Special Functions

erf	erfc	gamma	gammaln	inverseErf	inverseErfc

Logical

equal	greater	within

Bitwise

bitAnd	bitOr	bitShift	bitXOR	bitNot

Quaternions

quat	axisToQuat	quatToAxis	matrixToQuat	quatToMatrix	slerp

Sparse Matrices

Some applications require manipulation of large matrices the elements of which are mostly 0. In these applications, the use of sparse matrices can improve performance and reduce memory utilization.

Igor supports sparse matrices through the **MatrixSparse** operation which was added in Igor Pro 9.00. It uses the Intel Math Kernel Library Sparse BLAS routines and employs the libraries terminology and conventions.

Sparse Matrix Concepts

In this section we discuss some basic sparse matrix concepts as they apply to Igor. For a general primer on sparse matrices, see <https://en.wikipedia.org/wiki/Sparse_matrix>.

A normal matrix in Igor is a 2D wave in which each element of the matrix is stored in memory in a regular pattern of rows and columns. In the context of sparse matrices, we use the term "dense matrix" to refer to a normal matrix.

A sparse matrix in Igor is represented by a set of three 1D waves which define the non-zero elements of the matrix. Igor supports three formats, described below, for sparse matrix representation. The formats are called COO (coordinate format), CSC (compressed column format), and CSR (compressed row format). In the following sections, we use the term "sparse matrix" to mean a matrix defined by three 1D waves according to one of these formats.

You can create a sparse matrix equivalent to a dense matrix using the MatrixSparse TOCOO, TOCSR, and TOCSC conversion operations. Alternatively you can create it directly by creating three 1D waves and storing the appropriate values in them. You can create a dense matrix equivalent to a sparse matrix using TODENSE conversion operation.

MatrixSparse supports a number of math operations such as ADD (add two sparse matrices), MV (multiply a sparse matrix by a vector), SMSM (multiply two sparse matrices), and TRSV (solve a system of linear equations).

Sparse matrix operations work on single-precision and double-precision real and complex data. They do not support INFs or NaNs.

Sparse Matrix Formats

A sparse matrix in Igor is represented by a set of three 1D waves which define the non-zero elements of the matrix. Igor supports three sparse matrix storage formats named COO (coordinate format), CSR (compressed row format), and CSC (compressed column format).

MatrixSparse uses the CSR format except for operations that convert between formats.

For the purpose of illustrating these formats, we will use the example dense matrix from the Wikipedia sparse matrix web page:

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

The term nnz is short for "number of non-zero values" and appears below and in the documentation for MatrixSparse. In this example, nnz = 4.

COO Sparse Matrix Storage Format

"COO" is the shorthand name for "coordinate format". It is conceptually the simplest format and stores each non-zero matrix value along with the corresponding zero-based row and column indices.

In Igor terminology, COO format uses the following three waves:

W_COOValues stores each non-zero value in the matrix.

W_COORows stores the zero-based row index for each non-zero value in the matrix.

W_COOColumns stores the zero-based column index for each non-zero value in the matrix.

Our example matrix is represented in COO format like this:

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

These values can be read vertically as a set of ordered triplets: (5,1,0), (8,1,1), (3,2,2), and (6,3,1). The last of these tells us that the value 6 is the value of row 3, column 1.

The wave names W_COOValues, W_COORows, and W_COOColumns are used by MatrixSparse when it creates an output sparse matrix in COO format. When you specify an input sparse matrix, you can use any wave names.

CSR Sparse Matrix Storage Format

"CSR" is the shorthand name for "compressed sparse row". It is widely used because it is more efficient in terms of memory use and computational speed than COO.

MatrixSparse uses the CSR format except for operations that convert between formats.

In CSR format, the three 1D waves store the non-zero values, the zero-based column indices, and a "pointer" vector which is used to determine in which row each value appears.

In Igor terminology, CSR format uses the following three waves:

W_CSRValues stores each non-zero value in the matrix.

W_CSRColumns stores the zero-based column index for each non-zero value in the matrix.

W_CSRPointerB stores indices into W_CSRValues which are used to determine in which row a particular value appears.

Our example matrix is represented in CSR format like this:

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

Unlike the case of COO format, these values can not be read vertically as a set of ordered triplets. CSR is a bit more complicated.