Bringing interactivity into engineering courses with BERT-based Excel®-R applications

Florent Bourgeois, Prof.
Laboratoire de Génie Chimique, Université de Toulouse, CNRS
Toulouse, France

PRESENTATION OUTLINE

- Introduction
- Principles and implementation
- Examples in engineering education
- Conclusions

Introduction

- Engineers love Excel®!, perhaps the most (world)widely used "Engineer-Machine" interface for data processing.
- VBA (not a full OO programming language, as no inheritance nor function overloading, but includes classes and interfaces), interactive userform design, most engineers have some knowledge of VBA.
- Lacks power for data analysis, modeling and visualisation (not Excel®'s primary function).

BERT

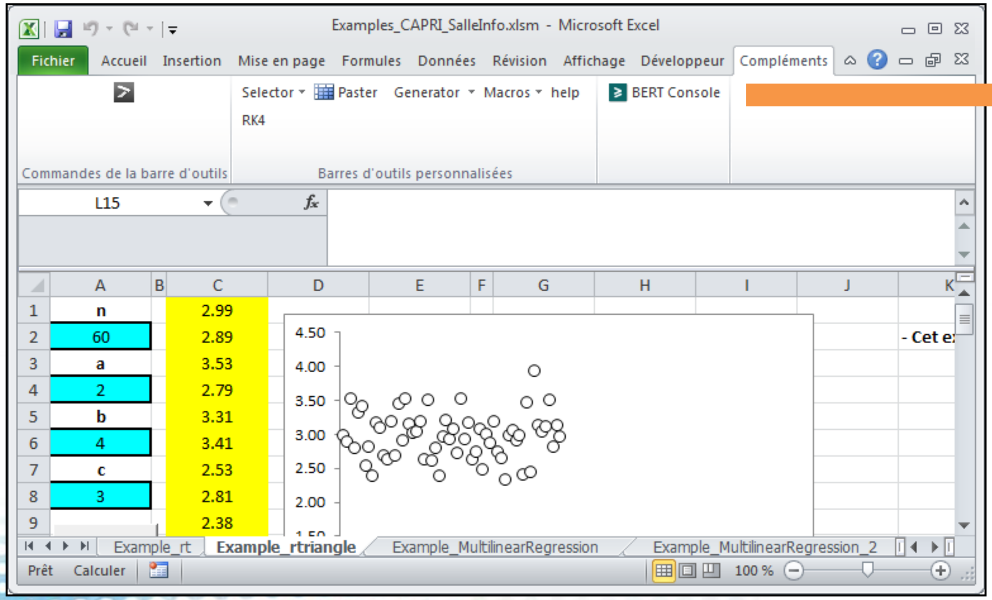
Basic Excel R Toolkit

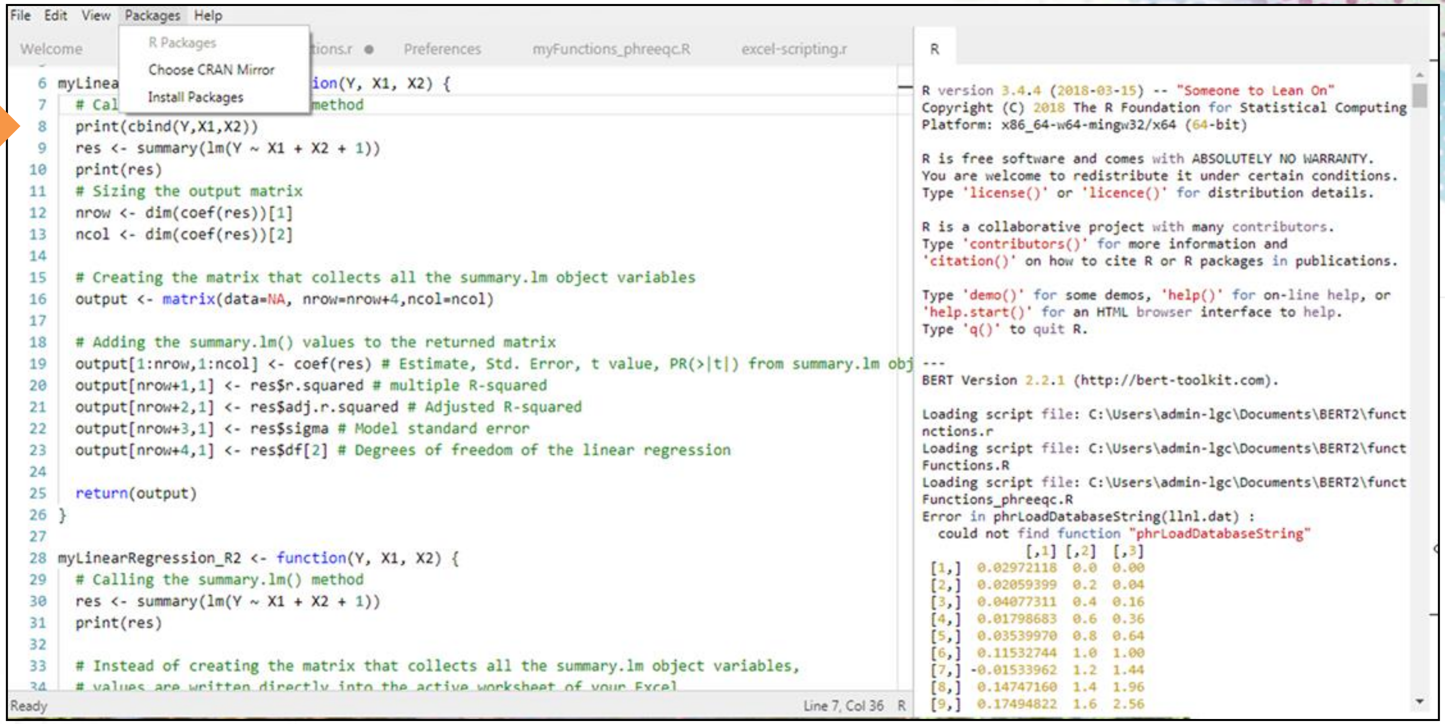
- BERT: Basic Excel R Toolkit (R console for Excel®)
- Free (from Structured Data, LLC)
- BERT version 2 (bert-toolkit.com)

- Virtually limitless capability for data analysis, modeling and visualisation (R's primary function), not limited to statistical data analysis.
- Free
- Still limited use and visibility amongst mainstream engineering community

- Installation from https://bert-toolkit.com/download-bert
- Excel® add-in ➔ BERT console (R editor)

BERT console (.R code)





- BERT's default startup folder = ~\Documents\BERT2\functions\
- Recommendation:
  - Place your .R files in BERT's default startup folder (unless change in "Preferences" file)
  - In BERT consolde: "File > New File" to write your R code (this creates a new tag in the BERT console)
  - Packages necessary for the R code are installed using the "Packages" menu in the BERT console

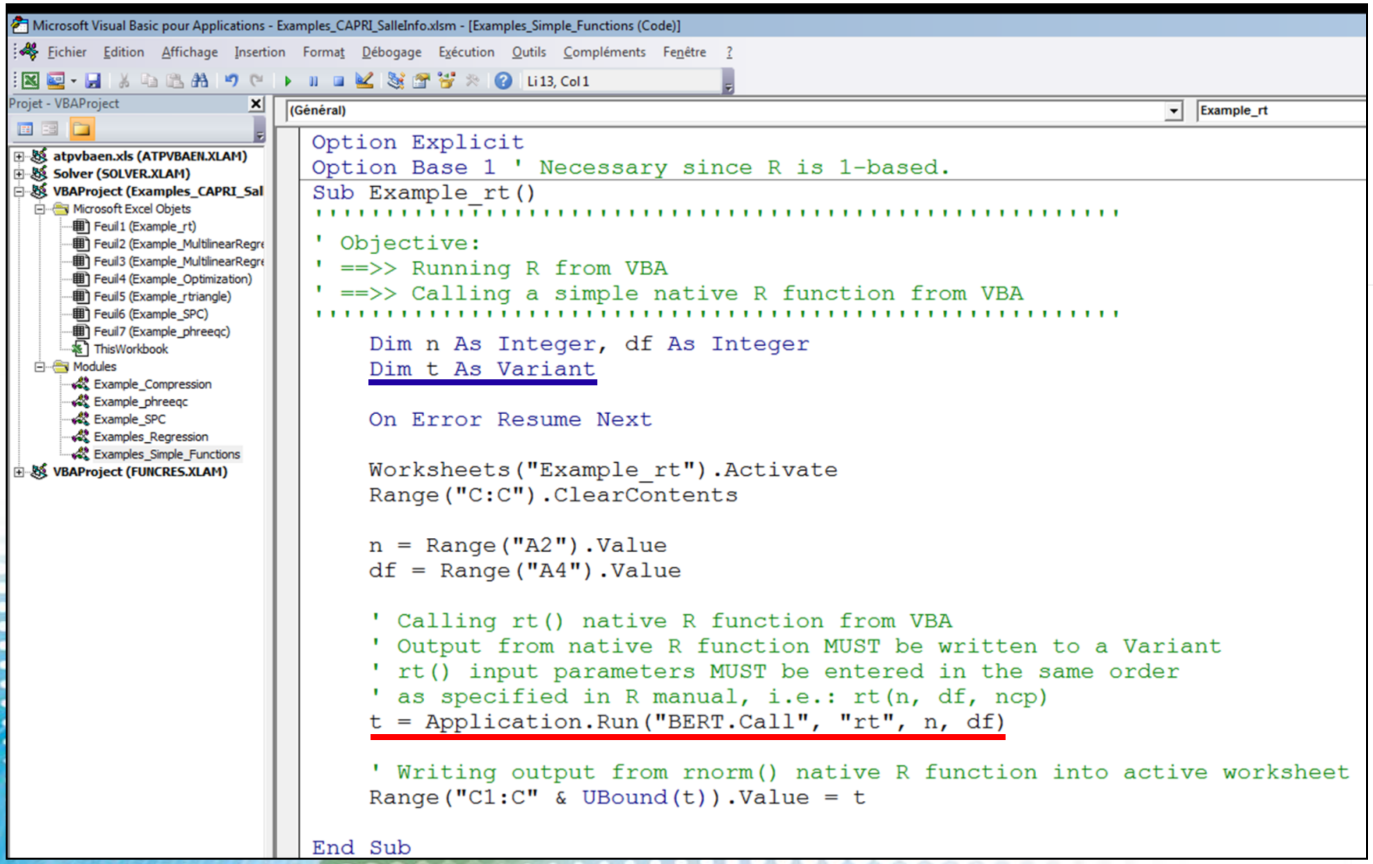
- **2 types of applications**
  - **Type 1**: Calling native and package-imported R functions from Excel® (VBA code, no R code, no use of the BERT console besides installation of R packages if needed)
  - **Type 2**: Calling user-defined R functions from Excel® (VBA code, R code, use of the BERT console)

- **Recommended VBA code structure for Type 1 applications**
  - **Section 1:** VBA reads the data from the Excel® spreadsheet as type Variant variables.
  - **Section 2:** VBA calls and pass Variant variables to (native, package-imported, own) R functions, using VBA Application.Run() call function.
  - **Section 3:** VBA and/or R writes the output from R functions into the Excel® spreadsheet.

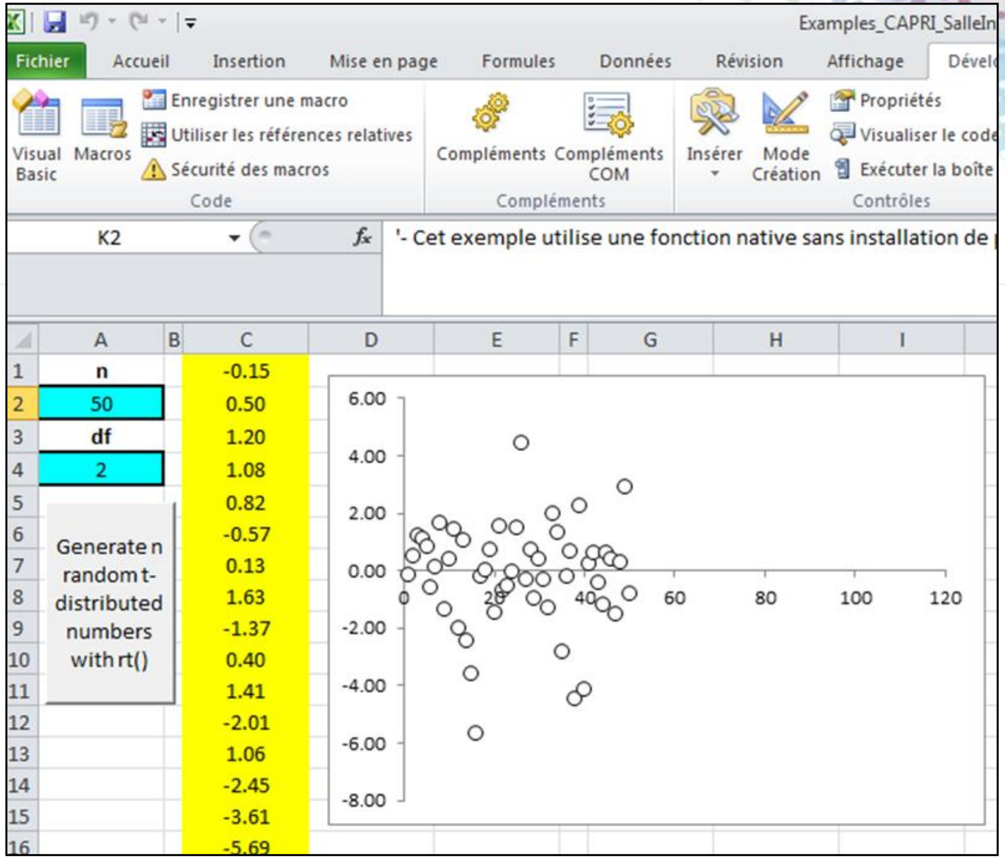
- **Note**
  - When running the Excel® file with built-in R code, there is no need to open the BERT console. The R code is therefore invisible to the end-user.

Type 1: Calling native and package-imported R functions from Excel® (VBA code, no R code, no use of the BERT console besides installation of R packages)

VBA editor



Excel® spreadsheet



Type 1: Calling native and package-imported R functions from Excel® (VBA code, no R code, no use of the BERT console besides installation of R packages)

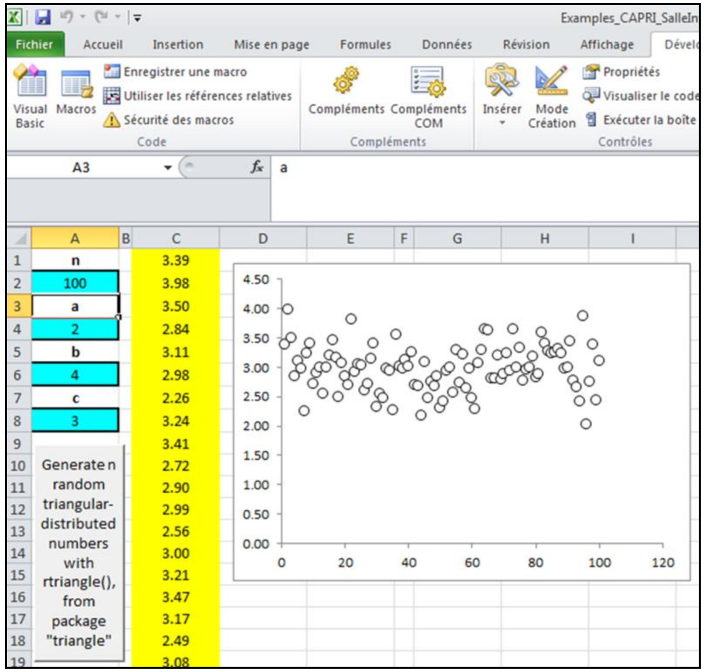
VBA editor

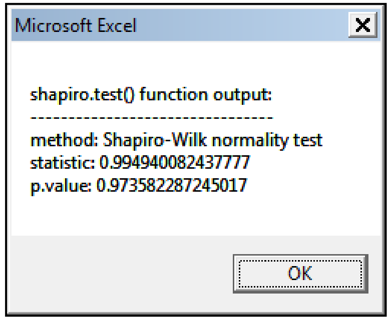
```
Sub Example_rtriangle()
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Objective:
' ==>> Running R from VBA
' ==>> Calling a simple R function from a specific R package.
'
' BEFORE RUNNING THIS CODE: R packages used by the VBA code must be installed prior to running
' In this example, install the "triangle" package from the BERT console:
' Packages > install Packages > ... Install
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    Dim n As Integer, a As Double, b As Double, c As Double

    ' Variables used to call R functions from VBA
    Dim lib As Variant ' Variable used to load a library when running the macro
    Dim var1, var2, var3, var4 'Variant variables used as output to calls to Application.Run()

    On Error Resume Next

    ' Loading an R package from VBA
    ''''''''''''''''''''''''''''''''''''
    lib = Application.Run("BERT.Call", "library", "triangle")

    Worksheets("Example_rtriangle").Activate
    Range("C:C").ClearContents

    n = Range("A2").Value
    a = Range("A4").Value
    b = Range("A6").Value
    c = Range("A8").Value

    ' Calling rtriangle() function from R package "triangle" from VBA
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Output from native R function MUST be written to a Variant
    ' rtriangle() input parameters MUST be entered in the same order as specified in R manual,
    ' rtriangle(n, a=0, b=1, c=(a+b)/2)
    var1 = Application.Run("BERT.Call", "rtriangle", n, a, b, c)

    ' Writing output from rnorm() native R function into active worksheet
    Range("C1:C" & UBound(var1)).Value = var1
    Application.ScreenUpdating = True 'Useful so the Excel graphs are updated without delay
```

Requires installing the "triangle" package from the BERT console

```
    ' Calling shapiro.test() function and printing the function outputs
    ' The shapiro.test function outputs an instance of the "htest" class
    ' whose components include: statistic, p.value, method.
    ' Use the VBA debugger to see what's in var3.
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    var3 = Application.Run("BERT.Call", "shapiro.test", var1)
    Call MsgBox("shapiro.test() function output:" & vbCrLf & _
            "--------------------------------" & vbCrLf & _
            "method: " & var3(3, 1) & vbCrLf & _
            "statistic: " & var3(1, 1) & vbCrLf & _
            "p.value: " & var3(2, 1))

End Sub
```

Excel® spreadsheet





Type 2: Calling user-defined R functions from Excel® (VBA code, R code, use of the BERT console)

- **Purpose**
  - Ideal when needing to develop an "engineer friendly'" interface for a running R code.
  - Ideal for educators seeking to embed more advanced R functions, related to their engineering courses, into Excel®. This brings interactivity (and fun) into the teaching, which helps with student learning.

- **Principles**
  - Keep all calculations inside your R functions, limiting VBA to pass data back and forth between the Excel® spreadsheet and the R functions.

- **Recommended VBA code structure for Type 2 applications**
  - **Section 1:** VBA reads the data from the Excel® spreadsheet and stores them as type Variant variables.
  - **Section 2:** VBA calls and pass Variant variables to user-defined R functions, whose code (.R files) is in the ~\Documents\BERT2\functions\ folder.
  - **Section 3:** VBA and/or R writes the output from the R functions into the Excel® spreadsheet.

Type 2: Calling user-defined R functions from Excel® (VBA code, R code, use of the BERT console)

VBA editor

```
Sub myLinearRegression_VBA()
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Objective:
' ==>> Running R from VBA
' ==>> Calling your own R functions from VBA.
    Dim Ndata As Integer, i As Integer
    Dim X1 As Variant, X2 As Variant, Y As Variant
    Dim var1 As Variant 'Variant variables used as output to calls to Application.Run()

    On Error Resume Next

    Worksheets("Example_MultilinearRegression").Activate
    Ndata = Application.WorksheetFunction.CountA(Range("B:B")) - 1

    ' Reading the values in the Excel spreadsheet
    X1 = Range(Cells(2, 2), Cells(Ndata + 1, 2)).Value
    X2 = Range(Cells(2, 3), Cells(Ndata + 1, 3)).Value
    Y = Range(Cells(2, 4), Cells(Ndata + 1, 4)).Value

    ' Calling myLinearRegression() function from R-Script from VBA
    ' to estimate the linear model parameter: Y = b0 + b1*X1 + b2*X2
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    var1 = Application.Run("BERT.Call", "myLinearRegression_R", Y, X1, X2)
    Range(Cells(2, 7), Cells(2 + UBound(var1, 1) - 1, 7 + UBound(var1, 2) - 1)).Value = var1
End Sub
```

BERT console (.R code)

bert2-console

File Edit View Packages Help

Welcome | myFunctions.R × | functions.r ● | Preferences | myFunctions_phre

```

myLinearRegression_R <- function(Y, X1, X2) {
  # Calling the summary.lm() method
  print(cbind(Y,X1,X2))
  res <- summary(lm(Y ~ X1 + X2 + 1))
  print(res)
  # Sizing the output matrix
  nrow <- dim(coef(res))[1]
  ncol <- dim(coef(res))[2]

  # Creating the matrix that collects all the summary.lm object varia
  output <- matrix(data=NA, nrow=nrow+4,ncol=ncol)

  # Adding the summary.lm() values to the returned matrix
  output[1:nrow,1:ncol] <- coef(res) # Estimate, Std. Error, t value,
  output[nrow+1,1] <- res$r.squared # multiple R-squared
  output[nrow+2,1] <- res$adj.r.squared # Adjusted R-squared
  output[nrow+3,1] <- res$sigma # Model standard error
  output[nrow+4,1] <- res$df[2] # Degrees of freedom of the linear re

  return(output)
}
```

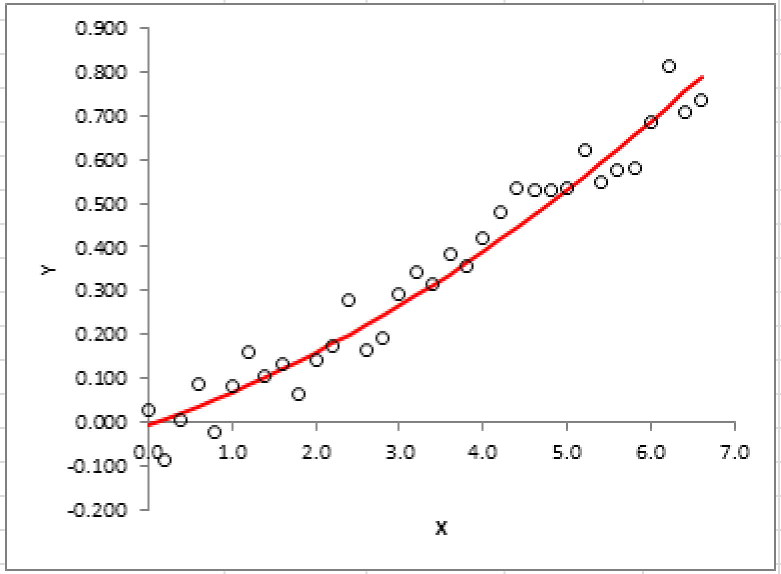
Type 2: Calling user-defined R functions from Excel® (VBA code, R code, use of the BERT console)

Click and Run your VBA code

	X	X^2	Y	Ymod
2	0.0	0.0	0.028	-0.006
3	0.2	0.0	-0.085	0.007
4	0.4	0.2	0.005	0.022
5	0.6	0.4	0.087	0.036
6	0.8	0.6	-0.024	0.052
7	1.0	1.0	0.082	0.068
8	1.2	1.4	0.161	0.085
9	1.4	2.0	0.105	0.102
10	1.6	2.6	0.131	0.120
11	1.8	3.2	0.065	0.139
12	2.0	4.0	0.143	0.158
13	2.2	4.8	0.173	0.178
14	2.4	5.8	0.278	0.199
15	2.6	6.8	0.163	0.221
16	2.8	7.8	0.192	0.243
17	3.0	9.0	0.292	0.266
18	3.2	10.2	0.341	0.289
19	3.4	11.6	0.316	0.313
20	3.6	13.0	0.386	0.338
21	3.8	14.4	0.357	0.363
22	4.0	16.0	0.423	0.390
23	4.2	17.6	0.480	0.416
24	4.4	19.4	0.537	0.444
25	4.6	21.2	0.532	0.472
26	4.8	23.0	0.529	0.501

Multilinear model: Y = b0+b1*X+b2*X^2		Estimate	Std. Error	t value
Intercept b0		-0.00592123	0.02610861	-0.226792
b1		0.06540543	0.01830726	3.572649
b2		0.00837007	0.00268090	3.122106
Multiple R-squared	0.9557			
Adjusted R-squared	0.9529			
Model Std. Error	0.0538			
df	31			

Output values from lm() are written into the active Excel worksheet directly from your own R function.



Output is formatted to resemble that of Excel's REGLIN() worksheet function

BERT console (.R code)

BERT Console

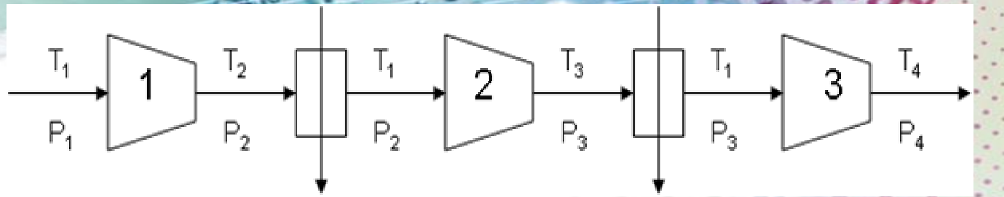
File Edit View Packages Help

Welcome | myFunctions.R | functions.r | Preferences | myFunctions_phreeqc.R

```r
myLinearRegression_R2 <- function(Y, X1, X2) {
  # Calling the summary.lm() method
  res <- summary(lm(Y ~ X1 + X2 + 1))
  print(res)

  # Instead of creating the matrix that collects all the summary.lm object v
  # values are written directly into the active worksheet of your Excel.
  range <- EXCEL$Application$get_Range("G1:J4")
  range$put_Value(coef(res)) # Estimate, Std. Error, t value, PR(>|t|) from
  range <- EXCEL$Application$get_Range("G5")
  range$put_Value(res$r.squared) # multiple R-squared
  range <- EXCEL$Application$get_Range("G6")
  range$put_Value(res$adj.r.squared) # Adjusted R-squared
  range <- EXCEL$Application$get_Range("G7")
  range$put_Value(res$sigma) # Model standard error
  range <- EXCEL$Application$get_Range("G8")
  range$put_Value(res$df[2]) # Degrees of freedom of the linear regression

  #return(res)
}
```

Rather than formatting R's output in the Excel spreadsheet using VBA, you can address an Excel® spreadsheet directly from one's R code using BERT's Excel® Scripting (COM) interface (e.g. for reading from and writing to specific cells in the Excel® spreadsheet).

Numerical optimization and chemical engineering example



- **The problem:**
  - 3-stage compression problem for ideal gas and adiabatic conditions.
  - Starting with inlet temperature T_1 and pressure P_1, and seeking an outlet pressure P_4, the problem consists in finding the inlet and outlet pressures P_2 and P_3 of the intermediate compression stage that yield the minimum compression work W. With $P_1<P_2<P_3<P_4$, k=1.4 and R the ideal gas constant, the compression work W can be expressed as

$$W = \frac{kRT_1}{k-1}\left[\left(\frac{P_2}{P_1}\right)^{(k-1)/k} + \left(\frac{P_3}{P_2}\right)^{(k-1)/k} + \left(\frac{P_4}{P_3}\right)^{(k-1)/k} - 1\right]$$

  - Students are invited to try the Nelder-Meade algorithm to solve the problem.

- **Step 1 :**
  - Search the CRAN archive for the R package you need. A quick web search indicates that Nelder-Mead algorithm is available in several R packages, e.g. "*optimization*" (https://cran.r-project.org/web/packages/optimization/index.html, Husmann et al.).
  - The "Package > Install Packages" menu from the BERT console is used to install the "*optimization*" package.
  - Write your (user-defined) R function, adding it to the ~\BERT2\functions\ folder.

- **Step 2 :**
  - Use of a Type 2 application, i.e. the whole numerical optimization solution to the problem is coded in a user-defined R function, and VBA is simply used to pass the data between the Excel® spreadsheet and the R code.
  - The VBA code makes a single call to the R function *Compression_R()*. The function loads the "*optimization*" package and uses the *optim_nm()* function, which implements the Nelder-Meade algorithm.

VBA editor

```
Sub Optimization_VBA()

' Section 0: variables déclaration
'-------------------------------
    ' All variables passed to your R function should be defined as Variant
    Dim k As Variant, R As Variant, T1 As Variant, P1 As Variant, P4 As Variant, P2init As Variant,
    'Variant variables used as output to Excel API function Application.Run()
    Dim var1 As Variant

    On Error Resume Next

    ' Clear the output range
    Range("B8").ClearContents

' SECTION 1: importing data from the Excel spreadsheet
'-----------------------------------------------------------
    Worksheets("Example_Optimization").Select
    k = Range("B1").Value
    R = Range("B2").Value
    T1 = Range("B3").Value
    P1 = Range("B4").Value
    P4 = Range("B5").Value
    P2init = Range("B6").Value
    P3init = Range("B7").Value

' SECTION 2 : Calling your own R function
' This is where all the calculations are done, VBA does none !
'-------------------------------------------------------------
    var1 = Application.Run("BERT.Call", "Compression_R", k, R, T1, P1, P4, P2init, P3init)

' SECTION 3: Writing output from your own R function back into the Excel spreadsheet
'-----------------------------------------------------------------------------------
' In this example, outputs from your R function are writtent to the active
' Excel spreadsheet directly from the R function utself.

End Sub
```

	A	B	C	D
1	k	1.4		Optimization example
2	R	8.3144598	J·mol−1·K−1	
3	T1	273	K	
4	P1	200	kPa	
5	P4	1000	kPa	
6	P2	342	kPa	
7	P3	585	kPa	
8	W	19837	J	

BERT console (.R code)

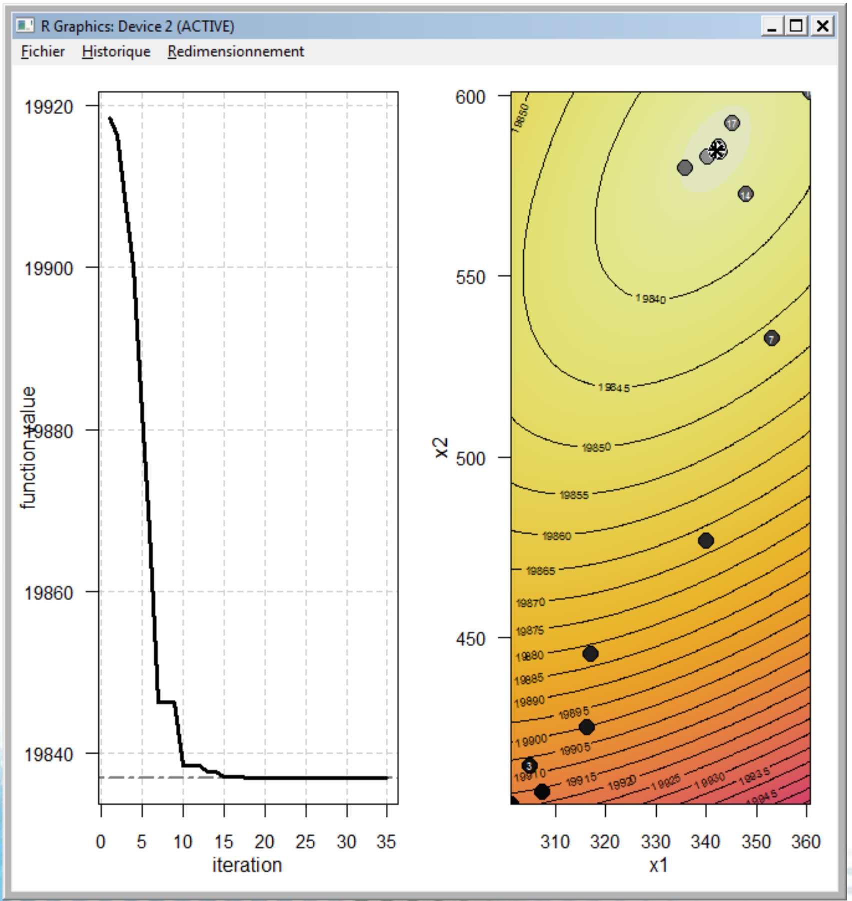
BERT Console

File Edit View Packages Help

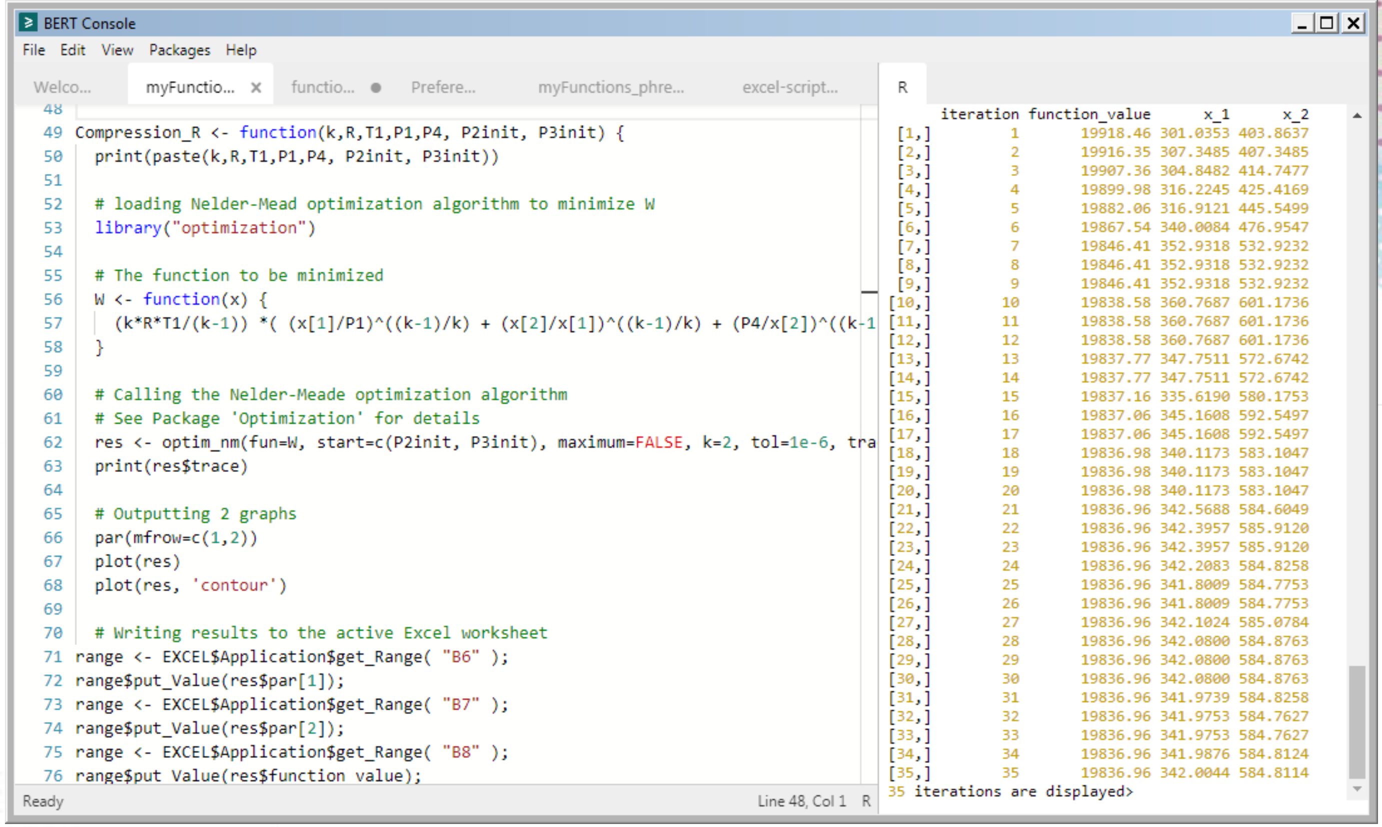
Welcome | myFunctions.R | functions.r | Preferences | myFunctions_phreeqc.R | excel-scripting.r

```r
Compression_R <- function(k,R,T1,P1,P4, P2init, P3init) {
  print(paste(k,R,T1,P1,P4, P2init, P3init))

  # loading Nelder-Mead optimization algorithm to minimize W
  library("optimization")

  # The function to be minimized
  W <- function(x) {
    (k*R*T1/(k-1)) *( (x[1]/P1)^((k-1)/k) + (x[2]/x[1])^((k-1)/k) + (P4/x[2])^((k-1)/k) - 1)
  }

  # Calling the Nelder-Meade optimization algorithm
  # See Package 'Optimization' for details
  res <- optim_nm(fun=W, start=c(P2init, P3init), maximum=FALSE, k=2, tol=1e-6, trace=TRUE)
  print(res$trace)

  # Outputting 2 graphs
  par(mfrow=c(1,2))
  plot(res)
  plot(res, 'contour')

  # Writing results to the active Excel worksheet
range <- EXCEL$Application$get_Range( "B6" );
range$put_Value(res$par[1]); # Pressure P2
range <- EXCEL$Application$get_Range( "B7" );
range$put_Value(res$par[2]); # Pressure P3
range <- EXCEL$Application$get_Range( "B8" );
range$put_Value(res$function_value); # Work W

}
```

- Pressures P_1 and P_2 are initialized in the Excel® spreadsheet.
- The VBA code calls the *compression_R()* function, which returns the work *W*.
- Here, the R code writes directly to the Excel® spreadsheet.



BERT console



- Graphs (here, a contour plot) and data (here, the trace of the optimization path) returned from the R functions are produced, as expected. **Data output by R functions are written to the BERT console.**
- Interfacing between Excel® and R is done in a few minutes time, top!
- Students can modify the Excel® interface as they wish, making the work interactive and fun while giving them the opportunity to interactively test the behaviour of the numerical algorithm, such as its sensivity to initialisation, or experiment with the design of the pumping system and more.

Statistical process control example : drawing and analysing control charts using the qcc package (https://cran.r-project.org/web/packages/qcc/index.html , Scrucca et al.)

BERT console

BERT Console

File Edit View Packages Help

Welcome | myFunctions.R × | functions.r ● | Preferences | myF

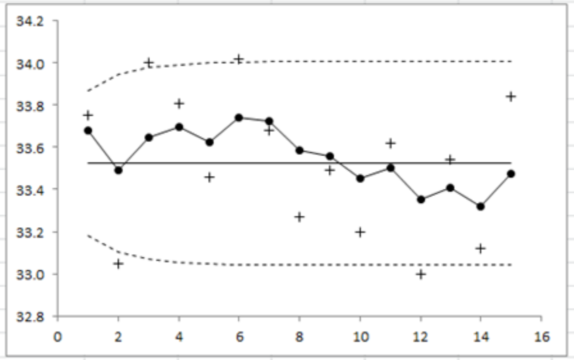
```r
ewma_R <- function(x, lambda, nsigmas) {

 # loading the 'qcc' ppackage
  library("qcc")

  # calling the ewma() function from the "qcc" package
  q <- ewma(data=x, lambda=lambda, nsigmas=nsigmas)
  summary(q)

  # Writing output values to the active Excel worksheet
  range <- EXCEL$Application$get_Range( "F2:G16" );
  range$put_Value(q$limits);
  range <- EXCEL$Application$get_Range( "H3" );
  range$put_Value(q$center);
  range <- EXCEL$Application$get_Range( "H5" );
  range$put_Value(q$std.dev);
}
```

VBA editor

```vba
Sub ewma_VBA()

' Section 0: variables déclaration
'-----------------------------------
    ' All variables passed to your R function should be defined as Variant
    Dim X As Variant, lambda As Variant, nsigmas As Variant
    'Variant variables used as output to Excel API function Application.Run()
    Dim var1 As Variant

    On Error Resume Next

' SECTION 1: importing data from the Excel spreadsheet
'--------------------------------------------------------
    Worksheets("Example_SPC").Activate
    X = Range("B2:B16").Value
    lambda = Range("C2").Value
    nsigmas = Range("C4").Value


' SECTION 2 : Calling your own R function
' This is where all the calculations are done, VBA does none !
'-------------------------------------------------------------
    var1 = Application.Run("BERT.Call", "ewma_R", X, lambda, nsigmas)

' SECTION 3: Writing output from your own R function back into the Excel spreadsheet
'------------------------------------------------------------------------------------
' In this example, outputs from your R function are writtent to the active
' Excel spreadsheet directly from the R function utself.

End Sub
```

x	lambda	Sample	LCL	UCL		ewma variable
33.75	0.3	1	33.18	33.87	center	33.68
33.05	nsigmas	2	33.10	33.94	33.5233333	33.49
34	2.7	3	33.07	33.98	std.dev	33.64
33.81		4	33.05	33.99	0.42616515	33.69
33.46		5	33.05	34.00		33.62
34.02		6	33.04	34.00		33.74
33.68		7	33.04	34.01		33.72
33.27		8	33.04	34.01		33.59
33.49		9	33.04	34.01		33.56
33.2		10	33.04	34.01		33.45
33.62		11	33.04	34.01		33.50
33		12	33.04	34.01		33.35
33.54		13	33.04	34.01		33.41
33.12		14	33.04	34.01		33.32
33.84		15	33.04	34.01		33.48

Call ewma_VBA()



- Not discussed here, it is implicit that you can use all the interfacing capability built into VBA (with userforms) to quickly and simply produce neat interactive user interfaces.

Geochemical example: Study of the relative thermodynamic stability (dissolution/precipitation) of two minerals in pure water, namely gypsum and anhydrite, as a function of temperature at 1 atm. (https://cran.r-project.org/web/packages/phreeqc/index.html , Charlton et al.)

VBA editor

```vba
Sub phreeqc_VBA()

' Section 0: variables déclaration
'---------------------------------
    ' All variables passed to your R function should be defined as Variant
    Dim Tmin As Variant, Tmax As Variant, Nsteps As Variant
    'Variant variables used as output to Excel API function Application.Run()
    Dim var1 As Variant
    ' variables used to get the size of the R output variable var1
    Dim nrow As Integer, ncol As Integer

    On Error Resume Next

    ' Clear the output range
    Range("C:N").ClearContents

' SECTION 1: importing data from the Excel spreadsheet
'--------------------------------------------------------
    Worksheets("Example_phreeqc").Activate
    Tmin = Range("A2").Value
    Tmax = Range("A4").Value
    Nsteps = Range("A6").Value

' SECTION 2 : Calling your own R function
' This is where all the calculations are done, VBA does none !
'--------------------------------------------------------------
    var1 = Application.Run("BERT.Call", "phreeqc_R", Tmin, Tmax, Nsteps)

' SECTION 3: Writing output from your own R function back into the Excel spreadsheet
'------------------------------------------------------------------------------------
    nrow = UBound(var1, 1)
    ncol = UBound(var1, 2)
    Range(Cells(1, 3), Cells(nrow, ncol + 2)).Value = var1


End Sub
```

BERT console

BERT Console

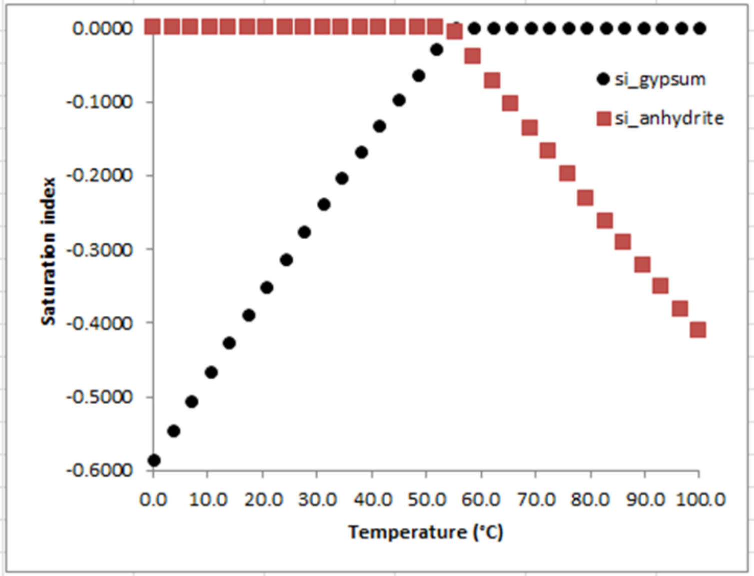
File Edit View Packages Help

Welcome | myFunctions.R × | functions.r | Preferences | myFunctions_phreeqc.R

```r
phreeqc_R <- function(Tmin, Tmax, Nsteps) {

# loading the 'phreeqc' package
library("phreeqc")

range <- EXCEL$Application$get_Range( "A4" )
x <- range$get_Value()
print(paste("x=",x))

# load the phreeqc.dat database
phrLoadDatabaseString(phreeqc.dat)

# Building a PHREEQC script and passing it the input temperatures
phrRunString(phreeqc_script(Tmin, Tmax, Nsteps))

# retrieve selected_output as a list of data.frame
so <- phrGetSelectedOutput()

 # returning the results output by PHREEQC
return(so$n1)
}

phreeqc_script <- function(Tmin, Tmax, Nsteps) {
# writing the PHREEQC script
input <- vector()

input <- c(input, "TITLE Example 2.--Temperature dependence of solubility")
input <- paste(input, "of gypsum and anhydrite")
input <- c(input, "SOLUTION 1 Pure water")
input <- c(input, "pH      7.0")
input <- c(input, "temp    25.0")
input <- c(input, "EQUILIBRIUM_PHASES 1")
input <- c(input, "Gypsum          0.0     1.0")
input <- c(input, "Anhydrite       0.0     1.0")
input <- c(input, "REACTION_TEMPERATURE 1")
# Creation of the modified line with new entries
Excel_input <- paste(Tmin," ",Tmax," in ",Nsteps," steps")
input <- c(input, Excel_input)
input <- c(input, "SELECTED_OUTPUT")
input <- c(input, "-file   ex2.sel")
input <- c(input, "-temperature")
input <- c(input, "-si     anhydrite  gypsum")
input <- c(input, "END")

return(input)
}
```

Geochemical example: Study of the relative thermodynamic stability (dissolution/precipitation) of two minerals in pure water, namely gypsum and anhydrite, as a function of temperature at 1 atm. (https://cran.r-project.org/web/packages/phreeqc/index.html , Charlton et al.)

	A	B	I	J	K	L	M
1	Tmin		pH	pe	temp.C.	si_anhydrite	si_gypsum
2	0		7	4	25.0	65535.0000	65535
3	Tmax		7.41	12.66	0.0	-0.5854	0
4	100		7.37	12.37	3.4	-0.5448	0
5	Nsteps		7.32	12.09	6.9	-0.5048	0
6	30		7.27	11.81	10.3	-0.4653	0
7			7.22	11.55	13.8	-0.4263	0
8	Call to VBA subroutine phreeqc_R		7.17	11.30	17.2	-0.3878	0
9			7.13	11.05	20.7	-0.3498	0
10			7.08	10.80	24.1	-0.3123	0
11			7.03	10.57	27.6	-0.2753	0
12			6.99	10.34	31.0	-0.2387	0
13			6.94	10.11	34.5	-0.2026	0
14			6.90	9.89	37.9	-0.1669	0
15			6.86	9.67	41.4	-0.1317	0
16			6.82	9.45	44.8	-0.0968	0
17			6.78	9.24	48.3	-0.0624	0
18			6.74	9.03	51.7	-0.0284	0
19			6.71	8.85	55.2	0	-0.0053
20			6.67	8.63	58.6	0	-0.0385
21			6.64	8.43	62.1	0	-0.0714
22			6.60	8.24	65.5	0	-0.1039
23			6.57	8.06	69.0	0	-0.1360
24			6.54	7.87	72.4	0	-0.1678
25			6.51	7.70	75.9	0	-0.1993
26			6.48	7.51	79.3	0	-0.2304
27			6.45	7.30	82.8	0	-0.2612
28			6.43	7.20	86.2	0	-0.2917
29			6.40	7.00	89.7	0	-0.3219
30			6.38	6.83	93.1	0	-0.3517
31			6.36	6.67	96.6	0	-0.3813
32			6.33	6.51	100.0	0	-0.4106



Conclusions

- With Excel®'s inescapability in the engineering community, both at university and at the work place, the ability to bring all of R's numerical strength into Excel® applications is a highly attractive proposition.

- BERT is a simple, efficient and free R-Excel® interoperability solution. Interfacing an existing R code with an Excel® worksheet adds interactivity and acceptability to one's R code. Very few lines of code (VBA and/or R) are necessary to embed your R code into Excel®.

- Interfacing Excel® and R for undergraduate engineering courses brings interactivity and fun into the learning of engineering courses, which can only help with the understanding of the course material.

- Beyond engineering education:

  - Development of excel® applications that take advantage of R's capabilities can be highly beneficial for practicing engineers.

  - Mixing advanced VBA and R computing can produce rather "sophisticated" engineering applications, with a user-interface that will never be a turn-off for any practicing engineer, as engineers (and others…) do love Excel® !

  - The combination of Excel® and R offers a highly competitive environment, technically and financially, for all engineering professionals and companies.

Thank you
for youR Excellent attention

Should you want to get in touch:

Florent.bourgeois@toulouse-inp.fr

https://lgc.cnrs.fr/annuaire/florent-bourgeois/

Special thanks to the following INP-ENSIACET undergraduate engineering students for their contribution:
Alexandre BARON, Florent CHAUVIN, Camille HERBIN, El Mehdi LAAISSAOUI, Robin MOINEAU and Roxanne TOUZÉ