Joint_Data  
    x As Double  
    y As Double  
End Type
```

```
Type Member_Data  
    J1 As Integer  
    J2 As Integer  
    Dens As Double  
    Ax As Double  
    E As Double
```

```

    lx As Double
    ly As Double
    Lh As Double
    Cx As Double
    Cy As Double
End Type

Public ChartWindow As Boolean
Public NP As Integer
Public DOF As Integer
Public NR As Integer
Public NJ As Integer
Public NM As Integer
Public Member() As Member_Data
Public Joint() As Joint_Data
Public NS As Integer 'no. of support
Public GS() As Double 'global stiffness
Public Rs() As Integer
Public DFR() As Double
Public Idm() As Integer
Public Idj() As Integer
Public Irj() As Integer
Public Psum() As Double
Public Pmm() As Double
Public Pj() As Double
Public Xs() As Double
Public Xm() As Double
Public Fm() As Double
Public XAx_Wd As Double
Public YAx_Wd As Double

Sub TRUSS2D()
On Error GoTo ErrMsg

Const NDJ = 2
Dim NLJ As Integer 'no of joint load

```

```

NJ = Cells(4, 2)
NM = Cells(5, 2)
'X-range:
XAx_Wd = Application.Max(Rows("10:10")) - Application.Min(Rows("10:10"))
'Y-range:
YAx_Wd = Application.Max(Rows("11:11")) - Application.Min(Rows("11:11"))
NS = Application.Count(Rows("22:22"))
NLJ = Application.Count(Rows("27:27"))
'Total number of joint displacement, NP:
NP = NDJ * NJ

ReDim Joint(NJ) As Joint_Data
ReDim Member(NM) As Member_Data
ReDim MS(4, 4, NM) As Double 'member stiffness matrix
ReDim GS(NP, NP) 'global stiffness matrix
ReDim Idm(4, NM) 'displacement index
ReDim Pj(NP) 'joint load vector
ReDim Pmm(NP, NM) 'fixed-end forces
ReDim Psum(NP) 'sum load matrix vector
ReDim Xs(NP) 'joint displacement vector
ReDim Xm(4, NM) 'member deformation
ReDim Fm(4, NM) 'member forces
ReDim Rs(NP)
ReDim spn(NS) As Integer

'=====
'Read Input Data
'=====

'Read joint coordinates
For i = 1 To NJ
    With Joint(i)
        .x = Cells(10, 1 + i)
        .y = Cells(11, 1 + i)
    End With
Next i

```

```

'Read element data
For i = 1 To NM
    With Member(i)
        .J1 = Cells(15, 1 + i)
        .J2 = Cells(16, 1 + i)
        .Dens = Cells(17, 1 + i)
        .Ax = Cells(18, 1 + i)
        .E = Cells(19, 1 + i)
        .lx = Joint(.J2).x - Joint(.J1).x
        .ly = Joint(.J2).y - Joint(.J1).y
        .Lh = Sqr(.lx ^ 2 + .ly ^ 2)
        .Cx = .lx / .Lh
        .Cy = .ly / .Lh
    End With
Next i

If ChartWindow = True Then Call PlotGeometry: Exit Sub
Application.ScreenUpdating = False

'Read joint load
For i = 1 To NLJ
    n = Cells(27, 1 + i)
    Pj(2 * n - 1) = Cells(28, 1 + i)
    Pj(2 * n) = Cells(29, 1 + i)
Next i

'restrain list and support (Rs) index
For i = 1 To NS
    spn(i) = Cells(22, 1 + i)
    Rs(2 * spn(i) - 1) = Cells(23, 1 + i)
    Rs(2 * spn(i)) = Cells(24, 1 + i)
Next i

DOF = 0
For i = 1 To NP
    If Rs(i) = 0 Then DOF = DOF + 1

```

```

Next i

'#restrained joint
NR = NP - DOF

ReDim DFX(NDJ * NS, DOF) As Double '[Kbf]according to Eq. 4.12
ReDim DFR(DOF, DOF) As Double '[Kff]^-1 according to Eq. 4.9
ReDim RJ(NDJ * NS) As Double 'support reactions
ReDim Idj(DOF) 'free displacement index
ReDim Irj(NDJ * NS) 'support displacement index

'=====
'Structure Analysis
'=====

Call Mindex(Idm, Idj, Irj)
Call Stiff_Mtx(MS, GS)
Call MInvers(DFX, DFR)
'generate load
Call Genload(Pmm, Psum)
'determine displacements
Call DISP(Xs, Xm)

'=====
'Print Result
'=====
'clear previous content
Dim LastRow
LastRow = ActiveSheet.UsedRange.Rows.Count
Range(Cells(35, 1), Cells(LastRow, 10)).ClearContents

'1.Print Load
For i = 1 To NJ
    Cells(34 + i, 1).NumberFormat = "0"
    Cells(34 + i, 1) = i
    Cells(34 + i, 2).NumberFormat = "0.000"
    Cells(34 + i, 2) = Psum(2 * i - 1)

```

```

        Cells(34 + i, 3).NumberFormat = "0.000"
        Cells(34 + i, 3) = Psum(2 * i)
Next i

'2. Print Joint displacement
For i = 1 To NJ
    Cells(34 + i, 4).NumberFormat = "0.00000"
    Cells(34 + i, 4) = Xs(2 * i - 1)
    Cells(34 + i, 5).NumberFormat = "0.00000"
    Cells(34 + i, 5) = Xs(2 * i)
Next i

'-----
'Member Forces
'-----
'{F}m=[K]m.{X}m

For i = 1 To NM
    For j = 1 To 4
        Fm(j, i) = 0
        For n = 1 To 4
            Fm(j, i) = Fm(j, i) + MS(j, n, i) * Xm(n, i)
        Next n
    Next j
Next i

'Print member foces
For i = 1 To NM
    With Member(i)
        Cells(34 + i, 6).NumberFormat = "0"
        Cells(34 + i, 6) = i & "./" & Format(.Lh, "0.0")
        Cells(34 + i, 7).NumberFormat = "0.000"
        Cells(34 + i, 7) = Pmm(Idm(1, i), i) + Fm(1, i)
        Cells(34 + i, 8).NumberFormat = "0.000"
        Cells(34 + i, 8) = Pmm(Idm(3, i), i) + Fm(3, i)
    End With
Next i

```

```

'-----
'Support Reactions
'-----

For i = 1 To 2 * NS
    For j = 1 To DOF
        RJ(i) = RJ(i) + DFX(i, j) * Xs(Idj(j))
    Next j
Next i

For i = 1 To NS
    Cells(34 + i, 9).NumberFormat = "0"
    Cells(34 + i, 9) = spn(i)
    Cells(34 + i, 10).NumberFormat = "0.000"
    Cells(34 + i, 10) = RJ(2 * i - 1) - Psum(Irj(2 * i - 1))
    Cells(34 + i, 11).NumberFormat = "0.000"
    Cells(34 + i, 11) = RJ(2 * i) - Psum(Irj(2 * i))
Next i

Call PlotGeometry
Call PlotDisplacement
Call PlotAxial

Range("A1").Select
Application.ScreenUpdating = True
Exit Sub

ErrMsg: MsgBox "Error, please check input ...", vbOKOnly + vbExclamation,
"TRUSS2D"

Range("A1").Select
Application.ScreenUpdating = True

End Sub

```


Module2 (TRUSS2D)

```
Option Explicit
Option Base 1
Public T() As Double 'transformation matrix
Public TT() As Double 'transpose [T], used in Module 3

'Indexing for matrix subscript
Sub Mindex(index() As Integer, IFr() As Integer, IFx() As Integer)

'Member end-displacements index:
For i = 1 To NM
    With Member(i)
        index(1, i) = 2 * .J1 - 1
        index(2, i) = 2 * .J1
        index(3, i) = 2 * .J2 - 1
        index(4, i) = 2 * .J2
    End With
Next i

'Joint free displacement index, IFr and support index, IFx
n = 1
j = 1
For i = 1 To NP
    If Rs(i) = 0 Then IFr(n) = i: n = n + 1
Next i

n = 1
For i = 1 To NS
    n = Cells(22, 1 + i)
    IFx(2 * i - 1) = 2 * n - 1
    IFx(2 * i) = 2 * n
Next i

End Sub
```

```

Sub Stiff_Mtx(MK() As Double, S() As Double)
Dim sms As Double
ReDim M(4, 4, NM) As Double

'Building global stiffness matrix
For i = 1 To NM
    With Member(i)
        sms = .Ax * .E / .Lh
        'Member stiffness
        MK(1, 1, i) = sms: MK(1, 3, i) = -sms
        MK(3, 1, i) = -sms: MK(3, 3, i) = sms
        'Member global stiffness
        M(1, 1, i) = sms * .Cx * .Cx: M(1, 2, i) = sms * .Cx * .Cy: M(1, 3, i) =
-M(1, 1, i): M(1, 4, i) = -M(1, 2, i)
        M(2, 1, i) = M(1, 2, i): M(2, 2, i) = sms * .Cy * .Cy: M(2, 3, i) = M(1,
4, i): M(2, 4, i) = -M(2, 2, i)
        M(3, 1, i) = M(1, 3, i): M(3, 2, i) = M(2, 3, i): M(3, 3, i) = M(1, 1,
i): M(3, 4, i) = M(1, 2, i)
        M(4, 1, i) = M(1, 4, i): M(4, 2, i) = M(2, 4, i): M(4, 3, i) = M(3, 4,
i): M(4, 4, i) = M(2, 2, i)
        'Storing members and superposition
        For j = 1 To 4
            For n = 1 To 4
                S(Idm(j, i), Idm(n, i)) = S(Idm(j, i), Idm(n, i)) + M(j, n, i)
            Next n
        Next j
    End With
Next i
End Sub

'Global matrix
Sub MInvers(SR() As Double, SD() As Double)
Dim n As Integer

'submatrix Stiffness, KRF
For i = 1 To DOF
    For j = 1 To 2 * NS
        SR(j, i) = GS(Irj(j), Idj(i))
    Next j
Next i

```

```

'submatrix Stiffness, KFF
For i = 1 To DOF
    For j = 1 To DOF
        SD(i, j) = GS(Idj(i), Idj(j))
    Next j
Next i

'Inverse []
For i = 1 To DOF
    For j = 1 To DOF
        If j <> i Then SD(i, j) = SD(i, j) / SD(i, i)
    Next j
    For n = 1 To DOF
        If n = i Then GoTo 10
        For j = 1 To DOF
            If j <> i Then SD(n, j) = SD(n, j) - SD(i, j) * SD(n, i)
        Next j
10 Next n
    For n = 1 To DOF
        If n <> i Then SD(n, i) = -SD(n, i) / SD(i, i)
    Next n
    SD(i, i) = 1 / SD(i, i)
Next i

End Sub

Sub Genload(Pa() As Double, Ps() As Double)
    ReDim TT(4, 4, NM) As Double
    ReDim Peq(NP, NM) 'equivalent load due to selfweight
    Dim Idx As Integer, W As Double

    'member fixed-end forces due to selfweight
    For i = 1 To NM
        With Member(i)
            W = .Dens * .Ax
            Pa(Idm(1, i), i) = .Cy * W * .Lh / 2
            Pa(Idm(2, i), i) = .Cx * W * .Lh / 2
        End With
    Next i
End Sub

```

```

        Pa(Idm(3, i), i) = .Cy * W * .Lh / 2
        Pa(Idm(4, i), i) = .Cx * W * .Lh / 2
    End With
Next i

'Transformation matrix (transpose)
For i = 1 To NM
    With Member(i)
        TT(1, 1, i) = .Cx: TT(3, 3, i) = TT(1, 1, i): TT(1, 2, i) = -.Cy: TT(3,
4, i) = TT(1, 2, i)
        TT(2, 1, i) = .Cy: TT(4, 3, i) = TT(2, 1, i): TT(2, 2, i) = .Cx: TT(4, 4,
i) = TT(2, 2, i)
    End With
Next i

'equivalent joint load due to selfweight in global system
For i = 1 To NM
    For j = 1 To 4
        For n = 1 To 4
            Peq(Idm(j, i), i) = Peq(Idm(j, i), i) + TT(j, n, i) * -Pa(Idm(n,
i), i)
        Next n
    Next j
Next i

'sum load = joint load + eq.load
For i = 1 To NP
    Ps(i) = Pj(i)
Next i

'load superposition
For i = 1 To NM
    For j = 1 To 4
        Ps(Idm(j, i)) = Ps(Idm(j, i)) + Peq(Idm(j, i), i)
    Next j
Next i

End Sub

```

```

Sub DISP(x() As Double, X2() As Double)
ReDim X2(4, NM) As Double
ReDim Xi(4, NM) As Double
ReDim T(4, 4, NM) As Double

'Joint Displacement {X} = [SD]^-1.{P}
For i = 1 To DOF
x(Idj(i)) = 0
    For j = 1 To DOF
        x(Idj(i)) = x(Idj(i)) + DFR(i, j) * Psum(Idj(j))
    Next j
Next i

'Numbering of global {X}(NP) to global {Xi} elemen
For i = 1 To NM
    For j = 1 To 4
        Xi(j, i) = x(Idm(j, i))
    Next j
Next i

'Transforming {Xi} to local coordinates
'Transformation matrix:
For i = 1 To NM
    With Member(i)
        T(1, 1, i) = .Cx: T(3, 3, i) = T(1, 1, i): T(1, 2, i) = .Cy: T(3, 4, i) =
T(1, 2, i)
        T(2, 1, i) = -.Cy: T(4, 3, i) = T(2, 1, i): T(2, 2, i) = .Cx: T(4, 4, i)
= T(2, 2, i)
    End With
Next i

'member deformation, tranformed{X} = [T].{Xi}
For i = 1 To NM
    For j = 1 To 4
        X2(j, i) = 0
        For n = 1 To 4
            X2(j, i) = X2(j, i) + T(j, n, i) * Xi(n, i)
        Next n
    Next j
Next i

```

```

        Next j
    Next i

End Sub

```

Module3 (TRUSS2D)

```

'This module consist of only code to create charts
Option Explicit
Option Base 1
Sub PlotGeometry()

On Error Resume Next

Dim Cx1, Cx2, Cy1, Cy2
ReDim mxVal(NM) As Variant
ReDim myVal(NM) As Variant
ReDim tagxVal(NM) As Variant
ReDim tagyVal(NM) As Variant

Dim am As Integer

'member coordinates
For i = 1 To NM
    With Member(i)
        Cx1 = Joint(.J1).x
        Cy1 = Joint(.J1).y
        Cx2 = Joint(.J2).x
        Cy2 = Joint(.J2).y
    End With
    mxVal(i) = Array(Cx1, Cx2)
    myVal(i) = Array(Cy1, Cy2)
    tagxVal(i) = Array((Cx1 + Cx2) / 2)
    tagyVal(i) = Array((Cy1 + Cy2) / 2)
Next i

ActiveSheet.ChartObjects("Chart 1").Activate

```

```

ActiveChart.ChartArea.Select
Selection.ClearContents

'creating joints
For i = 1 To NJ
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = Joint(i).x
        .SeriesCollection(i).Values = Joint(i).y
        .SeriesCollection(i).Name = "J" & i
    End With
    'node marker
    ActiveChart.PlotArea.Select
    ActiveChart.SeriesCollection(i).Select
    With Selection
        .MarkerBackgroundColorIndex = 3 'red
        .MarkerForegroundColorIndex = 3
        .MarkerStyle = xlCircle
        .MarkerSize = 6
        .Shadow = False
        .ApplyDataLabels AutoText:=True, LegendKey:= _
            False, ShowSeriesName:=True, ShowCategoryName:=False,
ShowValue:=False, _
            ShowPercentage:=False, ShowBubbleSize:=False
    End With

    ActiveChart.SeriesCollection(i).DataLabels.Select
    With Selection
        .HorizontalAlignment = xlCenter
        .VerticalAlignment = xlCenter
        .ReadingOrder = xlContext
        .Position = xlLabelPositionRight
        .Orientation = xlHorizontal
    End With
    ActiveChart.SeriesCollection(i).DataLabels.Select
    With Selection.Font
        .Name = "Arial"
        .FontStyle = "Regular"
    End With
End For

```



```

        .SeriesCollection(i).XValues = tagxVal(am)
        .SeriesCollection(i).Values = tagyVal(am)
        .SeriesCollection(i).Name = "M" & am
    End With
    ActiveChart.PlotArea.Select
    ActiveChart.SeriesCollection(i).Select
    With Selection
        .MarkerBackgroundColorIndex = 5
        .MarkerForegroundColorIndex = 5
        .MarkerStyle = xlSquare
        .Smooth = False
        .MarkerSize = 6
        .Shadow = False
        .ApplyDataLabels AutoText:=True, LegendKey:= _
            False,          ShowSeriesName:=True,          ShowCategoryName:=False,
ShowValue:=False, _
            ShowPercentage:=False, ShowBubbleSize:=False
    End With
    ActiveChart.SeriesCollection(i).DataLabels.Select
    With Selection.Font
        .Name = "Arial"
        .FontStyle = "Regular"
        .Size = 8
    End With
    am = am + 1
Next i

    ActiveChart.ChartArea.Select
    With ActiveChart
        .Axes(xlCategory, xlPrimary).HasTitle = True
        .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = _
            "NJ = " & Format(NJ, "0") & ", NM = " & Format(NM, "0")
    End With

    ActiveChart.ShowWindow = True

End Sub

```

```

Sub PlotDisplacement()
On Error Resume Next

Dim Cx1, Cx2, Cy1, Cy2
ReDim mxValbf(NM) As Variant, myValbf(NM) As Variant
ReDim mxValaf(NM) As Variant, myValaf(NM) As Variant

Dim am As Integer, scx As Single
Dim Xmax As Double, at_joint As Integer

    ActiveSheet.ChartObjects("Chart 2").Activate
    ActiveChart.ChartArea.Select
    Selection.ClearContents

'=====
'scale for displacement
'=====
'user scale: max displacement are drawn up to 10% of the axis range
'or follows formula: X-range or Y-range * 10% = Xmax

'determine max. joint-X and Y displacement
'(user function UXMax written at the end of this module):
Xmax = Application.index(UXMax(Xs, NP), 1)
at_joint = Application.index(UXMax(Xs, NP), 2)
scx = (Application.Max(XAx_Wd, YAx_Wd) * 10 / 100) / Abs(Xmax)

'member coordinates: before & after loading
For i = 1 To NM
    With Member(i)
        Cx1 = Joint(.J1).x
        Cy1 = Joint(.J1).y
        Cx2 = Joint(.J2).x
        Cy2 = Joint(.J2).y
        mxValbf(i) = Array(Cx1, Cx2)
        myValbf(i) = Array(Cy1, Cy2)
        Cx1 = Joint(.J1).x + scx * Xs(2 * .J1 - 1)
        Cy1 = Joint(.J1).y + scx * Xs(2 * .J1)
    End With

```

```

        Cx2 = Joint(.J2).x + scx * Xs(2 * .J2 - 1)
        Cy2 = Joint(.J2).y + scx * Xs(2 * .J2)
    mxValaf(i) = Array(Cx1, Cx2)
    myValaf(i) = Array(Cy1, Cy2)
End With
Next i

am = 1
'creating member lines: before loading
For i = 1 To NM
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = mxValbf(am)
        .SeriesCollection(i).Values = myValbf(am)
        .SeriesCollection(i).Name = " " " " " " " "
    End With
    ActiveChart.SeriesCollection(i).Select
    With Selection
        .MarkerStyle = xlNone
        .Smooth = False
    End With
    With Selection.Border
        .ColorIndex = 5
        .Weight = xlThin
        .LineStyle = xlContinuous
    End With
    am = am + 1
Next i

n = NM + NM
am = 1
'creating member lines: after loading
For i = NM + 1 To n
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = mxValaf(am)
        .SeriesCollection(i).Values = myValaf(am)

```

```

        .SeriesCollection(i).Name = "=""=""
    End With
    ActiveChart.SeriesCollection(i).Select
    With Selection
        .MarkerStyle = xlNone
        .Smooth = False
    End With
    With Selection.Border
        .ColorIndex = 7
        .Weight = xlMedium
        .LineStyle = xlContinuous
    End With
    am = am + 1
Next i

ActiveChart.ChartArea.Select
With ActiveChart
    .Axes(xlCategory, xlPrimary).HasTitle = True
    .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = _
        "Max Displacement, J." & at_joint & " = " & Format(Xmax, "0.00000")
End With

End Sub

Sub PlotAxial()
On Error Resume Next

Dim Cx1, Cx2, Cy1, Cy2
Dim Cmx1, Cmx2, Cmy1, Cmy2
ReDim mxVal(NM) As Variant
ReDim myVal(NM) As Variant
ReDim saxVal(NM) As Variant
ReDim sayVal(NM) As Variant
ReDim tagxVal(NM) As Variant
ReDim tagyVal(NM) As Variant

ReDim mjsxVal(NM + NM) As Variant

```

```

ReDim mpyVal(NM + NM) As Variant
ReDim acolor(NM + NM) As Variant
ReDim Pa_LM(2, NM) As Double
ReDim Pa_RM(2, NM) As Double
ReDim Pa_LT(2, NM) As Double
ReDim Pa_RT(2, NM) As Double

Dim am As Integer, sca As Single
Dim Amax As Double, at_member As Integer, L_I As Integer, L_II As Integer

    ActiveSheet.ChartObjects("Chart 3").Activate
    ActiveChart.ChartArea.Select
    Selection.ClearContents

'analogueous to plot displacement for scaling
Amax = Application.index(UAMax(Pmm, Fm, NM), 1)
at_member = Application.index(UAMax(Pmm, Fm, NM), 2)
sca = (Application.Max(XAx_Wd, YAx_Wd) * 10 / 100) / Abs(Amax)

For i = 1 To NM
    Pa_LM(2, i) = -(Pmm(Idm(1, i), i) + Fm(1, i))
    Pa_RM(2, i) = (Pmm(Idm(3, i), i) + Fm(3, i))
Next i

'transform member axial to global axis by using [TT]
'the drawing should be in global coordinates
For i = 1 To NM
    For j = 1 To 2
        For n = 1 To 2
            Pa_LT(j, i) = Pa_LT(j, i) + TT(j, n, i) * Pa_LM(n, i)
            Pa_RT(j, i) = Pa_RT(j, i) + TT(j, n, i) * Pa_RM(n, i)
        Next n
    Next j
Next i

For i = 1 To NM
    With Member(i)

```

```

Cx1 = Joint(.J1).x
Cy1 = Joint(.J1).y
Cx2 = Joint(.J2).x
Cy2 = Joint(.J2).y
'assign force coordinates
Cmx1 = Cx1 + sca * Pa_LT(1, i)
Cmy1 = Cy1 + sca * Pa_LT(2, i)
Cmx2 = Cx2 + sca * Pa_RT(1, i)
Cmy2 = Cy2 + sca * Pa_RT(2, i)
End With
'member coordinates
mxVal(i) = Array(Cx1, Cx2)
myVal(i) = Array(Cy1, Cy2)
'axial coordinates
saxVal(i) = Array(Cmx1, Cmx2)
sayVal(i) = Array(Cmy1, Cmy2)
'member-joint coordinates
mjsxVal(2 * i - 1) = Array(Cx1, Cmx1)
mjyVal(2 * i - 1) = Array(Cy1, Cmy1)
'mcolor: define color for positive/negative value
acolor(2 * i - 1) = -(Pmm(Idm(1, i), i) + Fm(1, i))

mjsxVal(2 * i) = Array(Cx2, Cmx2)
mjyVal(2 * i) = Array(Cy2, Cmy2)
acolor(2 * i) = Pmm(Idm(3, i), i) + Fm(3, i)
Next i

'creating member lines
For i = 1 To NM
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = mxVal(i)
        .SeriesCollection(i).Values = myVal(i)
        .SeriesCollection(i).Name = "=""=""
    End With
    ActiveChart.SeriesCollection(i).Select
    With Selection

```



```
'user function to determine maximum value of
'joint X1 and X2
```

```
Function UXMax(Xs, NP) As Variant
```

```
Dim Abs_FXMax As Double
```

```
Dim FXMax As Double
```

```
ReDim Idis(NP) As Double
```

```
'Joint displacement index
```

```
n = 1
```

```
For i = 2 To NP Step 2
```

```
    Idis(i - 1) = n
```

```
    Idis(i) = n
```

```
n = n + 1
```

```
Next i
```

```
Abs_FXMax = 0
```

```
n = 0
```

```
For i = 1 To NP
```

```
    If Abs(Xs(i)) > Abs_FXMax Then
```

```
        Abs_FXMax = Abs(Xs(i))
```

```
        FXMax = Xs(i)
```

```
        n = Idis(i)
```

```
        UXMax = Array(FXMax, n)
```

```
        'UXMax keeps origin value(+ or - value)
```

```
    End If
```

```
n = n + i
```

```
Next i
```

```
End Function
```

```
'user function to determine maximum value of
```

```
'axial force, F1
```

```
Function UAMax(Pmm, Fm, NM) As Variant
```

```
Dim Abs_FAMax As Double
```

```

Dim FAMax As Double

Abs_FAMax = 0
n = 0
For i = 1 To NM
    If Abs(Pmm(Idm(1, i), i) + Fm(1, i)) > Abs_FAMax Then
        Abs_FAMax = Abs(Pmm(Idm(1, i), i) + Fm(1, i))
        FAMax = -(Pmm(Idm(1, i), i) + Fm(1, i))
        n = i
        UAMax = Array(FAMax, n)
    End If
    If Abs(Pmm(Idm(3, i), i) + Fm(3, i)) > Abs_FAMax Then
        Abs_FAMax = Abs(Pmm(Idm(3, i), i) + Fm(3, i))
        FAMax = Pmm(Idm(3, i), i) + Fm(3, i)
        n = i
        UAMax = Array(FAMax, n)
    End If
Next i
End Function

```

Command Button (TRUSS2D)

```

Private Sub CommandButton1_Click()
    ChartWindow = True
    Call TRUSS2D
End Sub

Private Sub CommandButton2_Click()
    ChartWindow = False
    Call TRUSS2D
End Sub

```

Module1 (BOF)

```
'=====
'BOF v.1.0 Program for Beam on Elastic Foundation, 2004
'by: Gunthar Pangaribuan - Refer to the book:
'An Introduction to: Excel for Civil Engineers
'=====
```

Option Explicit

Public i As Integer, j As Integer, n As Integer

Type Joint_Data

 x As Double

 y As Double

 Ks As Double

 Sp As Double

End Type

Type Member_Data

 J1 As Integer

 J2 As Integer

 tm As Double

 bm As Double

 EI As Double

 lx As Double

 ly As Double

 Lh As Double

 Cx As Double

 Cy As Double

 Km1 As Double

 Km2 As Double

 Pres As Double

End Type

Public ChartWindow As Boolean

Public NP As Integer

```

Public DOF As Integer
Public NR As Integer
Public NJ As Integer
Public NM As Integer
Public Member() As Member_Data
Public Joint() As Joint_Data
Public Em As Double
Public Km() As Double
Public NS As Integer 'no. of support
Public GASAT() As Double 'global stiffness matrix
Public Rs() As Integer
Public DFR() As Double
Public Idm() As Integer
Public Idj() As Integer
Public Irj() As Integer
Public Psum() As Double
Public Pmm() As Double
Public Pj() As Double
Public Xs() As Double
Public Xm() As Double
Public Fm() As Double
Public mmomen1() As Double
Public mmomen2() As Double
Public shear1() As Double
Public shear2() As Double

Sub BOF()
On Error GoTo ErrMsg

Const NDJ = 2
Dim NLJ As Integer 'no of joint load
Dim Ksn As Integer

NJ = Cells(4, 2)
NM = Cells(5, 2)
Em = Cells(6, 2)

```

```

NS = Application.Count(Rows("23:23"))
NLJ = Application.Count(Rows("28:28"))
Ksn = Application.Count(Rows("12:12"))

'Total number of joint displacement, NP:
NP = NDJ * NJ

ReDim Joint(NJ) As Joint_Data
ReDim Member(NM) As Member_Data
ReDim ESAT(4, 4, NM) As Double 'member matrix refer to the book
ReDim GASAT(NP, NP) 'global stiffness matrix refer to the book
ReDim Idm(4, NM) ' displacement index
ReDim Pj(NP) 'joint load vector

ReDim Pmm(NP, NM) 'fixed-end forces
ReDim Psum(NP) 'sum load vector
ReDim Xs(NP) 'joint displacement vector
ReDim Xm(4, NM) ' member deformation
ReDim Fm(4, NM) 'member forces

ReDim mmomen1(NM)
ReDim mmomen2(NM)
ReDim shear1(NM)
ReDim shear2(NM)
ReDim Rs(NP)
ReDim spn(NS) As Integer
Dim mshear As Double

'=====
'Read Input Data
'=====

'Read joint data - in order
For i = 1 To NJ
    With Joint(i)
        .x = Cells(10, 1 + i)
        .y = Cells(11, 1 + i)
    End With

```

```

        .Ks = Cells(12, 1 + i)
        .Sp = .Ks * Xs(2 * i) 'soil pressure
    End With
Next i

'Read member data
For i = 1 To NM
    With Member(i)
        .J1 = Cells(16, 1 + i)
        .J2 = Cells(17, 1 + i)
        .tm = Cells(18, 1 + i)
        .bm = Cells(19, 1 + i)
        .EI = Em * 1 / 12 * .bm * .tm ^ 3
        .lx = Joint(.J2).x - Joint(.J1).x
        .ly = Joint(.J2).y - Joint(.J1).y
        .Lh = Sqr(.lx ^ 2 + .ly ^ 2)
        .Cx = .lx / .Lh
        .Cy = .ly / .Lh
        .Pres = Cells(20, 1 + i)
        .Km1 = 0.5 * .Lh * .bm * Joint(.J1).Ks
        .Km2 = 0.5 * .Lh * .bm * Joint(.J2).Ks
    End With
Next i

'end spring:
With Member(1)
    .Km1 = .Lh * .bm * Joint(.J1).Ks
End With
With Member(NM)
    .Km2 = .Lh * .bm * Joint(.J2).Ks
End With

If ChartWindow = True Then Call PlotGeometry: Exit Sub
Application.ScreenUpdating = False

'Read joint load
For i = 1 To NLJ
    n = Cells(28, 1 + i)

```

```

    Pj(2 * n - 1) = Cells(29, 1 + i)
    Pj(2 * n) = Cells(30, 1 + i)
Next i

'restrain list and support (Rs) index
For i = 1 To NS
    spn(i) = Cells(23, 1 + i)
    Rs(2 * spn(i) - 1) = Cells(24, 1 + i)
    Rs(2 * spn(i)) = Cells(25, 1 + i)
Next i

DOF = 0
For i = 1 To NP
    If Rs(i) = 0 Then DOF = DOF + 1
Next i

'#restrained joint (zero displacement)
NR = NP - DOF

ReDim DFX(NDJ * NS, DOF) As Double '[Kbf]according to Eq. 4.12
ReDim DFR(DOF, DOF) As Double '[Kff]^-1 according to Eq. 4.9
ReDim RJ(NDJ * NS) As Double 'support reactions
ReDim Idj(DOF) 'free displacement index
ReDim Irj(NDJ * NS) 'support displacement index
ReDim nAs_Now(NJ) As Integer
Dim nAs_Last As Integer
Dim ITR As Integer
Dim ITR_Continue As Boolean
ITR_Continue = True

ITR = 1

'clear previous content
Dim LastRow
LastRow = ActiveSheet.UsedRange.Rows.Count
Range(Cells(38, 1), Cells(LastRow, 16)).ClearContents

```

```

'=====
'Structure Analysis
'=====

Do

Call Mindex(Idm, Idj, Irj)
Call FEM(ESAT, GASAT)
Call MInvers(DFX, DFR)
Call Genload(Pmm, Psum)
Call DISP(Xs, Xm)

'Print used springs
For i = 1 To NM
    With Member(i)
        Cells(37 + i, 8).NumberFormat = "0.000"
        Cells(37 + i, 8) = .Km1
        Cells(37 + i, 9).NumberFormat = "0.000"
        Cells(37 + i, 9) = .Km2
    End With
Next i

n = 0
For i = 1 To NJ
're-input joint's Ks and Sp if Xs < 0
    With Joint(i)
        If Xs(i * 2) < 0 And .Ks > 0 Then
            .Ks = 0
            .Sp = .Ks * Xs(2 * i)
            n = n + 1
        End If
    End With
Next i

're-input member's soil K
For i = 1 To NM
    With Member(i)
        .Km1 = 0.5 * .Lh * .bm * Joint(.J1).Ks
    End With
Next i

```



```

        .Km2 = 0.5 * .Lh * .bm * Joint(.J2).Ks
    End With
Next i
With Member(1)
    .Km1 = .Lh * .bm * Joint(.J1).Ks
End With
With Member(NM)
    .Km2 = .Lh * .bm * Joint(.J2).Ks
End With

'check no. of active spring between 2 iterations
nAs_Now(ITR) = NJ - n
nAs_Last = nAs_Now(ITR - 1)

If nAs_Now(ITR) = nAs_Last Or n = 0 Then
    ITR_Continue = False
    Cells(34, 3) = ITR - 1
    Cells(35, 3) = nAs_Now(ITR)
End If

ITR = ITR + 1

Loop Until ITR_Continue = False

'=====
'Print Result
'=====
'1. Print Load
For i = 1 To NJ
    Cells(37 + i, 1).NumberFormat = "0"
    Cells(37 + i, 1) = i
    Cells(37 + i, 2).NumberFormat = "0.000"
    Cells(37 + i, 2) = Psum(2 * i - 1)
    Cells(37 + i, 3).NumberFormat = "0.000"
    Cells(37 + i, 3) = Psum(2 * i)
Next i

```

```

'2. Print Joint displacement and soil pressure
For i = 1 To NJ
    Cells(37 + i, 4).NumberFormat = "0.00000"
    Cells(37 + i, 4) = Xs(2 * i - 1)
    Cells(37 + i, 5).NumberFormat = "0.00000"
    Cells(37 + i, 5) = Xs(2 * i)
    Cells(37 + i, 6).NumberFormat = "0.000"
    With Joint(i)
        .Sp = .Ks * Xs(2 * i)
        Cells(37 + i, 6) = .Sp
    End With
Next i

'-----
'Member Forces
'-----
'{F}m=[S]m.{X}s or [F]m=[S].[A]T.{X}s
'there is already calculated S.A^T,
For i = 1 To NM
    For j = 1 To 4
        Fm(j, i) = 0
        For n = 1 To 4
            Fm(j, i) = Fm(j, i) + ESAT(j, n, i) * Xm(n, i)
        Next n
    Next j
Next i

'Print member foces
For i = 1 To NM
    With Member(i)
        'no/length
        Cells(37 + i, 7).NumberFormat = "0"
        Cells(37 + i, 7) = i & " / " & Format(.Lh, "0.00")
        'moment
        mmomen1(i) = Fm(1, i) + Pmm(Idm(1, i), i)
        Cells(37 + i, 10).NumberFormat = "0.000"
        Cells(37 + i, 10) = mmomen1(i)
    End With
Next i

```

```

mmomen2(i) = Fm(2, i) + Pmm(Idm(3, i), i)
Cells(37 + i, 11).NumberFormat = "0.000"
Cells(37 + i, 11) = mmomen2(i)
'spring
Cells(37 + i, 12).NumberFormat = "0.000"
Cells(37 + i, 12) = Fm(3, i)
Cells(37 + i, 13).NumberFormat = "0.000"
Cells(37 + i, 13) = Fm(4, i)
'shear
If Ksn = 0 Then
    'internal forces with support
    mshear = (Fm(1, i) + Fm(2, i)) / .Lh
    shear1(i) = mshear + Pmm(Idm(2, i), i)
    shear2(i) = -mshear + Pmm(Idm(4, i), i)
Else
    'as per reference - with springs
    mshear = (Fm(1, i) + Fm(2, i)) / .Lh
    shear1(i) = mshear
    shear2(i) = -mshear
End If
Cells(37 + i, 14).NumberFormat = "0.000"
Cells(37 + i, 14) = shear1(i)
Cells(37 + i, 15).NumberFormat = "0.000"
Cells(37 + i, 15) = shear2(i)
End With
Next i

'-----
'Support Reactions
'-----

For i = 1 To 2 * NS
    For j = 1 To DOF
        RJ(i) = RJ(i) + DFX(i, j) * Xs(Idj(j))
    Next j
Next i

For i = 1 To NS

```

```

Cells(37 + i, 16).NumberFormat = "0"
Cells(37 + i, 16) = spn(i)
Cells(37 + i, 17).NumberFormat = "0.000"
Cells(37 + i, 17) = RJ(2 * i - 1) - Psum(Irj(2 * i - 1))
Cells(37 + i, 18).NumberFormat = "0.000"
Cells(37 + i, 18) = RJ(2 * i) - Psum(Irj(2 * i))
Next i

'create graphs
Call PlotGeometry
Call PlotDisplacement
Call PlotMoment
Call PlotShear
Call PlotSoilPressure

Range("A1").Select
Application.ScreenUpdating = True
Exit Sub

ErrMsg: MsgBox "Error, please check input ...", vbOKOnly + vbExclamation,
"BOF"

Range("A1").Select
Application.ScreenUpdating = True
End Sub

```

Module2 (BOF)

```

Option Explicit
'Indexing for matrix subscript
Sub Mindex(index() As Integer, IFr() As Integer, IFx() As Integer)

'Member end-displacements index:
For i = 1 To NM
    With Member(i)
        index(1, i) = 2 * .J1 - 1
        index(2, i) = 2 * .J1
    End With

```

```

        index(3, i) = 2 * .J2 - 1
        index(4, i) = 2 * .J2
    End With
Next i

'Joint free displacement index, IFr and
'restraint (support) index, IFx
n = 1
For i = 1 To NP
    If Rs(i) = 0 Then IFr(n) = i: n = n + 1
Next i

n = 1
For i = 1 To NS
    n = Cells(23, 1 + i)
    IFx(2 * i - 1) = 2 * n - 1
    IFx(2 * i) = 2 * n
Next i

End Sub

Sub FEM(SAT() As Double, GS() As Double)
    ReDim ASAT(4, 4, NM) As Double

    'Assembly finite element matrix, Eq.4.16 & 4.17
    For i = 1 To NM
        With Member(i)
            'member SA^T matrix:
            'from internal forces F - deformation d relationship, F = S.d or F =
            S.A^T.X
            SAT(1, 1, i) = 4 * .EI / .Lh: SAT(1, 2, i) = 6 * .EI / .Lh ^ 2
            SAT(1, 3, i) = 2 * .EI / .Lh: SAT(1, 4, i) = -6 * .EI / .Lh ^ 2
            SAT(2, 1, i) = 2 * .EI / .Lh: SAT(2, 2, i) = 6 * .EI / .Lh ^ 2
            SAT(2, 3, i) = 4 * .EI / .Lh: SAT(2, 4, i) = -6 * .EI / .Lh ^ 2
            SAT(3, 2, i) = .Km1
            SAT(4, 4, i) = .Km2
        End With
        'Member global stiffness (member ASA^T)
        ASAT(1, 1, i) = 4 * .EI / .Lh: ASAT(1, 2, i) = 6 * .EI / .Lh ^ 2
    
```

```

    ASAT(1, 3, i) = 2 * .EI / .Lh: ASAT(1, 4, i) = -6 * .EI / .Lh ^ 2
    ASAT(2, 1, i) = 6 * .EI / .Lh ^ 2: ASAT(2, 2, i) = 12 * .EI / .Lh ^ 3 +
.Km1
    ASAT(2, 3, i) = 6 * .EI / .Lh ^ 2: ASAT(2, 4, i) = -12 * .EI / .Lh ^ 3
    ASAT(3, 1, i) = 2 * .EI / .Lh: ASAT(3, 2, i) = 6 * .EI / .Lh ^ 2
    ASAT(3, 3, i) = 4 * .EI / .Lh: ASAT(3, 4, i) = -6 * .EI / .Lh ^ 2
    ASAT(4, 1, i) = -6 * .EI / .Lh ^ 2: ASAT(4, 2, i) = -12 * .EI / .Lh ^ 3
    ASAT(4, 3, i) = -6 * .EI / .Lh ^ 2: ASAT(4, 4, i) = 12 * .EI / .Lh ^ 3 +
.Km2

    'Storing members and superposition(global ASA^T)
    For j = 1 To 4
        For n = 1 To 4
            GS(Idm(j, i), Idm(n, i)) = GS(Idm(j, i), Idm(n, i)) + ASAT(j, n,
i)
        Next n
    Next j
    End With
Next i

End Sub

'Global matrix
Sub MInvers(SR() As Double, SD() As Double)
    Dim n As Integer

    'submatrix Stiffness, KRF
    For i = 1 To DOF
        For j = 1 To 2 * NS
            SR(j, i) = GASAT(Irj(j), Idj(i))
        Next j
    Next i

    'submatrix Stiffness, KBF
    For i = 1 To DOF
        For j = 1 To DOF
            SD(i, j) = GASAT(Idj(i), Idj(j))
        Next j
    Next i

```

```

Inverse []
  For i = 1 To DOF
    For j = 1 To DOF
      If j <> i Then SD(i, j) = SD(i, j) / SD(i, i)
    Next j
    For n = 1 To DOF
      If n = i Then GoTo 10
      For j = 1 To DOF
        If j <> i Then SD(n, j) = SD(n, j) - SD(i, j) * SD(n, i)
      Next j
10  Next n
    For n = 1 To DOF
      If n <> i Then SD(n, i) = -SD(n, i) / SD(i, i)
    Next n
    SD(i, i) = 1 / SD(i, i)
  Next i

```

End Sub

```
Sub Genload(Pa() As Double, Ps() As Double)
```

```

ReDim TT(4, 4, NM) As Double
ReDim Peq(NP, NM) 'equivalent load
Dim Idx As Integer, w As Double

```

```
'member fixed-end forces due to pressure
```

```

For i = 1 To NM
  With Member(i)
    w = .Pres * .bm
    Pa(Idm(1, i), i) = -w * .Lh ^ 2 / 12
    Pa(Idm(2, i), i) = -w * .Lh / 2
    Pa(Idm(3, i), i) = w * .Lh ^ 2 / 12
    Pa(Idm(4, i), i) = -w * .Lh / 2
  End With
Next i

```

```
'equivalent joint load due to pressure
```

```

For i = 1 To NM
    For j = 1 To 4
        For n = 1 To 4
            Peq(Idm(j, i), i) = -Pa(Idm(j, i), i)
        Next n
    Next j
Next i

'sum load = joint load + eq.load
For i = 1 To NP
    Ps(i) = Pj(i)
Next i

'load superposition
For i = 1 To NM
    For j = 1 To 4
        Ps(Idm(j, i)) = Ps(Idm(j, i)) + Peq(Idm(j, i), i)
    Next j
Next i

End Sub

Sub DISP(x() As Double, Xi() As Double)
    ReDim T(4, 4, NM) As Double

    'Joint Displacement {X} = [SD]^-1.{P}
    For i = 1 To DOF
        x(Idj(i)) = 0
        For j = 1 To DOF
            x(Idj(i)) = x(Idj(i)) + DFR(i, j) * Psum(Idj(j))
        Next j
    Next i

    'Numbering of global {X}(NP) to global {Xi} member
    For i = 1 To NM
        For j = 1 To 4
            Xi(j, i) = x(Idm(j, i))
        Next j
    Next i

```



```

        Next j
    Next i

End Sub

```

Module3 (BOF)

```

Option Explicit
Option Base 1
Sub PlotGeometry()

On Error Resume Next

Dim Cx1, Cx2, Cy1, Cy2
ReDim mxVal(NM) As Variant
ReDim myVal(NM) As Variant
ReDim tagxVal(NM) As Variant
ReDim tagyVal(NM) As Variant

Dim am As Integer

'member coordinates
For i = 1 To NM
    With Member(i)
        Cx1 = Joint(.J1).x
        Cy1 = Joint(.J1).y
        Cx2 = Joint(.J2).x
        Cy2 = Joint(.J2).y
    End With
    mxVal(i) = Array(Cx1, Cx2)
    myVal(i) = Array(Cy1, Cy2)
    tagxVal(i) = Array((Cx1 + Cx2) / 2)
    tagyVal(i) = Array((Cy1 + Cy2) / 2)
Next i

ActiveSheet.ChartObjects("Chart 1").Activate
ActiveChart.ChartArea.Select

```

```

Selection.ClearContents

'creating joints + name tags
For i = 1 To NJ
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = Joint(i).x
        .SeriesCollection(i).Values = Joint(i).y
        .SeriesCollection(i).Name = "J" & i
    End With
    'node marker
    ActiveChart.PlotArea.Select
    ActiveChart.SeriesCollection(i).Select
    With Selection
        .MarkerBackgroundColorIndex = 3 'red
        .MarkerForegroundColorIndex = 3
        .MarkerStyle = xlCircle
        .MarkerSize = 6
        .Shadow = False
        .ApplyDataLabels AutoText:=True, LegendKey:= _
            False, ShowSeriesName:=True, ShowCategoryName:=False,
ShowValue:=False, _
            ShowPercentage:=False, ShowBubbleSize:=False
    End With

    ActiveChart.SeriesCollection(i).DataLabels.Select
    With Selection
        .HorizontalAlignment = xlCenter
        .VerticalAlignment = xlCenter
        .ReadingOrder = xlContext
        .Position = xlLabelPositionAbove
        .Orientation = xlHorizontal
        .Font.Name = "Arial"
        .Font.FontStyle = "Regular"
        .Font.Size = 8
    End With
Next i

ActiveSheet.ChartObjects("Chart 1").Activate

```



```

ActiveChart.SeriesCollection(i).Select
With Selection
    .MarkerBackgroundColorIndex = 5
    .MarkerForegroundColorIndex = 5
    .MarkerStyle = xlSquare
    .Smooth = False
    .MarkerSize = 6
    .Shadow = False
    .ApplyDataLabels AutoText:=True, LegendKey:= _
        False,          ShowSeriesName:=True,          ShowCategoryName:=False,
ShowValue:=False, _
        ShowPercentage:=False, ShowBubbleSize:=False
End With
ActiveChart.SeriesCollection(i).DataLabels.Select
With Selection
    .HorizontalAlignment = xlCenter
    .VerticalAlignment = xlCenter
    .Position = xlLabelPositionAbove
    .Orientation = xlHorizontal
    .Font.Name = "Arial"
    .Font.FontStyle = "Regular"
    .Font.Size = 8
End With
am = am + 1
Next i

ActiveChart.ChartArea.Select
With ActiveChart
    .Axes(xlCategory, xlPrimary).HasTitle = True
    .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = _
        "NJ = " & Format(NJ, "0") & ", NM = " & Format(NM, "0")
End With

ActiveChart.ShowWindow = True

End Sub

```

```

Sub PlotDisplacement()
On Error Resume Next

Dim Cx1, Cx2, Cy1, Cy2
ReDim mxValbf(NM) As Variant, myValbf(NM) As Variant
ReDim mxValaf(NM) As Variant, myValaf(NM) As Variant

Dim Xmax As Double, At_joint As Integer, am As Integer

    ActiveSheet.ChartObjects("Chart 2").Activate
    ActiveChart.ChartArea.Select
    Selection.ClearContents

Xmax = Application.index(UXMax(Xs, NP), 1)
At_joint = Application.index(UXMax(Xs, NP), 2)

'member coordinates: before & after loading
For i = 1 To NM
    With Member(i)
        Cx1 = Joint(.J1).x
        Cy1 = Joint(.J1).y
        Cx2 = Joint(.J2).x
        Cy2 = Joint(.J2).y
        mxValbf(i) = Array(Cx1, Cx2)
        myValbf(i) = Array(Cy1, Cy2)
        Cx1 = Joint(.J1).x + .Cy * Xs(2 * .J1)
        Cy1 = Joint(.J1).y + .Cx * Xs(2 * .J1)
        Cx2 = Joint(.J2).x + .Cy * Xs(2 * .J2)
        Cy2 = Joint(.J2).y + .Cx * Xs(2 * .J2)
        mxValaf(i) = Array(Cx1, Cx2)
        myValaf(i) = Array(Cy1, Cy2)
    End With
Next i

    am = 1
    'creating member lines: before loading
    For i = 1 To NM

```



```

        .LineStyle = xlContinuous
    End With
    am = am + 1
Next i

ActiveChart.ChartArea.Select
With ActiveChart
    .Axes(xlCategory, xlPrimary).HasTitle = True
    .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = _
    "Max Displacement, J." & At_joint & " = " & Format(Xmax, "0.00000")
End With

End Sub

Sub PlotMoment()
On Error Resume Next

Dim Cx1, Cx2, Cy1, Cy2
Dim Cmx1, Cmx2, Cmy1, Cmy2
ReDim mxVal(NM) As Variant
ReDim myVal(NM) As Variant
ReDim moxVal(NM) As Variant
ReDim moyVal(NM) As Variant
ReDim tagxVal(NM) As Variant
ReDim tagyVal(NM) As Variant

ReDim mjsxVal(NM + NM) As Variant
ReDim mjyVal(NM + NM) As Variant
ReDim mcolor(NM + NM) As Variant

Dim Mmax As Single, At_joint As Integer, am As Integer, L_I As Integer, L_II
As Integer, _
L_III As Integer, L_IV As Integer

    ActiveSheet.ChartObjects("Chart 3").Activate
    ActiveChart.ChartArea.Select
    Selection.ClearContents

```

```

Mmax = Application.index(UMMax(Pmm, Fm, NM), 1)
At_joint = Application.index(UMMax(Pmm, Fm, NM), 2)

For i = 1 To NM
    With Member(i)
        Cx1 = Joint(.J1).x
        Cy1 = Joint(.J1).y
        Cx2 = Joint(.J2).x
        Cy2 = Joint(.J2).y
        'assign coordinates
        Cmx1 = Cx1
        Cmy1 = Cy1 + mmomen1(i)
        Cmx2 = Cx2
        Cmy2 = Cy2 - mmomen2(i)
    End With
    'member coordinates
    mxVal(i) = Array(Cx1, Cx2)
    myVal(i) = Array(Cy1, Cy2)
    'moment coordinates
    moxVal(i) = Array(Cmx1, Cmx2)
    moyVal(i) = Array(Cmy1, Cmy2)
    'member-joint coordinates
    mjxVal(2 * i - 1) = Array(Cx1, Cmx1)
    mjyVal(2 * i - 1) = Array(Cy1, Cmy1)
    'mcolor: define color for positive/negative value
    mcolor(2 * i - 1) = mmomen1(i)
    mjxVal(2 * i) = Array(Cx2, Cmx2)
    mjyVal(2 * i) = Array(Cy2, Cmy2)
    mcolor(2 * i) = mmomen2(i)
Next i

'creating member lines
For i = 1 To NM
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = mxVal(i)
        .SeriesCollection(i).Values = myVal(i)
    End With
Next i

```



```

        .LineStyle = xlContinuous
    End With
    am = am + 1
Next i

L_III = L_II + 1
L_IV = L_II + NM + NM
am = 1
'creating joint lines to axis
For i = L_III To L_IV
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = mJxVal(am)
        .SeriesCollection(i).Values = mJyVal(am)
        .SeriesCollection(i).Name = " " " " " "
    End With
    ActiveChart.SeriesCollection(i).Select
    With Selection
        .MarkerBackgroundColorIndex = xlNone
        .MarkerForegroundColorIndex = xlNone
        .MarkerStyle = xlNone
        .Smooth = False
    End With
    With Selection.Border
        If mcolor(am) < 0 Then
            .ColorIndex = 3
        Else
            .ColorIndex = 5
        End If
        .Weight = xlMedium
        .LineStyle = xlContinuous
    End With
    am = am + 1

Next i

ActiveChart.ChartArea.Select

```

```

With ActiveChart
    .Axes(xlCategory, xlPrimary).HasTitle = True
    .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = _
        "Max Moment, J." & At_joint & " = " & Format(Mmax, "0.000")
End With

End Sub

Sub PlotShear()
On Error Resume Next

Dim Cx1, Cx2, Cy1, Cy2
Dim Cmx1, Cmx2, Cmy1, Cmy2
ReDim mxVal(NM) As Variant
ReDim myVal(NM) As Variant
ReDim tagxVal(NM) As Variant
ReDim tagyVal(NM) As Variant
ReDim shxval(NM) As Variant
ReDim shyval(NM) As Variant
ReDim sjxVal(NM + NM) As Variant
ReDim sjyVal(NM + NM) As Variant
ReDim mcolor(NM + NM) As Variant

Dim Smax As Double, At_member As Integer, am As Integer, L_I As Integer, _
L_II As Integer, L_III As Integer, L_IV As Integer

    ActiveSheet.ChartObjects("Chart 4").Activate
    ActiveChart.ChartArea.Select
    Selection.ClearContents

Smax = Application.index(USMax(Pmm, Fm, NM), 1)
At_member = Application.index(USMax(Pmm, Fm, NM), 2)

For i = 1 To NM
    With Member(i)
        Cx1 = Joint(.J1).x
        Cy1 = Joint(.J1).y

```



```

        .Smooth = False
    End With
    With Selection.Border
        .ColorIndex = 5
        .Weight = xlThin
        .LineStyle = xlContinuous
    End With
Next i

L_I = NM + 1
L_II = NM + NM
am = 1
'creating shear lines
For i = L_I To L_II
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = shxval(am)
        .SeriesCollection(i).Values = shyval(am)
        .SeriesCollection(i).Name = "=""=""
    End With
    ActiveChart.SeriesCollection(i).Select
    With Selection
        .MarkerBackgroundColorIndex = xlNone
        .MarkerForegroundColorIndex = xlNone
        .MarkerStyle = xlNone
        .Smooth = False
    End With
    With Selection.Border
        .ColorIndex = 7
        .Weight = xlThin
        .LineStyle = xlContinuous
    End With
    am = am + 1
Next i

L_III = L_II + 1
L_IV = L_II + NM + NM

```



```

Sub PlotSoilPressure()
On Error Resume Next

Dim Cx1, Cx2, Cy1, Cy2
Dim Cmx1, Cmx2, Cmy1, Cmy2
Dim Sp As Double
ReDim mxVal(NM) As Variant
ReDim myVal(NM) As Variant
ReDim spxVal(NM) As Variant
ReDim spyVal(NM) As Variant

ReDim spjxVal(NM + NM) As Variant
ReDim spjyVal(NM + NM) As Variant
ReDim mcolor(NM + NM) As Variant

Dim Spmax As Double, At_joint As Integer, am As Integer, L_I As Integer, L_II
As Integer

    ActiveSheet.ChartObjects("Chart 5").Activate
    ActiveChart.ChartArea.Select
    Selection.ClearContents

Spmax = Application.index(USPMax(Pmm, Fm, NM), 1)
At_joint = Application.index(USPMax(Pmm, Fm, NM), 2)

For i = 1 To NM
    With Member(i)
        Cx1 = Joint(.J1).x
        Cy1 = Joint(.J1).y
        Cx2 = Joint(.J2).x
        Cy2 = Joint(.J2).y
        'assign coordinates
        Cmx1 = Cx1 + .Cy * Joint(.J1).Sp
        Cmy1 = Cy1 + .Cx * Joint(.J1).Sp
        Cmx2 = Cx2 + .Cy * Joint(.J2).Sp
        Cmy2 = Cy2 + .Cx * Joint(.J2).Sp
    End With
'member coordinates

```

```

        mxVal(i) = Array(Cx1, Cx2)
        myVal(i) = Array(Cy1, Cy2)
'soil pressure coordinates
        spxVal(i) = Array(Cmx1, Cmx2)
        spyVal(i) = Array(Cmy1, Cmy2)
'member-joint coordinates
        spjxVal(2 * i - 1) = Array(Cx1, Cmx1)
        spjyVal(2 * i - 1) = Array(Cy1, Cmy1)
        'mcolor: define color for positive/negative value
        mcolor(2 * i - 1) = Sp
        spjxVal(2 * i) = Array(Cx2, Cmx2)
        spjyVal(2 * i) = Array(Cy2, Cmy2)
        mcolor(2 * i) = Sp
Next i

'creating member lines
For i = 1 To NM
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = mxVal(i)
        .SeriesCollection(i).Values = myVal(i)
        .SeriesCollection(i).Name = "=""=""
    End With
    ActiveChart.SeriesCollection(i).Select
    With Selection
        .MarkerBackgroundColorIndex = xlNone
        .MarkerForegroundColorIndex = xlNone
        .MarkerStyle = xlNone
        .Smooth = False
    End With
    With Selection.Border
        .ColorIndex = 5
        .Weight = xlThin
        .LineStyle = xlContinuous
    End With
Next i

```



```

L_I = NM + NM
am = 1
'creating soil pressure lines
For i = NM + 1 To L_I
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = spxVal(am)
        .SeriesCollection(i).Values = spyVal(am)
        .SeriesCollection(i).Name = """"
    End With
    ActiveChart.SeriesCollection(i).Select
    With Selection
        .MarkerBackgroundColorIndex = xlNone
        .MarkerForegroundColorIndex = xlNone
        .MarkerStyle = xlNone
        .Smooth = False
    End With
    With Selection.Border
        .ColorIndex = 7
        .Weight = xlThin
        .LineStyle = xlContinuous
    End With
am = am + 1
Next i

L_II = NM + NM + NM + NM
am = 1
'creating joint lines to axis
For i = L_I + 1 To L_II
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = spjxVal(am)
        .SeriesCollection(i).Values = spjyVal(am)
        .SeriesCollection(i).Name = """"
    End With
    ActiveChart.SeriesCollection(i).Select
    With Selection

```

```

        .MarkerBackgroundColorIndex = xlNone
        .MarkerForegroundColorIndex = xlNone
        .MarkerStyle = xlNone
        .Smooth = False
    End With
    With Selection.Border
        If mcolor(am) < 0 Then
            .ColorIndex = 3
        Else
            .ColorIndex = 5
        End If
        .Weight = xlMedium
        .LineStyle = xlContinuous
    End With
    am = am + 1
Next i

ActiveChart.ChartArea.Select
With ActiveChart
    .Axes(xlCategory, xlPrimary).HasTitle = True
    .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = _
        "Max Pressure, J." & At_joint & " = " & Format(Spmax, "0.000")
End With

End Sub

'user function to determine maximum value of joint translation
Function UXMax(Xs, NP) As Variant

Dim Abs_FXMax As Double
Dim FXMax As Double

Abs_FXMax = 0
n = 0
For i = 2 To NP Step 2
    If Abs(Xs(i)) > Abs_FXMax Then
        Abs_FXMax = Abs(Xs(i))
    End If

```

```

        FXMax = Xs(i)
        n = i / 2
        'UXMax keeps origin value(+ or - value)
        UXMax = Array(FXMax, n)
    End If
Next i

End Function

'user function to determine maximum value of member moment
Function UMMax(Pmm, Fm, NM) As Variant

Dim Abs_FMMMax As Double
Dim FMMMax As Double

Abs_FMMMax = 0
n = 0
For i = 1 To NM
    If Abs(mmomen1(i)) > Abs_FMMMax Then
        Abs_FMMMax = Abs(mmomen1(i))
        FMMMax = mmomen1(i)
        n = Member(i).J1
        UMMMax = Array(FMMMax, n)
    End If
    If Abs(mmomen2(i)) > Abs_FMMMax Then
        Abs_FMMMax = Abs(mmomen2(i))
        FMMMax = -mmomen2(i)
        n = Member(i).J2
        UMMMax = Array(FMMMax, n)
    End If
Next i

End Function

'user function to determine maximum shear force
Function USMax(Pmm, Fm, NM) As Variant

```

```

Dim Abs_FSMMax As Double
Dim FSMMax As Double

Abs_FSMMax = 0
n = 0
For i = 1 To NM
    If Abs(shear1(i)) > Abs_FSMMax Then
        Abs_FSMMax = Abs(shear1(i))
        FSMMax = -shear1(i)
        n = i
        USMax = Array(FSMMax, n)
    End If
    If Abs(shear2(i)) > Abs_FSMMax Then
        Abs_FSMMax = Abs(shear2(i))
        FSMMax = shear2(i)
        n = i
        USMax = Array(FSMMax, n)
    End If
Next i

End Function

'user function to determine maximum soil pressure
Function USPMax(Pmm, Fm, NM) As Variant

Dim Sp As Double
Dim Abs_FSPMax As Double
Dim FSPMax As Double

Abs_FSPMax = 0
n = 0
For i = 1 To NJ
    Sp = Joint(i).Ks * Xs(2 * i)
    If Abs(Sp) > Abs_FSPMax Then
        Abs_FSPMax = Abs(Sp)
        FSPMax = Sp
        n = i
    End If
Next i

```

```

        USPMax = Array(FSPMax, n)
    End If
Next i
End Function

```

Command Button (BOF)

```

Private Sub CommandButton1_Click()
    ChartWindow = True
    Call BOF
End Sub

Private Sub CommandButton2_Click()
    ChartWindow = False
    Call BOF
End Sub

```

Module1 (XLAT)

```

'=====
'XLAT v.1.0 Program for Laterally Loaded Structure, 2004
'by: Gunthar Pangaribuan - Refer to the book:
'An Introduction to Excel for Civil Engineers
'=====

Option Explicit

Public i As Integer, j As Integer, n As Integer

Type Joint_Data
    x As Double
    y As Double
    Ks As Double
    Spring As Double
    Sp As Double
End Type

```

Type Member_Data

J1 As Integer

J2 As Integer

Im As Double

bm As Double

EI As Double

lx As Double

ly As Double

Lh As Double

Km1 As Double

Km2 As Double

Pres1 As Double

Pres2 As Double

End Type

Public ChartWindow As Boolean

Public NP As Integer

Public DOF As Integer

Public NR As Integer

Public NJ As Integer

Public NM As Integer

Public Member() As Member_Data

Public Joint() As Joint_Data

Public Em As Double

Public Km() As Double

Public NS As Integer 'no. of support

Public GASAT() As Double 'global stiffness matrix

Public Rs() As Integer

Public DFR() As Double

Public Idm() As Integer

Public Idj() As Integer

Public Irj() As Integer

Public Psum() As Double

Public Pmm() As Double

Public Pj() As Double

Public Xs() As Double

```

Public Xm() As Double
Public Fm() As Double
Public mmomen1() As Double
Public mmomen2() As Double
Public shear1() As Double
Public shear2() As Double

Sub XLAT()
On Error GoTo ErrMsg

Const NDJ = 2
Dim NLJ As Integer 'no of joint load
Dim Jground As Integer 'excavation point

NJ = Cells(4, 2)
NM = Cells(5, 2)
Em = Cells(6, 2)
Jground = Cells(4, 5)

NS = Application.Count(Rows("23:23"))
NLJ = Application.Count(Rows("28:28"))
'Total number of joint displacement, NP:
NP = NDJ * NJ

ReDim Joint(NJ) As Joint_Data
ReDim Member(NM) As Member_Data
ReDim ESAT(4, 4, NM) As Double 'member matrix refer to the book
ReDim EASAT(4, 4, NM) As Double 'member matrix refer to the book
ReDim GASAT(NP, NP) 'global stiffness matrix refer to the book
ReDim Idm(4, NM) 'displacement index
ReDim Pj(NP) 'joint load vector

ReDim Pmm(NP, NM) 'fixed-end forces
ReDim Psum(NP) 'sum load vector
ReDim Xs(NP) 'joint displacement vector
ReDim Xm(4, NM) 'member deformation
ReDim Fm(4, NM) 'member forces

```

```

ReDim mmomen1(NM)
ReDim mmomen2(NM)
ReDim shear1(NM)
ReDim shear2(NM)
ReDim Rs(NP)
ReDim spn(NS) As Integer
Dim mshear As Double

'=====
'Read Input Data
'=====

'Read joint data - in order
For i = 1 To NJ
    With Joint(i)
        .x = Cells(10, 1 + i)
        .y = Cells(11, 1 + i)
        .Ks = Cells(12, 1 + i)
        .Spring = Cells(13, 1 + i)
        .Sp = .Ks * Xs(2 * i) 'soil pressure
    End With
Next i

'Read member data
For i = 1 To NM
    With Member(i)
        .J1 = Cells(16, 1 + i)
        .J2 = Cells(17, 1 + i)
        .Im = Cells(18, 1 + i)
        .bm = Cells(19, 1 + i)
        .EI = Em * .Im
        .lx = Joint(.J2).x - Joint(.J1).x
        .ly = Joint(.J2).y - Joint(.J1).y
        .Lh = Sqr(.lx ^ 2 + .ly ^ 2)
        .Pres1 = Cells(20, 1 + i)
        .Pres2 = Cells(21, 1 + i)
    End With
Next i

```



```

        .Km1 = 0.5 * .Lh * .bm * Joint(.J1).Ks + Joint(.J1).Spring
        .Km2 = 0.5 * .Lh * .bm * Joint(.J2).Ks
    End With
Next i

If ChartWindow = True Then Call PlotGeometry: Exit Sub
Application.ScreenUpdating = False

'Read joint load
For i = 1 To NLJ
    n = Cells(28, 1 + i)
    Pj(2 * n - 1) = Cells(29, 1 + i)
    Pj(2 * n) = Cells(30, 1 + i)
Next i

'restrain list and support (Rs) index
For i = 1 To NS
    spn(i) = Cells(23, 1 + i)
    Rs(2 * spn(i) - 1) = Cells(24, 1 + i)
    Rs(2 * spn(i)) = Cells(25, 1 + i)
Next i

DOF = 0
For i = 1 To NP
    If Rs(i) = 0 Then DOF = DOF + 1
Next i

'#restrained joint
NR = NP - DOF

ReDim DFX(NDJ * NS, DOF) As Double '[Kbf]according to Eq. 4.12
ReDim DFR(DOF, DOF) As Double '[Kff]^-1 according to Eq. 4.9
ReDim RJ(NDJ * NS) As Double 'support reactions
ReDim Idj(DOF) 'free displacement index
ReDim Irj(NDJ * NS) 'support displacement index
ReDim nAs_Now(NJ) As Integer
Dim nAs_Last As Integer

```

```

Dim Ksn As Integer
Dim ITR As Integer
Dim ITR_Continue As Boolean
ITR_Continue = True
ITR = 1

'clear previous content
Dim LastRow
LastRow = ActiveSheet.UsedRange.Rows.Count
Range(Cells(38, 1), Cells(LastRow, 18)).ClearContents

'=====
'Structure Analysis
'=====
Do

Call Mindex(Idm, Idj, Irj)
Call FEM(ESAT, EASAT, GASAT)
Call MInvers(DFX, DFR)
Call Genload(Pmm, Psum)
Call DISP(Xs, Xm)

'Print used springs
For i = 1 To NM
    With Member(i)
        Cells(37 + i, 8).NumberFormat = "0.000"
        Cells(37 + i, 8) = .Km1
        Cells(37 + i, 9).NumberFormat = "0.000"
        Cells(37 + i, 9) = .Km2
    End With
Next i

n = 0
For i = 1 To Jground - 1
're-input joint's Ks and Sp if Xs > 0
    With Joint(i)
        If Xs(i * 2) > 0 Then

```

```

        .Ks = 0
        .Sp = .Ks * Xs(2 * i)
        n = n + 1
    End If
End With
Next i

're-input member's soil K
For i = 1 To NM
    With Member(i)
        .Km1 = 0.5 * .Lh * .bm * Joint(.J1).Ks + Joint(.J1).Spring
        .Km2 = 0.5 * .Lh * .bm * Joint(.J2).Ks
    End With
Next i

'check no. of active spring between 2 iterations
nAs_Now(ITR) = NJ - n
nAs_Last = nAs_Now(ITR - 1)

If nAs_Now(ITR) = nAs_Last Or n = 0 Then
    ITR_Continue = False
    Cells(34, 3) = ITR - 1
    Cells(35, 3) = nAs_Now(ITR)
    Ksn = nAs_Now(ITR)
End If

ITR = ITR + 1

Loop Until ITR_Continue = False

'=====
'Print Result
'=====

'1.Print Load
For i = 1 To NJ
    Cells(37 + i, 1).NumberFormat = "0"

```

```

Cells(37 + i, 1) = i
Cells(37 + i, 2).NumberFormat = "0.000"
Cells(37 + i, 2) = Psum(2 * i - 1)
Cells(37 + i, 3).NumberFormat = "0.000"
Cells(37 + i, 3) = Psum(2 * i)
Next i

'2. Print Joint displacement/pressure
For i = 1 To NJ
    Cells(37 + i, 4).NumberFormat = "0.00000"
    Cells(37 + i, 4) = Xs(2 * i - 1)
    Cells(37 + i, 5).NumberFormat = "0.00000"
    Cells(37 + i, 5) = Xs(2 * i)
    Cells(37 + i, 6).NumberFormat = "0.000"
    With Joint(i)
        .Sp = .Ks * Xs(2 * i)
        Cells(37 + i, 6) = .Sp
    End With
Next i

'-----
'Member Forces
'-----
'{F}m=[S]m.{X}m or [F]m=[S].[A]T.[X]s
'there is already calculated S.A^T,
ReDim tshear(4, NM)

'produce moment + soil pressure
For i = 1 To NM
    For j = 1 To 4
        Fm(j, i) = 0
        For n = 1 To 4
            Fm(j, i) = Fm(j, i) + ESAT(j, n, i) * Xm(n, i)
        Next n
    Next j
Next i

```

```

'Print member data
For i = 1 To NM
    With Member(i)
        'no/length
        Cells(37 + i, 7).NumberFormat = "0"
        Cells(37 + i, 7) = i & "./" & Format(.Lh, "0.00")
        'moment
        mmomen1(i) = Fm(1, i) + Pmm(Idm(1, i), i)
        Cells(37 + i, 10).NumberFormat = "0.000"
        Cells(37 + i, 10) = mmomen1(i)
        mmomen2(i) = Fm(2, i) + Pmm(Idm(3, i), i)
        Cells(37 + i, 11).NumberFormat = "0.000"
        Cells(37 + i, 11) = mmomen2(i)
        'spring
        Cells(37 + i, 12).NumberFormat = "0.000"
        Cells(37 + i, 12) = Fm(3, i)
        Cells(37 + i, 13).NumberFormat = "0.000"
        Cells(37 + i, 13) = Fm(4, i)
        'shear
        mshear = (Fm(1, i) + Fm(2, i)) / .Lh
        shear1(i) = mshear
        shear2(i) = -mshear
        Cells(37 + i, 14).NumberFormat = "0.000"
        Cells(37 + i, 14) = shear1(i)
        Cells(37 + i, 15).NumberFormat = "0.000"
        Cells(37 + i, 15) = shear2(i)
    End With
Next i

'-----
'Support Reactions
'-----
For i = 1 To 2 * NS
    For j = 1 To DOF
        RJ(i) = RJ(i) + DFX(i, j) * Xs(Idj(j))
    Next j
Next i

```

```

For i = 1 To NS
    Cells(37 + i, 16).NumberFormat = "0"
    Cells(37 + i, 16) = spn(i)
    Cells(37 + i, 17).NumberFormat = "0.000"
    Cells(37 + i, 17) = RJ(2 * i - 1) - Psum(Irj(2 * i - 1))
    Cells(37 + i, 18).NumberFormat = "0.000"
    Cells(37 + i, 18) = RJ(2 * i) - Psum(Irj(2 * i))
Next i

'create graphs
Call PlotGeometry
Call PlotDisplacement
Call PlotMoment
Call PlotShear
Call PlotLoad

Range("A1").Select
Application.ScreenUpdating = True
Exit Sub

ErrMsg: MsgBox "Error, please check input ...", vbOKOnly + vbExclamation,
"XLAT"

Range("A1").Select
Application.ScreenUpdating = True

End Sub

```

Module2 (XLAT)

```

Option Explicit

'Indexing for matrix subscript
Sub Mindex(index() As Integer, IFr() As Integer, IFx() As Integer)

'Member end-displacements index:

```

```

For i = 1 To NM
    With Member(i)
        index(1, i) = 2 * .J1 - 1
        index(2, i) = 2 * .J1
        index(3, i) = 2 * .J2 - 1
        index(4, i) = 2 * .J2
    End With
Next i

'Joint free displacement index, IFr and
'restraint (support) index, IFx
n = 1
For i = 1 To NP
    If Rs(i) = 0 Then IFr(n) = i: n = n + 1
Next i

n = 1
For i = 1 To NS
    n = Cells(23, 1 + i)
    IFx(2 * i - 1) = 2 * n - 1
    IFx(2 * i) = 2 * n
Next i

End Sub

Sub FEM(SAT() As Double, ASAT() As Double, GS() As Double)

'Clearing array
For i = 1 To NP
    For j = 1 To NP
        GS(i, j) = 0#
    Next j
Next i

'Assembly finite element matrix, Eq.4.16 & 4.17
For i = 1 To NM
    With Member(i)

```

```

'member SA^T matrix:
'from internal forces F - deformation d relationship, F = S.d or F =
S.A^T.X
SAT(1, 1, i) = 4 * .EI / .Lh: SAT(1, 2, i) = 6 * .EI / .Lh ^ 2
SAT(1, 3, i) = 2 * .EI / .Lh: SAT(1, 4, i) = -6 * .EI / .Lh ^ 2
SAT(2, 1, i) = 2 * .EI / .Lh: SAT(2, 2, i) = 6 * .EI / .Lh ^ 2
SAT(2, 3, i) = 4 * .EI / .Lh: SAT(2, 4, i) = -6 * .EI / .Lh ^ 2
SAT(3, 2, i) = .Km1
SAT(4, 4, i) = .Km2
'Member global stiffness (member ASA^T)
ASAT(1, 1, i) = 4 * .EI / .Lh: ASAT(1, 2, i) = 6 * .EI / .Lh ^ 2
ASAT(1, 3, i) = 2 * .EI / .Lh: ASAT(1, 4, i) = -6 * .EI / .Lh ^ 2
ASAT(2, 1, i) = 6 * .EI / .Lh ^ 2: ASAT(2, 2, i) = 12 * .EI / .Lh ^ 3 +
.Km1
ASAT(2, 3, i) = 6 * .EI / .Lh ^ 2: ASAT(2, 4, i) = -12 * .EI / .Lh ^ 3
ASAT(3, 1, i) = 2 * .EI / .Lh: ASAT(3, 2, i) = 6 * .EI / .Lh ^ 2
ASAT(3, 3, i) = 4 * .EI / .Lh: ASAT(3, 4, i) = -6 * .EI / .Lh ^ 2
ASAT(4, 1, i) = -6 * .EI / .Lh ^ 2: ASAT(4, 2, i) = -12 * .EI / .Lh ^ 3
ASAT(4, 3, i) = -6 * .EI / .Lh ^ 2: ASAT(4, 4, i) = 12 * .EI / .Lh ^ 3 +
.Km2
'Storing members and superposition(global ASA^T)
For j = 1 To 4
    For n = 1 To 4
        GS(Idm(j, i), Idm(n, i)) = GS(Idm(j, i), Idm(n, i)) + ASAT(j, n,
i)
    Next n
Next j
End With
Next i

End Sub

'Global matrix
Sub MInvers(SR() As Double, SD() As Double)
Dim n As Integer

'submatrix Stiffness, KBF
For i = 1 To DOF
    For j = 1 To 2 * NS
        SR(j, i) = GASAT(Irj(j), Idj(i))
    Next j
Next i

```



```

        Next j
    Next i

    'submatrix Stiffness, KFF
    For i = 1 To DOF
        For j = 1 To DOF
            SD(i, j) = GASAT(Idj(i), Idj(j))
        Next j
    Next i

    'Inverse []
    For i = 1 To DOF
        For j = 1 To DOF
            If j <> i Then SD(i, j) = SD(i, j) / SD(i, i)
        Next j
        For n = 1 To DOF
            If n = i Then GoTo 10
            For j = 1 To DOF
                If j <> i Then SD(n, j) = SD(n, j) - SD(i, j) * SD(n, i)
            Next j
        Next j
    10 Next n
        For n = 1 To DOF
            If n <> i Then SD(n, i) = -SD(n, i) / SD(i, i)
        Next n
        SD(i, i) = 1 / SD(i, i)
    Next i

End Sub

Sub Genload(Pa() As Double, Ps() As Double)
    ReDim Peq(NP, NM) As Double 'equivalent load
    Dim Idx As Integer, w As Double, d As Double, sq As Double

    'member fixed-end forces due to pressure
    For i = 1 To NM
        With Member(i)
            d = Abs(.Pres1 - .Pres2)
            sq = Application.Min(.Pres1, .Pres2)

```

```

'rectangular part
w = sq * .bm
Pa(Idm(1, i), i) = -w * .Lh ^ 2 / 12 'moment
Pa(Idm(2, i), i) = -w * .Lh / 2 'horizontal
Pa(Idm(3, i), i) = w * .Lh ^ 2 / 12 'moment
Pa(Idm(4, i), i) = -w * .Lh / 2 'horizontal
'triangular part & summing
w = d * .bm
Pa(Idm(1, i), i) = Pa(Idm(1, i), i) + (-w * .Lh ^ 2 / 30) 'moment
Pa(Idm(2, i), i) = Pa(Idm(2, i), i) + (-w * .Lh * 3 / 20) 'horizontal
Pa(Idm(3, i), i) = Pa(Idm(3, i), i) + (w * .Lh ^ 2 / 20) 'moment
Pa(Idm(4, i), i) = Pa(Idm(4, i), i) + (-w * .Lh * 7 / 20) 'horizontal
End With
Next i

'equivalent joint load due to pressure
For i = 1 To NM
    For j = 1 To 4
        For n = 1 To 4
            Peq(Idm(j, i), i) = -Pa(Idm(j, i), i)
        Next n
    Next j
Next i

'sum load = joint load + eq.load
For i = 1 To NP
    Ps(i) = Pj(i)
Next i

'load superposition
For i = 1 To NM
    For j = 1 To 4
        Ps(Idm(j, i)) = Ps(Idm(j, i)) + Peq(Idm(j, i), i)
    Next j
Next i

End Sub

```

```

Sub DISP(x() As Double, Xi() As Double)
ReDim T(4, 4, NM) As Double

'Joint Displacement {X} = [SD]^-1.{P}
For i = 1 To DOF
x(Idj(i)) = 0
    For j = 1 To DOF
        x(Idj(i)) = x(Idj(i)) + DFR(i, j) * Psum(Idj(j))
    Next j
Next i

'Numbering of global {X}(NP) to global {Xi} member
For i = 1 To NM
    For j = 1 To 4
        Xi(j, i) = x(Idm(j, i))
    Next j
Next i

End Sub

```

Module3 (XLAT)

```

Option Explicit
Option Base 1
Sub PlotGeometry()
On Error Resume Next

Dim Cx1, Cx2, Cy1, Cy2
Dim Cmx1, Cmx2, Cmy1, Cmy2

ReDim mxVal(NM) As Variant
ReDim myVal(NM) As Variant
ReDim tagxVal(NM) As Variant
ReDim tagyVal(NM) As Variant

```

```

Dim am As Integer, L_I As Integer, L_II As Integer, L_III As Integer, L_IV As Integer
Dim L_V As Integer, L_VI As Integer, L_VII As Integer, L_VIII As Integer

'member coordinates
For i = 1 To NM
    With Member(i)
        Cx1 = Joint(.J1).x
        Cy1 = Joint(.J1).y
        Cx2 = Joint(.J2).x
        Cy2 = Joint(.J2).y
    End With
    mxVal(i) = Array(Cx1, Cx2)
    myVal(i) = Array(Cy1, Cy2)
    tagxVal(i) = Array((Cx1 + Cx2) / 2)
    tagyVal(i) = Array((Cy1 + Cy2) / 2)
Next i

ActiveSheet.ChartObjects("Chart 1").Activate
ActiveChart.ChartArea.Select
Selection.ClearContents

'creating joints + name tags
For i = 1 To NJ
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = Joint(i).x
        .SeriesCollection(i).Values = Joint(i).y
        .SeriesCollection(i).Name = "J" & i
    End With
    'node marker
    ActiveChart.PlotArea.Select
    ActiveChart.SeriesCollection(i).Select
    With Selection
        .MarkerBackgroundColorIndex = 3 'red
        .MarkerForegroundColorIndex = 3
        .MarkerStyle = xlCircle
        .MarkerSize = 6
    End With
Next i

```

```

        .Shadow = False
        .ApplyDataLabels AutoText:=True, LegendKey:= _
            False,          ShowSeriesName:=True,          ShowCategoryName:=False,
ShowValue:=False, _
            ShowPercentage:=False, ShowBubbleSize:=False
    End With

    ActiveChart.SeriesCollection(i).DataLabels.Select
    With Selection
        .HorizontalAlignment = xlCenter
        .VerticalAlignment = xlCenter
        .ReadingOrder = xlContext
        .Position = xlLabelPositionRight
        .Orientation = xlHorizontal
        .Font.Name = "Arial"
        .Font.FontStyle = "Regular"
        .Font.Size = 8
    End With
Next i

    ActiveSheet.ChartObjects("Chart 1").Activate
    ActiveChart.ChartArea.Select

    L_I = NJ + 1
    L_II = NJ + NM
am = 1
'creating member lines
For i = L_I To L_II
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = mxVal(am)
        .SeriesCollection(i).Values = myVal(am)
        .SeriesCollection(i).Name = " " " " " "
    End With
    ActiveChart.SeriesCollection(i).Select
    With Selection
        .MarkerBackgroundColorIndex = xlNone
        .MarkerForegroundColorIndex = xlNone

```

```

        .MarkerStyle = xlNone
        .Smooth = False
    End With
    With Selection.Border
        .ColorIndex = 5
        .Weight = xlThin
        .LineStyle = xlContinuous
    End With
    am = am + 1
Next i

L_III = L_II + 1
L_IV = L_II + NM
'creating member name tags
am = 1
For i = L_III To L_IV
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = tagxVal(am)
        .SeriesCollection(i).Values = tagyVal(am)
        .SeriesCollection(i).Name = "M" & am
    End With
    ActiveChart.PlotArea.Select
    ActiveChart.SeriesCollection(i).Select
    With Selection
        .MarkerBackgroundColorIndex = 5
        .MarkerForegroundColorIndex = 5
        .MarkerStyle = xlSquare
        .Smooth = False
        .MarkerSize = 6
        .Shadow = False
        .ApplyDataLabels AutoText:=True, LegendKey:= _
            False, ShowSeriesName:=True, ShowCategoryName:=False,
ShowValue:=False, _
            ShowPercentage:=False, ShowBubbleSize:=False
    End With
    ActiveChart.SeriesCollection(i).DataLabels.Select
    With Selection

```

```

        .HorizontalAlignment = xlCenter
        .VerticalAlignment = xlCenter
        .Position = xlLabelPositionRight
        .Orientation = xlHorizontal
        .Font.Name = "Arial"
        .Font.FontStyle = "Regular"
        .Font.Size = 8
    End With
    am = am + 1
Next i

ActiveChart.ChartArea.Select
With ActiveChart
    .Axes(xlCategory, xlPrimary).HasTitle = True
    .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = _
        "NJ = " & Format(NJ, "0") & ", NM = " & Format(NM, "0")
End With

'Draw pressure
ReDim pxVal(NM) As Variant
ReDim pyVal(NM) As Variant
ReDim pjxVal(NM + NM) As Variant
ReDim pjyVal(NM + NM) As Variant

For i = 1 To NM
    With Member(i)
        Cx1 = Joint(.J1).x
        Cy1 = Joint(.J1).y
        Cx2 = Joint(.J2).x
        Cy2 = Joint(.J2).y
        'assign coordinates
        Cmx1 = Cx1 + .Pres1
        Cmy1 = Cy1
        Cmx2 = Cx2 + .Pres2
        Cmy2 = Cy2
    End With

```

```

'pressure coordinates
    pxVal(i) = Array(Cmx1, Cmx2)
    pyVal(i) = Array(Cmy1, Cmy2)
'member-joint coordinates
    pjxVal(2 * i - 1) = Array(Cx1, Cmx1)
    pjyVal(2 * i - 1) = Array(Cy1, Cmy1)
    pjxVal(2 * i) = Array(Cx2, Cmx2)
    pjyVal(2 * i) = Array(Cy2, Cmy2)
Next i

L_V = L_IV + 1
L_VI = L_IV + NM

am = 1
'creating pressure lines
For i = L_V To L_VI
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = pxVal(am)
        .SeriesCollection(i).Values = pyVal(am)
        .SeriesCollection(i).Name = ""
    End With
    ActiveChart.SeriesCollection(i).Select
    With Selection
        .MarkerBackgroundColorIndex = xlNone
        .MarkerForegroundColorIndex = xlNone
        .MarkerStyle = xlNone
        .Smooth = False
    End With
    With Selection.Border
        .ColorIndex = 7
        .Weight = xlThin
        .LineStyle = xlContinuous
    End With
    am = am + 1
Next i

```



```

L_VII = L_VI + 1
L_VIII = L_VI + NM + NM

am = 1
'creating joint lines to axis
For i = L_VII To L_VIII
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = pjxVal(am)
        .SeriesCollection(i).Values = pjyVal(am)
        .SeriesCollection(i).Name = "=""=""
    End With
    ActiveChart.SeriesCollection(i).Select
    With Selection
        .MarkerBackgroundColorIndex = xlNone
        .MarkerForegroundColorIndex = xlNone
        .MarkerStyle = xlNone
        .Smooth = False
    End With
    With Selection.Border
        .ColorIndex = 5
        .Weight = xlMedium
        .LineStyle = xlContinuous
    End With
am = am + 1
Next i

ActiveChart.ShowWindow = True

End Sub

Sub PlotDisplacement()
On Error Resume Next

Dim Cx1, Cx2, Cy1, Cy2
ReDim mxValbf(NM) As Variant, myValbf(NM) As Variant
ReDim mxValaf(NM) As Variant, myValaf(NM) As Variant

```

```
Dim Xmax As Double, At_joint As Integer, am As Integer
```

```
    ActiveSheet.ChartObjects("Chart 2").Activate  
    ActiveChart.ChartArea.Select  
    Selection.ClearContents
```

```
Xmax = Application.index(UXMax(Xs, NP), 1)  
At_joint = Application.index(UXMax(Xs, NP), 2)
```

```
'member coordinates: before & after loading
```

```
For i = 1 To NM  
    With Member(i)  
        Cx1 = Joint(.J1).x  
        Cy1 = Joint(.J1).y  
        Cx2 = Joint(.J2).x  
        Cy2 = Joint(.J2).y  
        mxValbf(i) = Array(Cx1, Cx2)  
        myValbf(i) = Array(Cy1, Cy2)  
        Cx1 = Joint(.J1).x + Xs(2 * .J1)  
        Cy1 = Joint(.J1).y + Xs(2 * .J1)  
        Cx2 = Joint(.J2).x + Xs(2 * .J2)  
        Cy2 = Joint(.J2).y + Xs(2 * .J2)  
        mxValaf(i) = Array(Cx1, Cx2)  
        myValaf(i) = Array(Cy1, Cy2)  
    End With  
Next i
```

```
    am = 1
```

```
'creating member lines: before loading
```

```
For i = 1 To NM  
    With ActiveChart  
        .SeriesCollection.NewSeries  
        .SeriesCollection(i).XValues = mxValbf(am)  
        .SeriesCollection(i).Values = myValbf(am)  
        .SeriesCollection(i).Name = " " " " " "  
    End With
```

```

ActiveChart.SeriesCollection(i).Select
With Selection
    .MarkerStyle = xlNone
    .Smooth = False
End With
With Selection.Border
    .ColorIndex = 5
    .Weight = xlMedium
    .LineStyle = xlContinuous
End With
am = am + 1
Next i

n = NM + NM
am = 1
'creating member lines: after loading
For i = NM + 1 To n
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = mxValaf(am)
        .SeriesCollection(i).Values = myValaf(am)
        .SeriesCollection(i).Name = " " " " " " " "
    End With
ActiveChart.SeriesCollection(i).Select
With Selection
    .MarkerStyle = xlNone
    .Smooth = False
End With
With Selection.Border
    .ColorIndex = 7
    .Weight = xlMedium
    .LineStyle = xlContinuous
End With
am = am + 1
Next i

ActiveChart.ChartArea.Select

```

```

With ActiveChart
    .Axes(xlCategory, xlPrimary).HasTitle = True
    .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = _
        "Max Displacement, J." & At_joint & " = " & Format(Xmax, "0.00000")
End With

End Sub

Sub PlotMoment()
On Error Resume Next

Dim Cx1, Cx2, Cy1, Cy2
Dim Cmx1, Cmx2, Cmy1, Cmy2
ReDim mxVal(NM) As Variant
ReDim myVal(NM) As Variant
ReDim moxVal(NM) As Variant
ReDim moyVal(NM) As Variant
ReDim tagxVal(NM) As Variant
ReDim tagyVal(NM) As Variant

ReDim mjsxVal(NM + NM) As Variant
ReDim mjyVal(NM + NM) As Variant
ReDim mcolor(NM + NM) As Variant

Dim Mmax As Single, At_joint As Integer, am As Integer, L_I As Integer, L_II
As Integer, _
L_III As Integer, L_IV As Integer

    ActiveSheet.ChartObjects("Chart 3").Activate
    ActiveChart.ChartArea.Select
    Selection.ClearContents

Mmax = Application.index(UMMax(Pmm, Fm, NM), 1)
At_joint = Application.index(UMMax(Pmm, Fm, NM), 2)

For i = 1 To NM
    With Member(i)
        Cx1 = Joint(.J1).x

```

```

    Cy1 = Joint(.J1).y
    Cx2 = Joint(.J2).x
    Cy2 = Joint(.J2).y
    'assign coordinates
    Cmx1 = Cx1 + mmomen1(i)
    Cmy1 = Cy1
    Cmx2 = Cx2 - mmomen2(i)
    Cmy2 = Cy2
End With
'member coordinates
    mxVal(i) = Array(Cx1, Cx2)
    myVal(i) = Array(Cy1, Cy2)
'moment coordinates
    moxVal(i) = Array(Cmx1, Cmx2)
    moyVal(i) = Array(Cmy1, Cmy2)
'member-joint coordinates
    mjxVal(2 * i - 1) = Array(Cx1, Cmx1)
    mjyVal(2 * i - 1) = Array(Cy1, Cmy1)
    'mcolor: define color for positive/negative value
    mcolor(2 * i - 1) = mmomen1(i)
    mjxVal(2 * i) = Array(Cx2, Cmx2)
    mjyVal(2 * i) = Array(Cy2, Cmy2)
    mcolor(2 * i) = mmomen2(i)
Next i

'creating member lines
For i = 1 To NM
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = mxVal(i)
        .SeriesCollection(i).Values = myVal(i)
        .SeriesCollection(i).Name = " " " " " "
    End With
    ActiveChart.SeriesCollection(i).Select
    With Selection
        .MarkerBackgroundColorIndex = xlNone
        .MarkerForegroundColorIndex = xlNone
    End With
End For

```

```

        .MarkerStyle = xlNone
        .Smooth = False
    End With
    With Selection.Border
        .ColorIndex = 5
        .Weight = xlThin
        .LineStyle = xlContinuous
    End With
Next i

L_I = NM + 1
L_II = NM + NM
am = 1
'creating moment lines
For i = L_I To L_II
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = moxVal(am)
        .SeriesCollection(i).Values = moyVal(am)
        .SeriesCollection(i).Name = "=""=""
    End With
    ActiveChart.SeriesCollection(i).Select
    With Selection
        .MarkerBackgroundColorIndex = xlNone
        .MarkerForegroundColorIndex = xlNone
        .MarkerStyle = xlNone
        .Smooth = False
    End With
    With Selection.Border
        .ColorIndex = 7
        .Weight = xlThin
        .LineStyle = xlContinuous
    End With
    am = am + 1
Next i

```



```

End Sub

Sub PlotShear()
On Error Resume Next

Dim Cx1, Cx2, Cy1, Cy2
Dim Cmx1, Cmx2, Cmy1, Cmy2
ReDim mxVal(NM) As Variant
ReDim myVal(NM) As Variant
ReDim tagxVal(NM) As Variant
ReDim tagyVal(NM) As Variant
ReDim shxval(NM) As Variant
ReDim shyval(NM) As Variant
ReDim sjxVal(NM + NM) As Variant
ReDim sjyVal(NM + NM) As Variant
ReDim mcolor(NM + NM) As Variant

Dim Smax As Double, At_member As Integer, am As Integer, L_I As Integer, _
L_II As Integer, L_III As Integer, L_IV As Integer

    ActiveSheet.ChartObjects("Chart 4").Activate
    ActiveChart.ChartArea.Select
    Selection.ClearContents

Smax = Application.index(USMax(Pmm, Fm, NM), 1)
At_member = Application.index(USMax(Pmm, Fm, NM), 2)

For i = 1 To NM
    With Member(i)
        Cx1 = Joint(.J1).x
        Cy1 = Joint(.J1).y
        Cx2 = Joint(.J2).x
        Cy2 = Joint(.J2).y
        'assign coordinates
        Cmx1 = Cx1 - shear1(i)
        Cmy1 = Cy1
    End With

```



```

        .LineStyle = xlContinuous
    End With
Next i

L_I = NM + 1
L_II = NM + NM
am = 1
'creating shear lines
For i = L_I To L_II
    With ActiveChart
        .SeriesCollection.NewSeries
        .SeriesCollection(i).XValues = shxval(am)
        .SeriesCollection(i).Values = shyval(am)
        .SeriesCollection(i).Name = "="" "" "" "" """
    End With
    ActiveChart.SeriesCollection(i).Select
    With Selection
        .MarkerBackgroundColorIndex = xlNone
        .MarkerForegroundColorIndex = xlNone
        .MarkerStyle = xlNone
        .Smooth = False
    End With
    With Selection.Border
        .ColorIndex = 7
        .Weight = xlThin
        .LineStyle = xlContinuous
    End With
    am = am + 1
Next i

L_III = L_II + 1
L_IV = L_II + NM + NM
am = 1
'creating joint lines to axis
For i = L_III To L_IV
    With ActiveChart
        .SeriesCollection.NewSeries

```



```

ReDim mxVal(NM) As Variant
ReDim myVal(NM) As Variant
ReDim lxVal(NM) As Variant
ReDim lyVal(NM) As Variant

ReDim lxxVal(NM + NM) As Variant
ReDim lyyVal(NM + NM) As Variant
Dim am As Integer, L_I As Integer, L_II As Integer

    ActiveSheet.ChartObjects("Chart 5").Activate
    ActiveChart.ChartArea.Select
    Selection.ClearContents

For i = 1 To NM
    With Member(i)
        Cx1 = Joint(.J1).x
        Cy1 = Joint(.J1).y
        Cx2 = Joint(.J2).x
        Cy2 = Joint(.J2).y
        'assign coordinates
        Cmx1 = Cx1 + Psum(2 * .J1)
        Cmy1 = Cy1
        Cmx2 = Cx2 + Psum(2 * .J2)
        Cmy2 = Cy2
    End With
    'member coordinates
    mxVal(i) = Array(Cx1, Cx2)
    myVal(i) = Array(Cy1, Cy2)
    'load coordinates
    lxVal(i) = Array(Cmx1, Cmx2)
    lyVal(i) = Array(Cmy1, Cmy2)
    'member-joint coordinates
    lxxVal(2 * i - 1) = Array(Cx1, Cmx1)
    lyyVal(2 * i - 1) = Array(Cy1, Cmy1)
    lxxVal(2 * i) = Array(Cx2, Cmx2)
    lyyVal(2 * i) = Array(Cy2, Cmy2)
Next i

```



```

        .MarkerStyle = xlDiamond
        .Smooth = False
        .MarkerSize = 6
        .Shadow = False
    End With
    With Selection.Border
        .ColorIndex = 1
        .Weight = xlMedium
        .LineStyle = xlContinuous
    End With
    am = am + 1
Next i

End Sub

'user function to determine maximum value of joint translation
Function UXMax(Xs, NP) As Variant

Dim Abs_FXMax As Double
Dim FXMax As Double

Abs_FXMax = 0
n = 0
For i = 2 To NP Step 2
    If Abs(Xs(i)) > Abs_FXMax Then
        Abs_FXMax = Abs(Xs(i))
        FXMax = Xs(i)
        n = i / 2
        'UXMax keeps origin value(+ or - value)
        UXMax = Array(FXMax, n)
    End If
Next i

End Function

```

'user function to determine maximum value of member moment

Function UMMax(Pmm, Fm, NM) As Variant

Dim Abs_FMMMax As Double

Dim FMMMax As Double

Abs_FMMMax = 0

n = 0

For i = 1 To NM

 If Abs(mmomen1(i)) > Abs_FMMMax Then

 Abs_FMMMax = Abs(mmomen1(i))

 FMMMax = mmomen1(i)

 n = Member(i).J1

 UMMax = Array(FMMMax, n)

 End If

 If Abs(mmomen2(i)) > Abs_FMMMax Then

 Abs_FMMMax = Abs(mmomen2(i))

 FMMMax = -mmomen2(i)

 n = Member(i).J2

 UMMax = Array(FMMMax, n)

 End If

Next i

End Function

'user function to determine maximum shear force

Function USMax(Pmm, Fm, NM) As Variant

Dim Abs_FSMMax As Double

Dim FSMMax As Double

Abs_FSMMax = 0

n = 0

For i = 1 To NM

 If Abs(shear1(i)) > Abs_FSMMax Then

 Abs_FSMMax = Abs(shear1(i))

 FSMax = -shear1(i)

```

        n = i
        USMax = Array(FSMax, n)
    End If
    If Abs(shear2(i)) > Abs_FSMax Then
        Abs_FSMax = Abs(shear2(i))
        FSMax = shear2(i)
        n = i
        USMax = Array(FSMax, n)
    End If
Next i

End Function

'user function to determine maximum soil pressure
Function USPMax(Pmm, Fm, NM) As Variant

Dim Sp As Double
Dim Abs_FSPMax As Double
Dim FSPMax As Double

Abs_FSPMax = 0
n = 0
For i = 1 To NJ
    Sp = Joint(i).Ks * Xs(2 * i)
    If Abs(Sp) > Abs_FSPMax Then
        Abs_FSPMax = Abs(Sp)
        FSPMax = Sp
        n = i
        USPMax = Array(FSPMax, n)
    End If
Next i

End Function

```


Command Button (XLAT)

```
Private Sub CommandButton1_Click()  
ChartWindow = True  
    Call XLAT  
End Sub
```

```
Private Sub CommandButton2_Click()  
ChartWindow = False  
    Call XLAT  
End Sub
```

Module1 (FDC)

```
Option Explicit  
Sub FDC()
```

```
'=====
```

'FDC v.1.0 Program for One Dimensional Consolidation Analysis, 2004
'by: Gunthar Pangaribuan - Refer to the book:
'An Introduction to: Excel for Civil Engineers
'=====

```
On Error GoTo Check
```

```
Dim h As Double, t As Double, cv As Double, Tv As Double, m As Integer, n As  
Long
```

```
Dim FDCCase As Integer, i As Integer, j As Integer
```

```
Dim AIP As Double, ATP As Double
```

```
im = Cells(4, 2) 'no. of layer  
m = 10 * im  
h = Cells(3, 2) 'height of layer  
t = Cells(5, 5) 'time  
cv = Cells(5, 2) 'coef. of consolidation  
FDCCase = Cells(3, 5) 'Case of calculation
```

```

ReDim dZ(0 To im) As Double, IP(0 To im) As Double, TP(0 To im) As Double

ReDim rsdZ(0 To m) As Double, rsIP(0 To m) As Double, rsTP(0 To m) As Double
Const Beta = 1 / 6

For i = 0 To im
    dZ(i) = i * h / im
    IP(i) = Cells(9 + i, 2)
Next i

're-input dZ and IP to obtain a smooth isochrone curve
n = 1
rsdZ(0) = dZ(0)
rsIP(0) = IP(0)
For i = 0 To im - 1
    For j = 1 To 10
        rsIP(j) = j * (IP(i + 1) - IP(i)) / 10
        rsdZ(j) = j * (dZ(i + 1) - dZ(i)) / 10
        rsIP(n) = rsIP(j) + IP(i)
        rsdZ(n) = rsdZ(j) + dZ(i)
        n = n + 1
    Next j
Next i

Dim LastRow
LastRow = ActiveSheet.UsedRange.Rows.Count
Range(Cells(9, 1), Cells(LastRow, 4)).ClearContents

If FDCCase = 1 Then

'open layered
    Tv = cv * t / (h / 2) ^ 2
    n = 1.5 * m ^ 2 * Tv

    ReDim u(0 To m + 1, 0 To n + 1) As Double

```

```

u(0, 0) = 0

For i = 0 To n
    u(0, i + 1) = 0 'first row = 0
    u(m, i + 1) = 0 'last row = 0
Next i

For i = 1 To m - 1
    u(i, 0) = rsIP(i) '> first row to < last row
Next i

'Finite difference approximation:
For j = 0 To n
    For i = 1 To m - 1
        u(i, j + 1) = u(i, j) + Beta * (u(i - 1, j) + u(i + 1, j) - 2 *
u(i, j))
    Next i
Next j

ElseIf FDCCase = 2 Then

'half closed layered
Tv = cv * t / h ^ 2
n = 6 * m ^ 2 * Tv

ReDim u(0 To m + 1, 0 To n + 1) As Double

u(0, 0) = 0

For i = 0 To n
    u(0, i + 1) = 0 'first row = 0
Next i

For i = 1 To m
    u(i, 0) = rsIP(i)
Next i

```

```

'Finite difference approximation:
  For j = 0 To n
    For i = 1 To m - 1
      u(i, j + 1) = u(i, j) + Beta * (u(i - 1, j) + u(i + 1, j) - 2 *
u(i, j))
    Next i
    'on impermeabel boundary (at m points):
    i = m: u(i, j + 1) = u(i, j) + Beta * (2 * u(i - 1, j) - 2 * u(i,
j))
  Next j

Else

  GoTo Check

End If

'Result: after t pressure
For i = 1 To m
  rsTP(i) = u(i, n + 1)
Next i

'Result
For i = 0 To im
  Cells(9 + i, 1).NumberFormat = "0.00"
  Cells(9 + i, 1) = dZ(i)
  Cells(9 + i, 2).NumberFormat = "0.00"
  Cells(9 + i, 2) = IP(i)
  Cells(9 + i, 3).NumberFormat = "0.00"
  Cells(9 + i, 3) = rsTP(10 * i)
Next i

'Determine area under Isochrone
'using trapezoidal approximation
'in H/m unit
AIP = 0
ATP = 0
For i = 0 To m - 1

```


Command Button (FDC)

```
Private Sub CommandButton1_Click()  
    Call FDC  
End Sub
```