CONTENTS

ビジュアルヘルプ – MatrixOp

MatrixOp コマンドのヘルプ

※ MatrixOp は、バージョンごとにさまざまな機能が追加されているコマンドです（特にバージョン７以降）。古いバージョンではできなかったことができるようになっていますので、過去のバージョンを持っている方は、本ドキュメントを確認することを推奨します。

MatrixOp を使って、これまでのコードを書き替えることができます。
これにより、コードを短くしたり、ウェーブの複製によるメモリの消費を抑えることができます。

簡単な例：

```
Function fn(Wave inWave)
        Duplicate/O inWave,outWave
        Redimension/D outWave
        FFT /DEST=fOutWave outWave
        Duplicate/O fOutWave,absOutWave
        absOutWave=abs(fOutWave)
        Variable ss=sum(absOutWave)
        return ss
end
```

という関数があるとします。
ウェーブを上書きしないように複製があります。
これを、まずは単純に MatrixOp を使って書き換えると次のようになります。

```
Function fn(Wave inWave)
        MatrixOp/O outWave = fp64(inWave)
        MatrixOp/O fOutWave = fft(outWave,1)
        MatrixOp/O absOutWave = abs(fOutWave)
        MatrixOp/O ss = sum(absOutWave)
        return ss[0]
end
```

各代入の結果を１つの MatrixOp にまとめます。

```
Function fn(Wave inWave)
        MatrixOP/O ss=sum(abs(fft(fp64(inWave),1)))
        return ss[0]
end
```

このようにすると、処理の途中でウェーブが大量に生成されることなく、結果を得ることができます。

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

MatrixOp コマンドは *expression* を評価し、結果を *destWave* に格納します。

expression には、リテラル数値、数値変数、数値ウェーブ、および以下に説明する演算子と関数のセットを含めることができます。
MatrixOp はテキストウェーブ、文字列、または構造体をサポートしません。

MatrixOp は、標準的な Igor ウェーブフォーム代入や行列演算よりも高速で、場合によっては可読性にも優れています。

MatrixOp の概要については、ヘルプ Using MatrixOp（Analysis.ihf）を参照してください。

パラメーター

destWave

代入式の対象となるウェーブを指定します。*destWave* は実行時に作成されます。既に存在する場合、/O フラグを使って上書きする必要があります。そうしないと、コマンドはエラーを返します。

コマンドが完了すると、*destWave* は *expression* によって示される次元とデータ型を持ちます。特に、*expression* が複素量に評価される場合、*destWave* は複素数ウェーブである可能性があります。*expression* がスカラーに評価される場合、*destWave* は 1x1 のウェーブです。

/FREE フラグを含めると、*destWave* はフリーウェーブとして作成されます。

デフォルトでは、*destWave* のデータ型は、オペランドのデータ型および代入の右辺における演算の性質に依存します。*expression* が整数ウェーブのみを参照する場合、*destWave* も整数ウェーブとなる可能性がありますが、スカラーを伴うほとんどの演算では、*destWave* は倍精度ウェーブに変換されます。詳細については、ヘルプ MatrixOp Data Promotion Policy を参照してください。

destWave がコマンド実行前に存在していても、MatrixOp は *expression* が示す通りそのデータ型と次元を変更する可能性があります。

MatrixOp 関数（uint16 や fp32 など）を使って数値型を強制できます。

expression

expression とは、ウェーブ、ローカル変数、グローバル変数、リテラル数値を、以下のセクションで列挙する行列演算演算子および行列演算関数と共に参照する数学的表現です。

オペランドには任意のデータ型の組み合わせを使用できます。特に、*expression* 内で実数型と複素数型を混在させることが可能です。MatrixOp は、Wave/C などの型宣言に関係なく、実行時に入力のデータ型と適切な出力データ型を決定します。

演算子

\+ スカラー間の加算、行列の加算、または行列の各要素へのスカラー（実数または複素数）の加算します。

\- あるスカラーから別のスカラーを減算、行列の引き算、または行列の各要素からスカラーを減算します。スカラーから行列を引くことは定義されていません。

* ２つのスカラーの乗算、行列とスカラーの乗算、または同じ次元の２つのウェーブの要素ごとに乗算します。

/ ２つのスカラーの除算、行列のスカラーの除算、または同じ次元の２つのウェーブの要素ごとに除算します。

スカラーの行列による除算はサポートされていませんが、代わりに乗算を用いた rec 関数を使用できます。

x 行列を乗算します（小文字の x のみ）。

この演算子の前後にはスペースを挿入する必要があります。行列の乗算では、左側の行列の列数が右側の行列の行数と等しい必要があります。

. 内積の一般化形式です。式 a.b では、a と b は任意の数値型であっても、同じ要素数を持つことが前提となります。この演算子は、a と b の両方が 1 次元配列であるかのように、連続する要素同士の積の和を返します。

複素数ウェーブ a と b に対して、この演算子は MatrixOp における sum(a*conj(b)) に相当する結果を返します。MatrixDot 関数は sum(b*conj(a)) を返します。

^t 行列の転置です。これは後置演算子で、^t は行列ウェーブの名前の後に記述されることを意味します。

^h エルミート転置です。これは後置演算子であり、行列ウェーブの名前の後に ^h が現れることを意味します。

&& 論理 AND 演算子はすべての実数データ型をサポートし、値が 0 または 1 の符号付きバイト数値トークンを結果として生成します。この演算は要素単位で作用し、オペランドウェーブの各要素に対して実行されます。

|| 論理 OR 演算子はすべての実数データ型をサポートし、値が 0 または 1 の符号付きバイト数値トークンを結果として生成します。この演算は要素単位で作用し、オペランドウェーブの各要素に対して実行されます。

MatrixOp は、+= のような演算子の組み合わせをサポートしていません。

この表は MatrixOp 演算子の優先順位を示しています。

MatrixOp 演算子		優先順位
^h	^t	最高
x	.	
*	/	
+	-	
&&	||	最低

括弧を使って評価順序を強制できます。

演算子の優先順位が同じ場合、演算子の結合性は右から左になります。
これは、a*b/c が a*(b/c) と同等であることを意味します。

関数一覧

これらの関数が MatrixOp とともに使用できます。

数と算術

e	enoise	nf	i	nan	maxAB
minAB	maxMagAB	minMagAB	mod		

三角関数

acos	asin	atan	atan2	cos	hypot
phase	sin	sqrt	tan		

指数関数

acosh	asinh	asinh	cosh	exp	expIntegralE1
expm	ln	log	powC	powR	
sinh					

複素数

cmplx	conj	imag	magSqr	p2Rect	phase
powC	r2Polar	eal			

丸めと切り捨て

abs	ceil	clip	floor	limit	mag
round					

形式変換

cmplx					
fp32	fp64				
int8	int16	int32	int32	int32	uint16
uint32					

データのプロパティ

numCols	umPoints	numRows	umType
waveChunks	waveLayers	wavePoints	
waveX	aveY	aveZ	aveT

データ特性評価

averageCols	binMean	inVar	rossCovar	chol	det
frobenius	integrate	indexCols	indexRows	intMatrix	
maxCols	axRows	axVal			
mean	inCols	inRows	inVal		
normP	neNorm				
productCol	productCols	productDiagonal	roductRows		
sgn					
sum	umBeams	umCols	umND	umRows	umSqr
trace	arBeams	arCols			

データの作成と抽出

beam	atCols	atRows	ol	colRepeat	rowRepeat
chunk	const	decimateMinMax	etDiag	dentity	nsertMat
IndexMatch	nv	ayer	layerStack	rec	removeCol
removeCols	setType	subRange	subWaveC	subWaveR	
tridiag	waveIndexSet	waveMap	zeroMat		

データの変換

addCols	addRows	bitReverseCol	diagonal	diagRC
median	normalize	normalizeCols	normalizeRows	
redimension	replace	replaceNaNs	replaceInfs	
reverseCol	reverseCols	reverseRow	reverseRows	
rotateChunks	rotateCols	rotateLayers	rotateRows	rowDiff
scale	scaleCols	spliceCols	scaleChunks	scaleLayers
setCol	setColsRange	setNaNs	setOffDiag	setRow
shiftVector	subtractCols	subtractMean	subtractMin	subtractRows
transposeVol	zapINFs	zapNaNs	zapZeros	

時間ドメイン

asyncCorrelation	convolve	correlate	limitProduct	syncCorrelation

周波数ドメイン

chirpZ	chirpZf	fft	ifft
FST	FCT	FSST	FSCT
FSST2	FSCT2		

行列

backwardSub	chol	covariance	det	diagonal
diagRC	forwardSub	frobenius	getDiag	identity
inv	kronProd	outerProduct	setOffDiag	tensorProduct
trace				

特殊関数

erf	erfc	gamma	gammaln	inverseErf	inverseErfc

論理式

equal	greater	within

ビット単位

bitAnd	bitOr	bitShift	bitXOR	bitNot

クオータニオン（四元数）

quat	axisToQuat	quatToAxis	matrixToQuat	quatInverse
quatFromSpherical	quatToMatrix	slerp		

関数の詳細

abs(*w*)	**実数の絶対値、または複素数の大きさを返します。**
acos(*w*)	***w* のアークコサインを返します。**

acosh(*w*) ***w* の双曲アークコサインを返します。**

Igor Pro 7.0 で追加されました。

addCols(*w,dc*) **1D ウェーブ *dc* の要素を *w* の対応する列に加算した行列を返します。**

```
out = w + dc[q]
```

Igor Pro 9.0 で追加されました。

addRows(*w,dr*) **1D ウェーブ *dr* の要素を *w* の対応する行に加算した行列を返します。**

```
out = w + dr[p]
```

Igor Pro 9.0 で追加されました。

asin(*w*) ***w* のアークサインを返します。**

asinh(*w*) ***w* の双曲アークサインを返します。**

Igor Pro 7.0 で追加されました。

asyncCorrelation(*w*) **実数値入力行列ウェーブ *w* に対する非同期スペクトル相関行列を返します。**詳細は syncCorrelation のセクションを参照してください。

atan(*w*) ***w* のアークタンジェントを返します。**

atan2(*y,x*) **実数 *y/x* のアークタンジェントを返します。**

atanh(*w*) ***w* の双曲アークタンジェントを返します。**

Igor Pro 7.0 で追加されました。

averageCols(*w*) **行列 *w* の列の平均値を含む (1xcolumns) ウェーブを返します。**これは sumCols(w)/numRows(w) と同等です。

Igor Pro 7.0 で追加されました。

axisToQuat(*ax*) 後半の**クォータニオンを使った関数**のセクションを参照してください。

backwardSub(*U,c*) **行列方程式 *U*x=*c* の列ベクトル解を返します。**ここで *U* は上三角行列を表す (N×N) ウェーブであり、*c* は N 行の列ベクトルです。c に余分な列がある場合、それらは無視されます。

理想的には、*U* と *c* は SP（単精度）ウェーブまたは DP（倍精度）ウェーブ（実数または複素数ウェーブ）であるべきです。他の数値データ型もサポートされていますが、わずかなパフォーマンスの低下を伴います。

この関数は通常、行列を下三角行列と上三角行列に分解する（例：コレスキー分解）後の連立一次方程式の解法に使われ、次のような形で表現される。

```
MatrixOp/O solVector=backwardSub(U,forwardSub(L,b))
```

ここで、U と L は上三角因子と下三角因子です。

Igor Pro 7.0 で追加されました。

beam(*w,row,col*) ***w* が 3D ウェーブの場合、ビーム関数は *w*[*row*][*col*][] で定義されるウェーブ内のデータに対応する 1D 配列を返します。**つまり、指定された行と列の全要素を全層から抽出した 1D 配列を返します。
ImageTransform の getBeam も参照してください。

***w* が 4D ウェーブである場合、*w*[*row*][*col*][][] を含む 2D 配列を返します。**つまり、全レイヤーおよび全チャンクから指定された行と列の全要素で構成される行列を返します。

ビーム関数は、レイヤーごとに動作しないという点で特別なクラスに属します。したがって、そのパラメーターのいずれかの代わりに複合式を使うことはできません。

ビーム関数は最優先権を持ちます。

binMean(*w,binWave*)

指定された *binWave* によって指定されたデータのビンの平均値を含む倍精度行列を返します。*binWave* は、インデックスのペアを含む 1D のウェーブです。ペアの最初の要素はビンの開始行インデックスであり、ペアの 2 番目の要素はビンの終了行インデックスです。ビンの重複は許可されますが、各ペアの最初のインデックスは 2 番目のインデックス以下でなければなりません。*binWave* 内のペアの順序は制限されません。*binWave* は実数でなければならず、NaN や INF を含んではいけません。*w* の各列に対してビンの平均値が計算されます。したがって、*w* が (m x n) 行列で *binWave* が 2k ポイントから構成される場合、この関数は平均値の (k x n) 行列を返します。

binVar(*w,binWave*)

上記の binMean() と同様ですが、平均値ではなくビンの分散を返します。

bitAnd(*w1,w2*)

***w1* と同じ次元と数値型を持つ配列を返します。各要素は *w1* と *w2* のビット単位の AND 演算に対応します。**

w2 は 1 つの数値または *w1* と同じ次元の行列のいずれかです。*w1* と *w2* は両方とも実数整数型でなければなりません。

Igor Pro 7.0 で追加されました。

bitOr(*w1,w2*)

***w1* と同じ次元と数値型を持つ配列を返します。各要素は *w1* と *w2* のビット単位の OR 演算に対応します**。

w2 は 1 つの数値、または *w1* と同じ次元の行列のいずれかです。*w1* と *w2* は両方とも実数整数型でなければなりません。

Igor Pro 7.0 で追加されました。

bitReverseCol (*w,c,order*)

列 *c* の要素を並べ替えた行列 *w* を返します。

order は次のいずれかの値です：

0： インデックスの直接バイナリ反転
1： アダマール（Hadamard）順序
2： ダイアディック／ペイリー／グレイ順序
3： 変更のない順序

bitReverseCol は、FFT や高速ウォルシュ・アダマール変換などの高速変換アルゴリズムにおけるエントリの順序変更に使用できます。w の行数は 2 のべき（冪）でなければなりません。

bitReverseCol を使った例については、本ヘルプ後半の例を参照してください。

Igor Pro 9.0 で追加されました。

bitShift(*w1*,*w2*)

***w1* と同じ次元と数値型を持つ配列を、*w2* で指定された量だけシフトして返します。**

w2 は１つの数値、または *w1* と同じ次元の行列のいずれかです。*w1* と *w2* は両方とも実数整数型でなければなりません。

w2 が正の値の場合、シフトは左方向に行われます。*w2* が負の値の場合、シフトは右方向に行われます。

Igor Pro 7.0 で追加されました。

bitXor(*w1*,*w2*)

***w1* と同じ次元と数値型を持つ配列を返します。各要素は *w1* と *w2* のビット単位の XOR 演算に対応します。**

w2 は１つの数値または *w1* と同じ次元の行列のいずれかです。*w1* と *w2* は両方とも実数整数型でなければなりません。

Igor Pro 7.0 で追加されました。

bitNot(*w*)

***w* と同じ次元と数値型を持つ配列を返します。各要素は、対応する *w* の要素のビット単位の補数です。**

Igor Pro 7.0 で追加されました。

catCols(*w1*,*w2*)

w2* の列を *w1* の列に連結します。w1* と *w2* は行数が同じで、データ型が同じである必要があります。

Igor Pro 7.0 で追加されました。

catRows(*w1*,*w2*)

w2* の行を *w1* の行に連結します。w1* と *w2* は列数が同じで、データ型が同じである必要があります。

Igor Pro 7.0 で追加されました。

cbrt(*w*)

w* の要素の立方根を返します。w* が複素数の場合、cbrt は主立方根（虚部が正の根）を返します。

Igor Pro 8.0 で追加されました。

ceil(*z*)

***z* より大きい最小の整数を返します。**

chirpZ(*data*,*A*,*W*,*M*)

以下の定義によるコンターに対して計算された 1D ウェーブデータの Chirp Z 変換を返します。

$$z_k = AW^{-k}$$

$$k = 0,1,...M-1$$

ここで *A* と *W* はともに複素数で、列 {x(n)} に対する標準的な Z 変換は次のように定義されます。

$$X(z) = \sum_{k=0}^{N-1} x(n) z^{-k}$$

出力の位相は、単位円上の ChirpZ 変換の結果と FFT の結果を一致させるために反転されることに注意してください。

chirpZf(*data*,*f1*,*f2*,*df*)

Chirp Z 変換を返します。ただし、パラメーターは実数値の開始周波数 *f1*、終了周波数 *f2*、および周波数分解能 *df* で指定されます。この場合、*A* と *W* の両方が単位振幅を持つため、変換は単位円に制限されます。

chol(*w*)

正定値対称行列 *w* に対して、w=U^t x U を満たす Cholesky（コレスキー）分解 U を返します。計算では *w* の上三角部分のみが実際に使われることに注意してください。

chunk(*w,n*)

4D ウェーブ *w* からチャンク *n* を返します。

Igor Pro 7.0 で追加されました。

clip(*w,low,high*)

ウェーブ *w* の値を、*low* パラメーターと *high* パラメーターで指定された範囲にクリッピングして返します。*w* に NaN または INF 値が含まれる場合、それらは変更されません。結果は、*low* と *high* 入力パラメーターの範囲に関係なく、入力ウェーブ *w* と同じ数値型を保持します。

cmplx(*re,im*)

２つの実数トークンから複素数トークンを返します。*re* と *im* は同じ次元数を持つ必要があります。

Igor Pro 7.0 で追加されました。

col(*w,c*)

行列ウェーブ *w* から列 *c* を返します。

colRepeat(*w,n*)

ウェーブ *w* のデータを *n* 個の同一列で構成する行列を返します。*w* が 2D ウェーブの場合、全データを単一列として扱います。高次元データは層単位でサポートされます。*n*<2 の場合、MatrixOp はエラーを返します。

Igor Pro 7.0 で追加されました。

conj(*matrixWave*)

入力式の複素共役を返します。

const(*r,c,val*)

(*r* x *c*) 行列を返します。すべての要素は *val* と等しくなります。返される行列のデータ型は *val* と同じです。zeroMat も参照してください。

Igor Pro 7.0 で追加されました。

convolve(*w1,w2,opt*)

***w1* と *w2* の畳み込みをオプション *opt* に従って返します。**結果の次元は *w1* と *w2* の最大次元によって決定され、行数は（必要に応じて）偶数になるようパディングされます。サポートされるオプションには、循環畳み込み用の *opt*=0 と非因果畳み込み用の *opt*=4 が含まれます。

高速 2D 畳み込みにおいて、*w1* が画像、*w2* が同じ数値型の正方カーネルである場合、*opt*=-1 または *opt*=-2 を使用できます。*opt*=-1 の場合、境界での畳み込みはゼロパディングを用いて評価されます。*opt*=-2 の場合、パディングは *w1* を境界で反射した形となります。整数ウェーブを扱う場合、カーネルは内部でその要素の和によって正規化されます。浮動小数点ウェーブ用のカーネルは変更されません。

w1 の画像を *w2* のより小さなポイント拡がり関数で畳み込むには、以下を使用できます。

opt=-1： 画像の境界をゼロで埋める場合
opt=-2： 境界値を反転させることで境界をパディングしたい場合

負数のオプションは、*w1* と *w2* が同じ数値型であることが要求される非常に最適化された畳み込み計算のために設計されています。ポイント拡がり関数のサイズが約 13×13 より大きい場合、正数のオプションを使って畳み込みを計算する方が効率的になる可能性があります。

correlate(*w1,w2,opt*)

オプション *opt* を指定して *w1* と *w2* の相関を返します。結果の次元は *w1* と *w2* の最大次元によって決定され、行数は（必要に応じて）偶数に

なるようパディングされます。サポートされるオプションには、円相関用の *opt*=0 と非因果相関用の *opt*=4 が含まれます。

cos(*w*) **_w_ のコサインを返します。**

cosh(*w*) **_w_ の双曲コサインを返します。**

Igor Pro 7.0 で追加されました。

covariance(*w*) **サンプル共分散行列を返します。**

w が行数 x 列数の実数行列である場合、返される値は列数 x 列数の行列 Q で、その要素は

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

ここで

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

です。

crossCovar (*w1, w2, opt*) **1D ウェーブ _w1_ と _w2_ の交差共分散を返します。**オプションパラメーター *opt* は、生の交差共分散を求める場合は 0 に、結果をゼロオフセットで 1 に正規化したい場合は 1 に設定できます。*w1* の行数が N、*w2* の行数が M の場合、返されるベクトルの長さは N+M-1 となります。交差共分散は、各入力の平均値を差し引き、相関計算、オプションの正規化を実行して算出されます。/NODC フラグ付き Correlate も参照してください。

decimateMinMax(*w,N*) **行列 _w_ を列単位で分割します。**各列は *N* 個の区間に分割され、各区間は最小値と最大値で表されます。

RxC 実数値入力 *w* に対する出力は 2*N*×C 次元の配列で、極値は {min0, max0, min1, max1...} の順序で並べられます。

丸め誤差のため、R が *N* の整数倍でない場合、連続するビンが同じ要素数を表さない可能性があります。この場合、ビンのサイズの丸め誤差の正確な形式を期待すべきではありません。

Igor Pro 9.0 で追加されました。

det(*w*) **行列 _w_ の行列式に対応するスカラーを返します。***w* は実数でなければなりません。

diagonal(*w*) **_w_ と同じ行数を持つ正方行列を作成します。**対角要素は *w* の最初の列から取得され、それ以外の要素はすべて 0 です。入力が既存のウェーブではない場合（他の関数の結果など）、DiagRC を使ってください。

diagRC(*w,rows,cols*) ***rows* × *cols* の 2D 行列を返します。**対角線上の要素は *w* の要素から順次埋められ、それ以外の行列要素はすべて 0 に設定されます。*w* の次元は重要ではありません。*w* の要素総数が対角線上の要素数より少ない場合、すべての要素が使われ、残りの対角線上の要素は 0 に設定されます。

e **自然対数の底を返します。**

enoise(*w*) **_w_ と同じ次元のトークンを返します。各要素は enoise(w[i][j]) が返す値に設定されます。**

equal(*w1,w2*)

等価の場合は 1、そうでない場合は 0 を返す符号なしバイト結果を返します。結果の次元は最大のパラメーターの次元と一致します。

w1 と *w2* のいずれか一方または両方が定数（すなわち、1行1列）である場合があります。

w1 と *w2* が定数ではない場合、それらは同じ行数と列数を持たなければなりません。*w1* または *w2* が複数のレイヤーを持つ場合、それらは同じレイヤー数を持つか、あるいは一方だけが単一レイヤーでなければなりません。

両方のパラメーターは実数または複素数のいずれかです。実数パラメーターと複素数パラメーターの比較は 0 を返します。

erf(*w*)

***w* の実数値に対する誤差関数（erf コマンドのヘルプを参照）を返します。**

erfc(*w*)

***w* の実数値に対する補誤差関数（erfc コマンドのヘルプを参照）を返します。**

exp(*w*)

w* に対する指数関数を返します。w* は実数または複素数、スカラーまたは行列であることができます。

expIntegralE1(*w*)

実数引数 *w* に対する指数積分を返します。ExpIntegralE1 コマンドのヘルプも参照してください。

Igor Pro 9.0 で追加されました。

expm(*w*)

実数値の正方行列 *w* の行列指数関数を返します。

Igor Pro 9.0 で追加されました。

FCT(*w,dir*)

後半の**三角関数変換関数**のセクションを参照してください。

fft(*w,options*)

***w* の FFT を返します。**

w は行数が偶数でなければなりません。

options にはフラグ用のバイナリフィールドが含まれます。ゼロ中心化を無効化したい場合（FFT コマンドの /Z フラグを参照）、ビット 1 を 1 に設定してください。その他のビットは予約済みです。

MatrixOp はウェーブスケーリングをサポートしていないため、FFT 演算と同じウェーブスケーリング変化を生成しません。

floor(*w*)

w* より小さい最大の整数を返します。w* が複素数の場合、この関数は実部と虚部に別々に適用されます。

forwardSub(*L,b*)

行列方程式 *Lx=b* の列ベクトル解を返します。ここで *L* は下三角行列を表す (N×N) ウェーブで、*b* は N 行の列ベクトルです。*b* に余分な列がある場合は無視されます。

理想的には、*L* と *b* は SP（単精度）ウェーブまたは DP（倍精度）ウェーブ（実数または複素数ウェーブ）であるべきです。他の数値データ型もサポートされていますが、わずかなパフォーマンスの低下を伴います。

この関数は通常、行列を下三角行列と上三角行列に分解した（例：コレスキー分解）後の連立一次方程式の解法に使われ、次のような形で表現されます：

```
MatrixOp/O solVector=backwardSub(U,forwardSub(L,b))
```

ここで、U と L は上三角因子および下三角因子です。

Igor Pro 7.0 で追加されました。

fp32(*w*) ***w* を 32 ビット単精度浮動小数点表現に変換します。**詳細は後半の /NPRM フラグも参照してください。

Igor Pro 7.0 で追加されました。

fp64(*w*) ***w* を 64 ビット単精度浮動小数点表現に変換します。**詳細は後半の /NPRM フラグも参照してください。

Igor Pro 7.0 で追加されました。

Frobenius(*w*) **行列のフロベニウス（Frobenius）ノルムを返します。**これは、すべての要素の絶対値の二乗の和の平方根として定義されます。

FST(*w,dir*) 後半の**三角関数変換関数**のセクションを参照してください。

FSST(*w,dir*) 後半の**三角関数変換関数**のセクションを参照してください。

FSCT(*w,dir*) 後半の**三角関数変換関数**のセクションを参照してください。

FSST2(*w,dir*) 後半の**三角関数変換関数**のセクションを参照してください。

FSCT2(*w,dir*) 後半の**三角関数変換関数**のセクションを参照してください。

gamma(*w*) **実数引数 *w* に対するガンマ関数を返します。**gamma コマンドのヘルプも参照してください。

Igor Pro 9.0 で追加されました。

gammaln(*w*) **実数引数 *w* に対するガンマ関数の自然対数を返します。**gammln コマンドのヘルプも参照してください。

Igor Pro 9.0 で追加されました。

getDiag(*w2d,d*) ***w2d* の対角線 *d* を含む 1D ウェーブを返します。***d*=0 は主対角線、*d*>0 は上対角線、*d*<0 は下対角線に対応します。

Igor Pro 7.0 で追加されました。

greater(*a,b*) ***a* > *b* の真偽を符号なしバイトで返します。***a* と *b* は両方とも実数である必要がありますが、一方または両方が定数でも構いません（上記の equal() を参照）。結果の次元は、最大のパラメーターの次元と一致します。

greaterOrEqual(*a,b*) ***a* >= *b* の真偽を符号なしバイトで返します。***a* と *b* は両方とも実数である必要がありますが、一方または両方が定数でも構いません（上記の equal() を参照）。結果の次元は、最大のパラメーターの次元と一致します。

Igor Pro 9.0 で追加されました。

hypot(*w1,w2*) ***w1* と *w2* の二乗和の平方根を返します。**

Igor Pro 7.0 で追加されました。

ifft(*w,options*) ***w* の逆フーリエ変換（IFFT）を返します。**

options はビット単位のパラメーターで、以下のように定義されます：

Bit 0： 結果を実数に強制する。IFFT コマンドの /C フラグと同じ
Bit 1： 中心ゼロを無効にする
Bit 2： 結果をスワップする

ビット設定の詳細については、ヘルプ Setting Bit Parameters を参照してください。

MatrixOp はウェーブスケーリングをサポートしていないため、IFFT コマンドと同じウェーブスケーリング変化を生成しません。

identity(*n,m*)
identity(*n*)

単位行列である計算オブジェクトを作成します。引数 *n* を単独で使った場合、作成される単位行列は (*n* × *n*) の正方行列で、対角要素が 1（残りの要素は 0 に設定）となります。両方の引数を使った場合、関数は (*n* × *m*) のゼロ行列を作成し、その対角要素を 1 で埋めます。
注意：単位行列は実行時に生成され、特定の操作の目的でのみ存続します。

imag(*w*)

***w* の虚部を返します。**

indexCols(*w*)

***w* と同じ次元のトークンを返します。各要素はその列インデックスに等しくなります。**これは、標準的なウェーブ代入文の右辺で使われる q に相当する MatrixOp です。w の列数が 2^{32} 未満の場合、返されるトークンは符号なし 32 ビット整数です。それ以外の場合は倍精度浮動小数点数です。

Igor Pro 8.0 で追加されました。

IndexMatch (*w,val* [,*tol*])

指定された行列 *w* の行と列を指定する 2 列のウェーブを返します。この行と列では、オプションの誤差許容値 *tol* を条件として *val* との一致が見つかります。入力 w は 1D または 2D です。それ以上の次元はサポートされていません。

indexRows(*w*)

***w* と同じ次元のトークンを返します。各要素はその行インデックスに等しくなります。**これは、標準のウェーブ代入文の右辺で使われる p に相当する MatrixOp です。w の行数が 2^{32} 未満の場合、返されるトークンは無符号 32 ビット整数です。それ以外の場合は倍精度浮動小数点数です。

Igor Pro 8.0 で追加されました。

inf()

INF を返します。

Igor Pro 7.0 で追加されました。

insertMat(*s,d,r,c*)

行列 *s* を、行 *r* と列 *c* から開始して行列 *d* に挿入します。ウェーブ *s* と *d* は同じ数値データ型である必要があります。挿入範囲はウェーブ *d* の次元に合わせて切り詰められます。*r* と *c* は両方とも非負である必要があります。

Igor Pro 7.0 で追加されました。

int8(*w*)

***w* を 8 ビット符号付き整数表現に変換します。**詳細は後半の /NPRM フラグも参照してください。

Igor Pro 7.0 で追加されました。

int16(*w*)

***w* を 16 ビット符号付き整数表現に変換します。**詳細は後半の /NPRM フラグも参照してください。

Igor Pro 7.0 で追加されました。

int32(*w*)

***w* を 32 ビット符号付き整数表現に変換します。**詳細は後半の /NPRM フラグも参照してください。

Igor Pro 7.0 で追加されました。

integrate(*w,opt*)

***w* の累積和を返します。**

opt=0 の場合、和はウェーブ全体に対して実行され、2D ウェーブの列をあたかも 1 つの大きな列の一部であるかのように扱います。
opt=1 の場合、2D ウェーブの各列ごとに累積和が個別に計算されます。

Igor Pro 7.0 で追加されました。

intMatrix(*w*)

***w* と同じ次元の倍精度行列を返します。**

返される行列の各要素は、その要素の左上にある *w* の対応する要素すべての和です。つまり、

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

この関数の有用性は、次の関係式から明らかです：

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

Igor Pro 7.0 で追加されました。

inv(*w*)

正方行列 *w* の逆行列を返します。

w が可逆でない場合、この演算は全要素が NaN に設定された同じ次元の行列を返します。

inverseErf(*w*)

***w* の実数値に対する逆誤差関数（inverseErf コマンドのヘルプを参照）を返します。**

Igor Pro 7.0 で追加されました。

inverseErfc(*w*)

***w* の実数値に対する逆補完誤差関数（inverseErfc コマンドのヘルプを参照）を返します。**

Igor Pro 7.0 で追加されました。

kronProd(*u,v*)

行列 *u* と *v* のクロネッカー（Kronecker）積を返します。

u の次元が M×N で *v* の次元が P×Q の場合、返されるブロック行列の次元は MP×NQ となります。これは次のように与えられます。

$$u \otimes v = \begin{bmatrix} u_{11}v_{11} & u_{11}v_{112} & \cdots & u_{11}v_{1q} & \cdots & \cdots & u_{1n}v_{11} & u_{1n}v_{12} & \cdots & u_{1n}v_{1q} \\ u_{11}v_{21} & u_{11}v_{22} & \cdots & u_{11}v_{2q} & \cdots & \cdots & u_{1n}v_{21} & u_{1n}v_{22} & \cdots & u_{1n}v_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ u_{11}v_{p1} & u_{11}v_{p2} & \cdots & u_{11}v_{pq} & \cdots & \cdots & u_{1n}v_{p1} & u_{1n}v_{p2} & \cdots & u_{1n}v_{pq} \\ \vdots & \vdots & & \vdots & & & \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots & & & \vdots & \vdots & & \vdots \\ u_{m1}v_{11} & u_{m1}v_{12} & \cdots & u_{m1}v_{1q} & \cdots & \cdots & u_{mn}v_{11} & u_{mn}v_{12} & \cdots & u_{mn}v_{1q} \\ u_{m1}v_{21} & u_{m1}v_{22} & \cdots & u_{m1}v_{2q} & \cdots & \cdots & u_{mn}v_{21} & u_{mn}v_{22} & \cdots & u_{mn}v_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & & \vdots \\ u_{m1}v_{p1} & u_{m1}v_{p2} & \cdots & u_{m1}v_{pq} & & & u_{mn}v_{p1} & u_{mn}v_{p2} & & u_{mn}v_{pq} \end{bmatrix}$$

kronProd を使った例に、後半の例を参照してください。

Igor Pro 9.0 で追加されました。

layer(*w,n*[,*chunk*])

3D または 4D ウェーブのレイヤーを返します。

chunk を省略し、*w* が 3D ウェーブの場合、layer 関数は *w* からレイヤー *n* を返します。
chunk を省略し、*w* が 4D ウェーブである場合、layer 関数は *w* から chunk 0 のレイヤー *n* を返します。

chunk を含める場合、*w* は 4D ウェーブでなければならず、layer 関数は *w* から指定された chunk の *n* 番目のレイヤーを返します。

いずれの場合においても、*w* は式ではなく実際のウェーブでなければなりません。

Igor Pro 7.0 で追加されました。
オプションの *chunk* パラメーターは Igor Pro 9.0 で追加されました。

layerStack(*w,n*)

4D 入力ウェーブ *w* とレイヤー番号 *n* を受け取り、*w* の各チャンクのレイヤー *n* のデータを順に含む 3D ウェーブ（スタック）を返します。*w*** はウェーブであり、派生トークンであってはなりません。

Igor Pro 9.0 で追加されました。

例：

```
// 4D RGB ウェーブの赤色チャンネルから 3D スタックを作成する
MatrixOP/O redChannelStack=layerStack(myRGBMovie,0)
```

limit(*w,low,high*)

ウェーブ *w* の値を、*low* パラメーターと *high* パラメーターの間でクリッピングして返します。clip() 関数とは異なり、limit() は +-INF の値を指定された [*low*,*high*] 範囲にクリップします。結果は、*low* と *high* 入力パラメーターの範囲に関係なく、入力ウェーブ *w* と同じ数値型を保持します。

limitProduct(*w1,w2*)

ウェーブ *w1* と *w2* の部分的な要素ごとの乗算を返します。

w1 の次元が *w2* の次元以上であると仮定します。*w2* の次元が NxM の場合、この関数は *w1* と同じ次元の行列を返します。この行列の最初の NxM 要素には *w1* と *w2* の対応する要素の積が格納され、残りの要素はゼロに設定されます。

この関数は、カーネル *w2* のサイズが入力 *w1* のサイズよりもはるかに小さいフィルタリング用途で使用されるように設計されています。

Igor Pro 7.0 で追加されました。

log(*w*) **トークン *w* の対数（底 10）を返します。***w* は実数または複素数、スカラーまたは行列です。

log2(*w*) **トークン *w* の 2 進対数を返します。***w* は実数または複素数、スカラーまたは行列です。

Igor Pro 9.0 で追加されました。

ln(*w*) **トークン *w* の自然対数を返します。***w* は実数または複素数、スカラーまたは行列です。

mag(*w*) ***w* の各要素の絶対値を含む実数値のウェーブを返します。**これは abs 関数と同等です。

magSqr(*w*) **実数 *w* の二乗、または複素数 *w* の絶対値の二乗を含む実数ウェーブを返します。**

maxAB(*a,b*) **実数 *a* と *b* のうち大きい方を返します。**

maxAB は NaN または複素数の入力に対応していません。

Igor Pro 7.0 で追加されました。

maxMagAB(*a,b*) **トークン *a* と *b* の各データポイントについて、maxMagAB は絶対値が大きい方を返します。**

maxMagAB は複素数をサポートしていません。

Igor Pro 9.0 で追加されました。

maxCols(*w*) **ウェーブ *w* の各列の最大値を含む 1D ウェーブを返します。***w* が複素数ウェーブの場合、出力は *w* の各列の最大絶対値となります。

maxCols は NaN をサポートしていません。

Igor Pro 7.0 で追加されました。

maxRows(*w*) **ウェーブ *w* の各行の最大値を含む 1D ウェーブを返します。***w* が複素数ウェーブの場合、出力には *w* の各行の最大絶対値が含まれます。

maxRows は NaN をサポートしていません。

Igor Pro 8.0 で追加されました。

maxVal(*w*) **ウェーブ *w* の最大値を返します。***w* が複素数の場合、*w* の最大絶対値を返します。

w が実数の場合、返される数値の型は *w* と同じです。
w が複素数の場合、結果は実数となり、w の各要素の絶対値の最大値を表します。*w* が倍精度の場合、結果は倍精度となります。それ以外の場合は単精度となります。

w が 3D または 4D ウェーブの場合、maxVal は (1 x 1 x layers x chunks) のデータトークンを返します。

maxVal は NaN をサポートしていません。

mean(*w*) **_w_ の平均値を返します。**

median(*w, r, c*) **_w_ の要素の _r_ 行 × _c_ 列の近傍について、中央値を含む行列を返します。** 中央値は Igor の median() 関数が返す値と同等です。中央値は NaN をスキップしますが、INF はスキップしません。入力が倍精度の場合、戻り値は倍精度型です。それ以外の場合は単精度型です。*r* と *c* の両方が正の場合に中央値が返されます。それ以外の場合は元の *w* を返します。境界沿いでのデータの反射や複製は行われません。代わりに、要素が対応する境界から *r* または *c* 未満の場合、利用可能な値のみを用いて中央値が計算されます。

minCols(*w*) **ウェーブ _w_ の各列の最小値を含む 1D ウェーブを返します。***w* が複素数ウェーブの場合、出力には *w* の各列の最小絶対値が含まれます。

minCols は NaN をサポートしていません。

Igor Pro 8.0 で追加されました。

minAB(*a,b*) **実数 _a_ と _b_ のうち小さい方を返します。**

minAB は複素数をサポートしていません。

Igor Pro 9.0 で追加されました。

minMagAB(*a,b*) **トークン _a_ と _b_ の各データポイントについて、絶対値が小さい方を返します。**

minMagAB は複素数をサポートしていません。

Igor Pro 9.0 で追加されました。

minRows(*w*) **ウェーブ _w_ の各行の最小値を含む 1D ウェーブを返します。***w* が複素数ウェーブの場合、出力には *w* の各行の最小絶対値が含まれます。

minRows は NaN をサポートしていません。

Igor Pro 8.0 で追加されました。

minVal(*w*) **ウェーブ _w_ の最小値を返します。***w* が複素数の場合、この関数は *w* の最小絶対値を返します。

w が実数の場合、返される数値の型は *w* と同じです。

w が複素数の場合、結果は実数となり、*w* の要素の絶対値の最小値を表します。*w* が倍精度の場合、結果は倍精度となります。それ以外の場合は単精度となります。

w が 3D または 4D ウェーブである場合、minVal は (1 × 1 × layers × chunks) のデータトークンを返します。

minVal は NaN をサポートしていません。

mod(*w,b*) **_w_ を _b_ で割った余りを返します。***b* はスカラーまたは *w* と同じ次元の行列です。

Igor Pro 7.0 で追加されました。

nan() **NaN を返します。**

normalize(*w*)

ベクトルまたは行列の正規化されたバージョンを返します。正規化は、返されるトークンの絶対値が 1 となるように行われます。ただし、すべての要素が 0 である場合は例外で、その場合は出力が変更されません。

normalizeCols(*w*)

実数ウェーブ *w* の各列を、その列の全要素の二乗和の平方根で割ります。

normalizeRows(*w*)

実数ウェーブ *w* の各行を、その行の全要素の二乗和の平方根で割ります。

normP(*w,pn*)

行列 *w* の p-ノルムを返します。行列 *w* は以下によって定義されます。

$$\left\|W\right\|_p = \left(\sum_{j=0}^{cols} \sum_{i=0}^{rows} \left| w[i][j] \right|^p \right)^{1/p}$$

pn=2 の場合、MatrixOp Frobenius（フロベニウス）関数に相当します。

Igor Pro 9.0 で追加されました。

not(*w*)

***w* と同じ次元の符号なしバイト配列を返します。**各要素は次の二進テストに対応します。

w[i][j]==0? 1:0

w は任意の実数型です。

Igor Pro 10.0 で追加されました。

numCols(*w*)

w* の列数を返します。w* が 1D の場合、関数は 1 を返します。

numPoints(*w*)

***w* のレイヤー内のポイント数を返します。**

numRows(*w*)

***w* の行数を返します。**

numType(*w*)

***w* の数値型を返します。**

0： *w* は通常の数字
1： *w* は +/-INF
2： *w* は NaN

oneNorm(*w*)

行列 *w* の 1-ノルムを返します。これは列の絶対和の最大値として定義されます。

$$\left\|W\right\| = \max_{0 \le j \le cols} \left(\sum_{i=0}^{rows} \left| w[i][j] \right| \right)$$

Igor Pro 9.0 で追加されました。

outerProduct(*w1,w2*)

M ポイントを含む 1D ウェーブ *w1* と N ポイントを含む 1D ウェーブ *w2* に対して、外積は M 行 N 列の行列を返します。その (i,j) 要素は次の通りです。

```
out[i][j] = w1[i] * conj(w2[j])
```

Igor Pro 8.0 で追加されました。

p2Rect(*w*)

***w* の各要素を極座標表現から直交座標表現に変換します。**

phase(*w*)

***w* の各要素の位相を phase=atan2(y,x) を使って計算した実数値のウェーブを返します。**

Pi

π を返します。

powC(*w1,w2*)	***w1* と *w2* が実数または複素数である場合、複素数値の $w1^{w2}$ を返します。**
powR(*x,y*)	**実数 *x* と *y* に対して x^y を返します。**
productCol(*w,c*)	**ウェーブ *w* の列 *c* の要素の積を含む（1 x 1）ウェーブ *a* を返します。**出力は倍精度実数または複素数です。 Igor Pro 7.0 で追加されました。
productCols(*w*)	**各要素が、対応する列の全要素の積である（1 x cols）のウェーブを返します。**出力は倍精度実数または複素数です。 Igor Pro 7.0 で追加されました。
productDiagonal(*w,d*)	**ウェーブ *w* の対角線上の要素の積を含む（1 x 1）ウェーブを返します。***d*=0 は主対角線、*d*>0 は上対角線、*d*<0 は下対角線に対応します。出力は倍精度実数または複素数です。 Igor Pro 7.0 で追加されました。
productRow(*w,r*)	**ウェーブ *w* の行 *r* の全要素の積を含む（1 x 1）ウェーブを返します。**出力は倍精度実数または複素数です。 Igor Pro 7.0 で追加されました。
productRows(*w*)	**対応する行の全要素の積を各要素とする（1 × 行）のウェーブを返します。**出力は倍精度実数または複素数です。 Igor Pro 7.0 で追加されました。
quat(*arg*)	後半の**三角関数変換関数**のセクションを参照してください。
quatFromSpherical (*theta, phi*)	後半の**三角関数変換関数**のセクションを参照してください。
quatInverse(*inQuat*)	後半の**三角関数変換関数**のセクションを参照してください。
quatToEuler (*qIn,mode*)	後半の**三角関数変換関数**のセクションを参照してください。
quatToAxis(*qIn*)	後半の**三角関数変換関数**のセクションを参照してください。
quatToMatrix(*qIn*)	後半の**三角関数変換関数**のセクションを参照してください。
r2Polar(*w*)	***w* の各要素に対して r2polar 関数と同等の処理を実行します。**つまり、各複素数 (x+iy) を極座標表現 r,theta に変換します（ x+iy=rexp(itheta) ）。
real(*w*)	***w* の実部を返します。**
rec(*w*)	***w* の各要素の逆数を返します。**

redimension(*w,nr,nc*)	***w* から（*nr* x *nc*）行列を返します。***w* のデータは連続的に（次元に関係なく列単位で）出力に配置されます。出力サイズが *w* より大きい場合、残りのデータポイントは 0 に設定されます。*w* が複数のレイヤーを含む場合、新しい次元はレイヤーごとに適用されます。例えば：
	
	`Make/N=(10,20,30) ddd = p + 10q + 100r`
	`MatrixOp/O aa = redimension(ddd,25,1)`

は、ウェーブ aa を (25,1,30) 次元で生成します。

removeCol(*w, c*) **行列 *w* から列 *c* を削除したコピーを返します。**

removeCols(*w, mask*) **マスクされた列を除去した行列 *w* のコピーを返します。**ここで *mask* は、*w* の列数以上をポイント数とする実数値ウェーブです。*mask* ウェーブの対応するエントリが 0 でない場合、行列 *w* の列が除去されます。

replace (*w,findVal, replacementVal*) **ウェーブ *w* 内の *findVal* の出現箇所をすべて *replacementVal* で置換します。**ウェーブ *w* の次元および数値型は保持されます。*replacementVal* は *w* と同じ数値型に変換されるため、切り捨てが生じる可能性があります。

replaceInfs (*w, replacementVal*) **すべての INF エントリが *replacementVal* で置換されたウェーブ *w* のコピーを返します。***w* が複素数の場合、実部または虚部のいずれかが INF であれば置換が行われます。*replacementVal* が実数で *w* が複素数の場合、置換されたエントリの虚数成分は 0 に設定されます。*w* が実数の場合に複素数の置換値を指定するとエラーとなります。

replaceNaNs (*w, replacementVal*) **ウェーブ *w* 内の NaN をすべて *replacementVal* で置換します。**ウェーブ *w* の次元は保持されます。*replacementVal* は *w* と同じ数値型に変換されるため、切り捨てが発生する可能性があります。

reverseCol(*w,c*) **列 *c* の順序を逆にした配列 *w* を返します。**

Igor Pro 7.0 で追加されました。

reverseCols(*w*) **すべての列を逆順にした配列 *w* を返します。**

Igor Pro 7.0 で追加されました。

reverseRow(*w,r*) **行 *r* を逆順にした配列 *w* を返します。**

Igor Pro 7.0 で追加されました。

reverseRows(*w*) **すべての行を逆順にした配列 *w* を返します。**

Igor Pro 7.0 で追加されました。

rotateChunks(*w,n*) ***w* の最後の *n* 個のチャンクを [0, *n*-1] のチャンクに移動した行列を返します。***n* が負の場合、最初の abs(*n*) 個のチャンクがデータの末尾に移動されます。*n* に NaN を渡すことはエラーです。

Igor Pro 7.0 で追加されました。

rotateCols(*w,nc*) **2D ウェーブ *w* の列を回転させ、最後の *nc* 列をデータの列 [0,*nc*-1] に移動します。***nc* が負の場合、最初の abs(*nc*) 列が列 [n-1-*nc* ,n-1] に移動されます。ここで n は列の総数です。*nc* に NaN を渡すとエラーになります。*nc* が列数より大きい場合、実際の回転は mod(*nc*,actualCols) となります。

rotateLayers(*w,n*) **最後の *n* 層を層 [0,*n*-1] に移動した行列を返します。***n* が負の場合、最初の abs(*n*) 層がデータの末尾に移動されます。*n* に NaN を渡すとエラーになります。

Igor Pro 7.0 で追加されました。

rotateRows(*w,nr*) **2D ウェーブ *w* の行を回転させ、最後の *nr* 行をデータの行 [0,*nr*-1] に移動します。***nr* が負の場合、最初の abs(*nr*) 行が [n-1-*nr*, n-1] の行に移動されます（n は総行数）。*nr* に NaN を渡すとエラーになります。

nr が行数より大きい場合、実際の回転は mod(*nr*, actualRows) となります。

round(*z*) **z を最も近い整数に丸めます。丸め方は「ゼロから離れる方向」です。**

row(*w,r*) **行列 *w* から行 *r* を返します。**返される行は (1xC) のウェーブで、C は *w* の列数です。1D ウェーブに変換するには、Redimension/N=(C) を使ってください。ImageTransform getRow のヘルプも参照してください。

rowDiff(*w*) **行列 *w* の次元が行数 × 列数の場合、この関数は次元 (行数 - 1) × 列数の行列 M を返します。**この行列のすべての要素は次の関係を満たします。

```
M[i][j] = w[i+1][j] - w[i][j]
```

Igor Pro 10.0 で追加されました。

rowRepeat(*w,n*) **ウェーブ *w* のデータを *n* 行の同一データで構成する行列を返します。***w* が 2D ウェーブの場合、全データを 1 つの列として扱います。高次元データはレイヤーごとに処理されます。*n*<2 の場合、MatrixOp はエラーを返します。

Igor Pro 7.0 で追加されました。

scale(*w,low,high*) **ウェーブ *w* の値を、*low* パラメーターと *high* パラメーターの間でスケーリングして返します。***w* に NaN または INF 値が含まれる場合、それらは変更されません。結果は、*low* と *high* 入力パラメーターの範囲に関係なく、*w* と同じ数値型を保持します。

scaleChunks (*w, scalingW*) ***w* を *scalingW* の連続する係数でスケーリングしたチャンクを含むウェーブを返します。**ここで *w* はウェーブトークン（複合式ではない）でなければなりません。結果のウェーブは *w* と同じ次元を持ちます。*w* が 3D ウェーブ（またはそれ以下の次元）の場合、*w* の全データは 1 つのチャンクとして *scalingW* の最初のエントリでスケーリングされます。

scaleCols(*w1,w2*) ***w1* と同じ次元の行列を返します。*w1* の各列は、1D ウェーブ *w2* の対応する行の値でスケーリングされます。***w2* の行数は *w1* の列数と等しくなければなりません。

Igor Pro 7.0 で追加されました。

scaleLayers (*w, scalingW*) ***w* を *scalingW* の連続する係数でスケーリングしたレイヤーを含むウェーブを返します。***w* が 2D または 1D ウェーブの場合、*w* の全データは単一のレイヤーとして *scalingW* の最初の要素でスケーリングされます。

scaleRows(*w1,w2*) ***w1* の各行を 1D ウェーブ *w2* の対応する行の値でスケーリングした、*w1* と同じ次元の行列を返します。***w2* の行数は *w1* の行数と等しくなければなりません。

Igor Pro 7.0 で追加されました。

select(*w,sr,sc*) **行列 *w* の要素のうち、行インデックスが (p%*sr*)=0 を満たし、かつ列インデックスが (q%*sc*)=0 を満たすものからなる行列を返します。**最初の行と列は常に選択されます。

sr と *sc* は実数、有限、かつ非負でなければなりません。次元内の全要素を含めるために、これらを 0 に設定できます。

Igor Pro 9.0 で追加されました。

setCol(*w2d,c,w1d*) ***w1d* の内容を列 *c* に格納した *w2d* のデータを返します。**

w1d の要素数は、*w2d* の行数以上でなければなりません。*w2d* と *w1d* は、両方とも実数か、両方とも複素数でなければなりません。

Igor Pro 7.0 で追加されました。

setColsRange (*w,low,high*) ***w* の各列を [*low*,*high*] の範囲にスケーリングした行列を返します。***w* は実数値でなければなりません。

Igor Pro 9.0 で追加されました。

setNaNs(*w,mask*) ***mask* ウェーブが 0 以外である箇所には NaN が格納されたウェーブ *w* のデータを返します。**ウェーブ *w* は任意の数値型で指定できます。

mask は *w* と同じ次元で実数でなければなりません。通常、別の式の出力結果となります。例えば、*w* が 5 より大きい場合に目的の値をすべて NaN に設定するには次のようにします。

```
MatrixOp/O out = setNaNs(w,greater(w,5))
```

Igor Pro 7.0 で追加されました。

setOffDiag(*w,d,w1*) ***w1* の内容を対角線 *d* に格納した *w* のデータを返します。**

d=0 は *w* の主対角線で、*d*>0 は上対角線、*d*<0 は下対角線に対応します。*w* と *w1* は両方とも実数でも、両方とも複素数でもかまいません。

Igor Pro 7.0 で追加されました。

setRow(*w2d,r,w1d*) ***w1d* の内容が行 *r* に格納された *w2d* のデータを返します。**

w1d の要素数は、*w2d* の列数以上でなければなりません。*w2d* と *w1d* は、両方とも実数か、両方とも複素数でなければなりません。

Igor Pro 7.0 で追加されました。

setType(*w, wType*) ***w* と同じ次元のウェーブを、*wType* で指定されたデータ型で返します。**サポートされているウェーブタイプは、2、4、8、16、32、72、80、96 です。これらは、fp32、fp64、int8、int16、int32、uint8、uInt16、uInt32 に対応します。これらの値は、1 で OR 演算を行うことで、対応する複素数タイプを取得することもできます。

sgn(*w*) ***w* の各要素の符号を返します。**負の数には -1 を返し、それ以外には 1 を返します。複素数は受け付けません。

shiftVector(*w,n,val*) **1D 行ベクトル *w* の要素を *n* 個分シフトし、移動した要素を *val* で埋めます。***val* は *w* のデータ型と一致する必要があり、複素数 *w* の場合は cmplx(a,b) の形で表現されるべきです。

sin(*z*) ***z* のサインを返します。**

sinh(*z*) ***z* の双曲サインを返します。**

Igor Pro 7.0 で追加されました。

slerp(*qIn1,qIn2,frac*) 後半の**クォータニオンを使った関数**のセクションを参照してください。

spliceCols(*w,c,d,ic*) **行列 *w* の *c* 列目から開始して、行列 *d* の *ic* 列を *w* に行を挿入した行列を返します。**行列 *d* は行列 *w* と同じ行数を持つ必要があり、両者は同じ数値型でなければなりません。*ic* は非負の整数でなければなりません。*ic*

が行列 *d* の列数を超える場合、関数は必要に応じて *d* の列を繰り返し挿入します。

Igor Pro 9.0 で追加されました。

sq(*w*)

行列 *w* の各要素の二乗を返します。複素数 z に対しては、返される値は z*z で、magsqr(z) ではありません。

Igor Pro 9.0 で追加されました。

sqrt(*z*)

z* の平方根を返します。z* が実数の場合、負の数に対しては NaN を返します。*z* が複素数の場合、*z* の複素平方根を返します。

subRange (*w,rs,re,cs,ce*)

ウェーブ *w* の連続した部分集合を、開始行 *rs* から終了行 *re* まで、開始列 *cs* から終了列 *ce* まで返します。これは Duplicate/R と似ていますが、次元スケーリングとラベルは保持されません。

subtractCols(*w, wc*)

***w* の各列から 1D ウェーブ *wc* を差し引いた *w* と同じ次元のウェーブを返します。**ウェーブ *wc* は、*w* の行数と同じ数のポイントを持つ必要があります。

subtractMean(*w,opt*)

ウェーブ *w* の平均値を計算し、ウェーブの値から平均値を差し引いた値を返します（*opt*=0）。各列の平均値を計算し、その列から平均値を差し引きます（*opt*=1）。各行の平均値を行値から差し引きます（*opt*=2）。

subtractMin(*w*)

各要素から *w* の最小値を差し引いた *w* と同じ次元のウェーブを返します。この関数は複素数値の *w* をサポートしていません。

subtractRows(*w, wr*)

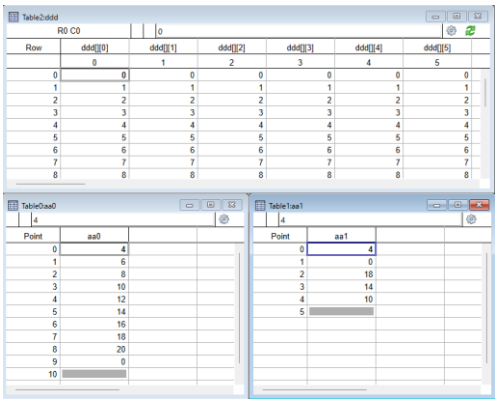
***w* の各行から 1D ウェーブ *wr* を差し引いた *w* と同じ次元のウェーブを返します。**ウェーブ *wr* は、*w* の列数と同じ数のポイントを持つ必要があります。

subWaveC (*w,r,c,count*[,*stride*])

ウェーブ *w* の列に沿ってサンプリングされたデータのサブセットを返します。このサブセットは、行 *r* と列 *c* の要素から始まる *count* 個の要素を含みます。デフォルトでは *stride*=1 で、サンプリングは連続的です。
負のストライドを指定すると、開始要素から逆方向にサンプリングできます。
サンプリングが配列の境界をいずれかの方向で超える場合、このコマンドはエラーを返します。

例：

```
Make/O/N=(22,33) ddd=x
MatrixOP/O/P=1 aa0=subWaveC(ddd,4,5,10,2)
//      aa0={4,6,8,10,12,14,16,18,20,0}
MatrixOP/O/P=1 aa1=subWaveC(ddd,4,5,5,-4)
//      aa1={4,0,18,14,10}
```

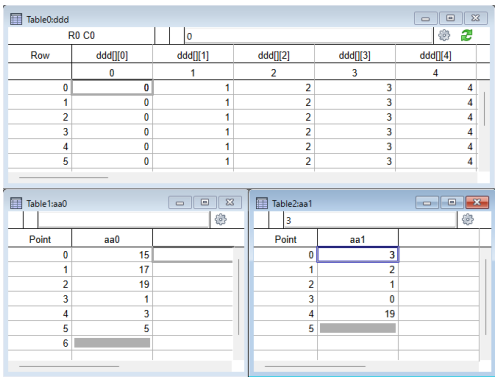


subWaveR (*w,r,c,count*[,*stride*])

ウェーブ *w* の行に沿ってサンプリングされたデータのサブセットを返します。行 *r*、列 *c* の要素から始まる *count* 個の要素を含みます。デフォルトでは *stride*=1 で、サンプリングは連続的です。
負のストライドを指定すると、開始要素から逆方向にサンプリングできます。
サンプリングが配列の境界をいずれかの方向で超える場合、このコマンドはエラーを返します。

例：

```
Make/O/N=(10,20) ddd=y
// 右境界を跨ぐフォワードサンプリング
MatrixOP/O/P=1 aa0=subWaveR(ddd,4,15,6,2)
//       aa0={15,17,19,1,3,5}
// 左境界を跨ぐ逆サンプリング
MatrixOP/O/P=1 aa1=subWaveR(ddd,2,3,5,-1)
//       aa1={3,2,1,0,19}
```



sum(*z*)

式 *z* のすべての要素の合計を返します。

sumBeams(*w*)

3D ウェーブ *w* の全ビームについて、全レイヤーにわたる和を含む（n×m）行列を返します。

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

ビームは Z 方向の 1D 配列です。

sumBeams は非レイヤー状関数で、*w* が別の式の結果ではなく、適切な 3D ウェーブであることが要求されます。

sumCols(*w*)

n×m 入力ウェーブ *w* の m 個の列の和を含む（1×m）行列を返します。

$$out_{j} = \sum_{i=0}^{nRows-1} w_{ij}$$

sumND(*w*)

トークン *w* の次元に関係なく、その和を含む 1 ポイントのウェーブを返します。

Igor Pro 9.0 で追加されました。

例：
次の例は、sumND とレイヤー単位で動作する sum 関数の違いを示しています。

```
Make/O/N=(5,6,3) w3D = z
MatrixOP/O/P=1 sumOut=sum(w3D)          // 3 レイヤーの合計を出力
MatrixOP/O/P=1 sumNDOut=sumND(w3D)      // 1 つの合計を出力
```

sumRows(*w*)

n×m 入力ウェーブ *w* の各行の和を含む（n×1）行列を返します。

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

sumSqr(*w*)

***w* の全要素の絶対値の二乗の和を返します。**

syncCorrelation(*w*)

実数値入力行列ウェーブ *w* に対する同期スペクトル相関行列を返します。asyncCorrelation コマンドも参照してください。

相関行列は、*w* の各列からその平均値を差し引き、得られた行列をその転置行列と乗算し、最後にすべての要素を（nrows-1）で除算することで計算されます。ここで nrows は *w* の行数です。

tan(*w*)

***w* のタンジェントを返します。**

tanh(*w*)

***w* の双曲タンジェントを返します。**

Igor Pro 7.0 で追加されました。

tensorProduct(*w1,w2*)

2D 行列 *w1* と *w2* のテンソル積である 2D 行列を返します。例えば、２つの（2×2）行列のテンソル積は次のように与えられます。

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

Igor Pro 7.0 で追加されました。

Trace(*w*)

w* の対角要素の和である実数または複素数のスカラーを返します。w* が正方行列でない場合、和は行と列のインデックスが同じ要素に対して計算されます。

transposeVol(*w,mode*)

3D ウェーブ *w* に対して、モードパラメーターの値に応じて転置された 3D ウェーブを返します。

mode=1： output=w[p][r][q]
mode=2： output=w[r][p][q]
mode=3： output=w[r][q][p]
mode=4： output=w[q][r][p]
mode=5： output=w[q][p][r]

transposeVol は非階層関数で、*w* が別の式の結果ではなく、適切な 3D ウェーブであることが必要です。

triDiag(*w1,w2,w3*)

w1* が上対角線、*w2* が主対角線、*w3* が下対角線となる三対角行列を返します。w2* が n 個のポイントを持つ場合、*w1* と *w3* は n-1 個のポイントを持つことが期待されます。ウェーブは任意の数値型で、返されるウェーブは入力に対応する数値型を持ちます。

uint8(*w*)

トークン *w* を 8 ビットの符号なし整数表現に変換します。詳細は下記の /NPRM フラグも参照してください。

Igor Pro 7.0 で追加されました。

uint16(*w*)

トークン *w* を 16 ビットの符号なし整数表現に変換します。詳細は下記の /NPRM フラグも参照してください。

Igor Pro 7.0 で追加されました。

uint32(*w*)

トークン *w* を 32 ビットの符号なし整数表現に変換します。詳細は下記の /NPRM フラグも参照してください。

Igor Pro 7.0 で追加されました。

varBeams(*w*)

実数値 3D ウェーブ *w* のビームの分散を含む行列を返します。

varBeams は非階層関数で、*w* が別の式の結果ではなく、適切な 3D ウェーブであることが必要です。

w が 1 つのレイヤーで構成される場合、返される分散はすべて NaN となります。

Igor Pro 9.0 で追加されました。

varCols(*w*)

***w* の各列に対応する分散を含む (1 x cols) のウェーブを返します。**

waveX(*w*)

純粋なウェーブの引数 *w* に対して、ウェーブの X スケーリングによって決定される各ポイントの X 値を返します。

Igor Pro 9.0 で追加されました。

waveY(*w*)

純粋なウェーブの引数 *w* に対して、ウェーブの Y スケーリングによって決定される各ポイントの Y 値を返します。

Igor Pro 9.0 で追加されました。

waveZ(*w*)

純粋なウェーブの引数 *w* に対して、ウェーブの Z スケーリングによって決定される各ポイントの Z 値を返します。

Igor Pro 9.0 で追加されました。

waveT(*w*)

純粋なウェーブの引数 *w* に対して、ウェーブの T スケーリングによって決定される各ポイントの T 値を返します。

Igor Pro 9.0 で追加されました。

waveIndexSet (*w1,w2,w3*)

***w2* の値に応じて、*w1* または *w3* から値を取得した *w1* と同じ次元の行列を返します。**具体的には以下の方法を使います。

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

w1 と *w2* は同じ行数と列数を持たなければなりません。*w1* と *w3* は数値型が一致しなければなりません。*w2* は符号なし型であってはなりません。

w2 の値は、*w3* へのポイント番号インデックスとして使われます。*w3* は実際の次元に関わらず 1D ウェーブとして扱われます。

w2 のインデックス値が *w3* の要素数以上である場合、そのインデックス値は範囲外となります。この場合、出力値は *w1* から取得され、あたかもインデックス値が負であるかのように扱われます。

waveMap(*w1*,*w2*) ***w1* [*w2* [i][j]] の値を含む、*w2* と同じ次元の配列を返します。**出力のデータ型は *w1* と同じです。*w2* の値は、*w1* 配列への 1D 整数インデックスとして扱われます。IndexSort コマンドも参照してください。

waveChunks(*w*) **ウェーブ *w* 内のチャンク数を返します。**

Igor Pro 7.0 で追加されました。

waveLayers(*w*) **ウェーブ *w* 内のレイヤー数を返します。**

Igor Pro 7.0 で追加されました。

wavePoints(*w*) **ウェーブ *w* 内のポイント数を返します。**

Igor Pro 7.0 で追加されました。

within (*w*,*low*,*high*[,*opt*]) **オプションのパラメーター *opt* が使われない場合、この関数は *w* と同じ次元の配列を返します。*w* の対応する要素が *low* と *high* の間にある場合（*low* <= *w*[i][j] <= *high*）、その要素は 1 に設定され、それ以外の場合は 0 に設定されます。*opt* が使われた場合、関数は指定範囲内の値に対して 1 を返し、それ以外の値に対して *opt* を返します。**特に、*opt* を NaN に設定することで、例えば以下のような形式の表現ができます。

```
zapNaNs(within(w,low,high,NaN))
```

パラメーター *low* と *high* は実数でなければならず、*low*<*high* を満たす必要があります。下限を除去するには *low*=-INF、上限を除去するには *high*=INF を使用できます。*w* に NaN が含まれる場合、対応する出力は 0 (または *opt*) となります。

オプションパラメーター *opt* は Igor Pro 10.0 で追加されました。

zapINFs(*w*) ***w* の連続データをすべて INF 要素を除去した 1 列のウェーブを返します。**入力 *w* は 1D または 2D ですが、それ以上の次元はサポートされていません。

Igor Pro 9.0 で追加されました。

zapNaNs(*w*) ***w* の連続データを NaN 要素をすべて除去した状態で含む 1 列のウェーブを返します。**入力 *w* は 1D または 2D ですが、それ以上の次元はサポートされていません。

Igor Pro 9.0 で追加されました。

zapZeros(*w*) ***w* と同じ数値型を持つ 1D ウェーブを返します。すべてのゼロ要素が除去されます。**入力 *w* は 1D または 2D のウェーブのみです。

zeroMat(*r,c,nt*) **数値型 '*nt*' の (*r* x *c*) 行列を返します。すべての要素はゼロに設定されます。**サポートされる型については WaveType を参照してください。上記の const も参照してください。

Igor Pro 7.0 で追加されました。

三角関数変換関数

FCT(*w,dir*) **1D ウェーブ *w* に対して、高速な実数・実数コサイン変換を計算します。**

前方変換（*dir*=1）は次のように定義されます。

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

区間の数は n=numpnts(w)-1 です。

逆変換（*dir*=-1）は次のように定義されます。

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

詳細については、以降の**三角変換**のセクションを参照してください。

Igor Pro 8.0 で追加されました。

FST(*w,dir*)

1D ウェーブ *w* に対して、高速な実数・実数サイン変換を計算します。

前方変換（*dir*=1）は次のように定義されます。

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

ここで、n=numpnts(w) です。

逆変換（*dir*=-1）は次のように定義されます。

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

詳細については、以降の**三角変換**のセクションを参照してください。

Igor Pro 8.0 で追加されました。

FSST(*w,dir*)

1D ウェーブ関数 *w* に対する高速な実数・実数スタガードサイン変換を計算します。

前方変換（*dir*=1）は次のように定義されます。

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

ここで、n=numpnts(w) です。

逆変換（*dir*=-1）は次のように定義されます。

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

詳細については、以降の**三角変換**のセクションを参照してください。

Igor Pro 8.0 で追加されました。

FSCT(*w,dir*)

1D ウェーブ *w* に対して、高速な実数・実数スタガードコサイン変換を計算します。

前方変換（*dir*=1）は次のように定義されます。

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

逆変換（*dir*=-1）は次のように定義されます。

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

詳細については、以降の**三角変換**のセクションを参照してください。

Igor Pro 8.0 で追加されました。

FSST2(*w,dir*)

1D ウェーブ *w* に対して、高速な実数・実数スタガード 2 サイン変換を計算します。

前方変換（*dir*=1）は次のように定義されます。

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

逆変換（*dir*=-1）は次のように定義されます。

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

詳細については、以降の**三角変換**のセクションを参照してください。

Igor Pro 8.0 で追加されました。

FSCT2(*w,dir*)

1D ウェーブ *w* に対して、高速な実数・実数スタガード 2 コサイン変換を計算します。

前方変換（*dir*=1）は次のように定義されます。

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

逆変換（*dir*=-1）は次のように定義されます。

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

詳細については、以降の**三角変換**のセクションを参照してください。

Igor Pro 8.0 で追加されました。

クォータニオン（四元数）を使った関数

これらの関数は、クォータニオントークンを作成・操作したり、クォータニオンの結果を返したりします。これらは Igor Pro 8.0 以降で追加されました。
MatrixOp におけるクォータニオンの背景情報については、ヘルプ MatrixOp Quaternion Data Tokens を参照してください。

quat(*arg*)

***arg* をクオータニオントークンに変換します。**

args は次のようなものになります。

- 実数クオータニオンに変換されるスカラー
- 純虚数クォータニオンに変換される 1×3 または 3×1 のウェーブ
- 別のクォータニオン呼び出しからの出力

クォータニオントークンに対する演算は、クォータニオンの演算規則に従います。

引数は複素数であってはなりません。生成されるクォータニオントークンは quat 関数によって正規化されません。

詳細はヘルプ MatrixOp Quaternion Data Tokens を参照してください。

Igor Pro 8.0 で追加されました。

quatFromSpherical (*theta, phi*)

球面極座標の角度 *theta* と *phi* に対応する MatrixOP クオータニオントークンを表す、４要素の 1D ウェーブを返します。

Igor Pro 10.0 で追加されました。

quatInverse (*inQuat*)

入力クォータニオンの逆数に対応する MatrixOP クォータニオントークンを表す、４要素の 1D ウェーブを返します。

Igor Pro 10.0 で追加されました。

quatToMatrix(*qIn*)

***qIn* がクォータニオントークンでない場合、それをクォータニオントークンに変換し、正規化して、同等の 4x4 ホモジニアス回転行列を返します。**

Igor Pro 8.0 で追加されました。

quatToAxis(*qIn*)

qIn がクオータニオントークンでない場合、それをクオータニオントークンに変換し、正規化して、対応する回転軸と回転角度を返します。結果は４要素のウェーブで、最初の３要素が回転軸を定義し、最後の要素がラジアン単位の回転角度です。

Igor Pro 8.0 で追加されました。

quatToEuler (*qIn, mode*)

***qIn* がクオータニオントークンでない場合、それをクオータニオントークンに変換し、ラジアンで表された等価なオイラー角を含む 3x1 ウェーブを返します。**

返されるオイラー角は、φ（X 軸周りの回転）、θ（Y 軸周りの回転）、ψ（Z 軸周りの回転）です。

mode パラメーターは回転順序を定義します。サポートされるモードは以下の通りです：121、123、131、132、212、213、231、232、312、313、321、323。1 は X 軸、2 は Y 軸、3 は Z 軸を指定します。次を参照してください：<https://www.euclideanspace.com/maths/geometry/rotations/conversions/quaternionToEuler/quat_2_euler_paper_ver2-1.pdf>

Igor Pro 8.0 で追加されました。

axisToQuat (*ax*)

***ax* は、回転軸を最初の３要素に、回転角度（ラジアン単位）を最後の要素に持つ４要素のウェーブです。この回転を表す四元数成分を含む４要素のウェーブを返します。**

Igor Pro 8.0 で追加されました。

slerp (*qIn1, qIn2,frac*)

クオータニオン qIn1 とクオータニオン qIn2 の間で球面線形補間を実行します。*frac* は 0 から 1 の間の値で、*qIn1* から *qIn2* への回転量を表します。

この関数は、回転後のクオータニオンを表すクオータニオン成分を含む４要素のウェーブを返します。これは例えば、一連の回転をアニメーション化する時に役立ちます。

Igor Pro 8.0 で追加されました。

ウェーブパラメーター

MatrixOp は 2D ウェーブ（行列）を扱うために設計されましたが、1D、3D、4D のウェーブにも対応します。

1D ウェーブは１列の行列として扱われます。
3D および 4D のウェーブは、各レイヤーが行列であるかのように、レイヤーごとに処理されます。

式内でウェーブのサブセットを参照できます。
サポートされるサブセットは２種類のみです。
１つの要素に評価されるもの（スカラーとして扱われます）と、１つ以上のレイヤーに評価されるものです。
例えば：

wave1d[a]	スカラー
wave2d[a][b]	スカラー
wave3d[a][b][c]	スカラー
wave3d[][][a]	3D ウェーブからのレイヤー
wave3d[][][a,b]	3D ウェーブからの a から b までのレイヤー
wave3d[][][a,b,c]	3D ウェーブからの a から b までの c 毎のレイヤー

MatrixOp 関数には、任意の次元のウェーブをパラメーターとして渡すことができます。
例えば：

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

MatrixOp では、代入の両側で同じ 3D ウェーブを使うことは許可されていません。

```
MatrixOp/O wave3D = wave3D + 3          // 不許可
```

詳細については、ヘルプ MatrixOp Wave Data Tokens を参照してください。

フラグ

/AT=*allocationType*
MatrixOP におけるメモリ割り当て方法のコントロールを可能にします。

allocationType = 0： 汎用的なメモリ割り当て
allocationType = 1： Intel のスケーラブルな割り当て
allocationType = 2： Intel のスケーラブルで整合性のある割り当て

デフォルトでは、MatrixOP はスケーラブルかつ整合性のある割り当てを使います。

Igor Pro 9.0 で追加されました。

/C
destWave に対して複素数ウェーブ参照を提供します。省略された場合、MatrixOp は *destWave* に対して実数ウェーブ参照を作成します。このウェーブ参照により、ユーザー定義関数の後続のステートメントで出力ウェーブを参照できます。

/FREE
destWave をフリーウェーブとして作成します。関数内でのみ許可され、単純な名前またはウェーブ参照構造体のフィールドが指定されている場合に限ります。

Igor Pro 6.1 以降が必要です。上級プログラマーのみ対象です。
詳細についてはヘルプ Free Waves を参照してください。

/FS
自動マルチスレッド処理やベクトル化された実装を使わず、コマンドを順次実行するように強制します。

/NTHR=*n*

3D ウェーブの結果を計算するために使うスレッド数を設定します。各スレッドは入力の 1 つのレイヤーの結果を計算します。

/NTHR を省略した場合、計算はメインスレッドのみによって実行されます。

n = 0：　コンピューター上のプロセッサ数と同じ数のスレッドを使います。
n > 0：　n は使うスレッド数を指定します。スレッド数を増やすことでパフォーマンスが向上する場合もありますが、必ずしも向上するとは限りません。

/NPRM

MatrixOp 式における数値データ型の自動昇格を制限します。

デフォルトでは、MatrixOp は数値データ型を昇格させ、演算結果が妥当な精度となるようにします。状況によっては、切り捨てやオーバーフローのリスクがあっても、結果を特定のデータ型として保持したい場合があります。/NPRM フラグを指定すると、式内で最も高精度なデータ型を使って出力ウェーブを作成します。例えば、B が 16 ビットウェーブ、C が 8 ビットウェーブである式 A=B+C では、出力ウェーブ A は 16 ビットウェーブとなります。

符号なし数値型は、すべてのオペランドが符号なしの場合にのみ生成されます。

データ昇格が必要な場合、/NPRM は無視されます。
例えば：

```
Make/B/U wave2
MatrixOp/O/NPRM wave1 = -wave2
```

int8、int16 などの MatrixOp 関数を使って、任意のトークンの数値型を正確にコントロールできます。

浮動小数点型から整数型への特異値の変換は、以下の規則に従います。

1. NaN は 0 に変換される
2. INF は宛先の整数型の最大値に変換される
3. -INF は、目的の整数型の最小値に変換される

/NV=*mode*

Intel MKL のベクトル化関数の使用をコントロールします。

mode=0：　非ベクトル化実行を要求する。これにより、追加スレッドの生成を防止し、プリエンプティブスレッド内で MatrixOP が呼び出された時の速度向上が期待できる
mode=1：　デフォルト。可能な限りベクトル化された関数を使う

Igor Pro 9.0 で追加されました。

/O

destWave が既に存在する場合、上書きします。

/P=*printMode*

MatrixOp 評価の結果の出力をコントロールします。

MatrixOp がメインスレッドで実行されていない場合、/P は無視されます。

printMode は 0 から 2 の間の値です。

0：　出力は行われない（デフォルト）
1：　式の評価結果をレイヤーごとの順序でコマンドウィンドウの履歴エ

リアに出力し、結果を指定のウェーブに保存する
2： 式の評価結果をレイヤーごとの順序でコマンドウィンドウの履歴エリアに出力するが、結果値を宛先ウェーブに作成または保存しない。宛先ウェーブが既に存在する可能性がある場合、コマンドによって宛先ウェーブが変更されない場合でも、/O フラグを含める必要がある

MatrixOp を通常通り動作させ、その後出力ウェーブを表示したい場合は /P=1 を使ってください。

ウェーブを生成または変更せずに MatrixOp の結果を出力したい場合は /P=2 を使ってください。

値は 16 桁の精度で順次出力されます。出力は行と列を表すようにフォーマットされていません。

コマンドの結果が 3D または 4D ウェーブである場合、結果はレイヤーごとに出力されます。

後半の「MatrixOp の出力例」のセクションを参照してください。

Igor Pro 8.0 で追加されました。

/S 既存の宛先ウェーブが MatrixOp/O コマンド内で定義されている場合、そのウェーブの次元スケーリング、単位、ウェーブノートを保持します。宛先ウェーブがフリーウェーブの場合、このフラグは効果を持ちません。

/T=*k* 結果をテーブル形式で表示します。*k* はユーザーが閉じようとした時の動作を指定します。

k = 0： 通常のダイアログ（デフォルト）
k = 1： ダイアログ無しで Kill

詳細

MatrixOp の一般的な形式は次の通りです。

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

destWave には MatrixOp によって生成されるウェーブ、または MatrixOp/O によって上書きされるウェーブを指定します。

destWave は単純なウェーブ名、部分的なデータフォルダーパス、または完全なデータフォルダーパスを指定できます。
ユーザー定義関数内では、単純なウェーブ名、または /O オプションが指定されている場合は既存のウェーブを参照するウェーブ参照を指定できます。

expression とは、1 つ以上のデータトークンと、上に列挙した組み込みの MatrixOp 関数と MatrixOp 演算子を組み合わせた数学的表現です。
MatrixOp は、ウェーブフォーム代入文で使われる p、q、r、s、または x、y、z、t 記号をサポートしません。

データトークンにはウェーブ、変数、およびリテラル数値が含まれます。

オペランドには任意のデータ型の組み合わせを使用できます。
特に、*expression* 内で実数型と複素数型を混在させることが可能です。
MatrixOp は、Wave/C などの型宣言に関係なく、実行時に入力データの型と適切な出力データ型を決定します。

詳細については、ヘルプ Using MatrixOp を参照してください。

三角関数変換

三角関数変換関数は Igor Pro 8.0 で追加されました。

FST、FCT、FSST、FSCT、FSST2、FSCT2 は、Intel MKL ライブラリ関数を使って実装されています。
これらの変換は、自動的にマルチスレッド処理で実行される場合があります。

前方変換と逆変換の定義式は、Intel のドキュメントに従います（<https://software.intel.com/en-us/mkl-developer-reference-c-trigonometric-transforms-implemented>を参照）。

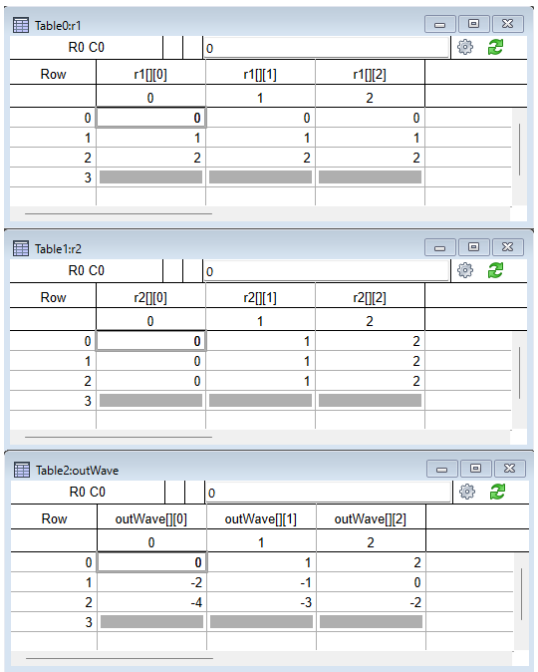
MatrixOP の実装では、すべての入力および出力配列はゼロベースです。
これを、スタガード２コサイン関数の以下の例で示します。

```
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
```

例

これらの例に加えて、ヘルプ MatrixOp Optimization Examples を参照してください。

```
// 次の行列を以降の例の前半で使います
Make/O/N=(3,3) r1=x,r2=y

// 行列の加算とスカラーによる行列の乗算
MatrixOp/O outWave=r1+r2-3*r1
```



つまり

r1[0][0]+r2[0][0]-3*r1[0][0] = 0+0-3*0 = 0
r1[1][0]+r2[1][0]-3*r1[1][0] = 1+0-3*1 = -2
r1[2][0]+r2[2][0]-3*r1[2][0] = 2+0-3*2 = -4

のようになります。

```
// 行列の恒等関数を使う
MatrixOp/O outWave=Identity(3) x r1

// 別の計算のために恒常的な単位行列を作成
MatrixOp/O id4=Identity(4)

// Trace 関数の使用
MatrixOp/O outWave=(Trace(r1)*identity(3) x r1)-3*r1

// 行列乗算における逆行列関数 Inv() の使用
MatrixOp/O outWave=Inv(r2) x r2

// 行列式関数 Det() の使用
MatrixOp/O outWave=Det(r1)+Det(r2)

// 転置後置演算子の使用
MatrixOp/O outWave=r1^t+(r2-r1)^t-r2^t

// 実データと複合データの混合の使用
Variable/C complexVar=cmplx(1,2)
MatrixOp/O outWave=complexVar*r2 - Cmplx(2,4)*r1

// エルミート（Hermitian）転置演算子
MatrixOp/O outWave=Trace(complexVar*r2)^h -Trace(cmplx(2,4)*r1)^h

// その場での操作 + 複素数への変換
MatrixOp/O r1=r1*cmplx(1,2)

// 2D 空間フィルター filterWave を使った画像フィルタリング
MatrixOp/O filteredImage=IFFT(FFT(srcImage,2)*filterWave,3)

// 正のシフト
Make/O w={0,1,2,3,4,5,6}
MatrixOp/O w=shiftVector(w,2,77)
Print w
// 出力  w[0]= {77,77,0,1,2,3,4}

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

// kronProd 関数を使ったアダマール（Hadamard）行列の生成
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 の例
Make/N=8 index = p
MatrixOP/O/P=1 out = bitReverseCol(index,0,0)  // 直接のバイナリ反転
                                               // 出力： out = {0,4,2,6,1,5,3,7}
```

```
MatrixOP/O/P=1 out = bitReverseCol(index,0,1)  // アダマール（Hadamard）順
                                               // 出力： out = {0,4,6,2,3,7,5,1}
MatrixOP/O/P=1 out = bitReverseCol(index,0,2)  // Dyadic/Paley/Gray 順
                                               // 出力： out = {0,1,3,2,6,7,5,4}
```

MatrixOp 出力例

```
// 結果の値を１つ出力する
Make/O/N=(10,5) m2D=(p+1)*(q+1)
MatrixOP/O/P=1 aa=sum(m2D)
// 出力： aa={825}

// 実数の 2D の結果を出力
Make/O/N=(1,4) ddd=enoise(3)
MatrixOp/O/P=1 aa=ddd
// 出力： aa={-2.273916482925415,-2.327789783477783,1.286988377571106,-0.8658701777458191}

// 高次元の結果をレイヤーごとに出力
Make/O/N=(3,4,3) ddd=z+1
MatrixOP/O/P=1 aa=mean(ddd)
// 出力：  aa={1}          // 各レイヤーは 1x1 の値となる
          aa={2}
          aa={3}
```

参考文献

syncCorrelation と 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.

参照

ヘルプ Using MatrixOp、FastOp

Igor の行列ルーチンの詳細はヘルプ Matrix Math Operations を参照してください

デモ

FFT Swapping Demo

Kalman Filter Example

（File→Example Experiments→Analysis）