

MatrixOp

Both *alpha* and *beta* must be real and default to 1.

Use /B=0 to omit the *beta*matC* term.

Example

```
Function Demo()      // Demonstrates various ways to call MatrixMultiplyAdd
// Create input waves
Make/O/N=(3,3) matA = p + 10*q
Make/O/N=(3,3) matB = p == q

Print "=== matC = matA x matB ==="
Make/O/N=(3,3) matC = 0
MatrixMultiplyAdd /B=0 matA, matB, matC
Print matC

Print "=== matC = 2*matA x matB ==="
Make/O/N=(3,3) matC = 0
MatrixMultiplyAdd /A=2 /B=0 matA, matB, matC
Print matC

Print "=== Free Wave Using WaveClear = matA x matB ==="
Make/O/N=(3,3) matC = 0           // matC wave ref is cleared by WaveClear
WaveClear matC                   // MatrixMultiplyAdd creates a free output wave
MatrixMultiplyAdd /B=0 matA, matB, matC
Print matC

Print "=== Free Wave Using /ZC = matA x matB ==="
Make/O/N=(3,3) matC = 0           // matC wave ref is cleared by /ZC
MatrixMultiplyAdd /B=0 /ZC matA, matB, matC
Print matC

Print "=== Create dest wave using string variable ==="
String dest = "matDest1"
Make/O/N=(3,3) $dest = 0
WAVE destWave = $dest
MatrixMultiplyAdd /B=0 matA, matB, destWave
Print destWave
End
```

See Also

MatrixOp and **MatrixMultiply** for more efficient matrix operations.

Matrix Math Operations on page III-138 for more about Igor's matrix routines.

FastOp for additional efficient non-matrix operations.

MatrixOp

MatrixOp [/C /FREE /NTHR=*n* /O /S] *destwave* = *expression*

The MatrixOp operation evaluates *expression* and stores the result in *destWave*.

expression may include literal numbers, numeric variables, numeric waves, and the set of operators and functions described below. MatrixOp does not support text waves, strings or structures.

MatrixOp is faster and in some case more readable than standard Igor waveform assignments and matrix operations.

See **Using MatrixOp** on page III-140 for an introduction to MatrixOp.

Parameters

destWave

Specifies a destination wave for the assignment expression. *destWave* is created at runtime. If it already exists, you must use the /O flag to overwrite it or the operation returns an error.

When the operation is completed, *destWave* has the dimensions and data type implied by *expression*. In particular, it may be complex if *expression* evaluates to a complex quantity. If *expression* evaluates to a scalar, *destWave* is a 1x1 wave.

If you include the /FREE flag then *destWave* is created as a free wave.

By default the data type of *destWave* depends on the data types of the operands and the nature of the operations on the right-hand side of the assignment. If *expression* references integer waves only, *destWave* may be an integer wave too but most operations with a scalars convert *destWave* into a double precision wave. See **MatrixOp Data Promotion Policy** on page III-145 for further discussion.

Even if *destWave* exists before the operation, MatrixOp may change its data type and dimensionality as implied by *expression*.

You can force the number type using MatrixOp functions such as `uint16` and `fp32`.

expression

expression is a mathematical expression referencing waves, local variables, global variables and literal numbers together with MatrixOp functions and MatrixOp operators as listed in the following sections.

You can use any combination of data types for operands. In particular, you can mix real and complex types in *expression*. MatrixOp determines data types of inputs and the appropriate output data type at runtime without regard to any type declaration such as `Wave/C`.

Operators

+	Addition between scalars, matrix addition or addition of a scalar (real or complex) to each element of a matrix.
-	Subtraction of one scalar from another, matrix subtraction, or subtracting a scalar from each element of a matrix. Subtraction of a matrix from a scalar is not defined.
*	Multiplication between two scalars, multiplication of a matrix by a scalar, or element-by-element multiplication of two waves of the same dimensions.
/	Division of two scalars, division of a matrix by a scalar, or element-by-element division between two waves of the same dimensions. Division of a scalar by a matrix is not supported but you can use the <code>rec</code> function with multiplication instead.
x	Matrix multiplication (lower case x symbol only). This operator <i>must</i> be preceded and followed by a space. Matrix multiplication requires that the number of columns in the matrix on the left side be equal to the number of rows in the matrix on the right.
.	Generalized form of a dot product. In an expression $a.b$ it is expected that a and b have the same number of points although they may be of arbitrary numeric type. The operator returns the sum of the products of the sequential elements as if both a and b were 1D arrays. For complex binary operand waves a and b , this operator returns the MatrixOp equivalent of $\text{sum}(a*\text{conj}(b))$. The MatrixDot function returns $\text{sum}(b*\text{conj}(a))$.
^t	Matrix transpose. This is a postfix operator meaning that ^t appears after the name of a matrix wave.
^h	Hermitian transpose. This is a postfix operator meaning that ^h appears after the name of a matrix wave.
&&	Logical AND operator supports all real data types and results in a signed byte numeric token with the value of either 0 or 1. The operation acts on an element by element basis and is performed for each element of the operand waves.
	Logical OR operator supports all real data types and results in a signed byte numeric token with the value of either 0 or 1. The operation acts on an element by element basis and is performed for each element of the operand waves.

MatrixOp does not support operator combinations such as +=.

This table shows the precedence of MatrixOp operators:

MatrixOp Operator	Precedence
^h ^t	Highest
x .	
* /	
+ -	
&&	Lowest

You can use parentheses to force evaluation order.

Operators that have the same precedence associate from right to left. This means that $a * b / c$ is equivalent to $a * (b / c)$.

Functions

These functions are available for use with MatrixOp.

<code>abs (w)</code>	Absolute value of a real number or the magnitude of a complex number.
----------------------	---

<code>acos(w)</code>	Arc cosine of w .
<code>acosh(w)</code>	Inverse hyperbolic cosine of w . Added in Igor Pro 7.00.
<code>addCols(w, dc)</code>	Returns a matrix where elements of the 1D wave dc are added to the corresponding column in w : $\text{out} = w + dc[q]$ Added in Igor Pro 9.00.
<code>addRows(w, dr)</code>	Returns a matrix where elements of the 1D wave dr are added to the corresponding row in w : $\text{out} = w + dr[p]$ Added in Igor Pro 9.00.
<code>asin(w)</code>	Arc sine of w .
<code>asinh(w)</code>	Inverse hyperbolic sine of w . Added in Igor Pro 7.00.
<code>asyncCorrelation(w)</code>	Asynchronous spectrum correlation matrix for a real valued input matrix wave w . See <code>syncCorrelation</code> for details.
<code>atan(w)</code>	Arc tangent (inverse tangent) of w .
<code>atan2(y, x)</code>	Arc tangent (inverse tangent) of real y/x .
<code>atanh(w)</code>	Inverse hyperbolic tangent of w . Added in Igor Pro 7.00.
<code>averageCols(w)</code>	Returns a (1xcolumns) wave containing the averages of the columns of matrix w . This is equivalent to <code>sumCols(w)/numRows(w)</code> . Added in Igor Pro 7.00.
<code>axisToQuat(ax)</code>	See Functions Using Quaternions on page V-573.
<code>backwardSub(U, c)</code>	Returns a column vector solution for the matrix equation $Ux=c$, where U is an (NxN) wave representing an upper triangular matrix and c is a column vector of N rows. If c has additional columns they are ignored. Ideally, U and c should be either SP or DP waves (real or complex). Other numeric data types are supported with a slight performance penalty. This function is typically used in solving linear equations following a matrix decomposition into lower and upper triangular matrices (e.g., Cholesky), with an expression of the form: $\text{MatrixOp/O solVector} = \text{backwardSub}(U, \text{forwardSub}(L, b))$ where U and L are the upper and lower triangular factors. Added in Igor Pro 7.00.
<code>beam(w, row, col)</code>	When w is a 3D wave the beam function returns a 1D array corresponding to the data in the beam defined by $w[\text{row}][\text{col}][]$. In other words, it returns a 1D array consisting of all elements in the specified row and column from all layers. See also ImageTransform <code>getBeam</code> . When w is a 4D wave it returns a 2D array containing $w[\text{row}][\text{col}][][]$. In other words, it returns a matrix consisting of all elements in the specified row and column from all layers and all chunks. The beam function belongs to a special class in that it does not operate on a layer by layer basis. It therefore does not permit compound expressions in place of any of its parameters. The beam function has the highest precedence.

<code>bitAnd(w1, w2)</code>	<p>Returns an array of the same dimensions and number type as <i>w1</i> where each element corresponds to the bitwise AND operation between <i>w1</i> and <i>w2</i>.</p> <p><i>w2</i> can be either a single number or a matrix of the same dimensions as <i>w1</i>. Both <i>w1</i> and <i>w2</i> must be real integer numeric types.</p> <p>Added in Igor Pro 7.00.</p>
<code>bitOr(w1, w2)</code>	<p>Returns an array of the same dimensions and number type as <i>w1</i> where each element corresponds to the bitwise OR operation between <i>w1</i> and <i>w2</i>.</p> <p><i>w2</i> can be either a single number or a matrix of the same dimensions as <i>w1</i>. Both <i>w1</i> and <i>w2</i> must be real integer numeric types.</p> <p>Added in Igor Pro 7.00.</p>
<code>bitReverseCol(w, c, order)</code>	<p>Returns the matrix <i>w</i> with the entries of column <i>c</i> reordered.</p> <p><code>bitReverseCol</code> was added in Igor Pro 9.00.</p> <p><i>order</i> is one of the following values:</p> <ul style="list-style-type: none"> 0: Direct binary reversal of the index 1: Hadamard order 2: Dyadic/Paley/Gray order 3: Unchanged order <p><code>bitReverseCol</code> can be used for changing the order of entries in fast transform algorithms such as FFT and the fast Walsh-Hadamard transform. The number of rows in <i>w</i> must be a power of 2.</p> <p>See <i>Examples</i> below for an example using <code>bitReverseCol</code>.</p>
<code>bitShift(w1, w2)</code>	<p>Returns an array of the same dimensions and number type as <i>w1</i> shifted by the amount specified in <i>w2</i>. When <i>w2</i> is positive the shifting is to the left. When it is negative the shifting is to the right.</p> <p><i>w2</i> can be either a single number or a matrix of the same dimensions as <i>w1</i>. Both <i>w1</i> and <i>w2</i> must be real integer numeric types.</p> <p>Added in Igor Pro 7.00.</p>
<code>bitXor(w1, w2)</code>	<p>Returns an array of the same dimensions and number type as <i>w1</i> where each element corresponds to the bitwise XOR operation between <i>w1</i> and <i>w2</i>.</p> <p><i>w2</i> can be either a single number or a matrix of the same dimensions as <i>w1</i>. Both <i>w1</i> and <i>w2</i> must be real integer numeric types.</p> <p>Added in Igor Pro 7.00.</p>
<code>bitNot(w)</code>	<p>Returns an array of the same dimensions and number type as <i>w</i> where each element is the bitwise complement of the corresponding element in <i>w</i>.</p> <p>Added in Igor Pro 7.00.</p>
<code>catCols(w1, w2)</code>	<p>Concatenates the columns of <i>w2</i> to those of <i>w1</i>. <i>w1</i> and <i>w2</i> must have the same number of rows and the same number type.</p> <p>Added in Igor Pro 7.00.</p>
<code>catRows(w1, w2)</code>	<p>Concatenates the rows of <i>w2</i> to those of <i>w1</i>. <i>w1</i> and <i>w2</i> must have the same number of columns and the same number type.</p> <p>Added in Igor Pro 7.00.</p>
<code>cbrt(w)</code>	<p>Returns the cube root of the elements of <i>w</i>. When <i>w</i> is complex <code>cbrt</code> returns the principal cube root (the root with a positive imaginary number).</p> <p>Added in Igor Pro 8.00.</p>

<code>ceil(z)</code>	Smallest integer larger than z .
<code>chirpZ(data, A, W, M)</code>	<p>Chirp Z Transform of the 1D wave data calculated for the contour defined by</p> $z_k = AW^{-k},$ $k = 0, 1, \dots, M - 1.$ <p>Here both A and W are complex and the standard z transform for a sequence $\{x(n)\}$ is defined by</p> $X(z) = \sum_{k=0}^{N-1} x(n)z^{-k}.$ <p>The phase of the output is inverted to match the result of the ChirpZ transform on the unit circle with that of the FFT.</p>
<code>chirpZf(data, f1, f2, df)</code>	Chirp Z Transform except that the transform parameters are specified by real-valued starting frequency $f1$, end frequency $f2$ and frequency resolution df . The transform is confined to the unit circle because both A and W have unit magnitude.
<code>chol(w)</code>	Returns the Cholesky decomposition U of a positive definite symmetric matrix w such that $w = U^t \times U$. Note that only the upper triangle of w is actually used in the computation.
<code>chunk(w, n)</code>	Returns chunk n from 4D wave w . Added in Igor Pro 7.00.
<code>clip(w, low, high)</code>	Returns the values in the wave w clipped between the <i>low</i> and the <i>high</i> parameters. If w contains NaN or INF values, they are not modified. The result retains the same number type as the input wave w irrespective of the range of the <i>low</i> and <i>high</i> input parameters.
<code>cmplx(re, im)</code>	Returns a complex token from two real tokens. <i>re</i> and <i>im</i> must have the same dimensionality. Added in Igor Pro 7.00.
<code>col(w, c)</code>	Returns column c from matrix wave w .
<code>colRepeat(w, n)</code>	Returns a matrix that consists of n identical columns containing the data in the wave w . If w is a 2D wave, it is treated as if it were a single column containing all the data. Higher dimensions are supported on a layer-by-layer basis. MatrixOp returns an error if $n < 2$. Added in Igor Pro 7.00.
<code>conj(matrixWave)</code>	Complex conjugate of the input expression.
<code>const(r, c, val)</code>	Returns an $(r \times c)$ matrix where all elements are equal to <i>val</i> . The data type of the returned matrix is the same as that of <i>val</i> . See also <code>zeroMat</code> below. Added in Igor Pro 7.00.

`convolve (w1, w2, opt)`

Convolution of $w1$ with $w2$ subject to options opt . The dimensions of the result are determined by the largest dimensions of $w1$ and $w2$ with the number of rows padded (if necessary) so that they are even. Supported options include $opt=0$ for circular convolution and $opt=4$ for acausal convolution.

For fast 2D convolutions where $w1$ is an image and $w2$ is a square kernel of the same numeric type, you can use $opt=-1$ or $opt=-2$. When $opt=-1$ the convolution at the boundaries is evaluated using zero padding. When $opt=-2$ the padding is a reflection of $w1$ about the boundaries. When working with integer waves the kernel is internally normalized by the sum of its elements. The kernel for floating point waves remain unchanged.

To convolve an image in $w1$ with a smaller point spread function in $w2$ you can use:

$opt=-1$ if you want to pad the image boundaries with zeros or

$opt=-2$ if you want to pad the boundaries by reflecting image values about each boundary.

The negative options are designed for a very optimized convolution calculation which requires that $w1$ and $w2$ have the same numeric type. If the size of the point spread function is larger than about 13x13 it may become more efficient to compute the convolution using the positive options.

`correlate (w1, w2, opt)`

Correlation of $w1$ with $w2$ subject to options opt . The dimensions of the result are determined by the largest dimensions of $w1$ and $w2$ with the number of rows padded (if necessary) so that they are even. Supported options include $opt=0$ for circular correlation and $opt=4$ for acausal correlation.

`cos (w)`

Cosine of w .

`cosh (w)`

Hyperbolic cosine of w . Added in Igor Pro 7.00.

`covariance (w)`

Returns the sample covariance matrix. Added in Igor Pro 8.00.

If w is a rows x cols real matrix then the returned value is the cols x cols matrix Q with elements

$$q_{ij} = \frac{1}{rows-1} \sum_{i=0}^{rows-1} (w_{ij} - \bar{w}_j)(w_{ik} - \bar{w}_k),$$

where

$$\bar{w}_k = \frac{1}{rows} \sum_{i=0}^{rows-1} w_{ik}.$$

`crossCovar (w1, w2, opt)`

Returns the cross-covariance for 1D waves $w1$ and $w2$. The options parameter opt can be set to 0 for the raw cross-covariance or to 1 if you want the results to be normalized to 1 at zero offset. If $w1$ has N rows and $w2$ has M rows then the returned vector is of length N+M-1. The cross-covariance is computed by subtracting the mean of each input followed by correlation and optional normalization. See also **Correlate** with the /NODC flag.

<code>decimateMinMax(<i>w</i>, <i>N</i>)</code>	<p>Decimates the matrix <i>w</i> on a column by column basis. Each column is divided into <i>N</i> bins and each bin is represented by its minimum and maximum values.</p> <p>For an RxC real valued input <i>w</i>, the output is an array of dimensions 2Nx<i>C</i> and the extrema are ordered as in {min0,max0,min1,max1...}.</p> <p>Because of roundoff, sequential bins may not represent the same number of elements when R is not an integer multiple of <i>N</i>. In this case you not count on the exact form of the bin size roundoff.</p> <p>The decimateMinMax keyword was added in Igor Pro 9.00.</p>
<code>det(<i>w</i>)</code>	Returns a scalar corresponding to the determinant of matrix <i>w</i> , which must be real.
<code>diagonal(<i>w</i>)</code>	Creates a square matrix that has the same number of rows as <i>w</i> . All elements are zero except for the diagonal elements which are taken from the first column in <i>w</i> . Use DiagRC if the input is not an existing wave (such as a result from another function).
<code>diagRC(<i>w</i>, <i>rows</i>, <i>cols</i>)</code>	2D matrix of dimensions <i>rows</i> by <i>cols</i> . All matrix elements are set to zero except those of the diagonal which are filled sequentially from elements of <i>w</i> . The dimensionality of <i>w</i> is unimportant. If the total number of elements in <i>w</i> is less than the number of elements on the diagonal then all elements will be used and the remaining diagonal elements will be set to zero.
<code>e</code>	Returns the base of the natural logarithm.
<code>equal(<i>w1</i>, <i>w2</i>)</code>	<p>Returns unsigned byte result with 1 for equality and zero otherwise. The dimensionality of the result matches the dimensionality of the largest parameter.</p> <p>Either or both <i>w1</i> and <i>w2</i> can be constants (i.e., one row by one column).</p> <p>If <i>w1</i> and <i>w2</i> are not constants, they must have the same number of rows and columns. If <i>w1</i> or <i>w2</i> have multiple layers, they must either have the same number of layers or one of them must have a single layer.</p> <p>Both parameters can be either real or complex. A comparison of a real with a complex parameter returns zero.</p>
<code>erf(<i>w</i>)</code>	<p>Returns the error function (see erf) for real values in <i>w</i>.</p> <p>Added in Igor Pro 7.00.</p>
<code>erfc(<i>w</i>)</code>	<p>Returns the complementary error function (see erfc) for real values in <i>w</i>.</p> <p>Added in Igor Pro 7.00.</p>
<code>exp(<i>w</i>)</code>	Exponential function for <i>w</i> which can be real or complex, scalar or a matrix.
<code>expIntegralE1(<i>w</i>)</code>	Returns the exponential integral for real arguments <i>w</i> . See also ExpIntegralE1 . Added in Igor Pro 9.00.
<code>expm(<i>w</i>)</code>	Returns the matrix exponential of a real-valued square matrix <i>w</i> . Added in Igor Pro 9.00.
<code>FCT(<i>w</i>, <i>dir</i>)</code>	See Trigonometric Transform Functions on page V-571.
<code>fft(<i>w</i>, <i>options</i>)</code>	<p>FFT of <i>w</i>.</p> <p><i>w</i> must have an even number of rows.</p> <p><i>options</i> contains a binary field flag. Set bit 1 to 1 if you want to disable the zero centering (see /Z flag in the FFT operation). Other bits are reserved.</p> <p>MatrixOp does not support wave scaling and therefore it does not produce the same wave scaling changes as the FFT operation.</p>

<code>floor (w)</code>	Largest integer smaller than w . If w is complex the function is separately applied to the real and imaginary parts.
<code>forwardSub (L, b)</code>	<p>Returns a column vector solution for the matrix equation $Lx=b$, where L is an (NxN) wave representing a lower triangular matrix and b is a column vector of N rows. If b has additional columns they are ignored.</p> <p>Ideally, L and b should be either SP or DP waves (real or complex). Other numeric data types are supported with a slight performance penalty.</p> <p>This function is typically used in solving linear equations following a matrix decomposition into lower and upper triangular matrices (e.g., Cholesky), with an expression of the form:</p> <p><code>MatrixOp/O solVector=backwardSub (U, forwardSub (L, b))</code></p> <p>where U and L are the upper and lower triangular factors.</p> <p>Added in Igor Pro 7.00.</p>
<code>fp32 (w)</code>	<p>Converts w to 32-bit single precision floating point representation. See also the /NPRM flag below for more information.</p> <p>Added in Igor Pro 7.00.</p>
<code>fp64 (w)</code>	<p>Converts w to 64-bit single precision floating point representation. See also the /NPRM flag below for more information.</p> <p>Added in Igor Pro 7.00.</p>
<code>Frobenius (w)</code>	Returns the Frobenius norm of a matrix defined as the square root of the sum of the squared absolute values of all elements.
<code>FST (w, dir)</code>	See Trigonometric Transform Functions on page V-571.
<code>FSST (w, dir)</code>	See Trigonometric Transform Functions on page V-571.
<code>FSCT (w, dir)</code>	See Trigonometric Transform Functions on page V-571.
<code>FSST2 (w, dir)</code>	See Trigonometric Transform Functions on page V-571.
<code>FSCT2 (w, dir)</code>	See Trigonometric Transform Functions on page V-571.
<code>gamma (w)</code>	Returns the gamma function for real arguments w . See also gamma . Added in Igor Pro 9.00.
<code>gammaLn (w)</code>	Returns the natural log of the gamma function for real arguments w . See also gammln . Added in Igor Pro 9.00.
<code>getDiag (w2d, d)</code>	<p>Returns a 1D wave that contains diagonal d of $w2d$. $d=0$ is the main diagonal, $d>0$ correspond to upper diagonals and $d<0$ to lower diagonals.</p> <p>Added in Igor Pro 7.00.</p>
<code>greater (a, b)</code>	Returns an unsigned byte for the truth of $a > b$. Both a and b must be real but one or both can be constants (see <code>equal ()</code> above). The dimensionality of the result matches the dimensionality of the largest parameter.
<code>greaterOrEqual (a, b)</code>	<p>Returns an unsigned byte for the truth of $a \geq b$. Both a and b must be real but one or both can be constants (see <code>equal ()</code> above). The dimensionality of the result matches the dimensionality of the largest parameter.</p> <p>Added in Igor Pro 9.00.</p>
<code>hypot (w1, w2)</code>	<p>Returns the square root of the sum of the squares of $w1$ and $w2$.</p> <p>Added in Igor Pro 7.00.</p>

<code>ifft(<i>w</i>, <i>options</i>)</code>	<p>IFFT of <i>w</i>.</p> <p><i>options</i> is a bitwise parameter defined as follows:</p> <p>Bit 0: Forces the result to be real, like the IFFT operation /C flag.</p> <p>Bit 1: Disables center-zero.</p> <p>Bit 2: Swaps the results.</p> <p>See Setting Bit Parameters on page IV-12 for details about bit settings.</p> <p>MatrixOp does not support wave scaling and therefore it does not produce the same wave scaling changes as the IFFT operation.</p>
<code>identity(<i>n</i>, <i>m</i>)</code> <code>identity(<i>n</i>)</code>	<p>Creates a computational object that is an identity matrix. If you use a single argument <i>n</i>, the identity created is an (<i>n</i><i>x</i><i>n</i>) square matrix with 1's for diagonal elements (the remaining elements are set to zero). If you use both arguments, the function creates an (<i>n</i><i>x</i><i>m</i>) zero matrix and fills its diagonal elements with 1's. Note that the identity is created at runtime and persists only for the purpose of the specific operation.</p>
<code>imag(<i>w</i>)</code>	Returns the imaginary part of <i>w</i> .
<code>indexCols(<i>w</i>)</code>	<p>Returns a token of the same dimensions as <i>w</i> where each element is equal to its column index. This is the MatrixOp equivalent of q used on righthand side of a standard wave assignment statement. The returned token is unsigned 32-bit integer if the number of columns in <i>w</i> is less than 2³² or double-precision floating point otherwise.</p> <p>Added in Igor Pro 8.00.</p>
<code>indexRows(<i>w</i>)</code>	<p>Returns a token of the same dimensions as <i>w</i> where each element is equal to its row index. This is the MatrixOp equivalent of p used on righthand side of a standard wave assignment statement. The returned token is unsigned 32-bit integer if the number of rows in <i>w</i> is less than 2³² or double-precision floating point otherwise.</p> <p>Added in Igor Pro 8.00.</p>
<code>inf()</code>	<p>Returns INF.</p> <p>Added in Igor Pro 7.00.</p>
<code>insertMat(<i>s</i>, <i>d</i>, <i>r</i>, <i>c</i>)</code>	<p>Inserts matrix <i>s</i> into matrix <i>d</i> starting at row <i>r</i> and column <i>c</i>. The waves <i>s</i> and <i>d</i> must be of the same numeric data type. The inserted range is clipped to the dimensions of the wave <i>d</i>. Both <i>r</i> and <i>c</i> must be non-negative.</p> <p>Added in Igor Pro 7.00.</p>
<code>int8(<i>w</i>)</code>	<p>Converts <i>w</i> to 8-bit signed integer representation. See also the /NPRM flag below for more information.</p> <p>Added in Igor Pro 7.00.</p>
<code>int16(<i>w</i>)</code>	<p>Converts <i>w</i> to 16-bit signed integer representation. See also the /NPRM flag below for more information.</p> <p>Added in Igor Pro 7.00.</p>
<code>int32(<i>w</i>)</code>	<p>Converts <i>w</i> to 32-bit signed integer representation. See also the /NPRM flag below for more information.</p> <p>Added in Igor Pro 7.00.</p>
<code>integrate(<i>w</i>, <i>opt</i>)</code>	<p>Returns a running sum of <i>w</i>.</p> <p>If <i>opt</i>=0 the sum runs over the entire input wave treating the columns of 2D waves as if they were part of one big column.</p> <p>If <i>opt</i>=1 the running sum is computed separately for each column of a 2D wave.</p> <p>Added in Igor Pro 7.00.</p>

`intMatrix(w)`

Returns a double-precision matrix of the same dimensions as w .

Each element of the returned matrix is the sum of all the corresponding elements of w that are above and to the left of it, i.e.,

$$out_{ij} = \sum_{m=0}^i \sum_{n=0}^j w_{mn}.$$

The utility of this function is apparent in the following relationship:

$$\sum_{i=x_1}^{x_2} \sum_{j=y_1}^{y_2} w_{ij} = out[x_2, y_2] - out[x_2, y_1 - 1] - out[x_1 - 1, y_2] + out[x_1 - 1, y_1 - 1].$$

`intMatrix` was added in Igor Pro 7.00.

`inv(w)`

Returns the inverse of the square matrix w .

If w is not invertible, the operation returns a matrix of the same dimensions where all elements are set to NaN.

`inverseErf(w)`

Returns the inverse error function (see **inverseErf**) for the real values in w .

Added in Igor Pro 7.00.

`inverseErfc(w)`

Returns the inverse complementary error function (see **inverseErfc**) for the real values in w .

Added in Igor Pro 7.00.

`kronProd(u, v)`

Returns the Kronecker product of matrices u and v .

`kronProd` was added in Igor Pro 9.00.

If u has dimensions $M \times N$ and v has dimensions $P \times Q$ then the returned block matrix has dimensions $M \times P \times N \times Q$. It is given by

$$u \otimes v = \begin{bmatrix} u_{11}v_{11} & u_{11}v_{12} & \cdots & u_{11}v_{1q} & \cdots & \cdots & u_{1n}v_{11} & u_{1n}v_{12} & \cdots & u_{1n}v_{1q} \\ u_{11}v_{21} & u_{11}v_{22} & \cdots & u_{11}v_{2q} & \cdots & \cdots & u_{1n}v_{21} & u_{1n}v_{22} & \cdots & u_{1n}v_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ u_{11}v_{p1} & u_{11}v_{p2} & \cdots & u_{11}v_{pq} & \cdots & \cdots & u_{1n}v_{p1} & u_{1n}v_{p2} & \cdots & u_{1n}v_{pq} \\ \vdots & \vdots & & \vdots & & & \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots & & & \vdots & \vdots & & \vdots \\ u_{m1}v_{11} & u_{m1}v_{12} & \cdots & u_{m1}v_{1q} & \cdots & \cdots & u_{mn}v_{11} & u_{mn}v_{12} & \cdots & u_{mn}v_{1q} \\ u_{m1}v_{21} & u_{m1}v_{22} & \cdots & u_{m1}v_{2q} & \cdots & \cdots & u_{mn}v_{21} & u_{mn}v_{22} & \cdots & u_{mn}v_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & & \vdots \\ u_{m1}v_{p1} & u_{m1}v_{p2} & \cdots & u_{m1}v_{pq} & & & u_{mn}v_{p1} & u_{mn}v_{p2} & & u_{mn}v_{pq} \end{bmatrix}$$

See *Examples* below for an example using `kronProd`.

<code>layer(w, n, chunk)</code>	<p>Returns a layer of a 3D or 4D wave.</p> <p>The layer function was added in Igor Pro 7.00.</p> <p>The optional <i>chunk</i> parameter was added in Igor Pro 9.00.</p> <p>If you omit <i>chunk</i> and <i>w</i> is a 3D wave, the layer function returns layer <i>n</i> from <i>w</i>.</p> <p>If you omit <i>chunk</i> and <i>w</i> is a 4D wave, the layer function returns layer <i>n</i> of chunk 0 from <i>w</i>.</p> <p>If you include <i>chunk</i>, <i>w</i> must be a 4D wave and the layer function returns layer <i>n</i> of the specified chunk from <i>w</i>.</p> <p>In all cases <i>w</i> must be a an actual wave and not an expression.</p>
<code>layerStack(w, n)</code>	<p>The layerStack function takes a 4D input wave <i>w</i> and a layer number <i>n</i> and returns a 3D wave (stack) that contains sequentially the data in layer <i>n</i> of each chunk of <i>w</i>. <i>w</i> must be a wave and not a derived token.</p> <p>layerStack was added in Igor Pro 9.00.</p> <p>Example:</p> <pre>// Create 3D stack from red channel of 4D RGB movie wave MatrixOP/O redChannelStack=layerStack(myRGBMovie,0)</pre>
<code>limitProduct(w1, w2)</code>	<p>Returns a partial element-by-element multiplication of waves <i>w1</i> and <i>w2</i>.</p> <p>It is assumed that the dimensions of <i>w1</i> are greater or equal to that of <i>w2</i>. If <i>w2</i> is of dimensions NxM then the function returns a matrix of the same dimensions as <i>w1</i> with the first NxM elements contain the product of the corresponding elements in <i>w1</i> and <i>w2</i> and the remaining elements set to zero.</p> <p>The function is designed to be used in filtering applications where the size of the kernel <i>w2</i> is much smaller than the size of the input <i>w1</i>.</p> <p>Added in Igor Pro 7.00.</p>
<code>log(w)</code>	Returns the log base 10 of a token <i>w</i> which can be real or complex, scalar or a matrix.
<code>log2(w)</code>	Returns the log base 2 of a token <i>w</i> which can be real or complex, scalar or a matrix. log2 was added in Igor Pro 9.00.
<code>ln(w)</code>	Returns the natural logarithm of a token <i>w2</i> which can be real or complex, scalar or a matrix.
<code>mag(w)</code>	Returns a real valued wave containing the magnitude of each element of <i>w</i> . This is equivalent to the <code>abs</code> function.
<code>magSqr(w)</code>	Returns a real value wave containing the square of a real <i>w</i> or the squared magnitude of complex <i>w</i> .
<code>maxAB(a, b)</code>	<p>Returns the larger of the two real numbers <i>a</i> and <i>b</i>.</p> <p>maxAB does not support NaN or complex inputs.</p> <p>Added in Igor Pro 7.00.</p>
<code>maxMagAB(a, b)</code>	<p>For each data point in the tokens <i>a</i> and <i>b</i>, maxMagAB returns the one with the larger absolute value.</p> <p>maxMagAB does not support complex numbers.</p> <p>Added in Igor Pro 9.00.</p>

<code>maxCols (w)</code>	<p>Returns a 1D wave containing the maximum value of each column in the wave <i>w</i>. If <i>w</i> is complex the output contains the maximum magnitude of each column of <i>w</i>.</p> <p><code>maxCols</code> does not support NaN.</p> <p>Added in Igor Pro 7.00.</p>
<code>maxRows (w)</code>	<p>Returns a 1D wave containing the maximum value of each row in the wave <i>w</i>. If <i>w</i> is complex the output contains the maximum magnitude of each row of <i>w</i>.</p> <p><code>maxRows</code> does not support NaN.</p> <p>Added in Igor Pro 8.00.</p>
<code>maxVal (w)</code>	<p>Returns the maximum value of the wave <i>w</i>. If <i>w</i> is complex it returns the maximum magnitude of <i>w</i>.</p> <p>When <i>w</i> is real, the returned number type is the same as <i>w</i>'s.</p> <p>When <i>w</i> is complex, the result is real and represents the maximum of the magnitudes of the elements of <i>w</i>. If <i>w</i> is double precision, then the result is double precision; otherwise it is single precision.</p> <p>When <i>w</i> is a 3D or 4D wave, <code>maxVal</code> returns a (1 x 1 x layers x chunks) data token.</p> <p><code>maxVal</code> does not support NaN values.</p>
<code>mean (w)</code>	<p>Returns the mean value <i>w</i>.</p>
<code>minAB (a, b)</code>	<p>Returns the smaller of the two real numbers <i>a</i> and <i>b</i>.</p> <p><code>minAB</code> does not support NaN or complex inputs.</p> <p>Added in Igor Pro 7.00.</p>
<code>minMagAB (a, b)</code>	<p>For each data point in the tokens <i>a</i> and <i>b</i>, <code>minMagAB</code> returns the one with the smaller absolute value.</p> <p><code>minMagAB</code> does not support complex numbers.</p> <p>Added in Igor Pro 9.00.</p>
<code>minCols (w)</code>	<p>Returns a 1D wave containing the minimum value of each column in the wave <i>w</i>. If <i>w</i> is complex the output contains the minimum magnitude of each column of <i>w</i>.</p> <p><code>minCols</code> does not support NaN.</p> <p>Added in Igor Pro 8.00.</p>
<code>minRows (w)</code>	<p>Returns a 1D wave containing the minimum value of each row in the wave <i>w</i>. If <i>w</i> is complex the output contains the minimum magnitude of each row of <i>w</i>.</p> <p><code>minRows</code> does not support NaN.</p> <p>Added in Igor Pro 8.00.</p>
<code>minVal (w)</code>	<p>Returns the minimum value of the wave <i>w</i>. If <i>w</i> is complex the function returns the minimum magnitude of <i>w</i>.</p> <p>When <i>w</i> is real, the returned number type is the same as <i>w</i>'s.</p> <p>When <i>w</i> is complex, the result is real and represents the minimum of the magnitudes of the elements of <i>w</i>. If <i>w</i> is double precision, then the result is double precision; otherwise it is single precision.</p> <p>When <i>w</i> is a 3D or 4D wave, <code>minVal</code> returns a (1 x 1 x layers x chunks) data token.</p> <p><code>minVal</code> does not support NaN values.</p>

<code>mod(w, b)</code>	Returns the remainder after dividing w by b . b can be a scalar or a matrix of the same dimensions as w . Added in Igor Pro 7.00.
<code>nan()</code>	Returns NaN. Added in Igor Pro 7.00.
<code>normalize(w)</code>	Normalized version of a vector or a matrix. Normalization is such that the returned token should have a unity magnitude except if all elements are zero, in which case output is unchanged.
<code>normalizeCols(w)</code>	Divides each column of the real wave w by the square root of the sum of the squares of all elements of the column.
<code>normalizeRows(w)</code>	Divides each row of the real wave w by the square root of the sum of the squares of all the elements in that row.
<code>normP(w, pn)</code>	Returns the p-norm of matrix w defined by $\ w\ _p = \left(\sum_{j=0}^{cols} \sum_{i=0}^{rows} w[i][j] ^p \right)^{1/p}.$ <p>The case of $pn=2$ is equivalent to the MatrixOp Frobenius function. Added in Igor Pro 9.00.</p>
<code>numCols(w)</code>	Returns the number of columns in the wave w . When w is 1D the function returns 1.
<code>numPoints(w)</code>	Returns the number of points in a layer of w .
<code>numRows(w)</code>	Returns the number of rows in w .
<code>numType(w)</code>	Number the number type of w : 0: w is a normal number 1: w is +/-INF 2: w is NaN
<code>oneNorm(w)</code>	Returns the 1-norm of matrix w defined as the maximum absolute column sum $\ w\ = \max_{0 \leq j \leq cols} \left(\sum_{i=0}^{rows} w[i][j] \right).$ <p>Added in Igor Pro 9.00.</p>
<code>outerProduct(w1, w2)</code>	For a 1D wave $w1$ containing M points and a 1D wave $w2$ containing N points, <code>outerProduct</code> returns an M by N matrix where the (i,j) element is: <code>out[i, j] = w1[i] * conj(w2[j])</code> Added in Igor Pro 8.00.
<code>p2Rect(w)</code>	Converts each element of w from polar to rectangular representation.
<code>phase(w)</code>	Returns a real valued wave containing the phase of each element of w calculated using <code>phase=atan2(y, x)</code> .
<code>Pi</code>	Returns π .
<code>powC(w1, w2)</code>	Complex valued $w1^{w2}$ where $w1$ and $w2$ can be real or complex.
<code>powR(x, y)</code>	Returns x^y for real x and y .

<code>productCol(w, c)</code>	Returns the a (1 x 1) wave containing the product of the elements in column <i>c</i> of wave <i>w</i> . The output is double precision real or complex. Added in Igor Pro 7.00.
<code>productCols(w)</code>	Returns a (1 x cols) wave where each entry is the product of all the elements in the corresponding column. The output is double precision real or complex. Added in Igor Pro 7.00.
<code>productDiagonal(w, d)</code>	Returns a (1 x 1) wave containing the product of the elements on the specified diagonal of wave <i>w</i> . <i>d</i> =0 is the main diagonal, <i>d</i> >0 correspond to upper diagonals and <i>d</i> <0 to lower diagonals. The output is double precision real or complex. Added in Igor Pro 7.00.
<code>productRow(w, r)</code>	Returns the a (1 x 1) wave containing the product of the elements in row <i>r</i> of wave <i>w</i> . The output is double precision real or complex. Added in Igor Pro 7.00.
<code>productRows(w)</code>	Returns a (1 x rows) wave where each entry is the product of all the elements in the corresponding row. The output is double precision real or complex. Added in Igor Pro 7.00.
<code>quat(arg)</code>	See Functions Using Quaternions on page V-573.
<code>quatToAxis(qIn)</code>	See Functions Using Quaternions on page V-573.
<code>quatToEuler(qIn, mode)</code>	See Functions Using Quaternions on page V-573.
<code>quatToMatrix(qIn)</code>	See Functions Using Quaternions on page V-573.
<code>r2Polar(w)</code>	Performs the equivalent of the r2polar function on each element of <i>w</i> , i.e., each complex number (x+iy) is converted into the polar representation r,theta with $x+iy=r*\exp(i*theta)$
<code>real(w)</code>	Returns the real part of <i>w</i> .
<code>rec(w)</code>	Returns the reciprocal of each element in <i>w</i> .
<code>redimension(w, nr, nc)</code>	Returns an (nr x nc) matrix from the data in the wave <i>w</i> . The data in <i>w</i> are moved contiguously (column by column regardless of dimensionality) into the output. If the output size is larger than <i>w</i> the remaining points are set to zero. If <i>w</i> contains more than one layer the new dimensions apply on a layer by layer basis. For example: Make/N=(10,20,30) ddd = p + 10*q + 100*r MatrixOp/O aa = redimension(ddd,25,1) creates a wave aa with dimensions (25,1,30). Added in Igor Pro 7.00.
<code>replace(w, findVal, replacementVal)</code>	Replace in wave <i>w</i> every occurrence of <i>findVal</i> with <i>replacementVal</i> . The wave <i>w</i> retains its dimensionality and number type. <i>replacementVal</i> is converted to the same number type as <i>w</i> which may cause truncation.
<code>replaceNaNs(w, replacementVal)</code>	Replaces every occurrence of NaN in the wave <i>w</i> with <i>replacementVal</i> . The wave <i>w</i> retains its dimensionality. <i>replacementVal</i> is converted to the same number type as <i>w</i> which may cause truncation.

<code>reverseCol(w, c)</code>	Returns array <i>w</i> with column <i>c</i> in reverse order. Added in Igor Pro 7.00.
<code>reverseCols(w)</code>	Returns array <i>w</i> with all columns in reverse order. Added in Igor Pro 7.00.
<code>reverseRow(w, r)</code>	Returns array <i>w</i> with row <i>r</i> in reverse order. Added in Igor Pro 7.00.
<code>reverseRows(w)</code>	Returns array <i>w</i> with all rows in reverse order. Added in Igor Pro 7.00.
<code>rotateChunks(w, n)</code>	Returns a matrix where the last <i>n</i> chunks of <i>w</i> are moved to chunks [0, <i>n</i> -1]. If <i>n</i> is negative then the first abs(<i>n</i>) chunks are moved to the end of the data. It is an error to pass NaN for <i>n</i> . Added in Igor Pro 7.00.
<code>rotateCols(w, nc)</code>	Rotates the columns of a 2D wave <i>w</i> so that the last <i>nc</i> columns are moved to columns [0, <i>nc</i> -1] of the data. If <i>nc</i> is negative the first abs(<i>nc</i>) columns are moved to columns [<i>n</i> -1- <i>nc</i> , <i>n</i> -1]. Here <i>n</i> is the total number of columns. It is an error to pass NaN for <i>nc</i> . If <i>nc</i> is greater than the number of columns then the effective rotation is mod(<i>nc</i> , actualCols).
<code>rotateLayers(w, n)</code>	Returns a matrix where the last <i>n</i> layers of <i>w</i> are moved to layers [0, <i>n</i> -1]. If <i>n</i> is negative then the first abs(<i>n</i>) layers are moved to the end of the data. It is an error to pass NaN for <i>n</i> . Added in Igor Pro 7.00.
<code>rotateRows(w, nr)</code>	Rotates the rows of a 2D wave <i>w</i> so that the last <i>nr</i> rows are moved to rows [0, <i>nr</i> -1] of the data. If <i>nr</i> is negative the first abs(<i>nr</i>) rows are moved to rows [<i>n</i> -1- <i>nr</i> , <i>n</i> -1] where <i>n</i> is the total number of rows. It is an error to pass NaN for <i>nr</i> . If <i>nr</i> is greater than the number of rows then the effective rotation is mod(<i>nr</i> , actualRows).
<code>round(z)</code>	Rounds <i>z</i> to the nearest integer. The rounding method is “away from zero”.
<code>row(w, r)</code>	Returns row <i>r</i> from matrix wave <i>w</i> . The returned row is a (1xC) wave where C is the number of columns in <i>w</i> . To convert it to a 1D wave use Redimension/N=(C). See also ImageTransform getRow.
<code>rowRepeat(w, n)</code>	Returns a matrix that consists of <i>n</i> identical rows containing the data in the wave <i>w</i> . If <i>w</i> is a 2D wave, it is treated as if it were a single column containing all the data. Higher dimensions are supported on a layer-by-layer basis. MatrixOp returns an error if <i>n</i> <2. Added in Igor Pro 7.00.
<code>scale(w, low, high)</code>	Returns the values in the wave <i>w</i> scaled between the <i>low</i> and the <i>high</i> parameters. If <i>w</i> contains NaN or INF values, they are not modified. The result retains the same number type as that of <i>w</i> irrespective of the range of the <i>low</i> and <i>high</i> input parameters.
<code>scaleCols(w1, w2)</code>	Returns a matrix of the same dimensions as <i>w1</i> where each column of <i>w1</i> is scaled by the value in the corresponding row of the 1D wave <i>w2</i> . The number of rows in <i>w2</i> must equal the number of columns in <i>w1</i> . Added in Igor Pro 7.00.
<code>scaleRows(w1, w2)</code>	Returns a matrix of the same dimensions as <i>w1</i> where each row of <i>w1</i> is scaled by the value in the corresponding row of the 1D wave <i>w2</i> . The number of rows in <i>w2</i> must equal the number of rows in <i>w1</i> . Added in Igor Pro 7.00.

<code>select(w, sr, sc)</code>	<p>Returns a matrix consisting of the elements of matrix <i>w</i> for which the row index satisfies $(p \% sr) == 0$ and the column index satisfies $(q \% sc) == 0$. The first row and column are always selected.</p> <p><i>sr</i> and <i>sc</i> must be real, finite, and non-negative.</p> <p>Added in Igor Pro 9.00.</p>
<code>setCol(w2d, c, w1d)</code>	<p>Returns the data in the <i>w2d</i> with the contents of <i>w1d</i> stored in column <i>c</i>.</p> <p><i>w1d</i> must have at least as many elements as the number of rows of <i>w2d</i>. <i>w2d</i> and <i>w1d</i> must be either both real or both complex.</p> <p>Added in Igor Pro 7.00.</p>
<code>setColsRange(w, low, high)</code>	<p>Returns a matrix in which each column of <i>w</i> is scaled to the range [<i>low</i>,<i>high</i>]. <i>w</i> must be real-valued.</p> <p><code>setColsRange</code> was added in Igor Pro 9.00.</p>
<code>setNaNs(w, mask)</code>	<p>Returns the data in the wave <i>w</i> with NaNs stored where the mask wave is non-zero. The wave <i>w</i> can be of any numeric type.</p> <p><i>mask</i> must have the same dimensions as <i>w</i> and must be real. It is usually the result of another expression. For example, to set all values in the destination to NaN where <i>w</i> is greater than 5:</p> <p><code>MatrixOp/ou = setNaNs(w, greater(w, 5))</code></p> <p>Added in Igor Pro 7.00.</p>
<code>setOffDiag(w, d, w1)</code>	<p>Returns the data in the <i>w</i> with the contents of <i>w1</i> stored in diagonal <i>d</i>.</p> <p><i>d</i>=0 is the main diagonal of <i>w</i>, <i>d</i>>0 correspond to upper diagonals and <i>d</i><0 to lower diagonals. <i>w</i> and <i>w1</i> can either be both real or both complex.</p> <p>Added in Igor Pro 7.00.</p>
<code>setRow(w2d r, w1d)</code>	<p>Returns the data in the <i>w2d</i> with the contents of <i>w1d</i> stored in row <i>r</i>.</p> <p><i>w1d</i> must have at least as many elements as the number of columns in <i>w2d</i>. <i>w2d</i> and <i>w1d</i> must be either both real or both complex.</p> <p>Added in Igor Pro 7.00.</p>
<code>sgn(w)</code>	<p>Returns the sign of each element in <i>w</i>. It returns -1 for negative numbers and 1 otherwise. It does not accept complex numbers.</p>
<code>shiftVector(w, n, val)</code>	<p>Shifts the element of a 1D row-vector <i>w</i> by <i>n</i> elements and fills the displaced elements with <i>val</i>, which must match the data type of <i>w</i> and should be expressed as <code>cmplx(a, b)</code> for complex <i>w</i>.</p>
<code>sin(z)</code>	Sine of <i>z</i> .
<code>sinh(z)</code>	Hyperbolic sine of <i>z</i> . Added in Igor Pro 7.00.
<code>slerp(qIn1, qIn2, frac)</code>	See Functions Using Quaternions on page V-573.
<code>spliceCols(w, c, d, ic)</code>	<p>Returns a matrix where <i>ic</i> columns from matrix <i>d</i> are spliced into matrix <i>w</i> starting at column <i>c</i> of <i>w</i>. Matrix <i>d</i> must have the same number of rows as matrix <i>w</i> and they must both have the same number type. <i>ic</i> must be non-negative integer. If <i>ic</i> is greater than the number of columns in the matrix <i>d</i>, the function repeats as many columns of <i>d</i> as necessary.</p> <p>Added in Igor Pro 9.00.</p>
<code>sq(w)</code>	<p>Returns the square of each element of the matrix <i>w</i>.</p> <p>For complex elements <i>z</i> the returned value is <i>z</i>*<i>z</i>, not <code>magsqr(z)</code>.</p> <p>Added in Igor Pro 9.00.</p>

<code>sqrt (z)</code>	Square root of <i>z</i> .
<code>subRange (w, rs, re, cs, ce)</code>	<p>Returns a contiguous subset of the wave <i>w</i> from starting row <i>rs</i> through ending row <i>re</i> and from starting column <i>cs</i> through ending column <i>ce</i>. This is similar to Duplicate/R except that dimension scaling and labels not preserved.</p> <p>Added in Igor Pro 7.00.</p>
<code>subtractMean (w, opt)</code>	Computes the mean of the real wave <i>w</i> and returns the values of the wave minus the mean value (<i>opt</i> =0). Computes the mean of each column and subtracts it from that column (<i>opt</i> =1). Subtracts the mean of each row from row values (<i>opt</i> =2).
<code>subWaveC (w, r, c, count[, stride])</code>	<p>Returns a subset of the data that is sampled along columns of the wave <i>w</i>, containing count elements starting with the element at row <i>r</i> and column <i>c</i>. By default <i>stride</i>=1 and the sampling is continuous.</p> <p>You can specify a negative stride to sample backwards from the starting element.</p> <p>The operation returns an error if the sampling would exceed the array bounds in either direction.</p> <p>For example:</p> <pre>Make/O/N=(22,33) ddd=x MatrixOP/O/P=1 aa=subWaveC(ddd,4,5,10,2) // aa={4,6,8,10,12,14,16,18,20,0} MatrixOP/O/P=1 aa=subWaveC(ddd,4,5,5,-4) // aa={4,0,18,14,10}</pre>
<code>subWaveR (w, r, c, count[, stride])</code>	<p>Returns a subset of the data that is sampled along rows of the wave <i>w</i>, containing count elements starting with the element at row <i>r</i> and column <i>c</i>. By default <i>stride</i>=1 and the sampling is continuous.</p> <p>You can specify a negative stride to sample backwards from the starting element.</p> <p>The operation returns an error if the sampling would exceed the array bounds in either direction.</p> <p>Examples:</p> <pre>Make/O/N=(10,20) ddd=y // Forward sampling across right boundary MatrixOP/O/P=1 aa=subWaveR(ddd,4,15,6,2) // aa={15,17,19,1,3,5} // Reverse sampling across left boundary MatrixOP/O/P=1 aa=subWaveR(ddd,2,3,5,-1) // aa={3,2,1,0,19}</pre>
<code>sum (z)</code>	Returns the sum of all the elements in expression <i>z</i> .

sumBeams (*w*)

Returns an (*n* × *m*) matrix containing the sum over all layers of all the beams of the 3D wave *w*:

$$out_{ij} = \sum_{k=0}^{nLayers-1} w_{ijk}.$$

A beam is a 1D array in the Z-direction.

sumBeams is a non-layered function which requires that *w* be a proper 3D wave and not the result of another expression.

sumCols (*w*)

Returns a (1 × *m*) matrix containing the sums of the *m* columns in the *n* × *m* input wave *w*:

$$out_j = \sum_{i=0}^{nRows-1} w_{ij}.$$

sumND (*w*)

Returns a 1-point wave containing the sum of the token *w* regardless of its dimensionality.

sumND was added in Igor Pro 9.00.

This example illustrates the difference between sumND and the sum function which operates on a layer-by-layer basis:

```
Make/O/N=(5,6,3) w3D = z
MatrixOP/O/P=1 sumOut=sum(w3D)          // Prints 3 layer sums
MatrixOP/O/P=1 sumNDOut=sumND(w3D)      // Prints single sum
```

sumRows (*w*)

Returns an (*n* × 1) matrix containing the sums of the *n* rows in the *n* × *m* input wave *w*:

$$out_i = \sum_{j=0}^{nCols-1} w_{ij}.$$

sumSqr (*w*)

Sum of the squared magnitude of all elements in *w*.

syncCorrelation (*w*)

Synchronous spectrum correlation matrix for a real valued input matrix wave *w*. See also `asyncCorrelation`.

The correlation matrix is computed by subtracting from each column of *w* its mean value, multiplying the resulting matrix by its transpose, and finally dividing all elements by (*nrows*-1) where *nrows* is the number of rows in *w*.

tan (*w*)

Tangent of *w*.

tanh (*w*)

Hyperbolic tangent of *w*. Added in Igor Pro 7.00.

tensorProduct (*w1*, *w2*)

Returns a 2D matrix that is the tensor product of the 2D matrices *w1* and *w2*. For example, the tensor product of two (2 × 2) matrices is given by:

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \otimes \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} \end{pmatrix}$$

Added in Igor Pro 7.00.

<code>Trace (w)</code>	Returns a real or complex scalar which is the sum of the diagonal elements of <i>w</i> . If <i>w</i> is not a square matrix, the sum is over the elements for which the row and column indices are the same.
<code>transposeVol (w, mode)</code>	<p>For 3D wave <i>w</i>, <code>transposeVol</code> returns a transposed 3D wave depending on the value of the <i>mode</i> parameter:</p> <p><i>mode</i>=1: output=<i>w</i>[p][r][q] <i>mode</i>=2: output=<i>w</i>[r][p][q] <i>mode</i>=3: output=<i>w</i>[r][q][p] <i>mode</i>=4: output=<i>w</i>[q][r][p] <i>mode</i>=5: output=<i>w</i>[q][p][r]</p> <p><code>transposeVol</code> is a non-layered function which requires that <i>w</i> be a proper 3D wave and not the result of another expression.</p>
<code>triDiag (w1, w2, w3)</code>	Returns a tri-diagonal matrix where <i>w1</i> is the upper diagonal, <i>w2</i> the main diagonal and <i>w3</i> the lower diagonal. If <i>w2</i> has <i>n</i> points then <i>w1</i> and <i>w3</i> are expected to have <i>n</i> -1 points. The waves can be of any numeric type and the returned wave has a numeric type that accommodates the input.
<code>uint8 (w)</code>	Converts <i>w</i> to 8-bit unsigned integer representation. See also the /NPRM flag below for more information. Added in Igor Pro 7.00.
<code>uint16 (w)</code>	Converts <i>w</i> to 16-bit unsigned integer representation. See also the /NPRM flag below for more information. Added in Igor Pro 7.00.
<code>uint32 (w)</code>	Converts <i>w</i> to 32-bit unsigned integer representation. See also the /NPRM flag below for more information. Added in Igor Pro 7.00.
<code>varBeams (w)</code>	<p>Returns a matrix containing the variances of the beams of the real-valued 3D wave <i>w</i>.</p> <p><code>varBeams</code> was added in Igor Pro 9.00.</p> <p><code>varBeams</code> is a non-layered function which requires that <i>w</i> be a proper 3D wave and not the result of another expression.</p> <p>When <i>w</i> consists of a single layer the returned variances are all NaN.</p>
<code>varCols (w)</code>	Returns a (1 x cols) wave where each element contains the variance of the corresponding column in <i>w</i> .
<code>waveX (w)</code>	<p>For pure wave argument <i>w</i>, <code>waveX</code> returns the X value for each point as determined by the wave's X scaling.</p> <p>Added in Igor Pro 9.00.</p>
<code>waveY (w)</code>	<p>For pure wave argument <i>w</i>, <code>waveY</code> returns the Y value for each point as determined by the wave's Y scaling.</p> <p>Added in Igor Pro 9.00.</p>
<code>waveZ (w)</code>	<p>For pure wave argument <i>w</i>, <code>waveZ</code> returns the Z value for each point as determined by the wave's Z scaling.</p> <p>Added in Igor Pro 9.00.</p>
<code>waveT (w)</code>	<p>For pure wave argument <i>w</i>, <code>waveT</code> returns the T value for each point as determined by the wave's T scaling.</p> <p>Added in Igor Pro 9.00.</p>
<code>waveIndexSet (w1, w2, w3)</code>	

Returns a matrix of the same dimensions as $w1$ with values taken either from $w1$ or from $w3$ depending on values in $w2$ using:

$$out[i][j] = \begin{cases} w1[i][j] & \text{if } w2[i][j] < 0 \\ w3[w2[i][j]] & \text{otherwise} \end{cases}.$$

$w1$ and $w2$ must have the same number of rows and columns. $w1$ and $w3$ must match in number type. $w2$ cannot be unsigned.

Values from $w2$ are used as point number indices into $w3$ which is treated like a 1D wave regardless of its actual dimensionality.

An index value from $w2$ is out-of-bounds if it is greater than or equal to the number of points in $w3$. In this case, the output value is taken from $w1$ as if the index value were negative.

<code>waveMap(w1, w2)</code>	Returns an array of the same dimensions as $w2$ containing the values $w1[w2[i][j]]$. The data type of the output is the same as that of $w1$. Values of $w2$ are taken as 1D integer indices into the $w1$ array. See also IndexSort .
<code>waveChunks(w)</code>	Returns the number of chunks in the wave w . Added in Igor Pro 7.00.
<code>waveLayers(w)</code>	Returns the number of layers in the wave w . Added in Igor Pro 7.00.
<code>wavePoints(w)</code>	Returns the number of points in the wave w . Added in Igor Pro 7.00.
<code>within(w, low, high)</code>	Returns an array of the same dimensions as w with the value 1 where the corresponding element of w is between low and high ($low \leq w[i][j] < high$). Added in Igor Pro 7.00. All parameters must be real. It is an error to pass a NaN as either <i>low</i> or <i>high</i> . It is also an error if $low \geq high$. If w contains NaNs, the corresponding outputs are 0.
<code>zapINFs(w)</code>	Returns a one-column wave containing the sequential data of w with all INF elements removed. The input w can be 1D or 2D but higher dimensions are not supported. Added in Igor Pro 9.00.
<code>zapNaNs(w)</code>	Returns a one-column wave containing the sequential data of w with all NaN elements removed. The input w can be 1D or 2D but higher dimensions are not supported. Added in Igor Pro 9.00.
<code>zeroMat(r, c, nt)</code>	Returns an $(r \times c)$ matrix of number type nt where all entries are set to zero. See WaveType for supported types. See also <code>const</code> above. Added in Igor Pro 7.00.

Trigonometric Transform Functions

FCT(*w*, *dir*)

Computes the fast, real to real cosine transform on 1D wave *w*.

Added in Igor Pro 8.00.

The forward transform (*dir*=1) is defined by:

$$F(k) = \frac{1}{n} \left[f(0) + f(n) \cos(k\pi) \right] + \frac{2}{n} \sum_{i=1}^{n-1} f(i) \cos\left(\frac{ik\pi}{n}\right), \quad k = 0, \dots, n$$

The number of intervals is $n = \text{numpts}(w) - 1$.

The inverse transform (*dir*=-1) is defined by:

$$f(i) = \frac{1}{2} \left[F(0) + F(n) \cos(i\pi) \right] + \sum_{k=1}^{n-1} F(k) \cos\left(\frac{ik\pi}{n}\right), \quad i = 0, \dots, n$$

See **Trigonometric Transforms** on page V-576 below for more information.

FST(*w*, *dir*)

Computes the fast, real to real sine transform on 1D wave *w*.

Added in Igor Pro 8.00.

The forward transform (*dir*=1) is defined by:

$$F(k) = \frac{2}{n} \sum_{i=1}^{n-1} f(i) \sin\left(\frac{ik\pi}{n}\right), \quad k = 1, \dots, n-1$$

where $n = \text{numpts}(w)$.

The inverse transform (*dir*=-1) is defined by:

$$f(i) = \sum_{k=1}^{n-1} F(k) \sin\left(\frac{ik\pi}{n}\right), \quad i = 1, \dots, n-1$$

See **Trigonometric Transforms** on page V-576 below for more information.

FSST(*w*, *dir*)

Computes the fast, real to real staggered sine transform on 1D wave *w*.

Added in Igor Pro 8.00.

The forward direction (*dir*=1) is defined by:

$$F(k) = \frac{1}{n} \sin\left(\frac{(2k-1)\pi}{2}\right) f(n) + \frac{2}{n} \sum_{i=1}^{n-1} f(i) \sin\left(\frac{i(2k-1)\pi}{2n}\right), \quad k = 1, \dots, n$$

where $n = \text{numpts}(w)$.

The inverse transform (*dir*=-1) is defined by:

$$f(i) = \sum_{k=1}^n F(k) \sin\left(\frac{i(2k-1)\pi}{2n}\right), \quad i = 1, \dots, n$$

See **Trigonometric Transforms** on page V-576 below for more information.

FSCT (*w*, *dir*)

Computes the fast, real to real, staggered cosine transform on 1D wave *w*.
Added in Igor Pro 8.00.

The forward direction (*dir*=1) is defined by:

$$F(k) = \frac{1}{n} f(0) + \frac{2}{n} \sum_{j=1}^{n-1} f(j) \cos\left(\frac{j\pi(2k+1)}{2n}\right), \quad k = 0, \dots, n-1$$

The inverse transform (*dir*=-1) is defined by:

$$f(i) = \sum_{k=0}^{n-1} F(k) \cos\left(\frac{(2k+1)i\pi}{2n}\right), \quad i = 0, \dots, n-1$$

See **Trigonometric Transforms** on page V-576 below for more information.

FSST2 (*w*, *dir*)

Computes the fast, real to real, staggered2 sine transform on 1D wave *w*.
Added in Igor Pro 8.00.

The forward direction (*dir*=1) is defined by:

$$F(k) = \frac{2}{n} \sum_{i=1}^n f(i) \sin\left(\frac{(2k-1)(2i-1)\pi}{4n}\right), \quad k = 1, \dots, n$$

The inverse transform (*dir*=-1) is defined by:

$$f(i) = \sum_{k=1}^n F(k) \cos\left(\frac{(2k-1)(2i-1)\pi}{4n}\right), \quad i = 1, \dots, n$$

See **Trigonometric Transforms** on page V-576 below for more information.

FSCT2 (*w*, *dir*)

Computes the fast, real to real, staggered2 cosine transform on 1D wave *w*.
Added in Igor Pro 8.00.

The forward direction (*dir*=1) is defined by:

$$F(k) = \frac{2}{n} \sum_{i=1}^n f(i) \cos\left(\frac{(2k-1)(2i-1)\pi}{4n}\right), \quad k = 1, \dots, n$$

The inverse transform (*dir*=-1) is defined by:

$$f(i) = \sum_{k=1}^n F(k) \cos\left(\frac{(2k-1)(2i-1)\pi}{4n}\right), \quad i = 1, \dots, n$$

See **Trigonometric Transforms** on page V-576 below for more information.

Functions Using Quaternions

These functions create and manipulate quaternion tokens or return quaternion results. They were added in Igor Pro 8.00. See **MatrixOp Quaternion Data Tokens** on page III-146 for background information on quaternions in MatrixOp.

`quat (arg)` Converts *arg* into a quaternion token.

arg can be:

- A scalar which is converted to a real quaternion
- A 1x3 or 3x1 wave which is converted to a pure imaginary quaternion
- The output from another `quat` call

Arithmetic on quaternion tokens obeys quaternion arithmetic rules.

arg must not be complex. The resulting quaternion token is not normalized by the `quat` function.

See **MatrixOp Quaternion Data Tokens** on page III-146 for details.

`quatToMatrix (qIn)`

Converts *qIn* into a quaternion token, if not already a quaternion token, normalizes it, and returns the equivalent 4x4 homogeneous rotation matrix.

`quatToAxis (qIn)`

Converts *qIn* into a quaternion token, if not already a quaternion token, normalizes it, and returns the equivalent axis of rotation and rotation angle. The result is a 4-element wave in which the first three elements define the rotation axis and the last element is the rotation angle in radians.

`quatToEuler (qIn, mode)`

Converts *qIn* into a quaternion token, if not already a quaternion token, and returns a 3x1 wave containing equivalent Euler angles expressed in radians.

The returned Euler angles are phi (rotation about the X axis), theta (rotation about the Y axis), and psi (rotation about the Z axis).

The *mode* parameter defines the rotation sequence. The supported modes are: 121, 123, 131, 132, 212, 213, 231, 232, 312, 313, 321 and 323. 1 designates the X axis, 2 the Y axis and 3 the Z axis. See, for example:

https://www.euclideanspace.com/maths/geometry/rotations/conversions/quaternionToEuler/quat_2_euler_paper_ver2-1.pdf

`axisToQuat (ax)` *ax* is a 4-element wave with the axis of rotation in the first 3 elements and the rotation angle in radians as the last element. Returns a 4-element wave containing the quaternion components representing this rotation.

`slerp (qIn1, qIn2, frac)`

Performs spherical linear interpolation between quaternion *qIn1* and quaternion *qIn2*. *frac* is a value between 0 and 1 representing the amount of desired rotation from *qIn1* to *qIn2*.

The function returns a 4-element wave containing the quaternion components that represent the rotated quaternion. This is useful, for example, in animating a sequence of rotations.

Wave Parameters

MatrixOp was designed to work with 2D waves (matrices) but also works with 1D, 3D and 4D waves. A 1D wave is treated like a 1-column matrix. 3D and 4D waves are treated on a layer-by-layer basis, as if each layer were a matrix.

MatrixOp

You can reference subsets of waves in *expression*. Only two types of subsets are supported: those that evaluate to a single element, which are treated as scalars, and those that evaluate to one or more layers. For example:

<code>wave1d[a]</code>	Scalar
<code>wave2d[a][b]</code>	Scalar
<code>wave3d[a][b][c]</code>	Scalar
<code>wave3d[][][a]</code>	Layer <i>a</i> from 3D wave
<code>wave3d[][][a,b]</code>	Layers <i>a</i> through <i>b</i> from 3D wave
<code>wave3d[][][a,b,c]</code>	Layers <i>a</i> through <i>b</i> stepping by <i>c</i> from 3D wave

You can pass waves of any dimensions as parameters to MatrixOp functions. For example:

```
Make/O/N=128 wave1d = x
MatrixOp/O outWave = powR(wave1d,2)
```

MatrixOp does not allow using the same 3D wave on both sides of the assignment:

```
MatrixOp/O wave3D = wave3D + 3    // Not allowed
```

See **MatrixOp Wave Data Tokens** on page III-141 for further discussion.

Flags

/AT=allocationType

/AT lets you control the method used to allocate memory in MatrixOP. It was added in Igor Pro 9.00.

Use *allocationType*=0 for generic memory allocation.

Use *allocationType*=1 for Intel scalable allocation.

Use *allocationType*=2 for Intel scalable and aligned allocation.

By default MatrixOP uses scalable and aligned allocation.

/C Provides a complex wave reference for *destWave*. If omitted, MatrixOp creates a real wave reference for *destWave*. The wave reference allows you to refer to the output wave in a subsequent statement of a user-defined function.

/FREE Creates *destWave* as a free wave. Allowed only in functions and only if a simple name or wave reference structure field is specified.

Requires Igor Pro 6.1 or later. For advanced programmers only.

See **Free Waves** on page IV-91 for more discussion.

/NTHR=n Sets the number of threads used to compute the results for 3D waves. Each thread computes the results for a single layer of the input.

By default (*/NTHR* omitted) the calculations are performed by the main thread only.

If *n*=0 the operation uses as many threads you have processors on your computer.

If *n*>0, *n* specifies the number of threads to use. More threads may or may not improve performance.

- /NPRM** Use /NPRM to restrict the automatic promotion of numeric data types in MatrixOp expressions.
- By default, MatrixOp promotes numeric data types so that operations result in reasonable accuracy. In some situations you may want to keep the results as a particular data type even at the risk of truncation or overflow. If you include the /NPRM flag, MatrixOp creates the destination wave using the highest precision data type in the expression. For example, an expression $A=B+C$ where B is 16-bit wave and C is an 8-bit wave results in a 16-bit wave A. Unsigned number types can result only when all operands are unsigned.
- /NPRM is ignored when data promotion is required. For example:
- ```
Make/B/U wave2
MatrixOp/O/NPRM wave1 = -wave2
```
- You can use MatrixOp functions such as int8, int16, etc., to precisely control the number type of any token.
- /NV=mode** Controls use of Intel MKL vectorized functions. /NV was added in Igor Pro 9.00.
- By default, *mode*=1, which uses vectorized functions where possible.
- Set *mode* to 0 to request non-vectorized execution. This may improve speed when MatrixOP is called in a preemptive thread by preventing spawning additional threads.
- /O** Overwrites *destWave* if it already exists.
- /P=printMode**
- Controls printing of the result of the MatrixOp evaluation. /P was added in Igor Pro 8.00.
- /P is ignored if MatrixOp is not running in the main thread.
- printMode* is a value between 0 and 2:
- 0: No printing is done (default).
  - 1: Prints the result from evaluating expression in layer-by-layer order to the history area of the command window and stores the result in the destination wave.
  - 2: Prints the result from evaluating expression in layer-by-layer order to the history area of the command window but does create or store the resulting values in the destination wave. If the destination wave may already exist, you must include the /O flag, even though the destination wave is not changed by the operation.
- Use /P=1 if you want MatrixOp to work normally and then to print the destination wave.
- Use /P=2 if you want MatrixOp to print its result without creating or affecting any waves.
- Values are printed sequentially using 16-digit precision. Output is not formatted to represent rows and columns.
- When the operation's result is a 3D or 4D wave, the results are printed on a layer-by-layer basis.
- See **MatrixOp Printing Examples** on page V-577 below.
- /S** Preserves the dimension scaling, units and wave note of a pre-existing destination wave in a MatrixOp/O command.
- /T=k** Displays results in a table. *k* specifies the behavior when the user attempts to close it.
- 0: Normal with dialog (default).
  - 1: Kills with no dialog.
  - 2: Disables killing.

### Details

MatrixOp has the general form:

```
MatrixOp [flags] destWave = expression
```

*destWave* specifies the wave created by MatrixOp or overwritten by MatrixOp/O.

From the command line, *destWave* can be a simple wave name, a partial data folder path or a full data folder path. In a user-defined function it can be a simple wave name or, if /O is present, a wave reference pointing to an existing wave.

*expression* is a mathematical expression that consists of one or more data tokens combined with the built-in MatrixOp functions and MatrixOp operators listed above. MatrixOp does not support the p, q, r, s, or x, y, z, t symbols that are used in waveform assignment statements.

Data tokens include waves, variables and literal numbers.

You can use any combination of data types for operands. In particular, you can mix real and complex types in *expression*. MatrixOp determines data types of inputs and the appropriate output data type at runtime without regard to any type declaration such as Wave/C.

See **Using MatrixOp** on page III-140 for more information.

### Trigonometric Transforms

The trigonometrics transform functions were added in Igor Pro 8.00.

FST, FCT, FSST, FSCT, FSST2 and FSCT2 are implemented using INTEL MKL library functions. The transforms may automatically execute in multiple threads.

The equations for the definitions of the forward and reverse transforms follow Intel's documentation (see <https://software.intel.com/en-us/mkl-developer-reference-c-trigonometric-transforms-implemented>).

In MatrixOp's implementation, all input and output arrays are zero based. This is illustrated by the following example for the staggered2 cosine:

```
Function DoStaggered2Cosine(inWave)
 Wave inWave

 Variable n = dimsize(inWave,0)
 Make/O/N=(n)/D outStaggered2CosTransform=0
 Variable i, k, theSum, frontFactor=2/n

 for(k=1; k<=n; k+=1)
 theSum=0
 for(i=1; i<=n; i+=1)
 theSum += inWave[i-1] * cos((2*k-1) * (2*i-1) * pi/(4*n))
 endfor
 outStaggered2CosTransform[k-1] = frontFactor * theSum
 endfor
End
```

### Examples

In addition to these examples, see **MatrixOp Optimization Examples** on page III-148.

The following matrices are used in these examples:

```
Make/O/N=(3,3) r1=x, r2=y
```

Matrix addition and matrix multiplication by a scalar:

```
MatrixOp/O outWave = r1+r2-3*r1
```

Using the matrix Identity function:

```
MatrixOp/O outWave = Identity(3) x r1
```

Create a persisting identity matrix for another calculation:

```
MatrixOp/O id4 = Identity(4)
```

Using the Trace function:

```
MatrixOp/O outWave = (Trace(r1)*identity(3) x r1)-3*r1
```

Using matrix inverse function Inv() with matrix multiplication:

```
MatrixOp/O outWave = Inv(r2) x r2
```

Using the determinant function Det():

```
MatrixOp/O outWave = Det(r1)+Det(r2)
```

Using the Transpose postfix operator:

```
MatrixOp/O outWave = r1^t+(r2-r1)^t-r2^t
```

Using a mix of real and complex data:

```
Variable/C complexVar = cmplx(1,2)
MatrixOp/O outWave = complexVar*r2 - Cmplx(2,4)*r1
```

**Hermitian transpose operator:**

```
MatrixOp/O outWave = Trace(complexVar*r2)^h -Trace(cmplx(2,4)*r1)^h
```

**In-place operation and conversion to complex:**

```
MatrixOp/O r1 = r1*cmplx(1,2)
```

**Image filtering using 2D spatial filter filterWave:**

```
MatrixOp/O filteredImage=IFFT(FFT(srcImage,2)*filterWave,3)
```

**Positive shift:**

```
Make/O w={0,1,2,3,4,5,6}
MatrixOp/O w=shiftVector(w,2,77)
Print w
// w[0]= {77,77,0,1,2,3,4}
```

**Negative shift:**

```
Make/O w={0,1,2,3,4,5,6}
MatrixOp/O w=shiftVector(w,(-2),77)
Print w
// w[0]= {2,3,4,5,6,77,77}
```

```
// Using KronProd function to generate Hadamard matrices
Function HadamardMatrix(int N) // N must be >= 2
```

```
 Make/FREE/N=(2,2) H2={{1,1},{1,-1}}
 Duplicate/FREE H2, tmp
 int i
 for(i=2; i<N; i+=1)
 MatrixOP/O/FREE tmp=KronProd(H2,tmp)
 endfor
 Duplicate/O tmp, Hadamard
End
```

```
// bitReverseCol
Make/N=8 index = p
MatrixOP/O/P=1 out = bitReverseCol(index,0,0) // Direct binary reversal
// Prints: out = {0,4,2,6,1,5,3,7}
MatrixOP/O/P=1 out = bitReverseCol(index,0,1) // Hadamard order
// Prints: out = {0,4,6,2,3,7,5,1}
MatrixOP/O/P=1 out = bitReverseCol(index,0,2) // Dyadic/Paley/Gray order
// Prints: out = {0,1,3,2,6,7,5,4}
```

### MatrixOp Printing Examples

```
// Print a single-valued result
Make/O/N=(10,5) m2D=(p+1)*(q+1)
MatrixOP/O/P=1 aa=sum(m2D)
aa={825}
```

```
// Print a real 2D result
Make/O/N=(1,4) ddd=enoise(3)
MatrixOp/O/P=1 aa=ddd
aa={-2.273916482925415,-2.327789783477783,1.286988377571106,-0.8658701777458191}
```

```
// Print higher-dimension result layer-by-layer
Make/O/N=(3,4,3) ddd=z+1
MatrixOP/O/P=1 aa=mean(ddd)
aa={1} // Each layer results in a 1x1 value
aa={2}
aa={3}
```

### References

syncCorrelation and asyncCorrelation:

Noda, I., Determination of Two-Dimensional Correlation Spectra Using the Hilbert Transform, *Applied Spectroscopy* 54, 994-999, 2000.

ChirpZ:

Rabiner, L.R., and B. Gold, *The Theory and Application of Digital Signal Processing*, Prentice Hall, Englewood Cliffs, NJ, 1975.