LabVIEW

G Math Toolkit Reference Manual

November 1996 Edition
Part Number 321290A-01

© Copyright 1996 National Instruments Corporation. All Rights Reserved.

Internet Support

support@natinst.com
E-mail: info@natinst.com
FTP Site: ftp.natinst.com
Web Address: http://www.natinst.com

Bulletin Board Support

BBS United States: (512) 794-5422
BBS United Kingdom: 01635 551422
BBS France: 01 48 65 15 59

Fax-on-Demand Support

(512) 418-1111

Telephone Support (U.S.)

Tel: (512) 795-8248
Fax: (512) 794-5678

International Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, Denmark 45 76 26 00,
Finland 09 527 2321, France 01 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186,
Israel 03 5734815, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456,
Mexico 5 520 2635, Netherlands 0348 433466, Norway 32 84 84 00, Singapore 2265886,
Spain 91 640 0085, Sweden 08 730 49 70, Switzerland 056 200 51 51, Taiwan 02 377 1200,
U.K. 01635 523545

National Instruments Corporate Headquarters

6504 Bridge Point Parkway Austin, TX 78730-5039 Tel: (512) 794-0100

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this manual is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

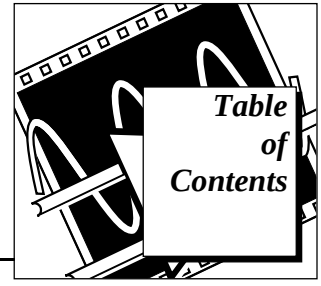
Trademarks

HiQ®, LabVIEW®, National Instruments™, natinst.com™, and The Software is the Instrument® are trademarks of National Instruments Corporation.

Product and company names listed are trademarks or trade names of their respective companies.

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.



About This Manual

Organization of the G Math Toolkit Reference Manual xi
Conventions Used in This Manual xii
Related Documentation xiii
Customer Communication xiii

Chapter 1
Introduction

G Math Toolkit Examples and Parameter Help 1-2
Using the Parser VIs 1-2
Example 1 1-3
Example 2 1-3
Example 3 1-4
Example 4 1-4
The Parser VIs in More Detail 1-4
Error Structure 1-8
Variables 1-10

Chapter 2
Parser VIs

Parser VI Descriptions 2-1
Eval Formula Node 2-1
Eval Formula String 2-2
Eval Multi-Variable Array 2-3
Eval Multi-Variable Scalar 2-4
Eval Parsed Formula Node 2-6
Eval Parsed Formula String 2-7
Eval Single-Variable Array 2-8
Eval Single-Variable Scalar 2-9
Parse Formula Node 2-10
Parse Formula String 2-11
Substitute Variables 2-12

Chapter 3
Data Visualization VIs

Data Visualization VI Descriptions 3-1
Animate Graphs 3-1
Common Intensity Maps 3-2
Contour Plot 3-3
Cut nD Surface 3-4
Draw Graph 3-5
Mesh 3D 3-6
Parametric Curve 2D 3-7
Parametric Curve 3D 3-8
Read Graphs 3-10
Write Graphs 3-11

Chapter 4
Ordinary Differential Equation VIs

Ordinary Differential Equation VI Descriptions 4-1
ODE Cash Karp 5th Order 4-1
ODE Euler Method 4-4
ODE Linear nth Order Numeric 4-7
ODE Linear nth Order Symbolic 4-9
ODE Linear System Numeric 4-11
ODE Linear System Symbolic 4-13
ODE Runge Kutta 4th Order 4-15

Chapter 5
Zero Finder VIs

Zero Finder VI Descriptions 5-1
Find All Zeros of f(x) 5-1
Newton Raphson Zero Finder 5-3
Nonlinear System Single Solution 5-4
Nonlinear System Solver 5-5
Polynomial Real Zero Counter 5-7
Ridders Zero Finder 5-8

Chapter 6
Optimization VIs

Optimization VI Descriptions 6-2
Brent with Derivatives 1D 6-2
Chebyshev Approximation 6-3
Downhill Simplex nD 6-4
Conjugate Gradient nD 6-5
Find All Minima 1D 6-7
Find All Minima nD 6-8
Fitting on a Sphere 6-9
Golden Section 1D 6-10
Levenberg Marquardt 6-12
Linear Programming Simplex Method 6-13
Pade Approximation 6-14

Chapter 7
1D Explorer VIs

1D Explorer VI Descriptions 7-1
Curve Length 7-1
Differentiation 7-2
Eval Polar to Rect 7-4
Eval Polar to Rect Optimal Step 7-5
Eval X-Y(a,t) 7-6
Eval X-Y(t) 7-7
Eval X-Y(t) Optimal Step 7-9
Eval y=f(a,x) 7-10
Eval y=f(x) 7-11
Eval y=f(x) Optimal Step 7-12
Integration 7-13
Limit 7-14
Zeroes and Extrema of f(x) 7-15

Chapter 8
2D Explorer VIs

2D Explorer VI Descriptions 8-1
Eval X-Y-Z(a,t1,t2) 8-1
Eval X-Y-Z(t1,t2) 8-3
Eval y=f(a,x1,x2) 8-4
Eval y=f(x1,x2) 8-6
Extrema of f(x1,x2) 8-7
Partial Derivatives of f(x1,x2) 8-8

Chapter 9
Function VIs

Function VI Descriptions 9-1
Bessel Function Jn(x) 9-1
Bessel Function Yn(x) 9-2
Bessel Polynomial 9-2
Beta Function 9-3
Binomial Coefficient 9-4
Chebyshev Polynomial 9-4
Continued Fraction 9-6
Cosine Integral 9-6
Gamma Function 9-7
Incomplete Beta Function 9-8
Incomplete Gamma Function 9-9
Jacobian Elliptic Function 9-11
Legendre Elliptic Integral 1st Kind 9-12
Sine Integral 9-13
Spike Function 9-14
Square Function 9-15
Step Function 9-16

Chapter 10
Transform VIs

Transform VI Descriptions 10-1
Buneman Frequency Estimator 10-1
Daubechies4 Function 10-2
Dual Signal FFT 10-3
Fractional FFT 10-3
Laplace Transform Real 10-5
Power Spectrum Fractional FFT 10-7
Prime FFT 10-7
Sparse FFT 10-8
Sparse Signal FFT 10-9
STFT Spectrogram 10-10
Unevenly Sampled Signal Spectrum 10-11
Walsh Hadamard 10-13
Walsh Hadamard Inverse 10-14
Wavelet Transform Daubechies4 10-15
Wavelet Transform Daubechies4 Inverse 10-16
WVD Spectrogram 10-19

Appendix A
Error Codes

Appendix B
References

Appendix C
Customer Communication

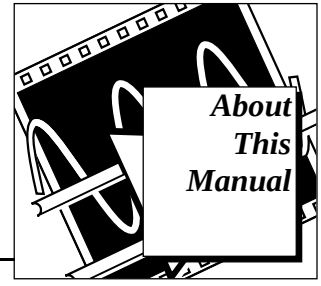
Glossary

Figures

Figure 3-1. The Meaning of Observer and Origin 3-9

Tables

Table A-1. G Math Toolkit Error Codes A-1
Table A-2. G Math Toolkit Parser Error Codes A-4



The *G Math Toolkit Reference Manual* describes the features, functions, and operation of the G Math Toolkit. With this toolkit and your LabVIEW or BridgeVIEW application, you can perform complex mathematical calculations. The G Math Toolkit is intended for use by scientists, engineers, and mathematicians, or anyone needing to solve mathematical problems in a simple, quick and efficient manner. It can also be used as an educational aid by those interested in learning and expanding their knowledge of mathematics. To use this manual effectively, you should be familiar with the G programming language and the basic theory behind the type of problem that you want to solve.

Organization of the G Math Toolkit Reference Manual

The *G Math Toolkit Reference Manual* is organized as follows:

- Chapter 1, *Introduction*, introduces the G Math Toolkit, presents some examples of how you can use this toolkit for your day-to-day mathematical tasks, and describes some of the basics of working with the G Math Toolkit.
- Chapter 2, *Parser VIs*, describes the VIs that act as an interface between the end user and the programming system. These VIs parse the formula, which is in the form of a string, and convert the formula string to a form that can be used for evaluating results.
- Chapter 3, *Data Visualization VIs*, describes the VIs that are used for plotting and visualizing data in several different forms. These include advanced methods such as animation, contour plots, and surface cuts.
- Chapter 4, *Ordinary Differential Equation VIs*, describes the VIs you can use to solve ordinary differential equations, both symbolically and numerically.
- Chapter 5, *Zero Finder VIs*, describes the VIs that find the zeros of 1D or *n*-dimension, linear or nonlinear functions (or system of functions).
- Chapter 6, *Optimization VIs*, describes the VIs you can use to determine local minima and maxima of real 1D or *n*-dimension

functions. You can choose between optimization algorithms based on derivatives of the function and algorithms working without these derivatives.

- Chapter 7, *1D Explorer VIs*, describes the VIs you can use to study real-valued 1D functions given in symbolic form. You can study different qualities of function graphs with and without additional parameters.
- Chapter 8, *2D Explorer VIs*, describes the VIs you can use to examine 2D functions given in symbolic form, where parameterization is allowed. You can numerically calculate extrema and partial derivatives.
- Chapter 9, *Function VIs*, describes the VIs you can use to evaluate some common mathematical functions.
- Chapter 10, *Transform VIs*, describes VIs that implement some transforms commonly used in mathematics and signal processing.
- Appendix A, *Error Codes*, contains the error codes for the G Math Toolkit.
- Appendix B, *References*, section contains references to the mathematical theory or algorithm implemented in each VI.
- Appendix C, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and manuals.
- The *Glossary* contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, mnemonics, and symbols.

Conventions Used in This Manual

The following conventions are used in this manual:

bold — Bold text denotes a parameter, menu name, menu item, or dialog box button or option.

italic — Italic text denotes emphasis, a cross reference, or an introduction to a key concept.

bold italic ☞ — Bold italic text accompanied by the icon to the left denotes a note, which alerts you to important information.

`monospace` — Text in this font denotes text or characters that are to be literally input from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk

drives, paths, directories, programs, subprograms, subroutines, device names, functions, variables, filenames, and extensions, and for statements and comments taken from program code.

<> Angle brackets enclose the name of a key on the keyboard–for example, <PageDown>.

- A hyphen between two or more key names enclosed in angle brackets denotes that you should simultaneously press the named keys–for example, <Control-Alt-Delete>.

<Control> Key names are capitalized.

Abbreviations, acronyms, metric prefixes, mnemonics, symbols, and terms are listed in the *Glossary*.

Related Documentation

The following documents contain information that you may find helpful as you read this manual:

- *LabVIEW User Manual*
- *LabVIEW Analysis VI Reference Manual*
- *HiQ for Macintosh User Manual*
- *HiQ for Macintosh Function Reference Manual*
- *HiQ for Windows User Manual*

Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual contains comment and configuration forms for you to complete. These forms are in Appendix C, *Customer Communication*, at the end of this manual.

Introduction

Chapter 1

The G Math Toolkit is a new approach to solving advanced mathematics and analysis problems. With the G Math Toolkit and the G programming language underlying LabVIEW and BridgeVIEW, you can graphically assemble your problem-solving program as a block diagram. In addition, the interactive front panels introduce a new level of interactivity to math and analysis problem solving. With the G Math Toolkit, you can put the compiled speed, connectivity, open architecture, and flexibility of G to work in your applications. This chapter explains what the G Math Toolkit is, and illustrates how you can use this toolkit for your day-to-day mathematical tasks.

Some of the advantages of G programming over classical procedural programming languages such as FORTRAN, C, Pascal, and so on, include:

- Visual programming
- Data driven system design
- Virtual instrumentation
- Platform independence

The graphical user interface combined with the data driven paradigm makes G the perfect tool for complex measurement and analysis tasks because of the commonality of such tasks with mathematical operations and algorithms. You can use G as a programming system for almost all kinds of numerically and symbolically oriented mathematical routines.

The core of the G Math Toolkit is the Parser VI Library, which makes manipulating formulas on LabVIEW front panels possible. All the G Math Toolkit VI libraries are interconnected, and leverage off each other. For example, the Ordinary Differential Equation (ODE) VI library is based on the Function and the Parser VI libraries.

The strength of the complete G Math Toolkit is not just the power of individual VIs in performing mathematical calculations, but also in combining VIs to solve extremely complex problems. In addition, you can interface real-world measurements to the mathematical algorithms

in order to obtain practical solutions. This flexibility makes the toolkit an extremely powerful analysis tool.

G Math Toolkit Examples and Parameter Help

Study the G Math Toolkit examples to learn some of the possible theoretical and practical applications of this package. These example libraries, installed in the `gmath` directory in your LabVIEW or BridgeVIEW `Examples` directory, are grouped into the following categories:

- `Graphics.llb` examples showing the graphics capabilities
- `Math.llb` examples from mathematics
- `Mechanix.llb` examples applied to the field of mechanics
- `Misc.llb` miscellaneous “real world” examples
- `Optimiz.llb` examples related to optimization
- `Sig_proc.llb` examples applied to signal processing

Also, many of the G Math Toolkit VIs have example text on the front panel of the VI showing how to input various parameters. Double-click on the VI icon to open the front panel of the VI and examine this helpful text.

Using the Parser VIs

The Parser VIs are a collection of VIs that directly connect an end user to the programming system. Until now, only the LabVIEW Formula Node could manage formula expressions. But the Formula Node is a pure programming tool, not directly accessible to the user from the front panel.

In many applications, being able to enter formulas directly from the front panel is extremely useful. This is possible using the G Math Toolkit. The following examples describe some typical scenarios where you might use the G Math Toolkit.

Example 1

Given a set of measurements (x_i, y_i), where $i = 0, 1, \ldots, n-1$, fit the set of data points into a model equation, such as the following.

$Y = a\sin(bX) + c\cos(dX)$

or

$Y = a + bX + c\exp(dX)$

or more generally

$Y = f(X, a, b, \ldots, c)$

(where $a, b, \ldots, c$ are the unknown model parameters)

G can calculate such optimal system parameters as

$$\sum_{i=0}^{n-1} (y_i - f(x_i, a, b, \ldots, c))^2 = \text{min!}$$

G uses the Levenberg Marquardt method to solve this minimization problem. Before now, the model equation had to be fixed in the formula node before running the program. With the Parser VIs, you can input the model equation on the front panel.

Example 2

Another typical example consists of the following three-step process.

1. Collect a set of discrete measurements
   (x_i, y_i, z_i) for $i = 0, 1, \ldots, n-1$
2. Fit this set based on a model such as
   $Z = f(X, Y, a, b, \ldots, c)$ with unknown parameters $a, b, \ldots, c$
3. Determining those points (x, y) with
   $z = f(x, y, a, b, \ldots, c) = 0$
   or
   $z = f(x, y, a, b, \ldots, c) = \text{max!}$
   or
   $z = f(x, y, a, b, \ldots, c) = \text{min!}$

An easy way to input discrete measurements on a LabVIEW front panel makes calculating the roots, minima, and maxima for a general model simple for your end user. Notice that at the beginning of the previous process, the correct formula $Z = f(X, Y, a, b, \ldots, c)$ is not known.

Example 3

Another application for the G Math Toolkit is to help control an x-y step motor or a robot control that positions objects in 2D or 3D space during runtime. The path of the object can be calculated with the VIs found in this toolkit.

Another 2D and 3D application of the G math toolkit is for surface description. Wings of airplanes, among other parts of machines and instruments in the real technical world, can be described by mathematical curves and surfaces. In nondestructive testing (using ultrasound, eddy currents, or X-rays), a prescan of the structure under test is initially done, followed by a more accurate scan in areas where the measured and expected values differ.

Because it is virtually impossible to store the measurements of all the curves and surfaces of an entire structure, it is necessary to take a set of coarse measurements as a preparatory step. The prescan of the wing of a plane is then followed by a more accurate scan of a smaller part of this wing. Because the Parser VIs can handle formulas on the front panel, you can use them to calculate the 2D and 3D curves of the wing in an effective manner.

Example 4

The study of solutions of differential equations (especially parameter studies) is not only a question of appropriate numeric algorithms such as the Euler method, Runge Kutta method, or the Cash Karp method but also a question of formula manipulation. With the G Math Toolkit you can manipulate differential equations on the front panel. See the examples in the `math.llb` example collection for sample versions of this approach to solving differential equations.

The Parser VIs in More Detail

A Parser VI scans an input string and interprets this string as a formula, or a collection of formulas. Then, the Parser VI transforms the formulas into numeric calculations and outputs the results.The Parser VI routines deal only with real numbers.

There are some differences between the parser in the G Math Toolkit and the Formula Node found in the original LabVIEW package. The following table outlines these differences.

Element	Formula Node	Parser VI Routines
Variables	No restrictions	Only *a*, *a*0, ..., *a*9, ... *z*, *z*0, ..., *z*9, are valid
Binary functions	max, min, mod, rem	Not available

Element	Formula Node	Parser VI Routines
More complex math functions	Not available	gamma, ci, si, spike, step, square
Logical, conditional, inequality, equality	?:, ||, &&, !=, ==, <, >, <=, >=	Not available, except for the Eval Formula Node VI
π	pi	pi(1) = π, pi(2) = 2π 2pi or 2(pi) will return an error

The precedence of operators is the same for the G Math Toolkit Parser VIs as in the Formula Nodes of G. Refer to Chapter 2, *Parser VIs*, for more information on specific VIs.

All functions in the G Math Toolkit use the following syntax:

```
function (argument)
```

The following table lists the functions you can use with the Parser VIs.

Function	Corresponding G Math Function name	Description
abs(x)	Absolute Value	Returns the absolute value of *x*.
acos(x)	Inverse Cosine	Computes the inverse cosine of *x*.
acosh(x)	Inverse Hyperbolic Cosine	Computes the inverse hyperbolic cosine of *x* in radians.
asin(x)	Inverse Sine	Computes the inverse sine of *x* in radians.
asinh(x)	Inverse Hyperbolic Sine	computes the inverse hyperbolic sine of *x* in radians.
atan(x)	Inverse Tangent	Computes the inverse tangent of *x* in radians.
atanh(x)	Inverse Hyperbolic Tangent	Computes the inverse hyperbolic tangent of *x* in radians.
ci(x)	Cosine Integral	Computes the cosine integral of *x* where *x* is any real number.

Function	Corresponding G Math Function name	Description
ceil(x)	Round to +Infinity	Rounds *x* to the next higher integer (smallest integer $\geq x$.)
cos(x)	Cosine	Computes the cosine of *x* in radians.
cosh(x)	Hyperbolic Cosine	Computes the hyperbolic cosine of *x* in radians.
cot(x)	Cotangent	Computes the cotangent of *x* in radians (1/tan(x)).
csc(x)	Cosecant	Computes the cosecant of *x* in radians (1/sin(x)).
exp(x)	Exponential	Computes the value of *e* raised to the power *x*.
expm1(x)	Exponential(Arg)–1	Computes the value of *e* raised to the power of $x - 1$ ($e^x - 1$)
floor(x)	Round to –Infinity	Truncates x to the next lower integer (Largest integer $\leq x$)
gamma(x)	Gamma Function	$\Gamma(n + 1) = n!$ for all natural numbers *n*.
getexp(x)	Mantissa and exponent	Returns the exponent of *x*.
getman(x)	Mantissa and exponent	Returns the mantissa of *x*.
int(x)	Round to nearest integer	Rounds its argument to the nearest even integer.
intrz	Round toward zero	Rounds *x* to the nearest integer between *x* and zero.
ln(x)	Natural Logarithm	Computes the natural logarithm of *x* (to the base *e*).
lnpl(x)	Natural Logarithm (Arg + 1)	Computes the natural logarithm of (x + 1).

Function	Corresponding G Math Function name	Description
log(x)	Logarithm Base 10	Computes the logarithm of *x* (to the base 10).
log2(x)	Logarithm Base 2	Computes the logarithm of *x* (to the base 2).
pi(x)	Represents the value $\pi = 3.14159...$	pi(*x*) = *x**π pi(1) = π pi(2.4) = 2.4*π
rand( )	Random Number (0–1)	Produces a floating-point number between 0 and 1.
sec(x)	Secant	Computes the secant of *x* (1/cos(*x*)).
si(x)	Sine Integral	Computes the sine integral of *x* where *x* is any real number.
sign(x)	Sign	Returns 1 if *x* is greater than 0. Returns 0 if *x* is equal to 0. Returns –1 if *x* is less than 0.
sin(x)	Sine	Computes the sine of *x* in radians.
sinc(x)	Sinc	Computes the sine of *x* divided by *x* in radians (sin(*x*)/*x*).
sinh(x)	Hyperbolic Sine	Computes the hyperbolic sine of *x* in radians.
spike(x)	Spike function	spike(*x*) returns: 1 if $0 \leq x \leq 1$ 0 for any other value of *x*.
sqrt(x)	Square Root	Computes the square root of *x*.

Function	Corresponding G Math Function name	Description
square(x)	Square function	square (x) returns: 1 if $2n \leq x \leq (2n+1)$ 0 if $2n+1 \leq x \leq (2n+2)$ where x is any real number and n is any integer.
step(x)	Step function	step(x) returns: 0 if $x < 0$ 1 if any other condition obtains.
tan(x)	Tangent	Computes the tangent of x in radians.
tanh(x)	Hyperbolic Tangent	Computes the hyperbolic tangent of x in radians.

Error Structure

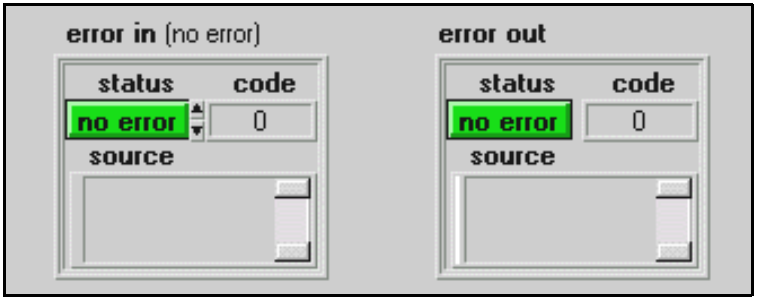
The Parser VIs use the following error handling structure. This structure consists of a Boolean **status** button, a signed 32-bit integer numeric **code** indicator, and a string **source** indicator. These error handler components are explained below:

status is TRUE if an error occurred. If **status** is TRUE, this VI does not perform any operations.

code is the error code number identifying the error.

source is a message explaining the error in more detail.

The default **status** of the **error in** structure is TRUE (no error), indicated by an error **code** of 0.

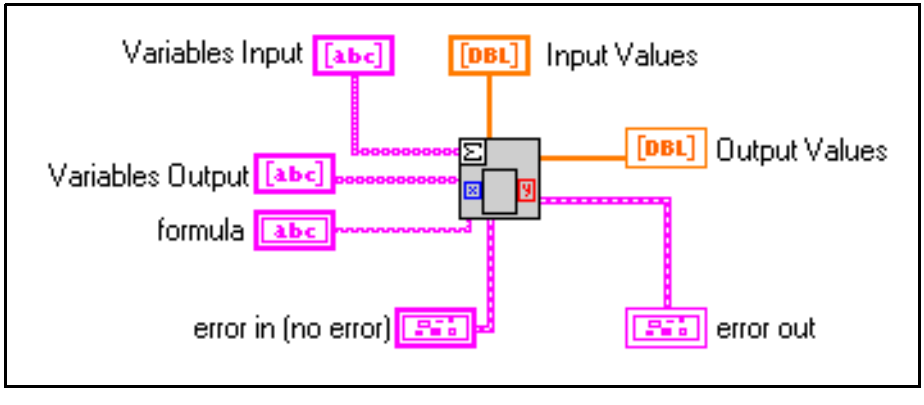


With this structure, you can programmatically check the accuracy of formulas and control the data flow in case of errors. The application uses the **source** field of the error handling structure as a storage for a incorrect or invalid formula input. This field displays limited error descriptions if an error is detected in your program. See Appendix A, *Error Codes*, for the error codes and the messages of the Parser VI routines.

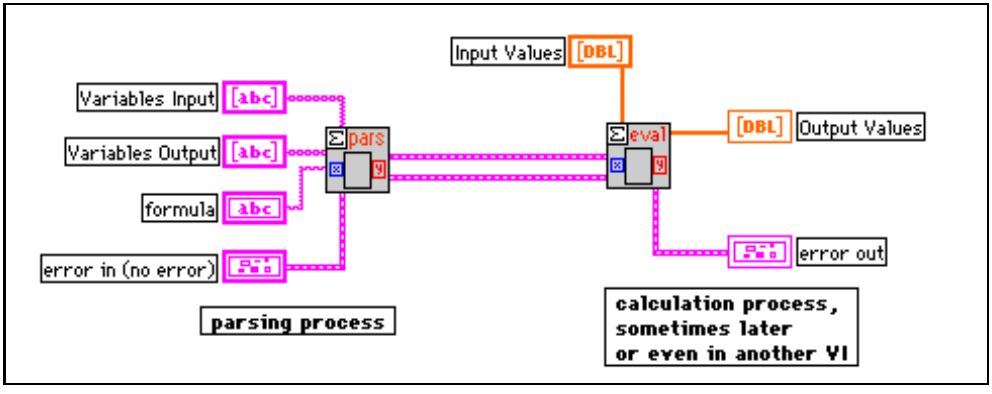
There are three main kinds of Parser routines:

- The VIs representing the functionality of the LabVIEW Formula Node (the Eval Formula Node, Parse Formula Node, and Eval Parsed Formula Node VIs)
- The VIs analyzing a simple string as one formula (the Eval Formula String, Parse Formula String, and Eval Parsed Formula String VIs)
- The VIs producing whole sets of function values (the Eval Single-Variable Array, Function Explorer, and Eval Multi-Variable Scalar VIs)

The first two categories of Parser VIs can be further divided into two subcategories, the *direct* form and the *indirect* form. As an example, the direct version of the Eval Formula Node VI is represented by the following block diagram.



On the other hand, the indirect forms split the VI in two subVIs, as shown in the following illustration. You can use the indirect form in larger applications, where a two-step process (parsing and then evaluating) is more efficient.



Variables

The Parser VIs accept only the following variables:

a, *a*0,…, *a*9

b, *b*0,…, *b*9

.

.

.

z, *z*0,…, *z*9

☞ **Note:** ***For variable and function names, only lowercase letters are allowed. The toolkit interprets capital letters as errors.***

For the sake of uniqueness, all numbers in exponential notation use the 1E-1 convention (with the capital letter E). Using 1e-1 (with the lowercase letter e) results in an error message.

The following table shows some common error codes, the description of the error, and an error example.

Code	Error Description	Error Example
0	No error	sin(x)
1	Bracket problem at the beginning	1+x)
2	Incomplete function expression	sin(x)+
3	Bracket problem	()
4	Bracket problem at the end	(1+x
5	Wrong decimal point	1,2 (US)
6	Wrong number format	1e-3 instead of 1E-3
7	Wrong function call	sin()
8	Not a valid function	sins(x)
9	Incomplete expression	x+
10	Wrong variable name	a11
11	Wrong letter	sin(X)
12	Too many decimal points	1.23.45
21	Contains more than one variable	1+x+y4
22	Inconsistency in variables or numbers	Depends on application
23	Contains variables	Depends on application
24	Variables output problem	Depends on application

Parser VIs

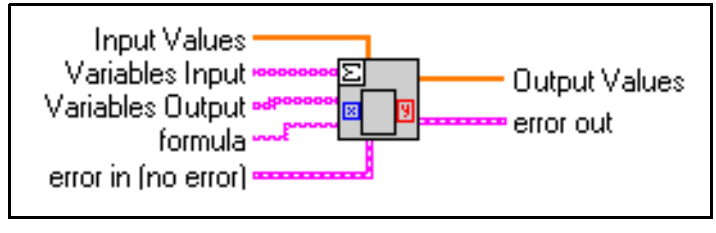
Chapter 2

The Parser VIs act as an interface between the end user and the programming system. These VIs parse the formula, which is in the form of a string, and convert the formula string to a form that can be used for evaluating results. The Parser VIs are found in the PARSER.LLB library.

Parser VI Descriptions

Eval Formula Node

Functionally equivalent to the Formula Node in LabVIEW, but with variables that can be entered on the front panel.



[DBL] **Input Values** is an array of numbers with a one-to-one relation to **Variables Input**.

[abc] **Variables Input** is an array of input strings, each of which represents a valid variable name.

[abc] **Variables Output** is an array of output strings, each of which represents a valid variable name.

abc **formula** is a string consisting of one or more formulas separated by semicolons. Each formula is built up by **Variables Input** on the right side of an equation and **Variables Output** on the left side.

error in (no error) describes error conditions occurring before this VI executes. If an error has already occurred, this VI returns the value of the **error in** cluster in **error out**. The VI executes normally only if no incoming error exists; otherwise it merely passes the **error in** value to **error out**. Refer to the *Error Structure* section of Chapter 1, *Introduction*, in this manual for a description of the **error in** cluster components.

Output Values is a 1D array of numbers corresponding to **Variables Output** and **formula**.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Example

The following inputs

formula: y = 3*x + 4*z; p = q^2 – 5;

Variables Input: [x, z, q]

Input Values: [1,2,3]

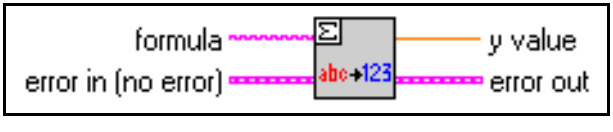
Variables Output: [y, p]

result in

Output Values: [11.00, 4.00]

Eval Formula String

Interprets a string as a numeric calculation and determines the result.



formula is a string representing the calculation without any variables. It consists only of numbers and mathematical functions.

error in (no error) describes error conditions occurring before this VI executes. If an error has already occurred, this VI returns the value of the **error in** cluster in **error out**. The VI executes normally only if no incoming error exists; otherwise it merely passes the **error in** value to **error out**. Refer to the *Error Structure* section of Chapter 1, *Introduction*, in this manual for a description of the **error in** cluster components.

y value is the result of the calculation.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Example

The following input

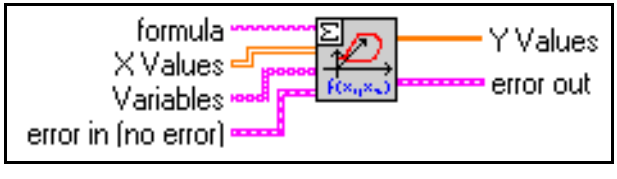
formula: sin(pi(1/2)) + 3*5 – 2

results in

y value: 14.00

Eval Multi-Variable Array

Calculates the function values of a given function at an arbitrarily given set of n-dimension points by $y_i = f(x_{1i}, x_{2i}, \ldots, x_{ni})$, where f is an n-dimension function given by the formula, and $(x_1, x_2, \ldots, x_n)$ are n independent variables.



formula is a string representing the n-dimension function under investigation.

X Values is a 2D array of X Values. Each row of the array represents the fixed values of each of the **Variables** of the n-dimension function. The other dimension of the array marks the different n-dimension points at which the function has to be calculated.

Variables is an array of strings. Each element of the array stands for a variable name of the n-dimension independent terms.

error in (no error) describes error conditions occurring before this VI executes. If an error has already occurred, this VI returns the value of the **error in** cluster in **error out**. The VI executes normally only if no incoming error exists; otherwise it merely passes the **error in** value to **error out**. Refer to the *Error Structure* section of Chapter 1, *Introduction*, in this manual for a description of the **error in** cluster components.

Y Values is the 1D array of the evaluated values, y_i, using **formula** (**X Values**). Each element of **Y** corresponds to the corresponding column of **X**.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Example

The following inputs

formula: 3*x1 + 4*x2 + x3^2

X Values: [1,0; –1, 4; 2,1]([1, –1, 2] are the values for [x_1, x_2, x_3] in the first iteration; [0, 4, 1] for the second)

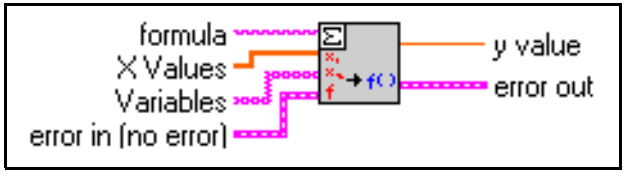
Variables: [x1, x2, x3] (for x_1, x_2, and x_3)

result in

Y Values: [3, 17]

Eval Multi-Variable Scalar

Calculates exactly one function value based on a given formula $y = f(x_1, x_2, \ldots, x_n)$.



formula is a string representing the formula of *n* independent **Variables**.

X Values is an array of X Values corresponding to the *n* **Variables**.

Variables is an array of strings representing the *n* independent variables of the given formula. There is a one-to-one relation between **Variables** and **X Values**.

error in (no error) describes error conditions occurring before this VI executes. If an error has already occurred, this VI returns the value of the **error in** cluster in **error out**. The VI executes normally only if no incoming error exists; otherwise it merely passes the **error in** value to **error out**. Refer to the *Error Structure* section of Chapter 1, *Introduction*, in this manual for a description of the **error in** cluster components.

y value is the value of **formula** (**X Values**).

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Note: ***This VI calculates one and only one value of a given n-dimension function. If you want to calculate a collection of function values, use the Eval Multi-Variable Array, Parse Formula String and Eval Parsed Formula String VIs.***

Example

The following inputs

formula: 3*x1 + 4*x2 + x3^2

X Values: [1, –1, 2]

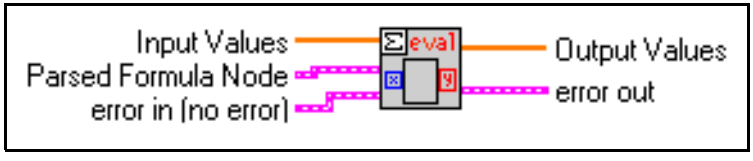
Variables: [x1, x2, x3]

result in

y value: 3.00

Eval Parsed Formula Node

The Eval Parsed Formula Node VI separates the parsing process from the evaluating process of the Eval Formula Node VI and improves the runtime behavior of a program containing the Eval Formula Node VIs at different locations. This VI has a direct connection from the Parse Formula Node VI and completes an Eval Formula Node VI calculation. You can wire the **Parsed Formula Node** input of this VI directly from the **Parsed Formula Node** output of the Parse Formula Node VI.



[DBL] **Input Values** is an array of numbers, each of which corresponds to the value that is given to each of the variables. These variables can be input in the **Variables Input** control of the Parse Formula Node VI. The coded form of these variables is available in the **Variables input decode** parameter of the **Parsed Formula Node** cluster.

[Cl] **Parsed Formula Node** is a cluster consisting of:

[DBL] **Y Values** is a 2D array of numbers representing a storage of detected and analyzed numbers of **formula** of the Parse Formula Node VI.

[I32] **Tables** is a 3D array with 3 columns. The first column contains a code that stands for the operator, the other two contain codes that stand for the operands.

[I32] **Variables input decode** is the intermediate and coded state of **Variables Input**. *See also* the Parse Formula Node VI.

[I32] **Variables output decode** is the intermediate and coded state of **Variables Output**. *See also* the Parse Formula Node VI.

You can wire the **Parsed Formula Node** cluster directly from the corresponding output of the Parse Formula Node VI.

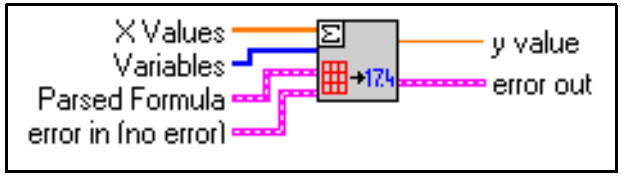
[Cl] **error in (no error)** describes error conditions occurring before this VI executes. If an error has already occurred, this VI returns the value of the **error in** cluster in **error out**. The VI executes normally only if no incoming error exists; otherwise it merely passes the **error in** value to **error out**. Refer to the *Error Structure* section of Chapter 1, *Introduction*, in this manual for a description of the **error in** cluster components.

Output Values is a 1D array of numbers corresponding to **Variables Output Decode**, **Y Values** and **Tables**.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Eval Parsed Formula String

Takes the output of Parse Formula String VI and fixes input values to calculate function values.



X Values is an array of X Values corresponding to **Variables**.

Variables is a 1D array of strings representing the independent variables. There is a one-to-one relation between **X Values** and **Variables**.

Parsed Formula is a cluster consisting of:

Y Values is a 1D array of numbers representing a storage of detected and analyzed numbers of **formula**.

Table is a 2D array with 3 columns. The first column contains a code that stands for the operator, the other two contain codes that stand for the operands.

This input can be wired directly from the corresponding output of the Parse Formula String VI.

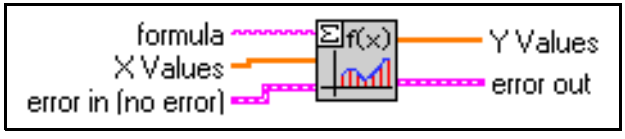
error in (no error) describes error conditions occurring before this VI executes. If an error has already occurred, this VI returns the value of the **error in** cluster in **error out**. The VI executes normally only if no incoming error exists; otherwise it merely passes the **error in** value to **error out**. Refer to the *Error Structure* section of Chapter 1, *Introduction*, in this manual for a description of the **error in** cluster components.

y value is the result of the interpretation process, such as the function value.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Eval Single-Variable Array

Calculates an array of function values at given points in a given interval by $y_i = f(x_i)$ where f is the one-dimensional (1D) function given by the user formula.



formula is a string representing the 1D function under investigation.

X Values is the given array of input values, x_i.

error in (no error) describes error conditions occurring before this VI executes. If an error has already occurred, this VI returns the value of the **error in** cluster in **error out**. The VI executes normally only if no incoming error exists; otherwise it merely passes the **error in** value to **error out**. Refer to the *Error Structure* section of Chapter 1, *Introduction*, in this manual for a description of the **error in** cluster components.

Y Values is the 1D array of function values of **formula** at the given points in the array **X Values**.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Example

To calculate the formula $y = x^2$, for x = 1, 2, 3, 4, and 5 the input arrays would be

formula: x^2

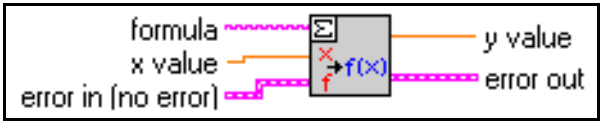
X Values: [1, 2, 3, 4, 5]

resulting in the output array

Y Values: [1, 4, 9, 16, 25]

Eval Single-Variable Scalar

Calculates exactly one function value of a given 1D function, $y = f(x)$, where f is the function specified by the user formula.



formula is a string representing the function under investigation. Only one variable can be integrated in this **formula**.

x value is the 1D point, *x*, at which the function value has to be calculated.

error in (no error) describes error conditions occurring before this VI executes. If an error has already occurred, this VI returns the value of the **error in** cluster in **error out**. The VI executes normally only if no incoming error exists; otherwise it merely passes the **error in** value to **error out**. Refer to the *Error Structure* section of Chapter 1, *Introduction*, in this manual for a description of the **error in** cluster components.

y value is the y value evaluated by the **formula** (**x value**).

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

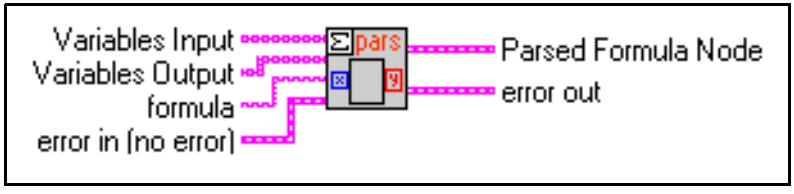
Note: ***This VI calculates only one value of a given 1D function. It is difficult to analyze the formula in the background. Therefore, if you want to calculate a collection of Eval Single-Variable Scalar VIs, use the Eval Single-Variable Array, Parse Formula String, and Eval Parsed Formula String VIs.***

Example

Refer to the previous Eval Single-Variable Array VI. The same example applies to this VI, except that the input **x value**, and the output **y value** are scalars.

Parse Formula Node

Analyzes the Eval Formula Node VI inputs and yields an intermediate state as an input for the Eval Parsed Formula Node VI.



Variables Input is a 1D array of input strings, each of them representing the name of a valid input variable.

Variables Output is a 1D array of output strings, each of them representing the name of a valid output variable.

formula is a string consisting of one or more formulas separated by semicolons.

error in (no error) describes error conditions occurring before this VI executes. If an error has already occurred, this VI returns the value of the **error in** cluster in **error out**. The VI executes normally only if no incoming error exists; otherwise it merely passes the **error in** value to **error out**. Refer to the *Error Structure* section of Chapter 1, *Introduction*, in this manual for a description of the **error in** cluster components.

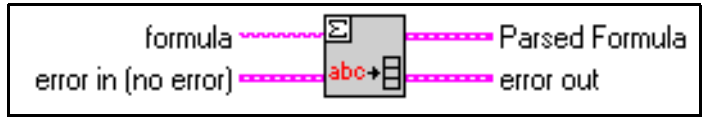
Parsed Formula Node is a cluster consisting of:

- **Y Values** is a 2D array of numbers representing a storage of detected and analyzed numbers of **formula**.
- **Tables** is a 3D array with 3 columns. The first column contains a code that stands for the operator, the other two contain codes that stand for the operands.
- **Variables Input Decode** is the intermediate and coded state of **Variables Input**.
- **Variables Output Decode** is the intermediate and coded state of **Variables Output**.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Parse Formula String

Analyzes a string as a formula and produces two numeric arrays. These arrays can be used by the Eval Parsed Formula String VI to produce X Values.



formula is a string representing the formula. The formula can contain any number of valid variables.

error in (no error) describes error conditions occurring before this VI executes. If an error has already occurred, this VI returns the value of the **error in** cluster in **error out**. The VI executes normally only if no incoming error exists; otherwise it merely passes the **error in** value to **error out**. Refer to the *Error Structure* section of Chapter 1, *Introduction*, in this manual for a description of the **error in** cluster components.

Parsed Formula is a cluster consisting of:

Y Values is a 1D array of numbers representing a storage of detected and analyzed numbers of **formula**.

Table is a 2D array with 3 columns. The first column contains a code that stands for the operator, the other two contain codes that stand for the operands.

This output can be wired directly to the coresponding input of the Eval Parsed Formula String VI.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

Usually, the Parse Formula String VI is used at the beginning of the calculation of function values. The Eval Parsed Formula String VI completes the calculation. This division works well if the analyzing process of the Parse Formula String VI finishes before the calculation processes. Keep this in mind for your own programming.

Substitute Variables

Substitutes a formula string by given rules. The rules have a parameter name - parameter content structure.

original formula is a string representing the formula, where parameter names stand for formulas.

Substitution Rules is an array of clusters describing the substitution rules.

- **parameter name** can have any length.
- **parameter content** must have a one-to-one relation with **parameter name**.

error in (no error).

formula after substitution is the final formula after all substitution rules are performed.

error out is the structure of the error handler is used as the output.

Note: ***A parameter name that begins with a capital letter E can produce unpredictable errors, if parts of the original string represent numbers like 1E-2. Avoid terms beginning with E in such cases.***

Example

The following inputs

original formula: ALPHA*cos(t) + beta

Substitution Rules: ALPHA → sin(t)
beta → 2 * t * exp(t)

result in

substitution formula:(sin(t))*cos(t) + (2*t*exp(t))

Data Visualization VIs

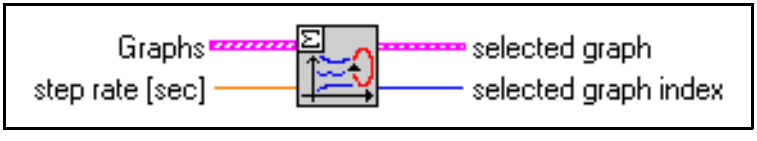
Chapter 3

The Data Visualization VIs are used for plotting and visualizing data in several different forms. These include advanced methods such as animation, contour plots, and surface cuts. They can be found in the `VISUALIZ.LLB` library.

Data Visualization VI Descriptions

Animate Graphs

Shows a collection of graphs and selects one graph from the whole collection of graphs. This VI works as an interactive tool.



Graphs is an input array of graphs to be animated. This array is made up of clusters each of which is composed of two arrays.

- **x values** is an array of the *x* values for a particular graph
- **y values** is an array of corresponding *y* values

step rate [sec] is the time between two consecutive graph representations in seconds.

selected graph is the selected graph. This cluster consists of two arrays.

- **x values** is an array of the *x* values for the selected graph
- **y values** is an array of corresponding *y* values

U32 **selected graph index** is the index of the graph you select, starting with 0.

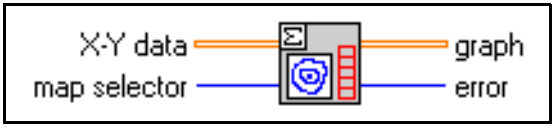
There are three possible modes of operation:

- **Automatic**—the VI sequentially displays each graph in the array, one after the other, at the speed given by **step rate [sec]**.
- **Forward**—you can manually step through each graph to be displayed, in the order of *increasing* array index.
- **Backward**—you can manually step through each graph to be displayed, in the order of *decreasing* array index.

Note: ***The* step rate [sec] *can be influenced by the user during runtime of the VI.***

Common Intensity Maps

Depicts an array of real values in a common intensity plot, with the data arranged on a rectangular grid.



[DBL] **X-Y data** is the 2D array of values $z = f(x, y)$, where x runs from 0 to x length and y runs from 0 to y length.

U16 **map selector** selects one item from a predefined list of Common Intensity Maps distributions. This list is shown in the table below. The value of **map selector** can range from 1 to 6.

[DBL] **graph** is a 2D array consisting of x and y values. You can wire this output directly to a graph for display.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes, if **map selector** is out of range.

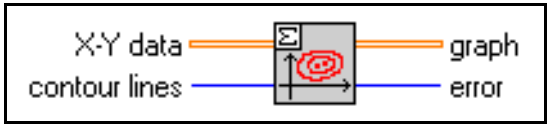
The following predefined color palettes are available.

Map Selector	Description	Number of Colors
1	256 fire	White, yellow, red, and blue
2	256 rainbow	Rainbow colors
3	256 log up	White and black in log up scale

Map Selector	Description	Number of Colors
4	256 log down	White and black in log down scale
5	32 color	Green, blue, and white
6	32 ice	Green and blue

Contour Plot

Depicts an array of real values on a contour plot, with the data arranged on a rectangular grid.



[DBL] **X-Y data** is the 2D array of values $z = f(x, y)$, where x runs from 0 to x length and y runs from 0 to y length.

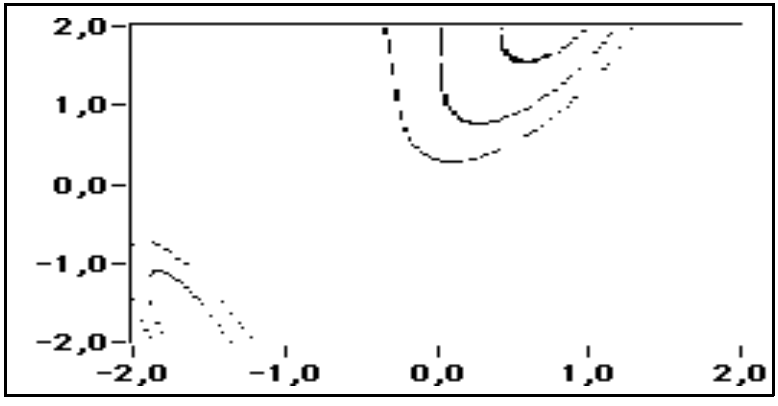
U32 **contour lines** is the number of **contour lines**. The default value is 5. This number can vary between 1 and 255.

[DBL] **graph** is a 2D array consisting of x and y values. You can wire this output directly to a graph for display.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes, if **contour lines** is out of range.

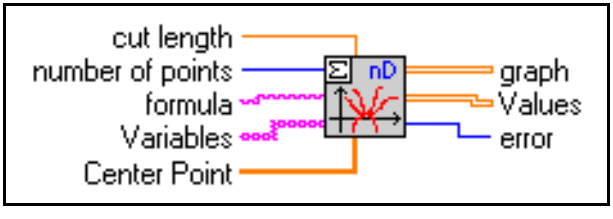
☞ **Note:** ***The axis scaling of the rectangular region can be influenced by attribute nodes.***

The function $f(x1, x2) = \sin(x1 * x1 - x2) - \cos(\sin(x2) - x1)$ is investigated in the interval (–2, 2) x (–2, 2) with the help of a contour plot as shown in the following figure.



Cut nD Surface

Depicts an *n*-dimension function in the form of *n* cuts along the *n* axes.



DBL **cut length** is the length of the n cuts.

U32 **number of points** is the number of points of each graph (cut).

abc **formula** is the formula of the *n*D function.

[abc] **Variables** is the *n* necessary variables to describe the *n*D function under observation.

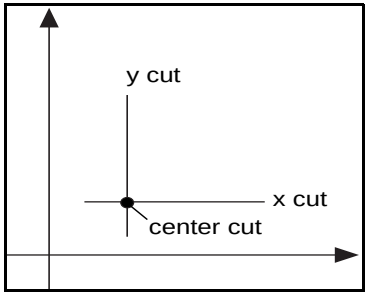
[DBL] **Center Point** is the center point of the *n* different cuts. (See Figure 3-1 in the Parametric Curve 3D VI description in this chapter.)

[DBL] **graph** is the resulting graph as combination of all cuts along the axes.

[DBL] **Values** is the same data as **graph** in the form of a 2D array of numbers.

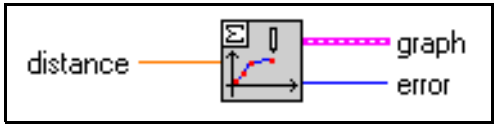
I32 **error**. See Appendix A, *Error Codes*, for a list of error codes. Fine tuning between **formula**, **Variables** and **Center Point** is necessary.

The following diagram explains the meaning of the center point and the cuts through the center points.



Draw Graph

Produces a graph interactively with the help of mouse actions.



distance is the minimal radial distance between two different points of the graph. In a neighborhood of radius **distance**, no other point of the graph can be placed.

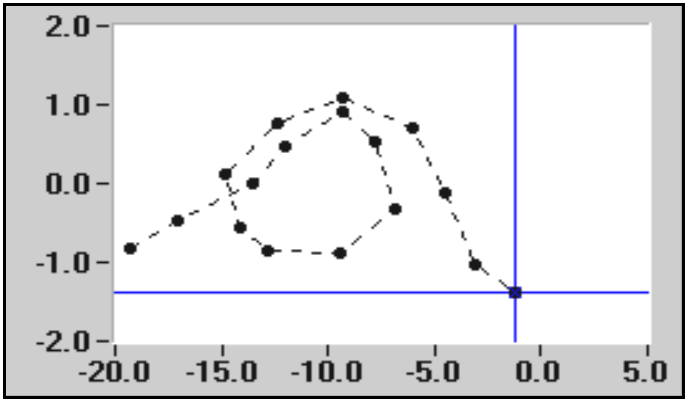
graph is a cluster consisting of two arrays holding the x and y data points drawn on the graph by the user.

error. See Appendix A, *Error Codes*, for a list of error codes, if **distance** ≤ 0 .

You can generate new points on the graph and delete existing points with this VI. Position the mouse cursor where you want to create or delete a point. Click the mouse and release it and the point is created or deleted. The graph stores the points in consecutive order. You can also clear all points by using the **CLEAR** button.

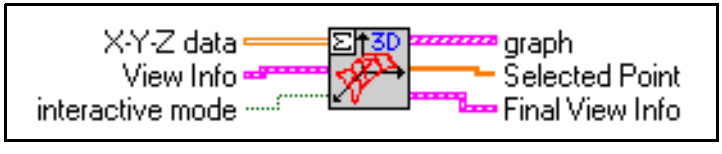
☞ **Note:** ***The value of* distance *can be adjusted by the user during runtime of the VI.***

The following diagram shows a generated graph. The cross hairs mark the current position of the cursor. You can add and delete points.



Mesh 3D

Depicts a surface under various conditions. You can completely control the position of the observer. This also lets you calculate the projections of a chosen point of the surface. This VI works as an interactive tool.



[DBL] **X-Y-Z data** is the visualized 2D data material of the curve. All points of the curve are given in the form (x, y, z).

View Info is a cluster of data that describes the viewpoint of the observer.

[DBL] **Origin** is the point the observer is looking at. (See Figure 3-1 for an illustration.) The default value is (0, 0, 0).

[DBL] **Axis** is the proportional factor of each axis. You can stretch or compress an axis independently of the other. The default value is (2, 2, 2).

psi describes the azimuth of the observer. This is the angle (in radians) of rotation about the z axis at the origin. When **psi** = 0, the viewpoint is along the x axis looking away from $x = 0$ toward the positive values of x and perpendicular to the y axis.

radius describes the distance from the origin to the actual position of the observer.

phi describes the elevation of the observer. This is the angle (in radians) made with the x-y plane by the line of sight from the observer to the origin.When **phi** = $\pi/2$, the viewpoint is directly above the z axis.

See Figure 3-1 for an explanation of psi, radius, and phi.

interactive mode. FALSE represents “not interactive,” TRUE stands for “interactive.” The default value is FALSE.

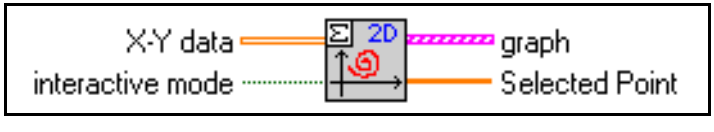
graph is an array of clusters, each consisting of to arrays containing the x and y values of the graph data determined by **View Info**.

Selected Point is the coordinates of exactly one selected point of the graph. This point can interactively be determined. The projections of this point on the x, y and z axis are also presented.

Final View Info is a collection of data describing the view point of the observer at the end of the interactive mode. The data in this cluster has the same structure as the data in the **View Info** cluster.

Parametric Curve 2D

Presents a 2D curve based on the data of an array. The VI can work in an interactively working mode, where the points of the curve can exactly be determined with the help of projections.



X-Y data is the 2D array of the x and y components of the curve.

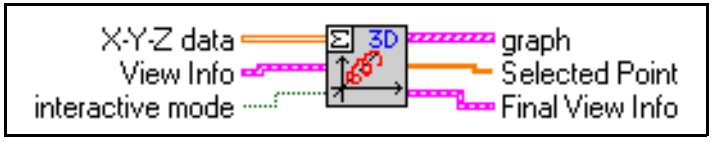
interactive mode. FALSE represents “not interactive,” TRUE stands for “interactive.” The default value is FALSE.

graph is the graphical presentation of the curve completed by the coordinates of a selected point on the curve.

Selected Point is the coordinates of the selected point of the curve.

Parametric Curve 3D

Depicts a curve in 3D under various conditions. You can control the position of the observer. This VI also lets you calculate the projections of a chosen point of the curve. This VI works as an interactive tool.



X-Y-Z data is the visualized 2D array of data of the curve. All points of the curve are given in the form (x, y, z).

View Info is a cluster of data that describes the view point of the observer.

Origin is the point the observer is looking at. (See Figure 3-1 for an illustration.) The default value is (0, 0, 0).

Axis is the proportional factor of each axis. One can stretch or compress an axis independently of the other. The default value is (2, 2, 2).

psi describes the azimuth of the observer. This is the angle (in radians) of rotation about the z axis at the origin. When **psi** = 0, the viewpoint is along the x axis looking away from $x = 0$ toward the positive values of x and perpendicular to the y axis.

radius describes the distance from the origin to the actual position of the observer.

phi describes the elevation of the observer. This is the angle (in radians) made with the x-y plane by the line of sight from the observer to the origin.When **phi** = $\pi/2$, the viewpoint is directly above the z axis.

interactive mode. FALSE represents “not interactive,” TRUE stands for “interactive.” The default value is FALSE.

graph is an array of clusters, each consisting of to arrays containing the *x* and *y* values of the graph data determined by **View Info**.

Selected Point is the coordinates of exactly one selected point of the graph. This point can interactively be determined. The projections of this point on the x, y and z axis are also presented.

Final View Info is a collection of data describing the view point of the observer at the end of the interactive mode. The data in this cluster has the same structure as the data in the **View Info** cluster.

Figure 3-1 illustrates the meaning of observer and origin.

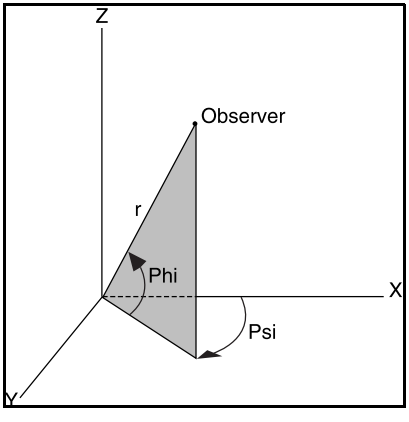
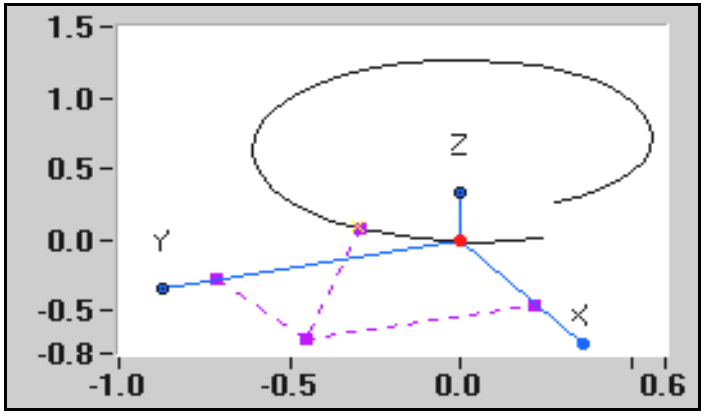


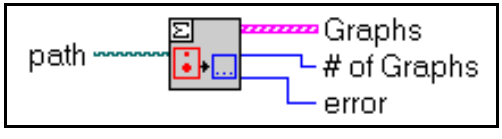
Figure 3-1. The Meaning of Observer and Origin

The following diagram shows the 3D curve $(\sin(t),\ \cos(t),\ t)$ from the view point of an outer observer. The projections are also shown.



Read Graphs

Reads graphs saved with the Write Graphs VI and displays the graph data.



path is the path to the file containing the graph data.

Graphs is the graphical representation of the saved graph data.

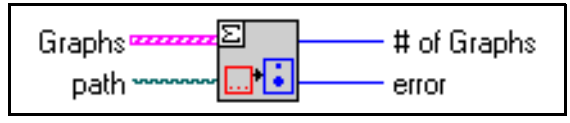
# of Graphs is the number of saved graphs.

error. See Appendix A, *Error Codes*, for a list of error codes. Common file errors can occur.

Note: ***See the description of the Write Graphs VI for details of the file format.***

Write Graphs

Writes a collection of graphs to a file in ASCII format.



Graphs is an array of clusters, each consisting of two arrays. These arrays consist of the *x* and *y* values for the graphs you are writing to a file.

path is the path to the location where you intend to write the file.You can provide a specific path, or wire this input to the Open/Create/Replace File VI.

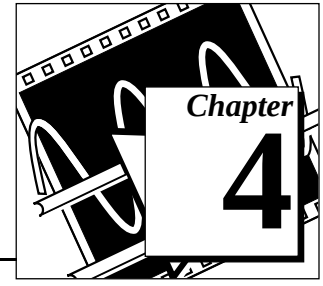
# of Graphs is number of graphs, which is determined in the **Graphs** cluster.

error. See Appendix A, *Error Codes*, for a list of error codes. Common file errors can also occur.

This VI produces a readable ASCII in a table with the following structure.

Row	Content	Discussion
1	Graphs	Identifies the table as a collection of graphs
2	Tab separated set of numbers, beginning with the number of graphs being written, followed by the number of points in each graph	Number of graphs and number of points in each of the graphs
3	The tab separated components of the first graph, beginning with the x components and followed by y components	Data of the first graph
4	The tab separated components of the second graph, beginning with the x components and followed by y components	Data of the second graph
. . .	. . .	. . .

Ordinary Differential Equation VIs

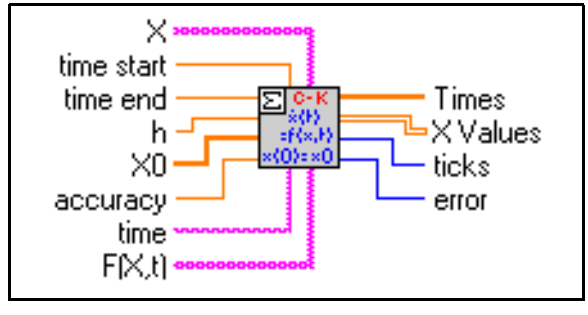


You can use these VIs to solve ordinary differential equations, both symbolically and numerically. They can be found in the `ODE.LLB` library.

Ordinary Differential Equation VI Descriptions

ODE Cash Karp 5th Order

The Cash Karp method solves ordinary differential equations with start conditions. The Cash Karp method works with an adaptive step rate and is computationally more efficient than the Euler method and the Runge Kutta method.



X is an array of strings of variables.

time start is the start point of the ODE. The default value is 0.

time end is the end point of the time interval under investigation. The default value is 1.0.

h is the step rate at the beginning. The default value is 0.1.

X0 is the vector of the start condition $x_{10},\ldots, x_{n0}$. There is a one-to-one relation between the components of **X0** and **X**.

accuracy controls the accuracy of the solutions. The default value is 0.0, which specifies the maximum deviation of the calculated solution from the actual solution.

time is the string denoting the time variable. The default variable is *t*.

F(X,t) is a 1D array of strings representing the right sides of the differential equations.

Times is a 1D array representing the time steps. The ODE Cash Karp method yields arbitrarily chosen time steps between **time start** and **time end**.

X Values is a 2D array of the solution vector $x_1,\ldots, x_n$. The top index runs over the time steps, the bottom index runs over the elements of $x_1,\ldots, x_n$.

ticks is the time in milliseconds for the whole calculation.

error. See Appendix A, *Error Codes*, for a list of error codes. Especially, the wrong inputs **X**, **X0**, and **F(X,t)** can produce errors.

The Cash Karp method is an embedded Runge Kutta formula and is based on a fifth order strategy (with six steps).

$$
\begin{aligned}
k_1 &= hF(X(t_n), t_n) \\
k_2 &= hF(X(t_n) + a_2 h, t_n + b_{21}k_1) \\
&\;\;\vdots \\
k_6 &= hF(X(t_n) + a_6 h, t_n + b_{61}k_1 + \ldots + b_{65}k_5)
\end{aligned}
$$

and $X(t_{n+1}) = X(t_n) + c_1 k_1 + \ldots + c_6 k_6$

$$X^*(t_{n+1}) = X(t_n) + c_1{}^* k_1 + \ldots + c_6{}^* k_6$$

with $t_{n+1} = t_n + h$

The

$a_2,\ldots,a_6$, $b_{21},\ldots,b_{65}$, $c_1,\ldots,c_6$, and $c_1{}^*,\ldots,c_6{}^*$

are fixed real numbers. This choice determines the quality of the method.

The actual step size h_{new} can be determined with the help of the **accuracy** value, the old step size **h** and the difference

$$\Delta = \left| X(t_{n+1}) - X^{*}(t_{n+1}) \right|$$

according to

$$h_{new} = h \left| \frac{accuracy}{\Delta} \right|^{\frac{1}{5}}$$

☞ **Note:** ***It may happen that the value of the last element in the* Times *indicator turns out to be greater than the value entered in the* time end *control. This is a property of the Cash Karp method. This method is very accurate, but you have no control of the step rate. In order to guarantee that the end point specified in* time end *is taken into consideration, the last step may turn out to be too long.***

Example

The following diagram shows the solution of the following system of ordinary differential equations in a 3D representation.

$$\frac{dx(t)}{dt} = 10(y(t) - x(t))$$

$$\frac{dy(t)}{dt} = x(t)(28 - z(t)) - y(t)$$

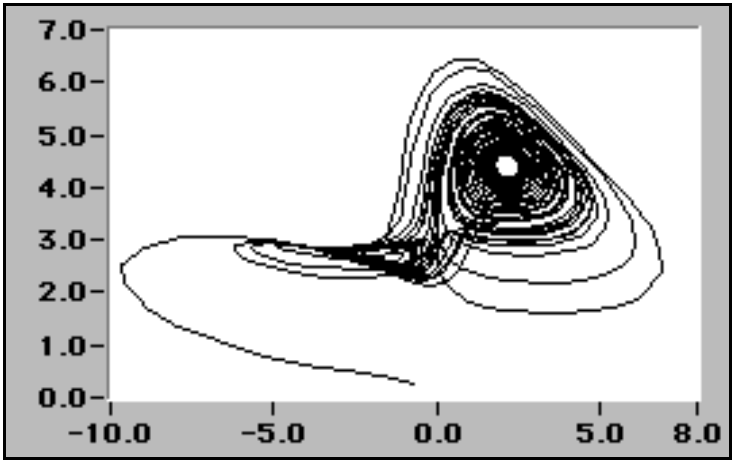
$$\frac{dz(t)}{dt} = x(t)y(t) - \frac{8}{3}z(t)$$

$$t \in [0, 40]$$

$$x(0) = 0.6$$

$$y(0) = 0.6$$

$$z(0) = 0.6$$



The above equations and boundary conditions are entered in the VI as:

time start: 0.00

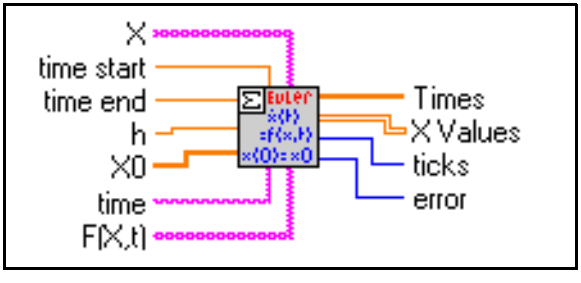
time end: 40.00

X0: [0.6, 0.6, 0.6]

F(X,t): [10*(y – x), x*(28 – z) – y, x*y – (8/3)*z]

ODE Euler Method

The Euler method solves ordinary differential equations with start conditions.



The general form of an ordinary differential equation (ODE) is

$$\text{differential equations} \quad \begin{cases} \dot{x}_1(t) = f_1(x_1(t),\ldots, x_n(t), t) \\ . \\ . \\ . \\ \dot{x}_n(t) = f_n(x_1(t),\ldots, x_n(t), t) \end{cases}$$

with

$$\text{starting conditions} \quad \begin{cases} x_1(t_0) = x_{10} \\ . \\ . \\ . \\ x_n(t_0) = x_{n0} \end{cases}$$

The functions $f_1,\ldots, f_n$, the numbers $x_{10},\ldots, x_{n0}$ and the start point $t = t_0$ are given. With the conventions

$F = (f_1,\ldots, f_n)$, $X(t) = (x_1(t),\ldots, x_n(t))$ and $X_0 = (x_{10},\ldots, x_{n0})$

we have

$$\text{vector form} \quad \begin{cases} \dot{X}(t) = F(X(t),t) \\ X(t_0) = X_0 \end{cases}$$

You have to determine functions **X** fulfilling the above equations.

X is an array of strings of variables.

time start is the start point of the ODE. The default value is 0.

time end is the end point of the time interval under investigation. The default value is 1.0.

h is the fixed step rate. The default value is 0.1.

X0 is the vector of the start condition $x_{10},\ldots, x_{n0}$. There is a one-to-one relation between the components of **X0** and **X**.

time is the string denoting the time variable. The default variable is *t*.

F(X,t) is a 1D array of strings representing the right sides of the differential equations.

Times is an array representing the time steps. The Euler method yields equidistant time steps between **time start** and **time end**.

X Values is a 2D array of the solution vector $x_1,\ldots, x_n$. The top index runs over the time steps, the bottom index runs over the elements of $x_1,\ldots, x_n$.

ticks is the time in milliseconds for the whole calculation.

error. See Appendix A, *Error Codes*, for a list of error codes. Especially, the wrong values of the inputs **X**, **X0**, and **F(X,t)** can produce errors.

The Euler method is the most basic and often useful strategy to solve ODEs. Beginning with t_0 and a fixed step rate *h* (usually relatively small) the new values

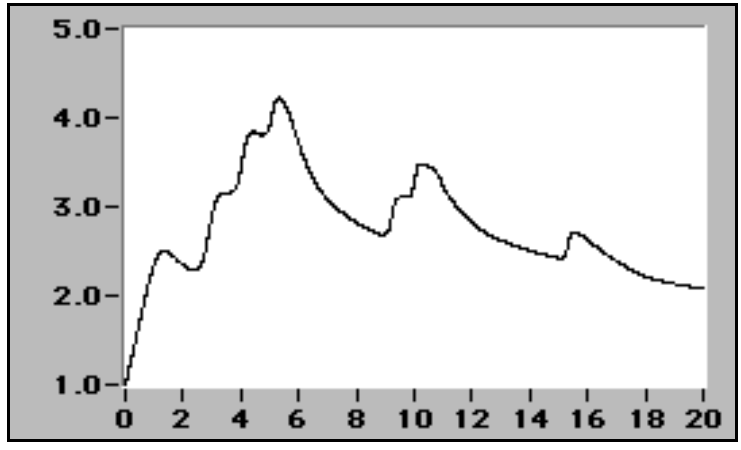
$$
\begin{aligned}
X(t_0 + h) &= X(t_0) + hF(X(t_0), t) \\
X(t_0 + 2h) &= X(t_0 + h) + hF(X(t_0 + h), t_0 + h) \\
&\vdots
\end{aligned}
$$

are calculated. This process stops, if **time start** + nh ≥ **time end**, where **time end** is the right endpoint of the time interval under investigation.

Example

The following diagram shows the solution of the following ordinary differential equation.

$$
\begin{aligned}
\frac{dx(t)}{dt} &= \sin(tx) + \operatorname{sinc}(t + x) + \cos(t - x) \quad t \in [0, 20] \\
x(0) &= 1
\end{aligned}
$$



The above equation and initial condition are entered on the front panel as:

time start: 0.00

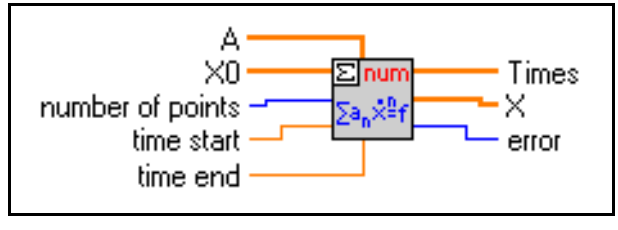
time end: 20.00

X0: 1.00

F(X,t): sin(t*x) + sinc(t + x) + cos(t – x)

ODE Linear nth Order Numeric

Solves an *n*th order homogeneous linear differential equation with constant coefficients in numeric form.



[DBL] **A** in the matrix of coefficients of the different derivatives of a function *x*(*t*), starting with the coefficient of the lowest order term. The coefficient of the highest order derivative is assumed to be equal to 1.0 and does not need to be entered.

[DBL] **X0** is a vector of the start conditions of the *n*th order differential equation, starting with the start condition of the lowest order derivative.

U32 **number of points** is the number of equidistant time points between **time start** and **time end**. The default value is 10.

DBL **time start** is the start point of the nth order linear system. The default value is 0.0.

DBL **time end** is the end point of the time interval under investigation. The default value is 1.0.

[DBL] **Times** is an array representing the time steps. The method yields equidistant time steps between **time start** end **time end**.

[DBL] **X** is the vector of the solution *x* at the equidistant time points.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes. Especially, discrepancies between the length of **X0** and **A** can produce errors.

Consider the *n*th order linear homogeneous differential equation

$$
\begin{aligned}
x^{(n)} + a_{n-1}x^{(n-1)} + \ldots + a_1 x^{(1)} + a_0 x &= 0 \\
with \quad x(0) &= x_{00} \\
x^{(1)}(0) &= x_{10} \\
&\vdots \\
x^{(n-1)}(0) &= x_{n-10}
\end{aligned}
$$

(0 represents the more general value of **time start**.) There is a strong connection between the equation

$$x^{(n)} + a_{n-1}x^{(n-1)} + \ldots + a_1 x^{(1)} + a_0 x = 0$$

and the zero finding problem

$$z^n + a_{n-1}z^{n-1} + \ldots + a_1 z + a_0 = 0$$

The *n* zeroes of the last equation determine the structure of the solution of the ODE. If we have *n* distinct complex zeroes $\lambda_1, \ldots, \lambda_n$, the general solution of the *n*th order differential equation can be expressed by

$$x(t) = \beta_1 \exp(\lambda_1 t) + \ldots + \beta_n \exp(\lambda_n t)$$

The unknowns $\beta_1,\ldots,\beta_n$ can be determined by the start condition

$$x(0) = \beta_1 + \ldots + \beta_n$$

$$x^{(1)}(0) = \beta_1\lambda_1 + \ldots + \beta_n\lambda_n$$

.

.

.

$$x^{(n-1)}(0) = \beta_1\lambda_1^{n-1} + \ldots + \beta_n\lambda_n^{n-1}$$

Note: ***The case of repeated eigenvalues $\lambda_1,\ldots,\lambda_n$ is more complex and is not treated here. An error code of*** `-23017` ***is given if this happens.***

Note: ***By convention, the value of the highest coefficient is taken as 1.0, and does not need to be entered in the* A *control. The other coefficients are entered starting with the lowest order coefficient.***

Example

To solve the differential equation

$x'' - 3\,x' + 2\,x = 0$

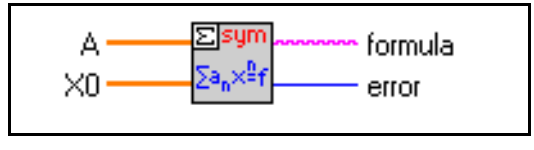
with the I.C. as with $x(0) = 2$ and $x'(0) = 3$

enter

A = [2, –3] and **X0** = [2, 3].

ODE Linear nth Order Symbolic

Solve an *n*th order homogeneous linear differential equation with constant coefficients in symbolic form.



A inputs the coefficients of the consecutive derivatives of a function $x(t)$, starting with the coefficient of the lowest order term. The coefficient of the highest order derivative is assumed to be equal to 1.0 and does not need to be entered.

X0 is a vector of the start conditions of the nth order differential equation, starting with the coefficient of the lowest order derivative.

formula is the symbolic solution.

error. See Appendix A, *Error Codes*, for a list of error codes. Discrepancies between the length of **X0** and **A** can produce errors.

The general solution has the following form (See the ODE Linear *n*th Order Numeric VI description in this chapter.):

$$x(t) = \beta_1 \exp(\lambda_1 t) + \ldots + \beta_n \exp(\lambda_n t)$$

with complex $\beta_1, \ldots, \beta_n$ and $\lambda_1, \ldots, \lambda_n$. But all inputs are real, and thus the solution also has this property. As a consequence, the symbolic solution is a linear combination of exp-, sin-, and cos-functions with real coefficients.

Note: ***Only the case of pairwise different $\lambda_1, \ldots, \lambda_n$ is treated. For the case of repeated eigenvalues, an error code of `-23017` is given.***

Note: ***By convention, the value of the highest coefficient is taken as 1.0, and does not need to be entered in the* A *control. The other coefficients are entered starting with the lowest order coefficient.***

Example

To solve the differential equation

$x'' - 3x' + 2x = 0$

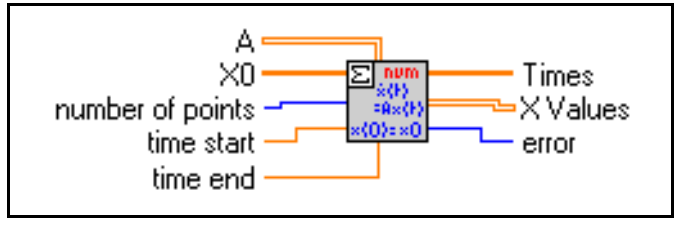
with the I.C. as with $x(0) = 2$ and $x'(0) = 3$

enter

A = [2, –3] and **X0** = [2, 3]

ODE Linear System Numeric

Solves an *n*-dimension homogeneous linear system of differential equations with constant coefficients, for a given start condition. The solution is based on the determination of the eigenvalues and eigenvectors of the underlying matrix **A**. The solution is given in numeric form.



[DBL] **A** is the *n* by *n* matrix describing the linear system.

[DBL] **X0** is the *n* vector describing the start condition.

U32 **number of points** is the number of equidistant time points between **time start** and **time end**. The default value is 10.

DBL **time start** is the start point of the linear system, that is, the time with **X**(**time start**) = **X0**. The default value is 0.0.

DBL **time end** is the end point of the time interval under investigation. The default value is 1.0.

[DBL] **Times** is an array representing the time steps. The method yields equidistant time steps between **time start** and **time end**.

[DBL] **X Values** is the matrix of the solution **X** at the equidistant time points.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes. Especially, discrepancies between the dimensions of **X0** and **A** can produce errors.

Linear systems can be described by

$$\frac{dX(t)}{dt} = AX(t)$$

$$X(0) = X_0$$

if **time start** = 0.

Here $X(t) = (x_0(t),\ldots, x_n(t))$ and **A** represents a *n* by *n* real matrix. The linear system can be solved by the determination of the eigenvalues and eigenvectors of **A**. Let *S* be the

set of all eigenvectors spanning the whole *n*-dimensional space. The transformation $Y(t) = SX(t)$ yields

$$\frac{dY(t)}{dt} = SAS^{-1}Y(t)$$
$$Y(0) = SX_0$$

The matrix SAS^{-1} has diagonal form, so that the solution is obvious. The solution *X(t)* can be determined by back-transformation $X(t) = S^{-1}Y(t)$.

☞ **Note:** ***No* ticks *output is implemented because the essential operation is the calculation of eigenvectors and eigenvalues of the matrix A. This operation is negligible for relatively small dimensions of A.***

☞ **Note:** ***This VI works fine for almost all cases of real matrices A which can have repeated eigenvalues, conjugate complex eigenvalues, and so on. The exception is the case of a singular eigenvector matrix, that is, a matrix in which the eigenvectors do not span the entire space. An error of `-23016` is given if the eigenvector matrix is singular.***

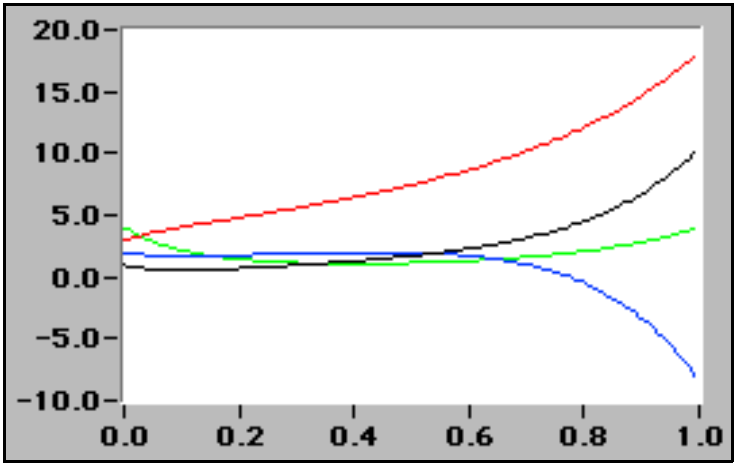
Example

The following diagram shows the four components of the solution of the linear differential equation described by the following system:

$$\begin{bmatrix} \frac{dx_1(t)}{dt} \\ \frac{dx_2(t)}{dt} \\ \frac{dx_3(t)}{dt} \\ \frac{dx_4(t)}{dt} \end{bmatrix} = \begin{bmatrix} -7 & -6 & 4 & -1 \\ -6 & 2 & 1 & -2 \\ 4 & 1 & 0 & 2 \\ -1 & -2 & 2 & -7 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix}$$

with

$$x_1(0) = 1$$
$$x_2(0) = 2$$
$$x_3(0) = 3$$
$$x_4(0) = 4$$



Enter the equations above on the front panel as shown:

A: [–7, –6, 4, 1; –6, 2, 1, –2; 4, 1, 0, 2; –1, –2, 2, –7]

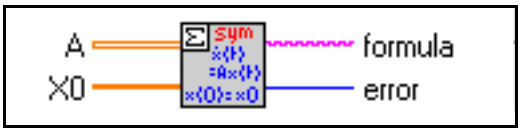
X0: [1, 2, 3, 4]

time start: 0.00

time end: 1.00

ODE Linear System Symbolic

Solves an *n*-dimension linear system of differential equations with a given start condition. The solution is based on the determination of the eigenvalues and eigenvectors of the underlying matrix. The solution is given in symbolic form.



[DBL] **A** is the *n* by *n* matrix describing the linear system.

[DBL] **X0** is the *n* vector describing the start condition.

abc **formula** is a string with the solution of the linear system in the standard formula notation of LabVIEW. The solution vector elements are separated by carriage return.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes. In particular, discrepancies between the dimensions of **X0** and **A** can produce errors.

☞ **Note:** ***This VI works fine for almost all cases of real matrices A which can have repeated eigenvalues, conjugate complex eigenvalues, and so on. The exception is the case of a singular eigenvector matrix, that is, a matrix in which the eigenvectors do not span the whole space. An error of*** `-23016` ***is given if the eigenvector matrix is singular.***

Example

The linear differential equation described by the following system:

$$\begin{bmatrix} \frac{dx_1(t)}{dt} \\ \frac{dx_2(t)}{dt} \\ \frac{dx_3(t)}{dt} \\ \frac{dx_4(t)}{dt} \end{bmatrix} = \begin{bmatrix} -7 & -6 & 4 & -1 \\ -6 & 2 & 1 & -2 \\ 4 & 1 & 0 & 2 \\ -1 & -2 & 2 & -7 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix}$$

with

$x_1(0) = 1$

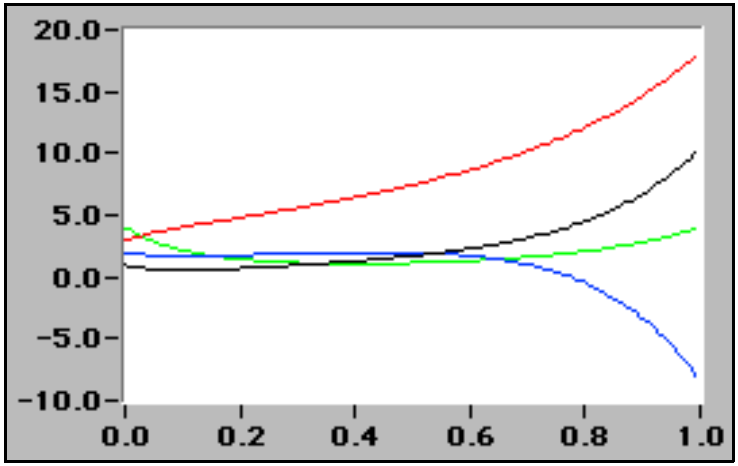
$x_2(0) = 2$

$x_3(0) = 3$

$x_4(0) = 4$

has the solution

+1.62*exp(–12.46*t) – 1.28*exp(–6.30*t) + 0.63*exp(1.34*t) + 0.04*exp(5.42*t)
+0.84*exp(–12.46*t) – 0.29*exp(–6.30*t) + 1.51*exp(1.34*t) – 0.06*exp(5.42*t)
–0.73*exp(–12.46*t) + 0.01*exp(–6.30*t) + 3.69*exp(1.34*t) + 0.02*exp(5.42*t)
+0.87*exp(–12.46*t) + 2.67*exp(–6.30*t) + 0.45*exp(1.34*t) + 0.01*exp(5.42*t)



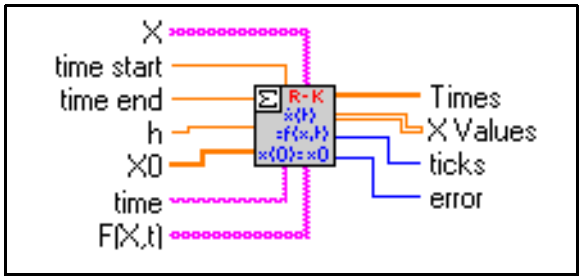
Enter the equations above on the front panel as follows:

A: [–7, –6, 4, 1; –6, 2, 1, –2; 4, 1, 0, 2; –1, –2, 2, –7]

X0: [1, 2, 3, 4]

ODE Runge Kutta 4th Order

The Runge Kutta method solves ordinary differential equations with start conditions. The Runge Kutta method works with a fixed step rate but with a higher degree of accuracy than the common Euler method.



X is an array of strings of variables.

time start is the start point of the ODE. The default value is 0.0.

time end is the end point of the time interval under investigation. The default value is 1.0.

h is the fixed step rate. The default value is 0.1.

X0 is the vector of the start condition $x_{10},\ldots, x_{n0}$. There is a one-to-one relation between the components of **X0** and **X**.

time is the string denoting the time variable. The default variable is *t*.

F(X,t) is an array of strings representing the right sides of the differential equations.

Times is an array representing the time steps. The Runge Kutta method yields equidistant time steps between **time start** and **time end**.

X Values is a 2D array of the solution vector $x_1,\ldots, x_n$. The top index runs over the time steps, the bottom index runs over the elements of $x_1,\ldots, x_n$.

ticks is the time in milliseconds for the whole calculation.

error. See Appendix A, *Error Codes*, for a list of error codes. Especially, wrong inputs **X**, **X0** and **F(X,t)** can produce errors.

The Runge Kutta method of 4th order works as a five stage process, more precisely

$$k_1 = hF(X(t_n), t_n)$$

$$k_2 = hF\left(X(t_n) + \frac{k_1}{2}, t_n + \frac{h}{2}\right)$$

$$k_3 = hF\left(X(t_n) + \frac{k_2}{2}, t_n + \frac{h}{2}\right)$$

$$k_4 = hF(X(t_n) + k_3, t_n + h)$$

and $X(t_{n+1}) = X(t_n) + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}$

with $t_{n+1} = t_n + h$

The method ends, if $t_n \geq$ **time end**.

Example

The following diagram shows the solution of the following system of ordinary differential equations:

$$\frac{dx(t)}{dt} = 10(y(t) - x(t))$$

$$\frac{dy(t)}{dt} = x(t)(28 - z(t)) - y(t)$$

$$\frac{dz(t)}{dt} = x(t)y(t) - \frac{8}{3}z(t)$$

$$t \in [0, 50]$$

$$x(0) = 1$$

$$y(0) = 1$$

$$z(0) = 1.$$

Enter these equations on the front panel as shown:

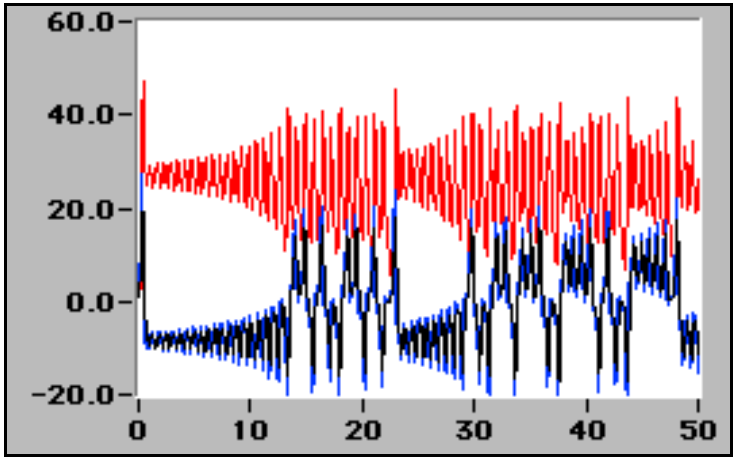
time start: 0.00

time end: 50.00

X: [x, y, z]

X0: [1, 1, 1]

F(X,t): [10*(y – x), x*(28 – z) – y, x*y – (8/3)*z]



Note: ***Even though there are actually three solutions, a first glance at the graph almost seems to show only two solutions. This is because the solutions for x and y are very similar, so they almost overlap.***

Zero Finder VIs

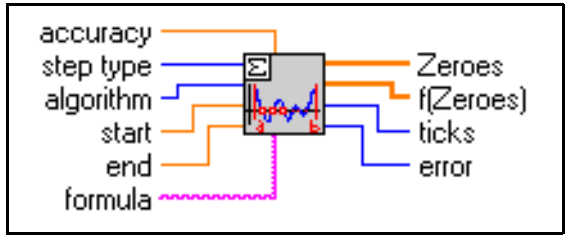
Chapter 5

These VIs find the zeros of 1D or *n*-dimension, linear or nonlinear functions (or system of functions). They are found in the library ZERO.LLB.

Zero Finder VI Descriptions

Find All Zeros of f(x)

Determines all zeros of a 1D function in a given interval.



DBL — **accuracy** controls the accuracy of the determined zeros. The default value is 1E-8, which specifies the maximum deviation of the calculated solution from the actual solution.

U16 — **step type**. A value of 0 represents a fixed number of function values, a value of 1 represents an optimal step size. In general, the second value leads to more accurate zeros.

U16 — **algorithm**. Use 0 for the Ridders method; 1 for the Newton Raphson method. The default value is 0.

DBL — **start** is the start point of the interval under investigation. The default value is 0.0.

DBL — **end** is the end point of the interval. The default value is 1.0.

abc — **formula** is a string describing the function.

[DBL] — **Zeros** are the determined zeros of **formula**.

[DBL] **f(Zeros)** are the function values of **zeros**. Usually, these values are very close to 0.

U32 **ticks** is the time in milliseconds to analyze the formula and to calculate the **zeros**.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes.

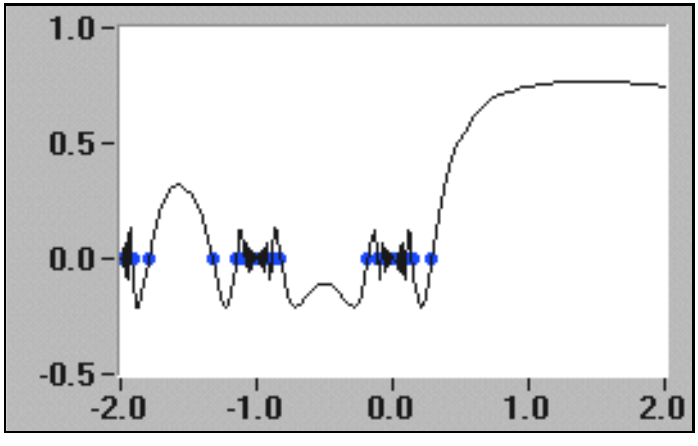
Example

The following diagram shows the graph and the zeros of sin(sinc(gamma(*x*))) in the interval (–2, 2). These values are entered on the front panel as:

start: –2.00

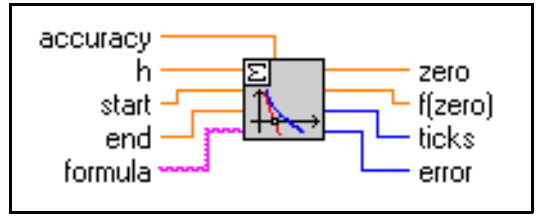
end: 2.00

formula: sin(sinc(gamma(x)))



Newton Raphson Zero Finder

Determines a zero of a 1D function close to two points with the help of the derivative of this 1D function. The two values form a search limit for the unknown zero of the 1D function.



DBL **accuracy** controls the accuracy of the zero determination. The default value is 1E-8, which specifies the maximum deviation of the calculated solution from the actual solution.

DBL **h** is the delta value to calculate the derivative of the given **formula**. The default value is 1E-8.

DBL **start** is the first point close to the **zero** that the algorithm is trying to determine. The default value is 0.0.

DBL **end** is the second point close to the **zero** that the algorithm is trying to determine. The default value is 1.0.

abc **formula** is a string representing the 1D function **zero**.

DBL **f(zero)** is the function value at the point given by **zero**.

DBL **zero** is only a good approximation for the exact value.

U32 **ticks** is the time in milliseconds for the calculation.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes.

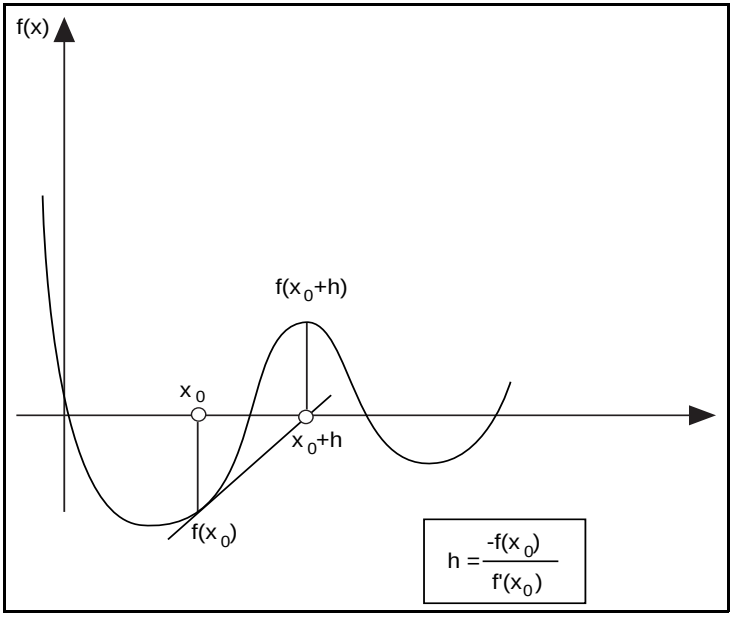
Let *f* be the given function. Use a method that combines the simple midpoint strategy and the Newton strategy

$$\text{midpoint strategy: } x_{new} = \frac{x_1 + x_2}{2}$$

$$\text{Newton strategy: } x_{new} = x_1 - \frac{f(x_1)}{f'(x_1)}$$

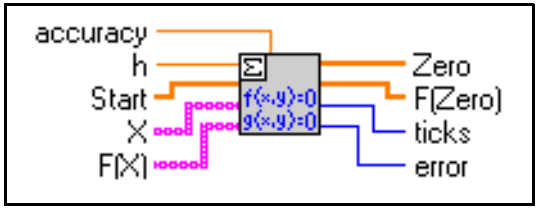
where x_1 and x_2 are given guesses with $f(x_1)*f(x_2) < 0$.

The following figure demonstrates the Newton strategy.



Nonlinear System Single Solution

Determines the solutions of nonlinear systems of equations in *n* dimensions beginning with a starting point in *n* dimensions.



DBL **accuracy** controls the accuracy of the determined zeros. The default value is 1E-8, which specifies the maximum deviation of the calculated solution from the actual solution.

DBL **h** is a small distance to calculate derivatives. The default value is 1E-8.

[DBL] **Start** is the start point in *n*D.

X is an array of strings defining the independent variables.

F(X) is an array of strings defining the functions in *n*D.

Zero is the determined zero of **F(X)**.

F(Zero) are the function values of the **Zero**. Usually, these values are very close to 0.

ticks is the time in milliseconds to analyze the formula and to produce the **Zero.**

error. See Appendix A, *Error Codes*, for a list of error codes.

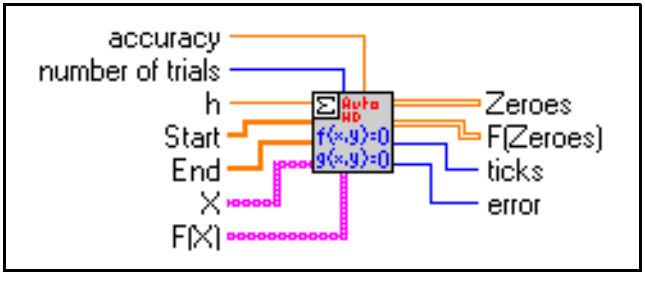
Let F be the n-dimension function and let X be a given point in n-dimensions.

Furthermore, let $f = \frac{1}{2}F*F$.

The algorithm is looking for such a vector P that $F(X + dP) \leq F(X)$ for all $0 \leq d \leq 1$. In a second step, an appropriate value d^* is calculated, so that $F(X + d^*P)$ is considerably smaller than $F(X)$. This process is repeated until $F(X) \approx 0$ is reached. What follows is an approximation for $F(X) = 0$.

Nonlinear System Solver

Determines a set of solutions of a nonlinear system of equations in *n*-dimensions beginning with a randomly chosen start point in *n*-dimensions.



accuracy controls the accuracy of the determined zeros. The default value is 1E-8.

number of trials is the elaborate number of randomly chosen start points. The algorithm starts with these points and is looking for zeros close to these points. The default value is 5.

h is a small distance to calculate derivatives. The default value is 1E-8.

Start is an array describing the left corner of the nD interval. The randomly chosen start points of the zero-finding algorithm can be found in the *n*-dimension rectangle spanned by **Start** and **End**.

End is an array describing the right corner of the *n*-dimension interval. The randomly chosen start points of the zero-finding algorithm can be found in the *n*-dimensional rectangle spanned by **Start** and **End**.

X is an array of strings defining the independent variables.

F(X) is an array of strings defining the functions in nD.

Zeros are the determined zeros of **F(X)**.

F(Zeros) are the function values of **zeros**. Usually, these values are very close to 0.

ticks is the time in milliseconds to analyze the formula and to produce the **zeros**.

error. See Appendix A, *Error Codes*, for a list of error codes.

Example

This algorithm is based on Nonlinear System Single Solution VI. This VI determines that the nonlinear system

$$\begin{aligned} 2*x + 3*y + z*z - 6 &= 0 \\ -4*x + y*y - 4*z + 7 &= 0 \\ x*x + y + z - 3 &= 0 \end{aligned}$$

has two solutions
(1.0000, 1.0000, 1.0000) and (–0.4050, 0.5931, 2.2429).

The above equations, with appropriate **Start** and **End** values are entered into the VI front panel as:

Start: [–1, –1, –1]

End: [4, 4, 4]

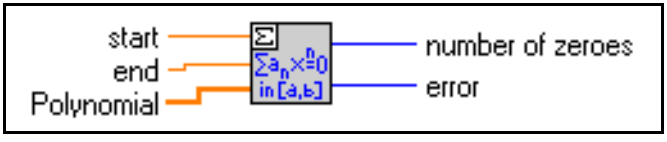
X: [x, y, z]

F(X): [2*x + 3*y + z*z – 6, –4*x + y*y – 4*z + 7, x*x + y + z – 3]

Note: ***Only the left side of the above equations needs to be entered into F(X). The VI assumes that the right side is zero.***

Polynomial Real Zero Counter

Calculates the number of zeros of a given real polynomial in a real interval without determining of the values of these zeros.



DBL **start** is the leftmost point of the interval. The default value is 0.0.

DBL **end** is the rightmost point of the interval. The default value is 0.0.

[DBL] **Polynomial** is an array representing the polynomial under investigation. The first element of this array relates to the constant coefficient of the **Polynomial**.

U32 **number of zeros** is the exact number of zeros of the **Polynomial** in the interval (**start**,**end**).

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes. When **start** > **end**, the application interprets it as an error situation.

The Sturm algorithm has to deal with two situations. Let $p(x)$ be a real polynomial in x, and let $p'(x)$ be the derivative of $p(x)$. If $d(x)$ denotes the greatest common divisor of $p(x)$ and $p'(x)$, $d(x) = gcd(p(x), p'(x))$, so $p(x)$ has multiple zeros, if and only if $d(x)$ is a nonconstant polynomial. In other words, the polynomial $p(x)/d(x)$ has only single zeros. Repeating this idea, you can combine the given polynomial $p(x)$ as the product of simple polynomials, each of them with single real zeros. The number of zeros of $p(x)$ is equal to the sum of all zeros of the defined simple polynomials having only single zeros.

If you need to determine the number of zeros of a real polynomial $p(x)$ with the single zero property, use the following Euclidean Algorithm.

$$
\begin{aligned}
p(x) &= q_1(x)p_1(x) - p_2(x) \\
p_1(x) &= q_2(x)p_2(x) - p_3(x) \\
&\vdots \\
p_{r-2}(x) &= q_{r-1}(x)p_{r-1}(x) - p_r(x) \\
p_{r-1}(x) &= q_r(x)p_r(x)
\end{aligned}
$$

where $p_1(x) = p'(x)$

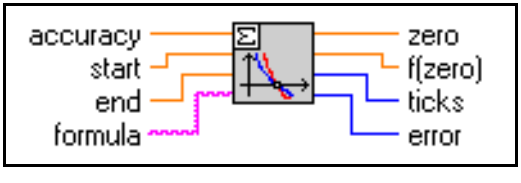
The Sturm's chain $(p(x), p_1(x),\ldots, p_r(x))$ determines two values W(**start**) and W(**end**). $W(x)$ is the number of sign changes of the chain $(p(x), p_1(x),\ldots, p_r(x))$. The number of all zeros of $p(x)$ is exactly equal to W(**end**) – W(**start**).

Note: ***The algorithm makes extensive use of polynomial operations like multiplication, division, and greatest common divisor (gcd). You can select these subVIs and reuse them in your own projects.***

Note: ***If you are interested in all real zeros of a real polynomial, the choices*** **start** $= -(|a_0| + \ldots + |a_n|)$ and **end** $= +(|a_0| + \ldots + |a_n|)$***, where a_i denotes the coefficients, are sufficient to determine this number.***

Ridders Zero Finder

Determines a zero of a 1D function in a given interval. The function has to be continuous and has to have different signs at the end points of the interval.



DBL **accuracy** controls the accuracy of the zero determination. The default value is 1E-8.

DBL DBL **start** is the leftmost point of the given interval. The default value is 0.0.

end is the rightmost point of the given interval. The default value is 1.0.

formula is a string representing the 1D function.

zero is the determined zero of **formula**.

f(zero) is the function value at the point **zero**. Usually, the value of **zero** is only a good approximation for the exact value.

ticks is the time in milliseconds for the whole calculation.

error. See Appendix A, *Error Codes*, for a list of error codes. When **start** > **end**, the application interprets it as an error condition. The function values at the points **start** and **end** must have different signs to guarantee the existence of a zero in (**start,end**).

Given the function $f(x)$ with $f(a)*f(b) < 0$.

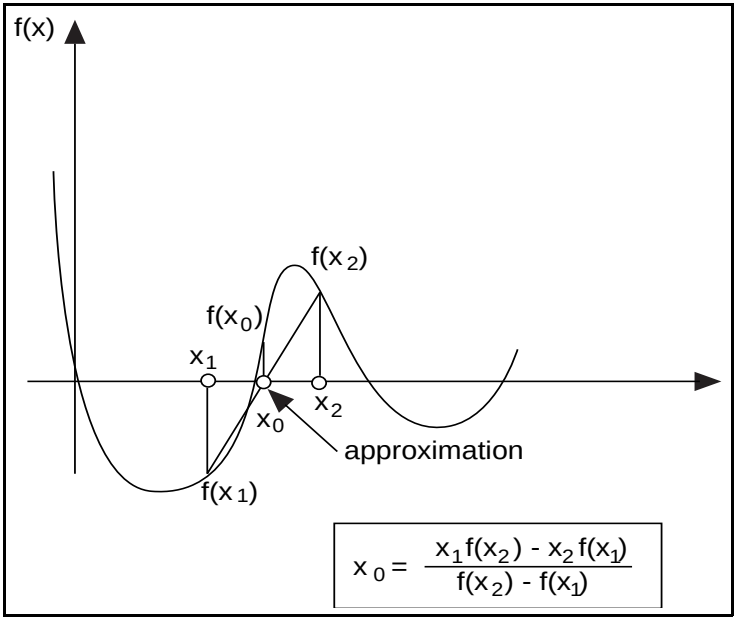
Ridders method determines $c = (a + b)/2$ and calculates the new guess

$$c_{new} = c + (c - a)\frac{sign(f(a) - f(b))f(c)}{\sqrt{f(c)^2 - f(a)f(b)}}$$

The triplets start, c_{new}, end are the base for the new iteration, depending on whether $f(\text{start})*f(c_{new}) < 0$ or $f(c_{new})*f(\text{end}) < 0$. The algorithm stops, if $|a - b| <$ **accuracy**.

Ridders method is very fast and reliable.

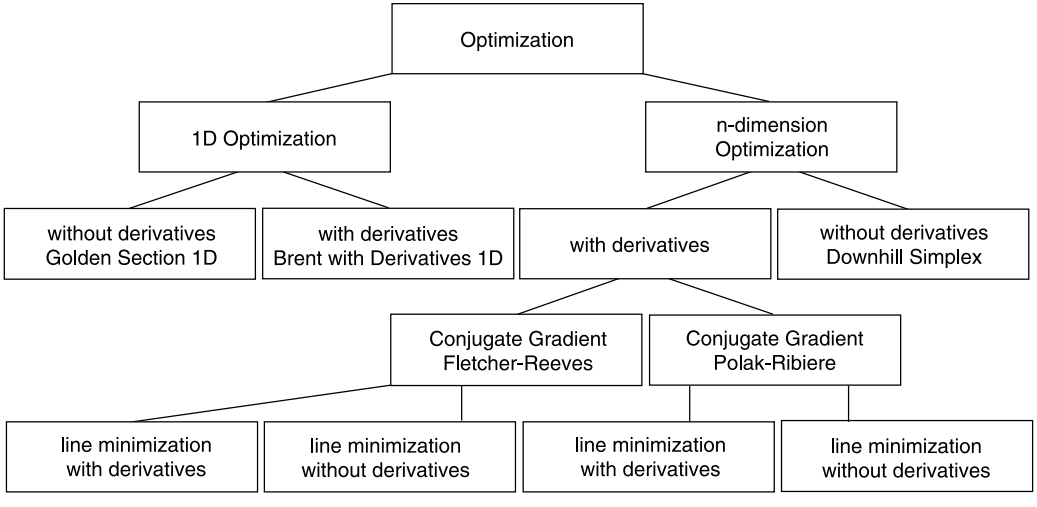
Ridders method is a generalization of the depicted estimation of a zero, as shown in the following illustration.



Optimization VIs

Chapter 6

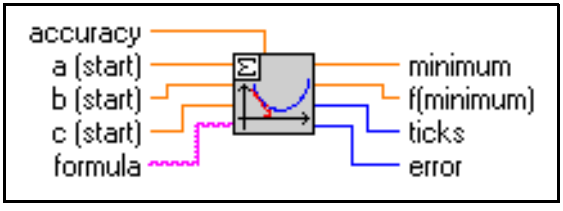
Use these VIs to determine local minima and maxima of real 1D or *n*-dimension functions. You can choose between optimization algorithms based on derivatives of the function and algorithms working without these derivatives. You can also use special methods like Linear Programming, Levenberg Marquardt in symbolic form, Pade, and Chebyshev Approximation. An overview of the optimization routines is shown in the following illustration.



Optimization VI Descriptions

Brent with Derivatives 1D

Determines a local minimum of a given 1D function in a given interval.



DBL **accuracy** controls the accuracy of the determined minimum of **formula**. The method stops, if two consecutive approximations differ not more than the value of **accuracy**.

DBL **a (start)** is the left point of the bracketing interval. The default value is 0.0.

DBL **b (start)** is the middle point of the bracketing interval. The default value is 0.0.

DBL **c (start)** is the right point of the bracketing interval. The default value is 0.0.

abc **formula** is a string describing the function under investigation. The Parser VIs check the syntax of this string.

DBL **minimum** is the determined local minimum of **formula**.

DBL **f(minimum)** is the function value of the determined local minimum.

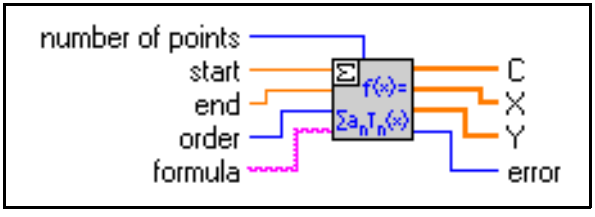
U32 **ticks** is the time in milliseconds for the whole calculation.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes.

Refer to the Golden Section 1D VI description for the definition of a bracketing interval. Here the derivative of **formula** is used. This leads to a faster algorithm, because the new bracketing interval has better properties than those of the Golden Section 1D VI.

Chebyshev Approximation

Determines a given function using Chebyshev polynomials.



U32 **number of points** is the number of equidistant points in the interval (**start**,**end**). The default value is 10.

DBL **start** is the start point of the interval. The default value is 0.0.

DBL **end** is the end point of the interval. The default value is 1.0.

U32 **order** is the degree of the Chebyshev approximation (that is, the number of different Chebyshev polynomials $T_0(x)$, $T_1(x)$,..., $T_n(x)$ describing the **formula)**. The default value is 3.

abc **formula** is a string describing the function under investigation. The Parser VIs check the syntax of this string.

[DBL] **C** is an array of coefficients belonging to $T_0(x)$, $T_1(x)$,... .

[DBL] **X** is the x values dividing (**start**,**end**) in equidistant subintervals.

[DBL] **Y** is the y values of the Chebyshev polynomial at points **X**.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes.

Let n be a given natural number. The function $f(x)$ can approximately be represented by

$$f(x) = c_0T_0(x) + \ldots + c_nT_n(x)$$

where $T_0(x),\ldots, T_n(x)$ are the first Chebyshev polynomials.

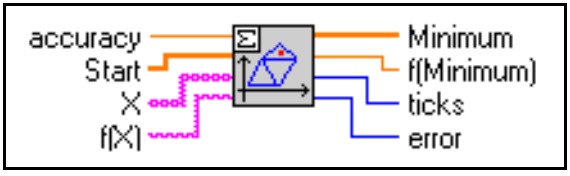
The $c_0,\ldots, c_n$ can be calculated as sums of the form

$$\sum_{k=1}^{n} f(x_k)T_j(x_k)$$

where $x_k = \cos\left(\frac{\pi k}{n}\right)$ for $k = 1,\ldots, n$

Downhill Simplex nD

Determines a local minimum of a function of *n* independent variables with the Downhill Simplex method.



accuracy controls the accuracy of the minimum. The method stops if two consecutive approximations differ no more than the value of **accuracy**. The default value is 1E-8.

Start is an array of points at which the optimization process is starting. These points form a simplex in *n*-dimension.

X is an array of strings representing the x variables.

f(X) is the string representing the function of the **X** variables.

Minimum is the determined local minimum in *n*-dimension.

f(Minimum) is the function value of **f(X)** at the determined minimum.

ticks is the time in milliseconds for the whole calculation.

error. See Appendix A, *Error Codes*, for a list of error codes. The Parser VIs check the syntax of the inputs.

The Downhill Simplex algorithm, also called the Nelder and Mead method, works without partial derivatives. The Downhill Simplex algorithm consists of catching the minimum of the function, **f(X),** with the help of simple geometrical bodies, specifically a simplex. A simplex in 2D is a triangle, a simplex in 3D is a tetrahedron and so on. You must have (n + 1) starting points, each of dimension n, forming the initial simplex. The user must enter only one point of these (n + 1). The (n + 1) dimensional simplex is automatically constructed. For the example given below ($f(x,y) = x\text{^}2 + y\text{^}2$), you must enter two numbers (describing exactly one point in 2D). The algorithm generates a new simplex by some elementary operations like reflections, expansions, and contractions. In the end, the minimum is concentrated in a very small simplex.

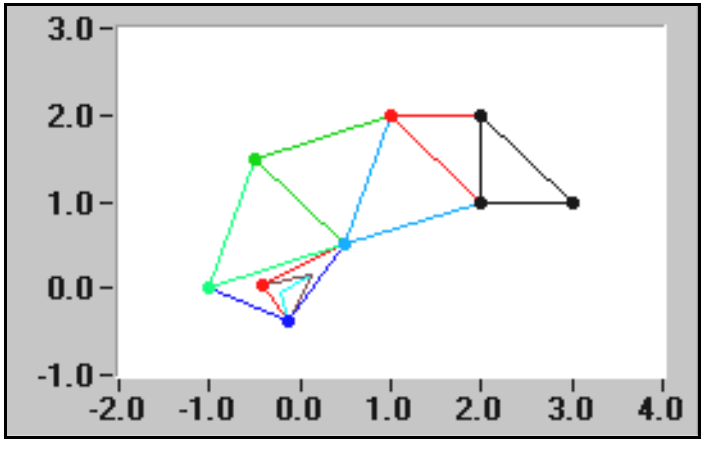
Example

The simplex sequence tending to the minimum (0,0) of the function $f(x, y) = x*x + y*y$ is shown in the following diagram. The function is entered on the front panel as:

Start: [3.2, 1]

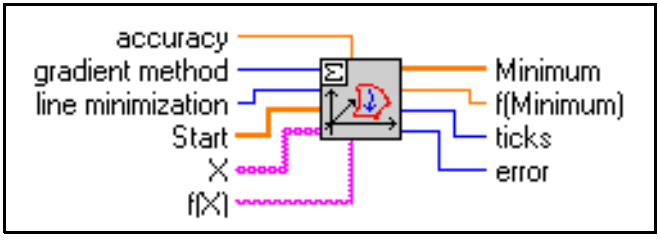
X: [x, y]

f(X): [x*x + y*y]



Conjugate Gradient nD

Determines a local minimum of a function of *n* independent variables with the Conjugate Gradient method.



DBL

accuracy controls the accuracy of the minimum. The method stops if two consecutive approximations differ no more than the value of **accuracy**. The default value is 1E-8.

U16

gradient method. A value of 0 represents the Fletcher Reeves method, a value of 1 represents the Polak Ribiere method. The default value is 0.

U16 **line minimization**. A value of 0 represents an algorithm without usage of the derivatives, a value of 1 represents an algorithm with usage of the derivatives. The default value is 0.

[DBL] **Start** is a point in *n*-dimension at which the optimization process starts.

[abc] **X** is an array of strings representing the **X** variables.

abc **f(X)** is the string representing the function of the **X** variables.

[DBL] **Minimum** is the determined local minimum in *n*-dimension.

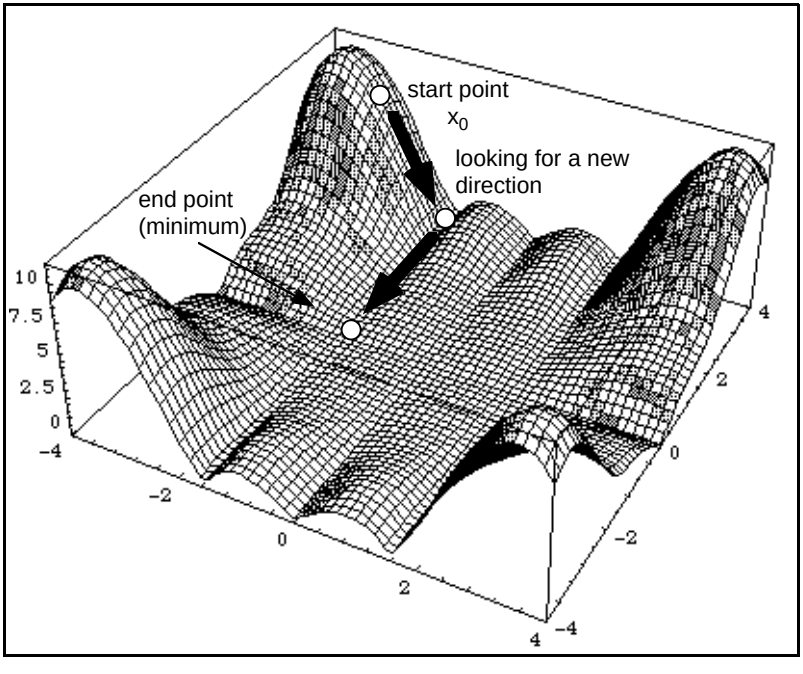
DBL **f(Minimum)** is the function value of **f(X)** at the determined minimum.

U32 **ticks** is the time in milliseconds for the whole calculation.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes. The Parser VIs check the syntax of the inputs.

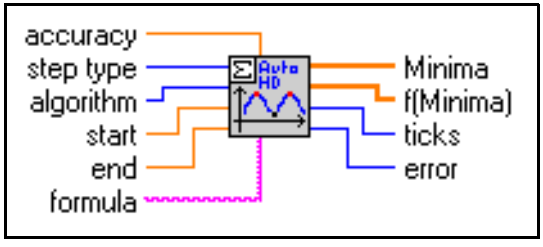
The Fletcher Reeves and the Polak Ribiere algorithm are based on the determination of best-suited directions and 1D subminimizations.

The following diagram shows a start point and a start direction. New points and new directions are calculated by the Conjugate Gradient nD VI.



Find All Minima 1D

Determines all local minima of a given function in a given interval.



DBL — **accuracy** controls the accuracy of the values of **Minima**. The method stops if two consecutive approximations differ no more than the value of **accuracy**. The default value is 1E-8.

U16 — **step type**. A value of 0 represents a fixed number of function values, a value of 1 represents an optimal step rate. In general, the second value leads to more accurate **Minima**. The default value is 0.

U16 — **algorithm**. A value of 0 represents the Golden Section method, a value of 1 represents the Brent method. The default value is 0.0.

DBL — **start** is the start point of the interval. The default value is 0.0.

DBL — **end** is the end point of the interval. The default value is 1.0.

abc — **formula** is a string representing the function under investigation.

[DBL] — **Minima** is an array of all found minima of **formula** in the interval (**start**, **end**).

[DBL] — **f(Mimima)** is the function values at the points **Minima**.

U32 — **ticks** is the time in milliseconds for the whole calculation.

I32 — **error**. See Appendix A, *Error Codes*, for a list of error codes.

☞ **Note:** ***If you want to find out the maxima of a function, you must take –function as the inputs. The –f(Minima) are the correct maximal function values.***

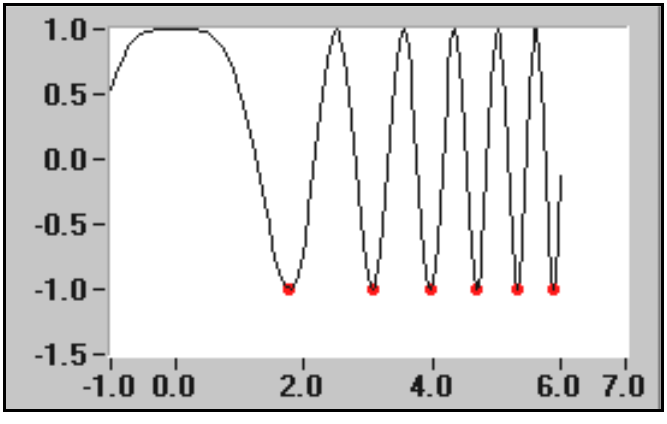
Example

Find all Minima 1D of $f(x) = \cos(x\text{^}2)$. All minima are determined. The following diagram shows the plot of $f(x)$. The boxes on the plot are the locations of the minima. The inputs are entered on the front panel as:

start: –1.0

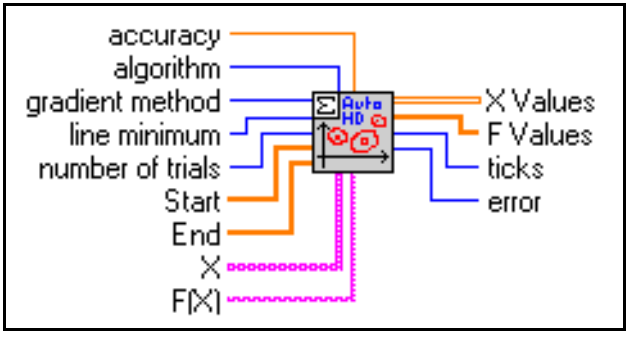
end: 6.0

formula: cos(x^2)



Find All Minima nD

Determines the minima of an *n*-dimension function in a given *n*-dimension interval.



DBL

accuracy controls the accuracy of the minima. The default value is 1E-8.

U16 **algorithm**. A value of 0 represents the Conjugate Gradient method, a value of 1 represents the Downhill Simplex method. The default value is 0.

U16 **gradient method**. A value of 0 represents the Fletcher Reeves method, a value of 1 represents the Polak Ribiere method. The default value is 0.

U16 **line minimum**. A value of 0 represents the line optimization without derivatives, a value of 1 represents the line optimization with derivatives. The default value is 0.

I32 **number of trials** is the number of the randomly chosen start points of the optimization process. These points belong to the interval (**start**, **end**). The default value is 5.

[DBL] **Start** is the start point in *n*-dimension.

[DBL] **End** is the end point in *n*-dimension.

[abc] **X** is an array of strings describing the *n* variables.

abc **F(X)** is a string describing the *n*-dimension function of **X**.

[DBL] **X Values** is a matrix describing all determined local minima.

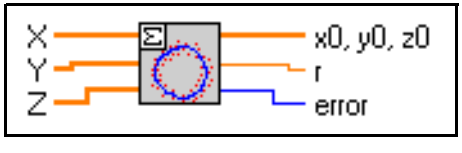
[DBL] **F Values** is the function values at the points **X Values**.

U32 **ticks** is the time in milliseconds for the whole calculation.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes.

Fitting on a Sphere

The algorithm determines the best spherical fit on a cloud of points in 3D.



[DBL] **X** is the x coordinates of the points of the cloud.

[DBL] **Y** is the y coordinates of the points of the cloud.

[DBL] **Z** is the z coordinates of the points of the cloud.

[DBL] **x0**, **y0**, **z0** are the calculated midpoints of the given cloud.

r is the calculated radius of the given cloud.

error. See Appendix A, *Error Codes*, for a list of error codes. Most error situations result from discrepancies between the sizes of **X**, **Y**, and **Z**.

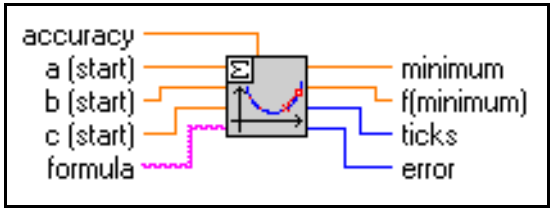
The min-functional is based on an uncommon idea:
find x_0, y_0, z_0 and r with

$$\sum_{i=1}^{n}((x_i - x_0)^2 + (y_i - y_0)^2 + (z_i - z_0)^2 - r^2)^2 = min!$$

This leads to a simple linear equation in x_0, y_0, and z_0.

Golden Section 1D

Determines a local minimum of a given 1D function with the help of a bracketing of the minimum. The Golden Section Search method is used.



accuracy controls the accuracy of the determined minimum of **formula**. The method stops, if two consecutive approximations differ not more than the value of **accuracy**. The default value is 1E-8.

a (start) is the left point of the bracketing interval. The default value is 0.0.

b (start) is the middle point of the bracketing interval. The default value is 0.0.

c (start) is the right point of the bracketing interval. The default value is 0.0.

formula is a string describing the function under investigation. The Parser VIs check the syntax of this string.

minimum is the determined local minimum of **formula**.

f(minimum) is the function value at the determined local minimum.

ticks is the time in milliseconds for the whole calculation.

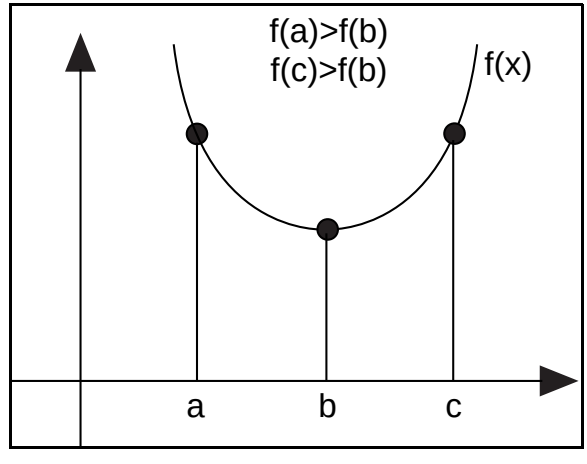
error. See Appendix A, *Error Codes*, for a list of error codes.

A bracketing triplet (a, b, c) of a 1D continuous function *f* is a combination of three points with $f(a) > f(b)$ and $f(c) > f(b)$. This guarantees the existence of a local minimum of *f* in the interval (a, c).

The Golden Section Search method determines beginning with a bracketing triplet (a, b, c) a new one with a considerably smaller expansion. Repeating this scheme often yields a good approximation of the local minimum. The new bracketing point is essentially calculated by the following equation.

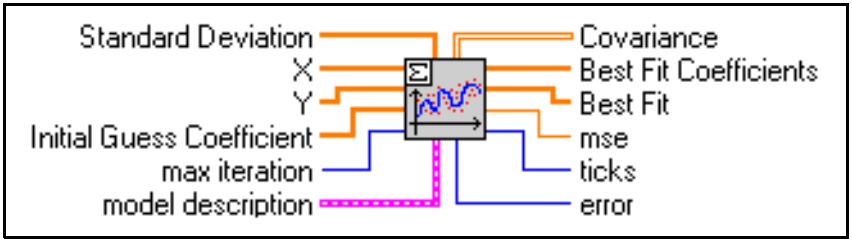
$$\left|\frac{x-b}{c-a}\right| = (\sqrt{5}-2) \quad \text{(Golden Section Search Method)}$$

The following diagram shows the relationship between *a*, *b*, *c* and *f*(*a*), *f*(*b*), *f*(*c*).



Levenberg Marquardt

Uses the Levenberg Marquardt method to determine a nonlinear set of coefficients that minimize a chi2 quantity.



[DBL] **Standard Deviation** is the standard deviation for data points. If they are equal or you do not know, leave this array empty. Internally, LabVIEW sets all data points to 1.0.

[DBL] **X** is the input array. The number of valid input points must be greater than zero and greater than the number of specified coefficients. The number of elements in **X** must be equal to the number of elements in **Y**.

[DBL] **Y** is the input array. The number of valid input points must be greater than zero and greater than the number of specified coefficients. The number of elements in **Y** must be equal to the number of elements in **X**.

[DBL] **Initial Guess Coefficient** denotes your initial-guessed solution.

[I32] **max iteration** is the maximum executing iteration. If the VI reaches maximum iteration without finding a solution, the function returns an error. You have to increase the **max iteration** or adjust the **Initial Guess Coefficient** to get a solution. The default value is 200.

model description is a cluster containing the fitting equation.

- [abc] **model** is a string describing the model equation.
- [abc] **Parameters** is an array of strings of the unknown parameters.
- [abc] **x** is a string describing the independent variable.

[DBL] **Covariance** is the matrix of covariances.

[DBL] **Best Fit Coefficients** is the set of coefficients that minimize the penalty (chi-squared) function. The chi-squared function is given by

$$\chi^2 = \sum_{i=0}^{N-1}\left(\frac{y_i - f(x;\, a_1,\ldots,\, a_M)}{\sigma_i}\right)$$

In this equation, (x_i, y_i) are the input data points, and $f(x_i;\, a_1,\ldots,\, a_M) = f(X, A)$ is the nonlinear function where $a_1,\ldots,a_M$ are coefficients. If the measurement errors are independent and normally distributed with constant, standard deviation $\sigma_i = \sigma$, this is also the least-square estimation.

The input arrays **X** and **Y** define the set of input data points. The VI assumes that you have prior knowledge of the nonlinear relationship between the *x* and *y* coordinates. That is, $f = f(X, A)$ where the set of coefficients, *A*, is determined by the Levenberg Marquardt algorithm.

Using this function successfully depends on how close your initial guess coefficients are to the solution. Therefore, it is always worth taking the time and effort to obtain good initial guess coefficients to the solution from any available resources before using the function.

[DBL] **Best Fit** is the fitted data.

DBL **mse** is the chi2 value.

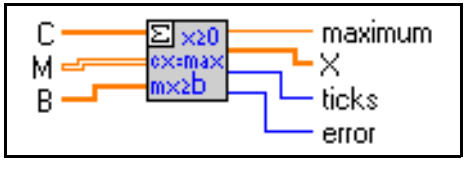
U32 **ticks** is the time in milliseconds for the whole calculation.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes.

More information on the Levenberg Marquardt method can be found in the Non-Linear Lev-Mar Fit VI description in the *LabVIEW Analysis VI Reference Manual*.

Linear Programming Simplex Method

Determines the solution of a linear programming problem.



[DBL] **C** is a vector describing the linear functional to maximize.

[DBL] **M** is a matrix describing the different constraints.

[DBL] **B** is a vector describing the right sides of the constraints inequalities.

DBL **maximum** is the maximal value of a **x** under the constraints.

[DBL] **X** is the solution vector.

U32 **ticks** is the time in milliseconds for the whole calculation.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes. The nonexistence of a solution *x* leads to an error.

The optimization problem

$$cx = max!$$

with the constraints

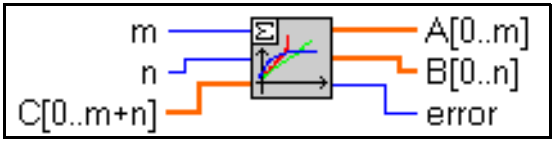
$$x \geq 0 \text{ and } mx \geq b.$$

Here $\mathbf{X} = (x_1,\ldots, x_n)$, $\mathbf{C} = (c_1,\ldots, c_n)$, $\mathbf{B} = (b_1,\ldots, b_k)$ and **M** a *k* by *n* matrix. Now you must decide whether or not an optimal vector *x* does exist, and if so, determine this vector *x*. The solution of a linear programming problem is a two-step process. The first step transforms the original problem into a problem in restricted normal form (essentially without inequalities in the formulation). The second step consists of the solution of this restricted normal form problem.

Note: ***The previous formulation seems to be special. But there are many ways to reformulate terms. For instance,*** $dx \leq e$ ***is equivalent to*** $|-dx \geq -e|$***, and,*** $dx = e$ ***is equivalent to the combination*** $dx \geq e$ ***and*** $-dx \geq -e$***.***

Pade Approximation

Determines the coefficients of a rational polynomial to best suit a given set of first derivatives.



I32 **m** is the degree of the polynomial of the numerator.

I32 **n** is the degree of the polynomial of the denominator.

[DBL] **C[0..m+n]** is the array describing the first derivatives of the given function.

[DBL] **A[0..m]** is the polynomial of the numerator.

[DBL] **B[0..n]** is the polynomial of the denominator.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes. Especially, $m \geq n$ is necessary.

Let f be a given function with known values $f(0), f'(0), \ldots, f^{(n+m)}(0)$. There is a unique rational polynomial $(m \geq n)$

$$R(x) = \frac{a_0 + a_1 x + \ldots + a_m x^m}{1 + b_1 x + \ldots + b_n x^n}$$

with $R(0) = f(0), R'(0) = f'(0), \ldots, R^{(m+n)}(0) = f^{(m+n)}(0)$.

The rational polynomial can be determined by solving a special linear equation.

1D Explorer VIs

Chapter 7

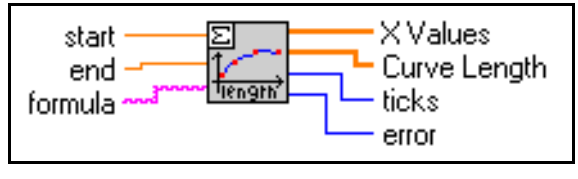
You can use this collection of VIs to study real-valued 1D functions given in symbolic form. You can study different qualities of function graphs with and without additional parameters. You can calculate and derive characteristic properties of these curves, such as length, integration, differentiation, extrema and zeroes. You can connect the **X Values** and **Y Values** outputs of the VIs to a suitable graph indicator for a visual representation of the waveform. These VIs are available in the `1DEXPLO.LLB` library.

Notice that some of the VIs are named with the words *Optimal Step*. For a function that has many maxima, minima, or singularities, the standard method (using equidistant points) may yield incorrect results. In such cases the Optimal Step method is better. This method starts off by taking equidistant points, but also checks the steepness of the curve between these points. In very steep portions of the graph (determined by the value of epsilon on the front panel), a new point is generated in between. As a rule, the smaller the value of epsilon, the more the points that are generated, and the better the graph.

1D Explorer VI Descriptions

Curve Length

Calculates the curve length of a 1D function between **start** and **end**.



DBL **start** is the start point of the interval under investigation. The default value is 0.0.

DBL **end** is the end point of the interval. The default value is 1.0.

formula is a string describing the function under investigation. You can use any valid symbol as the variable name. See Chapter 1, *Parser VIs*, in this manual.

X Values is the array of all regarded points in the interval (**start**, **end**).

Curve Length is an array of the values of the curve length of **formula** between **start** and **end** at all **X Values**.

ticks is the time in milliseconds to analyze the formula and to produce the **X Values** array and the **Curve Length** array.

error. See Appendix A, *Error Codes*, for a list of error codes. The accuracy of formula is verified by the Parser VIs.

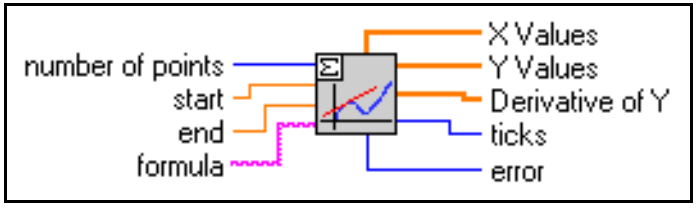
The curve length of a given function $f(t)$ between **start** and **end** can be calculated by the following equation.

$$L = \int_{start}^{end} \sqrt{1 + \left(\frac{df(t)}{dt}\right)^2}\, dt$$

This calculation is realized using the Integration VI, found in the `1DEXPLO.LLB` library, and that is why the calculations are based on the Runge Kutta method.

Differentiation

Calculates both function values and the values of the derivative of a given 1D function defined by a formula at equidistant points in an interval.



number of points is the number of all calculated points. The default value is 10.

start is the start point of the interval under investigation. The default value is 0.0.

end is the end point of the interval. The default value is 1.0.

formula is a string describing the function under investigation. You can use any valid symbol as the variable name. See Chapter 1, *Parser VIs*, in this manual.

X Values is the array of all equidistantly positioned points in the interval (**start**, **end**).

Y Values are the values of the function.

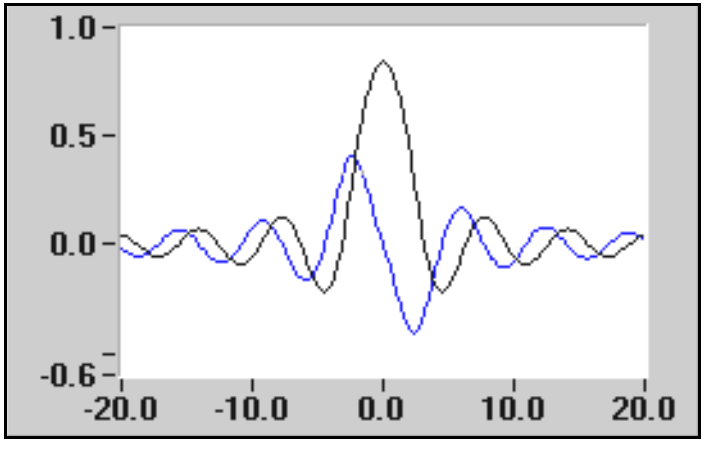
Derivative of Y are the values of the derivative of the function at the points **X Values**.

ticks is the time in milliseconds to analyze the formula and to produce the **X Values** array, **Y Values** array and **Derivative of Y** array.

error. See Appendix A, *Error Codes*, for a list of error codes. The accuracy of formula is verified by the Parser VIs.

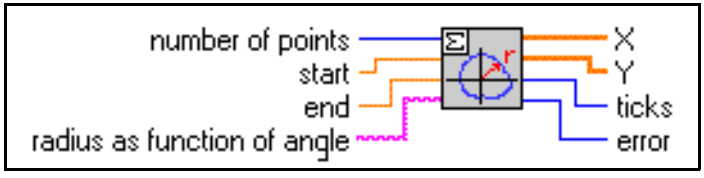
Note: ***Even though there is a modified method (Optimal Step) for functions, the G Math Toolkit does not have one for the Differentiation VI. If you are interested in highly accurate values of Differentiation, start with the symbolic differentiation (by hand). Then use the Eval y = f(x) Optimal Step VI, where f is the derivative of the function.***

The function and the derivative of $f(x) = \sin(\mathrm{sinc}(x))$ are investigated in the interval (–20, 20). The following diagram shows both $f(x)$ and $f'(x)$.



Eval Polar to Rect

Calculates the values of a polar parametric curve in 2D.



U32 — **number of points** is the number of all calculated points. The angle variable is splitted into equidistant subpoints. The default value is 10.

DBL — **start** is the start point of the interval under investigation. The default value is 0.0.

DBL — **end** is the end point of the interval. The default value is 1.0.

abc — **radius as function of angle** is a string representing the formula, $r = r(\varphi)$.

[DBL] — **X** is the array of the values of the first component, $r(\varphi) * \cos(r\varphi)$.

[DBL] — **Y** is the array of the values of the second component, $r(\varphi) * \sin(r\varphi)$.

U32 — **ticks** is the time in milliseconds to analyze the formula and to produce the **X** and the **Y** array.

I32 — **error**. See Appendix A, *Error Codes*, for a list of error codes. The accuracy of **radius as function of angle** is checked with the help of some of the Parser VIs.

Let φ be the angle and $r(\varphi)$ the radius in polar coordinate notation. Then it is

$$x = r(\varphi)\cos\varphi$$

$$y = r(\varphi)\sin\varphi$$

where φ runs over the specified interval.

Example

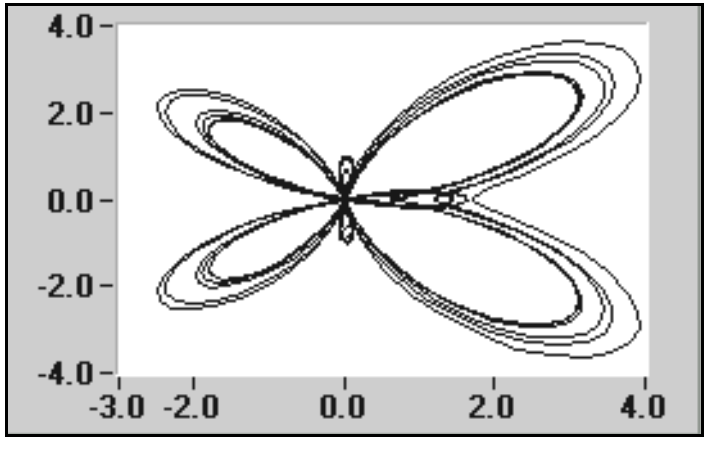
The function $r(t) = \exp(\cos(t)) - (2)*\cos(4*t) + \sin(t/12)\text{^}5$ describes a butterfly curve in the plane in polar coordinates, as shown in the following diagram. The diagram can be generated by entering the following values on the front panel:

number of points: 1000

start: 0

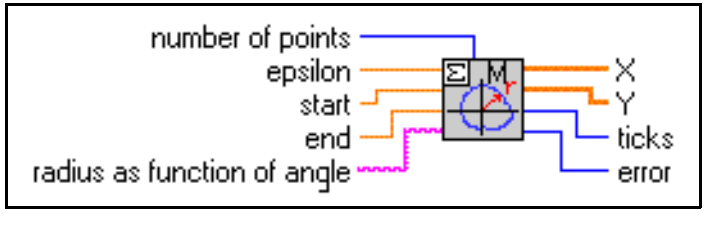
end: 30

radius as function of angle: exp(cos(t)) – 2*cos(4*t) + sin(t/12)^5



Eval Polar to Rect Optimal Step

Operates like the Eval Polar to Rect VI, but with a significantly higher degree of accuracy.



number of points is the number of calculated points at the beginning of the execution. The default value is 10. Usually, many more points will be added.

epsilon controls the construction of points inside the construction area. The default value is 0.05.

start is the start point of the interval under investigation. The default value is 0.0.

end is the end point of the interval. The default value is 1.0.

radius as function of angle is a string representing the formula $r = r(\varphi)$.

X is the array of the values of the first component, $r(\varphi)\cos(\varphi)$.

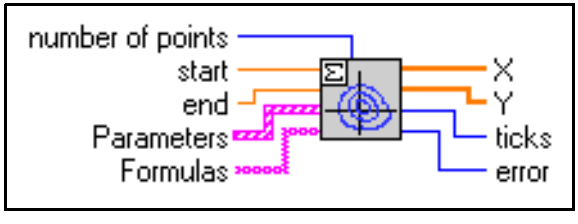
Y is the array of the values of the second component, $r(\varphi)\sin(\varphi)$.

ticks is the time in milliseconds to analyze the formula and to produce the **X** and the **Y** array.

error. See Appendix A, *Error Codes*, for a list of error codes. The accuracy of **radius as function of angle** is checked with the help of some of the Parser VIs.

Eval X-Y(a,t)

A generalized version of the Eval X-Y(t) VI with the possibility of adding some parameters into the formula.



number of points is the number of all calculated points. The independent variable is split into equidistant subpoints. The default value is 10.

start is the start point of the interval under investigation. The default value is 0.0.

end is the end point of the interval. The default value is 1.0.

Parameters is an array of clusters describing the parameters.

name of the parameter which uses the conventions of the Parser VIs.

value of the parameter.

Formulas is an array of two strings describing the two function components. You can use any valid symbol as the variable name. See Chapter 1, *Parser VIs*, in this manual.

X is the array of the values of the first component.

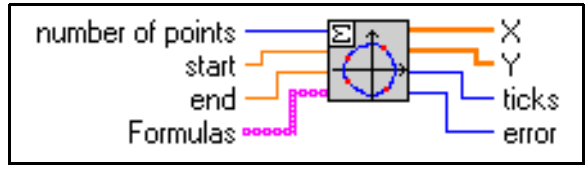
Y is the array of the values of the second component.

ticks is the time in milliseconds to analyze the **Formulas** and to produce the **X** and the **Y** array.

error. See Appendix A, *Error Codes*, for a list of error codes. The accuracy of **Formulas** and **Parameters** is checked with the help of some of the Parser VIs.

Eval X-Y(t)

Calculates the values of a function $(f(t), g(t))$, where t runs over an interval. Both components are given by Formulas. The t-values are chosen equidistantly in the interval.



number of points is the number of all calculated points. The independent variable is split into equidistant subpoints. The default value is 10.

start is the start point of the interval under investigation. The default value is 0.0.

end is the end point of the interval. The default value is 1.0.

Formulas is an array of two strings describing the two formula components. You can use any valid symbol as the variable name. See Chapter 1, *Parser VIs*, in this manual.

X is the array of the values of the first component.

Y is the array of the values of the second component.

ticks is the time in milliseconds to analyze the formula and to produce the **X** and the **Y** array.

error. See Appendix A, *Error Codes*, for a list of error codes.
The accuracy of **Formulas** is checked with the help of some of the Parser VIs.

Example

The functions $(t*\sin(t), \text{sqrt}(t)*\cos(t))$ describe a curve in the plane, with t ranging between (0, 12). The following diagram shows this curve.

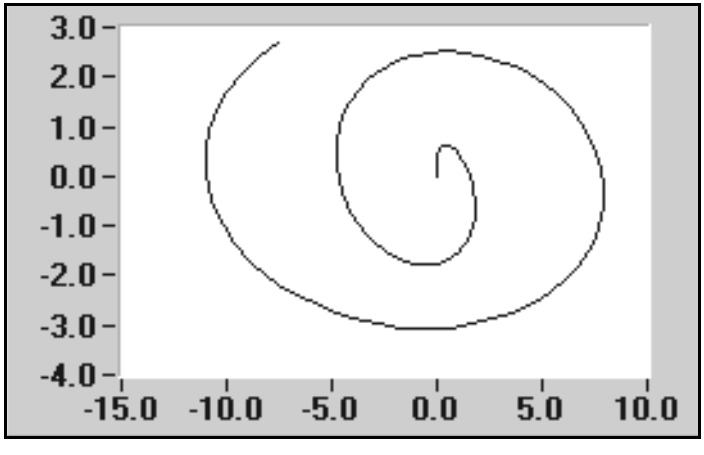
Enter the data on the front panel as shown:

number of points: 100

start: 0

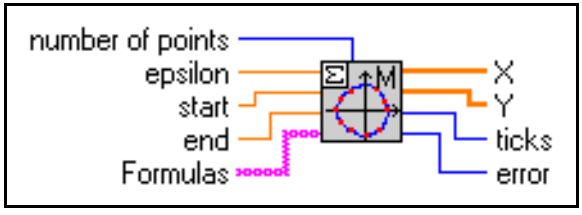
end: 12

Formulas: (t*sin(t), sqrt(t)*cos(t))



Eval X-Y(t) Optimal Step

Calculates the values of a more complex function $(f(t), g(t))$, where t runs over an interval. Both components are given by **Formulas**.



number of points is the number of calculated points at the beginning of the execution. The default value is 10. Usually, many more points will be added.

epsilon controls the construction of points in between. The default value is 0.05.

start is the start point of the interval under investigation. The default value is 0.0.

end is the end point of the interval. The default value is 1.0.

Formulas is an array of two strings describing the two function components. You can use any valid symbol as the variable name. See Chapter 1, *Parser VIs*, in this manual.

X is the array of the values of the first component.

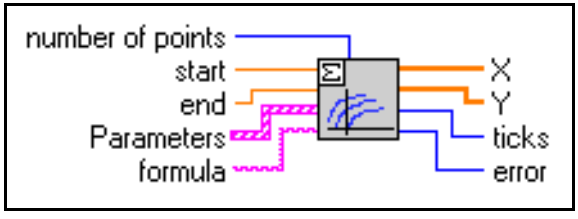
Y is the array of the values of the second component.

ticks is the time in milliseconds to analyze the formula and to produce the **X** and the **Y** array.

error. See Appendix A, *Error Codes*, for a list of error codes. The accuracy of **Formulas** is checked with the help of some of the Parser VIs.

Eval y=f(a,x)

A generalized version of the Eval y=f(x) VI, with the possibility of adding some parameters into the formula.



U32 **number of points** is the number of all calculated points. The independent variable is split into equidistant subpoints. The default value is 10.

DBL **start** is the start point of the interval under investigation. The default value is 0.0.

DBL **end** is the end point of the interval. The default value is 1.0.

[ ] **Parameters** is an array of clusters describing the parameters.

- abc **name** of the parameter which uses the conventions of the Parser VIs.
- DBL **value** of the parameter.

abc **formula** is a string describing the function under investigation. You can use any valid symbol as the variable name. See Chapter 1, *Parser VIs*, in this manual.

[DBL] **X** is the array of equidistant points between **start** and **end**.

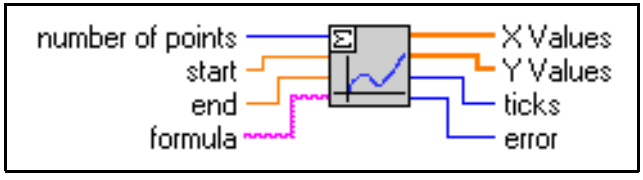
[DBL] **Y** is the function values at the points **X**.

U32 **ticks** is the time in milliseconds to analyze the formula and to produce the **X** and the **Y** array.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes. The accuracy of **formula** and **Parameters** is checked with the help of some of the Parser VIs.

Eval y=f(x)

Calculates the values of a 1D function given by a formula at equidistant points in an interval.



U32 — **number of points** is the number of all calculated points. The independent variable is split into equidistant subpoints. The default value is 10.

DBL — **start** is the start point of the interval under investigation. The default value is 0.0.

DBL — **end** is the end point of the interval. The default value is 1.0.

abc — **formula** is a string describing the function under investigation. You can use any valid symbol as the variable name. See Chapter 1, *Parser VIs*, in this manual.

[DBL] — **X Values** is the array of equidistant points between **start** and **end**.

[DBL] — **Y Values** is the array of function values from the corresponding points of **X Values**.

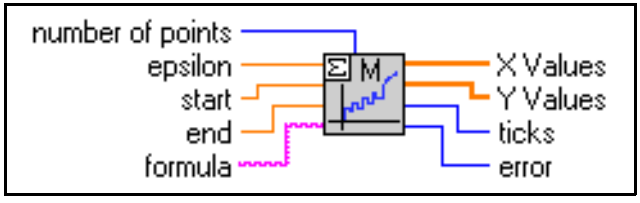
U32 — **ticks** is the time in milliseconds to analyze the formula and to produce the **X Values** array and the **Y Values** array.

I32 — **error**. See Appendix A, *Error Codes*, for a list of error codes. The accuracy of **formula** is verified by the Parser VIs.

☞ **Note:** ***You can directly connect the (X Values, Y Values) array to a graph indicator to see the result of Eval y=f(x) VI.***

Eval y=f(x) Optimal Step

Calculates the values of a more complex 1D function given by a formula.



U32 **number of points** is the number of calculated points at the beginning of the execution. The default value is 10. Usually, many more points will be added.

DBL **epsilon** controls the construction of points in the sense of optimal steps. The default value is 0.05. The smaller **epsilon** the more values will be produced.

DBL **start** is the start point of the interval under investigation. The default value is 0.0.

DBL **end** is the end point of the interval. The default value is 1.0.

abc **formula** is a string describing the function under investigation. You can use any valid symbol as the variable name. See Chapter 1, *Parser VIs*, in this manual.

[DBL] **X Values** is the array of all points in the interval (**start**, **end**). In particular, all start points produced by **number of points**, **start** and **end** belong to the array, but many further points belong too.

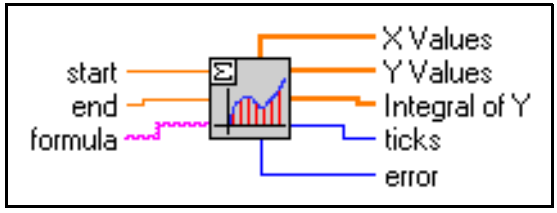
[DBL] **Y Values** are the function values at the points **X Values**.

U32 **ticks** is the time in milliseconds to analyze the formula and to produce the **X Values** and the **Y Values** array.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes. The accuracy of **formula** is verified by the Parser VIs.

Integration

Calculates both the function values and the integral of a 1D function between **start** and **end**.



DBL — **start** is the start point of the interval under investigation. The default value is 0.0.

DBL — **end** is the end point of the interval. The default value is 1.0.

abc — **formula** is a string describing the function under investigation. You can use any valid symbol as the variable name. See Chapter 1, *Parser VIs*, in this manual.

[DBL] — **X Values** is the array of all regarded points in the interval (**start**, **end**).

[DBL] — **Y Values** are the values of the function.

[DBL] — **Integral of Y** are the values of the integral of **formula** between **start** and **end** at all **X Values** values.

U32 — **ticks** is the time in milliseconds to analyze the formula and to produce the **X Values** array and the **Integral of Y** array.

I32 — **error**. See Appendix A, *Error Codes*, for a list of error codes. The accuracy of **formula** is verified by the Parser VIs.

Let *f(t)* be the given function. The calculation of

$$I = \int_{start}^{end} f(t)dt$$

can be reformulated as an ordinary differential equation

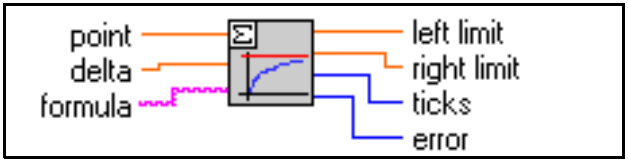
$$\frac{dI(s)}{ds} = f(s)$$

$$I(start) = 0$$

The algorithm is based on this reformulation. The Runge Kutta method is used for accuracy. See the ODE Runge Kutta 4th Order VI description in Chapter 4, *Ordinary Differential Equation VIs*, for more information.

Limit

Determines the left and right limits of a 1D function at a given point.



DBL **point** is the point at which the limits have to be calculated. The default value is 0.0.

DBL **delta** is the distance to the far left and right of **point**. The default value is 1E-10.

abc **formula** is a string describing the function under investigation. You can use any valid symbol as the variable name. See Chapter 1, *Parser VIs*, in this manual for more information.

DBL **left limit**. The accuracy is up to 8 decimal digits.

DBL **right limit**. The accuracy is up to 8 decimal digits.

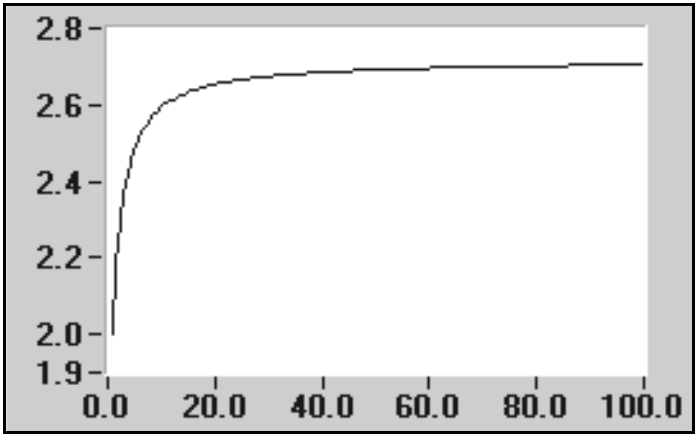
U32 **ticks** is the time in milliseconds to analyze the formula and to produce limits. Usually, the time is negligible for the limit operations.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes. The accuracy of **formula** is verified by the Parser VIs.

The algorithm calculates only the two values *f*(**point** – **delta**) and *f*(**point** + **delta**). Furthermore, delta is internally rounded to a power of 2.

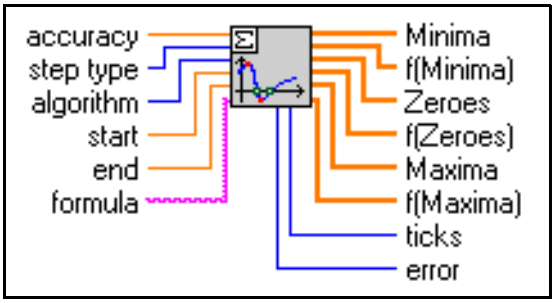
Note: ***A very small delta value can result in numerical inaccuracies. It is highly recommended that you take a value of delta = 1E-10 in all cases.***

The function $f(x) = (1 + 1/x)\hat{}x$ has the famous limit *e* (Euler) if *x* tends to infinity. You can use the Limit VI to determine the Euler number if you define $g(x) = (1 + x)\hat{}(1/x)$ for small positive *x*. By entering (1+*x*)^(1/*x*) in the formula control on the front panel, you can find the limit. The following diagram shows the convergence of *f*(*x*) to *e*.



Zeroes and Extrema of f(x)

Determines all zeroes and extrema of a 1D function in a given interval.



DBL **accuracy** controls the accuracy of the zeroes and the extrema. The default value is 1E-8.

U16 **step type**. A value of 0 (fixed function) represents uniformly spaced function values, a value of 1 (modified function) represents the optimal step size. In general, the second value leads to more accurate zeroes and extrema. The default value is 0.

U16 **algorithm**. A value of 0 selects the Ridders method; a value of 1 selects the Newton Raphson method. The default value is 0.

DBL **start** is the start point of the interval under investigation. The default value is 0.0.

DBL **end** is the end point of the interval. The default value is 1.0.

formula is a string describing the function.

Minima are the determined minimal values of **formula**.

f(Minima) are the function values at **Minima**.

Zeroes are the determined zeroes of **formula**.

f(Zeroes) are the function values of **Zeroes**. Usually, these values are very close to 0.

Maxima are the determined maximal values of **formula**.

f(Maxima) are the function values at **Maxima**.

ticks is the time in milliseconds to analyze the formula and to produce the **Minima**, **Zeroes**, and **Maxima**.

error. See Appendix A, *Error Codes*, for a list of error codes.

2D Explorer VIs

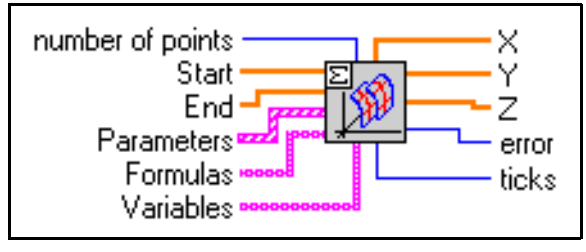
Chapter 8

You can use the 2D Explorer VIs to examine 2D functions given in symbolic form, where parameterization is allowed. You can numerically calculate extrema and partial derivatives. These VIs are available in the `2DEXPLO.LLB` library.

2D Explorer VI Descriptions

Eval X-Y-Z(a,t1,t2)

Describes a surface in 3D with two variables running over two different intervals. Additionally, you can integrate an arbitrary set of parameters, where any parameter gets its value with the help of an element of an array of parameter clusters on the front panel.



U32 — **number of points** describes the number of grid points for both variables. The default value is 25.

[DBL] — **Start** are the start points of both variables (that is, an array of length 2). The default values are (0,0).

[DBL] — **End** are the end points of both variables (that is, an array of length 2). The default values are (1,1).

[...] — **Parameters** is an array of clusters describing the parameters.

abc — **name** of the parameter which uses the same conventions as the Parser VIs.

DBL — **value** of the parameter.

Formulas is an array of three strings describing the three functions.

Variables is an array of two strings representing the two variables with respect to the naming conventions of the Parser VIs. The default variables are (t_1,t_2).

X is a 1D array of the x values at the grid points. These values are calculated on a 2D grid defined by the discrete values of the **Variables** (usually t_1 and t_2). The size of X is given by number of points.

Y is a 1D array of the y values at the grid points. These values are calculated on a 2D grid defined by the discrete values of the **Variables** (usually t_1 and t_2). The size of Y is given by number of points.

Z is a 1D array of the z values at the grid points. These values are calculated on a 2D grid defined by the discrete values of the **Variables** (usually t_1 and t_2). The size of Z is given by number of points.

ticks is the time effort for the whole calculation of the function values in milliseconds.

error. See Appendix A, *Error Codes*, for a list of error codes. Errors can result from incorrect function definitions and from discrepancies between the function definition and the array of variables.

Example

As an example, consider the ellipsoid given by $\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$.

Choosing the parameter values (in the **Parameters** cluster) as $a = 1$, $b = 2$, and $c = 3$, you can input the following three equations in the **Formulas** control on the front panel to get the **X**, **Y** and **Z** values of the ellipsoid.

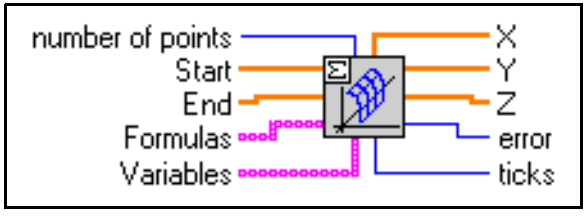
a*cos(t1)

b*sin(t1)*sin(t2)

c*sin(t1)*cos(t2)

Eval X-Y-Z(t1,t2)

Describes a surface in 3D with two variables running over two different intervals.



number of points describes the number of grid points for both variables. The default value is 25.

Start are the start points of variables (that is an array of length 2). The default value is (0,0).

End are the end points of both variables (that is, an array of length 2). The default value is (1,1).

Formulas is a 3-element array of formula strings representing a function definition of exactly two different variables. The naming conventions of the Parser VIs are valid.

Variables is an array of two strings representing the two variables with respect to the naming conventions of the Parser VIs. The default variables are (t_1,t_2).

X is the array of the *x* values at the grid points defined by the parameters.

Y is the array of the *y* values at the grid points defined by the parameters.

Z is the array of the *z* values at the grid points defined by the parameters.

ticks is the time effort for the whole calculation of all function values in milliseconds.

error. See Appendix A, *Error Codes*, for a list of error codes. Errors can result from incorrect function definitions and by discrepancies between the function definition and the array of variables.

Example

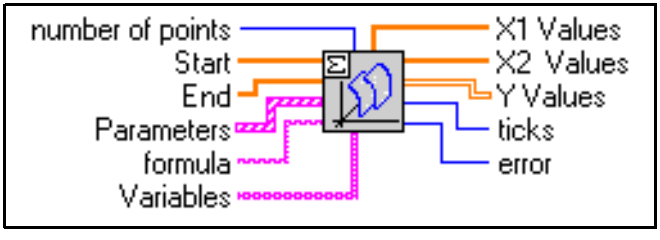
The three formulas

$$x(t1, t2) = \sin(t1)$$
$$y(t1, t2) = \cos(t1)\sin(t2)$$
$$z(t1, t2) = \cos(t1)\cos(t2)$$

describe a part of the unit sphere in 3D. In the **Formulas** control, only the right hand side of the above three equations needs to be entered.

Eval y=f(a,x1,x2)

Calculates a 2D array of function values defined on a grid. Additionally, you can integrate an arbitrary set of parameters, where any parameter gets its value with the help of an element of an array of parameter clusters on the front panel.



U32 **number of points** describes the number of grid points for both variables. The default value is 25.

[DBL] **Start** are the start points of both variables (that is, an array of length 2). The default value is (0,0).

[DBL] **End** are the end points of both variables (that is, an array of length 2). The default value is (1,1).

Parameters is an array of clusters describing the parameters.

abc **name** of the parameter which uses the same conventions as the Parser VIs.

DBL **value** of the parameter.

abc **formula** is a string describing a function, where both variables and parameter variables can be used.

[abc] **Variables** is an array of two strings representing the two variables with respect to the naming conventions of the Parser VIs. The default variables are ($x1$,$x2$).

[DBL] **X1 Values** is a 1D array of the used *x*1 arguments.

[DBL] **X2 Values** is a 1D array of the used *x*2 arguments.

[DBL] **Y Values** is the resulting 2D array of the function values.

U32 **ticks** is the time effort for the whole calculation of the function values in milliseconds.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes. Errors can result from incorrect function definitions and from discrepancies between the function definition and the array of variables. Additional errors may occur if the parameter variables violate the general parser rules or collide with the set of variables.

Example

The function $f(a, b, x_1, x_2) = a*\mathrm{sinc}(\mathrm{gamma}(x_1 + x_2)) - b*\sin(x_1)*\cos(x_2)$ with $a = 1$ and $b = 2$ is a common function in x_1 and x_2. The variables *a* and *b* stand for **Parameters**. Only the right hand side of this equation needs to be entered on the **Formula** control. Thus the values entered on the front panel are

Parameters: **parameters(0)**a

1.00

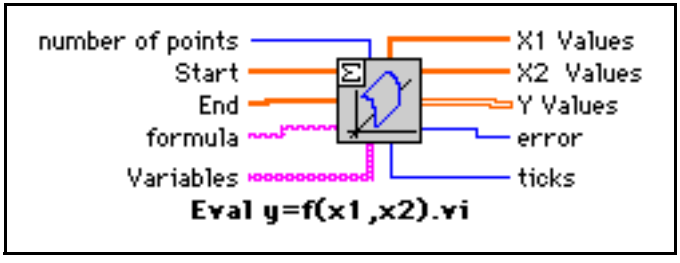
parameters(1)b

2.00

formula: a*sinc(gamma(x1 + x2)) – b*sin(x1)*cos(x2)

Eval y=f(x1,x2)

Calculates a 2D array of function values defined on a grid.



U32 **number of points** describes the number of grid points for both variables. The default value is 25.

[DBL] **Start** are the start points of both variables (that is, an array of length 2). The default value is (0,0).

[DBL] **End** are the end points of both variables (that is, an array of length 2). The default value is (1,1).

abc **formula** is a string representing a function definition of exactly two different variables. The naming conventions of the Parser VIs are valid.

[abc] **Variables** is an array of two strings representing the two variables with respect to the naming conventions of the Parser VIs. The default variables are (*x*1,*x*2).

[DBL] **X1 Values** is a 1D array of the used *x*1 arguments.

[DBL] **X2 Values** is a 1D array of the used *x*2 arguments.

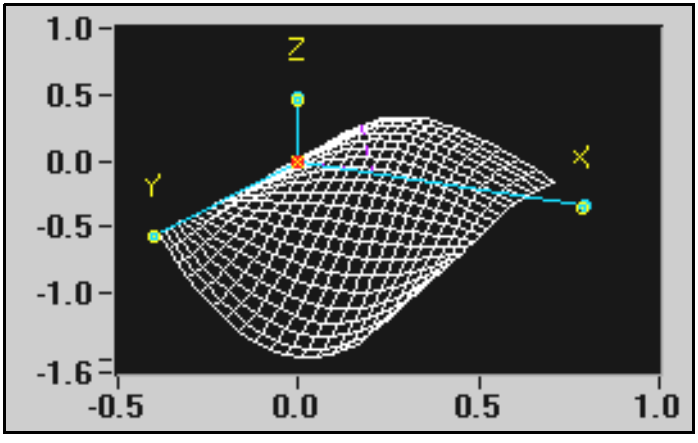
[DBL] **Y Values** is the resulting 2D array of the function values.

U32 **ticks** is the time effort for the whole calculation of all function values in milliseconds.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes. Errors can result from incorrect function definitions and from discrepancies between the function definition and the array of variables.

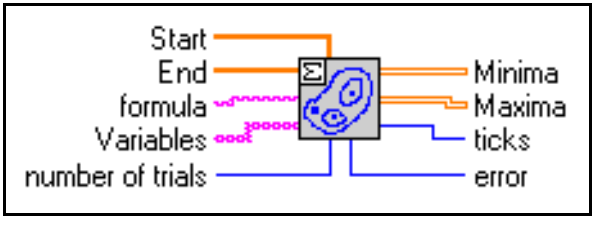
☞ **Note:** ***You can use the Data Visualization VI Library to visualize the results of the Eval y=f(x1,x2) VI. For more details, refer to the VI descriptions for the Contour Plot, Mesh 3D, and Common Intensity Maps VIs.***

The following diagram shows the visualization of the function $f(x1, x2) = \sin(3x1)*\cos(3*x2)$ in the interval (–2, 2) x (–2, 2).



Extrema of f(x1,x2)

The algorithm is looking for local extrema of a given function of two variables on a given rectangle.



[DBL] **Start** are the start points of both variables (that is, an array of length 2). The default values are (0,0).

[DBL] **End** are the end points of both variables (that is, an array of length 2). The default values are (1,1).

abc **formula** is a string describing a function.

[abc] **Variables** is an array of two strings representing the two variables with respect to the naming conventions of the Parser VIs. The default variables are (*x*1,*x*2).

U32 **number of trials** is the number of randomly spaced starting points of the algorithm.

Minima is a 2D array of all local minima of the given function. Any local minimum is presented by two coordinates.

Maxima is a 2D array of all local maxima of the given function. Any local maximum is presented by two coordinates.

ticks is the time effort for the whole calculation of the function values in milliseconds.

error. See Appendix A, *Error Codes*, for a list of error codes. Errors can result from incorrect function definitions and from discrepancies between the function definition and the array of variables.

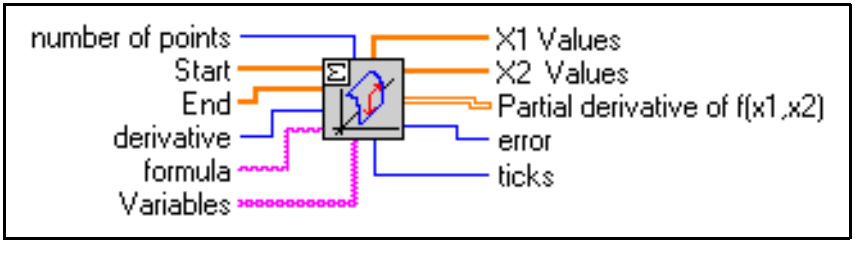
The absolute distance between two extrema must be equal or larger than 1E-6 for the two vectors to register as different from each other.

Note: ***While the number of trials can be very large, there is no guarantee you can find all or at least one zero, local minimum or local maximum of the given function. Moreover, such points do not exist in all cases.***

Note: ***Though the randomly chosen start points of the extrema algorithm belong to the initially given rectangle, sometimes the determined extrema can be found outside of the rectangle. In this case, these values will also be presented.***

Partial Derivatives of f(x1,x2)

Calculates a 2D array of the partial derivatives of a function of two independent variables.



number of points describes the number of grid points for both variables. The default value is 25.

Start are the start points of both variables (that is, an array of length 2). The default value is (0,0).

End are the end points of both variables (that is, an array of length 2). The default value is (1,1).

derivative is a value of 0 represents the partial derivative of the first variable. A value of 1 represents the partial derivative of the second variable.

formula is a string describing a function.

Variables is an array of two strings representing the two variables with respect to the naming conventions of the Parser VIs. The default value is ($x1$,$x2$).

Partial derivative of f(x1,x2) is the 2D array of the fixed partial derivative at the defined grid points. For a **derivative** of 0, the function $df(x1, x2)/dx1$ is calculated, for a **derivative** of 1 the function $df(x1, x2)/dx2$ is calculated.

X2 Values is the resulting 1D array.

Y Values is the resulting 2D array.

ticks is the time effort for the whole calculation of the function values in milliseconds.

error. See Appendix A, *Error Codes*, for a list of error codes. Errors can result from incorrect function definitions and discrepancies between the function definition and the array of variables. An additional error can occur, if the **derivative** is neither 0 nor 1.

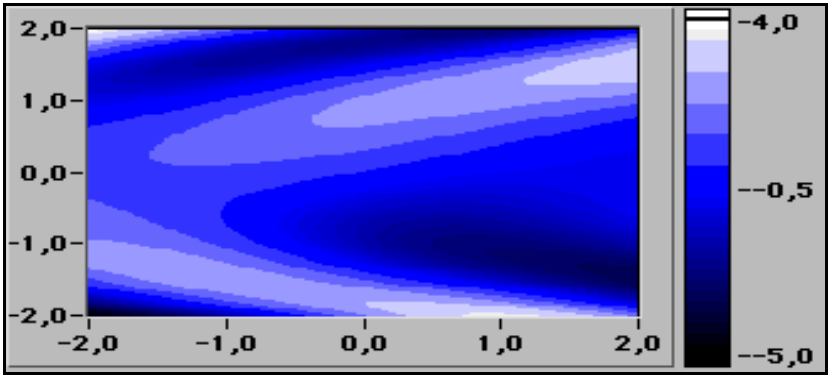
Example

The x_1-derivative of the function $f(x_1, x_2) = \sin(x_1 * x_1 - x_2) - \cos(\sin(x_2) - x_1)$ is investigated in the interval (–2, 2) by (–2, 2)! The values are obtained by entering the following on the front panel:

Start: [–2, –2]

End: [2, 2]

Formula: sin(x1*x1 – x2) – cos(sin(x2) – x1)



Function VIs

Chapter 9

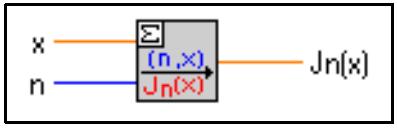
These are a group of VIs you can use to evaluate some common mathematical functions. They are available in the `FUNCTION.LLB` library.

Function VI Descriptions

Bessel Function Jn(x)

The Bessel function $J_n(x)$ is defined by

$$J_n(x) = \left(\frac{1}{2}x\right)^n \sum_{k=0}^{\infty} \frac{\left(\frac{-1}{4}x^2\right)^k}{k!\Gamma(n+k+1)} \text{ with } n = 0, 1, \ldots$$



DBL **x** is any real number.

U32 **n** is a non-negative integer.

DBL **Jn(x)** is the Bessel function of the first kind of order *n*.

Generally the power series for $J_n(x)$ is not computationally useful. More useful are recurrence relations and rational approximations of J_n and Y_n. The recurrences are

$$J_{n+1}(x) = \frac{2n}{x} J_n(x) - J_{n-1}(x)$$

$$Y_{n+1}(x) = \frac{2n}{x} Y_n(x) - Y_{n-1}(x)$$

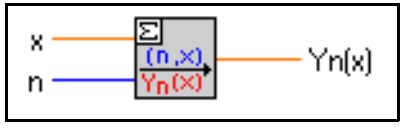
The treatment of the cases $x \le n$ and $x > n$ is completely different for the sake of accuracy.

Bessel Function Yn(x)

The Bessel function $Y_n(x)$ is defined by

$$Y_n(x) = \frac{J_n(x)\cos(n\pi) - J_{-n}(x)}{\sin(n\pi)} \text{ with } n = 0, 1, \ldots$$

based on the definition of the Bessel functions $J_n(x)$.



DBL **x** is any real number.

U32 **n** is a nonnegative integer.

DBL **Yn(x)** is the Bessel function of the second kind of order *n*.

See the description of the Bessel Function Jn(x) VI in this chapter for more information.

Bessel Polynomial

The Bessel polynomial $P_n(x)$ of order *n* is defined by a recurrence relation

$$P_n(x) = P_{n-1}(x) + \frac{x^2}{4(n-1)^2 - 1} P_{n-2}(x) \text{ for } n = 2, 3 \ldots$$

where $P_0(x) = 1$ and $P_1(x) = 1 + x$.

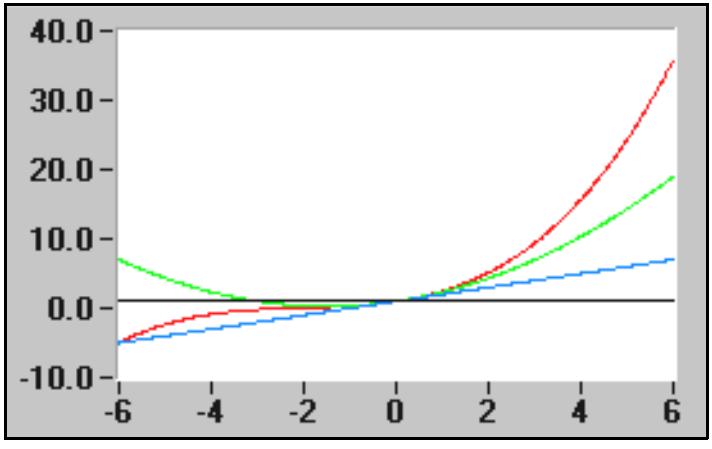
DBL **x** is any real number.

U32 **n** is a positive integer.

DBL Bessel polynomial (*n*, *x*) is the result of the calculation of $P_n(x)$ for the given values of *n* and *x*.

Note: ***Bessel Polynomials have strong connections to the Bessel filters. See the*** *LabVIEW Analysis VI Reference Manual* ***for details.***

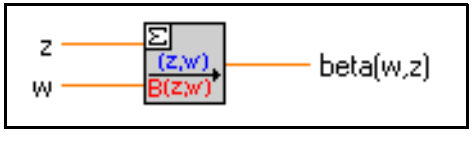
The following diagram shows the first four Bessel polynomials from bottom to top on the right side. The range of x is from [–6, 6].



Beta Function

The beta function $B(w, z)$ is defined by

$$B(w, z) = \frac{\Gamma(w)\Gamma(z)}{\Gamma(w+z)}$$



DBL **z** is any real number.

DBL **w** is any real number.

DBL **beta (w,z)** is the result of the calculation of $B(w, z)$ for the given values of w and z.

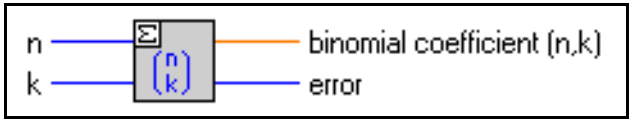
The calculation of $B(w, z)$ uses the Gamma Function VI.

☞ **Note:** ***B(w, z) is undefined, if w or z is a non-positive integer.***

Binomial Coefficient

The binomial coefficient is determined by

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$



U32 **n** is any non-negative integer.

U32 **k** is any non-negative integer.

DBL **binomial coefficient (n,k)** is the result of the calculation of the binomial coefficient for the given values of *n* and *k*.

I32 **error**. See Appendix A, *Error Codes*, for a list of Error Codes, if $n < k$.

Binomial coefficients can have many digits, even in the case of relatively small numbers *n* and *k*. The data type most suited for the binomial coefficient is a double real. You can directly calculate the factorial functions, *n!*, *k!*, and *(n – k)!*, with the Gamma Function VI.

Chebyshev Polynomial

The Chebyshev polynomial $T_n(x)$ is defined by

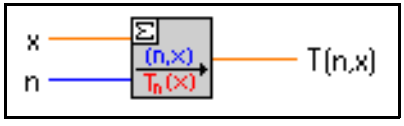
$$T_n(x) = \cos(n \arccos x) \text{ for } n = 0, 1, \ldots \text{ and real numbers } x.$$

These functions form the base of the so called Chebyshev approximation. For $i \neq j$ it is

$$\int_{-1}^{1} \frac{T_i(x)T_j(x)}{\sqrt{1-x^2}} dx = 0$$

All $T_n(x)$ form an orthogonal system over the weight function

$$\frac{1}{\sqrt{1-x^2}}$$



DBL **x** is the real argument.

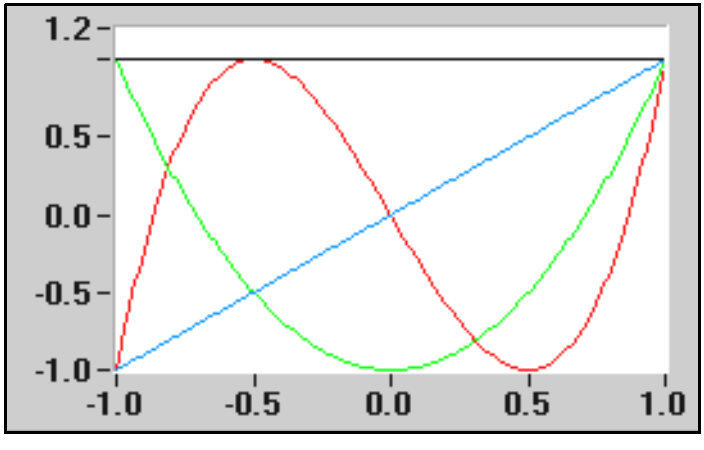
U32 **n** is the order of the Chebyshev polynomial.

DBL **T(n,x)** is the value of the *n*th Chebyshev polynomial at the point *x*.

☞ **Note:** ***The result of this definition does not look like a polynomial at first glance, but you can use trigonometric rules to show that T_n is a polynomial of degree n in the variable x.***

☞ **Note:** ***Chebyshev polynomials have strong connections to some very important filter types. See the*** *LabVIEW Analysis VI Reference Manual* ***for details.***

The following diagram shows the first four Chebyshev polynomials of degrees 0, 1, 2, and 3.

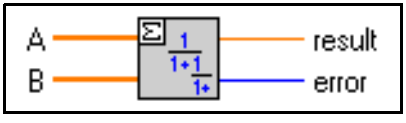


Continued Fraction

The continued fraction of two sequences $(a_0, a_1, \ldots, a_n)$ and $(b_0, b_1, \ldots, b_n)$ is defined by the following term.

$$result = \cfrac{a_0}{b_0 + \cfrac{a_1}{b_1 + \ldots}}$$

Continued fractions are invaluable tools for calculating special functions.



[DBL] **A** is the 1D array of the numerator part of the continued fraction.

[DBL] **B** is the 1D array of the denominator part of the continued fraction.

DBL **result** is a real value representing the result of the continued fraction.

I32 **error**. See Appendix A, *Error Codes*, for a list of Error Codes, if the dimensions of **A** and **B** are not equal.

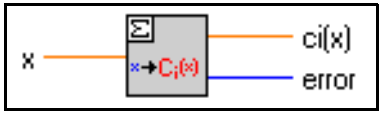
Cosine Integral

The cosine integral is defined by

$$ci(x) = \gamma + \ln x + \int_0^x \frac{\cos s - 1}{s} ds \text{ with the Euler constant } \gamma.$$

It is $ci(-x) = ci(x) - i\pi$.

The algorithm accepts only nonnegative real numbers as input.



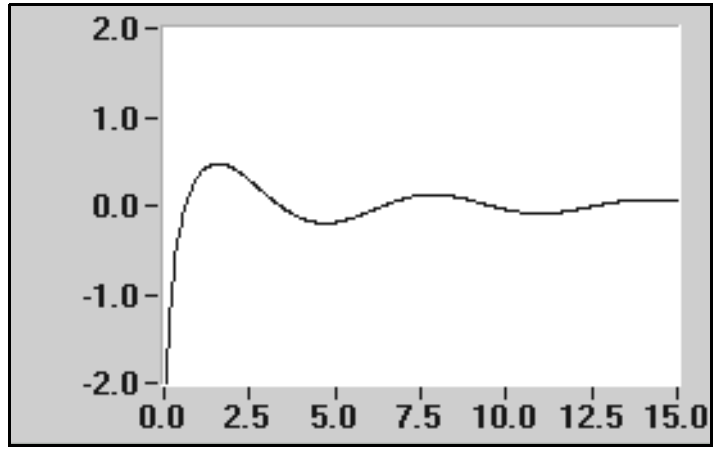
x is any real non-negative number.

ci(x) is the result of the calculation of the cosine integral for the given value of *x*.

error. See Appendix A, *Error Codes*, for a list of Error Codes, if $x \leq 0$.

The calculation of *ci(x)* can be done with the help of *E(x)*. See the Sine Integral VI description in this chapter for more information.

The following diagram shows the graph of the cosine integral, in the interval (0,15).

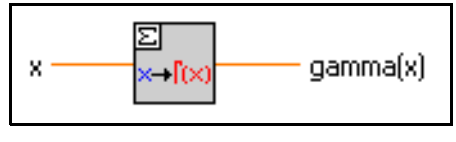


Gamma Function

The gamma function of *x* is the generalization of the common factorial function *n!*. The relation between these two functions is $\Gamma(n+1) = n!$ for all natural numbers *n*. The gamma function is defined by

$$\Gamma(x) = \int_0^x s^{x-1} \exp(-s)\,ds$$

for real and complex *x*, and has the property $\Gamma(x+1) = x\Gamma(x)$.



DBL **x** is any real number.

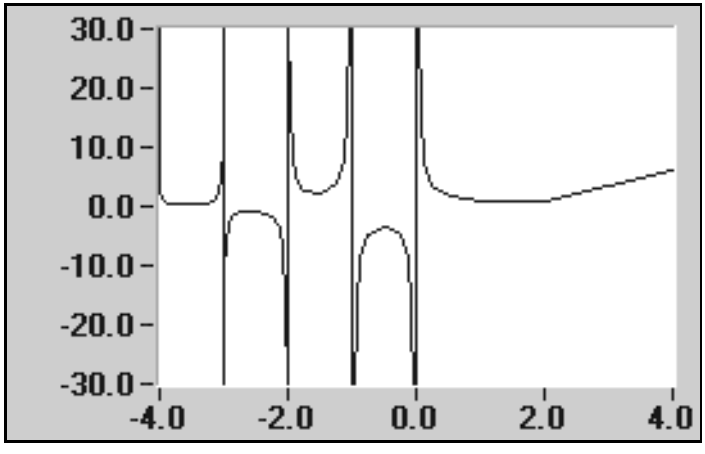
DBL **gamma (x)** has singularities for all nonpositive integers *x*.

The calculation of $\Gamma(x+1)$ is based on the Lanszos formula

$$\Gamma(x+1) = (x+5.5)^{x+0.5}\exp(-x-5.5)\sqrt{2\pi}\left[c_0 + \frac{c_1}{x+1} + \ldots + \frac{c_6}{x+6}\right]$$

with fixed and well calculated $c_0, c_1, \ldots, c_6$.

The following diagram shows the graph of the gamma function in the interval (–4, 4).

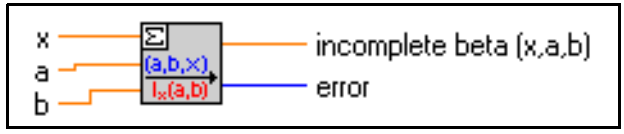


Incomplete Beta Function

The incomplete beta function is defined by

$$I_x(a, b) = \frac{1}{B(a,b)}\int_0^x s^{a-1}(1-s)^{b-1}ds \text{ with } a, b > 0 \text{ and } 0 \le x \le 1$$

where $B(a, b)$ denotes the beta function of *a* and *b*.



DBL **x** is a real number between 0 and 1

DBL **a** is a positive real number.

DBL **b** is a positive real number.

DBL **incomplete beta (x,a,b)** is the result of the calculation of $I_x(a,b)$ for the given values of x, a, and b.

I32 **error**. See Appendix A, *Error Codes*, for a list of Error Codes, if

$x \notin [0, 1]$

$a \leq 0$

$b \leq 0$

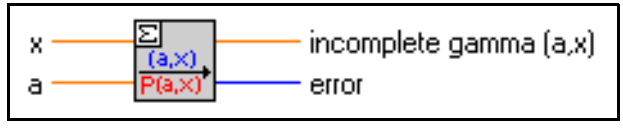
An efficient strategy for calculating of $I_x(a,b)$ is based on the continued fraction

$$I_x(a,b) = \frac{x^a(1-x)^b}{aB(a,b)}\left[\frac{1}{1+\frac{d_1}{1+d_{2\ldots}}}\right]$$ with d_i depending on i, a, b, and x.

Incomplete Gamma Function

The incomplete gamma function is defined by

$$P(a,x) = \frac{1}{\Gamma(a)}\int_0^x \exp(-s)s^{a-1}ds \text{ for } a > 0$$



DBL **x** is any real number.

a is a positive real number.

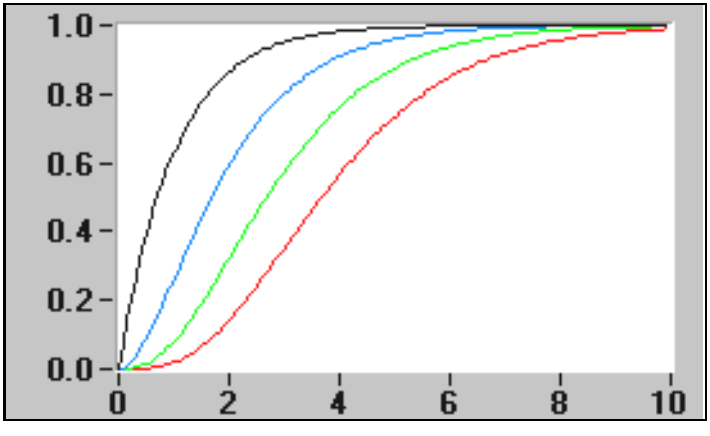
incomplete gamma (a,x) is the result of the calculation of $P(a, x)$ for the given values of a and x.

error. See Appendix A, *Error Codes*, for a list of Error Codes, if $a \le 0$.

The integral in the definition of $P(a, x)$ can approximately be derived by the following equation.

$$\int_0^x \exp(-s)s^{a-1}ds = \exp(-x)x^a \sum_{n=0}^{100} \frac{\Gamma(a)}{\Gamma(a+1+n)} x^n$$

The following diagram shows the incomplete gamma functions with $a = 0, 1, 2, 3$ from top to bottom.



Jacobian Elliptic Function

The value of *sn* (Jacobian Elliptic Function) is determined by the relation

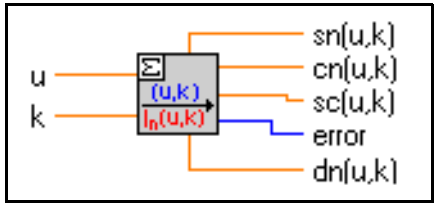
$$u = \int_0^{sn} \frac{ds}{\sqrt{(1-s^2)(1-k^2s^2)}}$$

where *u* and *k* are given real numbers. The other Jacobian Elliptic functions have the following definitions.

$$sn^2 + cn^2 = 1$$

$$k^2sn^2 + dn^2 = 1$$

$$sc = \frac{sn}{cn}$$



DBL **u** is any real number.

DBL **k** is a real number with $0 \le k \le 1$.

DBL **sn(u, k)**.

DBL **cn(u, k)**.

DBL **sc(u, k)**.

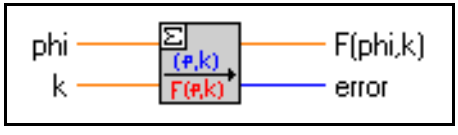
DBL **dn(u, k)**.

I32 **error codes**. See Appendix A, *Error Codes*, for a list of error codes if $k < 0$ or $k > 1$.

Legendre Elliptic Integral 1st Kind

The Legendre Elliptic Integral 1st kind is defined by the following equation.

$$F(\varphi, k) = \int_0^{\varphi} \frac{d\vartheta}{\sqrt{1 - k^2 \sin^2 \vartheta}}$$



DBL **phi** is any real number.

DBL **k** is a real number with $0 \le k \le 1$.

DBL **F(phi, k)** is the result of the calculation of *F(φ, k)* for the given values of φ and k.

I32 **error**. See Appendix A, *Error Codes*, for a list of Error Codes, if $k < 0$ or $k > 1$.

The calculation uses the relation

$$F(\varphi, k) = \sin\varphi R_F(\cos^2\varphi, 1 - k^2\sin^2\varphi, 1)$$

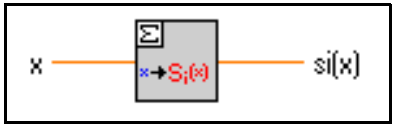
where R_F is the elliptic integral in the Carlson form

$$R_F(x, y, z) = \frac{1}{2}\int_0^{\infty} \frac{ds}{\sqrt{(s + x)(s + y)(s + z)}}$$

Sine Integral

The sine integral is defined by

$$si(x) = \int_0^x \frac{\sin s}{s} ds \text{ . It is } si(-x) = -si(x)$$



DBL **x** is any real number.

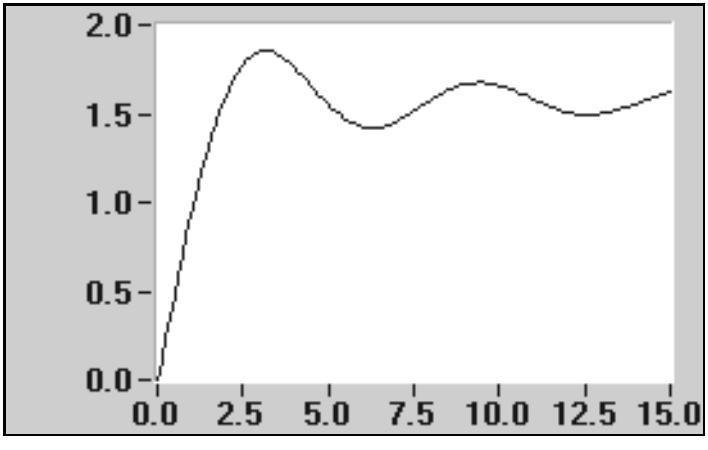
DBL **si(x)** is the result of the calculation of the sine integral for the given value of *x*.

An efficient algorithm is based on a continued fraction of the combined function

$$E(x) = -ci(x) + i\left(si(x) - \frac{\pi}{2}\right) \text{ more precisely on}$$

$$E(x) = \exp(-ix)\left(\frac{1}{1+ix-}\frac{1^2}{3+ix-}\frac{2^2}{5+ix-}\cdots\right)$$

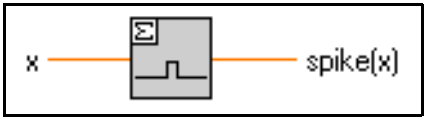
The following diagram shows the graph of the sine integral in the interval (0,15).



Spike Function

The spike function is defined by

$$spike(x) = \begin{cases} 1 & if \quad 0 \le x < 1 \\ 0 & else \end{cases}$$



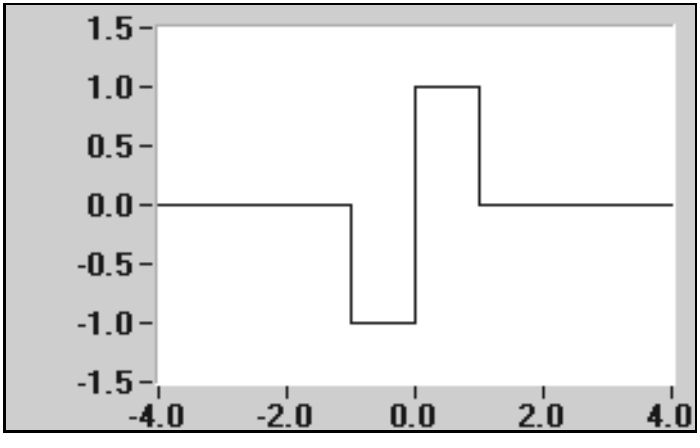
DBL **x** is any real number.

DBL **spike(x)** is the value of *spike(x)* for the given value of *x*.

Note: ***You can define more complex functions by variation and combination of the Step Function, Spike Function, and Square Function VIs, respectively.***

The diagram below illustrates the following example.

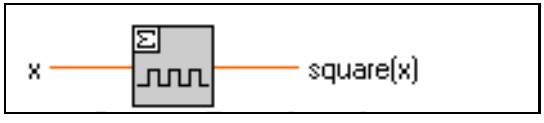
Example: $spike(x) - spike(-x)$ in the interval (–4.0, 4.0)



Square Function

The square function is defined by the following equation.

$$square(x) = \begin{cases} 1 & if \quad 2n \le x < 2n+1 \qquad n = \ldots, -1, 0, 1, \ldots \\ 0 & if \quad 2n+1 \le x < 2n+2 \quad n = \ldots, -1, 0, 1, \ldots \end{cases}$$



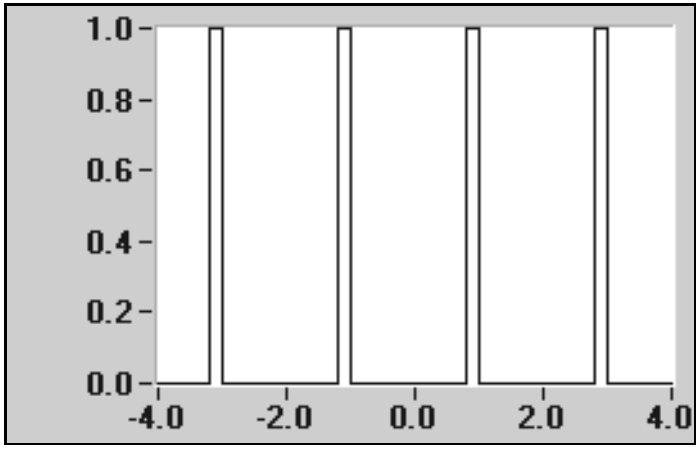
DBL **x** is any real number.

DBL **square(x)** is the value of *square(x)* for the given value of *x*.

☞ **Note:** ***You can define more complex functions by variation and combination of the Step Function, Spike Function and Square Function VIs, respectively.***

The diagram below illustrates the following example.

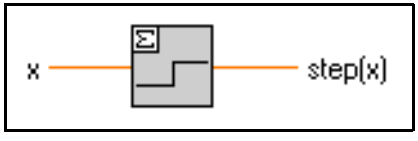
Example: $square(x)*square(x-0.8)$ in the interval (–4.0, 4.0)



Step Function

The step function is defined by

$$step(x) = \begin{cases} 0 & if\ x < 0 \\ 1 & else \end{cases}$$



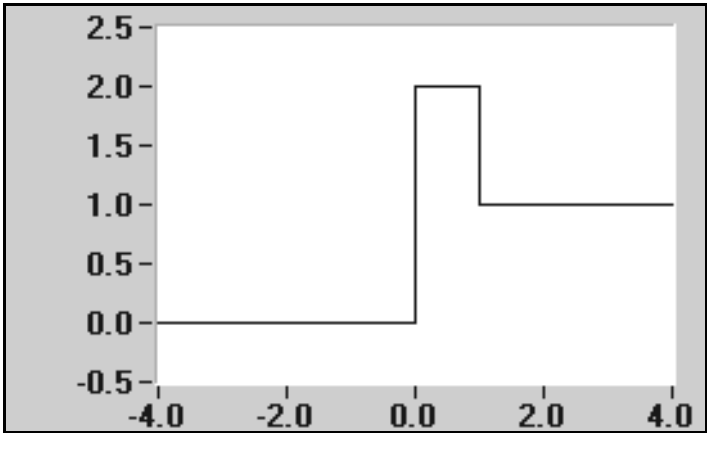
DBL **x** is any real number.

DBL **step(x)** is the value of *step(x)* for the given value of *x*.

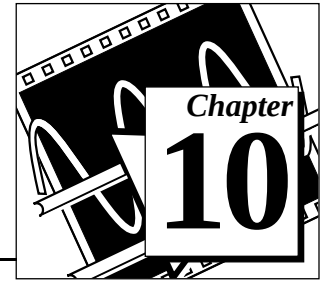
Note: ***You can define more complex functions by variation and combination of the Step Function, Spike Function, and Square Function VIs, respectively.***

The diagram below illustrates the following example.

Example: step(x) + spike(x) in the interval (–4.0, 4.0)



Transform VIs

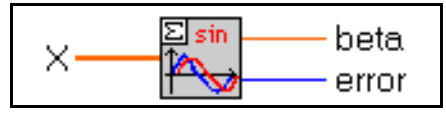


These VIs implement some transforms commonly used in mathematics and signal processing. They are available in the `TRANS.LLB` library.

Transform VI Descriptions

Buneman Frequency Estimator

This VI estimates the frequency of a given sine wave with unknown wavelength.



[DBL] **X** is the sampled signal at consecutive times.

DBL **beta** is the estimation of the frequency of the sine wave represented by **X**.

I32 **error** See Appendix A, *Error Codes*, for a list of error codes.

Sometimes, an underlying time signal is not exactly periodic with period *n*, where *n* denotes the size of the data array. How, then, do you determine the unknown period? The Buneman algorithm calculates the unknown frequency β by

$$\beta = b + \frac{n}{\pi}\operatorname{atan}\left(\frac{\sin\left(\frac{\pi}{n}\right)}{\cos\left(\frac{\pi}{n}\right) + \frac{|F_b(X)|}{|F_{b+b}(X)|}}\right)$$

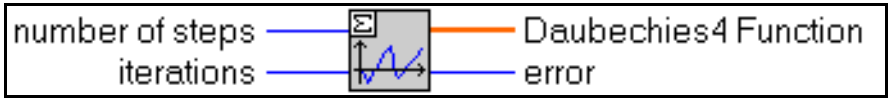
Here F_b denotes the value of the Fourier transform of the signal **X** at the frequency *b*. The value of *b* can be determined by the greatest value of

$$|F_b(X)|$$

The formula for β is exact for pure sine waves and a good estimation in all other cases.

Daubechies4 Function

The Daubechies4 function is calculated at evenly spaced sampled points.



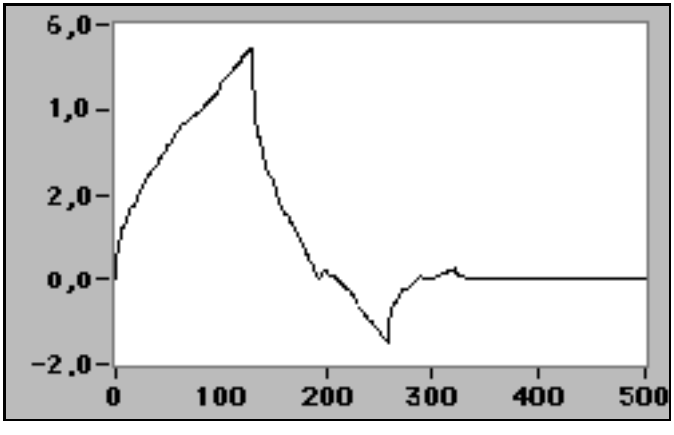
U32 **number of steps**. The algorithm calculates exactly 4 times the **number of steps** points of the Daubechies4 function. The default value is 128.

U32 **iterations** is the number of iterations. The quality of the output increases with the number of iterations. The default value is 10.

[DBL] **Daubechies4 Function**.

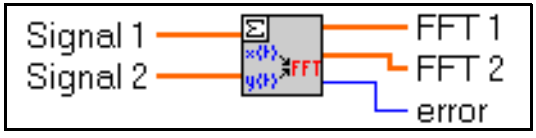
I32 **error**. See Appendix A for error codes.

The following figure shows the Daubechies4 function. This function is the base of a well known class of wavelet transforms.



Dual Signal FFT

One step computation of the Fourier transforms of two real signals of equal length.



[DBL] **Signal 1** is the array of the first signal. The length of **Signal 1** must equal the length of **Signal 2**.

[DBL] **Signal 2** is the array of the second signal. The length of **Signal 2** must equal the length of **Signal 1**.

[CDB] **FFT 1** is the complex Fourier transform of the first signal.

[CDB] **FFT 2** is the complex Fourier transform of the second signal.

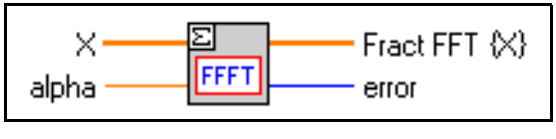
[I32] **error**. See Appendix A for error codes. Especially, the length of **Signal 1** and **Signal 2** have to be the same.

From two signals X and Y of the same length, an artificial signal $Z = X + iY$ can be constructed. From the complex Fourier transform of the complex signal Z the complex Fourier transforms of the signals X and Y can be simply derived. This method is much faster than the successive application of the real Fourier transform of X and Y.

☞ **Note:** ***This VI is especially useful, if you need to calculate a continued stream of Fourier transforms of real signals.***

Fractional FFT

Realizes the fractional fast Fourier transform of complex signals with arbitrarily chosen signal lengths.



[CDB] **X** is a given time signal. It may be complex.

[CDB] **alpha** can be any complex number. The default value is 0.00 + 0.00*i*.

[CDB] **Fract FFT {X}** is the fractional fast Fourier transform.

[I32] **error.** See Appendix A, *Error Codes*, for a list of error codes.

The fractional Fourier transform is a natural generalization of the classical Fourier transform. With $X = \{x_0, x_1, \ldots, x_{n-1}\}$ it is

$$Fractional\ FT\{X\}(j) = \sum_{k=0}^{n-1} \exp[-2\pi ijk\alpha]x_k$$

where α is a complex number.

The classical Fourier transform is a special case, namely

$$\alpha = \frac{1}{n}$$

There is a very efficient algorithm for calculating the fractional Fourier transform called Fractional Fast Fourier Transform(FFFT). This algorithm is based on the chirp z strategy. The starting point is the obvious algebraic relation $2jk = j^2 + k^2 - (k-j)^2$. With this relation it is

$$Fractional\ FFT\{X\}(j) = \exp(-\pi ij^2\alpha)\sum_{k=0}^{n-1} y_k z_{j-k}$$

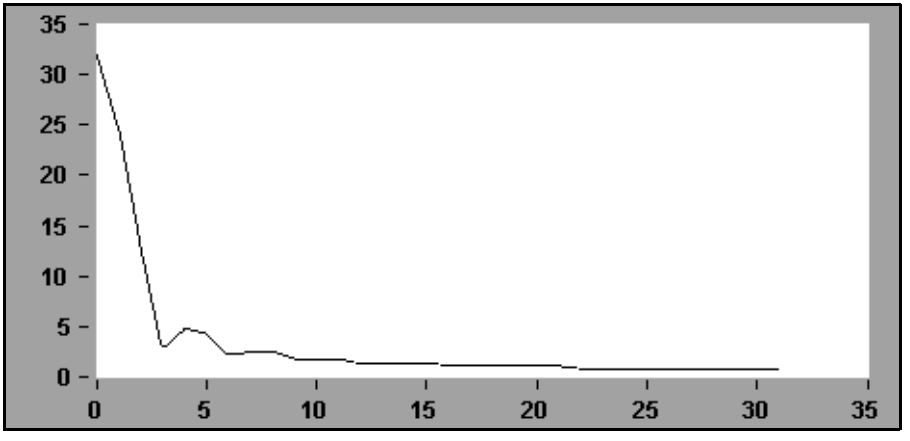
with

$$y_k = x_k \exp(-\pi ik^2\alpha) \text{ and } z_k = \exp(\pi ik^2\alpha)$$

It is possible to extend the signals Y and Z in such a manner that the above sum is a cyclic convolution.

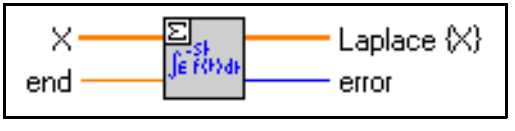
☞ **Note:** ***The Laplace transform is a special case of the fractional Fourier transform with alpha being purely imaginary. The fractional Fourier transform has many other interesting applications, such as analyzing signals with noninteger periodic components, high resolution trigonometric interpolation, and detecting lines in images.***

The following diagram shows the fractional fast Fourier transform of the function $f(x) = x$ with $\alpha = 0.1 + 0.0002i$.



Laplace Transform Real

Realizes the real Laplace transform of a real-time signal.



The real Laplace transform of a real signal *x(s)* is defined by

$$Laplace\{X\}(s) = \int_0^{\infty} x(t)\exp(-st)dt \text{ for } s \geq 0\text{, and } s \text{ real.}$$

Here, *x(t)* is defined for all $t \geq 0$. The discrete version of the Laplace transform of a discretely and evenly-sampled signal is a simple generation of the above continuous version.

[DBL] **X** is the array describing the evenly sampled time signal. The first element of this array belongs to $t = 0$, the last to $t =$ **end**.

DBL **end** is the instant in time of the last sample. The entire sample interval is between 0 and **end**.

Laplace {X} is the result of the Laplace transform as an array.

error. See Appendix A, *Error Codes*, for a list of error codes, if **end** is out of range.

The definition of the Laplace transform is not of much use if the time signal increases very rapidly with the time. The discrete version of the Laplace transform cannot fully detect the convergence behavior of the original definition.

The discrete version of the Laplace transform is computationally very expensive. An efficient strategy for the discrete Laplace transform is based on the Fractional Fast Fourier Transform (FFFT). The definition of the FFFT is as follows:

$$FFFT\{X\}(t) = \int_{-\infty}^{\infty} x(s)\exp(-i\alpha st)ds$$

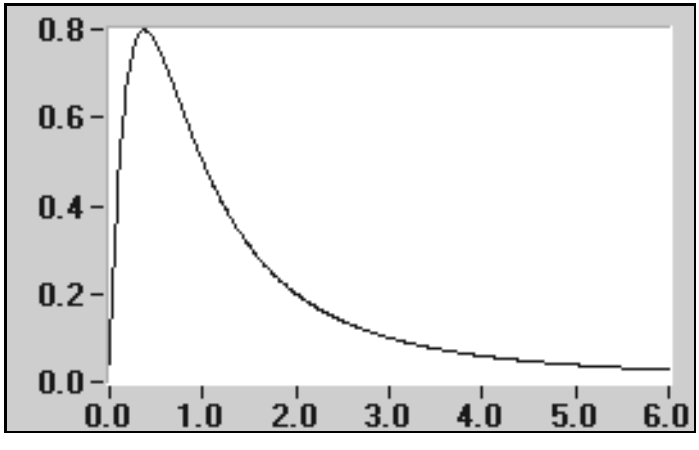
with an arbitrarily chosen complex α. See the description of the Fractional FFT VI in this chapter for more details.

Example

The following diagram shows the real Laplace transform of the function $f(t) = \sin(t)$ in the interval (0, 6). This is entered on the front panel as:

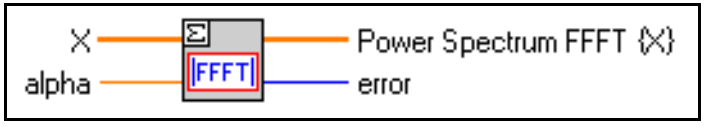
end: 6.00

X: values of sin(t) for $0 \le t \le 6$



Power Spectrum Fractional FFT

Yields the absolute values of the fractional Fourier transform of a complex signal.



[CDB] **X** is the given time signal.

CDB **alpha** can be any complex number. The default value is 0.00 + 0.00*i*.

[DBL] **Power Spectrum FFFT {X}** is the magnitude of the fractional FFT.

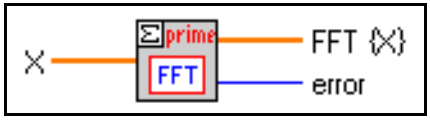
I32 **error**. See Appendix A, *Error Codes*, for a list of error codes.

The Power Spectrum Fractional FFT VI is defined by

$$Power\ Spectrum\ FFFT\{X\} = |FFFT\{X\}|$$

Prime FFT

Prime FFT is a special form of the Fourier transform. Use this VI if signal length is a prime.



[CDB] **X** is a complex array of prime length.

[CDB] **FFT {X}**.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes. Especially, the algorithm fails, if **X** does not have prime length.

☞ **Note:** ***The Prime FFT VI is computationally more efficient than classical algorithms. There are also special algorithms combining different Prime FFTs to realize the Fourier transform of signals of arbitrary length.***

Determining the Prime FFT is a five step process.

1. Determination of a generator g of the group modulo p, where p is the prime length of **X**. A generator g is such a natural number that
$1 < g < p \quad with \quad \{g^0, g^1, \ldots, g^{p-1}\} = \{1, 2, \ldots, p-1\}$

2. Reduction $\{x_0, x_1, \ldots, x_{p-1}\} \rightarrow \{x_1, x_2, \ldots, x_{p-1}\}$
3. Construction of two new signals each with a power of 2 length according to the following scheme:

first signal:

$\{0, 0, \ldots, x_{g^0}, x_{g^1}, \ldots, x_{g^{p-2}}, 0, 0, \ldots, 0\}$

second signal:

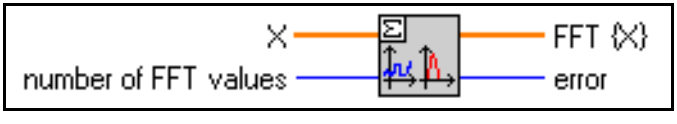
$$\left\{w^{g^{p-2}}, w^{g^{p-3}}, \ldots, w^{g^1}, w^{g^0}, w^{g^{p-2}}, \ldots, w^{g^1}, 0, 0, \ldots, 0\right\}$$

where $w = \exp\left\{-\frac{2\pi i}{p}\right\}$

4. Convolution of these two signals with the help of the Convolution Theorem.
5. Reduction and reorganization of the convolution result.

Sparse FFT

Computes the first elements of the Fourier transform of the input signal **X**. You can use this VI to perform an FFT on an array of complex numbers.



[CDB] **X** is the given time signal.

I32 **number of FFT values** is the number of interesting Fourier transform elements. The default value is 1.

[CDB] **FFT {X}** is the truncated Fourier transform of **X**.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes. The size of **X** has to be a multiple of the **number of FFT values**.

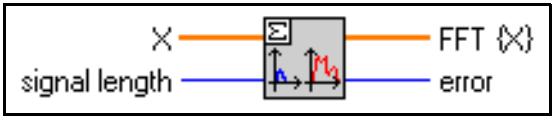
Let n be the length of X and m the number of FFT values. Then, the Fourier transform can be calculated by

$$FFT\{X\}(j) = \sum_{k=0}^{d-1} \exp\left(\frac{-2\pi ijk}{n}\right)\left\{\sum_{l=0}^{m-1} \exp\left[\frac{-2\pi ilk}{m}\right] x_{k+ld}\right\}$$

where $n = dm$ and $j = 0,\ldots, m-1$. In other words, the Fourier transform FFT {X} can be reduced to the combination of Fourier transforms of the length m. Under certain circumstances this is more efficient than the classical Fourier transform method. The efficiency depends strongly on the relation between m and n.

Sparse Signal FFT

The VI computes the Fourier transform of a signal **X**, a so called sparse signal.



[CDB] **X** is the nonzero part of the signal.

I32 **signal** length is the length of the combined (zero-padded) signal, signal length can not be smaller than m, where m is the length of X. The default value is 1.

[CDB] **FFT {X}** is the Fourier transform of the combined signal. The size of this array is **signal length**.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes. **signal length** must be a multiple of **X**.

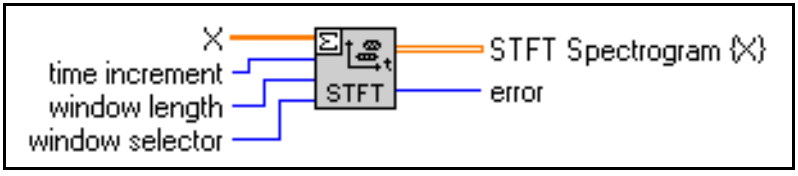
Let n be the length of the combined signal and m the length of **X**. Then it is

$$FFT\{X\}(j) = \sum_{k=0}^{m-1} \exp\left(\frac{-2\pi ikj}{n}\right) x_k$$

This can efficiently be calculated using the Fractional Fourier Transform. The efficiency depends strongly on the relation between m and n.

STFT Spectrogram

Computes the signal energy distribution in the joint time-frequency domain, using the Short-Time Fourier Transform (STFT) algorithm. This VI performs a sliding FFT.



[DBL] **X** is the time waveform.

I32 **time increment** is the base 2 logarithm of the time spacing, in samples, between each row of the Spectrogram output. For example, if you sampled the time waveform at f_S Hz, the spacing between the rows of Spectrogram is **time increment** /f_S second.

Increasing **time increment** decreases the computation time and reduces memory requirements, but also reduces time-domain resolution. Decreasing **time increment** improves time-domain resolution, but increases the computation time and memory requirements.

The default value is 1.

I32 **window length** is the actual length of the selected window. The default value is 1.

I32 **window selector** determines the type of analysis window the VI uses to compute **STFT Spectrogram {X}**. The **window selector** parameter can have the following values.

0: Rectangular

1: Blackman

2: Hamming

3: Hanning

4: Gaussian

The default is the Rectangular window.

[DBL] **STFT Spectrogram {X}** is a 2D array that describes the time waveform energy distribution in the joint time-frequency domain.

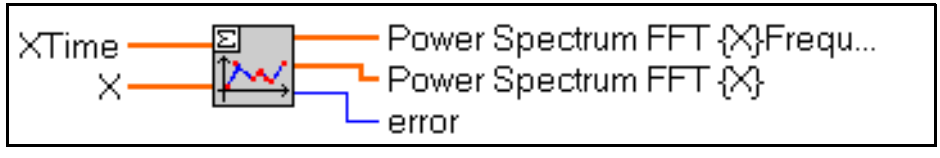
The number of rows (time axis) in **STFT Spectrogram {X}** is equal to the number of elements in the time waveform divided by **time increment**, and then rounded up. The number of columns (frequency axis) in **STFT Spectrogram {X}** is equal to

$$\frac{window\ length}{2} + 1$$

[I32] **error**. See Appendix A, *Error Codes*, for a list of error codes.

Unevenly Sampled Signal Spectrum

Calculates the power spectrum of a signal that is unevenly spaced in time.



[DBL] **XTime** is the discrete- and unevenly-spaced times.

[DBL] **X** represents the data material at times **XTime**. There is a one-to-one relation between **XTime** and **X**.

[DBL] **Power Spectrum FFT {X} Frequency** are the frequency points at which the power spectrum is calculated.

[DBL] **Power Spectrum FFT {X}** is the power spectrum, in the sense of the Lomb normalized periodogram.

[I32] **error**. See Appendix A, *Error Codes*, for a list of error codes, if **XTime** and **X** have different lengths.

The algorithm used is based on the Lomb normalized periodogram. Let the data x_k be given at the time points t_k, i.e. $X = \{x_0, x_1, \ldots, x_{n-1}\}$ and $XTime = \{t_0, t_1, \ldots, t_{n-1}\}$. Furthermore,

$$\bar{x} = \frac{1}{n}\sum_{k=0}^{n-1} x_k \text{ and } \sigma^2 = \frac{1}{n-1}\sum_{k=0}^{n-1}(x_k - \bar{x})^2$$

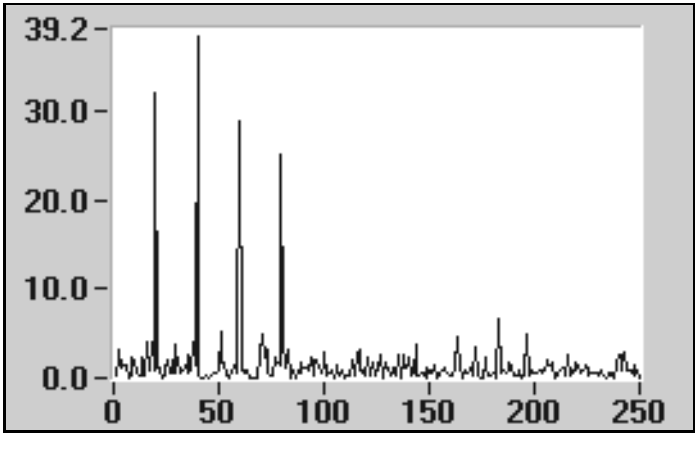
Then the Lomb normalized periodogram is defined as follows:

$$P(\omega) = \frac{1}{2\sigma^2}\left\{\frac{\left[\sum_{k=0}^{n-1}(x_k-\bar{x})\cos\omega(t_k-\tau)\right]^2}{\sum_{k=0}^{n-1}\cos^2\omega(t_k-\tau)} + \frac{\left[\sum_{k=0}^{n-1}(x_j-\bar{x})\sin\omega(t_k-\tau)\right]^2}{\sum_{k=0}^{n-1}\sin^2\omega(t_k-\tau)}\right\}$$

with

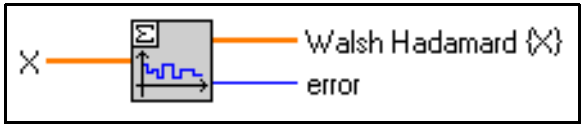
$$\tau = \frac{1}{2\omega}\arctan\left(\frac{\sum_{k=0}^{n-1}\sin 2\omega t_k}{\sum_{k=0}^{n-1}\cos 2\omega t_k}\right)$$

The following diagram shows the Fourier transform, of length 256, of a signal that has been sampled at unequal intervals of time. The signal is a combination of sine waves of frequencies 20, 40, 60, and 80 Hz. The duration of the signal is 1 sec. The sampling frequency was chosen as 256 Hz, giving the frequency resolution of 1 Hz.



Walsh Hadamard

Realizes the real Walsh Hadamard transform.



[DBL] **X** is an array of power of two length.

[DBL] **Walsh Hadamard {X}**.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes. Especially, the signal length of **X** has to be a power of 2.

Note: ***The Walsh Hadamard transform has similar properties to the more well known Fourier transform, but the computational effort is considerably smaller.***

The Walsh Hadamard transform is based on an orthogonal system consisting of functions of only two elements –1 and 1. For the special case of $n = 4$ the Walsh Hadamard transform of the signal $X = \{x_0, x_1, x_2, x_3\}$ can be noted in the following matrix form.

$$WH\{X\} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

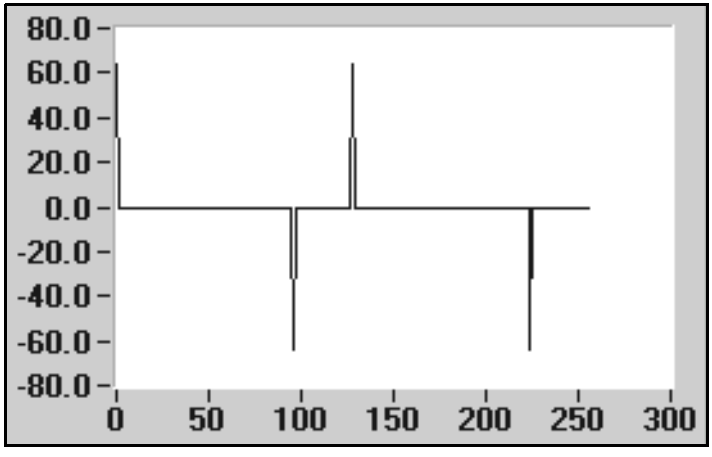
The matrices on the right hand side have simple construction rules. If WH_n and WH_{n+1} denote the Walsh Hadamard matrices of dimension 2^n and 2^{n+1}, respectively, the rule is

$$WH_{n+1} = \begin{bmatrix} WH_n & WH_n \\ WH_n & -WH_n \end{bmatrix}$$

where $-WH_n$ is meant in the element wise sense.

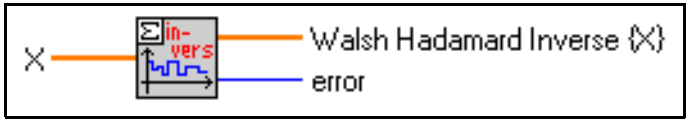
Note: ***The Walsh Hadamard transform fulfills the Convolution Theorem:*** $WH\{X*Y\} = WH\{X\}WH\{Y\}$

The following diagram shows the Walsh Hadamard transform of a pulse pattern signal of length 256, delay 32, and width 64.



Walsh Hadamard Inverse

Realizes the inverse of the real Walsh Hadamard transform.



[DBL] **X** is an array of power of 2 length.

[DBL] **Walsh Hadamard Inverse {X}**.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes. Especially, the signal length of **X** has to be the a power of 2.

If $WH\{X\}$ denotes the Walsh Hadamard transform of **X** and $WHI\{X\}$ the inverse Walsh Hadamard transform, then it is

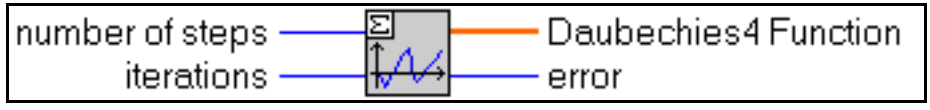
$$WHI\{WH\{X\}\} = WH\{WHI\{X\}\} = X$$

Furthermore, the following very simple formula is valid.

$WHI\{X\} = \frac{1}{n}WH\{X\}$, where n is the length of the signal **X**.

Wavelet Transform Daubechies4

Realizes the Wavelet transform based on the Daubechies4 function.



[DBL] **X** are the samples of the input signal. The length of the signal has to be a power of 2, otherwise an error code is given.

[DBL] **Wavelet Daubechies4 {X}**.

[I32] **error**. See Appendix A, *Error Codes*, for a list of error codes.

Note: ***Wavelet transforms form an extremely fast growing part of the general signal theory. Two important applications are compressions of data and solutions of large systems of linear equations.***

The Wavelet Transform Daubechies4 transform can be defined using the transformation matrix

$$C = \begin{bmatrix} c_0 & c_1 & c_2 & c_3 & & & & & & \\ c_3 & -c_2 & c_1 & -c_0 & & & & & & \\ & & c_0 & c_1 & c_2 & c_3 & & & & \\ & & c_3 & -c_2 & c_1 & -c_0 & & & & \\ & & & \cdot & & & & & & \\ & & & \cdot & & & & & & \\ & & & & & & & & & \\ & & & & & & c_0 & c_1 & c_2 & c_3 \\ & & & & & & c_3 & -c_2 & c_1 & -c_0 \\ c_2 & c_3 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & c_0 & c_1 \\ c_1 & -c_0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & c_3 & -c_2 \end{bmatrix}$$

Here blank entries signify zeroes. The numbers c_0, c_1, c_2, and c_3 have to fulfill certain orthogonal properties

$$
\begin{aligned}
c_0^2 + c_1^2 + c_2^2 + c_3^2 &= 1 \\
c_2 c_0 + c_3 c_1 &= 0 \\
c_3 - c_2 + c_1 - c_0 &= 0 \\
0c_3 - 1c_2 + 2c_1 - 3c_0 &= 0
\end{aligned}
$$

with the unique solution

$$c_0 = \frac{1+\sqrt{3}}{4\sqrt{2}}$$

$$c_1 = \frac{3+\sqrt{3}}{4\sqrt{2}}$$

$$c_2 = \frac{3-\sqrt{3}}{4\sqrt{2}}$$

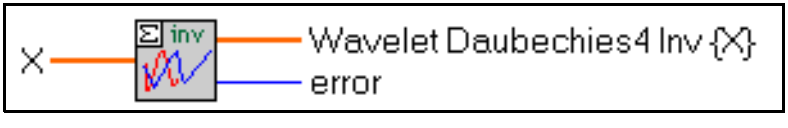
$$c_3 = \frac{1-\sqrt{3}}{4\sqrt{2}}$$

Note: ***You can solve the above system of nonlinear equations in c_0, c_1, c_2, and c_3 directly with the Nonlinear System Single Solution VI of this package.***

The Wavelet Daubechies4 transform of the array X is defined by
$Wavelet\ Daubechies4\{X\} = C*X$.

Wavelet Transform Daubechies4 Inverse

Realizes the inverse of the Wavelet transform based on the Daubechies4 function.



[DBL] **X** are the samples of the input signal. The length of the signal has to be a power of 2, otherwise an error code is given.

[DBL] **Wavelet Daubechies4 Inv {X}**.

[I32] **error**. See Appendix A, *Error Codes*, for a list of error codes.

The Wavelet Transform Daubechies4 Inverse transform can be defined with the help of the transformation matrix

$$C = \begin{bmatrix} c_0 & c_3 & & & & & & c_2 & c_1 \\ c_1 & -c_2 & & & & & & c_3 & -c_0 \\ c_2 & c_1 & c_0 & c_3 & & & & & \\ c_3 & -c_0 & c_1 & -c_2 & & & & & \\ & & & \cdot & & & & & \\ & & & \cdot & & & & & \\ & & & & c_2 & c_1 & c_0 & c_3 & \\ & & & & c_3 & -c_0 & c_1 & -c_2 & \\ & & & & & & c_2 & c_1 & c_0 & c_3 \\ & & & & & & c_3 & -c_0 & c_1 & -c_2 \end{bmatrix}$$

Here blank entries signify zeroes. The numbers c_0, c_1, c_2 and c_3 have to fulfill certain orthogonal properties, namely

$$\begin{aligned} {c_0}^2 + {c_1}^2 + {c_2}^2 + {c_3}^2 &= 1 \\ c_2 c_0 + c_3 c_1 &= 0 \\ c_3 - c_2 + c_1 - c_0 &= 0 \\ 0c_3 - 1c_2 + 2c_1 - 3c_0 &= 0 \end{aligned}$$

with the unique solution

$$c_0 = \frac{1+\sqrt{3}}{4\sqrt{2}}$$

$$c_1 = \frac{3+\sqrt{3}}{4\sqrt{2}}$$

$$c_2 = \frac{3-\sqrt{3}}{4\sqrt{2}}$$

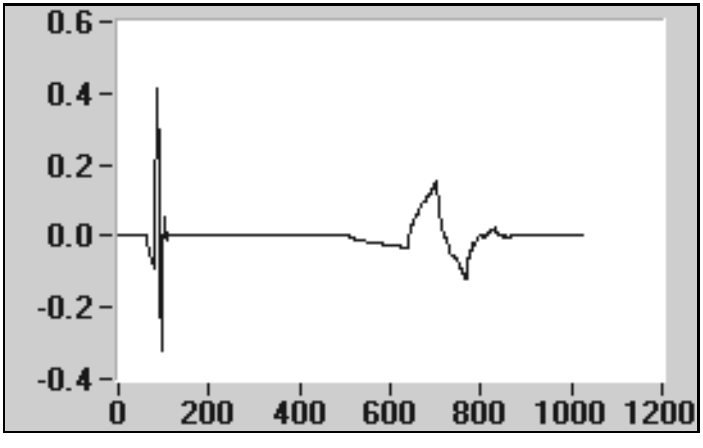
$$c_3 = \frac{1-\sqrt{3}}{4\sqrt{2}}$$

The inverse Wavelet Daubechies4 transform of the array **X** is defined by

$$Wavelet\ Daubechies4\ Inv\{X\} = C^{-1} * X$$

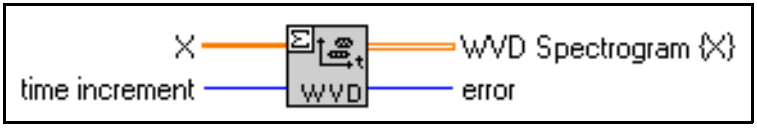
It is $CC^{-1} = C^{-1}C = I$ (Refer to the definition of the Wavelet Transform Daubechies4 VI.)

The following diagram shows the Wavelet Transform Daubechies4 Inverse VI of a function with two spikes at the points 13 and 69. The signal length is 1024.



WVD Spectrogram

Computes the signal energy distribution in the joint time-frequency domain using the Wigner-Ville distribution algorithm.



[DBL] **X** is the time waveform.

I32 **time increment** is the base 2 logarithm of the time spacing, in samples between each row of the WVD Spectrogram {X} output. For example, if you sample the time waveform at *fs* Hz, the spacing between the rows of WVD Spectrogram {X} is **time increment** /*fs* seconds. The default value is 1.

Increasing **time increment** decreases the computation time and reduces memory requirements, but also reduces time-domain resolution. Decreasing **time increment** improves time-domain resolution, but increases the computation time and memory requirements.

[DBL] **WVD Spectrogram {X}** is a 2D array that describes the time waveform energy distribution in the joint time-frequency domain.

I32 **error**. See Appendix A, *Error Codes*, for a list of error codes.

Error Codes

Appendix A

The following tables contain the error codes for the G Math Toolkit.

Table A-1. G Math Toolkit Error Codes

Error Code Number	Error Code Description
0	No error
–23001	Syntax error of parser
–23002	Discrepancy between function, variables and coordinates
–23003	Number of contours out of range
–23004	Number of color palettes out of range
–23005	Negative distance
–23006	Not a valid path
–23007	Not a graphs file
–23008	Wrong input, Euler method
–23009	Wrong input, Runge Kutta method
–23010	Wrong input, Cash Karp method
–23011	Nonpositive step rate
–23012	Nonpositive accuracy
–23013	Matrix vector conflict
–23014	A and X0 have different dimensions
–23015	Empty X0

Table A-1. G Math Toolkit Error Codes

Error Code Number	Error Code Description
–23016	Singular eigenvector matrix
–23017	Multiple roots
–23018	Left point is a root
–23019	Right point is a root
–23020	Left point greater than right point
–23021	Both function values have the same sign
–23022	Nonpositive accuracy or nonpositive delta x(h)
–23023	Wrong dimension of start
–23024	No root found
–23025	Nonvalid triplet (a,b,c)
–23026	No optimum found
–23027	Not exactly one variable
–23028	Wrong model equation
–23029	Levenberg Marquardt has failed
–23030	m >= n >= 0 is violated or the matrix of derivatives has the wrong dimension
–23031	No valid point
–23032	Maximum does not exist
–23033	Vectors have different dimensions or empty vectors
–23034	Ill conditioned system
–23035	Nonpositive number

Table A-1. G Math Toolkit Error Codes

Error Code Number	Error Code Description
–23036	Different parameters
–23037	Not exactly two functions
–23038	No variables in expression
–23039	Parameter problem
–23040	Derivative out of range
–23041	Not exactly two variables
–23042	Negative argument
–23043	Argument out of range (0,1]
–23044	Argument out of range [0,1]
–23045	n<k
–23046	Empty array
–23047	Argument out of range [0,100]
–23048	Invalid time increment
–23049	Invalid window length
–23050	Signal length not a multiple of number
–23051	Signal length not a power of two
–23052	Signal length not a prime and ≥ 5
–23053	Signal length not a power of two and ≥ 4
–23054	Non–unique variables

The following table shows the possible parser error codes for the G Math Toolkit.

Table A-2. G Math Toolkit Parser Error Codes

Error Code Number	Error Code Description	Error Example
0	No error	sin(x)
1	Bracket problem at the beginning	1+x)
2	Incomplete function expression	sin(x)+
3	Bracket problem	()
4	Bracket problem at the end	(1+x
5	Wrong decimal point	1,2 (US)
6	Wrong number format	1e–3 instead of 1E–3
7	Wrong function call	sin()
8	Not a valid function	sins(x)
9	Incomplete expression	x+
10	Wrong variable name	a11
11	Wrong letter	sin(X)
12	Too many decimal points	1.23.45
21	Contains more than one variable	1+x+y4
22	Inconsistency in variables or numbers	Depends on application
23	Contains variables	Depends on application
24	Variables output problem	Depends on application

References

Appendix B

The following references may prove helpful to you as you use the VIs and examples found in the G Math Toolkit.

Baily, David H and Paul N. Swartztrauber. “The Fractional Fourier Transform and Applications.” *Society of Industrial and Applied Mathematics Review*, vol. 33, no. 3, (September 1991): 389–404.

Crandall, R. E. *Projects in Scientific Computation.* Berlin: Springer, 1994.

Ecker, Joseph G. and Michael Kupferschmid. *Introduction to Operations Research.* New York: Krieger Publishing, 1991.

Fahmy, M. F. “Generalized Bessel Polynomials with Application to the Design of Bandpass Filters.” *Circuit Theory and Applications*, vol. 5, 1977: 337-342.

Gander, W. and J. Hrebícek. *Solving Problems in Scientific Computing using Maple and Matlab.* Berlin: Springer, 1993.

Lanczos, C. A. “Precision Approximation of the Gamma Function.” *Journal SIAM Numerical Analysis*, series B, vol. 1, 1964: 87–96.

Mitra, Sanjit K. and James F. Kaiser. *Handbook for Digital Signal Processing.* New York: John Wiley & Sons, 1993.

Oppenheim, Alan V. and Alan S. Willsky. *Signals and Systems.* New York: Prentice-Hall, 1983.

Press, William H., Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C, The Art of Scientific Computing.* Cambridge: Cambridge University Press, 1994.

Qian, Shie and Dapang Chen. *Joint Time-Frequency Analysis.* New York: Prentice Hall, 1996.

Rockey, K. C., H. R. Evans, D. W. Griffiths, D. A. Nethercot. *The Finite Element Method—A Basic Introduction for Engineers*. New York: John Wiley & Sons, 1983.

Wilkinson, J. H. and C. Reinsch. *Linear Algebra, Vol. 2 of Handbook for Automatic Computation*. New York: Springer, 1971.

Williams, John R. and Kevin Amaratunga. "Introduction to Wavelets in Engineering", *International Journal for Numerical Methods in Engineering*, vol. 37, 1994: 2365–2388.

Zwillinger, Daniel. *Handbook of Differential Equations*. San Diego: Academic Press, 1992.

Customer Communication

Appendix C

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve your technical problems and a form you can use to comment on the product documentation. When you contact us, we need the information on the Technical Support Form and the configuration form, if your manual contains one, about your system configuration to answer your questions as quickly as possible.

National Instruments has technical assistance through electronic, fax, and telephone systems to quickly provide the information you need. Our electronic services include a bulletin board service, an FTP site, a Fax-on-Demand system, and e-mail support. If you have a hardware or software problem, first try the electronic support systems. If the information available on these systems does not answer your questions, we offer fax and telephone support through our technical support centers, which are staffed by applications engineers.

Electronic Services

Bulletin Board Support

National Instruments has BBS and FTP sites dedicated for 24-hour support with a collection of files and documents to answer most common customer questions. From these sites, you can also download the latest instrument drivers, updates, and example programs. For recorded instructions on how to use the bulletin board and FTP services and for BBS automated information, call (512) 795-6990. You can access these services at:

United States: (512) 794-5422
Up to 14,400 baud, 8 data bits, 1 stop bit, no parity

United Kingdom: 01635 551422
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

France: 1 48 65 15 59
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

FTP Support

To access our FTP site, log on to our Internet host, `ftp.natinst.com`, as `anonymous` and use your Internet address, such as `joesmith@anywhere.com`, as your password. The support files and documents are located in the `/support` directories.

Fax-on-Demand Support

Fax-on-Demand is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access Fax-on-Demand from a touch-tone telephone at (512) 418-1111.

E-Mail Support (currently U.S. only)

You can submit technical support questions to the appropriate applications engineering team through e-mail at the Internet address listed below. Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.

`support@natinst.com`

Fax and Telephone Support

National Instruments has branch offices all over the world. Use the list below to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.

	Telephone	Fax
Australia	03 9879 9422	03 9879 9179
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Canada (Ontario)	905 785 0085	905 785 0086
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	09 527 2321	09 502 2930
France	01 48 14 24 24	01 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Israel	03 5734815	03 5734816
Italy	02 413091	02 41309215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	5 520 2635	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
U.K.	01635 523545	01635 523154

Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name _____

Company _____

Address _____

_____

Fax (___)_____ Phone (___) _____

Computer brand _____ Model _____ Processor_____

Operating system (include version number) _____

Clock speed _____MHz RAM _____MB_____ Display adapter

Mouse ___yes ___no Other adapters installed _____

Hard disk capacity _____MB _____Brand

Instruments used _____

_____

National Instruments hardware product model _____ Revision

Configuration _____

National Instruments software product _____ Version

Configuration _____

The problem is: _____

_____

_____

_____

_____

List any error messages: _____

_____

_____

The following steps reproduce the problem:_____

_____

_____

_____

_____

_____

G Math Hardware and Software Configuration Form

Record the settings and revisions of your hardware and software on the line to the right of each item. Complete a new copy of this form each time you revise your software or hardware configuration, and use this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

National Instruments Products

DAQ hardware _____

Interrupt level of hardware _____

DMA channels of hardware _____

Base I/O address of hardware _____

Programming choice _____

HiQ, NI-DAQ, LabVIEW, or BridgeVIEW version _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

Other Products

Computer make and model _____

Microprocessor _____

Clock frequency or speed _____

Type of video board installed _____

Operating system version _____

Operating system mode _____

Programming language _____

Programming language version _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: *G Math Toolkit Reference Manual*

Edition Date: November 1996

Part Number: 321290A-01

Please comment on the completeness, clarity, and organization of the manual.

_____

_____

_____

_____

_____

_____

_____

If you find errors in the manual, please record the page numbers and describe the errors.

_____

_____

_____

_____

_____

_____

_____

Thank you for your help.

Name _____

Title _____

Company _____

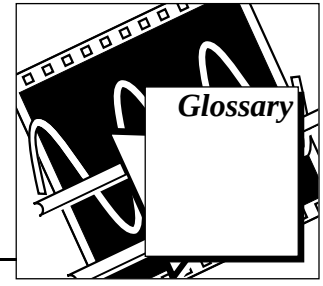
Address _____

_____

Phone (___)_____ Fax (___) _____

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway
Austin, TX 78730-5039

Fax to: Technical Publications
National Instruments Corporation
(512) 794-5678



Prefix	Meaning	Value
m-	milli-	10^{-3}
µ-	micro-	10^{-6}
n-	nano-	10^{-9}

Numbers/Symbols

°	degrees
ε	epsilon
%	percent
π	pi
1D	One-dimensional
2D	Two-dimensional
3D	Three-dimensional
nD	*n*-dimensional
f' or $\dot{f}$	derivative of f

A

array	Ordered, indexed set of data elements of the same type.
ASCII	American Standard Code for Information Interchange.

B

Bessel filters

These filters have a maximally flat response in both magnitude and phase. The phase response in the passband, which is usually the region of interest, is nearly linear. Bessel filters can be used to reduce nonlinear phase distortion inherent in all IIR filters.

Bessel function

The Bessel function of the first kind of order n, $J_n(x)$, is defined by

$$J_n(x) = \left(\frac{1}{2}x\right)^n \sum_{k=0}^{\infty} \frac{\left(\frac{-1}{4}x^2\right)^k}{k!\,\Gamma(n+k+1)}$$

with $n = 0, 1, \ldots$ The Bessel function of the second kind of order n, $Y_n(x)$, is defined by

$$Y_n(x) = \frac{J_n(x)\cos(n\pi) - J_{-n}(x)}{\sin(n\pi)} \text{ with } n = 0, 1, \ldots$$

Bessel polynomial

The Bessel polynomial $P_n(x)$ of order n is defined by a recurrence relation

$$P_n(x) = P_{n-1}(x) + \frac{x^2}{4(n-1)^2 - 1} P_{n-2}(x) \text{ for } n = 2, 3, \ldots$$

where $P_0(x) = 1$ and $P_1(x) = 1 + x$.

beta function

An integral defined by

$$B(w, z) = \int_0^1 t^{w-1}(1-t)^{z-1}\,dt \qquad (w > 0, z > 0)$$

or, in terms of the gamma function

$$B(w,z) = \frac{\Gamma(w)\Gamma(z)}{\Gamma(w+z)}$$

binomial coefficient

The binomial coefficient is given by

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

block diagram

Pictorial representation of a program or algorithm. In G, the block diagram, which consists of executable icons called nodes and wires that carry data between the nodes, is the source code for the VI. The block diagram resides in the block diagram window of your G development environment.

bracketing interval

A bracketing triplet (a, b, c) of a continuous 1D function *f* is a combination of three points with $f(a) > f(b)$ and $f(c) > f(b)$. This guarantees the existence of a local minimum of *f* in the interval (a, c).

bracketing of a minimum

A root of a continuous function is said to be bracketed by a pair of points, $a < b$, when $f(a)$ and $f(b)$ are of opposite signs. A minimum is bracketed when there are three points, $a < b < c$, with $f(a) > f(b)$ and $f(c) > f(b)$.

Brent Method

For solving nonlinear equations, methods that converge rapidly to the solution are unreliable because convergence may not occur unless started close enough to the actual solution. On the other hand, the more reliable methods are slower. The Brent method is a hybrid method that combines both the safety of the bisection method and the rapid convergence of inverse quadratic interpolation.

C

Cash Karp method

A numerical method for solving ordinary differential equations with start conditions. The Cash Karp method is an embedded Runge Kutta formula and is based on a fifth order strategy (with six steps). The Cash Karp method works with an adaptive step rate and is computationally more efficient than the Euler method or the Runge Kutta method.

chart

A 2D display of one or more plots, in which the display retains previous data, up to a maximum which you can define. The chart receives the data and updates the display point by point or array by array, retaining a certain number of past points in a buffer for display purposes.

Chebyshev polynomial

The Chebyshev polynomial, for real numbers x, is given by

$$T_n(x) = \cos(n \arccos(x))$$

This results in

$$T_0 = 1,\ T_1(x) = x,\ T_2(x) = 2x^2 - 1,\ T_3(x) = 4x^3 - 3x, \text{ and so on.}$$

chi-squared

A penalty function given by

$$\chi^2 = \sum_{i=0}^{N-1} \left(\frac{y_i - f(x_i;\ (a_1, \ldots, a_M))}{\sigma_i} \right)^2$$

In this equation, (x_i, y_i) are the input data points, and $f(x_i;\ a_1,\ldots,\ a_M) = f(X,A)$ is the nonlinear function where $a_1,\ldots,a_M$ are coefficients. If the measurement errors are independent and normally distributed with constant, standard deviation $\sigma_i = \sigma$, this is also the least-square estimation.

The input arrays **X** and **Y** define the set of input data points. It is assumed that one has prior knowledge of the nonlinear relationship between the x and y coordinates. That is, $f = f(X,A)$ where the set of coefficients, A, is determined by the Levenberg-Marquardt algorithm.

common intensity map/plot

A method of displaying 3 dimensions of data on a 2D plot with the use of color.

conjugate gradient method

This method can be used for multidimensional unconstrained minimization. It determines the local minimum of a function of n independent variables. The direction of the gradient is modified at each iteration. This is done by forming a sequence of conjugate (the inner product being orthogonal) search directions and tends to avoid slow convergence which may result by repeated

searching in the same direction. To minimize the function $f(x)$, given an initial guess x_0, we first calculate $g_0 = \nabla(f(x_0))$. The rest of the algorithm consists of following a sequence of steps until convergence. One of these steps consists of solving for a quantity β_{k+1}, where

$$\beta_{k+1} = (g_{k+1}^T g_{k+1})/(g_k^T g_k)$$

(Fletcher & Reeves)

or

$$\beta_{k+1} = ((g_{k+1} - g_k)^T g_{k+1})/(g_k^T g_k)$$

(Polak & Ribiere).

continued fraction

The continued fraction of two sequences $(a_0, a_1, \ldots, a_n)$ and $(b_0, b_1, \ldots, b_n)$ is defined by the following term.

$$\text{result} = \cfrac{a_0}{b_0 + \cfrac{a_1}{b_1 + \ldots}}$$

Continued fractions are valuable tools for calculating special functions.

contour plot

A plot where contour lines are used to connect points of equal value.

control

Front panel object for entering data to a VI interactively or to a subVI programmatically.

cosine integral

The cosine integral is defined by:

$$\mathrm{ci}(x) = \gamma + \ln x + \int_0^x \frac{\cos s - 1}{s} ds$$

with the Euler constant γ and x as any real non-negative number.

curve in 3D

A special parametric plot $(x(t), y(t), z(t))$, where the parameter t runs over a given interval.

D

data flow	Programming system consisting of executable nodes in which nodes execute only when they have received all required input data and produce output automatically when they have executed. G applications are dataflow systems.
Daubechies4 Function	A Daubechies wavelet with 4 coefficients.
dimension	Size and structure attribute of an array.
discrete	Having discontinuous values of the independent variable, usually time.
Downhill Simplex Method	Determines a local minimum of a function of n independent variables. The Downhill Simplex algorithm, also called the Nelder and Mead method, works without partial derivatives. The algorithm consists of catching the minimum of the function, $f(X)$, with the help of simple geometrical bodies, namely with a simplex. A simplex in 2D is a triangle, a simplex in 3D is a tetrahedron and so on. You must have $n + 1$ starting points, each of dimension n, forming the initial simplex. The user must enter only one point of these $(n+1)$. The $(n+1)$ dimensional simplex is automatically constructed. The algorithm generates a new simplex by some elementary operations like reflections, expansions, and contractions. In the end, the minimum is concentrated in a very small simplex.

E

eigenvalues	Values of λ for which the matrix equation $Ax = \lambda x$ has a nontrivial solution $(x \neq 0)$ are known as eigenvalues or characteristic values.
eigenvectors	The solutions of x for which the matrix equation $Ax = \lambda x$ has a nontrivial solution $(x \neq 0)$ are known as eigenvectors or characteristic vectors.
error in	The error structure entering a VI.
error out	The error structure leaving a VI.
error structure	The LabVIEW error structure consists of a Boolean status indicator, a numeric code indicator, and a string source indicator.

Euler method — A numerical method for solving ordinary differential equations with start conditions. This is a single-step method because it depends on information at only one point in time to advance to the next point.

extrema — Maxima and minima.

F

FFT — Fast Fourier Transform.

FFFT — Fractional Fast Fourier Transform.

Fletcher & Reeves — *See* Conjugate Gradient Method.

formula node — Node that executes formulas that you enter as text. This node is especially useful for lengthy formulas that would be cumbersome to build in block diagram form.

front panel — The interactive user interface of a VI. Modeled from the front panel of physical instruments, it is composed of switches, slides, meters, graphs, charts, gauges, LEDs, and other controls and indicators.

G

G — The LabVIEW graphical programming language.

gamma function — The gamma function of x is the generalization of the common factorial function $n!$. The relation between these two functions is $\Gamma(n+1) = n!$ for all natural numbers n. The gamma function is defined by

$$\Gamma(x) = \int_0^x s^{x-1} \exp(-s)\,ds$$

for real and complex x, and has the property $\Gamma(x+1) = x\Gamma(x)$.

golden section search

Determines a local minimum of a given 1D function with the help of a bracketing of the minimum. Consider a real valued 1D function $f(x)$, unimodal on the interval (a,c). The Golden Section Search method determines, beginning with a bracketing triplet (a,b,c), a new one with a considerably smaller expansion. Repeating this scheme often yields a good approximation of the local minimum. The new bracketing point, x, is essentially calculated by the following equation.

$$\left|\frac{x-b}{c-a}\right| = (\sqrt{5}-2)$$

graph

A 2D display of one or more plots. A graph receives and plots data as a block.

H

Hz

Hertz. Cycles per second.

I

incomplete beta function

The incomplete beta function is defined by:

$$I_x(a,b) = \frac{1}{B(a,b)}\int_0^x s^{a-1}(1-s)^{b-1}\mathrm{d}s$$

with $a, b > 0$ and $0 \le x \le 1$

where $B(a,b)$ denotes the beta function of a and b.

incomplete gamma function

The incomplete gamma function is defined by

$$P(a,x) = \frac{1}{\Gamma(a)}\int_0^x \exp(-s)s^{a-1}\mathrm{d}s$$

for $a > 0$.

indicator

Front panel object that displays output.

Inf

Digital display value for a floating-point representation of infinity.

J

Jacobian elliptic function

The value of *sn* (Jacobian Elliptic Function) is determined by the relation

$$u = \int_0^{sn} \frac{ds}{\sqrt{(1-s^2)(1-k^2s^2)}}$$

where *u* and *k* are given real numbers. The other Jacobian Elliptic functions have the following definitions

$$sn^2 + cn^2 = 1,\ k^2sn^2 + dn^2 = 1,\ sc = \frac{sn}{cn}$$

L

LabVIEW

Laboratory Virtual Instrument Engineering Workbench.

Legendre elliptic integral

The Legendre Elliptic Integral of the 1st kind is defined by the following equation.

$$F(\varphi, k) = \int_0^{\varphi} \frac{d\vartheta}{\sqrt{1-k^2\sin^2\vartheta}}$$

library

See VI Library.

linear programming

Suppose that we are given *f*, a linear function of the variables *x1, x2, x3,.....xn*, and a set of constraints on these variables in terms of linear inequalities. Linear Programming consists of methods for maximizing or minimizing *f*. Such problems are usually found in the areas of economics, distribution of goods, production, and approximation theory.

M

matrix — Two-dimensional array.

maxima — The maximum value(s) of a function.

minima — The minimum value(s) of a function.

N

NaN — Digital display value for a floating-point representation of not a number, typically the result of an undefined operation, such as $\frac{0}{0}$.

Newton Raphson method — Determines a zero of a 1D function close to two points with the help of the derivative of this 1D function. The two values form a search limit for the unknown zero of the 1D function. The Newton Raphson method (or Newton method) is an iterative method for solving equations of the form $f(x) = 0$ where the derivative of f, f', is continuous. Given two values, $x1$ and $x2$, with $f(x1)*f(x2) < 0$, first use the midpoint method to calculate

$$x(0) = (x1 + x2)/2 \quad (1)$$

Then use the Newton method to calculate

$$x(n+1) = x(n) - f(x(n))/f'(x(n)) \quad (2)$$

Replace $x(n)$ by $x(n+1)$ and repeat equation (2) till a certain termination condition is met. This termination condition could be either after a given number of repetitions, or if

$$|x(n+1) - x(n)| \leq \varepsilon$$

This is a fast and simple method.

nonsingular matrix — Matrix in which no row or column is a linear combination of the other rows or columns, respectively. In other words, the rows or columns are linearly independent. This matrix has a unique inverse, and it has a nonzero determinant.

O

ODE — Ordinary Differential Equation.

optimal step — For a function that has many maxima, minima, or singularities, the standard method (using equidistant points) may yield incorrect results and in such cases the optimal step method is much better. This method starts off with taking equidistant points, but also checks the steepness of the curve between these points. In very steep portions (determined by the value of epsilon on the front panel) of the graph, a new point is generated in between. As a rule, the smaller the value of epsilon, the more the points that are generated, and the better the graph.

P

Pade approximation — Determines the coefficients of a rational polynomial to best suit a given set of first derivatives. Let f be a given function with known values

$$f(0), f'(0), \ldots, f^{(n+m)}(0)$$

There exists a unique rational polynomial $(m \geq n)$

$$R(x) = \frac{a_0 + a_1 x + \ldots + a_m x^m}{1 + b_1 x + \ldots + b_n x^n}$$

with

$$R(0) = f(0), R'(0) = f'(0), \ldots, R^{(m+n)}(0) = f^{(m+n)}(0) .$$

The rational polynomial can be determined by solving a special linear equation.

parametric plot — A plot of the variables (such as x and y) which are given in terms of another variable t (called a parameter), resulting in the parametric equations $x = f(t)$ and $y = g(t)$. Each value of t determines a point (x,y). As t is varied, the point $(x,y) = (f(t), g(t))$ varies and traces a curve, that is plotted.

parser — A VI used to scan a string to determine the function of each of the elements in the string.

partial derivative — A derivative taken with respect to one of two or more independent variables, with the others being treated as constants.

platform — Computer and operating system.

plot — A graphical representation of an array of data shown either on a graph or a chart.

Polak & Ribiere — *See* Conjugate Gradient Method.

power spectrum — The magnitude of the FFFT.
fractional FFT

prime FFT — A special case of the FFT where the signal length is prime. The prime FFT VI is computationally more efficient than the classical FFT algorithms.

R

rational polynomial — A quotient of polynomials. The advantage that they have over polynomials for approximating functions is that they can be used to model functions with poles.

real Laplace transform — The Laplace transform for real *s*.

Ridders Method — Determines a zero of a 1D function in a given interval. The function has to be continuous and has to have different signs at the end points of the interval.

Let us be given the function $f(x)$ with

$$f(a)*f(b) < 0 .$$

Ridders method determines

$$c = (a + b)/2$$

and calculates the new guess

$$c_{\text{new}} = c + (c - a)\frac{\text{sign}(f(a) - f(b))f(c)}{\sqrt{f(c)^2 - f(a)f(b)}}$$

The triplets start, c_{new}, end are the base for the new iteration, depending on whether

$$f(\text{start}) * f(c_{\text{new}}) < 0$$

or

$$f(c_{\text{new}}) * f(\text{end}) < 0 .$$

The algorithm stops, if

$$|a - b| < \text{accuracy} .$$

Ridders' method is fast and reliable.

roots — Zeros of a function, i.e. values of the variable(s) for which the function is equal to zero.

Runge Kutta method — A numerical method for solving ordinary differential equations with start conditions. The Runge Kutta method works with a fixed step rate but with a higher degree of accuracy than the common Euler method.

S

scalar — Number capable of being represented by a point on a scale. A single value as opposed to an array. Scalar Booleans and clusters are explicitly singular instances of their respective data types.

simplex — A simple geometrical body. A simplex in 2D is a triangle, a simplex in 3D is a tetrahedron and so on. Used in finding the minimum of a function. See the Downhill Simplex nD VI for more information.

sine integral

The sine integral is defined by

$$\mathrm{si}(x) = \int_0^x \frac{\sin s}{s} \mathrm{d}s$$

We have that $\mathrm{si}(-x) = -\mathrm{si}(x)$ where x is any real number.

singular matrix

a square matrix in which a row or column is a linear combination of the other rows or columns, respectively. i.e. the rows or columns are linearly dependent. This matrix does not have an inverse, and its determinant is equal to zero.

sparse signals

Signals with a large number of zero values.

spectrogram

A particular representation of a signal that describes the distribution of the energy of the time waveform in the joint time-frequency domain.

spike function

The spike function is defined by

$$\mathrm{spike}(x) = \begin{cases} 1 & \text{if} \quad 0 \le x < 1 \\ 0 & \text{else} \end{cases}$$

square function

The square function is defined as

$$\mathrm{square}(x) = \begin{cases} 1 & \text{if} \quad 2n \le x < 2n + 1 \quad n = \ldots, -1, 0, 1, \ldots \\ 0 & \text{if} \quad 2n + 1 \le x < 2n + 2 \quad n = \ldots, -1, 0, 1, \ldots \end{cases}$$

where x is any real number.

step function

The step function is defined by

$$\mathrm{step}(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{else} \end{cases}$$

where x is any real number.

STFT

Short-Time Fourier Transform.

string

Representation of a value as text.

substitution of variables — A method (commonly used in calculus) for finding solutions to equations, where a variable is substituted in terms of another variable in order to make the equation tractable (solvable). It is also a good method to simplify complex terms.

subVI — VI used in the block diagram of another VI; comparable to a subroutine.

T

ticks — Time in milliseconds required for the entire calculation.

top-level VI — VI at the top of the VI hierarchy. This term distinguishes the VI from its subVIs.

U

unevenly sampled data — Data that has been sampled with non-equal sampling intervals.

V

vector — One-dimensional array.

VI Library — Special file that contains a collection of related VIs for a specific use. The extension of the file is `.llb`.

virtual instrument (VI) — LabVIEW program; so called because it models the appearance and function of a physical instrument.

W

Walsh Hadamard transform — A transform based on an orthogonal system consisting of functions of only two elements –1 and 1. Its properties are similar to the more well known Fourier transform, but the computational effort is considerably smaller.

wavelet transform — A transform used to represent a signal in the time-scale domain.

WVD — Wigner-Ville distribution. A form of time-frequency distribution used to represent a signal in the time-frequency domain.

X

X0 A vector of start (initial) conditions.

Z

zeros *See* roots.