CONTENTS

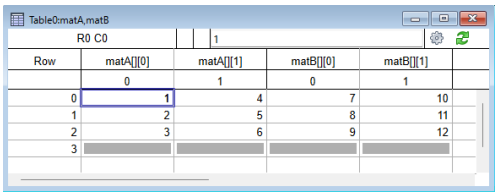
行列計算コマンド

行列計算を実行する基本的な方法は４つあります：正規ウェーブ表現、MatrixXXX コマンド、MatrixOp コマンド、MatrixSparse コマンド演算です。

正規ウェーブ表現

通常のウェーブ表現を使って、行列を他の行列やスカラーに加算できます。
また、行列をスカラーで乗算することもできます。
例えば：

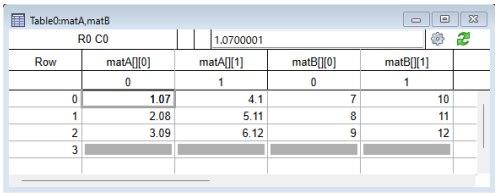
```
Make matA={{1,2,3},{4,5,6}}, matB={{7,8,9},{10,11,12}}
Edit matA, matB
```




```
matA = matA+0.01*matB
```

これは matA に新しい値を代入します。

```
matA = {{1.07,2.08,3.09},{4.1,5.11,6.12}}
```



MatrixXXX コマンド

Igor では、いくつかの行列演算が実装されています。
そのほとんどは「Matrix」という単語で始まる名前を持っています。
例えば、MatrixMultiply コマンドを使って一連の行列を乗算できます。
/T フラグを使うと、乗算に使う前に指定した行列のデータを転置することを指定できます。

Igor のマトリクスコマンドの多くは LAPACK ライブラリを使っています。
LAPACK の詳細については以下を参照してください：

LAPACK Users' Guide, Third Edition, 1999　(SIAM Publications, Philadelphia, ISBN: 0-89871-447-8)

または

LAPACK の Web サイト

<http://www.netlib.org/lapack/lug/lapack_lug.html>

特に断りがない限り、LAPACK ルーチンは実数および複素数ウェーブ、IEEE 単精度および倍精度行列をサポートします。
ほとんどの行列演算では、変数 V_Flag が作成され、演算が成功すると 0 に設定されます。
フラグが負の数に設定された場合、LAPACK ルーチンに渡されたパラメーターのいずれかが無効であることを示します。
フラグの値が正の場合、通常、入力行列の行／列のいずれかが問題の原因であることを示します。

MatrixOp コマンド

MatrixOp コマンドは行列式の実行効率を向上させ、構文を簡素化します。
例えば、次のコマンド：

```
MatrixOp matA = (matD - matB x matC) x matD
```

は、標準的な優先順位規則に従った行列の乗算および減算に相当します。

詳細は「MatrixOp の使用」のセクションを参照してください。

MatrixSparse コマンド

MatrixSparse コマンドは、要素の大部分が 0 である大規模行列の計算において、パフォーマンスの向上とメモリ使用量の削減を実現します。
詳細は「疎行列」のセクションを参照してください。

Matrix コマンド一覧

以下に行列の数学コマンドと関数を示します。

一般

- `MatrixCondition(`*`matrix`*`)`
  2D 正方行列 wave2D の条件数の推定逆数を返す
- `MatrixConvolve` *`coefMatrix, dataMatrix`*
  小さな係数行列カーネル Wave と srcWave を畳み込む
- `MatrixCorr [/COV][/DEGC]` *`waveA`* `[,` *`waveB`*`]`
  入力 1D ウェーブに対する相関、共分散、または相関行列の度合いを計算する
- `MatrixDet(`*`matrix`*`)`
  dataMatrix の行列式を返す
- `MatrixDot(`*`waveA, waveB`*`)`
  ２つの 1D ウェーブの内部（スカラー）積を計算する
- `MatrixFilter [/N=`*`n`*` /P=`*`p`*` /b=`*`b`*` /R=`*`roiWave`*` ]` *`Method dataMatrix`*
  対象データマトリックスに対して複数の標準的な画像フィルター処理のうちの１つを実行する
- `MatrixGLM [/Z]` *`matrixA, matrixB, waved`*
  ベクトル y の 2-norm を最小化する一般的な Gauss-Markov Linear Model problem（GLM）を解く
- `MatrixMultiply` *`matrixA`* `[/T],`*`matrixB`* `[/T] [,`*`additional matrices`*`]`
  行列式 A*B を計算し、結果を現在のデータフォルダー内に生成された M_product という名前の行列ウェーブに格納する
- `MatrixOp [/O]` *`destwave = expression`*
  式を評価し、結果を destWave に格納する
- `MatrixRank(`*`matrix`* `[,` *`maxConditionNumber`*`])`
  指定された条件数に従って matrixWaveA のランクを返す

- MatrixTrace(*matrix*)
  正方行列のトレース（対角要素の和）を計算する
- MatrixTranspose *matrix*
  行列の行と列を入れ替える

固有値、固有ベクトル、分解

- MatrixBalance [*flags*] *srcWave*
  N×N の実数または複素数行列を並べ替えまたは縮尺し、固有値や固有ベクトルなどの後続計算においてより安定した類似行列を得る
- MatrixEigenV [/O/B=*balance* /S=*sense* /X/R/L/Z] *matrixWave*
  LAPACK ルーチンを使って正方行列の固有値または固有ベクトルを計算する
- MatrixFactor [*flags*] *srcWave*
  概念的に matA および matB と呼ばれる 2 つの実数値出力行列を計算する
- MatrixGLM *matrixA*, *matrixB*, *waved*
  ベクトル y の 2-norm を最小化する一般的な Gauss-Markov Linear Model problem（GLM）を解く
- MatrixInverse [[/D]/P][/G][/O] *srcWave*
  行列の逆行列または擬似逆行列を計算する
- MatrixLUD *matrix*
  行列の LU 分解を計算する
- MatrixLUDTD *srcMain*, *srcUpper*, *srcLower*
  三対角行列の LU 分解を計算する
- MatrixReverseBalance [*flags*] *scaleWave*, *eigenvectorsWave*
  MatrixBalance を使ってバランス調整された行列に対して計算された eigenvectorsWave に含まれる左固有ベクトルまたは右固有ベクトルを逆変換する
- MatrixSchur [/Z] *matrix*
  N×N の非対称行列 srcMatrix に対して、固有値、A からの実 Schur、Schur ベクトル V の行列を計算する
- MatrixSVD *matrix*
  特異値分解アルゴリズムを使って、M×N matrixWave を 3 つの行列の積に分解する

線形方程式と最小二乗法

- MatrixGaussJ *matrixA*, *vectorsB*
  行列 A と列ベクトル b が与えられた場合、行列式 A*x=b を列ベクトル x について解く
- MatrixLinearSolve [/D={*subDiagonals*,*superDiagonals* }/M=*method* /O/Z]
  [/L][/U] *matrixA*, *matrix*
  線形連立方程式 matrixA * X=matrixB を解く
- MatrixLinearSolveTD [/Z] *upperW*, *mainW*, *lowerW*, *matrix*
  線形方程式系 TDMatrix * X = matrixB を解く
- MatrixLLS [/O/Z/M=*method*] *matrixA*, *matrix*
  QR/LQ 分解または SV 分解を用いて、M×N 行列 A を含む過決定または欠決定線形方程式系を解く
- MatrixLUBkSub *matrtixL*, *matrixU*, *index*, *vector*
  MatrixLUBkSub は LU 分解の逆代入を提供する
- MatrixSolve *method*, *matrixA*, *vector*
  MatrixLLS に置き換えられ、後方互換性のためだけに含まれている
- MatrixSVBkSub *matrixU*, *vectorW*, *matrixV*, *vector*
  SV 分解のための逆代入を行う

疎行列

- MatrixSparse
  疎行列に対する基本的な行列演算をサポートする

MatrixOp の使用

数学的計算を行う時、最初に選択するのは次のような標準的なウェーブの代入を記述することかもしれません。

```
wave1 = wave2 + wave3
```

右辺（RHS）の式が複雑な場合やウェーブが大きい場合、MatrixOp（「行列演算」の略称）を使うことで大幅な性能向上が得られます。
MatrixOp は当初 2D ウェーブに対する計算を目的として考案され、後に任意の次元のウェーブへ拡張されました。

MatrixOp の基本形を使うには、代入文の先頭にその名前を付けるだけです。

```
MatrixOp wave1 = wave2 + wave3
```

一般的な形式は次の通りで、

```
destWave = expression
```

両方のコマンドは似ているように見えますが、動作が異なります。

重要な違いは、MatrixOp がウェーブスケーリングを考慮せず純粋な配列要素に対して動作し、右辺（RHS）で x, y, z, t, p, q, r, s を使ったインデックス指定をサポートしない点です。

ウェーブの代入では、実行時に宛先ウェーブが存在している必要がありますが、MatrixOp は宛先ウェーブを自動的に作成します。
以下のコマンドでは：

```
Make/O wave2=1/(p+1), wave3=1
MatrixOp/O wave1 = wave2 + wave3
```

MatrixOp は単精度（single precision：SP）ウェーブ wave1 を作成し、結果をそこに格納します。

MatrixOp が宛先ウェーブを作成する場合、そのデータ型と次元は右辺の式に依存します。
前述の例では、wave2 と wave3 が SP であるため、MatrixOp は wave1 を SP として作成します。
宛先のデータ型と次元を決定するための MatrixOp のルールについては、「MatrixOp データプロモーションポリシー」のセクションで説明します。

これらの規則により、行列演算とウェーブの代入の間には以下の違いが生じます。

```
Make/O wave1 = 42          // wave1 は 128 ポイントのウェーブ。すべてのポイントを 42 に設定

MatrixOp/O wave1 = 42      // wave1 は 1 ポイントのウェーブで 42 に設定
```

代入文は右辺を評価し、結果をポイント単位で格納します。
このため、代入先のウェーブが右辺に現れると予期せぬ結果が生じることがあります。
次の例では、代入文の評価中に WaveMin(wave2) が変化します。

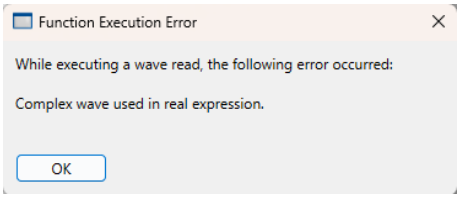
```
wave2 = wave2 - WaveMin(wave2)
```

対照的に、MatrixOp は宛先ウェーブに何かを格納する前に、宛先の全ポイントについて右辺を評価します。
その結果、このコマンドは期待通りに動作します：

```
MatrixOp/O wave2 = wave2 - minVal(wave2)
```

右辺に複素数が含まれる場合、もう１つの重要な違いが現れます。
例えば次を試してみると、

```
Make/O/C cwave2
Make/O wave1, wave3
wave1 = cwave2 + wave3
```



「Complex wave used in real expression（実数式に複素数ウェーブが使われました）」というエラーになります。

対照的に、MatrixOp は宛先のオブジェクトを複素数として作成するため、次のコマンドはエラーなく動作します。

```
MatrixOp/O wave1 = cwave2 + wave3
```

この例では、右辺（RHS）が実数ウェーブと複素数ウェーブを混合していますが、MatrixOp はこれを問題なく処理します。

標準の数学演算子に加え、MatrixOp は右辺のオペランドに適用できる一連の関数をサポートしています。
例としては abs、sin、cos、mean、minVal、maxVal などがあります。
MatrixOp リファレンス文書と、「カテゴリ別 MatrixOp 関数」のセクションにリストされているものを含め、さらに多くの関数が存在します。

MatrixOp データトークン

MatrixOp コマンドの一般的な形式は次の通りです：

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

右辺の式は、データトークンと行列演算演算子および行列演算関数の組み合わせです。
データトークンは次のいずれかです：

- リテラル数値
- Constant キーワードで宣言された数値定数
- 数値のローカル変数
- 数値のグローバル変数
- 数値のウェーブ
- 数値のウェーブレイヤー
- MatrixOp 関数の結果

式からユーザー定義関数または外部関数を呼び出すことはできません。

次の関数は、各種類のデータトークンを説明しています：

```
Constant kConstant = 234
Function Demo()
        // リテラル数値
        MatrixOp/O dest = 123

        // 定数
        MatrixOp/O dest = kConstant
```

```
        // ローカル変数
        Variable localVariable = 345
        MatrixOp/O dest = localVariable

        // グローバル変数
        Variable/G root:gVar = 456
        NVAR globalVariable = root:gVar
        MatrixOp/O dest = globalVariable

        // ウェーブ
        Make/O/N=(3,3) mat = p + 10*q
        MatrixOp/O tMat = mat^t

        // ウェーブレイヤー
        Make/O/N=(3,3,3) w3D = p + 10*q + 100*r
        MatrixOp/O layer1 = w3D[][][1]

        // MatrixOp 関数の出力
        MatrixOp/O invMat = inv(mat)
End
```

MatrixOp ウェーブデータトークン

MatrixOp は数値データとウェーブの次元のみを使います。
その他のプロパティ、特にウェーブのスケーリングは無視されます。
アプリケーションでウェーブのスケーリングが重要な場合は、MatrixOp の後に CopyScales または SetScale を実行するか、MatrixOp の代わりにウェーブの代入文を使ってください。

コマンドラインでは、ウェーブを式内で参照する時に、ウェーブ名のみ、ウェーブへの部分データフォルダーパス、またはウェーブへの完全データフォルダーパスを使用できます。
ユーザー定義関数内では、ウェーブを参照する時に、ウェーブ名のみ、または既存のウェーブを指すウェーブ参照変数を使用できます。
これらのウェーブ参照方法を総称して「ウェーブ参照」と呼びます。

ウェーブデータトークンは、ウェーブのサブセットを特定するインデックスで任意に修飾されたウェーブ参照で構成されます。
例えば：

```
Function Demo()
        // w3D に対する自動ウェーブ参照を作成
        Make/O/N=(3,4,5) w3D = p + 10*q + 100*r

        MatrixOp/O dest = w3D               // dest は 3D ウェーブ
        MatrixOp/O dest = w3D[0][1][2]      // dest は 1 ポイントを持つ 1D ウェーブ
        MatrixOp/O dest = w3D[0][1][]       // dest は 1 レイヤーを持つ 2D ウェーブ
End
```

MatrixOp がサポートするウェーブのサブセットは次の 2 種類のみです：

- スカラー（1 つの値）として評価される部分集合
- 1 つ以上のレイヤー（2D 配列）として評価される部分集合

行（w3D[1][][2]）または列（w3D[][1][2]）として評価される部分集合はサポートされていません。

スカラー

以下の式はスカラーに評価されます：

wave1d[*a*]	*a* はリテラル数値またはローカル変数です。MatrixOp はインデックス *a* を wave1d の有効範囲にクリップします。この式は参照されたウェーブ要素に等しいスカラーを評価します。
wave2d[*a*][*b*]	*a* と *b* はリテラル数値またはローカル変数です。MatrixOp は対応する次元の有効範囲にインデックスをクリップします。この式は参照されたウェーブ要素に等しいスカラーを評価します。
wave3d[*a*][*b*][*c*]	*a*、*b*、*c* はリテラル数値またはローカル変数です。MatrixOp は対応する次元の有効範囲にインデックスをクリップします。この式は参照されたウェーブ要素に等しいスカラーを評価します。

レイヤー

以下は 1 つ以上の 2D レイヤーに評価されます：

wave3d[][][*a*]	3D ウェーブのレイヤー *a* は 2D 行列として扱われます。最初の括弧ペアと 2 番目の括弧ペアは空でなければなりません。MatrixOp は *a* をレイヤーの有効範囲にクリップします。結果は 2D ウェーブです。
wave3d[][][*a*,*b*]	レイヤー *a* とレイヤー *b* の間の全レイヤーに対して式が評価されます。MatrixOp は *a* と *b* を有効なレイヤー範囲にクリップします。結果は 3D ウェーブです。
wave3d[][][*a*,*b*,*c*]	*expression* は、レイヤー *a* から始まり、レイヤー *b* までレイヤー数を *c* ずつ増加させて評価されます。*a*、*b*、*c* はスカラーでなければなりません。レイヤーは有効範囲にクリップされ、*c* は正の整数でなければなりません。

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使用できます。
例えば：

```
Make/O/N=128 wave1d = x
MatrixOp/O outWave = exp(wave1d)
```

MatrixOp は、方程式の両辺に同じ 3D ウェーブを含む式をサポートしていません：

```
MatrixOp/O wave3D = wave3D + 3      // 許可されない
```

代わりに次のように使います：

```
MatrixOp/O newWave3D = wave3D + 3
```

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、式内で実数型と複素数型を混在させることができます。
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。

MatrixOp 関数：　特定の種類の 2D データトークンを生成できます。
const 関数、zeroMat 関数：　指定された次元の固定値を含む行列を生成します。
identity 関数：　単位行列を生成し、triDiag 関数は 1D 入力ウェーブから三対角行列を生成します。
rowRepeat 関数、colRepeat 関数：　1D ウェーブから行列を構築します。
setRow 関数、setCol 関数、setOffDiag 関数：　1D ウェーブから 2D ウェーブの要素へデータを転送します。

MatrixOp とウェーブの次元

MatrixOp は 2D 配列（行列）の演算を最適化するために設計されましたが、様々な式において他のウェーブもサポートします。
ただし、ポイントがゼロのウェーブはサポートしていません。

1D ウェーブは 1 列の行列として扱われます。
3D と 4D ウェーブはレイヤーの配列として扱われるため、それらの代入はレイヤーごとに処理されます。

```
Make/O/N=(4,5) wave2 = q
Make/O/N=(4,5,6) wave3 = r
MatrixOp/O wave1 = wave2 * wave3    // 2D は 3D を 1 層ずつ乗算
```

wave1 は次元 (4,5,6) の 3D ウェーブとしてまとめられます。
wave1 の各レイヤーは、wave3 の対応するレイヤーの内容を、wave2 の内容と要素ごとに乗算した結果を含みます。

レイヤーごとの処理における例外は、beam や transposeVol などの特別な行列演算関数です。

一部の二項演算子は、ウェーブオペランドに対して次元の制限を設けています。
例えば、行列乗算（wave2 x wave3）では、wave2 の列数が wave3 の行数と等しい必要があります。
通常の乗算（wave2* wave3）では、両者の行数と列数が等しい必要があります。
例えば、

```
Make/O/N=(4,6) wave2
Make/O/N=(3,8) wave3
MatrixOp/O wave1 = wave2 * wave3    // エラー： "Matrix Dimensions Mismatch"
```

このルールの例外は、右辺のオペランドがウェーブ要素 1 つである場合です。

```
MatrixOp/O wave1 = wave2 * wave3[2][3]     // スカラー乗算に等しい
```

一部の MatrixOp 関数は行列の各列に対して操作を行い、1 行 N 列のウェーブを生成します。
これは Igor の他の部分で使うと混乱を招く可能性があります。
必要なものが N 行の 1D ウェーブである場合、ウェーブの次元を変更するか、MatrixOp コマンドに単純に転置演算子を含めることで対応できます。

```
Make/O/N=(4,6) wave2=gnoise(4)
MatrixOp/O wave1=sumCols(wave2)^t  // wave1 は 6 行からなる 1D ウェーブ
```

MatrixOp の演算子

MatrixOp は、右辺式で使用可能な 8 つの二項演算子と 2 つの後置演算子を定義します。
これらの演算子は、MatrixOp コマンドのヘルプの Operator のセクションに記載されています。

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。

基本演算子である +, -, *, / は、通常のウェーブ代入と同様に動作しますが、 1 つの例外があります。
- または / 演算が行列とスカラーを扱う場合、行列が最初のオペランドでスカラーが 2 番目のオペランドである場合にのみサポートされます。
例えば：

```
Make/O wave2 = 1
Variable var1 = 7
MatrixOp/O wave1 = wave2 - var1     // OK: wave2 の各ポイントから var1 を引く
```

```
MatrixOp/O wave1 = var1 - wave2     // エラー：行列からスカラーを引くことはできない
MatrixOp/O wave1 = var1 / wave2     // エラー：スカラーを行列で割ることはできない
```

スカラーを行列で割る演算は、逆数関数を使って次のように実現できます。

```
MatrixOp/O wave1 = scalar * rec(wave2)
```

ドット演算子「.」は、一般化された内積を計算するために使われます

```
MatrixOp/O wave1 = wave2 . wave3    // '.' の両側の空白は任意
```

x 演算子は行列の乗算を表します。

```
MatrixOp/O wave1 = wave2 x wave3    // 'x' の両側の空白は必須！
```

論理演算子 && と || は実数値データトークンに限定され、値 0 または 1 の符号なしバイト数値トークンを生成します。

後置演算子 ^t（行列の転置）と ^h（エルミート転置）は、それぞれ左側の最初のトークンに対して作用します。

```
MatrixOp/O wave1 = wave2^t + wave3
```

転置演算は優先順位が高いため、加算よりも先に実行されます。

左側のトークンは、例えば以下のような複合トークンである可能性があります。

```
MatrixOp/O wave1 = sumCols(wave2)^t
```

MatrixOp は、接尾辞演算子 ^t および ^h においてのみ ^ 文字をサポートします。
累乗演算には powR および powC 関数を使ってください。

MatrixOp の乗算とスケーリング

MatrixOp は、複数の乗算およびスケーリング機能を提供します。
これには行列とスカラーの乗算が含まれます。

```
MatrixOp/O wave1 = wave2 * scalar          // ウェーブの代入と同じ
```

ウェーブとウェーブの乗算は次です。

```
MatrixOp wave1 = wave2 * wave3             // レイヤーごとのサポートも含まれる
```

行列と行列の乗算は次です。

```
MatrixOp wave1 = wave2 x wave3
```

後者は MatrixMultiply コマンドと同等であり、複雑な複合式を 1 行で記述・実行できる利便性を備えています。

MatrixOp は、頻繁に使われる 2 つの特殊なスケーリング関数を追加します。
scaleCols 関数は各列を異なるスカラーで乗算し、scale 関数は入力値を指定された範囲にスケーリングします。

MatrixOp によるデータの再配置と抽出

MatrixOp は、行列内のデータを再配置したり、さらなる計算のためにデータのサブセットを抽出したりするためによく使われます。
ウェーブから要素やレイヤーを抽出するには、角括弧によるインデックス指定を使用できます。

```
MatrixOp destWave = wave1d[a]              // ウェーブのポイント a からスカラーを抽出
MatrixOp destWave = wave2d[a][b]           // ウェーブの要素 a,b からスカラーを抽出
MatrixOp destWave = wave3d[a][b][c]        // ウェーブの要素 a、b、c からスカラーを抽出
MatrixOp destWave = wave3d[][][a]          // 3D ウェーブからレイヤーを抽出
```

また、3D ウェーブからレイヤーを抽出することも可能です。
レイヤー a から開始し、レイヤー番号を c ずつ増加させてレイヤー b まで抽出します：

```
MatrixOp destWave = wave3d[][][a,b,c]
```

a、b、c はスカラーでなければなりません。
各スカラーは有効範囲にクリップされ、c は正の整数でなければなりません。

行、列、レイヤー、チャンクをループ処理する必要がある場合、MatrixOp 関数である row、col、layer、chunk を使って関連データを抽出することが役立ちます。
getDiag を使うと行列の対角要素を抽出できます。
サブレンジ（一部）を使えば複数の行または複数の列を抽出できます。

Rotate コマンドは 1D ウェーブに対して機能します。
MatrixOp はこれを高次元へ拡張し、次の関数を提供します：rotateRows、rotateCols、rotateLayers、rotateChunks。

MatrixOp transposeVol は ImageTransform transposeVol と類似していますが、複素数データ型をサポートします。

MatrixOp redimension 関数は、1D ウェーブを 2D 配列に変換するために設計されていますが、高次元ウェーブからデータを抽出することも可能です。

MatrixOp データプロモーションポリシー

MatrixOp は、右辺の式で使われるデータ型と演算に基づいて、宛先ウェーブのデータ型を選択します。
以下に例を示します（SP=Single Precision、DP=Double Precision）。

```
Make/O/B/U wave2, wave3                    // ２つの符号なしバイトウェーブを作成
MatrixOp/O wave1 = wave2                   // wave1 は符号なしバイト
MatrixOp/O wave1 = wave2 + wave3           // wave1 は符号なしワード（16 bit）
MatrixOp/O wave1 = wave2 * wave3           // wave1 は符号なしワード（16 bit）
MatrixOp/O wave1 = wave2 / wave3           // wave1 は SP
MatrixOp/O wave1 = magsqr(wave2)           // wave1 は SP
```

例では、コマンドの結果として取り得る範囲を表すために、宛先ウェーブのデータ型が変更される様子を示しています。
MatrixOp は、特定のウェーブに対してプロモーションが必要かどうかを判断するために、ウェーブ内のデータを検査しません。

Igor 変数は通常、倍精度データトークンとして扱われます。
MatrixOp は整数を含む変数に対して特別なルールを適用します。
例を以下に示します。

```
Make/O/B/U wave2                           // 符号なしバイトウェーブを作成
Variable vv = 2.1                          // DP 値を含む変数
MatrixOp/O wave1 = wave2*vv                // wave1 は DP
```

次に、変数を変更して様々な整数を格納します。

```
vv = 2
MatrixOp/O wave1 = wave2*vv                    // wave1 は符号なしワード（16 bit）
vv = 257
MatrixOp/O wave1 = wave2*vv                    // wave1 は符号なし整数（32 bit）
vv = 65538
MatrixOp/O wave1 = wave2*vv                    // wave1 は DP
```

これらの例は、MatrixOp が変数 vv を、その内容に適合するデータ型を持つトークンとして表現することを示しています。
変数が整数値を含まない場合、トークンは DP として扱われます。

MatrixOp は複雑な数学計算に便利であり、必要に応じて実数または複素数ウェーブを自動的に生成します。
例えば：

```
Make/O/C wave2, wave3                          // SP 複素数
Make/O wave4                                   // SP 実数
MatrixOp/O wave1 = wave2 * wave3 * wave4       // wave1 は SP 複素数
MatrixOp/O wave1 = abs(wave2*wave3) * wave4    // wave1 は SP 実数
```

この複雑なポリシーの例外は、入力が実数で負の場合に NaN を返す sqrt 関数です。

データ型のプロモーションを制限したい場合は、/NPRM フラグを使用できます。

```
Make/O/B/U wave2, wave3
MatrixOp/O wave1 = wave2 * wave3               // wave1 は符号なしワード（16 bit）
MatrixOp/O/NPRM wave1 = wave2 * wave3          // wave1 は符号なしバイト
```

多くの MatrixOp 関数は、処理前に整数トークンを SP に変換するため、/NPRM フラグの影響を受けません。
これには、すべての三角関数、chirp、chirpZ、fft、ifft、正規化関数、sqrt、log、exp、および前方/後方置換が含まれます。

MatrixOp 複合式

通常のデータトークンに対して動作するほとんどの MatrixOp 関数では、複合式を使用できます。
複合式では、ある MatrixOp 演算子または関数の結果を別の演算子または関数の入力として渡します。
例えば：

```
MatrixOp/O wave1 = sum(abs(wave2-wave3))
```

これは特に変換やフィルタリングにおいて便利です。

```
MatrixOp/O wave1 = IFFT(FFT(wave1,2)*filterWave,3)
```

一部の MatrixOp 関数（例：beam や transposeVol）は複合式をサポートしていません。
これらは 2 次元以上の入力が必要であるのに対し、複合式はレイヤーごとに評価されるためです。

```
Make/O/N=(10,20,30) wave3
MatrixOp/O wave1 = beam(wave3+wave3,5,5)       // エラー：不正な MatrixOp トークン
MatrixOp/O wave1 = beam(wave3,5+1,5)           // ここでは複合式が許される
```

MatrixOp 四元数データトークン

四元数 {w, x, y, z} は、振幅 w と三つのベクトル成分 x, y, z から構成され、以下の関係を満たします。

```
w = a, x = bi, y = cj, z = dk
```

a、b、c、d は実数です。
i、j、k は基本四元数単位であり、複素数における i に似ています。

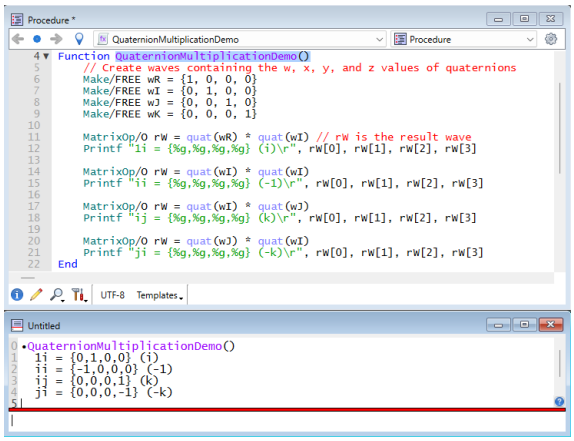
四元数は 3D グラフィックスにおける回転の表現によく用いられます。
Igor Pro 8 で MatrixOp 関数を通じて四元数のサポートが追加されました。

四元数の演算には独自の規則があります。
例えば、乗算は可換ではありません（ij は ji に等しくない）。

MatrixOp を使って四元数演算を実行するには、四元数トークンを作成し操作します。
例えば：

```
Function QuaternionMultiplicationDemo()
	// 四元数の w、x、y、z 値を含むウェーブを生成
	Make/FREE wR = {1, 0, 0, 0}
	Make/FREE wI = {0, 1, 0, 0}
	Make/FREE wJ = {0, 0, 1, 0}
	Make/FREE wK = {0, 0, 0, 1}

	MatrixOp/O rW = quat(wR) * quat(wI)		// rW は結果ウェーブ
	Printf "1i = {%g,%g,%g,%g} (i)¥r", rW[0], rW[1], rW[2], rW[3]

	MatrixOp/O rW = quat(wI) * quat(wI)
	Printf "ii = {%g,%g,%g,%g} (-1)¥r", rW[0], rW[1], rW[2], rW[3]

	MatrixOp/O rW = quat(wI) * quat(wJ)
	Printf "ij = {%g,%g,%g,%g} (k)¥r", rW[0], rW[1], rW[2], rW[3]

	MatrixOp/O rW = quat(wJ) * quat(wI)
	Printf "ji = {%g,%g,%g,%g} (-k)¥r", rW[0], rW[1], rW[2], rW[3]
End
```

この例では、wR、wI、wJ、wK は、四元数の w、x、y、z 値を含むウェーブです。
rW は MatrixOp からの結果ウェーブです。
quat(wR)、quat(wI)、quat(wJ) は MatrixOp 四元数トークンです。
これらは MatrixOp コマンドの実行中にのみ存在します。
MatrixOp は四元数演算を使って、四元数トークンを含む式を評価します。
右辺が評価された後、MatrixOp は結果をウェーブに格納します。
quat や matrixToQuat など、四元数座標を返す関数は、w、x、y、z 値を含む 4 要素のウェーブを返します。

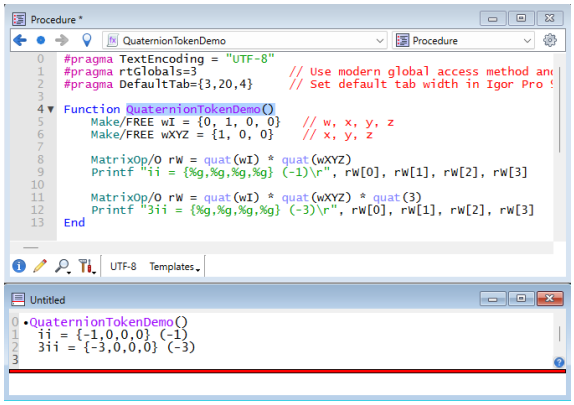


スカラーまたは x、y、z 値を含むウェーブから四元数トークンを作成することもできます。

```
Function QuaternionTokenDemo()
	Make/FREE wI = {0, 1, 0, 0}		// w, x, y, z
	Make/FREE wXYZ = {1, 0, 0}		// x, y, z

	MatrixOp/O rW = quat(wI) * quat(wXYZ)
	Printf "ii = {%g,%g,%g,%g} (-1)¥r", rW[0], rW[1], rW[2], rW[3]
```

```
        MatrixOp/O rW = quat(wI) * quat(wXYZ) * quat(3)
        Printf "3ii = {%g,%g,%g,%g} (-3)¥r", rW[0], rW[1], rW[2], rW[3]
End
```

quat(wXYZ) は、x、y、z を含むウェーブを、w 成分が 0 の純虚数四元数に変換します。
wXZY は、例えばこの例のように 3x1 のウェーブでも、1x3 のウェーブでもかまいません。
どちらも同じ四元数トークンを生成します。



quat(3) はスカラー値 3 を、x, y, z 成分が 0 である実数四元数に変換します。

quat への入力は複素数ではなく実数でなければなりません。

quat 関数は四元数を正規化しません。
別の MatrixOp 四元数演算への入力として使われた時に正規化されます。

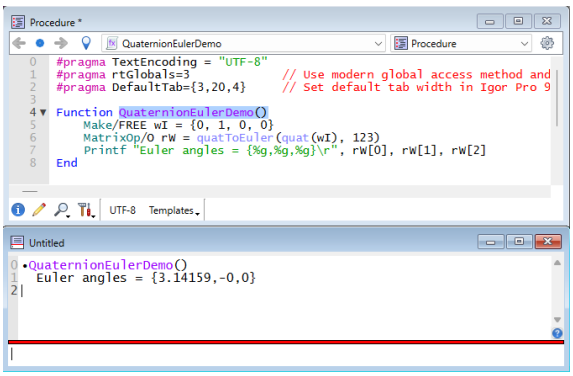
MatrixOp は次の四元数関数をサポートします：

quat、quatToMatrix、quatToAxis、axisToQuat、quatToEuler、slerp

これらは MatrixOp コマンドのヘルプに記載されています。

次の例は、四元数からオイラー角を生成するために quatToEuler を使う方法を示しています。

```
Function QuaternionEulerDemo()
        Make/FREE wI = {0, 1, 0, 0}
        MatrixOp/O rW = quatToEuler(quat(wI), 123)
        Printf "Euler angles = {%g,%g,%g}¥r", rW[0], rW[1], rW[2]
End
```



MatrixOp のマルチスレッド処理

一般的な CPU は複数のスレッドを実行できます。
一部の計算は並列実行に適しています。
MatrixOp を使ってスレッド処理を活用する方法はいくつかあります。

- ユーザー作成のプリエンプティブスレッド
  MatrixOp はスレッドセーフであるため、プリエンプティブスレッドから呼び出すことができます。詳細はヘルプ ThreadSafe Functions and Multitasking を参照してください。

- レイヤースレッド
  複数のレイヤーを含む式を評価する場合、/NTHR フラグを使って各レイヤーの計算を別々のスレッドで実

行できます。スレッドのオーバーヘッドを考慮すると、レイヤーごとの計算が 100 万 CPU サイクル以上かかる場合には /NTHR を使うことが合理的です。

- コマンドまたは関数の内部マルチスレッディング
  一部の MatrixOp 関数は、SP および DP データに対して自動的にマルチスレッディングされます。これには行列乗算、三角関数、hypotex、sqrt、erf、erfc、inverseErf、inverseErfc が含まれます。
  MultiThreadingControl コマンドは自動マルチスレッディングの微調整を提供しますが、通常はこれを弄る必要はありません。

MatrixOp のパフォーマンス

ほとんどの場合、MatrixOp はウェーブ代入や FastOp よりも高速です。
ただし、小さなウェーブの場合、追加のオーバーヘッドにより遅くなる可能性があります。

MatrixOp は浮動小数点データ型で最速に動作します。
最高速度を得るには、MatrixOp を呼び出す前に整数ウェーブを単精度浮動小数点に変換してください。

一部の MatrixOp 式は自動的にマルチスレッドで評価されます。
詳細は「MatrixOp のマルチスレッド処理」のセクションを参照してください。

MatrixOp 最適化の例

このセクションでは、MatrixOp を使ってパフォーマンスを向上させる例を示します。

- 行列操作コードを MatrixOp 呼び出しに置き換えます。
  例えば、

```
Make/O/N=(vecSize,vecSize) identityMatrix = p==q ? 1 : 0
MatrixMultiply matB, matC
identityMatrix -= M_Product
MatrixMultiply identityMatrix, matD
MatrixInverse M_Product
Rename M_Inverse, matA
```

  を次に置き換えます：

```
MatrixOp matA = Inv((Identity(vecSize) - matB x matC) x matD)
```

- ウェーブフォーム代入ステートメントを MatrixOp 呼び出しに置き換えます。
  例えば、

```
Duplicate/O wave2,wave1
wave1 = wave2*2
```

  を次に置き換えます：

```
MatrixOp/O wave1 = wave2*2
```

- 重複する部分式は一度だけ因数分解し計算します。
  例えば、

```
MatrixOp/O wave1 = var1*wave2*wave3
```

```
MatrixOp/O wave4 = var2*wave2*wave3
```

を次に置き換えます：

```
MatrixOp/O/FREE tmp = wave2*wave3  // 積は一度だけ計算する
MatrixOp/O wave1 = var1*tmp
MatrixOp/O wave4 = var2*tmp
```

- ウェーブの代入における ?: 条件演算子のインスタンスを MatrixOp 呼び出しに置き換えます。
  例えば、

```
wave1 = wave1[p]==0 ? NaN : wave1[p]
```

を次に置き換えます：

```
MatrixOp/O wave1 = setNaNs(wave1,equal(wave1,0))
```

カテゴリ別 MatrixOp 関数

このセクションでは、適切な関数を選択するのに役立つよう、MatrixOp 関数をカテゴリ別に一覧表示します。

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
```

四元数（クオータニオン）

```
quat              axisToQuat      quatToAxis      matrixToQuat    quatInverse

quatFromSpherical   quatToMatrix  slerp
```

疎行列

一部のアプリケーションでは、要素の大部分が 0 である大規模な行列の操作が必要となります。
こうしたアプリケーションでは、疎行列を使うことでパフォーマンスが向上し、メモリ使用量が削減されます。

Igor は、Igor Pro 9.0 で追加された MatrixSparse コマンドを通じて疎行列をサポートします。
これは Intel Math Kernel Library Sparse BLAS ルーチンを使い、ライブラリの用語と規約を採用しています。

疎行列の概念

このセクションでは、Igor に適用される疎行列の基本概念について説明します。
疎行列に関する一般的な入門については、https://en.wikipedia.org/wiki/Sparse_matrix を参照してください。

Igor における通常の行列は 2D ウェーブであり、行列の各要素はメモリ内に規則的な行と列のパターンで格納されます。
疎行列の文脈では、通常の行列を指すために「密行列」という用語を使います。

Igor における疎行列は、行列の非ゼロ要素を定義する 3 つの 1D ウェーブの集合によって表現されます。
Igor は疎行列の表現のために、以下に説明する 3 つの形式をサポートしています。
これらの形式は COO（座標形式）、CSC（圧縮列形式）、CSR（圧縮行形式）と呼ばれます。
以下のセクションでは、「疎行列」という用語を、これらの形式のいずれかに従って 3 つの 1D ウェーブによって定義される行列を意味するものとして使います。

MatrixSparse の TOCOO、TOCSR、TOCSC 変換コマンドを使って、密行列に相当する疎行列を作成できます。
または、 3 つの 1D ウェーブを作成し、適切な値をそれらに格納することで直接作成することもできます。
TODENSE 変換コマンドを使って、疎行列に相当する密行列を作成できます。

MatrixSparse は、ADD（2 つの疎行列の加算）、MV（疎行列とベクトルの乗算）、SMSM（2 つの疎行列の乗算）、TRSV（連立一次方程式の解法）など、数多くの数学演算をサポートしています。

疎行列演算は、単精度および倍精度の実数および複素数データに対して動作します。
INF や NaN はサポートしていません。

疎行列のフォーマット

Igor における疎行列は、行列の非ゼロ要素を定義する 3 つの 1D ウェーブの集合で表現されます。
Igor は COO（座標形式）、CSR（圧縮行形式）、CSC（圧縮列形式）という 3 つの疎行列保存形式をサポートしています。

MatrixSparse は、形式間の変換を行う操作を除き、CSR 形式を使います。

これらの形式を説明するために、Wikipedia の疎行列ウェブページにある例示用の密行列を使います。

```
0  0  0  0
5  8  0  0
0  0  3  0
0  6  0  0
```

nnz は「非ゼロ値の数（number of non-zero values）」の略語であり、以下および MatrixSparse コマンドのドキュメントに記載されています。
この例では、nnz = 4 です。

COO 疎行列ストレージフォーマット

「COO」は「座標フォーマット（coordinate format）」の略称です。
概念的には最も単純なフォーマットであり、非ゼロの行列値を、対応するゼロベースの行と列のインデックスと共に格納します。

Igor の用語では、COO フォーマットは以下の 3 つのウェーブを使います。

W_COOValues :　　行列内の各非ゼロ値を格納します。

W_COORow　　行列内の各非ゼロ値に対応するゼロベースの行インデックスを格納します。

W_COOColumns :　　行列内の各非ゼロ値に対応するゼロベースの列インデックスを格納します。

例の行列は COO フォーマットで次のように表わされます。

```
W_COOValues:        5  8  3  6
W_COORows:          1  1  2  3
W_COOColumns:       0  1  2  1
```

これらの値は、順序付きトリプレットの集合として縦方向に読み取ることができます：(5,1,0)、(8,1,1)、(3,2,2)、(6,3,1)。
最後のものは、値 6 が 3 行目 1 列目の値であることを示しています。

ウェーブの名前 W_COOValues、W_COORows、W_COOColumns は、MatrixSparse が COO フォーマットで出力疎行列を作成する時に使われます。
入力疎行列を指定する場合、任意のウェーブの名前を使用できます。

CSR 疎行列ストレージフォーマット

「CSR」は「圧縮疎行列（compressed sparse row）」の略称です。
メモリ使用量と計算速度の面で COO よりも効率的であるため、広く利用されています。

MatrixSparse は、フォーマット間の変換を行う操作を除き、CSR フォーマットを使います。

CSR フォーマットでは、３つの 1D ウェーブが非ゼロ値、ゼロベースの列インデックス、および各値がどの行に現れるかを決定するために使われる「ポインター」ベクトルを格納します。

Igor の用語では、CSR フォーマットは以下の３つのウェーブを使います。

W_CSRValues：　　行列内の各非ゼロ値を格納します。

W_CSRColumns：　　行列内の各非ゼロ値に対応するゼロベースの列インデックスを格納します。

W_CSRPointerB：　　特定の値がどの行に存在するかを決定するために使われる W_CSRValues へのインデックスを格納します。

例の行列は CSR フォーマットで以下のように表わされます。

```
W_CSRValues:        5  8  3  6
W_CSRColumns:       0  1  2  1
W_CSRPointerB:      0  2  3  4
```

COO フォーマットの場合とは異なり、これらの値は順序付きトリプレットの集合として縦方向に読み取ることができません。
CSR はもう少し複雑です。

これらのウェーブのうち最初の２つは順序対として読み取れます：(5,0)、(8,1)、(3,2)、(6,1)。
各順序対は値（例：6）と列（例：1）を指定しますが、その値がどの行に現れるかは示しません。

第３のウェーブ、W_CSRPointerB は、与えられた値がどの行に現れるかを決定するために使われます。
これは、表現される行列の各行に対する１つのインデックスと、行列内の非ゼロ値の数である nnz という追加のインデックスを含みます。
W_CSRPointerB[i] は、行 I の最初の非ゼロ値の W_CSRValues におけるゼロベースのインデックスです。

この例では、W_CSRPointerB の値を次のように解釈できます。

```
W_CSRPointerB[0] = 0        // 行 0 の最初の値は W_CSRValues のインデックス 0 に位置する *
W_CSRPointerB[1] = 0        // 行 1 の最初の値は W_CSRValues のインデックス 0 に位置する
W_CSRPointerB[2] = 2        // 行 2 の最初の値は W_CSRValues のインデックス 2 に位置する
W_CSRPointerB[3] = 3        // 行 3 の最初の値は W_CSRValues のインデックス 3 に位置する
W_CSRPointerB[4] = 4        // W_CSRValues 内の非ゼロ値の数は 4
```

* 行 0 には非ゼロ値が存在しないため、W_CSRPointerB[0] は W_CSRPointerB[1] と同じです。

MatrixSparse コマンドに CSR フォーマットの疎行列を指定する場合、nnz を指定する W_CSRPointerB の最後の要素はオプションであり、省略可能です。

ウェーブ名 W_CSRValues、W_CSRColumns、W_CSRPointerB は、MatrixSparse が CSR フォーマットで出力疎行列を作成する時に使われます。
入力疎行列を指定する場合、任意のウェーブ名を使用できます。

CSC 疎行列ストレージフォーマット

「CSC」は「圧縮疎列（compressed sparse column）」の略称です。
メモリ使用量と計算速度の面で、COO よりも効率的です。

Igor の用語では、CSC フォーマットは以下の３つのウェーブを使います。

W_CSCValues：　　行列内の各非ゼロ値を格納します。

W_CSCRows：　　行列内の各非ゼロ値に対応するゼロベースの行インデックスを格納します。

W_CSCPointerB： 特定の値がどの列に存在するかを決定するために使われる、W_CSCValues へのインデックスを格納します。

W_CSCPointerB ウェーブは、CSC において、CSR における W_CSRPointerB と同様の動作をします。

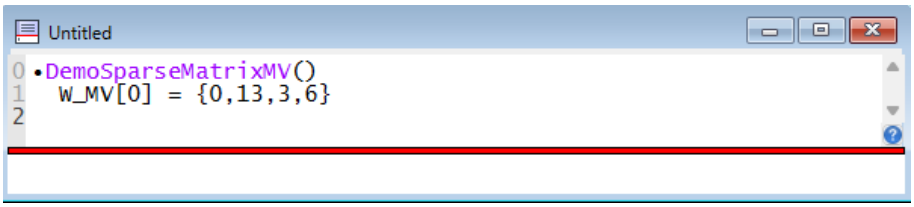
MatrixSparse コマンドに CSC フォーマットの疎行列を指定する場合、nnz を指定する W_CSCPointerB の最後の要素はオプションであり、省略可能です。

ウェーブの名前 W_CSCValues、W_CSCRows、W_CSCPointerB は、MatrixSparse が CSC フォーマットで出力疎行列を作成する時に使われます。
入力疎行列を指定する場合、任意のウェーブの名前を使用できます。

疎行列の例

MatrixSparse コマンドの動作を理解していただくために、MatrixSparse MV コマンドを使った疎行列とベクトルの乗算を示す簡単な例を以下に示します。

```
Function DemoSparseMatrixMV()
        // CSR フォーマットで Wikipedia の例の疎行列を定義
        Make/FREE/D values = {5, 8, 3, 6}       // 倍精度浮動小数点
        Make/FREE/L columns = {0, 1, 2, 1}      // 64-bit 符号あり整数
        Make/FREE/L ptrB = {0, 0, 2, 3, 4}      // 64-bit 符号あり整数

        // ベクトルを作成
        Make/FREE/D vector = {1, 1, 1, 1}      // 倍精度浮動小数点

        // 疎行列をベクトルで乗算
        MatrixSparse rowsA=4, colsA=4, csrA={values,columns,ptrB}, vectorX=vector, operation=MV

        // 出力疎行列用のウェーブ参照を作成
        WAVE W_MV      // MatrixSparse コマンド MV からの出力

        Print W_MV     // W_MV[0]= {0,13,3,6} を出力
End
```



先頭にある 3 つの Make コマンドは、自由ウェーブを使って CSR フォーマットの疎行列を定義します。
ウェーブの値は、単精度または倍精度浮動小数点、実数または複素数で、INF や NaN を含んではなりません。
この場合、インデックス、列、ptrB を含むウェーブは、64 ビットの符号付き整数でなければなりません。

次の Make コマンドは、values ウェーブと同じデータ型を持つベクトルを作成します。

疎入力行列は、rowsA、colsA、csrA キーワードによって定義されます。

入力ベクトルは vectorX キーワードで指定されます。

コマンドキーワードは実行する操作を指定します。
この場合は MV です。

この場合の出力を、現在のディレクトリに作成される W_MV という名前のウェーブとします。
これは values ウェーブと同じデータ型のベクトルです。

異なる MatrixSparse コマンドは、異なる入力を必要とし、異なる出力を生成します。

MatrixSparse コマンドのリスト

MatrixSparse がサポートするコマンドは以下の通りです。

コマンド	動作
ADD	２つの疎行列を加算し、疎な出力行列を生成します。 詳細は MatrixSparse ADD のヘルプを参照してください。
MM	疎行列と密行列の積を計算し、密行列を出力行列として生成します。 詳細は MatrixSparse MM のヘルプを参照してください。
MV	疎行列とベクトルの積を計算し、疎な出力行列を生成します。 詳細は MatrixSparse MV のヘルプを参照してください。
SMSM	２つの疎行列の積を計算し、疎な出力行列を生成します。 詳細は MatrixSparse SMSM のヘルプを参照してください。
TOCOO	入力行列（密行列、CSC フォーマット、または CSR フォーマット）に対応する COO フォーマットの疎な出力行列を生成します。 詳細は MatrixSparse TOCOO のヘルプを参照してください。
TOCSC	入力行列（密行列、COO フォーマット、または CSR フォーマット）に対応する CSC フォーマットの疎な出力行列を生成します。 詳細は MatrixSparse TOCSC のヘルプを参照してください。
TOCSR	入力行列（密行列、COO フォーマット、または CSC フォーマット）に対応する CSR フォーマットの疎な出力行列を生成します。 詳細は MatrixSparse TOCSR のヘルプを参照してください。
TODENSE	疎な入力行列（COO、CSC、または CSR フォーマット）に相当する密な出力行列を生成します。 詳細は MatrixSparse TODENSE のヘルプを参照してください。
TRSV	三角疎行列入力に対する連立一次方程式を解きます。 詳細は MatrixSparse TRSV のヘルプを参照してください。

MatrixSparse の入力

MatrixSparse コマンドへの入力は文書化されており、以下の概念的な行列、ベクトル、スカラー入力として理解できます。

Sparse matrix A は、rowsA と colsA キーワードと、cooA、csrA または cscA キーワードのいずれかによって定義されます。
上記の DemoSparseMatrixMV 例におけるこのコマンドでは、Sparse matrix A を指定するキーワードが鍵となります。

```
MatrixSparse rowsA=4, colsA=4, csrA={values,columns,ptrB}, vectorX=vector, operation=MV
```

Sparse matrix A は、１つ以上の疎行列入力を取るすべての MatrixSparse コマンドで使われます。
MatrixSparse 数学コマンド（ADD、MV、MM、SMSM、TRSV）では、入力疎行列が CSR フォーマットである必要があります。

Sparse matrix G は、rowsG と colsG キーワードと、cooG、csrG または cscG キーワードのいずれかによって定義されます。

Sparse matrix G は、２つの疎行列入力を取る MatrixSparse コマンド（ADD と SMSM）でのみ使われます。
これらのコマンドでは、入力疎行列が CSR フォーマットであることが必要です。

Matrix B は matrixB キーワードで定義され、１つ以上の密行列入力を取る MatrixSparse コマンド（現時点では MM コマンドのみ）で使われます。

Matrix C は matrixC キーワードで定義され、２つの密行列入力を取る MatrixSparse コマンド（現時点では MM コマンドのみ）で使われます。

Vector X は vectorX キーワードで定義され、１つ以上のベクトル入力を取る MatrixSparse コマンド（現時点では MM と TRSV コマンド）で使われます。

Vector Y は vector キーワードで定義され、２つのベクトル入力を取る MatrixSparse コマンド（現時点では MM コマンドのみ）で使われます。

Alpha と ***Alphai*** は、alpha と alphai キーワードで定義され、１つ以上のスカラー入力を取るすべての MatrixSparse コマンド（現時点では MM、MV、TRSV コマンド）で使われます。
Alphai は、複素数データに対する演算時にのみ使われます。

Beta と ***Betai*** は、beta と betai キーワードで定義され、２つのスカラー入力を取る MatrixSparse コマンド（現時点では MM と MV コマンド）で使われます。
Betai は、複素数データに対する演算時にのみ使われます。

MatrixSparse コマンドのデータ形式

コマンドのデータ形式は、MatrixSparse コマンドに使われるすべてのデータウェーブに必要なデータ形式です。
ここで「データウェーブ」とは、疎行列の表現における値ウェーブと、密行列を表す行列ウェーブを指します。

特定の MatrixSparse コマンドが sparse matrix A を入力として受け取る場合、ウェーブ（cooA、cscA、または csrA キーワードで指定される最初のウェーブ）が演算のデータ形式を決定します。

複数の入力データウェーブがある場合（例：２つの疎行列を加算する ADD コマンド）、すべての入力データウェーブのデータ形式は同一でなければなりません。

コマンドのデータ形式は単精度または倍精度の浮動小数点型でなければならず、実数または複素数であることができます。
MatrixSparse は　NaN または INF を含むウェーブをサポートしません。

出力ウェーブはコマンドのデータ形式を使って生成されます。

MatrixSparse の数学コマンド（ADD、MV、MM、SMSM、TRSV）では、入力疎行列が CSR フォーマットである必要があります。
疎行列を返す数学コマンド（ADD、SMSM）は、CSR フォーマットの出力疎行列を作成します。

変換コマンド（TOCOO、TOCSC、TOCSR、TODENSE）は、COO、CSC、CSR、または密行列形式での入力を受け付けます。

MatrixSparse インデックスのデータ形式

行または列のインデックス、あるいは値ウェーブへのインデックスを含むインデックスウェーブ（つまりポインターウェーブ）は、符号付き 64 ビット整数ウェーブである必要があり、通常は Make/L を使って作成されます。

MatrixSparse 変換

入力疎行列の変換済みバージョンに対して MatrixSparse を動作させるよう指定することも可能です。
利用可能な変換は、転置を表す T、エルミート変換を表す H、変換なし（デフォルト）を表す N で名付けられています。

opA キーワードは、MatrixSparse が Sparse matrix A の変換済みバージョンに対して操作を行うよう指示します。
例えば次のコマンド：

```
MatrixSparse rowsA=4, colsA=4, csrA={values,columns,ptrB}, opA=T, vectorX=vector, operation=MV
```

は、Sparse matrix A の転置済みバージョンに対して動作します。

opG キーワードは、MatrixSparse が Sparse matrix G の変換済みバージョンに対して操作を行うよう指示します。

オプションの疎行列に関する情報

MatrixSparse の sparseMatrixType キーワードを使うと、疎行列入力の特徴を表すオプション情報を指定できます。
疎行列入力の特性がわかっている場合は、sparseMatrixType を使ってこの情報を MatrixSparse に渡すことができます。
これにより、パフォーマンスが向上する場合があります。

sparseMatrixType キーワードの構文は次のとおりです：

```
sparseMatrixType={smType,smMode,smDiag}
```

すべてのパラメーターはキーワードです。

smType: GENERAL, SYMMETRIC, HERMITIAN, TRIANGULAR, DIAGONAL, BLOCK_TRIANGULAR, BLOCK_DIAGONAL

smMode: LOWER, UPPER

smDiag: DIAG, NON_DIAG

MatrixSparse コマンド

このセクションでは、MatrixSparse がサポートする各コマンドについて説明します。
疎行列に関する背景資料を読み、理解していることを前提とします。

以下のセクションでは、次の略語を使います：

シンボル	**意味**	**キーワードでの指定**
smA	Sparse matrix A	rowsA, colsA, csrA (1)
smG	Sparse matrix G	rowsG, colsG, csrG (1)
dmB	Dense matrix B	matrixB

dmC	Dense matrix C	matrixC
vX	Vector X	vectorX
vY	Vector Y	vectorY
alpha	スカラー値 alpha	alpha, 複素数入力に対しては alphai
beta	スカラー値 beta	beta, 複素数入力に対しては betai
smOut	出力疎行列	N/A (2)

(1) 行列形式変換コマンド TOCOO、TOCSC、TOCSR、TODENSE では、入力疎行列を COO フォーマットまたは CSC フォーマットで指定するために cooA および cscA キーワードも使用できます。
その他のすべてのコマンドでは、入力行列を CSR フォーマットで指定するために csrA および csrG を使う必要があります。

(2) 出力疎行列 smOut は、CSR フォーマットでウェーブ W_CSRValues、W_CSRColumns、W_CSRPointerB によって表現されます。

MatrixSparse ADD

ADD は Sparse matrix A と Sparse matrix G の和を計算します。
G は CSR フォーマットでなければなりません。
記号的には：

```
smOut = smA + smG
```

入力：CSR フォーマットの Sparse matrix A と Sparse matrix G

出力：CSR フォーマットの疎行列（W_CSRValues、W_CSRColumns、W_CSRPointerB で表現）

MatrixSparse ADD の例

```
Function DemoMatrixSparseADD()
        // CSR フォーマットで smA を作成
        Make/FREE/D/N=(11) valuesA = {1,25,26,44,16,22,28,5,11,36,42} // 倍精度浮動小数点
        Make/FREE/L/N=(11) columnsA = {0,4,4,7,2,3,4,0,1,5,6}         // 64-bit 符号付き整数
        Make/FREE/L/N=(6) ptrBA = {0,2,4,4,7,9}                       // 64-bit 符号付き整数

        // CSR フォーマットで smG を作成
        Make/FREE/D/N=(3) valuesG ={1,2,3}                            // 倍精度浮動小数点
        Make/FREE/L/N=(3) columnsG ={0,1,2}                           // 64-bit 符号付き整数
        Make/FREE/L/N=(4) ptrBG = {0,1,2,3,3,3,3,3,3}                 // 64-bit 符号付き整数

        // smA + smG を計算
        MatrixSparse rowsA=6, colsA=8, csrA={valuesA,columnsA,ptrBA}, rowsG=6, colsG=8,
csrG={valuesG,columnsG,ptrBG}, operation=ADD
        WAVE W_CSRValues, W_CSRColumns, W_CSRPointerB  // MatrixSparse コマンド ADD からの出力

        // 出力 CSR 疎行列を表す 1D ウェーブを出力
        Print W_CSRValues
        Print W_CSRColumns
        Print W_CSRPointerB
End
```

MatrixSparse MM

MM は疎行列と密行列の積を計算します。
記号的には：

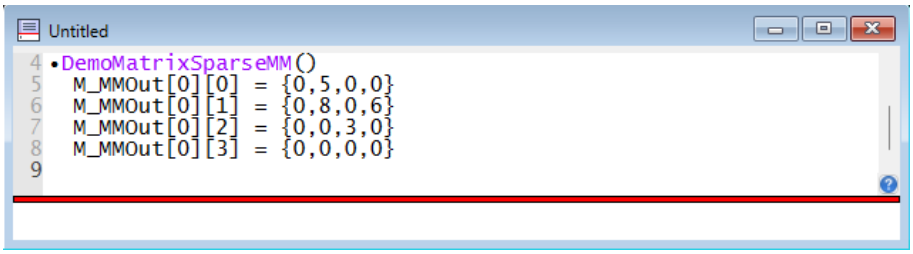
```
M_MMOut = alpha*smA*dmB + beta*dmC
```

入力：alpha、CSR フォーマットの Sparse matrix A、Dense matrix B、オプションで beta と Dense matrix C
beta キーワードを省略して beta をデフォルト値 0 のままにすると、beta*dmC 項は計算されないため、dmC 入力を指定する必要はありません。

出力：密行列 M_MMOut

MatrixSparse MM の例

```
Function DemoMatrixSparseMM()
	// CSR フォーマットで疎行列を定義
	Make/FREE/D values = {5, 8, 3, 6}		// 倍精度浮動小数点
	Make/FREE/L columns = {0, 1, 2, 1}		// 64-bit 符号付き整数
	Make/FREE/L ptrB = {0, 0, 2, 3, 4}		// 64-bit 符号付き整数

	// 密行列を作成
	Make/FREE/D matrix = { {1,0,0,0}, {0,1,0,0}, {0,0,1,0}, {0,0,0,1} }	// 倍精度浮動小数点

	// 疎行列を密行列で乗算
	MatrixSparse rowsA=4, colsA=4, csrA={values,columns,ptrB}, matrixB=matrix, operation=MM

	// 出力密行列に対するウェーブ参照を作成
	WAVE M_MMOut					// MatrixSparse コマンド MM からの出力

	Print M_MMOut
End
```



MatrixSparse MV

CSR フォーマットでなければならない疎行列とベクトルの積を計算し、出力ベクトルを生成します。
記号的には：

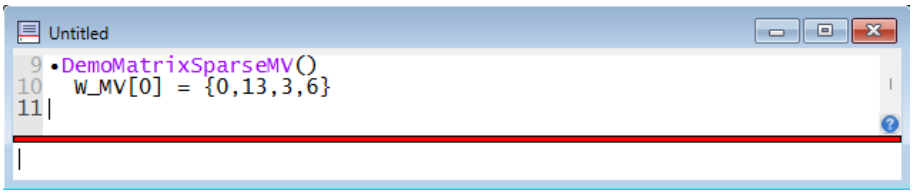
```
W_MV = alpha*smA*vX + beta*vY
```

入力：alpha、CSR フォーマットの Sparse matrix A、Vector X、オプションで beta と Vector Y
beta キーワードを省略して beta をデフォルト値 0 のままにすると、beta*vY 項は計算されないため、vY 入力を指定する必要はありません。

出力：ベクトル W_MV

MatrixSparse MV の例

```
Function DemoMatrixSparseMV()
        // CSR フォーマットで疎行列を定義
        Make/FREE/D values = {5, 8, 3, 6}              // 倍精度浮動小数点
        Make/FREE/L columns = {0, 1, 2, 1}             // 64-bit 符号付き整数
        Make/FREE/L ptrB = {0, 0, 2, 3, 4}             // 64-bit 符号付き整数

        // ベクトルを作成
        Make/FREE/D vector = {1, 1, 1, 1}              // 倍精度浮動小数点

        // 疎行列をベクトルで乗算
        MatrixSparse rowsA=4, colsA=4, csrA={values,columns,ptrB}, vectorX=vector, operation=MV

        // 出力ベクトルに対するウェーブ参照を作成
        WAVE W_MV                                  // MatrixSparse コマンド MV からの出力

        Print W_MV
End
```



MatrixSparse SMSM

SMSM は２つの疎行列の積を計算します。
記号的には：

```
        smOut = smA * smG
```

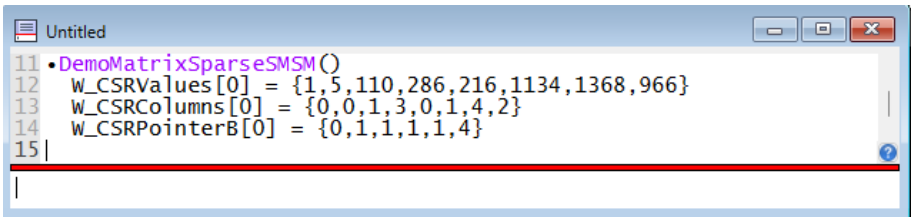
入力：CSR フォーマットの Sparse matrix A、CSR フォーマットの Sparse matrix G

出力：W_CSRValues、W_CSRColumns、W_CSRPointerB で表される CSR フォーマットの疎行列

MatrixSparse SMSM の例

```
Function DemoMatrixSparseSMSM()
        // CSR フォーマットで sparse matrix A を作成
        Make/FREE/D/N=(11) valuesA = {1,25,26,44,16,22,28,5,11,36,42} // 倍精度浮動小数点
        Make/FREE/L/N=(11) columnsA = {0,4,4,7,2,3,4,0,1,5,6}         // 64-bit 符号付き整数
        Make/FREE/L/N=(6) ptrBA = {0,2,4,4,7,9}                       // 64-bit 符号付き整数

        // CSR フォーマットで Sparse matrix G を作成
        Make/FREE/D/N=(8) valuesG = {1,10,26,6,14,38,15,23}           // 倍精度浮動小数点
        Make/FREE/L/N=(8) columnsG = {0,1,3,0,1,4,1,2}                // 64-bit 符号付き整数
        Make/FREE/L/N=(8) ptrBG = {0,1,3,3,3,3,6,8}                   // 64-bit 符号付き整数

        // MatrixA x MatrixG を計算
        MatrixSparse rowsA=6, colsA=8, csrA={valuesA,columnsA,ptrBA}, rowsG=8, colsG=5,
```

```
csrG={valuesG,columnsG,ptrBG}, operation=SMSM
        WAVE W_CSRValues, W_CSRColumns, W_CSRPointerB  // MatrixSparse コマンド SMSM からの出力

        // 出力 CSR 疎行列を表す 1D ウェーブを出力
        Print W_CSRValues
        Print W_CSRColumns
        Print W_CSRPointerB
End
```



MatrixSparse TOCOO

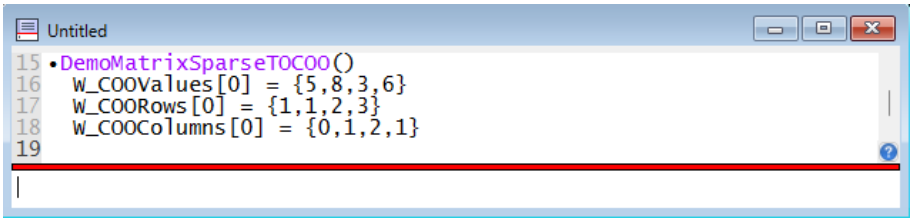
TOCOO は、入力行列（密行列、CSC フォーマット、または CSR フォーマットのいずれか）に対応する COO フォーマットの疎な出力行列を生成します。

入力：matrixB キーワードで指定される密行列、または cscA または csrA キーワードで指定される疎行列

出力：W_COOValues、W_COORows、W_COOColumns で表される COO フォーマットの疎行列

MatrixSparse TOCOO の例

```
Function DemoMatrixSparseTOCOO()
        // CSR フォーマットで Wikipedia の例の 4x4 行列を作成
        Make/FREE/D values = {5, 8, 3, 6}              // 倍精度浮動小数点
        Make/FREE/L columns = {0, 1, 2, 1}             // 64-bit 符号付き整数
        Make/FREE/L ptrB = {0, 0, 2, 3, 4}             // 64-bit 符号付き整数

        // CSR 行列から COO フォーマットの疎行列を作成
        MatrixSparse rowsA=4, colsA=4, csrA={values,columns,ptrB}, operation=TOCOO
        WAVE W_COOValues, W_COORows, W_COOColumns      // MatrixSparse コマンド TOCOO からの出力

        // COO 疎行列を表す 1D ウェーブを出力
        Print W_COOValues
        Print W_COORows
        Print W_COOColumns
End
```



MatrixSparse TOCSC

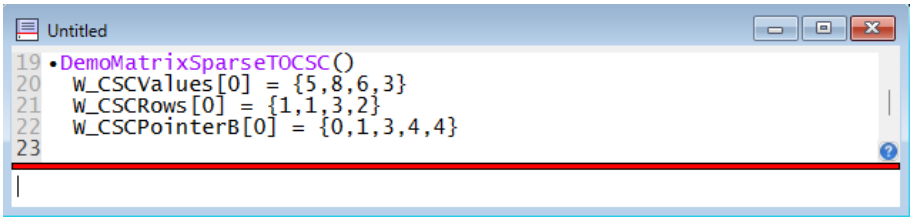
TOCSC は、入力行列（密行列、COO フォーマット、または CSR フォーマットのいずれか）に相当する CSC フォーマットの疎な出力行列を生成します。

入力：matrixB キーワードで指定される密行列、または cooA または csrA キーワードで指定される疎行列

出力：W_CSCValues、W_CSCRows、W_CSCPointerB で表される CSC フォーマットの疎行列

MatrixSparse TOCSC の例

```
Function DemoMatrixSparseTOCSC()
        // 密なフォーマットで Wikipedia の例の 4x4 行列を作成
        Make/FREE/D/N=(4,4) dense                // 倍精度浮動小数点
        dense[0][0] = {0,5,0,0}
        dense[0][1] = {0,8,0,6}
        dense[0][2] = {0,0,3,0}
        dense[0][3] = {0,0,0,0}

        // 密行列から CSC フォーマットの疎行列を作成
        // MatrixSparse は、このケースでは重要ではないにもかかわらず、rowsA と colsA を要求する
        MatrixSparse rowsA=4, colsA=4, matrixB=dense, operation=TOCSC
        WAVE W_CSCValues, W_CSCRows, W_CSCPointerB     // MatrixSparse コマンド TOCSC からの出力

        // CSC 疎行列を表す 1D ウェーブを出力
        Print W_CSCValues
        Print W_CSCRows
        Print W_CSCPointerB
End
```



MatrixSparse TOCSR

TOCSR は、入力行列（密行列、COO フォーマット、または CSC フォーマットのいずれか）に相当する CSR フォーマットの疎な出力行列を生成します。

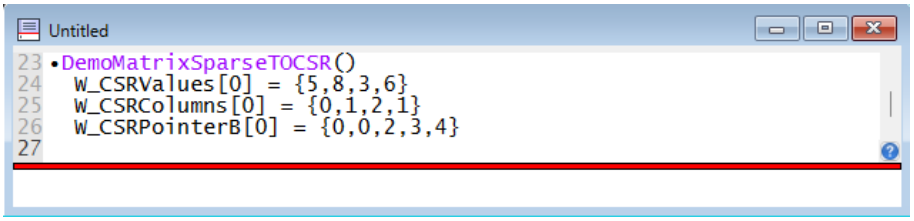
入力：matrixB キーワードで指定される密行列、または cooA または cscA キーワードで指定される疎行列

出力：W_CSRValues、W_CSRColumns、W_CSRPointerB で表される CSR フォーマットの疎行列

MatrixSparse TOCSR の例

```
Function DemoMatrixSparseTOCSR()
        // 密なフォーマットで Wikipedia の例の 4x4 行列を作成
        Make/FREE/D/N=(4,4) dense                // 倍精度浮動小数点
        dense[0][0] = {0,5,0,0}
        dense[0][1] = {0,8,0,6}
```

```
        dense[0][2] = {0,0,3,0}
        dense[0][3] = {0,0,0,0}

        // 密行列から CSC フォーマットの疎行列を作成
        // MatrixSparse は、このケースでは重要ではないにもかかわらず、rowsA と colsA を要求する
        MatrixSparse rowsA=4, colsA=4, matrixB=dense, operation=TOCSR
        WAVE W_CSRValues, W_CSRColumns, W_CSRPointerB  // MatrixSparse コマンド TOCSR からの出力

        // CSR 疎行列を表す 1D ウェーブを出力
        Print W_CSRValues
        Print W_CSRColumns
        Print W_CSRPointerB
End
```



MatrixSparse TODENSE

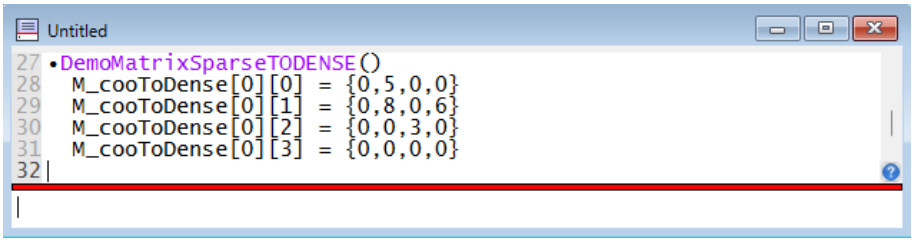
TODENSE は、COO、CSC、または CSR フォーマットの疎入力行列に相当する密出力行列を生成します。

入力：cooA、cscA、または csrA キーワードで指定される疎行列

出力：密行列 M_cooToDense、M_cscToDense、または M_csrToDense

MatrixSparse TODENSE の例

```
Function DemoMatrixSparseTODENSE()
        // COO フォーマットで Wikipedia の例の 4x4 行列を作成
        Make/FREE/D values = {5, 8, 3, 6}              // 倍精度浮動小数点
        Make/FREE/L rows = {1, 1, 2, 3}                // 64-bit 符号付き整数
        Make/FREE/L columns = {0, 1, 2, 1}             // 64-bit 符号付き整数

        // 疎行列から密行列を作成
        MatrixSparse rowsA=4, colsA=4, cooA={values,rows,columns}, operation=TODENSE
        WAVE M_cooToDense                       // MatrixSparse コマンド TODENSE からの出力

        // 密な出力行列を出力
        Print M_cooToDense
End
```



MatrixSparse TRSV

TRSV は三角疎行列 A の連立一次方程式を解きます。
記号的には出力行列 M_TRSVOut を求めます。
ここでは：

```
smA*M_TRSVOut = alpha*vX
```

入力：CSR フォーマットの Sparse matrix A、alpha、vector X

出力：密行列 M_TRSVOut

MatrixSparse TRSV の例

注記：2026 年 01 月時点のヘルプ内のコードでは {0, 0, 0} という結果となるため、一部編集が必要です。

```
// これは、Wikipedia ページの Row Reduction セクションにある例に基づいています。
//      https://en.wikipedia.org/wiki/System_of_linear_equations
// 連立方程式は次の通りです：
//              x + 3y - 2z = 5
//              3x + 5y + 6z = 7
//              2x + 47 + 3z = 8
// これにより、以下の拡張行列が得られます：
//              1       3       -2      5
//              3       5       6       7
//              2       4       3       8
// 解は： x=-15, y=8, z=2
// MatrixSparse の TRSV コマンドは三角化係数行列を必要とするため、
// Gauss-Jordan 消去法を使って得られた拡張行列の上三角版から開始します：
//              1       3       -2      5
//              0       1       -3      2
//              0       0       1       2
// MatrixSparse TOCSR を使って同等の疎行列を作成します。
// 次に、対応する解ベクトル {5, 5, 2} を作成します。
// 次に、MatrixSparse TRSV を呼び出し、解の集合{-15, 8, 2}を得ます。

Function DemoMatrixSparseTRSV()
        // 係数を表す密な上三角行列を作成
        Make/FREE/D/N=(3,3) utMat               // 倍精度浮動小数点
        utMat[0][0] = {1, 0, 0}                 // 列 0
        utMat[0][1] = {3, 1, 0}                 // 列 1
        utMat[0][2] = {-2, -3, 1}               // 列 2

        // CSR フォーマットで上三角行列の疎行列バージョンを作成
        MatrixSparse rowsA=3, colsA=3, matrixB=utMat, operation=TOCSR
        WAVE values = W_CSRValues
        WAVE columns = W_CSRColumns
        WAVE ptrB = W_CSRPointerB

        // 解ベクトルを作成
        Make/FREE/D/N=(3) vector = {5,2,2}      // 倍精度浮動小数点

        // 連立一次方程式を解く
        // sparseMatrixType={TRIANGULAR,UPPER,NON_DIAG}を追加すると、パフォーマンスが向上する可能性がある
        // この行を修正：MatrixSparse rowsA=3, colsA=3, csrA={values,columns,ptrB}, vectorX=vector,
        operation=TRSV
        MatrixSparse rowsA=3, colsA=3, csrA={values,columns,ptrB},
                sparseMatrixType={TRIANGULAR,UPPER,NON_DIAG}, vectorX=vector, operation=TRSV
        WAVE M_TRSVOut                          // MatrixSparse コマンド TRSV からの出力
```

```
        Print M_TRSVOut                         // {-15, 8, 2} となるはず
End
```