A Julia Implementation of Gensys

Gilberto Boareto & Daniel Coutinho

October, 2019

1 Introduction

This document is a white paper on our implementation of gensys, the solution method developed by Sims (2002). Our implementation follows closely Sims's paper and the description in Miao (2014). In general, one takes a model written as:

$$\Gamma_0 y_t = \Gamma_1 y_{t-1} + \Psi \varepsilon_t + \Pi \eta_t$$

In which y_t are the variables today, ε_t is a vector of exogenous shocks and η_t is a vector of endogenous errors, i.e. expectational errors.

Sims's method allows for a Γ_0 that is singular. Instead of determining some variables as *jump* and others as *predetermined*, as Blanchard and Khan (1980) or Klein (2000), Sims's allows it to be done endogenously by the algorithm.

Sims's method has another remarkable difference from the previous methods: instead of ”root counting” - checking that the number of eigenvalues larger than 1 equals the number of *jump* variables and the number of eigenvalues smaller than 1 equals the number of *predetermined* variables - Sims's algorithm relies on ensuring that the expectational errors are linear combinations of the exogenous shocks. We will go into more details in the next section.

We implement this in Julia. Julia is a modern programming language focused on scientific computing, trying to balance speed and ease to use. Julia has reached maturity only recently. However, there is a large number of packages written for Julia. Thomas Sargent is probably the most famous economist with ties with Julia, dedicating a whole section of his website to the language and how-to solve standard models in economics using Julia - although this does not include large scale DSGEs. The Federal Reserve Bank of New York (FED-NY) implemented their DSGE on Julia. By their own account, this sped up estimation ten fold and solving the model was eleven times faster. Given the time it takes to estimate even a medium DSGE model, these gains are not negligibles. Moreover, Julia has a large numerical linear algebra library that comes out of the box (you don't need to install anything besides Julia). This is valuable given that we will work with *linear* models.

This white paper is divided as follows: in the next section, we briefly discuss two methods of numerical linear algebra that will be used in our implementation: the Schur Decomposition (also called the QZ decomposition) and the Singular Value Decomposition (SVD); section three concisely presents the algorithm and our implementation; section four presents a small example from Gali (2008), which has analytical solution - this allows us to make sure that our implementation is sound; section five concludes.

2 Numerical Linear Algebra

We will use two decompositions of matrices: the Schur Decomposition and the Singular Value Decomposition (SVD). Both are closely related to the more well known eigenvalue decomposition:

$$A = P^{-1}\Lambda P$$

In which Λ is a diagonal matrix. In general, P is not orthonormal. The eigenvectors matrix, P, is orthonormal, i.e. $P^{-1} = P'$ in some cases, for example, if A is normal (Symmetric matrices are normal).

The Schur Decomposition tries to find a decomposition of the same form of the eigenvalue decomposition, but instead of imposing that Λ is diagonal, we now want that P$ be orthonormal. So the Schur Decomposition then finds:

$$A = Q'SQ$$

Remarkably S is always an upper triangular matrix.

We will be actually interested in the Generalized Schur Decomposition. To understand this one, lets first understand what is the generalized eigenvalue. In the usual eigenvalue problem, we want to find a vector $\mathbf{x}$ and a value λ such that:

$$Ax = \lambda x$$

Rewritting this in matricial terms, we want:

$$(A - \lambda I)x = 0$$

In which I stands for the identity matrix. Now, what if we want to solve the following problem:

$$Ax = \lambda Bx$$

We can find matrices such that $B = P^{-1}P$ and $A = P^{-1}\Lambda P$, in which Λ is a diagonal matrix.

The idea for the Generalized Schur Decomposition will be the same, but now we want to find matrices Q, Z, S, T such that:

1. $$Q$$

   and Z are orthonormal

2. $$S$$

   and T are upper triangular

3. $$A = QSZ$$

4. $$B = QTZ$$

5. $$\forall i$$

   S_{ii}, T_{ii} are not zero

6. The pairs (S_{ii}, T_{ii}) can be arranged in any order

Item 6 will be particularly useful in what follows

The SVD also follows the idea from the eigenvalue problem. Now the idea is to decompose A as USV', in which S is a diagonal matrix and both U and V are orthonormal.

3 An example

As a proof of concept, lets compare our code with the model of Gali (2008), chapter 3 - this is the simplest standard New Keynesian Model. Lets load first our code for Gensys:

```
using Plots

include(string(pwd(), "/src/gensys.jl"))
```

```
irf (generic function with 2 methods)
```

We also loaded the library Plots. We will calibrate exactly as Gali does, on page 52:

```
bet = 0.99
sig = 1
phi = 1
alfa = 1/3
epsilon = 6
theta = 2/3
phi_pi = 1.5
phi_y = 0.5/4
rho_v = 0.5

THETA = (1-alfa)/(1-alfa+alfa*epsilon)
lamb = (1-theta)*(1-bet*theta)/theta*THETA
kappa = lamb*(sig+(phi+alfa)/(1-alfa))
```

```
0.12750000000000006
```

We refer the reader to Gali's book for knowing what each parameter means. Next, we need to write the usual model in Sim's format. Let us repeat the three equations of the usual model:

$$\pi_t = \beta E_t(\pi_{t+1}) + \kappa \tilde{y}_t \tilde{y}_t = -\frac{1}{\sigma}(i_t - E_t(\pi_{t+1})) + E_t(\tilde{y}_{t+1}) i_t = \phi_\pi \pi_t + \phi_{\tilde{y}} \tilde{y}_t + v_t v_t = \rho_v v_{t-1} + \varepsilon_t^v$$

Gali's book gives an analytical solution for the impulse response functions for a shock v_t. This can be found on page 51. We implement them in Julia as well:

```
LAMBDA_v = ((1-bet*rho_v)*(sig*(1-rho_v)+phi_y)+kappa*(phi_pi-rho_v))^(-1)

y_irf(v) = -(1-bet*rho_v)*LAMBDA_v*v
pi_irf(v) = -kappa*LAMBDA_v*v
r_irf(v) = sig*(1-rho_v)*(1-bet*rho_v)*LAMBDA_v*v
i_irf(v) = (sig*(1-rho_v)*(1-bet*rho_v) - rho_v*kappa)*LAMBDA_v*v
v(v,uu) = rho_v*v + uu

v (generic function with 1 method)
```

And we compute a sample IRF for a positive 25 basis shock in the interest rate:

```
irfs_true = zeros(15,5)

irfs_true[1,1] = 0

uu = 0.25

for t in 2:15
    irfs_true[t,4] = v(irfs_true[t-1,4],uu)
    irfs_true[t,1] = pi_irf(irfs_true[t,4])
    irfs_true[t,2] = y_irf(irfs_true[t,4])
    irfs_true[t,3] = i_irf(irfs_true[t,4])
    irfs_true[t,5] = r_irf(irfs_true[t,4])
    global uu = 0
end
```

Now lets solve it numerically: the first step is to write $\eta_t^\pi = \pi_t - E_{t-1}\pi_t$ and $\eta_t^{\tilde{y}} = \tilde{y}_t - E_{t-1}\tilde{y}_t$. Now, use this to isolate to write the expectations as function of the true value plus the expectational error - $E_t(\pi_{t+1}) = \pi_t + \eta_t^\pi$ for example.

This allows us to write the system above as follows:

$$\pi_{t-1} - \kappa\tilde{y}_{t-1} + \beta\eta_t^\pi = \beta\pi_t \sigma\tilde{y}_{t-1} + i_{t-1} + \eta_t^\pi + \sigma\eta_t^{\tilde{y}} = \pi_t + \sigma\tilde{y}_t i_{t-1} - \phi_\pi\pi_{t-1} - \phi_{\tilde{y}}\tilde{y}_{t-1} - v_{t-1} = 0 v_t = \rho v_{t-1} + \varepsilon_t^v$$

And it is straight foward to write the matrices Γ_0, Γ_1, Ψ, Π, as follows:

```
GAMMA_0 = [bet    0     0  0;
           1      sig   0  0;
           0      0     0  0;
           0      0     0  1]

GAMMA_1 = [1       -kappa  0  0;
           0        sig    1  0;
           -phi_pi  -phi_y  1 -1;
           0        0      0  rho_v]

PSI = [0; 0; 0; 1]

PI = [bet  0;
      1    sig;
      0    0;
      0    0]

4×2 Array{Float64,2}:
 0.99  0.0
 1.0   1.0
 0.0   0.0
 0.0   0.0
```

Here are LaTeX versions of the matrices generated automatically with the package *latexify*, from the inputed matrices for Julia:

```
using Latexify

print(string("\$\$\\Gamma_0 = \$\$",latexify(GAMMA_0)))
```

$$\Gamma_0 =$$
$$\begin{bmatrix} 0.99 & 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{1}$$

```
print(string("\$\$\\Gamma_1 = \$\$",latexify(GAMMA_1)))
```

$$\Gamma_1 =$$
$$\begin{bmatrix} 1.0 & -0.12750000000000006 & 0.0 & 0.0 \\ 0.0 & 1.0 & 1.0 & 0.0 \\ -1.5 & -0.125 & 1.0 & -1.0 \\ 0.0 & 0.0 & 0.0 & 0.5 \end{bmatrix} \tag{2}$$

```
print(string("\$\$\\Psi = \$\$",latexify(PSI)))
```

$$\Psi =$$
$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{3}$$

```
print(string("\$\$\\Pi = \$\$",latexify(PI)))
```

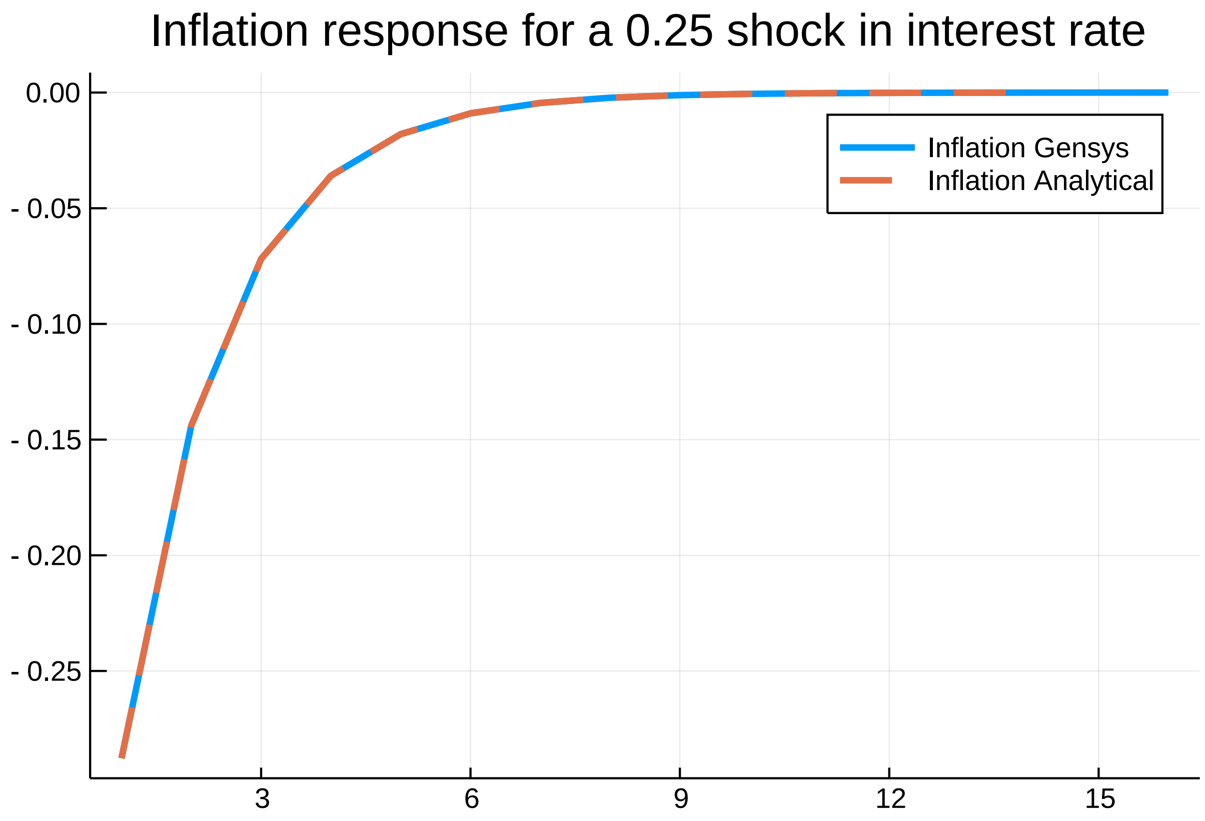
$$\Pi =$$
$$\begin{bmatrix} 0.99 & 0.0 \\ 1.0 & 1.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \end{bmatrix} \tag{4}$$

And we just call the function with this matrices:

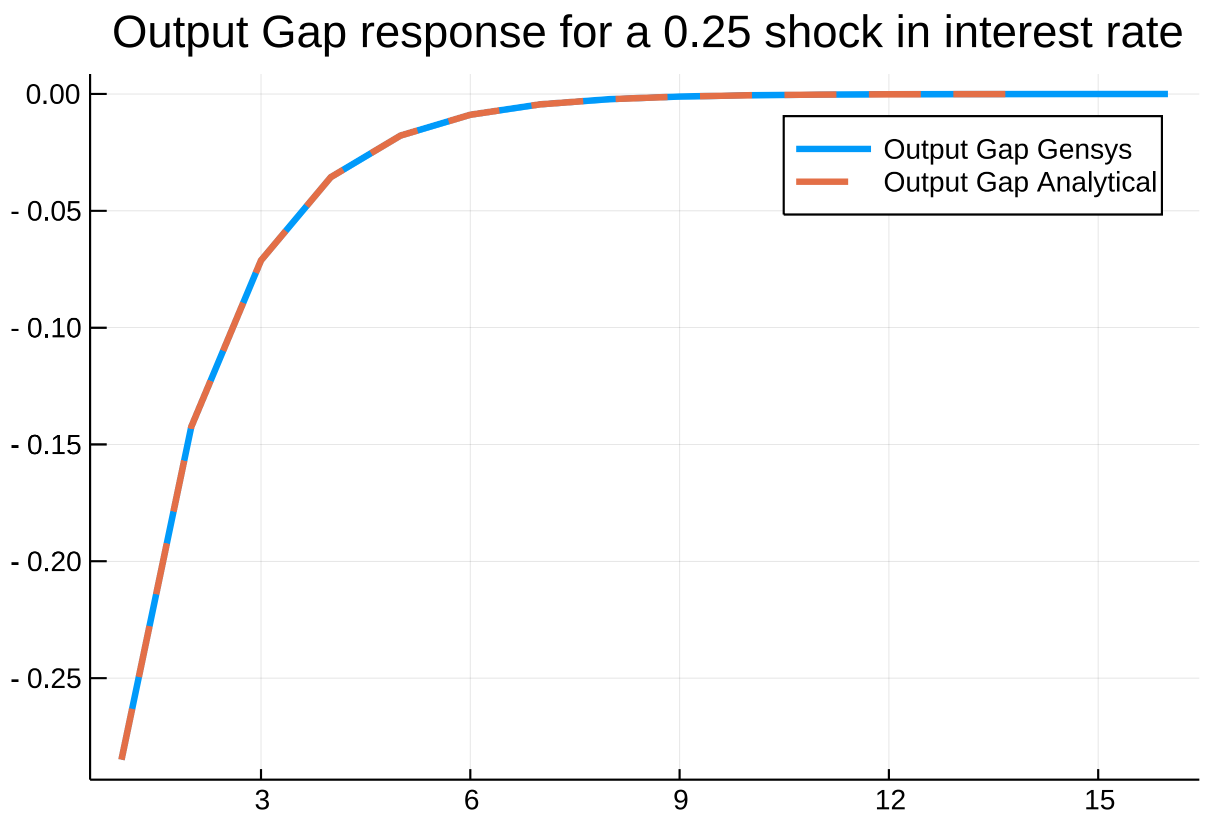
```
sol = gensys(GAMMA_0,GAMMA_1,PSI,PI)

irfs = irf(sol,15,0.25)

16×4 Array{Float64,2}:
 -0.0719323    -0.284908     0.106488     0.25
 -0.0359661    -0.142454     0.053244     0.125
 -0.0179831    -0.0712271    0.026622     0.0625
 -0.00899154   -0.0356135    0.013311     0.03125
 -0.00449577   -0.0178068    0.0066555    0.015625
 -0.00224788   -0.00890339   0.00332775   0.0078125
 -0.00112394   -0.00445169   0.00166388   0.00390625
 -0.000561971  -0.00222585   0.000831938  0.00195312
 -0.000280986  -0.00111292   0.000415969  0.000976562
 -0.000140493  -0.000556462  0.000207984  0.000488281
 -7.02464e-5   -0.000278231  0.000103992  0.000244141
 -3.51232e-5   -0.000139115  5.19961e-5   0.00012207
 -1.75616e-5   -6.95577e-5   2.5998e-5    6.10352e-5
 -8.7808e-6    -3.47788e-5   1.2999e-5    3.05176e-5
 -4.3904e-6    -1.73894e-5   6.49951e-6   1.52588e-5
 -2.1952e-6    -8.69471e-6   3.24976e-6   7.62939e-6
```

Lets compare the IRFs generated analyticaly and by our implementation of Gensys:

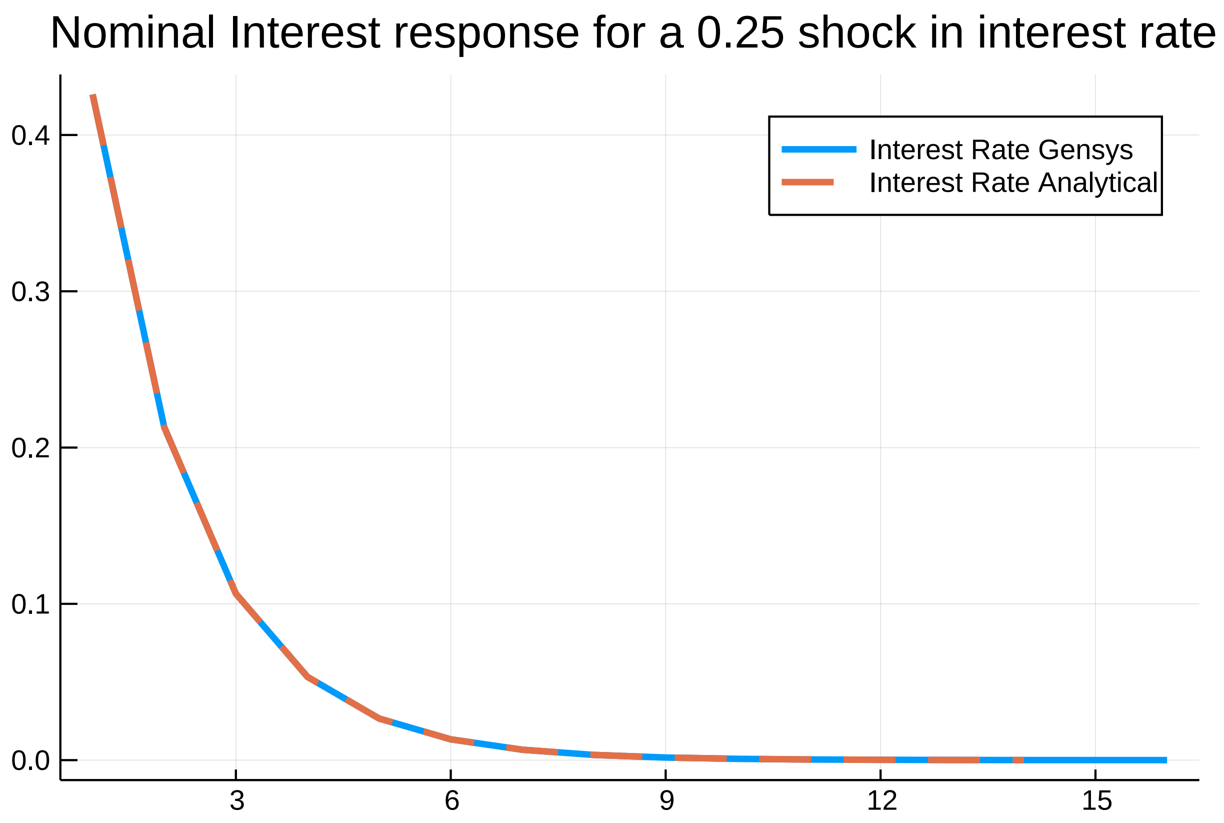
```
plot(4*irfs[:,1], label = "Inflation Gensys", w =3)
plot!(4*irfs_true[2:15,1], label = "Inflation Analytical", line = :dash, w =3)
title!("Inflation response for a 0.25 shock in interest rate")
```



```
plot(irfs[:,2], label = "Output Gap Gensys", w = 3)
plot!(irfs_true[2:15,2], label = "Output Gap Analytical", line = :dash, w = 3)
title!("Output Gap response for a 0.25 shock in interest rate")
```

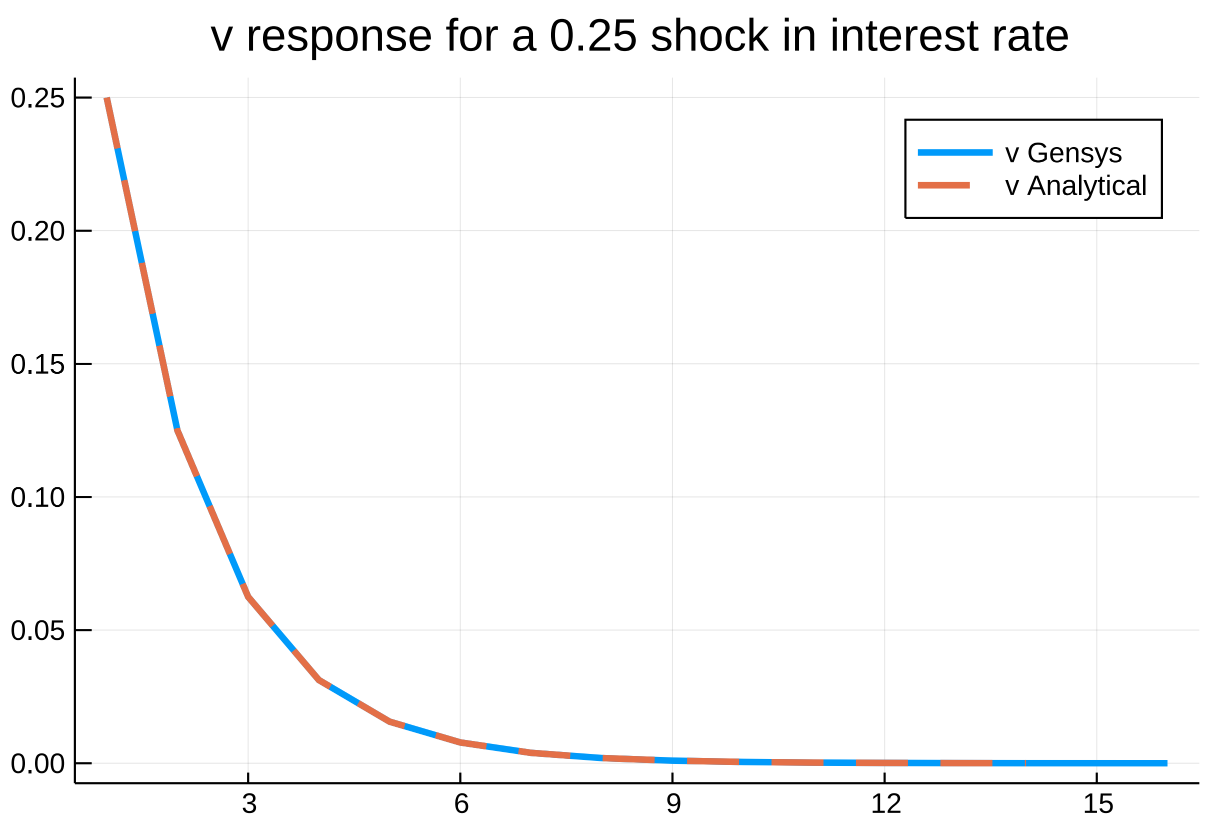


```
plot(4*irfs[:,3], label = "Interest Rate Gensys", w = 3)
plot!(4*irfs_true[2:15,3], label = "Interest Rate Analytical", line = :dash, w = 3)
title!("Nominal Interest response for a 0.25 shock in interest rate")
```



```
plot(irfs[:,4], label = "v Gensys", w = 3)
plot!(irfs_true[2:15,4], label = "v Analytical", line = :dash, w = 3)
```

```
title!("v response for a 0.25 shock in interest rate")
```



The impulse response are identical between the two methods, which is a strong evidence of the reliability and precision of our implementation of Gensys, Gensys itself and Julia capabilities to handle linear algebra routines.