Costain, Nakov and Petit (2021) - Replication codes

https://github.com/borjapetit/costainnakovpetit2021

This repository includes all the codes required to replicate the results in the paper **"Flattening of the Phillips curve with state-dependent prices and wages"**, by James Costain (Banco de España), Anton Nakov (ECB and CEPR) and Borja Petit (CUNEF).

The model is solved in two steps. First, we use Fortran to solve for the steady state and to linearize the dynamic system. Then, we use Matlab to implement Klein's complex QZ decomposition solution method and to compute the Impulse Response Functions. We also make use of Dynare to compute the best fitting Calvo parameters reported in table 6 of the paper.

Structure of the codes

It is very important that you keep the same structure of folders and subfolders

```
Main directory
   ⌊ compiledfiles
   ⌊ figures
      ⌊ figs
      ⌊ pdfs
   ⌊ matlab
   ⌊ tables
   ⌊ textfiles
      ⌊ _dyn
      ⌊ _irfs
         ⌊ _figs_irfs
      ⌊ _ss
         ⌊ _figs_hist
```

Folder	Description
`../compiledfiles/`	This folder contains the `.mod` files produced at compilation.
`../figures/`	This folder contains the Matlab codes that generate the figures of the paper.
`../matlab/`	This folder contains a number of Matlab codes that are used to implement Klein's decomposition, and some other auxiliary functions.
`../tables/`	This folder contains a set of Matlab codes that produce the tables shown in the paper.
`../textfiles/`	This folder contains text files with data and the solution to the model.

Root	File	Description
../	dynamics.f90	This code computes the Jacobian of the dynamic system and stores it in ../textfiles/_dyn/Vxy_dyn.txt where x and y refer to the specific version solved (see below).
	main.f90	This code contains the execution of the program.
	parameters.f90	This code contains the parameters of the model as well as some general-purpose functions.
	solution.f90	This code solves the firms' and workers' problems and computes the invariant distribution of prices/wages.
	toolkit.f90	This code contains a set of general-purpose functions and subroutines, including optimization routines used in the code.
../figures/	fig_2.m	Generate figures 2 and D1 of the paper.
	fig_3.m	Generate figure 3 of the paper.
	fig_4.m	Generate figure 4 of the paper.
	fig_5.m	Generate figure 5 of the paper.
	fig_6.m	Generate figure 6 of the paper.
	fig_7.m	Generate figure 7 and table 8 of the paper.
	fig_8.m	Generate figure 8 of the paper.
../matlab/	solve_dyn.m	This code reads the .txt files produced with Fortran and implements Klein's QZ decomposition to generate the IRFs.
	extract_dyn.m	Given a Jacobian matrix, this code fills the matrices to configure the problem for Klein's QZ decomposition.
	extract_ss.m	Takes a vector with all the results from the steady-state and fills vectors and matrices.
	kleinsolve.m	Implementation of Klein's QZ decomposition.
	parameters.m	Defines parameters used in other codes.
	plot_irf.m	Generate the figure with IRFs.
	calvow.mod	Dynare code with Calvo’s model.
	calvo_matching.m	Function used to generated table 6.
	stoch_simul.m	Dynare code.
../tables/	table_4.m	Generate Table 4 of the paper.
	table_5.m	Generate Table 5 of the paper.
	table_6.m	Generate Table 6 of the paper.
	table_7.m	Generate Table 7 of the paper.
../textfiles/	calibprams.txt	Values of the calibrated parameters.
	data_pc.mat	Data used in section 4.2.3.
	data_pdfprices.txt	Empirical histogram of price changes.
	data_pdfwages.txt	Empirical histogram of wage changes.

Compilation command

To compile the Fortran codes run the following command:

```
cd [set your own working directory]

gfortran [optional: compilation flags] -J $(pwd)/compiledfiles
toolkit.f90 parameters.f90 solution.f90 dynamics.f90 main.f90 -o lpw
```

You should use the following compilation flags:

```
-fopenmp -O3 -ffixed-line-length-150 -fmax-stack-var-size=1000000
```

You can also compile the codes using Intel's compiler `ifort`:

```
ifort [optional: compilation flags] /object:%cd%\compiledfiles\
/module:%cd%\compiledfiles toolkit.f90 parameters.f90 solution.f90
dynamics.f90 main.f90 -o lpw
```

with standard compilation flags `/openmp` and `/O3`

> **IMPORTANT**
>
> Before compiling the code, make sure you have specified your working directory in the variable `path` in `parameters.f90`. All the input/output operations depend on that variable.

Model versions

The different versions of the model, combining different noise parameters and inflation rates, are named according to `Vxy` where `x` refers to the noise parameters and `y` to the inflation rate. In particular:

Adjustment cost / Inflation rate	**2%**	**-1%**	**0%**	**4%**	**8%**	**-2%**	**1%**
Baseline	V10	V11	V12	V13	V14	V15	V16
Semi-flexible prices and sticky wages	V20	V21	V22	V23	V24	V25	V26
Flexible prices and sticky wages	V30	V31	V32	V33	V34	V35	V36
Sticky prices and semi-flexible wages	V40	V41	V42	V43	V44	V45	V46
Sticky prices and flexible wages	V50	V51	V52	V53	V54	V55	V56
Flexible prices and flexible wages	V60	V61	V62	V63	V64	V65	V66

This table includes all the versions that can be computed, which are a few more than the ones shown in the paper (versions in the second and fourth rows are not in the paper). We also allow the inflation rate to take two different extra values to compute the Phillips curve slope presented in the paper:

Inflation rate	
Inflation rate of 4.63% (US, 1980-2000)	V17
Inflation rate of 2.01% (US, 2000-2020)	V18

While the inflation rates are defined within the codes, the (baseline) noise parameters are taken from the calibrated parameters in `../textfiles/calibparams.txt`. Semi-flexible prices/wages case is computed by diving the corresponding noise parameter by 10, and the flexible prices/wages case by diving the corresponding (baseline) noise parameter by 100. Thus, changing the baseline value of the noise parameters in `../textfiles/calibparams.txt`. will affect the noise parameters in all versions.

Running Fortran codes

The Fortran code offers 7 possibilities:

1. Solve only the steady state

   The user is asked to specify which version to solve
2. Solve the steady state and the dynamics

   The user is asked to specify which version to solve
3. Calibrate the parameters of the model

   The user asked which algorithm to use for calibration: Nelder-Mead or a Newton-based algorithm
4. Solve for different noise parameters

   The code solves the steady-state and the dynamics for all the combinations of noise parameters and the baseline inflation rate. (Version V10, V20, V30, V40, V50, and V60)
5. Solve for different inflation rates

   The code solves the steady-state and the dynamics for all possible inflation rates, and the baseline level of noise parameters. (Versions V10, V11, V12, V13, V14, V15 and V16)
6. Solve for all possible cases

   The code solves the steady-state and the dynamics for all possible combinations of noise parameters and inflation rates that are shown in the paper: V10, V12, V13, V14, V15, V30, V32, V33, V34, V35, V50, V52, V53, V54, V55, V60, V62, V63, V64, and V65. The user can choose to solve all versions, including those not shown in the paper, by modifying `main.f90`.
7. Solve for pre and post inflation rates

   The code solves the steady-state and the dynamics for an inflation rate of 4.63% (US, 1980-2000) and 2.01% (US, 2000-2020). (Versions V17 and V18)

Solving for the steady-state of any version takes around 36 minutes, while the dynamics take around 7 minutes. Thus, solving the model for all the relevant cases (options 6 and 7 above) may take around 16-17 hours. (*)

To save in computing time, the program will automatically load the best initial guess available, using the steady-state solution to version V10 (included in this repository) as the default one.

Each version generates two text files with the solution:

- `Vxy_ss.txt` (251KB) with the state-state prices, the value functions of firms and workers, and the distribution of prices and wages.
    - Stored in `../textfiles/_ss/`
- `Vxy_dyn.txt` (4.2Gb) with the Jacobian of the dynamic system. The program `solve_dyn.m` converts this file into a .mat file, which reduces the file size to 67Mb.
    - Stored in `../textfiles/_dyn/`

Running Matlab codes

Once the solution to one or more versions is generated, we use Matlab to solve for the dynamics of the model implementing Klein's complex QZ decomposition. This is done in the program `solve_dyn.m`.

The first thing the user needs to do is to set the working directory and to specify the case or cases to be solved in the variable `cases`. For example, `cases = [ 10 20 30 ]` will solve the dynamics for versions V10, V20 and V30.

The `solve_dyn.m` program will:

- Load the solution from Fortran (`Vxy_dyn.txt`) and convert it into a `.mat` file (67 Mb)
    - Save `Vxy_dyn.mat` (305 Mb) in folder `../textfiles/_dyn/`
- Generate the histogram of price and wages changes
    - Save the figure (`Vxy_hist.pdf`) in folder `../textfiles/_ss/_figs_hist/`
- Implement Klein's complex QZ decomposition returning the (unique) matrices P and F in:

$$\begin{pmatrix} z_{t+1} \\ k_{t+1} \end{pmatrix} = P \cdot \begin{pmatrix} z_t \\ k_t \end{pmatrix} + \begin{pmatrix} \epsilon_t \\ 0 \end{pmatrix}; \quad d_t = F \cdot \begin{pmatrix} z_t \\ k_t \end{pmatrix}$$

  where k_t is the vector of state variables, z_t the vector of stochastic variables, d_t the vector of jump variables and ϵ_t is the vector of iid shocks. The matrix P is saved as `STATEDYNAMICS`, and the matrix F as `JUMPS`.
- Simulate the path of model's variables during `TT` periods after a monetary policy shock.
- Save the variables' paths, and the `STATEDYNAMICS` and `JUMPS` matrices
    - Save `Vxy_irf.mat` (305 Mb) it folder `../textfiles/_irfs/`
- Generate the IRFs
    - Save the figure (`Vxy_irf.pdf)` in folder `../textfiles/_irfs/_figs_irfs/`

Running `solve_dyn.m` takes around 2 hours and 20 minutes per version, except for those with flexible prices and wages which take around 90 minutes. Computing the IRFs for all relevant cases takes at least 40 hours. (*)

List of files required to generate figures and tables

Generating all the figures and tables included in the paper, requires the following files:

V10_ss.mat (251 Mb)	V10_dyn.mat (68 Mb)	V10_irf.mat (306 Mb)
V12_ss.txt (251 Mb)	V12_dyn.mat (68 Mb)	V12_irf.mat (307 Mb)
V13_ss.txt (251 Mb)	V13_dyn.mat (68 Mb)	V13_irf.mat (306 Mb)
V14_ss.txt (251 Mb)	V14_dyn.mat (68 Mb)	V14_irf.mat (307 Mb)
V15_ss.txt (251 Mb)	V15_dyn.mat (68 Mb)	V15_irf.mat (305 Mb)
V30_ss.txt (251 Mb)	V30_dyn.mat (57 Mb)	V30_irf.mat (299 Mb)
V32_ss.txt (251 Mb)	V32_dyn.mat (57 Mb)	V32_irf.mat (299 Mb)
V33_ss.txt (251 Mb)	V33_dyn.mat (57 Mb)	V33_irf.mat (299 Mb)
V34_ss.txt (251 Mb)	V34_dyn.mat (57 Mb)	V34_irf.mat (299 Mb)
V35_ss.txt (251 Mb)	V35_dyn.mat (57 Mb)	V35_irf.mat (298 Mb)
V50_ss.txt (251 Mb)	V50_dyn.mat (31 Mb)	V50_irf.mat (284 Mb)
V52_ss.txt (251 Mb)	V52_dyn.mat (31 Mb)	V52_irf.mat (286 Mb)
V53_ss.txt (251 Mb)	V53_dyn.mat (31 Mb)	V53_irf.mat (285 Mb)
V54_ss.txt (251 Mb)	V54_dyn.mat (31 Mb)	V54_irf.mat (285 Mb)
V55_ss.txt (251 Mb)	V55_dyn.mat (31 Mb)	V55_irf.mat (283 Mb)
V60_ss.txt (251 Mb)	V60_dyn.mat (20 Mb)	V60_irf.mat (276 Mb)
V62_ss.txt (251 Mb)	V62_dyn.mat (20 Mb)	V62_irf.mat (276 Mb)
V63_ss.txt (251 Mb)	V63_dyn.mat (20 Mb)	V63_irf.mat (276 Mb)
V64_ss.txt (251 Mb)	V64_dyn.mat (20 Mb)	V64_irf.mat (275 Mb)
V65_ss.txt (251 Mb)	V65_dyn.mat (20 Mb)	V65_irf.mat (275 Mb)
V17_ss.txt (251 Mb)	V17_dyn.mat (68 Mb)	V17_irf.mat (306 Mb)
V18_ss.txt (251 Mb)	V18_dyn.mat (68 Mb)	V18_irf.mat (306 Mb)

Additionally, when solving version V10, the program will also generate the following files used to generated figures 3 and 8

_V10_lambda_ss.txt (24 Kb)
_V10_pi_ss.txt (24 Kb)
_V10_piw_ss.txt (7 Mb)
_V10_rho_ss.txt (98 Mb)

(*) Computing time based on:

- Mac Mini M1 2020 running under MacOS 11.4
- GNU Fortran compiler version 10.2.0
- 8 threads for parallelization
- Matlab R2020a 64-bit (with Dynare 4.6.4)