This guidebook applies to TI-Nspire™ software version 3.6. To obtain the latest version of the documentation, go to education.ti.com/guides.
Important Information

Except as otherwise expressly stated in the License that accompanies a program, Texas Instruments makes no warranty, either express or implied, including but not limited to any implied warranties of merchantability and fitness for a particular purpose, regarding any programs or book materials and makes such materials available solely on an "as-is" basis. In no event shall Texas Instruments be liable to anyone for special, collateral, incidental, or consequential damages in connection with or arising out of the purchase or use of these materials, and the sole and exclusive liability of Texas Instruments, regardless of the form of action, shall not exceed the amount set forth in the license for the program. Moreover, Texas Instruments shall not be liable for any claim of any kind whatsoever against the use of these materials by any other party.

License

Please see the complete license installed in

C:\Program Files\TI Education\<TI-Nspire™ Product Name>\license.

© 2006 - 2013 Texas Instruments Incorporated
## Contents

Important Information ......................................................... 2

Expression Templates .......................................................... 5

**Alphabetical Listing** ....................................................... 11
   A ........................................................................ 11
   B ........................................................................ 19
   C ........................................................................ 22
   D ........................................................................ 37
   E ........................................................................ 43
   F ........................................................................ 49
   G ........................................................................ 56
   I ........................................................................ 61
   L ........................................................................ 67
   M ........................................................................ 80
   N ........................................................................ 88
   O ........................................................................ 95
   P ........................................................................ 98
   Q ...................................................................... 104
   R ...................................................................... 106
   S ...................................................................... 119
   T ...................................................................... 136
   U ...................................................................... 146
   V ...................................................................... 147
   W ...................................................................... 148
   X ...................................................................... 149
   Z ...................................................................... 150

Symbols .................................................................................. 156

Empty (Void) Elements ......................................................... 177

Shortcuts for Entering Math Expressions ............................. 179

EOS™ (Equation Operating System) Hierarchy .................. 181

Error Codes and Messages .................................................... 183
Expression Templates

Expression templates give you an easy way to enter math expressions in standard mathematical notation. When you insert a template, it appears on the entry line with small blocks at positions where you can enter elements. A cursor shows which element you can enter.

Use the arrow keys or press tab to move the cursor to each element’s position, and type a value or expression for the element. Press enter or ctrl enter to evaluate the expression.

### Fraction template

Example:

\[
\frac{12}{8.2}
\]

Note: See also / (divide), page 158.

### Exponent template

Example:

\[2^3 = 8\]

Note: Type the first value, press ^, and then type the exponent. To return the cursor to the baseline, press right arrow (→).

Note: See also ^ (power), page 158.

### Square root template

Example:

\[
\sqrt{4} = 2 \quad \sqrt{\{9,a,4\}} = \{3, \sqrt[3]{a}, 2\} = \frac{2}{\sqrt{4} \sqrt[3]{\{9,16,4\}}} = \{3,4,2\}
\]

Note: See also √ (square root), page 167.
Nth root template

\[ \sqrt[n]{a} \]

Note: See also \( \text{root}() \), page 116.

Example:

\[
\begin{align*}
3\sqrt[3]{8} & = 2 \\
3\sqrt[3]{8,37,15} & = \{2,3,2.46621\}
\end{align*}
\]

Exponent template

\[ e^x \]

Natural exponential \( e \) raised to a power

Note: See also \( e^x() \), page 43.

Example:

\[
\begin{align*}
e^1 & = 2.71828182846
\end{align*}
\]

Log template

\[ \log_b(x) \]

Calculates log to a specified base. For a default of base 10, omit the base.

Note: See also \( \log() \), page 76.

Example:

\[
\begin{align*}
\log_4(2) & = 0.5
\end{align*}
\]

Piecewise template (2-piece)

\[ \begin{cases} 
  f(x) & \text{if condition} \\
  g(x) & \text{if other condition}
\end{cases} \]

 Lets you create expressions and conditions for a two-piece piecewise function. To add a piece, click in the template and repeat the template.

Note: See also \( \text{piecewise}() \), page 99.

Example:

\[ f(x) = \begin{cases} 
  x+1, & x > 1 \\
  \text{undef}, & x \leq 1
\end{cases} \]

Piecewise template (N-piece)

\[ \psi \]

Lets you create expressions and conditions for an \( N \)-piece

Example:
**Piecewise template (N-piece)**


**Create Piecewise Function**

Piecewise Function
Number of function pieces ?

OK Cancel

**Note:** See also `piecewise()`, page 99.

**System of 2 equations template**

\[
\begin{align*}
\text{System of Equations} \\
\text{Number of equations} & \quad 3 \quad \text{OK Cancel}
\end{align*}
\]

Creates a system of two linear equations. To add a row to an existing system, click in the template and repeat the template.

**Note:** See also `system()`, page 135.

**System of N equations template**

Lets you create a system of $N$ linear equations. Prompts for $N$.

**Note:** See also `system()`, page 135.

**Absolute value template**

**Note:** See also `abs()`, page 11.

**Example:**

\[
\begin{align*}
\text{Absolute value template} \\
\text{Note: See also abs()}, \text{ page 11.}
\end{align*}
\]
dd°mm's's'ss'' template

Example:

\[
30°15'10''
\]

\[
0.528011
\]

Lets you enter angles in \(dd°mm's's'ss''\) format, where \(dd\) is the number of decimal degrees, \(mm\) is the number of minutes, and \(ss.ss\) is the number of seconds.

Matrix template (2 x 2)

Example:

\[
\begin{bmatrix}
1 & 2.5 \\
3 & 4
\end{bmatrix}
\]

\[
\begin{bmatrix}
5 & 10 \\
15 & 20
\end{bmatrix}
\]

Creates a 2 x 2 matrix.

Matrix template (1 x 2)

Example:

\[
\text{crossP} \begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & 0 & 2
\end{bmatrix}
\]

Matrix template (2 x 1)

Example:

\[
\begin{bmatrix}
5 \\
8
\end{bmatrix}
\]

\[
\begin{bmatrix}
0.01 \\
0.05 & 0.08
\end{bmatrix}
\]

Matrix template (m x n)

The template appears after you are prompted to specify the number of rows and columns.

Example:

\[
\begin{bmatrix}
4 & 2 & 6 \\
1 & 2 & 3 \\
5 & 7 & 9
\end{bmatrix}
\]

\[
\begin{bmatrix}
4 & 2 & 9
\end{bmatrix}
\]
Matrix template \((m \times n)\)  

**Note:** If you create a matrix with a large number of rows and columns, it may take a few moments to appear.

---

Sum template \(\sum\)  

\[
\sum_{n=3}^{7} (n)
\]

**Note:** See also \(\sum\) (sumSeq), page 168.

---

Product template \(\Pi\)  

\[
\Pi_{n=1}^{5} \left( \frac{1}{n} \right)
\]

**Note:** See also \(\Pi\) (prodSeq), page 167.

---

First derivative template  

\[
\frac{d}{dx}(\text{expression})
\]

The first derivative template can be used to calculate first derivative at a point numerically, using auto differentiation methods.

**Note:** See also \(d()\) (derivative), page 166.

---

Second derivative template  

\[
\frac{d^2}{d[x]^2}(\text{expression})
\]

---

Expression Templates 9
**Second derivative template**

The second derivative template can be used to calculate second derivative at a point numerically, using auto differentiation methods.

*Note:* See also `d()` (derivative), page 166.

\[
\frac{d^2}{dx^2}(x^3)_{x=3}
\]

**Definite integral template**

The definite integral template can be used to calculate the definite integral numerically, using the same method as `nInt()`.

*Note:* See also `nInt()`, page 91.

Example:

\[
\int_0^{10} x^2 \, dx
\]
Alphabetical Listing

Items whose names are not alphabetic (such as +, !, and >) are listed at the end of this section, page 156. Unless otherwise specified, all examples in this section were performed in the default reset mode, and all variables are assumed to be undefined.

A

\[
\text{abs}(\text{Value1}) \Rightarrow \text{value} \\
\text{abs}(\text{List1}) \Rightarrow \text{list} \\
\text{abs}(\text{Matrix1}) \Rightarrow \text{matrix}
\]

Returns the absolute value of the argument.

Note: See also Absolute value template, page 7.

If the argument is a complex number, returns the number's modulus.

\[
\text{amortTbl}(\text{NPmt}, N, I, PV, \{Pmt\}, \{FV\}, \{PpY\}, \{CpY\}, \{PmtAt\}, \{\text{roundValue}\}) \Rightarrow \text{matrix}
\]

Amortization function that returns a matrix as an amortization table for a set of TVM arguments.

\(\text{NPmt}\) is the number of payments to be included in the table. The table starts with the first payment.

\(N, I, PV, Pmt, FV, PpY, CpY\), and \(PmtAt\) are described in the table of TVM arguments, page 144.

- If you omit \(Pmt\), it defaults to \(Pmt=\text{tvmPmt}(N, I, PV, FV, PpY, CpY, PmtAt)\).
- If you omit \(FV\), it defaults to \(FV=0\).
- The defaults for \(PpY, CpY\), and \(PmtAt\) are the same as for the TVM functions.

\(\text{roundValue}\) specifies the number of decimal places for rounding. Default=2.

The columns in the result matrix are in this order:
Payment number, amount paid to interest, amount
amortTbl() paid to principal, and balance.

The balance displayed in row \( n \) is the balance after payment \( n \).

You can use the output matrix as input for the other amortization functions \( \Sigma \text{Int()} \) and \( \Sigma \text{Prn()} \), page 168, and \( \text{bal()} \), page 19.

BooleanExpr1 and BooleanExpr2 \( \Rightarrow \) Boolean expression

BooleanList1 and BooleanList2 \( \Rightarrow \) Boolean list

BooleanMatrix1 and BooleanMatrix2 \( \Rightarrow \) Boolean matrix

Returns true or false or a simplified form of the original entry.

Integer1 and Integer2 \( \Rightarrow \) integer

Compares two real integers bit-by-bit using an and operation. Internally, both integers are converted to signed, 64-bit binary numbers. When corresponding bits are compared, the result is 1 if both bits are 1; otherwise, the result is 0. The returned value represents the bit results, and is displayed according to the Base mode.

You can enter the integers in any number base. For a binary or hexadecimal entry, you must use the 0b or 0h prefix, respectively. Without a prefix, integers are treated as decimal (base 10).

In Hex base mode:

\[
\begin{align*}
0h7AC36 & \quad 0h3D5F & \quad 0h2C16 \\
\end{align*}
\]

Important: Zero, not the letter O.

In Bin base mode:

\[
\begin{align*}
0b100101 & \quad 0b100 & \quad 0b100 \\
\end{align*}
\]

In Dec base mode:

\[
\begin{align*}
37 & \quad 0b100 & \quad 4 \\
\end{align*}
\]

Note: A binary entry can have up to 64 digits (not counting the 0b prefix). A hexadecimal entry can have up to 16 digits.

angle()

angle(\( Value \)) \( \Rightarrow \) value

Returns the angle of the argument, interpreting the argument as a complex number.

In Degree angle mode:

\[
\begin{align*}
\text{angle}(0 + 2 \cdot i) & \quad 90 \\
\end{align*}
\]
angle()\[\text{Catalog}>\]

In Gradian angle mode:

\[
\text{angle}(0+3\cdot i) = 100
\]

In Radian angle mode:

\[
\begin{align*}
\text{angle}(1+i) & = 0.785398 \\
\text{angle}\left(1+2\cdot i, 3+0\cdot i, 0-4\cdot i\right) & = \{1.10715, 0., -1.5708\} \\
\text{angle}\left(1+2\cdot i, 3+0\cdot i, 0-4\cdot i\right) & = \left\{\frac{\pi}{2}, -\tan^{-1}\left(\frac{1}{2}\right), 0, -\frac{\pi}{2}\right\}
\end{align*}
\]

\text{angle}(List1) \Rightarrow list

\text{angle}(Matrix1) \Rightarrow matrix

Returns a list or matrix of angles of the elements in List1 or Matrix1, interpreting each element as a complex number that represents a two-dimensional rectangular coordinate point.

\text{ANOVA} \[\text{Catalog}>\]

\text{ANOVA} List1, List2[, List3, ..., List20][, Flag]

Performs a one-way analysis of variance for comparing the means of two to 20 populations. A summary of results is stored in the \text{stat.results} variable. (page 131)

\text{Flag}=0 \text{ for Data, Flag}=1 \text{ for Stats}

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{stat.F}</td>
<td>Value of the F statistic</td>
</tr>
<tr>
<td>\text{stat.PVal}</td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>\text{stat.df}</td>
<td>Degrees of freedom of the groups</td>
</tr>
<tr>
<td>\text{stat.SS}</td>
<td>Sum of squares of the groups</td>
</tr>
<tr>
<td>\text{stat.MS}</td>
<td>Mean squares for the groups</td>
</tr>
<tr>
<td>\text{stat.dfError}</td>
<td>Degrees of freedom of the errors</td>
</tr>
<tr>
<td>\text{stat.SSError}</td>
<td>Sum of squares of the errors</td>
</tr>
<tr>
<td>\text{stat.MSError}</td>
<td>Mean square for the errors</td>
</tr>
<tr>
<td>\text{stat.sp}</td>
<td>Pooled standard deviation</td>
</tr>
</tbody>
</table>
### ANOVA2way

**ANOVA2way** \( List1, List2[, List3, ..., List10][, levRow] \)

Computes a two-way analysis of variance for comparing the means of two to 10 populations. A summary of results is stored in the `stat.results` variable. (See page 131.)

\( LevRow=0 \) for Block

\( LevRow=2, 3, ..., Len-1, \) for Two Factor, where \( Len = \text{length}(List1) = \text{length}(List2) = ... = \text{length}(List10) \) and \( Len / LevRow \equiv \{2, 3, ..., Len-1\} \)

Outputs: Block Design

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.F</td>
<td>F statistic of the column factor</td>
</tr>
<tr>
<td>stat.PVal</td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>stat.df</td>
<td>Degrees of freedom of the column factor</td>
</tr>
<tr>
<td>stat.SS</td>
<td>Sum of squares of the column factor</td>
</tr>
<tr>
<td>stat.MS</td>
<td>Mean squares for column factor</td>
</tr>
<tr>
<td>stat.FBlock</td>
<td>F statistic for factor</td>
</tr>
<tr>
<td>stat.PValBlock</td>
<td>Least probability at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>stat.dfBlock</td>
<td>Degrees of freedom for factor</td>
</tr>
<tr>
<td>stat.SSBlock</td>
<td>Sum of squares for factor</td>
</tr>
<tr>
<td>stat.MSBlock</td>
<td>Mean squares for factor</td>
</tr>
<tr>
<td>stat.dfError</td>
<td>Degrees of freedom of the errors</td>
</tr>
<tr>
<td>stat.SSErr</td>
<td>Sum of squares of the errors</td>
</tr>
<tr>
<td>stat.MSErr</td>
<td>Mean squares for the errors</td>
</tr>
<tr>
<td>stat.s</td>
<td>Standard deviation of the error</td>
</tr>
</tbody>
</table>

### COLUMN FACTOR Outputs
<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.Fcol</td>
<td>F statistic of the column factor</td>
</tr>
<tr>
<td>stat.PValCol</td>
<td>Probability value of the column factor</td>
</tr>
<tr>
<td>stat.dfCol</td>
<td>Degrees of freedom of the column factor</td>
</tr>
<tr>
<td>stat.SSCol</td>
<td>Sum of squares of the column factor</td>
</tr>
<tr>
<td>stat.MSCol</td>
<td>Mean squares for column factor</td>
</tr>
</tbody>
</table>

ROW FACTOR Outputs

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.FRow</td>
<td>F statistic of the row factor</td>
</tr>
<tr>
<td>stat.PValRow</td>
<td>Probability value of the row factor</td>
</tr>
<tr>
<td>stat.dfRow</td>
<td>Degrees of freedom of the row factor</td>
</tr>
<tr>
<td>stat.SSRow</td>
<td>Sum of squares of the row factor</td>
</tr>
<tr>
<td>stat.MSRow</td>
<td>Mean squares for row factor</td>
</tr>
</tbody>
</table>

INTERACTION Outputs

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.FInteract</td>
<td>F statistic of the interaction</td>
</tr>
<tr>
<td>stat.PValInteract</td>
<td>Probability value of the interaction</td>
</tr>
<tr>
<td>stat.dfInteract</td>
<td>Degrees of freedom of the interaction</td>
</tr>
<tr>
<td>stat.SSInteract</td>
<td>Sum of squares of the interaction</td>
</tr>
<tr>
<td>stat.MSInteract</td>
<td>Mean squares for interaction</td>
</tr>
</tbody>
</table>

ERROR Outputs

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.dfError</td>
<td>Degrees of freedom of the errors</td>
</tr>
<tr>
<td>stat.SSError</td>
<td>Sum of squares of the errors</td>
</tr>
<tr>
<td>stat.MSError</td>
<td>Mean squares for the errors</td>
</tr>
<tr>
<td>s</td>
<td>Standard deviation of the error</td>
</tr>
</tbody>
</table>
Ans

\[ \text{Ans} \Rightarrow \text{value} \]

Returns the result of the most recently evaluated expression.

\[ \begin{array}{l l}
56 & 56 \\
56 + 4 & 60 \\
60 + 4 & 64 \\
\end{array} \]

approx()

\[ \text{approx}(\text{Value1}) \Rightarrow \text{number} \]

Returns the evaluation of the argument as an expression containing decimal values, when possible, regardless of the current Auto or Approximate mode.

This is equivalent to entering the argument and pressing [ctrl] [enter].

\[ \begin{array}{l l}
\text{approx}\left(\frac{1}{3}\right) & 0.333333 \\
\text{approx}\left(\frac{1 + 1}{3 \cdot 9}\right) & \{0.333333, 0.111111\} \\
\text{approx}\left(\{\sin(\pi), \cos(\pi)\}\right) & \{0., -1.\} \\
\text{approx}\left(\sqrt{2} \sqrt{3}\right) & \begin{bmatrix} 1.41421 & 1.73205 \end{bmatrix} \\
\text{approx}\left(\frac{1}{3} \frac{1}{9}\right) & \begin{bmatrix} 0.333333 & 0.111111 \end{bmatrix} \\
\text{approx}\left(\{\sin(\pi), \cos(\pi)\}\right) & \{0., -1.\} \\
\text{approx}\left(\sqrt{2} \sqrt{3}\right) & \begin{bmatrix} 1.41421 & 1.73205 \end{bmatrix} \\
\end{array} \]

approx\((\text{List1})\) \Rightarrow \text{list}

approx\((\text{Matrix1})\) \Rightarrow \text{matrix}

Returns a list or matrix where each element has been evaluated to a decimal value, when possible.

\[ \begin{array}{l l}
\text{approxFraction()} \Rightarrow \text{value} \\
Value \text{\textbf{\textasciitilde} approxFraction([\text{Tol}])} \Rightarrow \text{value} \\
List \text{\textbf{\textasciitilde} approxFraction([\text{Tol}])} \Rightarrow \text{list} \\
Matrix \text{\textbf{\textasciitilde} approxFraction([\text{Tol}])} \Rightarrow \text{matrix} \\
\end{array} \]

Returns the input as a fraction, using a tolerance of \(\text{Tol}\). If \(\text{Tol}\) is omitted, a tolerance of \(5.\text{E}-14\) is used.

Note: You can insert this function from the computer keyboard by typing @\textasciitilde approxFraction(...).
### approxRational()

**approxRational**\((\text{Value}, \text{Tol})\) \(\Rightarrow\) \text{value}

**approxRational**\((\text{List}, \text{Tol})\) \(\Rightarrow\) \text{list}

**approxRational**\((\text{Matrix}, \text{Tol})\) \(\Rightarrow\) \text{matrix}

Returns the argument as a fraction using a tolerance of \(\text{Tol}\). If \(\text{Tol}\) is omitted, a tolerance of 5.E-14 is used.

| \(\text{approxRational}(0.333, 5 \cdot 10^{-5})\) | \(\frac{333}{1000}\) |
| \(\text{approxRational}([0.2, 0.33, 4.125], 5 \cdot 10^{-14})\) | \[
\begin{bmatrix}
1 & 33 & 33 \\
5 & 100 & 8
\end{bmatrix}
\] |

### arccos()  
See \(\cos^{-1}\), page 29.

### arccosh()  
See \(\cosh^{-1}\), page 30.

### arccot()  
See \(\cot^{-1}\), page 31.

### arccoth()  
See \(\coth^{-1}\), page 32.

### arccsc()  
See \(\csc^{-1}\), page 34.

### arccsch()  
See \(\csch^{-1}\), page 35.

### arcsec()  
See \(\sec^{-1}\), page 119.

### arcsech()  
See \(\sech^{-1}\), page 120.
arcsin()  
See \( \sin^{-1}() \), page 126.

arcsinh()  
See \( \sinh^{-1}() \), page 127.

arctan()  
See \( \tan^{-1}() \), page 137.

arctanh()  
See \( \tanh^{-1}() \), page 138.

augment()  

<table>
<thead>
<tr>
<th>( \text{augment}(\text{List1, List2}) \Rightarrow \text{list} )</th>
<th>( \text{augment}([1, 3, 2], [5, 4]) )</th>
<th>( {1, 3, 2, 5, 4} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{augment}(\text{Matrix1, Matrix2}) \Rightarrow \text{matrix} )</td>
<td>( \begin{bmatrix} 1 &amp; 2 \ 3 &amp; 4 \end{bmatrix} \rightarrow m1 )</td>
<td>( \begin{bmatrix} 1 &amp; 2 \ 3 &amp; 4 \end{bmatrix} )</td>
</tr>
<tr>
<td></td>
<td>( \begin{bmatrix} 5 \ 6 \end{bmatrix} \rightarrow m2 )</td>
<td>( \begin{bmatrix} 5 \ 6 \end{bmatrix} )</td>
</tr>
<tr>
<td></td>
<td>( \text{augment}(m1, m2) )</td>
<td>( \begin{bmatrix} 1 &amp; 2 &amp; 5 \ 3 &amp; 4 &amp; 6 \end{bmatrix} )</td>
</tr>
</tbody>
</table>

avgRC()  

| \( \text{avgRC}(\text{Expr1, Var [=} \text{Value}[, \text{Step}]) \Rightarrow \text{expression} \) | \( x:=2 \) | 2 |
|\( \text{avgRC}(\text{Expr1, Var [=} \text{Value}[, \text{List}]) \Rightarrow \text{list} \) | \( \text{avgRC}(x^2-x+2, x) \) | 3.001 |
|\( \text{avgRC}(\text{List1, Var [=} \text{Value}[, \text{Step}]) \Rightarrow \text{list} \) | \( \text{avgRC}(x^2-x+2, x, 1) \) | 3.1 |
|\( \text{avgRC}(\text{Matrix1, Var [=} \text{Value}[, \text{Step}]) \Rightarrow \text{matrix} \) | \( \text{avgRC}(x^2-x+2, x, 3) \) | 6 |

Returns a new list that is \( \text{List2} \) appended to the end of \( \text{List1} \).

Returns a new matrix that is \( \text{Matrix2} \) appended to \( \text{Matrix1} \). When the "," character is used, the matrices must have equal row dimensions, and \( \text{Matrix2} \) is appended to \( \text{Matrix1} \) as new columns. Does not alter \( \text{Matrix1} \) or \( \text{Matrix2} \).

Returns the forward-difference quotient (average rate of change).

\( \text{Expr1} \) can be a user-defined function name (see \( \text{Func} \)).
When Value is specified, it overrides any prior variable assignment or any current "|" substitution for the variable.

Step is the step value. If Step is omitted, it defaults to 0.001.

Note that the similar function centralDiff() uses the central-difference quotient.

bal()

bal(NPmt, N, I, PV, [Pmt], [FV], [PpY], [CpY], [PmtAt], [roundValue]) ⇒ value

Amortization function that calculates schedule balance after a specified payment.

N, I, PV, Pmt, FV, PpY, CpY, and PmtAt are described in the table of TVM arguments, page 144.

NPmt specifies the payment number after which you want the data calculated.

N, I, PV, Pmt, FV, PpY, CpY, and PmtAt are described in the table of TVM arguments, page 144.

• If you omit Pmt, it defaults to Pmt=tvmPmt(N, I, PV, FV, PpY, CpY, PmtAt).
• If you omit FV, it defaults to FV=0.
• The defaults for PpY, CpY, and PmtAt are the same as for the TVM functions.

roundValue specifies the number of decimal places for rounding. Default=2.

bal(NPmt, amortTable) calculates the balance after payment number NPmt, based on amortization table amortTable. The amortTable argument must be a matrix in the form described under amortTbl(), page 11.

Note: See also Σnt() and ΣPm(), page 168.
*Base2

**Integer1 ▼ Base2 ➞ integer**

*Note:* You can insert this operator from the computer keyboard by typing \( @>\text{Base2}. \)

Converts *Integer1* to a binary number. Binary or hexadecimal numbers always have a 0b or 0h prefix, respectively. Use a zero, not the letter O, followed by b or h.

`0b binaryNumber`

`0h hexadecimalNumber`

A binary number can have up to 64 digits. A hexadecimal number can have up to 16.

Without a prefix, *Integer1* is treated as decimal (base 10). The result is displayed in binary, regardless of the Base mode.

Negative numbers are displayed in "two's complement" form. For example,

\(-1\) is displayed as

`0hFFFFFFFFFFFFF` in Hex base mode

`0b111...111` (64 1's) in Binary base mode

\(-2^{63}\) is displayed as

`0h8000000000000000` in Hex base mode

`0b100...000` (63 zeros) in Binary base mode

If you enter a decimal integer that is outside the range of a signed, 64-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range. Consider the following examples of values outside the range.

\(2^{63}\) becomes \(-2^{63}\) and is displayed as

`0h8000000000000000` in Hex base mode

`0b100...000` (63 zeros) in Binary base mode

\(2^{64}\) becomes 0 and is displayed as

`0h0` in Hex base mode

`0b0` in Binary base mode

\(-2^{63} - 1\) becomes \(2^{63} - 1\) and is displayed as

`0h7FFFFFFF...F` in Hex base mode

`0b111...111` (64 1's) in Binary base mode
**Base10**

*Integer1 ▶ Base10 ⇒ integer*

**Note:** You can insert this operator from the computer keyboard by typing `@>Base10`.

Converts *Integer1* to a decimal (base 10) number. A binary or hexadecimal entry must always have a 0b or 0h prefix, respectively.

- `0b binaryNumber`  
- `0h hexadecimalNumber`

Zero, not the letter O, followed by b or h.

A binary number can have up to 64 digits. A hexadecimal number can have up to 16.

Without a prefix, *Integer1* is treated as decimal. The result is displayed in decimal, regardless of the Base mode.

---

**Base16**

*Integer1 ▶ Base16 ⇒ integer*

**Note:** You can insert this operator from the computer keyboard by typing `@>Base16`.

Converts *Integer1* to a hexadecimal number. Binary or hexadecimal numbers always have a 0b or 0h prefix, respectively.

- `0b binaryNumber`  
- `0h hexadecimalNumber`

Zero, not the letter O, followed by b or h.

A binary number can have up to 64 digits. A hexadecimal number can have up to 16.

Without a prefix, *Integer1* is treated as decimal (base 10). The result is displayed in hexadecimal, regardless of the Base mode.

If you enter a decimal integer that is too large for a signed, 64-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range. For more information, see ▶ *Base2*, page 20.
binomCdf() \[\text{Catalog > }\]

binomCdf(n, p) \(\Rightarrow\) number

binomCdf(n, p, lowBound, upBound) \(\Rightarrow\) number if lowBound and upBound are numbers, list if lowBound and upBound are lists

binomCdf(n, p, upBound) for \(P(0 \leq X \leq \text{upBound})\) \(\Rightarrow\) number if upBound is a number, list if upBound is a list

Computes a cumulative probability for the discrete binomial distribution with \(n\) number of trials and probability \(p\) of success on each trial.

For \(P(X \leq \text{upBound})\), set lowBound=0

binomPdf() \[\text{Catalog > }\]

binomPdf(n, p) \(\Rightarrow\) number

binomPdf(n, p, XVal) \(\Rightarrow\) number if \(XVal\) is a number, list if \(XVal\) is a list

Computes a probability for the discrete binomial distribution with \(n\) number of trials and probability \(p\) of success on each trial.

C

ceiling() \[\text{Catalog > }\]

ceiling(Value) \(\Rightarrow\) value

Returns the nearest integer that is \(\geq\) the argument.

The argument can be a real or a complex number.

Note: See also floor().

ceiling(List) \(\Rightarrow\) list

ceiling(Matrix) \(\Rightarrow\) matrix

Returns a list or matrix of the ceiling of each element.

centralDiff() \[\text{Catalog > }\]

centralDiff(Expr, Var \([=\text{Value}],[\text{Step}]\)) \(\Rightarrow\) expression

centralDiff(Expr, Var \([\text{Step}]\) | Var=\(\text{Value}\)) \(\Rightarrow\) expression

centralDiff(\(\cos(x), x\)) \(x=\frac{\pi}{2}\) \(\Rightarrow\) -1.
centralDiff()

centralDiff(Expr1,Var [=Value][,List]) ⇒ list

centralDiff(List1,Var [=Value][,Step]) ⇒ list

centralDiff(Matrix1,Var [=Value][,Step]) ⇒ matrix

Returns the numerical derivative using the central difference quotient formula.

When Value is specified, it overrides any prior variable assignment or any current "|" substitution for the variable.

Step is the step value. If Step is omitted, it defaults to 0.001.

When using List1 or Matrix1, the operation gets mapped across the values in the list or across the matrix elements.

Note: See also avgRC().

char()

char(Integer) ⇒ character

Returns a character string containing the character numbered Integer from the handheld character set.

The valid range for Integer is 0-65535.

\( \chi^2 \)

\( \chi^2 \) 2way obsMatrix

\( \chi^2 \) 2way obsMatrix

Computes a \( \chi^2 \) test for association on the two-way table of counts in the observed matrix obsMatrix. A summary of results is stored in the stat.results variable. (page 131)

For information on the effect of empty elements in a matrix, see “Empty (Void) Elements,” page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.( \chi^2 )</td>
<td>Chi square stat: sum (observed - expected)^2/expected</td>
</tr>
</tbody>
</table>
### Output variable Description

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.PVal</td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>stat.df</td>
<td>Degrees of freedom for the chi square statistics</td>
</tr>
<tr>
<td>stat.ExpMat</td>
<td>Matrix of expected elemental count table, assuming null hypothesis</td>
</tr>
<tr>
<td>stat.CompMat</td>
<td>Matrix of elemental chi square statistic contributions</td>
</tr>
</tbody>
</table>

### \( \chi^2 \) Cdf()

\( \chi^2 \) Cdf\((lowBound,upBound,df) \Rightarrow \text{number if lowBound and upBound are numbers, list if lowBound and upBound are lists} \)

### \( \chi^2 \) GOF

\( \chi^2 \) GOF\( \text{obsList,expList,df} \)

\( \chi^2 \) GOF\( \text{obsList,expList,df} \)

Performs a test to confirm that sample data is from a population that conforms to a specified distribution. \( \text{obsList} \) is a list of counts and must contain integers. A summary of results is stored in the \( \text{stat.results} \) variable. (See page 131.)

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.
$\chi^2$Pdf()

$\chi^2\text{Pdf}(XVal, df) \Rightarrow \text{number if } XVal \text{ is a number, list if } XVal \text{ is a list}$

$\text{chi2Pdf}(XVal, df) \Rightarrow \text{number if } XVal \text{ is a number, list if } XVal \text{ is a list}$

Computes the probability density function (pdf) for the $\chi^2$ distribution at a specified $XVal$ value for the specified degrees of freedom $df$.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.

ClearAZ

Clears all single-character variables in the current problem space.

If one or more of the variables are locked, this command displays an error message and deletes only the unlocked variables. See unLock, page 147.

ClrErr

Clears the error status and sets system variable $\text{errCode}$ to zero.

The Else clause of the Try...Else...EndTry block should use ClrErr or PassErr. If the error is to be processed or ignored, use ClrErr. If what to do with the error is not known, use PassErr to send it to the next error handler. If there are no more pending Try...Else...EndTry error handlers, the error dialog box will be displayed as normal.

Note: See also PassErr, page 98, and Try, page 141.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing $\text{[ adjacent}$ instead of $\begin{array}{c}\text{[ enter}\end{array}$ at the end of each line. On the computer keyboard, hold down Alt and press Enter.
**colAugment**

\[
\text{colAugment}(\text{Matrix1, Matrix2}) \Rightarrow \text{matrix}
\]

Returns a new matrix that is Matrix2 appended to Matrix1. The matrices must have equal column dimensions, and Matrix2 is appended to Matrix1 as new rows. Does not alter Matrix1 or Matrix2.

\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
5 & 6
\end{bmatrix} \rightarrow \text{m1} \\
\begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix} \\
\begin{bmatrix}
5 & 6
\end{bmatrix} \rightarrow \text{m2}
\]

\[
\text{colAugment(m1,m2)} = \begin{bmatrix}
1 & 2 \\
3 & 4 \\
5 & 6
\end{bmatrix}
\]

**colDim**

\[
\text{colDim}(<\text{Matrix}>) \Rightarrow \text{expression}
\]

Returns the number of columns contained in Matrix.

Example:

\[
\text{colDim}\left(\begin{bmatrix}
0 & 1 & 2 \\
3 & 4 & 5
\end{bmatrix}\right) = 3
\]

**colNorm**

\[
\text{colNorm}(<\text{Matrix}>) \Rightarrow \text{expression}
\]

Returns the maximum of the sums of the absolute values of the elements in the columns in Matrix.

Example:

\[
\text{colNorm}\left(\begin{bmatrix}
1 & -2 & 3 \\
4 & 5 & -6
\end{bmatrix}\right) = 9
\]

**conj**

\[
\text{conj}(\text{Value1}) \Rightarrow \text{value}
\]

\[
\text{conj}(\text{List1}) \Rightarrow \text{list}
\]

\[
\text{conj}(\text{Matrix1}) \Rightarrow \text{matrix}
\]

Returns the complex conjugate of the argument.

\[
\text{conj}\left(\begin{bmatrix}
1+2\cdot i \\
2 & 1-3\cdot i \\
i & -7
\end{bmatrix}\right) = \begin{bmatrix}
1-2\cdot i \\
2 & 1+3\cdot i \\
i & -7
\end{bmatrix}
\]

**constructMat**

\[
\text{constructMat}(\text{Expr, Var1, Var2, numRows, numCols}) \Rightarrow \text{matrix}
\]

Returns a matrix based on the arguments.

\[
\text{constructMat}\left(\frac{1}{i+j},i,j,3,4\right) = \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1
\end{bmatrix}
\]

**Alphabetical Listing**

26
evaluating \( \text{Expr} \) for each incremented value of \( \text{Var1} \) and \( \text{Var2} \).

\( \text{Var1} \) is automatically incremented from 1 through \( \text{numRows} \). Within each row, \( \text{Var2} \) is incremented from 1 through \( \text{numCols} \).

<table>
<thead>
<tr>
<th>CopyVar</th>
<th>( \text{Var1}, \text{Var2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>CopyVar</td>
<td>( \text{Var1}, \text{Var2} ) copies the value of variable ( \text{Var1} ) to variable ( \text{Var2} ), creating ( \text{Var2} ) if necessary. Variable ( \text{Var1} ) must have a value.</td>
</tr>
<tr>
<td>If ( \text{Var1} ) is the name of an existing user-defined function, copies the definition of that function to function ( \text{Var2} ). Function ( \text{Var1} ) must be defined.</td>
<td></td>
</tr>
<tr>
<td>( \text{Var1} ) must meet the variable-naming requirements or must be an indirection expression that simplifies to a variable name meeting the requirements.</td>
<td></td>
</tr>
</tbody>
</table>

| CopyVar | \( \text{Var1}, \text{Var2} \). copies all members of the \( \text{Var1} \). variable group to the \( \text{Var2} \). group, creating \( \text{Var2} \). if necessary. |
| Var1. must be the name of an existing variable group, such as the statistics \( \text{stat.nn} \) results, or variables created using the \( \text{LibShortcut()} \) function. If \( \text{Var2} \). already exists, this command replaces all members that are common to both groups and adds the members that do not already exist. If one or more members of \( \text{Var2} \). are locked, all members of \( \text{Var2} \). are left unchanged. |

<table>
<thead>
<tr>
<th>\text{corrMat()} ( \text{Catalog} &gt; )</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{corrMat} ( [\text{List1}, \text{List2}[,\ldots,\text{List20}]] )</td>
</tr>
<tr>
<td>Computes the correlation matrix for the augmented matrix ( [\text{List1}, \text{List2}, \ldots, \text{List20}] ).</td>
</tr>
</tbody>
</table>
**cos()**

\[ \cos(Value) \Rightarrow value \]

\[ \cos(List) \Rightarrow list \]

\[ \cos(Value) \] returns the cosine of the argument as a value.

\[ \cos(List) \] returns a list of the cosines of all elements in List.

**Note:** The argument is interpreted as a degree, gradian or radian angle, according to the current angle mode setting. You can use °, G, or r to override the angle mode temporarily.

**cos(squareMatrix) ⇒ squareMatrix**

Returns the matrix cosine of squareMatrix. This is not the same as calculating the cosine of each element.

When a scalar function \( f(A) \) operates on \( squareMatrix \) (A), the result is calculated by the algorithm:

Compute the eigenvalues \( \lambda_i \) and eigenvectors \( V_i \) of A.

squareMatrix must be diagonalizable. Also, it cannot have symbolic variables that have not been assigned a value.

Form the matrices:

\[
B = \begin{bmatrix}
\lambda_1 & 0 & \cdots & 0 \\
0 & \lambda_2 & \cdots & 0 \\
0 & 0 & \cdots & 0 \\
0 & 0 & \cdots & \lambda_n \\
\end{bmatrix}
\text{ and } X = [V_1, V_2, \ldots, V_n]
\]

Then \( A = X B X^{-1} \) and \( f(A) = X f(B) X^{-1} \). For example, \( \cos(A) = X \cos(B) X^{-1} \) where:

\[
\cos(B) =
\]

In Degree angle mode:

\[
\begin{array}{c}
\cos\left(\frac{\pi}{4}\right) \\
\cos(45) \\
\cos\left(\{0,60,90\}\right) \\
\end{array}
\]

\[
\begin{array}{c}
0.707107 \\
0.707107 \\
\{1,0.5,0\} \\
\end{array}
\]

In Gradian angle mode:

\[
\begin{array}{c}
\cos\left(\{0,50,100\}\right) \\
\end{array}
\]

\[
\{1,0.707107,0\}
\]

In Radian angle mode:

\[
\begin{array}{c}
\cos\left(\frac{\pi}{4}\right) \\
\cos(45) \\
\end{array}
\]

\[
\begin{array}{c}
0.707107 \\
0.707107 \\
\end{array}
\]

\[
\begin{array}{ccc}
0.212493 & 0.205064 & 0.121389 \\
0.160871 & 0.259042 & 0.037126 \\
0.248079 & 0.090153 & 0.218972 \\
\end{array}
\]

28  **Alphabetical Listing**
All computations are performed using floating-point arithmetic.

\[
\begin{bmatrix}
\cos(\lambda_1) & 0 & \ldots & 0 \\
0 & \cos(\lambda_2) & \ldots & 0 \\
0 & 0 & \ldots & 0 \\
0 & 0 & \ldots & \cos(\lambda_n)
\end{bmatrix}
\]

\[\cos()\]

In Degree angle mode:
\[
\cos^\circ\{(1)\}
\]
\[0.\]

In Gradian angle mode:
\[
\cos^\circ\{(0)\}
\]
\[100.\]

In Radian angle mode:
\[
\cos^\circ\{(0,0,2,0.5)\}
\]
\[\{1.5708,1.36944,1.0472\}\]

\[\cos()\]

Returns the matrix inverse cosine of \(\text{squareMatrix1}\).
This is not the same as calculating the inverse cosine of each element. For information about the calculation method, refer to \(\cos()\).

\(\text{squareMatrix1}\) must be diagonalizable. The result always contains floating-point numbers.

\[\cosh()\]

In Degree angle mode:
\[
\cosh(\{1\})
\]

\[1.73485+0.064606i\]
\[-1.49086+2.10514i\]
\[-0.725533+1.51594i\]
\[0.623491+0.77836i\]
\[-2.08316+2.63205i\]
\[1.79018-1.27182i\]

To see the entire result, press \(\uparrow\) and then use \(\downarrow\) and \(\leftarrow\) to move the cursor.
\( \text{cosh}() \)

\[ \text{cosh}(Value1) \] returns the hyperbolic cosine of the argument.

\[ \text{cosh}(List1) \] returns a list of the hyperbolic cosines of each element of \( List1 \).

\[ \text{cosh}(\text{squareMatrix1}) \Rightarrow \text{squareMatrix} \]

Returns the matrix hyperbolic cosine of \( \text{squareMatrix1} \). This is not the same as calculating the hyperbolic cosine of each element. For information about the calculation method, refer to \( \text{cos}() \).

\( \text{squareMatrix1} \) must be diagonalizable. The result always contains floating-point numbers.

\[ \text{cosh}^{-1}() \]

\[ \text{cosh}^{-1}(Value1) \Rightarrow \text{value} \]

\[ \text{cosh}^{-1}(List1) \Rightarrow \text{list} \]

\[ \text{cosh}^{-1}(Value1) \] returns the inverse hyperbolic cosine of the argument.

\[ \text{cosh}^{-1}(List1) \] returns a list of the inverse hyperbolic cosines of each element of \( List1 \).

**Note:** You can insert this function from the keyboard by typing \( \text{arccosh}(\ldots) \).

\[ \text{cosh}^{-1}(\text{squareMatrix1}) \Rightarrow \text{squareMatrix} \]

Returns the matrix inverse hyperbolic cosine of \( \text{squareMatrix1} \). This is not the same as calculating the inverse hyperbolic cosine of each element. For information about the calculation method, refer to \( \text{cos}() \).

\( \text{squareMatrix1} \) must be diagonalizable. The result always contains floating-point numbers.
**cot()**

\[ \text{cot}(Value1) \Rightarrow \text{value} \]
\[ \text{cot}(List1) \Rightarrow \text{list} \]

Returns the cotangent of \( Value1 \) or returns a list of the cotangents of all elements in \( List1 \).

**Note:** The argument is interpreted as a degree, gradian or radian angle, according to the current angle mode setting. You can use °, G, or r to override the angle mode temporarily.

In Gradian angle mode:
\[ \text{cot}(50) \Rightarrow 1. \]

In Radian angle mode:
\[ \text{cot}\left\{1,2,1,3\right\} \Rightarrow \left\{0.642093, 0.584848, 7.01525\right\} \]

**cot⁻¹()**

\[ \text{cot}^{-1}(Value1) \Rightarrow \text{value} \]
\[ \text{cot}^{-1}(List1) \Rightarrow \text{list} \]

Returns the angle whose cotangent is \( Value1 \) or returns a list containing the inverse cotangents of each element of \( List1 \).

**Note:** The result is returned as a degree, gradian or radian angle, according to the current angle mode setting.

**Note:** You can insert this function from the keyboard by typing \text{arccot}(\ldots)\).

In Degree angle mode:
\[ \text{cot}^{-1}(1) \Rightarrow 45 \]

In Gradian angle mode:
\[ \text{cot}^{-1}(1) \Rightarrow 50 \]

In Radian angle mode:
\[ \text{cot}^{-1}(1) \Rightarrow 0.785398 \]

**coth()**

\[ \text{coth}(Value1) \Rightarrow \text{value} \]
\[ \text{coth}(List1) \Rightarrow \text{list} \]

Returns the hyperbolic cotangent of \( Value1 \) or returns a list of the hyperbolic cotangents of all elements of \( List1 \).
coth⁻¹()  

\[
\text{coth}^{-1}(\text{Value1}) \Rightarrow \text{value} \\
\text{coth}^{-1}(\text{List1}) \Rightarrow \text{list}
\]

Returns the inverse hyperbolic cotangent of Value1 or returns a list containing the inverse hyperbolic cotangents of each element of List1.

**Note:** You can insert this function from the keyboard by typing `arccoth (...)`.

count()  

\[
\text{count}(\text{Value1 or List1} [, \text{Value2 or List2} [...]]) \Rightarrow \text{value}
\]

Returns the accumulated count of all elements in the arguments that evaluate to numeric values.

Each argument can be an expression, value, list, or matrix. You can mix data types and use arguments of various dimensions.

For a list, matrix, or range of cells, each element is evaluated to determine if it should be included in the count.

Within the Lists & Spreadsheet application, you can use a range of cells in place of any argument.

Empty (void) elements are ignored. For more information on empty elements, see page 177.

countif()  

\[
\text{countif}(\text{List}, \text{Criteria}) \Rightarrow \text{value}
\]

Returns the accumulated count of all elements in List that meet the specified Criteria.

**Criteria** can be:

- A value, expression, or string. For example, 3 counts only those elements in List that simplify to the value 3.
- A Boolean expression containing the symbol \(?\).
countif() as a placeholder for each element. For example, ?<5 counts only those elements in List that are less than 5.

Within the Lists & Spreadsheet application, you can use a range of cells in place of List.

Empty (void) elements in the list are ignored. For more information on empty elements, see page 177.

Note: See also sumIf(), page 135, and frequency(), page 54.

cPolyRoots()

cPolyRoots(Poly, Var) ⇒ list

cPolyRoots(ListOfCoeffs) ⇒ list

The first syntax, cPolyRoots(Poly, Var), returns a list of complex roots of polynomial Poly with respect to variable Var.

Poly must be a polynomial in expanded form in one variable. Do not use unexpanded forms such as y\(^2\)*y+1 or x*x+2*x+1

The second syntax, cPolyRoots(ListOfCoeffs), returns a list of complex roots for the coefficients in ListOfCoeffs.

Note: See also polyRoots(), page 101.

crossP()

crossP(List1, List2) ⇒ list

Returns the cross product of List1 and List2 as a list. List1 and List2 must have equal dimension, and the dimension must be either 2 or 3.

crossP(Vector1, Vector2) ⇒ vector

Returns a row or column vector (depending on the arguments) that is the cross product of Vector1 and
crossP()

Vector2.

Both Vector1 and Vector2 must be row vectors, or both must be column vectors. Both vectors must have equal dimension, and the dimension must be either 2 or 3.

csc()

\[ \text{csc}(\text{Value1}) \Rightarrow \text{value} \]
\[ \text{csc}(\text{List1}) \Rightarrow \text{list} \]

Returns the cosecant of Value1 or returns a list containing the cosecants of all elements in List1.

In Degree angle mode:

\[ \text{csc}(45) \Rightarrow 1.41421 \]

In Gradian angle mode:

\[ \text{csc}(50) \Rightarrow 1.41421 \]

In Radian angle mode:

\[ \text{csc}\left\{ \frac{\pi}{1}, \frac{\pi}{2}, \frac{\pi}{3} \right\} \Rightarrow \{1.1884, 1.1547\} \]

\[ \text{csc}^{-1}() \]

\[ \text{csc}^{-1}(\text{Value1}) \Rightarrow \text{value} \]
\[ \text{csc}^{-1}(\text{List1}) \Rightarrow \text{list} \]

Returns the angle whose cosecant is Value1 or returns a list containing the inverse cosecants of each element of List1.

Note: The result is returned as a degree, gradian or radian angle, according to the current angle mode setting.

Note: You can insert this function from the keyboard by typing \texttt{arccsc} (...) .
\textbf{csch()}

\texttt{csch(Value1) ⇒ value}

\texttt{csch(List1) ⇒ list}

Returns the hyperbolic cosecant of \textit{Value1} or returns a list of the hyperbolic cosecants of all elements of \textit{List1}.

\textbf{csch\(^{-1}\)()}

\texttt{csch\(^{-1}\)(Value) ⇒ value}

\texttt{csch\(^{-1}\)(List1) ⇒ list}

Returns the inverse hyperbolic cosecant of \textit{Value1} or returns a list containing the inverse hyperbolic cosecants of each element of \textit{List1}.

\textbf{CubicReg}

\texttt{CubicReg \(X, Y[, [Freq[, Category[, Include]]]]\)}

Computes the cubic polynomial regression \(y=a\times x^3+b\times x^2+c\times x+d\) on lists \(X\) and \(Y\) with frequency \(Freq\). A summary of results is stored in the \textit{stat.results} variable. (See page 131.)

All the lists must have equal dimension except for \textit{Include}.

\(X\) and \(Y\) are lists of independent and dependent variables.

\textit{Freq} is an optional list of frequency values. Each element in \textit{Freq} specifies the frequency of occurrence for each corresponding \(X\) and \(Y\) data point. The default value is 1. All elements must be integers \(\geq 0\).

\textit{Category} is a list of numeric or string category codes for the corresponding \(X\) and \(Y\) data.

\textit{Include} is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.
<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression equation: $a \cdot x^3 + b \cdot x^2 + c \cdot x + d$</td>
</tr>
<tr>
<td>stat.a, stat.b, stat.c, stat.d</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.R²</td>
<td>Coefficient of determination</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the regression</td>
</tr>
<tr>
<td>stat.XReg</td>
<td>List of data points in the modified $X$ List actually used in the regression based on restrictions of $Freq$, $Category$ List, and Include Categories</td>
</tr>
<tr>
<td>stat.YReg</td>
<td>List of data points in the modified $Y$ List actually used in the regression based on restrictions of $Freq$, $Category$ List, and Include Categories</td>
</tr>
<tr>
<td>statFreqReg</td>
<td>List of frequencies corresponding to stat.XReg and stat.YReg</td>
</tr>
</tbody>
</table>

**cumulativeSum()**

```
cumulativeSum(List1) \Rightarrow list
```

Returns a list of the cumulative sums of the elements in List1, starting at element 1.

```
cumulativeSum(List1) \Rightarrow matrix
```

Returns a matrix of the cumulative sums of the elements in Matrix1. Each element is the cumulative sum of the column from top to bottom.

An empty (void) element in List1 or Matrix1 produces a void element in the resulting list or matrix. For more information on empty elements, see page 177.

**Cycle**

```
Cycle
```

Transfers control immediately to the next iteration of the current loop (For, While, or Loop).

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing ENTER instead of ENTER at the end of each line. On the computer keyboard, hold down Alt and press Enter.

```plaintext
Function listing that sums the integers from 1 to 100 skipping 50.
```

Define $g() = Func$
```
Local temp, i
0 \to temp
For i=1,100,1
If i=50
Cycle
temp+i \to temp
EndFor
Return temp
EndFunc
```

Return $g()$ 5000
Cylind

Vector+Cylind

Note: You can insert this operator from the computer keyboard by typing @>Cylind.

Displays the row or column vector in cylindrical form [r, θ, z].

Vector must have exactly three elements. It can be either a row or a column.

\[
\begin{bmatrix} 2 & 2 & 3 \end{bmatrix} \rightarrow \text{Cylind} \Rightarrow \begin{bmatrix} 2.82843 & \angle 0.785398 & 3.0 \end{bmatrix}
\]

\[\text{dbd}(\text{date1},\text{date2}) \Rightarrow \text{value}\]

Returns the number of days between \(\text{date1}\) and \(\text{date2}\) using the actual-day-count method.

\(\text{date1}\) and \(\text{date2}\) can be numbers or lists of numbers within the range of the dates on the standard calendar. If both \(\text{date1}\) and \(\text{date2}\) are lists, they must be the same length.

\(\text{date1}\) and \(\text{date2}\) must be between the years 1950 through 2049.

You can enter the dates in either of two formats. The decimal placement differentiates between the date formats.

**MM.DDYY** (format used commonly in the United States)

**DDMM.YY** (format use commonly in Europe)

\[\text{dbd} \left( 12.3103,1.0104 \right) \Rightarrow 1 \]

\[\text{dbd} \left( 1.0107,6.0107 \right) \Rightarrow 151 \]

\[\text{dbd} \left( 3112.03,101.04 \right) \Rightarrow 1 \]

\[\text{dbd} \left( 101.07,106.07 \right) \Rightarrow 151 \]

\[\text{DD} \]

\[\text{Expr1} \rightarrow \text{DD} \Rightarrow \text{valueList1}\]

\[\text{DD} \Rightarrow \text{listMatrix1}\]

\[\text{DD} \Rightarrow \text{matrix}\]

Note: You can insert this operator from the computer keyboard by typing @>DD.

Returns the decimal equivalent of the argument

In Degree angle mode:

\[\begin{bmatrix} 1.5^\circ \end{bmatrix} \rightarrow \text{DD} \Rightarrow 1.5^\circ \]

\[\begin{bmatrix} 45^\circ 22'14.3'' \end{bmatrix} \rightarrow \text{DD} \Rightarrow 45.3706^\circ \]

\[\begin{bmatrix} \{ 45^\circ 22'14.3'',60^\circ 00'' \} \end{bmatrix} \rightarrow \text{DD} \Rightarrow \{ 45.3706^\circ,60^\circ \} \]
expressed in degrees. The argument is a number, list, or matrix that is interpreted by the Angle mode setting in gradians, radians or degrees.

In Gradian angle mode:

\[
1 \times \text{DD} = \frac{9}{10}^\circ
\]

In Radian angle mode:

\[
(1.5) \times \text{DD} = 85.9437^\circ
\]

Number1 \(\times\) Decimal \(\Rightarrow\) value

List1 \(\times\) Decimal \(\Rightarrow\) value

Matrix1 \(\times\) Decimal \(\Rightarrow\) value

Note: You can insert this operator from the computer keyboard by typing @>Decimal.

Displays the argument in decimal form. This operator can be used only at the end of the entry line.

Define \(\text{Var} = \text{Expression}\)
Define \(\text{Function(Param1, Param2, ...)} = \text{Expression}\)

Defines the variable \(\text{Var}\) or the user-defined function \(\text{Function}\).

Parameters, such as \(\text{Param1}\), provide placeholders for passing arguments to the function. When calling a user-defined function, you must supply arguments (for example, values or variables) that correspond to the parameters. When called, the function evaluates \(\text{Expression}\) using the supplied arguments.

\(\text{Var}\) and \(\text{Function}\) cannot be the name of a system variable or built-in function or command.

Note: This form of \text{Define} is equivalent to executing the expression: \(\text{expression} \rightarrow \text{Function (Param1, Param2)}\).
Define Function(Param1, Param2, ...) = Func
    Block
EndFunc

Define Program(Param1, Param2, ...) = Prgm
    Block
EndPrgm

In this form, the user-defined function or program can execute a block of multiple statements.

Block can be either a single statement or a series of statements on separate lines. Block also can include expressions and instructions (such as If, Then, Else, and For).

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing [→] instead of [enter] at the end of each line. On the computer keyboard, hold down Alt and press Enter.

Note: See also Define LibPriv, page 39, and Define LibPub, page 40.

Define LibPriv

Define LibPriv Var = Expression
Define LibPriv Function(Param1, Param2, ...) = Expression
Define LibPriv Function(Param1, Param2, ...) = Func
    Block
EndFunc
Define LibPriv Program(Param1, Param2, ...) = Prgm
    Block
EndPrgm

Operates the same as Define, except defines a private library variable, function, or program. Private functions and programs do not appear in the Catalog.

Note: See also Define, page 38, and Define LibPub, page 40.

Define g(x,y) = Func
    If x > y Then
    Return x
    Else
    Return y
    EndIf
EndFunc

g(3,-7) 3

Define g(x,y) = Prgm
    If x > y Then
    Disp x, " greater than " . . . . y 
    Else
    Disp x, " not greater than " . . . . y
    EndIf
EndPrgm

g(3,-7) 3 greater than -7
Define LibPub

Define LibPub Var = Expression
Define LibPub Function(Param1, Param2, ...) = Expression
Define LibPub Function(Param1, Param2, ...) = Func
    Block
EndFunc
Define LibPub Program(Param1, Param2, ...) = Prgm
    Block
EndPrgm

Operates the same as Define, except defines a public library variable, function, or program. Public functions and programs appear in the Catalog after the library has been saved and refreshed.

Note: See also Define, page 38, and Define LibPriv, page 39.

DelVar

DelVar Var1, Var2 [, Var3] ...
DelVar Var.

Deletes the specified variable or variable group from memory.

If one or more of the variables are locked, this command displays an error message and deletes only the unlocked variables. See unLock, page 147.

DelVar Var, deletes all members of the Var, variable group (such as the statistics stat.nn results or variables created using the LibShortcut() function). The dot (.) in this form of the DelVar command limits it to deleting a variable group; the simple variable Var is not affected.

<table>
<thead>
<tr>
<th>DeltaList()</th>
<th>See ΔList(), page 73.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>DelVar</th>
<th>Catalog &gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 \rightarrow a</td>
<td>2</td>
</tr>
<tr>
<td>( (a+2)^2 )</td>
<td>16</td>
</tr>
<tr>
<td>DelVar a</td>
<td>Done</td>
</tr>
<tr>
<td>( (a+2)^2 )</td>
<td>&quot;Error: Variable is not defined&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DeltaList()</th>
<th>See ΔList(), page 73.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>DelVar</th>
<th>Catalog &gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>( aa.a=45 )</td>
<td>45</td>
</tr>
<tr>
<td>( aa.b=5.67 )</td>
<td>5.67</td>
</tr>
<tr>
<td>( aa.c=78.9 )</td>
<td>78.9</td>
</tr>
</tbody>
</table>
| getVarInfo() | \( aa.a \) "NUM" "\{\}"
| getVarInfo() | \( aa.b \) "NUM" "\{\}"
| getVarInfo() | \( aa.c \) "NUM" "\{\}"
| DelVar aa. | Done |
| getVarInfo() | "NONE" |

40 Alphabetical Listing
\textbf{delVoid()}

\texttt{delVoid(List1) \Rightarrow list}

Returns a list that has the contents of \textit{List1} with all empty (void) elements removed.

For more information on empty elements, see page 177.

\textbf{det()}

\texttt{det(squareMatrix[, Tolerance]) \Rightarrow expression}

Returns the determinant of \textit{squareMatrix}.

Optionally, any matrix element is treated as zero if its absolute value is less than \textit{Tolerance}. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, \textit{Tolerance} is ignored.

- If you use \texttt{ctrl enter} or set the Auto or Approximate mode to Approximate, computations are done using floating-point arithmetic.
- If \textit{Tolerance} is omitted or not used, the default tolerance is calculated as:
  \[ 5 \times 10^{-14} \max(\text{dim}(\text{squareMatrix}))\text{rowNorm}(\text{squareMatrix}) \]

\textbf{diag()}

\texttt{diag(List) \Rightarrow matrix}
\texttt{diag(rowMatrix) \Rightarrow matrix}
\texttt{diag(columnMatrix) \Rightarrow matrix}

Returns a matrix with the values in the argument list or matrix in its main diagonal.

\texttt{diag(squareMatrix) \Rightarrow rowMatrix}

Returns a row matrix containing the elements from the main diagonal of \textit{squareMatrix}.

\textit{squareMatrix} must be square.
**dim()**

```
dim(List) ⇒ integer
```

Returns the dimension of List.

```
dim(Matrix) ⇒ list
```

Returns the dimensions of matrix as a two-element list (rows, columns).

```
dim(String) ⇒ integer
```

Returns the number of characters contained in character string String.

```
dim({0,1,2})
```

3

```
dim([[1 -1]
      [2 -2]
      [3  5]])
```

{3,2}

```
dim("Hello")
dim("Hello" & "there")
```

5

11

**Disp**

```
Disp[exprOrString1 [, exprOrString2] ...]
```

Displays the arguments in the Calculator history. The arguments are displayed in succession, with thin spaces as separators.

Useful mainly in programs and functions to ensure the display of intermediate calculations.

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing ‹ instead of ‹enter› at the end of each line. On the computer keyboard, hold down Alt and press Enter.

```
Define chars(start,end)=Prgm
   For i,start,end
      Disp i," ",char(i)
   EndFor
EndPrgm
```

Done

```
chars[240,243]
```

240 0
241 ñ
242 0
243 0

Done

**►DMS**

```
Value ►DMS
List ►DMS
Matrix ►DMS
```

In Degree angle mode:

```
(45.371)►DMS 45°22′15.6″
{45.371,60}►DMS {45°22′15.6″,60°}
```

**Note:** You can insert this operator from the computer keyboard by typing @>DMS.

Interprets the argument as an angle and displays the equivalent DMS (DDDDD°MM'SS.ss") number.

See °, ′, ″ on page 172 for DMS (degree, minutes,
DMS

seconds) format.

**Note:** DMS will convert from radians to degrees when used in radian mode. If the input is followed by a degree symbol °, no conversion will occur. You can use DMS only at the end of an entry line.

dotP()

**dotP(List1, List2) ⇒ expression**

Returns the “dot” product of two lists.

**dotP(Vector1, Vector2) ⇒ expression**

Returns the “dot” product of two vectors.

Both must be row vectors, or both must be column vectors.

E

e^0

e^0(Value1) ⇒ value

Returns e raised to the Value1 power.

**Note:** See also e exponent template, page 6.

**Note:** Pressing [ ▲ ] to display e^ is different from pressing the character [ E ] on the keyboard.

You can enter a complex number in re^iθ polar form. However, use this form in Radian angle mode only; it causes a Domain error in Degree or Gradian angle mode.

e^0(List1) ⇒ list

Returns e raised to the power of each element in List1.
$e^0$

$e^x(\text{squareMatrix}) \Rightarrow \text{squareMatrix}$

Returns the matrix exponential of $\text{squareMatrix}$. This is not the same as calculating $e$ raised to the power of each element. For information about the calculation method, refer to $\text{cos}()$.

$squareMatrix$ must be diagonalizable. The result always contains floating-point numbers.

<table>
<thead>
<tr>
<th>$e^x$ key</th>
<th>1</th>
<th>5</th>
<th>3</th>
<th>782.209</th>
<th>559.617</th>
<th>456.509</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>680.546</td>
<td>488.795</td>
<td>396.521</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>1</td>
<td>524.929</td>
<td>371.222</td>
<td>307.879</td>
<td></td>
</tr>
</tbody>
</table>
eigVl() \[\text{Catalog} > \]

\[\text{eigVl(squareMatrix)} \Rightarrow \text{list}\]

Returns a list of the eigenvalues of a real or complex squareMatrix.

squareMatrix is first balanced with similarity transformations until the row and column norms are as close to the same value as possible. The squareMatrix is then reduced to upper Hessenberg form and the eigenvalues are computed from the upper Hessenberg matrix.

In Rectangular complex format mode:

\[
\begin{bmatrix}
-1 & 2 & 5 \\
3 & -6 & 9 \\
2 & -5 & 7 \\
\end{bmatrix} \Rightarrow \text{m1}
\]

\[
\begin{bmatrix}
-1 & 2 & 5 \\
3 & -6 & 9 \\
2 & -5 & 7 \\
\end{bmatrix}
\]

eigVl(m1)

\[\{-4.40941,2.20471+0.763006\cdot i,2.20471-0\cdot i\}\]

To see the entire result, press \(\uparrow\) and then use \(\downarrow\) and \(\uparrow\) to move the cursor.

Else

See If, page 61.

Elself

If BooleanExpr1 Then

Block1

Elseif BooleanExpr2 Then

Block2

Elseif BooleanExprN Then

BlockN

EndIf

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing \(\downarrow\) instead of \(\text{enter}\) at the end of each line. On the computer keyboard, hold down \(\text{Alt}\) and press \(\text{Enter}\).

EndFor

See For, page 52.

EndFunc

See Func, page 55.
Uses the Euler method to solve the system
\[
\frac{d\text{depVar}}{d\text{Var}} = \text{Expr(Var, depVar)}
\]
with depVar(Var0)=depVar0 on the interval [Var0,VarMax]. Returns a matrix whose first row
defines the Var output values and whose second row
defines the value of the first solution component at
the corresponding Var values, and so on.

Expr is the right-hand side that defines the ordinary
differential equation (ODE).

\[
y' = 0.001y^2(100-y) \quad \text{and} \quad y(0)=10
\]
\[
euler\{0.001\cdot y\cdot(100-y), y, \{0,100\}, 10, 1\}
\]
\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
10 & 10.9 & 11.8712 & 12.9174 & 14.042
\end{array}
\]

To see the entire result, press \(\uparrow\) and then use \(<\) and \(>\)
to move the cursor.

\[
\begin{align*}
y'1&=y1+0.1\cdot y1\cdot y2 \\
y'2&=3\cdot y2-y1\cdot y2 \\
\end{align*}
\]

with \(y1(0)=2\) and \(y2(0)=5\)
\[
euler\left\{\begin{array}{c}
ym1+0.1\cdot y1\cdot y2 \\
3\cdot y2-y1\cdot y2 \\
\end{array} , \{y1,y2\} , \{0.5\} , \{2.5\} , 1 \right\}
\]
\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
2 & 1 & 1 & 3.27 & 243.
\end{array}
\]
\[
5 & 10 & 30 & 90 & 90 & -2070.
\]
**euler()**

SystemOfExpr is the system of right-hand sides that define the system of ODEs (corresponds to order of dependent variables in ListOfDepVars).

ListOfExpr is a list of right-hand sides that define the system of ODEs (corresponds to the order of dependent variables in ListOfDepVars).

Var is the independent variable.

ListOfDepVars is a list of dependent variables.

\{Var0, VarMax\} is a two-element list that tells the function to integrate from Var0 to VarMax.

ListOfDepVars0 is a list of initial values for dependent variables.

VarStep is a nonzero number such that \(\text{sign}(\text{VarStep}) = \text{sign}(\text{VarMax}-\text{Var0})\) and solutions are returned at \(\text{Var0}+i\cdot\text{VarStep}\) for all \(i=0,1,2,...\) such that \(\text{Var0}+i\cdot\text{VarStep}\) is in \([\text{var0}, \text{VarMax}]\) (there may not be a solution value at VarMax).

eulerStep is a positive integer (defaults to 1) that defines the number of euler steps between output values. The actual step size used by the euler method is \(\text{VarStep} / \text{eulerStep}\).

---

**Exit**

Exit

Exits the current For, While, or Loop block.

Exit is not allowed outside the three looping structures (For, While, or Loop).

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing \(\text{[ \_ ]}\) instead of \(\text{[ enter]}\) at the end of each line. On the computer keyboard, hold down Alt and press Enter.
\textbf{exp()} \hspace{10cm} \textbf{\texttt{e} key}

\begin{align*}
\text{exp(Value1)} & \Rightarrow \text{value} \\
\text{Returns } e \text{ raised to the Value1 power.}
\end{align*}

\begin{tabular}{|c|c|}
\hline
\text{e}^1 & 2.71828 \\
\text{e}^2 & 8103.08 \\
\hline
\end{tabular}

\textbf{Note:} See also \texttt{e} exponent template, page 6.

You can enter a complex number in re^{\theta} polar form. However, use this form in Radian angle mode only; it causes a Domain error in Degree or Gradian angle mode.

\text{exp(List1)} \Rightarrow \text{list}

\text{Returns } e \text{ raised to the power of each element in List1.}

\text{exp(squareMatrix1)} \Rightarrow \text{squareMatrix}

\text{Returns the matrix exponential of squareMatrix1.}

This is not the same as calculating \texttt{e} raised to the power of each element. For information about the calculation method, refer to \texttt{cos()}.

\textit{squareMatrix1} must be diagonalizable. The result always contains floating-point numbers.

\textbf{expr()} \hspace{10cm} \textbf{Catalog >}

\textbf{expr(}String\textbf{)} \Rightarrow \text{expression}

\text{Returns the character string contained in String as an expression and immediately executes it.}

\begin{itemize}
\item \texttt{"Define cube(x)=x^3" } \Rightarrow \text{funcstr}
\item \texttt{"Define cube(x)=x^3" } \Rightarrow \text{funcstr}
\end{itemize}

\begin{tabular}{|c|c|c|c|}
\hline
1 & 5 & 3 & 782.209 \\
4 & 2 & 1 & 680.546 \\
6 & -2 & 1 & 524.929 \\
\hline
\end{tabular}

\texttt{cube(2)} \Rightarrow 8

\textbf{ExpReg} \hspace{10cm} \textbf{Catalog >}

\textbf{ExpReg} \, X, \, Y \, [\, \texttt{Freq} \, [\, \texttt{Category}, \, \texttt{Include}]\,]

\text{Computes the exponential regression } y = a \cdot (b)^x \text{ on lists } X \text{ and } Y

\text{with frequency } Freq. \text{ A summary of results is stored in the stat.results variable. (See page 131.)}

\text{All the lists must have equal dimension except for } \texttt{Include}.

\text{X and Y are lists of independent and dependent variables.}

\text{Freq} \text{ is an optional list of frequency values. Each element in } Freq

\text{specifies the frequency of occurrence for each corresponding } X
and \( Y \) data point. The default value is 1. All elements must be integers \( \geq 0 \).

*Category* is a list of numeric or string category codes for the corresponding \( X \) and \( Y \) data.

*Include* is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression equation: ( a \cdot (b) \cdot x )</td>
</tr>
<tr>
<td>stat.a, stat.b</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.r²</td>
<td>Coefficient of linear determination for transformed data</td>
</tr>
<tr>
<td>stat.r</td>
<td>Correlation coefficient for transformed data ( x, \ln(y) )</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals associated with the exponential model</td>
</tr>
<tr>
<td>stat.ResidTrans</td>
<td>Residuals associated with linear fit of transformed data</td>
</tr>
<tr>
<td>stat.XReg</td>
<td>List of data points in the modified ( X ) List actually used in the regression based on restrictions of ( Freq, Category ) List, and Include Categories</td>
</tr>
<tr>
<td>stat.YReg</td>
<td>List of data points in the modified ( Y ) List actually used in the regression based on restrictions of ( Freq, Category ) List, and Include Categories</td>
</tr>
<tr>
<td>stat.FreqReg</td>
<td>List of frequencies corresponding to ( stat.XReg ) and ( stat.YReg )</td>
</tr>
</tbody>
</table>

\[ \text{factor()} \]

*factor(rationalNumber)* returns the rational number factored into primes. For composite numbers, the computing time grows exponentially with the number of digits in the second-largest factor. For example, factoring a 30-digit integer could take more than a day, and factoring a 100-digit number could take more than a century.

To stop a calculation manually,
factor()

• Windows®: Hold down the F12 key and press Enter repeatedly.
• Macintosh®: Hold down the F5 key and press Enter repeatedly.
• Handheld: Hold down the @ key and press enter repeatedly.

If you merely want to determine if a number is prime, use isPrime() instead. It is much faster, particularly if rationalNumber is not prime and if the second-largest factor has more than five digits.

FCdf()

\[ FCdf(\text{lowBound}, \text{upBound}, \text{dfNumer}, \text{dfDenom}) \Rightarrow \text{number} \]

If \( \text{lowBound} \) and \( \text{upBound} \) are numbers, \( \text{list} \) if \( \text{lowBound} \) and \( \text{upBound} \) are lists

Computes the F distribution probability between \( \text{lowBound} \) and \( \text{upBound} \) for the specified \( \text{dfNumer} \) (degrees of freedom) and \( \text{dfDenom} \).

For \( P(X \leq \text{upBound}) \), set \( \text{lowBound} = 0 \).

Fill

\[ \text{Fill} \ Value, \ matrixVar \Rightarrow \ matrix \]

Replaces each element in variable \( \text{matrixVar} \) with \( \text{Value} \). \( \text{matrixVar} \) must already exist.

\[ \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \rightarrow \text{amatrix} \quad \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \]

Fill 1.01,\text{amatrix} Done

\[ \begin{array}{c} \text{amatrix} \\ 1.01 \end{array} \rightarrow \begin{array}{c} 1.01 \\ 1.01 \end{array} \]

\[ \begin{bmatrix} 1,2,3,4,5 \end{bmatrix} \rightarrow \text{alist} \quad \begin{bmatrix} 1,2,3,4,5 \end{bmatrix} \]

Fill 1.01,\text{alist} Done

\[ \text{alist} \rightarrow \begin{bmatrix} 1.01,1.01,1.01,1.01,1.01 \end{bmatrix} \]

50 Alphabetical Listing
**FiveNumSummary**

**FiveNumSummary X, [Freq], [Category, Include]**

Provides an abbreviated version of the 1-variable statistics on list \( X \). A summary of results is stored in the `stat.results` variable.

(See page 131.)

\( X \) represents a list containing the data.

\( Freq \) is an optional list of frequency values. Each element in \( Freq \) specifies the frequency of occurrence for each corresponding \( X \) and \( Y \) data point. The default value is 1.

\( Category \) is a list of numeric category codes for the corresponding \( X \) data.

\( Include \) is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

An empty (void) element in any of the lists \( X, Freq, \) or \( Category \) results in a void for the corresponding element of all those lists.

For more information on empty elements, see page 177.

---

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stat.MinX</code></td>
<td>Minimum of ( x ) values.</td>
</tr>
<tr>
<td><code>stat.Q_1X</code></td>
<td>1st Quartile of ( x ).</td>
</tr>
<tr>
<td><code>stat.MedianX</code></td>
<td>Median of ( x ).</td>
</tr>
<tr>
<td><code>stat.Q_3X</code></td>
<td>3rd Quartile of ( x ).</td>
</tr>
<tr>
<td><code>stat.MaxX</code></td>
<td>Maximum of ( x ) values.</td>
</tr>
</tbody>
</table>

---

**floor()**

\[ \text{floor}(Value) \Rightarrow \text{integer} \]

Returns the greatest integer that is \( \leq \) the argument. This function is identical to `int()`.

The argument can be a real or a complex number.

\[ \text{floor}(List) \Rightarrow \text{list} \]
\[ \text{floor}(Matrix) \Rightarrow \text{matrix} \]

Returns a list or matrix of the floor of each element.

**Note:** See also `ceiling()` and `int()`.

---

Alphabetical Listing 51
For

For Var, Low, High [, Step]
   Block
EndFor

Executes the statements in Block iteratively for each value of Var, from Low to High, in increments of Step.

Var must not be a system variable.

Step can be positive or negative. The default value is 1.

Block can be either a single statement or a series of statements separated with the "::" character.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing ▼ instead of [enter] at the end of each line. On the computer keyboard, hold down Alt and press Enter.

format()

format(Value[, formatString]) ⇒ string

Returns Value as a character string based on the format template.

formatString is a string and must be in the form: "F[n]", "S[n]", "E[n]", "G[n][c]", where [] indicate optional portions.

F[n]: Fixed format. n is the number of digits to display after the decimal point.

S[n]: Scientific format. n is the number of digits to display after the decimal point.

E[n]: Engineering format. n is the number of digits after the first significant digit. The exponent is adjusted to a multiple of three, and the decimal point is moved to the right by zero, one, or two digits.

G[n][c]: Same as fixed format but also separates digits to the left of the radix into groups of three. c specifies the group separator character and defaults to a comma. If c is a period, the radix will be shown as a comma.

Define g()=Func
   Local tempsum, step, i
   0 → tempsum
   1 → step
   For i,1,100,step
      tempsum+i → tempsum
   EndFor
   EndFunc

<table>
<thead>
<tr>
<th>format(1.234567, &quot;f3&quot;)</th>
<th>&quot;1.235&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>format(1.234567, &quot;s2&quot;)</td>
<td>&quot;1.23e0&quot;</td>
</tr>
<tr>
<td>format(1.234567, &quot;e3&quot;)</td>
<td>&quot;1.235e0&quot;</td>
</tr>
<tr>
<td>format(1.234567, &quot;g3&quot;)</td>
<td>&quot;1.235&quot;</td>
</tr>
<tr>
<td>format(1234.567, &quot;g3&quot;)</td>
<td>&quot;1,234.567&quot;</td>
</tr>
<tr>
<td>format(1.234567, &quot;g3,r;&quot;)</td>
<td>&quot;1:235&quot;</td>
</tr>
</tbody>
</table>
format() [Rc]: Any of the above specifiers may be suffixed with the Rc radix flag, where c is a single character that specifies what to substitute for the radix point.

fPart() fPart(Expr1) ⇒ expression
fPart(List1) ⇒ list
fPart(Matrix1) ⇒ matrix

Returns the fractional part of the argument.

For a list or matrix, returns the fractional parts of the elements.

The argument can be a real or a complex number.

F'Pdf() F’Pdf(XVal,dfNumer,dfDenom) ⇒ number if XVal is a number,
list if XVal is a list

Computes the F' distribution probability at XVal for the specified dfNumer (degrees of freedom) and dfDenom.

freqTable►list() freqTable►list(List1,freqIntegerList) ⇒ list

Returns a list containing the elements from List1 expanded according to the frequencies in freqIntegerList. This function can be used for building a frequency table for the Data & Statistics application.

List1 can be any valid list.

freqIntegerList must have the same dimension as List1 and must contain non-negative integer elements only. Each element specifies the number of times the corresponding List1 element will be repeated in the result list. A value of zero excludes the corresponding List1 element.

Note: You can insert this function from the computer keyboard by typing freqTable►list(…).
Empty (void) elements are ignored. For more information on empty elements, see page 177.

Returns a list containing counts of the elements in List1. The counts are based on ranges (bins) that you define in binsList.

If binsList is \{b(1), b(2), ..., b(n)\}, the specified ranges are \(? \leq b(1), b(1) < ? \leq b(2), ..., b(n-1) < ? \leq b(n), b(n) > ?\}. The resulting list is one element longer than binsList.

Each element of the result corresponds to the number of elements from List1 that are in the range of that bin. Expressed in terms of the countIf() function, the result is \{ countIf(list, ? \leq b(1)), countIf(list, b(1) < ? \leq b(2)), ..., countIf(list, b(n-1) < ? \leq b(n)), countIf(list, b(n) > ?) \}.

Elements of List1 that cannot be "placed in a bin" are ignored. Empty (void) elements are also ignored. For more information on empty elements, see page 177.

Within the Lists & Spreadsheet application, you can use a range of cells in place of both arguments.

Note: See also countIf(), page 32.

Performs a two-sample F test. A summary of results is stored in the stat.results variable. (See page 131.)
For $H_a: \sigma_1 > \sigma_2$, set $Hypothesis > 0$
For $H_a: \sigma_1 \neq \sigma_2$ (default), set $Hypothesis = 0$
For $H_a: \sigma_1 < \sigma_2$, set $Hypothesis < 0$

For information on the effect of empty elements in a list, see Empty (Void) Elements, page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.F</td>
<td>Calculated $F$ statistic for the data sequence</td>
</tr>
<tr>
<td>stat.PVal</td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>stat.dfNumer</td>
<td>Numerator degrees of freedom = $n_1-1$</td>
</tr>
<tr>
<td>stat.dfDenom</td>
<td>Denominator degrees of freedom = $n_2-1$</td>
</tr>
<tr>
<td>stat.sx1, stat.sx2</td>
<td>Sample standard deviations of the data sequences in List 1 and List 2</td>
</tr>
<tr>
<td>stat.x1_bar, stat.x2_bar</td>
<td>Sample means of the data sequences in List 1 and List 2</td>
</tr>
<tr>
<td>stat.n1, stat.n2</td>
<td>Size of the samples</td>
</tr>
</tbody>
</table>

Define a piecewise function:

```
Define g(x)=Func
    If x<0 Then
        Return 3*cos(x)
    Else
        Return 3-x
    EndIf
EndFunc
```

Result of graphing $g(x)$

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing [ ] instead of [ ] at the end of each line. On the computer keyboard, hold down Alt and press Enter.
**gcd()**

**gcd(Number1, Number2) ⇒ expression**

Returns the greatest common divisor of the two arguments. The gcd of two fractions is the gcd of their numerators divided by the lcm of their denominators.

In Auto or Approximate mode, the gcd of fractional floating-point numbers is 1.0.

**gcd(List1, List2) ⇒ list**

Returns the greatest common divisors of the corresponding elements in List1 and List2.

**gcd(Matrix1, Matrix2) ⇒ matrix**

Returns the greatest common divisors of the corresponding elements in Matrix1 and Matrix2.

**geomCdf()**

**geomCdf(p,lowBound,upBound) ⇒ number if lowBound and upBound are numbers, list if lowBound and upBound are lists**

**geomCdf(p,upBound) for P(1≤X≤upBound) ⇒ number if upBound is a number, list if upBound is a list**

Computes a cumulative geometric probability from lowBound to upBound with the specified probability of success p.

For P(X≤upBound), set lowBound = 1.

**geomPdf()**

**geomPdf(p,XVal) ⇒ number if XVal is a number, list if XVal is a list**

Computes a probability at XVal, the number of the trial on which the first success occurs, for the discrete geometric distribution with the specified probability of success p.
**getDenom()**

**getDenom(Fraction1) ⇒ value**

Transforms the argument into an expression having a reduced common denominator, and then returns its denominator.

<table>
<thead>
<tr>
<th>Fraction1</th>
<th>getDenom(Fraction1)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[\frac{x+2}{y-3}]</td>
<td>getDenom</td>
<td>3</td>
</tr>
<tr>
<td>[\frac{2}{7}]</td>
<td>getDenom</td>
<td>7</td>
</tr>
<tr>
<td>[\frac{1}{x} + \frac{y^2+y}{y^2}]</td>
<td>getDenom</td>
<td>30</td>
</tr>
</tbody>
</table>

**getLangInfo()**

**getLangInfo() ⇒ string**

Returns a string that corresponds to the short name of the currently active language. You can, for example, use it in a program or function to determine the current language.

- English = "en"
- Danish = "da"
- German = "de"
- Finnish = "fi"
- French = "fr"
- Italian = "it"
- Dutch = "nl"
- Belgian Dutch = "nl_BE"
- Norwegian = "no"
- Portuguese = "pt"
- Spanish = "es"
- Swedish = "sv"

**getLockInfo()**

**getLockInfo(Var) ⇒ value**

Returns the current locked/unlocked state of variable Var.

- value = 0: Var is unlocked or does not exist.
- value = 1: Var is locked and cannot be modified or deleted.

See **Lock**, page 76, and **unLock**, page 147.
getMode() returns a value representing the current setting of the ModeNameInteger mode.

getMode(0) returns a list containing number pairs. Each pair consists of a mode integer and a setting integer.

For a listing of the modes and their settings, refer to the table below.

If you save the settings with getMode(0) → var, you can use setMode(var) in a function or program to temporarily restore the settings within the execution of the function or program only. See setMode(), page 122.

<table>
<thead>
<tr>
<th>Mode Name</th>
<th>Mode Integer</th>
<th>Setting Integers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Digits</td>
<td>1</td>
<td>1=Float, 2=Float1, 3=Float2, 4=Float3, 5=Float4, 6=Float5, 7=Float6, 8=Float7, 9=Float8, 10=Float9, 11=Float10, 12=Float11, 13=Float12, 14=Fix0, 15=Fix1, 16=Fix2, 17=Fix3, 18=Fix4, 19=Fix5, 20=Fix6, 21=Fix7, 22=Fix8, 23=Fix9, 24=Fix10, 25=Fix11, 26=Fix12</td>
</tr>
<tr>
<td>Angle</td>
<td>2</td>
<td>1=Radian, 2=Degree, 3=Gradian</td>
</tr>
<tr>
<td>Exponential Format</td>
<td>3</td>
<td>1=Normal, 2=Scientific, 3=Engineering</td>
</tr>
<tr>
<td>Real or Complex</td>
<td>4</td>
<td>1=Real, 2=Rectangular, 3=Polar</td>
</tr>
<tr>
<td>Auto or Approx.</td>
<td>5</td>
<td>1=Auto, 2=Approximate</td>
</tr>
<tr>
<td>Vector Format</td>
<td>6</td>
<td>1=Rectangular, 2=Cylindrical, 3=Spherical</td>
</tr>
<tr>
<td>Base</td>
<td>7</td>
<td>1=Decimal, 2=Hex, 3=Binary</td>
</tr>
</tbody>
</table>
getNum

\textbf{getNum}(\textit{FractionI}) \Rightarrow value

Transforms the argument into an expression having a reduced common denominator, and then returns its numerator.

\begin{align*}
x & = 5, \quad y = 6 & 6 \\
ggetNum \left( \frac{x+2}{y-3} \right) & = 7 \\
ggetNum \left( \frac{2}{7} \right) & = 2 \\
ggetNum \left( \frac{1}{x}, \frac{1}{y} \right) & = 11
\end{align*}

ggetType

\textbf{getType}(\textit{var}) \Rightarrow string

Returns a string that indicates the data type of variable \textit{var}.

If \textit{var} has not been defined, returns the string "NONE".

\begin{align*}
\{1,2,3\} & \rightarrow \text{temp} & \{1,2,3\} \\
ggetType(\text{temp}) & = \text{"LIST"} \\
3 \cdot i & \rightarrow \text{temp} & 3 \cdot i \\
ggetType(\text{temp}) & = \text{"EXPR"} \\
\text{DelVar} \text{ temp} & \quad \text{Done} \\
ggetType(\text{temp}) & = \text{"NONE"}
\end{align*}

ggetVarInfo

\textbf{getVarInfo()} \Rightarrow matrix or string

\textbf{getVarInfo}([LibNameString]) \Rightarrow matrix or string

\textbf{getVarInfo}() returns a matrix of information (variable name, type, library accessibility, and locked/unlocked state) for all variables and library objects defined in the current problem.

If no variables are defined, \textbf{getVarInfo}() returns the string "NONE".

\textbf{getVarInfo}([LibNameString]) returns a matrix of information for all library objects defined in library \textit{LibNameString}. \textit{LibNameString} must be a string (text enclosed in quotation marks) or a string variable.

If the library \textit{LibNameString} does not exist, an error occurs.
### getVarInfo()

Note the example, in which the result of `getVarInfo()` is assigned to variable `vs`. Attempting to display row 2 or row 3 of `vs` returns an "Invalid list or matrix" error because at least one of elements in those rows (variable `b`, for example) reevaluates to a matrix.

This error could also occur when using `Ans` to reevaluate a `getVarInfo()` result.

The system gives the above error because the current version of the software does not support a generalized matrix structure where an element of a matrix can be either a matrix or a list.

```
\[
\begin{align*}
a &:= 1 \\
b &:= \begin{bmatrix} 1 & 2 \\ 1 & 3 \\ 1 & 7 \end{bmatrix} \\
c &:= \begin{bmatrix} 1 & 3 & 7 \\ 1 & 3 & 7 \end{bmatrix} \\
\text{vs} &:= \text{getVarInfo()} \\
\text{vs}[1] &:= \begin{bmatrix} a & \text{"NUM"} & \text{""} \end{bmatrix} \\
\text{vs}[2] &:= \begin{bmatrix} b & \text{"MAT"} & \text{""} \\ c & \text{"MAT"} & \text{""} \end{bmatrix} \\
\text{vs}[1,1] &:= 1 \\
\text{vs}[2] &:= \text{"Error: Invalid list or matrix"} \\
\text{vs}[2,1] &:= \begin{bmatrix} 2 & \text{""} \end{bmatrix}
\end{align*}
\]
```

### Goto

**Goto** `labelName`

Transfers control to the label `labelName`.

`labelName` must be defined in the same function using a `Lbl` instruction.

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing ` chemotherapy` instead of `enter` at the end of each line. On the computer keyboard, hold down `Alt` and press `Enter`.

```
\[
\text{Define } g() := \text{Func} \\
\quad \text{Local } temp, i \\
\quad 0 \rightarrow temp \\
\quad 1 \rightarrow i \\
\quad \text{Lbl top} \\
\quad \text{temp+i} \rightarrow temp \\
\quad \text{If } i < 10 \text{ Then} \\
\quad \text{i+1} \rightarrow i \\
\quad \text{Goto top} \\
\quad \text{EndIf} \\
\quad \text{Return } temp \\
\quad \text{EndFunc}
\]
```

### ▶ Grad

**Expr▶Grad ⇒ expression**

Converts `Expr1` to gradian angle measure.

**Note:** You can insert this operator from the computer keyboard by typing `@>Grad`.

In Degree angle mode:

```
\[
1.5 ▶ \text{Grad} \quad \begin{bmatrix} 1.66667^\circ \end{bmatrix}
\]
```

In Radian angle mode:

```
\[
1.5 ▶ \text{Grad} \quad \begin{bmatrix} 95.493^\circ \end{bmatrix}
\]
```
identity()

<table>
<thead>
<tr>
<th>identity(Integer) ⇒ matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns the identity matrix with a dimension of Integer.</td>
</tr>
<tr>
<td>Integer must be a positive integer.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>identity(4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0</td>
</tr>
<tr>
<td>0 1 0 0</td>
</tr>
<tr>
<td>0 0 1 0</td>
</tr>
<tr>
<td>0 0 0 1</td>
</tr>
</tbody>
</table>

If

<table>
<thead>
<tr>
<th>If BooleanExpr Then Block EndIf</th>
</tr>
</thead>
<tbody>
<tr>
<td>If BooleanExpr evaluates to true, executes the single statement Statement or the block of statements Block before continuing execution.</td>
</tr>
<tr>
<td>If BooleanExpr evaluates to false, continues execution without executing the statement or block of statements.</td>
</tr>
<tr>
<td>Block can be either a single statement or a sequence of statements separated with the &quot;::&quot; character.</td>
</tr>
<tr>
<td><strong>Note for entering the example:</strong> In the Calculator application on the handheld, you can enter multi-line definitions by pressing [ \text{ • } ] instead of [ \text{ enter } ] at the end of each line. On the computer keyboard, hold down Alt and press Enter.</td>
</tr>
<tr>
<td>If BooleanExpr Then Block1 Else Block2 EndIf</td>
</tr>
<tr>
<td>If BooleanExpr evaluates to true, executes Block1 and then skips Block2.</td>
</tr>
<tr>
<td>If BooleanExpr evaluates to false, skips Block1 but executes Block2.</td>
</tr>
<tr>
<td>Block1 and Block2 can be a single statement.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Define g(x)=Func</th>
</tr>
</thead>
<tbody>
<tr>
<td>If x&lt;0 Then Return x^2 EndIf</td>
</tr>
<tr>
<td>g(-2) 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Define g(x)=Func</th>
</tr>
</thead>
<tbody>
<tr>
<td>If x&lt;0 Then Return x Else Return x EndIf</td>
</tr>
<tr>
<td>g(12) 12</td>
</tr>
<tr>
<td>g(-12) 12</td>
</tr>
</tbody>
</table>
If

If BooleanExpr1 Then
   Block1
ElseIf BooleanExpr2 Then
   Block2
ElseIf BooleanExprN Then
   BlockN
EndIf

Allows for branching. If BooleanExpr1 evaluates to true, executes Block1. If BooleanExpr1 evaluates to false, evaluates BooleanExpr2, and so on.

Define \( g(x) = \) Func
   If \( x < 5 \) Then
      Return 5
   ElseIf \( x > -5 \) and \( x < 0 \) Then
      Return \( -x \)
   ElseIf \( x \geq 0 \) and \( x = 10 \) Then
      Return \( x \)
   ElseIf \( x = 10 \) Then
      Return 3
   EndIf
EndFunc

| \( g(-4) \) | 4 |
| \( g(10) \) | 3 |

**ifFn()**

\( \text{ifFn}(\text{BooleanExpr, Value\_If\_true [,Value\_If\_false [ ,Value\_If\_unknown]]}) \Rightarrow \text{expression, list, or matrix} \)

Evaluates the boolean expression \( \text{BooleanExpr} \) (or each element from \( \text{BooleanExpr} \)) and produces a result based on the following rules:

- \( \text{BooleanExpr} \) can test a single value, a list, or a matrix.
- If an element of \( \text{BooleanExpr} \) evaluates to true, returns the corresponding element from \( \text{Value\_If\_true} \).
- If an element of \( \text{BooleanExpr} \) evaluates to false, returns the corresponding element from \( \text{Value\_If\_false} \). If you omit \( \text{Value\_If\_false} \), returns undef.
- If an element of \( \text{BooleanExpr} \) is neither true nor false, returns the corresponding element \( \text{Value\_If\_unknown} \). If you omit \( \text{Value\_If\_unknown} \), returns undef.
- If the second, third, or fourth argument of the \( \text{ifFn()} \) function is a single expression, the Boolean test is applied to every position in \( \text{BooleanExpr} \).

**Note:** If the simplified \( \text{BooleanExpr} \) statement involves a list or matrix, all other list or matrix

\( \text{ifFn}([1,2,3] < 2.5, [5,6,7], [8,9,10]) \)
   \[ \{5,6,10\} \]

Test value of 1 is less than 2.5, so its corresponding \( \text{Value\_If\_True} \) element of 5 is copied to the result list.

Test value of 2 is less than 2.5, so its corresponding \( \text{Value\_If\_True} \) element of 6 is copied to the result list.

Test value of 3 is not less than 2.5, so its corresponding \( \text{Value\_If\_False} \) element of 10 is copied to the result list.

\( \text{ifFn}([1,2,3] < 2.5, [4,8,10]) \)
   \[ \{4,4,10\} \]

\( \text{Value\_If\_true} \) is a single value and corresponds to any selected position.

\( \text{ifFn}([1,2,3] < 2.5, [5,6,7]) \)
   \[ \{5,6,\text{undef}\} \]

\( \text{Value\_If\_false} \) is not specified. Undefined is used.
ifFn() arguments must have the same dimension(s), and the result will have the same dimension(s).

\[
\text{ifFn}\left\{\{2, "a"\}, \{2.5, 6, 7\}, \{9, 10\}, "err"\}\right\}
\]

One element selected from Value\_If\_true. One element selected from Value\_If\_unknown.

\[
\text{imag}\left(\begin{array}{cc}
1 & 2 \\
3 & 4 \\
\end{array}\right)
\]

\[
\text{imag}\left(\begin{array}{c}
1+2\cdot i \\
\end{array}\right)
\]

\[
\text{imag}\left(\begin{array}{c}
0, 1, i \\
\end{array}\right)
\]

\[
\text{imag}\left(\begin{array}{c}
0 \ 0 \\
3 \ 4 \\
\end{array}\right)
\]

\[
\text{imag}(Value) \Rightarrow value
\]

Returns the imaginary part of the argument.

\[
\text{imag}(List) \Rightarrow list
\]

Returns a list of the imaginary parts of the elements.

\[
\text{imag}(Matrix) \Rightarrow matrix
\]

Returns a matrix of the imaginary parts of the elements.

Indirection See #(), page 170.

\[
\text{inString}(srcString, subString, \text{Start}) \Rightarrow integer
\]

Returns the character position in string srcString at which the first occurrence of string subString begins.

Start, if included, specifies the character position within srcString where the search begins. Default = 1 (the first character of srcString).

If srcString does not contain subString or Start is > the length of srcString, returns zero.

\[
\text{int}(Value) \Rightarrow integer
\]

\[
\text{int}(List) \Rightarrow list
\]

\[
\text{int}(Matrix) \Rightarrow matrix
\]
int()  

Returns the greatest integer that is less than or equal to the argument. This function is identical to \texttt{floor()}. The argument can be a real or a complex number.

For a list or matrix, returns the greatest integer of each of the elements.

\[
\begin{array}{ll}
\text{intDiv()} & \text{Catalog >}\ \\
\text{intDiv(Number1, Number2) ⇒ integer} & \text{intDiv(-7, 2)} \quad -3 \\
\text{intDiv(List1, List2) ⇒ list} & \text{intDiv(4, 5)} \quad 0 \\
\text{intDiv(Matrix1, Matrix2) ⇒ matrix} & \text{intDiv}\{12, -14, -16\}, \{5, 4, 3\}\} \quad \{2, 3, 5\}
\end{array}
\]

Returns the signed integer part of \((Number1 \div Number2)\).

For lists and matrices, returns the signed integer part of \((\text{argument 1} \div \text{argument 2})\) for each element pair.

\[
\begin{array}{ll}
\text{interpolate()} & \text{Catalog >}\ \\
\text{interpolate(xValue, xList, yList, yPrimeList) ⇒ list} & \text{Differential equation:} \\
& y'= -3y + 6t + 5 \text{ and } y(0) = 5 \\
& \text{rk} = \text{rk2}\left[ -3y + 6t + 5, y, \{0, 10\}, 5.1 \right] \\
& \begin{bmatrix} 0, & 1, & 2, & 3, & 4, \end{bmatrix} \\
& \begin{bmatrix} 5, & 3.19499, & 5.00394, & 6.99957, & 9.00593, \end{bmatrix} \\
& \text{To see the entire result, press ▲ and then use ◄ and ► to move the cursor.}
\end{array}
\]

Use the \texttt{interpolate()} function to calculate the function values for the xvalue list:

\[
\begin{array}{ll}
xvalue list: & \text{seq}(i, 0, 10, 0.5) \\
xlist: & \text{mat}^\text{list}(\text{rk}[1]) \\
ylist: & \text{mat}^\text{list}(\text{rk}[2]) \\
yPrime list: & \text{3y} + 6t + 5y - \text{ylist} \text{ and } t - \text{xlist} \\
\end{array}
\]

\[
\begin{array}{ll}
xvalue list: & \{0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 6.5\} \\
xlist: & \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\} \\
ylist: & \{5.0, 3.19499, 5.00394, 6.99957, 9.00593, 10.997\} \\
yPrime list: & \{-3y + 6t + 5, y - xlist\} \\
\end{array}
\]

\texttt{interpolate(xvalue list, xlist, ylist, yPrime list)}
invχ²

invχ²(Area, df)

invChi2(Area, df)

Computes the Inverse cumulative χ² (chi-square) probability function specified by degree of freedom, df for a given Area under the curve.

invF

invF(Area, dfNumer, dfDenom)

invF(Area, dfNumer, dfDenom)

computes the Inverse cumulative F distribution function specified by dfNumer and dfDenom for a given Area under the curve.

invNorm

invNorm(Area, μ, σ)

Computes the inverse cumulative normal distribution function for a given Area under the normal distribution curve specified by μ and σ.

invt

invt(Area, df)

Computes the inverse cumulative student-t probability function specified by degree of freedom, df for a given Area under the curve.

iPart

iPart(Number) \Rightarrow integer
iPart(List1) \Rightarrow list
iPart(Matrix1) \Rightarrow matrix

Returns the integer part of the argument.

For lists and matrices, returns the integer part of each element.

The argument can be a real or a complex number.
Function to find the internal rate of return of an investment.

\[ \text{irr}(CF0, CFList[, CFFreq]) \Rightarrow \text{value} \]

Financial function that calculates internal rate of return of an investment.

*CF0* is the initial cash flow at time 0; it must be a real number.

*CFList* is a list of cash flow amounts after the initial cash flow CF0.

*CFFreq* is an optional list in which each element specifies the frequency of occurrence for a grouped (consecutive) cash flow amount, which is the corresponding element of *CFList*. The default is 1; if you enter values, they must be positive integers < 10,000.

**Note:** See also \( \text{mirr}() \), page 84.

### isPrime()

\[ \text{isPrime}(\text{Number}) \Rightarrow \text{Boolean constant expression} \]

Returns true or false to indicate if *number* is a whole number \( \geq 2 \) that is evenly divisible only by itself and 1.

If *Number* exceeds about 306 digits and has no factors \( \leq 1021 \), \( \text{isPrime}(\text{Number}) \) displays an error message.

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing \( \text{Alt} \) instead of \( \text{Enter} \) at the end of each line. On the computer keyboard, hold down Alt and press Enter.

### isVoid()

\[ \text{isVoid}(\text{Var}) \Rightarrow \text{Boolean constant expression} \]
\[ \text{isVoid}(\text{Expr}) \Rightarrow \text{Boolean constant expression} \]
\[ \text{isVoid}(\text{List}) \Rightarrow \text{list of Boolean constant expressions} \]

Returns true or false to indicate if the argument is a void data type.

For more information on void elements, see page 177.
**Lbl**

**Lbl** `labelName`

Defines a label with the name `labelName` within a function.

You can use a `Goto labelName` instruction to transfer control to the instruction immediately following the label.

`labelName` must meet the same naming requirements as a variable name.

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing `[·]` instead of `·` at the end of each line. On the computer keyboard, hold down `Alt` and press `Enter`.

```
Define g()=Func
   Local temp,i
   0 -> temp
   1 -> i
  Lbl top
   temp+i -> temp
   If i<10 Then
      i+1 -> i
   Goto top
   EndIf
   Return temp
EndFunc
```

**lcm()**

`lcm(Number1, Number2) ⇒ expression`

`lcm(List1, List2) ⇒ list`

`lcm(Matrix1, Matrix2) ⇒ matrix`

Returns the least common multiple of the two arguments. The `lcm` of two fractions is the `lcm` of their numerators divided by the `gcd` of their denominators. The `lcm` of fractional floating-point numbers is their product.

For two lists or matrices, returns the least common multiples of the corresponding elements.

| `lcm(6,9)` | 18 |
| `lcm(\begin{bmatrix} 1/3 & 14,16 \\ \frac{2}{15} & 7,5 \end{bmatrix})` | \( \begin{bmatrix} \frac{2}{3} & 14,80 \end{bmatrix} \) |

**left()**

`left(sourceString[, Num]) ⇒ string`

Returns the leftmost `Num` characters contained in character string `sourceString`.

If you omit `Num`, returns all of `sourceString`.

`left(List1[, Num]) ⇒ list`

| `left("Hello",2)` | "He" |
| `left(\{1,3,2,4\},3)` | \{1,3,2\} |
left()  

Returns the leftmost Num elements contained in List1.

If you omit Num, returns all of List1.

left(Comparison) \rightarrow expression

Returns the left-hand side of an equation or inequality.

libShortcut()

libShortcut(LibNameString, ShortcutNameString [, LibPrivFlag]) \rightarrow list of variables

Creates a variable group in the current problem that contains references to all the objects in the specified library document LibNameString. Also adds the group members to the Variables menu. You can then refer to each object using its ShortcutNameString.

Set LibPrivFlag=0 to exclude private library objects (default)
Set LibPrivFlag=1 to include private library objects

To copy a variable group, see CopyVar on page 27. To delete a variable group, see DelVar on page 40.

LinRegBx

LinRegBx X,Y[,Freq][,Category,Include]

Computes the linear regression \( y = a + b \cdot x \) on lists \( X \) and \( Y \) with frequency \( Freq \). A summary of results is stored in the stat.results variable. (See page 131.)

All the lists must have equal dimension except for Include.

\( X \) and \( Y \) are lists of independent and dependent variables.

Freq is an optional list of frequency values. Each element in Freq specifies the frequency of occurrence for each corresponding \( X \) and \( Y \) data point. The default value is 1. All elements must be integers \( \geq 0 \).

Category is a list of numeric or string category codes for the corresponding \( X \) and \( Y \) data.

Include is a list of one or more of the category codes. Only those
data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression Equation: $a+b\cdot x$</td>
</tr>
<tr>
<td>stat.a, stat.b</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.$r^2$</td>
<td>Coefficient of determination</td>
</tr>
<tr>
<td>stat.r</td>
<td>Correlation coefficient</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the regression</td>
</tr>
<tr>
<td>stat.XReg</td>
<td>List of data points in the modified X List actually used in the regression based on restrictions of Freq, Category List, and Include Categories</td>
</tr>
<tr>
<td>stat.YReg</td>
<td>List of data points in the modified Y List actually used in the regression based on restrictions of Freq, Category List, and Include Categories</td>
</tr>
<tr>
<td>stat.FreqReg</td>
<td>List of frequencies corresponding to stat.XReg and stat.YReg</td>
</tr>
</tbody>
</table>

Computes the linear regression $y = mx + b$ on lists $X$ and $Y$ with frequency $Freq$. A summary of results is stored in the `stat.results` variable. (See page 131.)

All the lists must have equal dimension except for `Include`.

$X$ and $Y$ are lists of independent and dependent variables.

`Freq` is an optional list of frequency values. Each element in `Freq` specifies the frequency of occurrence for each corresponding $X$ and $Y$ data point. The default value is 1. All elements must be integers $\geq 0$.

`Category` is a list of numeric or string category codes for the corresponding $X$ and $Y$ data.

`Include` is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.
<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression Equation: ( y = m \cdot x + b )</td>
</tr>
<tr>
<td>stat.m, stat.b</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.r²</td>
<td>Coefficient of determination</td>
</tr>
<tr>
<td>stat.r</td>
<td>Correlation coefficient</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the regression</td>
</tr>
<tr>
<td>stat.XReg</td>
<td>List of data points in the modified ( X ) List actually used in the regression based on restrictions of ( Freq, Category List, and Include Categories )</td>
</tr>
<tr>
<td>stat.YReg</td>
<td>List of data points in the modified ( Y ) List actually used in the regression based on restrictions of ( Freq, Category List, and Include Categories )</td>
</tr>
<tr>
<td>stat.FreqReg</td>
<td>List of frequencies corresponding to ( stat.XReg ) and ( stat.YReg )</td>
</tr>
</tbody>
</table>

**LinRegtIntervals**

For Slope. Computes a level \( C \) confidence interval for the slope.

\[
\text{LinRegtIntervals } X,Y,[F],0,[CLev]]
\]

For Response. Computes a predicted \( y \)-value, a level \( C \) prediction interval for a single observation, and a level \( C \) confidence interval for the mean response.

A summary of results is stored in the \( stat.results \) variable. (See page 131.)

All the lists must have equal dimension.

\( X \) and \( Y \) are lists of independent and dependent variables.

\( F \) is an optional list of frequency values. Each element in \( F \) specifies the frequency of occurrence for each corresponding \( X \) and \( Y \) data point. The default value is 1. All elements must be integers \( \geq 0 \).

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression Equation: ( a + b \cdot x )</td>
</tr>
<tr>
<td>stat.a, stat.b</td>
<td>Regression coefficients</td>
</tr>
</tbody>
</table>
### Output variable | Description
--- | ---
stat.df | Degrees of freedom
stat.r² | Coefficient of determination
stat.r | Correlation coefficient
stat.Resid | Residuals from the regression

For Slope type only

| Output variable | Description |
--- | ---
[stat.CLower, stat.CUpper] | Confidence interval for the slope
stat.ME | Confidence interval margin of error
stat.SESlope | Standard error of slope
stat.s | Standard error about the line

For Response type only

| Output variable | Description |
--- | ---
[stat.CLower, stat.CUpper] | Confidence interval for the mean response
stat.ME | Confidence interval margin of error
stat.SE | Standard error of mean response
[stat.LowerPred, stat.UpperPred] | Prediction interval for a single observation
stat.MEPred | Prediction interval margin of error
stat.SE Pred | Standard error for prediction
stat.y | \( a + b \cdot XVal \)

**LinRegtTest**

**LinRegtTest** \( X, Y[, Freq[, Hypoth]] \)

Computes a linear regression on the \( X \) and \( Y \) lists and a \( t \) test on the value of slope \( \beta \) and the correlation coefficient \( \rho \) for the equation \( y = a + \beta x \). It tests the null hypothesis \( H_0: \beta = 0 \) (equivalently, \( \rho = 0 \)) against one of three alternative hypotheses.

All the lists must have equal dimension.

\( X \) and \( Y \) are lists of independent and dependent variables.

\( Freq \) is an optional list of frequency values. Each element in \( Freq \)
specifies the frequency of occurrence for each corresponding \( X \) and \( Y \) data point. The default value is 1. All elements must be integers \( \geq 0 \).

*Hypoth* is an optional value specifying one of three alternative hypotheses against which the null hypothesis (\( H_0: \beta = \rho = 0 \)) will be tested.

For \( H_a: \beta \neq 0 \) and \( \rho \neq 0 \) (default), set *Hypoth* = 0
For \( H_a: \beta < 0 \) and \( \rho < 0 \), set *Hypoth* < 0
For \( H_a: \beta > 0 \) and \( \rho > 0 \), set *Hypoth* > 0

A summary of results is stored in the *stat.results* variable. (See page 131.)

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression equation: ( a + bx )</td>
</tr>
<tr>
<td>stat.t</td>
<td>( t )-Statistic for significance test</td>
</tr>
<tr>
<td>stat.PVal</td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>stat.df</td>
<td>Degrees of freedom</td>
</tr>
<tr>
<td>stat.a, stat.b</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.s</td>
<td>Standard error about the line</td>
</tr>
<tr>
<td>stat.SESlope</td>
<td>Standard error of slope</td>
</tr>
<tr>
<td>stat.r(^2)</td>
<td>Coefficient of determination</td>
</tr>
<tr>
<td>stat.r</td>
<td>Correlation coefficient</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the regression</td>
</tr>
</tbody>
</table>
linSolve()  

\[
\text{linSolve(SystemOfLinearEqns, Var1, Var2, \ldots)} \Rightarrow \text{list}
\]

\[
\text{linSolve(LinearEqn1 and LinearEqn2 and \ldots, Var1, Var2, \ldots)} \Rightarrow \text{list}
\]

\[
\text{linSolve(\{LinearEqn1, LinearEqn2, \ldots\}, Var1, Var2, \ldots)} \Rightarrow \text{list}
\]

\[
\text{linSolve(SystemOfLinearEqns, \{Var1, Var2, \ldots\})} \Rightarrow \text{list}
\]

\[
\text{linSolve(LinearEqn1 and LinearEqn2 and \ldots, \{Var1, Var2, \ldots\})} \Rightarrow \text{list}
\]

\[
\text{linSolve(\{LinearEqn1, LinearEqn2, \ldots\}, \{Var1, Var2, \ldots\})} \Rightarrow \text{list}
\]

Returns a list of solutions for the variables \(Var1, Var2, \ldots\).

The first argument must evaluate to a system of linear equations or a single linear equation. Otherwise, an argument error occurs.

For example, evaluating \(\text{linSolve(x=1 and x=2, x)}\) produces an "Argument Error" result.

\[\Delta\text{List()}\]

\[
\Delta\text{List(List1)} \Rightarrow \text{list}
\]

**Note:** You can insert this function from the keyboard by typing \(\text{deltaList\ldots}\).

Returns a list containing the differences between consecutive elements in \(List1\). Each element of \(List1\) is subtracted from the next element of \(List1\). The resulting list is always one element shorter than the original \(List1\).
list►mat()

Returns a matrix filled row-by-row with the elements from List.

elementsPerRow, if included, specifies the number of elements per row. Default is the number of elements in List (one row).

If List does not fill the resulting matrix, zeros are added.

Note: You can insert this function from the computer keyboard by typing list►mat(...).

ln()

Returns the natural logarithm of the argument.

For a list, returns the natural logarithms of the elements.

ln(squareMatrix1) ⇒ squareMatrix

Returns the matrix natural logarithm of squareMatrix1. This is not the same as calculating the natural logarithm of each element. For information about the calculation method, refer to cos() on.

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.
LnReg \( X, Y, [Freq] [, Category, Include] \)

Computes the logarithmic regression \( y = a + b \cdot \ln(x) \) on lists \( X \) and \( Y \) with frequency \( Freq \). A summary of results is stored in the \( \text{stat.results} \) variable. (See page 131.)

All the lists must have equal dimension except for \( \text{Include} \).

\( X \) and \( Y \) are lists of independent and dependent variables.

\( Freq \) is an optional list of frequency values. Each element in \( Freq \) specifies the frequency of occurrence for each corresponding \( X \) and \( Y \) data point. The default value is 1. All elements must be integers \( \geq 0 \).

\( Category \) is a list of numeric or string category codes for the corresponding \( X \) and \( Y \) data.

\( Include \) is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{stat.RegEqn} )</td>
<td>Regression equation: ( a + b \cdot \ln(x) )</td>
</tr>
<tr>
<td>( \text{stat.a, stat.b} )</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>( \text{stat.r}^2 )</td>
<td>Coefficient of linear determination for transformed data</td>
</tr>
<tr>
<td>( \text{stat.r} )</td>
<td>Correlation coefficient for transformed data (( \ln(x), y ))</td>
</tr>
<tr>
<td>( \text{stat.Resid} )</td>
<td>Residuals associated with the logarithmic model</td>
</tr>
<tr>
<td>( \text{stat.ResidTrans} )</td>
<td>Residuals associated with linear fit of transformed data</td>
</tr>
<tr>
<td>( \text{stat.XReg} )</td>
<td>List of data points in the modified ( X ) List actually used in the regression based on restrictions of ( Freq, Category ) List, and ( Include ) Categories</td>
</tr>
<tr>
<td>( \text{stat.YReg} )</td>
<td>List of data points in the modified ( Y ) List actually used in the regression based on restrictions of ( Freq, Category ) List, and ( Include ) Categories</td>
</tr>
<tr>
<td>( \text{stat.FreqReg} )</td>
<td>List of frequencies corresponding to ( \text{stat.XReg} ) and ( \text{stat.YReg} )</td>
</tr>
</tbody>
</table>
**Local**

Local \([ Var1, Var2 ] [, Var3 ] \) ...

Declares the specified \( \text{vars} \) as local variables. Those variables exist only during evaluation of a function and are deleted when the function finishes execution.

**Note:** Local variables save memory because they only exist temporarily. Also, they do not disturb any existing global variable values. Local variables must be used for **For** loops and for temporarily saving values in a multi-line function since modifications on global variables are not allowed in a function.

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing `>` instead of `enter` at the end of each line. On the computer keyboard, hold down **Alt** and press **Enter**.

---

**Lock**

Lock \([ Var1, Var2 ] [, Var3 ] \) ...

Lock \( \text{Var} \).

Locks the specified variables or variable group. Locked variables cannot be modified or deleted.

You cannot lock or unlock the system variable \( \text{Ans} \), and you cannot lock the system variable groups \( \text{stat.} \) or \( \text{tvm} \).

**Note:** The **Lock** command clears the Undo/Redo history when applied to unlocked variables.

See **unLock**, page 147, and **getLockInfo()**, page 57.

---

**log()**

log \( (\text{Value1}, \text{Value2}) \) ⇒ \( \text{value} \)

log \( (\text{List1}, \text{Value2}) \) ⇒ \( \text{list} \)

Returns the base-\( \text{Value2} \) logarithm of the first argument.

**Note:** See also **Log template**, page 6.
\textbf{log()} 

For a list, returns the base-\texttt{Value2} logarithm of the elements.

If the second argument is omitted, 10 is used as the base.

\begin{align*}
\text{list} \cdot \text{log} \cdot \text{base} \Rightarrow \text{Value1} \\
\text{base} \text{is defaults to 10.}
\end{align*}

\begin{align*}
\text{squareMatrix1} \Rightarrow \text{squareMatrix} \\
\text{log} \cdot \text{squareMatrix1} \cdot \text{Value} \\
\text{Returns the matrix base-\texttt{Value} logarithm of squareMatrix1. This is not the same as calculating the base-\texttt{Value} logarithm of each element. For information about the calculation method, refer to \texttt{cos}.}
\end{align*}

\text{squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.}

If the base argument is omitted, 10 is used as base.

\begin{align*}
\log_{10}(\{-3,1,2,5\}) \\
= "\text{Error: Non-real calculation}" \\
\end{align*}

\begin{align*}
\log_{10}(\{-3,1,2,5\}) \\
= \{0.477121+1.36438\cdot i, 0.079181, 0.69897\} \\
\end{align*}

\text{In Radian angle mode and Rectangular complex format:}

\begin{align*}
\log_{10}(\begin{pmatrix} 1 & 5 & 3 \\ 4 & 2 & 1 \\ 6 & -2 & 1 \end{pmatrix}) \\
= \begin{bmatrix} 0.795387+0.753438\cdot i \\ 0.194895-0.315095\cdot i \\ -0.115909-0.904706\cdot i \end{bmatrix} \\
= \{0.003993-0.64742\cdot i \\ 0.462485+0.27073\cdot i \\ 0.488304+0.7774\cdot i\} \\
\end{align*}

To see the entire result, press \texttt{up} and then use \texttt{left} and \texttt{right} to move the cursor.

---

\textbf{Logistic} 

\texttt{Logistic X, Y, [Freq] [, Category, Include]} 

Computes the logistic regression \( y = \frac{c}{1+a\cdot e^{-bx}} \) on lists \( X \) and \( Y \) with frequency \( Freq \). A summary of results is stored in the \texttt{stat.results} variable. (See page 131.)

All the lists must have equal dimension except for \texttt{Include}.

\( X \) and \( Y \) are lists of independent and dependent variables.

\( Freq \) is an optional list of frequency values. Each element in \( Freq \) specifies the frequency of occurrence for each corresponding \( X \) and \( Y \) data point. The default value is 1. All elements must be integers \( \geq 0 \).

\( Category \) is a list of numeric or string category codes for the corresponding \( X \) and \( Y \) data.

\( Include \) is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.
For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression equation: ( c/(1+a\cdot e^{-b\cdot x}) )</td>
</tr>
<tr>
<td>stat.a, stat.b, stat.c</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the regression</td>
</tr>
<tr>
<td>stat.XReg</td>
<td>List of data points in the modified ( X ) list actually used in the regression based on restrictions of ( Freq ), ( Category ) list, and ( Include ) categories</td>
</tr>
<tr>
<td>stat.YReg</td>
<td>List of data points in the modified ( Y ) list actually used in the regression based on restrictions of ( Freq ), ( Category ) list, and ( Include ) categories</td>
</tr>
<tr>
<td>stat.FreqReg</td>
<td>List of frequencies corresponding to ( stat.XReg ) and ( stat.YReg )</td>
</tr>
</tbody>
</table>

**LogisticD**

**LogisticD** \( X, [Iterations], [Freq[, Category, Include]] \)

Computes the logistic regression \( y = (c/(1+a\cdot e^{-b\cdot x})+d) \) on lists \( X \) and \( Y \) with frequency \( Freq \), using a specified number of \( Iterations \). A summary of results is stored in the \( stat.results \) variable. (See page 131.)

All the lists must have equal dimension except for \( Include \).

\( X \) and \( Y \) are lists of independent and dependent variables.

\( Freq \) is an optional list of frequency values. Each element in \( Freq \) specifies the frequency of occurrence for each corresponding \( X \) and \( Y \) data point. The default value is 1. All elements must be integers \( \geq 0 \).

\( Category \) is a list of numeric or string category codes for the corresponding \( X \) and \( Y \) data.

\( Include \) is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.
<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression equation: $c/(1+a\cdot e^{-bx}+d)$</td>
</tr>
<tr>
<td>stat.a, stat.b, stat.c, stat.d</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the regression</td>
</tr>
<tr>
<td>stat.XReg</td>
<td>List of data points in the modified X List actually used in the regression based on restrictions of Freq, Category List, and Include Categories</td>
</tr>
<tr>
<td>stat.YReg</td>
<td>List of data points in the modified Y List actually used in the regression based on restrictions of Freq, Category List, and Include Categories</td>
</tr>
<tr>
<td>stat.FreqReg</td>
<td>List of frequencies corresponding to stat.XReg and stat.YReg</td>
</tr>
</tbody>
</table>

**Loop**

Loop

Block

EndLoop

Repeatedly executes the statements in Block. Note that the loop will be executed endlessly, unless a Goto or Exit instruction is executed within Block.

Block is a sequence of statements separated with the `:` character.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing `←` instead of `·` at the end of each line. On the computer keyboard, hold down Alt and press Enter.
LU

**LU Matrix, lMatrix, uMatrix, pMatrix[, Tol]**

Calculates the Doolittle LU (lower-upper) decomposition of a real or complex matrix. The lower triangular matrix is stored in lMatrix, the upper triangular matrix in uMatrix, and the permutation matrix (which describes the row swaps done during the calculation) in pMatrix.

\[ lMatrix \cdot uMatrix = pMatrix \cdot \text{matrix} \]

Optionally, any matrix element is treated as zero if its absolute value is less than Tol. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, Tol is ignored.

- If you use [CTRL] [ENTER] or set the Auto or Approximate mode to Approximate, computations are done using floating-point arithmetic.
- If Tol is omitted or not used, the default tolerance is calculated as:
  \[ 5 \times 10^{-14} \times \max(\text{dim}(\text{Matrix})) \times \text{rowNorm}(\text{Matrix}) \]

The LU factorization algorithm uses partial pivoting with row interchanges.

**M**

**mat►list()**

**mat►list(Matrix) ⇒ list**

Returns a list filled with the elements in Matrix. The elements are copied from Matrix row by row.

**Note:** You can insert this function from the computer keyboard by typing mat►list(...).

**max()**

**max(Value1, Value2) ⇒ expression**

**max(List1, List2) ⇒ list**

max(2,3,1.4) 2.3
max({1,2},{4,3}) {1,3}
max()

**max(Matrix1, Matrix2) ⇒ matrix**

Returns the maximum of the two arguments. If the arguments are two lists or matrices, returns a list or matrix containing the maximum value of each pair of corresponding elements.

**max(List) ⇒ expression**

Returns the maximum element in list.

**max(Matrix1) ⇒ matrix**

Returns a row vector containing the maximum element of each column in Matrix1.

Empty (void) elements are ignored. For more information on empty elements, see page 177.

**Note:** See also min().

---

mean()

**mean(List[, freqList]) ⇒ expression**

Returns the mean of the elements in List.

Each freqList element counts the number of consecutive occurrences of the corresponding element in List.

**mean(Matrix1[, freqMatrix]) ⇒ matrix**

Returns a row vector of the means of all the columns in Matrix1.

Each freqMatrix element counts the number of consecutive occurrences of the corresponding element in Matrix1.

Empty (void) elements are ignored. For more information on empty elements, see page 177.
**median()**  

\[
\text{median}(\text{List}[, \text{freqList}]) \Rightarrow \text{expression}
\]

Returns the median of the elements in \( \text{List} \).

Each \( \text{freqList} \) element counts the number of consecutive occurrences of the corresponding element in \( \text{List} \).

\[
\text{median}(\text{Matrix1}[, \text{freqMatrix}]) \Rightarrow \text{matrix}
\]

Returns a row vector containing the medians of the columns in \( \text{Matrix1} \).

Each \( \text{freqMatrix} \) element counts the number of consecutive occurrences of the corresponding element in \( \text{Matrix1} \).

**Notes:**

- All entries in the list or matrix must simplify to numbers.
- Empty (void) elements in the list or matrix are ignored. For more information on empty elements, see page 177.

---

**MedMed**  

\[
\text{MedMed} \, X, Y[, \text{Freq}][, \text{Category}, \text{Include}]\]

Computes the median-median line \( y = (m \cdot x + b) \) on lists \( X \) and \( Y \) with frequency \( \text{Freq} \). A summary of results is stored in the \( \text{stat.results} \) variable. (See page 131.)

All the lists must have equal dimension except for \( \text{Include} \).

\( X \) and \( Y \) are lists of independent and dependent variables.

\( \text{Freq} \) is an optional list of frequency values. Each element in \( \text{Freq} \) specifies the frequency of occurrence for each corresponding \( X \) and \( Y \) data point. The default value is 1. All elements must be integers \( \geq 0 \).

\( \text{Category} \) is a list of numeric or string category codes for the corresponding \( X \) and \( Y \) data.

\( \text{Include} \) is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.
<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Median–median line equation: ( m \cdot x + b )</td>
</tr>
<tr>
<td>stat.m, stat.b</td>
<td>Model coefficients</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the median–median line</td>
</tr>
<tr>
<td>stat.XReg</td>
<td>List of data points in the modified ( X ) List actually used in the regression based on restrictions of ( Freq, ) Category List, and Include Categories</td>
</tr>
<tr>
<td>stat.YReg</td>
<td>List of data points in the modified ( Y ) List actually used in the regression based on restrictions of ( Freq, ) Category List, and Include Categories</td>
</tr>
<tr>
<td>stat.FreqReg</td>
<td>List of frequencies corresponding to ( stat.XReg ) and ( stat.YReg )</td>
</tr>
</tbody>
</table>

**mid()**

\[
\text{mid}(\text{sourceString}, \text{Start}[, \text{Count}]) \Rightarrow \text{string}
\]

Returns \( \text{Count} \) characters from character string \( \text{sourceString} \), beginning with character number \( \text{Start} \).

If \( \text{Count} \) is omitted or is greater than the dimension of \( \text{sourceString} \), returns all characters from \( \text{sourceString} \), beginning with character number \( \text{Start} \).

\( \text{Count} \) must be \( \geq 0 \). If \( \text{Count} = 0 \), returns an empty string.

**Catalog >**

\[
\begin{array}{ll}
\text{mid}("Hello there",2) & "ello there" \\
\text{mid}("Hello there",7,3) & "he" \\
\text{mid}("Hello there",1,5) & "Hello" \\
\text{mid}("Hello there",1,0) & "" \\
\end{array}
\]

\[
\begin{array}{ll}
\text{mid}\{9,8,7,6\},3 & \{7,6\} \\
\text{mid}\{9,8,7,6\},2,2 & \{8,7\} \\
\text{mid}\{9,8,7,6\},1,2 & \{9,8\} \\
\text{mid}\{9,8,7,6\},1,0 & \{\ldots\} \\
\end{array}
\]

\[
\begin{array}{ll}
\text{mid}\{"A","B","C","D"\},2,2 & \{"B","C"\} \\
\end{array}
\]

**Alphabetical Listing** 83
**min()**

- **min(Value1, Value2) ⇒ expression**
  \[ \min(2.3,1.4) = 1.4 \]
  \[ \min(\{1.2\}, \{-4,3\}) = \{-4,2\} \]

- **min(List1, List2) ⇒ list**
  \[ \min(\{0,1,-7,1.3,0.5\}) = -7 \]
  \[ \min(\begin{bmatrix} 1 & -3 & 7 \\ -4 & 0 & 0.3 \end{bmatrix}) = \begin{bmatrix} -4 \\ -3 \\ 0.3 \end{bmatrix} \]

- **min(Matrix1, Matrix2) ⇒ matrix**

  Returns the minimum of the two arguments. If the arguments are two lists or matrices, returns a list or matrix containing the minimum value of each pair of corresponding elements.

  **min(List) ⇒ expression**

  Returns the minimum element of List.

  **min(Matrix1) ⇒ matrix**

  Returns a row vector containing the minimum element of each column in Matrix1.

  **Note:** See also max().

**mirr()**

- **mirr(financeRate, reinvestRate, CF0, CFList [, CFFreq])**

  Financial function that returns the modified internal rate of return of an investment.

  **financeRate** is the interest rate that you pay on the cash flow amounts.

  **reinvestRate** is the interest rate at which the cash flows are reinvested.

  **CF0** is the initial cash flow at time 0; it must be a real number.

  **CFList** is a list of cash flow amounts after the initial cash flow CF0.

  **CFFreq** is an optional list in which each element specifies the frequency of occurrence for a grouped (consecutive) cash flow amount, which is the corresponding element of CFList. The default is 1; if you enter values, they must be positive integers < 10,000.

  **Note:** See also irr(), page 66.
**mod()**

\[ \text{mod}(Value_1, Value_2) \Rightarrow \text{expression} \]
\[ \text{mod}(List_1, List_2) \Rightarrow \text{list} \]
\[ \text{mod}(Matrix_1, Matrix_2) \Rightarrow \text{matrix} \]

Returns the first argument modulo the second argument as defined by the identities:

\[ \text{mod}(x, 0) = x \]
\[ \text{mod}(x, y) = x - y \lfloor x/y \rfloor \]

When the second argument is non-zero, the result is periodic in that argument. The result is either zero or has the same sign as the second argument.

If the arguments are two lists or two matrices, returns a list or matrix containing the modulo of each pair of corresponding elements.

**Note:** See also **remain()**, page 111

---

**mRow()**

\[ \text{mRow}(Value, Matrix_1, Index) \Rightarrow \text{matrix} \]

Returns a copy of Matrix_1 with each element in row Index of Matrix_1 multiplied by Value.

---

**mRowAdd()**

\[ \text{mRowAdd}(Value, Matrix_1, Index_1, Index_2) \Rightarrow \text{matrix} \]

Returns a copy of Matrix_1 with each element in row Index_2 of Matrix_1 replaced with:

Value \times \text{row Index}_1 + \text{row Index}_2

---

**MultReg**

\[ \text{MultReg} \ Y, X_1[, X_2[, X_3[, ..., X_{10}]]] \]

Calculates multiple linear regression of list \( Y \) on lists \( X_1, X_2, ..., X_{10} \). A summary of results is stored in the `stat.results` variable.

(See page 131.)
All the lists must have equal dimension.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.

### MultReg

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression Equation: $b_0+b_1\cdot x_1+b_2\cdot x_2+ ...$</td>
</tr>
<tr>
<td>stat.b0, stat.b1, ...</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.R$^2$</td>
<td>Coefficient of multiple determination</td>
</tr>
<tr>
<td>stat.yList</td>
<td>$yList = b_0+b_1\cdot x_1+ ...$</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the regression</td>
</tr>
</tbody>
</table>

### MultRegIntervals

**MultRegIntervals** $Y, X1[, X2[, X3, ...[, X10]]]$, $XValList[, CLevel]$

Computes a predicted $y$-value, a level $C$ prediction interval for a single observation, and a level $C$ confidence interval for the mean response.

A summary of results is stored in the `stat.results` variable. (See page 131.)

All the lists must have equal dimension.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression Equation: $b_0+b_1\cdot x_1+b_2\cdot x_2+ ...$</td>
</tr>
<tr>
<td>stat.$\hat{y}$</td>
<td>A point estimate: $\hat{y} = b_0 + b_1 \cdot x_l + ...$ for $XValList$</td>
</tr>
<tr>
<td>stat.diffError</td>
<td>Error degrees of freedom</td>
</tr>
<tr>
<td>stat.CLower, stat.CUpper</td>
<td>Confidence interval for a mean response</td>
</tr>
<tr>
<td>stat.ME</td>
<td>Confidence interval margin of error</td>
</tr>
<tr>
<td>stat.SE</td>
<td>Standard error of mean response</td>
</tr>
<tr>
<td>stat.LowerPred, stat.UpperPred</td>
<td>Prediction interval for a single observation</td>
</tr>
<tr>
<td>stat.MEPred</td>
<td>Prediction interval margin of error</td>
</tr>
</tbody>
</table>
### Output variable | Description
---|---
stat.SEPred | Standard error for prediction
stat.bList | List of regression coefficients, \{b0,b1,b2,\ldots\}
stat.Resid | Residuals from the regression

### MultRegTests

\textbf{MultRegTests} \( Y, X1[, X2[, X3,\ldots[, X10]]] \)

Multiple linear regression test computes a multiple linear regression on the given data and provides the global \( F \) test statistic and \( t \) test statistics for the coefficients.

A summary of results is stored in the \texttt{stat.results} variable. (See page 131.)

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.

#### Outputs

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression Equation: ( b0+b1\cdot x1+b2\cdot x2+ \ldots )</td>
</tr>
<tr>
<td>stat.F</td>
<td>Global ( F ) test statistic</td>
</tr>
<tr>
<td>stat.PVal</td>
<td>P-value associated with global ( F ) statistic</td>
</tr>
<tr>
<td>stat.R^2</td>
<td>Coefficient of multiple determination</td>
</tr>
<tr>
<td>stat.AdjR^2</td>
<td>Adjusted coefficient of multiple determination</td>
</tr>
<tr>
<td>stat.s</td>
<td>Standard deviation of the error</td>
</tr>
<tr>
<td>stat.DW</td>
<td>Durbin-Watson statistic; used to determine whether first-order auto correlation is present in the model</td>
</tr>
<tr>
<td>stat.dfReg</td>
<td>Regression degrees of freedom</td>
</tr>
<tr>
<td>stat.SSReg</td>
<td>Regression sum of squares</td>
</tr>
<tr>
<td>stat.MSReg</td>
<td>Regression mean square</td>
</tr>
<tr>
<td>stat.dfError</td>
<td>Error degrees of freedom</td>
</tr>
<tr>
<td>stat.SSError</td>
<td>Error sum of squares</td>
</tr>
<tr>
<td>stat.MSError</td>
<td>Error mean square</td>
</tr>
<tr>
<td>stat.bList</td>
<td>{b0,b1,\ldots} List of coefficients</td>
</tr>
<tr>
<td>Output variable</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>stat.tList</td>
<td>List of t statistics, one for each coefficient in the bList</td>
</tr>
<tr>
<td>stat.PList</td>
<td>List P-values for each t statistic</td>
</tr>
<tr>
<td>stat.SEList</td>
<td>List of standard errors for coefficients in bList</td>
</tr>
<tr>
<td>stat.yList</td>
<td>yList = b0+b1*x1+ . . .</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the regression</td>
</tr>
<tr>
<td>stat.sResid</td>
<td>Standardized residuals; obtained by dividing a residual by its standard deviation</td>
</tr>
<tr>
<td>stat.CookDist</td>
<td>Cook’s distance; measure of the influence of an observation based on the residual and leverage</td>
</tr>
<tr>
<td>stat.Leverage</td>
<td>Measure of how far the values of the independent variable are from their mean values</td>
</tr>
</tbody>
</table>

\[ N \]

**nand**

BooleanExpr1 \( \text{nand} \) BooleanExpr2 returns Boolean expression

BooleanList1 \( \text{nand} \) BooleanList2 returns Boolean list

BooleanMatrix1 \( \text{nand} \) BooleanMatrix2 returns Boolean matrix

Returns the negation of a logical and operation on the two arguments. Returns true, false, or a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

\[ \text{Integer1} \text{nand} \text{Integer2} \Rightarrow \text{integer} \]

Compresses two real integers bit-by-bit using a nand operation. Internally, both integers are converted to signed, 64-bit binary numbers. When corresponding bits are compared, the result is 1 if both bits are 1; otherwise, the result is 0. The returned value represents the bit results, and is displayed according to the Base mode.

You can enter the integers in any number base. For a binary or hexadecimal entry, you must use the 0b or 0h prefix, respectively. Without a prefix, integers are treated as decimal (base 10).
\( \text{\textbf{nCr}}() \)\n
**nCr(Value1, Value2) \( \Rightarrow \) expression**

For integer Value1 and Value2 with Value1 \( \geq \) Value2 \( \geq 0 \), \( \text{nCr}() \) is the number of combinations of Value1 things taken Value2 at a time. (This is also known as a binomial coefficient.)

\( \text{nCr}(Value, 0) \Rightarrow 1 \)

\( \text{nCr}(Value, \text{negInteger}) \Rightarrow 0 \)

\( \text{nCr}(Value, \text{posInteger}) \Rightarrow Value \cdot (Value - 1) \ldots (Value - \text{posInteger} + 1)/\text{posInteger}! \)

\( \text{nCr}(Value, \text{nonInteger}) \Rightarrow \text{expression}!/(\text{Value} - \text{nonInteger})! \cdot \text{nonInteger}! \)

\( \text{nCr}(\text{List1, List2}) \Rightarrow \text{list} \)

Returns a list of combinations based on the corresponding element pairs in the two lists. The arguments must be the same size list.

\( \text{nCr}(\text{Matrix1, Matrix2}) \Rightarrow \text{matrix} \)

Returns a matrix of combinations based on the corresponding element pairs in the two matrices. The arguments must be the same size matrix.

\( \text{nDerivative()} \)

**nDerivative(Expr1, Var=\text{Value}[,\text{Order}]) \Rightarrow \text{value}**

Returns the numerical derivative calculated using auto differentiation methods.

When Value is specified, it overrides any prior variable assignment or any current "|" substitution for the variable.

If the variable Var does not contain a numeric value, you must provide Value.

Order of the derivative must be 1 or 2.
**nDerivative()**

*Note:* The `nDerivative()` algorithm has a limitation: it works recursively through the unsimplified expression, computing the numeric value of the first derivative (and second, if applicable) and the evaluation of each subexpression, which may lead to an unexpected result.

Consider the example on the right. The first derivative of \( x(x^2+x)^{1/3} \) at \( x=0 \) is equal to 0. However, because the first derivative of the subexpression \( (x^2+x)^{1/3} \) is undefined at \( x=0 \), and this value is used to calculate the derivative of the total expression, `nDerivative()` reports the result as undefined and displays a warning message.

If you encounter this limitation, verify the solution graphically. You can also try using `centralDiff()`.

**newList()**

`newList(numElements) ⇒ list`

Returns a list with a dimension of `numElements`. Each element is zero.

**newMat()**

`newMat(numRows, numColumns) ⇒ matrix`

Returns a matrix of zeros with the dimension `numRows` by `numColumns`.

**nfMax()**

`nfMax(Expr, Var) ⇒ value`

`nfMax(Expr, Var, lowBound) ⇒ value`

`nfMax(Expr, Var, lowBound, upBound) ⇒ value`

`nfMax(Expr, Var) | lowBound≤Var≤upBound ⇒ value`

Returns a candidate numerical value of variable `Var` where the local maximum of `Expr` occurs.

If you supply `lowBound` and `upBound`, the function looks in the closed interval `[lowBound, upBound] for the local maximum.
nfMin()  

nfMin(Expr, Var) ⇒ value  
nfMin(Expr, Var, lowBound) ⇒ value  
nfMin(Expr, Var, lowBound, upBound) ⇒ value  
nfMin(Expr, Var, lowBound ≤ Var ≤ upBound) ⇒ value

Returns a candidate numerical value of variable Var where the local minimum of Expr occurs.

If you supply lowBound and upBound, the function looks in the closed interval [lowBound, upBound] for the local minimum.

nInt()  

nInt(Expr1, Var, Lower, Upper) ⇒ expression  

If the integrand Expr1 contains no variable other than Var, and if Lower and Upper are constants, positive ∞, or negative ∞, then nInt() returns an approximation of \( \int_{\text{Lower}}^{\text{Upper}} \) of (Expr1, Var, Lower, Upper). This approximation is a weighted average of some sample values of the integrand in the interval Lower < Var < Upper.

The goal is six significant digits. The adaptive algorithm terminates when it seems likely that the goal has been achieved, or when it seems unlikely that additional samples will yield a worthwhile improvement.

A warning is displayed (“Questionable accuracy”) when it seems that the goal has not been achieved.

Nest nInt() to do multiple numeric integration. Integration limits can depend on integration variables outside them.

nom()  

nom(effectiveRate, CpY) ⇒ value

Financial function that converts the annual effective interest rate effectiveRate to a nominal rate, given CpY as the number of compounding periods per year.

effectiveRate must be a real number, and CpY must be a real number > 0.

Note: See also eff(), page 44.
nor

BooleanExpr1 nor BooleanExpr2 returns Boolean expression
BooleanList1 nor BooleanList2 returns Boolean list
BooleanMatrix1 nor BooleanMatrix2 returns Boolean matrix

Returns the negation of a logical or operation on the two arguments. Returns true, false, or a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

Integer1 nor Integer2 ⇒ integer

Comparing two real integers bit-by-bit using a nor operation. Internally, both integers are converted to signed, 64-bit binary numbers. When corresponding bits are compared, the result is 1 if both bits are 1; otherwise, the result is 0. The returned value represents the bit results, and is displayed according to the Base mode.

You can enter the integers in any number base. For a binary or hexadecimal entry, you must use the 0b or 0h prefix, respectively. Without a prefix, integers are treated as decimal (base 10).

norm()

norm(Matrix) ⇒ expression
norm(Vector) ⇒ expression

Returns the Frobenius norm.

<table>
<thead>
<tr>
<th>norm(Matrix)</th>
<th>norm(Vector)</th>
</tr>
</thead>
<tbody>
<tr>
<td>norm([[1, 2], [3, 4]])</td>
<td>5.47723</td>
</tr>
<tr>
<td>norm([1, 2])</td>
<td>2.23607</td>
</tr>
<tr>
<td>norm([1, 2])</td>
<td>2.23607</td>
</tr>
</tbody>
</table>

normCdf()

normCdf(lowBound, upBound[, μ[, σ]]) ⇒ number if lowBound and upBound are numbers, list if lowBound and upBound are lists

Computes the normal distribution probability between lowBound and upBound for the specified μ (default=0) and σ (default=1).

For P(X ≤ upBound), set lowBound = 9E999.
**normPdf()**

**normPdf(XVal, μ, σ) ⇒ number if XVal is a number, list if XVal is a list**

Computes the probability density function for the normal distribution at a specified XVal value for the specified μ and σ.

---

**not**

**not BooleanExpr ⇒ Boolean expression**

Returns true, false, or a simplified form of the argument.

**not Integer1 ⇒ integer**

Returns the one’s complement of a real integer. Internally, Integer1 is converted to a signed, 64-bit binary number. The value of each bit is flipped (0 becomes 1, and vice versa) for the one’s complement. Results are displayed according to the Base mode.

You can enter the integer in any number base. For a binary or hexadecimal entry, you must use the 0b or 0h prefix, respectively. Without a prefix, the integer is treated as decimal (base 10).

If you enter a decimal integer that is too large for a signed, 64-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range. For more information, see ►Base2, page 20.

---

**nPr()**

**nPr(Value1, Value2) ⇒ expression**

For integer Value1 and Value2 with Value1 ≥ Value2 ≥ 0, nPr() is the number of permutations of Value1 things taken Value2 at a time.

**nPr(Value, 0) ⇒ 1**

**nPr(Value, negInteger) ⇒ 1 / (Value+1) × (Value+2) × ... × (Value-negInteger)**
\[ nPr \]

\[ nPr(\text{Value}, \text{posInteger}) \Rightarrow \text{Value} \cdot (\text{Value} - 1) \ldots (\text{Value} - \text{posInteger} + 1) \]

\[ nPr(\text{Value}, \text{nonInteger}) \Rightarrow \text{Value}! \left/ (\text{Value} - \text{nonInteger})! \right. \]

\[ nPr(\text{List1}, \text{List2}) \Rightarrow \text{list} \]

Returns a list of permutations based on the corresponding element pairs in the two lists. The arguments must be the same size list.

\[ nPr(\text{Matrix1}, \text{Matrix2}) \Rightarrow \text{matrix} \]

Returns a matrix of permutations based on the corresponding element pairs in the two matrices. The arguments must be the same size matrix.

\[ \text{npv}() \]

\[ \text{npv}(\text{InterestRate}, \text{CFO}, \text{CFList}, [\text{CFFreq}]) \]

Financial function that calculates net present value; the sum of the present values for the cash inflows and outflows. A positive result for npv indicates a profitable investment.

\text{InterestRate} is the rate by which to discount the cash flows (the cost of money) over one period.

\text{CF0} is the initial cash flow at time 0; it must be a real number.

\text{CFList} is a list of cash flow amounts after the initial cash flow \text{CF0}.

\text{CFFreq} is a list in which each element specifies the frequency of occurrence for a grouped (consecutive) cash flow amount, which is the corresponding element of \text{CFList}. The default is 1; if you enter values, they must be positive integers < 10,000.
**nSolve()**

\[ \text{nSolve}(\text{Equation}, \text{Var} = \text{Guess}) \Rightarrow \text{number or error string} \]

\[ \text{nSolve}(\text{Equation}, \text{Var} = \text{Guess}, \text{lowBound}) \Rightarrow \text{number or error string} \]

\[ \text{nSolve}(\text{Equation}, \text{Var} = \text{Guess}, \text{lowBound}, \text{upBound}) \Rightarrow \text{number or error string} \]

Iteratively searches for one approximate real numeric solution to \text{Equation} for its one variable. Specify the variable as:

\text{variable}

- or -

\text{variable} = \text{real number}

For example, \( x \) is valid and so is \( x=3 \).

\text{nSolve()} attempts to determine either one point where the residual is zero or two relatively close points where the residual has opposite signs and the magnitude of the residual is not excessive. If it cannot achieve this using a modest number of sample points, it returns the string “no solution found.”

\[ \frac{\text{nSolve}(x^2+5.5x-25=9x)}{3.84429} \]
\[ \frac{\text{nSolve}(x^2=4,x=1)}{-2.} \]
\[ \frac{\text{nSolve}(x^2=4,x=1)}{2.} \]

Note: If there are multiple solutions, you can use a guess to help find a particular solution.

---

**OneVar**

\[ \text{OneVar}[1,]X[,][Freq][,]Category[,Include]] \]

\[ \text{OneVar}[n,]X1,X2[,X3[,..,X20]] \]

Calculates 1-variable statistics on up to 20 lists. A summary of results is stored in the \text{stat.results} variable. (See page 131.)

All the lists must have equal dimension except for \text{Include}.

\text{Freq} is an optional list of frequency values. Each element in \text{Freq} specifies the frequency of occurrence for each corresponding \( X \) and \( Y \) data point. The default value is 1. All elements must be integers \( \geq 0 \).
**Category** is a list of numeric category codes for the corresponding \( X \) values.

**Include** is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

An empty (void) element in any of the lists \( X \), \( Freq \), or **Category** results in a void for the corresponding element of all those lists. An empty element in any of the lists \( X1 \) through \( X20 \) results in a void for the corresponding element of all those lists. For more information on empty elements, see page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{stat.} \bar{x} )</td>
<td>Mean of ( x ) values</td>
</tr>
<tr>
<td>( \text{stat.} \Sigma x )</td>
<td>Sum of ( x ) values</td>
</tr>
<tr>
<td>( \text{stat.} \Sigma x^2 )</td>
<td>Sum of ( x^2 ) values</td>
</tr>
<tr>
<td>( \text{stat.sx} )</td>
<td>Sample standard deviation of ( x )</td>
</tr>
<tr>
<td>( \text{stat.ox} )</td>
<td>Population standard deviation of ( x )</td>
</tr>
<tr>
<td>( \text{stat.n} )</td>
<td>Number of data points</td>
</tr>
<tr>
<td>( \text{stat.MinX} )</td>
<td>Minimum of ( x ) values</td>
</tr>
<tr>
<td>( \text{stat.Q}_1 ) ( X )</td>
<td>1st Quartile of ( x )</td>
</tr>
<tr>
<td>( \text{stat.MedianX} )</td>
<td>Median of ( x )</td>
</tr>
<tr>
<td>( \text{stat.Q}_3 ) ( X )</td>
<td>3rd Quartile of ( x )</td>
</tr>
<tr>
<td>( \text{stat.MaxX} )</td>
<td>Maximum of ( x ) values</td>
</tr>
<tr>
<td>( \text{stat.SSX} )</td>
<td>Sum of squares of deviations from the mean of ( x )</td>
</tr>
</tbody>
</table>

**BooleanExpr1** or **BooleanExpr2** returns **Boolean** expression

**BooleanList1** or **BooleanList2** returns **Boolean** list

**BooleanMatrix1** or **BooleanMatrix2** returns **Boolean** matrix

Returns true or false or a simplified form of the original entry.

\[
\begin{align*}
\text{Define } g(x) &- \text{Func} \\
& \quad \text{If } x \leq 0 \text{ or } x \geq 5 \\
& \quad \text{Goto end} \\
& \quad \text{Return } x \cdot 3 \\
& \quad \text{Lbl end} \\
& \text{EndFunc}
\end{align*}
\]

\[
g(3) \quad 9
\]

\[
g(0) \quad A \text{ function did not return a value}
\]
or

Returns true if either or both expressions simplify to true. Returns false only if both expressions evaluate to false.

Note: See xor.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing → instead of enter at the end of each line. On the computer keyboard, hold down Alt and press Enter.

\texttt{Integer1 \texttt{or} Integer2 \Rightarrow integer}

Compares two real integers bit-by-bit using an or operation. Internally, both integers are converted to signed, 64-bit binary numbers. When corresponding bits are compared, the result is 1 if either bit is 1; the result is 0 only if both bits are 0. The returned value represents the bit results, and is displayed according to the Base mode.

You can enter the integers in any number base. For a binary or hexadecimal entry, you must use the 0b or 0h prefix, respectively. Without a prefix, integers are treated as decimal (base 10).

If you enter a decimal integer that is too large for a signed, 64-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range. For more information, see \texttt{Base2}, page 20.

Note: See xor.

<table>
<thead>
<tr>
<th>In Hex base mode:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0h7AC36 or 0h3D5F</td>
</tr>
</tbody>
</table>

Important: Zero, not the letter O.

<table>
<thead>
<tr>
<th>In Bin base mode:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b100101 or 0b100</td>
</tr>
</tbody>
</table>

Note: A binary entry can have up to 64 digits (not counting the 0b prefix). A hexadecimal entry can have up to 16 digits.

**ord()**

\texttt{ord(String) \Rightarrow integer}

\texttt{ord(List1) \Rightarrow list}

Returns the numeric code of the first character in character string \texttt{String}, or a list of the first characters of each list element.

\begin{center}
\begin{tabular}{|l|c|}
\hline
Character & Numeric Code \\
\hline
\texttt{"hello"} & 104 \\
\texttt{char(104)} & "h" \\
\texttt{ord(char(24))} & 24 \\
\texttt{ord\{"alpha","beta"\}} & \{97,98\} \\
\hline
\end{tabular}
\end{center}
$P$

$\textbf{P} \uparrow \text{Rx()}$

\begin{align*}
P & \rightarrow \text{Rx}(\theta \text{Expr}, \theta \text{Expr}) \Rightarrow \text{expression} \\
P & \rightarrow \text{Rx}(\theta \text{List}, \theta \text{List}) \Rightarrow \text{list} \\
P & \rightarrow \text{Rx}(\theta \text{Matrix}, \theta \text{Matrix}) \Rightarrow \text{matrix}
\end{align*}

Returns the equivalent x-coordinate of the $(r, \theta)$ pair.

**Note:** The $\theta$ argument is interpreted as either a degree, gradian or radian angle, according to the current angle mode. If the argument is an expression, you can use °, G, or r to override the angle mode setting temporarily.

**Note:** You can insert this function from the computer keyboard by typing $P@>\text{Rx} \ (\ \ldots \ )$.

$\textbf{P} \uparrow \text{Ry()}$

\begin{align*}
P & \rightarrow \text{Ry}(\theta \text{Value}, \theta \text{Value}) \Rightarrow \text{value} \\
P & \rightarrow \text{Ry}(\theta \text{List}, \theta \text{List}) \Rightarrow \text{list} \\
P & \rightarrow \text{Ry}(\theta \text{Matrix}, \theta \text{Matrix}) \Rightarrow \text{matrix}
\end{align*}

Returns the equivalent y-coordinate of the $(r, \theta)$ pair.

**Note:** The $\theta$ argument is interpreted as either a degree, radian or gradian angle, according to the current angle mode.$^r$

**Note:** You can insert this function from the computer keyboard by typing $P@>\text{Ry} \ (\ \ldots \ )$.

$\textbf{PassErr}$

$\text{PassErr}$

Passes an error to the next level.

If system variable $\text{errCode}$ is zero, $\text{PassErr}$ does not do anything.

The $\text{Else}$ clause of the $\text{Try...Else...EndTry}$ block should use $\text{ClrErr}$ or $\text{PassErr}$. If the error is to be processed or ignored, use $\text{ClrErr}$. If what to do with the error is not known, use $\text{PassErr}$ to send it to the next error handler. If there are no more pending...
**PassErr**

**Try...Else...EndTry** error handlers, the error dialog box will be displayed as normal.

**Note:** See also **ClrErr**, page 25, and **Try**, page 141.

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing `[end]` instead of `[enter]` at the end of each line. On the computer keyboard, hold down `Alt` and press`Enter`.

### piecewise()

`piecewise(Expr1[, Cond1[, Expr2 [, Cond2[,...]]]])`

Returns definitions for a piecewise function in the form of a list. You can also create piecewise definitions by using a template.

**Note:** See also **Piecewise template**, page 6.

### poissCdf()

`poissCdf(\(λ\), lowBound, upBound) \Rightarrow number` if `lowBound` and `upBound` are numbers, `list` if `lowBound` and `upBound` are lists

`poissCdf(\(λ\), upBound)` for `P(0 \leq X \leq upBound)` \Rightarrow `number ` if `upBound` is a number, `list` if `upBound` is a list

Computes a cumulative probability for the discrete Poisson distribution with specified mean \(\lambda\).

For `P(X \leq upBound)`, set `lowBound`=0

### poissPdf()

`poissPdf(\(λ\), XVal) \Rightarrow number` if `XVal` is a number, `list` if `XVal` is a list

Computes a probability for the discrete Poisson distribution with the specified mean \(\lambda\).
Polar

Note: You can insert this operator from the computer keyboard by typing @>Polar.

Displays vector in polar form \([r \angle \theta]\). The vector must be of dimension 2 and can be a row or a column.

Note: ▶ Polar is a display-format instruction, not a conversion function. You can use it only at the end of an entry line, and it does not update ans.

Note: See also ▶ Rect, page 110.

complexValue ▶ Polar

Displays complexVector in polar form.

- Degree angle mode returns \((r \angle \theta)\).
- Radian angle mode returns \(re^{i\theta}\).

complexValue can have any complex form. However, an \(re^{i\theta}\) entry causes an error in Degree angle mode.

Note: You must use the parentheses for an \((r \angle \theta)\) polar entry.

In Radian angle mode:

\[
\begin{align*}
(3+4i)\text{Polar} &\Rightarrow e^{927295i} . 5 \\
\left(4 \angle \frac{\pi}{3}\right)\text{Polar} &\Rightarrow e^{1.0472i} . 4.
\end{align*}
\]

In Gradian angle mode:

\[
(4i)\text{Polar} \Rightarrow (4 \angle 100)
\]

In Degree angle mode:

\[
(3+4i)\text{Polar} \Rightarrow (5 \angle 53.1301)
\]

polyEval()

\[
polyEval(List1, Expr1) \Rightarrow expression
\]

\[
polyEval(List1, List2) \Rightarrow expression
\]

Interprets the first argument as the coefficient of a descending-degree polynomial, and returns the polynomial evaluated for the value of the second argument.

\[
polyEval\{1,2,3,4\},2 \Rightarrow 26
\]

\[
polyEval\{1,2,3,4\},\{2,7\} \Rightarrow \{26,262\}
\]
**polyRoots()**

\[
polyRoots(Poly, Var) \Rightarrow list
\]

\[
polyRoots(ListOfCoeffs) \Rightarrow list
\]

The first syntax, \( polyRoots(Poly, Var) \), returns a list of real roots of polynomial \( Poly \) with respect to variable \( Var \). If no real roots exist, returns an empty list: \{\}. 

\( Poly \) must be a polynomial in expanded form in one variable. Do not use unexpanded forms such as \( y^2y+1 \) or \( x\cdot x+2\cdot x+1 \)

The second syntax, \( polyRoots(ListOfCoeffs) \), returns a list of real roots for the coefficients in \( ListOfCoeffs \).

**Note:** See also \( cPolyRoots() \), page 33.

---

**PowerReg**

\[
\text{PowerReg} X, Y[, Freq][, Category, Include]]
\]

Computes the power regression \( y = (a\cdot x)^b \) on lists \( X \) and \( Y \) with frequency \( Freq \). A summary of results is stored in the \( \text{stat.results} \) variable. (See page 131.)

All the lists must have equal dimension except for \( Include \).

\( X \) and \( Y \) are lists of independent and dependent variables.

\( Freq \) is an optional list of frequency values. Each element in \( Freq \) specifies the frequency of occurrence for each corresponding \( X \) and \( Y \) data point. The default value is 1. All elements must be integers \( \geq 0 \).

\( Category \) is a list of numeric or string category codes for the corresponding \( X \) and \( Y \) data.

\( Include \) is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression equation: ( a\cdot x^b )</td>
</tr>
<tr>
<td>Output variable</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>stat.a, stat.b</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.r²</td>
<td>Coefficient of linear determination for transformed data</td>
</tr>
<tr>
<td>stat.r</td>
<td>Correlation coefficient for transformed data (ln(x), ln(y))</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals associated with the power model</td>
</tr>
<tr>
<td>stat.ResidTrans</td>
<td>Residuals associated with linear fit of transformed data</td>
</tr>
<tr>
<td>stat.XReg</td>
<td>List of data points in the modified X List actually used in the regression based on restrictions of Freq, Category List, and Include Categories</td>
</tr>
<tr>
<td>stat.YReg</td>
<td>List of data points in the modified Y List actually used in the regression based on restrictions of Freq, Category List, and Include Categories</td>
</tr>
<tr>
<td>stat.FreqReg</td>
<td>List of frequencies corresponding to stat.XReg and stat.YReg</td>
</tr>
</tbody>
</table>

**Prgm**

```
Prgm

Block

EndPrgm
```

Template for creating a user-defined program. Must be used with the Define, Define LibPub, or Define LibPriv command.

*Block* can be a single statement, a series of statements separated with the ":" character, or a series of statements on separate lines.

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing `[enter]` instead of `·` at the end of each line. On the computer keyboard, hold down Alt and press Enter.

Calculate GCD and display intermediate results.

```plaintext
Define proggcd(a,b)=Prgm

Local d
While b>0
  d:=mod(a,b)
  a:=b
  b:=d
  Disp a," ",b
EndWhile
Disp "GCD=",a
EndPrgm
```

**Done**

```
proggcd(4560,450)
```

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>450</td>
<td>60</td>
</tr>
<tr>
<td>60</td>
<td>30</td>
</tr>
<tr>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>GCD=30</td>
<td></td>
</tr>
</tbody>
</table>

**Done**

```
prodSeq()
```

See Π(), page 167.
**Product (PI)**

See Π (), page 167.

---

### product()

**product(List[, Start[, End]]) ⇒ expression**

Returns the product of the elements contained in `List`. `Start` and `End` are optional. They specify a range of elements.

**product(Matrix[, Start[, End]]) ⇒ matrix**

Returns a row vector containing the products of the elements in the columns of `Matrix`. `Start` and `End` are optional. They specify a range of rows.

Empty (void) elements are ignored. For more information on empty elements, see page 177.

---

### propFrac()

**propFrac(Value[, Var]) ⇒ value**

**propFrac(rational_number) returns rational_number as the sum of an integer and a fraction having the same sign and a greater denominator magnitude than numerator magnitude.**

**propFrac(rational_expression,Var) returns the sum of proper ratios and a polynomial with respect to Var. The degree of Var in the denominator exceeds the degree of Var in the numerator in each proper ratio. Similar powers of Var are collected. The terms and their factors are sorted with Var as the main variable.**

If `Var` is omitted, a proper fraction expansion is done with respect to the most main variable. The coefficients of the polynomial part are then made proper with respect to their most main variable first and so on.

You can use the `propFrac()` function to represent mixed fractions and demonstrate addition and subtraction of mixed fractions.
**QR Matrix, rMatrix[, Tol]**

Calculates the Householder QR factorization of a real or complex matrix. The resulting Q and R matrices are stored to the specified Matrix. The Q matrix is unitary. The R matrix is upper triangular.

Optionally, any matrix element is treated as zero if its absolute value is less than Tol. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, Tol is ignored.

- If you use `Ctrl+Enter` or set the Auto or Approximate mode to Approximate, computations are done using floating-point arithmetic.
- If Tol is omitted or not used, the default tolerance is calculated as:
  \[ 5 \times 10^{-14} \times \max(\text{dim}(\text{Matrix})) \times \text{rowNorm}(\text{Matrix}) \]

The QR factorization is computed numerically using Householder transformations. The symbolic solution is computed using Gram-Schmidt. The columns in qMatName are the orthonormal basis vectors that span the space defined by matrix.

**QuadReg**

**QuadReg X, Y[, Freq][, Category, Include]**

Computes the quadratic polynomial regression \( y=ax^2+bx+c \) on lists X and Y with frequency Freq. A summary of results is stored in the stat.results variable. (See page 131.)

All the lists must have equal dimension except for Include.

X and Y are lists of independent and dependent variables.

Freq is an optional list of frequency values. Each element in Freq specifies the frequency of occurrence for each corresponding X and Y data point. The default value is 1. All elements must be integers \( \geq 0 \).

Category is a list of numeric or string category codes for the...
corresponding \(X\) and \(Y\) data.

\textit{Include} is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression equation: (a x^2 + b x + c)</td>
</tr>
<tr>
<td>stat.a, stat.b, stat.c</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.R^2</td>
<td>Coefficient of determination</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the regression</td>
</tr>
<tr>
<td>stat.XReg</td>
<td>List of data points in the modified (X) List actually used in the regression based on restrictions of (Freq), (Category) List, and (Include) Categories</td>
</tr>
<tr>
<td>stat.YReg</td>
<td>List of data points in the modified (Y) List actually used in the regression based on restrictions of (Freq), (Category) List, and (Include) Categories</td>
</tr>
<tr>
<td>stat.FreqReg</td>
<td>List of frequencies corresponding to (stat.XReg) and (stat.YReg)</td>
</tr>
</tbody>
</table>

\textbf{QuartReg}

\texttt{QuartReg \textit{X}, \textit{Y}[, \textit{Freq}][, \textit{Category}, \textit{Include}][\textit{]}]

Computes the quartic polynomial regression
\(y = a x^4 + b x^3 + c x^2 + d x + e\) on lists \(X\) and \(Y\) with frequency \(Freq\).
A summary of results is stored in the \texttt{stat.results} variable. (See page 131.)

All the lists must have equal dimension except for \textit{Include}.

\(X\) and \(Y\) are lists of independent and dependent variables.

\(Freq\) is an optional list of frequency values. Each element in \(Freq\) specifies the frequency of occurrence for each corresponding \(X\) and \(Y\) data point. The default value is 1. All elements must be integers \(\geq 0\).

\textit{Category} is a list of numeric or string category codes for the corresponding \(X\) and \(Y\) data.

\textit{Include} is a list of one or more of the category codes. Only those
data items whose category code is included in this list are included in the calculation.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.RegEqn</td>
<td>Regression equation: ( a \cdot x^4 + b \cdot x^3 + c \cdot x^2 + d \cdot x + e )</td>
</tr>
<tr>
<td>stat.a, stat.b, stat.c, stat.d, stat.e</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.R^2</td>
<td>Coefficient of determination</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the regression</td>
</tr>
<tr>
<td>stat.XrReg</td>
<td>List of data points in the modified ( X ) List actually used in the regression based on restrictions of ( Freq, Category ) List, and Include Categories</td>
</tr>
<tr>
<td>stat.YReg</td>
<td>List of data points in the modified ( Y ) List actually used in the regression based on restrictions of ( Freq, Category ) List, and Include Categories</td>
</tr>
<tr>
<td>stat.FreqReg</td>
<td>List of frequencies corresponding to stat.XrReg and stat.YReg</td>
</tr>
</tbody>
</table>

\[ \text{R} \]

\[ \text{R} \gg \text{P\theta}(\cdot) \]

\[ \text{R} \gg \text{P\theta}(xValue, yValue) \Rightarrow value \]
\[ \text{R} \gg \text{P\theta}(xList, yList) \Rightarrow list \]
\[ \text{R} \gg \text{P\theta}(xMatrix, yMatrix) \Rightarrow matrix \]

Returns the equivalent \( \theta \)-coordinate of the \( (x, y) \) pair arguments.

**Note:** The result is returned as a degree, gradian or radian angle, according to the current angle mode setting.

**Note:** You can insert this function from the computer keyboard by typing \( \text{R} \gg \text{P\theta}(\cdot) \).

\[
\begin{align*}
\text{In Degree angle mode:} \\
\text{R} \gg \text{P\theta}(2,2) & \quad \Rightarrow 45. \\
\text{In Gradian angle mode:} \\
\text{R} \gg \text{P\theta}(2,2) & \quad \Rightarrow 50. \\
\text{In Radian angle mode:} \\
\text{R} \gg \text{P\theta}(3,2) & \quad \Rightarrow 0.588003 \\
\text{R} \gg \text{P\theta} \left( \begin{bmatrix} 3 & -4 & 2 \end{bmatrix}, \begin{bmatrix} 0 & \frac{\pi}{4} & 1.5 \end{bmatrix} \right) & \Rightarrow \begin{bmatrix} 0 & 2.94771 & 0.643501 \end{bmatrix}
\end{align*}
\]
\textbf{R►Pr()} \hspace{3cm} \textbf{Catalog >}

\begin{align*}
\text{R►Pr}(x, y) & \Rightarrow \text{value} \\
\text{R►Pr}(x\text{List}, y\text{List}) & \Rightarrow \text{list} \\
\text{R►Pr}(x\text{Matrix}, y\text{Matrix}) & \Rightarrow \text{matrix}
\end{align*}

Returns the equivalent \( r \)-coordinate of the \((x, y)\) pair arguments.

\textbf{Note:} You can insert this function from the computer keyboard by typing \texttt{R@>Pr} (\ldots).

---

\textbf{►Rad} \hspace{3cm} \textbf{Catalog >}

\begin{align*}
\text{Value1} \text{► Rad} & \Rightarrow \text{value} \\
\end{align*}

In Radian angle mode:

\begin{align*}
\text{R►Pr}(3,2) & \equiv 3.60555 \\
\text{R►Pr}\begin{bmatrix}3 & -4 & 2 \end{bmatrix} & \begin{bmatrix}0 & \frac{\pi}{4} & 1.5 \end{bmatrix} \\
& \begin{bmatrix}3 & 4.07638 & \frac{5}{2} \end{bmatrix}
\end{align*}

Converts the argument to radian angle measure.

\textbf{Note:} You can insert this operator from the computer keyboard by typing \texttt{@>Rad}.

---

\textbf{rand()} \hspace{3cm} \textbf{Catalog >}

\begin{align*}
\text{rand()} & \Rightarrow \text{expression} \\
\text{rand(#Trials)} & \Rightarrow \text{list}
\end{align*}

\text{rand()} returns a random value between 0 and 1.

\text{rand(#Trials)} returns a list containing \#Trials random values between 0 and 1.

---

\textbf{randBin()} \hspace{3cm} \textbf{Catalog >}

\begin{align*}
\text{randBin}(n, p) & \Rightarrow \text{expression} \\
\text{randBin}(n, p, \#Trials) & \Rightarrow \text{list}
\end{align*}

\text{randBin}(n, p) returns a random real number from a specified Binomial distribution.

\text{randBin}(n, p, \#Trials) returns a list containing \#Trials random real numbers from a specified Binomial distribution.
### randInt()

<table>
<thead>
<tr>
<th>randInt ( (\text{lowBound}, \text{upBound}) \Rightarrow \text{expression} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>randInt ( (\text{lowBound}, \text{upBound}) \Rightarrow \text{list} )</td>
</tr>
<tr>
<td>randInt ( (\text{lowBound}, \text{upBound}, #\text{Trials}) \Rightarrow \text{list} )</td>
</tr>
</tbody>
</table>

**Description:**
- randInt \( (\text{lowBound}, \text{upBound}) \) returns a random integer within the range specified by lowBound and upBound integer bounds.
- randInt \( (\text{lowBound}, \text{upBound}, \#\text{Trials}) \) returns a list containing \#Trials random integers within the specified range.

**Examples:**
- randInt(3,10) returns 3.
- randInt(3,10,4) returns \( \{9, 3, 4, 7\} \).

### randMat()

<table>
<thead>
<tr>
<th>randMat ( (\text{numRows}, \text{numColumns}) \Rightarrow \text{matrix} )</th>
</tr>
</thead>
</table>

**Description:**
- randMat \( (\text{numRows}, \text{numColumns}) \) returns a matrix of integers between -9 and 9 of the specified dimension.
- Both arguments must simplify to integers.

**Examples:**
- RandSeed 1147 done
- randMat(3,3)
  - \[
  \begin{bmatrix}
  8 & -3 & 6 \\
  -2 & 3 & -6 \\
  0 & 4 & -6 \\
  \end{bmatrix}
  \]

**Note:** The values in this matrix will change each time you press enter.

### randNorm()

<table>
<thead>
<tr>
<th>randNorm ( (\mu, \sigma) \Rightarrow \text{expression} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>randNorm ( (\mu, \sigma, #\text{Trials}) \Rightarrow \text{list} )</td>
</tr>
</tbody>
</table>

**Description:**
- randNorm \( (\mu, \sigma) \) returns a decimal number from the specified normal distribution. It could be any real number but will be heavily concentrated in the interval \( [\mu-3\sigma, \mu+3\sigma] \).
- randNorm \( (\mu, \sigma, \#\text{Trials}) \) returns a list containing \#Trials decimal numbers from the specified normal distribution.

**Examples:**
- RandSeed 1147 done
- randNorm(0,1)
  - 0.492541
- randNorm(3,4.5)
  - -3.54356
**randPoly()**

**randPoly(Var, Order) ⇒ expression**

Returns a polynomial in `Var` of the specified `Order`. The coefficients are random integers in the range \(-9\) through 9. The leading coefficient will not be zero.

`Order` must be 0-99.

**randSamp()**

**randSamp(List,#Trials[,noRepl]) ⇒ list**

Returns a list containing a random sample of `#Trials` trials from `List` with an option for sample replacement (`noRepl=0`), or no sample replacement (`noRepl=1`). The default is with sample replacement.

**RandSeed**

**RandSeed Number**

If `Number = 0`, sets the seeds to the factory defaults for the random-number generator. If `Number ≠ 0`, it is used to generate two seeds, which are stored in system variables `seed1` and `seed2`.

**real()**

**real(Value1) ⇒ value**

Returns the real part of the argument.

**real(List1) ⇒ list**

Returns the real parts of all elements.

**real(Matrix1) ⇒ matrix**

Returns the real parts of all elements.
Vector►Rect

Note: You can insert this operator from the computer keyboard by typing @>Rect.

Displays Vector in rectangular form \([x, y, z]\). The vector must be of dimension 2 or 3 and can be a row or a column.

Note: ►Rect is a display-format instruction, not a conversion function. You can use it only at the end of an entry line, and it does not update ans.

Note: See also ►Polar, page 99.

complexValue►Rect

Displays complexValue in rectangular form \(a+bi\). The complexValue can have any complex form. However, an \(r\angle\theta\) entry causes an error in Degree angle mode.

Note: You must use parentheses for an \((r\angle\theta)\) polar entry.

In Radian angle mode:

\[
\begin{bmatrix}
\pi \\
4 \cdot 3
\end{bmatrix}
\rightarrow\text{Rect}
\]

\[
\begin{bmatrix}
\pi \\
\frac{\pi}{3}
\end{bmatrix}
\rightarrow\text{Rect}
\]

11.3986

2.3+3.4641\cdot i

In Gradian angle mode:

\[
\begin{bmatrix}
1 \angle 100
\end{bmatrix}
\rightarrow\text{Rect}
\]

\[
\begin{bmatrix}
4 \angle 60
\end{bmatrix}
\rightarrow\text{Rect}
\]

2.3+3.4641\cdot i

Note: To type \(\angle\), select it from the symbol list in the Catalog.

ref()

\text{ref}(\text{Matrix}[\text{Mat}]) \rightarrow \text{matrix}

Returns the row echelon form of Matrix1.

Optionally, any matrix element is treated as zero if its absolute value is less than Tol. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, Tol is ignored.

- If you use Ctrl+Enter or set the Auto option.
**Approximate** mode to Approximate, computations are done using floating-point arithmetic.

- If `Tol` is omitted or not used, the default tolerance is calculated as:
  \[5 \times 10^{-14} \times \max(\text{dim}(\text{Matrix}1)) \times \text{rowNorm}(\text{Matrix}1)\]

Avoid undefined elements in `Matrix1`. They can lead to unexpected results.

For example, if `a` is undefined in the following expression, a warning message appears and the result is shown as:

\[
\begin{bmatrix}
  a & 1 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1 \\
\end{bmatrix}
\]

The warning appears because the generalized element `1/a` would not be valid for `a=0`.

You can avoid this by storing a value to `a` beforehand or by using the constraint `(^T)` operator to substitute a value, as shown in the following example.

\[
\begin{bmatrix}
  a & 1 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1 \\
\end{bmatrix} \left| a=0 \right.
\]

Note: See also `ref()`, page 118.

---

**`remain()`**

`remain(Value1, Value2) ⇒ value`

`remain(List1, List2) ⇒ list`

`remain(Matrix1, Matrix2) ⇒ matrix`

Returns the remainder of the first argument with respect to the second argument as defined by the identities:

- `remain(x, 0) x`
- `remain(x, y) x - y \cdot iPart(x/y)`

<table>
<thead>
<tr>
<th>Value1</th>
<th>Value2</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
<td>remain(7,0) 7</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>remain(7,3) 1</td>
</tr>
<tr>
<td>-7</td>
<td>-3</td>
<td>remain(-7,-3) 1</td>
</tr>
<tr>
<td>-7</td>
<td>-3</td>
<td>remain(-7,-3) 1</td>
</tr>
<tr>
<td>{12, 14, 16}</td>
<td>{9, 7, 5}</td>
<td>{3, 0, 1}</td>
</tr>
</tbody>
</table>
As a consequence, note that \( \text{remain}(x, y) - \text{remain}(x, y) \). The result is either zero or it has the same sign as the first argument.

**Note:** See also \( \text{mod}(\)\), page 85.

---

**Request**

\[ \text{Request} \text{promptString}, \text{var}[, \text{DispFlag} [, \text{statusVar}]] \]

\[ \text{Request} \text{promptString}, \text{func}(\text{arg1}, \ldots \text{argn})[, \text{DispFlag} [, \text{statusVar}]] \]

Programming command: Pauses the program and displays a dialog box containing the message \( \text{promptString} \) and an input box for the user’s response.

When the user types a response and clicks **OK**, the contents of the input box are assigned to variable \( \text{var} \).

If the user clicks **Cancel**, the program proceeds without accepting any input. The program uses the previous value of \( \text{var} \) if \( \text{var} \) was already defined.

The optional \( \text{DispFlag} \) argument can be any expression.

- If \( \text{DispFlag} \) is omitted or evaluates to 1, the prompt message and user’s response are displayed in the Calculator history.
- If \( \text{DispFlag} \) evaluates to 0, the prompt and response are not displayed in the history.

The optional \( \text{statusVar} \) argument gives the program a way to determine how the user dismissed the dialog box. Note that \( \text{statusVar} \) requires the \( \text{DispFlag} \) argument.

- If the user clicked **OK** or pressed **Enter** or **Ctrl+Enter**, variable \( \text{statusVar} \) is set to a value of 1.
- Otherwise, variable \( \text{statusVar} \) is set to a value of 0.

The \( \text{func}() \) argument allows a program to store the user’s response as a function definition. This syntax operates as if the user executed the command:
**Request**

Define $\text{func}(\text{arg1}, \ldots, \text{argn}) = \text{user's response}$

The program can then use the defined function $\text{func()}$. The $\text{promptString}$ should guide the user to enter an appropriate user's response that completes the function definition.

**Note:** You can use the Request command within a user-defined program but not within a function.

To stop a program that contains a Request command inside an infinite loop:

- Windows®: Hold down the F12 key and press Enter repeatedly.
- Macintosh®: Hold down the F5 key and press Enter repeatedly.
- Handheld: Hold down the [A on] key and press Enter repeatedly.

**Note:** See also RequestStr, page 113.

---

**RequestStr**

RequestStr $\text{promptString}, \text{var[1, DispFlag]}$

Programming command: Operates identically to the first syntax of the Request command, except that the user's response is always interpreted as a string. By contrast, the Request command interprets the response as an expression unless the user encloses it in quotation marks ("").

**Note:** You can use the RequestStr command within a user-defined program but not within a function.

To stop a program that contains a RequestStr command inside an infinite loop:

- Windows®: Hold down the F12 key and press Enter repeatedly.
- Macintosh®: Hold down the F5 key and press Enter repeatedly.
- Handheld: Hold down the [A on] key and press Enter repeatedly.

**Note:** See also Request, page 112.
RequestStr

requestStr_demo()

Response has 5 characters.

Return

Return[Expr]

Returns Expr as the result of the function. Use within a Func...EndFunc block.

Note: Use Return without an argument within a Prgm...EndPrgm block to exit a program.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing → instead of enter at the end of each line. On the computer keyboard, hold down Alt and press Enter.

right()

right(List1, Num) ⇒ list

Returns the rightmost Num elements contained in List1.
If you omit Num, returns all of List1.

right(sourceString[, Num]) ⇒ string

Returns the rightmost Num characters contained in character string sourceString.
If you omit Num, returns all of sourceString.

right(Comparison) ⇒ expression

Returns the right side of an equation or inequality.

rk23()

rk23(Expr, Var, depVar, {Var0, VarMax}, depVar0, VarStep[, diftol]) ⇒ matrix

rk23(SystemOfExpr, Var, ListOfDepVars, {Var0, VarMax},ListOfDepVars0, VarStep[, diftol]) ⇒

Differential equation:

\[ y' = 0.001y(100-y) \] and \( y(0)=10 \)
\textbf{rk23} \texttt{(ListOfExpr, Var,ListOfDepVars, \{Var0, VarMax\}, ListOfDepVars0, VarStep[, diftol])} \Rightarrow \texttt{matrix}

Uses the Runge-Kutta method to solve the system

\[
\frac{d \text{depVar}}{d \text{Var}} = \text{Expr(Var, depVar)}
\]

with \text{depVar(Var0)}=\text{depVar0} on the interval \text{[Var0, VarMax]}. Returns a matrix whose first row defines the \text{Var} output values as defined by \text{VarStep}. The second row defines the value of the first solution component at the corresponding \text{Var} values, and so on.

\textit{Expr} is the right hand side that defines the ordinary differential equation (ODE).

\textit{SystemOfExpr} is a system of right-hand sides that define the system of ODEs (corresponds to order of dependent variables in ListOfDepVars).

\textit{ListOfExpr} is a list of right-hand sides that define the system of ODEs (corresponds to order of dependent variables in ListOfDepVars).

\textit{Var} is the independent variable.

\textit{ListOfDepVars} is a list of dependent variables.

\{Var0, VarMax\} is a two-element list that tells the function to integrate from Var0 to VarMax.

\textit{ListOfDepVars0} is a list of initial values for dependent variables.

If \text{VarStep} evaluates to a nonzero number: \text{sign(VarStep)} = \text{sign(VarMax-Var0)} and solutions are returned at \text{Var0+i*VarStep} for all \text{i=0,1,2,...} such that \text{Var0+i*VarStep} is in \text{[Var0,VarMax]} (may not get a solution value at VarMax).

If \text{VarStep} evaluates to zero, solutions are returned at the "Runge-Kutta" \text{Var} values.

\textit{diftol} is the error tolerance (defaults to 0.001).

\begin{verbatim}
To see the entire result, press \textasciitilde and then use \textasciitilde and \textasciitilde to move the cursor.

\texttt{System of equations:
\begin{align*}
y' &= -y + 0.1 \cdot y \cdot y^2 \\
y^2 &= 3 \cdot y^2 - y \cdot y^2
\end{align*}
with y(0)=2 and y(2)=5}
\end{verbatim}

\begin{verbatim}
\texttt{ \texttt{rk23(0.001*y-(100-y),\{0,100\},10,1.0e-6)}
0. 1. 2. 3. 4. 10. 10.9367 11.9493 13.042 14.2189

To see the entire result, press \textasciitilde and then use \textasciitilde and \textasciitilde to move the cursor.

\texttt{System of equations:
\begin{align*}
y' &= -y + 0.1 \cdot y \cdot y^2 \\
y^2 &= 3 \cdot y^2 - y \cdot y^2
\end{align*}
with y(0)=2 and y(2)=5}
\end{verbatim}

\begin{verbatim}
\texttt{ \texttt{rk23(0.001*y-(100-y),\{0,100\},10,1.0e-6)}
0. 1. 2. 3. 4. 10. 10.9367 11.9493 13.042 14.2189

To see the entire result, press \textasciitilde and then use \textasciitilde and \textasciitilde to move the cursor.

\texttt{System of equations:
\begin{align*}
y' &= -y + 0.1 \cdot y \cdot y^2 \\
y^2 &= 3 \cdot y^2 - y \cdot y^2
\end{align*}
with y(0)=2 and y(2)=5}
\end{verbatim}

\begin{verbatim}
\texttt{ \texttt{rk23(0.001*y-(100-y),\{0,100\},10,1.0e-6)}
0. 1. 2. 3. 4. 10. 10.9367 11.9493 13.042 14.2189

To see the entire result, press \textasciitilde and then use \textasciitilde and \textasciitilde to move the cursor.

\texttt{System of equations:
\begin{align*}
y' &= -y + 0.1 \cdot y \cdot y^2 \\
y^2 &= 3 \cdot y^2 - y \cdot y^2
\end{align*}
with y(0)=2 and y(2)=5}
\end{verbatim}

\begin{verbatim}
\texttt{ \texttt{rk23(0.001*y-(100-y),\{0,100\},10,1.0e-6)}
0. 1. 2. 3. 4. 10. 10.9367 11.9493 13.042 14.2189

To see the entire result, press \textasciitilde and then use \textasciitilde and \textasciitilde to move the cursor.

\texttt{System of equations:
\begin{align*}
y' &= -y + 0.1 \cdot y \cdot y^2 \\
y^2 &= 3 \cdot y^2 - y \cdot y^2
\end{align*}
with y(0)=2 and y(2)=5}
\end{verbatim}
root()

\[ \text{root}(\text{Value}) \Rightarrow \text{root} \]
\[ \text{root}(\text{Value}_1, \text{Value}_2) \Rightarrow \text{root} \]

root(\text{Value}) returns the square root of \text{Value}.

root(\text{Value}_1, \text{Value}_2) returns the \text{Value}_2 root of \text{Value}_1. \text{Value}_1 can be a real or complex floating point constant or an integer or complex rational constant.

\textbf{Note:} See also \textbf{Nth root template}, page 6.

rotate()

\[ \text{rotate}(\text{Integer}_1[,\text{#ofRotations}]) \Rightarrow \text{integer} \]

Rotates the bits in a binary integer. You can enter \text{Integer}_1 in any number base; it is converted automatically to a signed, 64-bit binary form. If the magnitude of \text{Integer}_1 is too large for this form, a symmetric modulo operation brings it within the range. For more information, see \textbf{Base2}, page 20.

If \text{#ofRotations} is positive, the rotation is to the left. If \text{#ofRotations} is negative, the rotation is to the right.

The default is \texttt{-1} (rotate right one bit).

For example, in a right rotation:

Each bit rotates right.

0b00000000000001111010110000110101

Rightmost bit rotates to leftmost.

produces:

0b1000000000000111010110000110101

The result is displayed according to the Base mode.

\textbf{Important:} To enter a binary or hexadecimal number, always use the \texttt{0b} or \texttt{0h} prefix (zero, not the letter \texttt{O}).

rotate(\text{List}_1[,\text{#ofRotations}]) \Rightarrow \text{list}

Returns a copy of \text{List}_1 rotated right or left by \text{#ofRotations} elements. Does not alter \text{List}_1.

If \text{#ofRotations} is positive, the rotation is to the left. If \text{#ofRotations} is negative, the rotation is to the right.

The default is \texttt{-1} (rotate right one element).
rotate()  
\[ \text{rotate}(\text{String1}, \#\text{ofRotations}) \Rightarrow \text{string} \]  
Returns a copy of \text{String1} rotated right or left by \#\text{ofRotations} characters. Does not alter \text{String1}.

If \#\text{ofRotations} is positive, the rotation is to the left. If \#\text{ofRotations} is negative, the rotation is to the right.  
The default is \(-1\) (rotate right one character).

\[ \begin{align*}  
\text{rotate}("abcd") & \Rightarrow "dabc" \\
\text{rotate}("abcd", -2) & \Rightarrow "cdab" \\
\text{rotate}("abcd", 1) & \Rightarrow "bcda" 
\end{align*} \]

round()  
\[ \text{round}(\text{Value1}, \text{digits}) \Rightarrow \text{value} \]  
Returns the argument rounded to the specified number of digits after the decimal point.

\text{digits} must be an integer in the range 0-12. If \text{digits} is not included, returns the argument rounded to 12 significant digits.

Note: Display digits mode may affect how this is displayed.

\[ \begin{align*}  
\text{round}(1.234567, 3) & \Rightarrow 1.235 \\
\text{round}(\{\pi, \sqrt{2}, \ln(2)\}, 4) & \Rightarrow \{3.1416, 1.4142, 0.6931\} \\
\text{round}\left(\begin{bmatrix} \ln(5) & \ln(3) \\ \pi & e^1 \end{bmatrix}, 1\right) & \Rightarrow \begin{bmatrix} 1.6 & 1.1 \\ 3.1 & 2.7 \end{bmatrix} 
\end{align*} \]

rowAdd()  
\[ \text{rowAdd}(\text{Matrix1}, r\text{Index1}, r\text{Index2}) \Rightarrow \text{matrix} \]  
Returns a copy of \text{Matrix1} with row \text{rIndex2} replaced by the sum of rows \text{rIndex1} and \text{rIndex2}.

\[ \begin{align*}  
\text{rowAdd}\left(\begin{bmatrix} 3 & 4 \\ -3 & -2 \end{bmatrix}, 1, 2\right) & \Rightarrow \begin{bmatrix} 3 & 4 \\ 0 & 2 \end{bmatrix} 
\end{align*} \]

rowDim()  
\[ \text{rowDim}(\text{Matrix}) \Rightarrow \text{expression} \]  
Returns the number of rows in \text{Matrix}.

Note: See also \text{colDim()}, page 26.

\[ \begin{align*}  
\text{rowDim}(m1) & \Rightarrow 3 
\end{align*} \]
### rowNorm()

**rowNorm**(*Matrix*) \(\Rightarrow expression\)

Returns the maximum of the sums of the absolute values of the elements in the rows in *Matrix*.

**Note:** All matrix elements must simplify to numbers. See also **colNorm()**, page 26.

<table>
<thead>
<tr>
<th>rowNorm</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>[\begin{bmatrix} -5 &amp; 6 &amp; -7 \ 3 &amp; 4 &amp; 9 \ 9 &amp; -9 &amp; -7 \end{bmatrix}]</td>
<td></td>
</tr>
</tbody>
</table>

### rowSwap()

**rowSwap**(*Matrix1*, *rIndex1*, *rIndex2*) \(\Rightarrow matrix\)

Returns *Matrix1* with rows *rIndex1* and *rIndex2* exchanged.

<table>
<thead>
<tr>
<th>rowSwap(<em>mat</em>,1,3)</th>
<th>5 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>[\begin{bmatrix} 1 &amp; 2 \ 3 &amp; 4 \ 5 &amp; 6 \end{bmatrix}]</td>
<td></td>
</tr>
</tbody>
</table>

### ref()

**ref**(*Matrix1*, *Tol*) \(\Rightarrow matrix\)

Returns the reduced row echelon form of *Matrix1*.

<table>
<thead>
<tr>
<th>ref</th>
<th>66</th>
</tr>
</thead>
<tbody>
<tr>
<td>[\begin{bmatrix} -2 &amp; -2 &amp; 0 &amp; -6 \ 1 &amp; -1 &amp; 9 &amp; -9 \ -5 &amp; 2 &amp; 4 &amp; -4 \end{bmatrix}]</td>
<td></td>
</tr>
</tbody>
</table>

Optionally, any matrix element is treated as zero if its absolute value is less than *Tol*. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, *Tol* is ignored.

- If you use \[\text{ctrl} \text{enter}\] or set the **Auto** or **Approximate** mode to **Approximate**, computations are done using floating-point arithmetic.
- If *Tol* is omitted or not used, the default tolerance is calculated as:
  \[5 \times 10^{-14} \times \text{max}(\text{dim}(*Matrix1*)) \times \text{rowNorm(*Matrix1*)}\]

**Note:** See also **ref()**, page 110.
### sec()

** sec(Value) ⇒ value
sec(List) ⇒ list

Returns the secant of Value or returns a list containing the secants of all elements in List.

**Note:** The argument is interpreted as a degree, gradian or radian angle, according to the current angle mode setting. You can use °, ′, ″ to override the angle mode temporarily.

<table>
<thead>
<tr>
<th>sec(45)</th>
<th>1.41421</th>
</tr>
</thead>
<tbody>
<tr>
<td>sec({1,2,3,4})</td>
<td>{1.00015,1.00081,1.00244}</td>
</tr>
</tbody>
</table>

### sec^−1()

** sec^−1(Value) ⇒ value
sec^−1(List) ⇒ list

Returns the angle whose secant is Value or returns a list containing the inverse secants of each element of List.

**Note:** The result is returned as a degree, gradian, or radian angle, according to the current angle mode setting.

<table>
<thead>
<tr>
<th>sec^−1(1)</th>
<th>0.</th>
</tr>
</thead>
</table>

| sec^−1(sqrt(2)) | 50.0 |

| sec^−1({1,2,5}) | {0,1.04721,3.6944} |

**Note:** You can insert this function from the keyboard by typing arcsec (...).

### sech()

** sech(Value) ⇒ value
sech(List) ⇒ list

Returns the hyperbolic secant of Value or returns a list containing the hyperbolic secants of the List elements.

<table>
<thead>
<tr>
<th>sech(3)</th>
<th>0.099328</th>
</tr>
</thead>
<tbody>
<tr>
<td>sech({1,2,3,4})</td>
<td>{0.648054,0.198522,0.036619}</td>
</tr>
</tbody>
</table>
sech⁻¹

sech⁻¹(Value) ⇒ value
sech⁻¹(List) ⇒ list

Returns the inverse hyperbolic secant of Value or returns a list containing the inverse hyperbolic secants of each element of List.

Note: You can insert this function from the keyboard by typing arcsech ( ... ).

seq

seq(Expr, Var, Low, High[, Step]) ⇒ list

Increments Var from Low through High by an increment of Step, evaluates Expr, and returns the results as a list. The original contents of Var are still there after seq is completed.

The default value for Step = 1.

seqGen

seqGen(Expr, Var, depVar, {Var0, VarMax},
ListofInitTerms
[, VarStep[, CeilingValue]]]) ⇒ list

Generates a list of terms for sequence depVar(Var) =Expr as follows: Increments independent variable Var from Var0 through VarMax by VarStep, evaluates depVar(Var) for corresponding values of Var using the Expr formula and ListofInitTerms, and returns the results as a list.

seqGen(ListOrSystemOfExpr, Var, ListOfDepVars,
{Var0, VarMax} [,
 MatrixOfInitTerms[, VarStep[, CeilingValue]]]]) ⇒ matrix

Generate the first 5 terms of the sequence u(n) = u(n-1)²/2, with u(1)=2 and VarStep=1.

Example in which Var0=2:
seqGen()

Generates a matrix of terms for a system (or list) of sequences \( \text{ListOfDepVars}(\text{Var}) \)
\( = \text{ListOrSystemOfExpr} \) as follows: Increments independent variable \( \text{Var} \) from \( \text{Var0} \) through \( \text{VarMax} \) by \( \text{VarStep} \), evaluates \( \text{ListOfDepVars}(\text{Var}) \) for corresponding values of \( \text{Var} \) using \( \text{ListOrSystemOfExpr} \) formula and \( \text{MatrixOfInitTerms} \), and returns the results as a matrix.

The original contents of \( \text{Var} \) are unchanged after \( \text{seqGen()} \) is completed.

The default value for \( \text{VarStep} = 1 \).

seqn()

\( \text{seqn}(\text{Expr}(u, n[, \text{ListOfInitTerms}[, \text{nMax}[[, \text{CeilingValue}]]])) \Rightarrow \text{list} \)

Generates a list of terms for a sequence \( u(n)=\text{Expr}(u, n) \) as follows: Increments \( n \) from 1 through \( \text{nMax} \) by 1, evaluates \( u(n) \) for corresponding values of \( n \) using the \( \text{Expr}(u, n) \) formula and \( \text{ListOfInitTerms} \), and returns the results as a list.

\( \text{seqn}(\text{Expr}(u[, \text{nMax}[[, \text{CeilingValue}]]])) \Rightarrow \text{list} \)

Generates a list of terms for a non-recursive sequence \( u(n)=\text{Expr}(n) \) as follows: Increments \( n \) from 1 through \( \text{nMax} \) by 1, evaluates \( u(n) \) for corresponding values of \( n \) using the \( \text{Expr}(n) \) formula, and returns the results as a list.

If \( \text{nMax} \) is missing, \( \text{nMax} \) is set to 2500

If \( \text{nMax}=0 \), \( \text{nMax} \) is set to 2500

Note: \( \text{seqn()} \) calls \( \text{seqGen()} \) with \( n0=1 \) and \( nstep=1 \)
setMode()

**setMode**(modeNameInteger, settingInteger) ⇒ integer

setMode(list) ⇒ integer list

Valid only within a function or program.

**setMode**(modeNameInteger, settingInteger)
temporarily sets mode modeNameInteger to the new setting settingInteger, and returns an integer corresponding to the original setting of that mode. The change is limited to the duration of the program/function’s execution.

modeNameInteger specifies which mode you want to set. It must be one of the mode integers from the table below.

settingInteger specifies the new setting for the mode. It must be one of the setting integers listed below for the specific mode you are setting.

**setMode**(list) lets you change multiple settings. list contains pairs of mode integers and setting integers.

**setMode**(list) returns a similar list whose integer pairs represent the original modes and settings.

If you have saved all mode settings with **getMode**(0) → var, you can use **setMode**(var) to restore those settings until the function or program exits. See **getMode()**, page 58.

**Note**: The current mode settings are passed to called subroutines. If any subroutine changes a mode setting, the mode change will be lost when control returns to the calling routine.

**Note for entering the example**: In the Calculator application on the handheld, you can enter multi-line definitions by pressing [→] instead of [enter] at the end of each line. On the computer keyboard, hold down Alt and press Enter.

```
Set the Display Digits to Float1.
setMode(1, Float1)
```

<table>
<thead>
<tr>
<th>Mode Name</th>
<th>Mode Integer</th>
<th>Setting Integers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Digits</td>
<td>1</td>
<td>1=Float, 2=Float1, 3=Float2, 4=Float3, 5=Float4, 6=Float5, 7=Float6, 8=Float7, 9=Float8, 10=Float9, 11=Float10, 12=Float11, 13=Float12,</td>
</tr>
</tbody>
</table>

**Display approximate value of π using the default setting for Display Digits, and then display π with a setting of Fix2. Check to see that the default is restored after the program executes.**

```
Define prog1()=Prgm
Disp π
setMode(1,16)
Disp π
EndPrgm
```

```
prog1()
```

```
3.14159
3.14
```

**Done**
### Mode Name | Mode Integer | Setting Integers
---|---|---
**| ** | 14=Fix0, 15=Fix1, 16=Fix2, 17=Fix3, 18=Fix4, 19=Fix5, 20=Fix6, 21=Fix7, 22=Fix8, 23=Fix9, 24=Fix10, 25=Fix11, 26=Fix12
**Angle** | 2 | 1=Radian, 2=Degree, 3=Gradian
**Exponential Format** | 3 | 1=Normal, 2=Scientific, 3=Engineering
**Real or Complex** | 4 | 1=Real, 2=Rectangular, 3=Polar
**Auto or Approx.** | 5 | 1=Auto, 2=Approximate
**Vector Format** | 6 | 1=Rectangular, 2=Cylindrical, 3=Spherical
**Base** | 7 | 1=Decimal, 2=Hex, 3=Binary

### shift() `shift(Integer1, #ofShifts) ⇒ integer`
Shifts the bits in a binary integer. You can enter `Integer1` in any number base; it is converted automatically to a signed, 64-bit binary form. If the magnitude of `Integer1` is too large for this form, a symmetric modulo operation brings it within the range. For more information, see ► Base2, page 20.

If `#ofShifts` is positive, the shift is to the left. If `#ofShifts` is negative, the shift is to the right. The default is −1 (shift right one bit).

In a right shift, the rightmost bit is dropped and 0 or 1 is inserted to match the leftmost bit. In a left shift, the leftmost bit is dropped and 0 is inserted as the rightmost bit.

For example, in a right shift:

- Each bit shifts right.
- `0b0000000000000111101011000011010` produces:
  - In Bin base mode:
    - `shift(0b111101011000011010)`
    - `0b111101011000011010`
    - `shift(256,1)`
    - `0b1000000000`
  - In Hex base mode:
    - `shift(0h78E)`
    - `0h3C7`
    - `shift(0h78E, 2)`
    - `0h1E3`
    - `shift(0h78E, 2)`
    - `0h1E38`

**Important:** To enter a binary or hexadecimal number, always use the `0b` or `0h` prefix (zero, not the letter `O`).

Alphabetical Listing 123
The result is displayed according to the Base mode. Leading zeros are not shown.

```
shift(List1, #ofShifts) ⇒ list
```

Returns a copy of List1 shifted right or left by #ofShifts elements. Does not alter List1.

If #ofShifts is positive, the shift is to the left. If #ofShifts is negative, the shift is to the right. The default is −1 (shift right one element).

Elements introduced at the beginning or end of list by the shift are set to the symbol "undef".

```
shift(String1, #ofShifts) ⇒ string
```

Returns a copy of String1 shifted right or left by #ofShifts characters. Does not alter String1.

If #ofShifts is positive, the shift is to the left. If #ofShifts is negative, the shift is to the right. The default is −1 (shift right one character).

Characters introduced at the beginning or end of string by the shift are set to a space.

```
sign(Value1) ⇒ value
sign(List1) ⇒ list
sign(Matrix1) ⇒ matrix
```

For real and complex Value1, returns \( \frac{Value1}{\text{abs}(Value1)} \) when \( Value1 \neq 0 \).

Returns 1 if Value1 is positive. Returns −1 if Value1 is negative. sign(0) returns ±1 if the complex format mode is Real; otherwise, it returns itself.

sign(0) represents the unit circle in the complex domain.

For a list or matrix, returns the signs of all the elements.
**simult()**

\[ \text{simult(coeffMatrix, constVector[, Tol])} \Rightarrow \text{matrix} \]

Returns a column vector that contains the solutions to a system of linear equations.

Note: See also \text{linSolve()}, page 73.

\textit{coeffMatrix} must be a square matrix that contains the coefficients of the equations.

\textit{constVector} must have the same number of rows (same dimension) as \textit{coeffMatrix} and contain the constants.

Optionally, any matrix element is treated as zero if its absolute value is less than \textit{Tol}. This tolerance is used only if the matrix has floating-point entries and does not contain any symbolic variables that have not been assigned a value. Otherwise, \textit{Tol} is ignored.

- If you set the \text{Auto or Approximate} mode to \text{Approximate}, computations are done using floating-point arithmetic.
- If \textit{Tol} is omitted or not used, the default tolerance is calculated as:
  \[ 5 \times 10^{-14} \times \max(\text{dim(coeffMatrix)}) \times \text{rowNorm(coeffMatrix)} \]

\[ \text{simult(coeffMatrix, constMatrix[, Tol])} \Rightarrow \text{matrix} \]

Solves multiple systems of linear equations, where each system has the same equation coefficients but different constants.

Each column in \textit{constMatrix} must contain the constants for a system of equations. Each column in the resulting matrix contains the solution for the corresponding system.

Solve for \( x \) and \( y \):
\[
\begin{align*}
3x + 4y &= -1 \\
nx + 2y &= 1 \\
3x + 4y &= -1 \\
\end{align*}
\]

\[
\text{simult}\left( \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right) \Rightarrow \begin{bmatrix} -3 \\ 2 \end{bmatrix}
\]

The solution is \( x=-3 \) and \( y=2 \).

Solve:
\[
\begin{align*}
ax + by &= 1 \\
3x + 4y &= 2 \\
\end{align*}
\]

\[
\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \Rightarrow \text{mat}x1 \\
\text{simult}\left( \text{mat}x1, \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right) \Rightarrow \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}
\]

Solve:
\[
\begin{align*}
x + 2y &= 1 \\
3x + 4y &= -1 \\
x + 2y &= 2 \\
3x + 4y &= -3 \\
\end{align*}
\]

\[
\text{simult}\left( \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \begin{bmatrix} 1 & 2 \\ -1 & -3 \end{bmatrix} \right) \Rightarrow \begin{bmatrix} -3 & -7 \\ 2 & 9 \end{bmatrix}
\]

For the first system, \( x=-3 \) and \( y=2 \). For the second system, \( x=-7 \) and \( y=9/2 \).

**sin()**

\[ \text{sin(Value1)} \Rightarrow \text{value} \]

\[ \text{sin(List1)} \Rightarrow \text{list} \]

\text{sin(Value1)} \text{ returns the sine of the argument.}

In Degree angle mode:
\textbf{sin}() \hfill \textbf{key}

\texttt{sin(List1)} returns a list of the sines of all elements in \texttt{List1}.

\textbf{Note}: The argument is interpreted as a degree, gradian or radian angle, according to the current angle mode. You can use °, ′, or ″ to override the angle mode setting temporarily.

\begin{center}
\begin{tabular}{|c|c|}
\hline
\texttt{sin(pi/4)} & 0.707107 \\
\hline
\texttt{sin(45)} & 0.707107 \\
\texttt{sin([0,60,90])} & \{0, 0.866025, 1\} \\
\hline
\end{tabular}
\end{center}

In Gradian angle mode:

\begin{center}
\begin{tabular}{|c|}
\hline
\texttt{sin(50)} & 0.707107 \\
\hline
\end{tabular}
\end{center}

In Radian angle mode:

\begin{center}
\begin{tabular}{|c|}
\hline
\texttt{sin(pi/4)} & 0.707107 \\
\hline
\texttt{sin(45^c)} & 0.707107 \\
\end{tabular}
\end{center}

\texttt{sin(squareMatrix1)} ⇒ \textit{squareMatrix}

Returns the matrix sine of \texttt{squareMatrix1}. This is not the same as calculating the sine of each element. For information about the calculation method, refer to \texttt{cos()}.

\texttt{squareMatrix1} must be diagonalizable. The result always contains floating-point numbers.

\begin{center}
\begin{bmatrix}
1 & 5 & 3 \\
4 & 2 & 1 \\
6 & -2 & 1
\end{bmatrix}
\end{center}

\begin{center}
\begin{bmatrix}
0.9424 & -0.04542 & -0.031999 \\
-0.045492 & 0.949254 & -0.020274 \\
0.048739 & -0.00523 & 0.961051
\end{bmatrix}
\end{center}

\textbf{sin}^{-1}() \hfill \textbf{key}

\texttt{sin^{-1}(Value1)} ⇒ \textit{value}

\texttt{sin^{-1}(List1)} ⇒ \textit{list}

\texttt{sin^{-1}(Value1)} returns the angle whose sine is \texttt{Value1}.

\texttt{sin^{-1}(List1)} returns a list of the inverse sines of each element of \texttt{List1}.

\textbf{Note}: The result is returned as a degree, gradian or radian angle, according to the current angle mode setting.

\textbf{Note}: You can insert this function from the keyboard by typing \texttt{arcsin(...)}.

\texttt{sin^{-1}(squareMatrix1)} ⇒ \textit{squareMatrix}

In Degree angle mode:

\begin{center}
\begin{tabular}{|c|}
\hline
\texttt{sin^{-1}(1)} & 90. \\
\hline
\end{tabular}
\end{center}

In Gradian angle mode:

\begin{center}
\begin{tabular}{|c|}
\hline
\texttt{sin^{-1}(1)} & 100. \\
\hline
\end{tabular}
\end{center}

In Radian angle mode:

\begin{center}
\begin{tabular}{|c|}
\hline
\texttt{sin^{-1}([0,0.2,0.5])} & \{0., 0.201358, 0.523599\} \\
\hline
\end{tabular}
\end{center}

In Radian angle mode and Rectangular complex format mode:
\[ \sin^{-1}(\cdot) \]
Returns the matrix inverse sine of \( \text{squareMatrix1} \). This is not the same as calculating the inverse sine of each element. For information about the calculation method, refer to \( \cos() \).

\( \text{squareMatrix1} \) must be diagonalizable. The result always contains floating-point numbers.

\[ \sinh(\cdot) \]
\( \sinh(\text{Number1}) \Rightarrow \text{value} \)
\( \sinh(\text{List1}) \Rightarrow \text{list} \)

\( \sinh(\text{Value1}) \) returns the hyperbolic sine of the argument.

\( \sinh(\text{List1}) \) returns a list of the hyperbolic sines of each element of \( \text{List1} \).

\( \sinh(\text{squareMatrix1}) \Rightarrow \text{squareMatrix} \)
Returns the matrix hyperbolic sine of \( \text{squareMatrix1} \). This is not the same as calculating the hyperbolic sine of each element. For information about the calculation method, refer to \( \cos() \).

\( \text{squareMatrix1} \) must be diagonalizable. The result always contains floating-point numbers.

\[ \sinh^{-1}(\cdot) \]
\( \sinh^{-1}(\text{Value1}) \Rightarrow \text{value} \)
\( \sinh^{-1}(\text{List1}) \Rightarrow \text{list} \)

\( \sinh^{-1}(\text{Value1}) \) returns the inverse hyperbolic sine of the argument.

\( \sinh^{-1}(\text{List1}) \) returns a list of the inverse hyperbolic sines of each element of \( \text{List1} \).

Note: You can insert this function from the keyboard by typing \( \text{arcsinh}(\text{...}) \).

\( \sinh^{-1}(\text{squareMatrix1}) \Rightarrow \text{squareMatrix} \)
In Radian angle mode:
\(\sinh^{-1}(0)\)

Returns the matrix inverse hyperbolic sine of \(\text{squareMatrix1}\). This is not the same as calculating the inverse hyperbolic sine of each element. For information about the calculation method, refer to \(\text{cos}\) 0.

\(\text{squareMatrix1}\) must be diagonalizable. The result always contains floating-point numbers.

\[
\begin{bmatrix}
1 & 5 & 3 \\
4 & 2 & 1 \\
6 & -2 & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
0.041751 & 2.15557 & 1.1582 \\
1.46382 & 0.926568 & 0.112557 \\
2.75079 & -1.5283 & 0.57268
\end{bmatrix}
\]

**SinReg**

\(\text{SinReg}\ X, \ Y[, \ \text{Iterations},[, \ \text{Period},[, \ \text{Category}, \ \text{Include}]\]]\)

Computes the sinusoidal regression on lists \(X\) and \(Y\). A summary of results is stored in the \text{stat.results} variable. (See page 131.)

All the lists must have equal dimension except for \text{Include}.

\(X\) and \(Y\) are lists of independent and dependent variables.

\text{Iterations} is a value that specifies the maximum number of times (1 through 16) a solution will be attempted. If omitted, 8 is used. Typically, larger values result in better accuracy but longer execution times, and vice versa.

\text{Period} specifies an estimated period. If omitted, the difference between values in \(X\) should be equal and in sequential order. If you specify \text{Period}, the differences between \(x\) values can be unequal.

\text{Category} is a list of numeric or string category codes for the corresponding \(X\) and \(Y\) data.

\text{Include} is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

The output of \text{SinReg} is always in radians, regardless of the angle mode setting.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{stat.RegEqn}</td>
<td>Regression Equation: (a\sin(bx+c)+d)</td>
</tr>
<tr>
<td>Output variable</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>stat.a, stat.b, stat.c, stat.d</td>
<td>Regression coefficients</td>
</tr>
<tr>
<td>stat.Resid</td>
<td>Residuals from the regression</td>
</tr>
<tr>
<td>stat.XReg</td>
<td>List of data points in the modified X List actually used in the regression based on restrictions of Freq, Category List, and Include Categories</td>
</tr>
<tr>
<td>stat.YReg</td>
<td>List of data points in the modified Y List actually used in the regression based on restrictions of Freq, Category List, and Include Categories</td>
</tr>
<tr>
<td>stat.FreqReg</td>
<td>List of frequencies corresponding to stat.XReg and stat.YReg</td>
</tr>
</tbody>
</table>

**SortA**

**SortA** List1, List2, List3...

Sorts the elements of the first argument in ascending order.

If you include additional arguments, sorts the elements of each so that their new positions match the new positions of the elements in the first argument.

All arguments must be names of lists or vectors. All arguments must have equal dimensions.

Empty (void) elements within the first argument move to the bottom. For more information on empty elements, see page 177.

**SortD**

**SortD** List1, List2, List3...

Identical to SortA, except SortD sorts the elements in descending order.

Empty (void) elements within the first argument move to the bottom. For more information on empty elements, see page 177.
**Sphere**

*Vector* ► **Sphere**

**Note:** You can insert this operator from the computer keyboard by typing @> Sphere.

Displays the row or column vector in spherical form \([ρ \angle θ \angle φ]\).

*Vector* must be of dimension 3 and can be either a row or a column vector.

**Note:** ► **Sphere** is a display-format instruction, not a conversion function. You can use it only at the end of an entry line.

![Spherical Vector Diagram](image)

**sqrt()**

**sqrt(Value) ⇒ value**

**sqrt(List) ⇒ list**

Returns the square root of the argument.

For a list, returns the square roots of all the elements in List1.

**Note:** See also *Square root template*, page 5.
The results are displayed as a set of name-value pairs. The specific names shown are dependent on the most recently evaluated statistics function or command.

You can copy a name or value and paste it into other locations.

**Note:** Avoid defining variables that use the same names as those used for statistical analysis. In some cases, an error condition could occur. Variable names used for statistical analysis are listed in the table below.

<table>
<thead>
<tr>
<th>stat.a</th>
<th>stat.dfDenom</th>
<th>stat.MedianY</th>
<th>stat.Q3X</th>
<th>stat.SSBlock</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.AdjR²</td>
<td>stat.dfBlock</td>
<td>stat.MEPred</td>
<td>stat.Q3Y</td>
<td>stat.SSCol</td>
</tr>
<tr>
<td>stat.b</td>
<td>stat.dfCol</td>
<td>stat.MinX</td>
<td>stat.r</td>
<td>stat.SS</td>
</tr>
<tr>
<td>stat.b0</td>
<td>stat.dfError</td>
<td>stat.MinY</td>
<td>stat.r²</td>
<td>stat.SSY</td>
</tr>
<tr>
<td>stat.b1</td>
<td>stat.dfInteract</td>
<td>stat.MS</td>
<td>stat.RegEqn</td>
<td>stat.SSError</td>
</tr>
<tr>
<td>stat.b2</td>
<td>stat.dfReg</td>
<td>stat.MSSBlock</td>
<td>stat.Resid</td>
<td>stat.SSInteract</td>
</tr>
<tr>
<td>stat.b3</td>
<td>stat.dfNumer</td>
<td>stat.MSCol</td>
<td>stat.ResidTrans</td>
<td>stat.SSReg</td>
</tr>
<tr>
<td>stat.b4</td>
<td>stat.dfRow</td>
<td>stat.MSError</td>
<td>stat.σx</td>
<td>stat.SOCK</td>
</tr>
<tr>
<td>stat.b5</td>
<td>stat.DW</td>
<td>stat.MSInteract</td>
<td>stat.σy</td>
<td>stat.tList</td>
</tr>
<tr>
<td>stat.b6</td>
<td>stat.e</td>
<td>stat.MSReg</td>
<td>stat.σx1</td>
<td>stat.UpperPred</td>
</tr>
<tr>
<td>stat.b7</td>
<td>stat.ExpMatrix</td>
<td>stat.MSRaw</td>
<td>stat.αx²</td>
<td>stat.UpperVal</td>
</tr>
<tr>
<td>stat.b8</td>
<td>stat.F</td>
<td>stat.n</td>
<td>stat.Σx</td>
<td>stat.χ</td>
</tr>
<tr>
<td>stat.b9</td>
<td>stat.FBlock</td>
<td>Stat. ̂β</td>
<td>stat.Σx²</td>
<td>stat.X1</td>
</tr>
<tr>
<td>stat.b10</td>
<td>stat.Fcol</td>
<td>stat. ̂β1</td>
<td>stat.Σxy</td>
<td>stat.χ2</td>
</tr>
<tr>
<td>stat.bList</td>
<td>stat.FInteract</td>
<td>stat. ̂β2</td>
<td>stat.Σy</td>
<td>stat.χList</td>
</tr>
<tr>
<td>stat.χ²</td>
<td>stat.FreqReg</td>
<td>stat. ̂βDiff</td>
<td>stat.Σy²</td>
<td>stat.χList</td>
</tr>
<tr>
<td>stat.c</td>
<td>stat.Frow</td>
<td>stat.PList</td>
<td>stat.s</td>
<td>stat.XReg</td>
</tr>
<tr>
<td>stat.CLower</td>
<td>stat.Leverage</td>
<td>stat.PVal</td>
<td>stat.SE</td>
<td>stat.XVal</td>
</tr>
<tr>
<td>stat.CLowerList</td>
<td>stat.LowerPred</td>
<td>stat.PValBlock</td>
<td>stat.SEList</td>
<td>stat.XValList</td>
</tr>
<tr>
<td>stat.CompList</td>
<td>stat.LowerVal</td>
<td>stat.PValCol</td>
<td>stat.SEPred</td>
<td>stat.̄y</td>
</tr>
<tr>
<td>stat.CompMatrix</td>
<td>stat.m</td>
<td>stat.PValInteract</td>
<td>stat.sResid</td>
<td>stat.¿</td>
</tr>
</tbody>
</table>

```
xlist: {1,2,3,4,5} {1,2,3,4,5}
ylist: {4,8,11,14,17} {4,8,11,14,17}
LinRegMx xlist,ylist,1: stat.results
| "Title" | "Linear Regression (mx+b)"
| "RegEqn" | "m*x+b"
| "m" | 3.2 |
| "b" | 1.2 |
| "r²" | 0.996109 |
| "r" | 0.998053 |
| "Resid" | "{...}"
```

```
stat.values | "Linear Regression (mx+b)"
| "m*x+b" | 3.2 |
| 1.2 |
| 0.996109 |
| 0.998053 |
| "{0.4,0.4,0.2,0.0, 0.2}" |
```
stat.CookDist  stat.MaxX  stat.PValRow  stat.SEslope  stat.$\bar{y}$List
stat.CUpper  stat.MaxY  stat.Q1X  stat.sp  stat.YReg
stat.CUpperList  stat.ME  stat.Q1Y  stat.SS
stat.d  stat.MedianX

**Note:** Each time the Lists & Spreadsheet application calculates statistical results, it copies the "stat," group variables to a "stat#," group, where # is a number that is incremented automatically. This lets you maintain previous results while performing multiple calculations.

**stat.values**

**stat.values**

Displays a matrix of the values calculated for the most recently evaluated statistics function or command.

Unlike **stat.results, stat.values** omits the names associated with the values.

You can copy a value and paste it into other locations.

**stDevPop()**

\[
\text{stDevPop}(\text{List[,freqList]}) \Rightarrow \text{expression}
\]

Returns the population standard deviation of the elements in List.

Each freqList element counts the number of consecutive occurrences of the corresponding element in List.

**Note:** List must have at least two elements. Empty (void) elements are ignored. For more information on empty elements, see page 177.

\[
\text{stDevPop}(\text{Matrix1[,freqMatrix]}) \Rightarrow \text{matrix}
\]

Returns a row vector of the population standard deviations of the columns in Matrix1.

Each freqMatrix element counts the number of consecutive occurrences of the corresponding element in Matrix1.

**Note:** Matrix1 must have at least two rows. Empty (void) elements are ignored. For more information on empty elements, see page 177.
stDevSamp()  

\[ \text{stDevSamp}(\text{List}, \text{freqList}) \Rightarrow \text{expression} \]

Returns the sample standard deviation of the elements in List.

Each freqList element counts the number of consecutive occurrences of the corresponding element in List.

**Note:** List must have at least two elements. Empty (void) elements are ignored. For more information on empty elements, see page 177.

\[ \text{stDevSamp}(\text{Matrix}1, \text{freqMatrix}) \Rightarrow \text{matrix} \]

Returns a row vector of the sample standard deviations of the columns in Matrix1.

Each freqMatrix element counts the number of consecutive occurrences of the corresponding element in Matrix1.

**Note:** Matrix1 must have at least two rows. Empty (void) elements are ignored. For more information on empty elements, see page 177.

**Stop**

Programming command: Terminates the program.

Stop is not allowed in functions.

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing \[ \text{start} \] instead of \[ \text{enter} \] at the end of each line. On the computer keyboard, hold down Alt and press Enter.

**Store**

See \[ \text{\rightarrow(store)} \], page 175.
**string()**

\[ \text{string}(\text{Expr}) \Rightarrow \text{string} \]

Simplifies \( \text{Expr} \) and returns the result as a character string.

- \( \text{string}(1.2345) \Rightarrow "1.2345" \)
- \( \text{string}(1+2) \Rightarrow "3" \)

**subMat()**

\[ \text{subMat}(\text{Matrix1}[\text{startRow}], \text{startCol}, [\text{endRow}], [\text{endCol}]) \Rightarrow \text{matrix} \]

Returns the specified submatrix of \( \text{Matrix1} \).

Defaults: \( \text{startRow}=1, \text{startCol}=1, \text{endRow}=\text{last row}, \text{endCol}=\text{last column} \).

\[
\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\end{array} \rightarrow m1
\]

\[
\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\end{array}
\]

- \( \text{subMat}(m1, 2, 1, 3, 2) \Rightarrow 4 5 7 8 \)
- \( \text{subMat}(m1, 2, 2) \Rightarrow 5 6 8 9 \)

**Sum (Sigma)**

See \( \Sigma() \), page 168.

**sum()**

\[ \text{sum}([\text{List}, [\text{Start}, [\text{End}]]) \Rightarrow \text{expression} \]

Returns the sum of all elements in \( \text{List} \).

\( \text{Start} \) and \( \text{End} \) are optional. They specify a range of elements.

Any void argument produces a void result. Empty (void) elements in \( \text{List} \) are ignored. For more information on empty elements, see page 177.

- \( \text{sum}([1,2,3,4,5]) \Rightarrow 15 \)
- \( \text{sum}([a,2\cdot a,3\cdot a]) \Rightarrow "\text{Error: Variable is not defined}" \)
- \( \text{sum}(\text{seq}(n,n,1,10)) \Rightarrow 55 \)
- \( \text{sum}([1,3,5,7,9], 3) \Rightarrow 21 \)

- \( \text{sum}\left(\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6 \\
\end{array}\right) \Rightarrow \begin{array}{c}
5 \\
7 \\
9 \\
\end{array} \)
- \( \text{sum}\left(\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\end{array}\right) \Rightarrow \begin{array}{c}
12 \\
15 \\
18 \\
\end{array} \)
- \( \text{sum}\left(\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\end{array}, 2, 3\right) \Rightarrow \begin{array}{c}
11 \\
13 \\
15 \\
\end{array} \)

134  **Alphabetical Listing**
**sumIf()**

**sumIf(List, Criteria[, SumList]) ⇒ value**

Returns the accumulated sum of all elements in `List` that meet the specified `Criteria`. Optionally, you can specify an alternate list, `SumList`, to supply the elements to accumulate.

`List` can be an expression, list, or matrix. `SumList`, if specified, must have the same dimension(s) as `List`.

**Criteria** can be:

- A value, expression, or string. For example, `34` accumulates only those elements in `List` that simplify to the value 34.

- A Boolean expression containing the symbol `?` as a placeholder for each element. For example, `?<10` accumulates only those elements in `List` that are less than 10.

When a `List` element meets the `Criteria`, the element is added to the accumulating sum. If you include `SumList`, the corresponding element from `SumList` is added to the sum instead.

Within the Lists & Spreadsheet application, you can use a range of cells in place of `List` and `SumList`.

Empty (void) elements are ignored. For more information on empty elements, see page 177.

**Note:** See also `countIf()`, page 32.

**sumSeq()**

See Σ(), page 168.

**system()**

**system(Value1[, Value2[, Value3[, ...]]])**

Returns a system of equations, formatted as a list. You can also create a system by using a template.
**T (transpose)**

Matrix₁T ⇒ matrix

Returns the complex conjugate transpose of Matrix₁.

**Note:** You can insert this operator from the computer keyboard by typing `@t`.

**tan()**

\[ \text{tan}(\text{Value₁}) \Rightarrow \text{value} \]
\[ \text{tan}(\text{List₁}) \Rightarrow \text{list} \]

\( \text{tan}(\text{Value₁}) \) returns the tangent of the argument.

\( \text{tan}(\text{List₁}) \) returns a list of the tangents of all elements in List₁.

**Note:** The argument is interpreted as a degree, gradian or radian angle, according to the current angle mode. You can use °, ‰ or ′ to override the angle mode setting temporarily.

\[ \text{tan}(\text{squareMatrix₁}) \Rightarrow \text{squareMatrix} \]

Returns the matrix tangent of squareMatrix₁. This is not the same as calculating the tangent of each element. For information about the calculation method, refer to cos().

squareMatrix₁ must be diagonalizable. The result always contains floating-point numbers.
\[ \tan^{-1}(Value) \] returns the angle whose tangent is \( Value \).

\[ \tan^{-1}(List) \] returns a list of the inverse tangents of each element of \( List \).

Note: The result is returned as a degree, gradian or radian angle, according to the current angle mode setting.

Note: You can insert this function from the keyboard by typing \texttt{arctan(...)}.

\[ \tan^{-1}(squareMatrix) \] returns the matrix inverse tangent of \( squareMatrix \).

This is not the same as calculating the inverse tangent of each element. For information about the calculation method, refer to \texttt{cos()}.

\( squareMatrix \) must be diagonalizable. The result always contains floating-point numbers.

\[ \tanh(Value) \] returns the hyperbolic tangent of the argument.

\[ \tanh(List) \] returns a list of the hyperbolic tangents of each element of \( List \).

\[ \tanh(squareMatrix) \] returns the matrix hyperbolic tangent of \( squareMatrix \).

This is not the same as calculating the hyperbolic tangent of each element. For information about the calculation method, refer to \texttt{cos()}.

\( squareMatrix \) must be diagonalizable. The result always contains floating-point numbers.
tanh⁻¹()  
| tanh⁻¹(Value) ⇒ value |
| tanh⁻¹(List) ⇒ list |

- **tanh⁻¹(Value)** returns the inverse hyperbolic tangent of the argument.
- **tanh⁻¹(List)** returns a list of the inverse hyperbolic tangents of each element of List.

**Note:** You can insert this function from the keyboard by typing arctanh (...).

- **tanh⁻¹(squareMatrix) ⇒ squareMatrix**

  Returns the matrix inverse hyperbolic tangent of squareMatrix. This is not the same as calculating the inverse hyperbolic tangent of each element. For information about the calculation method, refer to cos().

  - squareMatrix must be diagonalizable. The result always contains floating-point numbers.

In Rectangular complex format:

- **tanh⁻¹(0)**
  
  0.

- **tanh⁻¹({1,2,3})**
  
  { undefined, 0.518046−1.5708i, 0.346574−1.570i }

  To see the entire result, press ▲ and then use ▼ and ▲ to move the cursor.

In Radian angle mode and Rectangular complex format:

- **tanh⁻¹(I| 1 5 3 | 4 2 1 | 6 2 1 |)**

  
  
  \[
  \begin{bmatrix}
  -0.099353 + 0.164058i & 0.267834 − 1.4908i \\
  -0.087596 − 0.725533i & 0.479679 − 0.94730i \\
  0.511463 − 2.08316i & 0.878563 + 1.790i
  \end{bmatrix}
  \]

  To see the entire result, press ▲ and then use ▼ and ▲ to move the cursor.

**tCdf()**

- **tCdf(lowBound, upBound, df) ⇒ number if lowBound and upBound are numbers, list if lowBound and upBound are lists**

  Computes the Student-t distribution probability between lowBound and upBound for the specified degrees of freedom df.

  For P(X ≤ upBound), set lowBound = 9E999.

**Text**

- **Text(promptString[, DispFlag])**

  Programming command: Pauses the program and displays the character string promptString in a dialog box.

  - When the user selects OK, program execution continues.
  - The optional flag argument can be any expression.
  - If DispFlag is omitted or evaluates to 1, the text message

  Define a program that pauses to display each of five random numbers in a dialog box.

  Within the Prgm...EndPrgm template, complete each line by pressing [Enter] instead of [enter]. On the computer keyboard, hold down Alt and press Enter.
is added to the Calculator history.

- If DispFlag evaluates to 0, the text message is not added to the history.

If the program needs a typed response from the user, refer to Request, page 112, or RequestStr, page 113.

Note: You can use this command within a user-defined program but not within a function.

Define text_demo()=
For i,1,5
  strinfo:="Random number " & string (rand(i))
  Text strinfo
EndFor
EndPrgm

Run the program:

text_demo()

Sample of one dialog box:

Then

See If, page 61.

tinterval

tinterval List[, Freq[, CLevel]]
(Data list input)

tinterval $\bar{x}$, sx, n[, CLevel]
(Summary stats input)

Computes a $t$ confidence interval. A summary of results is stored in the stat.results variable. (See page 131.)

For information on the effect of empty elements in a list, see "Empty (Void) Elements," page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.CLower, stat.CUpper</td>
<td>Confidence interval for an unknown population mean</td>
</tr>
<tr>
<td>stat.$\bar{x}$</td>
<td>Sample mean of the data sequence from the normal random distribution</td>
</tr>
</tbody>
</table>
### tinterval_2Samp

**tinterval_2Samp** List1,List2[,Freq1[,Freq2[,CLevel[,Pooled]]]]

(Data list input)

**tinterval_2Samp** \( \bar{x}_1, sx_1, n_1, \bar{x}_2, sx_2, n_2[, CLevel[, Pooled]] \)

(Summary stats input)

Computes a two-sample *t* confidence interval. A summary of results is stored in the `stat.results` variable. (See page 131.)

*Pooled*=1 pools variances; *Pooled*=0 does not pool variances.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.CLower, stat.CUpper</td>
<td>Confidence interval containing confidence level probability of distribution</td>
</tr>
<tr>
<td>stat.( \bar{x}_1-\bar{x}_2 )</td>
<td>Sample means of the data sequences from the normal random distribution</td>
</tr>
<tr>
<td>stat.ME</td>
<td>Margin of error</td>
</tr>
<tr>
<td>stat.df</td>
<td>Degrees of freedom</td>
</tr>
<tr>
<td>stat.( \bar{x}_1 ), stat.( \bar{x}_2 )</td>
<td>Sample means of the data sequences from the normal random distribution</td>
</tr>
<tr>
<td>stat.( \sigma x_1 ), stat.( \sigma x_2 )</td>
<td>Sample standard deviations for List 1 and List 2</td>
</tr>
<tr>
<td>stat.n1, stat.n2</td>
<td>Number of samples in data sequences</td>
</tr>
<tr>
<td>stat.sp</td>
<td>The pooled standard deviation. Calculated when <em>Pooled</em> = YES</td>
</tr>
</tbody>
</table>

### tPdf()

**tPdf**()

Computes the probability density function (pdf) for the Student-\( t \) distribution at a specified \( x \) value with specified degrees of freedom \( df \).
trace() \hspace{1cm} \textbf{Catalog >}

\textbf{trace(squareMatrix) ⇒ value}

Returns the trace (sum of all the elements on the main diagonal) of \textit{squareMatrix}.

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}
\]
\[
\begin{bmatrix}
a & 0 \\
1 & a
\end{bmatrix}
\]
\[
a:=12
\]
\[
\text{trace}\begin{bmatrix}
a & 0 \\
1 & a
\end{bmatrix}
\]

Try \hspace{1cm} \textbf{Catalog >}

\textbf{Try}

\textit{block1}

\textbf{Else}

\textit{block2}

\textbf{EndTry}

Executes \textit{block1} unless an error occurs. Program execution transfers to \textit{block2} if an error occurs in \textit{block1}. System variable \textit{errCode} contains the error code to allow the program to perform error recovery. For a list of error codes, see “Error codes and messages,” page 191.

\textit{block1} and \textit{block2} can be either a single statement or a series of statements separated with the “:” character.

\textbf{Note for entering the example:} In the Calculator application on the handheld, you can enter multi-line definitions by pressing \texttt{[·]} instead of \texttt{[enter]} at the end of each line. On the computer keyboard, hold down \texttt{Alt} and press \texttt{Enter}.

To see the commands \texttt{Try}, \texttt{ClrErr}, and \texttt{PassErr} in operation, enter the \texttt{eigenvals()} program shown at the right. Run the program by executing each of the following expressions.

\[
\begin{bmatrix}
-3 \\
-41 \\
5
\end{bmatrix}
\begin{bmatrix}
1 & 2 & -3.1
\end{bmatrix}
\]

\textbf{Note:} See also \texttt{ClrErr}, page 25, and \texttt{PassErr}, page 98.

Define \texttt{progl()}=\texttt{Prgm}

\texttt{Try}

\texttt{z:=z+1}

\texttt{Disp "z incremented."}

\texttt{Else}

\texttt{Disp "Sorry, z undefined."}

\texttt{EndTry}

\texttt{EndPrgm}

Done

\texttt{z:=1:progl()}

z incremented.

Done

\texttt{DelVar z:progl()}

Sorry, z undefined.

Done

Define \texttt{eigenvals(a,b)=Prgm}

© Program \texttt{eigenvals(A,B)} displays eigenvalues of \texttt{A+B}

\texttt{Try}

\texttt{Disp "A= ",a}

\texttt{Disp "B= ",b}

\texttt{Disp " "}

\texttt{Disp "Eigenvalues of A+B are:";eigVl(a*b)}

\texttt{Else}

\texttt{If errCode=230 Then}

\texttt{Disp "Error: Product of A+B must be a square}
tTest: Performs a hypothesis test for a single unknown population mean $\mu$ when the population standard deviation $\sigma$ is unknown. A summary of results is stored in the stat.results variable. (See page 131.)

Test $H_0: \mu = \mu_0$, against one of the following:

For $H_a: \mu < \mu_0$, set Hypoth<0
For $H_a: \mu \neq \mu_0$ (default), set Hypoth=0
For $H_a: \mu > \mu_0$, set Hypoth>0

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.t</td>
<td>$\frac{(\bar{x} - \mu_0)}{(\text{stdev} / \sqrt{n})}$</td>
</tr>
<tr>
<td>stat.PVal</td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>stat.df</td>
<td>Degrees of freedom</td>
</tr>
<tr>
<td>stat.$\bar{x}$</td>
<td>Sample mean of the data sequence in List</td>
</tr>
<tr>
<td>stat.sx</td>
<td>Sample standard deviation of the data sequence</td>
</tr>
<tr>
<td>stat.n</td>
<td>Size of the sample</td>
</tr>
</tbody>
</table>
**tTest_2Samp**

\( t\text{Test}_2\text{Samp} \left[ List1, List2, [Freq1, Freq2, [Hypoth, Pooled]] \right] \)

(Data list input)

\( t\text{Test}_2\text{Samp} \left[ \bar{x}_1, s\text{x}_1, n_1, \bar{x}_2, s\text{x}_2, n_2, [\text{Hypoth}, Pooled] \right] \)

(Summary stats input)

Computes a two-sample \( t \) test. A summary of results is stored in the \textit{stat.results} variable. (See page 131.)

Test \( H_0: \mu_1 = \mu_2 \), against one of the following:

- For \( H_a: \mu_1 < \mu_2 \), set \textit{Hypoth}<0
- For \( H_a: \mu_1 \neq \mu_2 \) (default), set \textit{Hypoth}=0
- For \( H_a: \mu_1 > \mu_2 \), set \textit{Hypoth}>0

\textit{Pooled}=1 pools variances
\textit{Pooled}=0 does not pool variances

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{stat.t}</td>
<td>Standard normal value computed for the difference of means</td>
</tr>
<tr>
<td>\textit{stat.PVal}</td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>\textit{stat.df}</td>
<td>Degrees of freedom for the ( t )-statistic</td>
</tr>
<tr>
<td>\textit{stat.\bar{x}_1, stat.\bar{x}_2}</td>
<td>Sample means of the data sequences in \textit{List 1} and \textit{List 2}</td>
</tr>
<tr>
<td>\textit{stat.sx1, stat.sx2}</td>
<td>Sample standard deviations of the data sequences in \textit{List 1} and \textit{List 2}</td>
</tr>
<tr>
<td>\textit{stat.n1, stat.n2}</td>
<td>Size of the samples</td>
</tr>
<tr>
<td>\textit{stat.sp}</td>
<td>The pooled standard deviation. Calculated when \textit{Pooled}=1.</td>
</tr>
</tbody>
</table>

**tvmFV()**

\( \text{tvmFV}(N, I, PV, Pmt, \{PpY\}, \{CpY\}, \{PmtAt\}) \Rightarrow \text{value} \)

Financial function that calculates the future value of money.

\textbf{Note:} Arguments used in the TVM functions are described in the table of TVM arguments, page 144. See also \texttt{amortTbl()}, page 11.
tvml()  

\( \text{tvml}(N, PV, Pmt, FV, [PpY], [CpY], [PmtAt]) \Rightarrow \text{value} \)

Financial function that calculates the interest rate per year.

**Note:** Arguments used in the TVM functions are described in the table of TVM arguments, page 144. See also `amortTbl()`, page 11.

**Example:**
\[ \text{tvml}(240, 100000, 1000, 0, 12, 12) \Rightarrow 10.5241 \]

---

tvmN()  

\( \text{tvmN}(i, PV, Pmt, FV, [PpY], [CpY], [PmtAt]) \Rightarrow \text{value} \)

Financial function that calculates the number of payment periods.

**Note:** Arguments used in the TVM functions are described in the table of TVM arguments, page 144. See also `amortTbl()`, page 11.

**Example:**
\[ \text{tvmN}(5, 0, 500, 77641, 12, 12) \Rightarrow 120. \]

---

tvmPmt()  

\( \text{tvmPmt}(N, I, PV, FV, [PpY], [CpY], [PmtAt]) \Rightarrow \text{value} \)

Financial function that calculates the amount of each payment.

**Note:** Arguments used in the TVM functions are described in the table of TVM arguments, page 144. See also `amortTbl()`, page 11.

**Example:**
\[ \text{tvmPmt}(60, 4, 30000, 0, 12, 12) \Rightarrow -552.496 \]

---

tvmPV()  

\( \text{tvmPV}(N, I, Pmt, FV, [PpY], [CpY], [PmtAt]) \Rightarrow \text{value} \)

Financial function that calculates the present value.

**Note:** Arguments used in the TVM functions are described in the table of TVM arguments, page 144. See also `amortTbl()`, page 11.

**Example:**
\[ \text{tvmPV}(48, 4, 50030000, 12, 12) \Rightarrow -3426.7 \]

---

<table>
<thead>
<tr>
<th>TVM argument*</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Number of payment periods</td>
<td>real number</td>
</tr>
<tr>
<td>I</td>
<td>Annual interest rate</td>
<td>real number</td>
</tr>
</tbody>
</table>

144  

*Alphabetical Listing*
<table>
<thead>
<tr>
<th>TVM argument*</th>
<th>Description</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PV</td>
<td>Present value</td>
<td>real number</td>
</tr>
<tr>
<td>Pmt</td>
<td>Payment amount</td>
<td>real number</td>
</tr>
<tr>
<td>FV</td>
<td>Future value</td>
<td>real number</td>
</tr>
<tr>
<td>PpY</td>
<td>Payments per year, default=1</td>
<td>integer &gt; 0</td>
</tr>
<tr>
<td>CpY</td>
<td>Compounding periods per year, default=1</td>
<td>integer &gt; 0</td>
</tr>
<tr>
<td>PmtAt</td>
<td>Payment due at the end or beginning of each period, default=end</td>
<td>integer (0=end, 1=beginning)</td>
</tr>
</tbody>
</table>

* These time-value-of-money argument names are similar to the TVM variable names (such as `tvm.pv` and `tvm.pmt`) that are used by the Calculator application's finance solver. Financial functions, however, do not store their argument values or results to the TVM variables.

**TwoVar**

**TwoVar** X, Y[, [Freq][, Category, Include]]

Calculates the TwoVar statistics. A summary of results is stored in the `stat.results` variable. (See page 131.)

All the lists must have equal dimension except for **Include**.

X and Y are lists of independent and dependent variables.

**Freq** is an optional list of frequency values. Each element in **Freq** specifies the frequency of occurrence for each corresponding X and Y data point. The default value is 1. All elements must be integers ≥ 0.

**Category** is a list of numeric category codes for the corresponding X and Y data.

**Include** is a list of one or more of the category codes. Only those data items whose category code is included in this list are included in the calculation.

An empty (void) element in any of the lists X, Freq, or Category results in a void for the corresponding element of all those lists. An empty element in any of the lists X1 through X20 results in a void for the corresponding element of all those lists. For more information on empty elements, see page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.x</td>
<td>Mean of x values</td>
</tr>
<tr>
<td>stat.Σx</td>
<td>Sum of x values</td>
</tr>
<tr>
<td>stat.Σx2</td>
<td>Sum of x2 values</td>
</tr>
<tr>
<td>Output variable</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>stat.sx</td>
<td>Sample standard deviation of $x$</td>
</tr>
<tr>
<td>stat.σx</td>
<td>Population standard deviation of $x$</td>
</tr>
<tr>
<td>stat.n</td>
<td>Number of data points</td>
</tr>
<tr>
<td>stat.Ȳ</td>
<td>Mean of $y$ values</td>
</tr>
<tr>
<td>stat.Σ$y$</td>
<td>Sum of $y$ values</td>
</tr>
<tr>
<td>stat.Σ$y^2$</td>
<td>Sum of $y^2$ values</td>
</tr>
<tr>
<td>stat.sy</td>
<td>Sample standard deviation of $y$</td>
</tr>
<tr>
<td>stat.σy</td>
<td>Population standard deviation of $y$</td>
</tr>
<tr>
<td>stat.Σ$xy$</td>
<td>Sum of $x$•$y$ values</td>
</tr>
<tr>
<td>stat.r</td>
<td>Correlation coefficient</td>
</tr>
<tr>
<td>stat.MinX</td>
<td>Minimum of $x$ values</td>
</tr>
<tr>
<td>stat.Q$_1X$</td>
<td>1st Quartile of $x$</td>
</tr>
<tr>
<td>stat.MedianX</td>
<td>Median of $x$</td>
</tr>
<tr>
<td>stat.Q$_3X$</td>
<td>3rd Quartile of $x$</td>
</tr>
<tr>
<td>stat.MaxX</td>
<td>Maximum of $x$ values</td>
</tr>
<tr>
<td>stat.MinY</td>
<td>Minimum of $y$ values</td>
</tr>
<tr>
<td>stat.Q$_1Y$</td>
<td>1st Quartile of $y$</td>
</tr>
<tr>
<td>stat.MedY</td>
<td>Median of $y$</td>
</tr>
<tr>
<td>stat.Q$_3Y$</td>
<td>3rd Quartile of $y$</td>
</tr>
<tr>
<td>stat.MaxY</td>
<td>Maximum of $y$ values</td>
</tr>
<tr>
<td>stat.Σ(x-Ȳ)$^2$</td>
<td>Sum of squares of deviations from the mean of $x$</td>
</tr>
<tr>
<td>stat.Σ(y-Ȳ)$^2$</td>
<td>Sum of squares of deviations from the mean of $y$</td>
</tr>
</tbody>
</table>

**unitV()**

$\text{unitV}(\text{Vector1}) \Rightarrow \text{vector}$

Returns either a row- or column-unit vector, depending on the form of $\text{Vector1}$.

$\text{Vector1}$ must be either a single-row matrix or a single-column matrix.
unLock

unLock Var1, Var2, Var3...
unLock Var.

Unlocks the specified variables or variable group. Locked variables cannot be modified or deleted.
See Lock, page 76, and getLockInfo(), page 57.

V

varPop()

varPop(List, freqList) ⇒ expression

Returns the population variance of List.

Each freqList element counts the number of consecutive occurrences of the corresponding element in List.

Note: List must contain at least two elements.

If an element in either list is empty (void), that element is ignored, and the corresponding element in the other list is also ignored. For more information on empty elements, see page 177.

varSamp()

varSamp(List, freqList) ⇒ expression

Returns the sample variance of List.

Each freqList element counts the number of consecutive occurrences of the corresponding element in List.

Note: List must contain at least two elements.

If an element in either list is empty (void), that element is ignored, and the corresponding element in the other list is also ignored. For more information on empty elements, see page 177.
**varSamp()**

```
varSamp() ⇒ matrix
```

empty elements, see page 177.

```
varSamp(Matrix1[,freqMatrix]) ⇒ matrix
```

Returns a row vector containing the sample variance of each column in `Matrix1`.

Each `freqMatrix` element counts the number of consecutive occurrences of the corresponding element in `Matrix1`.

If an element in either matrix is empty (void), that element is ignored, and the corresponding element in the other matrix is also ignored. For more information on empty elements, see page 177.

**Note:** `Matrix1` must contain at least two rows.

**W**

**warnCodes()**

```
warnCodes() ⇒ expression
```

```
warnCodes(Expr1, StatusVar) ⇒ expression
```

```
warnCodes(solve(sin(10*x)=x^2/x, x), warn)
```

Evaluates expression `Expr1`, returns the result, and stores the codes of any generated warnings in the `StatusVar` list variable. If no warnings are generated, this function assigns `StatusVar` an empty list.

`Expr1` can be any valid TI-Nspire™ or TI-Nspire™ CAS math expression. You cannot use a command or assignment as `Expr1`.

`StatusVar` must be a valid variable name.

For a list of warning codes and associated messages, see page 191.

**when()**

```
when() ⇒ expression
```

```
when(Condition, trueResult [,falseResult][, unknownResult]) ⇒ expression
```

Returns `trueResult`, `falseResult`, or `unknownResult`, depending on whether `Condition` is true, false, or unknown. Returns the input if there are too few
**when()**

Arguments to specify the appropriate result.

Omit both falseResult and unknownResult to make an expression defined only in the region where Condition is true.

Use an undef falseResult to define an expression that graphs only on an interval.

when() is helpful for defining recursive functions.

```
when(x<0, x+3), x=5      undef
```

**While**

```
While Condition
  Block
EndWhile
```

Executes the statements in Block as long as Condition is true.

Block can be either a single statement or a sequence of statements separated with the "::" character.

**Note for entering the example:** In the Calculator application on the handheld, you can enter multi-line definitions by pressing « instead of enter at the end of each line. On the computer keyboard, hold down Alt and press Enter.

```
Define sum_of_recip[n] = Func
  Local i, tempsum
  1 -> i
  0 -> tempsum
  While i<=n
    tempsum + 1/i -> tempsum
    i + 1 -> i
  EndWhile
  Return tempsum
EndFunc
```

```
  6
```

**xor**

BooleanExpr1 xor BooleanExpr2 returns Boolean expression

BooleanList1 xor BooleanList2 returns Boolean list

BooleanMatrix1 xor BooleanMatrix2 returns Boolean matrix

Returns true if BooleanExpr1 is true and BooleanExpr2 is false, or vice versa.

```
true xor true          false
5>3 xor 3>5          true
```
Returns false if both arguments are true or if both are false. Returns a simplified Boolean expression if either of the arguments cannot be resolved to true or false.

**Note:** See or, page 96.

**Integer1 xor Integer2⇒ integer**

Compares two real integers bit-by-bit using an xor operation. Internally, both integers are converted to signed, 64-bit binary numbers. When corresponding bits are compared, the result is 1 if either bit (but not both) is 1; the result is 0 if both bits are 0 or both bits are 1. The returned value represents the bit results, and is displayed according to the Base mode.

You can enter the integers in any number base. For a binary or hexadecimal entry, you must use the 0b or 0h prefix, respectively. Without a prefix, integers are treated as decimal (base 10).

If you enter a decimal integer that is too large for a signed, 64-bit binary form, a symmetric modulo operation is used to bring the value into the appropriate range. For more information, see ▶ Base2, page 20.

**Note:** See or, page 96.

---

**zInterval**

**zInterval σ,List[,Freq[,CLevel]]**

(Data list input)

**zInterval σ,\bar{x},n [,CLevel]**

(Summary stats input)

Computes a $z$ confidence interval. A summary of results is stored in the stat.results variable. (See page 131.)

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.
### zInterval_1Prop

**Catalog >**

**zInterval_1Prop** \( x, n, [CLlevel] \)

Computes a one-proportion \( z \) confidence interval. A summary of results is stored in the `stat.results` variable. (See page 131.)

\( x \) is a non-negative integer.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stat.CLower</code>, <code>stat.CUpper</code></td>
<td>Confidence interval containing confidence level probability of distribution</td>
</tr>
<tr>
<td><code>stat.\hat{p}</code></td>
<td>The calculated proportion of successes</td>
</tr>
<tr>
<td><code>stat.ME</code></td>
<td>Margin of error</td>
</tr>
<tr>
<td><code>stat.n</code></td>
<td>Number of samples in data sequence</td>
</tr>
</tbody>
</table>

### zInterval_2Prop

**Catalog >**

**zInterval_2Prop** \( x1, n1, x2, n2, [CLlevel] \)

Computes a two-proportion \( z \) confidence interval. A summary of results is stored in the `stat.results` variable. (See page 131.)

\( x1 \) and \( x2 \) are non-negative integers.

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stat.CLower</code>, <code>stat.CUpper</code></td>
<td>Confidence interval containing confidence level probability of distribution</td>
</tr>
<tr>
<td><code>stat.\hat{p} Diff</code></td>
<td>The calculated difference between proportions</td>
</tr>
</tbody>
</table>
### Output variable
<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Margin of error</td>
</tr>
<tr>
<td>First sample proportion estimate</td>
</tr>
<tr>
<td>Second sample proportion estimate</td>
</tr>
<tr>
<td>Sample size in data sequence one</td>
</tr>
<tr>
<td>Sample size in data sequence two</td>
</tr>
</tbody>
</table>

---

#### zInterval\_2Samp

**zInterval\_2Samp** \(\sigma_1, \sigma_2, List1, List2, [Freq1, Freq2, [CLevel]]\)

(Data list input)

**zInterval\_2Samp** \(\sigma_1, \sigma_2, \bar{x}_1, \bar{x}_2, n_1, n_2, [CLevel]\)

(Summary stats input)

Computes a two-sample \(\bar{z}\) confidence interval. A summary of results is stored in the `stat.results` variable. (See page 131.)

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.

---

### Output variable
<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidence interval containing confidence level probability of distribution</td>
</tr>
<tr>
<td>Sample means of the data sequences from the normal random distribution</td>
</tr>
<tr>
<td>Margin of error</td>
</tr>
<tr>
<td>Sample means of the data sequences from the normal random distribution</td>
</tr>
<tr>
<td>Sample standard deviations for List 1 and List 2</td>
</tr>
<tr>
<td>Number of samples in data sequences</td>
</tr>
<tr>
<td>Known population standard deviations for data sequence List 1 and List 2</td>
</tr>
</tbody>
</table>

---

#### zTest

**zTest** \(\mu_0, \sigma, List, [Freq, Hypoth]\)

(Data list input)

**zTest** \(\mu_0, \sigma, \bar{x}, n, [Hypoth]\)

(Summary stats input)

Performs a \(\bar{z}\) test with frequency `freqlist`. A summary of results
zTest

is stored in the stat.results variable. (See page 131.)

Test \( H_0: \mu = \mu_0 \), against one of the following:

For \( H_a: \mu < \mu_0 \), set Hypoth<0
For \( H_a: \mu \neq \mu_0 \) (default), set Hypoth=0
For \( H_a: \mu > \mu_0 \), set Hypoth>0

For information on the effect of empty elements in a list, see “Empty (Void) Elements,” page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.z</td>
<td>((\bar{x} - \mu_0) / (\sigma / \sqrt{n}))</td>
</tr>
<tr>
<td>stat.P Value</td>
<td>Least probability at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>stat.(\bar{x})</td>
<td>Sample mean of the data sequence in List</td>
</tr>
<tr>
<td>stat.sx</td>
<td>Sample standard deviation of the data sequence. Only returned for Data input.</td>
</tr>
<tr>
<td>stat.n</td>
<td>Size of the sample</td>
</tr>
</tbody>
</table>

zTest_1Prop

Computes a two-proportion z test. A summary of results is stored in the stat.results variable. (See page 131.)

\( x1, n1, x2, n2\[,\text{Hypoth}\]

\( x1 \) and \( x2 \) are non-negative integers.

Test \( H_0: p1 = p2 \), against one of the following:
zTest_2Prop

For $H_a: p_1 > p_2$, set $Hypoth>0$
For $H_a: p_1 \neq p_2$ (default), set $Hypoth=0$
For $H_a: p < p_0$, set $Hypoth<0$

For information on the effect of empty elements in a list, see
“Empty (Void) Elements,” page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.z</td>
<td>Standard normal value computed for the difference of proportions</td>
</tr>
<tr>
<td>stat.PVal</td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>stat.$\hat{p}$1</td>
<td>First sample proportion estimate</td>
</tr>
<tr>
<td>stat.$\hat{p}$2</td>
<td>Second sample proportion estimate</td>
</tr>
<tr>
<td>stat.$\hat{p}$</td>
<td>Pooled sample proportion estimate</td>
</tr>
<tr>
<td>stat.n1, stat.n2</td>
<td>Number of samples taken in trials 1 and 2</td>
</tr>
</tbody>
</table>

zTest_2Samp

zTest_2Samp $\sigma_1, \sigma_2, List1, List2[,Freq1[,Freq2[,Hypoth]]]]$

(Data list input)

zTest_2Samp $\sigma_1, \sigma_2, \bar{x}_1, n1, \bar{x}_2, n2[,Hypoth]$

(Summary stats input)

Computes a two-sample $z$ test. A summary of results is stored in
the stat.results variable. (See page 131.)

Test $H_0: \mu_1 = \mu_2$, against one of the following:
For $H_a: \mu_1 < \mu_2$, set $Hypoth<0$
For $H_a: \mu_1 \neq \mu_2$ (default), set $Hypoth=0$
For $H_a: \mu_1 > \mu_2$, $Hypoth>0$

For information on the effect of empty elements in a list, see
“Empty (Void) Elements,” page 177.

<table>
<thead>
<tr>
<th>Output variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.z</td>
<td>Standard normal value computed for the difference of means</td>
</tr>
<tr>
<td>stat.PVal</td>
<td>Smallest level of significance at which the null hypothesis can be rejected</td>
</tr>
<tr>
<td>stat.$\bar{x}$1, stat.$\bar{x}$2</td>
<td>Sample means of the data sequences in List1 and List2</td>
</tr>
<tr>
<td>Output variable</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>stat.sx1, stat.sx2</td>
<td>Sample standard deviations of the data sequences in List1 and List2</td>
</tr>
<tr>
<td>stat.n1, stat.n2</td>
<td>Size of the samples</td>
</tr>
</tbody>
</table>
## Symbols

### + (add)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>
| $Value_1 + Value_2 \Rightarrow value$ | Returns the sum of the two arguments. | $56+4 = 60$
| $List_1 + List_2 \Rightarrow list$ | Returns a list containing the sum of corresponding elements in $List_1$ and $List_2$. | $\left\{22,\frac{\pi}{2}\right\} + 11 = \left\{22,3,14159,1.5708\right\}$
| $Matrix_1 + Matrix_2 \Rightarrow matrix$ | Returns a matrix containing the sum of corresponding elements in $Matrix_1$ and $Matrix_2$. | $\left\{22,\frac{\pi}{2}\right\} + 12 = \left\{10,5,1.5708\right\}$

### - (subtract)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>
| $Value_1 - Value_2 \Rightarrow value$ | Returns $Value_1$ minus $Value_2$. | $6-2 = 4$
| $List_1 - List_2 \Rightarrow list$ | Subtracts each element in $List_2$ from the corresponding element in $List_1$. | $\left\{22,\frac{\pi}{2}\right\} - \left\{10,5,\frac{\pi}{2}\right\} = \left\{12,-1.8584,1.0\right\}$
| $Matrix_1 - Matrix_2 \Rightarrow matrix$ | Subtracts each element in $Matrix_2$ from the corresponding element in $Matrix_1$. | $\left[ \begin{array}{cc} 3 & 4 \\ 1 & 2 \end{array} \right] - \left[ \begin{array}{cc} 2 & 2 \\ 2 & 2 \end{array} \right] = \left[ \begin{array}{cc} 1 & 2 \\ -1 & 0 \end{array} \right]$
− (subtract)

returns the results.

Dimensions of the arguments must be equal.

\[ \text{Value} - \text{List1} \Rightarrow \text{list} \]
\[ \{10,15,20\} - 15 \Rightarrow \{5,0,5\} \]

Subtracts each List1 element from Value or subtracts Value from each List1 element, and returns a list of the results.

\[ \text{Value} - \text{Matrix1} \Rightarrow \text{matrix} \]
\[ \begin{bmatrix} 20 & 1 & 2 \\ 3 & 4 \end{bmatrix} - \begin{bmatrix} 19 & -2 \\ -3 & 16 \end{bmatrix} \]

\[ \text{Value} - \text{Matrix1} \] returns a matrix of Value times the identity matrix minus Matrix1. Matrix1 must be square.

\[ \text{Matrix1} - \text{Value} \Rightarrow \text{matrix} \]

\[ \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} - \begin{bmatrix} 7 & 8 \\ 7 & 8 \\ 7 & 8 \end{bmatrix} \]

\[ \text{Matrix1} - \text{Value} \] returns a matrix of Value times the identity matrix subtracted from Matrix1. Matrix1 must be square.

Note: Use − (dot minus) to subtract an expression from each element.

• (multiply)

\[ \text{Value1} \cdot \text{Value2} \Rightarrow \text{value} \]
\[ 2 \cdot 3.45 \Rightarrow 6.9 \]

Returns the product of the two arguments.

\[ \text{List1} \cdot \text{List2} \Rightarrow \text{list} \]
\[ \{1,2,3\} \cdot \{4,5,6\} \Rightarrow \{4,10,18\} \]

Returns a list containing the products of the corresponding elements in List1 and List2.

Dimensions of the lists must be equal.

\[ \text{Matrix1} \cdot \text{Matrix2} \Rightarrow \text{matrix} \]

Returns the matrix product of Matrix1 and Matrix2.

The number of columns in Matrix1 must equal the number of rows in Matrix2.

\[ \pi \cdot \{4,5,6\} \Rightarrow \{12.5664,15.708,18.8496\} \]

\[ \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 7 & 8 \\ 7 & 8 \\ 7 & 8 \end{bmatrix} \Rightarrow \begin{bmatrix} 42 & 48 \\ 105 & 120 \end{bmatrix} \]

\[ \pi \cdot \{4,5,6\} \]

Returns a list containing the products of Value and
Symbols

\[\times\] key

\(\text{multiply}\)

Each element in \(List1\).

\[\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot 0.01 = \begin{bmatrix} 0.01 & 0.02 \\ 0.03 & 0.04 \end{bmatrix}\]

\[6 \cdot \text{identity}\{3\} = \begin{bmatrix} 6 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 6 \end{bmatrix}\]

\(\text{Note: Use } \cdot \text{(dot multiply) to multiply an expression by each element.}\)

\[\div\] key

\(\text{divide}\)

Returns the quotient of \(Value1\) divided by \(Value2\).

\[
\frac{2}{3.45} = \frac{2}{3.45} = 0.57971
\]

\(\text{List1} / \text{List2} \Rightarrow \text{list}\)

Returns a list containing the quotients of \(List1\) divided by \(List2\).

\[
\frac{\{1,2,3\}}{\{4,5,6\}} = \{0.25, \frac{2}{5}, \frac{2}{5}\}
\]

Dimensions of the lists must be equal.

\(\text{Value} / \text{List1} \Rightarrow \text{list}\)

\[
\frac{6}{\{3,\sqrt{6}\}} = \{2,1,2.44949\}
\]

\(\text{List1} / \text{Value} \Rightarrow \text{list}\)

\[
\{7,9,2\} / 7.92 = \{1,1,1\}
\]

\(\text{Value} / \text{Matrix1} \Rightarrow \text{matrix}\)

\[
\begin{bmatrix} 7 & 9 & 2 \\ 7.92 \end{bmatrix}
\]

\(\text{Matrix1} / \text{Value} \Rightarrow \text{matrix}\)

\[
\begin{bmatrix} 1 \frac{1}{18} & 1 \frac{1}{14} & 1 \frac{1}{63} \\ 18 & 14 & 63 \end{bmatrix}
\]

\(\text{Note: Use } \div \text{ (dot divide) to divide an expression by each element.}\)

\[\wedge\] key

\(\text{power}\)

\(\text{Value1} ^ \text{Value2} \Rightarrow \text{value}\)

\[4^2 = 16\]

\(\text{List1} ^ \text{List2} \Rightarrow \text{list}\)

\[
\{2,4,6\} ^ {\{1,2,3\}} = \{2,16,216\}\]
^ (power)  \(^\text{key}\)

Returns the first argument raised to the power of the second argument.

**Note:** See also **Exponent template**, page 5.

For a list, returns the elements in List1 raised to the power of the corresponding elements in List2.

In the real domain, fractional powers that have reduced exponents with odd denominators use the real branch versus the principal branch for complex mode.

\[
\text{Value} \, ^\bigwedge \text{List1} \Rightarrow \text{list}
\]

Returns Value raised to the power of the elements in List1.

\[
\text{List1} \, ^\bigwedge \text{Value} \Rightarrow \text{list}
\]

Returns the elements in List1 raised to the power of Value.

\[
\text{squareMatrix1} \, ^\bigwedge \text{integer} \Rightarrow \text{matrix}
\]

Returns squareMatrix1 raised to the integer power.

\[
\begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix}^2
\]

\[
\begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix}^{-1}
\]

\[
\begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix}^{-2}
\]

\[
\begin{bmatrix}
7 & 10 \\
15 & 22
\end{bmatrix}
\]

\[
\begin{bmatrix}
-2 & 1 \\
3 & -1
\end{bmatrix}
\]

\[
\begin{bmatrix}
11 & -5 \\
2 & 2
\end{bmatrix}
\]

\[
\begin{bmatrix}
-15 & 7 \\
4 & 4
\end{bmatrix}
\]

x² (square)  \(^\text{x² key}\)

\[
\text{Value}^2 \Rightarrow \text{value}
\]

Returns the square of the argument.

\[
\text{List1}^2 \Rightarrow \text{list}
\]

Returns a list containing the squares of the elements in List1.

\[
\text{squareMatrix1}^2 \Rightarrow \text{matrix}
\]

Returns the matrix square of squareMatrix1. This is not the same as calculating the square of each element. Use \(^\text{key}^2\) to calculate the square of each element.
.\+ (dot add)  \[.\+ \text{ keys}\]

Matrix1 .\+ Matrix2 ⇒ matrix

\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
\end{bmatrix}
+ \begin{bmatrix}
10 & 30 \\
20 & 40 \\
\end{bmatrix}
= \begin{bmatrix}
11 & 32 \\
23 & 44 \\
\end{bmatrix}
\]

Value .\+ Matrix1 ⇒ matrix

\[
5 .\+ \begin{bmatrix}
10 & 30 \\
20 & 40 \\
\end{bmatrix}
= \begin{bmatrix}
15 & 35 \\
25 & 45 \\
\end{bmatrix}
\]

Matrix1 .\+ Matrix2 returns a matrix that is the sum of each pair of corresponding elements in Matrix1 and Matrix2.

Value .\+ Matrix1 returns a matrix that is the sum of Value and each element in Matrix1.

.- (dot subt.)  \[.- \text{ keys}\]

Matrix1 .- Matrix2 ⇒ matrix

\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
\end{bmatrix}
- \begin{bmatrix}
10 & 20 \\
30 & 40 \\
\end{bmatrix}
= \begin{bmatrix}
-9 & -18 \\
-27 & -36 \\
\end{bmatrix}
\]

Value .- Matrix1 ⇒ matrix

\[
5 .- \begin{bmatrix}
10 & 20 \\
30 & 40 \\
\end{bmatrix}
= \begin{bmatrix}
-5 & -15 \\
-25 & -35 \\
\end{bmatrix}
\]

Matrix1 .- Matrix2 returns a matrix that is the difference between each pair of corresponding elements in Matrix1 and Matrix2.

Value .- Matrix1 returns a matrix that is the difference of Value and each element in Matrix1.

\[\text{\times} \text{ (dot mult.)} \quad \text{\times \text{ keys}}\]

Matrix1 .\times Matrix2 ⇒ matrix

\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
\end{bmatrix}
\times \begin{bmatrix}
10 & 20 \\
30 & 40 \\
\end{bmatrix}
= \begin{bmatrix}
10 & 40 \\
90 & 160 \\
\end{bmatrix}
\]

Value .\times Matrix1 ⇒ matrix

\[
5 \times \begin{bmatrix}
10 & 20 \\
30 & 40 \\
\end{bmatrix}
= \begin{bmatrix}
50 & 100 \\
150 & 200 \\
\end{bmatrix}
\]

Matrix1 .\times Matrix2 returns a matrix that is the product of each pair of corresponding elements in Matrix1 and Matrix2.

Value .\times Matrix1 returns a matrix containing the products of Value and each element in Matrix1.

/ (dot divide)  \[./ \text{ keys}\]

Matrix1 ./ Matrix2 ⇒ matrix

\[
\begin{bmatrix}
1 & 2 \\
3 & 4 \\
\end{bmatrix}
./ \begin{bmatrix}
10 & 20 \\
30 & 40 \\
\end{bmatrix}
= \begin{bmatrix}
1 & 1 \\
\frac{1}{10} & \frac{1}{10} \\
\end{bmatrix}
\]

Value ./ Matrix1 ⇒ matrix

\[
5 ./ \begin{bmatrix}
10 & 20 \\
30 & 40 \\
\end{bmatrix}
= \begin{bmatrix}
\frac{1}{2} & \frac{1}{4} \\
\frac{1}{6} & \frac{1}{8} \\
\end{bmatrix}
\]

Matrix1 ./ Matrix2 returns a matrix that is the quotient of each pair of corresponding elements in Matrix1 and Matrix2.

Value ./ Matrix1 returns a matrix that is the quotient of Value and each element in Matrix1.
\(\cdot^\text{(dot power)}\)

\[
\begin{align*}
\text{Matrix1} \cdot^\wedge \text{Matrix2} & \Rightarrow \text{matrix} \\
\text{Value} \cdot^\wedge \text{Matrix1} & \Rightarrow \text{matrix}
\end{align*}
\]

\(\text{Matrix1} \cdot^\wedge \text{Matrix2}\) returns a matrix where each element in \(\text{Matrix2}\) is the exponent for the corresponding element in \(\text{Matrix1}\).

\(\text{Value} \cdot^\wedge \text{Matrix1}\) returns a matrix where each element in \(\text{Matrix1}\) is the exponent for \(\text{Value}\).

\(\neg\text{(negate)}\)

\[
\begin{align*}
\neg \text{Value1} & \Rightarrow \text{value} \\
\neg \text{List1} & \Rightarrow \text{list} \\
\neg \text{Matrix1} & \Rightarrow \text{matrix}
\end{align*}
\]

Returns the negation of the argument.

For a list or matrix, returns all the elements negated.

If the argument is a binary or hexadecimal integer, the negation gives the two’s complement.

\(\%\text{(percent)}\)

\[
\begin{align*}
\text{Value1} \% & \Rightarrow \text{value} \\
\text{List1} \% & \Rightarrow \text{list} \\
\text{Matrix1} \% & \Rightarrow \text{matrix}
\end{align*}
\]

Returns \(100\) for a list or matrix, returns a list or matrix with each element divided by 100.

\(=\text{(equal)}\)

\[
\text{Expr1}=\text{Expr2} \Rightarrow \text{Boolean expression}
\]

Example function that uses math test symbols: \(=, \neq, <, \leq, >, \geq\).
List1=List2 ⇒ Boolean list

Matrix1=Matrix2 ⇒ Boolean matrix

Returns true if Expr1 is determined to be equal to Expr2.

Returns false if Expr1 is determined to be not equal to Expr2.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing \(-\) instead of \(\text{enter}\) at the end of each line. On the computer keyboard, hold down \(\text{Alt}\) and press \(\text{Enter}\).

Expr1≠Expr2 ⇒ Boolean expression

List1≠List2 ⇒ Boolean list

Matrix1≠Matrix2 ⇒ Boolean matrix

Returns true if Expr1 is determined to be not equal to Expr2.

Returns false if Expr1 is determined to be equal to Expr2.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

Note: You can insert this operator from the keyboard by typing /=
< (less than)

Expr1 < Expr2 ⇒ Boolean expression

List1 < List2 ⇒ Boolean list

Matrix1 < Matrix2 ⇒ Boolean matrix

Returns true if Expr1 is determined to be less than Expr2.

Returns false if Expr1 is determined to be greater than or equal to Expr2.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

≤ (less or equal)

Expr1 ≤ Expr2 ⇒ Boolean expression

List1 ≤ List2 ⇒ Boolean list

Matrix1 ≤ Matrix2 ⇒ Boolean matrix

Returns true if Expr1 is determined to be less than or equal to Expr2.

Returns false if Expr1 is determined to be greater than Expr2.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

Note: You can insert this operator from the keyboard by typing <=

> (greater than)

Expr1 > Expr2 ⇒ Boolean expression

List1 > List2 ⇒ Boolean list

Matrix1 > Matrix2 ⇒ Boolean matrix

Returns true if Expr1 is determined to be greater than Expr2.

Returns false if Expr1 is determined to be less than or equal to Expr2.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.
$\geq$ (greater or equal)

$\text{Expr1} \geq \text{Expr2} \Rightarrow \text{Boolean expression}$

$\text{List1} \geq \text{List2} \Rightarrow \text{Boolean list}$

$\text{Matrix1} \geq \text{Matrix2} \Rightarrow \text{Boolean matrix}$

Returns true if $\text{Expr1}$ is determined to be greater than or equal to $\text{Expr2}$.

Returns false if $\text{Expr1}$ is determined to be less than $\text{Expr2}$.

Anything else returns a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

**Note:** You can insert this operator from the keyboard by typing $\geq$

$\Rightarrow$ (logical implication)

$\text{BooleanExpr1} \Rightarrow \text{BooleanExpr2} \Rightarrow \text{Boolean expression}$

$\text{BooleanList1} \Rightarrow \text{BooleanList2} \Rightarrow \text{Boolean list}$

$\text{BooleanMatrix1} \Rightarrow \text{BooleanMatrix2} \Rightarrow \text{Boolean matrix}$

$\text{Integer1} \Rightarrow \text{Integer2} \Rightarrow \text{Integer}$

Evaluates the expression $\text{not} \ <\text{argument1}>\ or\ <\text{argument2}>$ and returns true, false, or a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

**Note:** You can insert this operator from the keyboard by typing $\Rightarrow$
⇔ (logical double implication, XNOR)

BooleanExpr1 ⇔ BooleanExpr2 returns Boolean expression

BooleanList1 ⇔ BooleanList2 returns Boolean list

BooleanMatrix1 ⇔ BooleanMatrix2 returns Boolean matrix

Integer1 ⇔ Integer2 returns Integer

Returns the negation of an XOR Boolean operation on the two arguments. Returns true, false, or a simplified form of the equation.

For lists and matrices, returns comparisons element by element.

**Note:** You can insert this operator from the keyboard by typing <=>

\[5 \land 3 \text{ xor } 3 \land 5\]

\[5 \land 3 \leftrightarrow 3 \land 5\] false

3 \land 4 \rightarrow 7

3 \leftrightarrow 4 \rightarrow -8

\{1,2,3\} \land \{3,2,1\} \rightarrow \{2,0,2\}

\{1,2,3\} \leftrightarrow \{3,2,1\} \rightarrow \{-3,-1,-3\}

Ⅰ (factorial)

Value! ⇒ value

List! ⇒ list

Matrix! ⇒ matrix

Returns the factorial of the argument.

For a list or matrix, returns a list or matrix of factorials of the elements.

String1 & String2 ⇒ string

Returns a text string that is String2 appended to String1.
\[ d() \text{ (derivative)} \]

\[ d(Expr1, \text{Var}, \text{Order}) | \text{Var} = \text{Value} \Rightarrow \text{value} \]

\[ d(Expr1, \text{Var}, \text{Order}) \Rightarrow \text{value} \]

\[ d(List1, \text{Var}, \text{Order}) \Rightarrow \text{list} \]

\[ d(Mat\text{trix}1, \text{Var}, \text{Order}) \Rightarrow \text{matrix} \]

Except when using the first syntax, you must store a numeric value in variable Var before evaluating \( d() \). Refer to the examples.

\( d() \) can be used for calculating first and second order derivative at a point numerically, using auto differentiation methods.

Order, if included, must be 1 or 2. The default is 1.

**Note:** You can insert this function from the keyboard by typing `derivative(...)`. **Note:** See also First derivative, page 9 or Second derivative, page 9.

Note: The \( d() \) algorithm has a limitation: it works recursively through the unsimplified expression, computing the numeric value of the first derivative (and second, if applicable) and the evaluation of each subexpression, which may lead to an unexpected result.

Consider the example on the right. The first derivative of \( x(x^2+x)(1/3) \) at \( x=0 \) is equal to 0. However, because the first derivative of the subexpression \( x(x^2+x)(1/3) \) is undefined at \( x=0 \), and this value is used to calculate the derivative of the total expression, \( d() \) reports the result as undefined and displays a warning message.

If you encounter this limitation, verify the solution graphically. You can also try using `centralDiff()`.

\[ \int() \text{ (integral)} \]

\[ \int(Expr1, \text{Var}, \text{Lower}, \text{Upper}) \Rightarrow \text{value} \]

Returns the integral of \( Expr1 \) with respect to the variable \( \text{Var} \) from \( \text{Lower} \) to \( \text{Upper} \). Can be used to calculate the definite integral numerically, using the
\[ \text{Integral} \] \hspace{1cm} \text{Catalog >}

The same method as \text{nInt}().

**Note:** You can insert this function from the keyboard by typing \text{integral(...)}.  

**Note:** See also \text{nInt()}, page 91, and \text{Definite integral template}, page 10.

\[ \text{Square root} \] \hspace{1cm} \text{ctrl} \text{ sr} \text{ keys}

\[
\sqrt{\text{Value1}} \Rightarrow \text{value} \\
\sqrt{\text{List1}} \Rightarrow \text{list}
\]

Returns the square root of the argument.

For a list, returns the square roots of all the elements in \text{List1}.

**Note:** You can insert this function from the keyboard by typing \text{sqrt}(...)  

**Note:** See also \text{Square root template}, page 5.

\[ \text{Product} \] \hspace{1cm} \text{Catalog >}

\[
\prod(\text{Expr1, Var, Low, High}) \Rightarrow \text{expression}
\]

**Note:** You can insert this function from the keyboard by typing \text{prodSeq}(...) .

Evaluates \text{Expr1} for each value of \text{Var} from \text{Low} to \text{High}, and returns the product of the results.

**Note:** See also \text{Product template (Π)}, page 9.

\[
\prod(\text{Expr1, Var, Low, Low-1}) \Rightarrow 1 \\
\prod(\text{Expr1, Var, Low, High}) \Rightarrow \frac{1}{\prod(\text{Expr1, Var, High+1, Low-1})} \text{ if High < Low-1}
\]

The product formulas used are derived from the following reference:

Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. \textit{Concrete Mathematics: A Foundation}
Π() (prodSeq)

\[
\prod_{k=4}^{1} \left( \frac{1}{k} \right)
\]

\[
\prod_{k=4}^{1} \left( \frac{1}{k} \right) \prod_{k=2}^{4} \left( \frac{1}{k} \right) = \frac{1}{4}
\]

Catalog >

Σ() (sumSeq)

Σ(Expr1, Var, Low, High) ⇒ expression

Note: You can insert this function from the keyboard by typing sumSeq (…).

Evaluates Expr1 for each value of Var from Low to High, and returns the sum of the results.

Note: See also Sum template, page 9.

Σ(Expr1, Var, Low, Low−1) ⇒ 0
Σ(Expr1, Var, Low, High) ⇒ μ
Σ(Expr1, Var, High+1, Low−1) if High < Low−1

The summation formulas used are derived from the following reference:


ΣInt()

ΣInt(NPmt1, NPmt2, N, I, PV, [Pmt], [FV], [PpY], [CpY], [PmtAt], [roundValue]) ⇒ value
ΣInt(1,3,12,4.75,20000,12,12) = 213.48

ΣInt(NPmt1,NPmt2,amortTable) ⇒ value

Amortization function that calculates the sum of the interest during a specified range of payments.
ΣInt

NPmt1 and NPmt2 define the start and end boundaries of the payment range.

N, I, PV, Pmt, FV, PpY, CpY, and PmtAt are described in the table of TVM arguments, page 144.

- If you omit Pmt, it defaults to Pmt=tvmPmt(N, I, PV, FV, PpY, CpY, PmtAt).
- If you omit FV, it defaults to FV=0.
- The defaults for PpY, CpY, and PmtAt are the same as for the TVM functions.

roundValue specifies the number of decimal places for rounding. Default=2.

ΣInt(NPmt1, NPmt2, amortTable) calculates the sum of the interest based on amortization table amortTable. The amortTable argument must be a matrix in the form described under amortTbl(), page 11.

Note: See also ΣPrn(), below, and Bal(), page 19.

ΣPrn

ΣPrn(NPmt1, NPmt2, N, I, PV, Pmt, FV, PpY, CpY, roundValue) ⇒ value

ΣPrn(NPmt1, NPmt2, amortTable) ⇒ value

Amortization function that calculates the sum of the principal during a specified range of payments.

NPmt1 and NPmt2 define the start and end boundaries of the payment range.

N, I, PV, Pmt, FV, PpY, CpY, and PmtAt are described in the table of TVM arguments, page 144.

- If you omit Pmt, it defaults to Pmt=tvmPmt(N, I, PV, FV, PpY, CpY, PmtAt).
- If you omit FV, it defaults to FV=0.
- The defaults for PpY, CpY, and PmtAt are the same as for the TVM functions.

roundValue specifies the number of decimal places for rounding. Default=2.
\( \sum \text{Prn}() \)  

\( \sum \text{Prn}(NPmt1,NPmt2,amortTable) \) calculates the sum of the principal paid based on amortization table \( \text{amortTable} \). The \( \text{amortTable} \) argument must be a matrix in the form described under \( \text{amortTbl}() \), page 11.

**Note:** See also \( \sum \text{Int}() \), above, and \( \text{Bal}() \), page 19.

# (indirection)  

\# varNameString

Refer to the variable whose name is \( \text{varNameString} \). This lets you use strings to create variable names from within a function.

<table>
<thead>
<tr>
<th># varNameString</th>
<th># ( x ) &amp; ( y ) &amp; ( z )</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>xyz := 12</td>
<td>#( &quot;x&quot; &amp; &quot;y&quot; &amp; &quot;z&quot; )</td>
<td>12</td>
</tr>
</tbody>
</table>

Creates or refers to the variable \( \text{xyz} \).

| 10 → \( r \)     | 10 |
| "r" → \# \( s1 \) | "r" |
| \# \( s1 \)       | 10 |

Returns the value of the variable \( (r) \) whose name is stored in variable \( s1 \).

E (scientific notation)  

\( \text{mantissa} \times 10^\text{exponent} \)

Enters a number in scientific notation. The number is interpreted as \( \text{mantissa} \times 10^{\text{exponent}} \).

Hint: If you want to enter a power of 10 without causing a decimal value result, use \( 10^{\text{integer}} \).

**Note:** You can insert this operator from the computer keyboard by typing \( @ \). for example, type \( 2.3 @ E4 \) to enter \( 2.3 \times 10^4 \).

\( g \) (gradian)  

\( \text{Expr}g \Rightarrow \text{expression} \)

\( \text{List}g \Rightarrow \text{list} \)

\( \text{Matrix}g \Rightarrow \text{matrix} \)

In Degree, Gradian or Radian mode:
This function gives you a way to specify a gradian angle while in the Degree or Radian mode.

In Radian angle mode, multiplies \( \text{Expr1} \) by \( \pi/200 \).

In Degree angle mode, multiplies \( \text{Expr1} \) by \( \text{g}/100 \).

In Gradian mode, returns \( \text{Expr1} \) unchanged.

Note: You can insert this symbol from the computer keyboard by typing \( \text{g} \).

\[ \cos\left(50^g\right) \quad 0.707107 \]
\[ \cos\left(\left\{0,100^g,200^g\right\}\right) \quad \left\{1,0,-1\right\} \]

---

In Degree, Gradian or Radian angle mode:

\[ \cos\left(\frac{\pi}{4^\circ}\right) \quad 0.707107 \]
\[ \cos\left(\left\{0^\circ,\frac{\pi}{12^\circ},\left(\pi\right)^\circ\right\}\right) \quad \left\{1,0.965926,-1\right\} \]

---

This function gives you a way to specify a radian angle while in Degree or Gradian mode.

In Degree angle mode, multiplies the argument by \( 180/\pi \).

In Radian angle mode, returns the argument unchanged.

In Gradian mode, multiplies the argument by \( 200/\pi \).

Hint: Use \( \text{r} \) if you want to force radians in a function definition regardless of the mode that prevails when the function is used.

Note: You can insert this symbol from the computer keyboard by typing \( \text{r} \).

Value \( \text{r} \Rightarrow \text{value} \)

List \( \text{r} \Rightarrow \text{list} \)

Matrix \( \text{r} \Rightarrow \text{matrix} \)

---

This function gives you a way to specify a degree angle while in Gradian or Radian mode.

In Degree, Gradian or Radian angle mode:

\[ \cos\left(45^\circ\right) \quad 0.707107 \]

In Radian angle mode:
\( \degree \) (degree) \[ key \]

In Radian angle mode, multiplies the argument by \( \pi/180 \).

In Degree angle mode, returns the argument unchanged.

In Gradian angle mode, multiplies the argument by \( 10/9 \).

Note: You can insert this symbol from the computer keyboard by typing \( @d \).

\( °, ', " \) (degree/minute/second) \[ ctrl \ text 6 \ keys \]

\( dd \° mm′ ss.\″ \Rightarrow \) expression

\( dd \) A positive or negative number
\( mm \) A non-negative number
\( ss.\″ \) A non-negative number

Returns \( dd + (mm/60) + (ss.\″/3600) \).

This base-60 entry format lets you:

\( • \) Enter an angle in degrees/minutes/seconds without regard to the current angle mode.
\( • \) Enter time as hours/minutes/seconds.

Note: Follow \( ss.\″ \) with two apostrophes ("), not a quote symbol (").

\( \angle \) (angle) \[ ctrl \ text 6 \ keys \]

\[ [\text{Radius}, \angle \_\text{Angle}] \Rightarrow \text{vector} \]

(polar input)

\[ [\text{Radius}, \angle \_\text{Angle}, Z\_\text{Coordinate}] \Rightarrow \text{vector} \]

(cylindrical input)

\[ [\text{Radius}, \angle \_\text{Angle}, \angle \_\text{Angle}] \Rightarrow \text{vector} \]

(spherical input)

Returns coordinates as a vector depending on the Vector Format mode setting: rectangular, cylindrical, or spherical.

Note: You can insert this symbol from the computer keyboard by typing \( @< \).
(magnitude \( \angle \text{Angle} \)) \( \Rightarrow \) complexValue

(polar input)

Enters a complex value in \((r \angle \theta)\) polar form. The Angle is interpreted according to the current Angle mode setting.

\[
\begin{bmatrix}
5 & \angle 60^\circ & \angle 45^\circ \\
5. & 1.0472 & 0.785398
\end{bmatrix}
\]

In Radian angle mode and Rectangular complex format:

\[
5 + 3 \cdot i \left( 10 \angle \frac{\pi}{4} \right) = 2.07107 - 4.07107 \cdot i
\]

_ (underscore as an empty element)

See “Empty (Void) Elements,” page 177.

\[10^\()\]

\[10^\( \text{Value1} \) \Rightarrow \text{value}\]

\[10^\( \text{List1} \) \Rightarrow \text{list}\]

Returns 10 raised to the power of the argument.

For a list, returns 10 raised to the power of the elements in List1.

\[10^\( \text{squareMatrix1} \) \Rightarrow \text{squareMatrix}\]

Returns 10 raised to the power of squareMatrix1. This is not the same as calculating 10 raised to the power of each element. For information about the calculation method, refer to \( \cos() \).

squareMatrix1 must be diagonalizable. The result always contains floating-point numbers.

\[^\(^{-1}\) (reciprocal)\]

\[\text{Value1} ^\(^{-1}\) \Rightarrow \text{value}\]

\[\text{List1} ^\(^{-1}\) \Rightarrow \text{list}\]

Returns the reciprocal of the argument.

For a list, returns the reciprocals of the elements in List1.
$\text{squareMatrix} \, ^{-1} \Rightarrow \text{squareMatrix}$

Returns the inverse of \text{squareMatrix} \, 1. \text{squareMatrix} \, 1 must be a non-singular square matrix.

| (constraint operator)

\[
\begin{align*}
\text{Expr} | \text{BooleanExpr1} [\text{and} \text{ BooleanExpr2}]... \\
\text{Expr} | \text{BooleanExpr1} [\text{or} \text{BooleanExpr2}]...
\end{align*}
\]

The constraint ("\mid") symbol serves as a binary operator. The operand to the left of \mid is an expression. The operand to the right of \mid specifies one or more relations that are intended to affect the simplification of the expression. Multiple relations after \mid must be joined by logical "\text{and}" or "\text{or}" operators.

The constraint operator provides three basic types of functionality:

- Substitutions
- Interval constraints
- Exclusions

Substitutions are in the form of an equality, such as \text{x}=3 or \text{y}=\sin(x). To be most effective, the left side should be a simple variable. \text{Expr} \mid \text{Variable} = \text{value} will substitute \text{value} for every occurrence of \text{Variable} in \text{Expr}.

Interval constraints take the form of one or more inequalities joined by logical "\text{and}" or "\text{or}" operators. Interval constraints also permit simplification that otherwise might be invalid or not computable.

Exclusions use the "not equals" (/= or ≠) relational operator to exclude a specific value from consideration.

174 Symbols
If the variable $Var$ does not exist, creates it and initializes it to $Value$, $List$, or $Matrix$.

If the variable $Var$ already exists and is not locked or protected, replaces its contents with $Value$, $List$, or $Matrix$.

**Note:** You can insert this operator from the keyboard by typing `:=` as a shortcut. For example, type $\pi/4 =: myvar$.
© (comment)

© [text]

© processes text as a comment line, allowing you to annotate functions and programs that you create.

© can be at the beginning or anywhere in the line. Everything to the right of ©, to the end of the line, is the comment.

Note for entering the example: In the Calculator application on the handheld, you can enter multi-line definitions by pressing → instead of enter at the end of each line. On the computer keyboard, hold down Alt and press Enter.

Define \( g(n) = \text{Func} \)

© Declare variables
Local \( i, \) result
result:=0
For \( i, 1, n, 1 \) © Loop \( n \) times
result:=result\( +i^2 \)
EndFor
Return result
EndFunc

g(3)

14

0b, 0h

0b binaryNumber
0h hexadecimalNumber

Denotes a binary or hexadecimal number, respectively. To enter a binary or hex number, you must enter the 0b or 0h prefix regardless of the Base mode. Without a prefix, a number is treated as decimal (base 10).

Results are displayed according to the Base mode.

In Dec base mode:

\[
\begin{array}{c}
0b10+0hF+10 \\
27
\end{array}
\]

In Bin base mode:

\[
\begin{array}{c}
0b10+0hF+10 \\
0b11011
\end{array}
\]

In Hex base mode:

\[
\begin{array}{c}
0b10+0hF+10 \\
0h1B
\end{array}
\]
Empty (Void) Elements

When analyzing real-world data, you might not always have a complete data set. TI-Nspire™ Software allows empty, or void, data elements so you can proceed with the nearly complete data rather than having to start over or discard the incomplete cases.

You can find an example of data involving empty elements in the Lists & Spreadsheet chapter, under “Graphing spreadsheet data.”

The delVoid() function lets you remove empty elements from a list. The isVoid() function lets you test for an empty element. For details, see delVoid(), page 41, and isVoid(), page 66.

Note: To enter an empty element manually in a math expression, type “_” or the keyword void. The keyword void is automatically converted to a “_” symbol when the expression is evaluated. To type “_” on the handheld, press [ctrl] [\].

### Calculations involving void elements

The majority of calculations involving a void input will produce a void result. See special cases below.

<table>
<thead>
<tr>
<th>Calculation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>_</td>
</tr>
<tr>
<td>gcd(100,_)</td>
<td>_</td>
</tr>
<tr>
<td>3+_</td>
<td>_</td>
</tr>
<tr>
<td>{5,_,10} - {3,6,9}</td>
<td>{2,_,1}</td>
</tr>
</tbody>
</table>

### List arguments containing void elements

The following functions and commands ignore (skip) void elements found in list arguments.

- `count`, `countIf`, `cumulativeSum`, `freqTable`, `list`, `frequency`, `max`, `mean`, `median`, `product`, `stDevPop`, `stDevSamp`, `sum`, `sumIf`, `varPop`, and `varSamp`, as well as regression calculations, `OneVar`, `TwoVar`, and `FiveNumSummary` statistics, confidence intervals, and stat tests

<table>
<thead>
<tr>
<th>Function</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sum</code> ({2,_,3,5,6,6})</td>
<td></td>
<td>16.6</td>
</tr>
<tr>
<td><code>median</code> ({1,2,<em>,</em>,_,3})</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td><code>cumulativeSum</code> ({1,2,_,4,5})</td>
<td>{1,3,_,7,12}</td>
<td></td>
</tr>
<tr>
<td><code>cumulativeSum</code> {1,2,3,_,5,6}</td>
<td>{1,2,4,_,9,8}</td>
<td></td>
</tr>
</tbody>
</table>

`SortA` and `SortD` move all void elements within the first argument to the bottom.

- \{5,4,3,_,1\} → list1
- \{5,4,3,2,1\} → list2

`SortA list1,list2` → Done

<table>
<thead>
<tr>
<th>list1</th>
<th>list2</th>
</tr>
</thead>
<tbody>
<tr>
<td>{5,4,3,_,1}</td>
<td>{5,4,3,_,1}</td>
</tr>
<tr>
<td>{5,4,3,2,1}</td>
<td>{5,4,3,2,1}</td>
</tr>
</tbody>
</table>

`list1` \{1,3,4,5,_,\}

`list2` \{1,3,4,5,2\}
List arguments containing void elements

In regressions, a void in an X or Y list introduces a void for the corresponding element of the residual.

\[
\begin{align*}
\{1,2,3,\_\_\_\_\}_1 & \rightarrow \text{list1} & \{1,2,3,5\} \\
\{1,2,3,4,5\} & \rightarrow \text{list2} & \{1,2,3,4,5\} \\
\text{SortD list1,list2} & \text{ Done} \\
\text{list1} & \{5,3,2,1,\_\_\_\} \\
\text{list2} & \{5,3,2,1,4\} \\
\end{align*}
\]

\[
II:=\{1,2,3,4,5\}; \ t_2:=\{2,\_\_\_\_\_\_\}_2 \\
\{2,\_\_\_\_\_\_\}_2 \\
\text{LinRegMx II,t2} \quad \text{Done} \\
\text{stat.Resid} \\
\{0.434286, -0.862857, 0.014290, 0.44\} \\
\text{stat.XReg} \quad \{1,\_\_\_\_\_\_\, 4,5,\} \\
\text{stat.YReg} \quad \{2,\_\_\_\_\_\_\, 5,6,6\} \\
\text{stat.FreqReg} \quad \{1,\_\_\_\_\_\_\, 1,1\} \\
\]

An omitted category in regressions introduces a void for the corresponding element of the residual.

\[
\begin{align*}
\text{II:}\{1,3,4,5\}; \ t_2:=\{2,3,5,6,6\} \\
\text{cat:=}\{"M", "M", "F", "F"\}; \ incl:=\{"F"\} \\
\{"F"\} \\
\text{LinRegMx II,t2,1,cat,incl} \quad \text{Done} \\
\text{stat.Resid} \quad \{\_\_,\_\_,\_\_,\_\_,0,0\} \\
\text{stat.XReg} \quad \{\_\_,\_\_,4,5,\} \\
\text{stat.YReg} \quad \{\_\_,5,6,6\} \\
\text{stat.FreqReg} \quad \{\_\_,\_\_,1,1\} \\
\end{align*}
\]

A frequency of 0 in regressions introduces a void for the corresponding element of the residual.

\[
\begin{align*}
\text{II:}\{1,3,4,5\}; \ t_2:=\{2,3,5,6,6\} \\
\text{LinRegMx II,t2,1,0,1,1} \quad \text{Done} \\
\text{stat.Resid} \quad \{0.069231, -0.276923, 0.207692\} \\
\text{stat.XReg} \quad \{1,\_\_,4,5,\} \\
\text{stat.YReg} \quad \{2,\_\_,5,6,6\} \\
\text{stat.FreqReg} \quad \{1,\_\_,1,1\} \\
\end{align*}
\]
Shortcuts for Entering Math Expressions

Shortcuts let you enter elements of math expressions by typing instead of using the Catalog or Symbol Palette. For example, to enter the expression $\sqrt{6}$, you can type `sqrt(6)` on the entry line. When you press `enter`, the expression `sqrt(6)` is changed to $\sqrt{6}$. Some shortcuts are useful from both the handheld and the computer keyboard. Others are useful primarily from the computer keyboard.

From the Handheld or Computer Keyboard

<table>
<thead>
<tr>
<th>To enter this:</th>
<th>Type this shortcut:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi$</td>
<td>pi</td>
</tr>
<tr>
<td>$\theta$</td>
<td>theta</td>
</tr>
<tr>
<td>$\infty$</td>
<td>infinity</td>
</tr>
<tr>
<td>$\leq$</td>
<td>&lt;=</td>
</tr>
<tr>
<td>$\geq$</td>
<td>&gt;=</td>
</tr>
<tr>
<td>$\neq$</td>
<td>/=</td>
</tr>
<tr>
<td>$\Rightarrow$ (logical implication)</td>
<td>=&gt;</td>
</tr>
</tbody>
</table>
| $\leftrightarrow$ (logical double implication, XNOR) | <=>
| $\rightarrow$ (store operator) | =: |
| $||$ (absolute value) | abs(…) |
| $\sqrt{}$      | sqrt(…)            |
| $\Sigma()$ (Sum template) | sumSeq(…) |
| $\Pi()$ (Product template) | prodSeq(…) |
| $\sin^{-1}(), \cos^{-1}(), \ldots$ | arcsin(…), arccos(…), … |
| $\Delta List()$ | deltaList(…) |

Shortcuts for Entering Math Expressions 179
From the Computer Keyboard

<table>
<thead>
<tr>
<th>To enter this:</th>
<th>Type this shortcut:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$ (imaginary constant)</td>
<td>@i</td>
</tr>
<tr>
<td>$e$ (natural log base e)</td>
<td>@e</td>
</tr>
<tr>
<td>$E$ (scientific notation)</td>
<td>@E</td>
</tr>
<tr>
<td>$T$ (transpose)</td>
<td>@t</td>
</tr>
<tr>
<td>$r$ (radians)</td>
<td>@r</td>
</tr>
<tr>
<td>$°$ (degrees)</td>
<td>@d</td>
</tr>
<tr>
<td>$g$ (gradians)</td>
<td>@g</td>
</tr>
<tr>
<td>$∠$ (angle)</td>
<td>@&lt;</td>
</tr>
<tr>
<td>► (conversion)</td>
<td>@&gt;</td>
</tr>
<tr>
<td>►Decimal, ►approxFraction(), and so on.</td>
<td>@&gt;Decimal, @&gt;approxFraction(), and so on.</td>
</tr>
</tbody>
</table>
This section describes the Equation Operating System (EOS™) that is used by the TI-Nspire™ math and science learning technology. Numbers, variables, and functions are entered in a simple, straightforward sequence. EOS™ software evaluates expressions and equations using parenthetical grouping and according to the priorities described below.

### Order of Evaluation

<table>
<thead>
<tr>
<th>Level</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Parentheses ( ), brackets [], braces {}</td>
</tr>
<tr>
<td>2</td>
<td>Indirection (#)</td>
</tr>
<tr>
<td>3</td>
<td>Function calls</td>
</tr>
<tr>
<td>4</td>
<td>Post operators: degrees-minutes-seconds (°, ′, ″), factorial (!), percentage (%), radian (′), subscript ([]), transpose (^T)</td>
</tr>
<tr>
<td>5</td>
<td>Exponentiation, power operator (^)</td>
</tr>
<tr>
<td>6</td>
<td>Negation (⁻)</td>
</tr>
<tr>
<td>7</td>
<td>String concatenation (&amp;)</td>
</tr>
<tr>
<td>8</td>
<td>Multiplication (•), division (/)</td>
</tr>
<tr>
<td>9</td>
<td>Addition (+), subtraction (−)</td>
</tr>
<tr>
<td>10</td>
<td>Equality relations: equal (=), not equal (≠ or /=), less than (&lt;), less than or equal (≤ or ≤=), greater than (&gt;), greater than or equal (≥ or ≥=)</td>
</tr>
<tr>
<td>11</td>
<td>Logical not</td>
</tr>
<tr>
<td>12</td>
<td>Logical and</td>
</tr>
<tr>
<td>13</td>
<td>Logical or</td>
</tr>
<tr>
<td>14</td>
<td>xor, nor, nand</td>
</tr>
<tr>
<td>15</td>
<td>Logical implication (⇒)</td>
</tr>
<tr>
<td>16</td>
<td>Logical double implication, XNOR (⇔)</td>
</tr>
<tr>
<td>17</td>
<td>Constraint operator (“</td>
</tr>
<tr>
<td>18</td>
<td>Store (→)</td>
</tr>
</tbody>
</table>
Parentheses, Brackets, and Braces

All calculations inside a pair of parentheses, brackets, or braces are evaluated first. For example, in the expression 4(1+2), EOS™ software first evaluates the portion of the expression inside the parentheses, 1+2, and then multiplies the result, 3, by 4.

The number of opening and closing parentheses, brackets, and braces must be the same within an expression or equation. If not, an error message is displayed that indicates the missing element. For example, (1+2)/(3+4 will display the error message “Missing )”.

Note: Because the TI-Nspire™ software allows you to define your own functions, a variable name followed by an expression in parentheses is considered a “function call” instead of implied multiplication. For example a(b+c) is the function a evaluated by b+c. To multiply the expression b+c by the variable a, use explicit multiplication: a•(b+c).

Indirection

The indirection operator (#) converts a string to a variable or function name. For example, #("x"&"y"&"z") creates the variable name xyz. Indirection also allows the creation and modification of variables from inside a program. For example, if 10→r and "r"→s1, then #s1=10.

Post Operators

Post operators are operators that come directly after an argument, such as 5!, 25%, or 60°15′ 45″. Arguments followed by a post operator are evaluated at the fourth priority level. For example, in the expression 4^3!, 3! is evaluated first. The result, 6, then becomes the exponent of 4 to yield 4096.

Exponentiation

Exponentiation (^) and element-by-element exponentiation (^.) are evaluated from right to left. For example, the expression 2*3^2 is evaluated the same as 2^(3^2) to produce 512. This is different from (2^3)^2, which is 64.

Negation

To enter a negative number, press [-] followed by the number. Post operations and exponentiation are performed before negation. For example, the result of -x^2 is a negative number, and -9^2 = -81. Use parentheses to square a negative number such as (-9)^2 to produce 81.

Constraint ("|")

The argument following the constraint ("|") operator provides a set of constraints that affect the evaluation of the argument preceding the operator.
Error Codes and Messages

When an error occurs, its code is assigned to variable `errCode`. User-defined programs and functions can examine `errCode` to determine the cause of an error. For an example of using `errCode`, See Example 2 under the **Try** command, page 141.

**Note:** Some error conditions apply only to TI-Nspire™ CAS products, and some apply only to TI-Nspire™ products.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>A function did not return a value</td>
</tr>
<tr>
<td>20</td>
<td>A test did not resolve to TRUE or FALSE. Generally, undefined variables cannot be compared. For example, the test if a&lt;b will cause this error if either a or b is undefined when the If statement is executed.</td>
</tr>
<tr>
<td>30</td>
<td>Argument cannot be a folder name.</td>
</tr>
<tr>
<td>40</td>
<td>Argument error</td>
</tr>
<tr>
<td>50</td>
<td>Argument mismatch Two or more arguments must be of the same type.</td>
</tr>
<tr>
<td>60</td>
<td>Argument must be a Boolean expression or integer</td>
</tr>
<tr>
<td>70</td>
<td>Argument must be a decimal number</td>
</tr>
<tr>
<td>90</td>
<td>Argument must be a list</td>
</tr>
<tr>
<td>100</td>
<td>Argument must be a matrix</td>
</tr>
<tr>
<td>130</td>
<td>Argument must be a string</td>
</tr>
<tr>
<td>140</td>
<td>Argument must be a variable name. Make sure that the name:</td>
</tr>
<tr>
<td></td>
<td>• does not begin with a digit</td>
</tr>
<tr>
<td></td>
<td>• does not contain spaces or special characters</td>
</tr>
<tr>
<td></td>
<td>• does not use underscore or period in invalid manner</td>
</tr>
<tr>
<td></td>
<td>• does not exceed the length limitations</td>
</tr>
<tr>
<td></td>
<td>See the Calculator section in the documentation for more details.</td>
</tr>
<tr>
<td>160</td>
<td>Argument must be an expression</td>
</tr>
<tr>
<td>165</td>
<td>Batteries too low for sending or receiving Install new batteries before sending or receiving.</td>
</tr>
<tr>
<td>170</td>
<td>Bound</td>
</tr>
<tr>
<td>Error code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>180</td>
<td>Break</td>
</tr>
<tr>
<td>190</td>
<td>Circular definition</td>
</tr>
<tr>
<td>200</td>
<td>Constraint expression invalid</td>
</tr>
<tr>
<td>210</td>
<td>Invalid Data type</td>
</tr>
<tr>
<td>220</td>
<td>Dependent limit</td>
</tr>
<tr>
<td>230</td>
<td>Dimension</td>
</tr>
<tr>
<td>235</td>
<td>Dimension Error. Not enough elements in the lists.</td>
</tr>
<tr>
<td>240</td>
<td>Dimension mismatch</td>
</tr>
<tr>
<td>250</td>
<td>Divide by zero</td>
</tr>
<tr>
<td>260</td>
<td>Domain error</td>
</tr>
<tr>
<td>270</td>
<td>Duplicate variable name</td>
</tr>
<tr>
<td>280</td>
<td>Else and ElseIf invalid outside of If...EndIf block</td>
</tr>
<tr>
<td>290</td>
<td>EndTry is missing the matching Else statement</td>
</tr>
<tr>
<td>295</td>
<td>Excessive iteration</td>
</tr>
<tr>
<td>300</td>
<td>Expected 2 or 3-element list or matrix</td>
</tr>
<tr>
<td>310</td>
<td>The first argument of <code>nSolve</code> must be an equation in a single variable. It cannot contain a non-valued variable other than the variable of interest.</td>
</tr>
<tr>
<td>320</td>
<td>First argument of solve or cSolve must be an equation or inequality</td>
</tr>
<tr>
<td>Error code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>345</td>
<td>Inconsistent units</td>
</tr>
<tr>
<td>350</td>
<td>Index out of range</td>
</tr>
<tr>
<td>360</td>
<td>Indirection string is not a valid variable name</td>
</tr>
<tr>
<td>380</td>
<td>Undefined Ans</td>
</tr>
<tr>
<td></td>
<td>Either the previous calculation did not create Ans, or no previous calculation was entered.</td>
</tr>
<tr>
<td>390</td>
<td>Invalid assignment</td>
</tr>
<tr>
<td>400</td>
<td>Invalid assignment value</td>
</tr>
<tr>
<td>410</td>
<td>Invalid command</td>
</tr>
<tr>
<td>430</td>
<td>Invalid for the current mode settings</td>
</tr>
<tr>
<td>435</td>
<td>Invalid guess</td>
</tr>
<tr>
<td>440</td>
<td>Invalid implied multiply</td>
</tr>
<tr>
<td></td>
<td>For example, (x(x+1)) is invalid; whereas, (x^*(x+1)) is the correct syntax. This is to avoid confusion between implied multiplication and function calls.</td>
</tr>
<tr>
<td>450</td>
<td>Invalid in a function or current expression</td>
</tr>
<tr>
<td></td>
<td>Only certain commands are valid in a user-defined function.</td>
</tr>
<tr>
<td>490</td>
<td>Invalid in Try..EndTry block</td>
</tr>
<tr>
<td>510</td>
<td>Invalid list or matrix</td>
</tr>
<tr>
<td>550</td>
<td>Invalid outside function or program</td>
</tr>
<tr>
<td></td>
<td>A number of commands are not valid outside a function or program. For example, <code>Local</code> cannot be used unless it is in a function or program.</td>
</tr>
<tr>
<td>560</td>
<td>Invalid outside Loop..EndLoop, For..EndFor, or While..EndWhile blocks</td>
</tr>
<tr>
<td></td>
<td>For example, the Exit command is valid only inside these loop blocks.</td>
</tr>
<tr>
<td>565</td>
<td>Invalid outside program</td>
</tr>
<tr>
<td>570</td>
<td>Invalid pathname</td>
</tr>
<tr>
<td></td>
<td>For example, <code>\var</code> is invalid.</td>
</tr>
<tr>
<td>575</td>
<td>Invalid polar complex</td>
</tr>
<tr>
<td>580</td>
<td>Invalid program reference</td>
</tr>
<tr>
<td></td>
<td>Programs cannot be referenced within functions or expressions such as (1+p(x)) where (p) is a program.</td>
</tr>
<tr>
<td>600</td>
<td>Invalid table</td>
</tr>
<tr>
<td>605</td>
<td>Invalid use of units</td>
</tr>
<tr>
<td>610</td>
<td>Invalid variable name in a Local statement</td>
</tr>
<tr>
<td>Error code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>620</td>
<td>Invalid variable or function name</td>
</tr>
<tr>
<td>630</td>
<td>Invalid variable reference</td>
</tr>
<tr>
<td>640</td>
<td>Invalid vector syntax</td>
</tr>
<tr>
<td>650</td>
<td>Link transmission&lt;br&gt;A transmission between two units was not completed. Verify that the connecting cable is connected firmly to both ends.</td>
</tr>
<tr>
<td>665</td>
<td>Matrix not diagonalizable</td>
</tr>
<tr>
<td>670</td>
<td>Low Memory&lt;br&gt;1. Delete some data in this document&lt;br&gt;2. Save and close this document&lt;br&gt;If 1 and 2 fail, pull out and re-insert batteries</td>
</tr>
<tr>
<td>672</td>
<td>Resource exhaustion</td>
</tr>
<tr>
<td>673</td>
<td>Resource exhaustion</td>
</tr>
<tr>
<td>680</td>
<td>Missing (</td>
</tr>
<tr>
<td>690</td>
<td>Missing )</td>
</tr>
<tr>
<td>700</td>
<td>Missing “</td>
</tr>
<tr>
<td>710</td>
<td>Missing ]</td>
</tr>
<tr>
<td>720</td>
<td>Missing }</td>
</tr>
<tr>
<td>730</td>
<td>Missing start or end of block syntax</td>
</tr>
<tr>
<td>740</td>
<td>Missing Then in the If..EndIf block</td>
</tr>
<tr>
<td>750</td>
<td>Name is not a function or program</td>
</tr>
<tr>
<td>765</td>
<td>No functions selected</td>
</tr>
<tr>
<td>780</td>
<td>No solution found</td>
</tr>
<tr>
<td>800</td>
<td>Non-real result&lt;br&gt;For example, if the software is in the Real setting, $\sqrt{-1}$ is invalid.&lt;br&gt;To allow complex results, change the “Real or Complex” Mode Setting to RECTANGULAR or POLAR.</td>
</tr>
<tr>
<td>830</td>
<td>Overflow</td>
</tr>
<tr>
<td>850</td>
<td>Program not found&lt;br&gt;A program reference inside another program could not be found in the provided path during execution.</td>
</tr>
<tr>
<td>855</td>
<td>Rand type functions not allowed in graphing</td>
</tr>
<tr>
<td>Error code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>860</td>
<td>Recursion too deep</td>
</tr>
<tr>
<td>870</td>
<td>Reserved name or system variable</td>
</tr>
<tr>
<td>900</td>
<td>Argument error</td>
</tr>
<tr>
<td></td>
<td>Median-median model could not be applied to data set.</td>
</tr>
<tr>
<td>910</td>
<td>Syntax error</td>
</tr>
<tr>
<td>920</td>
<td>Text not found</td>
</tr>
<tr>
<td>930</td>
<td>Too few arguments</td>
</tr>
<tr>
<td></td>
<td>The function or command is missing one or more arguments.</td>
</tr>
<tr>
<td>940</td>
<td>Too many arguments</td>
</tr>
<tr>
<td></td>
<td>The expression or equation contains an excessive number of arguments and cannot be evaluated.</td>
</tr>
<tr>
<td>950</td>
<td>Too many subscripts</td>
</tr>
<tr>
<td>955</td>
<td>Too many undefined variables</td>
</tr>
<tr>
<td>960</td>
<td>Variable is not defined</td>
</tr>
<tr>
<td></td>
<td>No value is assigned to variable. Use one of the following commands:</td>
</tr>
<tr>
<td></td>
<td>• sto →</td>
</tr>
<tr>
<td></td>
<td>• :=</td>
</tr>
<tr>
<td></td>
<td>• Define</td>
</tr>
<tr>
<td></td>
<td>to assign values to variables.</td>
</tr>
<tr>
<td>965</td>
<td>Unlicensed OS</td>
</tr>
<tr>
<td>970</td>
<td>Variable in use so references or changes are not allowed</td>
</tr>
<tr>
<td>980</td>
<td>Variable is protected</td>
</tr>
<tr>
<td>990</td>
<td>Invalid variable name</td>
</tr>
<tr>
<td></td>
<td>Make sure that the name does not exceed the length limitations</td>
</tr>
<tr>
<td>1000</td>
<td>Window variables domain</td>
</tr>
<tr>
<td>1010</td>
<td>Zoom</td>
</tr>
<tr>
<td>1020</td>
<td>Internal error</td>
</tr>
<tr>
<td>1030</td>
<td>Protected memory violation</td>
</tr>
<tr>
<td>1040</td>
<td>Unsupported function. This function requires Computer Algebra System. Try TI-Nspire™ CAS.</td>
</tr>
<tr>
<td>1045</td>
<td>Unsupported operator. This operator requires Computer Algebra System. Try TI-Nspire™ CAS.</td>
</tr>
<tr>
<td>1050</td>
<td>Unsupported feature. This operator requires Computer Algebra System. Try TI-Nspire™ CAS.</td>
</tr>
<tr>
<td>Error code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>1060</td>
<td>Input argument must be numeric. Only inputs containing numeric values are allowed.</td>
</tr>
<tr>
<td>1070</td>
<td>Trig function argument too big for accurate reduction</td>
</tr>
<tr>
<td>1080</td>
<td>Unsupported use of Ans. This application does not support Ans.</td>
</tr>
</tbody>
</table>
| 1090       | Function is not defined. Use one of the following commands:  
|            | • Define  
|            | • :=  
|            | • sto →  
|            | to define a function. |
| 1100       | Non-real calculation  
|            | For example, if the software is in the Real setting, √(-1) is invalid.  
|            | To allow complex results, change the “Real or Complex” Mode Setting to RECTANGULAR or POLAR. |
| 1110       | Invalid bounds |
| 1120       | No sign change |
| 1130       | Argument cannot be a list or matrix |
| 1140       | Argument error  
|            | The first argument must be a polynomial expression in the second argument. If the second argument is omitted, the software attempts to select a default. |
| 1150       | Argument error  
|            | The first two arguments must be polynomial expressions in the third argument. If the third argument is omitted, the software attempts to select a default. |
| 1160       | Invalid library pathname  
|            | A pathname must be in the form xxxyyy, where:  
|            | • The xxx part can have 1 to 16 characters.  
|            | • The yyy part can have 1 to 15 characters.  
|            | See the Library section in the documentation for more details. |
| 1170       | Invalid use of library pathname  
|            | • A value cannot be assigned to a pathname using Define, :=, or sto →.  
|            | • A pathname cannot be declared as a Local variable or be used as a parameter in a function or program definition. |
| 1180       | Invalid library variable name.  
|            | Make sure that the name:  
|            | • Does not contain a period  
<p>|            | • Does not begin with an underscore |</p>
<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Does not exceed 15 characters</td>
<td>See the Library section in the documentation for more details.</td>
</tr>
<tr>
<td>1190 Library document not found:</td>
<td></td>
</tr>
<tr>
<td>• Verify library is in the MyLib folder.</td>
<td></td>
</tr>
<tr>
<td>• Refresh Libraries.</td>
<td>See the Library section in the documentation for more details.</td>
</tr>
<tr>
<td>1200 Library variable not found:</td>
<td></td>
</tr>
<tr>
<td>• Verify library variable exists in the first problem in the library.</td>
<td></td>
</tr>
<tr>
<td>• Make sure library variable has been defined as LibPub or LibPriv.</td>
<td></td>
</tr>
<tr>
<td>• Refresh Libraries.</td>
<td>See the Library section in the documentation for more details.</td>
</tr>
<tr>
<td>1210 Invalid library shortcut name.</td>
<td>Make sure that the name:</td>
</tr>
<tr>
<td>• Does not contain a period</td>
<td></td>
</tr>
<tr>
<td>• Does not begin with an underscore</td>
<td></td>
</tr>
<tr>
<td>• Does not exceed 16 characters</td>
<td></td>
</tr>
<tr>
<td>• Is not a reserved name</td>
<td>See the Library section in the documentation for more details.</td>
</tr>
<tr>
<td>1220 Domain error:</td>
<td>The tangentLine and normalLine functions support real-valued functions only.</td>
</tr>
<tr>
<td>1230 Domain error.</td>
<td>Trigonometric conversion operators are not supported in Degree or Gradian angle modes.</td>
</tr>
<tr>
<td>1250 Argument Error</td>
<td>Use a system of linear equations.</td>
</tr>
<tr>
<td>Example of a system of two linear equations with variables x and y:</td>
<td></td>
</tr>
<tr>
<td>3x+7y=5</td>
<td></td>
</tr>
<tr>
<td>2y-5x=-1</td>
<td></td>
</tr>
<tr>
<td>1260 Argument Error:</td>
<td>The first argument of nfMin or nfMax must be an expression in a single variable. It cannot contain a non-valued variable other than the variable of interest.</td>
</tr>
<tr>
<td>1270 Argument Error</td>
<td>Order of the derivative must be equal to 1 or 2.</td>
</tr>
<tr>
<td>1280 Argument Error</td>
<td></td>
</tr>
<tr>
<td>Error code</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>1290</td>
<td>Argument Error</td>
</tr>
<tr>
<td></td>
<td>Use a polynomial in expanded form in one variable.</td>
</tr>
<tr>
<td>1300</td>
<td>Argument Error</td>
</tr>
<tr>
<td></td>
<td>The coefficients of the polynomial must evaluate to numeric values.</td>
</tr>
<tr>
<td>1310</td>
<td>Argument error:</td>
</tr>
<tr>
<td></td>
<td>A function could not be evaluated for one or more of its arguments.</td>
</tr>
<tr>
<td>1380</td>
<td>Argument error:</td>
</tr>
<tr>
<td></td>
<td>Nested calls to domain() function are not allowed.</td>
</tr>
</tbody>
</table>
Warning Codes and Messages

You can use the `warnCodes()` function to store the codes of warnings generated by evaluating an expression. This table lists each numeric warning code and its associated message. For an example of storing warning codes, see `warnCodes()`, page 148.

<table>
<thead>
<tr>
<th>Warning code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>Operation might introduce false solutions.</td>
</tr>
<tr>
<td>10001</td>
<td>Differentiating an equation may produce a false equation.</td>
</tr>
<tr>
<td>10002</td>
<td>Questionable solution</td>
</tr>
<tr>
<td>10003</td>
<td>Questionable accuracy</td>
</tr>
<tr>
<td>10004</td>
<td>Operation might lose solutions.</td>
</tr>
<tr>
<td>10005</td>
<td>cSolve might specify more zeros.</td>
</tr>
<tr>
<td>10006</td>
<td>Solve may specify more zeros.</td>
</tr>
<tr>
<td>10007</td>
<td>More solutions may exist. Try specifying appropriate lower and upper bounds and/or a guess.</td>
</tr>
</tbody>
</table>
|              | Examples using `solve()`:
<p>|              | • solve(Equation, Var=Guess)||lowBound&lt;Var&lt;upBound |
|              | • solve(Equation, Var)||lowBound&lt;Var&lt;upBound |
|              | • solve(Equation, Var=Guess) |
| 10008        | Domain of the result might be smaller than the domain of the input. |
| 10009        | Domain of the result might be larger than the domain of the input. |
| 10012        | Non-real calculation |
| 10013        | $\infty^0$ or <code>undef^0</code> replaced by 1 |
| 10014        | <code>undef^0</code> replaced by 1 |
| 10015        | $1^\infty$ or <code>1^undef</code> replaced by 1 |
| 10016        | $1^undef$ replaced by 1 |
| 10017        | Overflow replaced by $\infty$ or $-\infty$ |
| 10018        | Operation requires and returns 64 bit value. |
| 10019        | Resource exhaustion, simplification might be incomplete. |
| 10020        | Trig function argument too big for accurate reduction. |
| 10021        | Input contains an undefined parameter. |</p>
<table>
<thead>
<tr>
<th>Warning code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>10022</td>
<td>Specifying appropriate lower and upper bounds might produce a solution.</td>
</tr>
<tr>
<td>10023</td>
<td>Scalar has been multiplied by the identity matrix.</td>
</tr>
<tr>
<td>10024</td>
<td>Result obtained using approximate arithmetic.</td>
</tr>
<tr>
<td>10025</td>
<td>Equivalence cannot be verified in EXACT mode.</td>
</tr>
<tr>
<td>10026</td>
<td>Constraint might be ignored. Specify constraint in the form &quot;\ Variable MathTestSymbol Constant&quot; or a conjunct of these forms, for example ‘x&lt;3 and x&gt;-12’</td>
</tr>
</tbody>
</table>
Support and Service

Texas Instruments Support and Service

General Information: North and South America

Home Page: education.ti.com
KnowledgeBase and e-mail inquiries: education.ti.com/support
Phone: (800) TI-CARES / (800) 842-2737
For North and South America and U.S. Territories

For Technical Support

Knowledge Base and support by e-mail: education.ti.com/support or ti-cares@ti.com
Phone (not toll-free): (972) 917-8324

For Product (Hardware) Service

Customers in the U.S., Canada, Mexico, and U.S. territories: Always contact Texas Instruments Customer Support before returning a product for service.

For All Other Countries:

For general information

For more information about TI products and services, contact TI by e-mail or visit the TI Internet address.

E-mail inquiries: ti-cares@ti.com
Home Page: education.ti.com

Service and Warranty Information

For information about the length and terms of the warranty or about product service, refer to the warranty statement enclosed with this product or contact your local Texas Instruments retailer/distributor.
## Index

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-</code></td>
<td>subtract</td>
<td>156</td>
</tr>
<tr>
<td><code>!</code></td>
<td>factorial</td>
<td>165</td>
</tr>
<tr>
<td><code>&quot;</code></td>
<td>second notation</td>
<td>172</td>
</tr>
<tr>
<td><code>#</code></td>
<td>indirection</td>
<td>170</td>
</tr>
<tr>
<td><code>&amp;</code></td>
<td>append</td>
<td>165</td>
</tr>
<tr>
<td><code>*</code></td>
<td>multiply</td>
<td>157</td>
</tr>
<tr>
<td><code>.-</code></td>
<td>dot subtraction</td>
<td>160</td>
</tr>
<tr>
<td><code>.*</code></td>
<td>dot multiplication</td>
<td>160</td>
</tr>
<tr>
<td><code>./</code></td>
<td>dot division</td>
<td>160</td>
</tr>
<tr>
<td><code>.*</code></td>
<td>dot power</td>
<td>161</td>
</tr>
</tbody>
</table>
Index

.+, dot addition .................................................. 160

/ 

/, divide .......................................................... 158

: 

:=, assign ........................................................ 175

^ 

^-1, reciprocal ............................................... 173

^, power .......................................................... 158

| 

|, constraint operator ........................................ 174

, 

' minute notation ................................................ 172

≠, not equal ..................................................... 162

≤, less than or equal ......................................... 163

≥, greater than or equal .................................... 164

>, greater than ............................................... 163

=, equal .......................................................... 161

Π 

Π, product ........................................................ 167

∑ 

∑(), sum .......................................................... 168

∑Int() .............................................................. 168

196  Index
\[ \sum \text{Prn}() \] \hspace{169}

\[ \sqrt{\text{, square root} \quad 167} \]

\[ \angle \text{ (angle) \quad 172} \]

\[ \int, \text{ integral} \quad 166 \]

\[ \Rightarrow \text{, logical implication} \quad 164, 179 \]

\[ \rightarrow \text{, store variable} \quad 175 \]
⇔, logical double implication .................................................. 165, 179

©, comment ................................................................. 176

°, degree notation .......................................................... 171
°, degrees/minutes/seconds ................................................. 172

0b, binary indicator ......................................................... 176
0h, hexadecimal indicator .................................................. 176

10^(), power of ten .......................................................... 173

2

2-sample F Test .............................................................. 54

A

abs(), absolute value ......................................................... 11
absolutely value
  template for ............................................................... 7-8
add, + ........................................................................... 156
amortization table, amortTbl() ............................................... 11, 19
amortization table, amortTbl() ............................................... 11, 19
and, Boolean operator ....................................................... 12
angle(), angle ................................................................ 12
angle, angle() ................................................................ 12
ANOVA, one-way variance analysis ..................................... 13
ANOVA2way, two-way variance analysis ........................................... 14
Ans, last answer ................................................................. 16
answer (last), Ans ............................................................. 16
append, & ........................................................................ 165
approx(), approximate ...................................................... 16
approximate, approx() ...................................................... 16
approxRational() ............................................................... 17
arccos(), cos⁻¹() ............................................................... 17
arccosh(), cosh⁻¹() ........................................................... 17
arccot(), cot⁻¹() ............................................................... 17
arccoth(), coth⁻¹() ........................................................... 17
arccsc(), csc⁻¹() ............................................................... 17
arccsch(), csch⁻¹() ........................................................... 17
arcsec(), sec⁻¹() ............................................................... 17
arcsech(), csech⁻¹() .......................................................... 17
arcsin(), sin⁻¹() ............................................................... 18
arcsinh(), sinh⁻¹() ............................................................ 18
arctan(), tan⁻¹() ............................................................... 18
arctanh(), tanh⁻¹() ........................................................... 18
arguments in TVM functions .................................................. 144
augment(), augment/concatenate ....................................... 18
augment/concatenate, augment() ....................................... 18
average rate of change, avgRC() .......................................... 18
avgRC(), average rate of change ........................................... 18

B

binary

  display, ►Base2 .................................................................. 20
  indicator, 0b .................................................................. 176
binomCdf() ...................................................................... 22
binomPdf() ...................................................................... 22

Boolean operators

  ⇒ .................................................................................. 164, 179
  ⇔ .................................................................................. 165
  and ............................................................................... 12
<table>
<thead>
<tr>
<th>Operator</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>nand</td>
<td>88</td>
</tr>
<tr>
<td>nor</td>
<td>92</td>
</tr>
<tr>
<td>not</td>
<td>93</td>
</tr>
<tr>
<td>or</td>
<td>96</td>
</tr>
<tr>
<td>xor</td>
<td>149</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cdf()</td>
<td>50</td>
</tr>
<tr>
<td>ceiling()</td>
<td>22</td>
</tr>
<tr>
<td>ceiling(), ceiling</td>
<td>22</td>
</tr>
<tr>
<td>centralDiff()</td>
<td>22</td>
</tr>
<tr>
<td>char()</td>
<td>23</td>
</tr>
<tr>
<td>character string, char()</td>
<td>23</td>
</tr>
<tr>
<td>characters</td>
<td></td>
</tr>
<tr>
<td>numeric code, ord()</td>
<td>97</td>
</tr>
<tr>
<td>string, char()</td>
<td>23</td>
</tr>
<tr>
<td>clear</td>
<td></td>
</tr>
<tr>
<td>error, ClErr</td>
<td>25</td>
</tr>
<tr>
<td>ClearAZ</td>
<td>25</td>
</tr>
<tr>
<td>ClErr, clear error</td>
<td>25</td>
</tr>
<tr>
<td>colAugment</td>
<td>26</td>
</tr>
<tr>
<td>colDim(), matrix column dimension</td>
<td>26</td>
</tr>
<tr>
<td>colNorm(), matrix column norm</td>
<td>26</td>
</tr>
<tr>
<td>combinations, nCr()</td>
<td>89</td>
</tr>
<tr>
<td>comment, ©</td>
<td>176</td>
</tr>
<tr>
<td>complex</td>
<td></td>
</tr>
<tr>
<td>conjugate, conj()</td>
<td>26</td>
</tr>
<tr>
<td>conj(), complex conjugate</td>
<td>26</td>
</tr>
<tr>
<td>constraint operator &quot;&quot;I&quot;&quot;</td>
<td>174</td>
</tr>
<tr>
<td>constraint operator, order of evaluation</td>
<td>181</td>
</tr>
<tr>
<td>construct matrix, constructMat()</td>
<td>26</td>
</tr>
<tr>
<td>constructMat(), construct matrix</td>
<td>26</td>
</tr>
<tr>
<td>convert</td>
<td></td>
</tr>
<tr>
<td>►Grad</td>
<td>60</td>
</tr>
<tr>
<td>►Rad</td>
<td>107</td>
</tr>
</tbody>
</table>
days between dates, dbd( ) ......................................................... 37

dbd( ), days between dates .................................................. 37
decimal
   angle display, ▶DD ................................................................. 37
   integer display, ▶Base10 ...................................................... 21
Define ..................................................................................... 38
Define LibPriv ........................................................................ 39
Define LibPub ......................................................................... 40
define, Define ......................................................................... 38
Define, define .......................................................................... 38
defining
   private function or program .............................................. 39
   public function or program ............................................... 40
definite integral
   template for ........................................................................... 10
degree notation, ° ................................................................. 171
degree/minute/second display, ▶DMS ...................................... 42
degree/minute/second notation ............................................. 172
delete
   void elements from list ..................................................... 41
deleting
   variable, DelVar .................................................................. 40
deltaList() ............................................................................. 40
DelVar, delete variable .......................................................... 40
delVoid( ), remove void elements ........................................ 41
derivatives
   first derivative, d( ) ............................................................ 166
   numeric derivative, nDeriv( ) ............................................... 90-91
   numeric derivative, nDerivative( ) ...................................... 89
det( ), matrix determinant ................................................. 41
diag( ), matrix diagonal ...................................................... 41
dim( ), dimension .................................................................. 42
dimension, dim( ) ............................................................... 42
Disp, display data ............................................................. 42
display as
   binary, ▶Base2 .................................................................... 20
cylindrical vector, ►Cylind

decimal angle, ►DD

decimal integer, ►Base10

degree/minute/second, ►DMS

hexadecimal, ►Base16

polar vector, ►Polar

rectangular vector, ►Rect

spherical vector, ►Sphere

display data, Disp

distribution functions

binomCdf() .................. 22

binomPdf() .................. 22

invNorm() .................. 65

invt() .................. 65

Invχ²() .................. 65

normCdf() .................. 92

normPdf() .................. 93

poissCdf() .................. 99

poissPdf() .................. 99

tCdf() .................. 138

tPdf() .................. 140

χ²2way() .................. 23

χ²Cdf() .................. 24

χ²GOF() .................. 24

χ²Pdf() .................. 25
divide, / .................. 158

dot

addition, .+ .................. 160
division, ./ .................. 160
multiplication, .* .................. 160
power, .^ .................. 161
product, dotP() .................. 43
subtraction, .- .................. 160
dotP(), dot product .................. 43

Index 203
E

e exponent

template for ................................................................. 6

e to a power, e^() ............................................................ 43, 48

E, exponent ......................................................................... 170

e^(), e to a power ............................................................... 43

eff(), convert nominal to effective rate .................................. 44

effective rate, eff() .............................................................. 44

eigenvalue, eigVl() ............................................................. 45

eigenvector, eigVc() ............................................................. 44

eigVc(), eigenvector ............................................................. 44

eigVl(), eigenvalue .............................................................. 45

else if, Elseif .................................................................... 45

else, Else ............................................................................ 61

Elseif, else if ...................................................................... 45

empty (void) elements ........................................................ 177

end

for, EndFor ........................................................................... 52

function, EndFunc .............................................................. 55

if, EndIf ............................................................................. 61

loop, EndLoop ................................................................... 79

program, EndPrgm ............................................................. 102

try, EndTry ......................................................................... 141

while, EndWhile .................................................................. 149

don function, EndFunc ........................................................ 55

don if, EndIf ....................................................................... 61

don loop, EndLoop ............................................................ 79

don while, EndWhile .......................................................... 149

EndTry, end try ................................................................... 141

EndWhile, end while .......................................................... 149

EOS (Equation Operating System) ........................................ 181

equal, = ............................................................................. 161

Equation Operating System (EOS) ........................................ 181

error codes and messages .................................................... 183, 191
## errors and troubleshooting

<table>
<thead>
<tr>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear error, ClrErr</td>
<td>25</td>
</tr>
<tr>
<td>pass error, PassErr</td>
<td>98</td>
</tr>
<tr>
<td>euler(), Euler function</td>
<td>46</td>
</tr>
<tr>
<td>evaluate polynomial, polyEval()</td>
<td>100</td>
</tr>
<tr>
<td>evaluation, order of</td>
<td>181</td>
</tr>
<tr>
<td>exclusion with &quot;</td>
<td>&quot; operator</td>
</tr>
<tr>
<td>exit, Exit</td>
<td>47</td>
</tr>
<tr>
<td>Exit, exit</td>
<td>47</td>
</tr>
<tr>
<td>exp(), e to a power</td>
<td>48</td>
</tr>
<tr>
<td>exponent, E</td>
<td>170</td>
</tr>
<tr>
<td>exponential regression, ExpReg</td>
<td>48</td>
</tr>
<tr>
<td>exponents</td>
<td></td>
</tr>
<tr>
<td>template for</td>
<td>5</td>
</tr>
<tr>
<td>expr(), string to expression</td>
<td>48</td>
</tr>
<tr>
<td>ExpReg, exponential regression</td>
<td>48</td>
</tr>
<tr>
<td>expressions</td>
<td></td>
</tr>
<tr>
<td>string to expression, expr()</td>
<td>48</td>
</tr>
</tbody>
</table>

## F

<table>
<thead>
<tr>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>factor(), factor</td>
<td>49</td>
</tr>
<tr>
<td>factor, factor( )</td>
<td>49</td>
</tr>
<tr>
<td>factorial, !</td>
<td>165</td>
</tr>
<tr>
<td>Fill, matrix fill</td>
<td>50</td>
</tr>
<tr>
<td>financial functions, tvmFV()</td>
<td>143</td>
</tr>
<tr>
<td>financial functions, tvmI( )</td>
<td>144</td>
</tr>
<tr>
<td>financial functions, tvmN( )</td>
<td>144</td>
</tr>
<tr>
<td>financial functions, tvmPmt( )</td>
<td>144</td>
</tr>
<tr>
<td>financial functions, tvmPV( )</td>
<td>144</td>
</tr>
<tr>
<td>first derivative</td>
<td></td>
</tr>
<tr>
<td>template for</td>
<td>9</td>
</tr>
<tr>
<td>FiveNumSummary</td>
<td>51</td>
</tr>
<tr>
<td>floor(), floor</td>
<td>51</td>
</tr>
<tr>
<td>floor, floor()</td>
<td>51</td>
</tr>
<tr>
<td>For</td>
<td>52</td>
</tr>
<tr>
<td>Index</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>for, For</td>
<td>52</td>
</tr>
<tr>
<td>For, for</td>
<td>52</td>
</tr>
<tr>
<td>format string, format( )</td>
<td>52</td>
</tr>
<tr>
<td>format(), format string</td>
<td>52</td>
</tr>
<tr>
<td>fpart(), function part</td>
<td>53</td>
</tr>
<tr>
<td>fractions</td>
<td></td>
</tr>
<tr>
<td>propFrac</td>
<td>103</td>
</tr>
<tr>
<td>template for</td>
<td>5</td>
</tr>
<tr>
<td>freqTable()</td>
<td>53</td>
</tr>
<tr>
<td>frequency()</td>
<td>54</td>
</tr>
<tr>
<td>Frobenius norm, norm()</td>
<td>92</td>
</tr>
<tr>
<td>Func, function</td>
<td>55</td>
</tr>
<tr>
<td>Func, program function</td>
<td>55</td>
</tr>
<tr>
<td>functions</td>
<td></td>
</tr>
<tr>
<td>part, fpart()</td>
<td>53</td>
</tr>
<tr>
<td>program function, Func</td>
<td>55</td>
</tr>
<tr>
<td>user-defined</td>
<td>38</td>
</tr>
<tr>
<td>functions and variables</td>
<td></td>
</tr>
<tr>
<td>copying</td>
<td>27</td>
</tr>
<tr>
<td>G</td>
<td></td>
</tr>
<tr>
<td>g, gradians</td>
<td>170</td>
</tr>
<tr>
<td>gcd(), greatest common divisor</td>
<td>56</td>
</tr>
<tr>
<td>geomCdf()</td>
<td>56</td>
</tr>
<tr>
<td>geomPdf()</td>
<td>56</td>
</tr>
<tr>
<td>get/return</td>
<td></td>
</tr>
<tr>
<td>denominator, getDenom( )</td>
<td>57</td>
</tr>
<tr>
<td>number, getNum( )</td>
<td>59</td>
</tr>
<tr>
<td>variables information, getVarInfo( )</td>
<td>57, 59</td>
</tr>
<tr>
<td>getDenom(), get/return denominator</td>
<td>57</td>
</tr>
<tr>
<td>getLangInfo(), get/return language information</td>
<td>57</td>
</tr>
<tr>
<td>getLockInfo(), tests lock status of variable or variable group</td>
<td>57</td>
</tr>
<tr>
<td>getModel(), get mode settings</td>
<td>58</td>
</tr>
<tr>
<td>getNum(), get/return number</td>
<td>59</td>
</tr>
<tr>
<td>getType(), get type of variable</td>
<td>59</td>
</tr>
</tbody>
</table>
getVarInfo(), get/return variables information ........................................ 59

go to, Goto .................................................................................. 60

Goto, go to ................................................................................ 60

gradian notation, g ....................................................................... 170

greater than or equal, ≥ ......................................................... 164

greater than, > ......................................................................... 163


greatest common divisor, gcd() ............................................. 56

groups, locking and unlocking .................................................. 76, 147

groups, testing lock status ......................................................... 57

H

hexadecimal

display, ▶Base16 ........................................................................ 21

indicator, 0h ............................................................................. 176

hyperbolic

arccosine, cosh⁻¹() .................................................................. 30

arcsine, sinh⁻¹() ...................................................................... 127

arctangent, tanh⁻¹() ................................................................ 138


cosine, cosh() ......................................................................... 29

sine, sinh() ............................................................................. 127

tangent, tanh() .................................................................... 137

I

identity matrix, identity() .............................................................. 61

identity(), identity matrix .......................................................... 61

if, If .......................................................................................... 61

If, if ........................................................................................... 61

ifFn() ........................................................................................ 62

imag(), imaginary part .................................................................. 63

imaginary part, imag() ................................................................. 63

indirection operator (#) .............................................................. 182

indirection, # .......................................................................... 170

inString(), within string ........................................................... 63

int(), integer .......................................................................... 63

Index 207
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intDiv()</td>
<td>integer divide</td>
</tr>
<tr>
<td>integer divide, intDiv()</td>
<td></td>
</tr>
<tr>
<td>integer part, iPart()</td>
<td></td>
</tr>
<tr>
<td>integer, int()</td>
<td></td>
</tr>
<tr>
<td>integral, (\int)</td>
<td></td>
</tr>
<tr>
<td>interpolate(), interpolate</td>
<td></td>
</tr>
<tr>
<td>inverse cumulative normal distribution (invNorm( ))</td>
<td></td>
</tr>
<tr>
<td>inverse, (^{-1})</td>
<td></td>
</tr>
<tr>
<td>invF()</td>
<td></td>
</tr>
<tr>
<td>invNorm(), inverse cumulative normal distribution)</td>
<td></td>
</tr>
<tr>
<td>invt()</td>
<td></td>
</tr>
<tr>
<td>Invx²()</td>
<td></td>
</tr>
<tr>
<td>iPart(), integer part</td>
<td></td>
</tr>
<tr>
<td>irr(), internal rate of return</td>
<td></td>
</tr>
<tr>
<td>internal rate of return, irr()</td>
<td></td>
</tr>
<tr>
<td>isPrime(), prime test</td>
<td></td>
</tr>
<tr>
<td>isVoid(), test for void</td>
<td></td>
</tr>
<tr>
<td>label, Lbl</td>
<td></td>
</tr>
<tr>
<td>language</td>
<td>get language information</td>
</tr>
<tr>
<td>Lbl, label</td>
<td></td>
</tr>
<tr>
<td>lcm, least common multiple</td>
<td></td>
</tr>
<tr>
<td>least common multiple, lcm</td>
<td></td>
</tr>
<tr>
<td>left(), left</td>
<td></td>
</tr>
<tr>
<td>left, left( )</td>
<td></td>
</tr>
<tr>
<td>length of string</td>
<td></td>
</tr>
<tr>
<td>less than or equal, (\leq)</td>
<td></td>
</tr>
<tr>
<td>LibPriv</td>
<td></td>
</tr>
<tr>
<td>LibPub</td>
<td></td>
</tr>
<tr>
<td>library</td>
<td>create shortcuts to objects</td>
</tr>
<tr>
<td>libShortcut( ), create shortcuts to library objects</td>
<td></td>
</tr>
<tr>
<td>linear regression, LinRegAx</td>
<td></td>
</tr>
</tbody>
</table>
linear regression, LinRegBx  ................................................. 68, 70
LinRegBx, linear regression ..................................................... 68
LinRegMx, linear regression .................................................... 69
LinRegIntervals, linear regression ............................................ 70
LinRegTest ................................................................. .......................... 71
linSolve() ........................................................................ 73
list to matrix, list\*mat( ) ....................................................... 74
list, conditionally count items in ........................................... 32
list, count items in ................................................................ 32
list\*mat( ), list to matrix ...................................................... 74
lists
    augment/concatenate, augment( ) ........................................ 18
cross product, crossP( ) ........................................................ 33
cumulative sum, cumulativeSum( ) .......................................... 36
differences in a list, Δlist( ) .................................................... 73
dot product, dotP( ) .............................................................. 43
empty elements in .................................................................. 177
list to matrix, list\*mat( ) ....................................................... 74
matrix to list, mat\*list( ) ...................................................... 80
maximum, max( ) ................................................................. 80
mid-string, mid( ) ................................................................. 83
minimum, min( ) ................................................................... 84
new, newList( ) ..................................................................... 90
product, product( ) ............................................................... 103
sort ascending, SortA ............................................................ 129
sort descending, SortD .......................................................... 129
summation, sum( ) ............................................................... 134-135
ln( ), natural logarithm ............................................................. 74
LnReg, logarithmic regression ................................................ 75
local variable, Local .............................................................. 76
local, Local ........................................................................... 76
Local, local variable .............................................................. 76
Lock, lock variable or variable group ....................................... 76
locking variables and variable groups ..................................... 76
Log
    template for ........................................................................ 6
Index 209
| logarithmic regression, LnReg | 75 |
| logarithms | 74 |
| logical double implication, ⇐ | 165 |
| logical implication, ⇒ | 164, 179 |
| logistic regression, Logistic | 77 |
| logistic regression, LogisticD | 78 |
| Logistic, logistic regression | 77 |
| LogisticD, logistic regression | 78 |
| loop, Loop | 79 |
| Loop, loop | 79 |
| LU, matrix lower-upper decomposition | 80 |

**M**

<p>| mat|list|, matrix to list | 80 |
|matrices| |
| augment/concatenate, augment( ) | 18 |
| column dimension, colDim() | 26 |
| column norm, colNorm() | 26 |
| cumulative sum, cumulativeSum() | 36 |
| determinant, det() | 41 |
| diagonal, diag() | 41 |
| dimension, dim() | 42 |
| dot addition, .+ | 160 |
| dot division, ./ | 160 |
| dot multiplication, .* | 160 |
| dot power, .^ | 161 |
| dot subtraction, .- | 160 |
| eigenvalue, eigVl() | 45 |
| eigenvector, eigVc() | 44 |
| filling, Fill | 50 |
| identity, identity() | 61 |
| list to matrix, list|mat|() | 74 |
| lower-upper decomposition, LU | 80 |
| matrix to list, mat|list|() | 80 |
| maximum, max() | 80 |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>minimum, min()</td>
<td>84</td>
</tr>
<tr>
<td>new, newMat()</td>
<td>90</td>
</tr>
<tr>
<td>product, product()</td>
<td>103</td>
</tr>
<tr>
<td>QR factorization, QR</td>
<td>104</td>
</tr>
<tr>
<td>random, randMat()</td>
<td>108</td>
</tr>
<tr>
<td>reduced row echelon form, ref()</td>
<td>118</td>
</tr>
<tr>
<td>row addition, rowAdd()</td>
<td>117</td>
</tr>
<tr>
<td>row dimension, rowDim()</td>
<td>117</td>
</tr>
<tr>
<td>row echelon form, ref()</td>
<td>110</td>
</tr>
<tr>
<td>row multiplication and addition, mRowAdd()</td>
<td>85</td>
</tr>
<tr>
<td>row norm, rowNorm()</td>
<td>118</td>
</tr>
<tr>
<td>row operation, mRow()</td>
<td>85</td>
</tr>
<tr>
<td>row swap, rowSwap()</td>
<td>118</td>
</tr>
<tr>
<td>submatrix, subMat()</td>
<td>134-135</td>
</tr>
<tr>
<td>summation, sum()</td>
<td>134-135</td>
</tr>
<tr>
<td>transpose, T</td>
<td>136</td>
</tr>
</tbody>
</table>

| Matrix Templates |
|------------------|------|
| matrix (1×2) | 8 |
| template for | 8 |
| matrix (2×1) | 8 |
| template for | 8 |
| matrix (2×2) | 8 |
| template for | 8 |
| matrix (m×n) | 8 |
| template for | 8 |
| matrix to list, mat►list() | 80 |

| Statistical Functions |
|-----------------------|------|
| max(), maximum | 80 |
| maximum, max() | 80 |
| mean(), mean | 81 |
| mean, mean() | 81 |
| median(), median | 82 |
| median, median() | 82 |
| medium-medium line regression, MedMed | 82 |

MedMed, medium-medium line regression | 82 |
mid-string, mid() | 83 |
mid(), mid-string | 83 |
min(), minimum | 84 |
minimum, min() .......................................................... 84
minute notation, ‘ .................................................. 172
mirr(), modified internal rate of return ......................... 84
mixed fractions, using propFrac() with .................... 103
mod(), modulo ....................................................... 85
mode settings, getMode() ......................................... 58
modes
  setting, setMode() .............................................. 122
modified internal rate of return, mirr() ..................... 84
modulo, mod() ..................................................... 85
mRow(), matrix row operation ................................ 85
mRowAdd(), matrix row multiplication and addition ...... 85
Multiple linear regression t test .............................. 87
multiply, * .......................................................... 157
MultReg ............................................................ 85
MultRegIntervals() ............................................. 86
MultRegTests() ................................................... 87
nand, Boolean operator .......................................... 88
natural logarithm, ln() ........................................... 74
nCr(), combinations .............................................. 89
nDerivative(), numeric derivative ............................ 89
negation, entering negative numbers ....................... 182
net present value, npv() ....................................... 94
new
  list, newList() .................................................. 90
  matrix, newMat() ............................................. 90
newList(), new list ............................................. 90
newMat(), new matrix .......................................... 90
nfMax(), numeric function maximum ....................... 90
nfMin(), numeric function minimum ....................... 91
nInt(), numeric integral ..................................... 91
nom(), convert effective to nominal rate .................. 91
nominal rate, nom() ........................................... 91
nor, Boolean operator .................................................. 92
norm(), Frobenius norm .................................................. 92
normal distribution probability, normCdf() .......................... 92
normCdf() ................................................................. 92
normPdf() ................................................................. 93
not equal, ≠ .................................................................. 162
not, Boolean operator ..................................................... 93
nPr(), permutations ......................................................... 93
npv(), net present value ................................................... 94
nSolve(), numeric solution ................................................. 95
nth root
   template for ................................................................ 6
numeric
derivative, nDeriv() .......................................................... 90-91
derivative, nDerivative() ...................................................... 89
integral, nInt() ............................................................... 91
solution, nSolve() ............................................................ 95

O
objects
   create shortcuts to library ............................................. 68
one-variable statistics, OneVar .......................................... 95
OneVar, one-variable statistics .................................... 95
operators
   order of evaluation .......................................................... 181
or (Boolean), or ............................................................... 96
or, Boolean operator ........................................................ 96
ord(), numeric character code ........................................ 97
+, add ........................................................................... 156

P
P►Rx(), rectangular x coordinate ....................................... 98
P►Ry(), rectangular y coordinate ....................................... 98
pass error, PassErr .......................................................... 98
Index

PassErr, pass error .......................................................... 98
Pdf() ................................................................. 53
percent, % ............................................................... 161
permutations, nPr() ................................................... 93
piecewise function (2-piece)
  template for ........................................................ 6
piecewise function (N-piece)
  template for ........................................................ 6
piecewise() .............................................................. 99
poissCdf() .............................................................. 99
poissPdf() .............................................................. 99
polar
  coordinate, R►Pr() ................................................ 107
  coordinate, R►Pθ() ................................................ 106
  vector display, ►Polar ........................................... 99
polyEval(), evaluate polynomial ................................... 100
polynomials
  evaluate, polyEval() ............................................ 100
  random, randPoly() ............................................. 109
PolyRoots() ............................................................ 101
power of ten, 10^() .................................................. 173
power regression, PowerReg ....................................... 101, 112-113, 138
power, ^ ............................................................... 158
PowerReg, power regression ......................................... 101
Prgm, define program ................................................. 102
prime number test, isPrime() ....................................... 66
probability density, normPdf() .................................... 93
prodSeq() ............................................................... 102
product(), product .................................................. 103
product, ∏() ........................................................... 167
  template for ...................................................... 9
product, product() .................................................. 103
programming
  define program, Prgm ......................................... 102
display data, Disp .................................................... 42
pass error, PassErr .................................................. 98
<table>
<thead>
<tr>
<th>programs</th>
<th>defining private library</th>
<th>39</th>
</tr>
</thead>
<tbody>
<tr>
<td>programs</td>
<td>defining public library</td>
<td>40</td>
</tr>
<tr>
<td>programs and programming</td>
<td>clear error, ClrErr</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>display I/O screen, Disp</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>end program, EndPrgm</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td>end try, EndTry</td>
<td>141</td>
</tr>
<tr>
<td></td>
<td>try, Try</td>
<td>141</td>
</tr>
<tr>
<td></td>
<td>proper fraction, propFrac</td>
<td>103</td>
</tr>
<tr>
<td></td>
<td>propFrac, proper fraction</td>
<td>103</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>QR factorization, QR</td>
</tr>
<tr>
<td>QR, QR factorization</td>
</tr>
<tr>
<td>quadratic regression, QuadReg</td>
</tr>
<tr>
<td>QuadReg, quadratic regression</td>
</tr>
<tr>
<td>quartic regression, QuartReg</td>
</tr>
<tr>
<td>QuartReg, quartic regression</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>R, radian</td>
</tr>
<tr>
<td>R►Pr(), polar coordinate</td>
</tr>
<tr>
<td>R►Pθ(), polar coordinate</td>
</tr>
<tr>
<td>radian, R</td>
</tr>
<tr>
<td>rand(), random number</td>
</tr>
<tr>
<td>randBin, random number</td>
</tr>
<tr>
<td>randInt(), random integer</td>
</tr>
<tr>
<td>randMat(), random matrix</td>
</tr>
<tr>
<td>randNorm(), random norm</td>
</tr>
<tr>
<td>random</td>
</tr>
<tr>
<td>matrix, randMat()</td>
</tr>
<tr>
<td>norm, randNorm()</td>
</tr>
<tr>
<td>number seed, RandSeed</td>
</tr>
<tr>
<td>Term</td>
</tr>
<tr>
<td>------------------------------------------</td>
</tr>
<tr>
<td>polynomial, randPoly()</td>
</tr>
<tr>
<td>random sample</td>
</tr>
<tr>
<td>randPoly(), random polynomial</td>
</tr>
<tr>
<td>randSamp()</td>
</tr>
<tr>
<td>RandSeed, random number seed</td>
</tr>
<tr>
<td>real(), real</td>
</tr>
<tr>
<td>real, real()</td>
</tr>
<tr>
<td>reciprocal, (^{-1})</td>
</tr>
<tr>
<td>rectangular-vector display, Rect</td>
</tr>
<tr>
<td>rectangular x coordinate, P(\times)Rx()</td>
</tr>
<tr>
<td>rectangular y coordinate, P(\times)Ry()</td>
</tr>
<tr>
<td>reduced row echelon form, ref()</td>
</tr>
<tr>
<td>ref(), row echelon form</td>
</tr>
<tr>
<td>regressions</td>
</tr>
<tr>
<td>cubic, CubicReg</td>
</tr>
<tr>
<td>exponential, ExpReg</td>
</tr>
<tr>
<td>linear regression, LinRegAx</td>
</tr>
<tr>
<td>linear regression, LinRegBx</td>
</tr>
<tr>
<td>logarithmic, LnReg</td>
</tr>
<tr>
<td>Logistic</td>
</tr>
<tr>
<td>logistic, Logistic</td>
</tr>
<tr>
<td>medium-medium line, MedMed</td>
</tr>
<tr>
<td>MultReg</td>
</tr>
<tr>
<td>power regression, PowerReg</td>
</tr>
<tr>
<td>quadratic, QuadReg</td>
</tr>
<tr>
<td>quartic, QuartReg</td>
</tr>
<tr>
<td>sinusoidal, SinReg</td>
</tr>
<tr>
<td>remain(), remainder</td>
</tr>
<tr>
<td>remainder, remain()</td>
</tr>
<tr>
<td>remove</td>
</tr>
<tr>
<td>void elements from list</td>
</tr>
<tr>
<td>Request</td>
</tr>
<tr>
<td>RequestStr</td>
</tr>
<tr>
<td>result values, statistics</td>
</tr>
<tr>
<td>results, statistics</td>
</tr>
<tr>
<td>return, Return</td>
</tr>
</tbody>
</table>
Index 217

Return, return .................................. 114
right(), right .................................. 114
right, right( ) .................................. 114
rk23(), Runge Kutta function ............ 114
rotate(), rotate .......................... 116
rotate, rotate( ) .................. 116
round(), round ........................ 117
round, round( ) .................. 117
row echelon form, ref( ) ............. 110
rowAdd(), matrix row addition ........ 117
rowDim(), matrix row dimension .... 117
rowNorm(), matrix row norm .......... 118
rowSwap(), matrix row swap .......... 118
ref(), reduced row echelon form ...... 118

S

sec "() , inverse secant .................. 119
sec(), secant .......................... 119
sech "() , inverse hyperbolic secant .... 120
sech(), hyperbolic secant ............. 119
second derivative
    template for .................................. 9
second notation, " ............................ 172
seq(), sequence .......................... 120
seqGen() .................................. 120
seqn() .................................. 121
sequence, seq() .......................... 120-121
set
    mode, setMode() .................. 122
setMode(), set mode .................. 122
settings, get current ................. 58
shift(), shift ........................ 123
shift, shift() .................. 123
sign(), sign .......................... 124
sign, sign() .................. 124
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>simult(), simultaneous equations</td>
<td>125</td>
</tr>
<tr>
<td>simultaneous equations, simul()</td>
<td>125</td>
</tr>
<tr>
<td>sin^(-1)( ), arcsine</td>
<td>126</td>
</tr>
<tr>
<td>sin( ), sine</td>
<td>125</td>
</tr>
<tr>
<td>sine, sin( )</td>
<td>125</td>
</tr>
<tr>
<td>sinh^(-1)( ), hyperbolic arcsine</td>
<td>127</td>
</tr>
<tr>
<td>sinh( ), hyperbolic sine</td>
<td>127</td>
</tr>
<tr>
<td>SinReg, sinusoidal regression</td>
<td>128</td>
</tr>
<tr>
<td>sinusoidal regression, SinReg</td>
<td>128</td>
</tr>
<tr>
<td>SortA, sort ascending</td>
<td>129</td>
</tr>
<tr>
<td>SortD, sort descending</td>
<td>129</td>
</tr>
<tr>
<td>sorting</td>
<td>129</td>
</tr>
<tr>
<td>ascending, SortA</td>
<td>129</td>
</tr>
<tr>
<td>descending, SortD</td>
<td>129</td>
</tr>
<tr>
<td>spherical vector display, ▶Sphere</td>
<td>130</td>
</tr>
<tr>
<td>sqrt( ), square root</td>
<td>130</td>
</tr>
<tr>
<td>square root</td>
<td>5</td>
</tr>
<tr>
<td>template for</td>
<td>5</td>
</tr>
<tr>
<td>square root, (\sqrt())</td>
<td>130, 167</td>
</tr>
<tr>
<td>standard deviation, stdDev( )</td>
<td>132-133, 147</td>
</tr>
<tr>
<td>stat.results</td>
<td>131</td>
</tr>
<tr>
<td>stat.values</td>
<td>132</td>
</tr>
<tr>
<td>statistics</td>
<td></td>
</tr>
<tr>
<td>combinations, nCr( )</td>
<td>89</td>
</tr>
<tr>
<td>factorial, !</td>
<td>165</td>
</tr>
<tr>
<td>mean, mean( )</td>
<td>81</td>
</tr>
<tr>
<td>median, median( )</td>
<td>82</td>
</tr>
<tr>
<td>one-variable statistics, OneVar</td>
<td>95</td>
</tr>
<tr>
<td>permutations, nPr( )</td>
<td>93</td>
</tr>
<tr>
<td>random norm, randNorm( )</td>
<td>108</td>
</tr>
<tr>
<td>random number seed, RandSeed</td>
<td>109</td>
</tr>
<tr>
<td>standard deviation, stdDev( )</td>
<td>132-133, 147</td>
</tr>
<tr>
<td>two-variable results, TwoVar</td>
<td>145</td>
</tr>
<tr>
<td>variance, variance( )</td>
<td>147</td>
</tr>
<tr>
<td>stdDevPop( ), population standard deviation</td>
<td>132</td>
</tr>
<tr>
<td>stdDevSamp( ), sample standard deviation</td>
<td>133</td>
</tr>
</tbody>
</table>
Stop command ................................................................. 133
store variable (→) .......................................................... 175
storing symbol, & .......................................................... 175
string dimension, dim( ) .................................................. 42
length ........................................................................ 42
string( ), expression to string ........................................... 134
strings append, & .......................................................... 165
character code, ord( ) ..................................................... 97
character string, char( ) ................................................. 23
expression to string, string( ) ........................................ 134
format, format( ) .......................................................... 52
formatting .................................................................... 52
indirection, # ............................................................. 170
left, left( ) ................................................................. 67
mid-string, mid( ) .......................................................... 83
right, right( ) ............................................................. 46, 64, 114, 148
rotate, rotate( ) .......................................................... 116
shift, shift( ) ............................................................. 123
string to expression, expr( ) .......................................... 48
using to create variable names ........................................ 182
within, InString .......................................................... 63
student-t distribution probability, tCdf( ) ......................... 138
student-t probability density, tPdf( ) ................................. 140
subMat(), submatrix ..................................................... 134-135
submatrix, subMat( ) ..................................................... 134-135
substitution with "|" operator .......................................... 174
subtract, - ................................................................... 156
sum of interest payments ............................................... 168
sum of principal payments ............................................. 169
sum( ), summation ....................................................... 134
sum, ∑( ) ................................................................. 168
template for ............................................................. 9
sumIf( ) .................................................................... 135
summation, sum() .......................................................... 134
sumSeq() ........................................................................ 135

system of equations (2-equation)
    template for ................................................................. 7

system of equations (N-equation)
    template for ................................................................. 7

\( T \)

t test, tTest ..................................................................... 142

T, transpose .................................................................... 136
tan\(^{-1}(\cdot)\), arctangent ..................................................... 137
tan(\cdot), tangent ............................................................... 136
tangent, tan(\cdot) .............................................................. 136
tanh\(^{-1}(\cdot)\), hyperbolic arctangent .................................... 138
tanh(\cdot), hyperbolic tangent ................................................ 137
tCdf(\cdot), studentt distribution probability ............................. 138

templates
    absolute value ............................................................. 7-8
    definite integral .......................................................... 10
    e exponent .................................................................... 6
    exponent ...................................................................... 5
    first derivative ............................................................ 9
    fraction ....................................................................... 5
    Log ............................................................................. 6
    matrix (1 \(\times\) 2) ........................................................... 8
    matrix (2 \(\times\) 1) ........................................................... 8
    matrix (2 \(\times\) 2) ........................................................... 8
    matrix (m \(\times\) n) .......................................................... 8
    nth root ....................................................................... 6
    piecewise function (2-piece) ............................................. 6
    piecewise function (N-piece) ............................................ 6
    product, \( \prod(\cdot) \) ....................................................... 9
    second derivative .......................................................... 9
    square root ................................................................... 5
    sum, \( \Sigma(\cdot) \) ........................................................... 9

\[220\] Index
system of equations (2-equation) .................................................. 7
system of equations (N-equation) .................................................. 7
test for void, isVoid( ) ................................................................. 66
Test_2S, 2-sample F test ............................................................... 54
Text command ............................................................................. 138
time value of money, Future Value ................................................ 143
time value of money, Interest ......................................................... 144
time value of money, number of payments .................................... 144
time value of money, payment amount .......................................... 144
time value of money, present value ................................................ 144
tInterval, t confidence interval ..................................................... 139
tInterval_2Samp, twosample t confidence interval ......................... 140
tPdf( ), student probability density .............................................. 140
trace( ) .......................................................................................... 141
transpose, T .................................................................................... 136
Try, error handling command ....................................................... 141
tTest, t test ..................................................................................... 142
tTest_2Samp, two-sample t test ...................................................... 143
TVM arguments ............................................................................ 144
tvmFV( ) ......................................................................................... 143
tvmI( ) ............................................................................................ 144
tvmN( ) ........................................................................................... 144
tvmPmt( ) ......................................................................................... 144
tvmPV( ) .......................................................................................... 144
two-variable results, TwoVar ......................................................... 145
TwoVar, two-variable results ......................................................... 145

U

unit vector, unitV( ) ....................................................................... 146
unitV( ), unit vector ....................................................................... 146
unLock, unlock variable or variable group ..................................... 147
unlocking variables and variable groups ...................................... 147
user-defined functions .................................................................. 38
user-defined functions and programs ........................................... 39-40
variable
    creating name from a character string ...................................... 182
variable and functions
    copying ....................................................................................... 27
variables
    clear all single-letter .................................................................. 25
delete, DelVar .................................................................................. 40
local, Local ......................................................................................... 76
variables, locking and unlocking ....................................................... 57, 76, 147
variance, variance( ) ......................................................................... 147
varPop( ) ............................................................................................ 147
varSamp( ), sample variance ............................................................. 147
vectors
    cross product, crossP( ) ................................................................... 33
cylindrical vector display, ▶Cylind ................................................... 37
dot product, dotP( ) ............................................................................ 43
unit, unitV( ) ...................................................................................... 146
void elements ..................................................................................... 177
void elements, remove ........................................................................ 41
void, test for ...................................................................................... 66

W

warnCodes( ), Warning codes ............................................................. 148
warning codes and messages .............................................................. 191
when( ), when .................................................................................... 148
when, when( ) .................................................................................... 148
while, While ......................................................................................... 149
While, while ......................................................................................... 149
with, | ............................................................................................... 174
within string, inString( ) ................................................................. 63

222  Index
<table>
<thead>
<tr>
<th>Alphabet</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>$x^2$, square</td>
<td>159</td>
</tr>
<tr>
<td></td>
<td>XNOR</td>
<td>165</td>
</tr>
<tr>
<td></td>
<td>xor, Boolean exclusive or</td>
<td>149</td>
</tr>
<tr>
<td>Z</td>
<td>$z\text{Interval}$, $z$ confidence interval</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>$z\text{Interval}_1\text{Prop}$, one-proportion $z$ confidence interval</td>
<td>151</td>
</tr>
<tr>
<td></td>
<td>$z\text{Interval}_2\text{Prop}$, two-proportion $z$ confidence interval</td>
<td>151</td>
</tr>
<tr>
<td></td>
<td>$z\text{Interval}_2\text{Samp}$, two-sample $z$ confidence interval</td>
<td>152</td>
</tr>
<tr>
<td></td>
<td>$z\text{Test}$</td>
<td>152</td>
</tr>
<tr>
<td></td>
<td>$z\text{Test}_1\text{Prop}$, one-proportion $z$ test</td>
<td>153</td>
</tr>
<tr>
<td></td>
<td>$z\text{Test}_2\text{Prop}$, two-proportion $z$ test</td>
<td>153</td>
</tr>
<tr>
<td></td>
<td>$z\text{Test}_2\text{Samp}$, two-sample $z$ test</td>
<td>154</td>
</tr>
<tr>
<td>Δ</td>
<td>$\Delta\text{list}$, list difference</td>
<td>73</td>
</tr>
<tr>
<td>X</td>
<td>$\chi^2\text{2way}$</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>$\chi^2\text{Cdf}$</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>$\chi^2\text{GOF}$</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>$\chi^2\text{Pdf}$</td>
<td>25</td>
</tr>
</tbody>
</table>