In this example the RHS mixes real and complex waves, something MatrixOp handles without problems.

In addition to the standard math operators, MatrixOp supports a list of functions that can be applied to operands on the RHS. Examples are `abs`, `sin`, `cos`, `mean`, `minVal` and `maxVal`. There are many more described in the **MatrixOp** reference documentation and listed below under **MatrixOp Functions by Category** on page III-149.

MatrixOp Data Tokens

A MatrixOp command has the general form:

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

The expression on the RHS is a combination of data tokens with MatrixOp operators and MatrixOp functions. A data token is one of the following:

- A literal number
- A numeric constant declared with the Constant keyword
- A numeric local variable
- A numeric global variable
- A numeric wave
- A numeric wave layer
- The result from a MatrixOp function

You can not call a user-defined function or external function from *expression*.

This function illustrates each type of data token:

```
Constant kConstant = 234
Function Demo()
   // Literal number
   MatrixOp/O dest = 123

   // Constant
   MatrixOp/O dest = kConstant

   // Local Variable
   Variable localVariable = 345
   MatrixOp/O dest = localVariable

   // Global Variable
   Variable/G root:gVar = 456
   NVAR globalVariable = root:gVar
   MatrixOp/O dest = globalVariable

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

   // Wave layer
   Make/O/N=(3,3,3) w3D = p + 10*q + 100*r
   MatrixOp/O layer1 = w3D[][][1]

   // Output from a MatrixOp function
   MatrixOp/O invMat = inv(mat)
End
```

MatrixOp Wave Data Tokens

MatrixOp uses only the numeric data and the dimensions of waves. All other properties, in particular wave scaling, are ignored. If wave scaling is important in your application, follow MatrixOp with **CopyScales** or **SetScale** or use a wave assignment statement instead of MatrixOp.

From the command line you can reference a wave in *expression* using the simple wave name, the partial data folder path to the wave or the full data folder path to a wave. In a user-defined function you can reference a wave using a simple wave name or a wave reference variable pointing to an existing wave. We call all of these methods of referencing waves "wave references".

A wave data token consists of a wave reference optionally qualified by indices that identify a subset of the wave. For example:

```
Function Demo()
   // Creates automatic wave reference for w3D
   Make/O/N=(3,4,5) w3D = p + 10*q + 100*r

   MatrixOp/O dest = w3D            // dest is a 3D wave
   MatrixOp/O dest = w3D[0][1][2]   // dest is a 1D wave with 1 point
   MatrixOp/O dest = w3D[0][1][]    // dest is a 2D wave with 1 layer
End
```

MatrixOp supports only two kinds of subsets of waves:

- A subset that evaluates to a scalar (a single value)
- A subset that evaluates to one or more layers (2D arrays)

Subsets that evaluate to a row (`w3D[1][][2]`) or a column (`w3D[][1][2]`) are not supported.

Scalars: The following expressions evaluate to scalars:

`wave1d[a]`	*a* is a literal number or a local variable. MatrixOp clips the index *a* to the valid range for *wave1d*. The expression evaluates to a scalar equal to the referenced wave element.
`wave2d[a][b]`	*a* and *b* are literal numbers or local variables. MatrixOp clips the indices to the valid range of the corresponding dimensions. The expression evaluates to a scalar equal to the referenced wave element.
`wave3d[a][b][c]`	*a, b* and *c* are literal numbers or local variables. MatrixOp clips the indices to the valid range of the corresponding dimensions. The expression evaluates to a scalar equal to the referenced wave element.

Layers: The following evaluate to one or more 2D layers:

`wave3d[][][a]`	Layer *a* from the 3D wave is treated as a 2D matrix. The first and second bracket pairs must be empty. MatrixOp clips *a* to the valid range of layers. The result is a 2D wave.
`wave3d[][][a,b]`	*expression* is evaluated for all layers between layer *a* and layer *b*. MatrixOp clips *a* and *b* to the valid range of layers. The result is a 3D wave.
`wave3d[][][a,b,c]`	*expression* is evaluated for layers starting with layer *a* and increasing to layer *b* incrementing by *c*. Layers are clipped to the valid range and *c* must be a positive integer. The result is a 3D wave.

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

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

MatrixOp does not support expressions that include the same 3D wave on both sides of the equation:

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

Instead use:

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