Institute for Management Information Systems

Vienna University of
Economics and Business Administration
Augasse 2-6
A-1090 Vienna

Assessing Executing MS Excel Macros and Functions with OOo Calc

Seminar Paper

PI 1076
MIS Seminar

Vienna, January 2008

Writer
Franz Müller, 0352250

Adviser
ao. Univ.Prof. Dr. Rony G. Flatscher

Table of Contents

1 Introduction

This paper tries to assess the executing of Microsoft Excel (Excel) macros and functions in OpenOffice.org Calc (Calc). The interoperability between Excel and Calc could be a very interesting issue for businesses as well as for individuals, because OpenOffice.org (OOo) can be used entirely free of any license fees [WhyOO]. To assess the extent of interoperability, an Internet research was made and several workbooks were created in Excel which contained functions and macros. These Excel workbooks were afterwards opened with and saved in Calc and the examples in it were tested.

In chapter 2 basic information about functions and macros are given. Firstly a short overview of functions and macros is given and furthermore the differences in terminology are explained. The first sub-chapter is about how to use functions in Excel and Calc. The second sub-chapter concentrates on handling macros in Excel and Calc. For each program the used programming languages are described briefly and the processes of recording and executing macros are shown. The third sub-chapter addresses the interoperability between Excel and Calc. Both the efforts in the past to enable the interoperability and the actual support in OOo 3 are shown. Additionally, possible lock-in-effects of Calc are discussed.

In chapter 3 the test cases which were developed to assess the interoperability and the findings are presented. The first sup-chapter deals with 336 functions in Excel, which were divided into eleven uniform groups. For each group the findings are presented graphically and in a textual way. In addition, functions which were not executable correctly are described in more detail. The second sub-chapter concentrates on the possibility to execute Excel macros in Calc. To assess the interoperability ten uniform categories were build and altogether 137 small macros were developed and tested.

2 Basics about Functions and Macros

Before basic information about functions and macros is given, some differences in terminology should be explained (see Table 1). The entire file, which accommodates the data is named workbook in Excel and spreadsheet in Calc. The tabbed sheet in a workbook respectively a spreadsheet is named worksheet in Excel and sheet in Calc. In both programs each individual cell is named cell. A further difference between Excel and Calc is the naming of callouts in cells which appear when the mouse pointer is positioned over the cell. In Excel such a callout is named commend, whereas in Calc it is named Note. [MigG06]

	Calc	**Excel**
Entire file	Spreadsheet	Workbook
Tabbed sheet	Sheet	Worksheet
Individual cell	Cell	Cell
Callouts in cells	Note	Commend

Table 1: Differences in terminology

In Calc and Excel there are a lot of build-in functions, which help the user to deal with complicated tasks. These functions handle categories like statistic, finance and so on. It is also possible to program own functions in Calc and Excel.

Macros are programs which automate tasks and can be embedded in documents [MigG06]. This automation of tasks will help to work more efficiently. Tasks which have to be executed for different data in the same way can be executed e. g. only by one click. But not only the possible reduction in working time is important. Furthermore, a possible source of error could be eliminated by macros. People, who have to do a lot of tasks which are very similar, tend to decrease their concentration, which will probably result in an increase in errors.

2.1 Handling Functions in Calc and Excel

In Calc and Excel cells can contain values like text, numbers or dates as well as formulas and their results [UseG07]. All formulas begin with an equals sign and may contain numbers, text and other data like format details to specify how numbers are to be formatted. Usually formulas will also contain arithmetic or logic operators. When using

formulas in such ways, multiplications and divisions will be calculated before additions and subtractions. A further way to use formulas is as starts of functions. It is also possible to nest functions in formulas or in functions. [UseG07]

Calc has about 350 build-in functions which cover topics like databases, date and time, finance and so on. To use these functions there are several possible ways. In Calc the function wizard, which can be accessed via the button  or **Insert | Function...**, helps the user to insert a function. The function wizard groups the functions to uniform categories. For each selected function the wizard shows background information and information concerning the needed parameters to use the function in the correct way (see Figure 1).

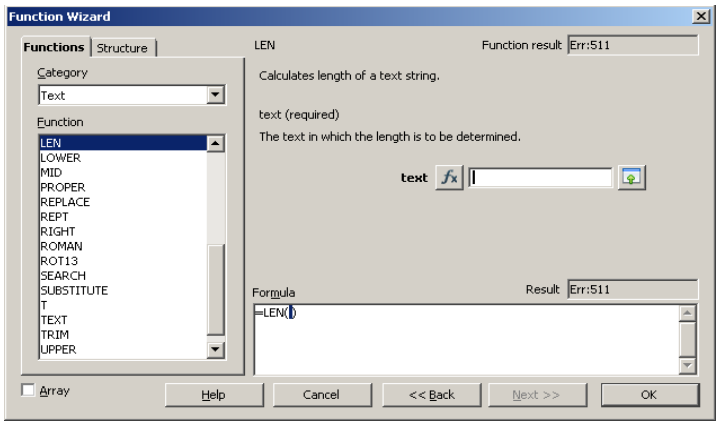


Figure 1: Function Wizard in Calc

In Excel there are about 340 build-in functions available, which cover the same topics as the functions in Calc. Excel also has a function wizard, which can be accessed via the button  or **Insert | Functions...**. The difference to the function wizard in Calc is that in Excel the function has to be chosen firstly (see Figure 2) and afterwards the main form of the function wizard is displayed. This form shows similar background information as the function wizard in Calc (see Figure 3).

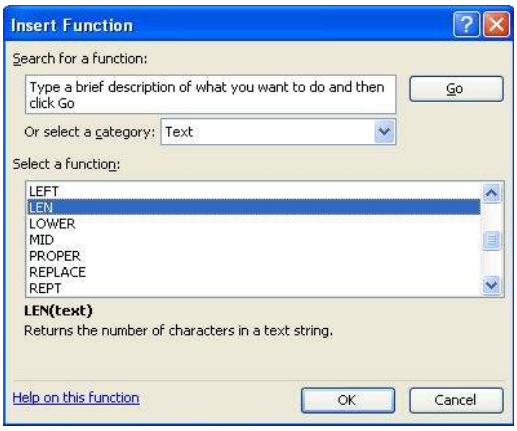


Figure 2: Choosing a function in Excel

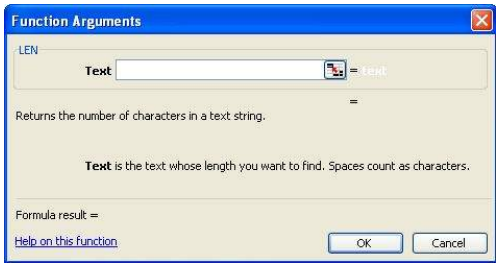


Figure 3: Function Wizard in Excel

In both programs, there are further ways to put functions into cells. One way is to type into a cell the equals sign followed by the name of the function and an opening bracket. The difference between the two programs is, that in Excel a short information is displayed concerning the function and the needed arguments, whereas in Calc no information is given.

It is also in both programs possible to fill a cell with a function by using a macro. How to record macros and use them to execute functions in cells will be described in the chapter 2.2 Handling Macros in Calc and Excel.

In Calc and Excel it is also possible to program own functions and use these functions afterwards in a cell. This could be handled with the programming languages OOo Basic in Calc and Visual Basic for Applications (VBA) in Excel. Such programmed macros can be saved as external programming modules (Add-ins) providing additional functions for working with spreadsheets respectively workbooks. The so programmed modules can be implemented and used from others. As the programming of external modules is not explained in this paper in more detail, please have a look at the help sections of each

program (Calc: *Add-in for Programming in OpenOffice.org Calc*; Excel: *About add-in programs*).

In OOo it is possible to write and integrate own UNO (Universal Network Objects) components. These components are called Add-Ons and can be added to the OOo menus and toolbars. For further information http://udk.openoffice.org/ and the help section *Integrating new UNO components* in Calc are recommended.

“An important difference between Excel and Calc regarding functions is that the values supplied to a function (which are called arguments in Excel and parameters in Calc) have different separators. Calc always uses semicolons to separate parameters in a function. Excel uses either commas or semicolons, depending on the system (for example commas on English systems, semicolons on German systems). Calc will generate a “#NAME?” error if you use a comma in place of a semi-colon.” [MigG06]

2.2 Handling Macros in Calc and Excel

As already mentioned above, macros are programs which automate tasks and can be embedded in a document [MigG06]. In other words, macros are saved sequences of commands or keystrokes that are stored for later use. Such commands allow a variety of functions, such as making decisions (e. g. if a value is less than zero, color it red), looping (e. g. if a value is greater 10, subtract 5 from it) or interacting with a person (e. g. asking the user for values to subtract). To start a macro quickly, it is possible to assign it to a keystroke or toolbar icon. [PitoA04]

In Excel, it is possible to use Visual Basic for Applications (VBA), which is an embeddable programming environment, to develop and modulate programs. Whereas Visual Basic is a stand-alone tool for creating e. g. separate software components, VBA is used to offer the same tools in the context of an existing application. VBA is a core component in the MS Office Suite and is integrated into Access, Excel, Outlook, PowerPoint and Word. [VBAFAQ]

The programming language in OOo is OpenOffice.org Basic (OOo Basic) which has been developed especially for the Office package and is firmly integrated in it [BASIPG].

OOo Basic as well as VBA belong to the family of Basic languages and therefore they are very similar to each other. The primary reason why VBA does not work in Calc without any developers' effort is that Calc uses a different method to access the spreadsheet (respectively workbook in Excel) and the cells. Specifically the objects, attributes and methods use different names which could cause a different behavior. [Porti04] Additionally to OOo Basic, OOo internally supports the scripting languages JavaScript, BeanShell and Python, although Python is not included with all distributions *(see* help section in Calc: *Assigning Scripts in OOo).*

2.2.1 Creation and Execution in Calc

To create a macro use **Tools | Macros | Record Macro** which will start the macro recorder. As long as the window *Record Macro* 

is displayed all commands and keystrokes will be recorded. In this example a macro is recorded, which inserts the actual date into the selected cell. To stop the recording mode, the button *Stop Recording* must be clicked which opens the *OpenOffice.org Basic Macros* dialog (see Figure 4).

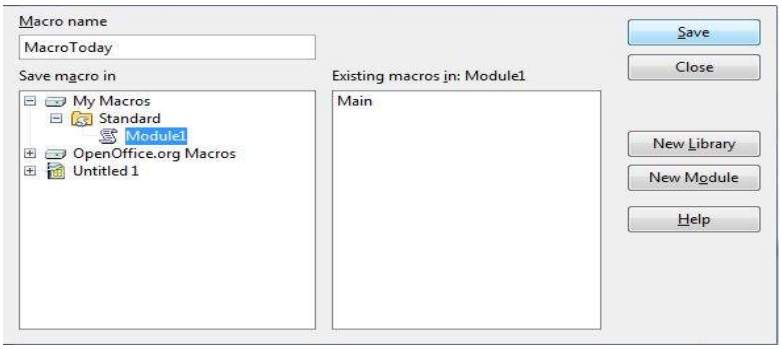


Figure 4: OpenOffice.org Basic macro dialog

The macro will be stored in the library named *Standard*, which is within the library container named *My Macros*. In this dialog it is also possible to create a new library and/or module to store the macro in. The following Figure 5 shows the organization of macros in OOo [GetSt08]:

- A library container contains one or more libraries, and each library is contained in one library container.
- A library contains zero or more modules, and each module is contained in one library.

- A module contains zero or more macros, and each macro is contained in one module.

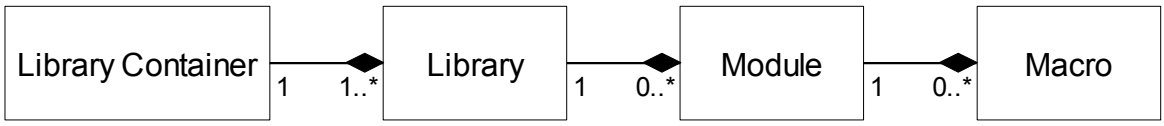


Figure 5: Macro Library hierarchy, [GetSt08]

After entering a name in the upper left corner of the *OpenOffice.org Basic Macros* dialog, the macro can be saved in the preferred module. In this example the standard library contains a module *Module1* with the macro *MacroToday* (see Figure 4).

To run the macro, **Tools | Macros | Run Macro** has to be chosen, which opens the *Macro Selector* dialog. In this dialog the newly created macro could be selected and executed with *Run*. A further possibility to run a macro is with **Tools | Macros | Organize Macros | OpenOffice.org Basic...**, which will open the *Macro Organizer* dialog. The second dialog is more powerful, because it is possible to edit, delete and organize macros. If in this dialog the button *Edit* is clicked, the macro will be opened in the Basic IDE (Integrated Development Environment). The entire code of the macro *MacroToday* is shown in the following Figure.

```
sub MacroToday
rem ----------------------------------------------------------------------
rem define variables
dim document   as object
dim dispatcher as object
rem ----------------------------------------------------------------------
rem get access to the document
document   = ThisComponent.CurrentController.Frame
dispatcher = createUnoService("com.sun.star.frame.DispatchHelper")

rem ----------------------------------------------------------------------
dim args1(0) as new com.sun.star.beans.PropertyValue
args1(0).Name = "StringName"
args1(0).Value = "=today()"

dispatcher.executeDispatch(document, ".uno:EnterString", "", 0, args1())
rem ----------------------------------------------------------------------
dispatcher.executeDispatch(document, ".uno:JumpToNextCell", "", 0, Array())

end sub
```

Figure 6: OOo Basic code of macro MacroToday

The code shows that each macro of a module is stored in subroutines which start with the keyword *sub* and end with the keywords *end sub*. [GetSt08]

The syntax and semantic of OOo Basic will not be described in more detail. As already mentioned, it is also possible in Calc to run macros which were written in Python, BeanShell or JavaScript. These types of macros will not be discussed as this paper deals with the interoperability of VBA Macros in Calc.

2.2.2 Creation and Execution in Excel

To record a macro in Excel, **Tools | Macro | Record New Macro...** has to be chosen. The difference to Calc is that the user has to decide before recording which name the macro should have and where it should be stored. The following Figure shows the *Record Macro* dialog where the user could write into the left upper corner the name of the macro.

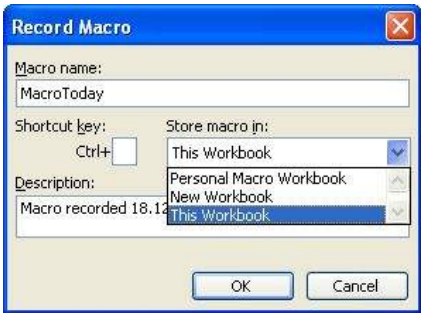


Figure 7: Record Macro dialog

The drop down box *Store macro in:* allows to store the Macro whether in a *Peronal Macro Workbook*, a *New Workbook* or *This Workbook*. It is also possible to define a *Shortcut key* to execute the recorded macro with only two keys. As long as the window *Stop Recording*  is displayed all commands and keystrokes will be recorded. With the button *Relative Reference* it is possible to record the commands relative to the actual reference. To run the macro **Tools | Macro | Macros...** must be chosen, which opens the dialog *Macro*. In this dialog it is possible to create, run, edit and delete a macro. If in this dialog the button *Edit* is clicked, the macro will be opened in the IDE as shown in Figure 8. Similar to OOo Basic in VBA the macro starts with the keyword *Sub* and ends with the keywords *End Sub*. Each macro is contained in one module and a module contains zero or more macros. [EinfMa]

```
Sub MacroToday()
'
' MacroToday Macro
' Macro recorded 18.12.2008 by Windows User
'


'
    ActiveCell.FormulaR1C1 = "=TODAY()"
    Range("A2").Select
End Sub
```

Figure 8: VBA code of macro MacroToday

2.3 Interoperability between Excel and Calc

The interoperability between Excel and Calc could be a very interesting issue for businesses, as OOo can be used entirely free of any license fees [WhyOO]. If someone thinks about a change from MS Office to OOo, not only the costing aspect should play a role. It is also important to know how well existing workbooks will work in an alternative office suite. Especially the functions and macros in existing documents should work in the same way after a change, because these documents could play a major role in core sectors of a company. The efforts which were made by employees to develop these documents could be very high. It is also possible that consultants were paid for developing macros, which should help employees to work more efficiently.

2.3.1 Efforts in the Past to enable the Interoperability

In early releases of OOo 2 there was no integrated solution to execute MS Office macros. Whereas the execution was not possible, the user could at least decide to keep the attached macros of MS Office files, so that they were still available for use in MS Office. [MigG06]

As OOo is distributed under the Lesser General Public License (LGPL), the complete source code of the program could be accessed and it is allowed to modify any part of the source or use fragments of it in other programs. Because of this licensing different vendors like Sun or Novell use source code developed by the OOo community and expand the functionality to provide own software suites. [GetSt08] As other vendors can expand

the functionality in different topics, it was already in early releases of OOo 2 possible to execute Excel macros in Calc in a limited way, provided that the alternative software suite was used.

Sun Microsystems released the source code for its StarOffice software to the open source community in October of the year 2000. Since this time, all versions of Sun's StarOffice have been using source code of OOo. Due to copyrights of third parties, the products do not provide exactly the same features. [GetSt08] For a better support of VBA, Sun provided an optional extension in StarOffice which could be enabled and disabled explicitly. This extension was implemented in Java and supported both, Excel and Word. [Sola07]

Also Novell has been working on an expanded software suite which uses the source code of OOo. One of the main goals of the office suite, which is named Go-oo, is the interoperability with documents from Microsoft. The api was implemented in C++ and supported only Excel macros. Whereas the VBA support in StarOffice was optional, in Go-oo it was enabled by default. [Sola07]

2.3.2 Interoperability in OOo 3

In February 2007, Sun released their proprietary VBA macro migration plug-in as Free Software. As the two solutions from Sun and Novell had the same goal, they overlapped in many areas. Because of this sub-optimal situation, both companies came to the agreement to share their resources and work together on a common project. As Novell used C++ and Sun used Java as implementation language, the developers had to evaluate the best way to bring both solutions together. They decided to go forward with C++ because OOo itself is implemented in C++. [SchJ07]

Whereas the outcomes of these efforts could have already been seen in later releases of OOo 2, OOo 3 has the best interoperability of Excel macros. In OOo 3 it is possible to specify the general properties for loading and saving Microsoft Office documents with VBA code with **Tools | Options | Load/Save | VBA Properties** (see Figure 9).

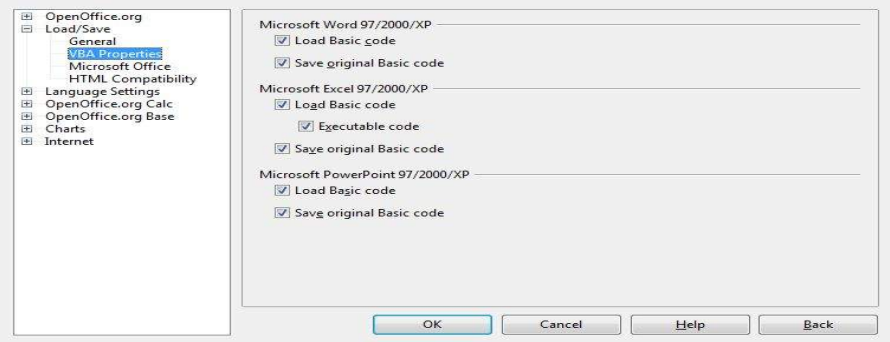


Figure 9: Options window in Calc

The Figure above shows that it is already possible to specify properties for Word, Excel and PowerPoint. The enabled option *Load Basic code* for Excel makes it possible to load and save the VBA code from an Excel document as a special OOo Basic module with the document. The disabled VBA code is visible in the OOo Basic IDE between *sub* and *end sub* and can furthermore be edited. If the document is saved in another format, the Basic code from the OOo Basic IDE is not saved. If the option *Executable code* is enabled, the VBA code will be loaded ready to be executable. Otherwise the VBA code will be commented out so it could be inspected but it will not be executable. The option *Save original Basic code* specifies that the original VBA code is held in a special internal memory for as long the document remains loaded in OOo. If the user decides to save the document in Microsoft format, the VBA code is saved again in unchanged form. When the document is not saved in Microsoft format, the VBA code is not saved with the document. It should be mentioned, that the option *Save original Basic code* takes precedence over the option *Load Basic code* (for further information see help section in Calc – *VBA Properties).*

To make the loading and executing of existing macros more secure, it is possible to adjust security levels in Calc. With **Tools | Options | OpenOffice.org | Security – Macro security...** it is possible to adjust four security levels and to specify trusted macro authors. Whereas the lowest level will allow to execute all macros without confirmation the highest level of security allows only macros from trusted file locations to run (for further information see help section in Calc – *Security*).

2.3.3 Possible Lock-in of Calc

“Vendor lock-in, or just lock-in, is the situation in which customers are dependent on a single manufacturer or supplier for some product (i. e., a good or service), or products, and cannot move to another vendor without substantial costs and/or inconvenience.” [Linfo]

In reference to the use of Calc and Excel three potential lock-in situations could occur when using Calc. The first one relates to the characteristics of the included file format respectively the possibility to save a document in different file formats. A second possible lock-in situation could occur, when users work with build-in functions which are only available in Calc but not in Excel. A further lock-in situation could occur when working with macros in Calc. The question is whether the embedded macros will work in Excel too or not.

OOo uses OpenDocument, which is an XML file format developed as an industry standard. Because of this file format, the files can easily be unzipped and read by any text editor. [MigG06] Furthermore, Calc is able to save the documents in Excel formats from versions 5.0 to XP, which shows that in such cases there is no vendor-lock-in.

When a Calc file is saved in Excel format, the build-in functions which are also available in Excel are executable afterwards. Problems occur, when in Calc build-in functions are used, which are not available in Excel. These functions will not work in Excel and could therefore cause a vendor-lock-in.

The third possible lock-in-effect refers to the handling of macros. When a macro is developed with OOo Basic, it is not possible to convert it to VBA so that it is executable with Excel afterwards. Whereas this causes a vendor-lock-in the situation is different when working with existing VBA macros. As already mentioned in the previous sub-chapter, it is possible to store the embedded VBA macros, to make a later execution in Excel possible.

3 Test Cases and Findings

The goal of this seminar paper is to assess the executing of Excel functions and macros in Calc. For this purpose several workbooks were developed in Excel which contain functions and macros. These workbooks were afterwards opened in Calc to test how well the functions and macros work.

The examples were developed with a German version of MS Excel 2003 and were tested with a German version of OOo Calc 3.

3.1 Excel Functions in Calc

In MS Excel 2003 there are 339 build-in functions available which cover the following categories:

- Database
- Date and Time
- Finance
- Information
- Logic
- Mathematics
- Search and Range
- Statistics
- Engineering
- Text and Data
- Trigonometry

Each of these categories and possible sub-categories are described briefly in the following sub-chapters. Explanations for categories and functions are mainly from the online-help of Microsoft [ListWF] and [JesE06] and will not be referenced individually. From the 339 functions, 336 were tested and are described in the following. The build-in functions *EUROCONVERT* and *SQL.REQUEST* were not tested, as these functions are counted among the external functions [ListWF]. The function PHONETIC which belongs to the category Text and Data was not executable correctly in Excel, so that this function was not tested in Calc.

3.1.1 Database Functions

As relational databases are mainly managed in tables, programs like Excel and Calc can be used to work with such databases. Excel has thirteen functions to analyze data which are organized in tables with headings and columns. Figure 10 shows the table which was the basis of the tested functions.

	A	B	C	D
1	Unternehmen	Datum	Artikel	Umsatz
2	Unternehmen W	24.11.2008	A	1200
3	Unternehmen W	25.11.2008	B	1000
4	Unternehmen X	25.11.2008	C	1600
5	Unternehmen W	26.11.2008	A	1440
6	Unternehmen Y	27.11.2008	D	980
7	Unternehmen W	28.11.2008	B	600
8	Unternehmen Y	01.12.2008	D	1100

Figure 10: Database in Excel

If the table is organized like in Figure 10, Excel identifies it as database. Important styling rules are, that field names are unique, that the heading is in the first row and that there are no empty rows within the database.

Figure 11 shows the findings of the database functions in an aggregated way. It shows that although the functions were converted correctly, some values were incorrect.

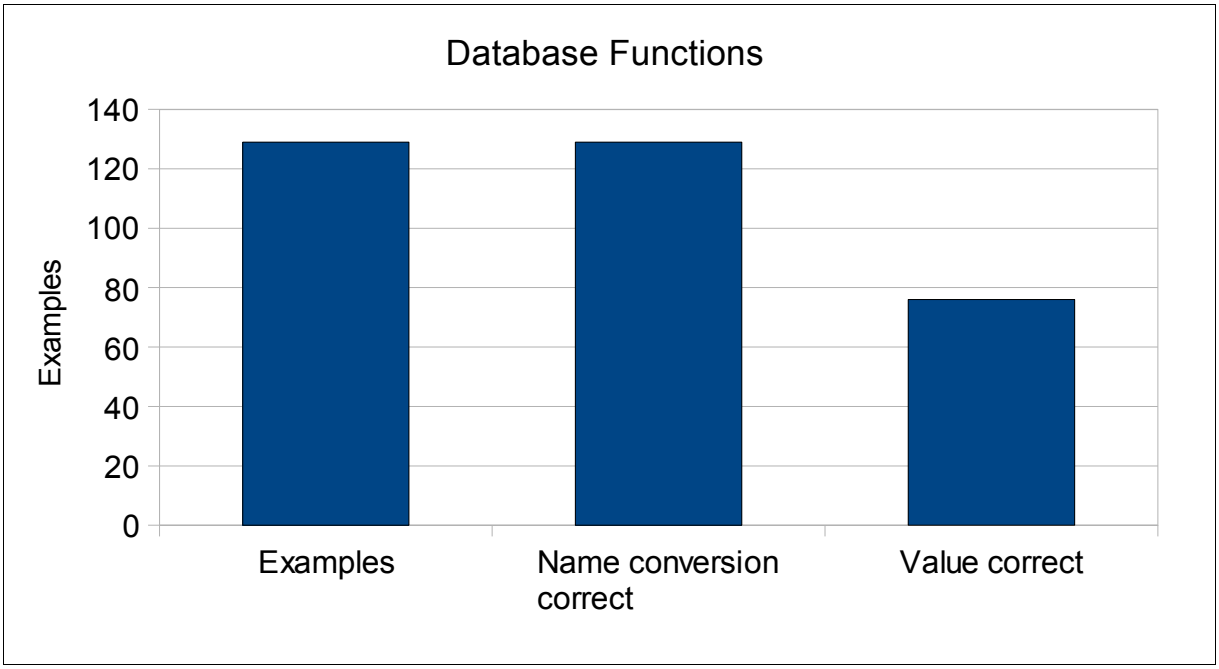


Figure 11: Database functions (aggregated)

Figure 12 shows the functions where the mistakes occurred, but the findings must be explained more accurately. Each function, excepting PIVOTDATENZUORDNEN, calculates a value for a field in the database. To calculate these values only for a fraction of the data, it is possible to define a criterion.

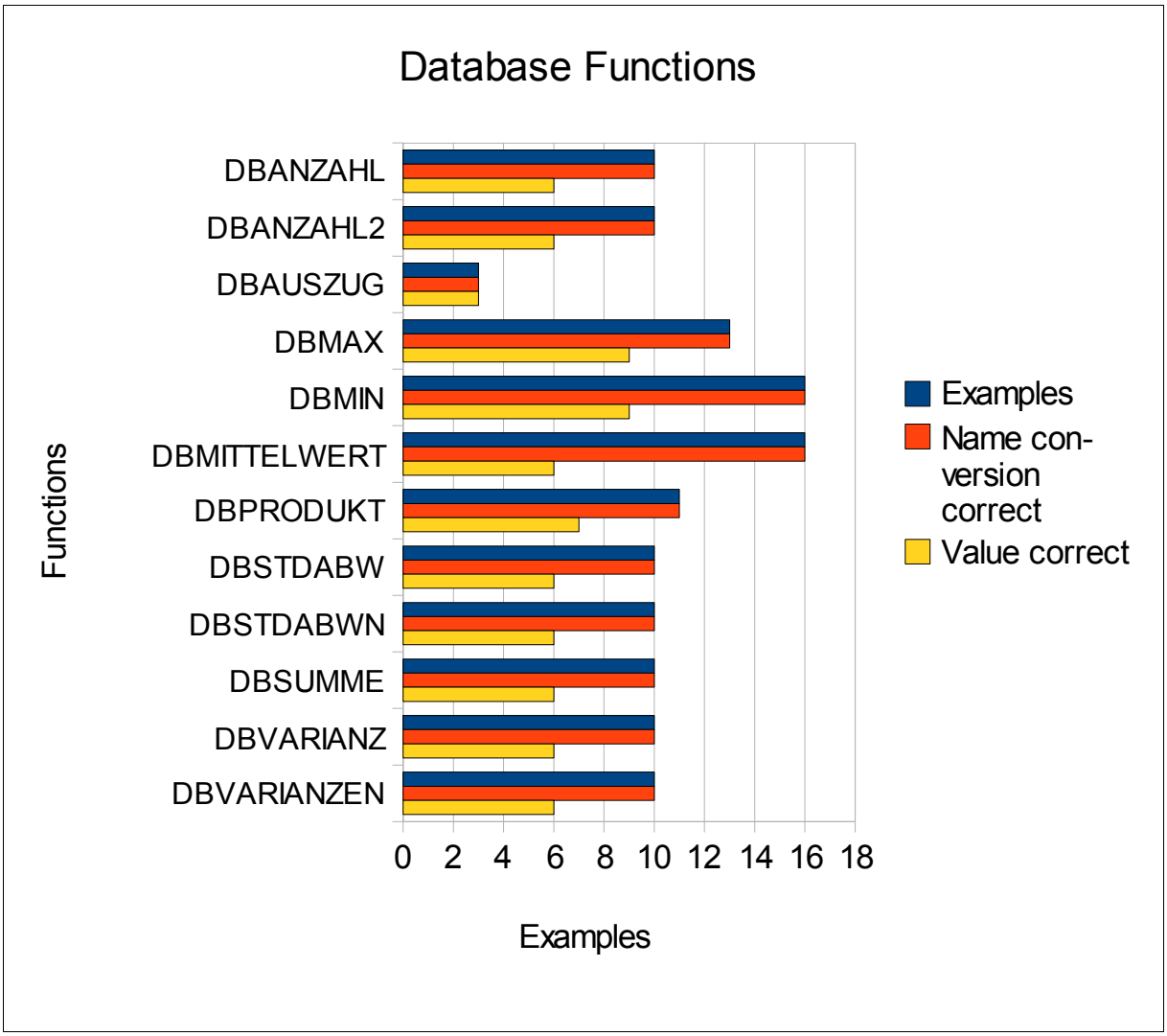


Figure 12: Database functions (separated)

Figure 13 shows the function DBANZAHL with a criterion to count only a part of the data.

	A	B	C	D	E
1	**DBANZAHL(Datenbank;***Datenbankfeld;* **Suchkriterien)**				
2					
3	**Datenbank**	*Datenbankfeld*	**Suchkriterien**		DBANZAHL
4	Datenbank	Umsatz	Umsatz		4
5			>1000		

Figure 13: Database function DBANZAHL

In Excel it is possible to use different operators to set criteria. The following table shows the possible operators in Excel and the applicability for each one in Calc.

Operator in Excel	Applicability in Calc
>	correct
<	correct
>=	correct
<=	correct
<>	correct
=	correct
*	incorrect
=*	incorrect
?	incorrect
=?	incorrect

Table 2: Operators in Excel and their applicability in Calc

The database function PIVOTDATENZUORDNEN returns data, which are stored in a PivotTable. This function is also executable in Calc.

3.1.2 Date and Time Functions

With functions from this category it is possible to make calculations with date values or to discover a certain status of a time value. Furthermore, it is possible to query the system time of the computer and work with it. It should be mentioned that Excel stores the date values as a serial number. This number starts with 1 (1900-01-01) and ends with 2958465 (9999.12.31). Digits which are on the right side of the decimal point represent the time.

In Figure 14 the findings of the aggregated functions of the category Date and Time are shown.

Figure 15 shows the findings for each individual function. The functions JAHR, MONAT and TAG convert a serial number to a year, a month or a day. Several problems occurred when working with these three functions. Figure 16 shows the original data which were used for the three functions and the return values for the function JAHR in Excel. Figure 17 shows the corresponding data in Calc.

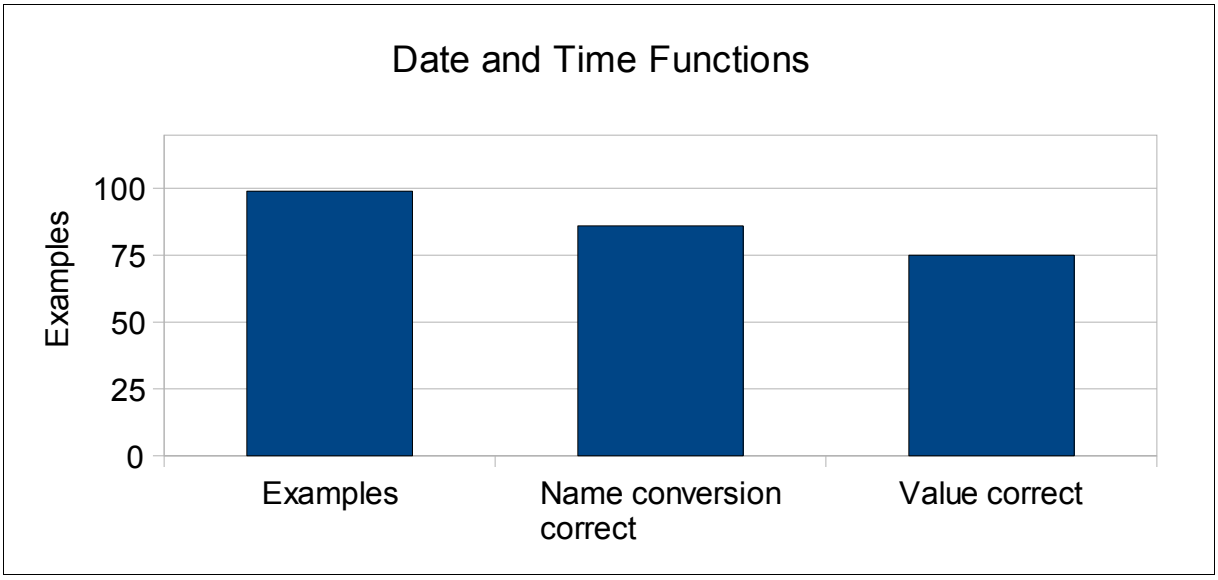


Figure 14: Date and Time Functions (aggregated)

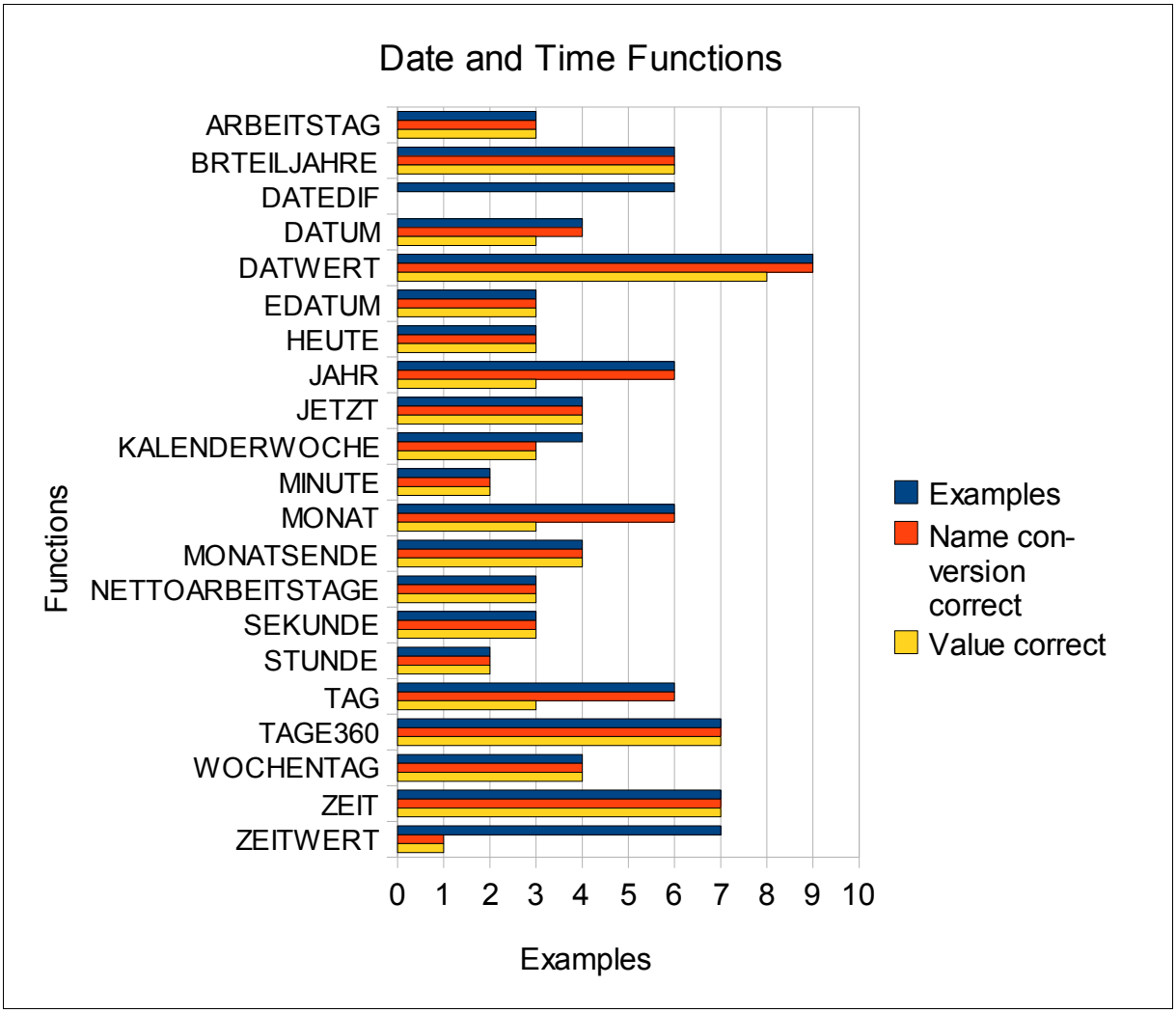
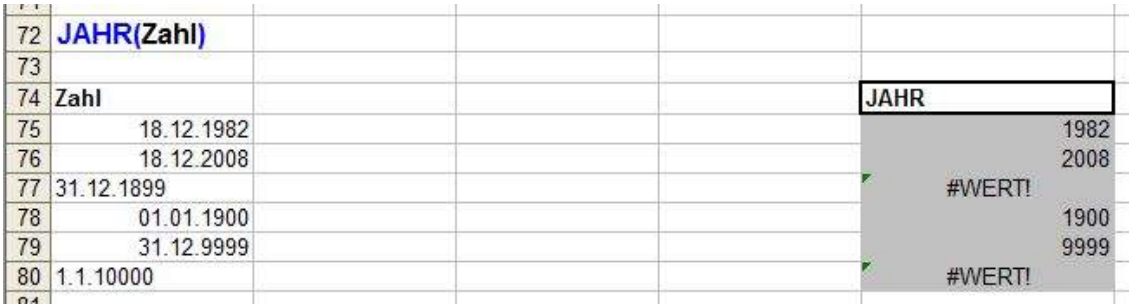


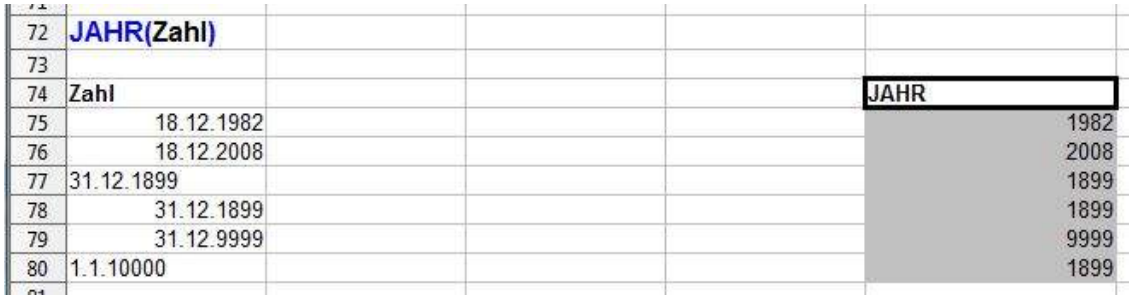
Figure 15: Date and Time Functions (separated)

As from Figures 16 and 17 could be seen, the first problem occurred when the Excel file was opened with Calc, because the value *01.01.1900* changed to the value *31.12.1899*. The next “problem” is that Calc is able to interpret more date values than Excel. Whereas Excel is only able to show year values between 1900 and 9999, Calc is able to extract the year value from dates before 1900. Both programs are not able to work with date values greater than 9999.



72	JAHR(Zahl)		
73			
74	Zahl		JAHR
75	18.12.1982		1982
76	18.12.2008		2008
77	31.12.1899		#WERT!
78	01.01.1900		1900
79	31.12.9999		9999
80	1.1.10000		#WERT!

Figure 16: Date values for the function JAHR in Excel



72	JAHR(Zahl)		
73			
74	Zahl		JAHR
75	18.12.1982		1982
76	18.12.2008		2008
77	31.12.1899		1899
78	31.12.1899		1899
79	31.12.9999		9999
80	1.1.10000		1899

Figure 17: Date values for the function JAHR in Calc

With the function DATEDIF it is possible to calculate the difference between two dates in years, months or days. This function is not executable in Calc.

The function DATUM has the arguments year, month and day and returns a serial number for these three arguments. Excel interprets the year value 108 as year 2008 whereas Calc returns an exception.

The function DATWERT converts a date in the form of text to a serial number. Whereas Excel is not able to convert dates before 1900, Calc is able to do this.

The function KALENDERWOCHE converts a serial number to a number representing where the week falls numerically with a year. In Excel the second argument (return_type) is optional and defines on which day the week begins. With the return type 1 (default) the week begins on Sunday whereas with the return type 2 on Monday. Calc is not able to provide an solution, when the return type is missing.

The function ZEITWERT converts a time in the form of text to a serial number. From the Figures 18 and 19 it could be seen that Calc is not able to interpret the most values.

197	ZEITWERT(Zeit)				
198					
199	Zeit				ZEITWERT
200	06:00:00				0,25
201	06:00 AM				0,25
202	06:00 PM				0,75
203	06:00:05 PM				0,75005787
204	01.01.2008 06:00				0,25
205	23:59:59				0,999988426
206	24:00				0

Figure 18: Time values for the function ZEITWERT in Excel

197	ZEITWERT(Zeit)				
198					
199	Zeit				ZEITWERT
200	06:00:00				0,25
201	06:00 AM				Err:502
202	06:00 PM				Err:502
203	06:00:05 PM				Err:502
204	01.01.2008 06:00				39448,25
205	23:59:59				1
206	24:00				1

Figure 19: Time values for the function ZEITWERT in Calc

3.1.3 Financial Functions

The functions for finance could be divided into different sub-categories, which are shown in Table 3.

Sub-categories	Counted functions
General Functions	4
Simple Interest Calculation	15
Compound Interest Calculation	3
Capital Budgeting	5
Amortization Calculation	7
Computation of Annuities	1
Sinking Fund Calculation	4
Bond Valuation	12

Table 3: Sub-categories of Financial Functions

The four functions from the category *General Functions* could be used in some of the other sub-categories. One of these functions is BW, which returns the present value of an investment. The function ZINS returns the interest rate per period of an annuity. With the function ZW, the future value of an investment could be returned. The last function of this group is ZZR, which returns the number of periods for an investment. Calc was able to execute all four functions (see Figure 20).

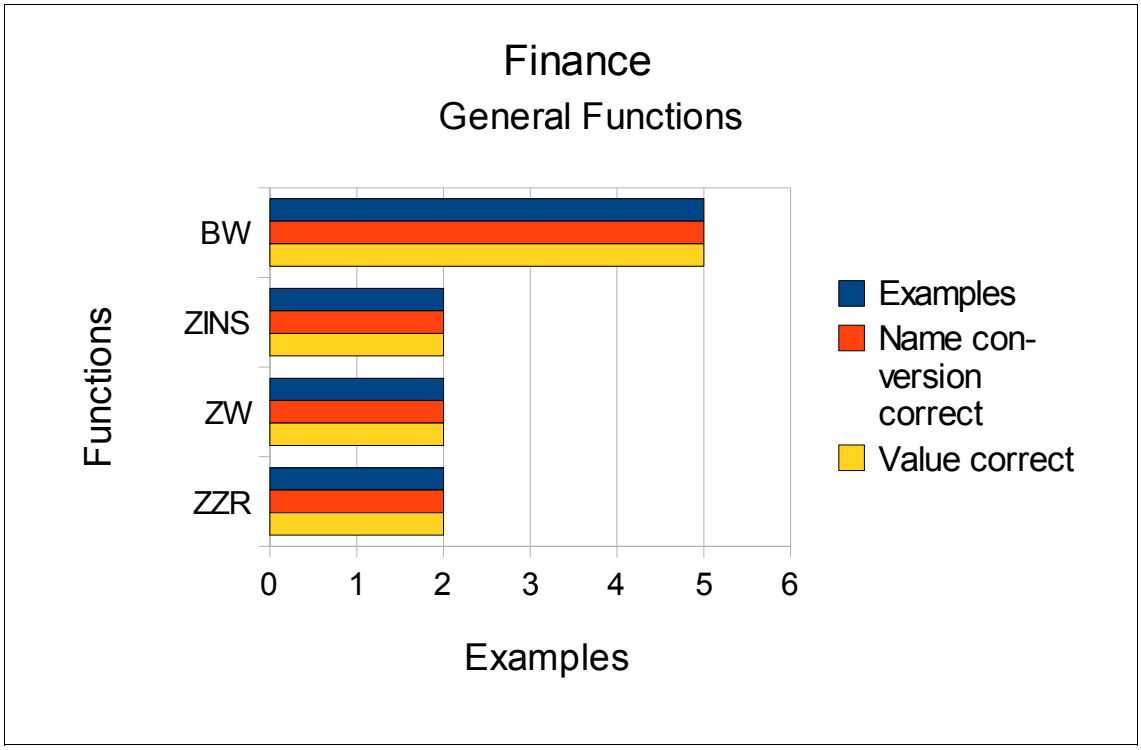


Figure 20: Finance – General Functions

When using functions for the simple interest calculation, interest is not added to the principal. From these functions the function KURSFÄLLIG is not executable in Calc. KURSFÄLLIG returns the price per 100 face value of a security that pays interest at maturity. The function RENDITEFÄLL returns the annual yield of a security that pays interest at maturity, which is also not executable in Calc (see Figure 21).

The function TBILLÄQUIV is not executable, which is used to return the bond-equivalent yield for a Treasury bill. The function TBILLKURS returns the price per 100 face value for a Treasury bill. Whereas this function is executable in Calc, the calculated values are different from those in Excel. In the Figures 22 and 23 the values for each program are shown.

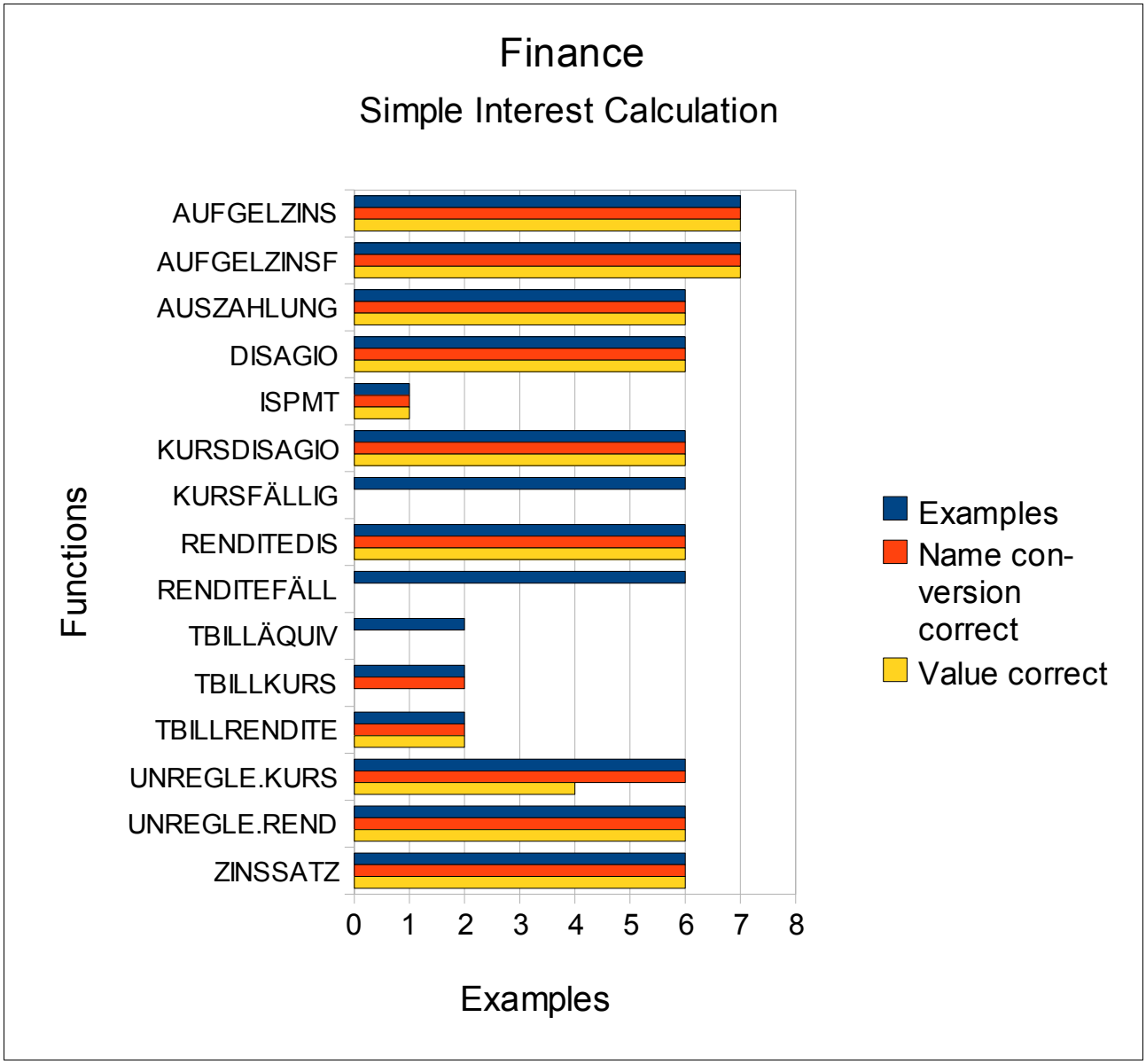


Figure 21: Finance – Functions for Simple Interest Calculation

104	TBILLKURS(Abrechnung;Fälligkeit;Disagio)				
105					
106	Abrechnung	Fälligkeit	Disagio		TBILLKURS
107	13.04.2008	01.09.2008	0,05		98,04166667
108	13.04.2008	01.09.2008	0,05		98,04166667

Figure 22: Function TBILLKURS in Excel

104	TBILLKURS(Abrechnung;Fälligkeit;Disagio)				
105					
106	Abrechnung	Fälligkeit	Disagio		TBILLKURS
107	13.04.2008	01.09.2008	0,05		98,06944444
108	13.04.2008	01.09.2008	0,05		98,06944444

Figure 23: Function TBILLKURS in Calc

With the three functions of the category *Compound Interest Calculation*, interest is added to the principal. All three functions were executable in Calc (see Figure 24).

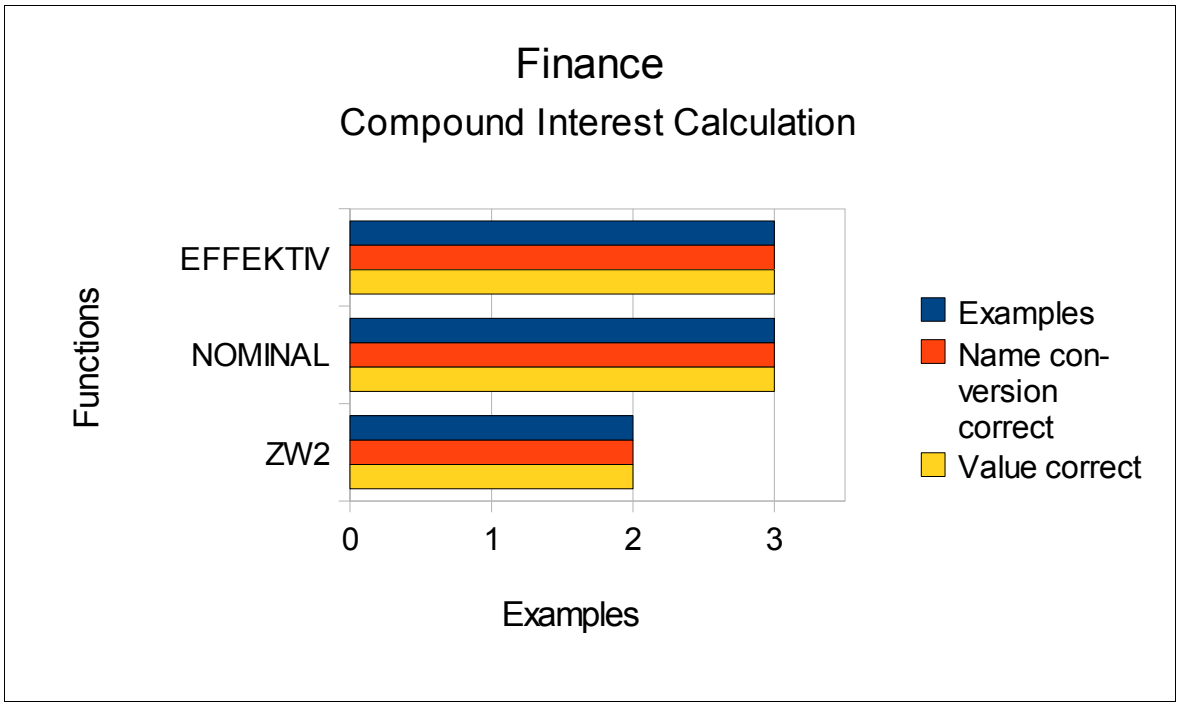


Figure 24: Finance – Functions for Compound Interest Calculation

In Figure 25 the functions for capital budgeting are displayed, which can further be divided into static and dynamic. The function XINTZINSFUSS, which returns the internal rate of return for a schedule of cash flows that is not necessarily periodic, was executable in Calc but the values were different to those in Excel.

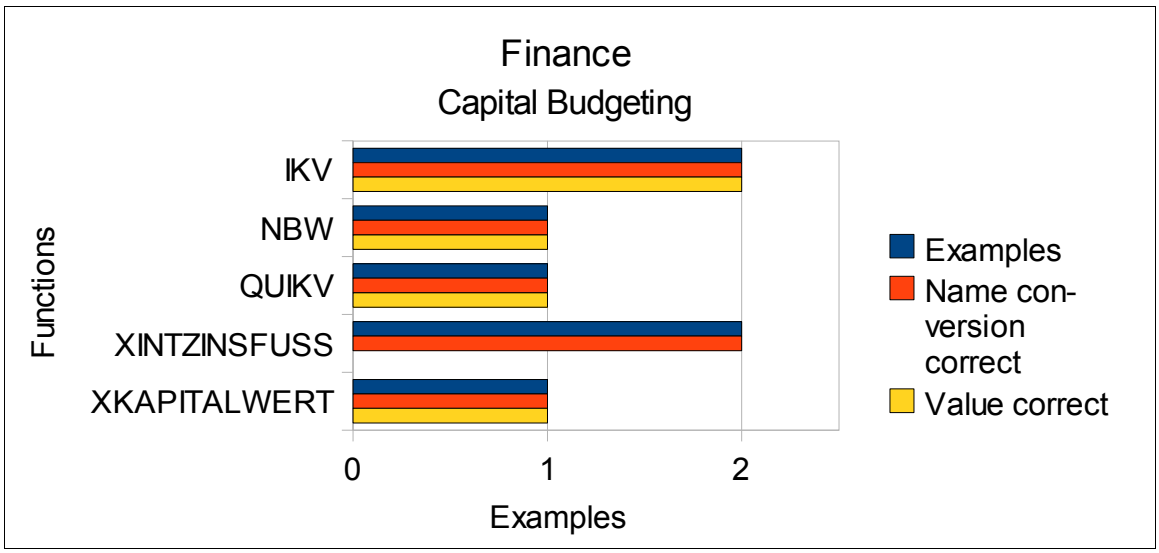


Figure 25: Finance – Functions for Capital Budgeting

Figures 26 and 27 show the different values of the function XINTZINSFUSS in Excel and Calc.

48	XINTZINSFUSS(Werte;Zeitpunkte;*Schätzwert*)			
49				
50	**Werte**	**Zeitpunkte**	*Schätzwert*	**XINTZINSFUSS**
51	-1000	01.01.2008		0,056031534076
52	300	01.02.2008		
53	220	01.04.2008		
54	160	01.06.2008		
55	180	01.08.2008		
56	160	01.10.2008		
57				
58	*Alternative*			
59	**Werte**	**Zeitpunkte**	*Schätzwert*	**XINTZINSFUSS**
60	-1000	01.01.2008	0,1	0,056031534076
61	300	01.02.2008		
62	220	01.04.2008		
63	160	01.06.2008		
64	180	01.08.2008		
65	160	01.10.2008		

Figure 26: Function XINTZINSFUSS in Excel

48	XINTZINSFUSS(Werte;Zeitpunkte;*Schätzwert*)			
49				
50	**Werte**	**Zeitpunkte**	*Schätzwert*	**XINTZINSFUSS**
51	-1000	01.01.2008		0,056031535669
52	300	01.02.2008		
53	220	01.04.2008		
54	160	01.06.2008		
55	180	01.08.2008		
56	160	01.10.2008		
57				
58	*Alternative*			
59	**Werte**	**Zeitpunkte**	*Schätzwert*	**XINTZINSFUSS**
60	-1000	01.01.2008	0,1	0,056031535669
61	300	01.02.2008		
62	220	01.04.2008		
63	160	01.06.2008		
64	180	01.08.2008		
65	160	01.10.2008		

Figure 27: Function XINTZINSFUSS in Calc

Figure 28 shows the functions for the amortization calculation, which were all executable in Calc.

For the sub-category C*omputation of Annuities,* there is only one function. The function RMZ returns the periodic payment for an annuity. All four examples of this function were executable in Calc.

In Figure 29 the four functions for the category S*inking Fund Calculation* are shown. All functions were executable in Calc and delivered correct values.

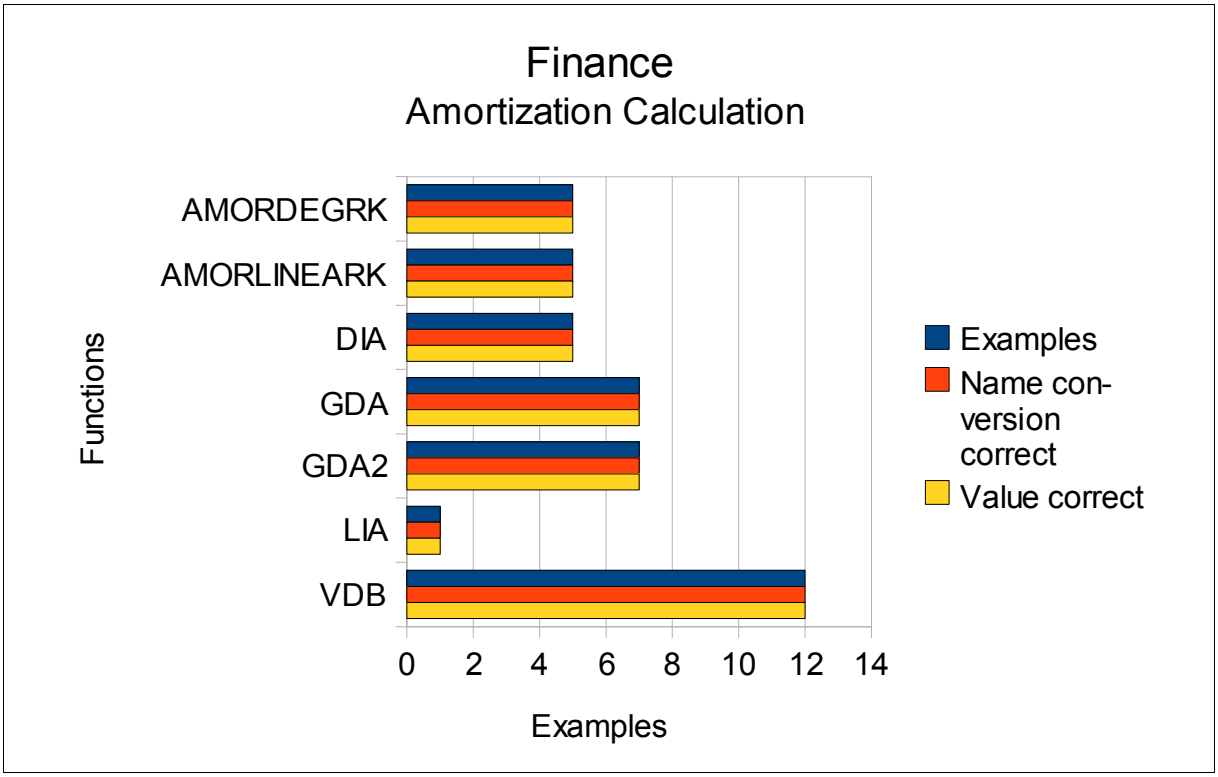


Figure 28: Finance – Functions for Amortization Calculation

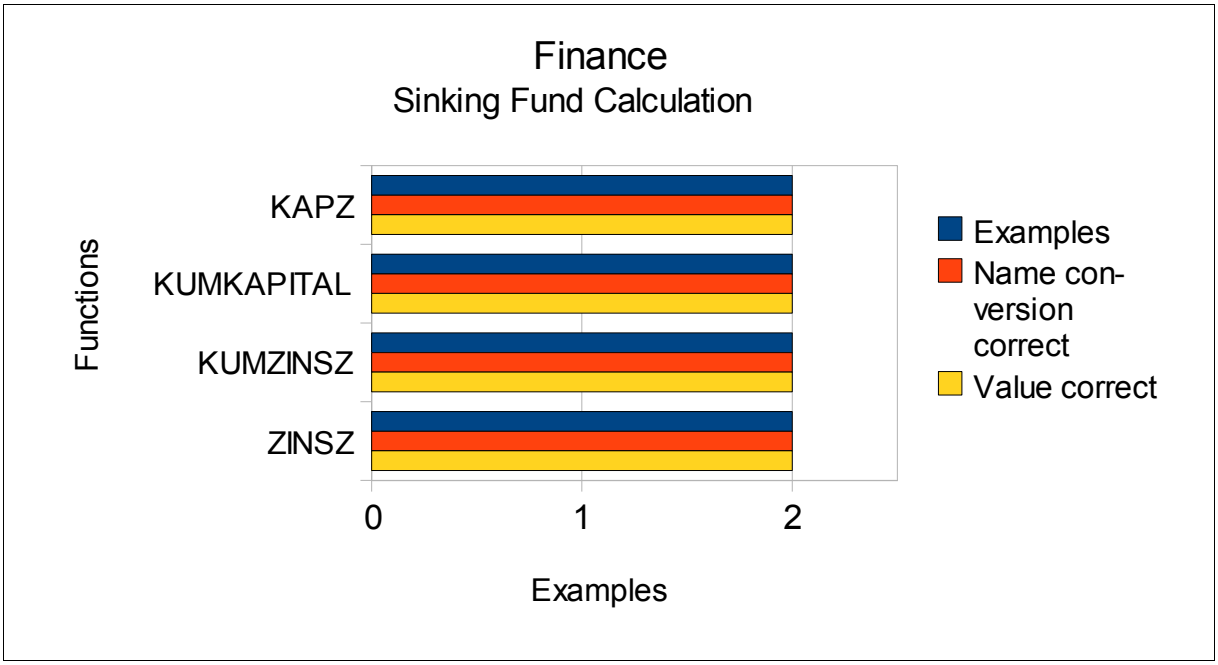


Figure 29: Finance – Functions for Sinking Fund Calculation

Figure 30 shows the last category of the functions for the category *Finance*. The Figure shows, that the two functions UNREGER.KURS and UNREGER.REND were not executable in Calc. UNREGER.KURS returns the price per 100 face value of a security

with an odd first period. The function UNREGER.REND returns the yield of a security with an odd first period.

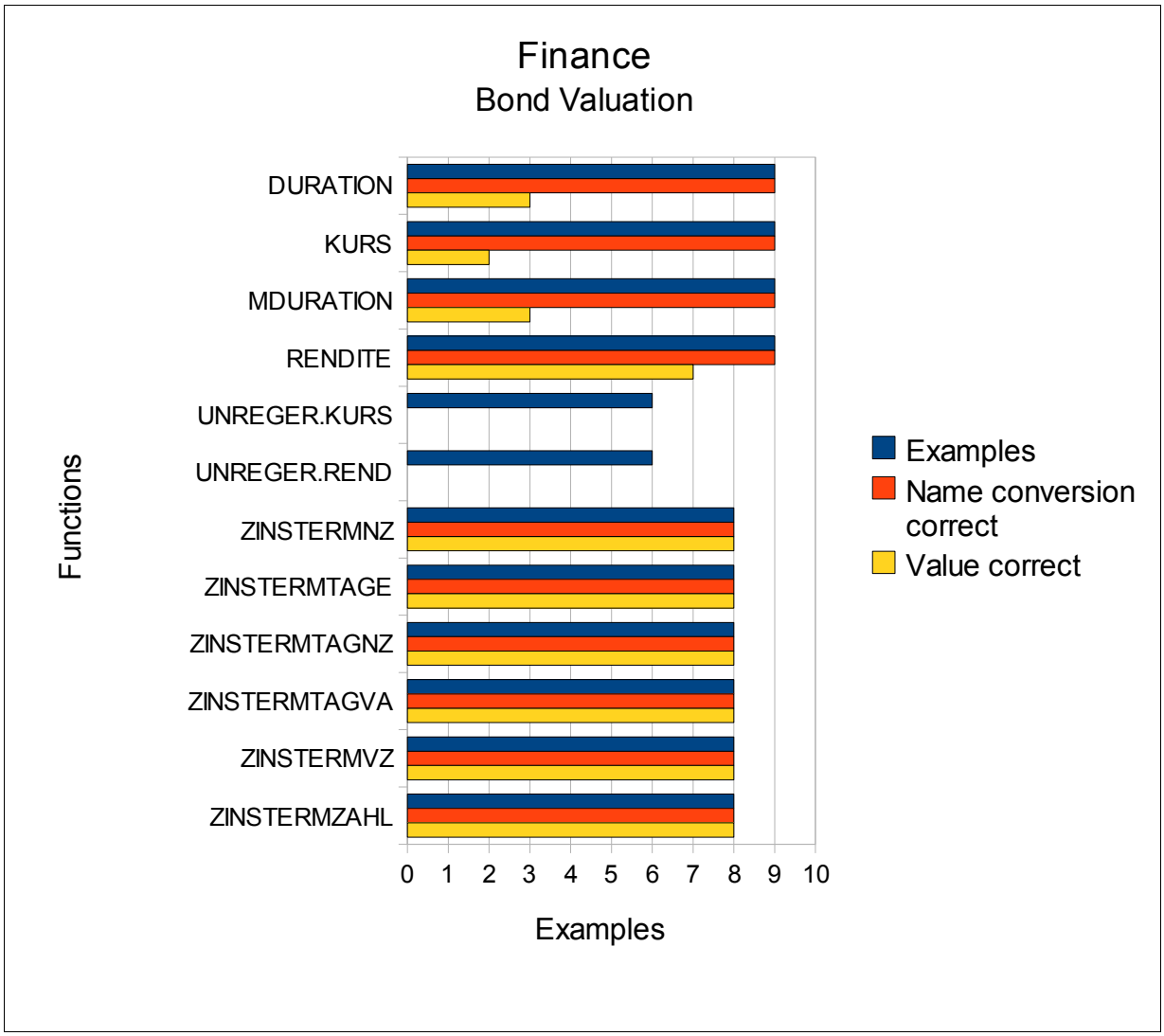


Figure 30: Finance – Bond Valuation

The four functions which are listed at the beginning of Figure 30, were executable in Calc but the calculated values differ from those in Excel. The following Figures show the functions with arguments and values in Excel and Calc.

The function DURATION returns the annual duration of a security with periodic interest payments (see Figures 31 and 32).

1	DURATION(Abrechnung;Fälligkeit;Nominalzins;Rendite;Häufigkeit;*Basis*)						
2							
3	Abrechnung	Fälligkeit	Nominalzins	Rendite	Häufigkeit	*Basis*	DURATION
4	13.04.2008	04.02.2010	0,05	0,025	1	0	1,761795308
5	13.04.2008	04.02.2010	0,05	0,025	1	1	1,764937385
6	13.04.2008	04.02.2010	0,05	0,025	1	2	1,761795308
7	13.04.2008	04.02.2010	0,05	0,025	1	3	1,764420879
8	13.04.2008	04.02.2010	0,05	0,025	1	4	1,761795308
9	13.04.2008	04.02.2010	0,05	0,025	1		1,761795308
10	13.04.2008	04.02.2010	0,05	0,025	1	1	1,764937385
11	13.04.2008	04.02.2010	0,05	0,025	2	1	1,740369943
12	13.04.2008	04.02.2010	0,05	0,025	4	1	1,726430973

Figure 31: Function DURATION in Excel

3	Abrechnun	Fälligkeit	Nominalzin	Rendite	Häufigkeit	*Basis*	DURATION
4	13.04.2008	04.02.2010	0,05	0,03	1	0	1,761795308
5	13.04.2008	04.02.2010	0,05	0,03	1	1	1,767669625
6	13.04.2008	04.02.2010	0,05	0,03	1	2	1,792350864
7	13.04.2008	04.02.2010	0,05	0,03	1	3	1,767160605
8	13.04.2008	04.02.2010	0,05	0,03	1	4	1,761795308
9	13.04.2008	04.02.2010	0,05	0,03	1		1,761795308
10	13.04.2008	04.02.2010	0,05	0,03	1	1	1,767669625
11	13.04.2008	04.02.2010	0,05	0,03	2	1	1,744138033
12	13.04.2008	04.02.2010	0,05	0,03	4	1	1,732305290

Figure 32: Function DURATION in Calc

KURS returns the price per 100 face value of a security that pays periodic interest (see Figures 33 and 34).

15	KURS(Abrechnung;Fälligkeit;Zins;Rendite;Rückzahlung;Häufigkeit;*Basis*)							
16								
17	Abrechnung	Fälligkeit	Zins	Rendite	Rückzahlung	Häufigkeit	*Basis*	KURS
18	13.04.2008	04.02.2010	0,05	0,025	100	1	0	104,3574829
19	13.04.2008	04.02.2010	0,05	0,025	100	1	1	104,3650226
20	13.04.2008	04.02.2010	0,05	0,025	100	1	2	104,3574829
21	13.04.2008	04.02.2010	0,05	0,025	100	1	3	104,3637832
22	13.04.2008	04.02.2010	0,05	0,025	100	1	4	104,3574829
23	13.04.2008	04.02.2010	0,05	0,025	100	1		104,3574829
24	13.04.2008	04.02.2010	0,05	0,025	100	1	1	104,3650226
25	13.04.2008	04.02.2010	0,05	0,025	100	2	1	104,3947289
26	13.04.2008	04.02.2010	0,05	0,025	100	4	1	104,4060213

Figure 33: Function KURS in Excel

17	Abrechnun	Fälligkeit	Zins	Rendite	Rückzahlung	Häufigkeit	*Basis*	KURS
18	13.04.2008	04.02.2010	0,05	0,03	100	1	0	104,3574829
19	13.04.2008	04.02.2010	0,05	0,03	100	1	1	104,3650226
20	13.04.2008	04.02.2010	0,05	0,03	100	1	2	104,3141498
21	13.04.2008	04.02.2010	0,05	0,03	100	1	3	104,3566591
22	13.04.2008	04.02.2010	0,05	0,03	100	1	4	104,3574829
23	13.04.2008	04.02.2010	0,05	0,03	100	1		104,3574829
24	13.04.2008	04.02.2010	0,05	0,03	100	1	1	104,3650226
25	13.04.2008	04.02.2010	0,05	0,03	100	2	1	104,3947289
26	13.04.2008	04.02.2010	0,05	0,03	100	4	1	104,4060213

Figure 34: Function KURS in Calc

The function MDURATION returns the Macauley modified duration for a security with an assumed par value of 100 (see Figures 35 and 36).

29	MDURATION(Abrechnung;Fälligkeit;Nominalzins;Rendite;Häufigkeit;*Basis*)						
30							
31	Abrechnung	Fälligkeit	Nominalzins	Rendite	Häufigkeit	*Basis*	MDURATION
32	13.04.2008	04.02.2010	0,05	0,025	1	0	1,718824691
33	13.04.2008	04.02.2010	0,05	0,025	1	1	1,721890132
34	13.04.2008	04.02.2010	0,05	0,025	1	2	1,718824691
35	13.04.2008	04.02.2010	0,05	0,025	1	3	1,721386224
36	13.04.2008	04.02.2010	0,05	0,025	1	4	1,718824691
37	13.04.2008	04.02.2010	0,05	0,025	1		1,718824691
38	13.04.2008	04.02.2010	0,05	0,025	1	1	1,721890132
39	13.04.2008	04.02.2010	0,05	0,025	2	1	1,718883895
40	13.04.2008	04.02.2010	0,05	0,025	4	1	1,7157078

Figure 35: Function MDURATION in Excel

3	Abrechnun	Fälligkeit	Nominalzin	Rendite	Häufigkeit	*Basis*	DURATION
4	13.04.2008	04.02.2010	0,05	0,03	1	0	1,761795308
5	13.04.2008	04.02.2010	0,05	0,03	1	1	1,767669625
6	13.04.2008	04.02.2010	0,05	0,03	1	2	1,792350864
7	13.04.2008	04.02.2010	0,05	0,03	1	3	1,767160605
8	13.04.2008	04.02.2010	0,05	0,03	1	4	1,761795308
9	13.04.2008	04.02.2010	0,05	0,03	1		1,761795308
10	13.04.2008	04.02.2010	0,05	0,03	1	1	1,767669625
11	13.04.2008	04.02.2010	0,05	0,03	2	1	1,744138033
12	13.04.2008	04.02.2010	0,05	0,03	4	1	1,732305290

Figure 36: Function DURATION in Calc

RENDITE returns the yield on a security that pays periodic interest (see Figures 37 and 38).

43	RENDITE(Abrechnung;Fälligkeit;Zins;Kurs;Rückzahlung;Häufigkeit;*Basis*)							
44								
45	Abrechnung	Fälligkeit	Zins	Kurs	Rückzahlung	Häufigkeit	*Basis*	RENDITE
46	13.04.2008	04.02.2010	0,05	104,357	100	1	0	0,025002668
47	13.04.2008	04.02.2010	0,05	104,357	100	1	1	0,025044246
48	13.04.2008	04.02.2010	0,05	104,357	100	1	2	0,025002668
49	13.04.2008	04.02.2010	0,05	104,357	100	1	3	0,025037421
50	13.04.2008	04.02.2010	0,05	104,357	100	1	4	0,025002668
51	13.04.2008	04.02.2010	0,05	104,357	100	1		0,025002668
52	13.04.2008	04.02.2010	0,05	104,357	100	1	1	0,025044246
53	13.04.2008	04.02.2010	0,05	104,357	100	1	1	0,025044246
54	13.04.2008	04.02.2010	0,05	104,357	100	1	1	0,025044246

Figure 37: Function RENDITE in Excel

45	Abrechnun	Fälligkeit	Zins	Kurs	Rückzahlung	Häufigkeit	*Basis*	RENDITE
46	13.04.2008	04.02.2010	0,05	104,36	100	1	0	0,025002668
47	13.04.2008	04.02.2010	0,05	104,36	100	1	1	0,025044246
48	13.04.2008	04.02.2010	0,05	104,36	100	1	2	0,024765481
49	13.04.2008	04.02.2010	0,05	104,36	100	1	3	0,024998122
50	13.04.2008	04.02.2010	0,05	104,36	100	1	4	0,025002668
51	13.04.2008	04.02.2010	0,05	104,36	100	1		0,025002668
52	13.04.2008	04.02.2010	0,05	104,36	100	1	1	0,025044246
53	13.04.2008	04.02.2010	0,05	104,36	100	1	1	0,025044246
54	13.04.2008	04.02.2010	0,05	104,36	100	1	1	0,025044246

Figure 38: Function RENDITE in Calc

3.1.4 Information Functions

The information functions can return information concerning the content (e. g. text, value, mistakes) and the status (e. g. formatting) of cells. The functions can be divided into three sub-categories (see Table 4).

Sub-categories	Counted functions
General Functions	7
Ist-Functions	11
Cell	1

Table 4: Sub-categories of Information Functions

In Figure 39 the findings of the general functions are shown. The function ANZAHLLEEREZELLEN counts the number of blank cells within a range. The difference between Calc and Excel is, that Calc interprets the input =”” as value whereas Excel interprets a cell with this value as blank.

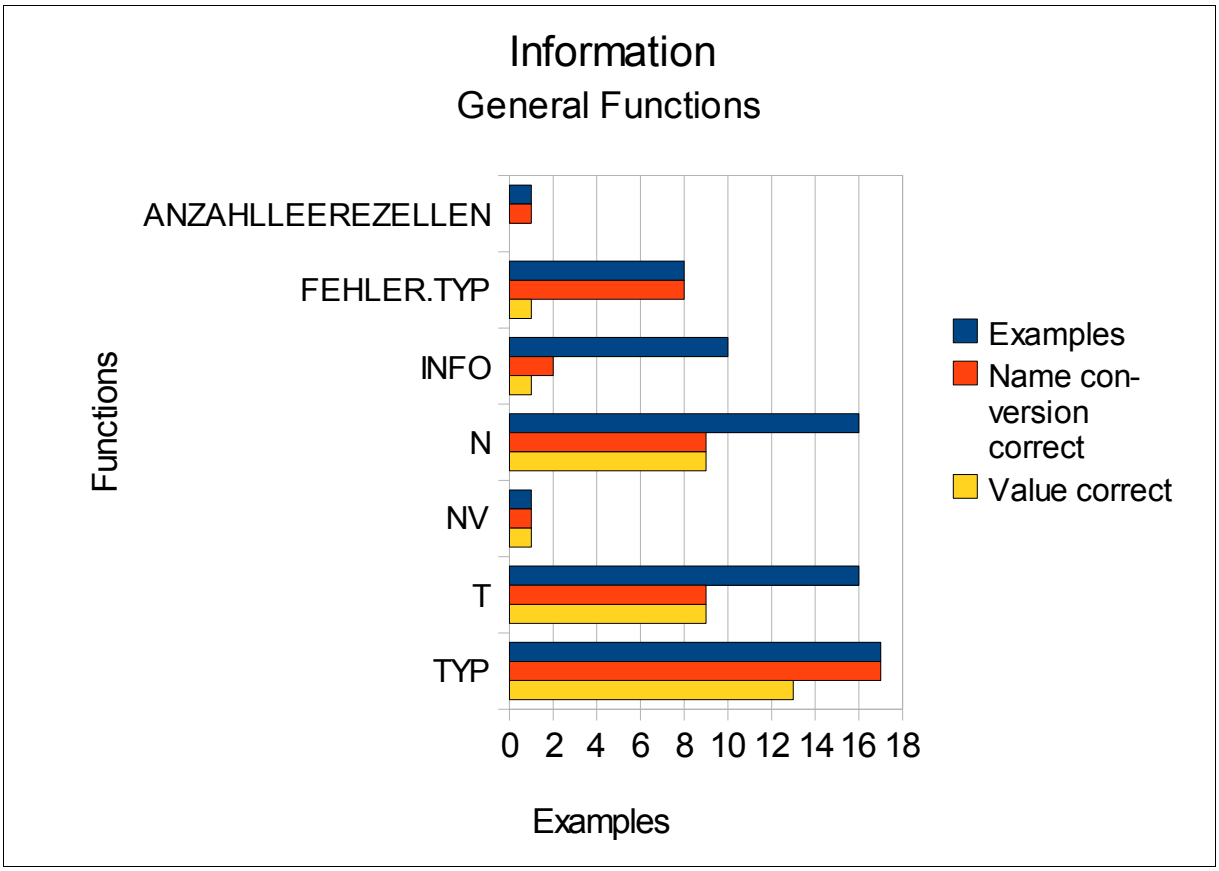


Figure 39: Information – General Functions

The function FEHLER.TYP returns a number corresponding to an error type. As in Excel there are different error types than in Calc, only one example returns the same value (when no error appears the value #NV is returned).

The function INFO returns information about the current operating environment (e. g. the used memory). As this function is Excel specific, only one example returns a value which was correct.

The function N returns a value converted to a number and the function T returns a value converted to a text. In Calc these functions work correctly with text, numbers, logical values, dates and time values. As in Excel there are different error types than in Calc the functions N and T do not work correctly with error types.

The function TYP returns a number indicating the data type of a value. This function does not work in Calc with the arguments =*""*, WAHR, FALSCH and references.

In Figure 40 the eleven Ist-Functions are shown, which enable the user to query special information of cells.

The function ISTFEHL returns TRUE if the value is any error value except #N/A. The error value #NV delivers in Calc the return value TRUE, whereas in Excel FALSE is returned.

The function ISTNV returns TRUE if the value is the #N/A error value. The error value #NV delivers in Calc the return value FALSE, whereas in Excel TRUE is returned. The other seven examples were executed correctly.

The function ISTZAHL returns TRUE if the value is a number. As Calc interprets the logic values TRUE and FALSE as numbers, for these values the function returns TRUE. In Excel the return value is FALSE.

The third sub-category of the information functions is *Cell*. This sub-category only has the function CELL, which is very powerful. It returns information about the formatting, location or content of a cell. As this function is Excel specific, only seven of the 54 examples work correctly in Calc.

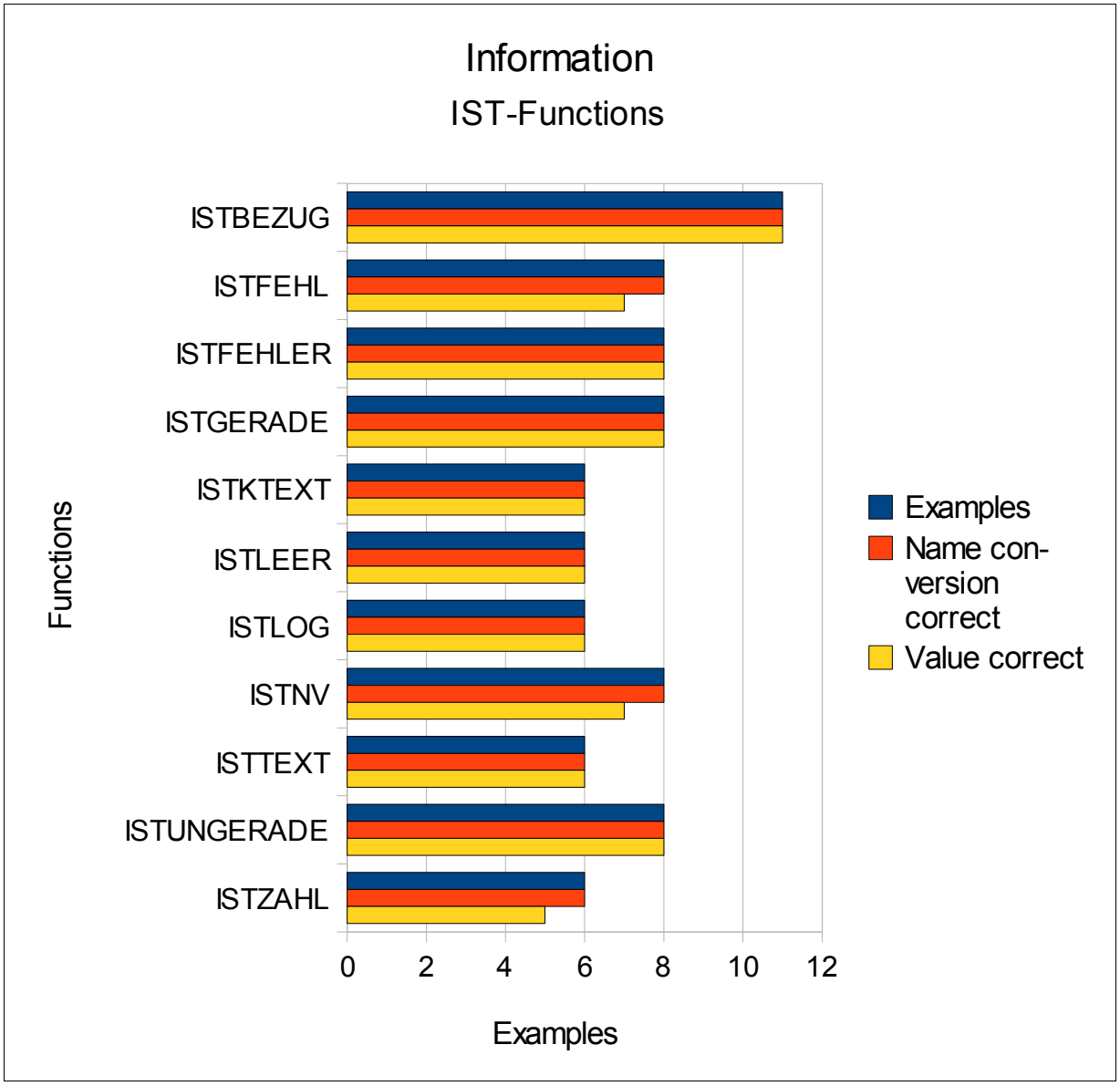


Figure 40: Information – IST-Functions

3.1.5 Logic Functions

Logic functions can create or combine logical values. With the function WENN a logical test can be specified, which can evaluate logical values and afterwards execute flows. Calc was able to execute all developed examples.

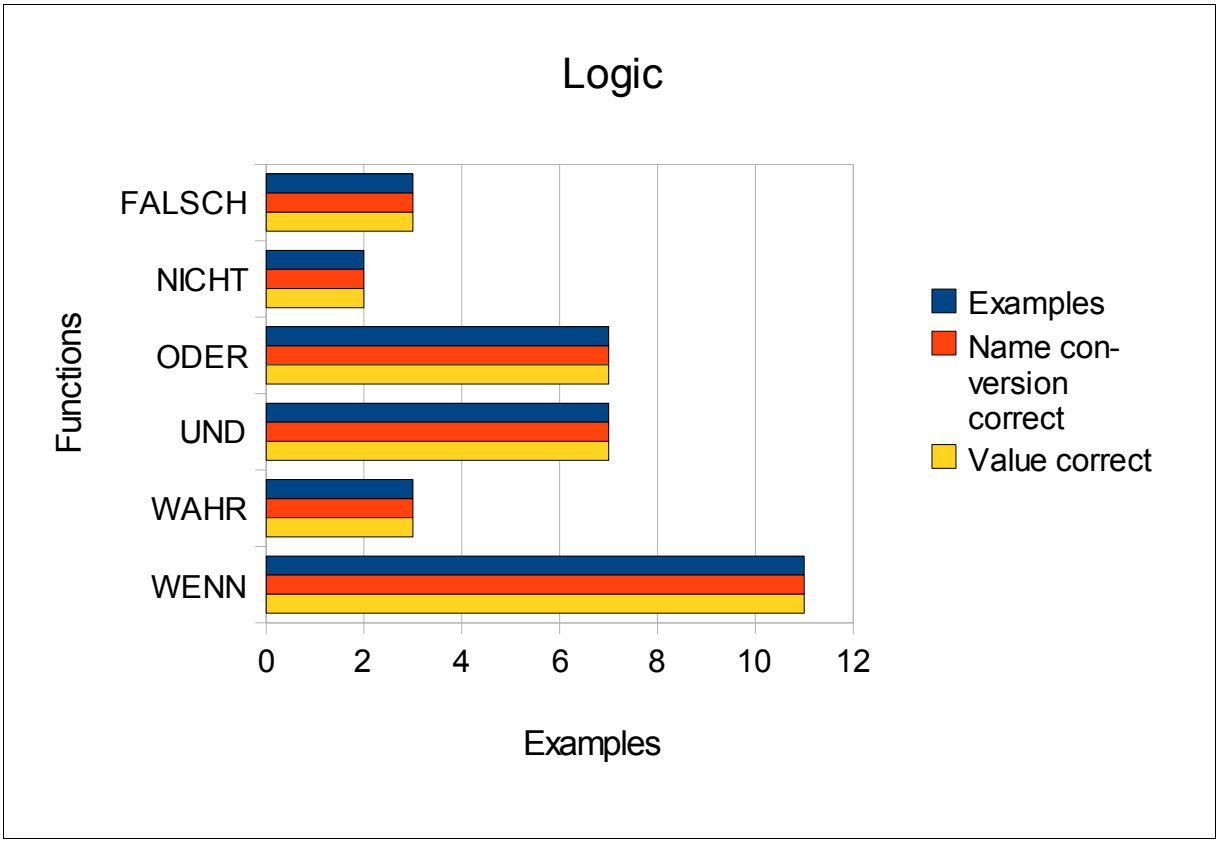


Figure 41: Logic Functions

3.1.6 Mathematical Functions

The functions for Mathematics can be divided into different sub-categories, which are shown in Table 5.

Sub-categories	Counted functions
Basic Arithmetic Operations	27
Advanced Arithmetic Operations	8
Rounding	11
Matrices	7

Table 5: Sub-categories of Mathematics

Figure 42 shows the functions for basic arithmetic operations, where only in one function a problem occurred. The function SUMME adds its arguments, but only numbers (text will be ignored). In Calc also the argument WAHR is added, which can result in a different return value.

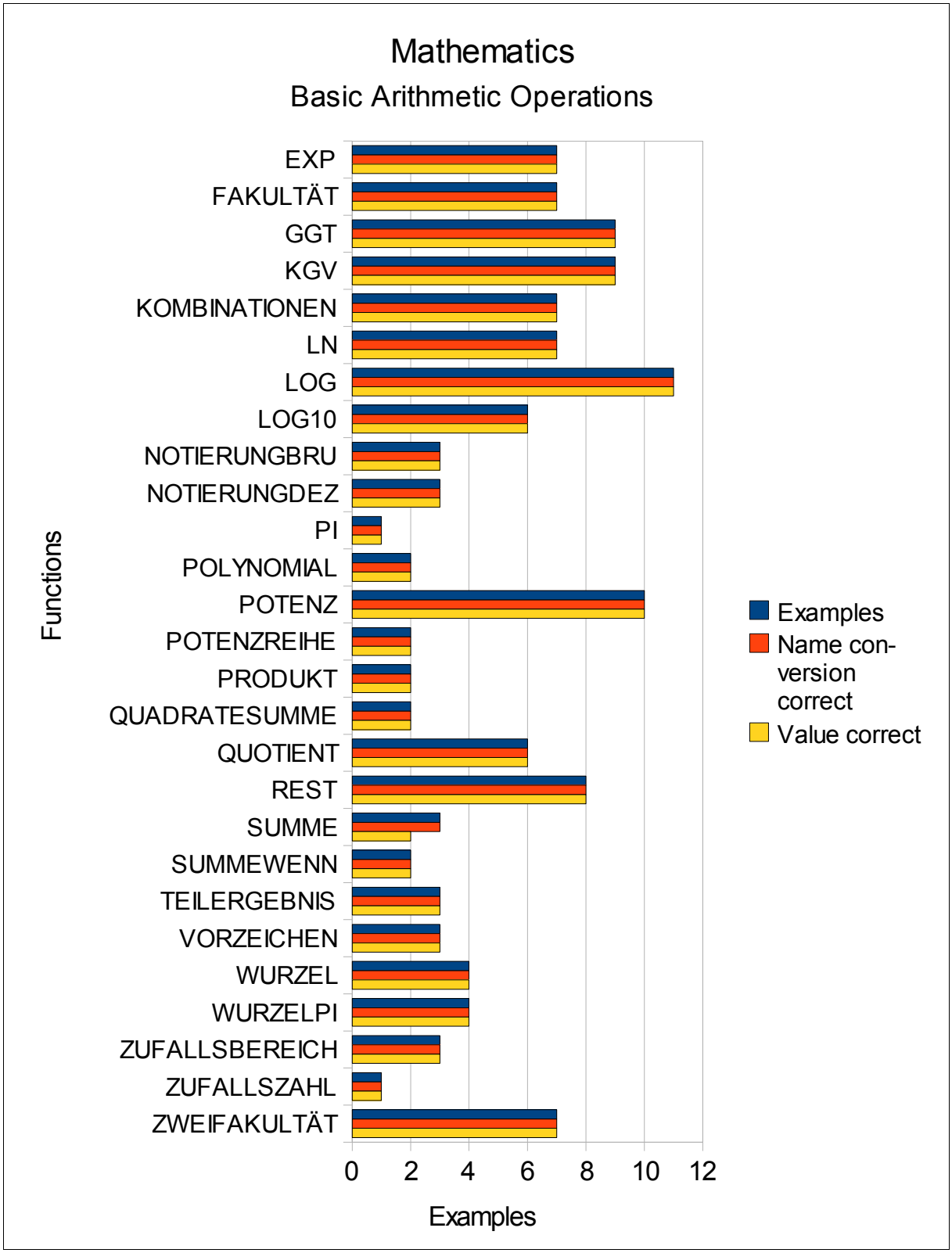


Figure 42: Mathematics – Basic Arithmetic Operations

Functions for advanced arithmetic operations are shown in Figure 43, which shows that all functions were executable in Calc but some did return different values. The functions

BESELL? return the modified Bessel functions In(x), Jn(x), Kn(x) and Yn(x). Differences in return values occurred only after the seventh digit right to the decimal point. As example the function for BESELLI is shown in both programs (see Figures 44 and 45).

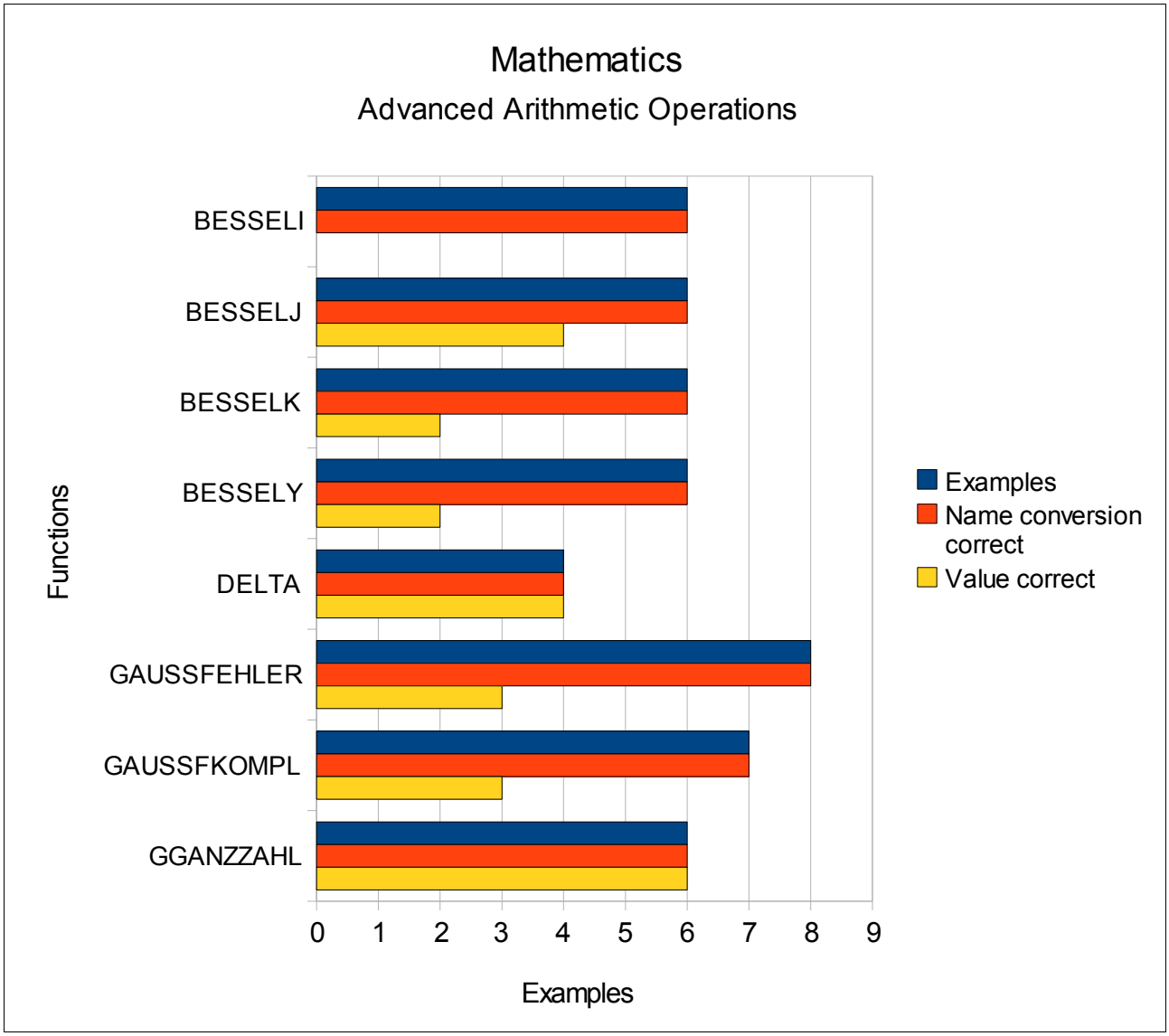


Figure 43: Mathematics – Advanced Arithmetic Operations

1	**BESSELI(x;n)**			
2				
3	x	n		BESSELI
4	0,25	0		1,01568613261
5	0,5	1		0,25789430329
6	1	2		0,13574766658
7	1,5	3		0,08077411362
8	2	4		0,05072857009
9	2	4,5		0,05072857009

Figure 44: Function BESSELI in Excel

1	BESSELI(x;n)			
2				
3	x	n		BESSELI
4	0,25	0		1,01568614122
5	0,5	1		0,25789430539
6	1	2		0,13574766977
7	1,5	3		0,08077411302
8	2	4		0,05072856998
9	2	4,5		0,05072856998

Figure 45: Function BESELI in Calc

The functions GAUSSFEHLER and GAUSSFKOMPL return the error function respectively the complementary error function. Both functions returned values which were different from those in Excel in some digits right to the decimal point.

The third sub-category of mathematical functions is used for rounding and contains eleven functions (see Figure 46), which were all executable correctly in Calc.

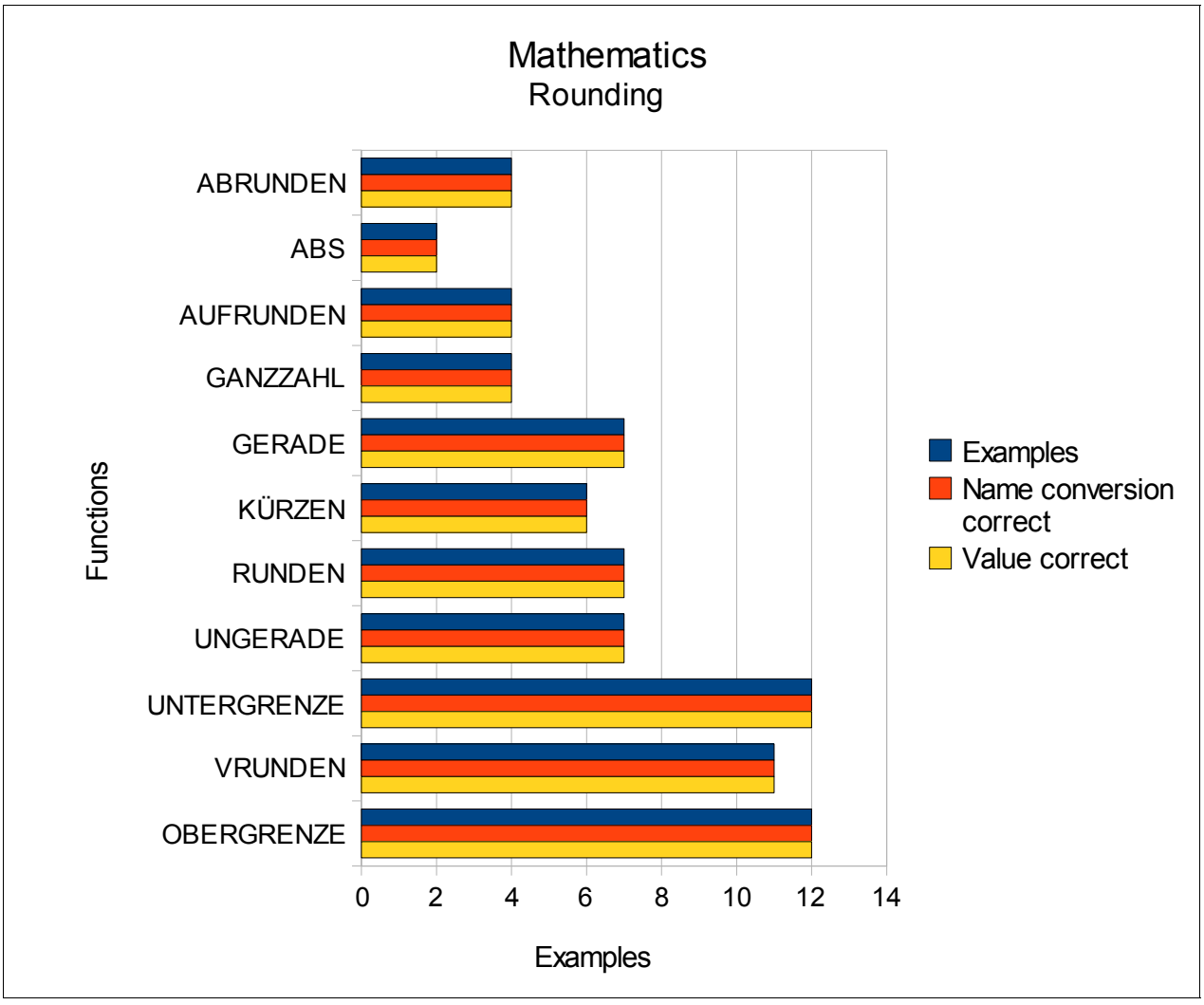


Figure 46: Mathematics – Functions for Rounding

Figure 47 shows the functions for the sub-category *Matrices*, which were all executable correctly in Calc.

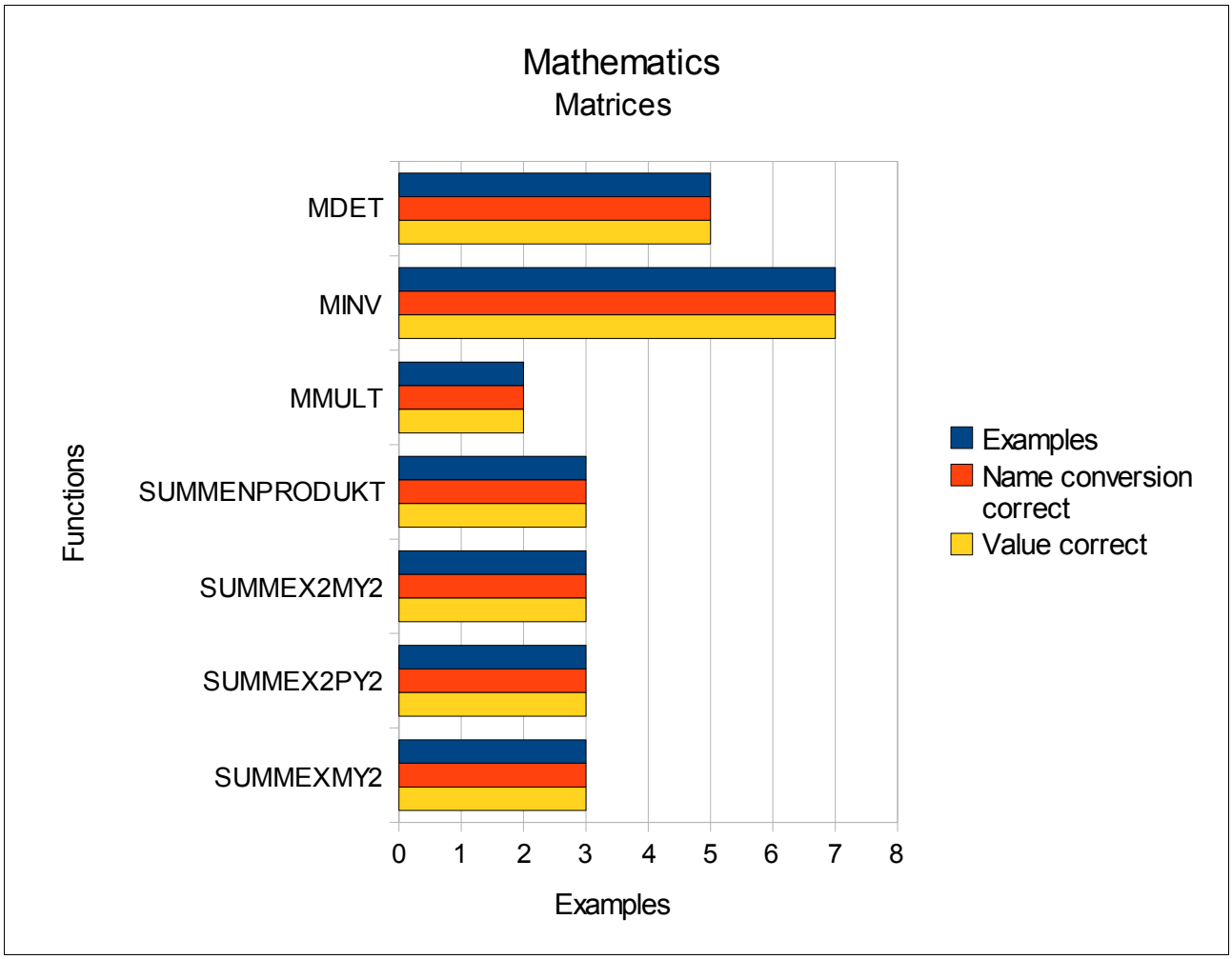


Figure 47: Mathematics – Functions for Matrices

3.1.7 Search and Range Functions

In Figure 48 the search and range functions are shown, which can be used to extract specific data from parts of a workbook.

The function ADDRESSE returns a reference as text to a single cell in a worksheet. In the Figures 49 and 50 the examples for this function in Excel and Calc are shown. The optional argument *Abs* decides, whether the references are absolute or relative (e. g. 1: row and column are absolute). The optional argument *A1* decides whether the source is in the form *A1* or *Z1S1* (if argument is empty: *A1*). The argument *Tabellenname* is also optional and can be used to refer to different sheets in the workbook.

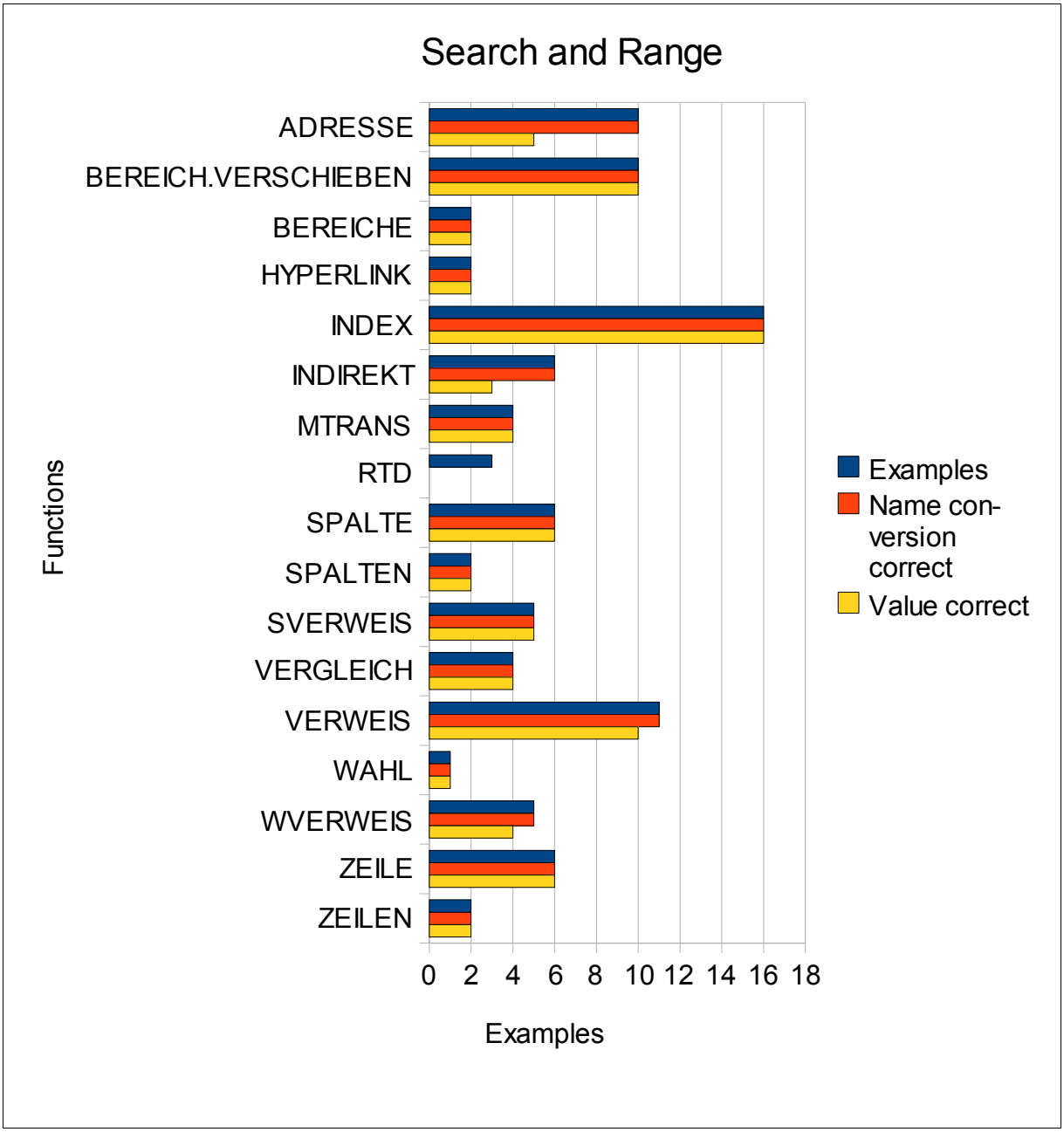


Figure 48: Search and Range Functions

9	ADRESSE(Zeile;Spalte;*Abs;A1;Tabellenname*)					
10						
11	Zeile	Spalte	*Abs*	*A1*	*Tabellenname*	ADRESSE
12	2	2				B2
13	2	2	1			B2
14	2	2	2			B$2
15	2	2	3			$B2
16	2	2	4			B2
17	2	2		WAHR		B2
18	2	2		1		B2
19	2	2		FALSCH		Z2S2
20	2	2		0		Z2S2
21	2	2			Tabelle	Tabelle!B2

Figure 49: Function ADRESSE in Excel

9	ADRESSE(Zeile; Spalte; Abs; A1; Tabellenname)					
10						
11	Zeile	Spalte	Abs	A1	Tabellenname	ADRESSE
12	2	2				B2
13	2	2	1			B2
14	2	2	2			B$2
15	2	2	3			$B2
16	2	2	4			B2
17	2	2		WAHR		#WERT!
18	2	2		1		#WERT!
19	2	2		FALSCH		#WERT!
20	2	2		0		#WERT!
21	2	2			Tabelle	#WERT!

Figure 50: Function ADRESSE in Calc

The function INDIREKT returns a reference indicated by a text value. Whereas the examples were executable in Calc when the text value was in the form *A1,* they where not executable when the text was in the form *Z1S1.*

The function RTD retrieves real-time data from a program that supports COM automation. The example, which was tested was not executable in Calc.

VERWEIS looks up values in a vector or an array. One example of this function returned a different value in Calc.

Also the function WVERWEIS, which looks in the top row of an array and returns the value of the indicated cell, returned a different value in Calc.

3.1.8 Statistical Functions

The functions for Statistics can be divided into different sub-categories, which are shown in Table 6.

Sub-categories	Counted functions
Counting	12
Mean	11
Likelihood	34
Variance	8
Symmetry	2
Regression	12

Table 6: Sub-categories of Statistics

In Figure 51 the findings for the twelve counting functions are shown.

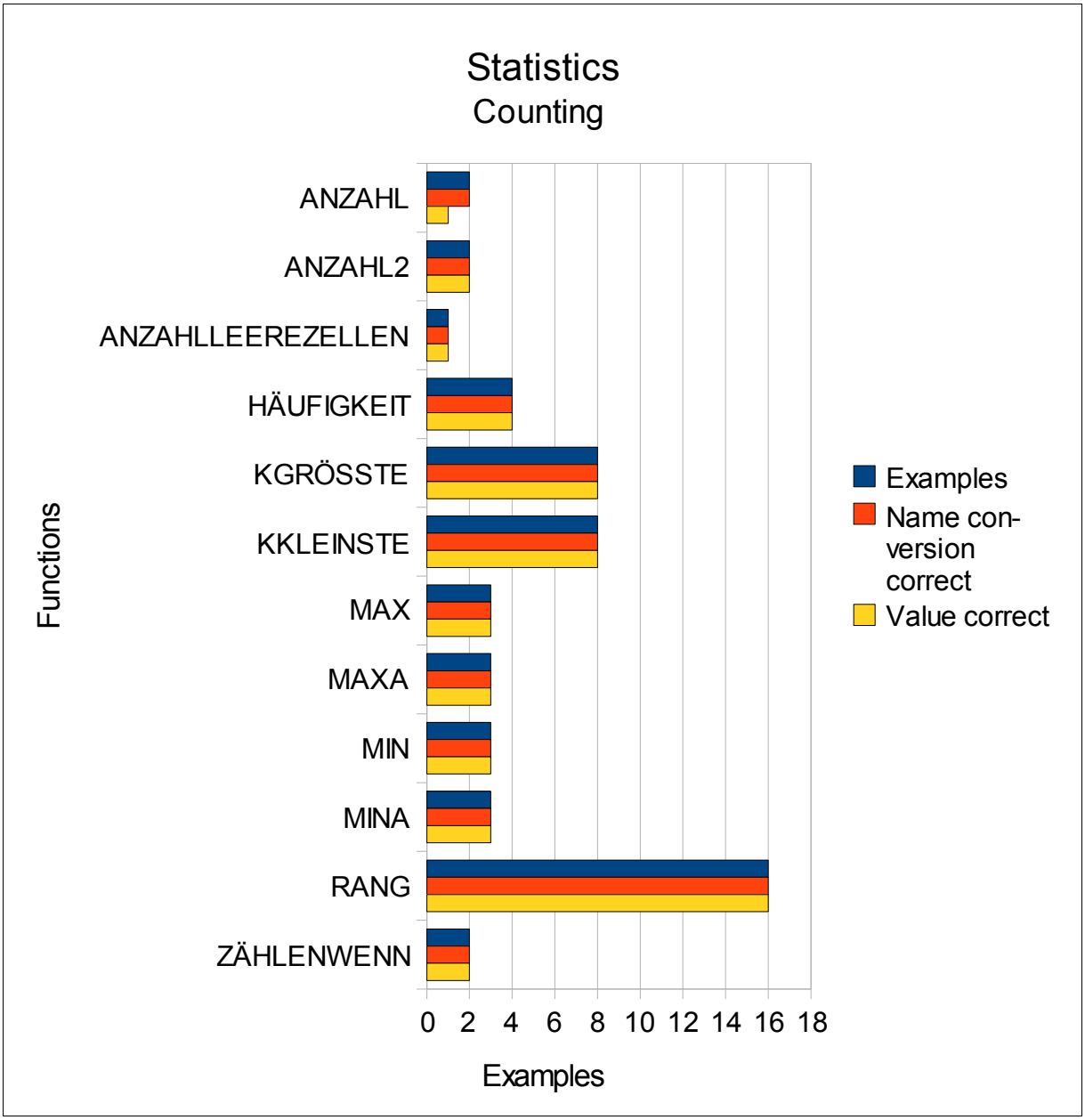


Figure 51: Statistics – Counting

The function ANZAHL counts how many numbers are in the list of arguments. There are differences in the outcome, because Calc interprets the value WAHR as 1, whereas Excel ignores this value.

In Figure 52 the findings for the statistical functions for the category *Mean* are shown. MEDIAN returns the median of the given numbers and MITTELABW returns the average of the absolute deviations of data points from their mean. The functions MITTELWERT

returns the average of its arguments and the function MODALWERT returns the most common value in a data set. The differences in these four functions occurred, because Calc interprets the value WAHR as 1 and Excel ignores it.

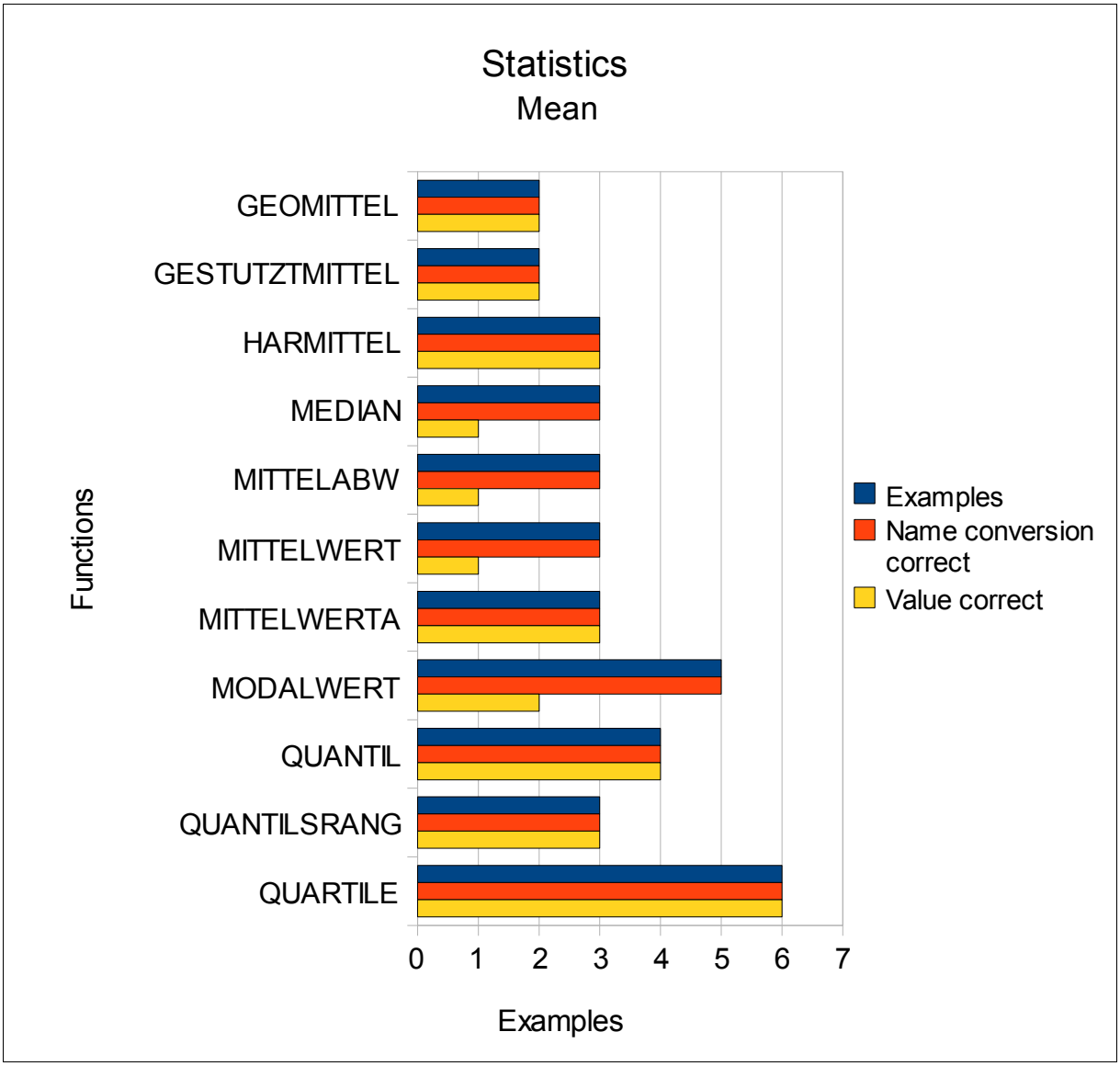


Figure 52: Statistics – Mean

In Figure 53 the findings of the category *Likelihood* are presented. The function CHIINV returns the inverse of the one-tailed probability of the chi-squared distribution whereas CHITEST returns the test for independence. The function CHIVERT returns the one-tailed probability of the chi-squared distribution. It was possible to execute these three functions in Calc, but the values differ from those in Excel. The differences occurred at the seventh and ninth digit right to the decimal point.

The function FTEST returns the result of an F-test, which determines the two-tailed probability that the variances in array1 and array2 are not significantly different.

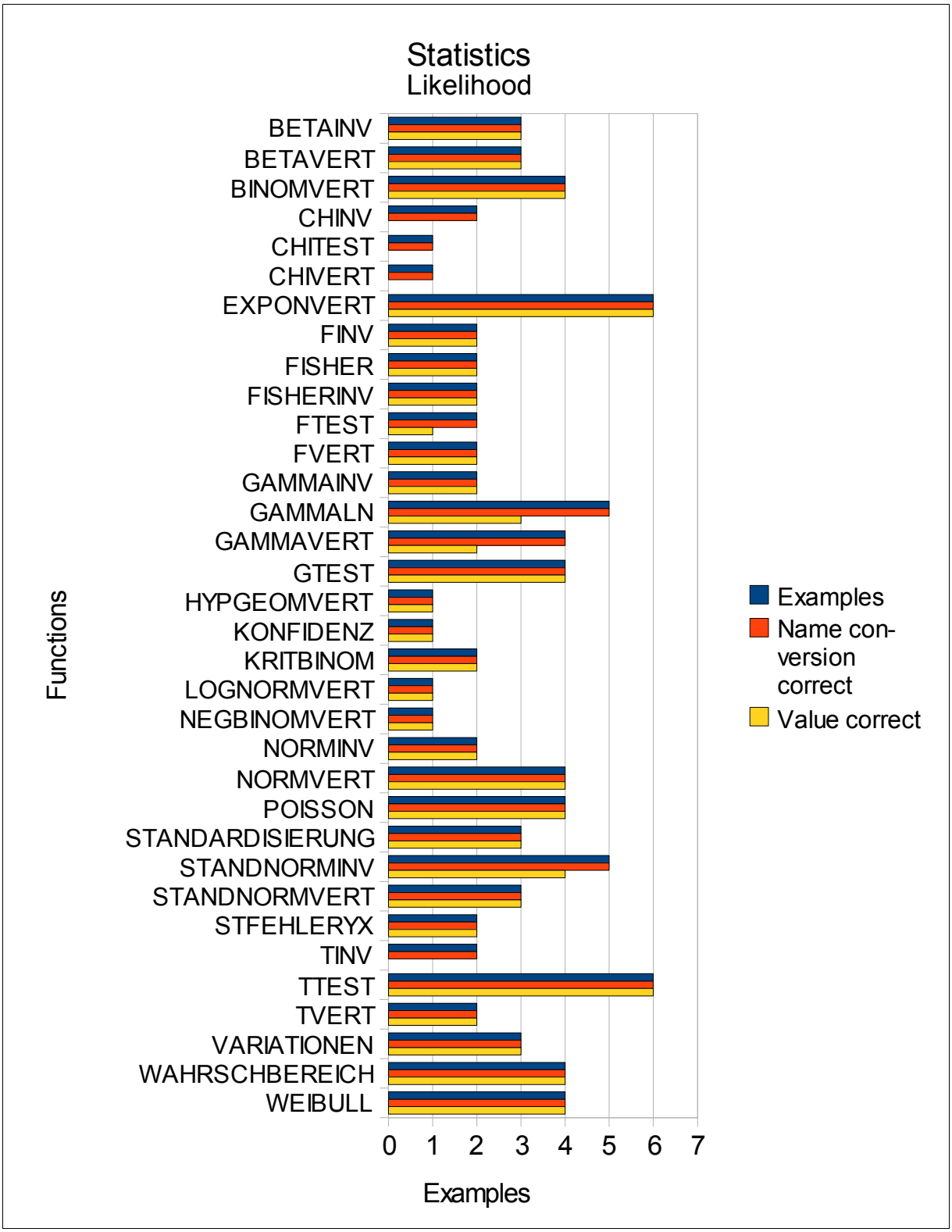


Figure 53: Statistics – Likelihood

The function GAMMALN returns the natural logarithm of the gamma function and GAMMAVERT returns the gamma distribution. In both functions there occurred some different values (see Figures 54 and 55).

107	GAMMALN(x)					
108						
109	x					GAMMALN
110	-1					#ZAHL!
111	0					#ZAHL!
112	1					-4,1716E-11
113	2					-8,57678E-11
114	3					0,69314718
115						
116						
117	GAMMAVERT(x;Alpha;Beta;Kumuliert)					
118						
119	x	Alpha	Beta	Kumuliert		GAMMAVERT
120	10	3,7	1,5	WAHR		0,9222811986
121	10	3,7	1,5	1		0,9222811986
122	10	3,7	1,5	FALSCH		0,0341162953
123	10	3,7	1,5	0		0,0341162953

Figure 54: Functions GAMMALN and GAMMAVERT in Excel

107	GAMMALN(x)					
108						
109	x					GAMMALN
110	-1					Err:502
111	0					Err:502
112	1					0,000000000000
113	2					0,000000000000
114	3					0,69314718
115						
116						
117	GAMMAVERT(x;Alpha;Beta;Kumuliert)					
118						
119	x	Alpha	Beta	Kumuliert		GAMMAVERT
120	10	3,7	1,5	WAHR		0,9222811979
121	10	3,7	1,5	1		0,9222811979
122	10	3,7	1,5	FALSCH		0,0341162953
123	10	3,7	1,5	0		0,0341162953

Figure 55: Functions GAMMALN and GAMMAVERT in Calc

The function STANDNORMINV returns the inverse of the standard normal cumulative distribution. This function returns a different value for 0,5.

The function TINV returns the inverse of the Student's t-distribution. As it can be seen from Figure 53, the function was executable in Calc but the values differ from those in Excel. The differences for the two examples occurred at the eighth and ninth digit right to the decimal point.

In Figure 56 the findings of the statistical functions of the category *Variance* are shown. The differences in the return values occurred, because Calc interprets the value WAHR as 1 and FALSCH as 0 and integrates these values into the calculation. Excel, on the other side, doesn't integrate the values WAHR and FALSCH into the calculation.

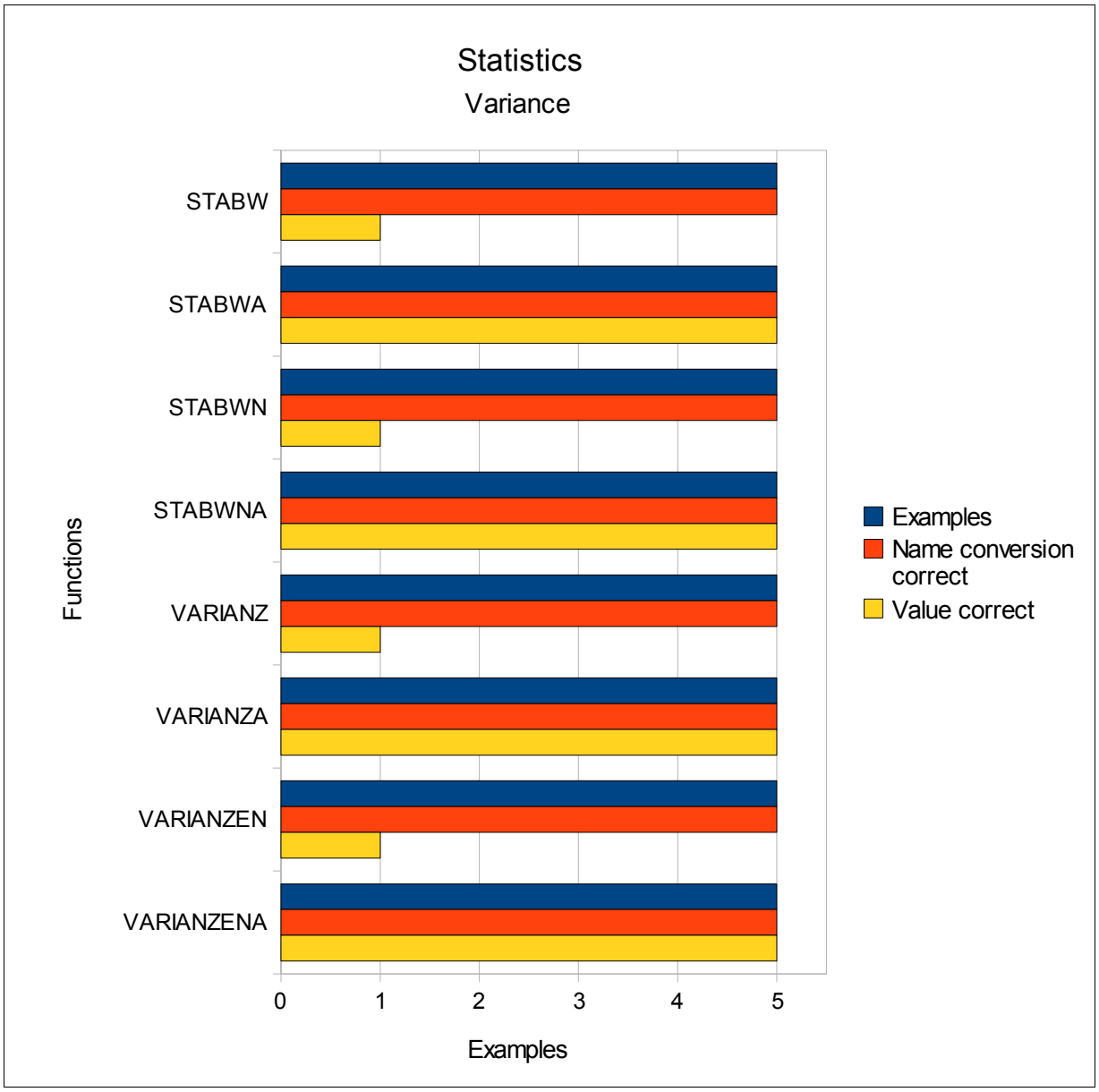


Figure 56: Statistics – Variance

The functions in Excel which end with A integrate the values WAHR and FALSCH into the calculation and therefore the values from these functions are the same as in Calc.

Figure 57 shows the two functions KURT and SCHIEFE from the statistical sub-category *Symmetry*. The function SCHIEFE returns the skewness of a distribution. There are

differences in the returned values, because of the same reason as in the function STABW, which is described above. Excel does not integrate the values WAHR and FALSCH into the calculation whereas Calc does this.

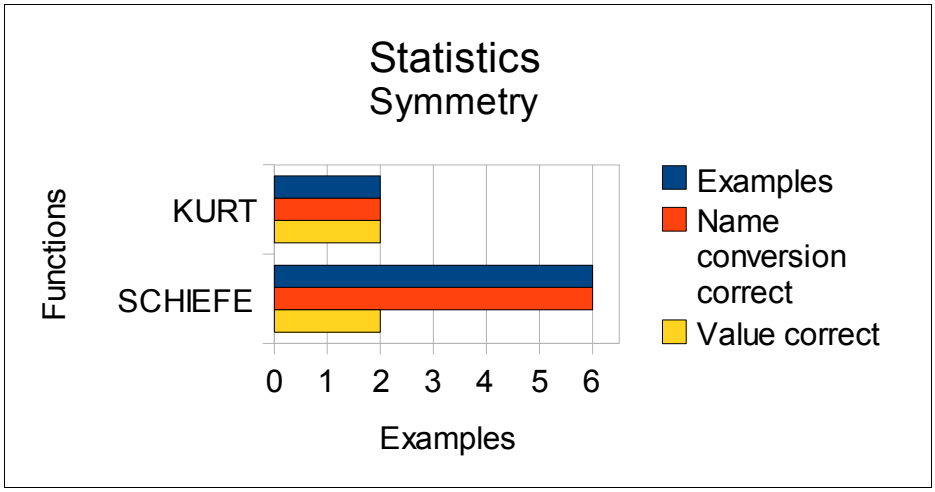


Figure 57: Statistics – Symmetry

In Figure 58 the functions for regression are shown.

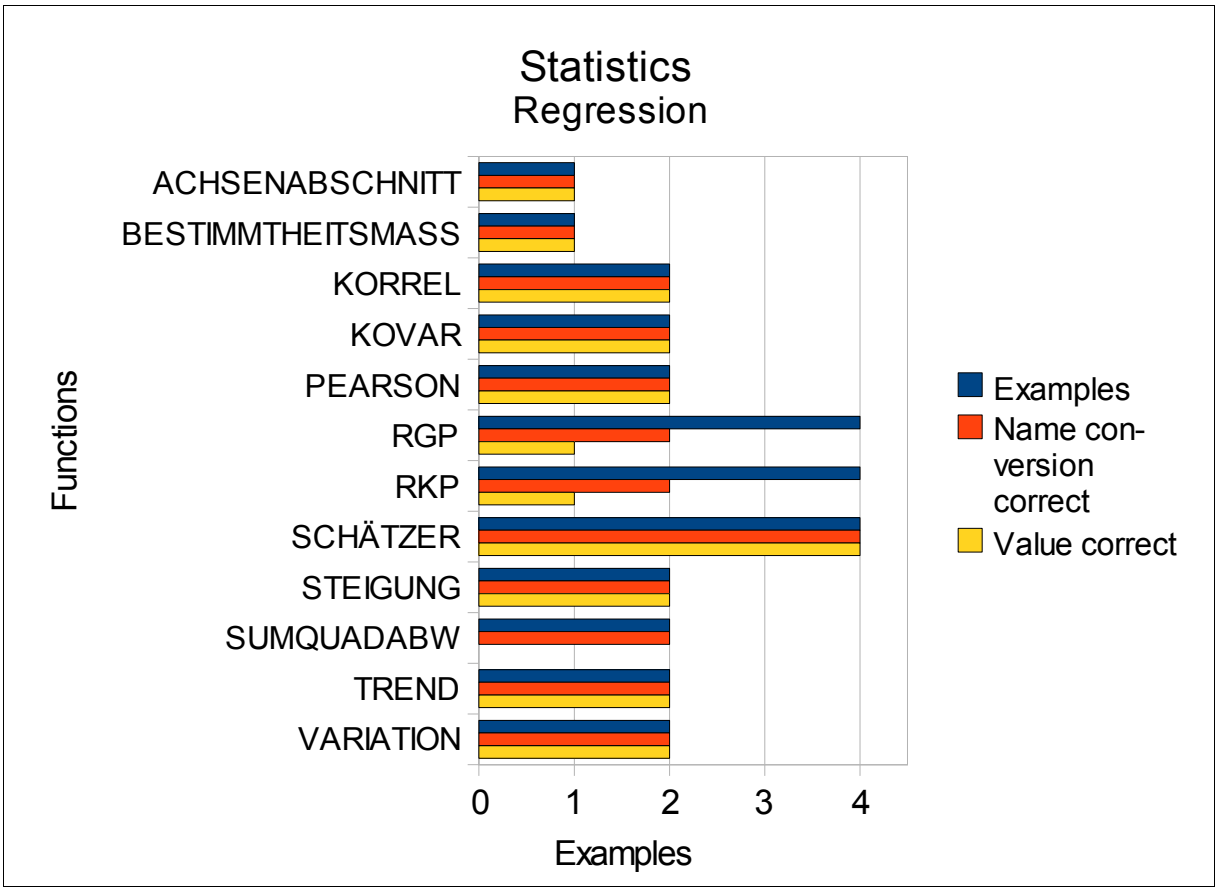


Figure 58: Statistics - Regression

The differences in SUMQUADABW are because of the different interpretation of the values WAHR and FALSCH (see STABW above). The function RGP returns the parameters of a linear trend and RKP returns the parameters of an exponential trend. Both functions were tested with the possible arguments but from all examples only one out of four has correct results.

3.1.9 Engineering

Functions from the category *Engineering* can be divided into two sub-categories. The transformation functions can be used to transform a value from a system to an other one, e. g. numbers or measures (see Figure 59). In the second sub-category are functions for complex data (see Figure 62).

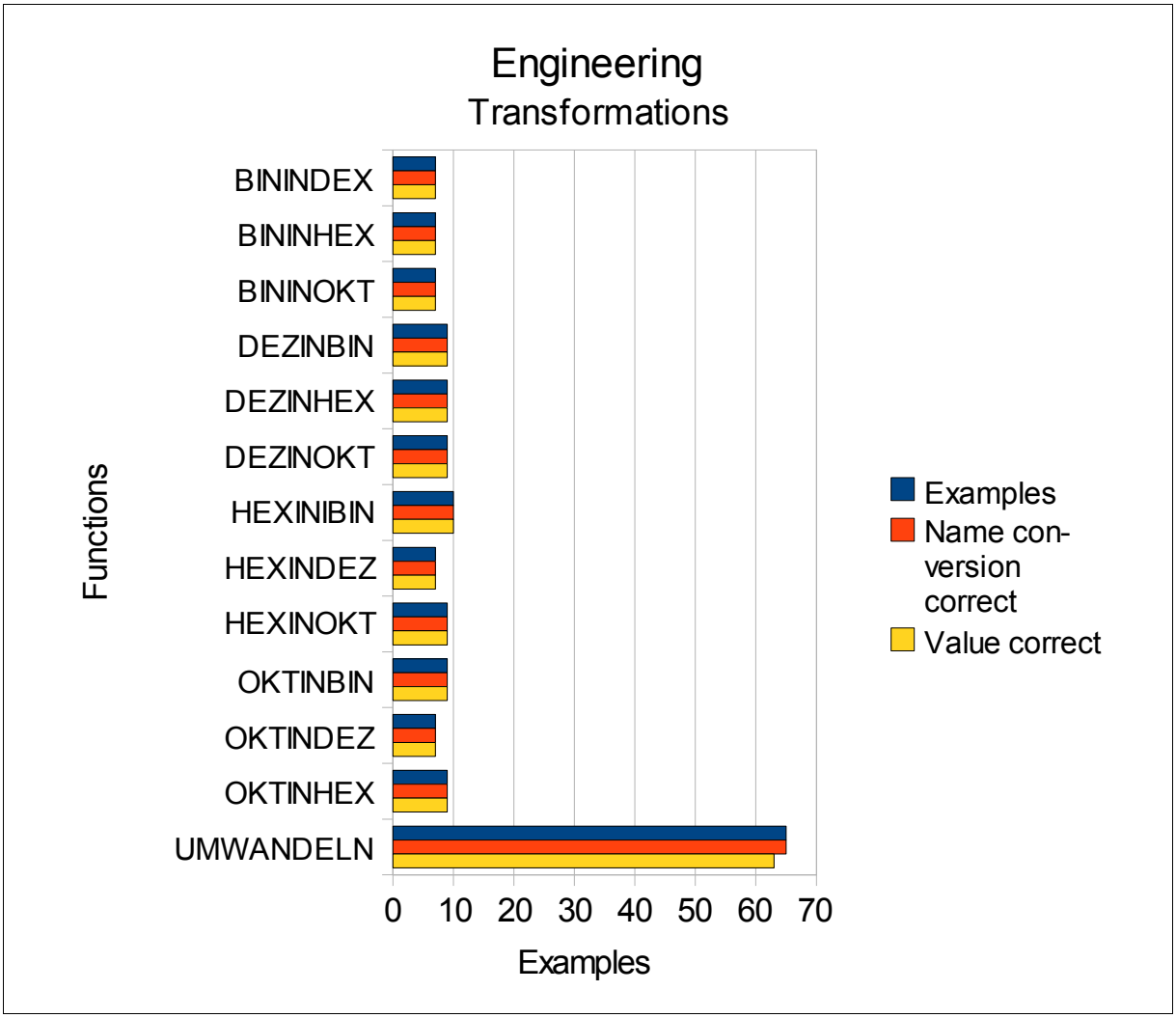


Figure 59: Engineering – Transformations

Figure 59 shows that hardly all examples of the category *Transformations* were executable in Calc. Only two examples of the function UMWANDELN delivered different return values.

In the Figures 60 and 61 the different values for transforming horsepower into watt respectively watt into horsepower are shown.

207	Leistung	Zahl	Von_Maßeinheit	In_Maßeinheit	UMWANDELN
208	Pferdestärke	1	HP	W	745,7010000000
209	Watt	1	W	HP	0,0013410201

Figure 60: Function UMWANDELN in Excel

207	Leistung	Zahl	Von_Maßeinheit	In_Maßeinheit	UMWANDELN
208	Pferdestärke	1	HP	W	745,6999214032
209	Watt	1	W	HP	0,0013410220

Figure 61: Function UMWANDELN in Calc

Figure 62 shows that the functions of category *Complex Data* were all executable in Calc but some return values were different from those in Excel.

The function IMDIV returns the quotient of two complex numbers. Whereas Excel was not able to return a quotient for the two arguments -2+3j and -2+3i, Calc returns the value 1.

The functions IMSUB and IMSUMME return the difference between two complex numbers respectively the sum of complex numbers. Both functions were not executable in Excel with the two arguments -2+3j and -2+3i. Calc returned the value 0 for IMSUB and -4+6j for IMSUMME.

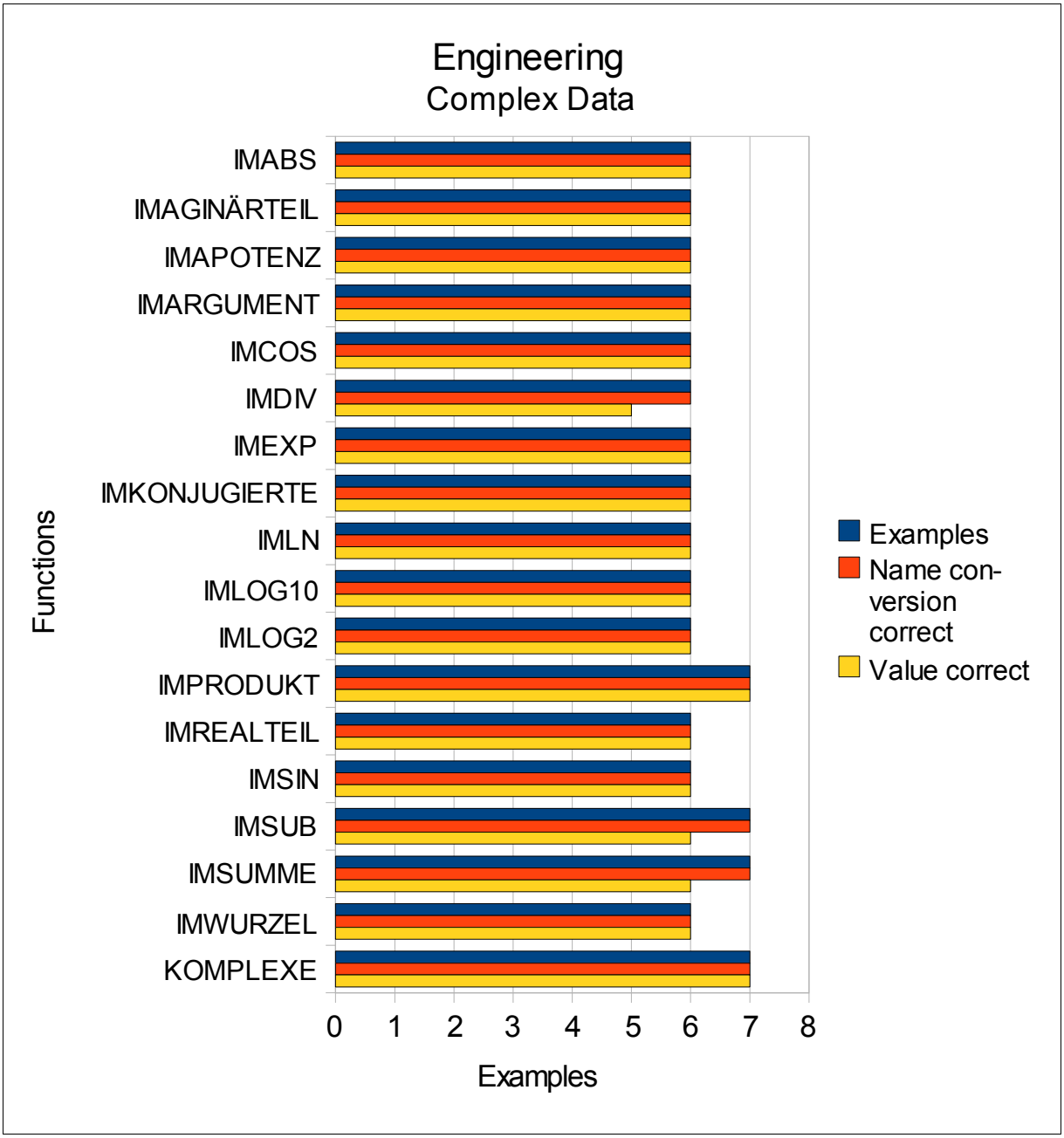


Figure 62: Engineering – Complex Data

3.1.10 Text and Data Functions

With functions for text and data, it is possible to transform text and data, to manipulate characters and to extract information concerning the text and data. As from Figure 63 could be seen, some functions were not executable in Calc. These functions are described briefly in the following. It should be mentioned that each one of these functions is

implemented in two modes in Excel. Whereas the “normal” functions are executable in Calc, the functions which work with Double-Byte-Characters are not executable.

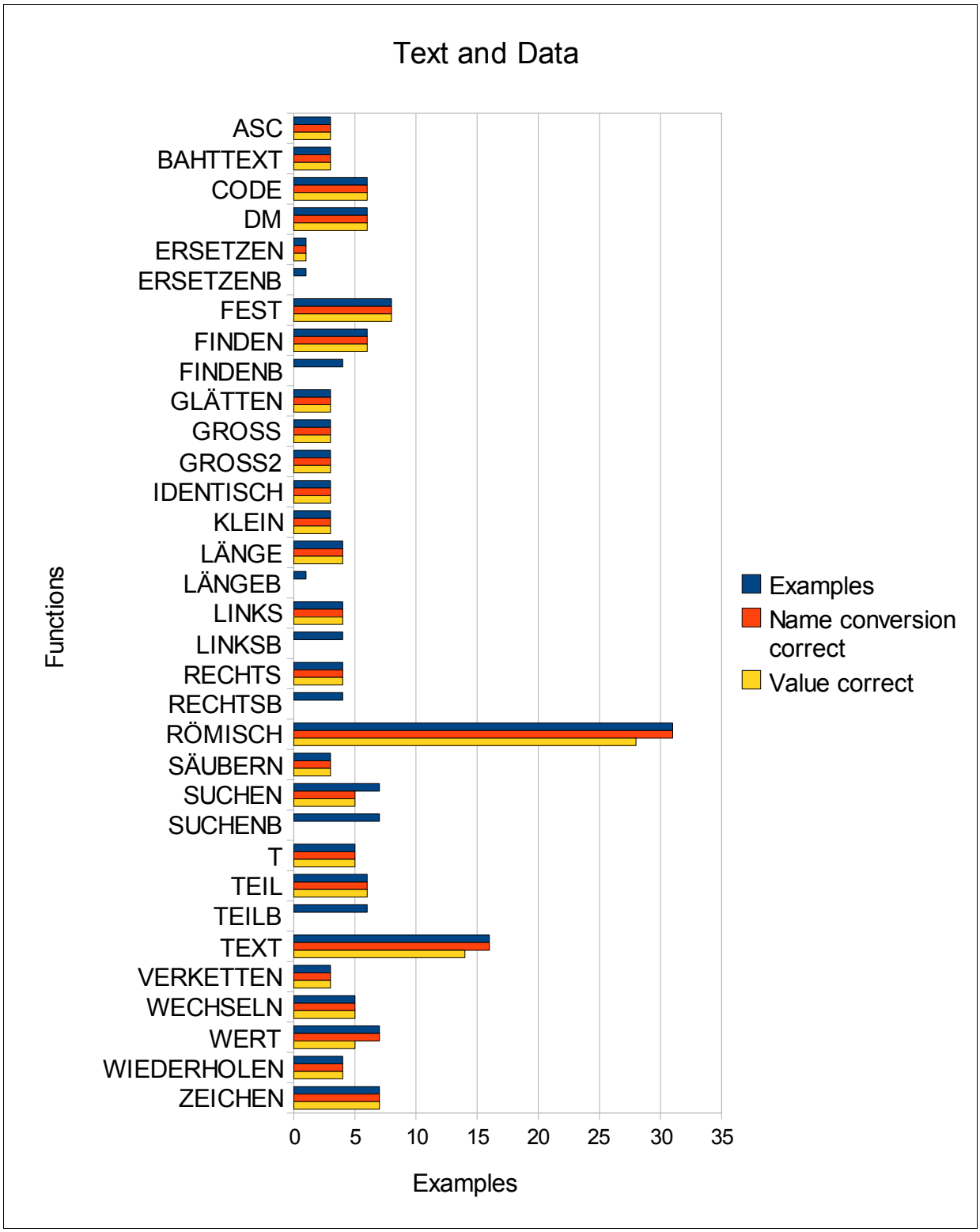


Figure 63: Text and Data Functions

ERSETZENB Replaces characters within text (for Double-Byte-Characters). FINDENB finds one text value within another (case-sensitive; for Double-Byte-Characters). LÄNGEB returns the number of characters in a text string (for Double-Byte-Characters). LINKSB returns the leftmost characters from a text value (for Double-Byte-Characters). RECHTSB returns the rightmost characters from a text value (for Double-Byte-Characters).
SUCHENB finds one text value within another (not case-sensitive; for Double-Byte-Characters).
TEILB returns a specific number of characters from a text string starting at the position you specify (for Double-Byte-Characters).

The function RÖMISCH converts an Arabic numeral to Roman, as text. It is possible to use different arguments to affect the length of the Roman text. Six of the seven possible arguments displayed the correct Roman text. When the function was executed with the argument FALSCH, Calc displayed a different but correct Roman text (see Figures 64 and 65).

217	Zahl	*Typ*			RÖMISCH
218	999	FALSCH			IM
219	1982	FALSCH			MLMXXXII
220	3999	FALSCH			MMMIM

Figure 64: Function RÖMISCH in Excel

217	Zahl	*Typ*			RÖMISCH
218	999	FALSCH			CMXCIX
219	1982	FALSCH			MCMLXXXII
220	3999	FALSCH			MMMCMXCIX
221					

Figure 65: Function RÖMISCH in Calc

With the function SUCHEN it is possible to find a text value within another (not case-sensitive). In Excel the user can decide to use the wild-cards * and ?, but in Calc functions with these arguments are not executable.

The function TEXT formats a number and converts it to text. An empty formatting argument and the argument @ were not executable in Calc.

With the function WERT a text can be converted to a number. In Excel it was not possible to convert the logic values WAHR and FALSCH to a number, although Excel saves these values as number. In Calc the output was 1 and 0.

3.1.11 Trigonometry

Functions for trigonometry can be used to make calculations with triangles. All 15 functions were executable in Calc (see Figure 66).

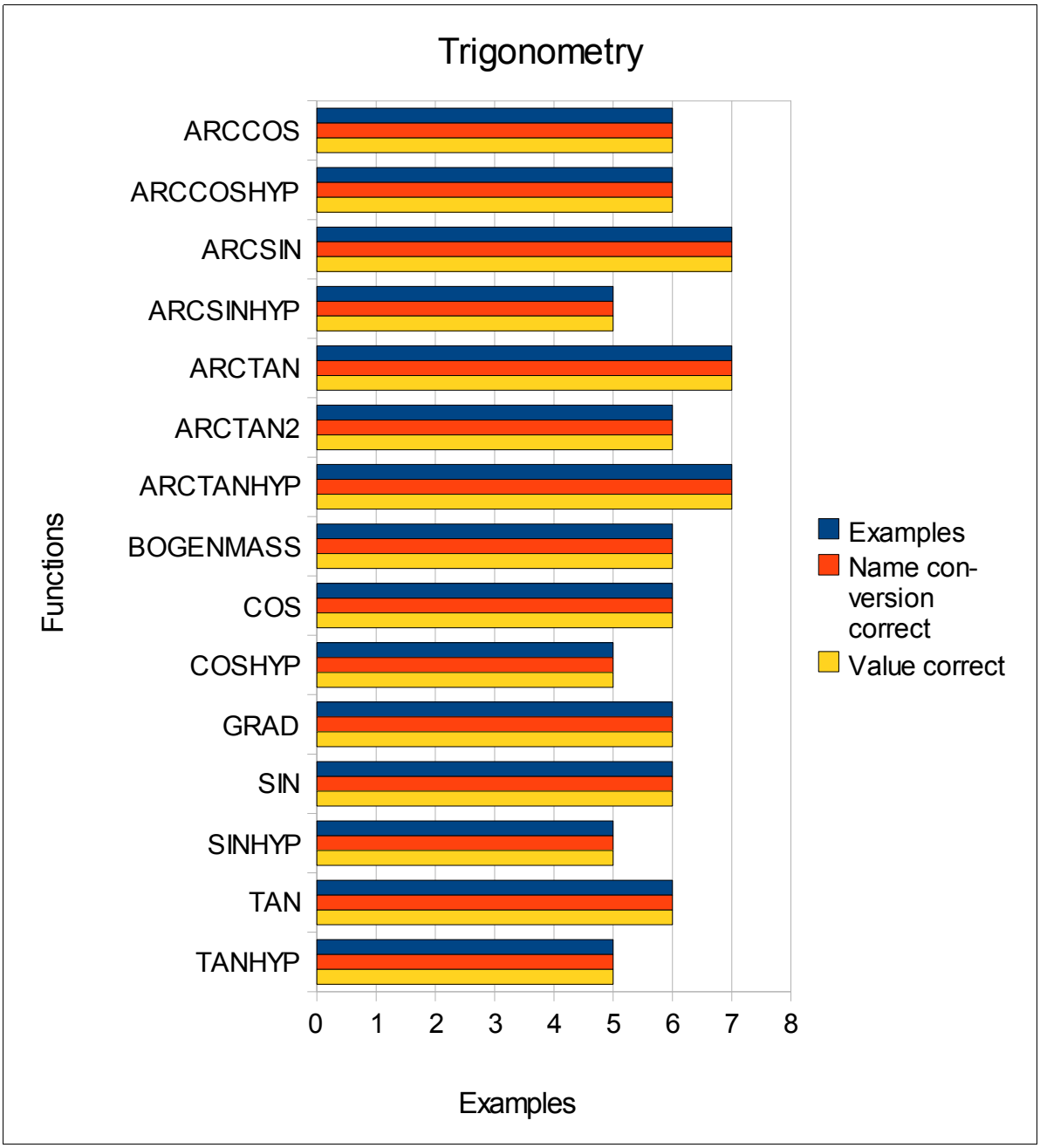


Figure 66: Functions for Trigonometry

3.2 Excel Macros in Calc

For assessing the execution of Excel macros in Calc nine workbooks were developed which handle different topics like calculating, formatting or diagrams. In Table 7 the categories, the number of contained macros and the number of executable macros are listed. These categories are described below.

Categories	Counted macros	Executable macros
Calculations	16	16
Cells and Ranges	22	20
Control Flow Statements	11	11
Diagrams	12	0
Filters	11	7
Formatting	30	23
Message Boxes	9	9
Error Handling	2	2
Printing	6	5
Worksheets	18	17

Table 7: Categories of macros

3.2.1 Calculations

The workbook *Calculations* contains macros which execute simple mathematical tasks like adding, subtracting, dividing, multiplying, potentiating and building the sum of numbers. Furthermore, rounding operations are executed. All of the 16 functions were executable in Calc correctly. As examples the macros AddReplaceA25 and RoundUpA26 are presented in the following.

Figure 67 shows the code for requesting a value from a cell, adding an amount to it and writing the new value into the cell. In the upper area the line `Option VBASupport 1` is made automatically by Calc and enables Calc to interpret the VBA-Code.

```
Rem Attribute VBA_ModuleType=VBAModule
Option VBASupport 1
Sub AddReplaceA25()

Dim number As Double

number = Range("A25").Value
number = number + 10
Range("A25").Value = number

End Sub
```

Figure 67: Macro to add values

In Figure 68 the code for the macro RoundUpA26 is shown. This macro uses the worksheet function *RoundUp* to round up the value in cell A26. The argument 1 defines to which digit the rounding should be executed. In this example the value 1,45 is rounded up to the value 1,5.

```
Sub RoundUpA26()

Range("A26").Value = Application.WorksheetFunction.RoundUp(Range("A26").Value, 1)

End Sub
```

Figure 68: Macro to round up values

3.2.2 Cells and Ranges

The workbook *Cells_Ranges* contains 22 macros which handle tasks like selecting a certain range (e. g. a row or a column), inserting and deleting rows and columns or hiding columns. Furthermore with some macros it is possible to query the range of the selection and to count cells in the selection.

Two of the 22 macros were not executable correctly and one macro delivers an error exception.

In Figure 69 two macros are shown, which should be able to select a certain range of cells. The macro RangeSelection5 was executable, it selects the range B26:E28. The cells in the sheet are selected with the property *Cells*. The first argument of this property is the row

number, the second argument is the column number. If the rows and columns are named like in the way of macro RangeSelection5, the macro works correctly in Calc.

The macro RangeSelection4 was not executable in Calc, because the columns are named with characters.

```
Sub RangeSelection4()

Range(Cells(25, "A"), Cells(29, "F")).Select

End Sub


Sub RangeSelection5()

Range(Cells(26, 2), Cells(28, 5)).Select

End Sub
```

Figure 69: Macros to select ranges

The macro in Figure 70 is able to select the 25th row and the 2nd column in a sheet by a union statement. Firstly, range objects are defined and afterwards these objects are set with the row and column. The two ranges are combined with the method *Union* and are selected with the method *Select*. Whereas this macro was executable in Excel, in Calc a runtime error occurred (see Figure 71).

```
Sub SelectRow25ColumnB()

Dim rangerow As Range
Dim rangecolumn As Range
Dim rangewhole As Range

Set rangerow = Range("25:25")
Set rangecolumn = Range("B:B")
Set rangewhole = Union(rangerow, rangecolumn)

rangewhole.Select

End Sub
```

Figure 70: Macros to combine ranges

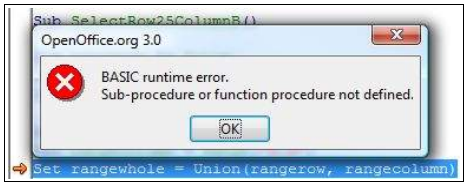


Figure 71: BASIC runtime error (ranges)

In Figure 72 two macros are shown to insert rows into a sheet. The macro Insert4ColumnsA was executable correctly in Calc. It selects the first four columns and uses the property *Insert* to insert as much columns as selected. *Shift:=xlToRight* shifts the selected columns to the right and inserts the new columns.

With the displayed macro InsertColumnA a new column should be inserted left to the selected one (column A), but this function did not work in Calc correctly.

```
Sub Insert4ColumnsA()

Columns("A:D").Insert Shift:=xlToRight

End Sub


Sub InsertColumnA()

Range("A:A").EntireColumn.Insert

End Sub
```

Figure 72: Macros to insert columns

3.2.3 Control Flow Statements

In the workbook *Control-Flow-Statements* eleven macros are stored, which handle statements like *for next*, *do until* and so on. The output of each macro was shown with dialog boxes. As all of the entire macros were executable in Calc, only one is shown in Figure 73. In the macro IfThenStatement a boolean variable is defined and set with the value TRUE. In the If-Then-Statement it is checked whether the value is true or false and a message box is displayed when the value is true.

```
Sub IfThenStatement()

Dim checkvalue As Boolean
checkvalue = True

If checkvalue = True Then
    MsgBox "IfThenStatement: Value is true"
End If

End Sub
```

Figure 73: Macro to execute a If-Then-Statement

3.2.4 Diagrams

With the macros in the workbook *Diagrams* it is possible to automate the process of developing diagrams. Furthermore it is possible to delete, hide and blend in existing diagrams. Three macros can be used to change the title, the position and the dimension.

None of the twelve macros were executable in Calc. Two macros are shown as example in Figure 74. The macro DeleteDiagramObjects selects with the property *ChartObjects* all charts in the active sheet and deletes these diagrams with the method *Delete*. The second macro uses also the property *ChartObjects*. When using a argument with this property, it is possible to select a certain diagram in the sheet. The method *Visible* is used to set the diagram visible or invisible.

```
Sub DeleteDiagramObjects()

ActiveSheet.ChartObjects.Delete

End Sub


Sub HideDiagramObject1()

ActiveSheet.ChartObjects(1).Visible = False

End Sub
```

Figure 74: Macros to delete and hide diagrams

3.2.5 Filters

The workbook *Filters* has eleven macros to handle filters in a workbook. The filters can be switched on or off, a certain filter can be set and filtered data can be extracted. Four macros were not executable in Calc and should be explained in the following.

In Figure 75 the macro is used to filter data with two arguments. In Excel only data are displayed which are lesser than 60 and from the category “A”. Calc shows all data which are lesser than 60, because Calc is not able to use both filters in the same time. Firstly, the argument “A” is used and then the argument “<60”, but without any respect to the first filtering. So only the last filter is executed.

```
Sub FilterAAndLesser60()

    Range("A25").AutoFilter Field:=1, Criteria1:="A", Operator:=xlAnd
    Range("A25").AutoFilter Field:=2, Criteria1:="<60", visibledropdown:=True

End Sub
```

Figure 75: Macro to filter data with two arguments (1)

The same problem occurred in Macro FilterBOrGreater70 (see Figure 76), in which only data should be displayed which are from the category “B” and greater than 70.

```
Sub FilterBOrGreater70()

With Range("A25")
    .AutoFilter Field:=1, Criteria1:="B", Operator:=xlOr
    .AutoFilter Field:=2, Criteria1:=">70", visibledropdown:=True
End With

End Sub
```

Figure 76: Macro to filter data with two arguments (2)

The Macro in Figure 77 extracts duplicates in a selected range. In the example below the range A25:A35 is selected and then the method *AdvancedFilter* is used. In the argument *action* the operation is decided: *xlFilterCopy* copies the data to another place whereas *xlFilterInPlace* filters the data on the origin place. The argument *copytorange* is optional

and decides where the data should be copied. With the argument *Unique*, which is optional, it can be decided whether duplicates should be eliminated or not.

```
Sub ExtractDuplicates()

Range("A25:A35").AdvancedFilter action:=xlFilterCopy, _
copytorange:=Range("A40"), Unique:=True

End Sub
```

Figure 77: Macro to extract duplicates

When this macro was being executed in Calc the following error message occured (see Figure 78).

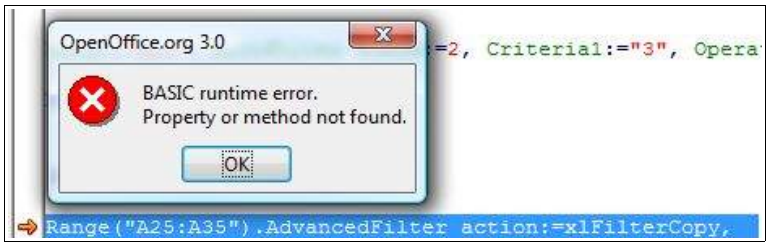


Figure 78: BASIC runtime error (ranges)

In Figure 79 the method *SpecialCells* is used to determine the visible cells (argument *xlVisible*). The method *Copy* copies this data to the place which is mentioned in the argument *Destination*.

```
Sub ExtractFilteredData()

Range("A25:B35").SpecialCells(xlVisible).Copy _
Destination:=Range("C40")

End Sub
```

Figure 79: Macro to extract filtered data

When this macro was being executed in Calc the error message in Figure 80 occured.

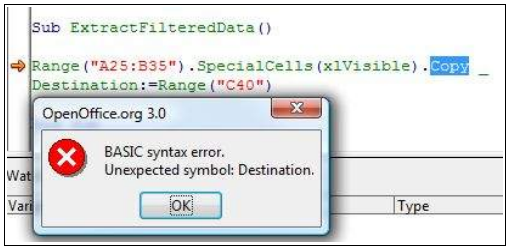


Figure 80: BASIC runtime error (filters)

3.2.6 Formatting

The macros in the workbook Formatting handle several formatting issues like querying and setting the font type, deleting content, formatting the border, deleting a format or copying data. Furthermore with some macros the height and width of rows and columns can be changed and the background can be colored. Also the text alignment is handled with some macros.

From the 30 macros in the workbook 23 were executable correctly.

In Figure 81 the macro SetDifferentFontValues1 is shown, which was not executable correctly in Calc. The error message shows, that the property *OutlineFont* is not known in Calc. If this row is deleted, the macro works well. The property *Selection* references to the selected cells in a sheet. With the property *Font* different font values can be assigned to the selection. In this example the size value is set to 14, the text is underlined and stroke through. The index for the color is set to the value 3, which stands for the color red.

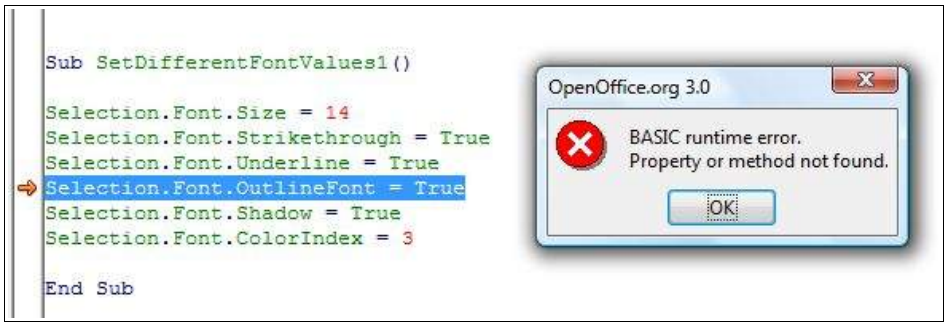


Figure 81: Code and BASIC runtime error (FontValues1)

In Figure 82 the same steps are made as in the above macro, but the way to set the font values is different. Firstly, an object *cell* from type *Range* is defined and for each cell in the selection the values are set. With the command *width* it is possible to set several values for an object without naming the object in every case. When executing this macro the same error occurred as in the macro SetDifferentFontValues1.

```
Sub SetDifferentFontValues2()

Dim cell As Range

For Each cell In Selection
    With Selection.Font
        .Size = 14
        .Strikethrough = True
        .Underline = True
        .OutlineFont = True
        .Shadow = True
        .ColorIndex = 3
    End With
Next cell

End Sub
```

Figure 82: Code and BASIC runtime error (FontValues2)

Figure 83 shows the macro to format the border for a cell with the color red. The property *LineStyle* of the object *Border* can have different values, in this example a dashed border is set. When this macro was executed neither the border was set correctly nor a error message occured.

```
Sub FormatBorder()

Selection.Borders.ColorIndex = 3
Selection.Borders.LineStyle = xlDash

End Sub
```

Figure 83: Macro to format a border

The first macro in Figure 84 sets the number format in cell *A25* to the value *#,##0*. To execute this task the property *NumberFormat* from the object *Range* is used. When this macro was being executed in Calc nothing happened.

```
Sub FormatCellA25()

Range("A25").NumberFormat = "#,##0"

End Sub


Sub FormatCellA27()

Range("A27").NumberFormat = "DD.MM.YYYY"

End Sub
```

Figure 84: Macros to set the number format

The second macro in Figure 84 sets the date format to the value *"DD.MM.YYYY"*. For the value 3000 the date should be displayed in the form *18.03.1908* but in Calc *DD.03.YYYY* was displayed.

The macro in Figure 85 uses the method *Find* of the object *Range* to search for the string *"def"*. The cell in which this string is found firstly is selected with the method *Activate*. As from the Figure could be seen, Calc was not able to execute the method *Find*.

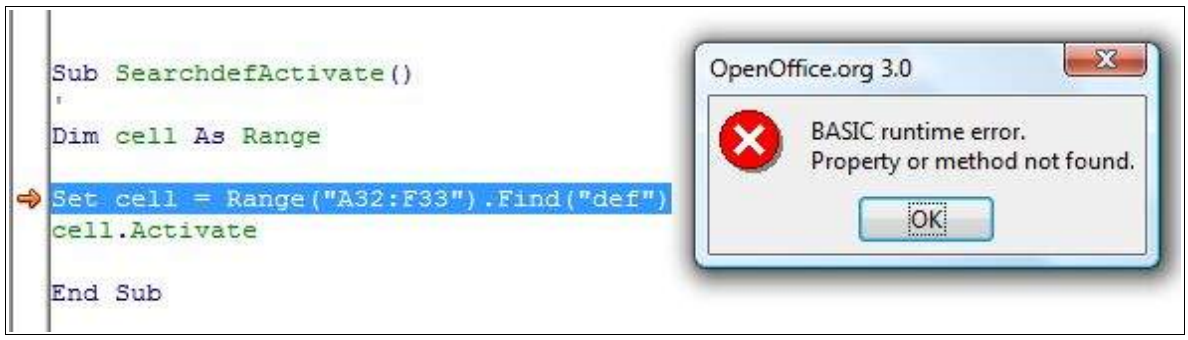


Figure 85: Macro to find a value and error message

Figure 86 shows a macro which repeats the value in the cell to fulfill the whole width of the cell. When this macro was being executed in Calc no fulfillment took place and no error message occured.

```
Sub HorizontalAlignmentFill()

Selection.HorizontalAlignment = xlFill

End Sub
```

Figure 86: Macro to fill a cell with the same values

3.2.7 Message Boxes and Error Handling

In the workbook *MessageBoxes_ErrorHandling* nine macros are included which show the user different message boxes (e. g. two or three buttons or a specific title). As all message boxes worked well in Calc only the VBA-Code for one is explained briefly.

The macro in Figure 87 puts out a message box with the text *"Hello World..."* and has three buttons. Furthermore the first button is set as default button. In Figure 88 the corresponding message box is shown.

```
Sub Message5()

MsgBox "Hello World (Abort/Retry/Ignore + DefaultButton1)", _
vbAbortRetryIgnore + vbDefaultButton1

End Sub
```

Figure 87: Macro for a message box

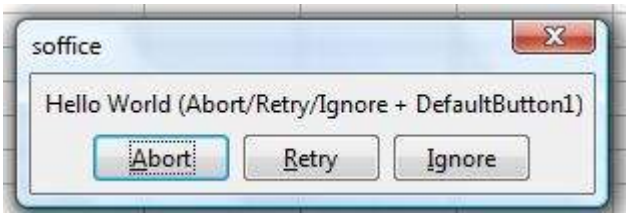


Figure 88: Message Box with three buttons

The two macros which should handle an error and make sure that no runtime error occurs did not work in Calc. As in Figure 89 could be seen both macros cause a runtime error in the line *ActiveSheet.Previous.Activate*, because the method *Activate* tries to activate the worksheet which is before the first one.

The macro ErrorHandling1 uses the command *On Error Resume Next* which activates an error handling routine. This command makes it possible, that the next command after the one where the error occurred, is being executed. The macro ErrorHandling2 uses the command *On Error GoTo,* which jumps to the line in the code which is quoted after the command, in the shown example the line *errhandling*.

```
Sub ErrorHandling1()

On Error Resume Next
ActiveSheet.Previous.Activate

End Sub



Sub ErrorHandling2()

On Error GoTo errhandling
ActiveSheet.Previous.Activate

errhandling:
    MsgBox "There was a problem"

End Sub
```

Figure 89: Macros for error handling

3.2.8 Printing

The workbook Printing contains six macros which execute tasks like setting and canceling the print area and printing sheets or only selections.

As only the macro PrintSelection worked correctly in Calc, all six macros are displayed in the Figure 90 and described in the following.

With the macro SetPrintArea an area in an worksheet can be selected and defined as print area. If the document is printed, only the selected area will be printed. With the macro CancelPrintArea the print area is deleted. For both macros the property *PrintArea* of the object is used.

The macros PrintSheet1 and PrintSheet1Copies3 are used to print out a special worksheet in the workbook. When the argument *copies* is used, the number of copies can be defined.

The macro PrintWithDialog uses the standard windows dialog for printing something. To achieve this dialog the property *Dialogs* form the object *Application* is used.

```
Rem Attribute VBA_ModuleType=VBAModule
Option VBASupport 1
Sub SetPrintArea()
'
ActiveSheet.PageSetup.PrintArea = Selection.Address

End Sub


Sub CancelPrintArea()
'
ActiveSheet.PageSetup.PrintArea = False

End Sub


Sub PrintSheet1()

Sheets(1).PrintOut

End Sub

Sub PrintSheet1Copies3()

Sheets(1).PrintOut copies:=3

End Sub

Sub PrintSelection()

Selection.PrintOut copies:=1, Collate:=True

End Sub

Sub PrintWithDialog()

Application.Dialogs(xlDialogPrint).Show

End Sub
```

Figure 90: Macros for printing

3.2.9 Worksheets

In the workbook *Worksheets* are macros which handle tasks like deleting, naming, hiding, and blending in sheets. It is also possible to protect sheets in the way that a user is not able to edit the content. With some macros it is also possible to copy sheets or the content of a sheet to another sheet or place.

From the eighteen macros one was not executable and three worked in a different way. The macro WSAdd in Figure 91 should add a new worksheet into the workbook. The difference between Calc and Excel is that Excel activated the new worksheet after adding. The macro WSAddLast also uses the method *Add* to add a new worksheet, but with an argument. With the argument *After* it is possible to decide after which worksheet the new one should be added. To assure that the worksheet is added after the last one, the worksheets are counted with the property *Count*.

```
Sub WSAdd()

Worksheets.Add

End Sub

Sub WSAddLast()

Worksheets.Add After:=Worksheets(Worksheets.Count)

End Sub
```

Figure 91: Macros for adding worksheets

In Figure 92 a macro is shown which deletes the third worksheet. The difference between Excel and Calc is that in Excel a dialog window was displayed which allows the user to discard the deletion. In Calc the worksheet was deleted without any dialog window.

```
Sub DeleteWS3wAlert()

Sheets(3).Delete

End Sub
```

Figure 92: Macros for deleting a worksheet

In Figure 93 the macro SelectWS23 is displayed, which is used to select the second and third worksheet. This macro neither worked nor displayed an error message.

```
Sub SelectWS23()

Sheets(Array(2, 3)).Select

End Sub
```

Figure 93: Macros for selecting two worksheets

4 Conclusions

This paper tried to assess the executing of Excel macros and functions in Calc.

In chapter 2 basics about functions and macros were given. These basics included the description of differences in terminology, e. g. the entire file is named spreadsheet in Calc and workbook in Excel. It was shown that there are not many differences in working with functions between Excel and Calc. The second sub-chapter showed that in Excel Visual Basic for Applications and in OpenOffice.org OOo Basic is used to implement macros. For each program the process of recording macros and the execution of the recorded macros were presented. The third sub-chapter dealt with the interoperability between Excel and Calc. It was mentioned that in the past the user had at least the possibility to keep the attached macros of Excel files, so that the macros were still available for use in Excel. Apart from this, the vendor specific solutions from Novell and Sun, which provided limited support of Excel, were described briefly. It was shown that these two vendors are working now together on a common project. In the sub-chapter *Possible Lock-in of Calc* the possible lock-in-effects which could occur when using Calc were discussed.

In chapter 3 the test cases and findings of functions and macros were presented. Eleven uniform categories and several sub-categories were build to test 336 functions. The findings show that in large part the functions were executable in Calc. In some test cases Calc was able to execute the functions but the return values were different in some digits right to the decimal point.

To assess the interoperability concerning macros ten uniform categories were build which handle different things like formatting, calculating and so on. The findings show that from the 137 macros 110 were executable correctly in Calc. The biggest problems occurred in macros which tried to work with diagrams. None of the twelve macros was executable in Calc.

Over all it can be said that Calc is already a very powerful program which can be used as alternative to Excel. Many Excel functions and macros are executable in Calc without any further effort. However, as not all Excel functions and macros are executable correctly in Calc it must be decided individually whether a change from Excel to Calc should be made or not.

References

[BASIPG] OpenOffice.org BASIC Programming Guide. http://wiki.services.openoffice.org/wiki/Documentation/BASIC_Guide, Retrieved on December 2008.

[EinfMa] Einführung in Makros: Ein Leitfaden. http://office.microsoft.com/de-de/help/HA012317941031.aspx?pid=CH010963501031, Retrieved on December 2008.

[GetSt08] Getting Started with OpenOffice.org 3. Published 2008-10-26. http://documentation.openoffice.org/manuals/userguide3/, Retrieved on December 2008.

[JesE06] Jeschke, Egbert et. Al: Microsoft Excel – Funktionsverzeichnis. Referenz aller Funktionen in Excel 2000 bis 2003. Microsoft Press Deutschland, Unterschleißheim 2006.

[Linfo] The Linux Information Project - Vendor Lock-in Definition. http://www.linfo.org/vendor_lockin.html, Retrieved on December 2008.

[ListWF] List of worksheet functions (by category). Microsoft Office Online. http://office.microsoft.com/en-us/excel/HP052042111033.aspx, Retrieved on December 2008.

[MigG06] Migration Guide – Openoffice.org. Published 2006-05-10. http://documentation.openoffice.org/manuals/oooauthors2/0600MG-MigrationGuide.pdf, Retrieved on October 2008.

[PitoA04] Pitonyak, Andrew: OpenOffice.org Macros Explained. Hentzenwerke Publishing, Whitefish Bay 2004.

[Porti04] Porting Excel/VBA to Calc/StarBasic. Published 2004-06-05. http://documentation.openoffice.org/HOW_TO/various_topics/VbaStarBasicXref.pdf, Retrieved on November 2008.

[SchJ07] Schmidt, Juergen: Sun and Novell work together on a common OpenOffice.org VBA story. Blog at GullFOSS. Online since 2007-02-09. http://blogs.sun.com/GullFOSS/entry/sun_and_novell_work_together, Retrieved on October 2008.

[Sola07] Sola, Rajesh: A Tutorial On VBA Macros – Interoperability. OpenOffice.org Project Day@foss.in. Presented 2007-12-05. http://foss.in/2007/register/slides/ VBA_Macros_-__Interoperability_380.pdf, Retrieved on November 2008.

[UseG07] OpenOffice.org User Guide for Version 2.x. Published 2007-04-09. http://documentation.openoffice.org/manuals/index.html, Retrieved on October 2008.

[VBAFAQ] Visual Basic for Applications Frequently Asked Questions. http://msdn.microsoft.com/de-de/isv/bb190540(en-us).aspx, Retrieved on December 2008.

[WhyOO] Why OpenOffice.org. http://why.openoffice.org/, Retrieved on December 2008.

Table of Figures

Index of Tables