City University London

2005

Programming Excel/VBA Part II

Andreas Fring

Recap:

- general intro to excel (anatomy of the window)
- absolute, relative and mixed referencing (A1,A$1,A1$,A1)
- functions (=F(... ,....,.....))
  - lookup tables (VLOOKUP,HLOOKUP)
- VB editor
- user defined functions (UDF)
- codes involving lookup functions
- error messages
- declaration of constants
- declaration of variables
- select case (if blocks)

2

VBA Control Commands:

- What are control commands?
- If ... Then (already known)
- Select Case (already known)
- Do .... Loop
- For ... Next
- While ...Wend
- For each ... Next
- Goto
- With ... End With

3

Looping:

- Loops are mechanisms for repeating the same procedure several times, e.g. the same mathematical procedure, reading repeatedly rows or columns of a table, etc.
- There are two structures in VBA for this:
  Do ... Loop and For ... Next
- Do ... Loop is used when the loop terminates when a logical condition applies, e.g. a mathematical statement such as x<11 or the end of a data file is reached etc.

• Syntax: **Do** {**While**|**Until**} condition
[statements]
[**Exit Do**]
[statements]
Loop

4

- In the DO WHILE ...LOOP the looping continues <u>while</u> the condition is true.
- In the DO UNTIL ...LOOP the looping continues <u>until</u> the condition is true.
- EXIT DO terminates the looping.
- Warning: Make sure you do not construct infinite loops.
  In case this happens use: Ctl + Break to abort
- Example: Write a function which checks the following identity:

$$\sum_{a=1}^{n} a = \frac{n(n+1)}{2}$$

5

• Code:

```
Function GSUM(n)
a = 0
Do Until a = n + 1        or   (Do While a < n + 1)
    GSUM = GSUM + a
    a = a + 1
Loop
End Function
```

gives for instance: GSUM(112) ⇒ 6328 = 112 *113/2

• equivalently:

```
Do
    GSUM = GSUM + a
    If    a = n  Then Exit Do
    a = a + 1
Loop
```

6

• Nesting DO...LOOP: You can also nest DO...LOOP structures to produce more complicated structures

```
• Syntax: Do {While|Until} condition
              Do {While|Until} condition
                  Do {While|Until} condition
                  ......
                  Loop
              Loop
          Loop
```

• EXAMPLE: Let's verify the identity

$$\sum_{k=1}^{p}\sum_{n=1}^{k} n = \frac{1}{6}p(1+p)(2+p)$$


```
Function NEST(p)
 k = 1
 Do Until k = p + 1
    n = 1
    Do Until n = k + 1
       NEST = NEST + n
       n = n + 1
    Loop
    k = k + 1
 Loop
End Function
```

```
Function NESTSUM(p)
   NESTSUM = p * (1 + p) * (2 + p) / 6
End Function
```

NEST(p) = NESTSUM(p)

• For ... Next is used when you know in advance how many times you want to iterate

```
• Syntax: For counter = first To last [Step step]
               [statements]
               [Exit For]
               [statements]
          Next [counter]
```

· counter: number which counts the loops
· first/last: initial/final value of counter
· step: increment by which the counter is change in each iteration

• Code: (same output as GSUM)

```
Function GSUMNEXT(n)
   For a = 1 To n
      GSUMNEXT = GSUMNEXT + a
   Next a
End Function
```


• Using now Step verify:

$$\sum_{a=1}^{n} 2a = n(n+1)$$

• Code:

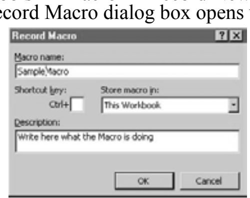
```
Function GSUMNEXT2(n)
   For a = 2 To 2*n  Step 2
      GSUMNEXT2 = GSUMNEXT2 + a
   Next a
End Function
```

gives for instance: GSUMNEXT2(112) ⇒ 12656 = 112*113

Macros:

• In Labsession 11 you have already seen how to write a subroutine (Macro) using the VBA editor. (not UDF)
• Alternatively you can also create them with the Macro recorder. In this way you do not need to know any VBA commands.

1) Recording a Macro:
   i) open a worksheet
   ii) select Tools → Macro → Record New Macro ↵
      ⇒ the record Macro dialog box opens up



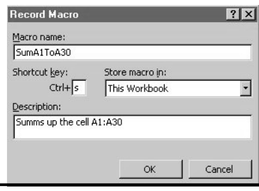
iii) enter Macro Name, e.g. "SumA1toA30"
   - not all names are allowed, such as function names, special signs in the name as !,?, blank,... are also not possible

iv) enter a letter for the shortcut key, e.g. "s"

v) store the macro somewhere, e.g. "This workbook"

vi) fill in the decription box, e.g. "sum up the cells A1:A30"

vii) Ok ↵ , the recording is on. Now all actions you carry out on the worksheet will be recorded and its code will be produced.



viii) For example:

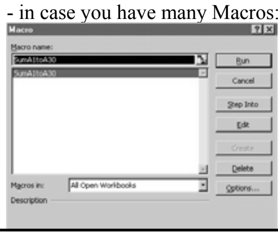
Carry out an action which sums up the cells A1:A30
- select a cell in your worksheet different from column A
- write: " The sum of the cells A1:A30 is: "
- select the adjacent cell and write: "=Sum(A1:A30)"
- the effect of this is that in the cell in which you wrote "=Sum(A1:A30)" this sum will be displayed
  · if a cell is empty its value contributes zero to the sum
  · you can now change the content of A1:A30 and the sum will be updated automatically

ix) - select Tools → Macro → Stop Recording ↵
- alternatively in the window on the worksheet select Stop Recording ↵
- if that window is now visible, you can make it appear by selecting Edit → Toolbars → Stop Recording ↵

13

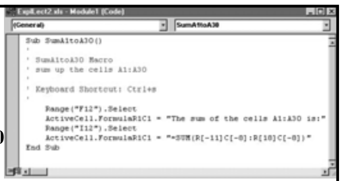
2) Viewing the code:

- The recording has produced a VBA code, which alternatively we could have programmed by hand:
- Let´s see what we have just produced:
- Select Tools → Macro → Macros ↵
  ⇒ a window called Macros opens up
- the window "Macro name" shows the name of the Macro
- in case you have many Macros: select Options ↵ to see the details of it (in case you do not remember)



- Select Edit ↵
  ⇒ the code we have just produced will show up

14



```
Sub SumA1toA30()
'
' SumA1toA30 Macro
' sum up the cells A1:A30
'
' Keyboard Shortcut: Ctrl+s
'
    Range("F12").Select
    ActiveCell.FormulaR1C1 = "The sum of the cells A1:A30 is:"
    Range("I12").Select
    ActiveCell.FormulaR1C1 = "=SUM(R[-11]C[-8]:R[18]C[-8])"
End Sub
```

15

3) Activating the Macro:

i) Select Tools → Macro → Macros ↵
  ⇒ a window called Macros opens up
  the macro's name appears in the window "Macro name:"
  · in case you have more than one, select the one you want
  Select Run ↵
  ⇒ what you have recorded before will be executed now

ii) Use the shortcut:
  - our example just: Ctl + s

iii) If you were editing the code:
  Select ▸ ↵
  ⇒ a window called Macros opens up ⇒ i)

iv) Using customized buttons or other objects:
  - we have to see first how to create those (see point 4):

15

• Example: We calculate once more

$$\sum_{a=1}^{n} a = \frac{n(n+1)}{2}$$

- first you have to fill in: 1→A1 , 2→A2, 3→A3 ... 30→A30
- you can do this by hand, but the faster way is to use "Series":
  - put 1 into cell A1:
  - select Edit → Fill → Series ↵
    ⇒ a window called Series opens up
  - Fill in: Series: • Column
    Type: • Linear
    Step value: 1
    Stop value: 30
  - Ok ↵

- activate the Macro ⇒ The sum of the cells A1:A30 is 465

17

4) Customized buttons (and other objects):

i) Make the "Forms toolbar" visible:
  Select Tools → Customize → Toolbars → ☑ Forms → Close↵

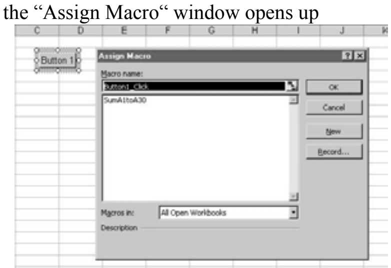
⇒ a new toolbar with possible form commands open up

18

ii) Assign a Macro to a button:
- Select Button (a grey box)

→ select a cell in your worksheet

⇒ the “Assign Macro“ window opens up



→ select the Macro which you want to attach to this button, e.g. SumA1toA30 → Ok ↵

⇒ it says now “Button #“ on your button

19

iii) Activating the Macro:

- Selecting now this button will activate the Macro you have attached to it, e.g. SumA1toA30

iv) Changing the button design:

• attach a better text to the button:

- select the right mouse button (moving first over the button)

⇒ a variety of commands opens up:

→ select Edit text ↵

→ type a meaningful text onto the button, e.g. Sum A1 to A30

⇒ Button 2 ⇒ Sum of A1 to A30

• change the size of the button:

- select the right mouse button (moving first over the button)

→ select Format Control ↵

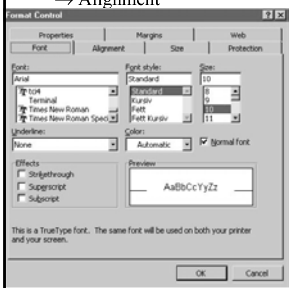
20

• change the size of the button:

- select the right mouse button (moving first over the button)

→ select Format Control ↵

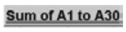
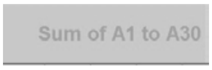
→ Alignment



→ ☑ Automatic size → Ok ↵

21

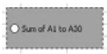
• similarly you can change the writing direction, the text fonts, the text and button size, the margins of the button, the colour, the status of the protection, etc.

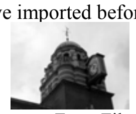
• You can also assign Macros to other objects:

· the ☑ symbol from the forms toolbar

· a text label *Aa* on the forms toolbar

· other symbols from the forms toolbar 

· a picture you have imported before such as



22

(Select Insert → Picture → From File or Clip Art → choose a picture)

Numerical Methods with Excel/VBA:

• Many problems in Mathematics, Physics, Economics, etc can only be solved in very idealized situations in an exact analytical fashion. Even solvable problems can often only be tackled with great effort.

• Numerical methods often lead to solutions which are extremely close to the correct answers. They lead very often quickly to some insights.

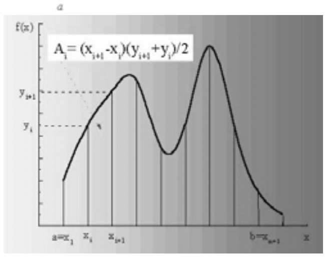
• Especially with the advance in computer technology, in terms of speed and storage capacity the limits of what can be computed are permanently pushed.

• Here we only have a glimpse at some methods to get an idea what could be possible and to apply the programming structures we have learned so far.

23

➤ Numerical Integration

• Recall: $I = \int_a^b f(x)dx$ = area below the curve $f(x)$



- Idea: approximate the integral by sums over trapezoidal areas :

$$I \approx \sum_{i=1}^{n} A_i$$

24

- Take the subdivision of the domain [a,b] to be evenly spaced:

$$x_{i+1} - x_i = \frac{b-a}{n} = \Delta$$

$\Longrightarrow$ Trapezoid rule for integration:

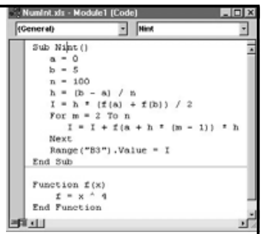
$$\boxed{I \approx \Delta\left[\frac{1}{2}(y_1 + y_{n+1}) + \sum_{i=2}^{n} y_i)\right]}$$

• Let us write a module (program) for this:

- Input: a ≡ lower integration limit
  b ≡ upper integration limit
  n ≡ number of subdivisions
  some function f(x) which we want to integrate
- Output: approximate value for the integral


```
Sub Nint()
  a = 0
  b = 5
  n = 100
  h = (b - a) / n
  I = h * (f(a) + f(b)) / 2
  For m = 2 To n
     I = I + f(a + h * (m - 1)) * h
  Next
  Range("B3").Value = I
End Sub

Function f(x)
  f = x ^ 4
End Function
```



Put the result onto the Excel sheet into the cell B3

• Example 1: $\int_0^5 x^4 dx = \frac{x^5}{5}\Big|_0^5 = 625$

- The program gives:

$n = 10 \quad I = 635.4063$

$n = 100 \quad I = 625.1042$

$n = 1000 \quad I = 625.0010$

$n = 10000 \quad I = 625.0000104$

• Example 2: $\int_0^\pi \sin(x)dx = -\cos(x)|_0^\pi = 2$

- Generate the π by 4 Arctan(1). In VBA this is written as 4 *Atn(1).
- The program yields:

$n = 10 \quad I = 1.9835$

$n = 100 \quad I = 1.999836$

$n = 1000 \quad I = 1.999998$

- So far we could have solved the intgrals also analytically, but not the next integral.

• Example 3:

$$\int \frac{\sin(x)}{x}dx = \text{Sinus Integral function}$$

$$\int_0^\pi \frac{\sin(x)}{x}dx \approx 1.85194$$

- How do we deal with the lower bound a=0? This is well defined analytically, but the computer can not handle 0/0, if we don‘t specify how to do that. Recipe: Just take the lower bound a to be a very small number, e.g.a=0.0000001.
- The program yields:

$n = 10 \quad I = 1.8493$

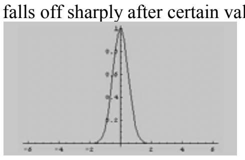
$n = 100 \quad I = 1.851911$

$n = 1000 \quad I = 1.851937$

• Example 4: $\int \exp(-2x^2)dx = \frac{1}{2}\sqrt{\frac{\pi}{2}}\ \text{Errorfunction}(x\sqrt{2}\,)$

$$\int_{-\infty}^{\infty} \exp(-2x^2)dx = \sqrt{\frac{\pi}{2}} \approx 1.25331$$

- How do we deal with infinity? Introduce a cut-off at some value large enough such that the mistake is small. This is possible because the integrant falls off sharply after certain values:



- Compute instead: $\int_{-5}^{5} \exp(-2x^2)dx$
- The program gives:

$n = 10 \quad I = 1.27134$

$n = 100 \quad I = 1.253314$

$n = 1000 \quad I = 1.2533$

• Different types of methods:

- Simpson‘s 1/3 rule (based on three adjacent points):

$$I = \int_a^b f(x)dx \approx \frac{\Delta}{3}\left[\sum_{i=1,3,5,\ldots}^{n-2} y_i + 4y_{i+1} + y_{i+2}\right]$$

- Simpson‘s 3/8 rule (based on four adjacent points):

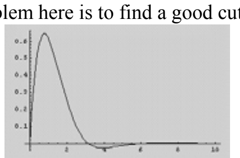
$$I = \int_a^b f(x)dx \approx \frac{3}{8}\Delta\left[\sum_{i=1,4,7,\ldots}^{n-3} y_i + 3y_{i+1} + 3y_{i+2} + y_{i+3}\right]$$

- Runge-Kutta methods, Monte Carlo integration,…
  · Here we do not derive these rules, but just take them as facts. See a different course on numerical methods for details.
- Let us implement the Simpson‘s 3/8 rule as a user defined function
- Implement the Simpson‘s 1/3 rule in Labsession 3.

```
Function Nintff(a, b, n)
  h = (b - a) / n
  I = 0
  For m = 1 To n - 2 Step 3
   I = I + (f(a+h*(m-1)) +3* f(a+h* m) +3* f(a+h*(m+1))+f(a+h*(m+2)))
  Next
  Nintff = I * h * 3 / 8
End Function
```

• Example 1: $2\int_0^\infty \sin(x)\exp(-x)dx = 1$

· A problem here is to find a good cut-off for the upper limit.



$\Longrightarrow$ b=10 ?

31

- Compare different integration methods:

b=10	Trapezoid:	1/3 Simpson	3/8 Simpson
n=10	0.83920049	0.97383313	0.93930204
n=100	0.99839888	1.00006056	1.00006255
n=1000	1.00004637	1.00006279	1.00006328
n=10000	1.00006265	1.00006279	1.00006284
b=20	Trapezoid:	1/3 Simpson	3/8 Simpson
n=10	0.43524312	0.61641151	0.49069239
n=100	0.99334224	0.99996411	0.9999185
n=1000	0.99993333	0.99999999	0.99999999
n=10000	0.99999933	1	1

· In this example we introduce an additional error though the cut-off.

· When the subdivision of the interval is large enough the three methods are almost equally good.

32

• Limitations:
  - The speed of the computer.
  - The accuracy of the numerical method used. (In principle there exist procedures to estimate the errors made.)
  - The accuracy of the computer, i.e. in computing functions used in the program and variables employed such as single or double precision.
  - Various other approximations such as division by zero, cut-offs for lower and upper bounds etc.

• There exist different types of numerical methods for other mathematical problems, such as solving polynomial equations, solving differential equations etc.

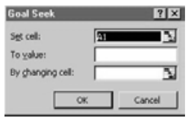
• Some methods are implemented in Excel as Built-in functions:

33

➤ Goal Seek

• Goal seek is a numerical routine implemented in Excel in form of a built-in function. It canbe used to solve equations.

• Usage: select Tools → Goal Seek ↵ → a dialog window opens



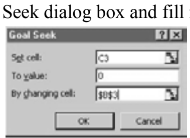
- Set cell contains the left hand side of an equation you want to solve
- To value contains the RHS of the equation
- By changing cell contains the variable of the equation

• Disadvantage: You have to guess a value near the answer.

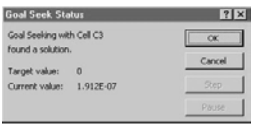
• Example: Solve the equation: $2x^2-9x-5=0$
  (We expect to find: $x_1=-1/2$ and $x_2=5$)
  - Type into the cell C3: =2*B3^2-9*B3-5
  - Type into the cell C4: =2*B4^2-9*B4-5

34

- Type into the cell B3 some starting value, e.g. -10
- open the Goal Seek dialog box and fill in



- OK ↵ →



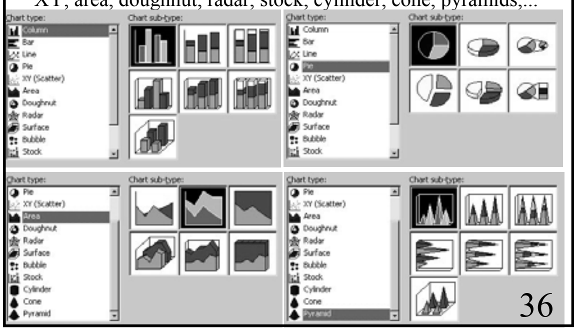
- The cell B3 and C3 have changed to -0.5 and 1.912E-07
- Repeat this process for the cells C4 and B3 to find the other solution. (You need a new guess for the starting value.)

• A more sophisticated method is the Excel Solver.

35

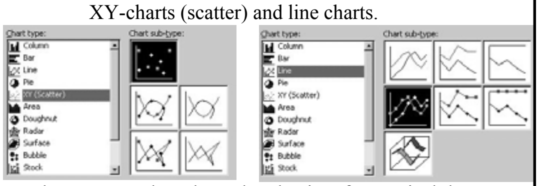
Charts (Graphs):

• Charts are ways to display data in a graphical way.
  - Excel offers various types of charts, such as column, bar, pie, XY, area, doughnut, radar, stock, cylinder, cone, pyramids,...



36

- Here we want to learn more about the most common types:
  XY-charts (scatter) and line charts.

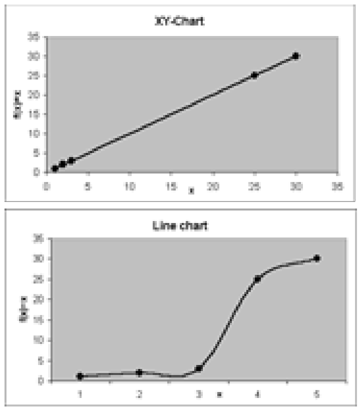


- XY charts are used to plot ordered pairs of numerical data, e.g. from a scientific experiment, mathematical functions, etc.
- Line charts are used when the x-values are textual, e.g. month of the year, names of people or companies, places, etc.

· These two types of charts should not be confused with each other, as their display is quite different, which is not suggested by their names

· Example:

37



We plot the data:

x	f(x)
1	1
2	2
3	3
25	25
30	30

38

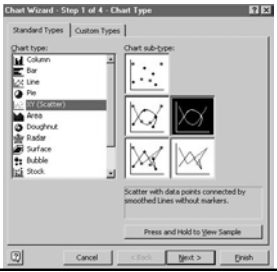
1) <u>Creating an XY/line chart</u>:

i) open a worksheet

ii) select the data you wish to display, e.g. cells A1:B30

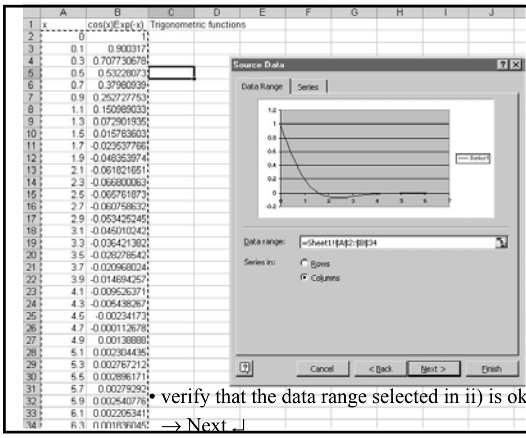
· in particular we want to see here how to plot a function f(x), e.g. the x are in A1:A30 and the f(x) in B1:B30

iii) open the chart wizard ⇒ a series of 4 dialog boxes open up



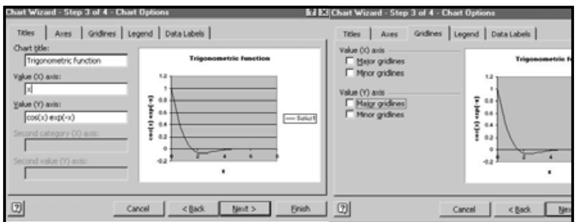
• specify the type and the sub-type of the chart → Next ↵

39

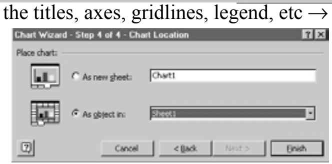


• verify that the data range selected in ii) is ok → Next ↵

40



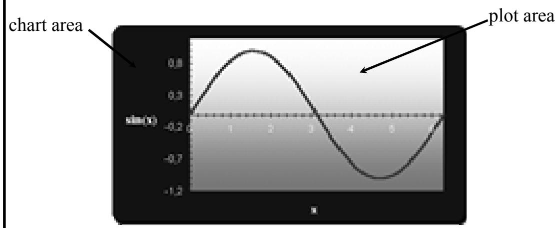
• specify the titles, axes, gridlines, legend, etc → Next ↵



• specify the location where the chart should stored→ Finish ↵
⇒ a chart will appear in the location you specified

41

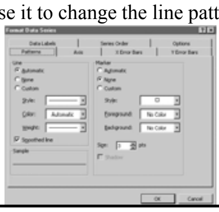
• For instance, if in some column (row) we had had some (densely enough) distributed x-values and in some other column (row) the corresponding values sin(x), we could have produced



• Most likely the design would not have been of this type, therefore ➔

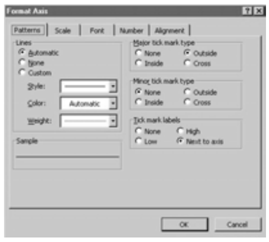
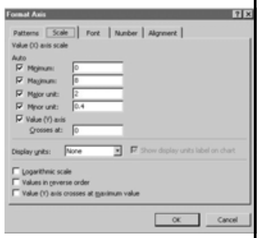
42

2) Modifying a chart:
• you can change the design of the presentation by selecting the objects you wish to modify
i) Formatting the plot area
- by default the plot area will be grey
- select the plot area ⇒ the “Format Plot Area“ window opens
- use it to change the colours of the background, frame, etc.
ii) Formatting the data series
- select the line ⇒ the “Format Data Series“ window opens
- use it to change the line pattern, data labels, error bars etc.




43

iii) Formatting the axis
- select the axis ⇒ the “Format Axis“ window opens
- use it to change the axis pattern and scale

iv) Modifying the chart options
- right select the chart area ⇒ Chart Options ↵
- use it to change titles, axes properties, gridlines, legends and data labels

44

v) Dynamical titles and axis labels
- the data are already linked in a dynamical way to the chart, this means if you change them the plot will change as well
- you can also do this with the title and axis labels
· type some text into a cell, e.g. “sin(x)“ into F1
· select the title or an axis label
· type “=“ into the Formular bar
· select again the cell where you wrote the text, e.g. F1
⇒ in the Formular bar the location of your text appears, e.g. =Sheet1!F1
· select the “✓“ to complete the process
⇒ Now, whenever you update the selected cell, e.g. F1, the text inside the chart will change accordingly
vi) Changing the default setting
- you might have a preferred chart style and if you do not want to repeat the previous steps use this style as default
· select the chart → Chart → Chart type ↵ Select as default ↵

45

3) Data input:
• There are various ways to fill in the cells with data:
i) You can fill in the data the pedestrian way by just typing them
ii) The data might be stored externally on some file resulting for instance as output from another program.
- Importing the data:
· select a cell on your worksheet for the first value
· select Data → Get External → Import Text File ↵
⇒ Text Import Wizard opens with a series of 3 dialog boxes
· answer questions about data and file type
· modify the field width
· select the data format → Finish ↵
· confirm the location where the data should be stored
iii) Use the fill function (see lecture on Macros)
iv) Use a VBA program to fill in the data
- see for instance Lab-session 1, task 4

46

v) Use the autofill function
- The autofill function determines automatically the entries of some cells given some starting values:
· fill in some starting values
e.g. 0 →A1, 0.1 →A2, =SIN(A1) →B1, =SIN(A2) →B2
· select the area of the starting values, e.g. A1:B2
· while you are on top of the selected area the cursor will be ✚
· move the cursor to the lower right corner of the selection, until the cursor changes from ✚ to +
· drag the fill handle down (or to the right) and the new cells will be filled based on the initial selection, e.g. 0.2 →A3, 0.3 →A4, =SIN(A3) →B3, =SIN(A4) →B4, etc.
· verify that Excel really filled in the sequence you wanted!!!
⇒ In our example we have now two column of data, which we can plot against each other in the chart

47

4) Data handling:
➤ Adding data to an existing chart:
- plot area → Source data → Series → add → X/Y values, name → Ok ↵
➤ Data → sort ≡ arrange selected data alphabetically, by data or numerically in ascending or descending order
➤ Data → filter ≡ allows to filter out certain data based on their location
➤ Data → validation ≡ allows to filter certain data based on a criterion you define, e.g. a certain range
➤ Data → subtotals ≡ computes totals and subtotals for selected columns and inserts them into the sheet
➤ Data → text to columns ≡ allows to change the data type

48

5) Curve fitting:

- On many occasions one has sets of ordered pairs of data $(x_1,...,x_n, y_1,...,y_n)$ which are related by a concrete function Y(X) e.g. some experimental data with a theoretical prediction

➤ suppose Y(X) is a linear function

$$Y = \alpha X + \beta$$

- Excel offers various ways to determine α and β
  i) SLOPE, INTERCEPT - functions
  based on the method of least square

$$\min = \sum_{i=1}^{n} [y_i - (\beta + \alpha x_i)]^2$$

SLOPE($y_1,...,y_n, x_1,...,x_n$) → α
INTERCEPT($y_1,...,y_n, x_1,...,x_n$) → β

49

- How does Excel compute this? (see other courses for derivation)

· mean values: $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$

· slope: $\alpha = \sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y}) / \sum_{i=1}^{n}(x_i-\bar{x})^2$

· intercept: $\beta = \bar{y} - \alpha\bar{x}$

· regression coefficient:

$$r = \sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y}) \Big/ \sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2 \sum_{i=1}^{n}(y_i-\bar{y})^2}$$

A good linear correlation between the x_i and y_i -values is $r \cong 1$.
With VBA we can write a code which does the same job, see Lab-session 5 of Part II.

50

ii) LINEST - function
this function is more sophisticated than the previous one
LINEST($y_1,...,y_n, x_1,...,x_n$, *constant*, *statistics*)
- if *constant* = TRUE or omitted the intercept is computed otherwise it is zero
- if *statistics* = TRUE the function returns regression statistic values with the output:

slope	**intercept**
standard error in the slope	**standard error in the intercept**
r-squared	**standard error in the y estimation**

- we restrict ourselves here to

51

- notice that LINEST is an array function, such that you have to prepare for an output bigger than one cell:
  · select a range for the output, e.g. 2×3 cells
  · type the function, e.g. =LINEST(.....)
  · complete with Ctrl + Shift + Enter

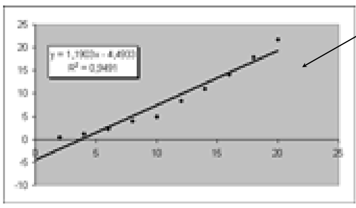
iii) adding a trendline
- this option also works for nonlinear, logarithmic, exponential ... correlations between the x- and y-values
  · choose an XY-chart with the subtype which has no line
  · right click any of the plotted points
  ⇒ Add Trendline windows opens
  · select the type of correlation, e.g. Linear, polynomial, ...
  · in Options decide if you want to add the computed equation the r-squared value etc on the chart

52

Example:
Consider the data:

2	0,4
4	1,2
6	2,3
8	4
10	5
12	8,3
14	11
16	14,1
18	17,9
20	21,8

assume linear correlation:
slope → 1.1903
intercept → -4,4933



with trendline adding

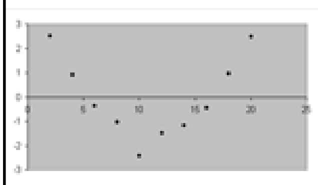
looks more or less linear?

53

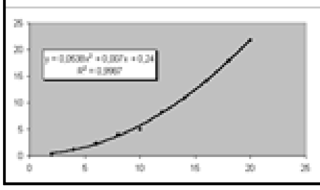
Compute the residuals, i.e. (the predicted values - the given ones):

(1.1903 x_i - 4.4933) - y_i →

2,512727
0,932121
-0,34848
-1,02909
-2,4097
-1,4903
-1,17091
-0,45152
0,967879
2,487273



not random!



quadratic fit is better!

54

Object Oriented Programming

- Premise: Everything we know in the Excel universe can be described as objects.
  - · There are about 200 objects in Excel.
  - · Our aim is to learn how to use them in VBA.
- objects can have <u>names</u>

  syntax: object("name")

  Expl.: Workbook ("Labsession5.xls"),
  Worksheet("Sums"), Range("trigdata"),
  Range("A1:A25"), ActiveCell, ActiveSheet,....
- objects can be used as <u>object variables</u>

  Expl.: Dim WB as object
  Set WB = Workbook ("Labsession5.xls")

  similar as the variables we already know, we can use WB instead of Workbook ("Labsession5.xls")

55

- objects are arranged to each other in a strict <u>hierachy</u>

  Excel application → workbook → worksheet → objectX → objectY → ...
  - • this hierachy has to be respected in the VBA syntax, e.g.

    workbook("book1.xls").worksheet ("sheet1").objectX.objectY

    not: worksheet ("sheet1"). workbook("book1.xls")....
  - • when referring to an object which is in an <u>active</u> workbook or sheet, you do not need to specify the entire hierachy

    Expl.:
    Range("A1")
  - • when it is in a <u>non-active</u> workbook and worksheet, you need to refer to the entire hierachy

    Expl.:
    workbook("book1.xls").worksheet ("sheet1").Range("A1")

56

- the <u>WITH ....END WITH</u> short hand
  - · this is a useful command which allows to avoid long hierachies

```
syntax: WITH objectX
            .objectY
            .objectZ
        END WITH
```

Expl.:

```
workbook("book1.xls").worksheet ("sheet1").Range("A1")
workbook("book1.xls").worksheet ("sheet1").Range("B25")
workbook("book1.xls").worksheet ("sheet1").Range("data")
instead: WITH workbook("book1.xls").worksheet ("sheet1")
            .Range("A1")
            .Range("B25")
            .Range("data")
         END WITH
```

57

- objects posses <u>properties</u>, can carry out <u>methods</u>, react to <u>events</u>

  objects → properties, methods, events

- the <u>properties</u> of objects are their characteristics

  syntax: object.property = property value

  Expl.:
  Range("A1").ColumnWidth = 10
  Name.Value = "This is Pi"
  Chart("temp").ChartType = "xlLine"
  - · the same kind of property can be associated to different objects

  Expl.:
  Range("A1").value = Range("B5").value
  (the value of cell B5 is assigned to cell A1)

58

- the <u>methods</u> (functions) are actions the object can initiate

  syntax: object.method [parameter := parameter value]

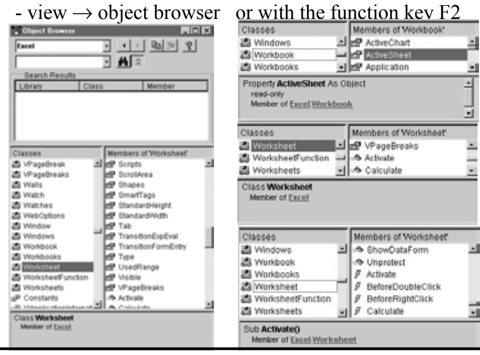
  Expl.:
  Range("A1:D4").Copy
  (copies the content of the cells A1 to D4 on the active worksheet)
  Range("A1:D4").Copy destination:=worksheet("T").Range("C5")
  (copies the content of the cells A1 to D4 on the active worksheet to the cells C5 to F8 on the worksheet named T)
- objects can change their properties as a reaction to an <u>event</u>

  syntax: object.event

  Expl.:
  worksheet("T1").Calculate
  (the object worksheet named "T1" is re-calculated and changes its properties)

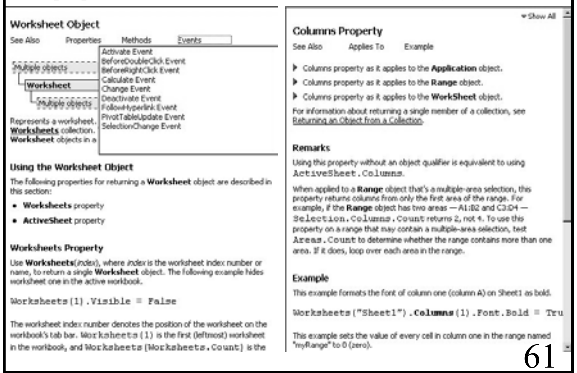
59

- the <u>object browser</u> provides you with the details of the properties, methods and events associated to particular objects
  - · it is activated in the VB editor
  - - view → object browser or with the function key F2



60

· clicking the question mark in the browser you can find out about the properties, methods and events related to an object:



61

➤ objects can be organized in <u>collections</u>

· members in same collection are on the same hierachical level

· you refer to a member of a collection just by a number

syntax: collection name(#)

Expl.:

worksheets(5) refers to the 5-th member in the worksheet collection

workbooks(3) refers to the 3-rd member in the workbook collection

names(6) refers to the 6-th member in the name collection

hyperlinks(1) refers to the 1-st member in the hyperlink collection

· note: worksheets ≠ worksheet , names ≠ name, etc

· collections can be created by using the <u>add</u>-method

syntax:
collection name.add [parameter1:= parameter value 1] , [:=]

62

Examples:

- x = 3.141592653589793
  y = true (the variables can be of different type)
  z = "too many names"
  Names.Add Name:="pi", RefersTo:=x
  Names.Add Name:="correct", RefersTo:=y
  Names.Add Name:="message", RefersTo:=z

  · you can refer to a member of the names collection as:
  · Names(2) → true in the VBA code
  · correct → true on the Excel sheet
- WITH worksheet(1)
  .Hyperlinks.Add .Range("B25"), http://www.city.ac.uk/
  END WITH
  Range("B25").Hyperlinks(1).Follow NewWindow:=True

  · inserts a hyperlink into cell B25 and executes it thereafter

63

Interactive In and Output

➤ We have already seen how to transfer data between the spreadsheet and VBA programs, by writing into cells and reading from cells:

- VBA program → spreadsheet
  Range("A1").value = 2
  (puts the value 2 into cell A1)
- spreadsheet → VBA program
  x = Range("A1").value
  (assigns the value of cell A1 to the variable x)

➤ Now we look at another useful technique, using message boxes.

- this is useful when you write a code for a user, who does not know about the VBA code, as you can provide more information

64

➤ <u>Message box:</u>

- displays a message in a dialog box and returns an integer value which depends on the answer of the user

syntax:
return = MsgBox(prompt [, buttons] [, title] [, helpfile ,context])

· parameters in [ ] are optional, i.e. you don't have to specify them

· when you omit the optional parameters you have to include the ,

· or:

syntax:
return = MsgBox(prompt:= "..." , title:= " ..."] ...)

· now you do not have to include the commas

· we will not treat here the helpfile and context option (they allow to display some help information)

65

prompt ≡ string expression, the text displayed in the dialog box (maximal 1024 characters)

title ≡ string expression, the text displayed in the title bar of the dialog box. When omited, it is the application name.

buttons ≡ a sum of several values specifying:

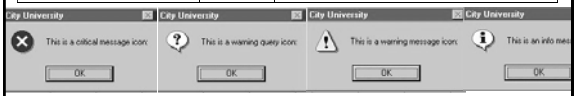
(a) the number and type of buttons:

Constant	Value	Description
vbOKOnly	0	**OK** button only
vbOKCancel	1	**OK** and **Cancel**
vbAbortRetryIgnore	2	**Abort**, **Retry**, and **Ignore**
vbYesNoCancel	3	**Yes**, **No**, and **Cancel**
vbYesNo	4	**Yes** and **No**
vbRetryCancel	5	**Retry** and **Cancel**

66

(b) the icon style

vbCritical	16	Display **Critical Message** icon
vbQuestion	32	Display **Warning Query** icon
vbExclamation	48	Display **Warning Message** icon
vbInformation	64	Display **Info Message** icon



(c) the default button

(this is the button selected when you just press return)

vbDefaultButton1	0	First button is default
vbDefaultButton2	256	Second button is default
vbDefaultButton3	512	Third button is default
vbDefaultButton4	768	Fourth button is default

67

(d) the modality of display

vbApplicationModal	0	The application stops until the user responds
vbSystemModal	4096	whole system stops until the user responds
vbMsgBoxHelpButton	16384	adds Help button
VbMsgBoxSetForeground	65536	MsgBox is foreground
vbMsgBoxRight	524288	Text is right aligned
vbMsgBoxRtlReading	1048576	text right-to-left

· select maximal one number from each of the groups (a) to (d)

· you can either use the Excel constant name or the number

e.g. buttons := 3 + 32

buttons := 35

buttons := **vbYesNoCancel** + **vbQuestion**

68

return ≡ a number between 1 and 7 which depends on the answer

· you can either use the Excel constant name or the number

Constant	**Return value**	**Selected button**
vbOK	1	**OK**
vbCancel	2	**Cancel**
vbAbort	3	**Abort**
vbRetry	4	**Retry**
vbIgnore	5	**Ignore**
vbYes	6	**Yes**
vbNo	7	**No**

· e.g. if the OK button is selected return has the value 1 or vbOK

69

▸ Examples:

```
Sub message1()
  MsgBox ("Do you know how to view pdf files?")
End Sub
or:     ret = MsgBox(Prompt:="Do you know how to view pdf files?")
Sub message2()
  prompt = "Do you know how to view pdf files?"
  title = "Programming Excel/VBA PartII"
  ret = MsgBox(prompt, , title)
End Sub
```

⇒ displays a message box with OK button

prompt: Do you know how to view pdf files?

title: Microsoft Excel (in message1)

title: Programming Excel/VBA PartII (in message2)

70

```
Sub message3()
  ......
  ret = MsgBox(prompt:=pr, Buttons:=3, Title:=ti)
End Sub
```

⇒ displays a message box with Yes/No/Cancel button

```
Sub message4()
  ......
  bu = vbYesNoCancel + vbQuestion
  ret = MsgBox(prompt:=pr, Buttons:=bu, Title:=ti)
End Sub
```

⇒ displays a message box with Yes/No/Cancel button and question mark icon (warning query icon)

71

```
Sub message5()
  pr = "Do you know how to view pdf files?"
  ti = "Programming Excel/VBA PartII"
111:
  ret = MsgBox(prompt:=pr, Buttons:=35, Title:=ti)
  If ret = vbYes Then        (or: ret = 6 then)
    ret = MsgBox("Good, you can print the lecture material", 48, ti)
  ElseIf ret = vbNo Then     (or: ret = 7 then)
    ret = MsgBox("By now you should know!", 16, ti)
  Else
    ret = MsgBox("Either you know or you don't. Decide!", 32, ti)
    GoTo 111
  End If
End sub
```

72

▸ Goto command:
- forces the program to go to a certain position

```
syntax:
        position:
           ........
           Goto  position
```

or:

```
syntax:
           Goto  position
           ......
        position
```

make sure can get out of this loop!!!!

73

▸ Input box:
- displays a prompt in a dialog box, waits for the user to enter a text or click a button, and returns a string containing the content of the text box.

```
syntax:
  return = InputBox(prompt [, title] [, default] [, xpos] [, ypos])
```

default ≡ a default output value

xpos/ypos ≡ horizontal/vertical distance of the left/upper edge of the dialog box from the left/top edge of the screen.

return ≡ string containing the content of the text box.

74

Arrays/Array functions

▸ Arrays are VBA variables which can store more than one item.
· the items held in an array are all of the same variable type
· one refers to an item by the array name and a number

syntax: declaration:	Dim Name(number)	
usage:	Name(x)	where $0 \le x \le$ number

· by default the indexing starts at 0

- Expl.: an array with three items named A

declaration: Dim A(2)
usage: A(0) = 5
A(1) = 3
A(2) = 6
note: A(3) is not defined

75

- You may change the index set from its default value

syntax: declaration:	Dim Name(x to y)	
usage:	Name(z)	where $x \le z \le y$

- Expl.: an array with three items named A

declaration: Dim A(8 to 10)
usage: A(8) = 5
A(9) = 3
A(10) = 6
note: A(6), A(7), A(11), A(12), ... are not defined

- Alternatively you can also use the array function

syntax: declaration:	Dim Name as variant
usage:	Name = array(x,y, ...,z)

· the indexing starts at zero, i.e. Name(0) = x

76

- Example 1:

```
Sub Example1()
    Dim A(8 To 10)
    A(8) = 2
    A(9) = 3
    A(10) = A(8) + A(9)
    Range("A10").Value = A(10)
End Sub
```

- writes 5 into the cell A10 of the active worksheet

- Example 2:

```
Sub Example2()
    Dim B As Variant
    B = Array(2, 3, 4, 5)
    Range("A13").Value = (B(0) + B(1)) /B(3)
End Sub
```

- writes 1 into the cell A13 of the active worksheet

77

▸ Multidimensional arrays are VBA variables which can hold more than one item related to several index sets (up to 60)
· e.g. a two dimensional array is a matrix

syntax: declaration:	Dim Name(num1,num2,num3,...)	
usage:	Name(x,y,z,...)	$0 \le x \le$ num1
		$0 \le y \le$ num2
		$0 \le z \le$ num3
		....

· the change of the index set is analogue to the one dimensional case

- Expl.: a 2 by 2 matrix $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$

declaration: Dim A(1 to 2,1 to 2)
usage: A(1,1) = a   A(1,2) = b
A(2,1) = c   A(2,2) = d

78

▸ Resizable arrays are arrays whose size is not fixed

syntax: declaration: Redim Name(x to y)
....
Redim Name(w to z)

· the first statement creates a one dimensional resizable array
· the second statement overwrites the first statement

syntax: declaration: Redim Name(x to y)
....
Redim preserve Name(w to z) $w \leq x$, $z \geq y$

· now the values in the array Name(x to y) will be saved

▸ Upper and lower bound function

· Lbound(RA) gives the lower bound of the array called RA
· Ubound(RA) gives the upper bound of the array called RA

79

- Expl.: Redim RA(1 to 10)
  x = Lbound(RA) (x = 1)
  y = Ubound(RA) (y = 10)
  Redim RA(12 to 19)
  x = Lbound(RA) (now x = 12)
  y = Ubound(RA) (now y = 19)

▸ Data exchange: Arrays can be used as an efficient way to exchange data between the Excel spreadsheet and the VBA program

• VBA program → spreadsheet
  Range("A1:B2").Value = A
  (puts the values of the array A into cells A1:B2)
• spreadsheet → VBA program
  Dim B As Variant
  B = Range("A1:B2").Value
  (assigns the values of cells A1:B2 to the array B)

80

- Expl.: The content of two 2 by 2 matrices in the cells A1:B2 and D1:E2 are read to two arrays A and B. The matrices are multiplied and the result is returned to the cells G1:H2.

```
Sub Matrix()
    Dim A, B As Variant          arrays have to be variants
    Dim C(1 To 2, 1 To 2)
    A = Range("A1:B2").Value
    B = Range("D1:E2").Value
    For i = 1 To 2               the indexing starts at 1
        For j = 1 To 2
            C(i, j) = A(i, 1) * B(1, j) + A(i, 2) * B(2, j)
        Next j
    Next i
    Range("G1:H2").Value = C
End Sub
```

81

▸ MMULT is an Excel array function which returns the product of two arrays

syntax: MMULT(array name1 , array name2)

- Expl.: MMULT(“A1:B2“ , “D1:E2“)
  ⇒ returns the same product as the previous VBA program
- notice that MMULT is an array function, such that you have to prepare for an output bigger than one cell: (recall LINEST)
  · select a range for the output, e.g. 2×2 cells
  · type the function, e.g. = MMULT(.....)
  · complete with Ctrl + Shift + Enter
- notice also: MMULT is an Excel function not VBA function

82

▸ The Split Function returns an array consisting of substrings from a string expression in which each substring is separated by a delimiter which can be specified

syntax: Split(expression [, delimiter] [, limit])

expression ≡ a string expression
delimiter ≡ the character which separates the substrings (the default value is space)
limit ≡ the maximum number of substrings to be returned (the default value is –1, that is all substrings)

- Expl.: Dim x as variant
  x = Split(“Today is Tuesday“)
  ⇒ x(1) = “Today“ x(2) = “is“ x(3) = “Tuesday“
  or: x = Split(“a,b,c,d,e,f,g“ , “,“ , 3)
  ⇒ x(1) = “a“ x(2) = “b“ x(3) = “c,d,e,f,g“

83

▸ The Join Function returns a string consisting of the values in a string array separated by a specified delimiter

syntax: Join(sourcearray [, delimiter])

sourcearray ≡ an array containing strings
delimiter ≡ the character which separates the substrings (the default value is space)

- Expl.: Dim x(1 to 3)
  x(1) = “Today“
  x(2) = “is“
  x(3) = “Tuesday“
  y = Join(x)
  ⇒ y = “Today is Tuesday“

84

- similarly:

y = “Today “ & “is “ & “Tuesday“

⇒ y = “Today is Tuesday“

· in addition:

Dim x as integer
x = 8
y = “Today “ & “is “ & “Tuesday the “ & x & “-th of March“

⇒ y = “Today is Tuesday the 8-th of March“

· here the individual components do not have to be of string type (8 is an integer)

85

Customized User Forms (CUF)

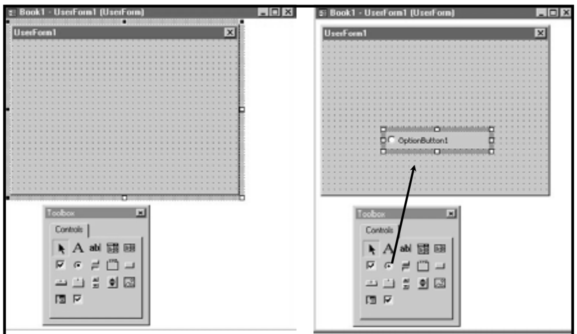
➤ CUF are user defined dialog boxes
(similar to MsgBox and InputBox, but far more flexible)

- CUF can be used to display, enter and modify informations to the Excel worksheet as well as to the VBA program
- Expl.: most standard windows dialog boxes, such as setup windows, wizard windows etc.

Creating and designing a CUF:

i) open the user form inside the VB editor
   - select Insert → UserForm ↵
     ⇒ a new user form object with the default name “UserForm1“ is created
   - if the UserForm is not visible:
     select View → Toolbox ↵

86



87

ii) add controls to the user form
   - from the toolbox drag and drop the icon of a particular control to the UserForm
   - move and resize the control
   - having several controls, they might influence each other
   - possible options for CUF controls:

Label ≡ A Text added to the CUF to provide general information.

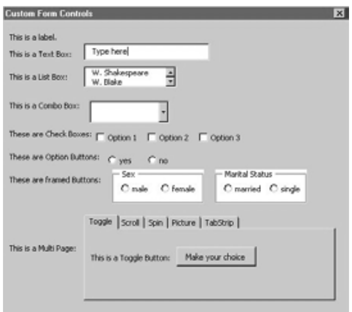
CommandButton ≡ A button that initiates an action.

TextBox ≡ A box in which you can type text.
The text can be linked to a cell on the worksheet.

ListBox ≡ A box that contains a list of items.
The text can be linked to a cell on the worksheet

Combo box ≡ A text box with a drop-down list box.
You can either type or select a choice in the box.
The text can be linked to cells.

88



89

Check Box ≡ An option you can turn on or off by selecting or clearing it.
More than one check box selected at a time is possible.

Option Button ≡ A button used to select only one of a group of options.

Frame ≡ A panel in which groups of related controls are organized.

ScrollBar ≡ A control that scrolls through a range of values when you click the scroll arrows or when you drag the scroll box.

SpinButton ≡ A button that can be attached to a cell or a text box.
Selecting the up arrow increases the value and selecting the down arrow decreases the value.

MultiPage ≡ A page that allows you to organize controls in form of several pages.

TabStrip ≡ Displays Tabs you can use to organize other controls.

Toggle button ≡ A button that remains pressed in when selected, and then releases when it is clicked again.

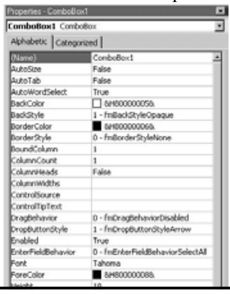
RefEdit ≡ Enables you to select a range in the worksheet.

Image ≡ A control that embeds a picture into a form.

90

iii) modify and adjust the entries in the Properties Window

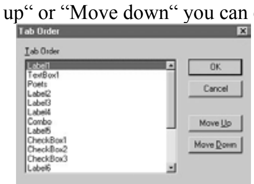
- depending on the selected control, the Properties Window contains various types of properties (see examples)
- if the Properties Window is not visible:
  select View → Properties Window ↵



91

iv) check and modify the tab order

- the tab order is the sequence in which the controls receive the focus
- inside the VB editor select View → TabOrder ↵
  ⇒ a window displaying the controls in a particular order opens
- with “Move up“ or “Move down“ you can change this order

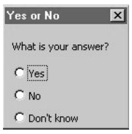


v) adjust the VBA-code to your needs

- inside the VB editor select View → Code ↵
- program now the code according to your needs

92

Expl.:



Create a CUF with title “Yes or No“ and a label saying “What is your answer?“ The form should contain three OptionButtons with text: “Yes“, “No“, “Don‘t know“. When “Yes“ is selected write “Y“ into the cell “D10“, when “No“ is selected “N“ and “0“ for “Don‘t know“.

<u>Running a user form</u>:

- select “Run“ as for “Sub“
- or select function key F5
- or attach a Commandbutton to it on a worksheet

93

- open a user form
- add a label with text “What is your answer?“
- add three OptionButtons with specified test
- change the caption in the UserForm Properties Window to “Yes or No“
- view the code of the UserForm, it will say
- Private Sub UserForm_Click()

```
Private Sub UserForm_Click()


End Sub
```

· “private“ means the procedure can only be used in the current workbook

· “Click“ indicates which event will run this code (in the Code window there is a ListBox which offers other possibilities, such as “DblClick“, etc.)

94

- complete the code as follows

```
Private Sub UserForm_Click()
    If OptionButton1.Value Then
        Range("D10").Value = "Y"
    ElseIf OptionButton2.Value Then
        Range("D10").Value = "N"
    ElseIf OptionButton3.Value Then
        Range("D10").Value = 0
    End If
End Sub
```

· OptionButton1, OptionButton2, OptionButton3 are the names of the OptionButtons. Depending on whether the Option is selected or not they are returned as “True“ or “False“, respectively.

95

Customized User Forms (II)

• <u>SpinButtons:</u>

- add a SpinButton to a user form
- important properties of the SpinButton are:
  · “Min“ and “Max“ are the values which define the interval in which the Spinvalues are varied
  · “SmallChange“ defines the step size by which the Spinvalue varies
  · “ControlSource“ links the value to a cell on the worksheet
- to link the SpinButton value to a TextBox, change the code as :

```
Sub SB1_change()
    TB1.Value = SB1.Value
End Sub
```

the name of the SpinButton is SB1

the name of the TextBox is TB1

96

Expl.: Create a CUF with title “Trigometric Functions“. The form should have a SpinButton which allows to vary a value x from 0 to 2π. This value should be displayed in a TextBox. The form should have three more TextBoxes which display the sin(x), cos(x) and tan(x).

- add a SpinButton to the user form
- change its name to “SB1“
- in the Properties Window set “Min“ to “0“, “Max“ to “200“ and “SmallChange“ to “5“
  (now when we click though the SpinButton it takes on the values 0,5,10,15,....190,195,200)
- add five labels with text “x=“, “sin(x)=“, “cos(x)=“,“tan(x)=“, “π“. To be able to write “π“, select in the Properties Window font “Mathematica1“, then type p
- add four TextBoxes named “TB1“, ... , “TB4“

97

• <u>ListBox:</u>

- add a ListBox to a user form
- important properties of the ListBox are:
  · “ControlSource“ links the selected value to a cell on the worksheet
  · “RowSource“ fills the list displayed in the ListBox
  (e.g. put a1:a20 then the list will contain the values in there)
- alternatively you can fill the list with an array in the VB code
-Expl.:

```
Private Sub UserForm_Click()
  Dim pp As Variant
  pp = Array("W. Shakespeare", "W. Blake", "J.W. von Goethe",
                  "F. Schiller", "Dante", "Cervantes", "Homer")
  Poets.List = pp
End Sub
```

the name of the ListBox is Poets

98

• <u>ComboBox:</u>

- add a ComboBox to a user form
- important properties of the ComboBox are:
  · “RowSource“ fills the list displayed in the ListBox
  (e.g. it could be two columns, say a1:b20)
  · “ControlSource“ links the selected value to a cell on the worksheet
  · “ColumnCount“ is the number of values displayed in the ComboBox
  (e.g. when you have more than one you might just want to display a few of them)
  · “BoundColumn“ denotes the number of the column related to the value of the ComboBox
  (e.g. 2 could be the second column out of 5)

99

- Expl.: we have the following values stored in two columns:

W. Shakespeare	1564
W. Blake	1757
J.W. von Goethe	1749

· setting now BoundColumn = 2 , ColumnCount = 1 has the effect that the names will be displayed in the ComboBox, but not the birth years related to the value
· change the VB code to:

```
Private Sub CoB1_Change()
    Range("k14").Value = CoB1.Value
End Sub
```

· the name of the ComboBox is CoB1 here
· the birth year is stored in CoB1.Value
· this value is then associated to the cell k14

100

• <u>ToggleButton:</u>

- add a ToggleButton to a user form
- important properties of the ToggleButton are:
  · the name of the Button is associated to the boolean values “true“ or “false“ which you can use in the VB program
- Expl.:

the name of the ToggleButton is ToB

```
Private Sub ToB_Click()
  If ToB Then
    Range("c2").Value = "Toggle is yes"
  Else
    Range("c2").Value = "Toggle is no"
  End If
End Sub
```

101

<u>Announcements</u>

- ➤ There are no more Lab sessions!
- ➤ The exam will take place
  11-th of May 2005
- ➤ The entire lecture and the Lab-sessions including the solutions can be obtained from
  http://www.staff.city.ac.uk/~fring/ExcelVBA/index.html

102