# `gensys` C++ `mex` Function and `Julia` Comparison

Callum Jones*

New York University

These are brief notes on a C++ implementation of `gensys` as a Matlab `mex` function. It significantly speeds up computation relative to the original `gensys` implementation by using LAPACK routines to compute the Schur (QZ) decomposition associated with Sims' (2002) solution of rational expectations models. I compare an easy change to the Matlab files which significantly speeds up the Matlab routines making them comparable to Julia routines.

The approach will be to write as a `mex` function the component of the `gensys` solution method which reorders the output of the necessary Schur (or QZ) decomposition. Why this Schur decomposition is needed is described in the appendix here. Using the C++ function significantly speeds up computation of the rational expectations solution.

## 1 `mex` function

A `mex` function is a Matlab executable, allowing you to run routines written in C++ (or C or Fortan) directly in Matlab, in the style of a Matlab function. These can be very useful if we want to run numerical routines which are much faster in programming languages other than Matlab's, while maintaining the nice interface and ease of matrix operations that Matlab offers.

A `mex` function requires a *gateway routine*. This needs to be named `mexFunction` and takes four parameters: `nlhs`, `plhs[]`, `nrhs`, and `prhs[]`. The two inputs `nlhs` and `nrhs` are integers specifying the number of LHS (output) and RHS (input) arguments. The input and output matrices are of type `mxArray*`, that is, they point to Matlab arrays. They are stored sequentially in `plhs[]` and `prhs[]`. Once we have extracted the Matlab input objects, we can write C++ routines as normal on those objects.[1]

For the `gensys` application, the `mex` gateway function is written in `qz_.cpp` and has the following structure:

*Department of Economics, New York University. Email: `callum.jones@nyu.edu`. All remaining errors are mine. Date: January 26, 2016.

[1] A note on data organization in C++: the elements of the matrix are stored column-wise, so that, for example, for the following matrix:

$$Z = \begin{bmatrix} a & b \\ c & d \end{bmatrix},$$

in C++, $Z$ is stored as `Z[0] = a`, `Z[1] = c`, `Z[2] = b` and `Z[3] = d`.

```
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
 // Extract and setup objects and call LAPACK QZ decomposition routine
}
```

In the header, we include the `mex` header file, `mex.h` to use the Matlab `mex` library. This defines `mex` types and functions, which are needed to make the input objects compatible with LAPACK routine types. Here, I use the LAPACK routine `zgges` to do the Schur decomposition. For `zgges`, I need to use `double*` types. To copy data from `mxArray*` to `double*`, we need to use the function `mxGetPr`. `mxCreateDoubleMatrix` will return a pointer to an object of type `mxArray`. `mxDestroyArray` is used for memory management.

(See the file `qz_.cpp` for the gateway function and for how the `mex` objects are created and used together with the LAPACK routines.)

## 2 Compiling

To compile the `mex` file, we need to link to Matlab's `mex` compiler. Then we need to link to the right folders for the LAPACK and math libraries in the `makefile` and adjust the suffix of the function file to be correct for the system compiled on. For example, for Macs, this suffix is `mexmaci64`. The `makefile` is:

```
MEX      = /Applications/MATLAB_R2013a_Student.app/bin/mex
SDIR     = .
LPDIR    = $(HOME)/Documents/lib/clapack
FLIB     = $(HOME)/Documents/lib/clapack/F2CLIBS
BLDIR    = $(HOME)/Documents/lib/clapack/BLAS
INDIR    = $(HOME)/Documents/lib/clapack/INCLUDE

LDFLAGS  = -I$(SDIR) -I$(INDIR) -lm -L$(BLDIR) $(FLIB)/libf2c.a \
           $(LPDIR)/lapack_LINUX.a $(LPDIR)/blas_LINUX.a

qz_.mexmaci64 :
 $(MEX) qz_.cpp $(LDFLAGS)
clean :
 rm -f *.mexmaci64
```

To compile the makefile with name `makefile` from the Matlab command line, run `!make`. This creates the function `qz_.mexmaci64`.

## 3 New files

With the new Schur decomposition function, we do not need the files `qzswitch`, `qzdiv` and `qzdivct`. The lines of the `gensys` code where the QZ decomposition is computed and reordered are deleted and replaced with the `mex` function:

```
[A, B, Q, Z] = qz_(GAM0,GAM1) ;
```

The remainder of the `gensys` function is unchanged.

## 4 Testing

The Matlab function `test_qz` tests that the QZ decomposition outputs the correct results and for the Ireland (2004) model with 128 variables (including 120 quarter long-rates) that the solution from the original `gensys` and new `gensys_` are the same (Fig 1). The speed increase is: $\frac{0.9380 \text{ s}}{0.1040 \text{ s}} = 9\times$.

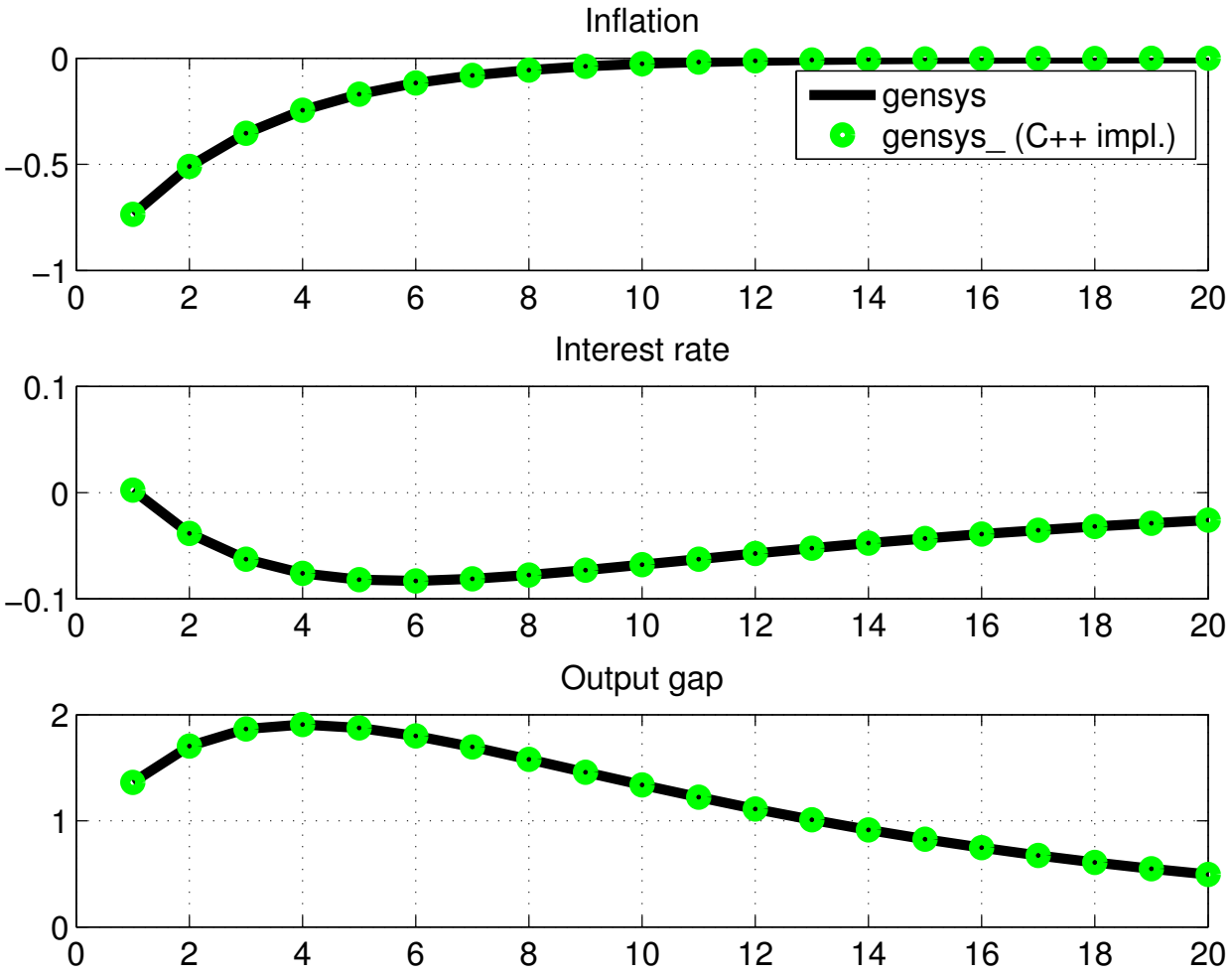


**Figure 1: IRF for both implementations**. They give the same solution.

## 5 Comparison of `gensys` to Julia implementation

A recent effort at the FRBNY interestingly develops Julia routines that solve and maximize the likelihood of the FRBNY's DSGE model.[2] Their Julia routines result in significant speed-ups of between 10% and 75% beyond the baseline Matlab routine. As they discuss, it is hard to disentangle the performance improvement due to the actual implementation and the performance improvement due to the difference in programming/execution environments. I show in this section that a simple two-line change to the baseline Matlab files in the FRBNY's implementation can result in Matlab performance times that are much closer to the Julia implementation.

As described in the appendix the main computational effort in solving a linear rational expectations model is the QZ decomposition and the reordering of the matrices of the decomposition to match those matrices' generalized eigenvalues in ascending order. The original `gensys` implementation does this reordering in separate functions `qzdiv`, `qzswitch` and `qzdivct`. However, the same computation can be done very efficiently in Matlab using the `ordqz` function.

To test the performance differences, I time how long it takes to complete three iterations of the function which minimizes the posterior of the FRBNY, as in the test functions in the FRBNY Julia implementation and as in the latest version of the Matlab implementation of the FRBNY model. Ideally the Matlab implementation of the model would be further optimized, for example, with all output prompts suppressed. Also, the comparison between the two methods would ideally be initialized at the same starting value and Hessian. While I don't do this, the comparison between the original Matlab benchmark, the adjusted Matlab implementation and the Julia implementation is informative.

| **Time relative to Matlab original `gensys` benchmark** | | |
|---|---|---|
| Matlab - original `gensys` | Matlab - with `ordqz` | Julia - `optimize` test |
| 1.00 | 0.60 | 0.47 |

As the table shows, making the straightforward adjustment of the functions which reorder the solution matrices provides a significant speed up in the Matlab environment – this performance improvement was captured in the C++ implementation above.

## A The QZ decomposition in the linear RE solution

Write a linear rational expectations model in matrix form as:

$$\tilde{\Gamma}_0 \mathbf{y}_t = \tilde{\Gamma}_1 \mathbf{y}_{t-1} + \tilde{C} + \tilde{\Psi}\varepsilon_t, \tag{1}$$

[2] See Marco Del Negro, Marc Giannoni, Pearl Li, Erica Moszkowski and Micah Smith, *The FRBNY DSGE Model Meets Julia*, December 2015.

where $\mathbf{y}_t$ is the state vector, defined by $\mathbf{y}_t = [\ \mathbf{y}'_{1,t} \quad \mathbf{y}'_{2,t} \quad \mathbb{E}_t\mathbf{z}'_{t+1}\ ]'$ where $\mathbf{y}_{1,t}$ is a $(n_1 \times 1)$ vector of exogenous and endogenous variables, and $\mathbf{y}_{2,t}$ is a $(n_2 \times 1)$ vector with those endogenous variables for which conditional expectations appear. The vector $\mathbf{z}_{t+1}$ of size $(k \times 1)$ contains leads of $\mathbf{y}_{2,t}$. The dimension of $\mathbf{y}_t$ is $(n \times 1)$, where $n = n_1 + n_2 + k$. Assume $\varepsilon_t$ to be a $(l \times 1)$ vector of serially uncorrelated processes. The matrices $\tilde{\Gamma}_0$ and $\tilde{\Gamma}_1$ are $(n_1 + n_2) \times n$ matrices, $\tilde{C}$ is $(n_1 + n_2) \times 1$ and $\tilde{\Psi}$ is $(n_1 + n_2) \times l$.

We seek a solution of the model (1) in the form of a VAR(1). There are $(n_1 + n_2)$ equations in model (1), and so because of the presence of expectations, it is not possible to invert $\tilde{\Gamma}$. Sims' (2002) proposal is to append to (1) expectations revisions which will be solved as part of the solution. Let $\eta_t$ be the vector of expectations revisions:

$$\eta_t = \mathbb{E}_t\mathbf{z}_t - \mathbb{E}_{t-1}\mathbf{z}_t, \tag{2}$$

where $\mathbb{E}_t\eta_{t+j} = 0$ for $j \geq 1$. For example, if $z_t = y_{2,t}$, then $\eta_t$ are forecast revisions.

Augment the system defined by Equation (1) with the $k$ equations from Equation (2):

$$\Gamma_0\mathbf{y}_t = C + \Gamma_1\mathbf{y}_{t-1} + \Psi\varepsilon_t + \Pi\eta_t. \tag{3}$$

where the matrices $\Gamma_0$, $\Gamma_1$,$C$,$\Psi$, and $\Pi$ are of conformable dimensions. $\Gamma_0$ is now an $n \times n$ matrix, which we will invert with a Schur (QZ) decomposition, and impose conditions such that we can remove the $\eta_t$ from the system. It is this QZ decomposition which can be sped up in the original `gensys` implementation.

To solve (3) as Sims (2002), take a Schur (QZ) decomposition of $(\Gamma_0, \Gamma_1)$ to get $Q'\Lambda Z' = \Gamma_0$ and $Q'\Omega Z' = \Gamma_1$, where $\Lambda$ and $\Omega$ are both upper triangular. The matrices $Q$ and $Z$ are unitary, so that $QQ' = I$ and $ZZ' = I$. Pre-multiply model equation by $Q$ and define $w_t = Z'y_t$ to rewrite the system as:

$$\Lambda w_t = \Omega w_{t-1} + Q(C + \Psi\varepsilon_t + \Pi\eta_t).$$

Define $w_{1,t} = Z'_1 y_t$ and $w_{2,t} = Z'_2 y_t$. $\Lambda$ and $\Omega$ are upper triangular and have the property that the generalized eigenvalues of $(\Gamma_0, \Gamma_1)$ are ratios of diagonal elements of $\Omega$ and $\Lambda$. Rearrange the system so that the explosive eigenvalues correspond to the lower right blocks of $\Lambda$ and $\Sigma$, partitioning $w_t$ and rewriting the system as:

$$\begin{pmatrix} \Lambda_{11} & \Lambda_{12} \\ 0 & \Lambda_{22} \end{pmatrix} \begin{pmatrix} w_{1,t} \\ w_{2,t} \end{pmatrix} = \begin{pmatrix} \Omega_{11} & \Omega_{12} \\ 0 & \Omega_{22} \end{pmatrix} \begin{pmatrix} w_{1,t-1} \\ w_{2,t-1} \end{pmatrix} + \begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix} (C + \Psi\varepsilon_t + \Pi\eta_t).$$

The lower block of the system are those equations which correspond to the $m$ explosive generalised eigenvalues of $(\Gamma_0, \Gamma_1)$. The lower set of equations are not affected by $w_{1,t}$. Isolate these:

$$\Lambda_{22} w_{2,t} = \Omega_{2,2} w_{2,t-1} + Q_2 \left(C + \Psi\varepsilon_t + \Pi\eta_t\right).$$

For stability, the $\eta_t$ needs to offset the effect of $\varepsilon_t$ on $w_{2,t}$. To see this, solve $w_{2,t}$ forward:

$$w_{2,t} = (\Lambda_{22} - \Omega_{22})^{-1} Q_2 C - \sum_{j=1}^{\infty} \left(\Omega_{22}^{-1}\Lambda_{22}\right)^{j-1} \Omega_{22}^{-1} Q_2 (\Psi \varepsilon_{t+j} + \Pi \eta_{t+j}).$$

This says that $w_{2,t}$ requires having in hand all future values of $\varepsilon_t$ and $\eta_t$ at time $t$. Take expectations of this expression at time $t$ to get:

$$w_{2,t} = (\Lambda_{22} - \Omega_{22})^{-1} Q_2 C - \mathbb{E}_t \sum_{j=1}^{\infty} \left(\Omega_{22}^{-1}\Lambda_{22}\right)^{j-1} \Omega_{22}^{-1} Q_2 \Psi \varepsilon_{t+j}.$$

Also take expectations at time $t+1$ to get:

$$w_{2,t} = (\Lambda_{22} - \Omega_{22})^{-1} Q_2 C - \mathbb{E}_{t+1} \sum_{j=1}^{\infty} \left(\Omega_{22}^{-1}\Lambda_{22}\right)^{j-1} \Omega_{22}^{-1} Q_2 \Psi \varepsilon_{t+j} - \Omega_{22}^{-1} Q_2 \Pi \eta_{t+1}.$$

Note the left hand side has not changed, so equating these two expressions implies:

$$Q_2 \Pi \eta_{t+1} = \Omega_{22} \sum_{j=1}^{\infty} \left(\Omega_{22}^{-1}\Lambda_{22}\right)^{j-1} \Omega_{22}^{-1} Q_2 \left(\mathbb{E}_t \varepsilon_{t+j} - \mathbb{E}_{t+1} \varepsilon_{t+j}\right).$$

This says that for the system to be stable the expectations revisions must offset the effect that shocks $\varepsilon_t$ have on the explosive component of the system, $w_{2,t}$. Expectations revisions ensure that the system is placed on the saddle path to stability. For this to be true, Sims (2002) shows that what is required for a unique solution is that the number of explosive eigenvalues of $(\Gamma_0, \Gamma_1)$, $m$ equals the number of variables which appear as expectations in the system, $k$. Under this condition, the system is on a saddle path to a steady-state from any initial condition (there are weaker conditions just for stability.) If this is true, and if the solution is stable then there is a matrix $\Phi$ such that $Q_1 \Pi = \Phi Q_2 \Pi$. By premultiplying the system by $[I_{n-p}, -\Phi]$, the coefficient on $\eta_t$ is $Q_1 \Pi - \Phi Q_2 \Pi$. Since existence of a solution requires $Q_1 \Pi = \Phi Q_2 \Pi$, the $\eta_t$ drop out of (1), so that a solution is:

$$\mathbf{y}_t = S_0 + S_1 \mathbf{y}_{t-1} + S_2 \varepsilon_t + S_y \mathbb{E}_t \sum_{j=1}^{\infty} M^{j-1} \Omega_{22}^{-1} Q_2 \Psi \varepsilon_{t+j},$$

where:

$$H = Z \begin{pmatrix} \Sigma_{11}^{-1} & -\Sigma_{11}^{-1}(\Sigma_{12} - \Phi\Sigma_{22}) \\ 0 & I \end{pmatrix}, \qquad S_0 = H \begin{pmatrix} Q_1 - \Phi Q_2 \\ (\Sigma_{22} - \Omega_{22})^{-1} Q_2 \end{pmatrix} C,$$

and

$$S_1 = H \begin{pmatrix} \Omega_{11} & \Omega_{12} - \Phi\Omega_{22} \\ 0 & 0 \end{pmatrix}, \qquad S_2 = H \begin{pmatrix} Q_1 - \Phi Q_2 \\ 0 \end{pmatrix} \Psi, \qquad S_y = -H \begin{pmatrix} 0 \\ I_m \end{pmatrix}.$$