Monetary Policy Models - PS2

Ricardo Semião

2024-11-08

Setup

This document was made using Jupyter Notebook to present the results of a Dynare/Octave code. Dynare files must be compiled, such that Dynare code can be run in independent cells. Still, I present the components of each `.mod` file here in code cells, and run the full code with the `oct2py`'s `%%octave` cell magic.

For starters, lets load the cell magic and other relevant modules.

```
from io import StringIO
from shutil import move

import pandas as pd

import re

%load_ext oct2py.ipython
```

Now, we can set some initial configurations, like suppress warnings, and set the Dynare path.

```
%%octave

warning('off', 'Octave:shadowed-function');
graphics_toolkit('gnuplot');

addpath C:\dynare\6.2\matlab
```

Lastly, this document is rendered with quarto. Using powershell, one runs:

```
!powershell quarto render ps2_main.ipynb --to pdf
```

Question 1

Item 1.

First, we define new ρ parameters, and update the model:

```
parameters rho1 rho2 se;

rho1 = 0.95;
rho2 = -0.5;
```

```
model;
x = rho1*x(-1)+rho2*x(-2)+e;
end;
```

Now, we can run the file that contains all the code above. Then, move the pictures to the correct location.

```
%%octave
```

```
cd dynare_scripts/ar2/
dynare ar2
```

```
move(
    'dynare_scripts/ar2/ar2/graphs/ar2_IRF_e.eps',
    'figures/ar2_IRF_e_095.eps'
)
```

All the outputs that Dynare generates are important to understand if the model is making sense. But here, we are more interested in the IRFs. I'll present them only in the Item 3. section.

Item 2.

One can use matlab loops:

```
rhos=[0.9, 0.7, 0.35];
irfs = nan(1,T,3);

dynare ar2.mod noclearall;

for i=1:length(rhos)
    set_param_value('rho',rhos(i));
    [info, oo_, options_, M_] = stoch_simul(M_, options_, oo_, var_list_);
    irfs(1,:,i) = oo_.irfs.x_e';
end
```

Or the same approach I used in PS1, using REGEX to change the `.mod` file contents. I will use the `re` module to find the `rho1 = ...` line in the existing file, replace it, run it and save the figure.

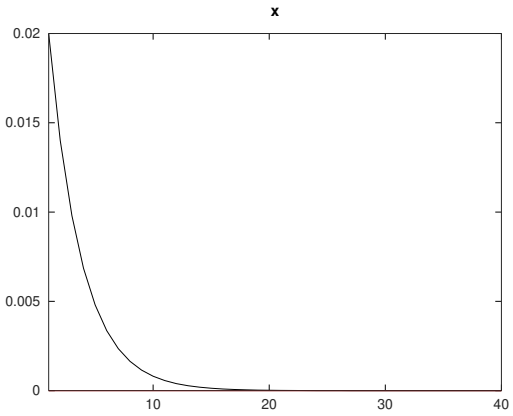
```
mod_path = 'dynare_scripts/ar2/ar2.mod'

for rho1 in [0.9, 0.7, 0.35]:
    with open(mod_path, 'r', encoding='utf-8') as file:
        mod_lines = file.readlines()

    mod_lines = [
        re.sub(r'rho1 = [0-9.]+;', f'rho1 = {rho1};', x)
        for x in mod_lines
    ]

    with open(mod_path, 'w', encoding='utf-8') as file_out:
        file_out.writelines(mod_lines)

    %octave dynare ar2

    move(
        'dynare_scripts/ar2/ar2/graphs/ar2_IRF_e.eps',
        f'figures/ar2_IRF_e_{rho1}.eps'
    )
```

Again, the IRFs will be presented in the next section.

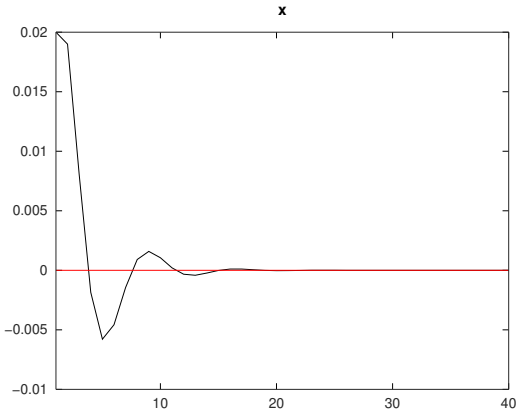
Item 3.

Now, lets run the original file, *ar1.mod*, and save its IRF.

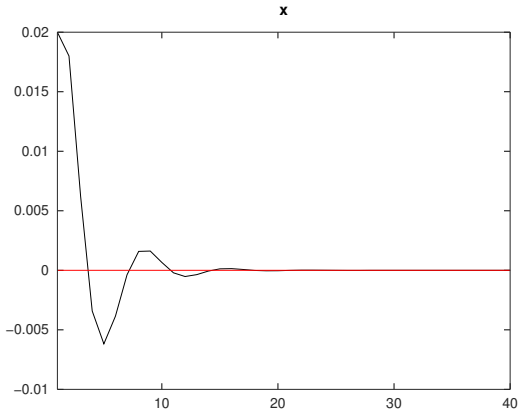
And now we can plot the results:



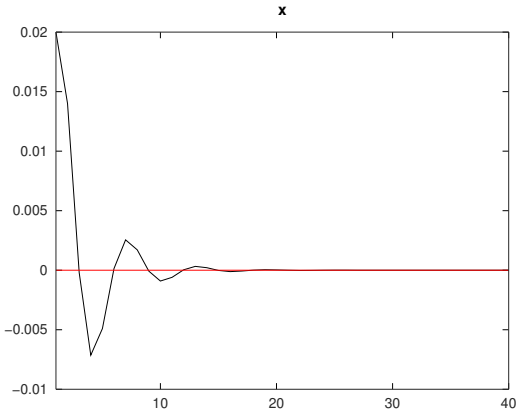
$AR(1), \quad \rho = 0.95$



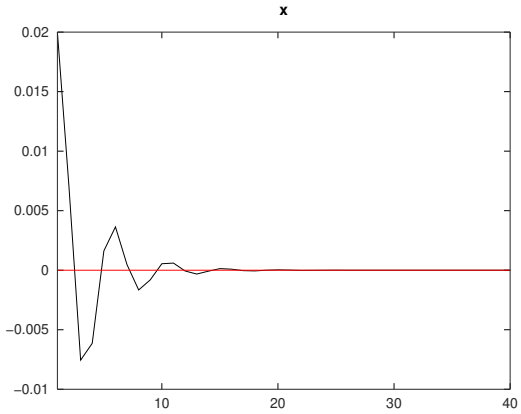
$AR(2), \quad \rho = 0.95$



$AR(2), \quad \rho = 0.90$



$AR(2), \quad \rho = 0.70$



$AR(2), \quad \rho = 0.35$

We can see that the oscillation in the shock response is smaller for bigger values of ρ. This makes sense, since we're giving a higher weight to the past values in the law of motion of the shock, yielding a more stabilized path.

Additionally, we can see that, without the negative ρ_2, effectively, with $\rho_2 = 0$, there is no oscillation, as expected. The response converges monotonically to zero.

Question 2

I'll answer both items together.

Original

The loop below run the model changing the orders to 1, 2, and 3, with the regex `r'stoch_simul(order = [1-3],'`.

```
mod_path = 'dynare_scripts/jermann98/jermann98.mod'

with open(mod_path, 'r', encoding='utf-8') as file:
    mod_lines = file.readlines()

for order in [1, 2, 3]:
    mod_lines = [
        re.sub(r'stoch_simul(order = [1-3],', f'stoch_simul(order = {order},', x)
        for x in mod_lines
    ]

    with open(mod_path, 'w', encoding='utf-8') as file_out:
        file_out.writelines(mod_lines)

    %octave dynare jermann98

    move(
        'dynare_scripts/jermann98/jermann98/graphs/jermann98_IRF_ez.eps',
        f'figures/jermann98_IRF_ez_o{order}.eps'
    )
```

Altered

Lets change the parameters and run the two orders again. I choose to change $\alpha : 0.819 \to 0.85$ and $\xi : 1/4.3 \to 1/4.5$. Both these changes should reduce the risk premia response to the shock, as (i) the past consumption has higher dependence on the past and (ii) the investment gets more costly to adjust, both implies in slower adjusts on the model overall.

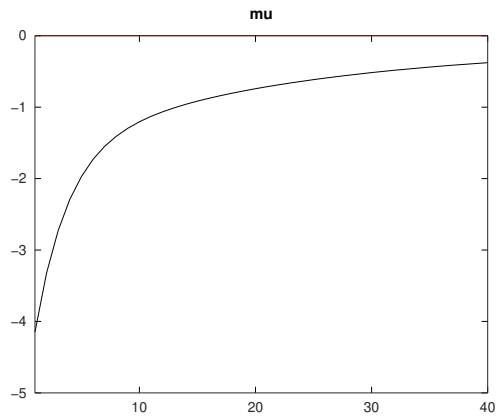
```
with open(mod_path, 'r', encoding='utf-8') as file:
    mod_lines = file.readlines()

mod_lines = [
    re.sub(r'chihab\s*= [0-9./];,', 'chihab = 0.85;',
        re.sub(r'xi\s*= [0-9./];,', 'xi = 1/4.5;',
            x
        )
    )
    for x in mod_lines
]
```

Now we can run the same loop as before, with new picture names:

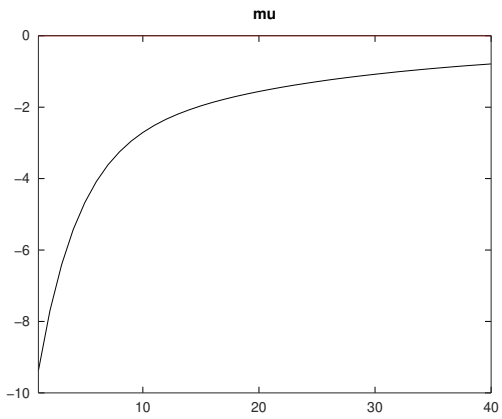
```
for order in [1, 2, 3]:
    mod_lines = [
        re.sub(r'stoch_simul(order = [1-3],', f'stoch_simul(order = {order},', x)
        for x in mod_lines
    ]

    with open(mod_path, 'w', encoding='utf-8') as file_out:
        file_out.writelines(mod_lines)
```

```
%octave dynare jermann98

move(
    'dynare_scripts/jermann98/jermann98/graphs/jermann98_IRF_ez.eps',
    f'figures/jermann98_IRF_ez_o{order}_alt.eps'
)
```

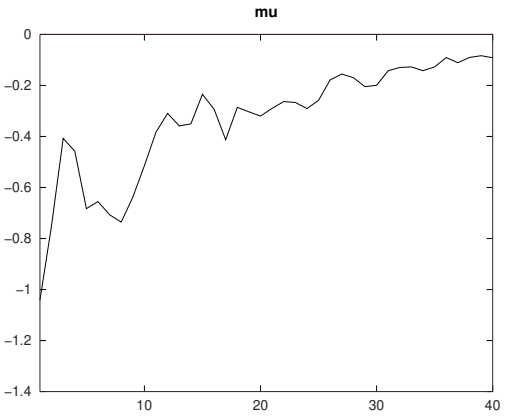
And now we can plot the results:



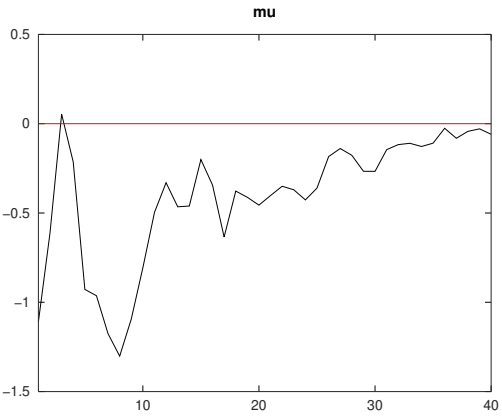
order = 1, original model



order = 1, altered model



order = 2, original model



order = 2, altered model

We can see that mainly in the first order, we have a slower adjustment, as expected. The second and third order approximations present much more oscillation, which could indicate problems in the code.

Obs: the 3rd order figures couldn't be handled by latex. I am uploading the relevant *.eps* files thus.