EXCEL-VBA

Mitschrifthilfe zum Kurs **für Einsteiger - 1. Auflage**

Tag 1 (Das Kennenlernen)

- VBA Editor
- Bevor es losgeht!
- Klassen/Objekte
- Objektkatalog
- Prozeduren - Allgemein
- Übungen und Hausaufgaben

Tag 2

- Prozeduren
  - Gültigkeitsbereiche
  - Deklaration
  - Initialisierung
- Excel Klassen
  - Application
  - Workbook/-s
  - Worsheet/-s
  - Range/Cells

Tag 3

- Bedingte Anweisungen
  - If-Else
  - Select-Case
  - Operatoren
    - Vergleich-
    - Logik-

Tag 4

- Schleifen
  - Do…Loop
  - For…Next

** Alle Angaben basieren auf Office 2013 - Abweichungen in anderen Versionen möglich*

** (→) = nächster Schritt; (…) = ausgelassener Text; (Kursiv) = Erklärungen/Anmerkungen*

Kontakt: sebastian.lebioda@hs-merseburg.de

EXCEL-VBA

Mitschrifthilfe zum Kurs **für Einsteiger - 1. Auflage**

Tag 5+6+7 (Komplexaufgabe 1 - Chromatogramm)

- Einfaches numerisches Differenzieren
- Messwerte vergleichen und eingrenzen
- WorksheetFunctions anwenden
- Einfaches numerisches Integrieren

Tag 8+9 (Komplexaufgabe 2 - Regression)

- Solver
- UserForm

Tag 10 Arrays

- Deklaration/Initialisierung

** Alle Angaben basieren auf Office 2013 - Abweichungen in anderen Versionen möglich*

** (→) = nächster Schritt; (…) = ausgelassener Text; (Kursiv) = Erklärungen/Anmerkungen*

Kontakt: sebastian.lebioda@hs-merseburg.de

Tag 1

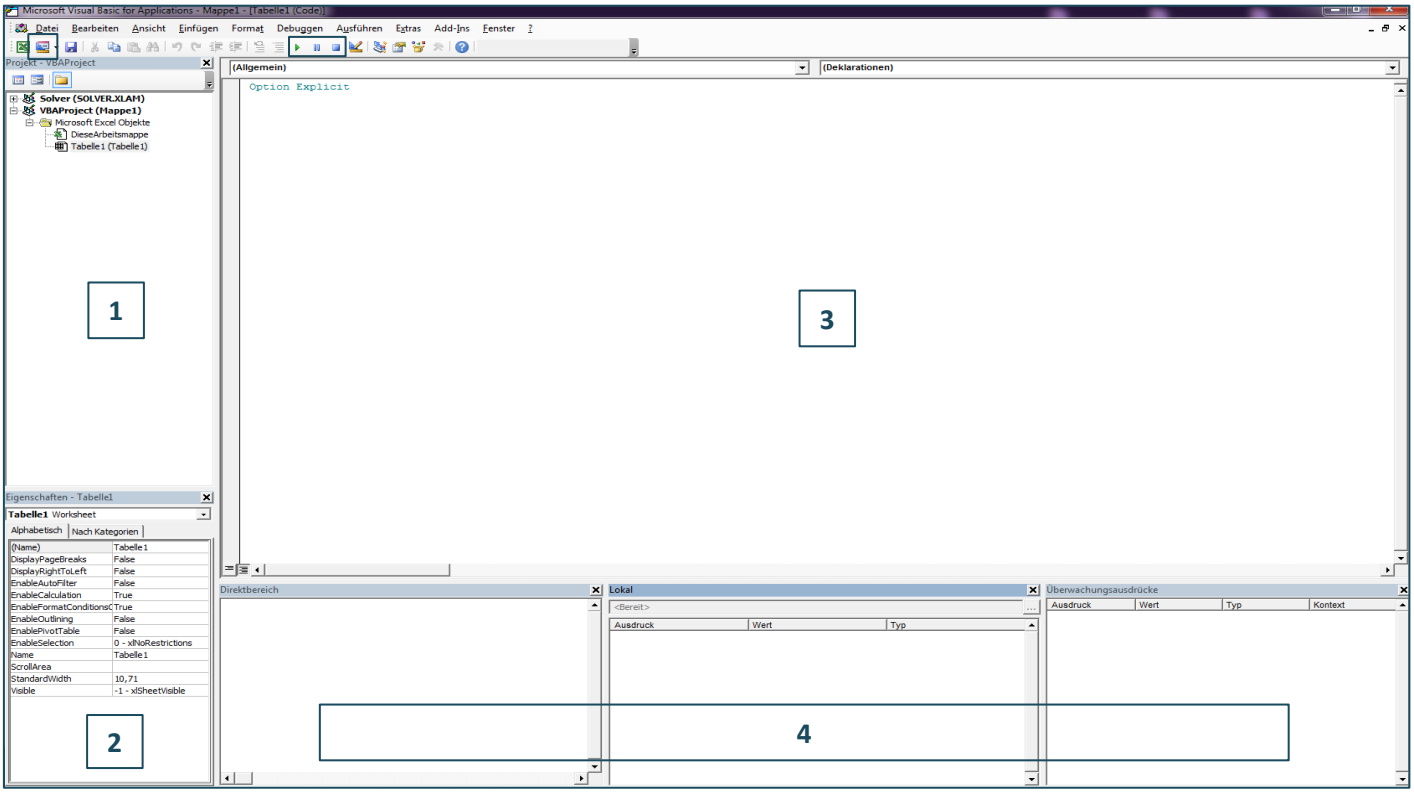
VBA-Editor

Aktivierung der Registerkarte "Entwicklertools"

-> Excel Optionen -> Menüband anpassen -> Häkchen beim entsprechenden Reiter

Öffnen des VBA-Editors

-> Symbol oder (Alt+F11)



* [1] Projektexplorer: Organisation der Objekte die Teil eines Projekts sind
* [2] Eigenschaften der Projektobjekte
* [3] Programmierbereich
* [4] Diverse Fenster als Programmierunterstützung
* Einfügen neuer Projektobjekte
* Programm starten, unterbrechen oder zurücksetzen

Bevor es losgeht !

* **Hilfreiche Schaltflächen aktivieren**
  Einrücken: Für bessere Code-Übersichtlichkeit
  Auskommentieren: Für Testzwecke

-> im Editor "Ansicht" -> "Symbolleisten" -> Anpassen -> Symbolleiste -> Häkchen bei "Bearbeiten"

* **Option Explicit einschalten**
  Statement im Modulkopf: Für saubereres programmieren

-> im Editor "Extras" -> Optionen -> Reiter (Editor) -> Häkchen bei Variablendeklaration erforderlich

* **Fenster aktivieren**
  Für debugging und zur Erkenntnisgewinnung

-> im Editor "Ansicht" -> "Symbolleisten" -> Lokal-, Direkt- und Überwachungsfenster auswählen

Urheber: Sebastian Lebioda - HS Merseburg

Tag 1

Klassen/Objekte

Notizen

* VBA (Visual Basic for Applications) ist ein spezialisierter Ableger für Office-Produkte der Programmierungsprache Visual Basic.

* VBA ist objektorientiert aufgebaut (OOP-ObjektOrientierte Programmierung), d.h. das alle Möglichkeiten als unterschiedliche Strukturen vorliegen, die miteinander hierarisch verknüft sein

* Diese Strukturen werden in der OOP Klasse genannt und sind eine Art Konstruktionsbezeichnung für ein Objekt - ähnlich unserer Gene (Klassen), mit denen z.B. Muskeln (Objekte) gebildet werden können

* In einer Klasse werden Eigenschaften, Methoden und Ereignisse beschrieben
  - Eigenschaft Muskel: rot, weiß, faserartig, biegsam, …
  - Methoden Muskel: wachsen, schmerzen, …
  - Ereignisse Muskel: Längenkontraktion durch Nervenimpuls, …

* Klassenstruktur ist durch Vererbung, Assoziation,… organisiert
  Vererbung: Eine Klasse erbt Eigenschaften einer Basisklasse

Grafische Darstellung einer Klasse nach UML (Unified Modeling Language)

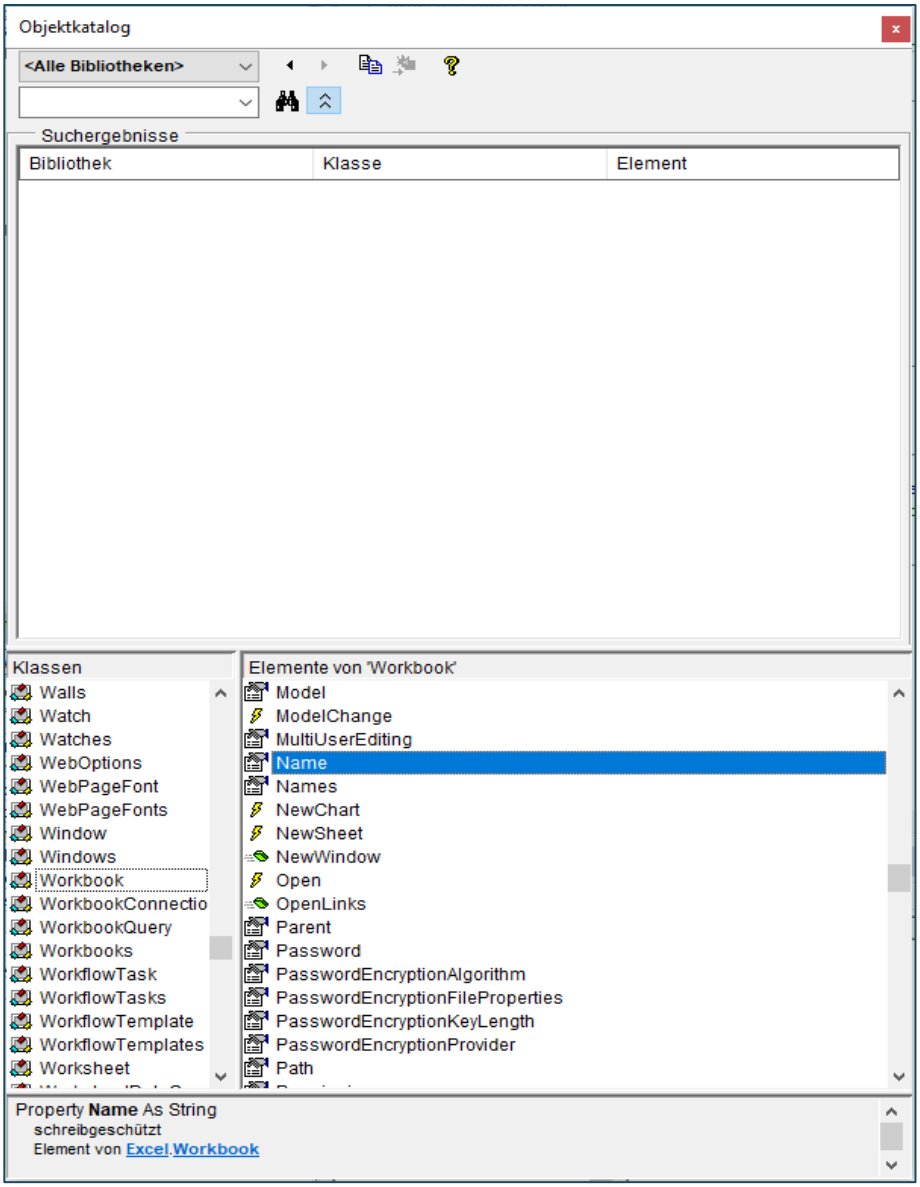
Klasse	Muskel	Workbook
Attribute 	Farbe Aufbau Beweglichkeit	Name HasPassword Saved
Methoden (Operationen) 	Wachsen() Schmerzen() Kontraktieren()	NewWindow() SaveAs()

Tag 1

Objektkatalog

Notizen

* Die Organisation aller Klassen ist im **Objektkatalog** dargestellt
  -> im VBA Editor Symbol  klicken
* Klassen werden in Bibliotheken gesammelt



Übung: Objektkatalog (Zusammen)

* Schreiben Sie in den Programmierbereich des VBA-Editors folgende Sub-Prozedur

```
Sub Objektkatalog1()
    Debug.Print DateTime.Minute(DateTime.Time)
    'Debug.Print DateTime.Time
    'Debug.Print WorksheetFunction.IsNumber(10)
    'Debug.Print Math.Exp(1)
End Function
```

* (') mit Hochkomma angeführter Text, wird vom Compiler ignoriert

Tag 1

Prozeduren

Notizen

* Innerhalb einer Prozedur wird der Programmcode ausgeführt

Allgemeiner Prozeduraufbau

```
[Gültigkeit]Prozedurname(Übergabepara1,...)|[Proz.namendekl.]
    Dim Variable1 [, VariableX] As Datentyp      'Deklarationsteil

    Variable1 = Wert                             'Initialisierungsteil
    VariableX = Variable1 * Wert

    ...                                          'Aufgabenteil
    ...

   [Exit Prozedur]
End Prozedur
```

* Prozedurnamen dürfen nur einmal vergeben werden und müssen mit einem Buchstaben beginnen, innerhalb des Gültigkeitsbereichs eindeutig sein und nur Buchstaben, Ziffern und den Unterstrich enthalten

* Prozedurnamen dürfen keine Schlüsselwörter sein (Public,Sub,..)

Sub-Prozedur

```
Sub Hypotenuse1([Übergabeparameter])
    Dim Kat, Ankat, Hypo As Integer

    Kat = 1
    Ankat = 2

    Hypo = Math.Sqr(Kat ^ 2 + Ankat ^ 2)
End Sub
```

Function-Prozedur

* Functions haben einen Rückgabewert der gleich Functionname ist. Dieser wird im Prozedurkopf deklariert.

* Keine Return-Anweisung möglich!

```
Function Hypotenuse2([Übergabeparameter]) As Double
    Dim Kat, Ankat As Integer

    Kat = 1
    Ankat = 2

    Hypotenuse2 = Math.Sqr(Kat ^ 2 + Ankat ^ 2)
End Function
```

Tag 1

Übungen und Hausaufgaben

Notizen

Übung1_1: Kreisfläche berechnen (Zusammen)

Erstellen Sie eine Sub-Prozedur ohne Übergabeparameter die zur Berechnung einer Kreisfläche dient. Nutzen Sie dazu das vorherige Beispiel und den Haltemodus, um das Ergebnis zu überprüfen.

Übung1_2: Schnittpunkt zweier Geraden (Selbstständig)

Erstellen Sie eine Sub-Prozedur ohne Übergabeparameter die den Schnittpunkt zweier Geraden ausrechnet. Achten Sie auf unterschiedliche Vorzeichen der Steigungen.

Bsp: y_1=3x+10 und y_2=-3x+20

Übung1_3: Kreisfläche berechnen (Zusammen)

Berechnen Sie die Kreisfläche wie in Übung 1_1, nur jetzt in einer Function-Prozedur. Der Durchmesser soll als Übergabeparameter an die Funktion übergeben werden.

Übung1_4: Schnittpunkt zweier Geraden (Selbstständig)

Berechnen Sie den Schnittpunkt zweier Geraden wie in Übung1_2 nur jetzt in einer Function-Prozedur. Achten Sie darauf das sie 2 Prozeduren benötigen.

Übung1_5: Beladung <-> Anteil

Erstellen Sie 2 Functions die Beladung in Anteil und umgekehrt ausgeben.

Übung1_6: Nullstellen Quadr. Funktion

Erstellen Sie Functions die die Nullstellen einer Quadratischen Funktion ausgeben (pq-Formel). Übergeben Sie die Koeffizienten a, b und c an die Funktionen.

Allgemeine Gleichung: y = a*x^2 + b*x + c

Beispielgleichung: y = 3*x^2 - 6*x + 1 Lsg.: x1=1,816|x2=0,184

Tag 2

Prozeduren

Notizen

Gültigkeitsbereiche

* Zur Erinnerung - Aufbau eines Prozedurkopfes (siehe Tag1)

```
[Gültigkeit]Prozedurname(Übergabepara1,...)|[Proz.namendekl.]
    Gültigkeit Variable1 [, VariableX] As Datentyp
    ...

End Sub
```

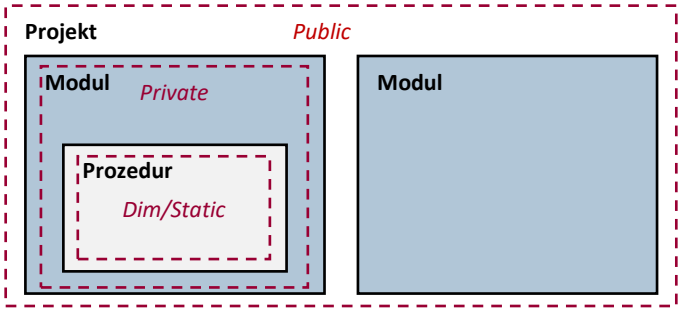
* Die Erreichbarkeit und Lebensdauer von Prozeduren/Variablen können über folgende Schlüsselwörter verändert werden.

`[Gültigkeit]` von Prozeduren (Sub und Function)

`Private`	* Prozedur ist nur innerhalb eines Moduls aufrufbar
`Public`	* ist gesetzt, wenn nichts angeben ist * Prozedur ist von überall im Projekt aufrufbar * hat in Userform keine globale Gültigkeit!

`Gültigkeit` von Variablen

`Dim/ReDim`	* in einer Prozedur deklariert: nur innerhalb der Prozedur gültig. * Variable ist nach Prozedurablauf aus dem Speicher * ReDim wird für die dyn. Änderung der Größe eines Arrays verwendet.
`Static`	* in einer Prozedur deklariert: nur innerhalb der Prozedur gültig. Variable ist nach Prozedurablauf nicht aus dem Speicher * Variable ist erst aus dem Speicher, wenn das Programm zurückgesetzt wurde
`Private`	* Privatedeklarierte Variablen müssen im Modulkopf, über sämtlichen Prozeduren deklariert werden * sind nur innerhald des Moduls gültig * Variable ist erst aus dem Speicher, wenn das Programm zurückgesetzt wurde
`Public`	* Publicdeklarierte Variablen müssen im Modulkopf, über sämtlichen Prozeduren deklariert werden * Variable ist überall im Projekt gültig * Variable ist erst aus dem Speicher, wenn das Programm zurückgesetzt wurde



Tag 2

Prozeduren

Notizen

Deklaration

```
[Gültigkeit]Prozedurname(ByRef/ByVal Übergabeparameter1 As ...)
    Gültigkeit Variable1 [, VariableX] As Datentyp
    .
End Sub
```

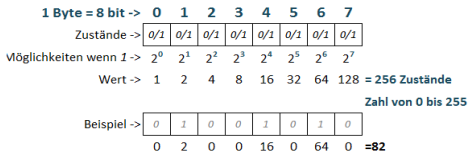
ByRef
* Wenn nichts steht, ist ByRef aktiv (Standard)
* der Übergabeparameter wird durch einen **Verwei**s gespeichert, dadurch haben Änderung an ihm überall Bestand

ByVal
* Der **Wert** des Übergabeparameters wird gespeichert, dadurch haben Änderung an ihm keinen Bestand

```
[Gültigkeit]Prozedurname(ByRef/ByVal Übergabeparameter1 As ...)
    Gültigkeit Variable1 [, VariableX] As Datentyp
    .
End Sub
```

* für effizientes programmieren, muss der Größenbereich der Variable als Datentyp angegeben werden
* Datentypen benötigen aufgrund ihres Wesens unterschiedlich viel Speicherplatz

Darstellung eines Bytes



Wichtigsten Datentypen in VBA [1]

Datentyp	Speicherbedarf in Byte	Bereich	Möglicher Präfix
Boolescher Wert	2	True oder False	bln
Byte	1	0 bis 255	byt
Single	4	-3,402823 E38 bis -1.401298 E-45 1.401298 E-45 bis 3,402823 E38	sng
Double	8	-1,79... E308 bis -4,94... E-324 4,94.... E-324 bis 1,79... E308	dbl
Integer	2	-32 768 bis 32 767	int
Long	4	-2 147 483 648 bis 2 147 483 647	lng
String	10 Byte + 2 Byte/Zeichen	0 bis ca. 2 Milliarden	str
Variant mit Zahlen	16	bis Bereich von Double	var
Variant mit Zeichen	32bit-> 22Byte+L 64bit-> 24Byte+L	0 bis ca. 2 Milliarden	
Objekt	4	beliebiger Objektverweis	obj

* Präfixe dienen der besseren Code-Lesbarkeit und werden der Variable vorangestellt. z.B. intCounter, objRange1,…

Kein Muss aber gängig!
Weitere Regeln, suche "Namenskonvention VBA"

1 Microsoft - OnlineHilfe: https://docs.microsoft.com/de-de/office/vba/language/reference/user-interface-help/data-type-summary

Tag 2

Prozeduren

Notizen

Initialisierung

* Beim Initialisieren werden den Variablen Startwerte zugewiesen

```
[Gültigkeit]Prozedurname(Übergabepar1,...)|...
    Dim Variable1 [, VariableX] As Datentyp

    Variable1 = Wert
    .
    .
```

* Beim Initialisieren von Objekten (Objektvariablen) muss die Set - Anweisung benutzt werden

```
    Dim objVariable As Objekt

    Set objVariable = Objekt
    Set objVariable = [New] Objekt

    Set objvariable = Nothing
```

* Das Erzeugen eines Objekts von einer Klasse heisst Instanziierung
* Das erzeugte Objekt ist dann eine Instanz dieser Klasse
* Es können mehrere Instanzen der selben Klasse erzeugt werden
* Objekte sollten immer nach dem Gebrauch entladen werden!!!
* Argumente in Methoden z.B. *Sub Name(FileName, FileFormat,…* lassen sich auch so schreiben:
  Sub SubName(FileName, FileFormat,…
  Sub SubName FileName:="C:\..." , FileFormat:=52, …

With Anweisung

* Lange Objektstrukturen lassen sich so verkürzen

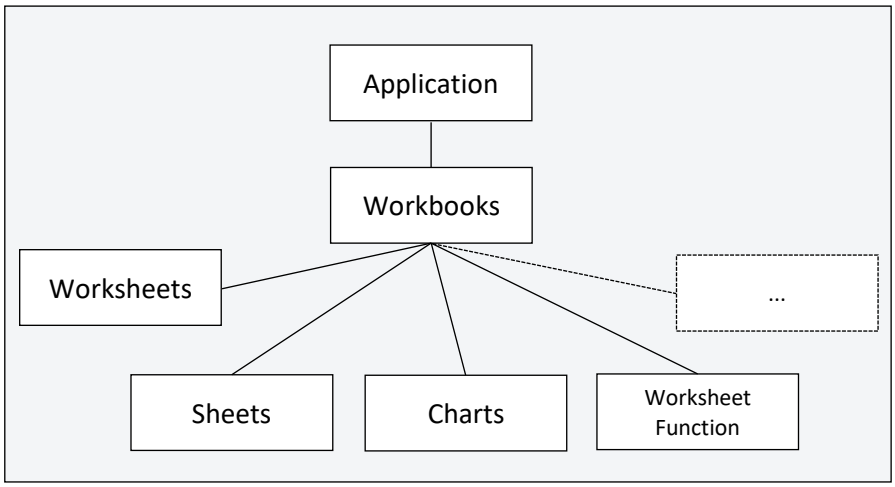
```
statt:
    ThisWorkbook.Worksheets(1).Range("A1").Value
    ThisWorkbook.Worksheets(1).Range("A2").value
    ThisWorkbook.Worksheets(1).Range("A3").value
    ThisWorkbook.Worksheets(1).Range("A4").Value


so:
With ThisWorkbook.Worksheets(1)
    .Range("A1").Value
    .Range("A2").Value
    .Range("A3").Value
    .Range("A4").Value
End With
```

Tag 2

Excel Klassen/Objekte (Auswahl)

Notizen



Application

* stellt Eigenschaften und Methoden zur Verfügung die Excel insgesamt betreffen

.Wait	Unterbricht den Programmcode bis zu einer bestimmten Uhrzeit
.Calculation	Steuert die Arbeitsmappenberechnung
.ScreenUpdating	Steuert die Bildschirmaktualisierung
.Workbooks	
.ThisWorkbook	
.ActiveWorkbook	
.WorksheetFunction	
.Worksheets	
:	

[Strg+Untbr] unterbricht den Programmcode!!!

Rückgabewert ist ein Objekt

Workbooks

* (-s) bedeutet, dass es sich um ein Collection-Objekt handelt
* Verwaltung aller geöffneten Excel-Anwendungen

.Add	Erstellt eine neue Arbeitsmappe
.Open	Öffnet eine Arbeitsmappe
.Close	Schließt eine Arbeitsmappe
.Count	Anzahl aller geöffneten Excel-Anwendungen
.SaveAs	Speichert eine Arbeitsmappe
:	

.Add - gibt ein Workbook-Objekt zurück

ThisWorkbook

* Eigenschaft der Application-Klasse und gibt ein Workbook-Objekt zurück
* Representiert die Arbeitsmappe in der der aktuelle Code läuft

ActiveWorkbook

* Eigenschaft der Application-Klasse und gibt ein Workbook-Objekt zurück
* Representiert die aktive (ausgewählte) Arbeitsmappe

Tag 2

Excel Klassen/Objekte (Auswahl)

Notizen

Worksheets

* (-s) bedeutet, dass es sich um ein Collection-Objekt handelt
* Verwaltung aller vorhandenen Tabellenblätter einer Arbeitsmappe

.Add	Erstellt ein neues Tabellenblatt
.Copy	Kopiert ein Tabellenblatt
.Move	Bewegt ein Tabellenblatt
.Visible	Steuert die Sichtbarkeit eines Tabellenblattes

Worksheet

* Ist ein Element der Worksheets-Klasse
* Wird angesprochen über Position, Tabellenblattnamen oder Codenamen

Die Sheets-Klasse ist analog zur Worksheets-Klasse, nur das die Sheets-Klasse noch Chart-Sheets beinhaltet

```
Set objSheet = Worksheets(1)
Set objSheet = Worksheets("Eingabenblatt")
Set objSheet = Tabelle1      (Codename nicht Tabellenblattname!!!)
```

.Activate	Aktiviert ein Tabellenblatt
.Add	Erstellt ein neues Tabellenblatt
.Delete	Löscht ein Tabellenblatt
.UsedRange	Bereich in einem TB der Werte enthält
.Range	Bereich in einem Tabellenblatt
.Cells	Bereich in einem Tabellenblatt (Koordinaten)

Worksheets-Eigenschaften die ein Range-Objekt zurückgeben

Range

* Ist ein Element der Worksheet-Klasse
* Wird angesprochen über ("SpaltenbuchstabeZeile")

```
Set objRange = Worksheets(1).Range("A1")
Set objRange = Worksheets(1).Range("B1:B10")
Set objRange = Worksheets(1).Range("C1, C5, C10")
Set objRange = Worksheets(1).Range("D:D")
Set objRange = Worksheets(1).Range("5:5")
```

.Value	Wert einer Zelle (variabler Datentyp)
:	

Auch interessant: Application.Union - Function

Cells

* Ist ein Element der Worksheet-Klasse
* Wird angesprochen über (Zeile, Spalte)

```
Set objRange = Worksheets(1).Cells(1,1)
Set objRange = Worksheets(1).Range(Cells(1, 2), Cells(10, 2))
```

.Value	Wert einer Zelle (variabler Datentyp)

Tag 3

Bedingte Anweisungen [1]

If

```
If Bedingung Then
   Anweisungen für Bedingung = Wahr
End If
.............................................

If Bedingung Then Anweisungen für Bedingung = Wahr
```

* Besteht die Anweisung für Bedingung aus nur einer Zeile, dann o. End If

If-Else

Zweiseitig bzw. Mehrseitig

```
If Bedingung Then
   Anweisungen für Bedingung = WAHR
[Else]
   [Anweisungen für Bedingung = FALSCH]
End If
```

Verschachtelt

```
If Bedingung1 Then
    If Bedingung2 Then
       Anweisungen für Bedingung2 = WAHR
       ...
       ...
    Else
       [Anweisungen für Bedingung2= FALSCH]
    End If
Else
    [Anweisungen für Bedingung1 = FALSCH]
End If
```

* WAHR-gesteuert

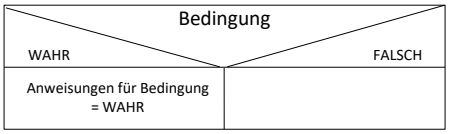
Mehrstufig

```
If Bedingung1 Then
    Anweisungen für Bedingung1 = WAHR
ElseIf Bedingung2 Then
    Anweisungen für Bedingung2 = WAHR
    .
    .
Else
    [Anweisungen für Bedingung1+2. = FALSCH]
End If
```

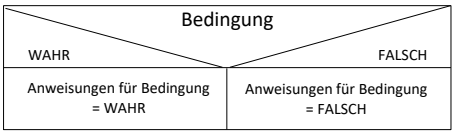
* FALSCH-gesteuert

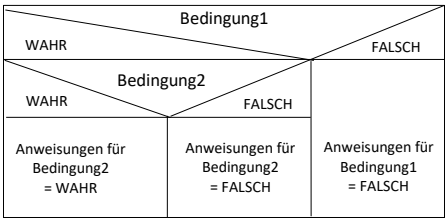
Notizen

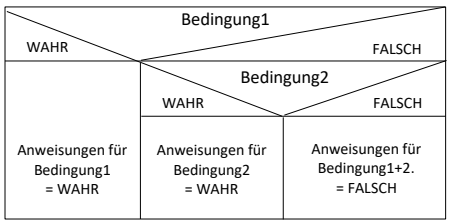
Struktogramm (Nassi-Shneidermann-Diagramme



Vorsicht bei der Prüfung von Functions mit booleschen Rückgabewert. Eine zusätzliche Prüfung auf True/False ist unnötig!







[1]: "Grundkurs Programmieren mit Visual Basic", Heinrich Wimmer, Hanser Verlag 2009, Seite 70 bis 74

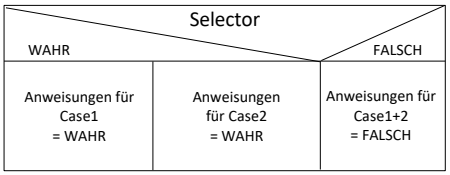
Tag 3

Bedingte Anweisungen

Notizen

Select Case

```
Select Case Selector
    Case1 Bedingung
          Anweisungen
    Case2 Bedingung
          Anweisungen
          .
          .
    Case Else
          Anweisungen
End Select
```



* Verwendung vorwiegend bei mehrseitiger Auswahl (Fallauswahl)

Operatoren [1]

Vergleichsoperatoren

	True wenn,
a < b	a kleiner b
a <= b	a kleiner gleich b
a > b	a größer b
a >= b	a größer gleich b
a = b	a gleich b
a <> b	a ungleich b
Is	Vergleich von zwei Objektreferenzen Bei Gleichheit Ergebnis = True
Like	Vergleich von zwei Zeichenfolgen

Logische Operatoren

Ausdrücke						
a	**b**	**And**	**Or**	**Xor**	**Not A**	**Eqv**
True	**True**	*True*	*True*	*False*	*False*	*True*
True	**False**	*False*	*True*	*True*	*False*	*False*
False	**True**	*False*	*True*	*True*	*True*	*False*
False	**False**	*False*	*False*	*False*	*True*	*True*

[1] https://docs.microsoft.com/de-de/office/vba/language/reference/user-interface-help/operator-summary

Tag 3

Übungen

Notizen

Übung3_1: Zahlprüfung (Zusammen und Selbstständig)

Erstellen Sie eine Zelle für die Eingabe einer Zahl. Diese soll auf folgende Eigenschaften geprüft werden.: Ist der Wert eine…

* Zahl?, Größer 0?, Größer 50?, Größer 100?;
* zwischen 0 und 50?, zwischen 50 und 100?, Größer 100?
* Positiv oder Negativ? und eine ungerade Zahl?

Nutzen Sie dafür die Bedingungen aus den vorherigen Seiten.

Übung3_2: Bedingung der Übung1_2 hinzufügen (Selbstständig)

Fügen Sie der bereits erstellten Sub-Prozedur aus Übung 1_2 eine Prüfung hinzu, die den Fehler abfängt, wenn die Steigungen der Geraden gleich groß sind.

Übung3_3: Top of 3

Bauen Sie eine Function, die aus einer Auswahl von 3 Zahlen die höchste ausgibt. Nutzen Sie dafür die If-Else-Anweisung … und nicht die Max-Function.

Übung3_4: Würfelkombinationen ausrechnen (Kniffel)

Rechnen Sie entsprechend der "Kniffel"-Regeln die Punkte aus, die pro Wurf theoretisch möglich sind.
Nutzen Sie dafür eine Sub- oder wahlweise mehrere Function Prozeduren und natürlich bedingte Anweisungen. Viel Spass!

5 Würfel

1	2	3	4	5

Gewinnkombinationen (x -> beliebiger Würfel)

1	1	1	x	x	1-6 -> Dreierpasch -> Summe
1	1	1	1	x	1-6 -> Viererpasch -> Summe
1	1	2	2	2	Full House -> 2 + 3 gleiche Würfel = 25Pkt.
1	2	3	4	x	kl. Str. -> 4 Würfel in einer Reihe = 30Pkt.
1	2	3	4	5	gr. Str. -> 5 Würfel in einer Reihe = 40Pkt.
1	1	1	1	1	Hauptgewinn -> 5 gleiche Würfel = 50Pkt
1	1	2	2	2	Chance -> Summe

Beispiel

5	3	5	5	4	Dreierpasch mit 5 = 22Pkt.
4	1	3	6	5	kleine Strasse mit 5 = 30Pkt.
2	5	2	5	2	Full House = 25Pkt.

Tag 4

Schleifen

Notizen

Do Loop - Kopfgesteuert

```
Do [{ While | Until } Bedingung]
     [Anweisungen]
     [Exit Do ]
     [Anweisungen]
Loop
```

[Strg+Untbr] unterbricht den Programmcode!!!

Do Loop - Fußgesteuert

```
Do
     [Anweisungen]
     [Exit Do]
     [Anweisungen]
Loop [{ While | Until } Bedingung]
```

* Für Aufgaben die kein definiertes Ende haben
  z.B. Grenzwertaufgaben, Iterationen, usw.

For Next

```
For Zaehler = start To end [ Step Schrittweite ]
        [Anweisung]
        [Exit For ]
        [Anweisung]
Next [ Zaehler ]
```

* Für Aufgaben die ein definiertes Ende haben
  z.B. Messwerte bearbeiten, Bereiche auswerten, usw.
* Vorsicht: Zähler hat nach Beendigung den Wert= Endwert + 1

For Each Next

```
For Each element In group
        [Anweisung]
        [Exit For]
        [Anweisung]
 Next [elemene]
```

* Für die Bearbeitung von Elemente einer Auflistung (Collection)
  z.B. Workbook In Workbooks, Arrays, usw.

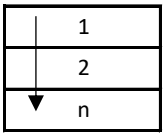
Tag 4

Übungen

Notizen

Übung4_1: Schleife 1

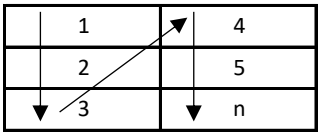
Bauen Sie eine Schleife nach folgendem Muster



Sie können die Kenntlichmachung durch Werte bzw. Hintergrundfarben realisieren

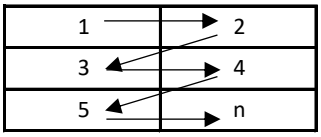
Übung4_2: Schleife 2

Bauen Sie eine Schleife nach folgendem Muster



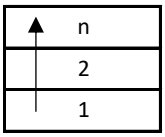
Übung4_3: Schleife 3

Bauen Sie eine Schleife nach folgendem Muster



Übung4_4: Schleife 4

Bauen Sie eine Schleife nach folgendem Muster



Tag 5+6

Komplexaufgabe 1

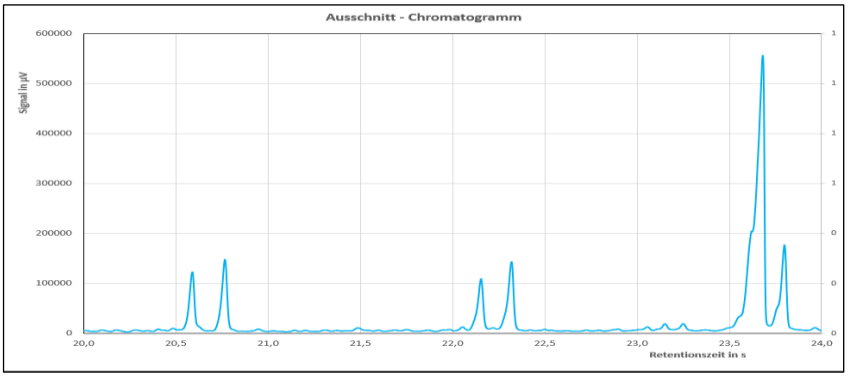
Notizen

Betrachten Sie die letzten Tage als Bestückung ihres Werkzeugkastens mit den für sie notwendigsten Anweisungen die VBA zur Verfügung stellt. Nun gilt es all das zu nutzen und auf praktische Aufgaben anzuwenden.

In der ersten Komplexaufgabe soll ein Chromatogramm ausgewertet werden. Folgende Ansprüche an das Programm werden festgesetzt:

- Identifikation alles Peaks und die Angabe aller Maxima
- Berechnung der Peakflächen
- Filtermöglichkeit nach Peakhöhe und Peakfläche

Das folgende Diagramm enthält einen Ausschnitt des beispielhaften Chromatogramms



Zur Bewältigung solch einer Aufgabe ist es ratsam Teilaufgaben zu erzeugen. Da es in der Regel mehrere Lösungswege gibt, ist der folgende nur exemplarisch zu betrachten.

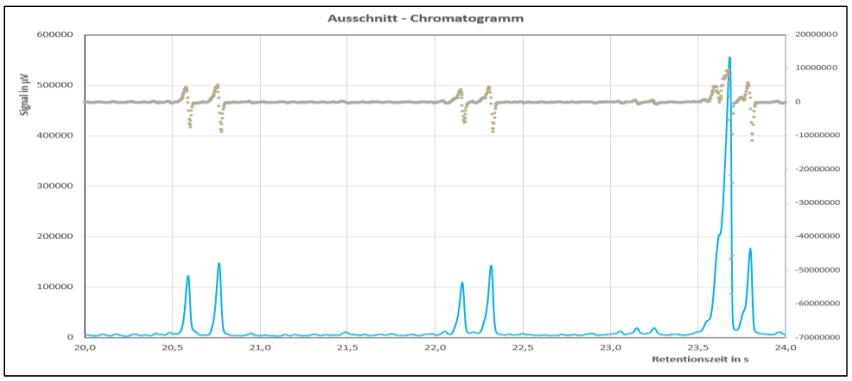
Teilaufgaben:
1. Steigung der Signalwerte über n Punkte errechnen.
2. Innerhalb der Steigungen die Zeitintervalle der Nulldurchgänge bestimmen.
3. Mit Hilfe der Zeitintervalle die Peakmaxima in den Rohdaten suchen.
4. Ausgehend der Peakmaxima, den Peak links - bzw. rechtsseitig integrieren (Peakfläche berechnen).
5. Peaks abhängig von Höhe und Fläche gefiltert anzeigen.

Tag 5+6

Komplexaufgabe 1

Notizen

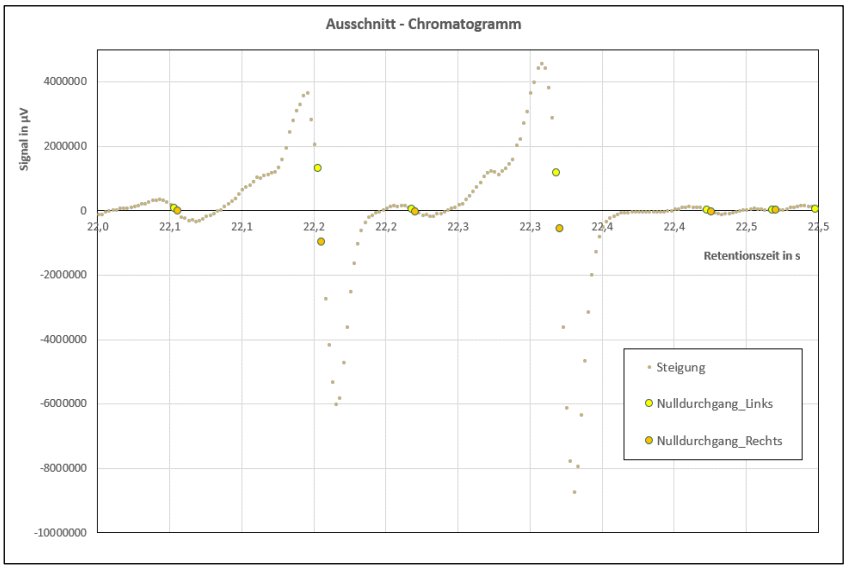
1. Steigung der Signalwerte über n Punkte errechnen.
Peakmaxima (Messreihen mit Spitzenwerten) können leicht bestimmt werden, indem die 1. Ableitung bzw. nur die Steigung einer Geraden über n Signalwerte berechnet wird. Die Steigung wechselt beim Peakmaxima das Vorzeichen - d.h. jeder Nulldurchgang ist ein Peak.



2. Innerhalb der Steigungen die Zeitintervalle der Nulldurchgänge bestimmen.
Für die Nullstellenfindung müssen zunächst die zeitlichen Grenzen dieser gefunden werden. Das gelingt mit der einfachen Bedingung:

If Steigung_i > 0 And Steigung_i+1 < 0 Then *Anweisungen*

Das Resultat ist im folgenden Bild zu sehen.

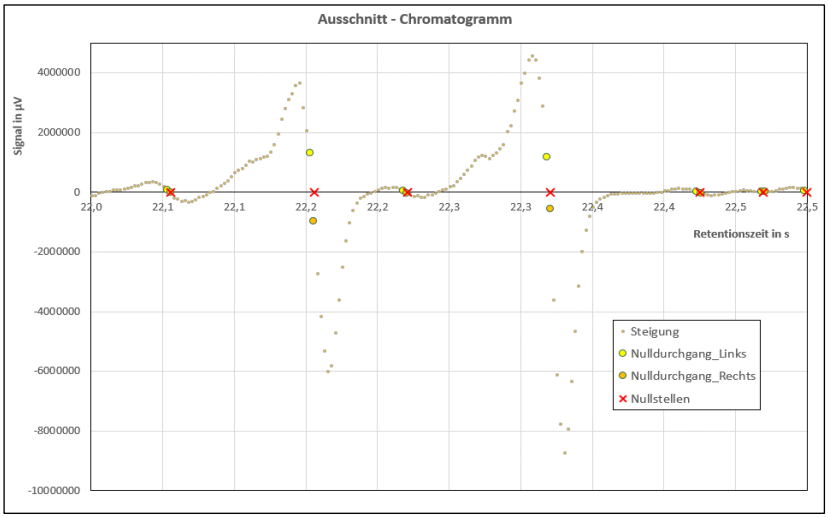


Tag 7

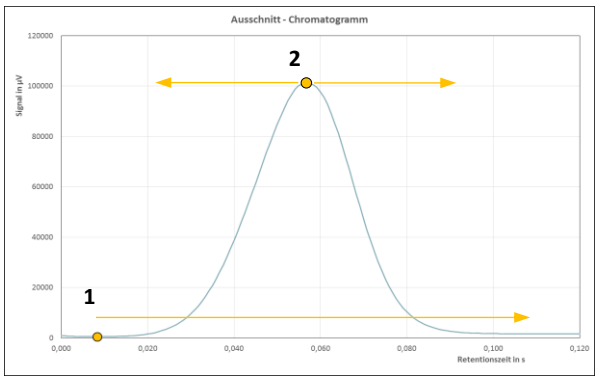
Komplexaufgabe 1

Notizen

3. Mit Hilfe der Zeitintervalle die Peakmaxima in den Rohdaten suchen.
Nachdem die zeitlichen Intervallgrenzen der Peaks ermittelt wurden, können diese genutzt werden, um die Peakmaxima innerhalb der Rohdaten zu finden.
Mit Hilfe der WorksheetFunctions "Max" und "Match" ist dies einfach zu realisieren.



Vorübung 1: Peakfläche integrieren.



Integrieren Sie den obigen Peak numerisch mir Hilfe der

$$Ober-/Untersumme \quad A_{Peak} = \Sigma\left((x_{i+1} - x_i) \cdot y_{i+1}\right)$$

und

$$Trapezregel \quad A_{Peak} = \frac{1}{2} \cdot \Sigma\left((x_{i+1} - x_i) \cdot (y_i + y_{i+1})\right)$$

innerhalb der Zeilen 18 und 145.

Vorübung 2:
Gehen Sie vor wie bei Vorübung 1., fangen Sie diesmal beim Peakmaximum an (Zeile 76). Integrieren Sie nacheinander die linke und die rechte Peakseite. Wählen Sie ein geeignetes Abbruchkriterium aus.

TAG 7

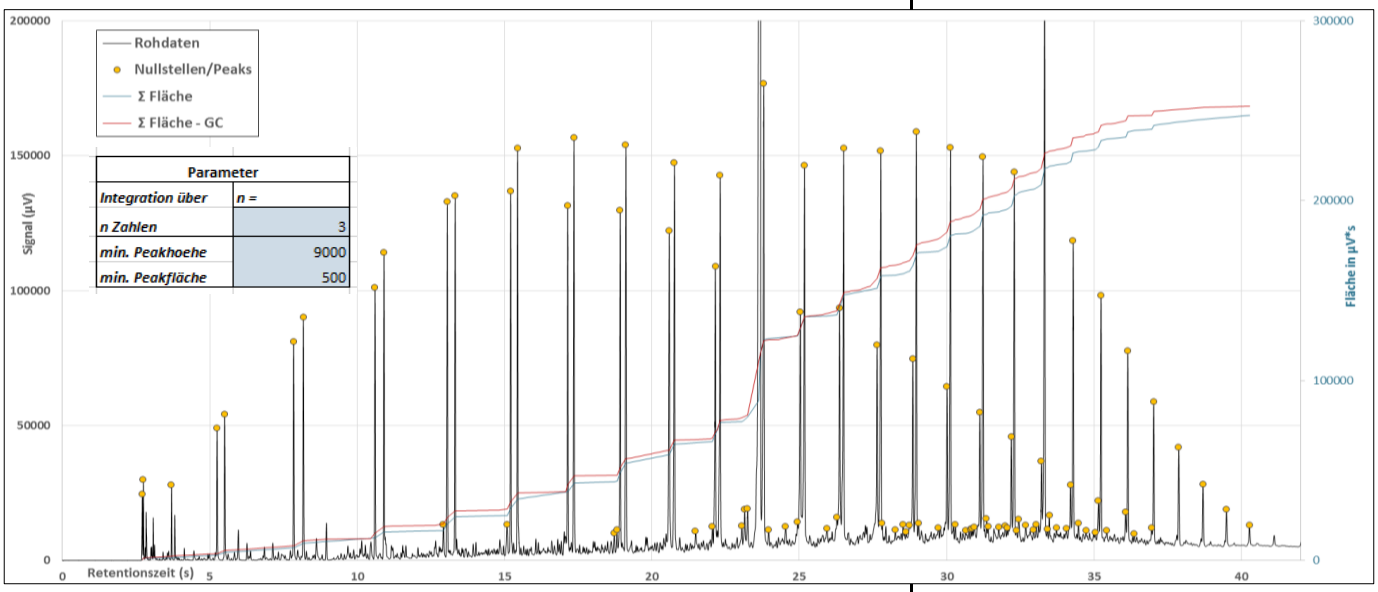
Komplexaufgabe 1

Notizen

4. Ausgehend der Peakmaxima, den Peak links - bzw. rechtsseitig integrieren (Peakfläche berechnen).
Analog zur Vorübung 4.2 können jetzt alle gefundenen Peaks integriert werden.

5. Peaks abhängig von Höhe und Fläche gefiltert anzeigen.
Für eine übersichtliche Auswertung des Chramatogramms ist es notwendig, sich nur die relevanten Peaks anzeigen zu lassen. Mit einer einfachen If-Else-Anweisung ist es sehr leicht möglich.
Der Vorteil gleich zu Beginn jeden gefundenen Peak zu berücksichtigen und auszuwerten ist der, dass dieses nur einmal gemacht werden muss. Das Peakfiltern wird ausschließlich auf das "Filtern" reduziert (ohne Neuberechnung der Peaks).

Das Resultat kann so aussehen.



Wer mehr machen möchte, folgende Punkte sind noch notwendig:

- Basislinienkorrektur
Einen Drift der Basislinie wurde nicht berücksichtigt, die Peakintegration bezieht sich auf y=0. Gut zu erkennen ist dies im Chromatogramm zum Ende hin. Dort werden auch kleinere Peaks erkannt, da die Peaks die entsprechende Peakhöhe haben, was aber auf eine steigende Basislinie zurückzuführen ist.

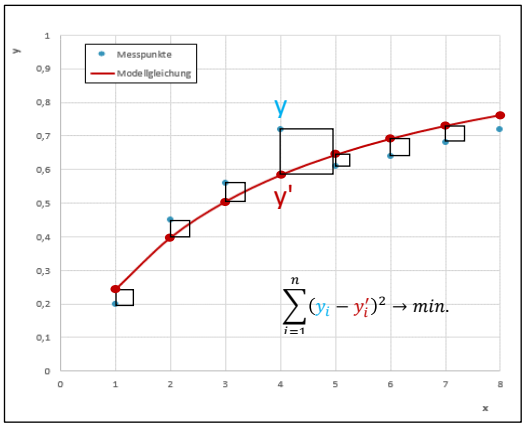
- Gauß-Fit
Peaks können auch durch eine Gauß-Verteilung ausgedrückt werden. Gerade für Peaks die nicht sauber getrennt wurden (Überlappung), ist dies eine gute Möglichkeit, die Peakfläche herauszufinden.

TAG 8

Komplexaufgabe 2 [1]

Die zweite Komplexaufgabe befasst sich mit der Erstellung von Regressionskurven. Dafür werden zur Berechnung das Excel-AddIn "Solver" und zur besseren Darstellung ein UserForm-Objekt verwendet.

Die Regressionskurven werden über die Methode der kleinsten Fehlerquadrate (FQS) bestimmt, wobei Solver die Suche nach den Konstanten übernimmt.



Solver

Vorteile der Nutzung über VBA

* Zellenbezüge können tabellenübergreifend angeben werden
* Zielwert kann ein Zellenbezug sein
* Ausführung in Schleifen

Verweis setzen

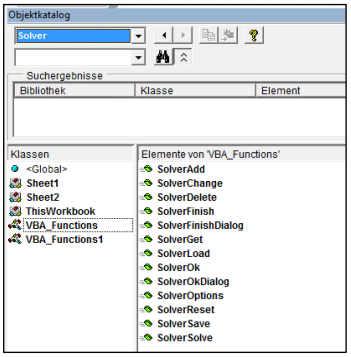
* Solver ist ein Add-In-Programm und muss für die Verwendung in VBA referenziert werden, um dessen Klassen nutzen zu können.
  VBA-Editor -> Verweise -> Häkchen bei Solver
* Anschließend sind die Klassen im Objektkatalog sichtbar. ->

Wichtigste Solver- Klassen

* Durch den Verweis erhält man Zugriff auf die Klassen vom Solver. Nachfolgend sind einige dieser aufgelistet.
* **SolverReset()**
  Setzt alle Werte auf Standard
* **SolverOk(SetCell, MaxMinVal, ValueOf, ByChange, [….])**
  Setzt Werte für Zielzelle, Zielwert und Veränderbare Zelle(n)
* **SolverOkDialog(SetCell, MaxMinVal, ValueOf, ByChange, [….])**
  Analog zu SolverOk + Dialogfenster
* **SolverAdd(CellRef, Relation, FormulaText)**
  Fügt Nebenbedingungen hinzu
* **SolverSolve(UserFinish, ShowRef)**
  Löst Berechnung aus.
  UserFinish:=True bewirkt, dass anschließend kein Bestätigungsdialog erscheint.
  Mit ShowRef lassen sich auftretende Fehler interpretieren.

Notizen

Achtung! Solver muss unter Optionen -> AddIns -> Excel AddIns -> GeheZu aktiviert sein



[1] https://www.solver.com/excel-solver-vba-functions

Tag 9

Komplexaufgabe 2

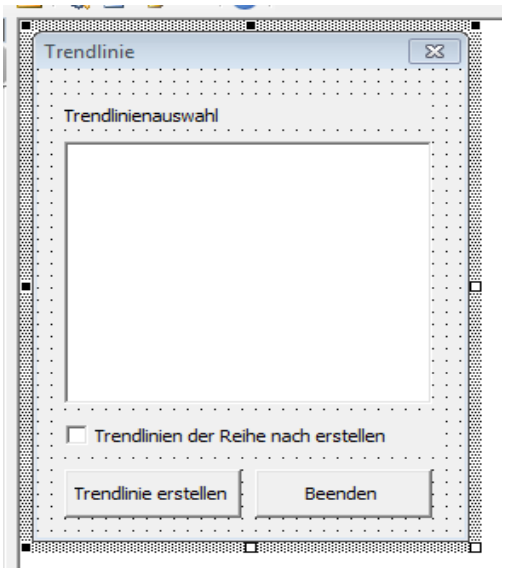
Notizen

UserForm

Für eine bessere Bedienbarkeit und Handhabe eignen sich UserForms.
Diese Formulare werden eingefügt über:
-> Einfügen -> UserForm

Bestückt werden UF mit Steuerelementen - diese werden in der Toolsammlung angezeigt und per Drag and Drop in die UF gezogen.

Erstellen Sie folgende UF mit folgenden Steuerelementen:
1x Label (Bezeichnungsfeld)
1x ListBox (Listenfeld)
1x Kontrollkästchen (CheckBox)
2x CommandButtons (Befehlsschaltfläche)



Tag 10

Arrays

Notizen

* Ein Array entspricht einem Datenfeld bestehend aus einzelnen Elementen eines Datentyps, die über Indizes angesprochen werden.
  Ein Array kann analog zu einem Tabellenblatt gesehen werden. So entspricht z.B. arrD(1048576) der Spalte D, die 1048577 Zellen (Elemente) besitzt.
  Array dienen als Aufbewahrungsort für sortierte Variablen, da die Ablage in Tabellenblättern sehr rechenintensiv sein kann.
  Die Arbeit mit Arrays verschlanken zum einen den Code und machen ihn wesentlich effektiver.

Elemente von Arrays fangen bei 0 (Null) an!

Deklaration/Initialisierung

Statische Arrays

```
Dim arrSt1(10) As Datentyp
 'Array mit 11 Elementen (0, 1, 2, ...10)

Dim arrSt2(5 To 10) As Datentyp
 'Array von Indizes 5 bis 10 = 6 Element

arrSt1(0) = 1.1
arrSt1(1) = 1.2
arrSt1(2) = 1.3

arrSt2(1) = 1.1          'Fehler!!!

Dim arrSt1(10, 10) As Datentyp
'2 dimensionales Array mit 11*11 Elementen

arrSt1(0, 0) = 1.1
arrSt1(0, 1) = 1.2

Dim arrSt1(10, 10, 10) As Datentyp
'3 dimensionales Array mit 11*11*11 Elementen
```

Beim Überprüfen der Arrays mit Haltepunkten arbeiten

Vorsicht bei höheren Arrays, ... Rechenaufwand steigt stark an mit den Dimensionen!

Dynamische Arrays

```
Dim arrDy1() As Datentyp
 'Array mit variabler Größe
 'Kann so noch nicht benutzt werden

ReDim arrDy1(10)
 'Neudimensionierung auf 11 Elemente
 'Vorsicht! bei ReDim werden Elemente gelöscht
ReDim Preserve arrDy1(20)
 'durch Preserve bleiben die Elemente erhalten
```

* LBound(arrayname, [dimension]) [1]
  Funktion, die die unterste Arraygröße einer Dimension ausgibt
* UBound(arrayname, [dimension])
  Funktion, die die oberste Arraygröße einer Dimension ausgibt

[1] https://docs.microsoft.com/de-de/office/vba/language/reference/user-interface-help/ubound-function