Complex matrices in gretl

Allin Cottrell

August 28, 2019

1 Introduction

As of mid-August 2019, gretl has native support for complex matrices. Not all of hansl's matrix functions accept complex input, but we have enabled a sizable subset of these functions which we believe should suffice for most econometric purposes. Complex support may be extended in future as need arises.

A first point to note is that complex values are treated as a special case of the hansl `matrix` type; there's no `complex` type as such. Complex scalars fall under the `matrix` type as 1×1 matrices; the hansl `scalar` type is only for real values (as is the `series` type).

This document explains how to create and manipulate complex matrices, and discusses some questions of backward compatibility.

2 Creating a complex matrix

The unique explicit constructor for complex matrices is the `complex()` function.[1] This takes two arguments, giving the real and imaginary parts respectively, and sticks them together, as in

```
C = complex(A, B)
```

Four cases are supported, as follows. (Other than in the subscripts, i denotes $\sqrt{-1}$.)

- `A` and `B` are both $m \times n$ real matrices Then `C` is an $m \times n$ complex matrix such that $c_{ij} = a_{ij} + b_{ij}\, i$.
- `A` and `B` are both scalars: `C` is a 1×1 complex matrix such that $c = a + b\, i$.
- `A` is an $m \times n$ real matrix and `B` is a scalar: `C` is an $m \times n$ matrix such that $c_{ij} = a_{ij} + b\, i$.
- `A` is a scalar and `B` is an $m \times n$ real matrix: `C` is an $m \times n$ matrix such that $c_{ij} = a + b_{ij}\, i$.

In addition, complex matrices may naturally arise as the result of certain computations.

With both real and complex matrices in circulation, one may wish to determine whether a particular matrix is complex. The function `iscomplex()` can tell you. Passed an identifier, it returns 1 if it names a complex matrix, 0 if it names a real matrix, or `NA` otherwise.

3 Indexation

Indexation of complex matrices works as with real matrices, on the understanding that each element of a complex matrix is a complex pair. So for example `C[i,j]` gets you the complex pair at row `i`, column `j` of `C`, in the form of a 1×1 complex matrix.

[1] At present gretl does not have a notation such as Julia's `C = A + B*im` for defining a complex matrix.

If you wish to access just the real or imaginary part of a given element, or range of elements, you can use the functions `Re()` or `Im()`, as in

```
scalar rij = Re(C[i,j])
```

which gets you the real part of c_{ij}.

In addition the dummy selectors `real` and `imag` can be used to assign to just the real or imaginary component of a complex matrix. Here are two examples:

```
# replace the real part of C with random normals
C[real] = mnormal(rows(C), cols(C))

# set the imaginary part of C to all zeros
C[imag] = 0
```

The replacement must be either a real matrix of the same dimensions as the target, or a scalar.

Further, the `real` and `imag` selectors may be combined with regular selectors to access specific portions of a complex matrix for either reading or writing. Examples:

```
# retrieve the real part of a submatrix of C
matrix R = C[1:2,1:2][real]

# set the imaginary part of C[3,3] to y
C[3,3][imag] = y
```

4 Operators

Most of the operators available for working with real matrices are also available for complex ones; this includes the "dot-operators" which work element-wise or by "broadcasting" vectors. Moreover, "mixed" operands are accepted, as in `D = C + A` where `C` is complex and `A` real. The result, `D`, will be complex. In such cases the real operand is treated as a complex matrix with an all-zero imaginary part.

The operators *not* defined for complex values are:

- Those that include the tests ">" or "<", since complex values as such cannot be compared as greater or lesser (though they can be compared as equal or not equal).
- The (real) modulus operator (percent sign), as in `x % y` which gives the remainder on division of `x` by `y`.

The "`'`" operator is available in both unary form (transpose), as in `B = A'`, and binary form (transpose-multiply), as in `C = A'B`. But note that for complex `A` this means the conjugate transpose, A^{H} or A^*. If you need the non-conjugated transpose you can use `transp()` — see section 5.

You may wish to note, although none of gretl's explicit regression functions (or commands) accept complex input you can calculate parameter estimates for a least-squares regression of complex Y ($T \times 1$) on complex X ($T \times k$) via `B = X \ Y`.

5 Functions

To give an idea of what works, and what doesn't work, for complex matrices, we'll walk through the hansl function-space using the categories employed in gretl's online "Function reference" (under the Help menu in the GUI program).

5.1 Linear algebra

The functions that accept complex arguments at present are: `cholesky`, `det`, `ldet`, `eigensym` (for Hermitian matrices), `ffti`, `inv`, `ginv`, `hdprod`, `mexp`, `mlog`, `qrdecomp`, `rank`, `svd`, `tr`, and `transp`. Note, however, that `mexp` and `mlog` require that the input matrix be diagonalizable, and `cholesky` requires a Hermitian matrix.

In addition the new functions `eiggen2` and `fft2` are complex-supporting versions of `eigengen` and `fft`, respectively (see section 7 for details). And there are the complex-only functions `ctrans`, which gives the conjugate transpose,[2] and `schur` for the Schur factorization (see section 9).

5.2 Matrix building

Given what was said in section 2 above, several of the functions in this category should be thought of as applying to the real or imaginary part of a complex matrix (for example, `ones` and `mnormal`), and are of course usable in that way. However, some of these functions can be applied to complex matrices as such, namely, `diag`, `diagcat`, `lower`, `upper`, `vec`, `vech` and `unvech`.

Please note: when `unvech` is applied to a suitable real vector it produces a symmetric matrix, but when applied to a complex vector it produces a Hermitian matrix.

The only functions *not* available for complex matrices are `cnameset` and `rnameset`. That is, you cannot name the columns or rows of such matrices (although this restriction could probably be lifted without great difficulty).

5.3 Matrix shaping

The functions that accept complex input are: `cols`, `rows`, `mreverse`, `mshape`, `selifc`, `selifr` and `trimr`.

The functions `msortby`, `sort` and `dsort` are excluded for the reason mentioned in section 4.

5.4 Statistical

Supported for complex input: `meanc`, `meanr`, `sumc`, `sumr`, `prodc` and `prodr`. And that's all.

5.5 Mathematical

In the matrix context, these are functions that are applied element by element. For complex input the following are supported: `log`, `exp` and `sqrt`, plus all of the trigonometric functions with the exception of `atan2`.

In addition there are the complex-only functions `cmod` (complex modulus, also accessible via `abs`), `carg` (complex "argument"), `conj` (complex conjugate), `Re` (real part) and `Im` (imaginary part). Note that $\text{carg}(z) = \text{atan2}(y, x)$ for $z = x + y\,i$.

5.6 Transformations

The functions `cum` and `diff` may be applied to complex matrices, but no others.

6 File input/output

Complex matrices should be stored and retrieved correctly in the XML serialization used for gretl session files (`*.gretl`).

[2] The `transp` function gives the straight (non-conjugated) transpose of a complex matrix.

The functions `mwrite` and `mread` work in two modes: binary mode if the filename ends with "`.bin`" and text mode otherwise. Both modes handle complex matrices correctly if both the writing and the reading are to be done by gretl, but for exchange of data with "foreign" programs text mode will *not* work for complex matrices as a whole. The options are:

- In text mode, use `mwrite` and `mread` on the two parts of a complex matrix separately, and reassemble the matrix in the target program.
- Use binary mode (on the whole matrix), if this is supported for the given foreign program.

At present binary mode transfer of complex matrices is supported for octave, python and julia. Listing 1 shows some examples: we export a complex matrix to each of these programs in turn; calculate its inverse in the foreign program; then verify that the result as imported back into gretl is the same as that calculated in gretl.

Listing 1: Exporting and importing complex matrices

```
set seed 34756
matrix C = complex(mnormal(3,3), mnormal(3,3))
D = inv(C)

mwrite(C, "C.bin", 1)

foreign language=octave
  C = gretl_loadmat('C.bin');
  gretl_export(inv(C), 'oct_D.bin');
end foreign

oct_D = mread("oct_D.bin", 1)
eval D - oct_D

foreign language=python
   import numpy as np
   C = gretl_loadmat('C.bin')
   gretl_export(np.linalg.inv(C), 'py_D.bin')
end foreign

py_D = mread("py_D.bin", 1)
eval D - py_D

foreign language=julia
  C = gretl_loadmat("C.bin")
  gretl_export(inv(C), "jl_D.bin")
end foreign

jl_D = mread("jl_D.bin", 1)
eval D - jl_D
```

7 Backward compatibility

Compatibility issues arise in two contexts, both related to the fact that gretl offered some degree of support for complex matrices before they became full citizens of the hansl polity.

1. The functions `fft` (fast Fourier transform for real input) and `eigengen` (eigenvalues and/or eigenvectors of a non-symmetric real matrix) returned complex matrices in what we may call the "legacy" representation. In the case of `fft` and the eigenvalues from `eigengen` this took the form of a regular gretl matrix with real values in the first (or odd-numbered) column(s) and imaginary parts in the second (or even-numbered) column(s). Since calculating with such matrices using the standard matrix operators would result in nonsense, we provided the tailored functions `cmult` and `cdiv`.

   In the case of complex eigenvectors from `eigengen`—well, you probably don't want to know, but if you do, consult the help text for `eigengen`; they were not easy for a user to handle!

2. The function packages `cmatrix` and `ghosts`. These are relatively recent additions to gretl, designed to support frequency-domain analysis. Prior to the development of native complex-matrix functionality, `cmatrix` was needed as an dependency for `ghosts` (multivariate spectral analysis).

So what happens with these functions and packages under the new regime?

Our resolution on the two built-in functions is this:

- `fft` and `eigengen` continue to behave exactly as before. They do not accept complex input and they produce old-style output. In the documentation they will be marked as legacy functions, not for use in newly written hansl code.
- We have added new counterpart functions, `fft2` and `eiggen2`. These accept either real or complex input and they produce new-style complex output in both cases.

On the affected packages: `cmatrix` is no longer required, and will not be supported any more: attempting to load this package will produce an error message in the next gretl release. We will make available an updated version of `ghosts` which uses gretl's native complex functionality.

Further note: It should be mentioned that the `ffti` function is backward compatible: the new functionality is just a superset of the old. The input must be complex, and is accepted by `ffti` in either the legacy or the new format. The output is real if the input is Hermitian, which in this case means that the first (zero-frequency) row is real and the remaining rows are conjugate symmetrical about their mid-point. That's the only case that can arise when `ffti` is used on output from the old `fft` (which only accepted real input). Otherwise (non-Hermitian complex input, which can arise only under the new scheme) the output will be complex, and in the new format.

8 Devels corner: internals and testing

In the C programming language a double-precision complex value is basically just two regular "doubles" contiguous in memory (but with a good deal of useful syntactic sugar). Since gretl matrices are in column-major order this means that an $m \times n$ complex matrix is in a sense equivalent to a $2m \times n$ real matrix, with real parts on the odd-numbered rows and imaginary parts on the even ones.

In certain special conditions it may be useful to be able to switch the interpretation of a given matrix. To this end there's a "hidden" function `_setcmplx`; it takes a matrix argument followed by an optional boolean: 1 (the default) to switch the interpretation from real to complex, or 0 to go from complex to real. While it's always possible to go from complex to real, the opposite switch fails if the matrix in question has an odd number of rows. Example:

```
? matrix C = complex({1,2;3,4},{0,1;0,1})
Generated matrix C
```

```
? print C
C (2 x 2)

 1 + 0i   2 + 1i
 3 + 0i   4 + 1i

? C = _setcmplx(C, 0)
? print C
C (4 x 2)

  1   2
  0   1
  3   4
  0   1
```

We also mentioned above (section 7) that prior to offering native complex matrix support gretl employed an *ad hoc* representation of such matrices, with real and imaginary parts in alternating columns. A second hidden function, `_cswitch`, can be used to convert between these formats.

As with `_setcmplx`, `_cswitch` takes a matrix argument followed by an optional boolean: 1 (the default) converts an old-style complex matrix to new-style; 0 performs the inverse operation. Examples:

```
? matrix C = complex({1,2;3,4},{0,1;0,1})
Generated matrix C
? print C
C (2 x 2)

 1 + 0i   2 + 1i
 3 + 0i   4 + 1i

? C = _cswitch(C, 0)
Replaced matrix C
? print C
C (2 x 4)

  1   0   2   1
  3   0   4   1

? C = _cswitch(C, 1)
Replaced matrix C
? print C
C (2 x 2)

 1 + 0i   2 + 1i
 3 + 0i   4 + 1i
```

Should they turn out to be useful for more than just testing, one or both of these hidden functions might be renamed and exposed for general use.

9 Signatures of new functions

In this final section we recap (and elaborate where necessary) on the signatures of functions that are specific to the new treatment of complex matrices.

`carg(C)`:
Argument: complex matrix C.
Returns: real matrix, the "argument" of C.

`cmod(C)`:
Argument: complex matrix C.
Returns: real matrix, the modulus of C.

`complex(A, B)`:
Arguments: A (real matrix or scalar), B (real matrix or scalar).
See section 2 for details.

`conj(C)`:
Argument: complex matrix C.
Returns: complex matrix, the complex conjugate of C.

`ctrans(C)`:
Argument: complex matrix C.
Returns: complex matrix, the conjugate transpose of C.

`eiggen2(A, &V, &W)`:
Argument 1: square matrix A, either real or complex.
Argument 2 (optional): address of matrix.
Argument 3 (optional): address of matrix.
Returns: complex column vector, the eigenvalues of A. If argument 2 is provided it retrieves a complex matrix holding the right eigenvectors of A; if argument 3 is provided it retrieves a complex matrix holding the left eigenvectors of A.

`fft2(X)`:
Argument: matrix X, either real or complex.
Returns: complex matrix holding the discrete FFT of X, by column.

`Re(C)`:
Argument: complex matrix C.
Returns: real matrix, the real part of C.

`Im(C)`:
Argument: complex matrix C.
Returns: real matrix, the imaginary part of C.

`schur(A, &Z, &w)`:
Argument 1: square complex matrix A.
Argument 2 (optional): address of matrix.
Argument 3 (optional): address of matrix.
Returns: complex upper triangular matrix T. If argument 2 is provided it retrieves a complex matrix Z holding the Schur vectors associated with A and T, such that $A = ZTZ^{\mathrm{H}}$. If argument 3 is provided it retrieves the eigenvalues of A in a complex column vector.