Mohd. Kamaruddin Abd. Hamid

DKK 3352
Computer Aided Chemical Engineering (Excel/VBA)

Department of Chemical Engineering,
Faculty of Chemical & Natural Resources Engineering,
Universiti Teknologi Malaysia.

CONTENTS

COURSE OUTLINE

Department & Faculty: Department of Chemical Engineering Faculty of Chemical & Natural Resources Engineering	**Page 1 of 3**
Subject & Code: Computer Aided Chemical Engineering (DKK 3342) Total Lecture Hours: 8 hours x 2 Meetings	**Semester: 2 Academic Session: 2005/2006**

Lecturer : **Engr. Mohd. Kamaruddin Abd. Hamid**
Room No. : **N01-203**
Tel. No. : **07-5535517 (Office), 013-7417808 (HP)**
E-mail : **kamaruddin@fkkksa.utm.my**

Synopsis : This course introduces students to a solid introduction to programming concepts as well as numerical methods and statistical analysis. Students will receive instruction in the use of Excel spreadsheet basic and advanced features, VBA (Visual Basic for Applications) and macro programming as well as Excel's statistical functions. Students should master general programming concepts as well as gain an appreciation of formal problem solving methodology. Example problems draw from the chemical engineering field whereby the student learns to apply appropriate software or numerical methods. Problems will be taken from the areas of material and energy balances, thermodynamics, transport, kinetics, data fitting and analysis of experimental data and steady state and dynamic modelling.

Course Objectives : By the end of the course, students should be able to:

1. Successfully employ prerequisite Excel skills (opening and saving workbooks, printing basics, entering data, formatting cell contents, cell and range concepts, relative and absolute addresses, copying and pasting cell ranges, using the fill handle to generate sequences of values, shortcut keys, etc.
2. Properly organize material in a worksheet (titles, input values, other parameter values, formulas, results)
3. Create effective Excel charts (x-y, scatter, line, surface) including selection of appropriate trend lines
4. Generate plots (charts) from parametric information
5. Successfully employ Excel's built-in functions, basic math functions, computing sums, trigonometric functions, advanced math functions, conditional formatting, etc.
6. Solve problems requiring vector and matrix math including transpose, inversion, determinants, solving systems of linear equations
7. Solve problems and interpret results from linear regression using excel functions, two-coefficient linear regression models, polynomial regression and linear regression using Excel's regression analysis package
8. Understand how to select the "best" regression equation
9. Able to recognize and set up problems involving iterative solutions, including the use of solver
10. Able to solve optimization problems using solver
11. Be able to record, modify and write Excel macros. Understand how macros can modify the contents and display of cells.
12. Understand and successfully employ basic VBA programming concepts including data types, variables, accessing Excel built-in functions
13. Successfully employ programming structures including IF-THEN-ELSE, SELECT CASE, FOR-NEXT, DO [WHILE/UNTIL] LOOP
14. Effectively use the VBA debugger to uncover and correct errors in program logic
15. Solve a wide variety of chemical engineering and related engineering and mathematics problems.

Prepared by: Name: Mohd. Kamaruddin Abd. Hamid Signature: Date:	**Certified by: (Course Coordinator) Name: Signature: Date:**

COURSE OUTLINE

Department & Faculty: Department of Chemical Engineering Faculty of Chemical & Natural Resources Engineering	**Page 2 of 3**
Subject & Code: Computer Aided Chemical Engineering (DKK 3342) Total Lecture Hours: 8 hours x 2 Meetings	**Semester: 2 Academic Session: 2005/2006**

Chapter	Topic
1	**Introduction to Computer Aided Chemical Engineering** Background: Importance of this course to the engineering professional; Course Objectives; Suggested Resources; Prerequisite Computer/Excel Skills
2	**Introduction to Excel** Spreadsheet Basics; Problem Identification; Pointing to Enter Information; Absolute vs. Relative Addresses; Named Cells and Ranges; Built-In Functions; Error Messages in Excel; Formatting Cells; Freezing and Splitting Panes; Organization of Information **Homework 1**
3	**Graphing with Excel** Types of Charts: XY (Scatter) Charts, Line Charts; Modifying the Elements of a Chart; Adding Additional Sets of Data to a Chart; Parametric Plotting; Plotting Random Values; Trendlines; Error Bars; Surface Plots
4	**Excel Functions** Introduction to Excel Functions; Excel's Built-In Functions; Functions Listed by Category; Math; Trigonometry; Matrix Functions; Other Math and Trigonometry; Logical; Data and Time; Text and Data; Engineering; Information; Lookup and Reference; Statistical **Project 1**
5	**Matrix Operations in Excel** Matrix Manipulations: Vectors, Matrices, and Arrays; Basic Matrix Operations; Matrix Subtraction and Scalar Multiplication; Multiplying Two Matrices; Transposing Matrices; Inverting Matrices; Determinant of a Matrix; Solving Systems of Linear Equations
6	**Linear Regression in Excel** Linear Regression using Excel Functions; Linear Regression using Excel's Trend Line Capability; Other Two-Coefficient Linear Regression Models: Linear, Logarithmic, Polynomial, Power, Exponential, Moving Average; Polynomial Regression; Linear Regression using Excel's Regression Analysis Package **Homework 2, Project 2**
7	**Iterative Solutions using Excel** Introduction; Graphical Solution; Using User-Defined Functions in Excel; Other Iterative solution Methods; Direct Substitution Method, In Cell Iteration; Introduction to Excel's Solver; Optimization using the Solver; Nonlinear Regression; Testing the Regression Result

COURSE OUTLINE

Department & Faculty: Department of Chemical Engineering Faculty of Chemical & Natural Resources Engineering	**Page 3 of 3**
Subject & Code: Computer Aided Chemical Engineering (DKK 3342) Total Lecture Hours: 8 hours x 2 Meetings	**Semester: 2 Academic Session: 2005/2006**

Chapter **Topic**

8 **Using Macros in Excel**

Introduction; Macros and Viruses; Recording Macros; Programmed Macros (VBA): Changing the Value in the Active Cell, Changing Properties of the Active Cell, Changing the Selected Cell, Selecting a Range of Cells, Changing the Values in a Range of Cells, Changing the Properties of a Range of Cells

9 **Programming in Excel with VBA**

Visual Basic for Applications (VBA) Overview; Projects, Forms and Modules: Visibility Issues, Adding Buttons; Flowcharts; NS Diagrams; Getting Back the Power of Excel (Referencing Excel within VBA); Viewing the Members of the ATPVBA (or other Libraries) Object Browser; VBA Language Elements: Data Types, Variables, Programming Structures, Compiler Options; Declaring Variables: Scalars, Arrays; Programming Structures: If…Then…Else Conditional Statements, SELECT CASE Conditional Statements, FOR…[EACH]…NEXT Looping Structure, DO…[WHILE|UNTIL]…Looping Structure; Using the VBA Debugger; VBA Debugging Tips; Types of Errors: Syntax Errors, Run-Time Errors, Logical Errors; Invoking the Debugger; Toolbar Buttons; Using Debugging Tools Effectively: Break Point, Stepping Through Code, Run to Cursor, Locals Window, Watch Window

Homework 3, Project 3

Teaching Methodology : Instruction on basic topics; Instructor-guided demonstrations of features; Hands-on workshops with a maximum of 2 persons per computer; Detailed course notes

References :
- Power Programming with VBA/Excel (Chapra), Prentice Hall, ISBN 0-130-47377-4.
- A Guide to Microsoft Excel 2002 for Scientists and Engineers (Lienme), Elsevier, ISBN 0-750-65613-1.
- Spreadsheet Tools for Engineers Using Excel (Gottfried), McGraw-Hill, ISBN 0-072-48066-6.
- Problem Solving in Chemical Engineering with Numerical Methods (Cutlip), Prentice Hall, ISBN 0-138-62566-2.
- VBA for Dummies (Mueller), John Wiley, ISBN 0-764-53989-2.
- Excel 2000 Programming for Dummies (Walkenbach), John Wiley, ISBN 0-764-50566-1.

Assessment : Your final grade will be determined by proportionally weighting performance in the following areas:
- **3 Homeworks - 30%**
- **3 Projects - 60%**
- **Attendance and Participation – 10%**

COMPUTER SCIENCE OR SPREADSHEET ENGINEERING?

An Excel/VBA-Based Programming and Problem Solving Course

DANIEL G. CORONELL
Rose-Hulman Institute of Technology • Terre Haute, IN 47803

Over the past two decades, chemical engineering practice has been profoundly influenced by advances in computer hardware and software, stimulating debate within the academic community on how students should be prepared for computer applications in the "real world." At the crux of this debate is the relative importance of including a traditional introductory computer science course in the chemical engineering curriculum. Without question, in the "old days" the pathway for applying computers to the solution of engineering problems was to write your own program from scratch, usually in Fortran.

Today, industry is much less likely to engage their engineers with such tasks. The expectation is that commercial software vendors will supply them with user-friendly software packages that require little or no programming skills. A recent survey[1] by CACHE indicated that computing in the workplace for entry-level chemical engineers is clearly on the rise (over two-thirds of the approximately 300 respondents spent at least one-half of their workday at their computer). It was found that most of the time spent working on the computer involved user-friendly commercial software packages, with the most common application being Microsoft Excel. Nearly three-quarters of the respondents were not expected by their employers to be competent in any programming language. The most common programming language being used, and the one most highly recommended for inclusion in the chemical engineering curricula, was Visual Basic.

Based on these results, it might be argued that graduating chemical engineers would be more suitably equipped to contribute in an industrial setting if they were taught how to effectively use Excel rather than how to write a computer program in a language they may never use again. The numerous books,[2-5] trade journal articles,[6-8] software vendors,[9-12] and consultants[13-15] that demonstrate the use of Excel in engineering analyses underscore this point. This notion, however, overlooks the value of learning how to logically formulate a problem-solving strategy that is inherent in any programming course. Moreover, it omits the necessary exposure to programming concepts (*e.g.*, loops, decision constructs, *etc.*) for the fraction of students who may be required to do some type of programming in an R&D setting or in graduate school.

This paper describes a compromise approach that combines instruction on the use of Excel as well as computer programming concepts by way of Excel's macro programming language, Visual Basic for Applications (VBA). The benefit to students is that they can learn the practical aspects of "spreadsheet engineering" as well as the more generally applicable concepts of computer programming. Additionally, they gain a clearer understanding of how the course material applies to their future profession since the course is taught within the chemical engineering department. The benefit to the instructor is the ability to consolidate the presentation of course material through the use of a single software package. This paper describes the format and content of a freshman-level course that has been designed to replace the more traditional introductory computer science course.

COURSE FORMAT

Programming and Computation for Chemical Engineers is a two-credit-hour course that chemical engineering majors at Rose-Hulman Institute of Technology are required to take in the spring quarter of their freshman year. The class meets

***Dan Coronell** received his BS from the University of Illinois-Urbana and his PhD from the Massachusetts Institute of Technology, both in chemical engineering. After graduation he worked in the chemical, semiconductor, and engineering software industries for over nine years before joining the faculty at Rose-Hulman Institute of Technology, where he is presently Associate Professor.*

two times per week for fifty-minute periods over a ten-week quarter. At this point in the curriculum, students have typically completed two quarters of calculus and chemistry and one quarter of physics. They are also concurrently enrolled in a freshman-level design class that introduces them to many concepts of importance to chemical engineers[16]. The early introduction of chemical engineering concepts into their curriculum provided by the two courses is beneficial to the students, as will be discussed below.

Prior to the first meeting of this newly redesigned course, a survey was conducted to assess the level of expertise in using Excel and experience with any programming language. Approximately two-thirds of the 66 students that were originally registered for the two sections responded to the survey. The first part of the survey asked the students to select one of five different categories that best characterized their ability to use Excel. The selection options included

- Power user
- Pretty comfortable using it to process data and make plots
- Have used it before several times and know the basics
- Have only started using it since my freshman year
- Have never used it before

The results are summarized in Table 1. While all of the respondents had used Excel to some extent, most of the course material consisted of techniques and applications that the students had never been exposed to in the past.

The second part of the survey asked students to identify any programming languages they had previously learned in coursework or through work experience. As can be seen in Table 2 below, two-thirds of the respondents had no previous programming experience. Note that a few respondents had experience in more than one type of programming language.

The classroom instructional technology at Rose-Hulman greatly facilitated the execution of this type of course. Every classroom is equipped with a laptop computer projector and wireless network capabilities. In addition, each student at Rose-Hulman is issued a laptop computer at the beginning of their freshman year. The students were required to bring their laptop computers to class each day. A typical 50-minute class period was discretized into 20 minutes of traditional lecture, 20 minutes of computer laboratory, and 10 minutes of discussion and reflection.

TABLE 1
Survey of Students' Level of Expertise Using Excel

	Power User	*Pretty Comfortable*	*Know Basics*	*Just Started*	*Never Used Before*
# of Respondents	2	17	13	10	0

TABLE 2
Survey of Students' Prior Programming Experience

	Visual Basic	*C/C++ C#*	*Java HTML*	*Pascal*	*Matlab*	*None*
# of Respondents	10	9	5	1	1	28

The initial lecture period would usually include an instructor-led example problem with students following along on their laptops, followed by a computer lab assignment on a problem related to the lecture topic. The students were free to work together and to ask questions during this time. The last few minutes of class were used to obtain closure on the subject matter where the computer lab solution would be provided, the relevance of the material would be reinforced, and any remaining questions would be answered. All in-class quizzes and homework assignments were submitted electronically as Excel workbooks.

While most of the students had not yet taken any core chemical engineering courses, every opportunity was taken to expose the students to the kinds of problems they would see later in the curriculum. This served to benefit the students in several ways. First, they became more engaged in learning the programming and the problem-solving concepts when it was demonstrated that these fundamentals could be applied to chemical engineering-related problems. This was true in spite of the fact that they possessed only a cursory understanding of the underlying fundamentals at this point in their education. Another benefit was that, early in the curriculum, students were exposed to a sample of what the chemical engineering profession entails. It was observed that many of the students were interested in chemical engineering for reasons ranging from the desire to follow the path of a family member or close friend to a desire to have a good paying job when they graduated. Approximately 5% (3/66) of them ended up changing their major after learning more about the chemical engineering profession. In the following section, the specific learning objectives and content of the course are described.

COURSE OBJECTIVES AND CONTENT

In a broader perspective, the objective of *Programming and Computation for Chemical Engineers* is to begin the process of introducing the computer as an engineering problem-solving tool. As described in the preceding section, the approach to satisfying this objective relies upon active learning, relevant example applications, and modern classroom instructional technology. This high-level objective of the newly redesigned course was refined into the specific learning objectives of

- Becoming proficient at using Excel to perform scientific and engineering calculations and graphical analysis.
- Understanding the essential elements of structured and object-oriented programming as it applies to VBA.
- Being able to construct customized VBA-based spreadsheet functions to enhance the engineering problem-solving capabilities of Excel.

The first one-quarter of the course consisted of formal instruction on spreadsheet techniques and tools. This is illustrated in Figure 1a, where the students' progression of learning begins with basic operations in a worksheet cell and progresses outward to increasingly complex worksheet operations. The remaining three-quarters of the course was devoted to instruction on VBA programming in Excel. Since VBA is a separate application that is integrated into the Excel environment, this involved two distinct aspects—learning the VBA programming language and learning how to interface with Excel from a VBA program. The latter required the students to work with Excel's so-called Object Model, a heirarchical, object-oriented interface to the Excel application that facilitates manipulation of all the elements of an Excel workbook.

A synopsis of the programming elements of VBA that were included in the course is shown in Figure 1b. VBA contains many of the same elements that are common to other programming languages—variables, loops, decision constructs, *etc*. Thus, the students obtain a conceptual understanding that enables them to more easily learn a different progamming language, such as C++ or Java. This was an important consideration since some of the students will continue their education in graduate school where they may be involved in computational research requiring programming skills in other languages.

EXAMPLE APPLICATIONS

In the preceding section, the general features of Excel spreadsheets and VBA programs that the students were exposed to, and which underly the course learning objectives, were described. The students developed their skills at using these tools by applying them to a number of engineering-related problems. The types of applications that were included and the specific example problem they were asked to solve are summarized in Table 3.

Most of the applications were first explored using a spreadsheet-only approach, then subsequently implemented in a VBA program. It is again emphasized that while the students did not possess a deep understanding of how the design equations for a particular application were derived, their appreciation for the usefulness of the computer skills they were acquiring was nonetheless heightened. Moreover, they finished the course with a better understanding of what was ahead of them in the chemical engineering curriculum.

One of the applications listed in Table 3 is the numerical solution of ordinary differential equations (ODEs). The students learned how to solve ODEs using both the explicit Euler method as well as the 4th-order Runge-Kutta method. As an example, the following differential equation representing the draining of a cylindrical tank was numerically solved by applying the 4th-order Runge-Kutta method, using both the spreadsheet and VBA program implementations.

$$\frac{dh}{dt} = -\left(\frac{d_{hole}}{d_{tank}}\right)^2 \sqrt{2gh}$$

In this equation, h is the height of fluid in the tank, t is the cumulative draining time, d_{hole} is the diameter of the drain hole located at the bottom of the tank, d_{tank} is the tank diameter, and g is the gravitational acceleration constant. This equation is widely known as Torricelli's formula, and possesses an exact solution. This enabled the students to also explore the concepts of integration step size and associated error, as well.

The two implementations are shown in Figure 2 below. The students solved this problem using the spreadsheet implementation early in the quarter, and subsequently revisited the problem after learning how to create customized VBA function procedures. This helped them to appreciate the advantages of the VBA approach, including the conciseness of the implementation

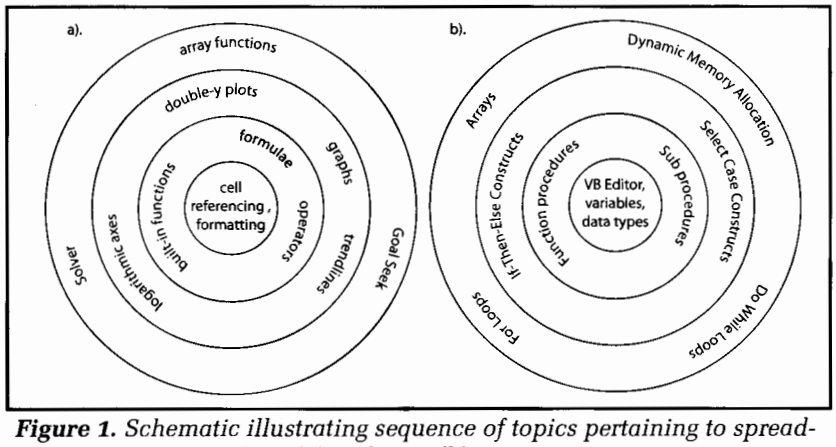


Figure 1. *Schematic illustrating sequence of topics pertaining to spreadsheet (a) and VBA (b) instruction.*

TABLE 3
Applications of Computer Skills Included in Course

Application	*Implementation*	*Example*
Engineering formula involving transcendental functions	Excel VBA	Pressure dynamics and release rate of choked flow from a high-pressure gas cylinder
Parameter estimation	Excel VBA	Determination of first-order rate constant from experimental data using the integral method
Solution of linear systems of algebraic equations	Excel	Steady-state material balance
Solution of nonlinear algebraic equations	Excel VBA	Determination of friction factor using the Colebrook equation
Numerical integration	Excel VBA	Sizing of a gas-liquid scrubber
Numerical solution of ordinary differential equations	Excel	Draining of a tank using Torricelli's formula

and the ability to reuse the function to solve a different problem by simply redefining the ODE in the VBA function. Additionally, the VBA implementation reduces the possibility of introducing a typographical error since the Runge-Kutta terms do not have to be retyped for each problem.

SUMMARY AND CONCLUDING REMARKS

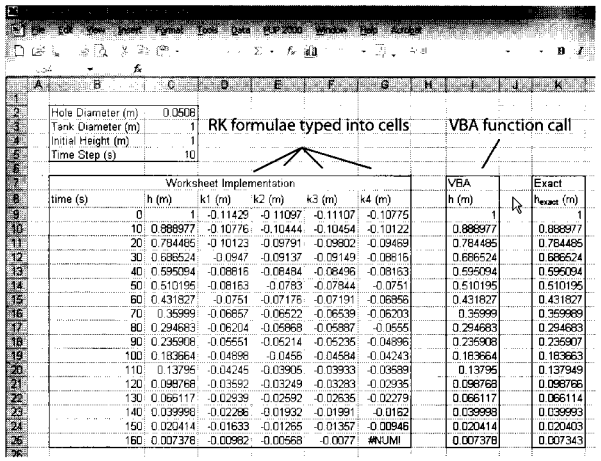
A newly redesigned freshman programming course for chemical engineers that supplants the traditional introductory computer science course has been described in this paper. The course focuses on the use of Excel spreadsheet and VBA programming techniques to solve engineering-related problems in response to the needs of industry as unveiled in a recent CACHE survey. The course also serves as a metric for students to assess their interest and aptitude for the chemical engineering profession at an earlier point in their curriculum than previously allowed.

Some remaining issues will have to be addressed, one of which includes the lack of suitable textbooks that relate to the course content. Many textbooks on Excel and VBA programming are available, but no textbook could be identified that focused on engineering-related applications. The students responded quite positively to learning programming skills within the context of solving engineering-related problems. A textbook that is tailored to such a class would greatly facilitate the continued offering of the course. Additionally, in order for the students to sustain and leverage the skills obtained in the course, a department-wide involvement to incorporate Excel/VBA problem-solving methods where applicable into their higher-level engineering courses must be initiated and sustained. The familiar saying, "Use it or lose it!" applies here.

It is also important to acknowledge that while spreadsheets and Visual Basic programs are useful for solving many relatively routine engineering problems as illustrated in the preceding section, some problems require more sophisticated computational tools. This becomes apparent to the students later in the curriculum as they take courses in fluid mechanics, transport phenomena, thermodynamics, reactor engineering, and design. Here they will learn about software packages designed to perform computational fluid dynamics calculations, predict detailed fluid properties, optimize process flowsheets, and more. Thus, the *Programming and Computation for Chemical Engineers* course represents the commencement of their education in using the computer as an engineering tool. The real value of the approach outlined here is, perhaps, that the students can more clearly and immediately see that computers can be programmed to efficiently solve chemical engineering problems.

a).

Excel worksheet for implementing spreadsheet and VBA-based solution to ODE.



Hole Diameter (m)	0.0508									
Tank Diameter (m)	1									
Initial Height (m)	1									
Time Step (s)	10									
		Worksheet Implementation						VBA		Exact
time (s)		h (m)	k1 (m)	k2 (m)	k3 (m)	k4 (m)		h (m)		h_{exact} (m)
0		1	-0.11429	-0.11097	-0.11107	-0.10775		1		1
10		0.888977	-0.10776	-0.10444	-0.10454	-0.10122		0.888977		0.888977
20		0.784485	-0.10123	-0.09791	-0.09802	-0.09469		0.784485		0.784485
30		0.686524	-0.0947	-0.09137	-0.09149	-0.08816		0.686524		0.686524
40		0.595094	-0.08816	-0.08484	-0.08496	-0.08163		0.595094		0.595094
50		0.510195	-0.08163	-0.0783	-0.07844	-0.0751		0.510195		0.510195
60		0.431827	-0.0751	-0.07175	-0.07191	-0.06856		0.431827		0.431827
70		0.35899	-0.06857	-0.06522	-0.06539	-0.06203		0.35999		0.359989
80		0.294683	-0.06204	-0.05868	-0.05887	-0.0555		0.294683		0.294683
90		0.235908	-0.05551	-0.05214	-0.05235	-0.04896		0.235908		0.235907
100		0.183664	-0.04898	-0.0456	-0.04584	-0.04243		0.183664		0.183663
110		0.13795	-0.04245	-0.03905	-0.03933	-0.03589		0.13795		0.137949
120		0.098768	-0.03592	-0.03249	-0.03283	-0.02935		0.098768		0.098766
130		0.066117	-0.02939	-0.02592	-0.02635	-0.02279		0.066117		0.066114
140		0.039998	-0.02286	-0.01932	-0.01991	-0.0162		0.039998		0.039993
150		0.020414	-0.01633	-0.01265	-0.01357	-0.00946		0.020414		0.020403
160		0.007378	-0.00982	-0.00568	-0.0077	#NUM!		0.007378		0.007343

b).

VBA code called from worksheet cell formula.

```
Option Explicit

Function RungeKutta(t As Double, y As Double, dt As Double) As Double
'This function integrates a single ODE using a 4 term Runge-Kutta method.
'The ODE is defined in the private function dydt.

    Dim k1 As Double, k2 As Double, k3 As Double, k4 As Double

    k1 = dt * dydt(t, y)
    k2 = dt * dydt(t + 0.5 * dt, y + 0.5 * k1)
    k3 = dt * dydt(t + 0.5 * dt, y + 0.5 * k2)
    k4 = dt * dydt(t + dt, y + k3)

    RungeKutta = y + (k1 + 2 * k2 + 2 * k3 + k4) / 6

End Function

Private Function dydt(t As Double, y As Double) As Double

    Dim dhole As Double, dtank As Double
    Const g As Double = 9.8066

    dhole = 0.0508
    dtank = 1

    dydt = -(dhole / dtank) ^ 2 * Sqr(2 * g * y)

End Function
```

Figure 2. *Runge-Kutta solution of an ODE describing draining of a tank using a spreadsheet (a) and a VBA program (b).*

It is also important to acknowledge that while spreadsheets and Visual Basic programs are useful for solving many relatively routine engineering problems as illustrated in the preceding section, some problems require more sophisticated computational tools. This becomes apparent to the students later in the curriculum as they take courses in fluid mechanics, transport phenomena, thermodynamics, reactor engineering, and design. Here they will learn about software packages designed to perform computational fluid dynamics calculations, predict detailed fluid properties, optimize process flowsheets, and more. Thus, the *Programming and Computation for Chemical Engineers* course represents the commencement of their education in using the computer as an engineering tool. The real value of the approach outlined here is, perhaps, that the students can more clearly and immediately see that computers can be programmed to efficiently solve chemical engineering problems.

REFERENCES

1. Edgar, T., "Computing Through the Curriculum: An Integrated Approach for Chemical Engineering," CACHE Fall 2003 Newsletter, <http://www.che.utexas.edu/cache/newsletters/fall2003_cover.html>
2. Bloch, S.C., *Excel for Engineers and Scientists,* 2nd ed., Wiley, New York, (2003)
3. Filby, G., *Spreadsheets in Science and Engineering*, Springer-Verlag, New York, (1998)
4. Kral, I.H., *The Excel Spreadsheet for Engineers and Scientists*, Pearson Education, New Jersey, (1997)
5. Orvis, W. J., *Excel for Scientists and Engineers*, 2nd ed., Sybex, San Francisco, (1996)
6. Peress, J., "Working with Non-Ideal Gases," *Chem. Eng. Prog.*, p. 39, March (2003)
7. Jevric, J., and M.E. Fayed, "Shortcut Distillation Calculations via Spreadsheets," *Chem. Eng. Prog.*, p.60, December (2002)
8. Anthony, J., "Elements of Calculation Style," *Chem. Eng. Prog.*, p.50, November 2001.
9. Chemeng Software, <http://www.chemengsoftware.com>
10. ChemSheet Software, <http://gttserv.lth.rwth-aachen.de/~cg/Software/ChemSheet/IndexFrame.htm>
11. Excel Unit Conversion, <http://www.unit-conversion.com/excel.htm>
12. The Chemical Engineers' Resource Page, <http://www.cheresources.com>
13. Beyond Technology, <http://www.beyondtechnology.com>
14. Emagenit, <http://www.emagenit.com>
15. Spreadsheet World, <http://www.spreadsheetworld.com>
16. Sauer, S.G., "Freshman Design in Chemical Engineering," *Chem. End. Ed.*, **38**, 222(2004) ❐

Introduction to Computer Aided Chemical Engineering

Background: Importance of This Material to the Engineering Professional

The widespread use of desktop computers has lead to the development of a variety of software products that can be used in the solution of engineering problems. Generally these fall into two categories:

- Programming languages: Fortran, C, C++, Pascal, etc.
- Mathematics software: Matlab, MathCAD, Maple, Polymath, etc.

In the past, the computer was only used for the difficult and time consuming task of modeling and simulating unit operations, processes and control systems. Routine calculations were carried out using hand-held calculators using essentially the same techniques as used in the slide rule era.

To a large extent, current chemical engineering homework, exams, and other assignments mostly center around hand calculations and paper and pencil "reports". Limiting the use of computers to solving "difficult problems" was justified because most "languages" would require large amounts of time to be spent

1. Deriving the model equations for the specific problem (basically "setting up the problem" but not solving it)
2. Finding appropriate numerical methods to solve the problem (e.g., if you needed to integrate a set of discrete data values, what methods "algorithms" are available)
3. Write/enter/debug the program code
4. Run and analyze the results for validity and precision.

It was soon recognized that the second and third tasks were minor contributions to the "learning of subject matter" in most chemical engineering courses but they represented the most time consuming and frustrating parts of a computer assignment. The computer enabled students to solve realistic problems, but the time spent on "non-chemical engineering" subject matter was much too long.

With the introduction of interactive numerical software a major change in chemical engineering education was made possible. This change has been called a "paradigm shift" by Fogler. Using interactive numerical software the student's main task was once again returned to being "set up the model equations." The interactive program was responsible for providing accurate
DKK 3352 – Computer-Aided Chemical Engineering solutions displaying the results in easy to interpret numerical and graphical representations without being involved in setting up the algorithms to perform the individual operations. For example, data values could be sorted without having to writing and implementing a sorting algorithm.

Several problems exist however in interactive numerical software:
1. Current textbooks still assume most homework problems must be solvable by hand
2. Some "interactive numerical software" is expensive and even if licensed on university computers is not generally available where the student solves problems
3. There is a significant learning curve for products that are especially comprehensive (including learning complicated command structures that are not generally intuitive or easy to learn). An example of this is having to remember that all storage of information in Matlab is as a matrix and must be constantly accounted for when referencing data (even for handling constants).

In selecting Excel and VBA the department has sought to retaining the advantages of interactive numerical software and permit enhancing native capabilities of the product using Macros and Programs. Students should feel comfortable with the Excel "interface" and adapt easily to the VBA environment.

Most importantly, this powerful combination should allow students to quickly solve trivial problems and with slightly more effort solve problems of considerable complexity.

Course Objectives

Course Objectives: The overall goal of the course is to provide a solid introduction to programming concepts as well as numerical methods and statistical analysis. Students will be introduced to formal problem solving methodology. Example problems projects draw from the chemical engineering field whereby the student learns to apply appropriate software or numerical methods. Problems will be taken from the areas of material and energy balances, thermodynamics, transport, kinetics, data fitting and analysis of experimental data and steady state and dynamic modeling.

Suggested Resources

Many resources are available to assist learning the material covered in this course. Frequently the web can be successfully searched to locate relevant procedures and code.

Printed (textbooks) are also a valuable source of information. These can be found or ordered from the bookstore as well as online sources. Some of the sources used in developing this course include:

- *Power Programming with VBA/Excel* (Chapra) Prentice Hall, ISBN 0-130-47377-4
- *A Guide to Microsoft Excel 2002 For Scientists and Engineers* (Lienme) Elsevier, ISBN 0-750-65613-1
- *Spreadsheet Tools for Engineers Using Excel* (Gottfried) McGraw-Hill, ISBN 0-072-48066-6
- *Problem Solving in Chemical Engineering with Numerical Methods* (Cutlip) Prentice Hall, ISBN 0-138-62566-2
- *VBA For Dummies* (Mueller) Wiley, ISBN 0-764-53989-2
- *Excel 2000 Programming For Dummies* (Walkenbach) Wiley, ISBN 0-764-50566-1

Prerequisite Computer/Excel Skills

It is expected that all students in this course possess a familiarity and capability with certain basic computer and spreadsheet skills and concepts. These have most likely been acquired in basic engineering and chemical engineering courses as well as by “osmosis”. If you do not feel you have any of these skills, you should individually make efforts to acquire them as soon as possible.

A partial list of these prerequisite skills is:

- Familiarity with Window Operating System (Windows XP)
- Microsoft GUI (Graphical User Interface)
- Microsoft Excel Window Nomenclature (Title Bar, Control Buttons, Menu Bar, Toolbars, etc.)
- Microsoft Excel Nomenclature (Workbook, Spreadsheet, Cell, Range, Label, Number, Formula, etc.)
- Starting Programs (Launching)
- Opening and Saving Workbooks
- Printing Basics

- Entering Data (Labels, Numbers, Formulas, etc).
- Formatting Cell Contents (Basics such as Font Selection, Font Size, Font Color, Alignment, Border, Background, Numerical Format, etc.)
- Cell and Range Concepts (A14, A1:C5)
- Relative and Absolute Addresses (A1, A$1, $A1, A1)
- Copying and Pasting Cell Ranges
- Pasting Formulas and Pasting Values
- Using the Fill Handle to Generate Sequences of Values
- Shortcut Keys (Cntl-C, Cntl-V, Cntl-X, Cntl-Z, Cntl-Y, etc.)
- Exiting Excel

Introduction to Excel

Introduction

Microsoft Excel is a graphical spreadsheet program that runs under the Microsoft Windows environment on PCs or on Apple Macintosh Computers running Apple system software. Excel is the oldest spreadsheet program to use a Graphical User Interface, and its interface is one of the most advanced.

Its strength lies in its ease of use for new users and its speed of use for experienced users. For most tasks there are at least two ways of doing the same operation. This provides new users with a simple method, and experienced users with a shortcut method.

Spreadsheets are used for a wide variety of tasks from the simple presentation of tables of numbers to complex simulations of scientific processes and production of computer-based training materials.

At its simplest, Excel can be used for entering and printing data. It contains extensive formatting features, including the use of colour, borders, a choice of font styles, sizes and colours, and an assortment of number formats.

Features of data can often be most easily seen when the data are represented visually, and production and editing of charts is very straightforward process.

Spreadsheets were originally used to carry out calculations. In addition to providing the facility for the user to construct formulae, Excel now provides a wide range of built-in functions.

Complex analytical tools are available as "**Add Ins**". These features are not installed by default, and have to be installed later from the **Tools** menu. Examples of these tools are the **Analysis Toolpak** which provides statistical functions and the **Solver** tool for solving linear programming problems.

In addition, **NAG Add Ins** are also available. These consist of a suite of statistical routines that can be added to an Excel application.

It is also possible to use the high level language **Visual Basic for Applications** in order to automate tasks and carry out complex operations, and a programming environment is provided for its use.

The Excel Interface

The Excel interface follows many Microsoft Windows conventions that users of other Windows programs will be familiar with. There is the main Excel window, menus, toolbars, a status line and sub-windows.

Starting Excel

To use Excel you need to be logged on to the network, and running **Microsoft Windows XP**. To run the program, click on the **<Start>** icon on the taskbar to display the **Start** menu and then move the mouse pointer to **Programs**, **MS Office** and click on **Microsoft Excel 2003**.

The Opening Display

Excel may take a minute or so to load, but once it has done so the screen should look like the diagram shown below.

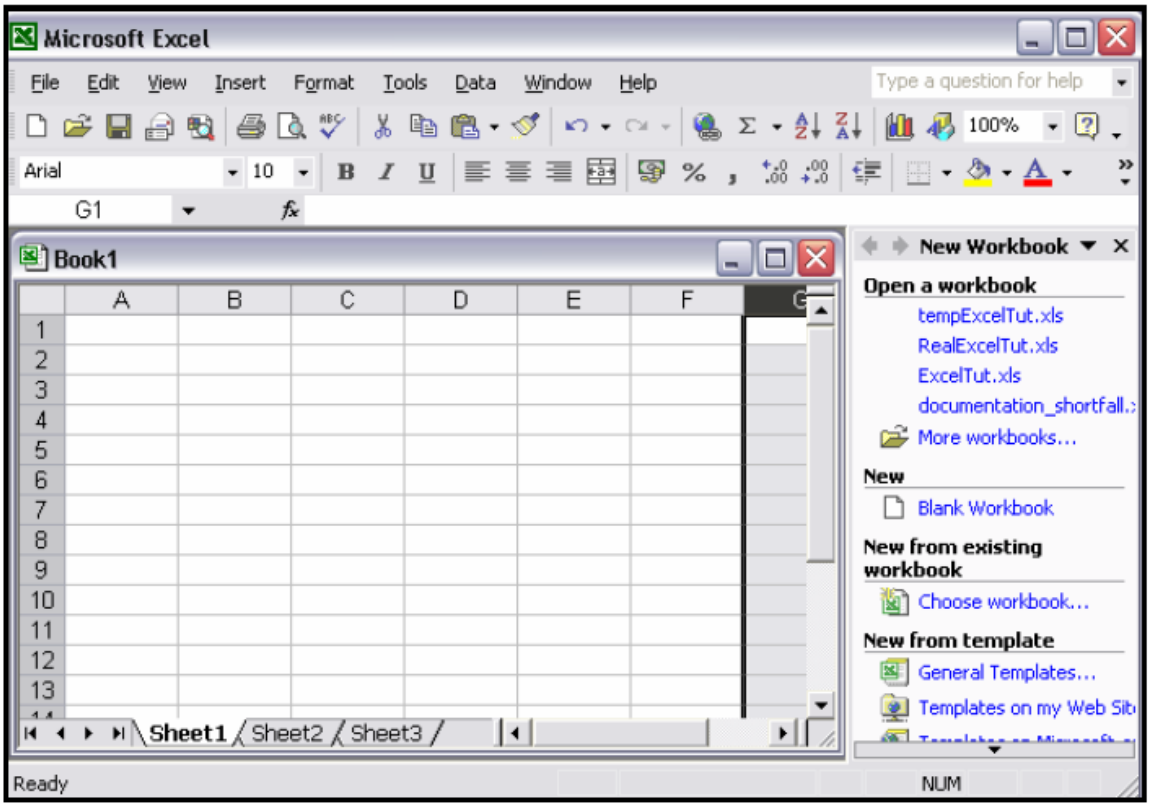


Figure 1. The Opening Excel Screen

The main Excel window contains a title bar, a collection of menus, two toolbars, a formula bar and a status bar. If one of the sub-windows becomes larger than the main window, or if it shifts outside the area of the main window, then the main window will also develop scroll bars. The panel on the right can be closed by clicking on its close icon.

Menus

Excel has nine main menus available when you are working on a worksheet. As far as is possible these menus are organised and labelled in the same way as the menus in Word 2003. The list of menu options available changes when you are viewing charts. The menus follow the standard Windows format of having the File and Edit menus to the left hand side of the screen, and the Window and Help menus on the right.
The menus are described in more detail below.

Main Menu File

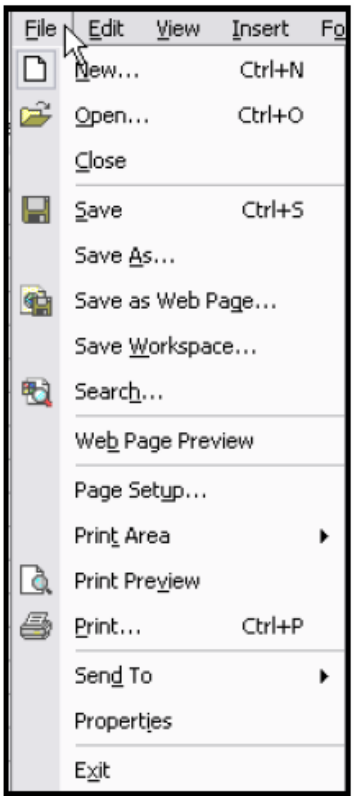


Figure 2. The File Menu

The File menu deals with all the file-handling aspects of Excel.

Options are available to open an existing file, create a new file, save a file or print a file.

The **Save As** facility allows an Excel file to be saved in a different file format such as a text file, or an HTML document for use on the web.

Saving the **Workspace** will save the sheets in the way they are currently arranged on screen.

Excel has some of its own file management capabilities, so there is a **Search** capability to search for a sheet.

The file can be previewed before it is printed, and the area to be printed can be specified using the **Print Area** option.

The **Send To** option is used to send the contents of a worksheet either as an email attachment, or in the body of an email message.

There is also the option of saving summary information about the current sheet, using the **Properties** option.

Finally the command to **Exit** Excel is here.

Edit

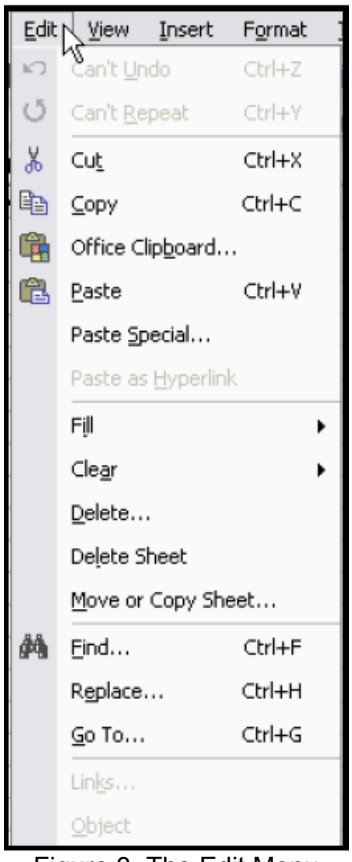


Figure 3. The Edit Menu

The **Edit** menu also shares much in common with other MS Windows applications. This menu contains commands for undoing the last action. **Cut**, **Copy** and **Paste** commands are available and there are some special copy options for certain elements. In addition to the normal Paste option there is a **Paste Special** option that allows the information that is being pasted to be linked − e.g. from another worksheet. It is also possible to paste only part of the information, e.g. just the data but not the formulae that created the data.

The **Fill** option allows the insertion of a series such as months of the year.

The **Delete** and **Clear** options relate to the removal of data inside a range of cells, the removal of columns or rows in a worksheet, or the removal of an entire sheet.

Move or **Copy Sheet** options allow the current sheet

to be moved or copied either to a different position in the current file, or to a different file.

Find and **Replace** functions are also in this menu.

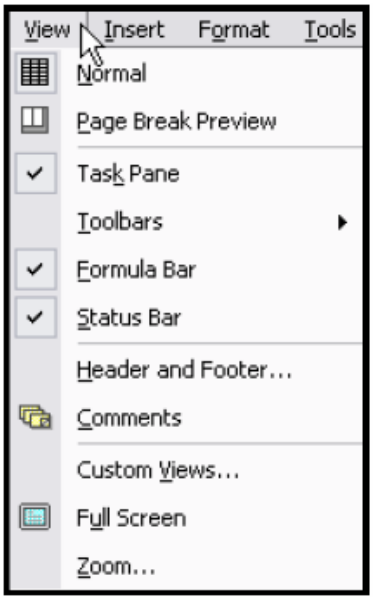


Figure 4. The View Menu

View

The **View** menu deals with how the Excel interface is displayed.

It allows the user to choose whether or not to display particular objects such as **menus**, **headers** and **footers**, **toolbars**, the **task pane**, comments, the **status bar** and the **formula bar**.

From here the available **toolbars** can be customised by adding or removing buttons.

A **Zoom** facility is included to set the magnification factor.

Insert

The **Insert** menu allows insertion of objects into an Excel worksheet.

Examples of objects include charts and macros.

Objects from other Windows programs can also be imported, e.g. drawings, photographs or sound clips.

In addition, hyperlinks can be included.

The **Function** command allows selection of a function from a list when creating a formula. Functions available are grouped into categories such as mathematical, statistical, financial or text.

The command dealing with the naming of ranges (rectangular blocks of cells) is also here.

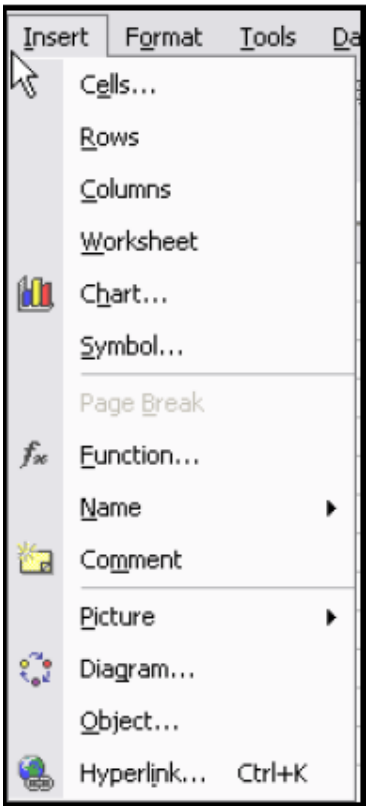


Figure 5. The Insert Menu

Format

The **Format** menu relates to the way the information is displayed.

From options within the Format menu, the typeface, size, style and colour of the content can be specified.

Excel provides the facility to save a collection of formatting options as a style.

This simplifies the process of producing spreadsheets with consistent formatting.

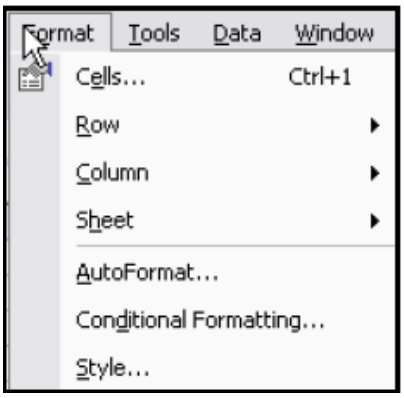


Figure 6. The Format Menu

Tools

The **AutoCorrect** option can correct common mistakes, such as unintentional use of the **<Caps Lock>** key or spelling errors.

The second set of commands allows several users to edit the workbook simultaneously.

The third set of commands relates to performing "what-if" analysis on the spreadsheet. The **Solver** option is a tool for solving linear programming problems.

The final group of commands includes the **Macro** command which allows the user to record macros and use the Visual Basic for Applications (**VBA**) programming environment.

Add-Ins can increase the functionality of Excel by adding commands, routines, menu items and functions to the base set provided with Excel.

Finally there are options to **Customize** the toolbars and menus, and an **Options** command that brings up a 13-page dialog box of settings that can be modified.

Figure 7. The Tools Menu

Data

The **Data** menu concerns using Excel as a flat-file database. The **Filte**r option enables a dataset to be searched.

The **Sort** Options allows sorting on one or more columns. The **Consolidate** command is one of the ways Excel uses its 3- D spreadsheet capabilities to compile the information from several sheets into one summary sheet.

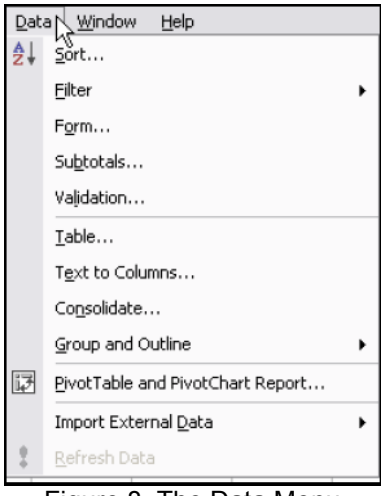


Figure 8. The Data Menu

The **PivotTable** command at the bottom of the menu is one of the most powerful features in Excel, allowing information from a spreadsheet to be viewed in a dynamic 3-dimensional model.

Finally, the **Import External Data** command runs the Microsoft Query program which allows a relational database, e.g. Microsoft Access to be queried, and inserts the resulting data into Excel for analysis.

Window

The **Window** menu is found in many MS Windows programs, and it contains a series of commands which allow the movement, selection and resizing of sub-windows in Excel.

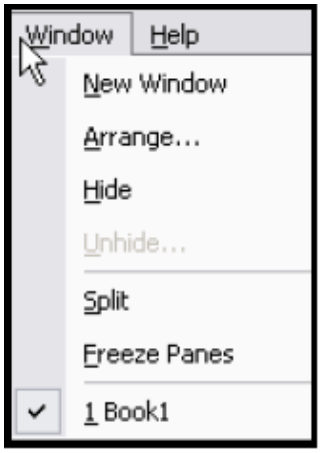


Figure 9. The Window Menu

It may be necessary to work with several sub-windows open at once. The **Arrange** command enables the sub-windows to be arranged in various ways.

The middle section concerns the active worksheet. The **Split** command allows up to four different areas of the same spreadsheet to be viewed simultaneously.

Freeze Panes enables column and row headings to be kept static when scrolling through a document. Both these features are extremely useful for managing large worksheets.

Help

This menu contains the commands for the Excel Help System. If the Office Assistant is installed, this can be accessed by clicking on the help icon on the **Standard** toolbar. This allows searching for help on a particular topic and can be configured to provide automatic help for common tasks.

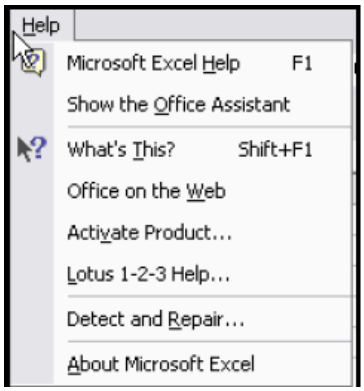


Figure 10. The Help Menu

Users who have previously used **Lotus 1-2-3** (or As-Easy-As) can access help specifically directed at them.

Spreadsheet Basics

In this course all spreadsheets must follow a standard format and observe other rules regarding organization and appearance. These rules and format will be developed as the course unfolds.

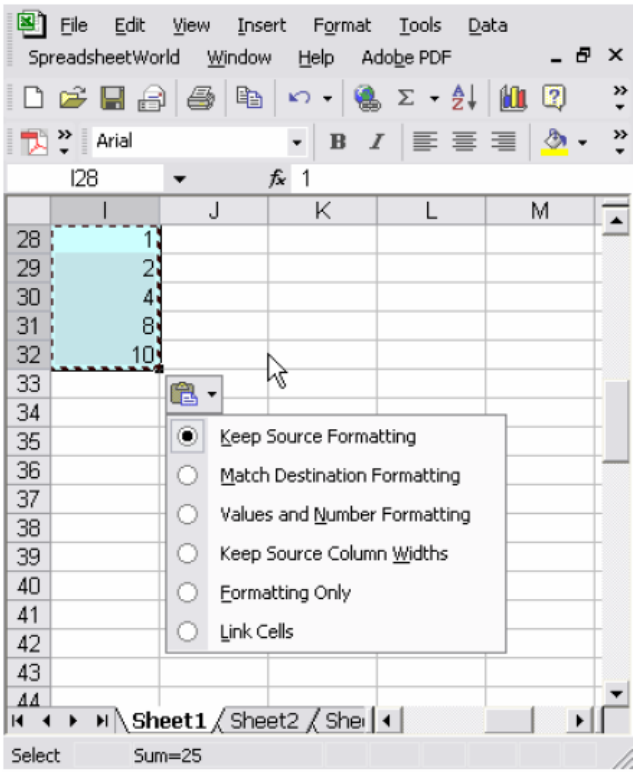
Problem Identification: All spreadsheets must contain a standard “boiler plate” for student and problem identification. This information appears in the “home” position, A1. Note the alignment of the information (right, left, etc).

A	B	C	D	E	F	G
Name:	Mohd. Kamaruddin Abd. Hamid					
Course:	DKK 3352					
Date:	12-Feb-06					
Assignment:	Classwork 1					
Description	Buble Point Calculaton for An Idel Binary Mixture Using Antoine's Equation (n-Pentane/n-Hexane) 1=n-pentane, 2=n-hexane					

Pointing to Enter Information: Normally entering cell references (B14, or B5:C10) is tedious and prone to error. It is usually preferable to enter such information by “pointing” to the desired cell or range of cells. Simply use the mouse to click on or drag over the desired item and press “Enter” when complete.

Once acquired, this skill is second nature and much faster than typing cell references.

Using Paste Options During a Copy Operation: New to Excel XP is the opportunity to format cells with or without formatting when pasting information. Notice in the screen capture below, the clipboard icon allows one to paste with various options. Practice with this feature to see how it functions.



Absolute vs Relative Addresses: Normally we type addresses or point to cells simply to refer to the information stored at those locations. However, when one copies and pastes cells containing addresses we can have Excel automatically "offset" those addresses relative to the movement of the copy from the original. This is a concept you should already be familiar with. Cells addresses can be "switched" between absolute and relative by repeatedly pressing the F4 key while you are editing the address in a formula.

Named Cells and Ranges: One of the more powerful but often overlooked features of Excel is to use "names" to represent cell or range addresses. For example, it will be much easier to refer to the temperature T as T rather than B12 (or B12 to be safe) and Pc rather than B13. You can associate names with cell values (contents) simply by putting the symbol for the name in the left cell and the value immediately to the right (as shown below).

Input Variables			
Description	Symbol	Value	Units
Temperature	T	450	K
Critical Pressure	Pc	111.3	atm
Critical Temperature	Tc	405.5	K
Gas Constant	Rgas	0.08206	atm L / gmol K

Then highlight the cell values AND names by dragging over them and select "Insert/Name/Create" from the top menu to view the following submenu:

Click "left" in this case and the names are assigned.

Certain names are "illegal" (those that would be confused with actually cell addresses (T1 or R3 for example since there are cells in the spreadsheet that already have those names). You also cannot use "C" or "R" by themselves, hence the use of Rgas in the example. Excel will attempt to "fix-up" these cases by using an underscore character after the name R_ or T1_ and you can employ these as well.

Input Vectors		
Description	Symbol	Value
Antoine's Data	A_1	6.85221
	B_1	1064.63
	C_1	232

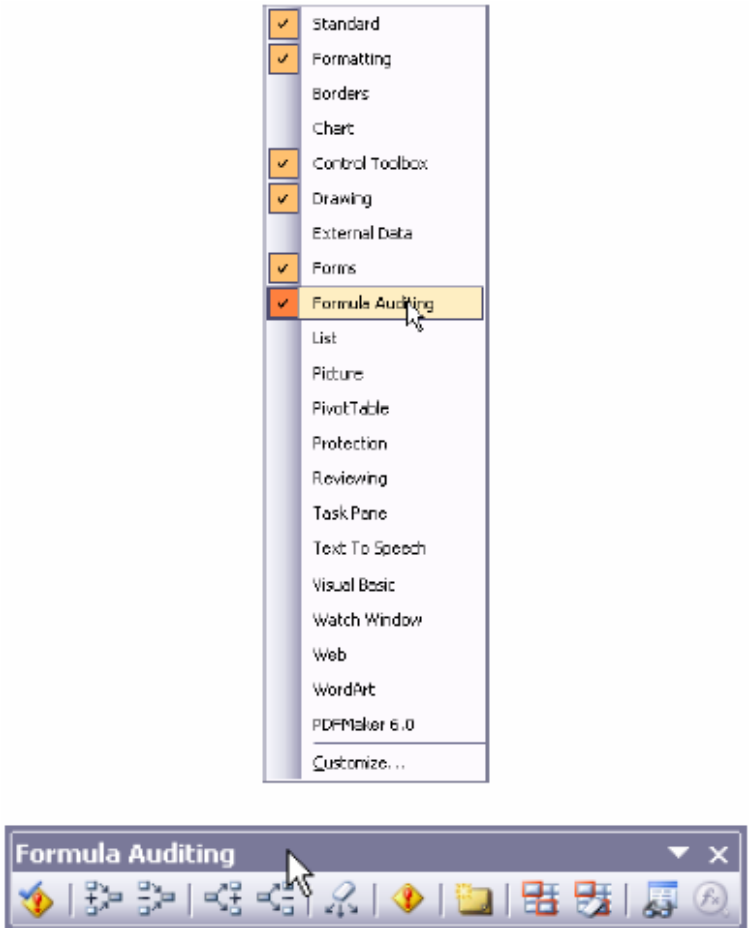
This allows you to use the defined names directly in formulas, for example:

= 27/64*(Rgas^2*Tc^2/Pc)

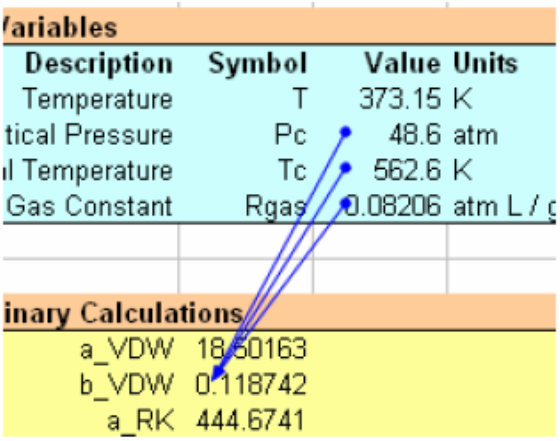
Formula Auditing

An important capability is to be able to visualize the “flow” of information on your worksheet, especially if you are having trouble remembering what a specific formula represents.

Make sure you have made the **Formula Auditing** toolbar visible. View/Toolbars.

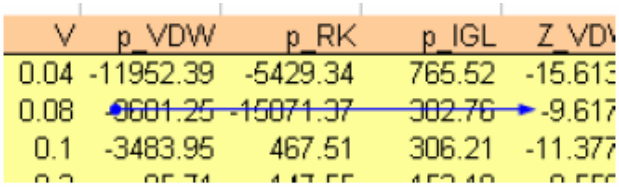


Click on a formula in your worksheet. To learn where the data used in this cell is located press the “trace precedents” (the second button).



Pressing it again will show where those cells get their values from, etc. Press the "remove all arrows" button to remove all arrows.

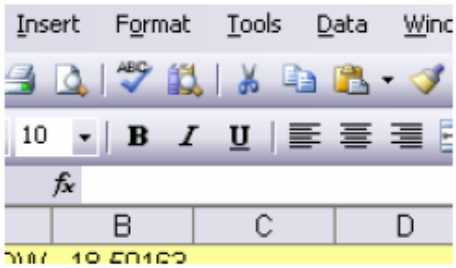
To learn where the data (formula) in a cell is used, press the "trace dependents" (the fourth button).

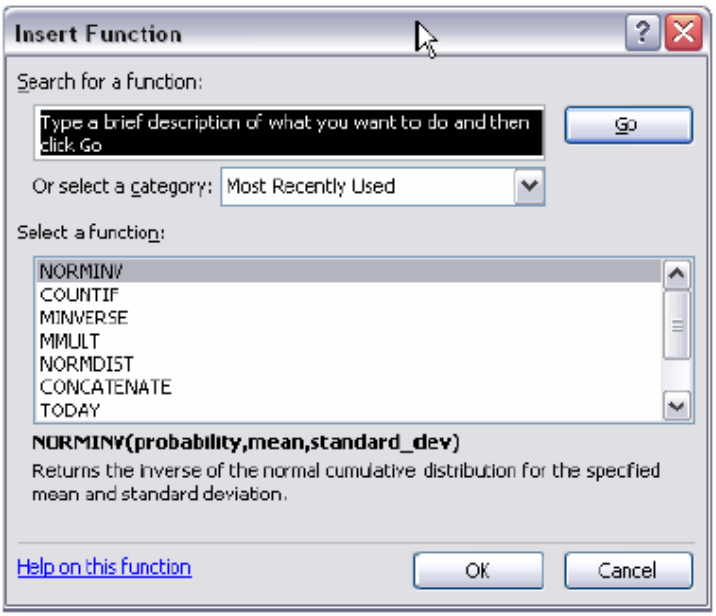


Pressing it again will show where those values are used, etc.

Built-In Functions: You are probably aware that Excel provides many "built-in" functions for a variety of operations. For example: =AVERAGE(B5:B9). Again, using the mouse to indicate (by dragging) the range to be considered is the easiest way to complete the formula. You DO need to type the closing parenthesis before pressing "Enter". We will review the very extensive list of Excel functions in the future.

There is a convenient way to "jog" one's memory if you forget the name or spelling of a function name. Just click on the "Formula Bar" f(x) icon.



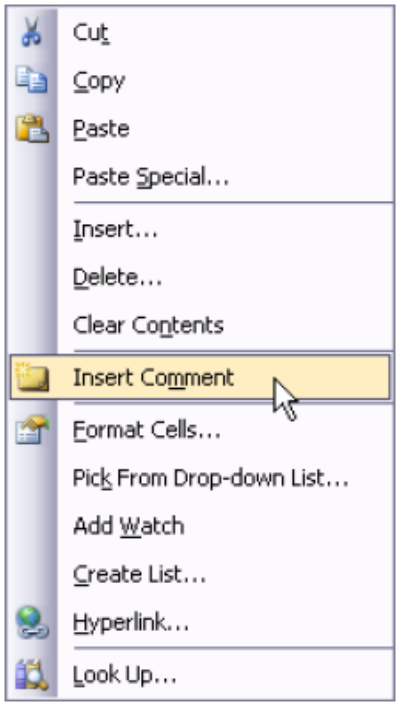


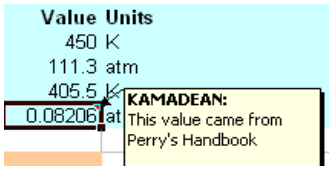
Error Messages in Excel: There are a number of very brief error messages you may see while typing or reviewing the spreadsheet. The most common messages are:

#DIV/0 Divide by 0

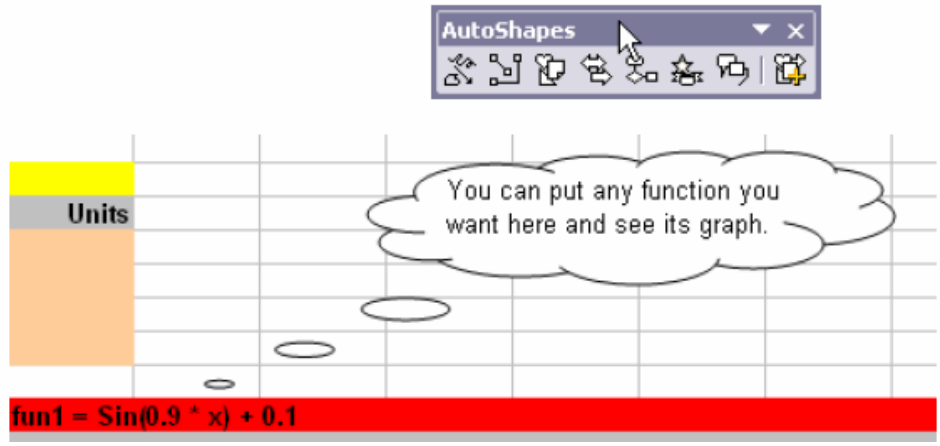
#N/A	Result “not available”. Some error has caused Excel to have no value to place in this cell.
#Name?	Unable to recognize the name used (did you forget an “_” character??) Check “Insert/Name/Define” to see what names are “in use” in the spreadsheet.
#NUM!	Unable to display the number (usually caused by a math function returning a value too large or too small to display).
#REF!	Invalid cell reference (pointing outside the allowed range)
#VALUE!	Wrong type of argument for function, etc. SIN(“happy”) or SIN(B3) where B3 contains “Assignment 1”

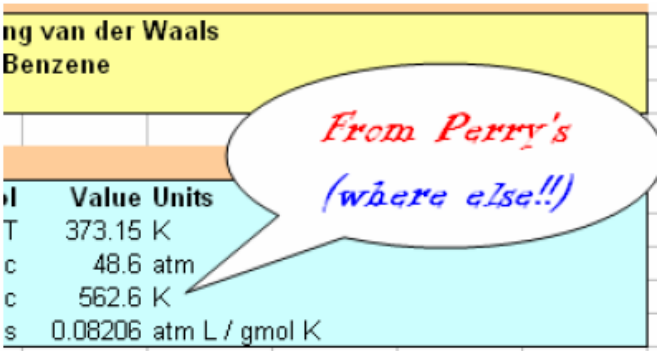
Cell Comments: It is frequently desirable to draw attention to some item or result on a worksheet. This can be done using cell comments. Just right click on the cell and select “Insert Comment”. Cells with comments have a “triangle” in the corner and when you hover over the cell the comment shows. You can also set the comment to be visible and locate it where you desire on the worksheet.





AutoShapes (Callouts): A more interesting and versatile way of documenting your worksheet is to use “callouts” or other “autoshapes”. Again, you will have to make this toolbar visible.





You can choose the font, color, size, etc of items in “callouts”.

Formatting Cells: (will be discussed in more detail later)

Freezing and Splitting Panes: (left for student to read)

Organization of Information: Typically spreadsheets contain a number of standard elements including:

- Titles: Identification
- Input Values: Parameters related to the problem specification (usually scalars)
- Other Parameter Values: Values needed by the formulas but not part of the problem statement (for example, MW’s, conversion factors, etc).
- Formulas: Calculations area
- Results: Computer values (possibly redisplayed) in a convenient format

Usually these can be arranged in a “top down” fashion although other arrangements are possible.

In our course, we will employ a fixed format for the **title information.** It is required that all programs clearly show the function, value and units of all input data as well as clearly identifying the program output and answer.

Identification	
Name:	Mohd. Kamaruddin Abd. Hamid
Course:	DKK 3352
Date:	12-Feb-06
Assignment:	HW 1

Description
Determination of Z-factor using van der Waals and Redlick-Kwong EOS for Ammonia

Problem Parameters

Description	Symbol	Value	Units
Temperature	T	450	K
Critical Pressure	Pc	111.3	atm
Critical Temperature	Tc	405.5	K
Gas Constant	Rgas	0.08206	atm L / gmol K

Problem Vectors

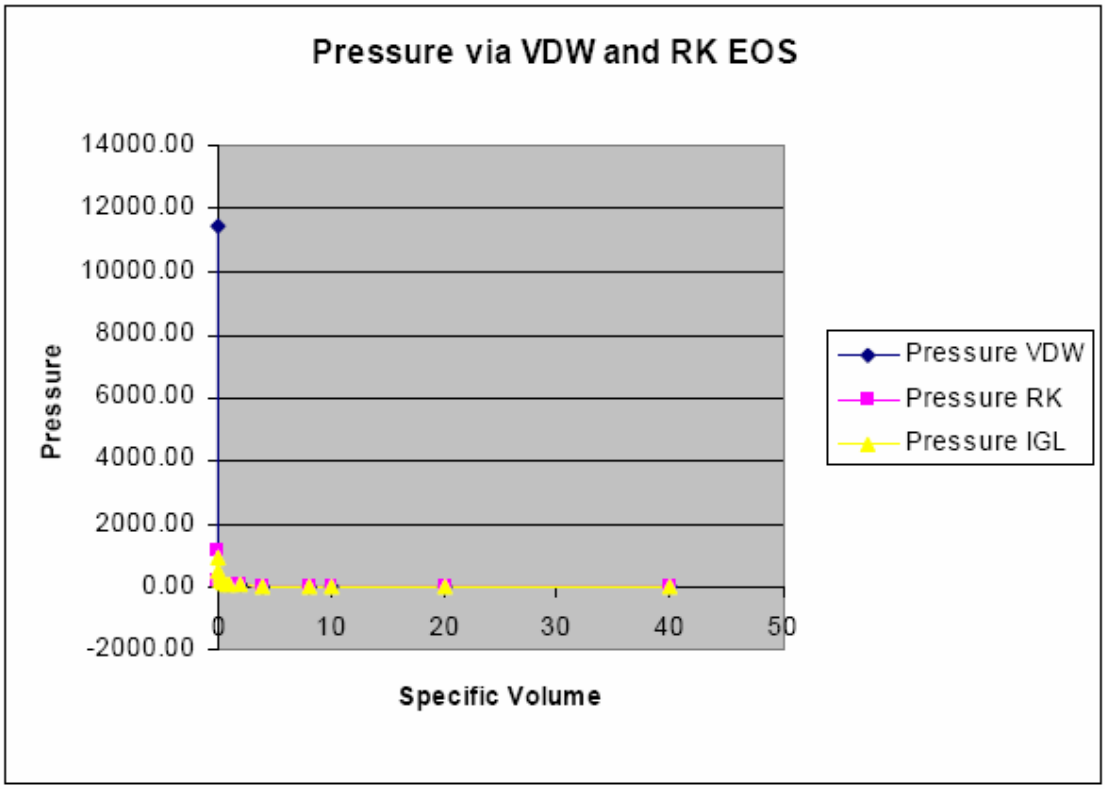
Description	Symbol	Value	Units
Specific Volume	V	0.04	L / gmol
		0.08	
		0.1	
		0.2	
		0.4	
		0.8	
		1	
		2	
		4	
		8	
		10	
		20	
		40	
		80	

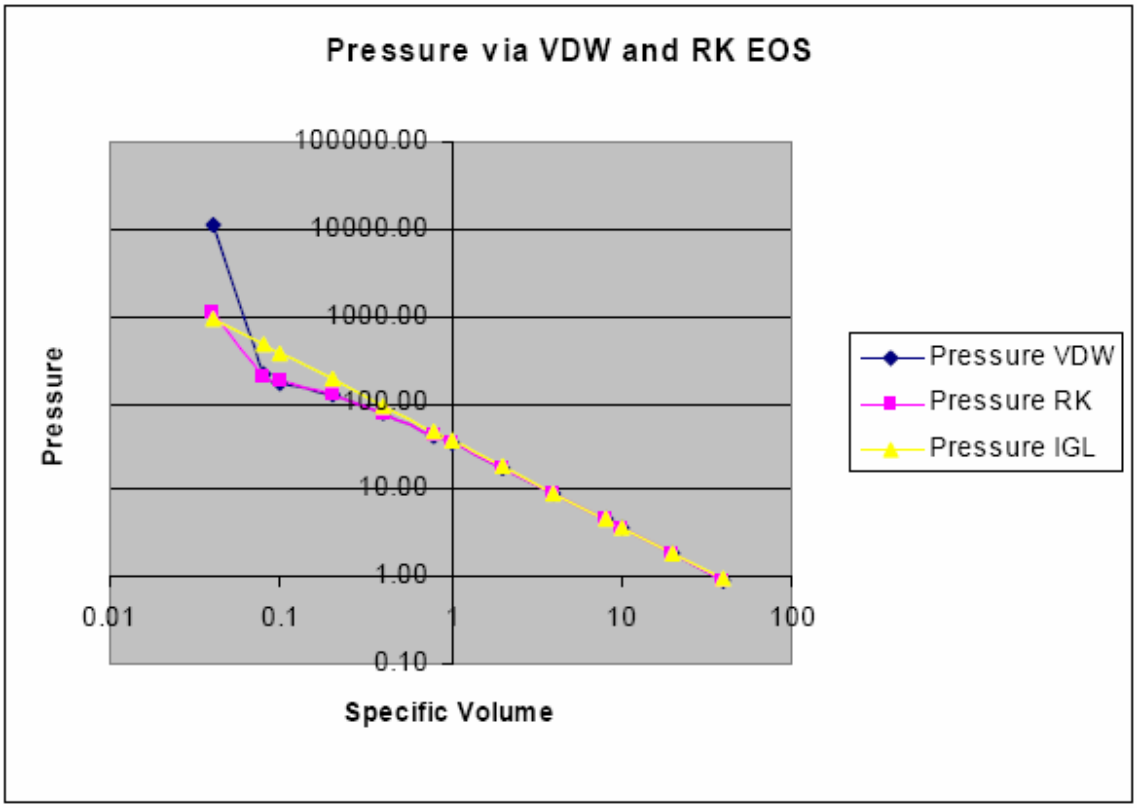
Additional Problem Data

a_VDW	4.196946
b_VDW	0.037371
a_RK	85.63687
b_RK	0.025903

Calculations	V	p_VDW	p_RK	p_IGL	Z_VDW	Z_RK
	0.04	11424.08	1088.04	923.18	12.3748	1.1786
	0.08	210.47	206.11	461.59	0.4560	0.4465
	0.1	169.92	177.72	369.27	0.4602	0.4813
	0.2	122.14	122.75	184.64	0.6615	0.6648
	0.4	75.60	75.01	92.32	0.8189	0.8126

0.8	41.86	41.59	46.16	0.9069	0.9011
1	34.16	33.97	36.93	0.9252	0.9200
2	17.77	17.71	18.46	0.9622	0.9592
4	9.06	9.04	9.23	0.9810	0.9794
8	4.57	4.57	4.62	0.9905	0.9896
10	3.66	3.66	3.69	0.9924	0.9917
20	1.84	1.84	1.85	0.9962	0.9958
40	0.92	0.92	0.92	0.9981	0.9979
80	0.46	0.46	0.46	0.9990	0.9990

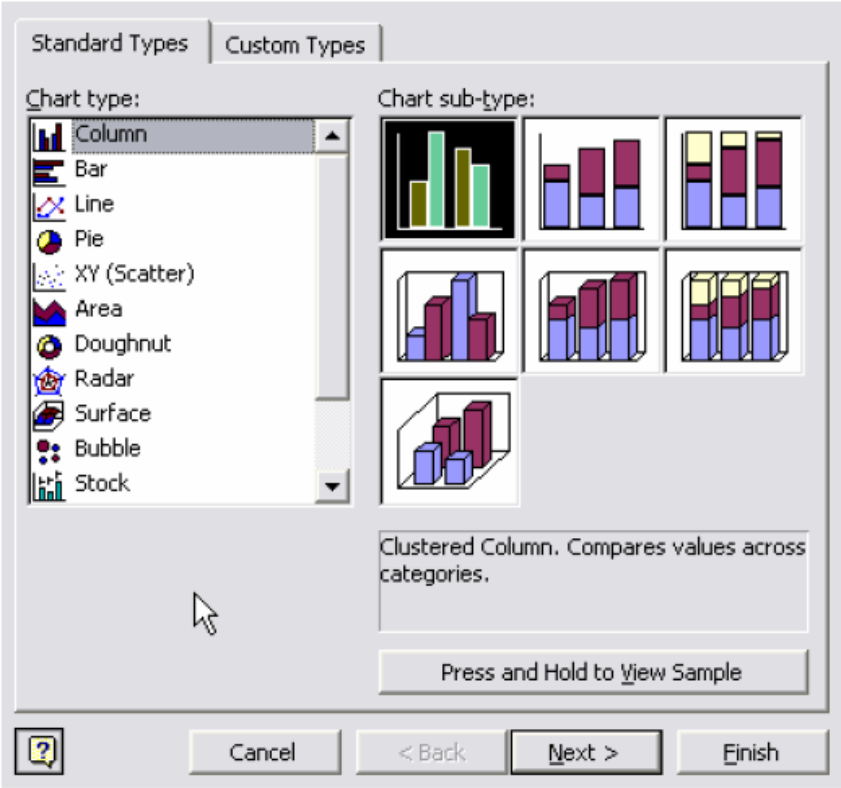



Pressure via VDW and RK EOS
100000.00
10000.00
1000.00
100.00
10.00
1.00
0.10
0.01
0.1
1
10
100
Pressure
Specific Volume
Pressure VDW
Pressure RK
Pressure IGL

Graphing with Excel

Introduction

Types of Charts: Excel offers the user some 300 different chart formats including:



The most useful and frequently employed for engineers is “XY (Scatter) Charts”. This type is sometimes confused with a similar appearing type which is generally NOT used called “Line Charts”.

XY (Scatter) Charts allow for the independent specification of x and y values (hence, no functionality is assumed) and data can be thought of as the “hits on a dart board.”

In the case of **Line Charts** the data values are associated with their linear position in the row or column of data they are in. Hence if you try to plot col x vs col y using a Line Chart you get two series on one plot, “y” vs the column number and “x” vs the column number. Usually Line Charts are used when the data being plotted is available for a “textual” index (such as the days of the week or for different chemical species).

Charts can be either “**embedded**” (existing as a component of the sheet itself) or as a **separate** component (a chartsheet). In this course we will generally keep charts on the sheet they derive their data from. This makes plotting the whole assignment easier.

Terminology that is employed by Excel to become familiar with includes:

- chart area
- plot area
- chart title
- x and y axis titles
- data series (a “line” on the chart)
- legend
- gridline
- markers

- smoothing options

As an example, prepare 101 data values in a spreadsheet. Use names to establish xmin, xmax=2*PI(), npts, and dx=(xmax-xmin)/(npts-1)

xmin	0.0000
xmax	6.2832
npts	101.0000
dx	0.0628

IMPORTANT NOTE: From this point on in the course, the designation [n] means to supply the specific cell address for the item in square brackets. This will be of the form ColRow such as A13 or B4. Since everyone will be free to set up worksheets individually you will need to supply the appropriate address for your situation.

Generate a table for the values of "n", "x", "y1", "y2":

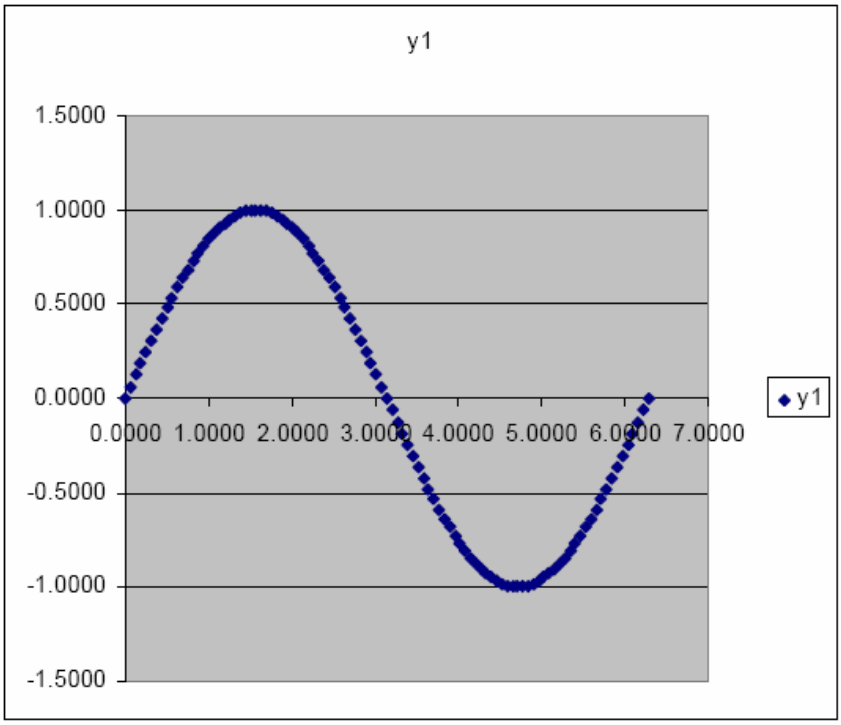
- n (generate a series from 0..1..100 by entering 0 and 1 and dragging)
- Now working with the top row (n=0) enter the following:
- [x] = xmin + [n]*dx
- [y1] = sin([x])
- [y2] = cos([2x])
- Now copy the "x", "y1" and "y2" formulas to the rest of the rows.

Note in the table below, the rows associated with n=5,6,…,95,96 have been hidden to conserve space. You may wish to do the same after generating the values.

n	x	y1	y2
0	0.0000	0.0000	1.0000
1	0.0628	0.0628	0.9921
2	0.1257	0.1253	0.9686
3	0.1885	0.1874	0.9298
4	0.2513	0.2487	0.8763
97	6.0947	-0.1874	0.9298
98	6.1575	-0.1253	0.9686
99	6.2204	-0.0628	0.9921
100	6.2832	0.0000	1.0000

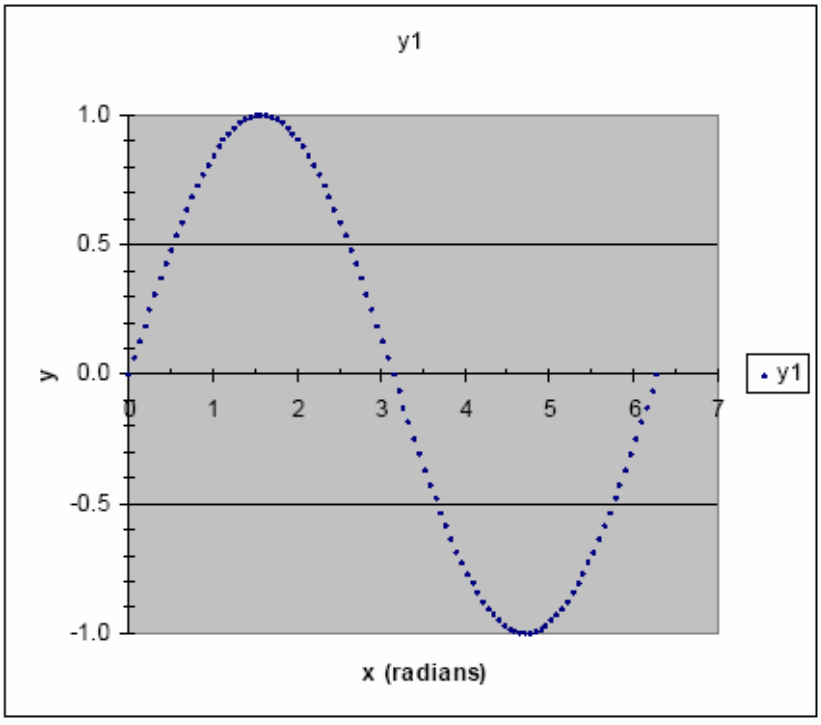
Note: This is not good "programming practice" since we have two different places we are referencing "101" points that are not "synched". That is, if we change the number npts=101 to npts=51 we still have 101 points in the table. In future assignments we will find ways to improve this.

- Insert/Chart/XY(Scatter)
- Drag to Highlight the data in columns x and y1 INCLUDING the headings "x" and "y1".
- Finish the chart



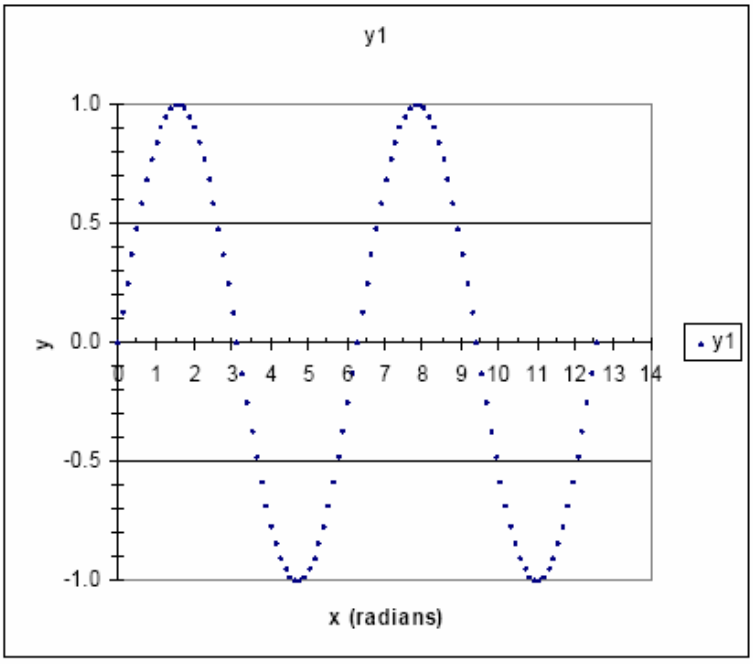
Modify the following elements of the graph:

- change symbol size to “2”
- uncheck “auto scale”
- y-axis: min = -1.0, max = 1.0, decimal places = 1
- x-axis: min = 0.0, max = auto, major = 1.0, minor = 0.50, decimal places = 0
- minor tic marks cross
- yaxis title “y”
- xaxis title “x (radians)”

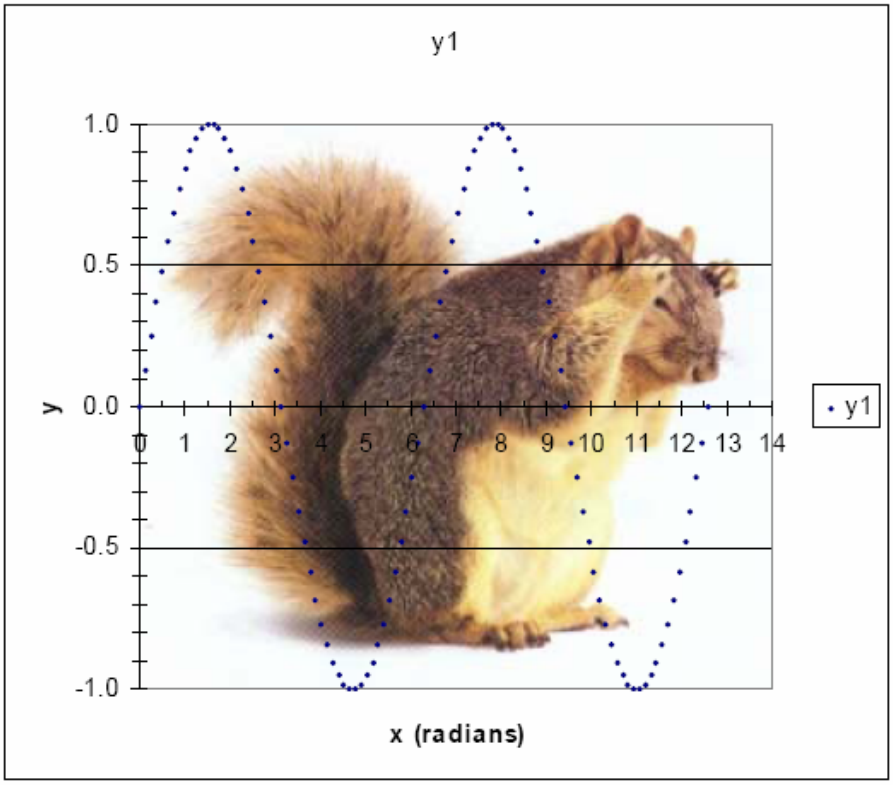


Now, **change the value of xmax** to =4*pi() to demonstrate the chart is “dynamically” linked to the data ranges.

Also, set plot “area color” to “**none**”. This reproduces much better in reports than the default gray and should be considered “standard” for this course unless otherwise indicated.

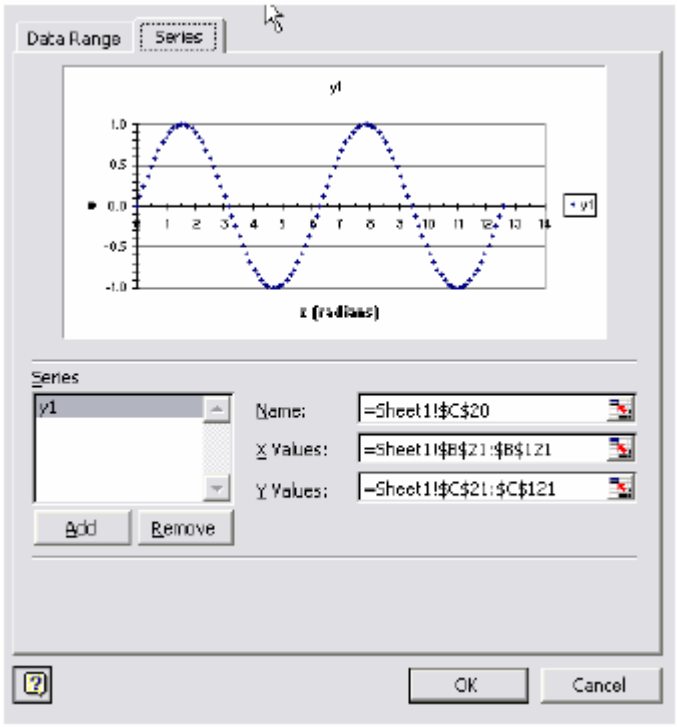


You can also **supply other textures** or even a “graphic” or background picture in place of the now clear plot area” using “Fill Effects/Picture/Select Picture”.

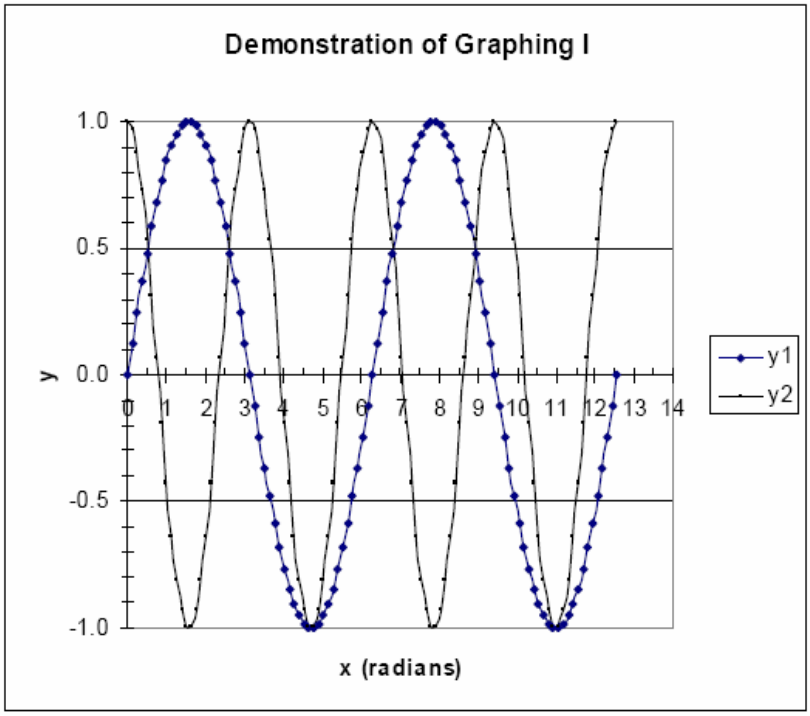


Adding additional sets of data to the chart:

- Click near the edge of the chart (handles appear)
- Right click and select **Source Data**



- Click “add” and add the second series of data x vx y2
- Add graph title in “chart options”
- Select “reasonable” colors for lines and symbols (blue/black/etc).
- Select solid lines with “smoothing” to show shape of curves (since we are graphing smooth trig functions this is appropriate).

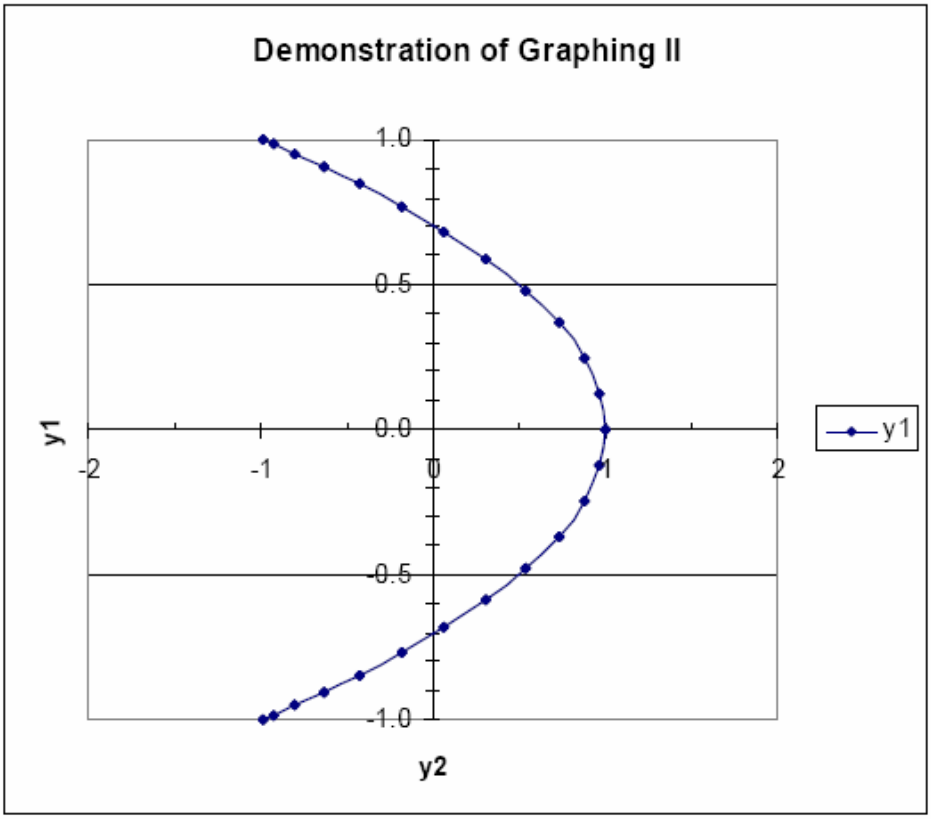


Parametric Plotting: In a parametric plot we draw functions on the x-axis and y-axis which are both functions of a third parameter. For example: y1=f(x), y2=f(x). Plot y1 on the y-axis and y2 on the x-axis.

Rather than starting from scratch, copy the current graph and paste and modify the copy.

Delete the old second series (y2). Change the first series x-axis to the y2 range.

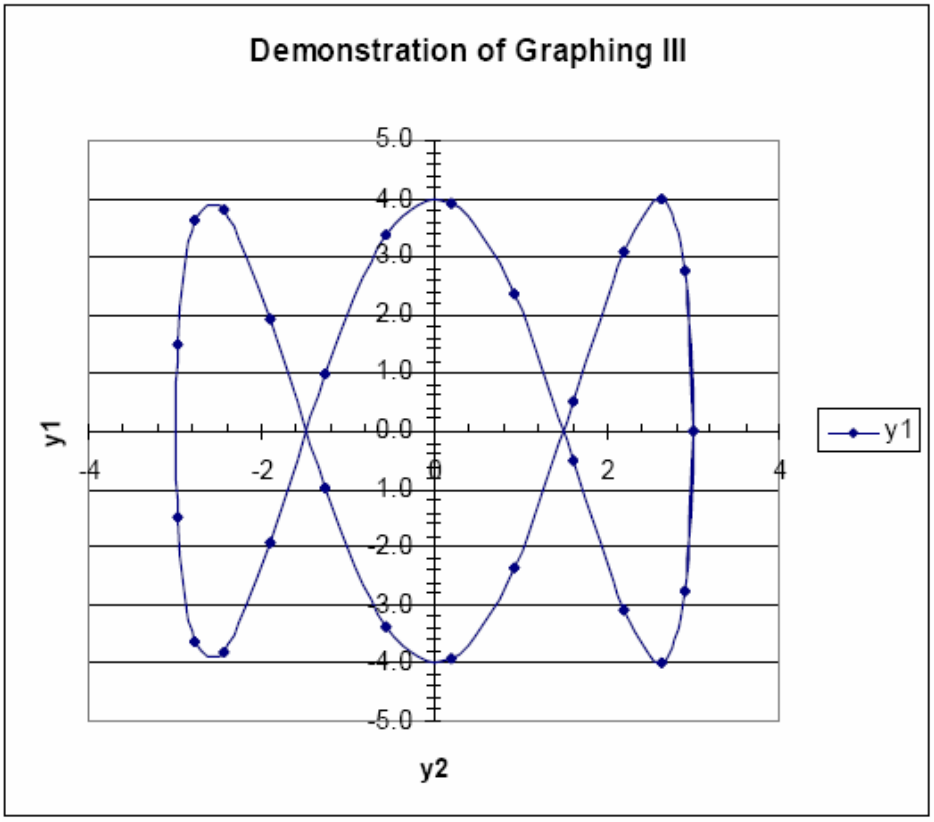
Note that this is no longer the graph of a mathematical function (that is, there is not a single value y=f(x) but rather multiple-values).



Modify the y1 and y2 function definitions to allow new parameters aa,bb,cc,dd

- y1=aa*sin(bb*x)
- y2=cc*cos(dd*x)

Make new named variable for aa, bb, cc, and dd and set them equal to the following: aa=4, bb=6, cc=3, dd=2. You should change the scaling for both axes to be “auto” (also major and minor tic mark values).

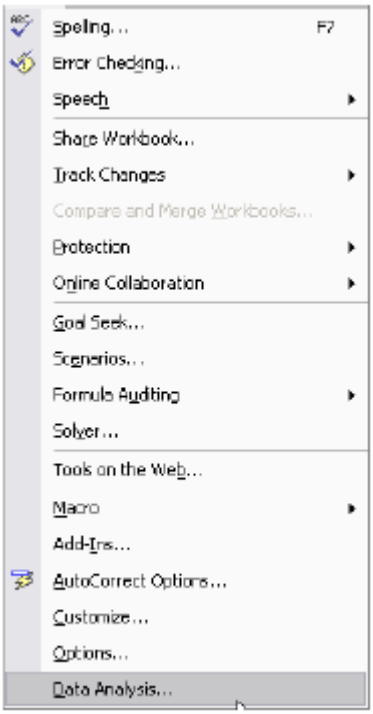


Plotting random values: Although we will consider random numbers again when we study **statistical functions,** it is interesting to see how graphing these functions help us understand their function. For example: suppose we wish to simulate someone playing "darts" who is able to (on average) aim for the bull's-eye (which is 60 inches off the ground) but they only get 95% of their darts within 12 inches of the center (that is, 24 inches of spread around the centerline). In statistical terms, this implies the standard deviation is 6 inches (since ±2 standard deviation contains about 95% of the values).

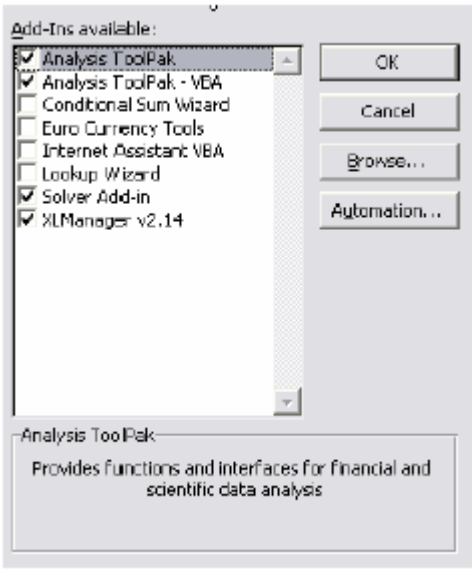
Set up a simulation showing someone throwing a dart 200 times and plot the results. Consider x (height) and y (left-right) to be independent.

We will use one of the Data Analysis Toolbox Functions (Random Number Generator) to generate the necessary data.

If the Analysis ToolPak has been installed and selected (made available) you will see "Data Analysis" as a selectable item in the "Tools" menu.



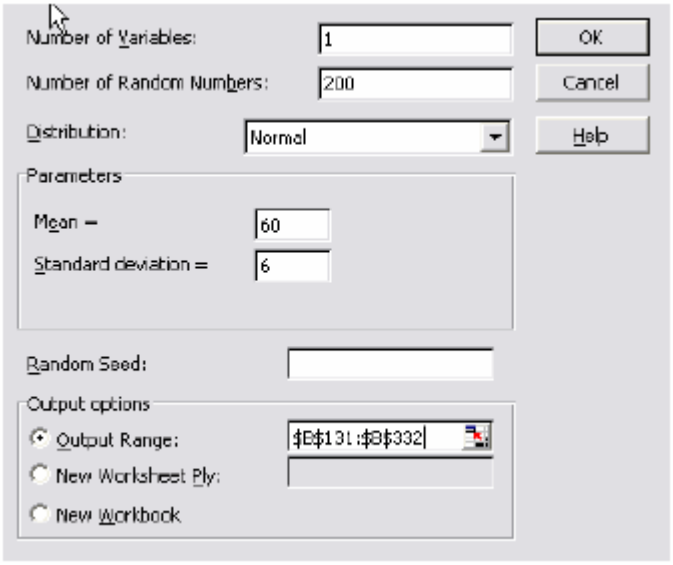
If you do not see “Data Analysis” you must select it from the list of add-ins installed for Excel. Simply check the box “Analysis ToolPak”. If you do not see this item, you will need to first install the Add-in from your Excel disks.



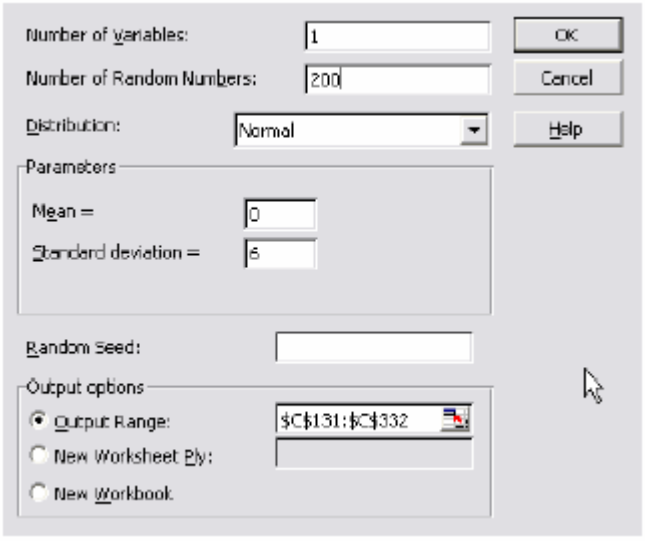
Prepare the spreadsheet to receive the data.

- Set up a column containing the values of n (1..200)
- Label columns xpos and ypos
- Choose “Data Analysis” from the “Tools” menu
- Select Random Number Generator
- Enter the appropriate data for “xpos”
    - Number of Random Numbers: 200
    - Mean = 60
    - Standard Deviation = 6

- Designate the location to receive the values generated by selecting “Output Range”



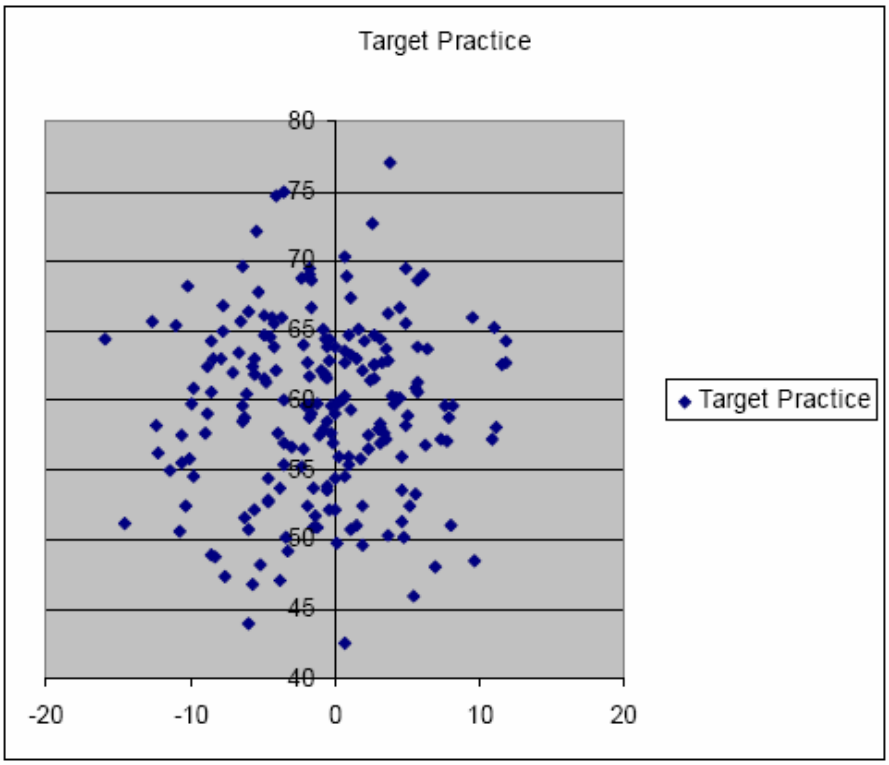
- Repeat to generating the appropriate data for “ypos”
  - Number of Random Numbers: 200
  - Mean = 0
  - Standard Deviation = 6



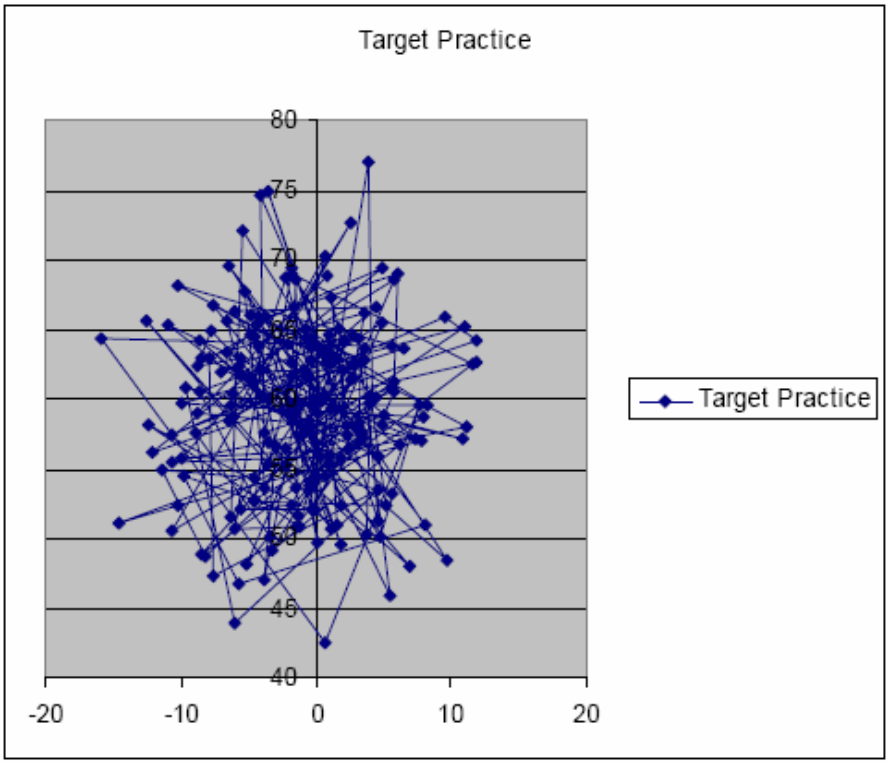
Your output should be similar to that shown below (only the first few columns are printed).

n	xpos	ypos
0	62.6877	11.7855
1	59.5560	-1.9741
2	50.1988	4.7057
3	59.5192	7.6165

Prepare a scatter plot using the “default options” for your xpos and ypos data. Your output should look approximately like below (change chart parameters as necessary). Be sure to plot the correct data on the right axis.



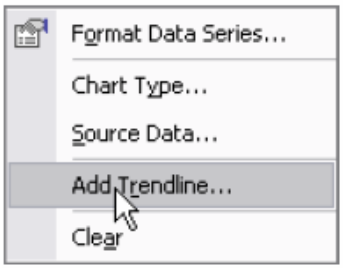
Note: We do NOT want to drawn "lines" connecting points in this case because there is no functional relationship between one dart toss and the next!



Trendlines: Trendlines represent a "curve fit through your data" using one of several methods. Knowing when to use a "trendline" rather than "connecting points" depends on the use of the chart. Generally when you wish to represent data that has experimental

error or other noise associated with the data you need a trendline. Also, if your output should look approximately like below (change chart parameters as necessary). Be sure to plot the correct data on the right axis.

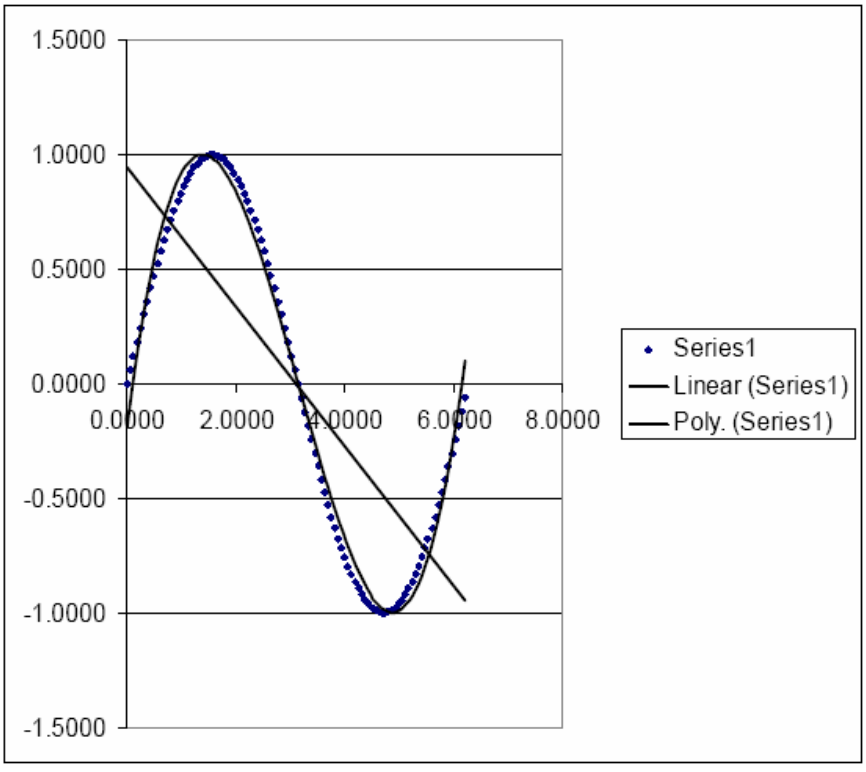
Trendline is selected by right clicking on any data point associated with a series of data. The choices are linear (least squares), logarithmic, polynomial, power (function), exponential, and moving average.



Generate data (n=51) for [y]=sin([x]) for 0<x<pi. Use xincr=pi()/51 and xvalue=xstart+[pt]*xincr

xincr	0.0616	
xstart	0.0000	
pt	xvalue	yvalue
0	0.0000	0.0000
1	0.0616	0.0616
2	0.1232	0.1229
3	0.1848	0.1837
4	0.2464	0.2439

Add two trendlines… one for “linear” and one for “polynomial (order 3)”



Error Bars: Left for student to read.

Surface Plots: Surface plots represent the functional behavior for functions of two variables (such as P=f(T,V)) or $z = x^2 + y^2$

Prepare a plot of the function $z = x^2 + y^2$ for $1<x<4$ by 0.1 and $2<y<6$ by 0.2

- Generate values for x (enter the values 1 and 1.1 and drag down to 4.0)
- Generate values for y (enter the values of 2 and 2.2 and drag across to 6.0)
- Enter the correct formula for the first (corner) cell
    - o =A2^2+B1^2 (for my spreadsheet)

	A	B	C	D	E	F	G
1		2	2.2	2.4	2.6	2.8	3
2	1	5					
3	1.1						
4	1.2						
5	1.3						
6	1.4						
7	1.5						

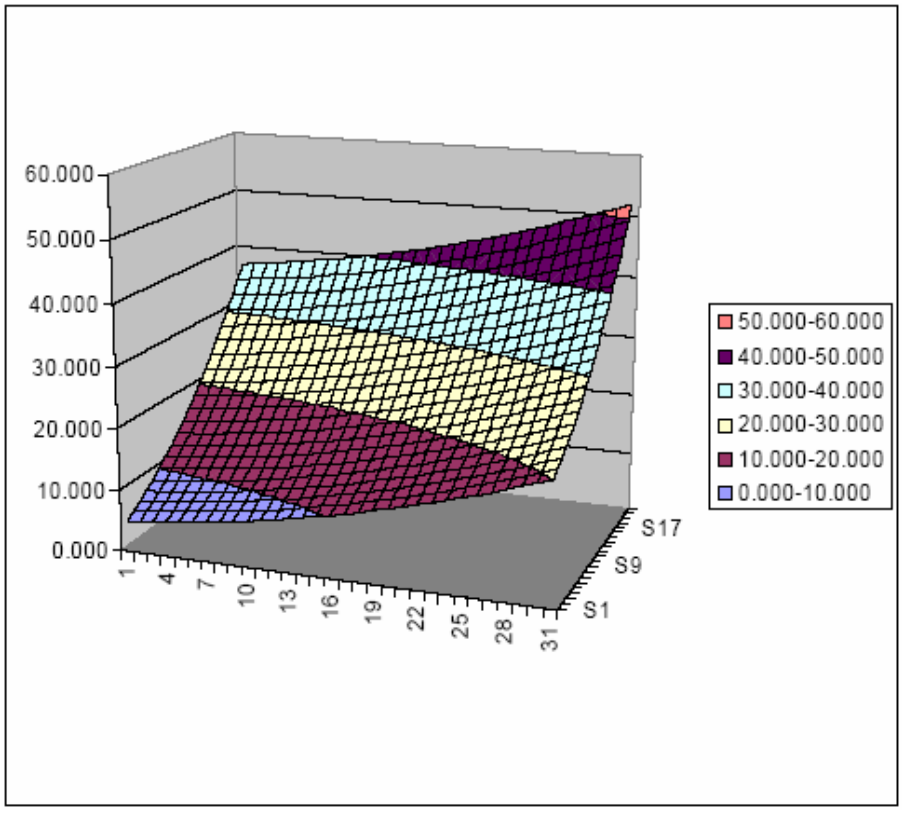
- Copy the corner cell formula to the whole range. Note the result.

	A	B	C	D	E	F
1		2	2.2	2.4	2.6	2.8
2	1	5	29.84	896.1856	803155.4	6.45E+11
3	1.1	26.21	1577.39	3291307	1.15E+13	1.32E+26
4	1.2	688.4041	2962058	1.96E+13	5.16E+26	2.84E+53
5	1.3	473901.9	9E+12	4.65E+26	4.83E+53	3.1E+107
6	1.4	2.25E+11	8.1E+25	2.23E+53	2.8E+107	1.8E+215
7	1.5	5.04E+22	6.56E+51	5E+106	8.3E+214	#NUM!
8	1.6	2.54E+45	4.3E+103	2.5E+213	#NUM!	#NUM!
9	1.7	6.47E+90	1.9E+207	#NUM!	#NUM!	#NUM!
10	1.8	4.2E+181	#NUM!	#NUM!	#NUM!	#NUM!
11	1.9	#NUM!	#NUM!	#NUM!	#NUM!	#NUM!
12	2	#NUM!	#NUM!	#NUM!	#NUM!	#NUM!
13	2.1	#NUM!	#NUM!	#NUM!	#NUM!	#NUM!
14	2.2	#NUM!	#NUM!	#NUM!	#NUM!	#NUM!
15	2.3	#NUM!	#NUM!	#NUM!	#NUM!	#NUM!

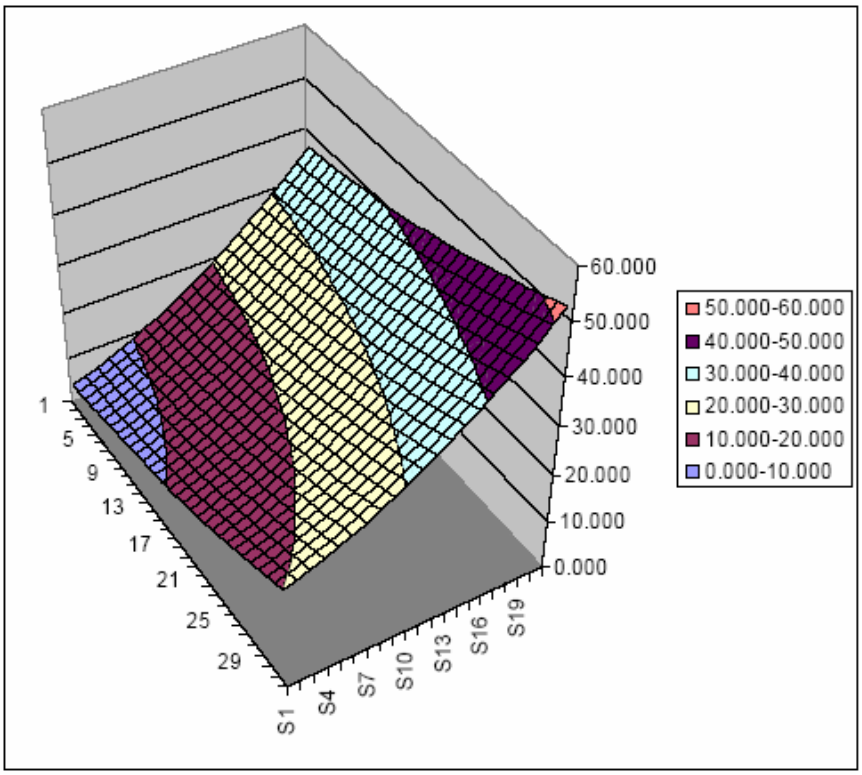
The problem is that we have not used the right “absolute addresses” for rows and columns. Determine the correct address for the corner cell and paste in the spreadsheet again.

	A	B	C	D	E	F
1		2	2.2	2.4	2.6	2.8
2	1	5	5.84	6.76	7.76	8.84
3	1.1	5.21	6.05	6.97	7.97	9.05
4	1.2	5.44	6.28	7.2	8.2	9.28
5	1.3	5.69	6.53	7.45	8.45	9.53
6	1.4	5.96	6.8	7.72	8.72	9.8
7	1.5	6.25	7.09	8.01	9.01	10.09
8	1.6	6.56	7.4	8.32	9.32	10.4
9	1.7	6.89	7.73	8.65	9.65	10.73
10	1.8	7.24	8.08	9	10	11.08
11	1.9	7.61	8.45	9.37	10.37	11.45
12	2	8	8.84	9.76	10.76	11.84
13	2.1	8.41	9.25	10.17	11.17	12.25

Insert a new chart of type “surface chart”. Highlight the entire data area INCLUDING the HEADINGS before starting the chart so the “axes” will be properly notated.



Try grabbing the corner “handles” and changing the viewpoint of this chart!


60.000
50.000
40.000
30.000
20.000
10.000
0.000
50.000-60.000
40.000-50.000
30.000-40.000
20.000-30.000
10.000-20.000
0.000-10.000
1
5
9
13
17
21
25
29
S1
S4
S7
S10
S13
S16
S19

Excel Functions

Introduction to Excel Functions

Functions are referred to by a unique (descriptive) name and an argument list (in parentheses). If used in a cell as a formula, one would preface the function with "+"

- PI()
- SIN(x)
- MAX(2,4,6)
- SUM(B1:B3, moredata, D2:D14)

Excel's ability to specify individual cells or cell ranges by (1) use of the mouse or by (2) using named ranges makes filling in argument lists easier.

Hints And Tips	**Display or hide formulas** Switch between displaying formulas and their values on a worksheet • Press CTRL + ` (grave accent).

Excel's Built-in Functions

Many of Excel's functions are available without any special reference however some of the more sophisticated functions are only available once the "Analysis ToolPak" is added in.

In addition to the "built-in" functions, users can write their own functions to simplify the construction of spreadsheets. In fact, this ability is one of the first uses we will put VBA to. For example, we might choose to express the PVT behavior of gases using the Redlich-Kwong equation of state.

Functions listed by category:

1. Math and Trigonometry

Math

ABS Returns the absolute value of a number
CEILING Rounds a number to the nearest integer or to the nearest multiple of significance
DEGREES Converts radians to degrees
EVEN Rounds a number up to the nearest even integer
EXP Returns e raised to the power of a given number
FACT Returns the factorial of a number
FLOOR Rounds a number down, toward zero
INT Rounds a number down to the nearest integer
LN Returns the natural logarithm of a number
LOG Returns the logarithm of a number to a specified base
LOG10 Returns the base-10 logarithm of a number
MOD Returns the remainder from division
MROUND Returns a number rounded to the desired multiple
ODD Rounds a number up to the nearest odd integer
PI Returns the value of pi
POWER Returns the result of a number raised to a power

PRODUCT Multiplies its arguments
QUOTIENT Returns the integer portion of a division
RADIANS Converts degrees to radians
RAND Returns a random number between 0 and 1
RANDBETWEEN Returns a random number between the numbers you specify
ROUND Rounds a number to a specified number of digits
ROUNDDOWN Rounds a number down, toward zero
ROUNDUP Rounds a number up, away from zero
SIGN Returns the sign of a number
SQRT Returns squareroot of a number
SUM Adds the arguments
SUMIF Adds the cells specified by a given criteria
COUNTIF Counts the cells specified by a given criteria

Examples of Math

=ABS(-3.2)	3.2
=CEILING(3.2,1)	4
=DEGREES(PI()/4)	45
=EVEN(5.2)	6
=EXP(2)	7.389056
=FLOOR(4.3,1)	4
=INT(3.4)	3
=LN(10)	2.302585
=LOG(10,EXP(1))	2.302585
=LOG10(10)	1
=MOD(5,2)	1
=MROUND(10.1112,0.1)	10.1
=ODD(3.4)	5
=PI()	3.141593
=POWER(10,2)	100
=PRODUCT(2,3,4)	24
=QUOTIENT(5,2)	2
=RADIANS(45)	0.785398
=RAND()	0.042837
=RANDBETWEEN(2,99)	27
=ROUND(5.3333,2)	5.33
=ROUNDDOWN(10.332,2)	10.33
=ROUNDUP(10.332,2)	10.34
=SIGN(-3.4)	-1
=SUMIF(A1:A5,">160000",B1:B5)	Sum the values in column B if the corresponding values in column A are greater than 160000
=COUNTIF(A1:A5,"apples")	Count the number of cells with “apples” in the column A
=COUNTIF(B1:B5,">55")	Count the number of cells with a value greater than 55 in column A

Trig

SIN Returns the sine of the given angle
COS Returns the cosine of a number
TAN Returns the tangent of a number
ASIN Returns the arcsine of a number
ACOS Returns the arccosine of a number
ATAN Returns the arctangent of a number
ATAN2 Returns the arctangent from x- and y-coordinates

SINH Returns the hyperbolic sine of a number
COSH Returns the hyperbolic cosine of a number
TANH Returns the hyperbolic tangent of a number
ACOSH Returns the inverse hyperbolic cosine of a number
ASINH Returns the inverse hyperbolic sine of a number
ATANH Returns the inverse hyperbolic tangent of a number

Matrix Functions

MDETERM Returns the matrix determinant of an array
MINVERSE Returns the matrix inverse of an array
MMULT Returns the matrix product of two arrays

Other Math and Trig

COMBIN Returns the number of combinations for a given number of objects
LCM Returns the least common multiple
GCD Returns the greatest common divisor
FACTDOUBLE Returns the double factorial of a number
SERIESSUM Returns the sum of a power series based on the formula
ROMAN Converts an arabic numeral to roman, as text
MULTINOMIAL Returns the multinomial of a set of numbers

2. Logical

IF Specifies a logical test to perform
OR Returns TRUE if any argument is TRUE
AND Returns TRUE if all its arguments are TRUE
NOT Reverses the logic of its argument
TRUE Returns the logical value TRUE
FALSE Returns the logical value FALSE

Examples of Logicals

Logical Operators = > >= < <= <>
False = 0 True <> 0

=IF(0,"0 is true", "0 is false")	0 is false
=IF(1,"1 is true", "1 is false")	1 is true
=IF(0.5,"0.5 is true", "0.5 is false")	0.5 is true
=IF(-1,"-1 is true", "-1 is false")	-1 is true
=IF(1>2,"blue","orange")	orange
=IF(C1>C2,34,53)	53
=AND(TRUE,TRUE,FALSE)	FALSE
=AND(2>1,4>3,-5>-7)	TRUE
=NOT(4<3)	TRUE
=OR(4<3,5>6)	FALSE

3. Date and Time

DATE Returns the serial number of a particular date
DATEVALUE Converts a date in the form of text to a serial number
DAY Converts a serial number to a day of the month
DAYS360 Calculates the number of days between two dates based on a 360-day year

EDATE Returns the serial number of the date that is the indicated number of months before or after the start date
EOMONTH Returns the serial number of the last day of the month before or after a specified number of months
HOUR Converts a serial number to an hour
MINUTE Converts a serial number to a minute
MONTH Converts a serial number to a month
NETWORKDAYS Returns the number of whole workdays between two dates
NOW Returns the serial number of the current date and time
SECOND Converts a serial number to a second
TIME Returns the serial number of a particular time
TIMEVALUE Converts a time in the form of text to a serial number
TODAY Returns the serial number of today's date
WEEKDAY Converts a serial number to a day of the week
WEEKNUM Converts a serial number to a number representing where the week falls numerically with a year
WORKDAY Returns the serial number of the date before or after a specified number of workdays
YEAR Converts a serial number to a year
YEARFRAC Returns the year fraction representing the number of whole days between start_date and end_date

Examples of Time and Date Functions

=TODAY()	2/12/2006
=NOW()	2/12/2006 13:33
=YEAR(TODAY())	2006
=MONTH(TODAY())	2
=MONTH(NOW())	2
=DAY(NOW())	12
=HOUR(NOW())	13
=MINUTE(NOW())	33
=SECOND(NOW())	16
=TIMEVALUE("11:20 am")	0.472222222
=DATEVALUE("07/29/2004")	38197
=TIME(5,30,0)	5:30 AM
=WEEKDAY(NOW())	5
=YEARFRAC("7/28/2004","3/30/1948")	56.32777778

4. Text and Data

ASC Changes full-width (double-byte) English letters or katakana within a character string to half-width (single-byte) characters
BAHTTEXT Converts a number to text, using the ß (baht) currency format
CHAR Returns the character specified by the code number
CLEAN Removes all nonprintable characters from text
CODE Returns a numeric code for the first character in a text string
CONCATENATE Joins several text items into one text item
DOLLAR Converts a number to text, using the $ (dollar) currency format
EXACT Checks to see if two text values are identical
FIND Finds one text value within another (case-sensitive)
FIXED Formats a number as text with a fixed number of decimals
JIS Changes half-width (single-byte) English letters or katakana within a character string to full-width (double-byte) characters
LEFT Returns the leftmost characters from a text value
LEN Returns the number of characters in a text string

LOWER Converts text to lowercase
MID Returns a specific number of characters from a text string starting at the position you specify
PROPER Capitalizes the first letter in each word of a text value
REPLACE Replaces characters within text
REPT Repeats text a given number of times
RIGHT Returns the rightmost characters from a text value
SEARCH Finds one text value within another (not case-sensitive)
SUBSTITUTE Substitutes new text for old text in a text string
T Converts its arguments to text
TEXT Formats a number and converts it to text
TRIM Removes spaces from text
UPPER Converts text to uppercase
VALUE Converts a text argument to a number

Examples of Text

=CHAR(65)	A
=CODE("Auburn")	65
=CONCATENATE("Aub","urn")	Auburn
=EXACT("Auburn","auburn")	FALSE
=FIND("burn","Auburn")	3
=FIND("i","Missississippi",6)	8
=FIXED(PI(),4)	3.1416
=FIXED(PI(),7)	3.1415927
=FIXED(1000*PI(),-2)	3,100
=FIXED(1000*PI(),-2,TRUE)	3100
=LEFT("Auburn",3)	Aub
=RIGHT("Auburn",3)	urn
=MID("Auburn",2,4)	ubur
=LEN("auburn")	6
=LOWER("AbCdEfG")	abcdefg
=UPPER("AbCdEfG")	ABCDEFG
=REPLACE("Auburn",3,LEN("freeze"),"freeze")	Aufreeze
=REPT("Aub",5)	AubAubAubAubAub
=SEARCH("bu","Auburn")	3
=SEARCH("u","Auburn",FIND("u","Auburn",1)+1)	4
=SUBSTITUTE("Auburn","u","U")	AUbUrn
=TEXT(5.13,"0.000")	5.130
=CONCATENATE("The answer is ",TEXT(PI()/2,"0.000")," approximately!")	The answer is 1.571 approximately!

5. Engineering

BESSELI Returns the modified Bessel function In(x)
BESSELJ Returns the Bessel function Jn(x)
BESSELK Returns the modified Bessel function Kn(x)
BESSELY Returns the Bessel function Yn(x)
BIN2DEC Converts a binary number to decimal
BIN2HEX Converts a binary number to hexadecimal
BIN2OCT Converts a binary number to octal
COMPLEX Converts real and imaginary coefficients into a complex number
CONVERT Converts a number from one measurement system to another
DEC2BIN Converts a decimal number to binary
DEC2HEX Converts a decimal number to hexadecimal
DEC2OCT Converts a decimal number to octal

DELTA Tests whether two values are equal
ERF Returns the error function
ERFC Returns the complementary error function
GESTEP Tests whether a number is greater than a threshold value
HEX2BIN Converts a hexadecimal number to binary
HEX2DEC Converts a hexadecimal number to decimal
HEX2OCT Converts a hexadecimal number to octal
IMABS Returns the absolute value (modulus) of a complex number
IMAGINARY Returns the imaginary coefficient of a complex number
IMARGUMENT Returns the argument theta, an angle expressed in radians
IMCONJUGATE Returns the complex conjugate of a complex number
IMCOS Returns the cosine of a complex number
IMDIV Returns the quotient of two complex numbers
IMEXP Returns the exponential of a complex number
IMLN Returns the natural logarithm of a complex number
IMLOG10 Returns the base-10 logarithm of a complex number
IMLOG2 Returns the base-2 logarithm of a complex number
IMPOWER Returns a complex number raised to an integer power
IMPRODUCT Returns the product of two complex numbers
IMREAL Returns the real coefficient of a complex number
IMSIN Returns the sine of a complex number
IMSQRT Returns the square root of a complex number
IMSUB Returns the difference between two complex numbers
IMSUM Returns the sum of complex numbers
OCT2BIN Converts an octal number to binary
OCT2DEC Converts an octal number to decimal
OCT2HEX Converts an octal number to hexadecimal

6. Information

CELL Returns information about the formatting, location, or contents of a cell
COUNTBLANK Counts the number of blank cells within a range
ERROR.TYPE Returns a number corresponding to an error type
INFO Returns information about the current operating environment
ISBLANK Returns TRUE if the value is blank
ISERR Returns TRUE if the value is any error value except #N/A
ISERROR Returns TRUE if the value is any error value
ISEVEN Returns TRUE if the number is even
ISLOGICAL Returns TRUE if the value is a logical value
ISNA Returns TRUE if the value is the #N/A error value
ISNONTEXT Returns TRUE if the value is not text
ISNUMBER Returns TRUE if the value is a number
ISODD Returns TRUE if the number is odd
ISREF Returns TRUE if the value is a reference
ISTEXT Returns TRUE if the value is text
N Returns a value converted to a number
NA Returns the error value #N/A
TYPE Returns a number indicating the data type of a value

7. Lookup and Reference

ADDRESS Returns a reference as text to a single cell in a worksheet
AREAS Returns the number of areas in a reference
CHOOSE Chooses a value from a list of values
COLUMN Returns the column number of a reference

COLUMNS Returns the number of columns in a reference
HLOOKUP Looks in the top row of an array and returns the value of the indicated cell
HYPERLINK Creates a shortcut or jump that opens a document stored on a network server, an intranet, or the Internet
INDEX Uses an index to choose a value from a reference or array
INDIRECT Returns a reference indicated by a text value
LOOKUP Looks up values in a vector or array
MATCH Looks up values in a reference or array
OFFSET Returns a reference offset from a given reference
ROW Returns the row number of a reference
ROWS Returns the number of rows in a reference
RTD Retrieves real-time data from a program that supports COM automation
TRANSPOSE Returns the transpose of an array
VLOOKUP Looks in the first column of an array and moves across the row to return the value of a cell

8. Statistical

Common Statistical

AVERAGE Returns the average of its arguments
MAX Returns the maximum value in a list of arguments
RANK Returns the rank of a number in a list of numbers
MEDIAN Returns the median of the given numbers
MIN Returns the minimum value in a list of arguments
LARGE Returns the k-th largest value in a data set
SMALL Returns the k-th smallest value in a data set

Other Statistical

AVEDEV Returns the average of the absolute deviations of data points from their mean
AVERAGEA Returns the average of its arguments, including numbers, text, and logical values
BETADIST Returns the cumulative beta probability density function
BETAINV Returns the inverse of the cumulative beta probability density function
BINOMDIST Returns the individual term binomial distribution probability
CHIDIST Returns the one-tailed probability of the chi-squared distribution
CHIINV Returns the inverse of the one-tailed probability of the chi-squared distribution
CHITEST Returns the test for independence
CONFIDENCE Returns the confidence interval for a population mean
CORREL Returns the correlation coefficient between two data sets
COUNT Counts how many numbers are in the list of arguments
COUNTA Counts how many values are in the list of arguments
COVAR Returns covariance, the average of the products of paired deviations
CRITBINOM Returns the smallest value for which the cumulative binomial distribution is less than or equal to a criterion value
DEVSQ Returns the sum of squares of deviations
EXPONDIST Returns the exponential distribution
FDIST Returns the F probability distribution
FINV Returns the inverse of the F probability distribution
FISHER Returns the Fisher transformation
FISHERINV Returns the inverse of the Fisher transformation
FORECAST Returns a value along a linear trend
FREQUENCY Returns a frequency distribution as a vertical array
FTEST Returns the result of an F-test
GAMMADIST Returns the gamma distribution

GAMMAINV Returns the inverse of the gamma cumulative distribution
GAMMALN Returns the natural logarithm of the gamma function, Γ(x)
GEOMEAN Returns the geometric mean
GROWTH Returns values along an exponential trend
HARMEAN Returns the harmonic mean
HYPGEOMDIST Returns the hypergeometric distribution
INTERCEPT Returns the intercept of the linear regression line
KURT Returns the kurtosis of a data set
LINEST Returns the parameters of a linear trend
LOGEST Returns the parameters of an exponential trend
LOGINV Returns the inverse of the lognormal distribution
LOGNORMDIST Returns the cumulative lognormal distribution
MAXA Returns the maximum value in a list of arguments, including numbers, text, and logical values
MINA Returns the smallest value in a list of arguments, including numbers, text, and logical values
MODE Returns the most common value in a data set
NEGBINOMDIST Returns the negative binomial distribution
NORMDIST Returns the normal cumulative distribution
NORMINV Returns the inverse of the normal cumulative distribution
NORMSDIST Returns the standard normal cumulative distribution
NORMSINV Returns the inverse of the standard normal cumulative distribution
PEARSON Returns the Pearson product moment correlation coefficient
PERCENTILE Returns the k-th percentile of values in a range
PERCENTRANK Returns the percentage rank of a value in a data set
PERMUT Returns the number of permutations for a given number of objects
POISSON Returns the Poisson distribution
PROB Returns the probability that values in a range are between two limits
QUARTILE Returns the quartile of a data set
RSQ Returns the square of the Pearson product moment correlation coefficient
SKEW Returns the skewness of a distribution
SLOPE Returns the slope of the linear regression line
STANDARDIZE Returns a normalized value
STDEV Estimates standard deviation based on a sample
STDEVA Estimates standard deviation based on a sample, including numbers, text, and logical values
STDEVP Calculates standard deviation based on the entire population
STDEVPA Calculates standard deviation based on the entire population, including numbers, text, and logical values
STEYX Returns the standard error of the predicted y-value for each x in the regression
TDIST Returns the Student's t-distribution
TINV Returns the inverse of the Student's t-distribution
TREND Returns values along a linear trend
TRIMMEAN Returns the mean of the interior of a data set
TTEST Returns the probability associated with a Student's t-test
VAR Estimates variance based on a sample
VARA Estimates variance based on a sample, including numbers, text, and logical values
VARP Calculates variance based on the entire population
VARPA Calculates variance based on the entire population, including numbers, text, and logical values
WEIBULL Returns the Weibull distribution
ZTEST Returns the two-tailed P-value of a z-test

Matrix Operations in Excel

Matrix Manipulations: Vectors, Matrices, and Arrays.

In this section we consider the topic of Vectors, Matrices and Arrays and their application in solving Linear Equations and other linear algebra problems.

Simultaneous linear equations occur frequently in engineering in such areas as heat conduction, molecular diffusion, fluid mechanics and in data regression. Excel's "Solver" feature will be used in a later chapter to solve more complicated linear and nonlinear systems of equations.

Generally the term **matrix** (from mathematics) and **array** (from Excel) can be used interchangeably to refer to data organized in row and column fashion. Matrices consisting of a single row or a single column are called **vectors**. Even though the functions are "named" with matrix there is no help in Excel under "matrix" only "array".

Typical Linear Equation Set and Corresponding Matrices

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2$$
$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

$$\text{Where } [A] = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad [X] = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad [B] = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Is presented in math as

$$[A][X] = [B]$$

and has the solution $[X] = [A]^{-1}[B]$

How Excel Handles Matrix Math

Matrix operations are handled in two different fashions in Excel. Addition of matrices and scalar multiplication are handled by conventional cell arithmetic (copying cell formulas) whereas advanced matrix operations such as transposition, multiplication and inversion are handled by **matrix (array) functions**.

Key to understanding the use of matrix operations is the concept of the matrix (array) formula. Such a formula uses matrix functions and returns a result that can be a matrix, a vector, or a scalar, depending on the computations involved. ***Whatever the result may be, an area on the spreadsheet of precisely the correct size must be selected before the formula is typed in (otherwise you will either lose some of the answer or get added and possibly confusing information).***

After typing such a formula, you "enter" it with three keys pressed at once: CTRL, SHIFT and ENTER. This indicates that a matrix (array) result really is desired. It also designates the entire selected range as the desired location for the answer. To modify or delete the formula, select the entire region beforehand.

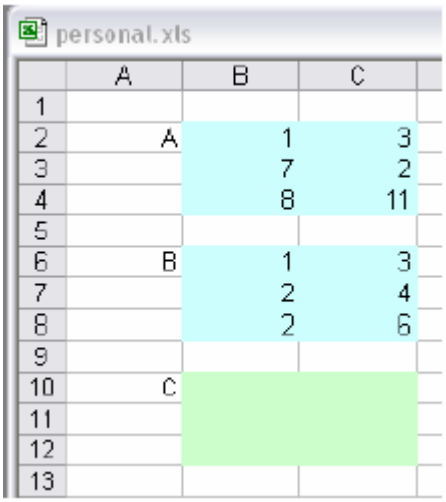
When matrix computations are performed in this way, the "result areas" will be updated immediately whenever any of the numbers in the "input areas" change (unless automatic recomputation has been turned off). This can be a great help when one wishes to evaluate the effects of changes in assumptions, initial conditions, etc.. This feature, coupled with the ability to see matrices, complete with identification of the rows and

columns (i.e. in the form that we have termed tables), will often make the spreadsheet environment the preferred choice for computation, if not for communication.

Basic Matrix Operations

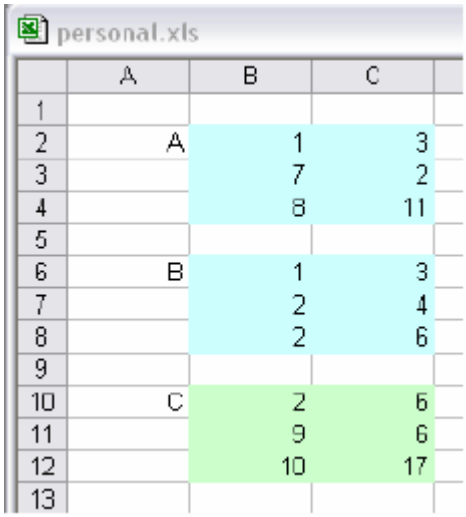
Matrix Addition: [C] = [A] + [B]

Method 1: Corresponding elements will be added using “cutting and pasting”.



	A	B	C
1			
2	A	1	3
3		7	2
4		8	11
5			
6	B	1	3
7		2	4
8		2	6
9			
10	C		
11			
12			
13			

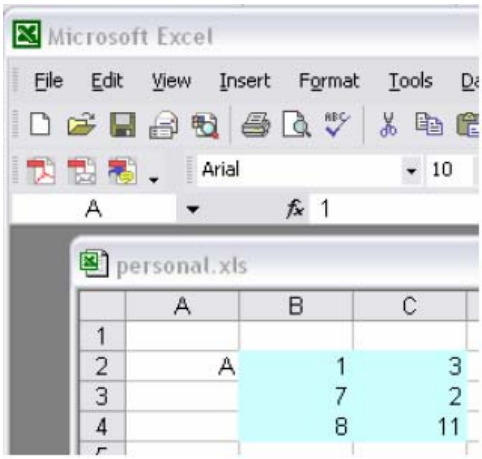
Type the formula =B2+B6 in cell B10 and copy and paste into the cells in region B10:C12.



	A	B	C
1			
2	A	1	3
3		7	2
4		8	11
5			
6	B	1	3
7		2	4
8		2	6
9			
10	C	2	6
11		9	6
12		10	17
13			

Method 2: Matrix operator (+) will be used with “named ranges”.

1. CLEAR AREA “C” FIRST. Highlight and name the cells from B2:C4 as “A”. This is done by typing “A” in the “name field” of the function picker.



2. Similarly highlight and name the range B6:C8 as “B”.
3. Highlight the **destination range** and type the following formula:

   =A+B *but do not press “enter” instead press “control-shift-enter” to complete the formula.* This will introduce { } characters around the formula {=A+B} which indicate an array operation. *YOU CANNOT SIMPLY TYPE THE BRACES.*

You should see the result in the area highlighted (in green above).

Matrix Subtraction and Scalar Multiplication

You can use either of these methods to subtract (element by element) or multiply (all elements by the same value). For example:

{=6*A} would produce a new array with all values in A multipled by 6.

Multiplying Two Matrices

Matrix multiplication requires that the two matrices are “conformable” (that is, appropriate number of rows and columns. The number of columns in the first matrix must equal the number of rows in the second matrix. That is, you can multiple A(2,5)xB(5,3) because the “inner” numbers are the same. The size of the result is governed by the “outer” numbers, in this case (2,3).

This should also suggest that AxB ≠ BxA since the result of AxB would be C(2,3) and the result of BxA is C(3,2).

To multiply two matrices, use the MMULT function.

=MMULT(first_matrix, second_matrix)

Remember you must highlight the destination matrix BEFORE completing the formula with Shift-Control-Enter!

Matrix Multiplication		
[A] 3x2	1	2
	3	4
	5	6
[B] 2x1	7	
	8	
[A][B]	23	
	53	
	83	

Transposing A Matrices

The mathematical operation of “transposing” a matrix is simply to switch the “rows” with the “columns”. Hence, a row vector’s transpose is a column vector and the transpose of a 2x3 matrix is a 3x2 matrix.

To take the transpose of a matrix, use the TRANSPOSE function.

Matrix Transpose			
[A] 3x2	1	2	
	3	4	
	5	6	
[A-Tran]	1	3	5
	2	4	6

Inverting A Matrices

The mathematical operation of “inverting” a matrix requires that two conditions are met:

1. The matrix must be “square” (same number of rows and columns)
2. The matrix must be “nonsingular”

A matrix is “singular” is any of the following are true:

1. Any row or column contains all zeros
2. Any two rows or columns are identical
3. Any row or column is a linear combination of other rows or columns.

To take the inverse of a matrix, use the MINVERSE function.

Matrix Inverse			
[A] 3x3	2	2	9
	3	4	8
	5	6	7
[A-Inv]	1	-2	1
	-0.95	1.55	-0.55
	0.1	0.1	-0.1

Determinant of A Matrix

The determinant of a matrix is a single value and is often encountered in solving systems of equations. Only square matrices have a determinant.

Since the calculation of a determinant involves a significant number of calculations, it is subject to “round-off” error. When the determinant is essentially “zero” (that is, 1x10-14 or less) it can be considered “zero”.

To find the determinant of a matrix, use the MDETERM function.

Matrix Determinant			
[A] 3x3	2	2	9
	3	4	8
	5	6	7
[A-Det]	-20		

Matrix Determinant			
[A] 3x3	2	2	9
	3	4	8
	-1	-1	-4.5
[A-Det]	0		

Note, the last line is the same as the first line multiplied by -0.5.

Solving Systems of Linear Equations

We now have the necessary tools to solve systems of linear equations. Here are the steps:

1. Write the equations in matrix form (coefficient matrix) x [unknown vector] = right hand side vector. [A][x] = [b].
2. Invert the coefficient matrix $[A]^{-1}$
3. Multiply both sides of the equation by the inverted coefficient matrix. This is the solution matrix.

In the example below, the solution “x” is

=MMULT([A]$^{-1}$,[b])

Solution of Simultaneous Linear Equations

[A]			[x]	[b]
1	1	4	0.0000	5
2	3	1	1.0000	4
3	1	2	1.0000	3

[A]-1		
-0.2083	-0.0833	0.4583
0.0417	0.4167	-0.2917
0.2917	-0.0833	-0.0417

Linear Regression in Excel

Introduction

Most students are familiar with the idea of “drawing a straight line through a set of data points” (a so-called “least squares” line). Lines of this sort (and the parameters derived from them (slope, intercept) are part of a larger statistical analysis called “regression”. In this section we limit our discussion to “linear regression” which refers to the “best fit” straight line drawn through data points.

Linear Regression Using Excel Function

There are three main Excel functions used in linear regression:

- =SLOPE()
- =INTERCEPT()
- =RSQ() “R-squared” value (that is, coefficient of determination)

Let’s consider a case where we “control” the scatter of data that is synthetically generated:

Start with an equation such as “y = 2x + 5” and generate points over the interval of 0.0<x<10.0 by 0.5.

y=slope*x + int

Let use generate some “random” noise that is a number between -2<noise<2 uniformly selected.

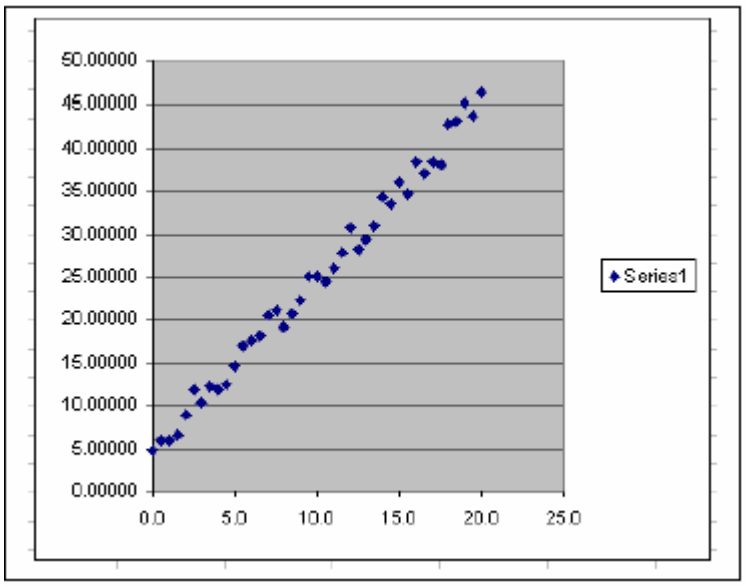
- **=randbetween(min_noise,max_noise)**

(unfortunately this only generates the values of -2, -1, 0, 1, or 2)

- **=(max_noise-min_noise)*RAND()+min_noise**

Now generate a scatter-plot showing the series “x” and “y”.

	A	B	C	D
1				
2	slope	2		
3	int	5		
4	min_noise	-2		
5	max_noise	2		
6				
7				y(raw)+
8	x	y(raw)	noise	noise
9	0.0	5	-0.30286	4.69714
10	0.5	6	-0.12353	5.87647
11	1.0	7	-1.11974	5.88026
12	1.5	8	-1.42164	6.57836
13	2.0	9	-0.11590	8.88410
14	2.5	10	1.89665	11.89665
15	3.0	11	-0.68577	10.31423
16	3.5	12	0.29106	12.29106
17	4.0	13	-1.24840	11.75160
18	4.5	14	-1.58091	12.41909
19	5.0	15	-0.52076	14.47924
20	5.5	16	0.95862	16.95862
21	6.0	17	0.45251	17.45251
22	6.5	18	0.02236	18.02236
23	7.0	19	1.40914	20.40914
24	7.5	20	1.14984	21.14984



Use the appropriate Excel functions to “estimate” the slope and intercept (as well as the R-sq value).

Computed Values	
Slope	1.937921
Intercept	5.521325
Rsq	0.991318

Computed Values	
Slope	=SLOPE(D9:D49,A9:A49)
Intercept	=INTERCEPT(D9:D49,A9:A49)
Rsq	=RSQ(D9:D49,A9:A49)

Note: Your values will be slightly different due to the “random” numbers being used in this example.

Linear Regression Using Excel's Trend Line Capability

It would be helpful to be able to see the regression line determined by Excel plotted against the data it was derived from. Fortunately this is very easy to produce by using Excel’s “tread line” capability.

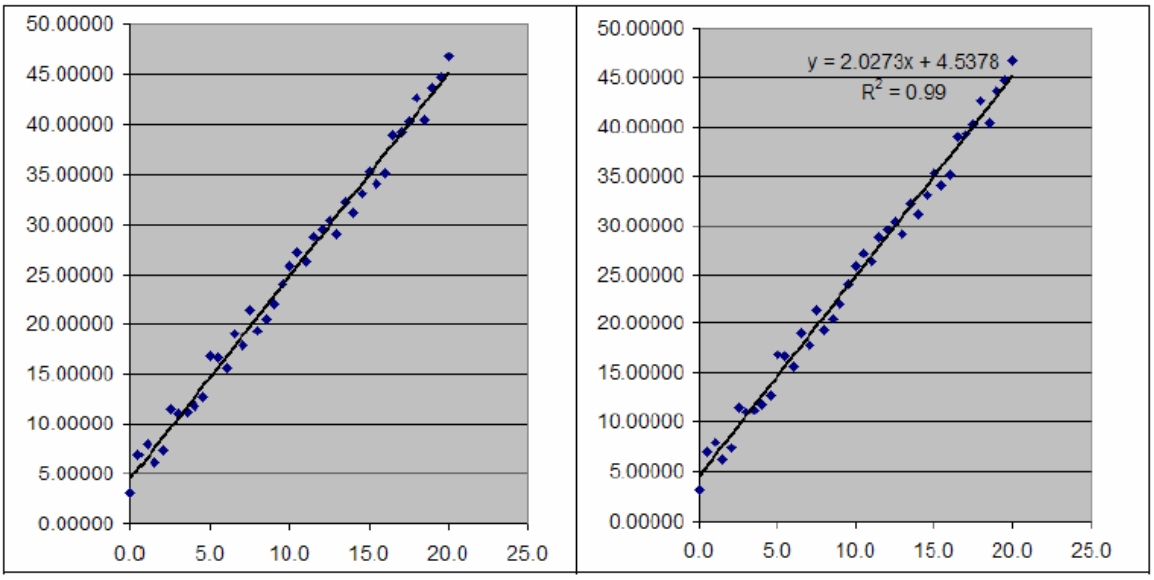
In order to produce the line manually one would solve the equation

$y_i = \text{slope}(x_i) + \text{int}$

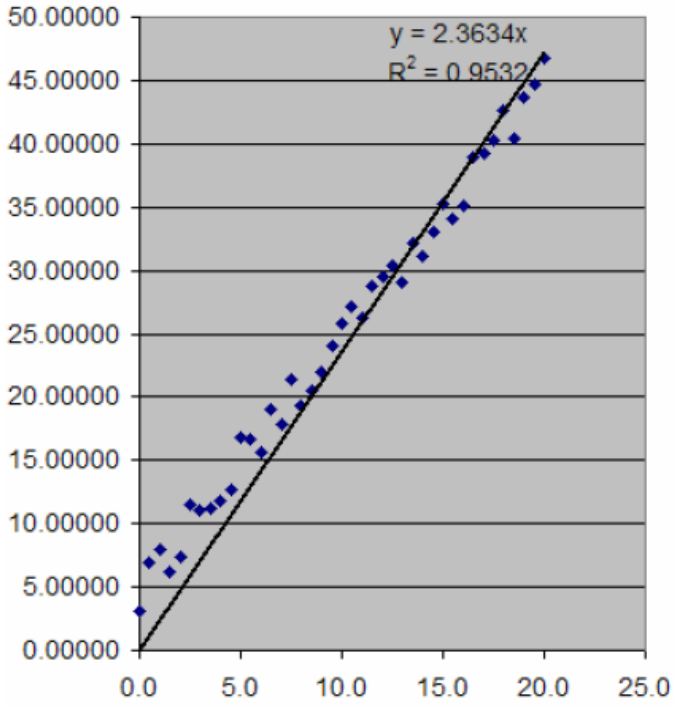
where i =1 .. npts.

We can simply return to the last graph, highlight the series to be associated with a trend line (we have only one series), and right click and select “Add Trendline”.

We can also specify (as Options) whether we want the numerical values of slope and intercept (as well as R-sq) on the graph.



Further, we can specify if we want to "force" the regression line through the origin (zero) or any other value we wish. When this is done the value of the slope is recalculated to properly "best fit" the data with this additional constraint.

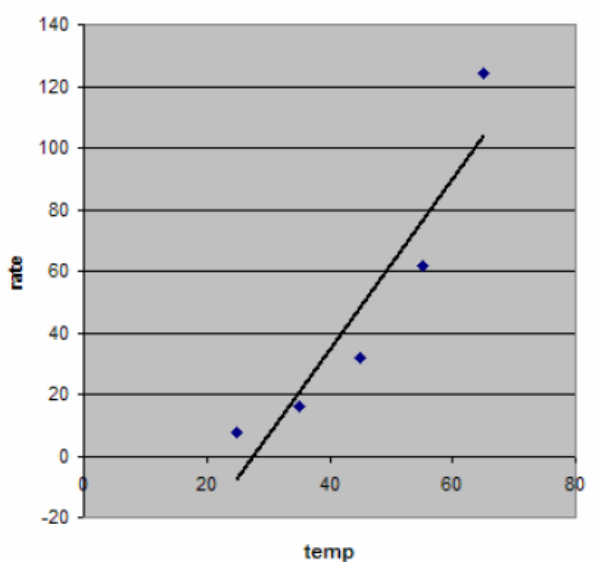


Other Two-Coefficient Linear Regression Models

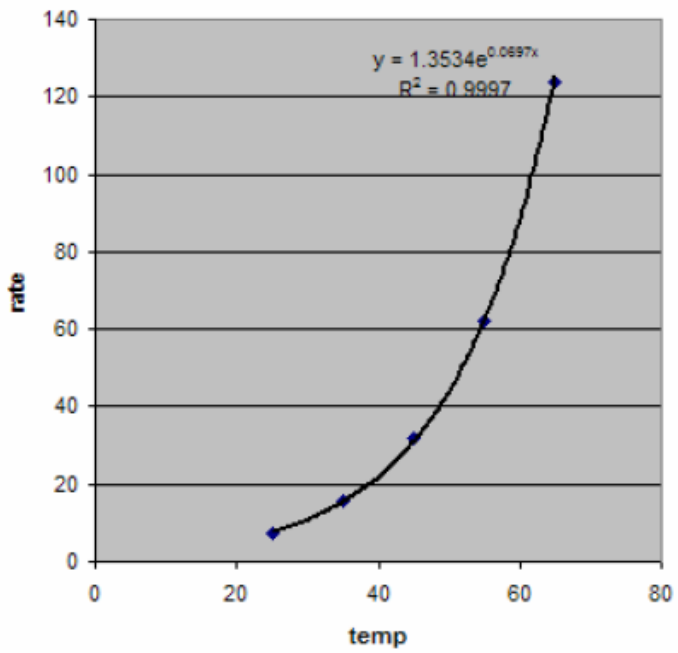
Often the data being represented does not follow a linear relationship. For example, if one is considering the rate of a reaction, generally the rate doubles for each 10 degrees (Celsius) increase in temperature.

Suppose we have collected the following data. If we prepare a linear plot as we did previously and fit a linear regression line to the data, we obtain a very poor plot. That is because the data would be expected to fit a semilog relationship (log of rate vs temp).

temp	rate
25	7.54
35	15.82
45	31.72
55	61.93
65	124.02



Instead we can retain the “linear” display of the data but fit the trend line to an exponential relationship. This is selected from the “Type” tab of the Format Trendline dialogue.



There are a number of “types” of regression fits available. These include

- Linear
- Logarithmic
- Polynomial
- Power
- Exponential
- Moving Average

All of these (with the exceptions of Polynomial and Moving Average) employ two regression coefficients and are hence termed Two-Coefficient Regression Models.

There are two important points to consider when selecting a “regression form”:

1. In some cases, the form of the functionality is known “a priori” and therefore regardless of the “fit” one should employ the correct functionality.
2. In other cases, the form of the functionality is not known and the objective is to produce a “pleasing (visually) line” to represent the data. Try each available function to see what fits best.

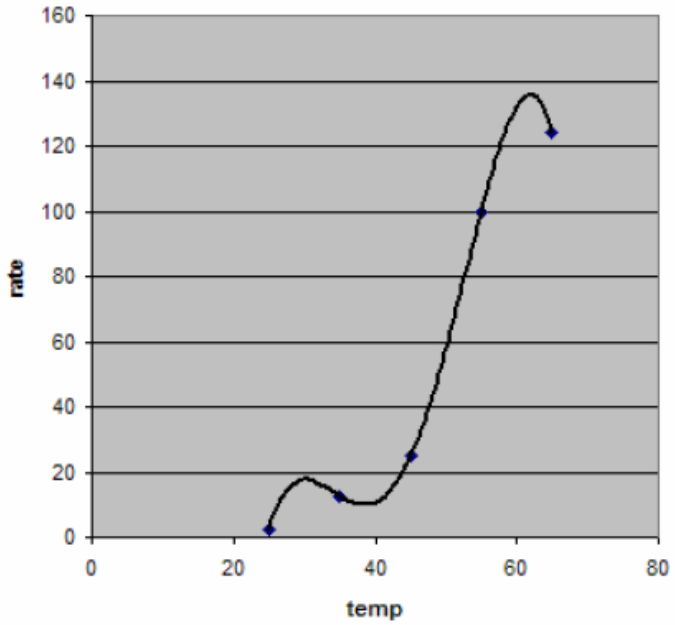
Polynomial Regression

One can sometimes account for “curvature” by employing a “higher order” function (polynomial regression). This fits the data to a function of the form:

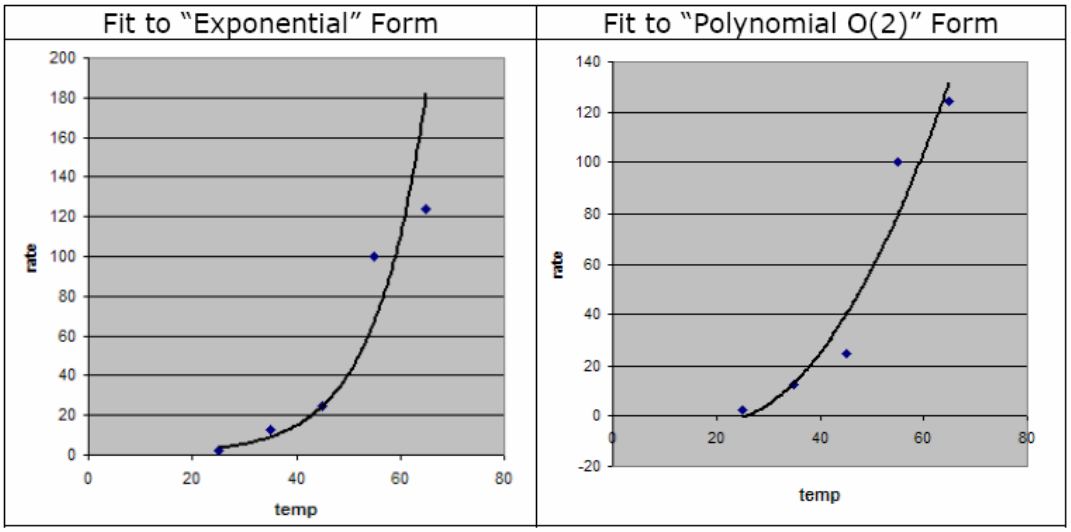
$Y = b_0 + b_1X + b_2X^2 + b_3X^3 \ldots$

In Excel (using the built-in trendline capabilities), all terms MUST be employed (you cannot force $b_2=0$ for example). In the next section, we consider how to use ANY kind of linear model.

When one is fitting data using the polynomial model, remember not to try to use “too high” an order of the equation in order to “better fit the data points” to “the line”. If one gets carried away doing this, very bad fits can arise where the regression curve “wildly bends around” while perfectly fitting the data points.

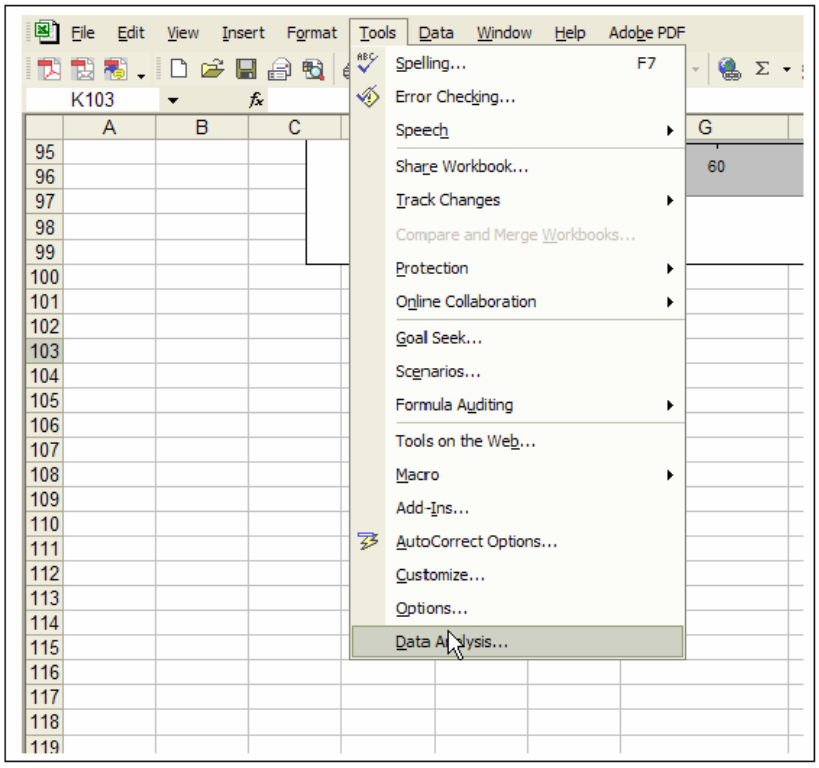


In this case, we fit 5 data points to a 4th degree polynomial. One is visually compelled to believe there are two local maximums and a local minimum around 40. In fast, the experiment data doesn’t suggest that at all. Both of the below better represent the “experimental data”.



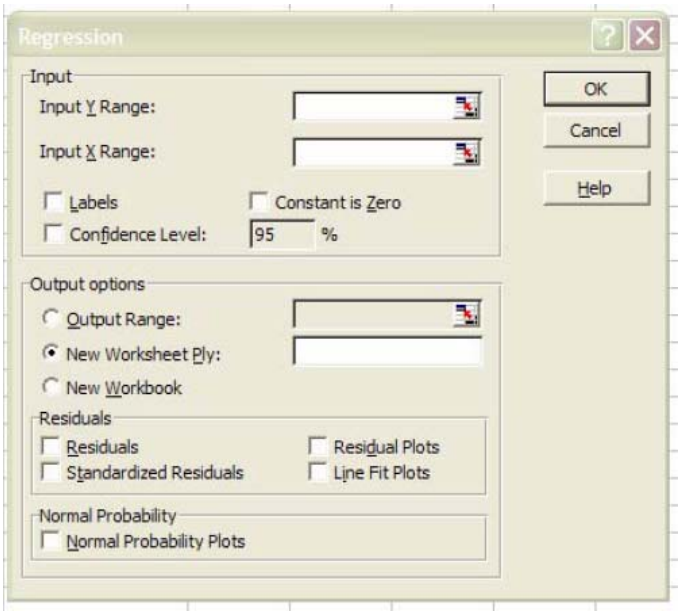
Linear Regression Using Excel's Regression Analysis Package

Excel's Linear Regression Analysis Package is part of the "Data Analysis" Add-In. This Add-In must be available (installed) AND must be selected (activated) before it can be used. This is done by checking the Analysis Toolpak in the Tools/Add-Ins menu.

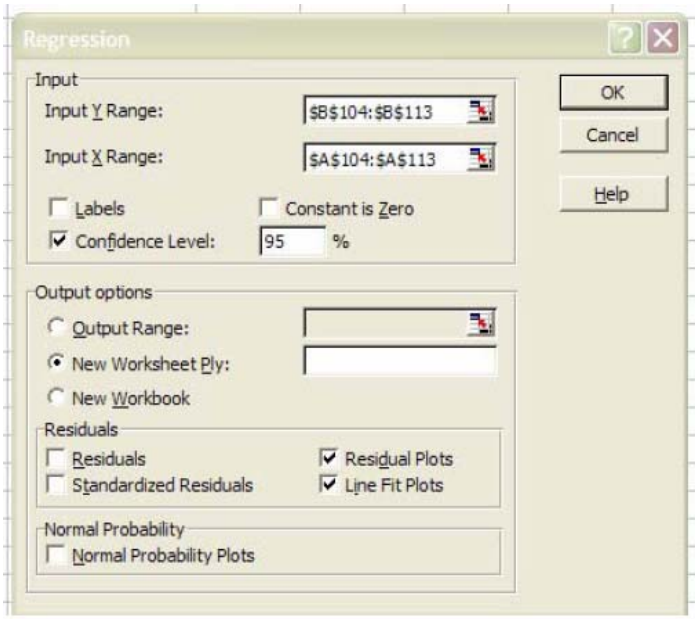


Using the data below

Time	Temp
0	298
1	299
2	301
3	304
4	306
5	309
6	312
7	316
8	319
9	322



Identify the “x” and “y” data ranges from the spreadsheet using the “to the spreadsheet” buttons. Also select the options shown below and direct the analysis to a “new worksheet ply”.



Summary Output

SUMMARY OUTPUT	
Regression Statistics	
Multiple R	0.99318635
R Square	0.986419126
Adjusted R Square	0.984721517
Standard Error	1.045915577
Observations	10

Analysis of Variance (Error) We will discuss the “F” statistic later in the course. Ignore for the present.

ANOVA					
	df	*SS*	*MS*	*F*	*Significance F*
Regression	1	635.6484848	635.6484848	581.0637119	9.35281E-09
Residual	8	8.751515152	1.093939394		
Total	9	644.4			

Here are the “results”. The “coefficients” are the “answers”. Intercept and Variable 1 are the “intercept” and “slope” (or coefficient of the “first ‘x’ variable” (we have only one in this case)).

	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*Lower 95.0%*	*Upper 95.0%*
Intercept	296.1090909	0.614740869	481.6811535	3.86444E-19	294.691495	297.5266868	294.691495	297.5266868
X Variable 1	2.775757576	0.115151515	24.10526316	9.35281E-09	2.510217534	3.041297618	2.510217534	3.041297618

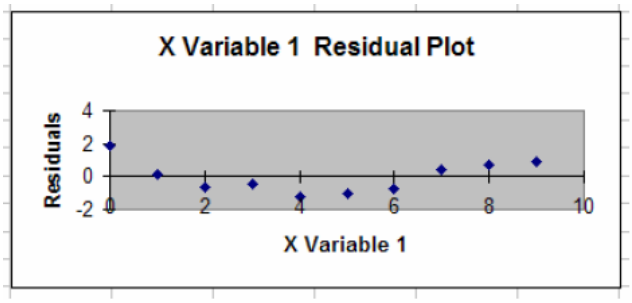
Also of note, we have an estimate of the confidence on the intercept and slope. In this case, we are 95% certain the data comes from a population where

- 294.69 < intercept < 297.52
- 2.5102 < slope < 3.0413

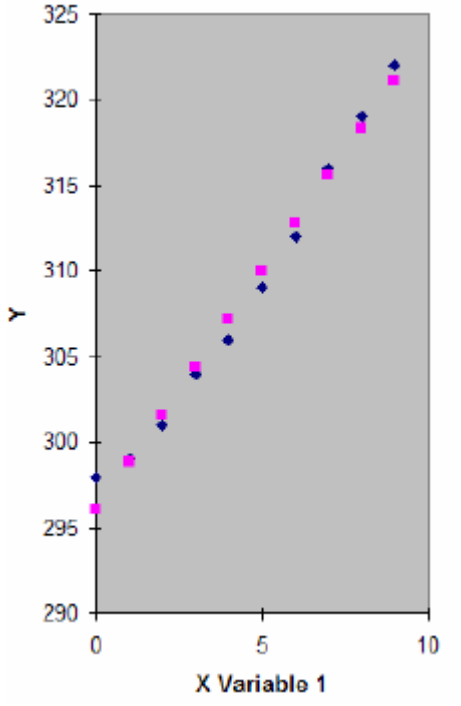
We will speak more about this “language” and its interpretation when we focus on statistics later in the course.

RESIDUAL OUTPUT		
Observation	*Predicted Y*	*Residuals*
1	296.1090909	1.890909091
2	298.8848485	0.115151515
3	301.6606061	-0.660606061
4	304.4363636	-0.436363636
5	307.2121212	-1.212121212
6	309.9878788	-0.987878788
7	312.7636364	-0.763636364
8	315.5393939	0.460606061
9	318.3151515	0.684848485
10	321.0909091	0.909090909

Putting the X values in the regression function, produces these “predicted y” values and “residuals” (differences).



This is a plot of the “residuals” (useful to spot “bad data values” and also “trends”).



This is a plot of the “observed y” and “predicted y”.

Note that there is an option to force the “constant to zero” as well as choosing a different “confidence level” (for example, 90% or 99.99%)

Using “Regression Analysis” for other functions

Suppose we wish to find the coefficient that fit the following function:

- $y = b0 + b_1x + b_2x^2 + b_3/x + b_4/x^2 + b_5 \log_{10}(x)$

Set up the problem as before except put in additional columns to hold values of x^2, $1/x$, $1/x^2$ and $\log_{10}(x)$.
Then, when you set the “input x range” highlight all the columns holding data. When you run the analysis, the “coefficients” will be associated with “var 1”, “var 2”, “var 3”, etc.

As an example, let’s try to add another term to our time-temperature data.

- $temp = b_0 + b_1(time) + b_2(time^{1.5})$

Temp	Time	Time^1.5
298	0	0
299	1	1.00
301	2	2.83
304	3	5.20
306	4	8.00
309	5	11.18
312	6	14.70
316	7	18.52
319	8	22.63
322	9	27.00

SUMMARY OUTPUT

Regression Statistics	
Multiple R	0.999228095
R Square	0.998456786
Adjusted R Square	0.998015868
Standard Error	0.37691359
Observations	10

ANOVA

	df	*SS*	*MS*	*F*	*Significance F*
Regression	2	643.405553	321.7027765	2264.494215	1.44375E-10
Residual	7	0.994446981	0.142063854		
Total	9	644.4			

	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*Lower 95.0%*	*Upper 95.0%*
Intercept	297.7453764	0.313227546	950.5721389	3.7659E-19	297.0047135	298.4860433	297.0047135	298.4860433
X Variable 1	0.891276455	0.258360231	3.449476186	0.010696909	0.280304731	1.502248179	0.280304731	1.502248179
X Variable 2	0.616290522	0.083402387	7.389363095	0.000150781	0.419075356	0.813505688	0.419075356	0.813505688

Based on the analysis, the regression equation is:

- temp = 297.745 + 0.8913(time) + 0.6163(time$^{1.5}$)

A note on "F-value". The F-statistic allows us to gauge which of several models is "better" in a particular statistical sense. When you compare models, the one with the larger "F" value is the "better". Our first model has an F=581.06 and the second model has an F=2264.49, hence the second model better fits the data.

A note on "P-values". The P-values are the probability that the coefficient shown for the parameter is actually "zero" (hence, this term really isn't in the model and should be eliminated and "rerun"). For example, the probability that the intercept is "zero" and not 297.74 is 3.76x10^{-19}. Generally if a P value is less than 0.05 (that is, 5% probability) then the term is NOT zero and IS significant and should be left in the model. In our case, CLEARLY the intercept is NOT zero. Another way of looking at this is to investigate the "95% confidence interval". We can be 95% confident that the actual value of the intercept is between:

- 297.004 < intercept < 298.486

and obviously the value "0" is not in that interval.

The coefficient of (time) is 0.8913 and the probability of this coefficient actually being "zero" is only 0.01069 or 1.069%. Looked at another way, when we collect data of this sort 100 times, only 1 time in the 100 would we be justified in eliminating the "time" term from the model and rerunning it.

The coefficient of (time^1.5) is 0.00015 and therefore is significant and should be left in the model.

Iterative Solutions Using Excel

Introduction

Sometimes it is “hard to see the forest for the trees”. As engineers, we work with equations so often we sometimes forget what we are really doing. When we “solve” and equation, we are really finding the value of an unknown quantity that makes both sides of the equation equal in value. Usually we have simple or even trivial rules to follow for how to do this.

For example, to find the area of a circle, we employ the formula:

$$A = \pi D^2 / 4$$

To find the area of a circle with a diameter of 2 ft we plug-in (trivial) the value of 2 for D and carry out the calculations. This gives the right hand side (RHS) value of “3.141592” so we say the value of A is “3.131592” also.

Suppose we needed to find the diameter when the volumetric flowrate is 1.0 ft^3/s and the average velocity is 3 ft/s.

From fluids we know:

$$Q = V A$$

So knowing that $A = \pi D^2/4$ we have

$$Q = V \pi D^2/4$$

Or substituting (plugging in) the values we have gives

$$1 = 3 \pi D^2/4$$

Now we employ the simple rule “rearrange and simplify” to give

$$\frac{(4)(1)}{3\pi} = D^2$$

Now the LHS is 0.4244 so the RHS must be 0.4244 so we take the square root and find D = 0.65147.

At no time did we think about or use the words “finding the root” of the equation, but in actually that’s exactly what we did. D = 0.65147 is the root of the equation 1 = 3 π D2/4 and we could have found the value of the root just like we do when we are “told” to find the root.

For example: What is the root of the equation $2x^2+2x-3=0$?

Graphical Solution

One way to “find the root” is to graph the equation f(x) against “x” and see where we cross the x-axis.
Rather than typing in the “typical expression” for the function, let’s learn how to write UDF’s in Excel.

A UDF (**User Defined Function**) is simply a function that you create yourself with VBA. UDFs are often called "Custom Functions". A UDF can remain in a code module attached

to a workbook, in which case it will always be available when that workbook is open. Alternatively you can create your own add-in containing one or more functions that you can install into Excel just like a commercial add-in.

UDFs can be accessed by code modules too. Often UDFs are created by developers to work solely within the code of a VBA procedure and the user is never aware of their existence.

Like any function, the UDF can be as simple or as complex as you want. Let's start with an easy one...

A Function to Calculate the Area of a Rectangle

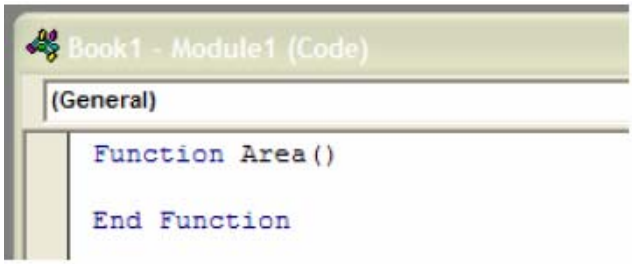
This is the calculation to be done:

AREA = LENGTH x WIDTH

Open a new workbook and then open the Visual Basic Editor (Tools > Macro > Visual Basic Editor or ALT+F11).

You will need a module in which to write your function so choose Insert > Module. Into the empty module type: **Function Area** and press ENTER. The Visual Basic Editor completes the line for you and adds an End Function line as if you were creating a subroutine.

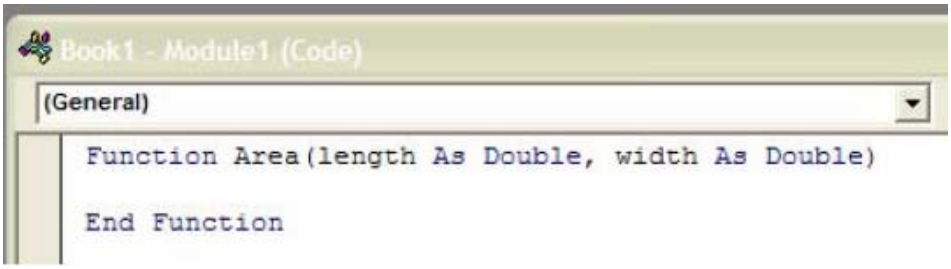
So far it looks like this...



Place your cursor between the brackets after "Area". We are going to specify the "arguments" that our function will take (an argument is a piece of information needed to do the calculation). Type **length as double, width as double** and click in the empty line underneath. Note that as you type, a scroll box pops-up listing all the things appropriate to what you are typing.

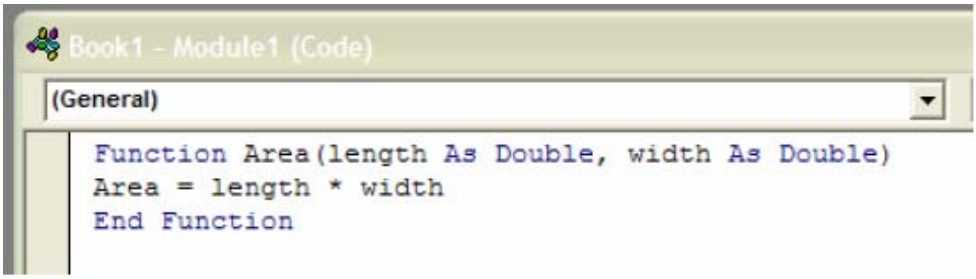
This feature is called Auto List Members. *If it doesn't appear either it is switched off (turn it on at Tools > Options > Editor) or you might have made a typing error earlier.* It is a very useful check on your syntax.

Find the item you need and double-click it to insert it into your code. You can ignore it and just type if you want. Your code now looks like this...



Declaring the data type of the arguments is not obligatory but makes sense. You could have typed length, width and left it as that, but warning Excel what data type to expect helps your code run more quickly and picks up errors in input. The double data type refers to number (which can be very large) and allows whole numbers and decimals.

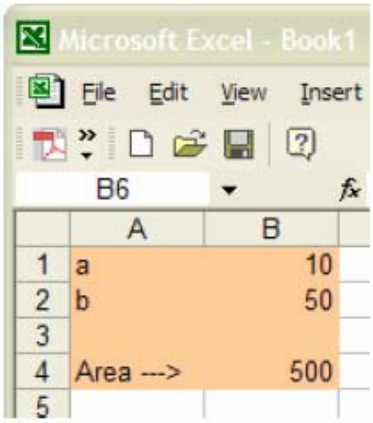
Now for the calculation itself. In the empty line first press the TAB key to indent your code (making it easier to read) and type Area = length * width. Here's the completed code...



Switch to the Excel window and enter data for variables “a” and “b” and use the “name / create” to assign these as range variables.

Now enter the function in the cell next to “Area --->”

=area(a,b)



Advanced feature: Sometimes, a function's arguments can be optional. In this example we could make the “width” argument optional. Supposing the rectangle happens to be a square with “length” and “width” equal. To save the user having to enter two arguments we could let them enter just the “length” and have the function use that value twice. So the function knows when it can do this we must include an IF Statement to help it decide.

Change the code so that it looks like this...

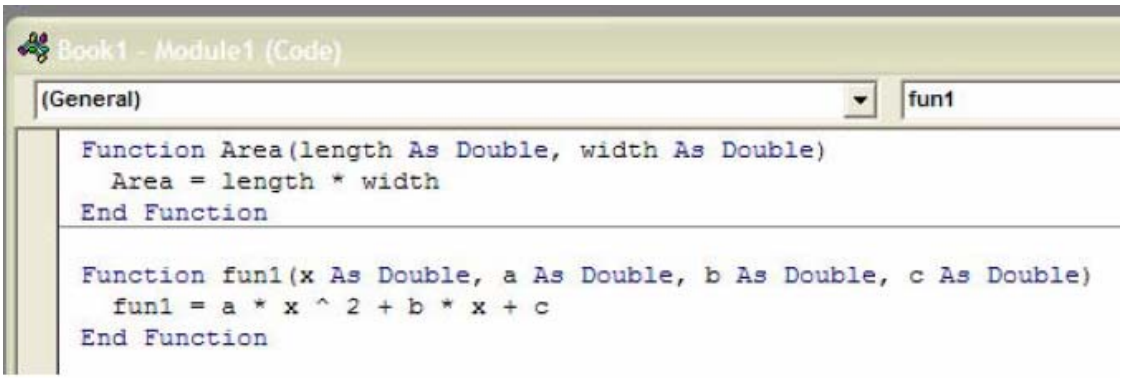
Function Area(length As Double, Optional width As Variant)

```
        If IsMissing(width) Then
                Area = length * length
        Else
                Area = length * width
        End If
End Function
```

Note that the data type for "width" has been changed to Variant to allow for null values. The function now allows the user to enter just one argument e.g. =area(A1). The IF Statement in the function checks to see if the "width" argument has been supplied and calculates accordingly.

Now we are ready to return to the graphing of the function $2x^2+2x-3=0$.

Return to the VBA editor and enter the function fun1 as below.



1. Return to the spreadsheet and enter data for _a, _b, and _c.
2. Enter the data for xmin and dx
3. Generate values from 1 to 51 in column "A"
4. Enter the formula for the first "x" value

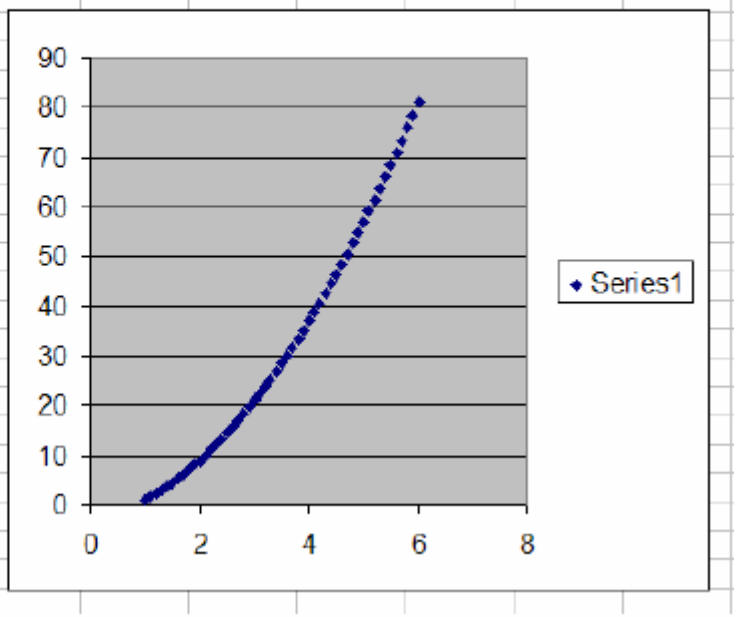
   =(A11-1)*dx+xmin

5. Copy the formula for "x" to the other cells
6. Enter the formula for the first "f(x)" value

   =fun1(B11,_a,_b,_c)

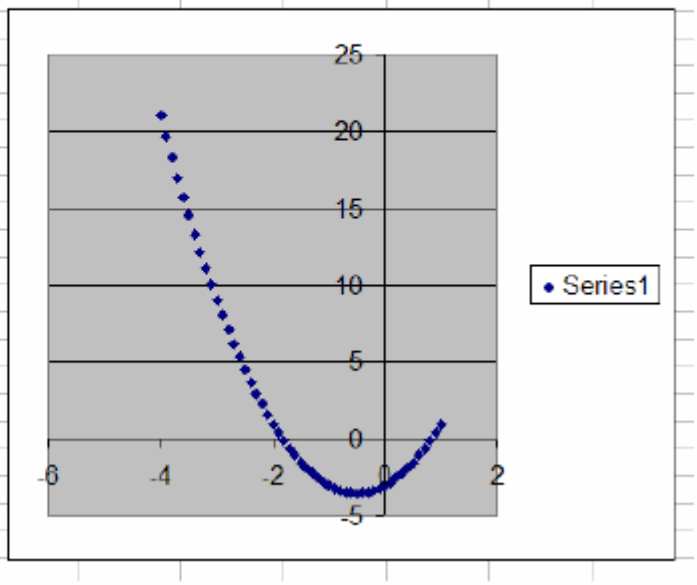
7. Copy the formula for "f(x)" to the other cells
8. Graph the function x vs f(x)

5					
6	_a	2		xmin	1
7	_b	2		dx	0.1
8	_c	-3			
9					
10	n	x	f(x)		
11	1	1	1		
12	2	1.1	1.62		
13	3	1.2	2.28		
14	4	1.3	2.98		
15	5	1.4	3.72		
16	6	1.5	4.5		
17	7	1.6	5.32		
18	8	1.7	6.18		
19	9	1.8	7.08		
20	10	1.9	8.02		
21	11	2	9		
22	12	2.1	10.02		
23	13	2.2	11.06		
24	14	2.3	12.18		
25	15	2.4	13.32		
26	16	2.5	14.5		
27	17	2.6	15.72		
28	18	2.7	16.98		
29	19	2.8	18.28		
30	20	2.9	19.62		
31	21	3	21		

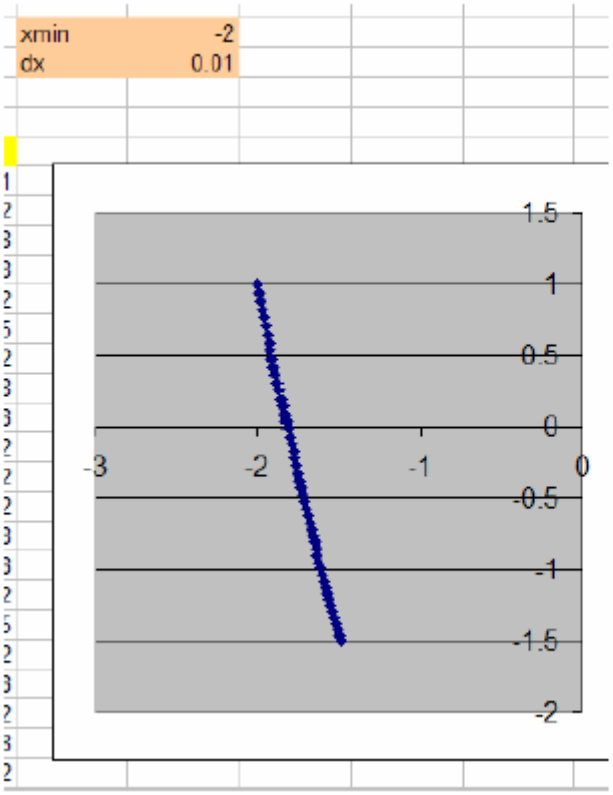


Replace the “x” values with xmin=-4 and dx=0.1

5					
6	_a	2		xmin	-4
7	_b	2		dx	0.1
8	_c	-3			
9					
10	n	x	f(x)		
11	1	-4	21		
12	2	-3.9	19.62		
13	3	-3.8	18.28		
14	4	-3.7	16.98		
15	5	-3.6	15.72		
16	6	-3.5	14.5		
17	7	-3.4	13.32		
18	8	-3.3	12.18		
19	9	-3.2	11.08		
20	10	-3.1	10.02		
21	11	-3	9		
22	12	-2.9	8.02		
23	13	-2.8	7.08		
24	14	-2.7	6.18		
25	15	-2.6	5.32		
26	16	-2.5	4.5		
27	17	-2.4	3.72		
28	18	-2.3	2.98		
29	19	-2.2	2.28		
30	20	-2.1	1.62		
31	21	-2	1		



Replace the “x” values with xmin=-2 and dx=0.01



17	-1.84	0.0912
18	-1.83	0.0378
19	-1.82	-0.0152
20	-1.81	-0.0678
21	-1.8	-0.12

At this point we have "located" the root (value where RHS and LHS are "approximately equal") to an accuracy of 0.01. It is between -1.83<root<-1.82 since that is where the sign of f(x) changes.

Note: This suggests we could write some additional Excel statements to develop a way to "count" the number of roots in an interval.

Note: It is also obvious that instead of "generating" values for "x" we are free to simply type values in column "B" and see if we can "guess" a value that makes column "C" be approximately "0.0".

Note: We can also place the cursor over the point nearest where the function crosses the axis and the "popup" will report the value. This technique is also helpful when identifying maxima, minima or other points of interest.

In this introduction, we have seen three ways to identify roots:

1. Graph the function and look for where the function crosses the axis
2. Tabular the function and look for where the function changes sign
3. Guess values repeatedly

Other Iterative Solution Methods

Direct Substitution Method

Suppose we wish to solve the equation $x = \frac{x^3 + 12}{17}$

Start by “solving” (isolating) one of the terms for “x” explicitly, that is, get “x” only on one side of the equation. Depending on the equation to be solved, there may be several different rearrangements.

Call the LHS variable xC (calculated) and the variable in the RHS xG (guessed).

We will proceed by assigning values for xG and xC will be computed (LHS).

1. Enter the “guess” value in underneath xG (0.8 in this case)
2. Enter the formula underneath xC (=(A4^3+12)/17 in this case)
3. Enter the formula to copy the value of the calculated “x” to be the next guess (that is, type =B4 in cell A5)

	A	B	C
1	**Direct Substitution Example**		
2			
3	**xG**	**xC**	
4	0.8	0.736	
5	0.736		
6			
7			

4. Copy the formula in cell A5 into the cells underneath it.
5. Copy the formula in cell B4 into the cells underneath it.

Note that you can try different starting values and “should” still converge (but perhaps to a different value).

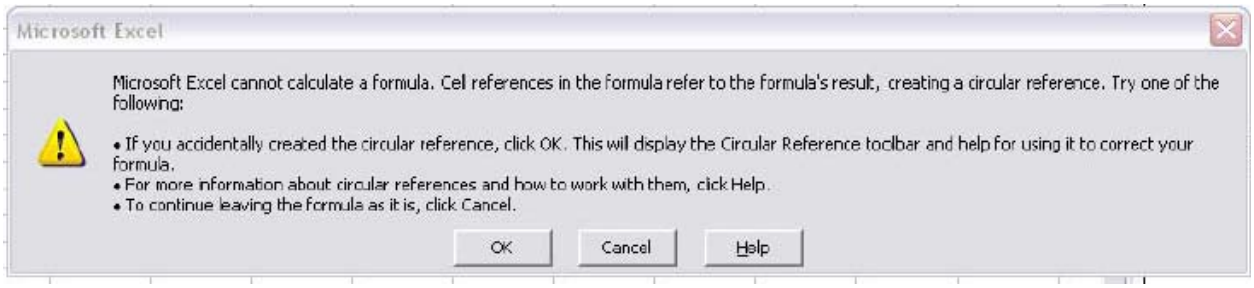
	A	B	C
1	**Direct Substitution Example**		
2			
3	**xG**	**xC**	
4	0.8	0.736	
5	0.736	0.729335	
6	0.729335	0.728703	
7	0.728703	0.728644	
8	0.728644	0.728638	
9	0.728638	0.728638	
10	0.728638	0.728638	
11	0.728638	0.728638	
12	0.728638	0.728638	
13	0.728638	0.728638	
14			

	A	B	C
1	**Direct Substitution Example**		
2			
3	**xG**	**xC**	
4	2	1.176471	
5	1.176471	0.801667	
6	0.801667	0.736189	
7	0.736189	0.729353	
8	0.729353	0.728705	
9	0.728705	0.728644	
10	0.728644	0.728638	
11	0.728638	0.728638	
12	0.728638	0.728638	
13	0.728638	0.728638	
14			

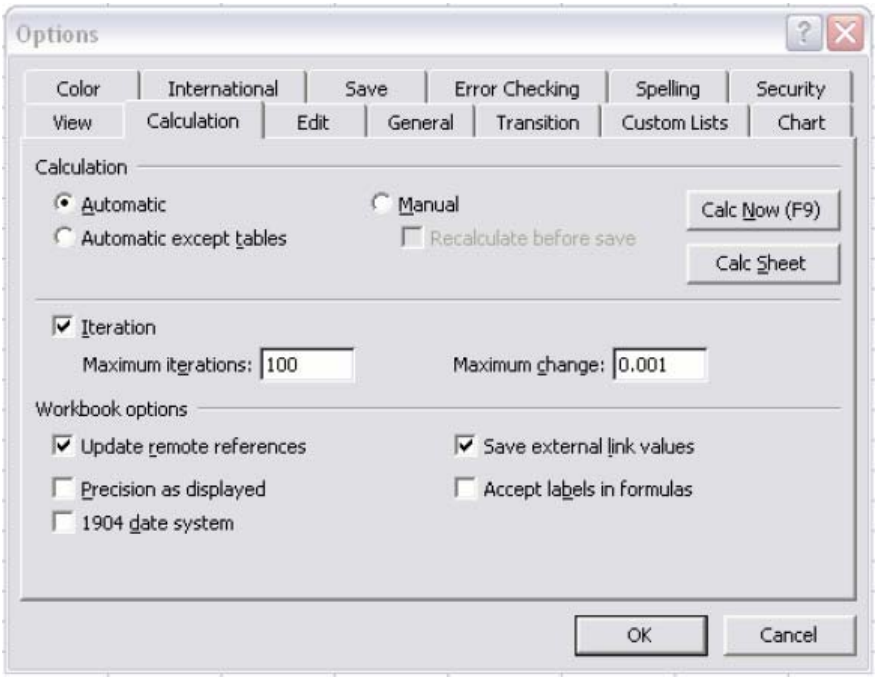
<u>In Cell Iteration</u>

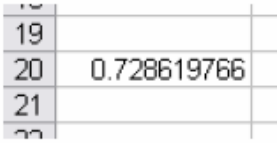
We can be even more clever (if we are careful) by allowing Excel to update the calculated value from the guessed value formula automatically. Usually Excel will try to prevent you from doing this since it is considered an error to have a so-called “circular reference”. However, if we inform Excel that we are trying to do this and “limit” the number of times it does the substitution, all will be “okay”.

Let’s go to cell A20 and type in =(A20^3+12)/17



Open the “Tools/Options” menu and check “Iteration”. Click “OK”



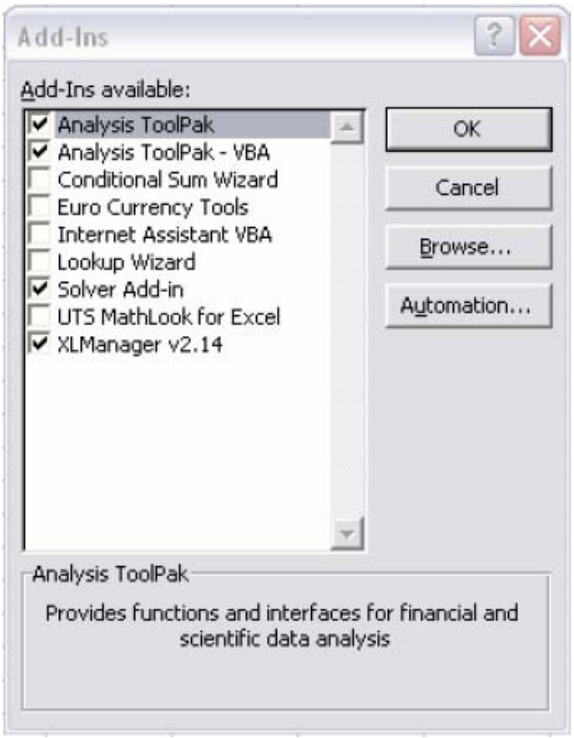


The “answer” appears in the cell A20. Although pretty “slick” this has several limitations. The starting value for the iteration is always 0.0 and therefore, if you have multiple roots you probably will not be able to rearrange the equation to necessary find all the roots.

Introduction to Excel's Solver

The solver is an Add-In that has to be installed before use. As with the Statistical Package, it is part of Excel but not installed by default.

Go to “Tools/Add-Ins”.

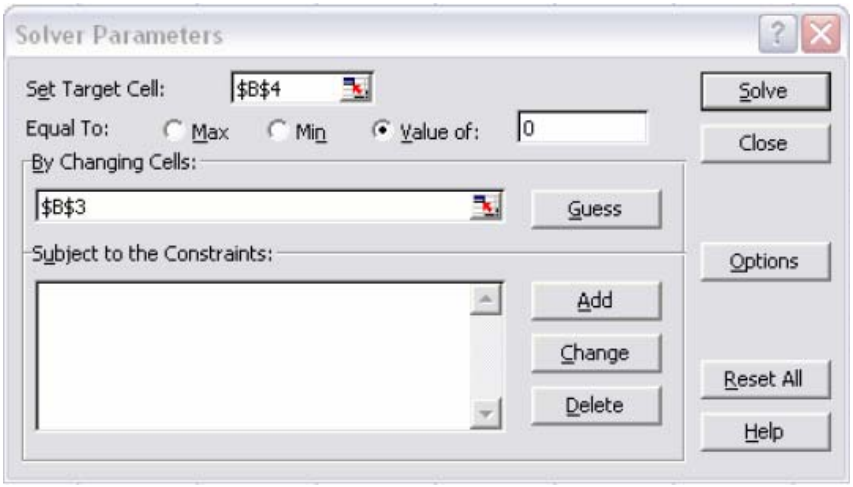


Once installed, Solver is easy to use. Returning to the problem we have been considering, setup the following data:

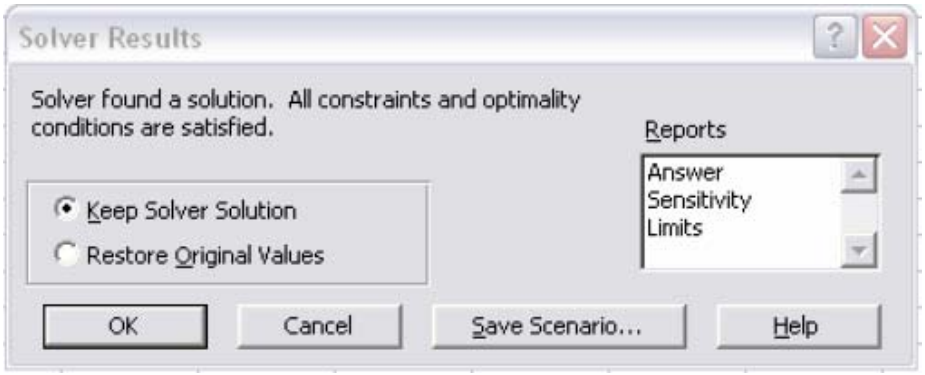
Type the guess value in B3 (3.6) and the formula =B3^3-17*B3+12 in B4

	A	B
1	**Solver Example**	
2		
3	**Guess:**	3.6
4	**Formula:**	-2.544

Now, start Solver by selecting “Tools/Solver” and fill in the values as shown:



Press “Solve” to see both cells update.



	A	B
1	**Solver Example**	
2		
3	**Guess:**	3.710213579
4	**Formula:**	-1.55514E-07

Note that we found the value of x (=3.710213) that makes the function equal to 0.0 (actually -1.555E-7 in this case).

The idea behind solver is very simple… Excel will control the value of the target cell (to make it equal to the value specified) by varying the value in the "Changing Cell". It is obviously required that when one changes the value in the "Changing Cell" it affects the answer in the "Target Cell".

Optimization Using the Solver

Single and Multiple Variable Optimization

The Solver can be used to determine unconstrained and constrained optimization problems (mins and maxs).

Type	Vars	Typical Function	Find	Constraints
1	1	$y=x^3-14x^2+3$	x@max(y)	none
2	1	$y=x^3-14x^2+3$	x@max(y)	3<x<7
3	2+	$z=x^3-3x^2+y^3$	x,y@min(z)	none
4	2+	$z=x^3-3x^2+y^3$	x,y@min(z)	x>3 x+y<5

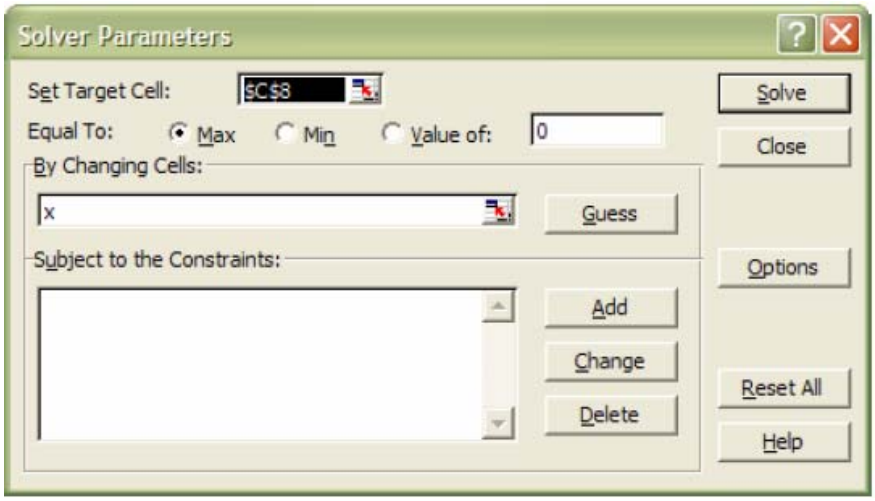
Type 1 – Simple 1 Var Unconstrained

$$y = 10+8x-x^2$$

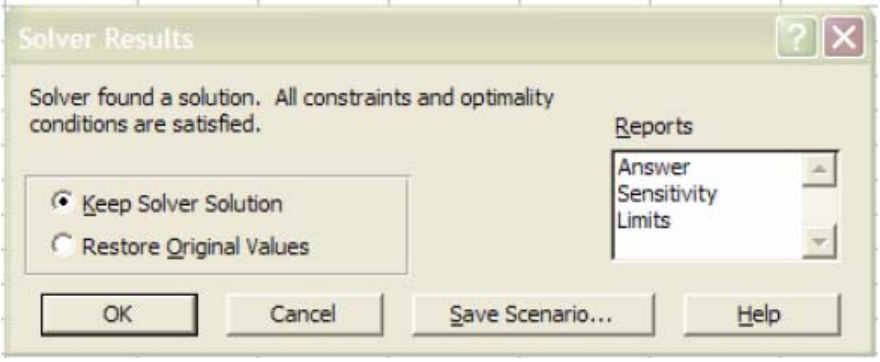
Guess	By Changing Cell
y=f(x)	Target Cell

x	2
y=f(x)	=10+8*x-x^2

x	2
y=f(x)	22

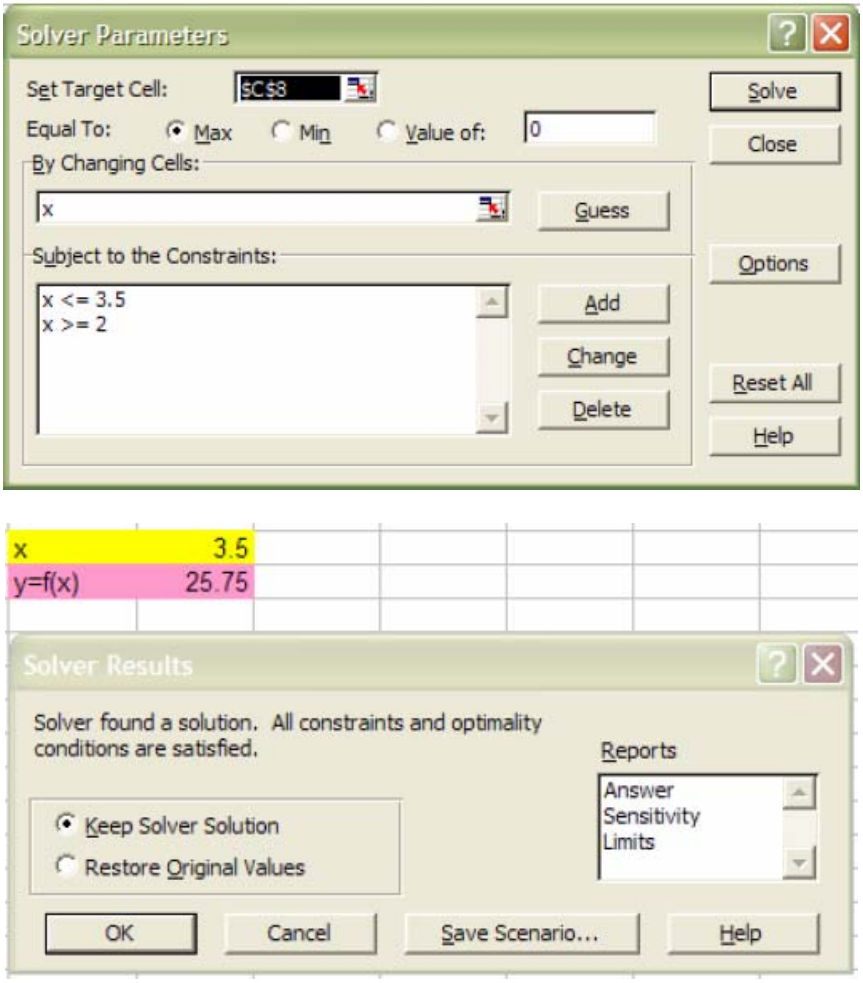


x	4
y=f(x)	26



Type 2 – Simple 1 Var with Constraints

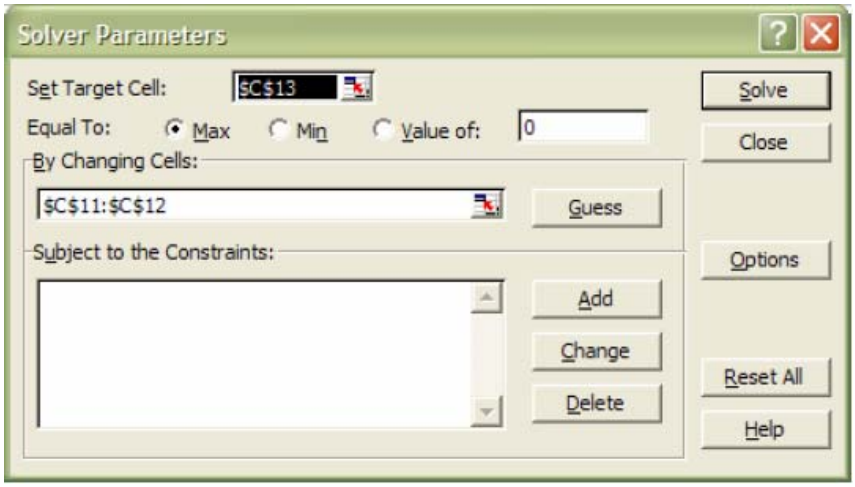
Suppose we wish to limit our search to the region 2<x<4.



Type 3 – Multi Variable Unconstrained

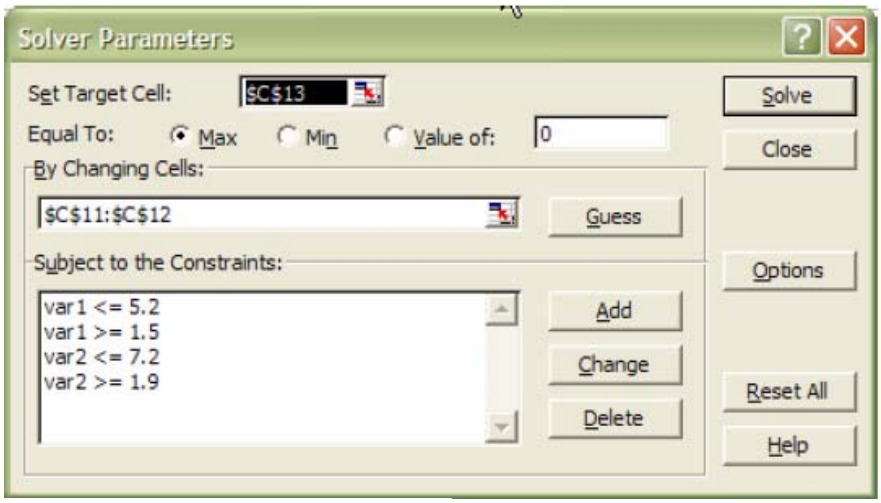
var1	2
var2	2
u=f(var1,var2)	-0.3784

var1	2
var2	2
u=f(var1,var2)	=SIN(var1)*COS(var2)



var1	1.5708
var2	-1.8E-07
u=f(var1,var2)	1

Note that for periodic functions there may be many (or infinite) max's or min's. Therefore, Excel will converge to whatever solution it happens upon in its search. In order to have more control, you should provide constraints for a region in which you know only one solution exists.



var1	4.71239
var2	3.14159
u=f(var1,var2)	1

Always graph or otherwise test the solution to be sure you have converged to the appropriate root.

Nonlinear Regression

In order to implement non-linear regression we take a more "mathematical" approach than we did when we employed matrix functions to "automatically" determine the solution of linear systems of equations.

The problem we will solve is to take a set of vapor pressure data and fit the parameters of the Antoine Equation using nonlinear regression.

In this case:

$$\log\left(p^*\right) = A - \frac{B}{C+T}$$

A quick investigation shows there is no way to "linearize" this equation so we must employ nonlinear regression techniques to establish the values of A, B, and C for a particular substance. In our case we will use data from a different source than the textbook, namely, n-pentane's vapor pressure as reported in Perry's Handbook.

Start by entering the available data (T, p*) as well as providing for "guesses" for A, B, and C. I have "named" these as conA, conB, and conC.

Vapor Pressure of Propane

	Guesses
ConA	5
ConB	1000
ConC	250

T (C)	p* (mm Hg)
-76.6	1.0
-62.5	5.0
-50.1	10.0
-40.2	20.0
-29.2	40.0
-22.2	60.0
-12.6	100.0
1.9	200.0
18.5	400.0
36.1	760.0

In regression analysis one minimizes the sum of the squared error (SSE) which is defined as

$$SSE = \sum_{i=1}^{N} \left(y_i - y_p\right)^2$$

where yi are the original data, yp are the predicted values using assumed values for A, B, C.

Then add columns for log(p*), y, y_p, error and error^2

where error = y_i-y_p

The value for y_p (D9) are derived from =ConA-(ConB/(ConC+A9)) and the others are obvious.

ALSO, add the entry for SSE which is the sum of the "error^2" values. A portion of the results are shown below.

Vapor Pressure of n-Pentane

	Guesses	Started
ConA	6.663363701	5
ConB	1002.383476	1000
ConC	227.8789998	250

SSE	0.01254

T (C)	p* (mm Hg)	y=log(p*)	y_p	error	error²	p*pred	diff
-76.6	1.0	0.00000	0.03731	-0.03731	0.00139	1.08970	-0.08970
-62.5	5.0	0.69897	0.60223	0.09674	0.00936	4.00161	0.99839
-50.1	10.0	1.00000	1.02500	-0.02500	0.00062	10.59243	-0.59243
-40.2	20.0	1.30103	1.32242	-0.02139	0.00046	21.00958	-1.00958
-29.2	40.0	1.60206	1.61812	-0.01606	0.00026	41.50711	-1.50711
-22.2	60.0	1.77815	1.78983	-0.01168	0.00014	61.63541	-1.63541
-12.6	100.0	2.00000	2.00716	-0.00716	0.00005	101.66166	-1.66166
1.9	200.0	2.30103	2.30098	0.00005	0.00000	199.97836	0.02164
18.5	400.0	2.60206	2.59490	0.00716	0.00005	393.46146	6.53854
36.1	760.0	2.88081	2.86615	0.01466	0.00021	734.77529	25.22471

Date From Perry's Handbook

We use Solver to minimize SSE by varying A, B, and C.

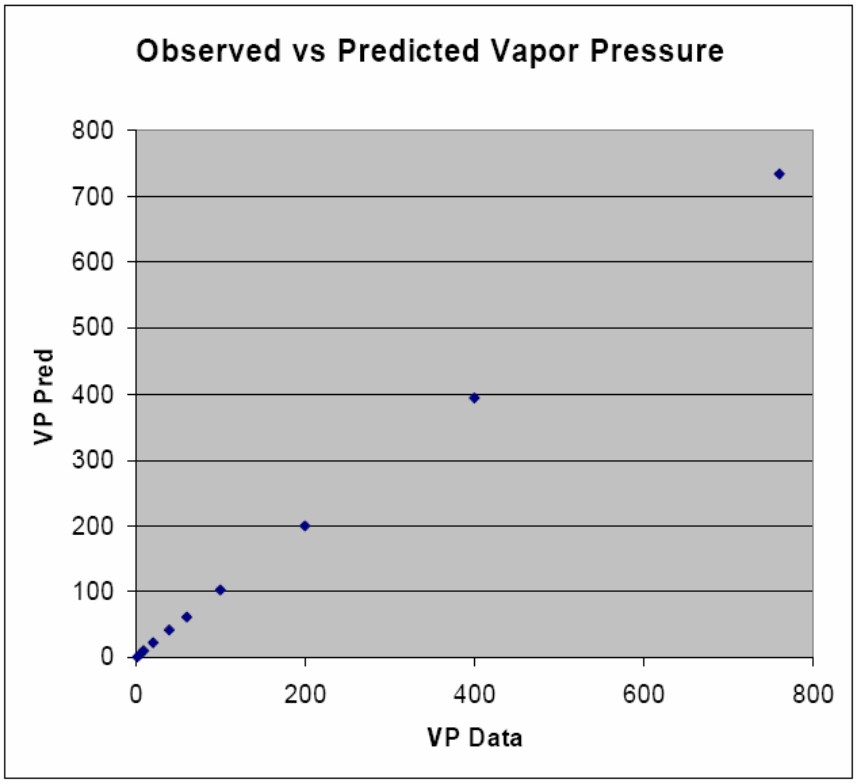
Testing the Regression Result

In order to verify the success of the regression we should plot the original data against the predicted values obtained by employing the constants determined by the nonlinear regression.

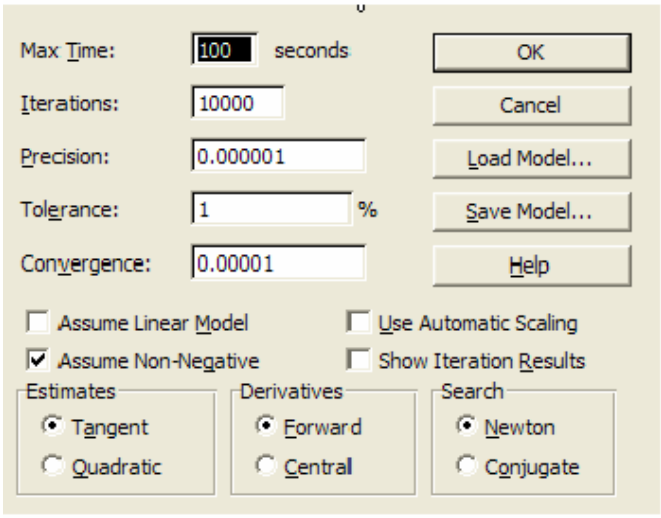
Note, this is not the same as plotting the original values AND the predicted values against the independent variable (temperature, in this case).

One can also “visualize” the fit by plotting the “error” or deviation, as it is called in statistics. The data needed for both these plots can easily be developed (as shown).

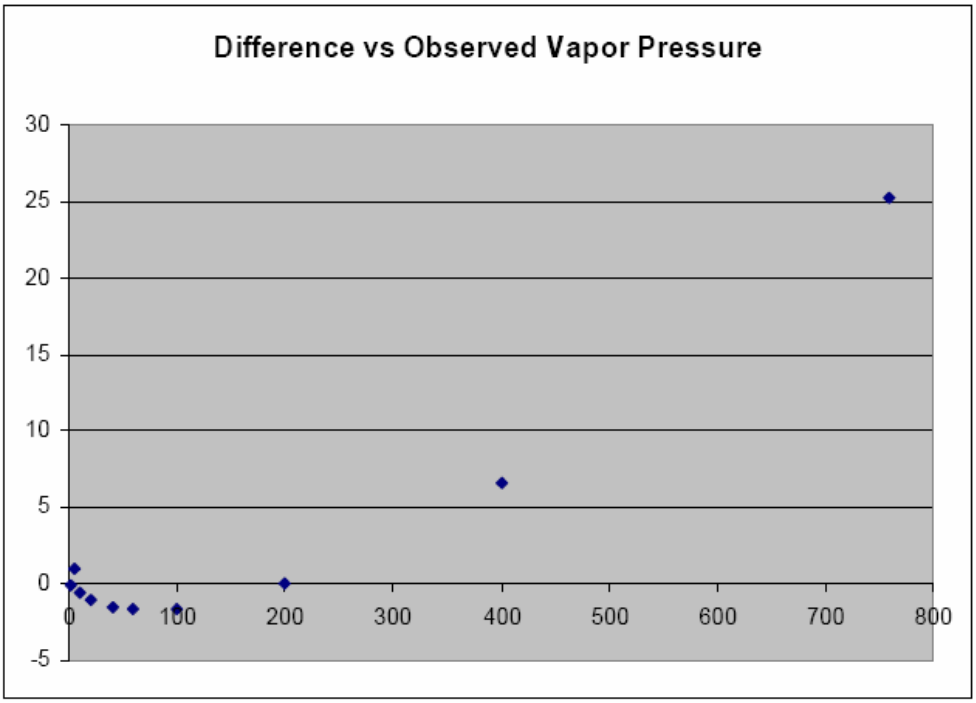
REMEMBER TO START A NEW CHART IN A “CLEAR” AREA OF THE WORKSHEET !!



Since the fit was not “superlative” one should consider if the method needed tighter tolerance, more iterations, or more time. Use the Options button of the Solver menu to adjust these.



Reviewing the new results:



Since the fit STILL is not “superlative” one should consider

1. The quality of the data
2. Did WE make a mistake in the analysis (error in formula, etc).
3. Does the deviation data suggest systematic error? If so, can a better model be proposed and fit with additional parameters.

It should be noted, the book has analyzed the data for water (very accurately known) and they had available 42 data points (whereas we have only 10 data points to estimate the coefficients for n-pentane.

The values listed in Felder are:

	Felder	Solver/Perry’s
A	6.84471	6.663363
B	1060.793	1002.3834
C	231.541	227.87899
Temp Range	13.3 to 36.8 ^{o}C	-76.6 to 36.1 ^{o}C

Using Macros in Excel

Introduction

Before becoming familiar with VBA (Visual Basic for Applications) we will focus on a related “capability” of Microsoft Office Products to record and replay user-entered keystrokes and other input device information (mouse movements, clicks, etc).

A “macro” refers to an “abbreviation”, that is, something that refers to something else. The postal abbreviation OH represents OHIO and Cntl-C is an “abbreviation” for “Edit/Copy”. We are very used to pressing “icons” on toolbars and having the expected actions take place and these are nothing more than “visual macros” (for example, pressing the “File Save” or “Print” icons).

One of the main advantages of macros is that they can “flawlessly replay” complex sequences of keystrokes (thus saving us a lot of time and effort). One of the major disadvantages of macros is that they can “mindlessly and swiftly replay” a complex sequence of keystrokes that should never have been applied to the task at hand. Thus, macros are “double edge swords” with both advantages and disadvantages.

A similar example of this might be to develop a macro to build the “boilerplate” (template) of identification information at the start of a new spreadsheet. By assigning this macro to the keystroke “Cntl-Q” one could get a fast start on a new assignment. On the other hand, if you were working on some other part of the spreadsheet and accidently pressed “Cntl-Q” it is possible you might overwrite dozens of cells of information and formulas because you didn’t start running the macro from row/column A1.

The easiest way to create a macro in Excel is just to let Excel “listen in” on what keys/etc you are pressing and store these to be later executed by a single keystroke. These “saved keystrokes” can be viewed and edited later.

Another way to create a macro is to start from scratch using the VBA Editor.

The largest limitation of macros is that they cannot accept information “on the fly” (that is, there is no “argument list” to pass information to the macro). Hence, you can write a macro to “add three” to the cell immediately to the left of the current cell, but you cannot write a macro to add “some number” to that same cell.

Macros and Viruses

Macros (from unknown or untrusted sources) are potentially dangerous (acting as a virus) and capable of damaging a computer and its information. Previous versions of Excel were configured to run macros “automatically” however, current versions provide four levels of security to limit the dangers macros represent.

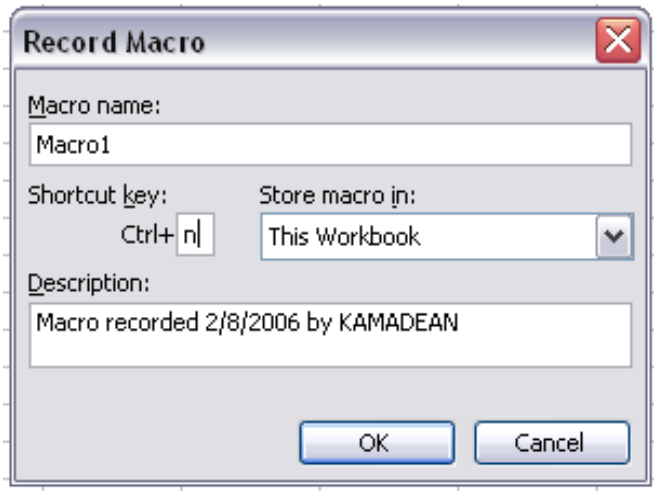
Security:

- Very High – Only macros in installed in trusted locations (folders) are allowed to run (you must maintain a list of the trusted locations)
- High – Only macros “digitally signed” by trusted sources are allowed to run (you must maintain a list of trusted sources)
- Medium – Excel will prompt to run macros not from trusted sources.
- Low – All macros allowed to run without warning.

Recording Macros

Macro 1: Typing on the Diagonal – "Cntl n"

1. Start a new worksheet. Move the focus (selected cell) to C3.
2. Begin recording the macro (Tools/Macros/Record New Macro).



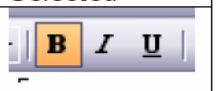
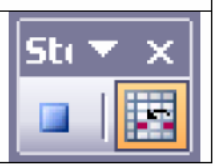
   Fill in ONLY the "n".
   Note 1: Shift "N" and "n" will be different macros.
   Note 2: You may name the macro (optional)
3. **IMPORTANT: When you first start to record a macro, a "Stop Recording" Toolbar should appear (having two buttons).**

Note: The appearance of this toolbar varies with the version of Excel being used.

The first button is pressed to STOP recording (don't do that now).

The second button selected either "absolute mode" or "relative mode". This button functions similarly to the "Bold" button. When "down" it is selected and the mode is "relative". When "up" the button is unselected and the mode is "absolute".

Unfortunately, it is hard to see which mode is active because the icon of the spreadsheet is very large compared to the size of the button.

	Unselected	Selected
Normal/Bold		
Absolute/Relative		

4. For this macro, we wish to have the recording mode be “absolute”. This is the same issue in Excel when $-signs are used in cell formulas (ie., A10 is an absolute address).

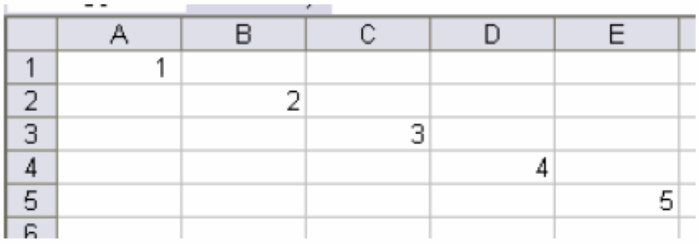
5. ***IMPORTANT: For “unknown reasons” sometimes the “stop recording” toolbar will fail to appear when you start recording a new macro.***

   What has happened is that the “stop recording toolbar” has gotten “unchecked” in the “View/Toolbars” menu. If this happens, you should stop recording the current macro (delete it later) by selecting “Tools/Macro/Stop Recording” and then selecting”
   “View/Toolbars/Customize/[x] Stop Recording”

   This will cause the “stop recording” toolbar to appear. It is suggested that you drag the “stop recording” toolbar into the standard toolbars and leave it permanently docked there (it will always be visible and in a convenient location).

   Note that the “stop recording” toolbar does not appear in the “toolbar list” unless you are actually recording a macro. It ALWAYS appears in the “customize list”.

6. Follow these typing directions exactly
   Move to A1 and enter “1”
   Move to B2 and enter “2”
   Move to C3 and enter “3”
   Move to D4 and enter “4”
   Move to E5 and enter “5”
   Move to F6.
7. Press the stop recording button.

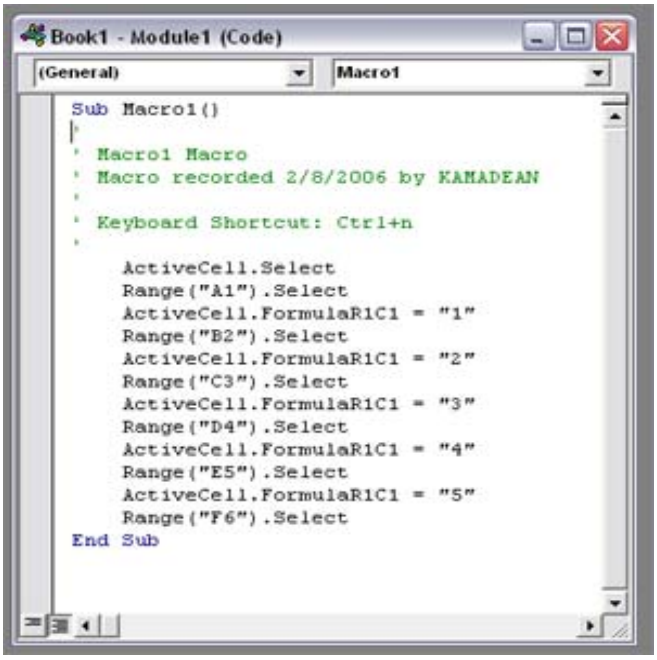


	A	B	C	D	E
1	1				
2		2			
3			3		
4				4	
5					5
6					

8. Test your macro by going to a new sheet (Sheet2) and pressing “Cntl-n”

Try the macro in several other places (move your cursor to different locations before pressing “Cntl-n”. Notice that wherever you locate the cursor, the data will be created in the locations we originally put the data (A1-E5).

Let’s edit the macro to see what we recorded. Tools/Macro/Macros/Edit



Notice that we can change any of these items and our macro will be appropriately edited. Hence, you can make mistakes and “fix them” or make the macro do new things.

In this case, change the values to 0, 0, 0, 0, 0. Test your macro. You could also change the addresses to A5, B4, C3, D2, E1. You could also type additional lines to add more “typing” or delete some lines.

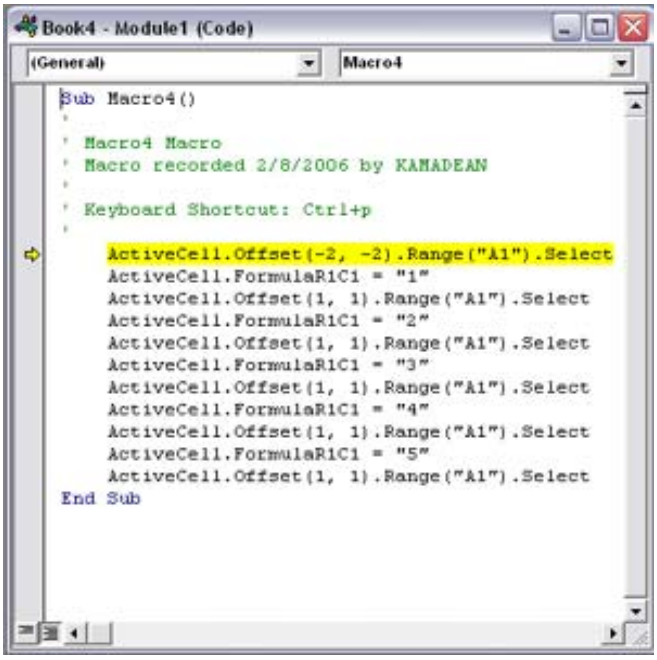
Relative Mode: This time, do the same thing EXCEPT when you start the recording, make sure “relative” mode is selected.

Start in cell C3 again.
Start to record your macro.
Name this macro “Cntl-p”
Switch to relative mode.
Do the same typing
End the macro.

Test your macro as before but this time CAREFULLY.

Return to C3 and try your macro.
Go to E5 and try your macro.
Got to A1 and try your macro.

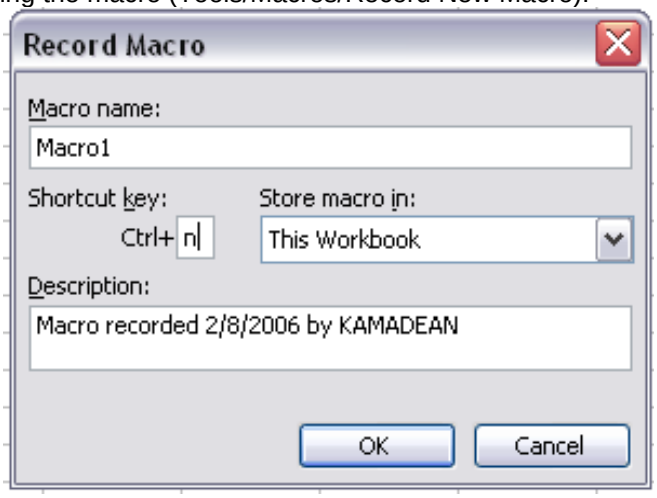
Notice that the latter case produces an error since the macro instructs the computer to go to a position that doesn’t exist.



Macro 2: Making a Boilerplate (Name Template)

CLOSE your current workbook.
Start a new workbook.
Make the current cell be A1.

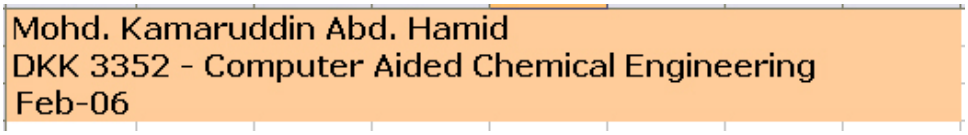
1. Make sure the following are selected:
   - regular (NOT BOLD)
   - Arial
   - 10 point font size
2. Begin recording the macro (Tools/Macros/Record New Macro).



IMPORTANT: For this example, use the RELATIVE MODE.

3. Type your name and other information (similar to below)
   Mohd. Kamaruddin Abd. Hamid
   DKK 3352 - Computer Aided Chemical Engineering
   February 2006
4. Highlight cells A1:H3 and (1) change the background color to “tan” (2) make the characters “bold” (3) change the font size to “12” and change the font to “Verdana”.

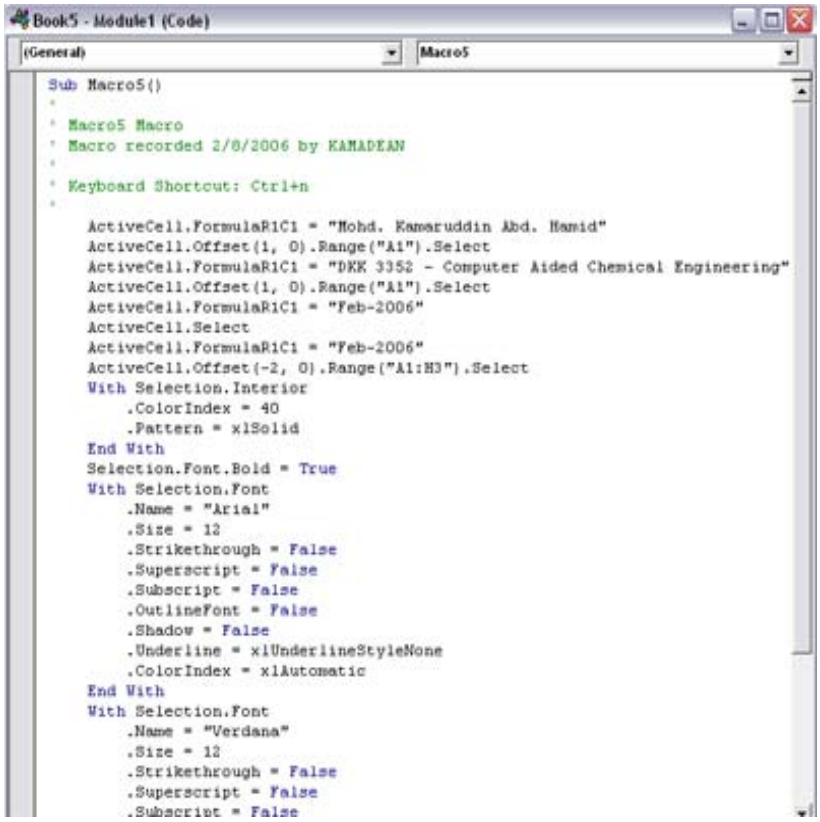
5. Click the “STOP RECORDING” button.



6. Test your macro by going to a new sheet (Sheet2) and pressing “Cntl-n”

Try the macro in several other places. Notice that wherever you locate the cursor, the data will be placed in that location (this is a relative reference). If this doesn't work, you did not get the relative/absolute mode setting correct.

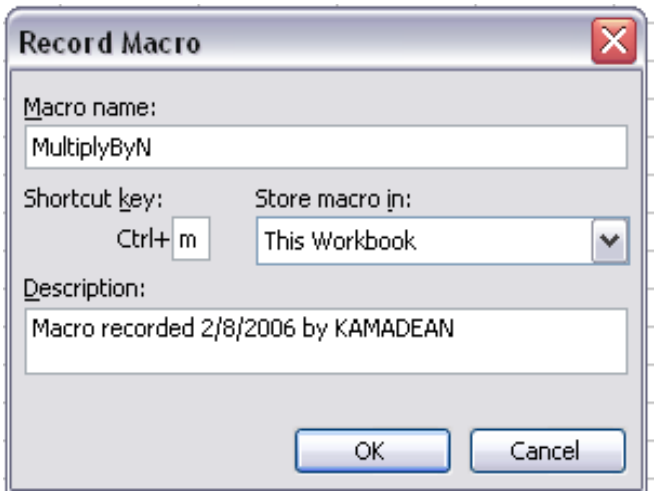
Let's edit the macro to see what we recorded. Tools/Macro/Macros/Edit “selected macro”



Notice that we can change any of these items and our macro will be appropriately edited. Hence, you can make mistakes and “fix them” or make the macro do new things.

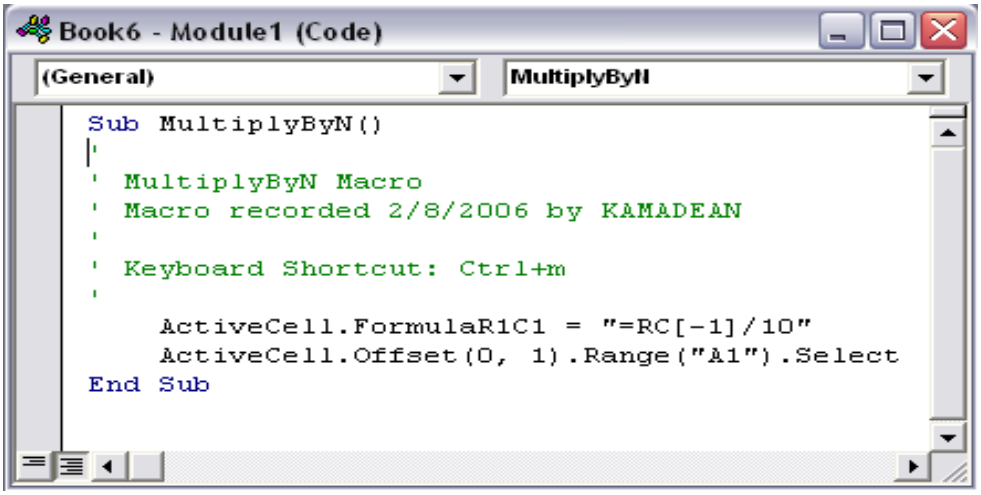
Macro 3: Operating On a Cell's Value

In this macro we will enter a number in a cell based on the contents of the cell immediately to the left of the cell, namely, we will multiply the value by 10.



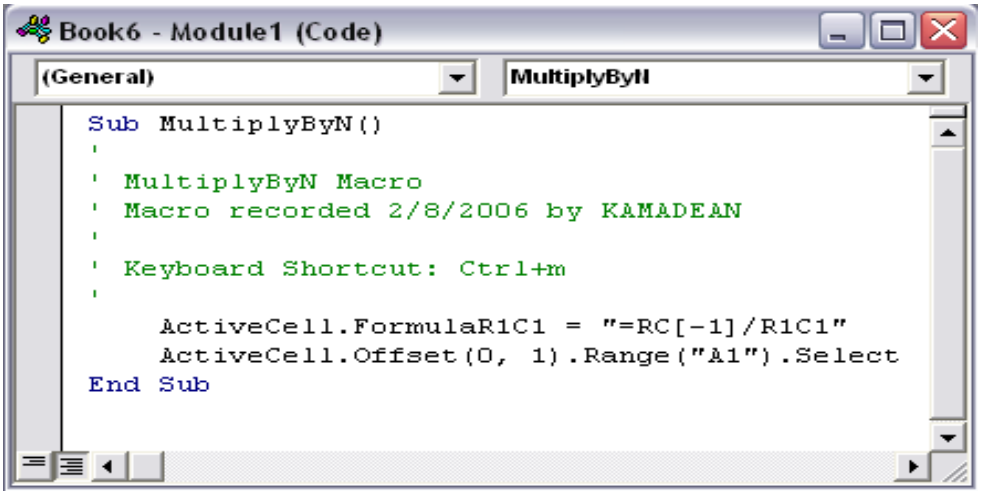
1. Start on a new “Sheet”. Type the number “123.456” in cell D4.
2. Locate the cursor in cell E4.
3. Start recording a macro (Cntl-m)
4. Type the following in E4: =D4/10
5. Use the cursor keys to move the cursor to the cell just to the right of E4 (namely F4).
6. Stop the recording.
7. Repeated pressing Cntl-m should cause the adjacent cells to contain the desired values.

123.456 12.3456 1.23456 0.123456 0.012346 0.001235 0.00012 3 1.23E-05



Let’s alter the macro to take the value stored in A1 and use it in place of “10”. Notice that the value in A1 is refered to by the general notation R1C1. We could have gotten this same result if when we were recording the macro we pressed the F4 key after pointing to the cell A1.

Notice that the numbers are getting bigger now because we have A1=0.1 (dividing by 0.1)!



Programmed Macros (VBA)

In this section we consider writing macros “from scratch” (that is, by creating them with the VBA Editor rather than recording them).

Macros are stored in Modules in a fashion similar to functions. The main difference is that macros perform an action (do something) rather than return a value (calculate something). Also, macros do not have “arguments”. Functions that do not return a value are called “subprograms”.

Let’s begin by writing a macro that makes the background color of the cells in the range A1:D6 “yellow”.

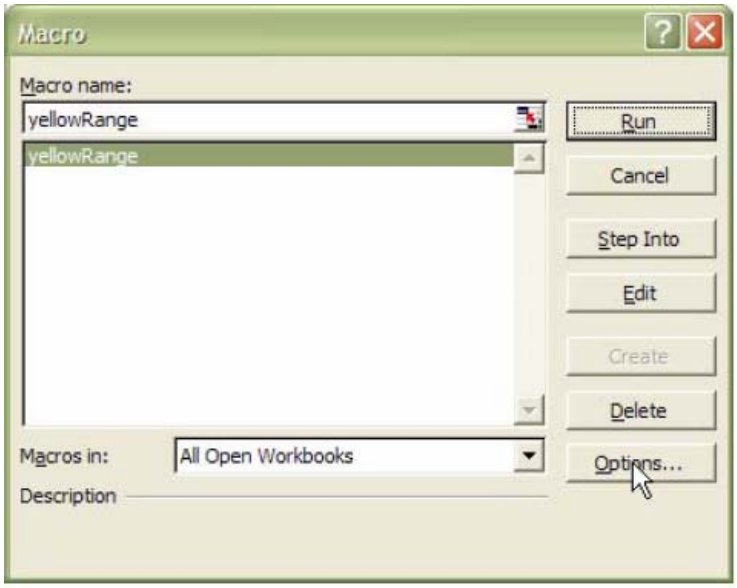
Macro 4: Yellow Range

Start a new worksheet with the “current cell” being A1.

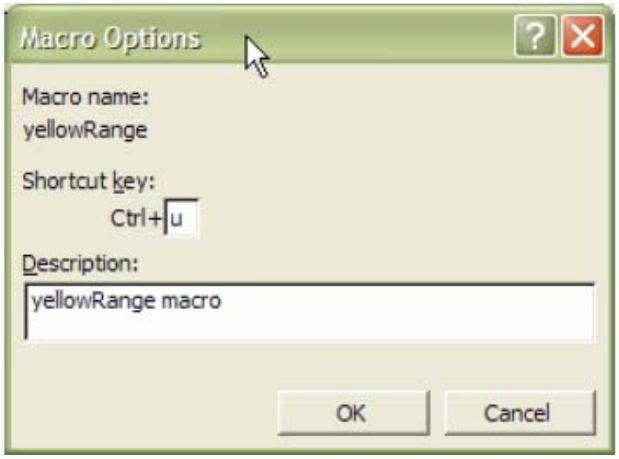
1. Add a new subprogram called “yellowRange” to Module1. If you do not have other files open, you should need to Insert a Module before you start the subprogram (Insert/Module from the VBA editor “Insert” menu).

```
Public Sub yellowRange()
Range("A1:D6").Interior.ColorIndex = 6
End Sub
```

2. Return to the worksheet and open the Tools/Macro/Macros menu.



We can run the macro (without a hotkey) by pressing “Run” …



… or we can select the “Options…” menu and assign the macro to a key (Cntl-u for example).

In this example, ColorIndex is a predefined set of colors numbered 1-56

ColorName	ColorIndex	RGB Values
Black	1	0,0,0
White	2	255,255,255
Red	3	255,0,0
Green	4	0,255,0
Blue	5	0,0,255
Yellow	6	255,255,0
Magneta	7	255,0,255
Cyan	8	0,255,255
(more)	9-56	

3. We can also choose colors based on their RGB values directly.

Change your macro to the following:

```
Public Sub yellowRange()
Range("A1:D6").Interior.Color = RGB(127, 63, 0)
`  This is some kind of brown color
End Sub
```

4. We can also “select” (leave highlighted) the range of cells by adding the following:

```
Public Sub yellowRange()
Range("A1:D6").Interior.Color = RGB(127, 63, 0)
Range("A1:D6").Select
End Sub
```

5. In order to operate on the cells WE have selected (instead of knowing which cells are in our range beforehand, use the following:

```
Public Sub yellowRange()
Range(ActiveWindow.RangeSelection.Address).Interior.ColorIndex = 4
End Sub
```

Make sure you have some cells selected before you run the macro again. You should make the cells “Green” by this macro.

6. In order to put a value in the selected cells use the following:

```
Public Sub yellowRange()
Range(ActiveWindow.RangeSelection.Address).Interior.ColorIndex = 4
Range(ActiveWindow.RangeSelection.Address).Value = "DKK3352"
Range(ActiveWindow.RangeSelection.Address).Font.Size = 6
Range(ActiveWindow.RangeSelection.Address).Font.Color = 1
Range(ActiveWindow.RangeSelection.Address).Font.Bold = True
End Sub
```

This will put the characters “DKK3352” in green cells in a very small red font with bold turned on.

7. In order to read a value out of a cell and put it in others, use the following:

```
Public Sub yellowRange()
Range(ActiveWindow.RangeSelection.Address).Value = Cells(2, 4).Value
'   Cells(2, 4) refers to "D2"
End Sub
```

8. In order to operate on values in a cell, it is easier to assign them to “local variable” and then perform the calculations and assignment.

```
Public Sub yellowRange()
  Dim n1 As Single
  Dim n2 As Single
  Dim n3 As Single
  n1 = Cells(2, 4).Value  ' This is D2
  n2 = Cells(2, 5).Value  ' This is E2
  n3 = n1 + n2            ' Sum the two values
  Range(ActiveWindow.RangeSelection.Address).Value = n3
End Sub
```

9. In this example, we read the value from A1 and write variations of it in A2, A3, and A4.

```
Public Sub yellowRange()
  Dim n1 As Single
  n1 = Cells(1, 1).Value  ' This is A1
  Cells(2, 1).Value = n1
  Cells(3, 1).Value = 2 * n1
  Cells(4, 1).Value = 3 * n1
End Sub
```

	A	B
1	8	
2	8	
3	16	
4	24	
5		
6		

Programming in Excel with VBA

Introduction

Hiding “behind the scenes” in Microsoft Office products is a powerful programming environment that most users are unaware of. This programming language is Visual Basic for Applications (VBA). Thus far in the course we have made use of two VBA capabilities: (1) user defined functions and (2) user defined macros (subprograms). VBA is closely related to, but not identical to, Visual Basic. Persons who are proficient with VBA will have little difficulty becoming proficient with Visual Basic.

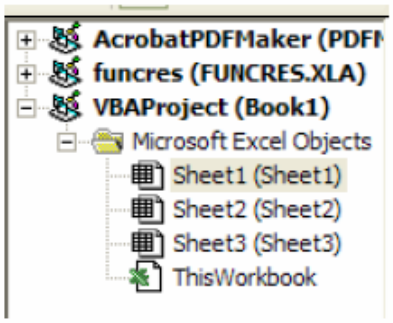
Visual Basic for Applications (VBA) Overview

Although VBA is a complete programming environment there is one significant limitation and several operational issues.

The limitation is that one cannot develop standalone modules (executable programs) that can be distributed. Code developed using VBA in Excel (for example) is a part of the Excel worksheet and must be run from Excel. This can cause problems if one distributes a “program” developed under one version of Excel and one attempts to run it under a different version. Normally the differences are small between VBA versions but even a small difference can cause major problems. This also means that “programs” developed with “today’s Excel” may not run properly when one upgrades one’s software to “tomorrow’s Excel”. Generally code which uses straightforward features (like using the “SIN” function or the “TODAY” function) will cause little problems. Most problems arise from use of functions and features much closer to the “operating system”.

Since we have already used the VBA Editor, we will not detail how to get into it. Once in the editor one sees three or more “panels” where different activities take place:

- The most obvious panel is the Development Panel (or development area) where code is written and forms are developed. Forms are similar to “menus” and similar “popups” where data is entered or results are displayed.
- The upper panel on the left is called the Project Panel. In the area one selects what will be displayed in the Development Panel. By default there are four items in a new worksheet.



Material developed for the “sheet1” development area are available ONLY to sheet1. Items to be common (shared) between ALL worksheets should be associated with the “ThisWorkbook” development area.

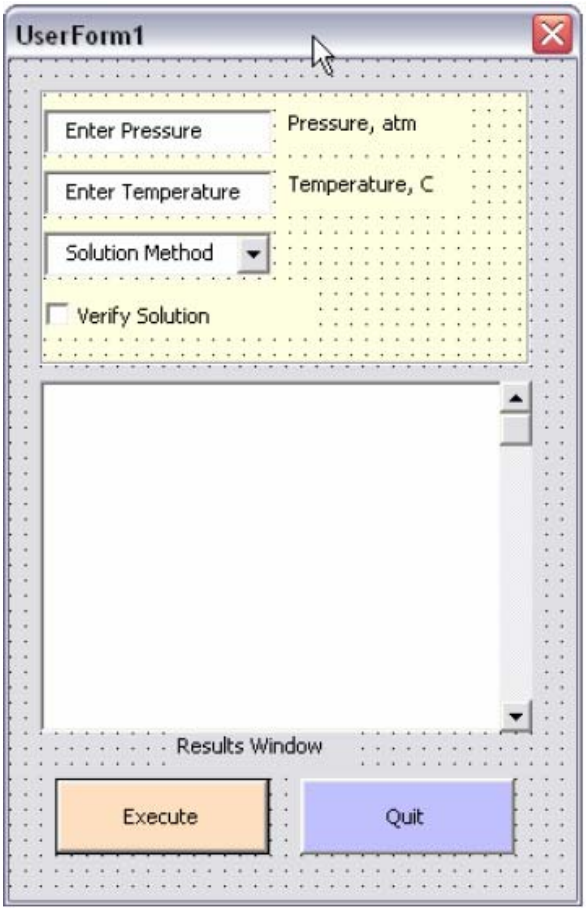
- The bottom left panel is called the “Properties Panel” which provides information about the properties of the currently selected item (more about this later).
- Another frequently used panel is the “Immediate Window (Panel)” which is not displayed by default. Use the “View” menu to show/hide this window or press

Cntl-G. One can immediately evaluate or cause events to happen by typing in this window.

Projects, Forms and Modules

In VBA, a **project** is a collection of components that make up a complete program. Projects typically contain:

- one or more **forms** to collect (input) and present (output) information.



- one or more **modules** that contain variable definitions (pi=3.141592) and program code (functions, subprograms, etc).
- a single **workbook** shown as “This Workbook” in the project panel.
- one of more individual **sheets** (again used to collect and present information).

Visibility Issues

Proper use of these functional components is important in that the location where data and code imposes limits (controls) over where the data and code can be used.

Program elements stored in a **module** are accessable from any project source. As we have seen, having several Excel xls files open at the same time makes all the functions written in any **module** available to any of the **workbooks** or **sheets**. Generally program **functions** and **subprograms** are written in **modules** rather than in specific sheets.

Program elements stored in the **workbook** are available to all sheets but not to other elements of the **project**.

Program elements stored in a **sheet** are only available within that **sheet**. This means that one can have different functions with the same name used in different sheets. Sheet1!Fun1, Sheet2!Fun1, etc.

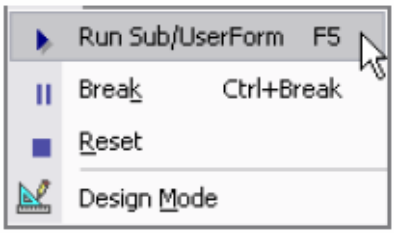
Example Subprogram

Let's begin by writing a subprogram procedure called "prog1" which demonstrates a few structural elements of the VBA language:

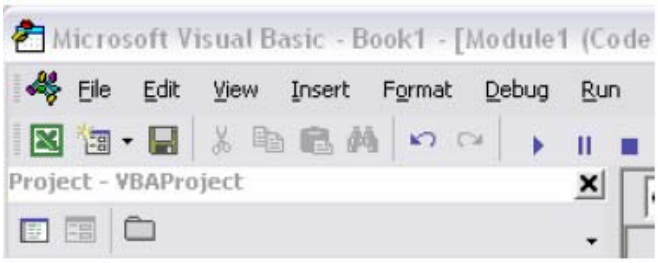
Starting with a blank worksheet, Insert a new module (Module1) and add the following subprogram to the module.

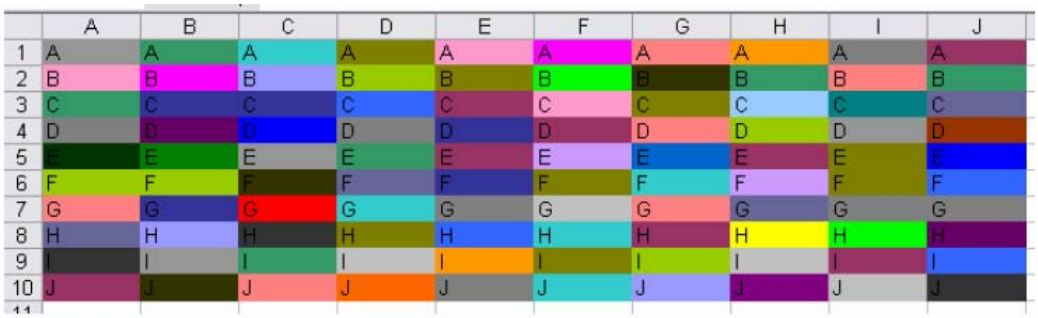
```
Public Sub prog1()
  firstletter = "A"
  For i = 1 To 10
    For j = 1 To 10
      red = Int(256 * Rnd())
      blue = Int(256 * Rnd())
      green = Int(256 * Rnd())
      Cells(i, j).Value = Chr(i + Asc(firstletter) - 1)
      Cells(i, j).Interior.Color = RGB(red, green, blue)
    Next j
  Next i
End Sub
```

Press F5 or use the Run / Run Sub/UserForm to excute the subprogram prog1.



Click on the "Xls" icon to see the results. Note that your code will be executed in Sheet1 since it is the default sheet.





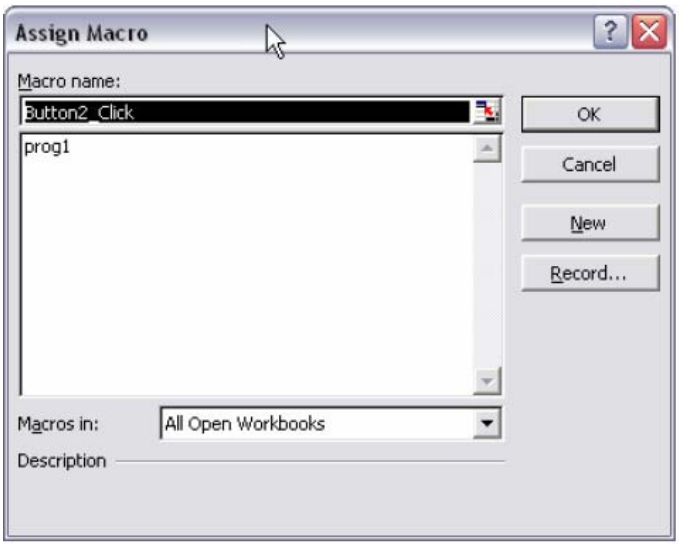
Now go to Sheet2 and execute the subprogram by selecting Tools / Macro / Macros / Run or simply press Alt F8. Do this repeatedly to see the random nature of this subprogram.

To simplify the process of executing our prog1() subprogram we will add a button and "attach it" to the prog1 code.

Begin by returning to Sheet1 (this is where we will put our Button).

Click the "Add Button" button from the Forms toolbar. (If this menu is not available, you can add it from View Toolbars.)

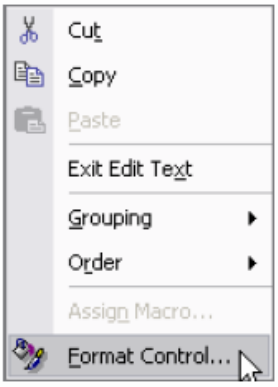
Drag over the region where you want the button to appear or just click to get the "default size".



Associate clicking the button with our prog1 subprogram. (Select and click OK).

Resize the button if you desire.

Right-Click and “Edit Text” to change the caption for the button.



You can also change other “appearance issues” for the button by using the Format Control menu.

Program: Generating Unique Random Numbers

Program Statement: Write a procedure that develops a set of “m” integer numbers each of which is in the range n1…n2 with no duplicates. Include logic to detect invalid situations.

For example: m=5 n1=10 n2=20
Solution: 13, 11, 19, 20, 18

Another: m=10 n1=5 n2=10
Solution: NONE (invalid)

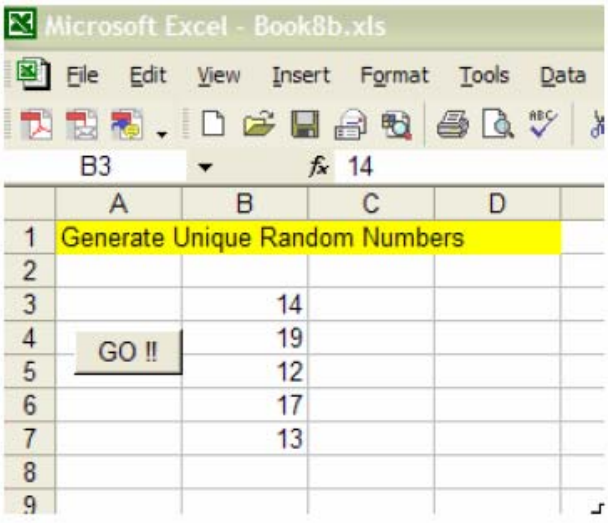
Another: m=55 n1=0 n2=100
Solution: TOO BIG FOR DIMENSIONING (invalid)

(Program follows on next page)

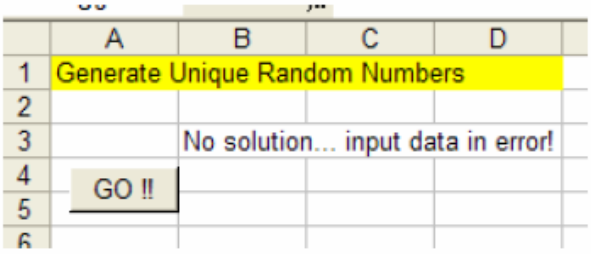
Note: A button was added to call the subprogram uniqueRandom().

```
Public Sub uniqueRandom()
Dim m As Integer
Dim n1 As Integer
Dim n2 As Integer
Dim maxN As Integer
Dim nRand As Integer
Dim found As Integer
Dim i As Integer
Dim IsUnique As Boolean
Dim IsInvalid As Boolean
Dim soln(50) As Integer

' Acquire Problem Data
m = 55
n1 = 10
n2 = 20

' Determine If Valid Case
maxN = n2 - n1 + 1
IsValid = (m <= maxN) And (m <= 50)

' Process Valid Case or Report Error
If IsValid Then

  ' Determine Solution
  found = 1
  soln(1) = GenerateRandomNumber(n1, n2)
  Do While found < m
    nRand = GenerateRandomNumber(n1, n2)
    IsUnique = True
    For i = 1 To found
      If soln(i) = nRand Then IsUnique = False
    Next i

    If IsUnique Then
      found = found + 1
      soln(found) = nRand
    End If
  Loop

  ' Report Solution
  For i = 1 To m
    Cells(2 + i, 2).Value = soln(i)
  Next i

Else
  ' Report As Invalid
  Cells(3, 2).Value = "No solution... input data in error!"
End If

End Sub
Public Function GenerateRandomNumber(a As Integer, b As Integer) As Integer

  Dim range As Integer
  Dim floatran As Single

  range = b - a
  floatran = range * Rnd()
  GenerateRandomNumber = Int(a + floatran)

End Function
```

Here is the output from a valid case (m=5, n1=10, n2=20):



Here is the output from an invalid case (m=55, n1=10, n2=20):



Getting Back the Power of Excel

One of the initial frustrations of using VBA is the “lack” of corresponding functions and procedures that one may be used to from using Excel itself.

Remembering that each product (Excel and VBA) are separate independent products one should not be too surprised that each has its own unique capabilities. For example, Excel is a “number cruncher” and has hundreds of mathematical and statistical functions. VBA, on the other hand, is a general purpose computing language whose audiences is much broader than “number cruching engineers”. Therefore, in order to access the Excel functions we have gotten used to, we need to appropriately “reference them”.

Simple Excel functions are fairly easy to “get to”. For example, consider the “Sum” function that might be used to add the values in the range A1:C5.

You can call most standard Excel worksheet functions in VBA procedures using the syntax:

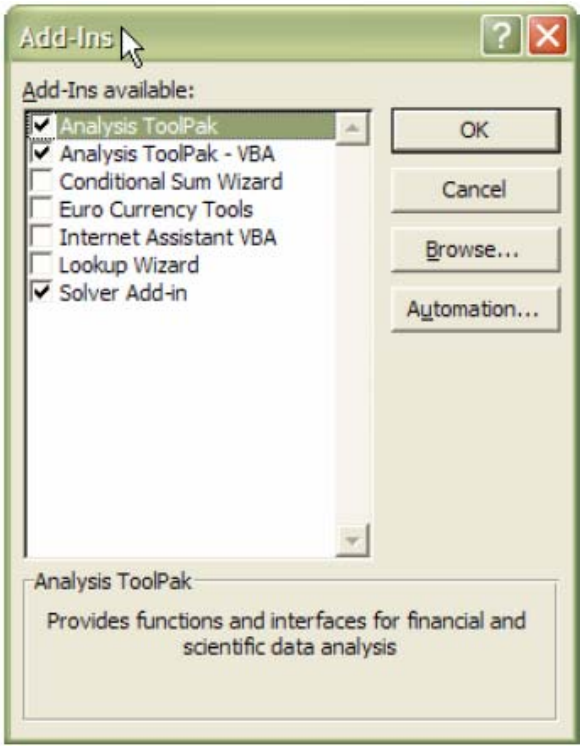
```
Result = Application.WorksheetFunction.Sum(Range("A1:C5"))
```

where Application.WorksheetFunction allows VBA to locate the function “Sum”. We also need to “adjust” the argument information.

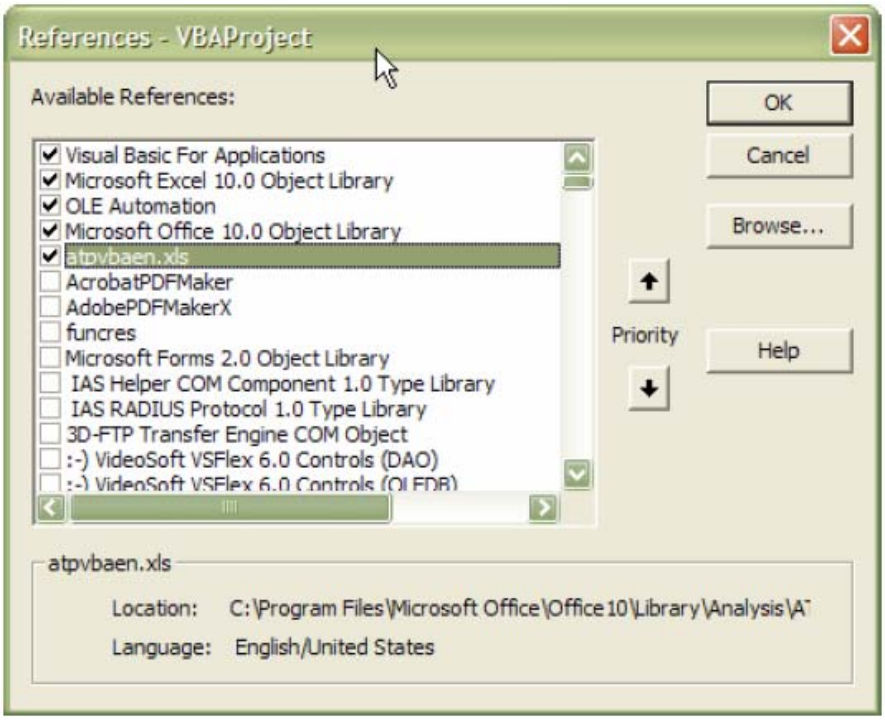
On the other hand, other functions are “less standard” and may belong to various installed programs (add-ins).

For example, suppose we wish to round a number in VBA to the closest “2” using the MROUND function. Since this function comes from the Analysis Toolpak we need to take two steps to be able to use that function.

1. From the Excel window, activate the Analysis ToolPak (VBA) from the Add-Ins menu (note this is a separate add-in from the ATP for Excel).



2. From the VBA window, activate the Analysis ToolPak VBA (ATPVBA) in the Tools/References menu. That item is atpvbaen.xls (EN=English)

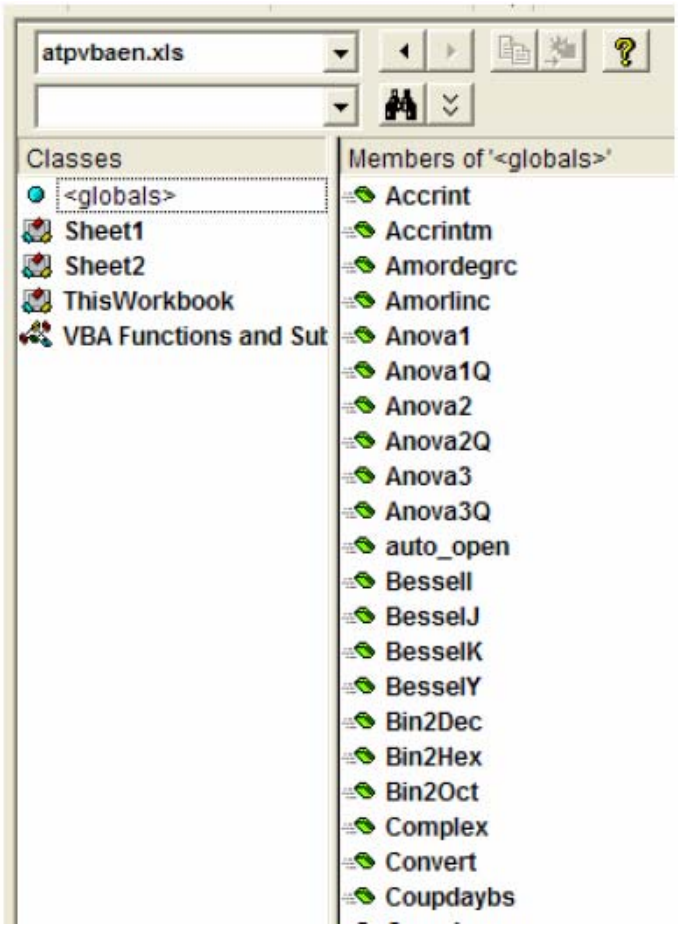


You will now be able to access the **mround** and other math functions in the following fashion:

```
Sub DemoRound
   Dim result As Single
   Dim x as Single
   x = 3.2
   result = mround(x)
   Cells(2,2) = result
End Sub
```

Viewing the Members of the ATPVBA (or other Libraries)

In order to view the procedures available one can view the members (Objects) that are currently loaded in VBA. Simply press F2 (or use the View/Object Browser). By default, one sees “<All Libraries>” but we can select “atpvbaen.xls” to see what we are interested in:



We can see our old favorites such as BesselJ as well as MRound and andBetween.

Another example, suppose we wish to employ the following “Excel” formula when we are using VBA:

=norminv(0.3, 70, 10)

Suppose we have a normally distributed variable (Gaussian distribution) where the average exam grade is 70 and the standard deviation of grades is 10 (that is, 95% of students have a grade between 50 and 90). 30% of students have “what grade or less”? The answer is: 64.756

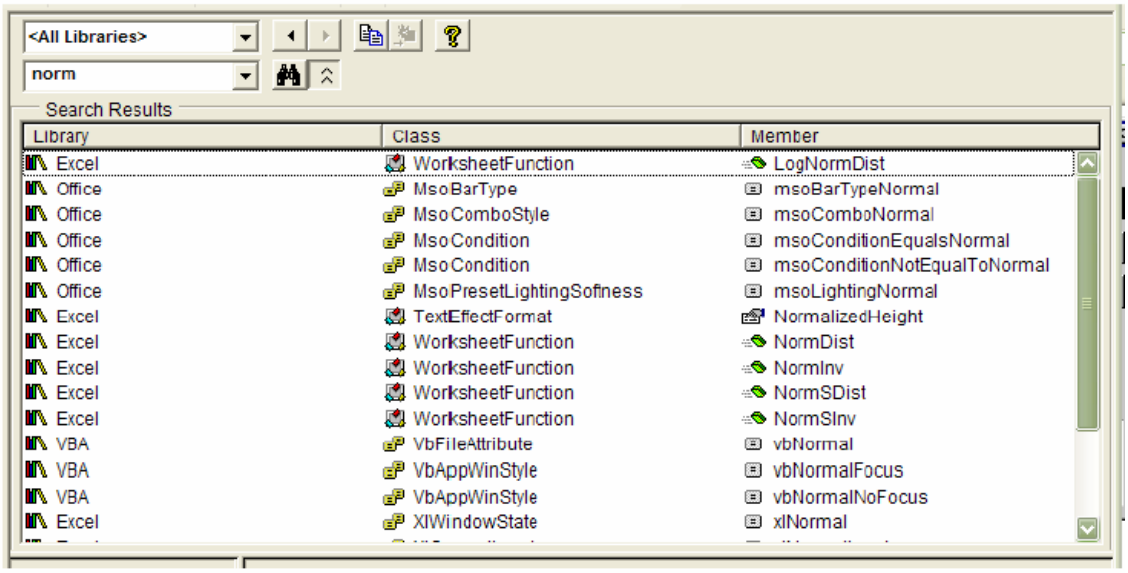
```
Sub test()
  Dim result As Variant
  result = Application.WorksheetFunction.NormInv(0.3, 70, 10)
  Cells(3, 1) = "Answer:"
  Cells(3, 2) = result
End Sub
```

	A	B
1		
2		
3	Answer:	64.756
4		
5		

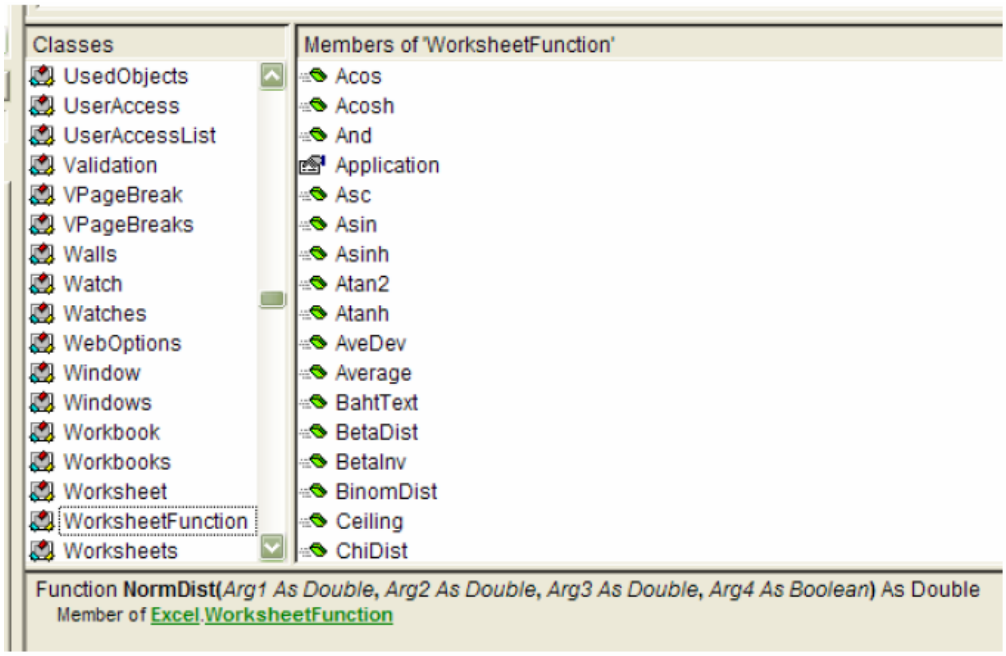
Note that we can also use the Object Browser to search for items we may not be sure of the syntax or location of.

For example, we will search in “All Libraries” for the string “norm”.

Use the “binoculars” to initiate the search.



Notice that “norm” matches a lot of different “members” (LogNormDist, NormDist, NormInv, etc). These familiar (sort of) functions are members of the “WorksheetFunction” group. Clicking on the “Class” WorksheetFunctions will show ALL members in the “bottom” window as shown below:



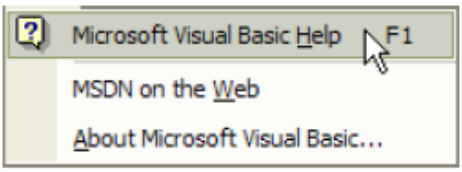
Here are the majority of the “expected” functions we apply that are not in the Analysis ToolPak VBA.

VBA Language Elements

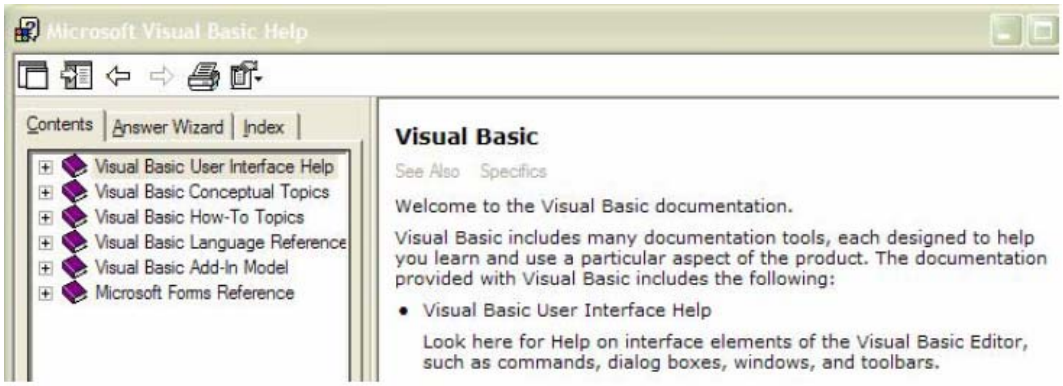
Before we “get going”, let’s talk about “getting help”. Buried within VBA is a plethora of information about VBA topics. Much of this doesn’t come to the surface when you do a “traditional help” inquiry (that is, typing a question in the search window). However, if you understand the layout of the help system, usually you can find the information you seek with a minimum of effort.

Starting help:

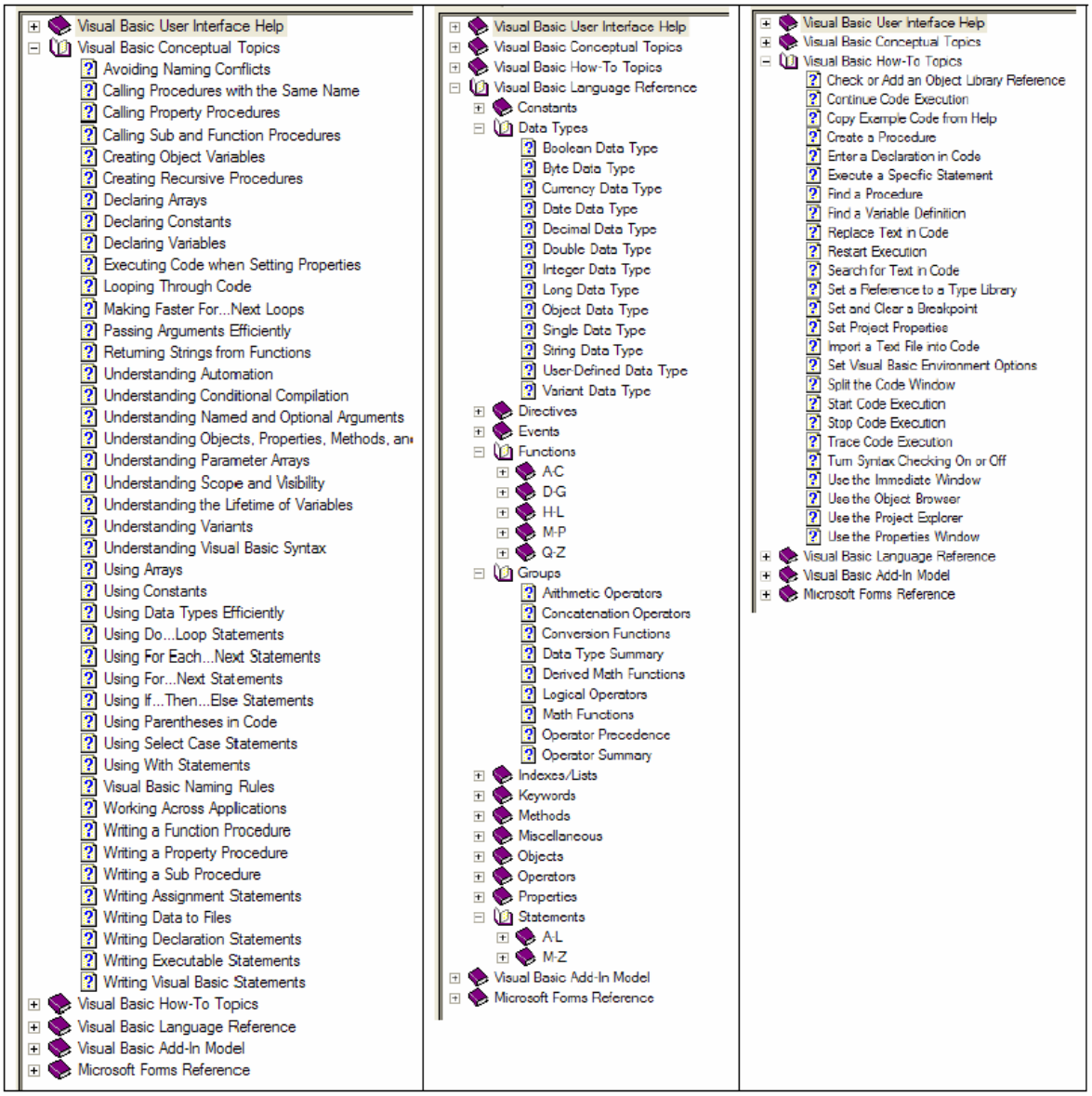
From within VBA, pull down Help/Microsoft Visual Basic Help. *Make sure you expand the “help window” to be able to easily see the text and examples.*



The help is divided into sections. The most useful sections are “Conceptual Topics”, “Language Reference” and “How-To Topics”.



Time spend “reading” can save much time debugging later !!



Data Types

The following is a summary of basic variable types supported in VBA.

Type	Memory	Type-Declaration Character	Description
Byte	1 byte	none	Positive whole number ranging from 0 through 255 that can be represented as a binary value.
Boolean	2 bytes	none	**True** or **False** value
Integer	2 bytes	%	Whole numbers ranging from -32,768 through 32,767.
Long (*long integer*	4 bytes	&	Whole numbers ranging from -2,147,483,648 through 2,147,483,647.
Single	4 bytes	!	Single-precision floating-point number (with decimal points) ranging from -3.402823E38 to 3.402823E38.
Double	8 bytes	#	Double-precision floating-point number (which is more precise for very large or very small numbers) ranging from -1.79769313486232E308 to 1.79769313486232E308.
Currency	8 bytes	@	Large numbers between -922,337,203,685,477.5808 and 922,337,203,685,477.5807 (15 digits to left of decimal and 4 digits to the right of the decimal).
Date	8 bytes	none	Represents dates from January 1, 100 through December 31, 9999.
Object	4 bytes	none	An instance of a class or object reference.
String	10 bytes + 1 byte per char	$	Series of any ASCII characters.
String (*fixed-length*)	length of string	none	Series of any ASCII characters, of a pre-defined length.
Variant	min 16 bytes	none	Any kind of data except fixed-length String data and user-defined types.

Declaring Variables

What are Variables?

Variables are used to store information temporarily. As a program runs, it holds values temporarily in memory. Variables are names that the program associates with specific locations in memory. The value to which the program refers in these areas can change throughout a session of program operation.

Each variable has a specific ***type*** that indicates how much memory the data requires and the operations that can be performed on that kind of data.

The **Dim** (stands for *dimension*) statement is used to declare variables and allocate storage space. It may appear in a *General Declarations* section at the top of a code module –orimmediately following a procedure declaration. For example ...

```
Sub SampleCode()
Dim MyNumber as Integer
```

```
' Other program statements go here
End Sub
```

Variables declared within a procedure are not available to other procedures and they only retain values for the life of that procedure. This refers to a variable's ***scope***. Variables declared at a module-level (or within a form's General Declarations section) are available to all procedures within that module or form -and- they continue to retain assigned values for the life of the program. However, these variables are not available to procedures outside of the module in which they are declared. One alternative to the *Dim* statement is the **Public** keyword, which expands the scope of the variable to make it available to other procedures outside it's own module or form. It's a better practice to use the narrowest possible scope for your variables.

> *Note: Other approaches to scope involve **Global**, **Private**, and **Static** declaration. Also, very similar to variables are **Constants**. You may want to learn about how to use these at some point.*

Explicit declaration (in contrast to *implicit* declaration) means requiring the declaration of a variable BEFORE you use it. By default, you are NOT forced to declare a variable, but undeclared variables are problematic. Using the ***Option Explicit*** statement forces you to declare every variable. This also helps you catch errors. You should place the Option Explicit statement at the head of the General Declarations section (at the top) in your modules.

> *Note: You can ensure that the Option Explicit statement will be added automatically by checking* **Require Variable Declaration** *in the* Editor *tab of the* Tools | Options *dialog of Office's integrated development environment (module) interface.*

A *type clause* (i.e., *... as Integer* -or- *... as String*) or appending a *type-declaration character* is optional, but it is good practice for this to always be explicitly assigned by you. Valid data types are Byte, Boolean, Integer, Long, Currency, Single, Double, Date, String (*variable-length text*), String * *length* (*fixed-length text*), Object, Variant, a user-defined type, or a specific object type. The compiler will default to the type *Variant* for a variable that does not have a type specified. A Variant behaves like a chameleon, as it can become whatever type is required for the data assigned to it. This is usually undesirable because it is not memory efficient, it slows performance as VBA has to determine what type of data the Variant represents, and it can result in problematic type conflict errors. Variants DO serve a useful purpose, however, when it is known that the variable type cannot be determined - such as when capturing freeform entry by users.

What are Arrays?

An *array* is a group of variables, usually of the same data type, arranged contiguously in memory. These variables share a common name. Each variable within an array is called an **element** and you use a number (an index) to tell them apart. An **index** simply means an identifying number.

In VBA there are two categories of arrays:

- ***fixed-size** array* - always remains the same size
- ***dynamic** array* - size can change at run-time

For now, we'll focus on the fixed-size array. You declare a fixed-size array just as you would a regular variable, except that you also need to specify the number of elements in the array. This is done in parentheses following the name of the array as you declare it. For example:

```
Dim MyOneDimensionalArray(10)
```

or

```
Dim MyTwoDimensionalArray(10,10)
```

In these examples, only the upper bound of the arrays have been specified. The default lower bound is 0 (for a zero-based array). However you can change the default lower bound so the index for the first element will be 1 by placing an **Option Base 1** statement in the General Declarations section.

Another way to specify a lower bound is to provide it explicitly using the **To** keyword. Here is an example of explicitly declaring the upper and lower bounds for a 50 element array beginning after an index of 100:

```
Dim MyArray(101 to 150)
```

A one-dimensional array is useful for storing a series of values. It may help to think of it as a list. For example, you may declare an array of 7 string variables so that each element corresponds to the name of a different day of the week.

```
Sub OneDimensionalArray()
Dim a as Integer
Dim aDayOfWeek(7) as String

' Initialize elements of one-dimensional array

aDayOfWeek(1) = "Monday"
aDayOfWeek(2) = "Tuesday"
aDayOfWeek(3) = "Wednesday"
aDayOfWeek(4) = "Thursday"
aDayOfWeek(5) = "Friday"
aDayOfWeek(6) = "Saturday"
aDayOfWeek(7) = "Sunday"

' Loop through the array and display each of ' it's elements

    For a = 1 To 7
        MsgBox aDayOfWeek(a)
    Next a

End Sub
```

A two-dimensional array is similar to a table in a database. One dimension represents the number of rows in the table, while the other specifies how many columns (fields) you are defining. Therefore, each element in a two-dimensional array can be referred to by its X and Y coordinates. You can efficiently process a multidimensional array by using nested loops.

```
Sub TwoDimensionalArray()
  Dim c as Integer
  Dim r as Integer
  Dim Counter as Integer
  Dim aMatrix(3, 3) as Integer


  For c = 1 To 3
    For r = 1 To 3

    ' Increment a counter variable

      Counter = Counter + 1

    ' Populate an element of the array with counter value

      aMatrix(r, c) = Counter
    Next r
  Next c

End Sub
```

A three-dimensional array is like a cube. Most arrays are only one- or two-dimensional. Although VBA allows you to create arrays of up to 60 dimensions, a need for more than a three-dimensional array would be rare. And, because VBA allocates memory space for every possible element in a declared array, regardless of whether or not you actually store a value in that element, avoid declaring an array any larger than is necessary. Always remain mindful of the size of your arrays, since they can have significant memory requirements.

Arrays are so powerful, as they makes it so much easier to work with groups of variables. This enables you to develop shorter and simpler procedures, because you can use loops that deal efficiently with your variables.

Programming Structures

If...Then...Else Conditional Statements

Decision Structures in VBA test conditions and then execute code segments depending on the test's results. This includes a familiar **If...** statement.

There are various forms of the **If...** construct. A single-line syntax is often used when only one statement is to be executed upon a successful test condition. It looks like this:

If *condition* Then *statement1* [Else *statement2*]

The example below will set the SalesTax variable to .0825 if the condition is True (CustomerState is "TX"); otherwise, the SalesTax variable will be set to 0 (zero).

```
If CustomerState="TX" Then SalesTax=.0825 Else SalesTax=0
```

Notice there's no **End If** used in the single line statement above.

The multi-line (or *block*) syntax will be easier to read if your decision structure has numerous conditions.

If *condition* Then

```
        statement1
[ ElseIf
        statement2 ]
[ Else
        statement3 ]
End If
```

Notice this example uses **ElseIf** to include additional conditions, and it *DOES* require an **End If**.

```
If sin(x)<0 Then
  fun1=-sin(x)
ElseIf sin(x)>0 AND sin(x) <0.5 Then
  fun1=0.5
Else
  fun1=sin(x)
End If
```

In this construct, the program goes through the conditions until it finds one that is True. Even if there are actually multiple True conditions, only the *FIRST* condition that valuates as True will be executed. If none of the conditions are satisfied, there is an optional **Else a**vailable as a *catch all* if you need it.

And, finally, here is one more example that shows how you can also nest conditional statements within other **If...** statements.

```
If CustomerState="TX" Then
    If CustomerCity="Houston" Then
      SalesTax=.0835
    ElseIf CustomerCity="Dallas" Then
      SalesTax=.0865
    Else
      SalesTax=.0825
    End if
ElseIf CustomerState="VA" Then
    If CustomerCity="Virginia Beach" Then
      SalesTax=.0675
    ElseIf CustomerCity="Vienna" Then
      SalesTax=.0685
    Else
      SalesTax=.0615
    End if
Else
    SalesTax=0
End If
```

The same example without indenting. It is nearly impossible to determine how this code will function.

```
If CustomerState="TX" Then
If CustomerCity="Houston" Then
SalesTax=.0835
ElseIf CustomerCity="Dallas" Then
SalesTax=.0865
Else
SalesTax=.0825
End if
ElseIf CustomerState="VA" Then
If CustomerCity="Virginia Beach" Then
SalesTax=.0675
ElseIf CustomerCity="Vienna" Then
SalesTax=.0685
Else
SalesTax=.0615
End if
Else
SalesTax=0
End If
```

SELECT CASE Conditional Statements

Decision Structures in VBA test conditions and then execute code segments depending on the test's results. The **Select Case** structure (also called "case statement") is often used in place of complex IF structures.

If you see the need for more than one ElseIf, you should consider a Case structure. An advantage of this construct over the **If...Then...Else** structure is that your code will be more readable and efficient.

The Select Case structure contains only one test expression, in the first line of the structure. Each Case statement in the Select Case structure is then compared against the test expression. If a match is found, the statements accompanying the Case are executed. Only the statement block of the first matching Case expression is executed.

A **Case Else** expression may be added at the end of the structure to execute if none of the Case expressions evaluates to True.

The syntax of the Select Case structure is shown below:

```
Select Case testexpression

    Case expression1
        statement1

    Case expression2
        statement2

    Case expressionN
        statementN

    [ Case Else
        statementX ]

End Select
```

expression : Delimited list of one or more of the following forms: *expression*, *expression*

To *expression*, **Is** *comparisonoperator expression*. The **To** keyword specifies a range of values. If you use the **To** keyword, the smaller value must appear before **To**. Use the **Is** keyword with comparison operators (except **Is** and **Like**) to specify a range of values. If not supplied, the **Is** keyword is automatically inserted.

You can use multiple expressions or ranges in each **Case** clause. For example, the following line is valid:

```
Case 1 To 4, 7 To 9, 11, 13, Is > MaxNumber
```

You also can specify ranges and multiple expressions for character strings. In the following example, **Case** matches strings that are exactly equal to everything, strings that fall between nuts and soup in alphabetic order, and the current value of TestItem:

```
Case "everything", "nuts" To "soup", TestItem
```

Example 1 CASE Construct:

```
Select Case aStudentsGrade

        Case Is > 90
                Grade = "A"

        Case Is > 80
                Grade = "B"

        Case Is > 70
                Grade = "C"

        Case Is > 65
                Grade = "D"

        Case Else
                Grade = "F"

End Select
```

Example 2 CASE Construct:

```
Quantity = InputBox("Enter Quantity:  ")

Select Case Quantity

        Case 0 To 24 : Discount = 0.1
        Case 25 To 49 : Discount = 0.15
        Case 50 To 100 : Discount = 0.20
        Case Is >100 : Discount = 0.25

End Select
MsgBox "Discount: " & Discount
```

FOR ... [EACH] ... NEXT Looping Structure

A looping structure can be used to execute code repetitively. The **For...Next** loop repeatedly executes a line or block of code a specified number of times.

The syntax of the For ... Next structure looks like this:

For *counter* = *start* To *end* [Step *increment*]
codeblock
Next *counter*

You can use any kind of numeric expression for the start and end values, including literals and variables.

The default Step increment is **1**. You can count in descending order by specifying a Step increment of **-1**.

The following example loops through a worksheets collection backward and deletes all but the first worksheet in the workbook.

Example (1) FOR Construct:

```
Sub DeleteSheets()

  Dim pagenum as Integer

' Suspend alerts so the user won't be notified repeatedly
' about removing sheets

  Application.DisplayAlerts = False

  '  Loop through a workbook's worksheets collection

    For pagenum = Workbooks("Book1").Worksheets.Count To 2 Step -1

    '  Delete all but the first worksheet from this file

      With Workbooks("Book1").Worksheets(pagenum).Delete

    Next pagenum

End Sub
```

Example (2) FOR Construct:

```
Sub DeleteSheets()

  Dim r as Integer
  Dim c as Integer
  Dim x(10,10) as Integer

    For r = 1 To 10

      For c = 10 To r STEP -1

        x(r,c)= c

      Next c

    Next r

End Sub
```

An important variation of the For ... Next loop enables us to loop through objects without needing to know how many items are in a collection. This is accomplished with a **For...Each...Next** loop.

It looks like this :

For Each *item* in *collection*
codeblock
Next

This is an efficient way to loop through a collection of objects and a very fast technique especially for looping through a range.

Example (3) FOR Construct:

```
Sub RangeLoop()

  Dim rCell As Range
  Dim MyRange As Range


' Initialize an object variable for a range

  Set MyRange = Worksheets("Sheet1").Range("A1:A30")


' Loop through each cell in the range

  For Each rCell In MyRange.Cells


  ' DO SOMETHING ... example, print cell address
  ' to VBE Immediate (debug) window

    Debug.Print rCell.Address

  Next rCell

End Sub
```

DO ... [WHILE|UNTIL] ... Looping Structure

The **DO ... Loop** structure comes in two flavors: **Do While** and **Do Until**.

The Do While structure will execute a code block as long as a given condition is true.

The Do Until structure will execute a code block up to the point when a given condition becomes true (or as long as the condition is false).

While and *Until* can be used at either the beginning or end of a looping structure, as long as they make a valid expression.

Syntax 1:

```
Do [While|Until condition]
        codeblock
Loop
```

Example DO WHILE LOOP Construct:

```
Do While cnt < 100

  ' DO SOMETHING ... example, print the counter
  ' value to VBE Immediate (debug) window

    Debug.Print cnt

  ' Increment the counter

    cnt = cnt + 1

Loop
```

Syntax 2:

```
Do
        codeblock
Loop [While|Until condition]
```

Example DO LOOP UNTIL Construct:

```
Do

  ' DO SOMETHING ... example, print the counter
  ' value to VBE Immediate (debug) window

    Debug.Print cnt

  ' Increment the counter

    cnt = cnt + 1

Loop Until cnt = 100
```

In these examples, the code block will be executed as long as the counter variable is less than 100. In the first example, the code will be run only if the condition is true. In the second example, the code will be run and then the condition will be evaluated to see if it should loop again.

Using the VBA Debugger

Sooner or later you will encounter the VBA Debugger. Most likely you will make a fatal mistake in your code and when executing it be taken automatically to the debugger (for punishment). A few people will deliberately invoke the debugger to “check out” some element of their code.

Most people consider the debugger to be “complicated” and “scary” and shun it like “influenza” (get your flu shots, please!!). But once you make friends with the debugger, you will find it to be both “essential to your sanity” and “downright indespensible”.

VBA Debugging Tips

If you've worked with VBA for any length of time, you know that it's relatively easy to make errors. In fact, most VBA programmers can do so with very little effort on their part. The process of locating and correcting errors in your VBA code is known as debugging.

Types of Errors

There are basically three types of errors that can occur:

- **Syntax errors:**
  A variety of errors related to entering the code itself. These include incorrectly spelled keywords, mismatched parentheses, and a wide variety of other errors. Excel flags your syntax errors and you can't execute your code until they are correct.
- **Run-time errors:**
  These are the errors that occur while your code is executing. There are many, many types of run-time errors. For example, if your code refers to an object that doesn't exist, you'll get a run-time error. Excel displays a message when there is a run-time error.
- **Logical errors:**
  These are errors that occur through faulty programming. Logical errors may or may not cause a run-time error. In many cases they will simply produce incorrect results.

Debugging your code is the process of finding and correcting run-time errors and logical errors.

Eight Bug Reduction Tips

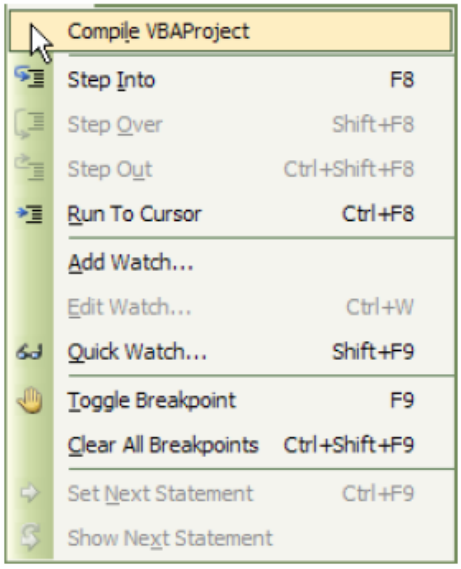
Here are a few tips that will help you keep the number of bugs to a minimum.

1. **Use an Option Explicit at the beginning of your module.** Doing so will require that you define the data type for every variable that you use. It's a bit more work, but you'll avoid the common error of misspelling a variable name. And there's a nice side benefit: Your routines will often run faster.
2. **Format your code with indentation.** I've found that using indentation to delineate code segments is quite helpful. If you have several nested For...Next loops, for example, consistent indentation will make it much easier to keep track of them all.

3. **Use lots of comments.** Nothing is more frustrating than revisiting code that you wrote six months ago - and not having a clue as to how it works. Adding a few comments to describe your logic can save you lots of time down the road.
4. **Keep your subroutines and functions simple.** Writing your code in smaller modules, each of which has a single, well-defined purpose, makes it much easier to debug them.
5. **Use the macro recorder to help you identify properties and methods.** If I can't remember the name or syntax of a property or method, I've found that it's often quicker to simply record a macro and look at the recorded code.
6. **Consider a different approach.** If you're having trouble getting a particular routine to work correctly, you might want to scrap the idea and try something completely different. In most cases, Excel offers several alternative methods of accomplishing the same thing.
7. **Understand Excel's debugger.** Although it can be a bit daunting at first, you'll find that Excel's debugger is an excellent tool. Invest some time and get to know it. I used VBA for quite a while before I took the time to learn how the debugger works (it's well documented in the online help). I spent about an hour learning the details, and I estimate that it has saved me dozens of hours in wasted time.

Invoking the Debugger

From the VBA Editor… pull-down the Debug menu… (for novices!!)… or



…or employ the Debug toolbar… (for REAL programmers!!)… To show the Debug toolbar, select “View / Toolbars / [x] Debug”

The major functions and their explanation (stolen shamelessly from the Help systems)…

Toolbar Buttons

 **Design Mode**
Turns design mode off and on.

 **Run Sub/UserForm** or **Run Macro**
Runs the current procedure if the cursor is in a procedure, runs the **UserForm** if a **UserForm** is currently active, or runs a macro if neither the **Code** window nor a **UserForm** is active.

 **Break**
Stops execution of a program while it is running and switches to break mode.

 **Reset**
Clears the execution stack and module level variables and resets the project.

 **Toggle Breakpoint, F9**
Sets or removes a breakpoint at the current line.

 **Step Into, F8**
Executes code one statement at a time.

 **Step Over, Shift-F8**
Executes code one procedure or statement at a time in the **Code** window.

 **Step Out, Control-Shift-F8**
Executes the remaining lines of a procedure in which the current execution point lies.

 **Locals Window**
Displays the **Locals** window.

 **Immediate Window**
Displays the **Immediate** window.

 **Watch Window, Control-W**
Displays the **Watch** window.

 **Quick Watch, Shift-F9**
Displays the **Quick Watch** dialog box with the current value of the selected expression.

 **Call Stack**
Displays the **Calls** dialog box, which lists the currently active procedure calls (procedures in the application that have started but are not completed).

Using Debugging Tools Effectively

Compile your Code

Once you're written your VBA procedures you should manually compile them to detect errors in syntax. This is done via the Debug/Compile menu. Once you achieved clean compiles without errors, you are ready to undertake the most important step in programming: testing and debugging. Too many programmers become complacent once the code is written and they get a clean compile. These are only the first steps. The most important steps, and the ones programmers like the least, are debugging and testing code.

Immediate Window

The Immediate Window is a window in the VBE in which you can enter commands and view and change the contents of variables while you code is in Break mode or when no macro code is executing. (Break mode is the state of VBA when code execution is paused at a break point (see Breakpoints, below). To display the Immediate Window, press CTRL+G or choose it from the View menu.

In the Immediate Window, you can display the value of a variable by using the ? command. Simply type ? followed by the variable name and press Enter. VBA will display the contents of the variable in the Immediate Window. For example,

```
?ActiveCell.Address
$A$10
```

You can also execute VBA commands in the Immediate Window by omitting the question mark and entering the command followed by the Enter key:

```
Application.EnableEvents=True
```
or
```
Range("A1").Value = 1234
```

The Immediate Window won't let you enter VBA code snippets and execute them together because the Immediate Windows executes what you enter when you press the Enter key. However, you can combine several "logical" lines of code in to a single "physical" line of code using the ':' character, and execute this entire command. For example, to display each element of the array variable Arr use the following in the Immediate Window.

```
For N= LBound(Arr) To UBound(Arr): Debug.Print Arr(N) : Next N
```

The Immediate Window always acts as if there were no Option Explicit statement in the active code module; that is, you don't have to declare variables you might use in Immediate Window commands. In fact, this is prohibited and you'll receive an error message if you attempt to use Dim in the Immediate Window.

Debug.Print

You can use the Debug.Print statement anywhere in your code to display messages or variable values in the Immediate Window. These statements don't require any confirmation or acknowledgement from the user so they won't affect the operation of your code. For example, you can send a message to the Immediate Window when a particular section of code is executed.

```
'
' some code
'
Debug.Print "Starting Code Section 1"
```

The liberal use of Debug.Print statements makes it easy to track the execution of your code. Debug.Print statements have no effect on the execution of your code and so it is safe to leave them in code projects that are distributed to end users. Debug.Print statements send messages to the Immediate Window, so you should have this window open in order to see the messages.

Unfortunately, there is no way to programmatically clear the Immediate Window. This is a shortcoming that has frustrated many programmers.

Debug.Assert

In Excel 2000 and later, you can use Debug.Assert statements to cause the code to break if a condition is not met. The syntax for Debug.Assert is: **Debug.Assert (condition)** where condition is some VBA code or expression that returns True (any numeric non-zero value) or False (a zero value). **If condition evaluates to False or 0, VBA breaks on that line (see Breakpoints, below).** For example, the following code will break on the Debug.Assert line because the condition (X < 100) is false.

```
Dim X As Long
X = 123
Debug.Assert (X < 100)
```

Debug.Assert is a useful way to pause code execution when special or unexpected conditions occur. It may seem backwards that Debug.Assert breaks execution when condition is False rather than True, but this peculiarity traces its roots back to early C-language compilers.

Remember, your end users don't want the code to enter break mode under any circumstances, so be sure to remove the statements before distributing your code, or use Conditional Compilation (see below) to create "release" and "debug" versions of your project. Note that Debug.Assert is not available in Excel97 or earlier versions.

Break Points

A break point is a setting on a line of code that tells VBA to pause execution immediately before that line of code is executed. Code execution is placed in what is called ***break mode***. When VBA is in break mode, you can enter commands in to the Immediate Window to display or change the values of variables.

To put a break point on a line of code, place the cursor on that line and press F9 or choose "Toggle Breakpoint" from the Debug menu.

To remove a break point, place the cursor on the line with the break point and press F9 or choose "Toggle Breakpoint" from the Debug menu.

When a line contains a break point, it is displayed with a brick colored background. Immediately before this line of code is executed, it will appear with a yellow background. Remember, when a break point is encountered, code execution is paused but that line of code has not yet executed. **You cannot place break points on blank lines, comment lines, or variable declaration lines (lines with Dim statements).**

After a break point is encountered, you can resume normal code execution by pressing F5 or choosing "Continue" from the Run menu, or stepping through the code line by line (see below).

Note that break points are not saved in the workbook file. If you close the file, all break points are removed. Breakpoints are preserved as long as the file is open.

Stepping Through Code

Normally, your code runs unattended. It executes until its logical end. However, when you are testing code, it is often useful to step through the code line by line, watching each line of code take effect. This makes it easy to determine exactly what line is causing incorrect behavior.

You can step through code line by line by pressing the F8 key to start the procedure in which the cursor is, or when VBA is paused at a break point. Pressing F8 causes VBA to execute each line one at a time, highlighting the next line of code in yellow. Note, the highlighted line is the line of code that will execute when you press F8. It has not yet been executed.

If your procedure calls another procedure, pressing F8 will cause VBA to step inside that procedure and execute it line by line.

You can use SHIFT+F8 to "Step Over" the procedure call. This means that the entire called procedure is executed as one line of code. This can make debugging simpler if you are confident that the problem does not lie within a called procedure.

When you are in a called procedure, you can use CTRL+SHIFT+F8 to "Step Out" of the current procedure. This causes VBA to execute until the end of the procedure is reached (an End Sub or Exit Sub statement) and then stop at the line of code immediately following the line which called the procedure.

Run To Cursor

VBA also supports "Run To Cursor". This is exactly what it sounds like. It tells VBA to execute code until the line on which the cursor is sitting is reached.

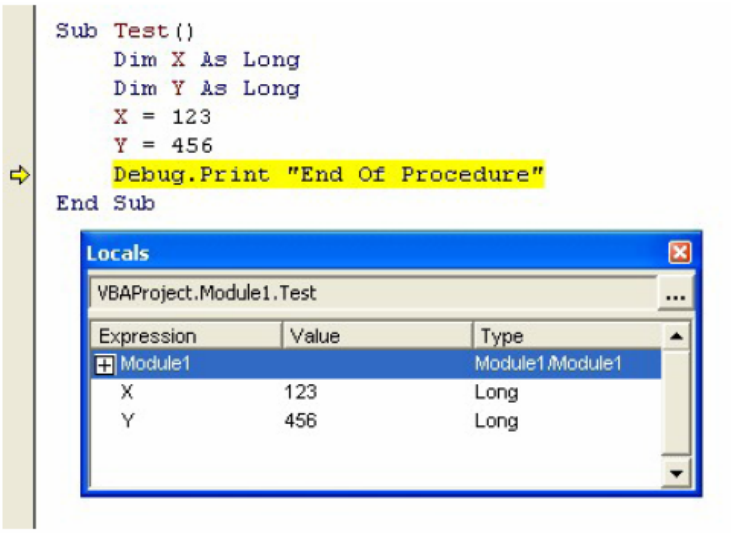
When this line is reach, VBA enters break mode. This is similar to putting a break point on a line of code, except that the break point is temporary. The second time that line of code is executed, code execution does not pause.

Locals Window

The Locals Window displays all the variables in a procedure (as well as global variables declared at the project or module level) and their values.

This makes it easy to see exactly what the value of each variable is, and where it changes, as you step through the code. **You can display the Locals Window by choosing it from the View menu.**

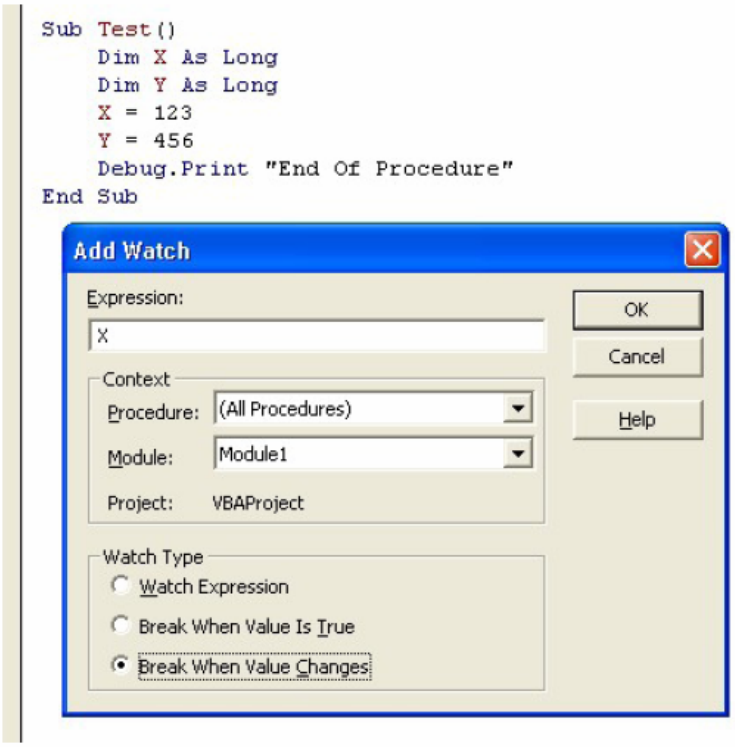
The Locals Window does not allow you to change the values of variables. It simply displays their names and values. The Locals Window is shown below. Note that the variables X and Y in procedure Test are displayed in the window. The line highlighted in yellow is the current line of execution -- it is the next line of code that VBA will execute.



Watch Window

The Watch Window allows you to "watch" a specific variable or expression and cause code execution to pause and enter break mode when the value of that variable or expression is True (non-zero) or whenever that variable is changed. (Note, this is not to be confused with the Watch object and the Watches collection).

To display the Watch Window, choose it from the View menu. To create a new watch, choose Add Watch from the Debug menu. This will display the Add Watch window, shown below.



There are three types of watches, shown in the Watch Type group box.

1. **"Watch Expression"** causes that watch to work much like the Locals Window display. It simply displays the value of a variable or expression as the code is executed.
2. **"Break When Value Is True"** causes VBA to enter break mode when the watch variable or expression is True (not equal to zero).
3. **"Break When Value Changes"** causes VBA to enter break mode when the value of the variable or expression changes value.

You can have many watches active in your project, and all watches are displayed in the Watch Window. This makes is simple to determine when a variable changes value.

Homework 1

Repeat the calculations and graphs that are displayed in Notes 1 for the following problem data:

Determine the compressibility factor “Z” for a set of values of specific volume at known temperature for benzene vapor using van der Waals equation of state (VDW) and the Redlich-Kwong equation of state (RK). Also prepare a suitable chart (graph) of pressure vs specific volume. Use the following problem data:

Description	Symbol	Value	Units
Substance		Benzene	
Temperature	T	373.15	K
Critical Pressure	Pc	48.6	atm
Critical Temperature	Tc	562.6	K
Gas Constant	Rgas	0.08206	atm L / gmol K

Values of specific volume to be used: (L/gmol) 0.04, 0.08, 0.1, 0.2, 0.4, 0.8, 1, 2, 4, 8, 10, 20, 40, 80.

RK	VDW
$P = \frac{RT}{(V-b)} - \frac{a}{V(V+b)\sqrt{T}}$	$(P + \frac{a}{V^2})(V-b) = RT$
$a = 0.42747 \frac{(R^2 Tc^{5/2})}{Pc}$	$a = \frac{27}{64} \frac{(R^2 Tc^2)}{Pc}$
$b = 0.08664 \frac{(RTc)}{Pc}$	$b = \frac{RTc}{8Pc}$

You should print the Excel worksheet as well as the FORMULA worksheet (use Control-` to switch to FORMULA view). Be sure to use the “class mandated” identification section as well as the “problem organization” suggested in the notes.

Homework 2

Correlation of Heat Transfer Data

This assignment provides the opportunity to practice using linear regression techniques as described recently in lecture.

Correlation of heat transfer data using dimensionless groups:

Heat transfer to a fluid flowing in a pipe has been considered by Geankoplis using the Buckingham Pi method and the result is that the Nusselt number (Nu) is a function of the Reynold's number (Re) and the Prandtl number (Pr).

where $N_u = hD/k$

$R_e = DV\rho/\mu$

$P_r = C_p\mu/k$

A widely used correlation for heat transfer in turbulent flow is the Sieder and Tate equation:

$$N_u = 0.023R_e^{0.8}P_r^{1/3}\left(\mu/\mu_w\right)^{0.14} \qquad \text{(A)}$$

which is of the general form

$$N_u = aR_e^{b}P_r^{c}\left(\mu/\mu_w\right)^{d} \qquad \text{(B)}$$

Data from Katz and Williams for heat transfer in a concentric tube exchanger with a 0.75 inch (No 16 BWG) inside tube and a 1.25 inch outside iron pipe is available in the table below for a particular oil.

Prepare a regression analysis to establish the following items:

1. Using an equation of the form of (A) determine the parameters that appear explicitly in (A). Hint: If you take the logs of both sides and rearrange the terms, you can form a linear equation.
2. Run the regression again without the viscosity term. Which model fits better? Is there statistical evidence that the term involving viscosity should not be kept in the model?

Prepare a report that addresses these issues.

Table B–16 Heat Transfer Data External to 3/4-inch OD Tubes[a]

Point	Re	Pr	μ/μ_w	Nu
1	49000	2.3	0.947	277
2	68600	2.28	0.954	348
3	84800	2.27	0.959	421
4	34200	2.32	0.943	223
5	22900	2.36	0.936	177
6	1321	246	0.592	114.8
7	931	247	0.583	95.9
8	518	251	0.579	68.3
9	346	273	0.29	49.1
10	122.9	1518	0.294	56
11	54.0	1590	0.279	39.9
12	84.6	1521	0.267	47
13	1249	107.4	0.724	94.2
14	1021	186	0.612	99.9
15	465	414	0.512	83.1
16	54.8	1302	0.273	35.9

[a]Williams, R. B., and Katz, D. L., *Trans. ASME*, *74*, 1307–1320 (1952).

Homework 3

This assignment provides the opportunity to practice using Excel Macro capability. This assignment requires you to demonstrate the correct functioning of the three macros described below to the instructor as well as turning in the “code” for your macros.

This assignment will require you to have read and understood the code and examples in the text.

In this assignment **YOU MUST WORK WITHOUT THE HELP OF YOUR FELLOW STUDENTS**. You are on your honor to meet this requirement. If you need assistance, see the instructor.

RECEIVING HELP FROM OTHER SOURCES WILL MERIT A GRADE OF ZERO FOR BOTH INDIVIDUALS.

Macros to be recorded from the keyboard

Write two (related) macros which are assigned to the Cntl-L and Cntl-R keys. These macros should move the data in the selected cell to the cell on the “left” or “right” respectively. By repeatedly pressing the Cntl-L or Cntl-R keys the data in the selected cell should continue to move in the intended direction.

Note: It is not necessary to check if the “destination” cell contains data. If it does, you can “overwrite it”.

Test your macros by entering the following formula in cell E7: =pi(). Then select cell E7 and demonstrate that you can move the data either right or left by repeatedly pressing either Cntl-L or Cntl-R.

Macros to be created using the editor

Write a macro that is assigned to the Cntl-E key. This macro should exchange the data between two cells that are adjacent to each other horizontally. The cell on the left should always be highlighted selected). By repeatedly pressing Cntl-E the values should exchange positions. Hint: You will need to copy to local variables (variables defined in the macro) the values of the two cells and then put the values in the correct locations. Since you don’t know the kind of data in those locations (as opposed to “integer” or “single” or “double”) you should declare these variables as “variant” (meaning ANY kind of data).

Test your macro by entering the characters “A” “B” “C” and “D” in cells A4 through D4. Select B4 and by pressing Cntl-E the data should “interchange” with the data in C4. The selected cell should remain as B4 so that repeated pressing of Cntl-E will continue to interchange the data.

Project 1

PROJECT 1 – Trial and Error Root Finding

In this assignment you will be employing many of the techniques used in Classwork 1 on a new project. The underlying idea is to determine the bubble-point temperature for an ideal binary mixture. You will be responsible for the identification of key equations as well as locating the necessary physical property data. The values provided below are for a different mixture.

There are several “rules” to be observed in this assignment:

1. Named variables are to be used for all scalar data items
2. You are to adhere to the problem identification section and the general format used in Classwork 1.
3. The solution methodology (trial and error root finding) must be that described below, that is, you are not free to solve the problem in another fashion (employing other techniques or capabilities of Excel).
4. Your program should accept data for the following items:

Input Variables			
Description	Symbol	Value	Units
Low Temperature	Tlow	60	C
High Temperature	Thigh	70	C
Number of Points	nPts	11	-
Liq Mole Fraction A	xmol1	0.10	-
System Pressure	P	760	mm Hg

5. Your program should accept data for the physical properties (Antoine's constants):

Physical Property Data			
Description	Symbol	Value	Units
Antoine's Data	A_1	6.85221	
	B_1	1064.63	
	C_1	232	
	A_2	6.87776	
	B_2	1171.53	
	C_2	224.366	

6. The bubble point of a solution can be determined by assuming (guessing) a temperature and computing the vapor pressures of the two components (via the Antoine’s Equation) and then calculating the vapor phase mole fractions of the two components. If the assumed temperature is correct, the mole fractions will sum to 1.0.
7. We will guess an interval (Tlow and Thigh) thought to contain the solution (bubblepoint). This interval will be divided into 10 intervals (11 points). A deltaT will be computed and a table of T values established based on the current interval.

Preliminary Calculations		
Temperature Inverval	dT	1

Calculations	n	T (C)
	0	60.00
	1	61.00
	2	62.00
	3	63.00
	4	64.00
	5	65.00
	6	66.00
	7	67.00
	8	68.00
	9	69.00
	10	70.00

In operation, we will type in different temperature intervals (say separated by 10 degrees) until we find the interval containing the bubblepoint and then reduce the interval by a factor of 10 repeatedly until the bubblepoint is known within 0.001 degrees C.

It is suggested you may wish to employ the following columns:

n	T (C)	T (K)	log10p1	log10p2	p1 (mmHg)	p2 (mmHg)	y1	y2	y1+y2
0	60.00	333.15	3.21	2.76	1607.74	572.75	0.21155	0.67825	0.88980
1	61.00	334.15	3.22	2.77	1654.48	592.11	0.21769	0.70118	0.91887

8. Solve the following three cases with your program:
   a. benzene-hexane system at 2.0 atm with xBenzene=0.2
   b. hexane-octane system at 1.0 atm with xHexane=0.3
   c. pentane-hexane system at 1.0 atm with xPentane=0.99

Project 2

PROJECT 2 – Use of Excel Matrix Functions (Linear Equations)

In this assignment you will be employing Excel matrix functions to solve a system of simulataneous linear equations that result from material balances on a steady state process with no recycle.

General Problem Description

Xylene, styrene, toluene and benzene are to be separated with the array of three distillation columns that is shown below where F, D, B, D1, B1, D2 and B2 are the molar flow rates in mol/min.

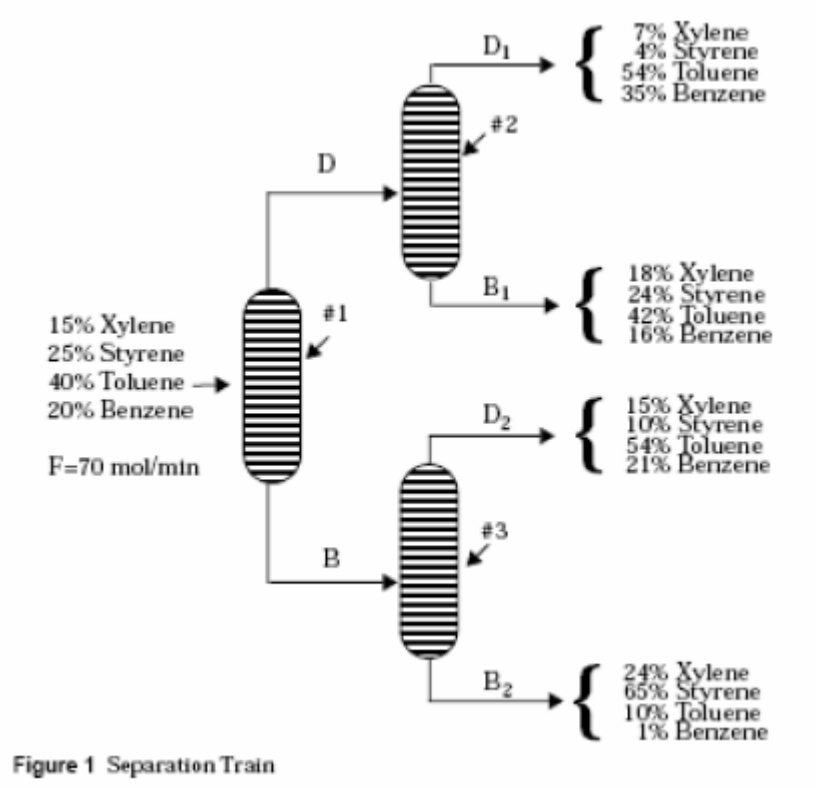


Figure 1 Separation Train

General approach:

1. Material balances on individual components on the overall separation train yield equations of the form:

   Xylene: 0.07D1 + 0.18B1 + 0.15D2 + 0.24B2 = 0.15 x 70

2. Overall balances and individual component balances on column #2 can be used to determine the molar flow rate and mole fractions from the equation of stream D from

   Molar Flow Rates: D = D1 + B1
   Xylene: XDxD = 0.07D1 + 0.18B1

3. Similarly, overall balances and individual component balances on column #3 can be used to determine the molar flow rate and mole fractions of stream B from the equation set

   Molar Flow Rates: B = D2 + B2
   Xylene: XBxB = 0.15D2 + 0.24B2

Using your knowledge of chemical engineering principles, set up the necessary matrix of equations to allow you to determine the following items:

(a) Calculate the molar flow rates of streams *D1, D2, B1* and *B2*.
(b) Determine the molar flow rates and compositions of streams *B* and *D*. U
(c) Determine the recovery of each component in each exiting stream in each column.
Recovery = amount in exit stream / amount in feed stream.

Project 3

PROJECT 3 – Simulation: "Interacting Tanks"

In this assignment you will be employing VBA to simulate a physical situation and explain the behavior observed.

The class will be working with independent teams of three students. Each team is responsible for providing the necessary fluid mechanics equations necessary to solve the problem as well as other data items.

Problem Description

Consider a system involving two standard steel 55 gallon tanks both of which have had the tops removed.

Tank #1 sits on the ground and contains a ½" circular hole in the side of the tank 12" above the bottom of the tank. Tank #1 also contains a ¼" circular hole in the side at ground level.

Tank #2 is elevated 30" higher than Tank #1 and is located 6" away from Tank #1. Tank #2 has two ½" circular holes in the side nearest to Tank #1. The first hole is located 12" above the bottom of the tank and the second hole is located 12" below the top of the tank.

At the start of the problem, Tank #2 is full and Tank #1 is half full of water.

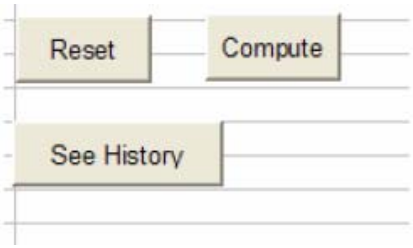
Program Requirements

Develop a general VBA program/spreadsheet to model the above situation, that is, your program should be able to receive "new data" and still function properly. This would include changes in the material (water), changes in the relative locations of the tanks (spacing) and changes in the characteristics of the holes (diameter, location). You do not have to allow for more or fewer holes.

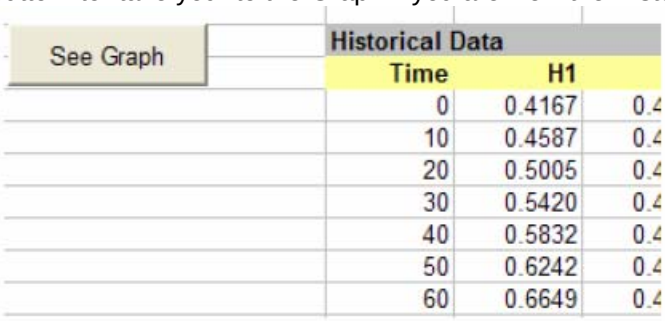
Issues (not comprehensive):

1. Problem data is to be provided in "friendly" units and converted to "standard units". This "conversion" should be performed on the spreadsheet (not using VBA). The VBA program will "read" the "standard units" data.
2. Employ a "Reset" and "Compute" button.

3. Provide a "History Report" that provides "snapshot" information about the progress of the simulation periodically. It is not desirable to have ALL the information since the size of the integration step may produce too much data.
   For example you might want to record the following items:
   - Time (since start)
   - H1, H2 (Height of water in Tank 1 and Tank 2)
   - mFlow2a, mFlow2b (mass flow of water leaving Tank 2 via each hole)
   - mFlow1a. mFlow1b (mass flow of water leaving Tank 1 via each hole)
   - %Cap1, %Cap2 (percentage of tank capacity occupied by water)
   - Other data you feel are helpful
4. Provide a graph plotting the H1 and H2 data from the "Historical Data".
5. Provide a "button" to "take you" to the Historical Data if you are viewing the Status Report.



6. Provide a “button” to “take you” to the Graph if you are view the Historical Data.



7. Your simulation basically involves solving two simultaneous ordinary differential equations. Using material balance principles, write expressions for dH1/dt and dH2/dt. Use a time-step of dT=1 second. You can then determine the change in H1 and H2 by multiplying the derivative by dT, that is:

   deltaH1 = dH1dt*dt and deltaH2 = dH2dt*dt

   therefore

   H1 = H1 + deltaH1 and H2 = H2 + deltaH2

8. You can assume a value of dt=1 second will be satisfactory (e.g., provides sufficient accuracy) and you can use an integer (or long) format. On the other hand, this would produce too much output (for the history and plot), therefore, data is only written to this area periodically, dTrep=10 seconds for example.
9. The “issue” of calculation of flow between the two tanks is left to the student/group. Your coursework in “fluids mechanics” should be helpful.
10. You do not need to do “input checking” in this assignment. Just be careful in entering your problem data.

Program Objectives:

Determine the maximum height (and time) that the water reached in Tank #1.
Determine the time for Tank #1 to completely empty.

Report Issues:

Your report should clearly indicate the equations and sources employed as well as the approach taken to modeling this problem.