Basic Matrix and Vector Functions written with VBA/Excel

Introduction

This release (October 2012) includes about 60 matrix and vector functions for excel users and macro (VBA) programmers. All these functions are written with the native VBA (Visual Basic for Applications) language of excel.

Why would you need matrix operations in excel?

- You want to implement complex calculations, simulations and optimizations easily with macros (VBA) in excel.
- You want to translate the functions and scripts written with mathematical applications like matlab into the excel VBA environment, to profit from built in functions of excel like charting, reporting and data storage.

finaquant® protos, the noncommercial calculation engine (.NET library) based on table functions

Finaquant offers this VBA software as an open source project for free. You can however support the continuity and quality of the project by donating, and by contributing to the related community forum. For supplementary information and updates visit the central download page of VBA functions and related community forum:

http://finaquant.com/download/matrixvectorvba

http://finaquant.com/forum/matrix-functions-vba-excel

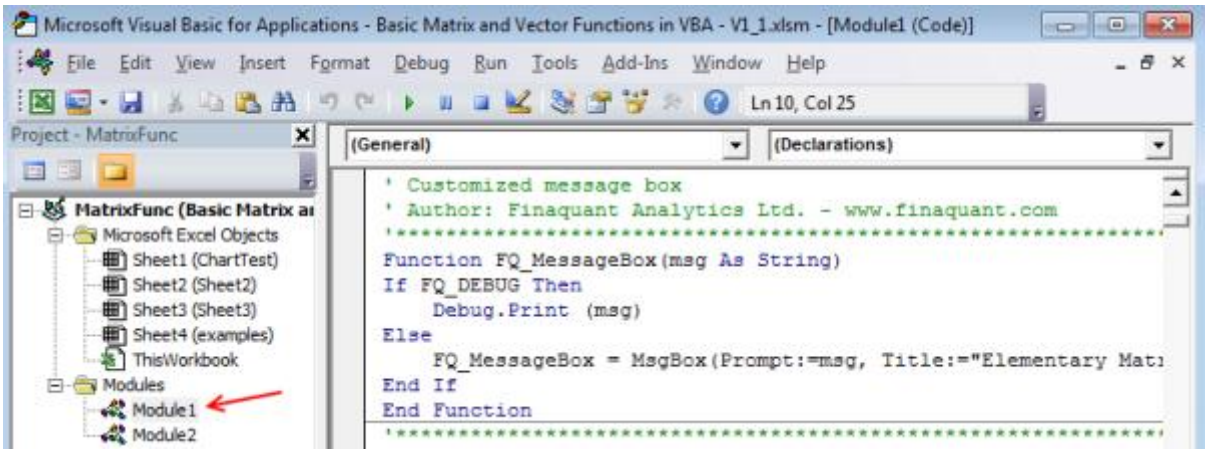
Finaquant Analytics Ltd.
Email: contactfq@finaquant.com twitter: @finaquant

Humrigenstrasse 45, CH-8704 Herrliberg, Switzerland, tel: +41 78 842 24 47

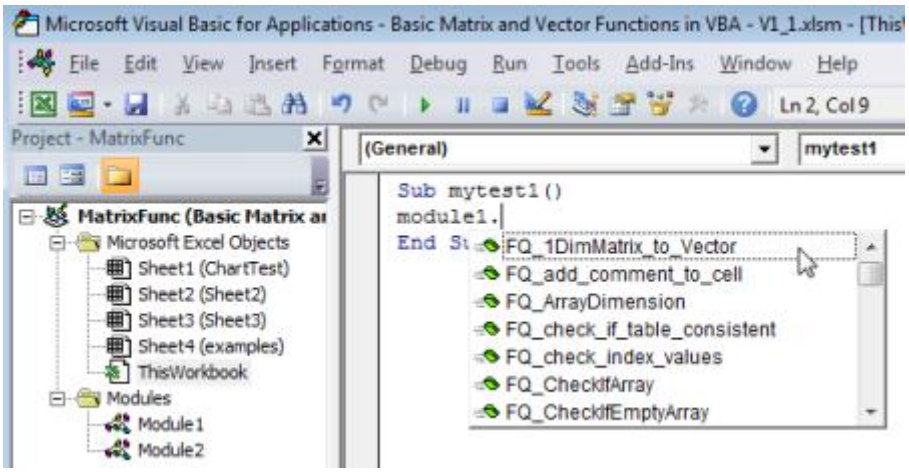
Contents

Getting started

All the matrix and vector functions reside in **Module1** section of the excel file you have downloaded (BasicMatrixAndVectorFunctionsInVBA-V1_1.xlsm). You can refer to all these functions in your own VBA procedures and functions in other modules and sections (like ThisWorkbook) of your excel file.



Module2 contains some very useful test procedures (sub in VBA) that test all the functions and procedures residing in Module1.

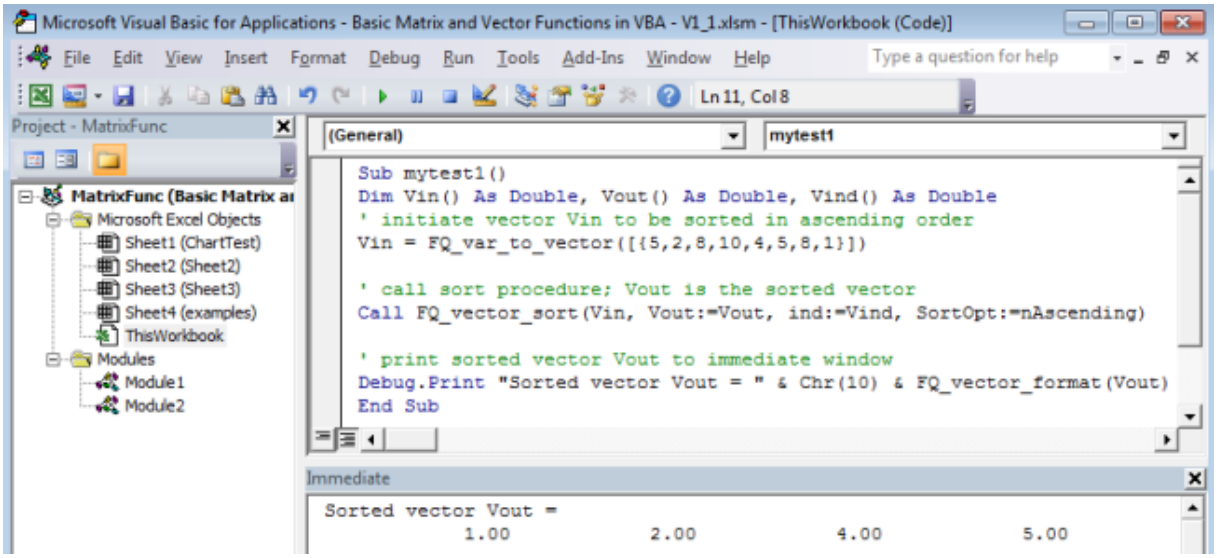


You can see the whole list of matrix and vector functions by tipping “module1.” in any VBE window. Note that all function names begin with “FQ_” (FQ for FinaQuant).

The VBA section of the excel file is initially password protected. You can download the password from the central download page for free in order to view and edit the VBA code in all modules. We recommend you strongly however, to make no changes in the original code unless it is absolutely necessary due to an urgent bug. You can always make your own corrections and extensions in other modules and sections. We would very much appreciate if you inform us about your corrections, improvements and extensions in the related forum at finaquant.com.

Your first VBA procedures with matrix and vector functions

Procedure mytest() in VBE section ThisWorkbook:




```
Sub mytest1()
Dim Vin() As Double, Vout() As Double, Vind() As Double
' initiate vector Vin to be sorted in ascending order
Vin = FQ_var_to_vector([{5,2,8,10,4,5,8,1}])
' call sort procedure; Vout is the sorted vector
Call FQ_vector_sort(Vin, Vout:=Vout, ind:=Vind, SortOpt:=nAscending)
' print sorted vector Vout to immediate window
Debug.Print "Sorted vector Vout = " & Chr(10) &
FQ_vector_format(Vout)
End Sub
```

This procedure creates first a vector with 8 elements, sorts them in ascending order, and prints the resultant sorted vector into the immediate window (press F5 to execute the procedure).

A second, and bigger example: Procedure TEST_FQS_matrix_inverse() in ThisWorkbook:

```
'*****************************************************************
' Procedure for testing FQS_matrix_inverse()
' Author: Finaquant Analytics Ltd. - www.finaquant.com
'*****************************************************************
Sub TEST_FQS_matrix_inverse()
Dim r_in As Range, r_out As Range, WorkBookName As String
' define input and output ranges
Set r_in = ThisWorkbook.Sheets("examples").Range("A4:D7")
Set r_out = ThisWorkbook.Sheets("examples").Range("F4:G5")
' call worksheet function
Call FQS_matrix_inverse(r_in, r_out)
End Sub
```

```
'*****************************************************************
' Example worksheet procedure (name starting with FQS_)
' Write inverse of the matrix to the given output range
' to illustrate general steps 1 (read), 2 (calculate) and 3 (write)
' Author: Finaquant Analytics Ltd. - www.finaquant.com
'*****************************************************************
Sub FQS_matrix_inverse(InputRange As Range, OutputRange As Range)
Dim M() As Double, Minv() As Double
On Error GoTo EH1    ' for capturing possible errors

' Step 1: Read all input data from worksheets
' Read matrix from the worksheet (input range)
M = FQ_range_to_matrix(InputRange)

' Step 2: Calculate results (outputs) with matrix and vector
functions
' Calculate inverse matrix
Minv = FQ_matrix_inverse(M)

' Step 3: Write results into excel sheets
Call FQ_matrix_to_range(Minv, OutputRange)
Exit Sub

EH1:     ' error handling
FQ_MessageBox ("Error in FQS_matrix_inverse: " & Err.Number & " - "
& Err.Description)
Err.Raise (Err.Number)
End Sub
```

	A	B	C	D	E	F	G	H	I
1	inputs		outputs						
2									
3	Matrix MatC					Matrix Inverse(MatC)			
4	32	25	29	13		0.025126	-0.00715	-0.01328	0.011129
5	58	47	20	95		0.034184	0.029321	0.006123	-0.04209
6	11	53	86	43		-0.00911	-0.01939	0.007839	0.019559
7	68	37	50	83		-0.03033	0.004468	0.003429	0.009909
8									

The procedure FQS_matrix_inverse illustrates the general pattern of a high-level function based on matrices and vectors. First, get all the relevant data as input parameters, second, make all the matrix and vector calculations without any reference to the data in sheets (or database), third write the results into sheets (or database). We recommend you to stick to this pattern in your own worksheet functions:

- Step 1: Read data from worksheets into matrices and vectors
- Step 2: Make all the calculations with arrays (i.e. matrices and vectors) and get the results
- Step 3: Write the results back into worksheets

The naming conventions we suggest for procedures and functions:

Use prefix "FQ_" for procedures with matrices and vectors as input and output parameters, without any reference to worksheets in the excel file.

Use prefix "FQS_" for procedures that read input data from worksheets, and writes results (i.e. outputs) back to worksheets, like the example above: FQS_matrix_inverse()

What is a matrix, and what is a vector for VBA functions

We defined a matrix as a two-dimensional array of double, with indices starting from 1:

Declaring a matrix with N rows and M columns (NxM) in VBA for excel:

```
Dim M(1 to N, 1 to M) as double
```

Similarly, a vector is defined as a one-dimensional array of type double, with indices starting from 1:

Declaring a vector of length nLen in VBA:

```
Dim V(1 to nLen) as double
```

There are two functions in Module1 whose only task is checking whether the given parameter is a real matrix or vector:

```
'Returns True if the argument Arr is a matrix; otherwise False
Function FQ_CheckIfMatrix(Arr As Variant) As Boolean
```

In order to qualify as a matrix, the argument Arr must be:

- A two dimensional array of type double
- With element indices (subscripts) starting from 1 (i.e. not from 0) for both dimensions

```
'Returns True if the argument Arr is a vector; otherwise False
Function FQ_CheckIfVector(Arr As Variant) As Boolean
```

In order to qualify as a vector, the argument Arr must be:

- A one dimensional array of type double
- With element indices (subscripts) starting from 1 (i.e. not from 0)

You can easily convert a matrix into a vector, or vice versa, with the available conversion functions. Please check the function list.

How to generate vectors

There are a number of ways and functions for initiating vectors.

1) Hard-wired, manual definition of element values:

```
Dim V1() as double, V2() as double
V1 = FQ_var_to_vector(Array(1, 3, 5,7))
' or
V2 = FQ_var_to_vector([{1,2,3,4,5,6}])
```

2) Vector with sequential values:

```
' creates vector [1, 3, 5, 7]
V1 = FQ_vector_sequence(StartValue:=1, Interval:=2, ElementCount:=4)
' creates vector [1, 1, 1, 1]
V1 = FQ_vector_sequence(StartValue:=1, Interval:=0, ElementCount:=4)
```

3) Vector with random values

```
' Elements with random values between 0 and 1
V1 = FQ_vector_rand(ElementCount:=4)
```

How to generate matrices

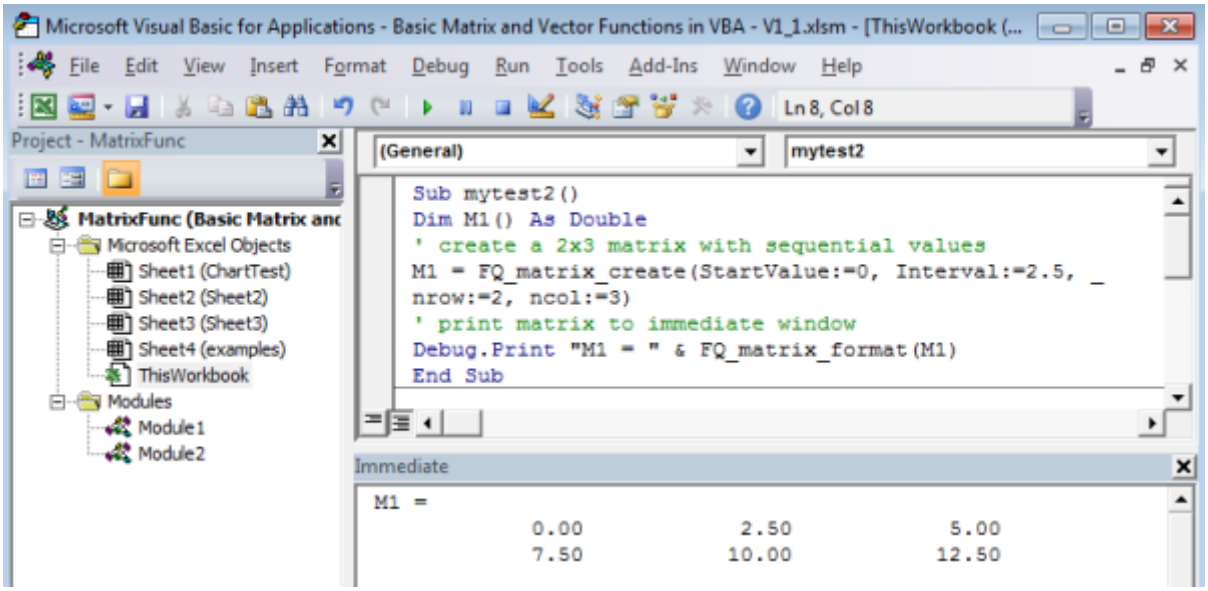
1) Manual definition of element values:

```
Dim M1() as double
' Create a 2x3 matrix
M1 = FQ_var_to_matrix([{1,2,3; 4,5,6}])
```

Note that the semicolon character is here the row separator (delimiter).

2) Matrix with sequential element values

```
' Create a 2x3 matrix with sequential values
M1 = FQ_matrix_create(StartValue:=0, Interval:2.5, nrow:=2, ncol:=3)
```



3) Matrix with random element values

```
' Create a 3x4 matrix with random element values between 0 and 1
M1 = FQ_matrix_rand(nrow:=3, ncol:=4)
```

```
' Create a 3x4 matrix with random element values between 0 and 100
M2 = FQ_matrix_scalar_multiply(M1, 100)
```

Getting and setting element values in VBA code (macro)

Matrices:

```
M1(2, 3) = 5.25 ' set value of matrix element,  nRow = 2, nCol = 3
X = M1(2,3)     ' get value of matrix element,  nRow = 2, nCol = 3
```

Vectors:

```
V1(3) = 4.5     ' Set value of vector element with index 3
X = V1(3)       ' Get value of vector element with index) 3
```

Alias names for functions and procedures

Assume, you would rather have the name `FQ_mmult` instead of `FQ_matrix_multiplication` for the matrix multiplication in linear algebra.

You don't need to cancel the password protection for Modul1 and change the name of the function there. As mentioned at the beginning, we strongly recommend not to make any changes in the original code unless it is strictly necessary due to a bug.

You can easily create an alias name in a third Module (remember: there are test functions in Module2). Just enter following lines in Module3:

```
' Alias name for FQ_matrix_multiplication
Function FQ_mmult(M1() As Double, M2() As Double) As Double()
FQ_mmult  = FQ_matrix_multiplication(M1,  M2)
End Function
```

You can also add new matrix and vector functions to the available list in Module3, in order to use them in other sections of the VBE (Visual Basic Editor), for example in ThisWorkbook.

Next steps

For updates and supplementary information visit the central download page:

http://finaquant.com/download/matrixvectorvba

For interesting VBA code examples, suggestions and questions see the related finaquant forum:

http://finaquant.com/forum/matrixvectorvba

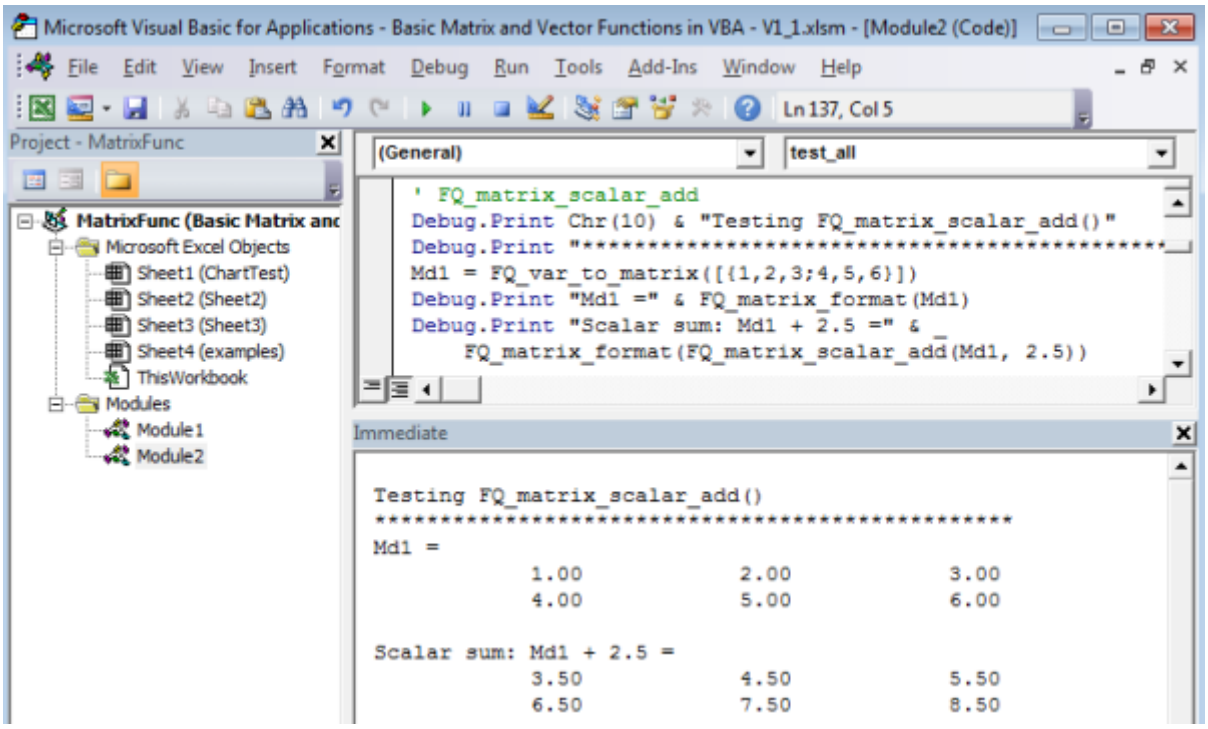
Can you use the VBA functions presented here for your personal and commercial projects? Do you want to contribute for the continuity and quality of the project? Yes, you can contribute by:

- Bringing interesting questions and ideas to related forum
- By donating generously (visit the central download page for donations)

List of functions

1. Matrix-scalar addition - matlab: M2 = M1 + x

```
Function FQ_matrix_scalar_add(M() As Double, x As Double) As
Double()    ' Adds a scalar number to all elements of matrix
```



2. Customized message box

```
Function FQ_MessageBox(msg As String)
```

3. Get array dimension

```
Function FQ_ArrayDimension(Arr As Variant) As Integer
' Returns the  number of dimensions in an array
```

4. Check if array

```
Function FQ_CheckIfArray(Arr As Variant) As Boolean
' Returns True if the argument is an array; otherwise False
```

5. Check if empty array

```
Function FQ_CheckIfEmptyArray(Arr As Variant) As Boolean
' Checks if empty array; returns True if array is empty
' - error if input argument Arr is not an array
```

6. Check if matrix

```
Function FQ_CheckIfMatrix(Arr As Variant) As Boolean
' Returns True if the argument is an matrix; otherwise False
' A matrix must be:
' - 2-dimensional array of data type Double
' - lowest element index must be 1 (lower bound)
```

7. Check if vector

```
Function FQ_CheckIfVector(Arr As Variant) As Boolean
' Returns True if the argument is a vector; otherwise False
' - A vector must be:
' - 1-dimensional array of data type Double or Long
' - lowest element index must be 1 (lower bound)
```

8. Convert variant array to matrix

```
Function FQ_var_to_matrix(Arr As Variant) As Double()
' Converts a variant array with numeric elements into a matrix
' - 1-dimensional variant array is converted into a 1xN horizontal
matrix
' - example 2x3 variant array: [{1,2,3; 4,5,6}]
' - error if dimension of variant array is not in set {0,1,2}
' - error if there is a non-numeric element in variant array
```

9. Convert variant array to vector

```
Function FQ_var_to_vector(Arr As Variant) As Double()
' Converts a variant array with numeric elements into a vector
' - example variant array: [{1,2,3,4,5,6}]
' - error if dimension of variant array is not 0 or 1
' - error if there is a non-numeric element in variant array
```

10. Format matrix for print

```
Function FQ_matrix_format(M As Variant) As String
' Converts a matrix into a printable formatted string
' for displaying matrices to user
' - returns "ERROR" if the input argument is not a matrix
```

11. Format vector for print

```
Function FQ_vector_format(V As Variant) As String
' Converts a vector into a printable formatted string
' for displaying vectors to user
' - returns "ERROR" if the input argument is not a matrix
```

12. Convert 1-dimensional matrix to vector

```
Function FQ_1DimMatrix_to_Vector(M() As Double) As Double()
' Converts a 1-dimensional (1xN or Nx1) matrix to a vector
' - returns the same vector if the input M is a vector
' - error if M is not a 1-dimensional matrix
```

13. Convert vector to a 1-dimensional matrix

```
Function FQ_Vector_to_1DimMatrix(V() As Double, malign As
MatrixAlignment) As Double()
' Converts Vector to a 1-dimensional matrix, either
' vertical or horizontal depending on matrix alignment argument
' - returns the same matrix (aligned as requested) if input V is a
1-dim matrix
' - error if M is not a vector or 1-dim matrix
```

14. Transpose matrix – matlab: M2 = M1'

```
Function FQ_matrix_transpose(M() As Double) As Double()
' Transpose matrix
' M2 = transpose(M1)  such that M1(j, i) = M2(i, j)
```

```
' - error if M is not a matrix
```

15. Inverse matrix – matlab: M2 = inv(M1)

```
Function FQ_matrix_inverse(M() As Double) As Double()
' Inverse matrix: Y = inv(M)
' - error if M is not a square matrix (ncol = nrow)
```

16. Read a worksheet range into variant array

```
Function FQ_range_to_variant(Rn As Range) As Variant
' Reads the values of a range of cells into a 2-dimensional
' NxM variant array (1 to N, 1 to M)
```

17. Write variant array values into a worksheet range

```
Sub FQ_variant_to_range(Arr As Variant, Rn As Range)
' Writes the values of a 2-dimensional variant array into a range in
excel starting from the upper left corner of the range (Cells(1,1))
' - error if array is empty or not initialized, or not 2-dim
```

18. Write values of a matrix into a worksheet range

```
Sub FQ_matrix_to_range(M() As Double, Rn As Range)
' Writes the values of a matrix (2-dim double) into a range in excel
starting from the upper left corner of the range (Cells(1,1))
' - error if M is not a matrix
```

19. Write values of a vector into a worksheet range

```
Sub FQ_vector_to_range(V() As Double, Rn As Range, Direction As
MatrixAlignment)

' Writes the values of a vector (1-dim double) into a range in excel
starting from the upper left corner of the range (Cells(1,1))
' - error if V is not vector
```

20. Read worksheet range into a matrix

```
Function FQ_range_to_matrix(R As Range) As Double()
' Reads a numeric range and converts it into a matrix with the same
' row and column size
' - error if any range value is not numeric; all cells in the range
' must be numeric and non-empty
```

21. Read worksheet range into a vector

```
Function FQ_range_to_vector(R As Range) As Double()
' Reads a numeric range row by row and writes the values into a
vector
' - error if any range value is not numeric; all cells in the range
' must be numeric and non-empty
```

22. Convert vector to matrix

```
Function FQ_vector_to_matrix(V() As Double, FillOption, nrow As
Long, ncol As Long) As Double()
```

```
' Writes the values of a vector into a matrix either row by row, or
' column by column.
' - matrix row/column size is given by parameters nrow and ncol
' - vector elements are recycled with mod() function, if matrix have
' more elements than the vector
' - error if V is not a vector
' - error of nrow < 1 or ncol < 1
' - error if invalid fill option
```

23. Convert matrix to vector

```
Function FQ_matrix_to_vector(M() As Double, FillOption) As Double()
' Reads the elements of a matrix either row by row, or column by
column,
' and writes their values into a vector
' - error if input argument M is not a matrix
' - error if invalid fill option
```

24. Create matrix with sequential element values

```
Function FQ_matrix_create(StartValue As Double, Interval As Double,
nrow As Long, ncol As Long) As Double()
Dim M() As Double
' Creates matrix with sequential element values with given row and
' column sizes. Fills matrix row-wise with numbers.
' - set Interval = 0 for constant element values
' - error input arguments nrow and ncol are not positive integers
```

25. Create matrix with random element values – matlab: M = rand(3, 4)

```
Function FQ_matrix_rand(nrow As Long, ncol As Long) As Double()
' Creates matrix with random element values between 0 and 1
' with given row and column sizes. Fills matrix row-wise with
numbers.
' - error input arguments nrow and ncol are not positive integers
```

26. Create vector with sequential element values – matlab: V = 1:2:9

```
Function FQ_vector_sequence(StartValue As Double, Interval As
Double, ElementCount As Long) As Double()
' Creates a vector with given length (ElementCount), start value
' and interval between subsequent elements.
' - error if ElementCount is not a positive integer
```

27. Create vector with random element values – matlab: V = rand(1,10)

```
Function FQ_vector_rand(ElementCount As Long) As Double()
' Creates a vector with random element values between 0 and 1
' with given vector length (ElementCount)
' and interval between subsequent elements.
' - error if ElementCount is not a positive integer
```

28. Check index (subscript) values

```
Function FQ_check_index_values(V As Variant, UpperLimit As Long,
LowerLimit As Long) As Boolean
' Check if all vector values are positive integers within limits
' i.e. return True if:
' 1) LowerLimit <= all values <= UpperLimit, and
' 2) all values are whole numbers
' - often used to check the validity of matrix or vector indices
' - error if input argument V is not a vector
```

29. Matrix partition - matlab : M(RowInd, ColInd)

```
Function FQ_matrix_partition(M() As Double, Optional RowInd As
Variant, Optional ColInd As Variant) As Double()
' returns partition of a matrix indicated by column and row index
vectors
' - no row/column index vector means, all rows/columns are selected
' - error if any index element is not a positive integer larger than
0
' - error if any index element is larger than row/col size of matrix

Vector partition – matlab: V(ind)
Function FQ_vector_partition(V() As Double, ind() As Double) As
Double()
```

30. Vector partition - matlab : V(ind)

```
Function FQ_vector_partition(V() As Double, ind() As Double) As
Double()
' returns partition of a vector indicated index vector ind
' - error if any index element is not a positive integer larger than
0
' - error if any index element is larger than vector length
```

31. Element sum of matrix – matlab: sum(M, dim)

```
Function FQ_matrix_element_sum(M() As Double, SumOption As
MatrixDirection) As Double()
' Returns the sum of elements of matrix M, either row or column wise
' - Rowwise sum returns a horizontal 1xNcol matrix
' - Columnwise sum returns a vertical 1 xNrow matrix
' - Element sum (all elements) returns a 1x1 matrix
' - error if M is not a matrix
' - error if SumOption is not 1 (nRowWiseSum) or 2 (nColWiseSum) or
3 (nElementSum)
```

32. Aggregate matrix – matlab: sum(M,d), min(M,d), max(M,d), avg(M,d), median(M,d)

```
Function FQ_matrix_aggregate(M() As Double, AggregateFunct As
String, AggregateOption As MatrixDirection) As Double()
' Applies the given aggregation function (sum, min, max, avg,
median)on the matrix, and returns a scalar number.
' - AggregateOption: nRowByRow, nColByCol or nAllElements
```

```
' -  nColByCol aggregation returns a 1xN horizontal matrix
' -  nRowByRow aggregation returns a Nx1 vertical matrix
' - nAllElements aggregation returns a 1x1 unity matrix
' - error if M is not a vector
' - error if unknown aggregation function
' - error if unknown aggregation option
```

33. Add a scalar number to all matrix elements – matlab: M2 = M1 + x

```
Function FQS_matrix_scalar_add(M As Range, x As Variant)
' Spreadsheet version of the function FQ_matrix_scalar_add
```

34. Add a scalar number to all vector elements

```
Function FQ_vector_scalar_add(V() As Double, x As Double) As
Double()
' Adds a scalar number to all elements of vector
' - error if V is not a vector
```

35. Multiply all elements of vector with a scalar number – matlab: M2 = M1 * x

```
Function FQ_vector_scalar_multiply(V() As Double, x As Double) As
Double()
' Multiplies all elements of vector with a scalar number x
' - error if V is not a vector
```

36. Aggregate vector – matlab: sum(V), min(V), max(V), avg(V), median(V)

```
Function FQ_vector_aggregate(V() As Double, AggregateFunct As
String) As Double
' Applies the given aggregation function (sum, min, max, avg,
median)
' on the vector, and returns a scalar number.
' - error if V is not a vector
' - error if unknown aggregation function
```

37. Number of elements in matrix

```
Function FQ_matrix_element_count(M() As Double)
' Returns the number of elements in matrix M
' - error if M is not a matrix
```

38. Matrix scalar multiplication – matlab: M2 = M1 * x

```
Function FQ_matrix_scalar_multiply(M() As Double, x As Double) As
Double()
' Multiplies all elements of matrix with a scalar number
' - error if M is not a matrix
```

39. Matrix vector multiplication – matlab: M * V, or V' * M (V: 1xN or Nx1 matrix)

```
Function FQ_matrix_vector_multiply(M() As Double, V() As Double,
MultiplyOption As Byte) As Double()
' Multiplies rows or columns of matrix M with corresponding elements
' of vector V
' MultiplyOption: nRowByRow, nColByCol
' - error if M is not a matrix, or if V is not a vector
' - error if vector size is not equal to row/column size of matrix
' depending on MultiplyOption
```

40.Sum of two equal sized matrices – matlab: M3 = M1 + M2

```
Function FQ_matrix_matrix_sum(M1() As Double, M2() As Double) As
Double()
' Adds up the elements of two matrices with identical row/column
sizes
' i.e. element-wise sum of two matrices
' - error if M1 and/or M2 is not a matrix
' - error if row and column sizes of matrices M1 and M2 are not
identical
```

41. Sum of two equal sized vectors – matlab: V3 = V1 + V2 (V: 1xN or Nx1 matrix)

```
Function FQ_vector_vector_sum(V1() As Double, V2() As Double) As
Double()
' Adds up the elements of two vectors with identical lengths
' i.e. element-wise sum of two equal-sized vectors
' - error if V1 and/or V2 is not a vector
' - error if lengths of vectors V1 and V2 are not identical
```

42. Element-wise multiplication of two equal-sized matrices – matlab: M3 = M1 .* M2

```
Function FQ_matrix_elementwise_multiply(M1() As Double, M2() As
Double) As Double()
' Elementwise multiplication of two equal-sized matrices
' R = M1 .* M2 (matlab notation)
' - error if M1 and/or M2 is not a matrix
' - error if row and column sizes of matrices M1 and M2 are not
identical
```

43. Element-wise division of two equal-sized matrices – matlab: M3 = M1 ./ M2

```
Function FQ_matrix_elementwise_divide(M1() As Double, M2() As
Double) As Double()
' Elementwise division of two equal-sized matrices
' R = M1 ./ M2 (matlab notation)
' - error if M1 and/or M2 is not a matrix
' - error if row and column sizes of matrices M1 and M2 are not
identical
```

44. Reverse the order of vector elements

```
Function FQ_vector_reverse(V() As Double) As Double()


' Reverse the order of vector elements; f.e. [1 4 3] --> [3 4 1]
' - error if V is not a vector
```

45. Matrix multiplication in linear algebra, C = A x B – matlab: M3 = M1 * M2

```
Function FQ_matrix_multiplication(M1() As Double, M2() As Double) As
Double()
' Matrix multiplication in linear algebra,  C = A x B
' - error of M1 and/or M2 is not a matrix
' - error if matrix sizes don't match: ncol of M1 must be equal to
nrow of B
```

46. Append matrix M2 to matrix M1 – matlab: M3 = [M1, M2], or M3 = [M1; M2]

```
Function FQ_matrix_append(M1() As Double, M2() As Double,
AppendOption As MatrixAlignment) As Double()
' Appends matrix M2 to M1 either vertically or horizontally
' AppendOption: AppendVertically or AppendHorizontally
' - returns empty array if both matrices are empty
' - returns M2 if M1 is empty
' - returns M1 if M2 is empty
' - error if M1 is not a matrix unless it is empty
' - error if M2 is not a matrix unless it is empty
' - error if matrix sizes don't match for an append operation
'   row sizes must be equal for horizontal append
'   column sizes must be equal for vertical append
```

47. Appends vector V2 to vector V1 – matlab: V3 = [V1, V2], or V3 = [V1; V2]

```
Function FQ_vector_vector_append(V1() As Double, V2() As Double) As
Double()
' Appends vector V2 to V1 such that result vector = [V1, V2]
' - returns empty array if both vectors are empty
' - returns V2 if V1 is empty
' - returns V1 if V2 is empty
' - error if V1 is not a vector unless it is empty
' - error if V2 is not a vector unless it is empty
' - error if V1 and/or V2 is not a vector unless they are empty
```

48. Apply mathematical function on matrix elements – matlab: sin(M), abs(M), fix(M), etc.

```
Function FQ_matrix_operation(M() As Double, OperationName As String)
As Double()
' Applies the given mathematical operation like abs(), fix(), sin()
etc.
' on all elements of the matrix M (all available single-argument VBA
functions)
' - error if input argument M is not a matrix
```

49. Apply mathematical function on vector elements – matlab: sin(V), abs(V), fix(V), etc.

```
Function FQ_vector_operation(V() As Double, OperationName As String)
As Double()
' Applies the given mathematical operation like abs(), fix(), sin()
etc.
' on all elements of the vector V (all available single-argument VBA
functions)
' - error if input argument V is not a vector
```

50. Assign to a vector partition – matlab: V1(ind) = V2

```
Function FQ_vector_partition_assign(V1() As Double, ind1() As
Double, V2() As Double) As Double()
' Assigns values of vector V2 to the partition of vector V1 selected
' by the index vector ind1. i.e. V1(ind1) = V2
' - error if V1 and/or V2 and/or ind1 are not vectors
' - error of length(V2) is not equal to length(ind1)
```

```
' - error if an index value in ind1 is not a positive integer
between 1 and length of V1
' - error if index vector ind1 is not unique with distinct index
values
```

51. Assign to a matrix partition – matlab: M1(RowInd, ColInd) = M2

```
Function FQ_matrix_partition_assign(M1() As Double, rowind1() As
Double, colind1() As Double, M2() As Double) As Double()
' Assigns values of matrix M2 to the partition of matrix M1 selected
' by the index vectors rowind1 and colind1: M1(rowind1, colind1) =
M2
' - error if M1 and/or M2 are not matrices
' - error if rowind1 and/or rowind2 are not vectors
' - error if length of vectors rowind1/colind1 are not equal to
row/column size of M2
' - error if index value in rowind1/colind1 are not a positive
integers between 1 and nrow1/ncol1
' - error if index vectors rowind1/colind1 are not unique with
distinct index values
```

52. Sort vector elements – matlab: [Vsorted, ind] = sort(V)

```
Sub FQ_vector_sort(Vin() As Double, Vout() As Double, ind() As
Double, SortOpt As SortOption)
' Sorts the elements of input vector Vin in ascending or descending
' order depending on SortOption
' - ind: index vector such that Vout = Vin(ind)
' - error if Vin is not a vector
' - assumption: minimum absolute difference between all element
values: MinElementDiff = 1/100000
```

53. Unique (distinct) vector elements – matlab: [Vunique, ind] = unique(V)

```
Sub FQ_vector_unique(Vin() As Double, Vout() As Double, ind() As
Double)
' Sorted output vector Vout contains distinct (unique) elements of
the input vector Vin in ascending order.
' - ind: index vector such that Vout = Vin(ind)
' - error if Vin is not a vector
```

54. Check if unique vector

```
Function FQ_vector_if_unique(V() As Double) As Boolean


' Returns True if vector Vin contains distinct (unique) element
values;
' otherwise returns false
' - error if Vin is not a vector
```

55. Find elements in vector – matlab: find(V > 1)

```
Sub FQ_vector_find_elements(V() As Double, ComparisonOperator As
String, SearchValue As Double, ind() As Double, IfFound As Boolean)
' Finds the positions (indices) of vector elements that satisfy the
' search criterion.
' Returns IfFound = True if there is at least a single match;
' otherwise returns false and empty index vector ind.
```

```
' - Example: Return all indices ind of vector elements such that
'   V(i) >= SearchValue
' - ComparisonOperator is a string from set {'=',  '<>', '>', '>=',
'<', '<='}
' - error if V is not a vector
' - error of undefined comparison operator not in the set above
```

56. Find vector indices ind such that V1 = V2(ind)

```
Sub FQ_vector_find_indices(V1() As Double, V2() As Double, ind() As
Double, V2containsV1 As Boolean)
' V2containsV1 = True: V2 contains all element values in V1; i.e.
'   V1 is a subset of V2
' - returned vector ind can be an empty vector
' - index value is 0 for an element of V1 not found in V2, for
example:
'   V1 = [1 2 3], V2 = [5 2 3 6] --> ind = [0 1 2]
' - error if V1 and/or V2 is not a vector
' Function alias name: FQ_vector_map_to_set_elements
```

57. Sort rows matrix in ascending or descending order

```
Sub FQ_matrix_sort(M1() As Double, Colind() As Double, Ms() As
Double, RowInd() As Double)
' Sort rows of matrix in either ascending or descending order,
' w.r.t. given column indices with vector ColInd
' ColInd = [1 3 -2] means order by 1., 3. and 2. columns, 2. in
descending order
' - returns empty matrix Ms if M1 is empty
' - returns row indices with RowInd, such that Ms = M1(RowInd, :)
' - error if the absolute value of a column index in ColInd is not a
'   positive integer between 1 and column size of M1
' - error if abs(ColInd) is not a unique vector
' - error if M1 is not a matrix
```

58. Create embedded chart

```
Sub FQ_create_embedded_chart(SheetName As String, DataRange As
Range, _
    Optional nLeft As Variant, Optional nTop As Variant, _
    Optional nWidth As Variant, Optional nHeight As Variant, _
    Optional ChartType As Variant)
' Creates an embedded chart on the given sheet
' type "MyChart.ChartType =" in VBE to see the available chart types
' - default values for size parameters nLeft, nTop, nWidth, nHeight:
'       50, 50, 300, 200

' - error if a worksheet with the given name does not exist
```

59. Convert variant array into a string array

```
Function FQ_var_to_str(Arr As Variant) As String()
' Converts the 1 or 2 dimensional variant array into a string array
' - error if array dimension arrdim = 0 or arrdim > 2
```

60. Create a data table

```
Function FQ_create_data_table(DataSet As Variant, nrows As Long,
ncols As Long, _
```

```
    row_names() As String, col_names() As String, row_desc() As
String, col_desc() As String) As DataTable
' Creates a mixed table with variant array as data set
' - returns empty table if there was an error
' - error if the size of DataSet doesn't match given nrows and ncols
' - error if other arrays row or column names, row or column
'   descriptions doesn't match corresponding nrows/
```

61. Check if data table is consistent

```
Function FQ_check_if_table_consistent(DTable As DataTable) As
Boolean
' Checks the consistence of a data table; returns true if it is
consistent
```

62. Add comment to a worksheet cell

```
Sub FQ_add_comment_to_cell(Rn As Range, Comment As String)
' Adds a comment to given cell (upper-left cell of a range)
' - overwrites an existing comment
```

63. Insert table into a worksheet range

```
Sub FQ_table_to_range(DTable As DataTable, Rn As Range,
IfAddComments As Boolean)
' Inserts table into the given range, starting from top-left cell
' - Inserts table only if it is consistent
' - Adds comments to row/column names if IfAddComments = True
' - Checks consistency of description arrays even if IfAddComments =
False
' - error if size of DataSet doesnot match nrows/ncols
' - error if other arrays row or column names, row or column
'   descriptions doesn't match corresponding nrows/ncols
```

64. Determinant of a matrix

```
Function FQ_determinant(M() As Double) As Double
' Determinant of a square matrix
' - error if M is not a matrix
```