MatrixOp is convenient for complex math calculations because it automatically creates real or complex destination waves as appropriate. For example:

```
Make/O/C wave2, wave3                          // SP complex
Make/O wave4                                   // SP real
MatrixOp/O wave1 = wave2 * wave3 * wave4       // wave1 is SP complex
MatrixOp/O wave1 = abs(wave2*wave3) * wave4    // wave1 is SP real
```

An exception to the complex policy is the `sqrt` function which returns NaN when its input is real and negative.

If you want to restrict data type promotion you can use the /NPRM flag:

```
Make/O/B/U wave2, wave3
MatrixOp/O wave1 = wave2 * wave3    // wave1 is unsigned word (16 bit).
```

A number of MatrixOp functions convert integer tokens to SP prior to processing and are not affected by the /NPRM flag. These include all the trigonometric functions, `chirp` and `chirpZ`, `fft`, `ifft`, normalization functions, `sqrt`, `log`, `exp` and forward/backward substitutions.

MatrixOp Compound Expressions

You can use compound expressions with most MatrixOp functions that operate on regular data tokens. In a compound expression you pass the result from one MatrixOp operator or function as the input to another. For example:

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

This is particularly convenient for transformations and filtering:

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

Some MatrixOp functions, such as `beam` and `transposeVol`, do not support compound expressions because they require input that has more than two-dimensions while compound expressions are evaluated on a layer by layer basis:

```
Make/O/N=(10,20,30) wave3
MatrixOp/O wave1 = beam(wave3+wave3,5,5)  // Error: Bad MatrixOp token
MatrixOp/O wave1 = beam(wave3,5+1,5)      // Compound expression allowed here
```

MatrixOp Quaternion Data Tokens

A quaternion {w,x,y,z} consists of an amplitude w and three vector components x, y, and z where:

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

a, b, c and d are real numbers. i, j, and k are the fundamental quaternion units, analogous to i in complex numbers.

Quaternions are often used to represent rotation in 3D graphics. Igor Pro 8 added support for quaternions via MatrixOp functions.

Quaternion arithmetic has its own rules. For example, multiplication is not commutative (ij is not equal to ji).

You can do quaternion arithmetic with MatrixOp by creating and then manipulating quaternion tokens. For example:

```
Function QuaternionMultiplicationDemo()
   // Create waves containing the w, x, y, and z values of quaternions
   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 is the result wave
   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
```

In this example, wR, wI, wJ and wK are waves containing the w, x, y, and z values of quaternions. rW is the result wave from MatrixOp. quat(wR), quat(wI) and quat(wJ) are MatrixOp quaternion tokens. They exist only during the execution of a MatrixOp command. MatrixOp uses quaternion arithmetic to evaluate expressions containing quaternion tokens. After the righthand side has been evaluated, MatrixOp stores the result in a wave. Functions that return quaternion coordinates, such as quat and matrixToQuat, return a 4-element wave containing the w, x, y and z values.

You can also create a quaternion token from a scalar or from a wave containing x, y, and z values:

```
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) converts the wave containing x, y, and z to a pure imaginary quaternion whose w component is 0. wXZY can be a 3x1 wave as in this example or it can be a 1x3 wave; both produce the same quaternion token.

quat(3) converts the scalar value 3 to a real quaternion whose x, y and z components are 0.

The input to quat must be real, not complex.

The quat function does not normalize the quaternion. It is normalized when it is used as an input to another MatrixOp quaternion operation.

MatrixOp supports the following quaternion functions:

quat, quatToMatrix, quatToAxis, axisToQuat, quatToEuler, and slerp

These are documented in the reference documenation for MatrixOp.

This example shows how to use quatToEuler to generale Euler angles from a quaternion:

```
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
```