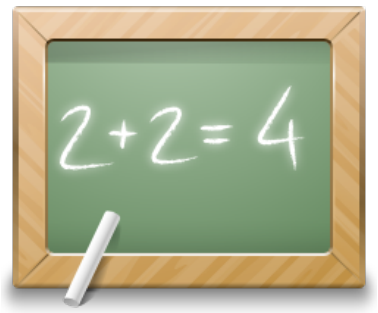
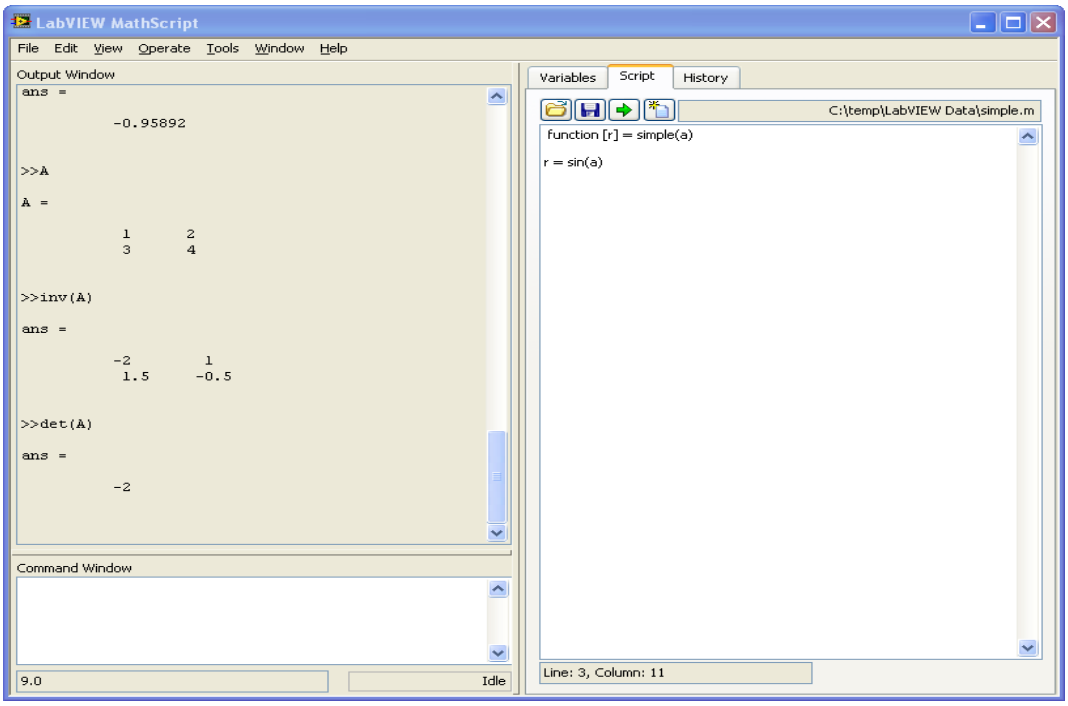
https://www.halvorsen.blog



Linear Algebra in LabVIEW

Hans-Petter Halvorsen, 2018-04-24



Preface

This document explains the basic concepts of Linear Algebra and how you may use LabVIEW for calculation of these problems.

This document is available for download from:

https://www.halvorsen.blog/documents/tutorials

For more information about LabVIEW, visit my Blog:

https://www.halvorsen.blog

and:

https://www.halvorsen.blog/documents/programming/labview/

Table of Contents

1 Introduction to LabVIEW

LabVIEW (short for **Lab**oratory **V**irtual **I**nstrumentation **E**ngineering **W**orkbench) is a platform and development environment for a visual programming language from National Instruments. The graphical language is named "G". Originally released for the Apple Macintosh in 1986, LabVIEW is commonly used for data acquisition, instrument control, and industrial automation on a variety of platforms including Microsoft Windows, various flavors of UNIX, Linux, and Mac OS X. Visit National Instruments at www.ni.com.

The code files have the extension ".vi", which is an abbreviation for "Virtual Instrument". LabVIEW offers lots of additional Add-Ons and Toolkits.

This paper is part of a series with LabVIEW papers:

- Introduction to LabVIEW
- Linear Algebra in LabVIEW
- Data Acquisition and Instrument Control in LabVIEW
- Control Design and Simulation in LabVIEW
- Signal Processing in LabVIEW
- Datalogging and Supervisory Control in LabVIEW
- System identification in LabVIEW
- Model based Control in LabVIEW
- Advanced Topics in LabVIEW

Each paper may be used independently of each other.

1.1 Dataflow programming

The programming language used in LabVIEW, also referred to as G, is a dataflow programming language. Execution is determined by the structure of a graphical block diagram (the LV-source code) on which the programmer connects different function-nodes by drawing wires. These wires propagate variables and any node can execute as soon as all its input data become available. Since this might be the case for multiple nodes simultaneously, G is inherently capable of parallel execution. Multi-processing and multi-threading hardware is automatically exploited by the built-in scheduler, which multiplexes multiple OS threads over the nodes ready for execution.

1.2 Graphical Programming

LabVIEW ties the creation of user interfaces (called front panels) into the development cycle. LabVIEW programs/subroutines are called virtual instruments (VIs). Each VI has three components: a block diagram, a front panel, and a connector panel. The last is used to represent the VI in the block diagrams of other, calling VIs. Controls and indicators on the front panel allow an operator to input data into or extract data from a running virtual instrument. However, the front panel can also serve as a programmatic interface. Thus a virtual instrument can either be run as a program, with the front panel serving as a user interface, or, when dropped as a node onto the block diagram, the front panel defines the inputs and outputs for the given node through the connector pane. This implies each VI can be easily tested before being embedded as a subroutine into a larger program.

The graphical approach also allows non-programmers to build programs simply by dragging and dropping virtual representations of lab equipment with which they are already familiar. The LabVIEW programming environment, with the included examples and the documentation, makes it simple to create small applications. This is a benefit on one side, but there is also a certain danger of underestimating the expertise needed for good quality "G" programming. For complex algorithms or large-scale code, it is important that the programmer possess an extensive knowledge of the special LabVIEW syntax and the topology of its memory management. The most advanced LabVIEW development systems offer the possibility of building stand-alone applications. Furthermore, it is possible to create distributed applications, which communicate by a client/server scheme, and are therefore easier to implement due to the inherently parallel nature of G-code.

1.3 Benefits

One benefit of LabVIEW over other development environments is the extensive support for accessing instrumentation hardware. Drivers and abstraction layers for many different types of instruments and buses are included or are available for inclusion. These present themselves as graphical nodes. The abstraction layers offer standard software interfaces to communicate with hardware devices. The provided driver interfaces save program development time. The sales pitch of National Instruments is, therefore, that even people with limited coding experience can write programs and deploy test solutions in a reduced time frame when compared to more conventional or competing systems. A new hardware driver topology (DAQmxBase), which consists mainly of G-coded components with only a few register calls through NI Measurement Hardware DDK (Driver Development Kit) functions, provides platform independent hardware access to numerous data acquisition

and instrumentation devices. The DAQmxBase driver is available for LabVIEW on Windows, Mac OS X and Linux platforms.

For more information about LabVIEW, visit my Blog: https://www.halvorsen.blog

1.4 LabVIEW MathScript RT Module

The LabVIEW MathScript RT Module is an add-on module to LabVIEW. With LabVIEW MathScript RT Module you can:

- Deploy your custom .m files to NI real-time hardware
- Reuse many of your scripts created with The MathWorks, Inc. MATLAB® software and others
- Develop your .m files with an interactive command-line interface
- Embed your scripts into your LabVIEW applications using the MathScript Node

2 Introduction to Linear Algebra

Given a matrix *A*:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix} \in R^{nxm}$$

Example:

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

2.1.1 Transpose

The **Transpose** of matrix *A*:

$$A^T = \begin{bmatrix} 0 & -2 \\ 1 & -3 \end{bmatrix}$$

2.1.2 Diagonal

The **Diagonal** elements of matrix *A* is the vector

$$diag(A) = \begin{bmatrix} a_{11} \\ a_{22} \\ \vdots \\ a_{pp} \end{bmatrix} \in R^{p=\min(x,m)}$$

The **Diagonal** matrix *Λ* is given by:

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \in R^{nxn}$$

Given the **Identity** matrix *I*:

$$I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \in R^{nxm}$$

2.1.3 Matrix Multiplication

Given the matrices $A \in R^{nxm}$ and $B \in R^{mxp}$, then

$$C = AB \in R^{nxp}$$

2.1.4 Matrix Addition

Given the matrices $A \in R^{nxm}$ and $B \in R^{nxm}$, then

$$C = A + B \in R^{nxm}$$

2.1.5 Determinant

Given a matrix $A \in R^{nxn}$, then the **Determinant** is given:

$$\det(A) = |A|$$

Given a *2x2* matrix

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \in R^{2x2}$$

Then

$$\det(A) = |A| = a_{11}a_{22} - a_{21}a_{12}$$

Notice that

$$\det(AB) = \det(A)\det(B)$$

and

$$\det(A^T) = \det(A)$$

2.1.6 Inverse Matrices

The **inverse** of a quadratic matrix $A \in R^{nxn}$ is defined by:

$$A^{-1}$$

if

$$AA^{-1} = A^{-1}A = I$$

For a *2x2* matrix we have:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \in R^{2x2}$$

The inverse A^{-1} is given by

$$A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix} \in R^{2x2}$$

2.2 Eigenvalues

Given $A \in R^{nxn}$, then the Eigenvalues is defined as:

$$\det(\lambda I - A) = 0$$

2.3 Solving Linear Equations

Given the linear equation

$$Ax = b$$

with the solution:

$$x = A^{-1}b$$

(Assuming that the inverse of *A* exists)

Example:

The equations

$$x_1 + 2x_2 = 5$$
$$3x_1 + 4x_2 = 6$$

may be written

$$Ax = b$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$

where

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$b = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$

2.4 LU factorization

LU factorization of $A \in R^{nxm}$ is given by

$$A = LU$$

where

L is a lower triangular matrix

U is a upper triangular matrix

Or sometimes LU factorization of $A \in R^{nxm}$ is given by

$$A = LU = LDU$$

where

D is a diagonal matrix

2.5 The Singular Value Decomposition (SVD)

The **Singular value Decomposition** (SVD) of the matrix $A \in R^{nxm}$ is given by

$$A = USV^T$$

where

U is a orthogonal matrix

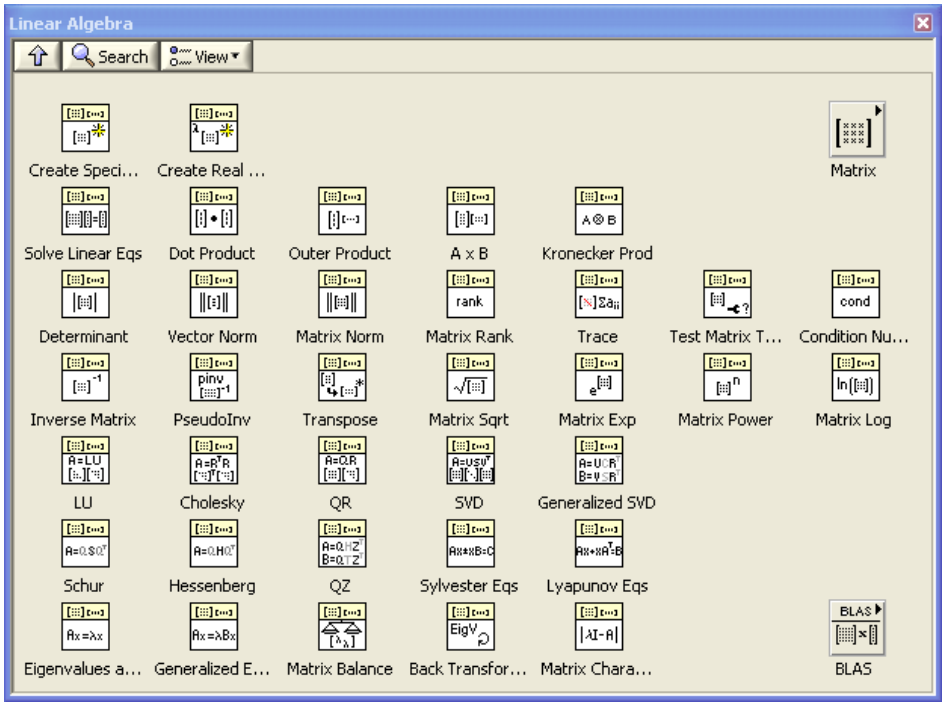
V is a orthogonal matrix

S is a diagonal singular matrix

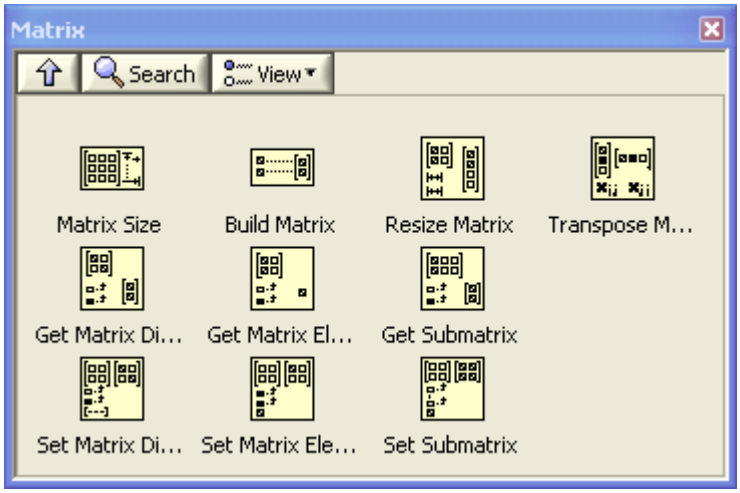
3 Linear Algebra Palette in LabVIEW

For an Introduction to LabVIEW, see the training: "**An Introduction to LabVIEW**". You may download it from my Blog: http://home.hit.no/~hansha/

Use the Linear Algebra Palette in order to solve Linear Algebra problems with the use of Graphical programming.

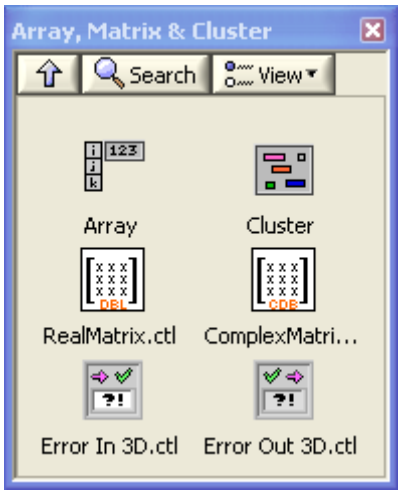


In the Matrix Sub Palette we have the following functions:



LabVIEW uses arrays to represents vectors and matrices. A vector is represented as a one dimensional array, while a matrix is represented as a two dimensional array.

In the Array, Matrix & Cluster Palette available from the Front Panel, we have the basic array and matrix controls:



3.1 Vectors

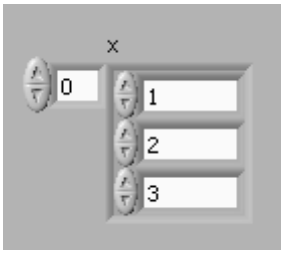
Given a vector *x*

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in R^n$$

Example: Vectors

$$x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Implementing a vector in the Front Panel:



3.2 Matrices

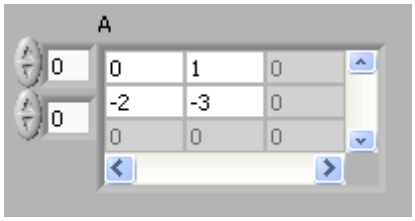
Given a matrix *A*:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix} \in R^{nxm}$$

Example: Matrices

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

Front Panel:



3.2.1 Transpose

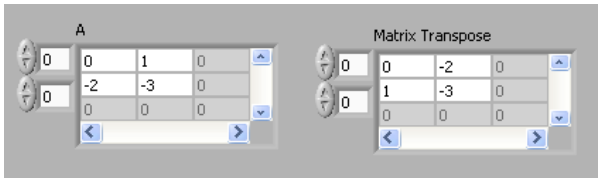
The **Transpose** of matrix *A*:

$$A^T = \begin{bmatrix} a_{11} & \cdots & a_{n1} \\ \vdots & \ddots & \vdots \\ a_{1m} & \cdots & a_{nm} \end{bmatrix} \in R^{mxn}$$

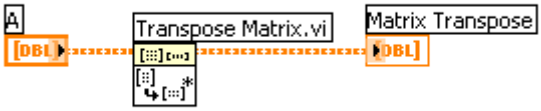
Example: Transpose

$$A^T = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}^T = \begin{bmatrix} 0 & -2 \\ 1 & -3 \end{bmatrix}$$

Front Panel:



Block Diagram:



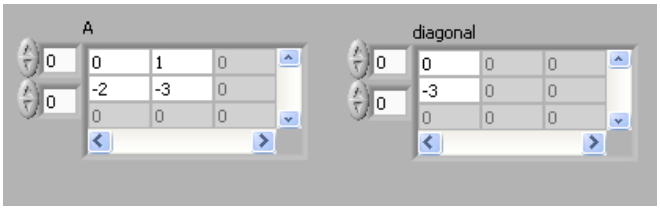
3.2.2 Diagonal

The **Diagonal** elements of matrix *A* is the vector

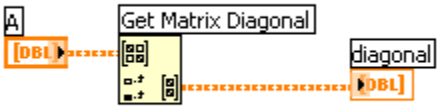
$$diag(A) = \begin{bmatrix} a_{11} \\ a_{22} \\ \vdots \\ a_{pp} \end{bmatrix} \in R^{p=\min(x,m)}$$

Example: Diagonal

Front Panel:



Block Diagram:



The **Diagonal** matrix *Λ* is given by:

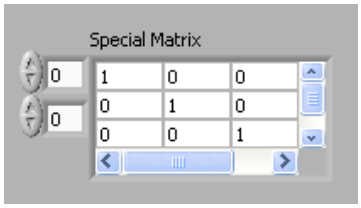
$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \in R^{nxn}$$

Given the **Identity** matrix *I*:

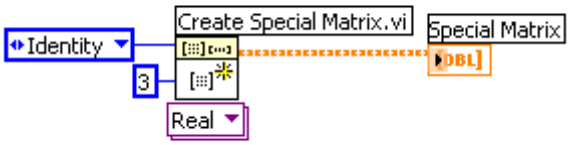
$$I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \in R^{nxm}$$

Example: Identity Matrix

Front Panel:



Block Diagram:



3.2.3 Matrix Multiplication

Given the matrices $A \in R^{nxm}$ and $B \in R^{mxp}$, then

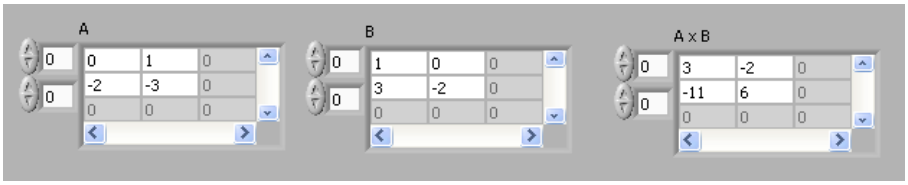
$$C = AB \in R^{nxp}$$

where

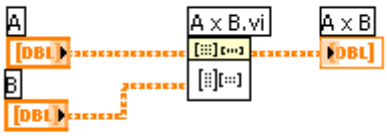
$$c_{jk} = \sum_{l=1}^{n} a_{jl} b_{lk}$$

Example: Matrix Multiplication

Front Panel:



Block Diagram:



Note!

$$AB \neq BA$$

$$A(BC) = (AB)C$$

$$(A + B)C = AC + BC$$

$$C(A + B) = CA + CB$$

→ Prove this in LabVIEW

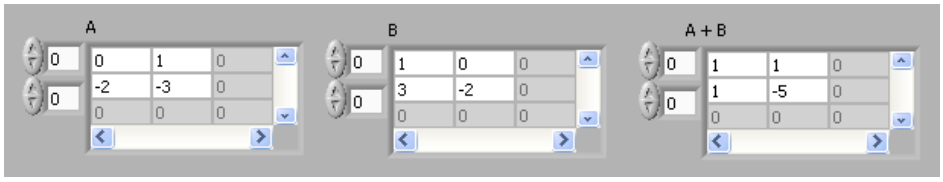
3.2.4 Matrix Addition

Given the matrices $A \in R^{nxm}$ and $B \in R^{nxm}$, then

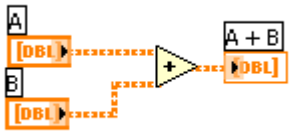
$$C = A + B \in R^{nxm}$$

Example: Matrix Addition

Front Panel:



Block Diagram:



Note! There is no special function for matrix addition, just use the standard add function in the Numeric palette.

3.2.5 Determinant

Given a matrix $A \in R^{nxn}$, then the **Determinant** is given:

$$\det(A) = |A|$$

Given a *2x2* matrix

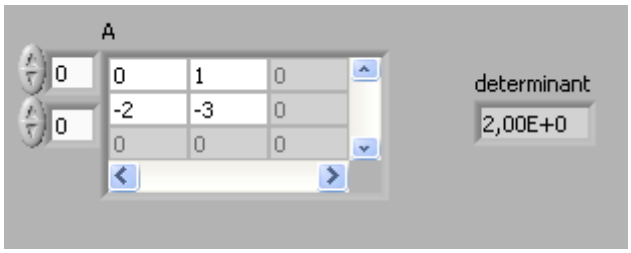
$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \in R^{2x2}$$

Then

$$\det(A) = |A| = a_{11}a_{22} - a_{21}a_{12}$$

Example: Determinant

Front Panel:



Block Diagram:

Notice that

$$\det(AB) = \det(A)\det(B)$$

and

$$\det(A^T) = \det(A)$$

→ Prove this in LabVIEW

3.2.6 Inverse Matrices

The **inverse** of a quadratic matrix $A \in R^{nxn}$ is defined by:

$$A^{-1}$$

if

$$AA^{-1} = A^{-1}A = I$$

For a *2x2* matrix we have:

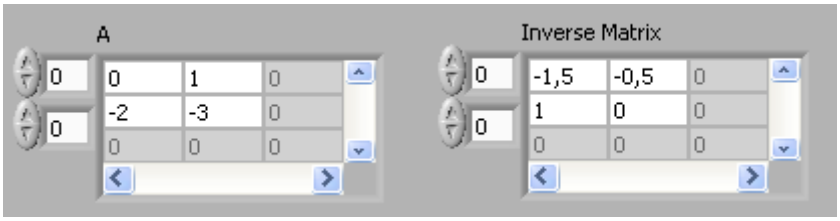
$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \in R^{2x2}$$

The inverse A^{-1} is given by

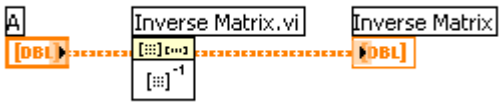
$$A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix} \in R^{2x2}$$

Example: Inverse

Front Panel:



Block Diagram:



Notice that:

$$AA^{-1} = A^{-1}A = I$$

→ Prove this in LabVIEW

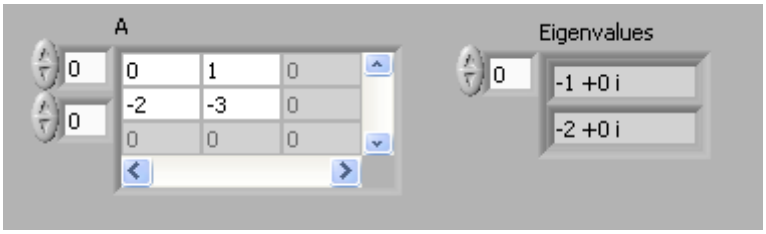
3.3 Eigenvalues

Given $A \in R^{nxn}$, then the Eigenvalues is defined as:

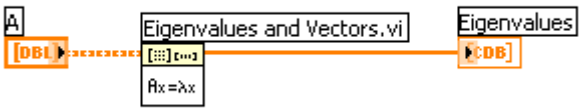
$$\det(\lambda I - A) = 0$$

Example: Eigenvalues

Front Panel:



Block Diagram:



3.4 Solving Linear Equations

Given the linear equation

$$Ax = b$$

with the solution:

$$x = A^{-1}b$$

(Assuming that the inverse of *A* exists)

Example: Solving Linear Equations

The equations

$$x_1 + 2x_2 = 5$$
$$3x_1 + 4x_2 = 6$$

may be written

$$Ax = b$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$

where

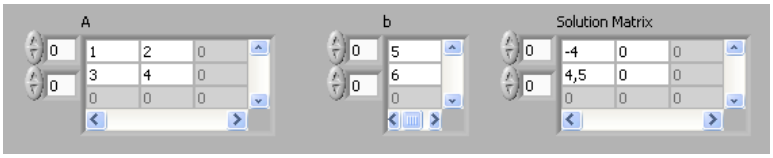
$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

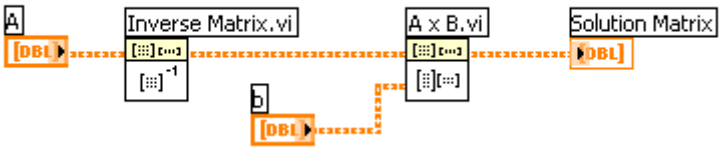
$$b = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$

The solution is:

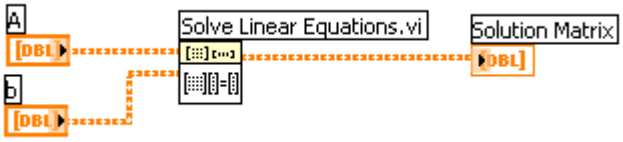
Front Panel:



Block Diagram:



Or:



3.5 LU factorization

LU factorization of $A \in R^{nxm}$ is given by

$$A = LU$$

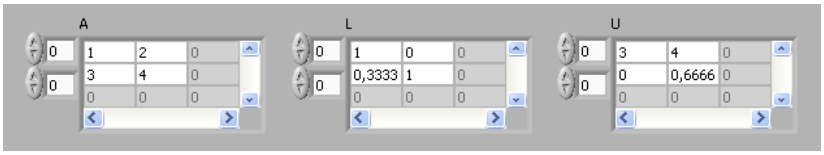
where

L is a lower triangular matrix

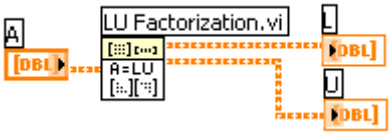
U is a upper triangular matrix

Example: LU Factorization

Front Panel:



Block Diagram:



Or sometimes LU factorization of $A \in R^{nxm}$ is given by

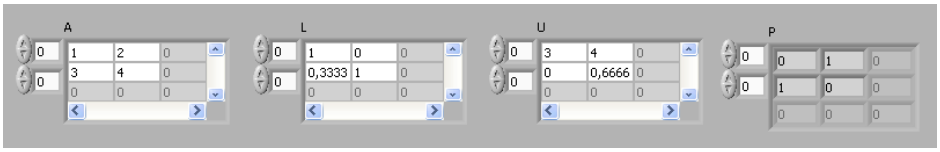
$$A = LU = LDU$$

where

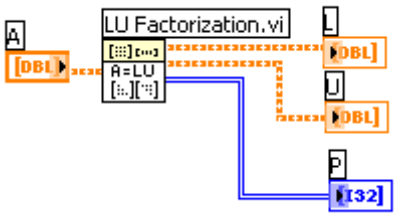
D is a diagonal matrix

Example: LU Factorization

Front Panel:



Block Diagram:



3.6 The Singular Value Decomposition (SVD)

The **Singular value Decomposition** (SVD) of the matrix $A \in R^{nxm}$ is given by

$$A = USV^T$$

where

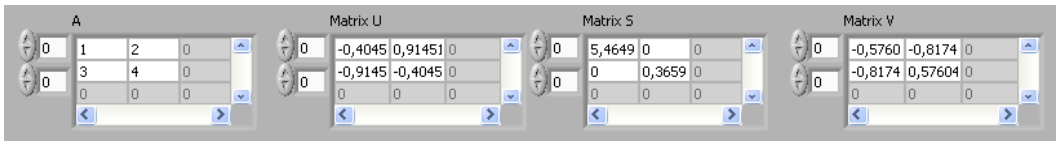
U is a orthogonal matrix

V is a orthogonal matrix

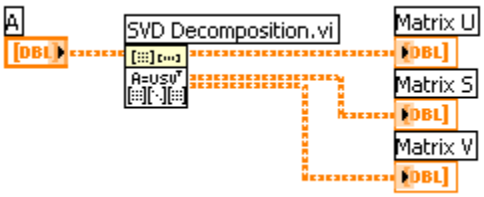
S is a diagonal singular matrix

Example: SVD Decomposition

Front Panel:



Block Diagram:



4 LabVIEW MathScript RT Module

You can work with LabVIEW MathScript through either of two interfaces: the “LabVIEW MathScript Interactive Window” or the “MathScript Node”.

You can work with LabVIEW MathScript RT Module through both interactive and programmatic interfaces. For an interactive interface in which you can load, save, design, and execute your .m file scripts, you can work with the “MathScript Interactive Window”. To deploy your .m file scripts as part of a LabVIEW application and combine graphical and textual programming, you can work with the “MathScript Node”.

The LabVIEW MathScript RT Module complements traditional LabVIEW graphical programming for such tasks as algorithm development, signal processing, and analysis. The LabVIEW MathScript RT Module speeds up these and other tasks by giving users a single environment in which they can choose the most effective syntax, whether textual, graphical, or a combination of the two. In addition, you can exploit the best of LabVIEW and thousands of publicly available .m file scripts from the web, textbooks, or your own existing m-script applications. LabVIEW MathScript RT Module is able to process your files created using the current MathScript syntax and, for backwards compatibility, files created using legacy MathScript syntaxes. LabVIEW MathScript RT Module can also process certain of your files utilizing other text-based syntaxes, such as files you created using MATLAB software. Because the MathScript RT engine is used to process scripts contained in a MathScript Windows or MathScript Node, and because the MathScript RT engine does not support all syntaxes, not all existing text-based scripts are supported.

LabVIEW MathScript RT Module supports most of the functionality available in MATLAB, the syntax is also similar.

For more details, see http://zone.ni.com/devzone/cda/tut/p/id/3257

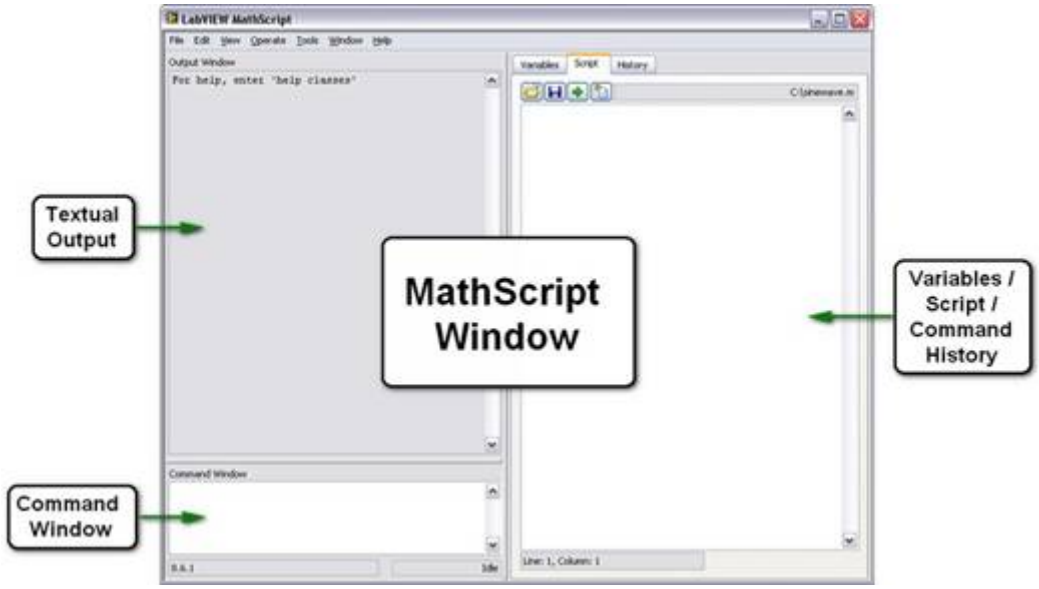
5 LabVIEW MathScript

Requires: **MathScript RT Module**

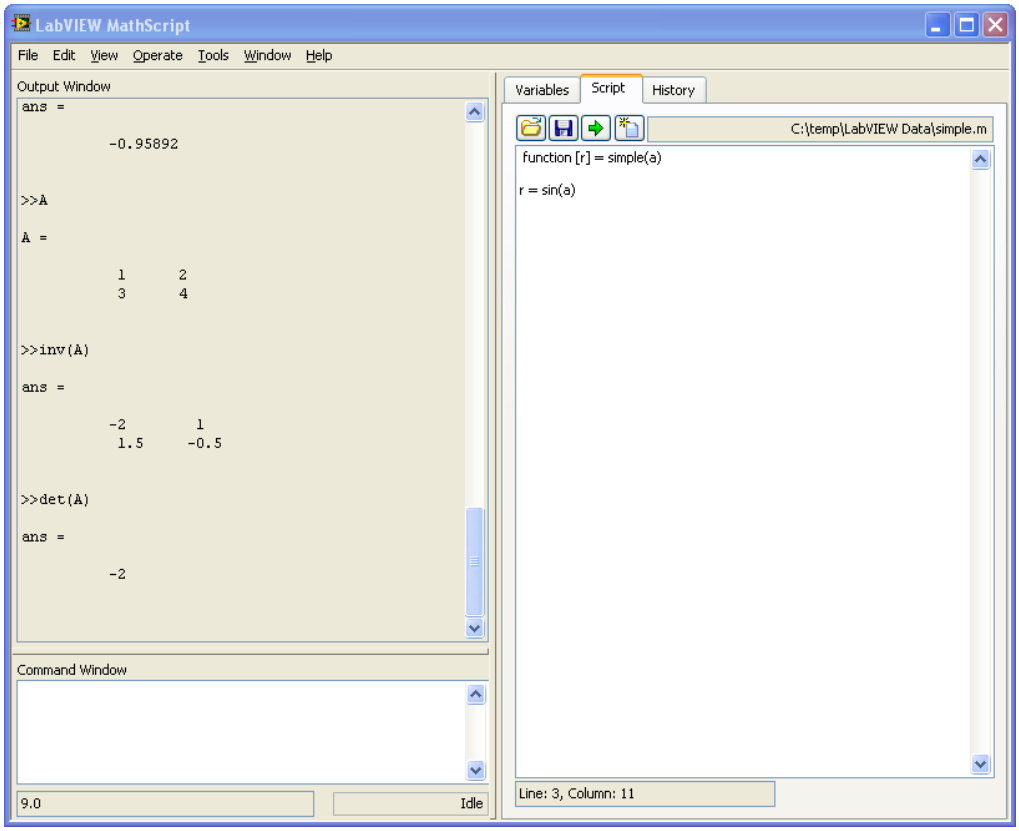
The “LabVIEW MathScript Window” is an interactive interface in which you can enter .m file script commands and see immediate results, variables and commands history. The window includes a command-line interface where you can enter commands one-by-one for quick calculations, script debugging or learning. Alternatively, you can enter and execute groups of commands through a script editor window.

As you work, a variable display updates to show the graphical / textual results and a history window tracks your commands. The history view facilitates algorithm development by allowing you to use the clipboard to reuse your previously executed commands.

You can use the “LabVIEW MathScript Window” to enter commands one at time. You also can enter batch scripts in a simple text editor window, loaded from a text file, or imported from a separate text editor. The “LabVIEW MathScript Window” provides immediate feedback in a variety of forms, such as graphs and text.



Example:



5.1 Help

You may also type help in your command window

```
>>help
```

Or more specific, e.g.,

```
>>help plot
```

5.2 Examples

I advise you to test all the examples in this text in LabVIEW MathScript in order to get familiar with the program and its syntax. All examples in the text are outlined in a frame like this:

```
>>
...
```

This is commands you should write in the **Command Window**.

You type all your commands in the Command Window. I will use the symbol "**>>**" to illustrate that the commands should be written in the Command Window.

Example: Matrices

Defining the following matrix

$$A = \begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix}$$

The syntax is as follows:

```
>> A = [1 2;0 3]
```

Or

```
>> A = [1,2;0,3]
```

If you, for an example, want to find the answer to

$$a + b, where\ a = 4, b = 3$$

```
>>a=4
>>b=3
>>a+b
```

MathScript then responds:

```
ans =
      7
```

MathScript provides a simple way to define simple arrays using the syntax: "**init:increment:terminator**". For instance:

```
>> array = 1:2:9
array =
 1 3 5 7 9
```

defines a variable named array (or assigns a new value to an existing variable with the name array) which is an array consisting of the values 1, 3, 5, 7, and 9. That is, the array starts at 1 (the init value), increments with each step from the previous value by 2 (the increment value), and stops once it reaches (or to avoid exceeding) 9 (the terminator value).

The increment value can actually be left out of this syntax (along with one of the colons), to use a default value of 1.

```
>> ari = 1:5
ari =
```

```
 1 2 3 4 5
```

assigns to the variable named ari an array with the values 1, 2, 3, 4, and 5, since the default value of 1 is used as the incrementer.

Note that the indexing is one-based, which is the usual convention for matrices in mathematics. This is atypical for programming languages, whose arrays more often start with zero.

Matrices can be defined by separating the elements of a row with blank space or comma and using a semicolon to terminate each row. The list of elements should be surrounded by square brackets: **[]**. Parentheses: **()** are used to access elements and subarrays (they are also used to denote a function argument list).

```
>> A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
A =
 16  3  2 13
  5 10 11  8
  9  6  7 12
  4 15 14  1
>> A(2,3)
ans =
 11
```

Sets of indices can be specified by expressions such as "2:4", which evaluates to [2, 3, 4]. For example, a submatrix taken from rows 2 through 4 and columns 3 through 4 can be written as:

```
>> A(2:4,3:4)
ans =
 11 8
 7 12
 14 1
```

A square identity matrix of size n can be generated using the function eye, and matrices of any size with zeros or ones can be generated with the functions zeros and ones, respectively.

```
>> eye(3)
ans =
 1 0 0
 0 1 0
 0 0 1
>> zeros(2,3)
ans =
 0 0 0
 0 0 0
>> ones(2,3)
ans =
 1 1 1
```

```
1 1 1
```

5.3 Useful commands

Here are some useful commands:

Command	Description
eye(x), eye(x,y)	Identity matrix of order x
ones(x), ones(x,y)	A matrix with only ones
zeros(x), zeros(x,y)	A matrix with only zeros
diag([x y z])	Diagonal matrix
size(A)	Dimension of matrix A
A'	Inverse of matrix A

5.4 Flow Control

This chapter explains the basic concepts of flow control in MathScript.

The topics are as follows:

- If-else statement
- Switch and case statement
- For loop
- While loop

5.4.1 If-else Statement

The if statement evaluates a logical expression and executes a group of statements when the expression is true. The optional elseif and else keywords provide for the execution of alternate groups of statements. An end keyword, which matches the if, terminates the last group of statements. The groups of statements are delineated by the four keywords—no braces or brackets are involved.

Example: If-Else Statement

Test the following code:

```
n=5
if n > 2
    M = eye(n)
elseif n < 2
    M = zeros(n)
else
    M = ones(n)
end
```

5.4.2 Switch and Case Statement

The switch statement executes groups of statements based on the value of a variable or expression. The keywords case and otherwise delineate the groups. Only the first matching case is executed. There must always be an end to match the switch.

Example: Switch and Case Statement

Test the following code:

```
n=2
switch(n)
    case 1
        M = eye(n)
    case 2
        M = zeros(n)
    case 3
        M = ones(n)
end
```

5.4.3 For loop

The for loop repeats a group of statements a fixed, predetermined number of times. A matching end delineates the statements.

Example: For Loop

Test the following code:

```
m=5
for n = 1:m
    r(n) = rank(magic(n));
end
r
```

5.4.4 While loop

The while loop repeats a group of statements an indefinite number of times under control of a logical condition. A matching end delineates the statements.

Example: While Loop

Test the following code:

```
m=5;
while m > 1
    m = m - 1;
    zeros(m)
end
```

5.5 Plotting

This chapter explains the basic concepts of creating plots in MathScript.

Topics:

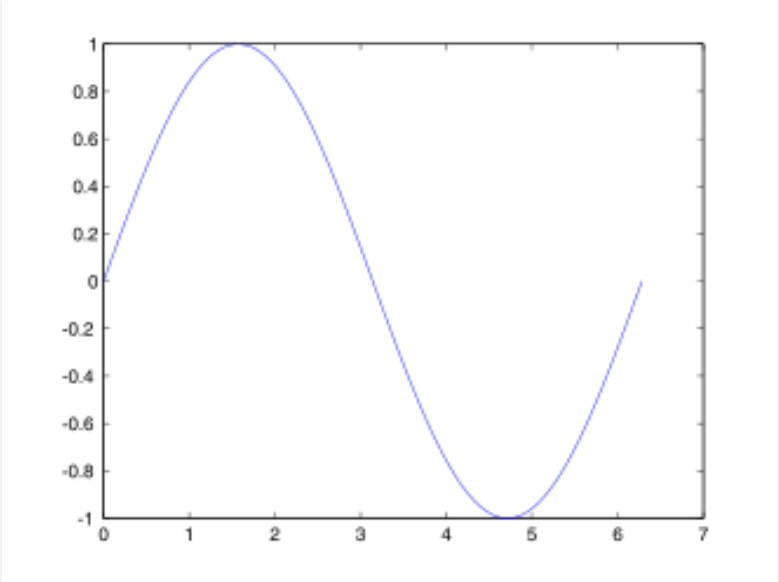
- Basic Plot commands

Example: Plotting

Function plot can be used to produce a graph from two vectors x and y. The code:

```
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y)
```

produces the following figure of the sine function:



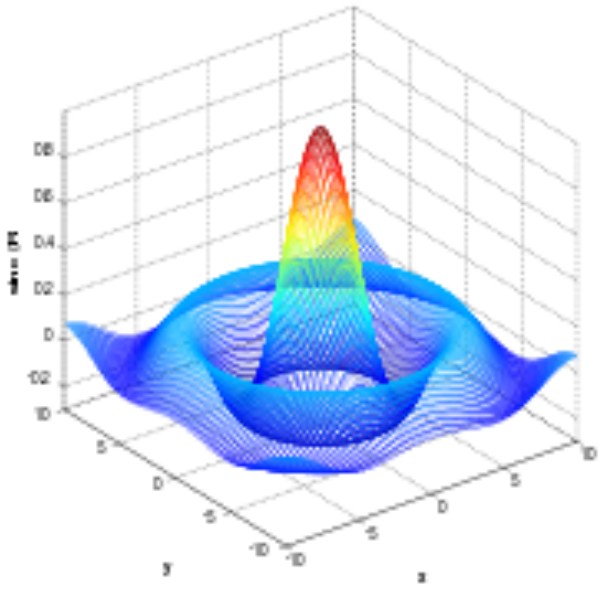
Example: Plotting

Three-dimensional graphics can be produced using the functions surf, plot3 or mesh.

```
[X,Y] = meshgrid(-10:0.25:10,-10:0.25:10);
f = sinc(sqrt((X/pi).^2+(Y/pi).^2));
mesh(X,Y,f);
axis([-10 10 -10 10 -0.3 1])
xlabel('{\bfx}')
ylabel('{\bfy}')
zlabel('{\bfsinc} ({\bfR})')
```

```
hidden off
```

This code produces the following 3D plot:



6 Linear Algebra Examples using MathScript

Requires: **MathScript RT Module**

Linear algebra is a branch of mathematics concerned with the study of matrices, vectors, vector spaces (also called linear spaces), linear maps (also called linear transformations), and systems of linear equations.

MathScript are well suited for Linear Algebra.

6.1 Vectors

Given a vector *x*

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in R^n$$

Example: Vectors

Given the following vector

$$x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
>> x=[1; 2; 3]
x =
     1
     2
     3
```

The **Transpose** of vector *x*:

$$x^T = [x_1 \quad x_2 \quad \cdots \quad x_n] \in R^{1xn}$$

```
>> x'
ans =
     1     2     3
```

The **Length** of vector *x*:

$$\|x\| = \sqrt{x^T x} = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

Orthogonality:

$$x^T y = 0$$

6.2 Matrices

Given a matrix *A*:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix} \in R^{nxm}$$

Example: Matrices

Given the following matrix:

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

```
>> A=[0 1;-2 -3]
A =
     0     1
    -2    -3
```

6.2.1 Transpose

The **Transpose** of matrix *A*:

$$A^T = \begin{bmatrix} a_{11} & \cdots & a_{n1} \\ \vdots & \ddots & \vdots \\ a_{1m} & \cdots & a_{nm} \end{bmatrix} \in R^{mxn}$$

Example: Transpose

Given the matrix:

$$A^T = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}^T = \begin{bmatrix} 0 & -2 \\ 1 & -3 \end{bmatrix}$$

```
>> A'
ans =
     0    -2
     1    -3
```

6.2.2 Diagonal

The **Diagonal** elements of matrix *A* is the vector

$$diag(A) = \begin{bmatrix} a_{11} \\ a_{22} \\ \vdots \\ a_{pp} \end{bmatrix} \in R^{p=\min(x,m)}$$

Example: Diagonal

Find the diagonal elements of matrix *A*:

```
>> diag(A)
ans =
     0
    -3
```

The **Diagonal** matrix *Λ* is given by:

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \in R^{nxn}$$

Given the **Identity** matrix *I*:

$$I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \in R^{nxm}$$

Example: Identity Matrix

Get the 3x3 Identity matrix:

```
>> eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1
```

6.2.3 Triangular

Lower Triangular matrix *L*:

$$L = \begin{bmatrix} . & 0 & 0 \\ \vdots & \ddots & 0 \\ . & \cdots & . \end{bmatrix}$$

Upper Triangular matrix *U*:

$$U = \begin{bmatrix} \cdot & \cdots & \cdot \\ 0 & \ddots & \vdots \\ 0 & 0 & \cdot \end{bmatrix}$$

6.2.4 Matrix Multiplication

Given the matrices $A \in R^{nxm}$ and $B \in R^{mxp}$, then

$$C = AB \in R^{nxp}$$

where

$$c_{jk} = \sum_{l=1}^{n} a_{jl} b_{lk}$$

Example: Matrix Multiplication

Matrix multiplication:

```
>> A=[0 1;-2 -3]
A =
     0     1
    -2    -3
>> B=[1 0;3 -2]
B =
     1     0
     3    -2
>> A*B
ans =
     3    -2
   -11     6
```

Note!

$$AB \neq BA$$

$$A(BC) = (AB)C$$

$$(A + B)C = AC + BC$$

$$C(A + B) = CA + CB$$

6.2.5 Matrix Addition

Given the matrices $A \in R^{nxm}$ and $B \in R^{nxm}$, then

$$C = A + B \in R^{nxm}$$

Example: Matrix Addition

Matrix addition:

```
>> A=[0 1;-2 -3]
>> B=[1 0;3 -2]
>> A+B
ans =
     1     1
     1    -5
```

6.2.6 Determinant

Given a matrix $A \in R^{nxn}$, then the **Determinant** is given:

$$\det(A) = |A|$$

Given a *2x2* matrix

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \in R^{2x2}$$

Then

$$\det(A) = |A| = a_{11}a_{22} - a_{21}a_{12}$$

Example: Determinant

Find the determinant:

```
A =
     0     1
    -2    -3
>> det(A)
ans =
     2
```

Notice that

$$\det(AB) = \det(A)\det(B)$$

and

$$\det(A^T) = \det(A)$$

Example: Determinant

Determinants:

```
>> det(A*B)
ans =
    -4
>> det(A)*det(B)
ans =
    -4
>> det(A')
ans =
     2
>> det(A)
ans =
     2
```

6.2.7 Inverse Matrices

The **inverse** of a quadratic matrix $A \in R^{nxn}$ is defined by:

$$A^{-1}$$

if

$$AA^{-1} = A^{-1}A = I$$

For a *2x2* matrix we have:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \in R^{2x2}$$

The inverse A^{-1} is given by

$$A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix} \in R^{2x2}$$

Example: Inverse Matrices

Inverse matrix:

```
A =
     0     1
    -2    -3
>> inv(A)
ans =
   -1.5000   -0.5000
    1.0000         0
```

Notice that:

$$AA^{-1} = A^{-1}A = I$$

→ Prove this in MathScript

6.3 Eigenvalues

Given $A \in R^{nxn}$, then the Eigenvalues is defined as:

$$\det(\lambda I - A) = 0$$

Example: Eigenvalues

Find the Eigenvalues:

```
A =
     0     1
    -2    -3
>> eig(A)
ans =
    -1
    -2
```

6.4 Solving Linear Equations

Given the linear equation

$$Ax = b$$

with the solution:

$$x = A^{-1}b$$

(Assuming that the inverse of *A* exists)

Example: Solving Linear Equations

Solving the following equation:

The equations

$$x_1 + 2x_2 = 5$$
$$3x_1 + 4x_2 = 6$$

may be written

$$Ax = b$$

$$\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}=\begin{bmatrix}5\\ 6\end{bmatrix}$$

where

$$A=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}$$

$$x=\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$

$$b=\begin{bmatrix}5\\ 6\end{bmatrix}$$

The solution is:

```
A =
     1     2
     3     4
>> b=[5;6]
b =
     5
     6
>> x=inv(A)*b
x =
   -4.0000
    4.5000
```

In MathScript you could also write *"x=A\b"*, which should give the same answer. This syntax can also be used when the inverse of A don't exists.

Example: Solving Linear Equations

Illegal operation:

```
>> A=[1 2;3 4;7 8]
>> x=inv(A)*b
??? Error using ==> inv
Matrix must be square.
>> x=A\b
x =
   -3.5000
    4.1786
```

6.5 LU factorization

LU factorization of $A \in R^{nxm}$ is given by

$$A = LU$$

where

L is a lower triangular matrix

U is a upper triangular matrix

The MathScript syntax is `[L,U]=lu(A)`

Example: LU Factorization

Find *L* and *U*:

```
>> A=[1 2;3 4]
>> [L,U]=lu(A)
L =
    0.3333    1.0000
    1.0000         0
U =
    3.0000    4.0000
         0    0.6667
```

Or sometimes LU factorization of $A \in R^{nxm}$ is given by

$$A = LU = LDU$$

where

D is a diagonal matrix

The MathScript syntax is `[L,U,P]=lu(A)`

Example: LU Factorization

Find *L, U* and *P*:

```
>> A=[1 2;3 4]
A =
     1     2
     3     4
>> [L,U,P]=lu(A)
L =
    1.0000         0
    0.3333    1.0000
U =
    3.0000    4.0000
         0    0.6667
P =
     0     1
     1     0
```

6.6 The Singular Value Decomposition (SVD)

The **Singular value Decomposition** (SVD) of the matrix $A \in R^{nxm}$ is given by

$$A = USV^T$$

where

U is a orthogonal matrix

V is a orthogonal matrix

S is a diagonal singular matrix

Example: SVD Decomposition

Find *S*, *V* and *D*:

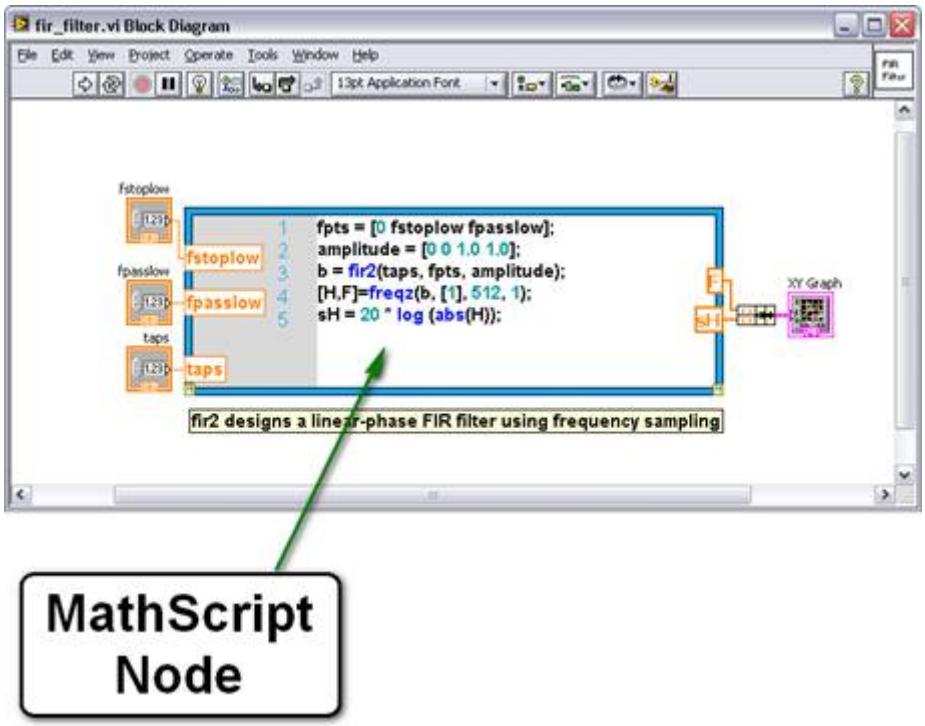
```
>> A=[1 2;3 4];
>> [U,S,V] = svd(A)
U =
   -0.4046   -0.9145
   -0.9145    0.4046
S =
    5.4650         0
         0    0.3660
V =
   -0.5760    0.8174
   -0.8174   -0.5760
```

6.7 Commands

Command	Description
`[L,U]=lu(A)` `[L,U,P]=lu(A)`	LU Factorization
`[U,S,V] = svd(A)`	Singular Value Decomposition (SVD)

7 MathScript Node

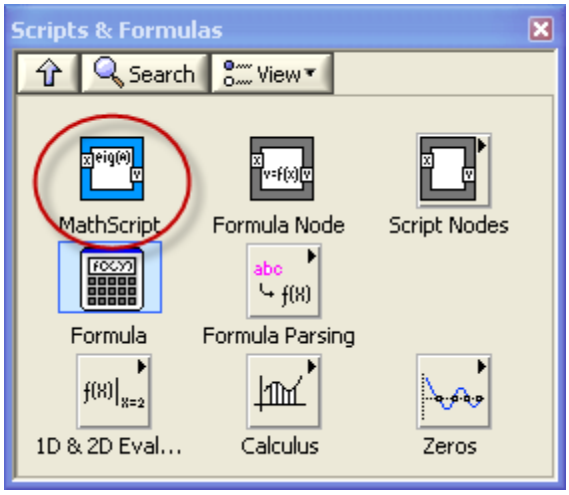
The “MathScript Node” offers an intuitive means of combining graphical and textual code within LabVIEW. The figure below shows the “MathScript Node” on the block diagram, represented by the blue rectangle. Using “MathScript Nodes”, you can enter .m file script text directly or import it from a text file.



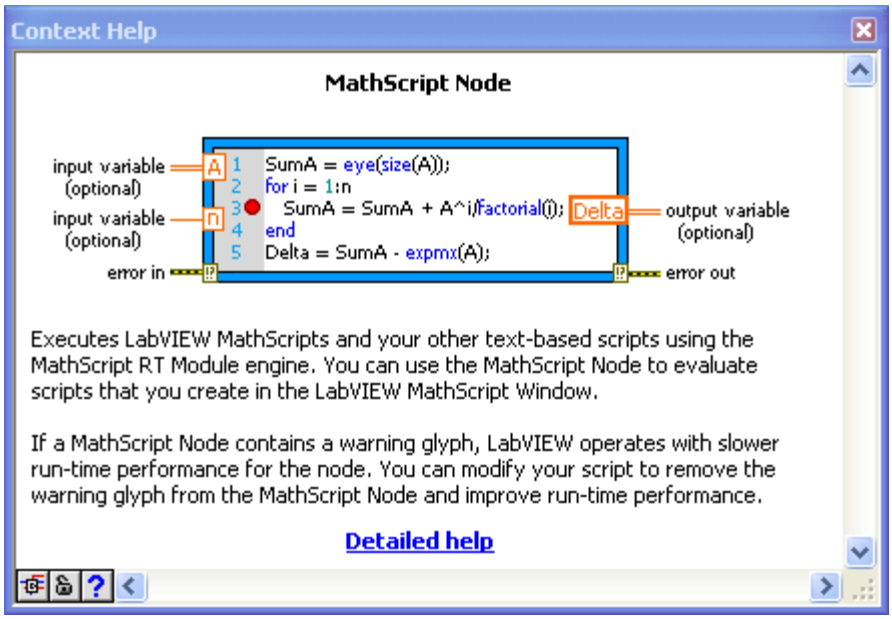
You can define named inputs and outputs on the MathScript Node border to specify the data to transfer between the graphical LabVIEW environment and the textual MathScript code.

You can associate .m file script variables with LabVIEW graphical programming, by wiring Node inputs and outputs. Then you can transfer data between .m file scripts with your graphical LabVIEW programming. The textual .m file scripts can now access features from traditional LabVIEW graphical programming.

The MathScript Node is available from LabVIEW from the Functions Palette: Mathematics → Scripts & Formulas

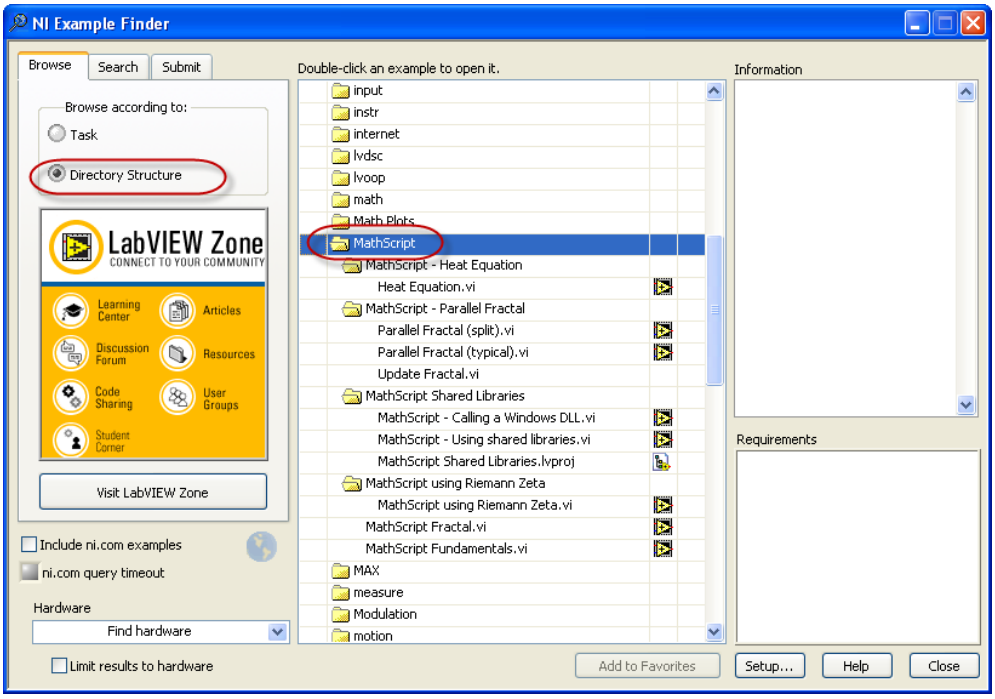


If you click Ctrl+H you get help about the MathScript Node:



Click “Detailed help” in order to get more information about the MathScript Node.

Use the NI Example Finder in order to find examples:



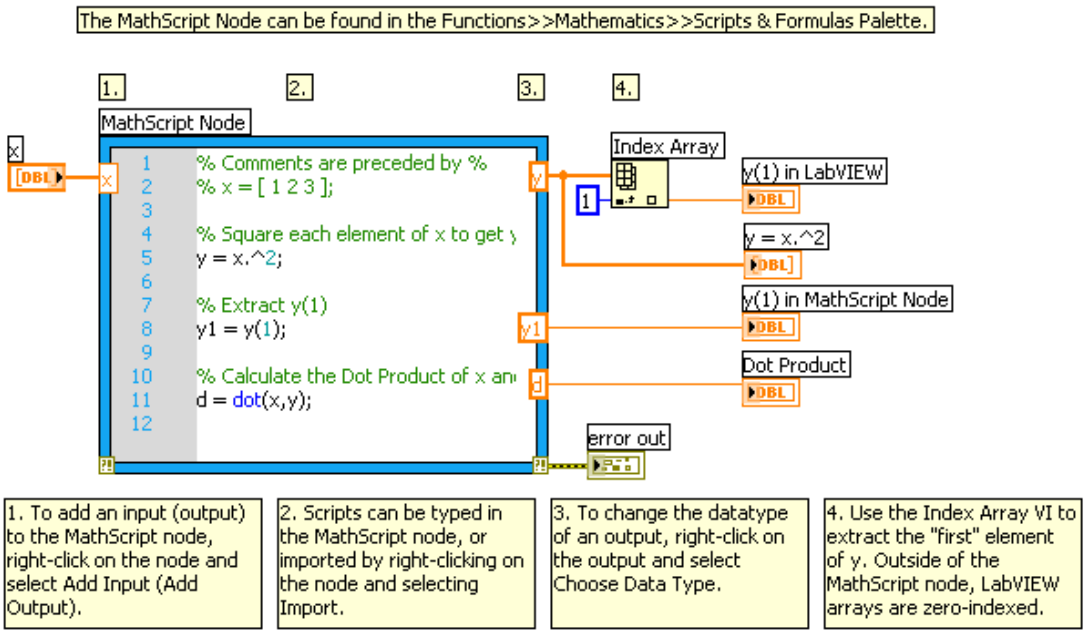
7.1 Transferring MathScript Nodes between Computers

If a script in a MathScript Node calls a user-defined function, LabVIEW uses the default search path list to link the function call to the specified .m file. After you configure the default search path list and save the VI that contains the MathScript Node, you do not need to reconfigure the MathScript search path list when you open the VI on a different computer because LabVIEW looks for the .m file in the directory where the .m file was located when you last saved the VI. However, you must maintain the same relative path between the VI and the .m file.

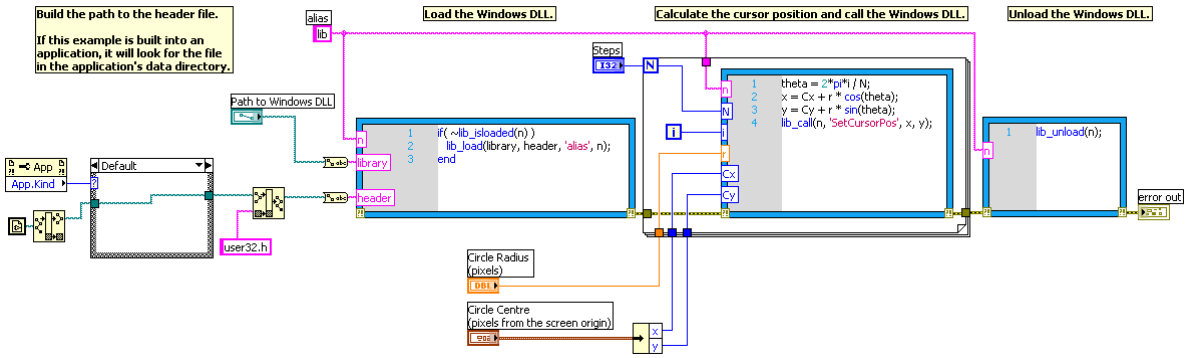
7.2 Examples

Example: Using the MathScript Node

Here is an example of how you use the MathScript Node. On the left border you connect input variables to the script, on the right border you have output variables. Right-click on the border and select “Add Input” or “Add Output”.

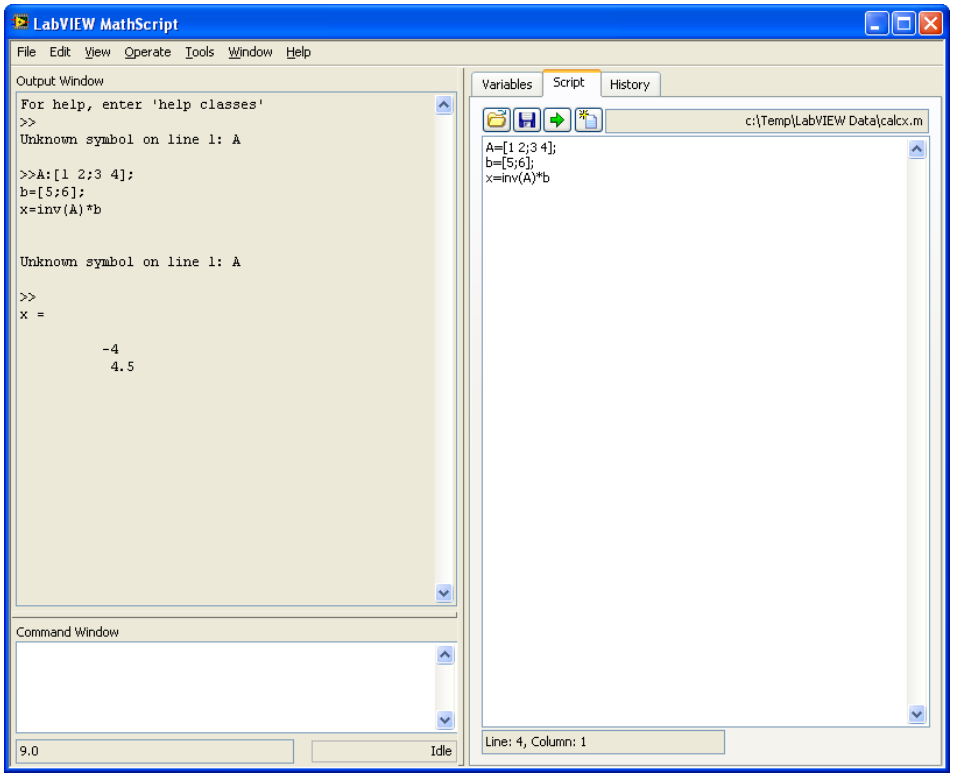


Example: Calling a Windows DLL:

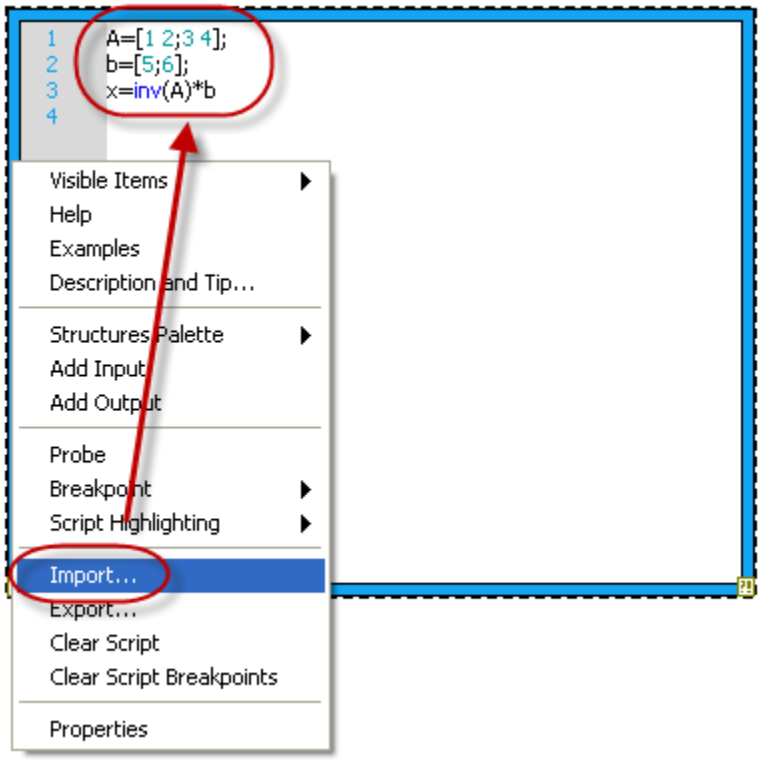


Example: Using m-files in the MathScript Node:

Use the LabVIEW MathScript to create a m-file script (or you may use MATLAB to create the same script):

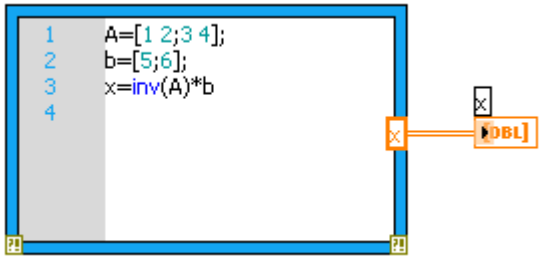


Right-click on the border of the MathScript Node and select “Import”, and then select the m-file you want to import into the Node.

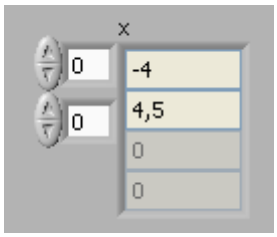


Right-click on the right border and select “Add Output”. Then right-click on the output variable and select “Create Indicator”.

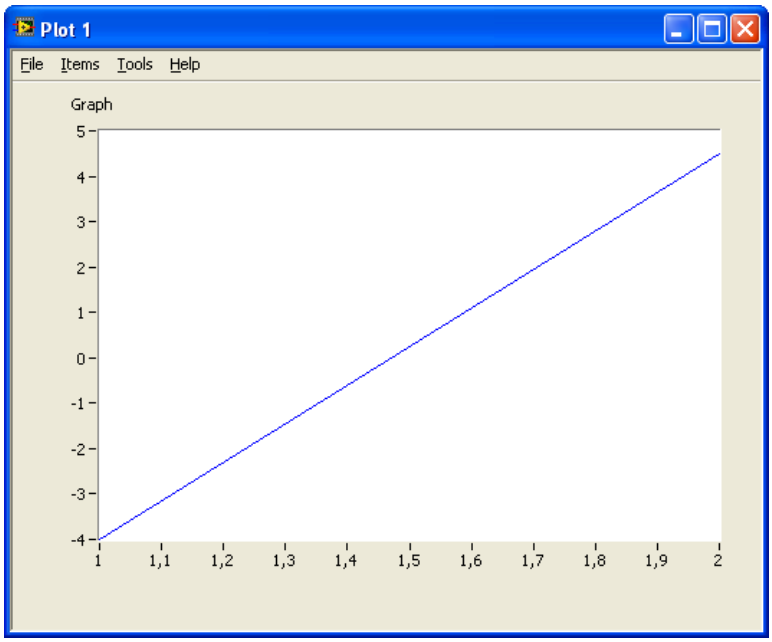
Block Diagram:



The result is as follows (click the Run button):



If you, e.g., add the following command in the MathScript Node: plot(x), the following window appears:



7.3 Exercises

Use the MathScript Node and test the same examples you did in the previous chapter (Chapter 6 - “Linear Algebra Examples using MathScript”)

8 Whats Next?

8.1 My Blog

For more information about LabVIEW, visit my Blog: https://www.halvorsen.blog

8.2 Training

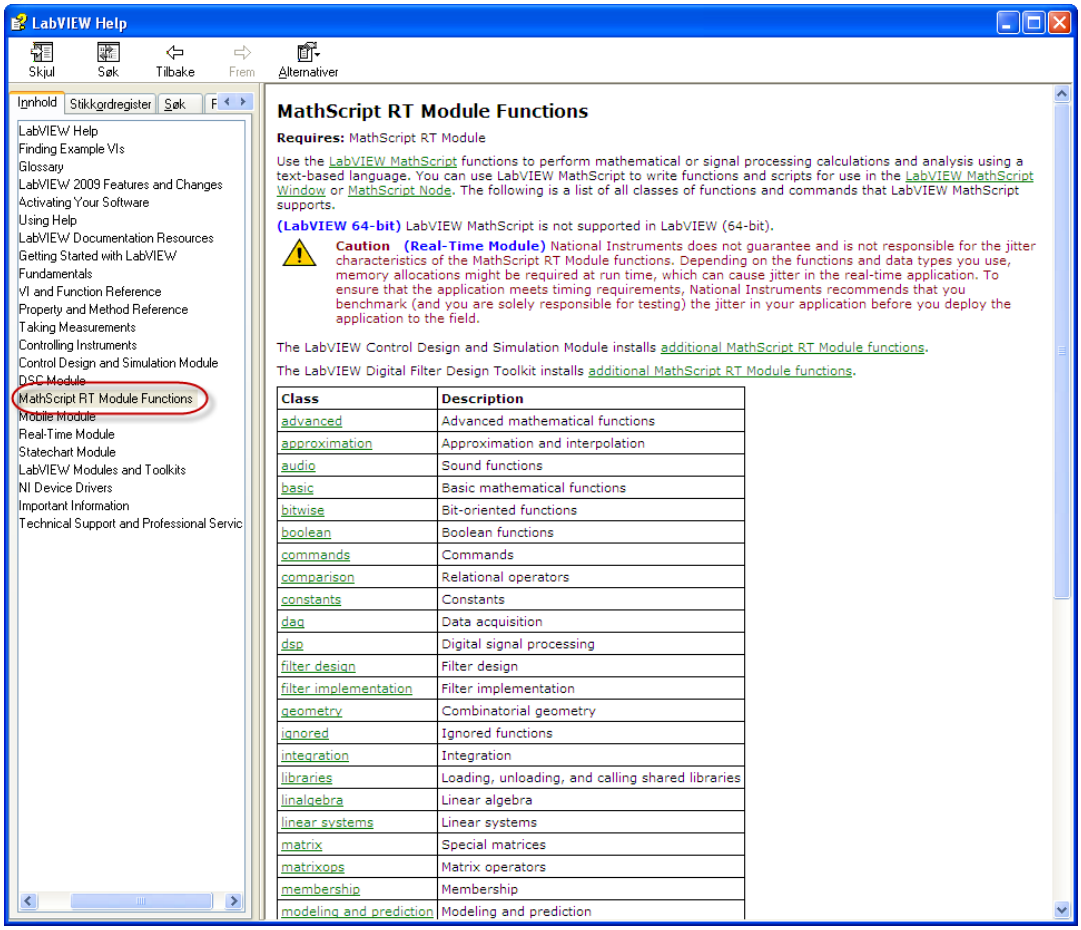
This Training is a part of a series with other Training Kits I have made, such as:

- Introduction to LabVIEW
- Data Acquisition in LabVIEW
- Control and Simulation in LabVIEW
- LabVIEW MathScript
- Linear Algebra in LabVIEW
- Datalogging and Supervisory Control in LabVIEW
- Wireless Data Acquisition in LabVIEW
- Intermediate Topics in LabVIEW
- Advanced Topics in LabVIEW

These Training Kits are available for download from my blog: https://www.halvorsen.blog

8.3 MathScript Functions

In the Help system there is detailed information about all the MathScript functions available. In addition to the MathScript RT Module functions, different add-on modules and toolkits installs additional functions. The LabVIEW Control Design and Simulation Module and LabVIEW Digital Filter Design Toolkit installs a lot of additional functions.


LabVIEW Help
Skjul
Søk
Tilbake
Frem
Alternativer
Innhold
Stikkordregister
Søk
LabVIEW Help
Finding Example VIs
Glossary
LabVIEW 2009 Features and Changes
Activating Your Software
Using Help
LabVIEW Documentation Resources
Getting Started with LabVIEW
Fundamentals
VI and Function Reference
Property and Method Reference
Taking Measurements
Controlling Instruments
Control Design and Simulation Module
DSC Module
MathScript RT Module Functions
Mobile Module
Real-Time Module
Statechart Module
LabVIEW Modules and Toolkits
NI Device Drivers
Important Information
Technical Support and Professional Servic
MathScript RT Module Functions
Requires: MathScript RT Module
Use the LabVIEW MathScript functions to perform mathematical or signal processing calculations and analysis using a text-based language. You can use LabVIEW MathScript to write functions and scripts for use in the LabVIEW MathScript Window or MathScript Node. The following is a list of all classes of functions and commands that LabVIEW MathScript supports.
(LabVIEW 64-bit) LabVIEW MathScript is not supported in LabVIEW (64-bit).
Caution (Real-Time Module) National Instruments does not guarantee and is not responsible for the jitter characteristics of the MathScript RT Module functions. Depending on the functions and data types you use, memory allocations might be required at run time, which can cause jitter in the real-time application. To ensure that the application meets timing requirements, National Instruments recommends that you benchmark (and you are solely responsible for testing) the jitter in your application before you deploy the application to the field.
The LabVIEW Control Design and Simulation Module installs additional MathScript RT Module functions.
The LabVIEW Digital Filter Design Toolkit installs additional MathScript RT Module functions.
Class
Description
advanced
Advanced mathematical functions
approximation
Approximation and interpolation
audio
Sound functions
basic
Basic mathematical functions
bitwise
Bit-oriented functions
boolean
Boolean functions
commands
Commands
comparison
Relational operators
constants
Constants
daq
Data acquisition
dsp
Digital signal processing
filter design
Filter design
filter implementation
Filter implementation
geometry
Combinatorial geometry
ignored
Ignored functions
integration
Integration
libraries
Loading, unloading, and calling shared libraries
linalgebra
Linear algebra
linear systems
Linear systems
matrix
Special matrices
matrixops
Matrix operators
membership
Membership
modeling and prediction
Modeling and prediction

Quick Reference

LabVIEW
Keyboard Shortcuts

Objects and Movement

Shift-click	Selects multiple objects; adds object to current selection.
↑↓→←(arrow keys)	Moves selected objects one pixel at a time.
Shift-↑↓→←	Moves selected objects several pixels at a time.
Shift-click (drag)	Moves selected objects in one axis.
Ctrl-click (drag)	Duplicates selected objects.
Ctrl-Shift-click (drag)	Duplicates selected objects and moves them in one axis.
Shift-resize	Resizes object while maintaining aspect ratio.
Ctrl-resize	Resizes object while maintaining center point.
Ctrl-Shift-resize	Resizes selected object while maintaining center point and aspect ratio.
Ctrl-drag a rectangle in open space	Adds more working space to the front panel or block diagram.
Ctrl-A	Selects all front panel or block diagram items.
Ctrl-Shift-A	Performs last alignment operation on objects.
Ctrl-D	Performs last distribution operation on objects.
Double-click open space	Adds a free label to the front panel or block diagram if automatic tool selection is enabled.
Ctrl-mouse wheel	Scrolls through subdiagrams of a Case, Event, or Stacked Sequence structure.
Spacebar (drag)	Disables preset alignment positions when moving labels or captions.
Ctrl-U	Reroutes all wires and rearranges block diagram objects automatically.

Debugging

Ctrl-↓	Steps into node.
Ctrl-→	Steps over node.
Ctrl-↓	Steps out of node.

Basic Editing

Ctrl-Z	Undoes last action.
Ctrl-Shift-Z	Redoes last action.
Ctrl-X	Cuts selected objects.
Ctrl-C	Copies selected objects
Ctrl-V	Pastes last cut or copied objects.

Navigating the LabVIEW Environment

Ctrl-E	Displays block diagram or front panel windows.
Ctrl-#	Enables or disables grid alignment. (Mac OS) Press the Command-* keys.
Ctrl-/	Maximizes and restores window.
Ctrl-T	Tiles front panel and block diagram windows.
Ctrl-F	Finds objects or text.
Ctrl-G	Searches VIs for next instance of object or text.
Ctrl-Shift-G	Searches VIs for previous instance of object or text.
Ctrl-Shift-F	Displays the **Search Results** window.
Ctrl-Tab	Cycles through LabVIEW windows.
Ctrl-Shift-Tab	Cycles through LabVIEW windows in reverse order.
Ctrl-Shift-N	Displays the **Navigation** window.
Ctrl-I	Displays the **VI Properties** dialog box.
Ctrl-L	Displays the **Error list** window.
Ctrl-Y	Displays the **History** window.
Ctrl-Shift-W	Displays the **All Windows** dialog box.
Ctrl-Space	Displays the **Quick Drop** dialog box. (Mac OS) Press the Command-Shift-Space keys.

Navigating the VI Hierarchy Window

Ctrl-D	Redraws the window.
Ctrl-A	Shows all VIs in the window.
Ctrl-click VI	Displays the subVIs and other nodes that make up the VI you select in the window.
Enter †	Finds next node that matches the search string.
Shift-Enter †	Finds previous node that matches the search string.

† After initiating a search by typing in the VI Hierarchy window.

File Operations

Ctrl-N	Creates a new, blank VI.
Ctrl-O	Opens an existing VI.
Ctrl-W	Closes the VI.
Ctrl-S	Saves the VI.
Ctrl-Shift-S	Saves all open files.
Ctrl-P	Prints the window.
Ctrl-Q	Quits LabVIEW.

Help

Ctrl-H	Displays the **Context Help** window. (Mac OS) Press the Command-Shift-H keys.
Ctrl-Shift-L	Locks the **Context Help** window.
Ctrl-? or F1	Displays the **LabVIEW Help**.

Refer to the **LabVIEW Help** for keyboard shortcut variations on other system locales and keyboard layouts.

Tools and Palettes

Ctrl	Switches to next most useful tool.
Shift	Switches to Positioning tool.
Ctrl-Shift over open space	Switches to Scrolling tool.
Spacebar†	Toggles between two most common tools.
Shift-Tab†	Enables automatic tool selection.
Tab†	Cycles through four most common tools if you disabled automatic tool selection by clicking the **Automatic Tool Selection** button. Otherwise, enables automatic tool selection.
↑↓→←	Navigates temporary **Controls** and **Functions** palettes.
Enter	Navigates into a temporary palette.
Esc	Navigates out of a temporary palette.
Shift-right-click	Displays a temporary version of the **Tools** palette at the location of the cursor.

† If automatic tool selection is disabled.

SubVIs

Double-click subVI	Displays subVI front panel.
Ctrl-double-click subVI	Displays subVI block diagram and front panel.
Drag VI icon to block diagram	Places that VI as a subVI on the block diagram.
Shift-drag VI icon to block diagram	Places that VI as a subVI on the block diagram with constants wired for controls that have non-default values.
Ctrl-right-click block diagram and select VI from palette	Opens the front panel of that VI.

Execution

Ctrl-R	Runs the VI.
Ctrl-.†	Stops the VI.
Ctrl-M	Changes to run or edit mode.
Ctrl-Run button	Recompiles the current VI.
Ctrl-Shift-Run button	Recompiles all VIs in memory.
Ctrl-↓†	Moves key focus inside an array or cluster.
Ctrl-↑†	Moves key focus outside an array or cluster.
Tab†	Navigates the controls or indicators according to tabbing order.
Shift-Tab†	Navigates backward through the controls or indicators.

† While the VI is running

Text

Double-click	Selects a single word in a string.
Triple-click	Selects an entire string.
Ctrl-→	Moves forward in string by one word.
Ctrl-←	Moves backward in string by one word.
Home	Moves to beginning of current line in string.
End	Moves to end of current line in string.
Ctrl-Home	Moves to beginning of entire string.
Ctrl-End	Moves to end of entire string.
Shift-Enter	Adds new items when entering items in enumerated type controls and constants, ring controls and constants, or Case structures.
Esc	Cancels current edit in a string.
Ctrl-Enter	Ends text entry.
Ctrl-=	Increases the current font size.
Ctrl--	Decreases the current font size.
Ctrl-0	Displays the **Font** dialog box.
Ctrl-1†	Changes to the Application font.
Ctrl-2†	Changes to the System font.
Ctrl-3†	Changes to the Dialog font.
Ctrl-4†	Changes to the current font.

† In the **Font** dialog box.

Wiring

Ctrl-B	Removes all broken wires.
Esc, right-click, or click terminal	While wiring, cancels a wire you started.
Single-click wire	Selects one segment.
Double-click wire	Selects a branch.
Triple-click wire	Selects entire wire.
A	While wiring, disables automatic wire routing temporarily.
Double-click	While wiring, tacks down wire without connecting it.
spacebar	While wiring, switches the direction of a wire between horizontal and vertical.
spacebar	While moving objects, toggles automatic wiring.
Ctrl-click input on function with two inputs	Switches the two input wires.
Shift-click	While wiring, undoes last point where you set a wire.

Note: The **Ctrl** key in these shortcuts corresponds to the **Option** or **Command** key on Mac OS and the **Alt** key on Linux.

Linear Algebra in LabVIEW

Hans-Petter Halvorsen

E-Mail: hans.p.halvorsen@usn.no

Web: https://www.halvorsen.blog

https://www.halvorsen.blog