Alexandria - User Guide

for Python and Matlab editions

Romain Legrand

Version 1.0

Alexandria - User Guide

Cover illustration: depiction of the ancient library of Alexandria, unknown artist.

To my wife, Mélanie.

To my sons, Tristan and Arnaud.

Foreword

Bayesian vector autoregressions have become the cornerstone of time-series econometrics. Despite the need for these models, there exist to these days very few software applications. On the commercial side, Eviews proposes some limited Bayesian VAR features since version 8. Only a few applications are available and implementation comes as a blackbox. On the non-commercial side, I had the priviledge to develop the BEAR toolbox when I was working at the European Central Bank, between 2014 and 2018. BEAR is a Matlab-based software for Bayesian time-series econometrics proposing a wide range of state-of-the art Bayesian econometric methodologies. It was initially developed internally to meet the needs of the European Central Bank, but it quickly gained notoriety and has now become a reference tool for many central banks and financial institutions across the world. BEAR remains to these days the most comprehensive application in the field of Bayesian time-series econometrics.

Despite this success, I eventually became unsatisfied with BEAR. One reason is technical: BEAR is developed under a simple functional programming paradigm. While working with functions only is fine for a small project, this approach finds its limits for large software projects like BEAR. In this case, proper object-oriented programming becomes a necessity to maintain efficient and well-structured code. Considering this question, it eventually became apparent that the only solution was the creation of a new software from scratch, built on the object-oriented paradigm.

The second reason is economical: BEAR is developed in Matlab, a mathematical language that remains to these days prominent among economists and econometricians. Matlab is a great platform permitting fast development and efficient execution of econometric applications, but it is also a commercial product. Matlab licenses are expensive and not affordable to everyone. Using Matlab thus excludes de facto many students and researchers, especially from emerging countries. In an era of fair opportunity and open source projects, I think this situation is not an option anymore.

For these reasons, I developed a new software called Alexandria. Alexandria provides a solution to these two issues. It is developed in pure object-oriented progamming, building a solid structure for long-term development. Also, Alexandria comes in two flavours: a Matlab version for veteran econometricians; and a Python version that offers a free, open source alternative that makes the product accessible to everyone, regardless of financial constraints. The two versions are strictly similar, so that the user really can pick whichever language is preferred.

A tribute to the ancient library of Alexandria, the name of the software reflects two main ideas. First, it is a library – in the computer science sense of the term – providing codes, programmes and applications for easy access to Bayesian time-series econometrics models. Second, it has the ambition of becoming an extensive collection of knowledge – as was the library of Alexandria in its time – with the hope of providing in the long run all the major models in the field of Bayesian time-series econometrics. I hope that you will enjoy using Alexandria as much as I enjoyed developing it.

Romain Legrand

Contents

CHAPTER 1

Installing Alexandria, Python edition

1.1 Python: an overview

Python is a high-level, interpreted, and general-purpose programming language. It owes its name to the BBC TV show "Monty Python's Flying Circus". It was initially designed in the late 1980's by Guido van Rossum at Centrum Wiskunde and Informatica in the Netherlands. It was first released in 1991 and developed further by the Python Software Foundation.

Relatively marginal until the 2000's, Python has been gaining in popularity since then. It represents today one of the most widely used programming languages, aside with other popular languages such as C# or Java. The Python community is estimated to more than 8 million users today.

There are several reasons which explain the success of Python. Python is powerful, yet one of the simplest languages to learn. It emphasizes readibility and conciseness, making development considerably faster under Python than under alternative frameworks. It is general, meaning it can be used to develop virtually any kind of application. Thanks to these qualities, many world-class software companies have chosen Python to write their applications, including Youtube, Google, Instagram, Netflix, Reddit, Spotify or Quora, to name just a few.

Python has also become increasingly popular amongst the data science community, as major applications were developed in Python or provided API to become Python-compatible. Scikit-Learn, TensorFlow, Torch, Keras, Numpy, Pandas or Matplotlib are examples of widely used data science applications that contributed to increase the popularity of Python among data scientists. While Matlab remains to these days dominant in the econometrics community, there is no doubt that Python will also play an increasingly important role in the field, making it a first-choice language for an open source econometrics software.

1.2 The Anaconda distribution

In theory, installing Python is straightforward. On the Python webpage, one can download the install file for the latest release and then proceed to the installation. In practice however, this option is not recommended. First, the basic Python installation is minimal and provides very little in terms of development tools. The only available feature is a small programme called Idle (Figure 1.1) which is hardly more than a raw Python console. Idle provides no code editor, no variable explorer and no graphic visualisation, which makes it practically unusable for development purposes.

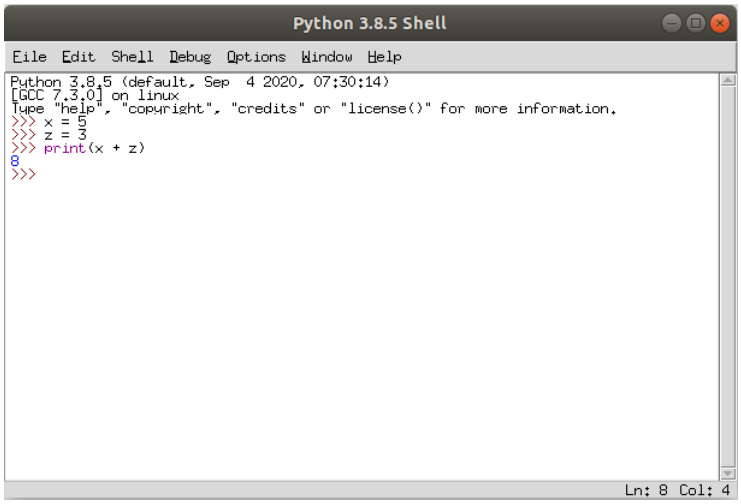


Figure 1.1: The Idle Python shell

Second, as any general programming language, base Python is extremely limited and requires additional libraries to provide suitable functionalities. With a basic installation those libraries need to be installed and maintained manually. This is impractical and creates a high risk of eventually generating conflicts between libraries, which may completely ruin the whole installation.

For these reasons it is recommended to use instead a distribution called Anaconda, which has by now become the industry standard. Anaconda is a free, open source software suite for scientific computing suitable for Windows, Linux and macOS. Anaconda makes Python more accessible thanks to a simplified Python installation and package management system.

Concretely, Anaconda does the following. First, it automatically installs Python 3 on the system, which avoids any kind of manual installation. The distribution also comes with over 250 packages automatically installed, comprising virtually any library that may be ever needed. Most importantly, it comes with its own management system that automatically handles the libraries updates and conflicts.

Second, Anaconda comes with a suite of Python development softwares, including two great applications: Spyder and Jupyter Notebook. Spyder (Figure 1.2) is a free and open-source integrated developement environment (IDE) for Python. It provides a code editor (left panel), a Python console for interactive execution (bottom right panel), and a variable/figure explorer (top right panel). Spyder provides everything needed for editing, analysis, debugging and visualisation. It also comes close to Matlab and RStudio in its design, so users of these programmes will find it easy to make the move to Python.

Jupyter Notebook (Figure 1.3) is a web-based environment for creating notebook documents containing text, code, and data. It combines the features of a classical text editor with an interactive execution environment allowing to execute code and display visual elements such as tables and figures. Jupyter makes it easy to design clear and beautiful data science documents. While Spyder is primarily intended for development, Jupyter is more suitable for the execution and visualisation of existing applications. In the end however, both applications can be used to write and run Python programmes.

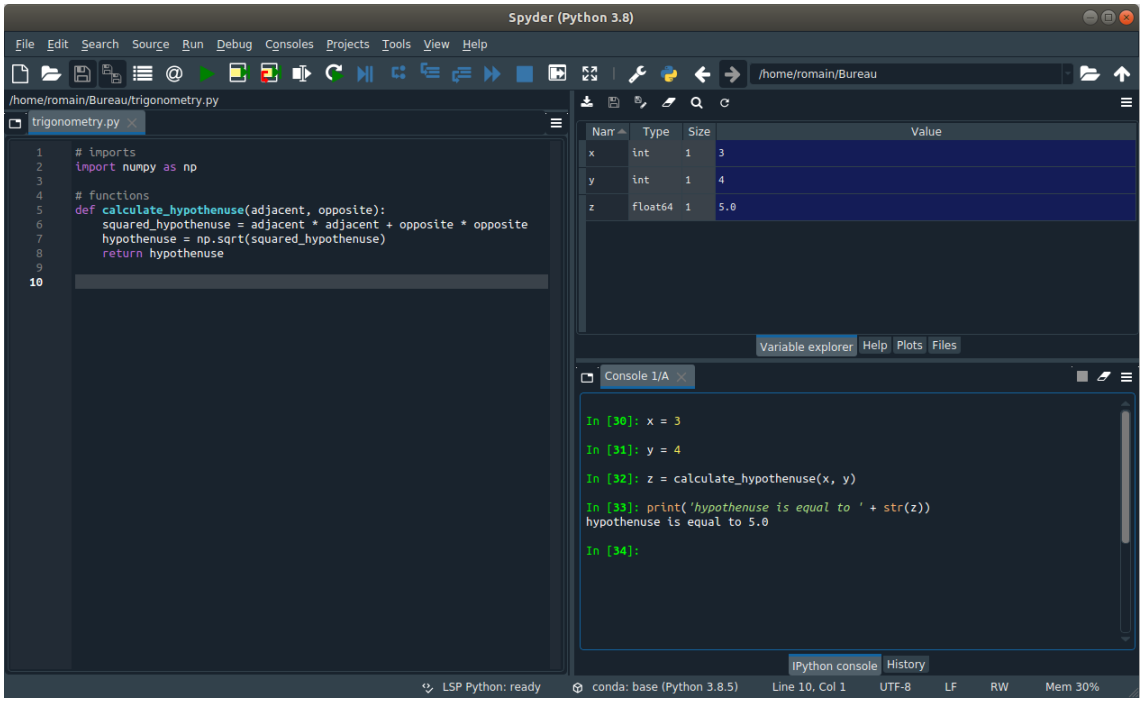


Figure 1.2: The Spyder environment

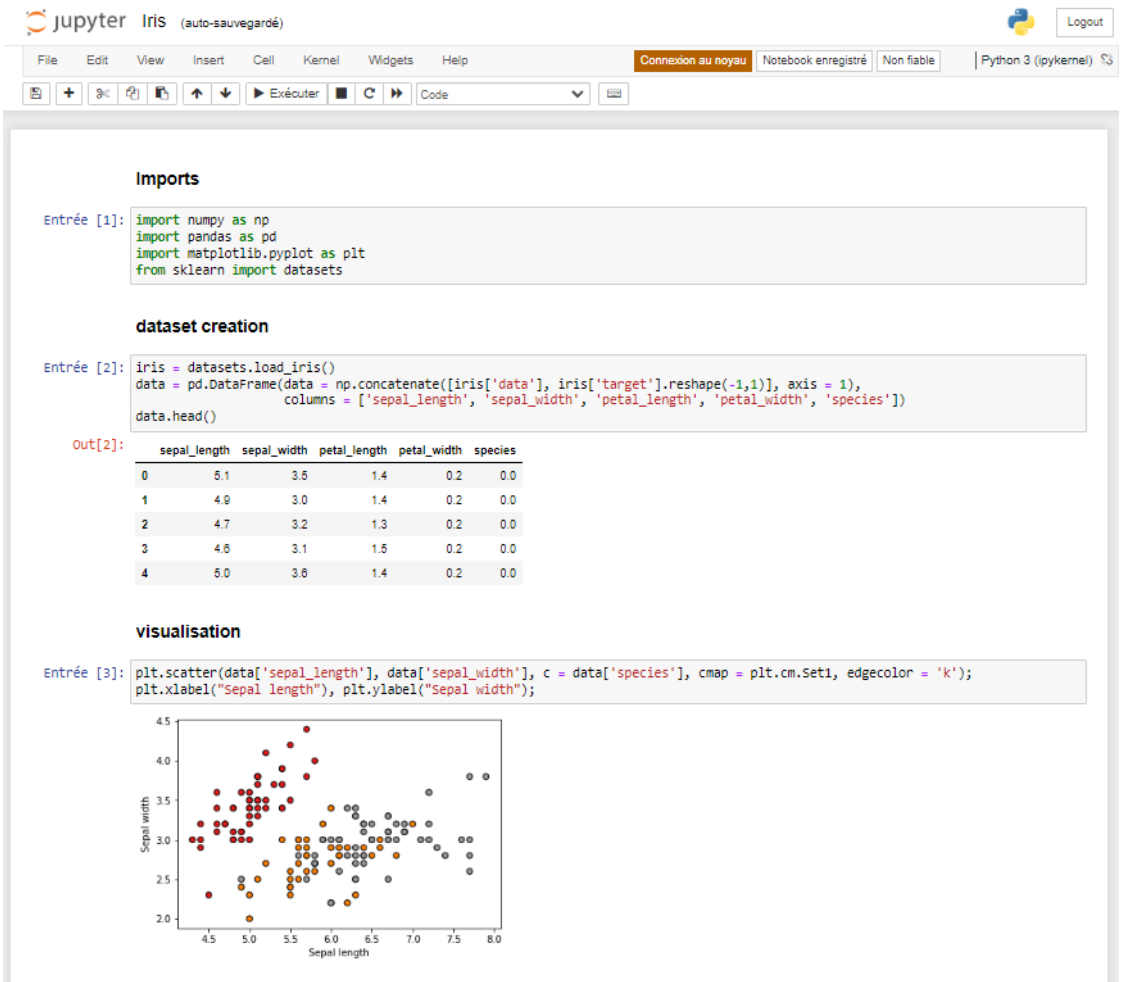


Figure 1.3: A Jupyter notebook

1.3 Anaconda installation on Windows

To install Anaconda on Windows, go first to the Anaconda distribution webpage (Figure 1.4). You may either register with an email or click "skip registration" to continue without email. You should then reach the download page (Figure 1.5). The page should automatically propose to download the latest version corresponding to Windows. Click on the download button to initiate the download of the installation file.

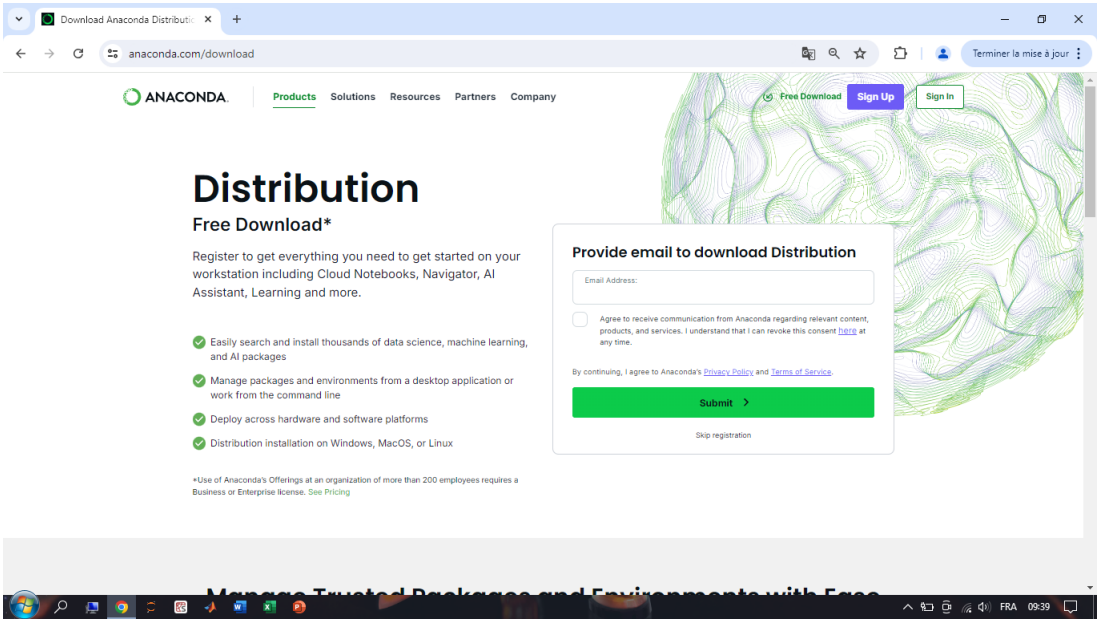


Figure 1.4: The Anaconda distribution webpage

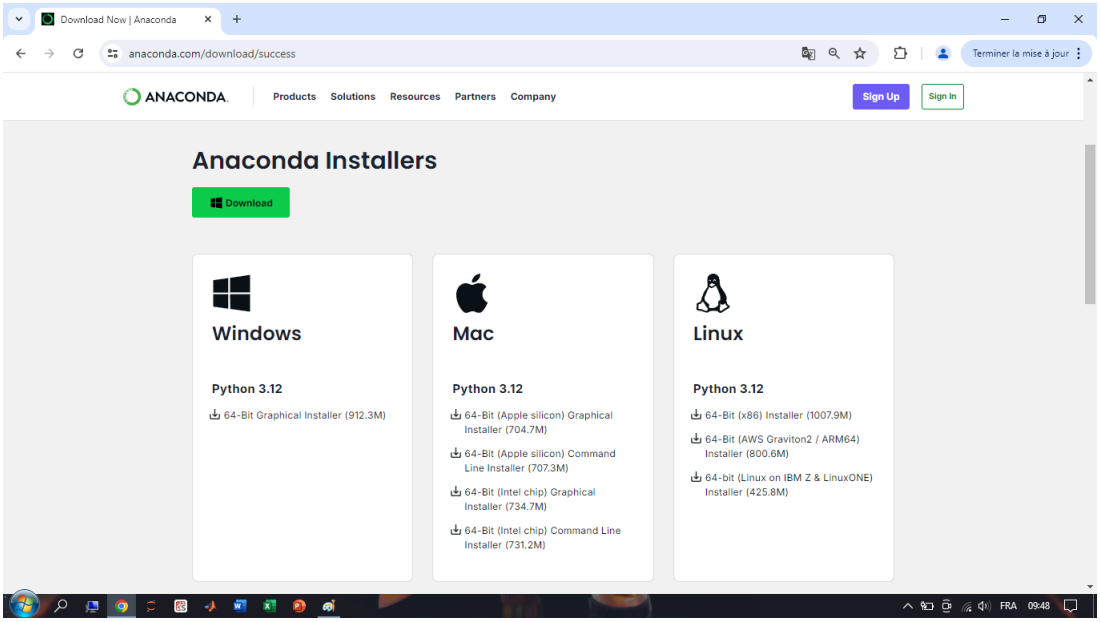


Figure 1.5: The Anaconda download webpage

Once download is completed, navigate to the folder containing the file (it should be the Downloads folder), and double-click on the file to start installation (Figure 1.6). Follow the steps and agree when prompted to eventually complete the setup. If you experience issues in the process, you may consult the Anaconda installation webpage.

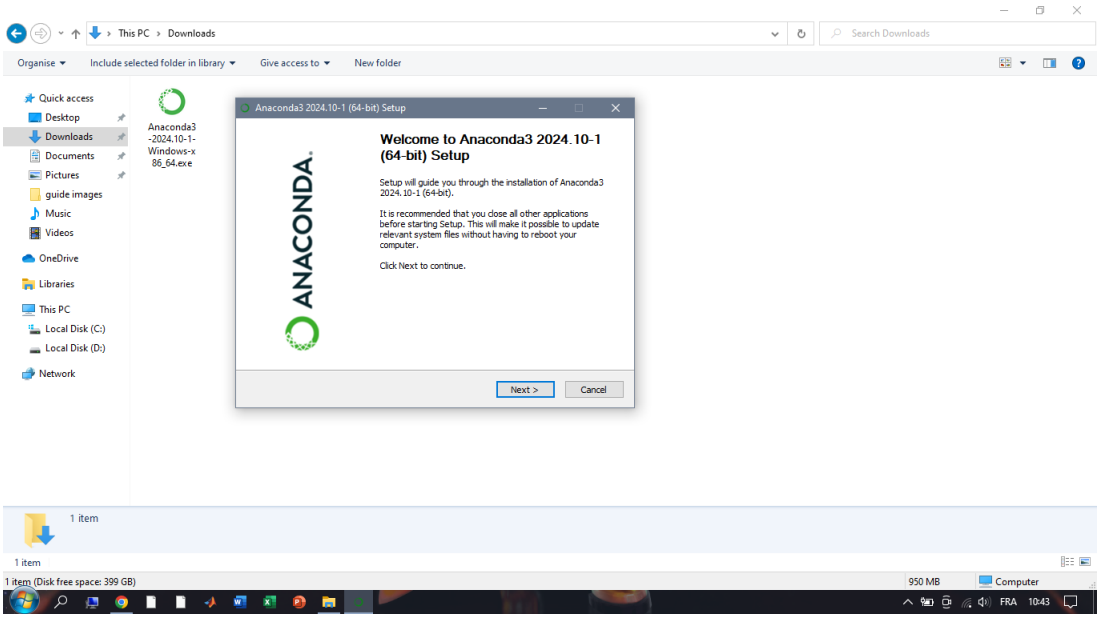
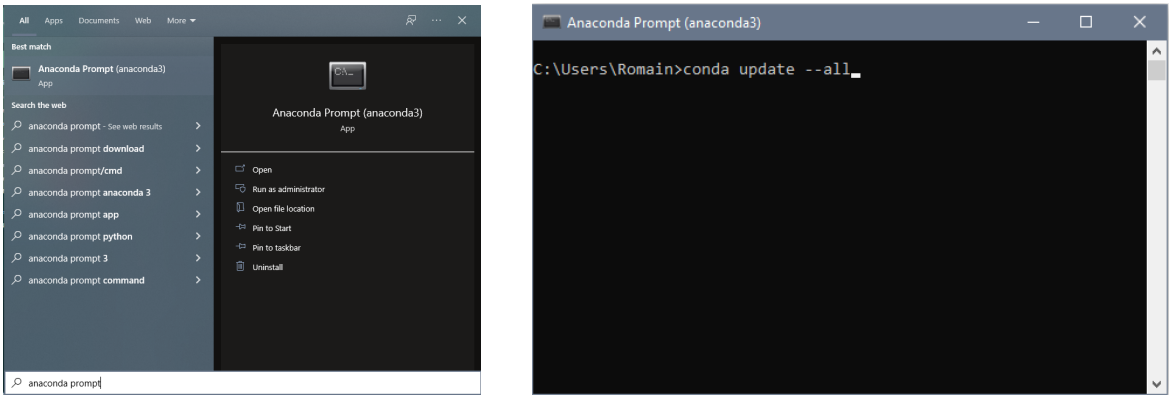


Figure 1.6: Anaconda installation on Windows

Anaconda is now installed on your computer along with Python and all the libraries you need. However, most of its material is outdated. As a final step, it is thus necessary to update the whole setup. To do so we will use the conda terminal, a facility provided by Anaconda. To open the conda terminal, search for "Anaconda prompt" in the Windows search bar (Figure 1.7, panel (a)). Then in the terminal, execute the command (Figure 1.7, panel (b)):

```
conda update --all
```

Agree when prompted and Anaconda will update Python and all the libraries, managing all possible conflicts.

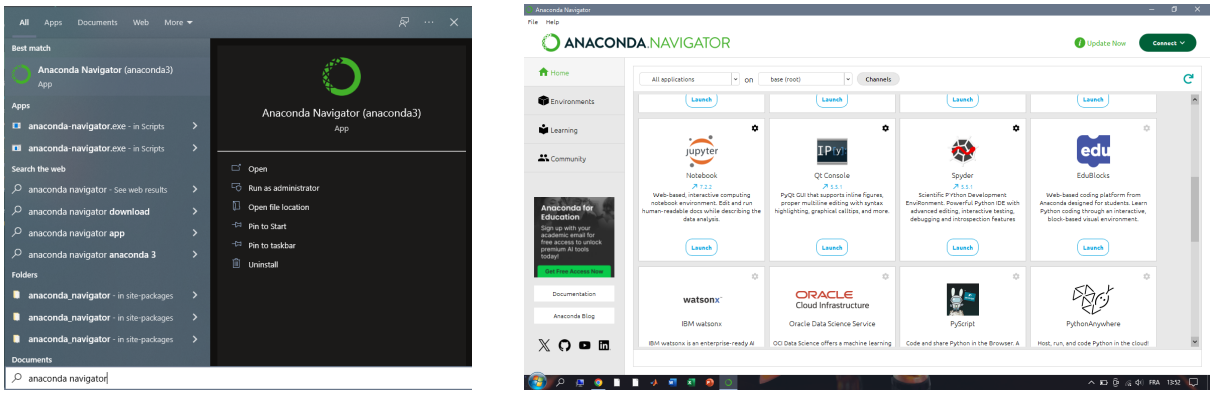


(a) Searching for the Anaconda prompt

(b) Updating Anaconda

Figure 1.7: The Anaconda prompt

Finally, to open Spyder or Jupyter Notebook you can use the Anaconda navigator. To open it, search for "Anaconda navigator" in the Windows search bar (Figure 1.8, panel (a)). Then in the navigator use the "Launch" button of Jupyter or Spyder to start the application (Figure 1.8, panel (b)). Alternatively, you may create shortcuts to launch these applications directly. In the Windows search bar, type "anaconda navigator", "jupyter notebook" or "spyder", then right-click and choose "Pin to Start" or "Pin to taskbar". You may also right-click and choose "Open file location", right-click the application icon in the folder and select "Send to" > "Desktop (create shortcut)".



(a) Searching for the Anaconda navigator

(b) The Anaconda navigator

Figure 1.8: The Anaconda navigator

1.4 Anaconda installation on Linux/macOS

As Linux and macOS are both based on Unix, they follow similar installation procedures. This section outlines the steps to follow, but for more details you may consult the Anaconda installation webpage.

To install Anaconda, go first to the Anaconda distribution webpage (Figure 1.4). You may either register with an email or click “skip registration” to continue without email. You should then reach the download page (Figure 1.5). The page should automatically propose to download the latest version corresponding to Mac and Linux. Click on the download button to initiate the download of the installation file, and wait until download is completed.

The installation file is a bash file (a file with ".sh" extension). To execute it, we will use the terminal. If you are unfamiliar with the terminal, simply type "terminal" in the application search bar (Figure 1.9) and click the terminal icon to open a terminal.

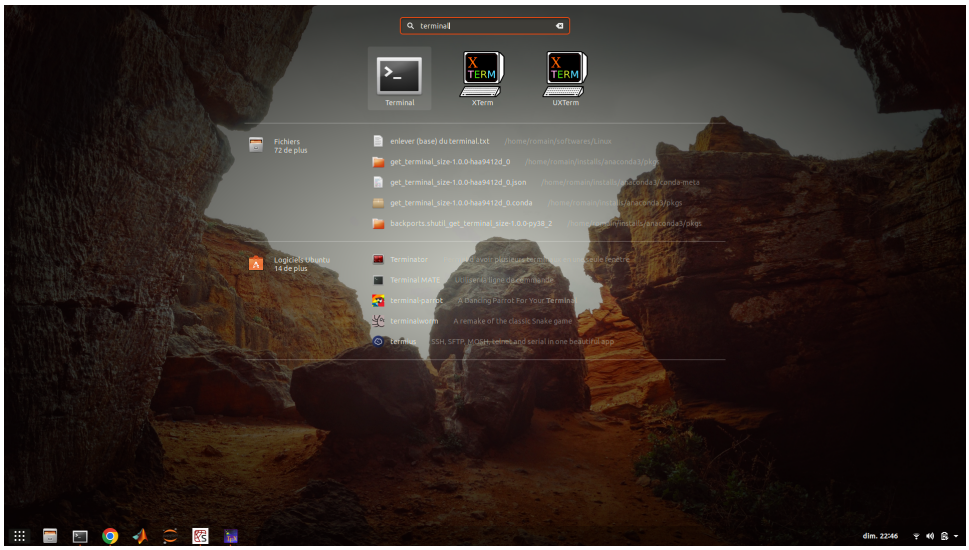


Figure 1.9: Searching for the terminal in Linux

Once the terminal is opened, search for the path of the folder where the installation file has been downloaded (it should be the Downloads folder), and note the name of the installation file. Then in the terminal execute the command:

```
bash path-to-folder/name-of-file
```

So for instance if the path to the folder containing the installation file is "~/Downloads" and the installation file is "Anaconda3-2024.10-1-Linux-x86_64.sh", then the command to execute is:

```
bash ~/Downloads/Anaconda3-2024.10-1-Linux-x86_64.sh
```

Executing this command in the terminal (Figure 1.10) will start the installation, and instructions will appear in the terminal. Follow the steps, agree when prompted and keep the default options to eventually complete the setup.

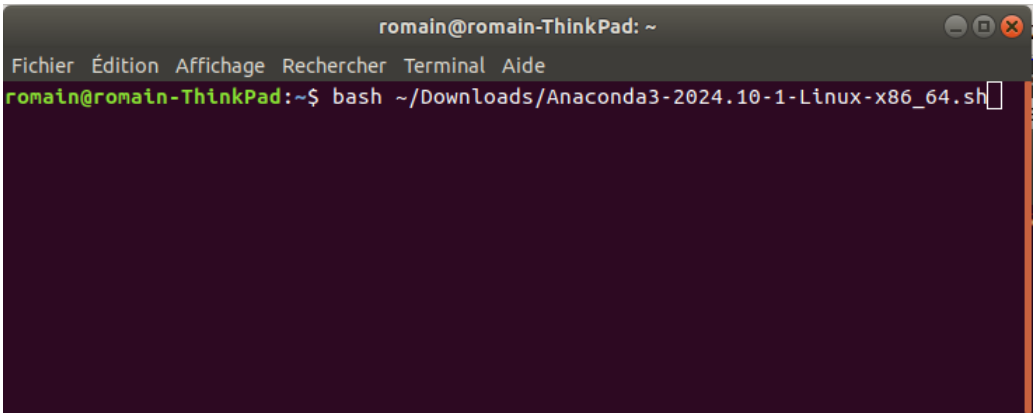


Figure 1.10: Executing the Anaconda bash install file

Anaconda is now installed on your computer along with Python and all the libraries you need. However, most of its material is outdated. As a final step, it is thus necessary to update the whole setup. To do, open a new terminal and execute the command (Figure 1.11):

```
conda update --all
```

Agree when prompted and Anaconda will update Python and all the libraries, managing all possible conflicts.

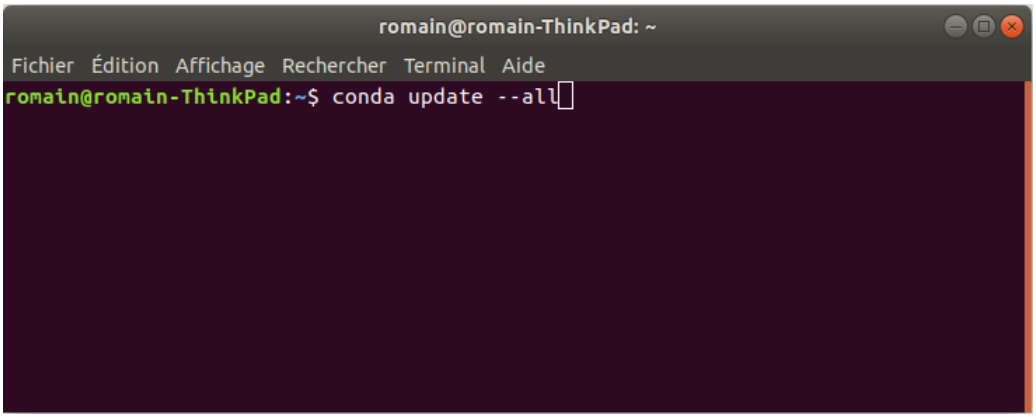


Figure 1.11: Updating Anaconda

Finally, to open Spyder or Jupyter Notebook you can use the Anaconda navigator. To access it, open a terminal and execute the command:

```
anaconda-navigator
```

This opens the navigator (Figure 1.12). You can then launch Jupyter Notebook or Spyder by pressing the corresponding Launch buttons in the navigator.

You may also directly start Jupyter or Spyder from the terminal by executing the commands:

```
jupyter-notebook
```

or:

```
spyder
```

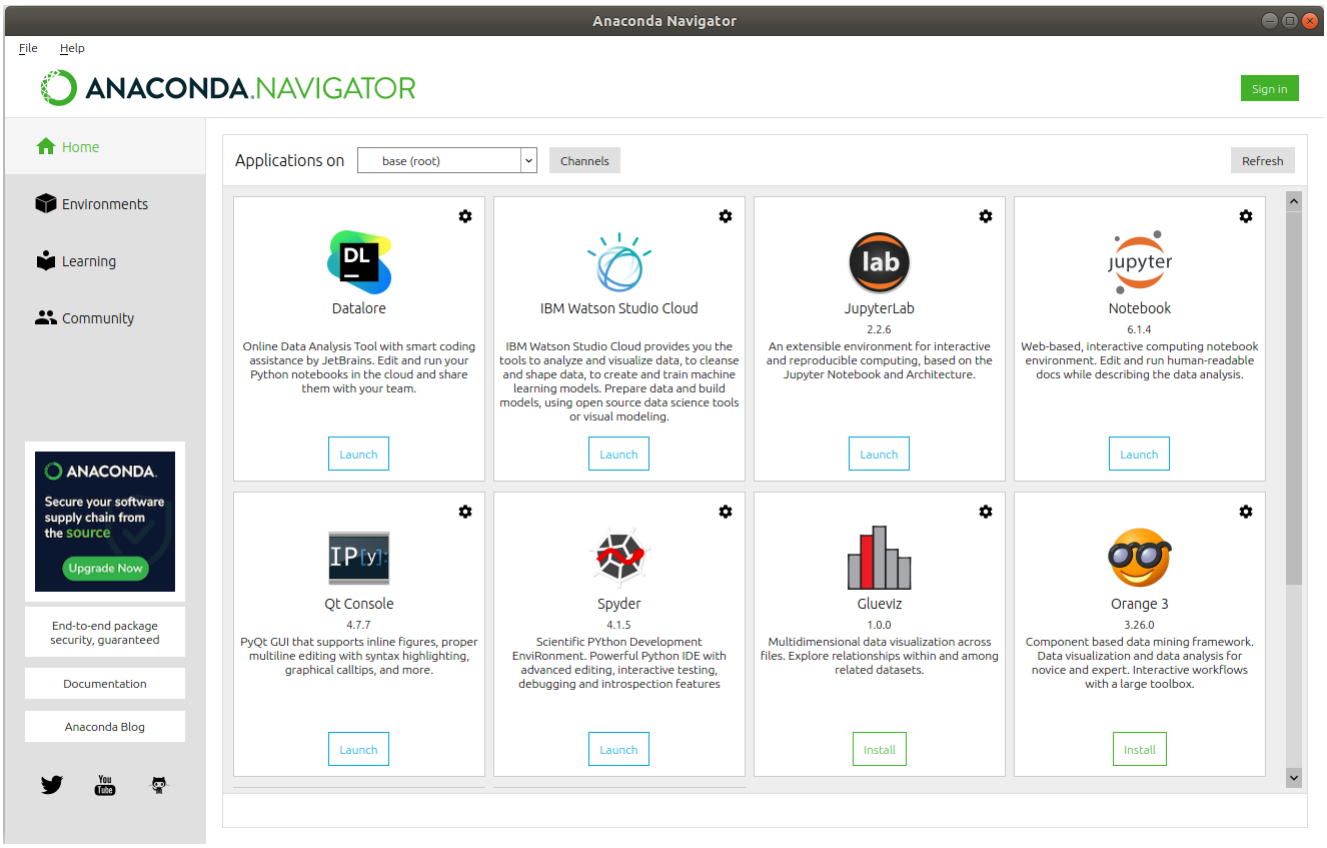


Figure 1.12: The Anaconda navigator

1.5 Alexandria: local installation

There are two options to install Alexandria: a local install that consists in creating a local copy of the folder containing the programmes for each of your projects, or a permanent install that installs the programme on your system once for all. For beginners, it may be easier to use the local installation as it is straightforward and more intuitive (one copy of the application is created for each project).

To proceed to a local install, you first need to recover the folder containing the Alexandria programmes. There are two possibilities to do so. You can go to the Alexandria website, navigate to Downloads on the left menu, and in the Toolbox section click on the link for the Python edition of the software (Figure 1.13). This will donwload a zip file containing the toolbox programme folder.

Figure 1.13: The Downloads page of the Alexandria website

Alternatively, you can visit the Github page of the project, click on the alexandria-python repo (it is a public repository), then click on (Figure 1.14):

```
Code -> Download ZIP
```

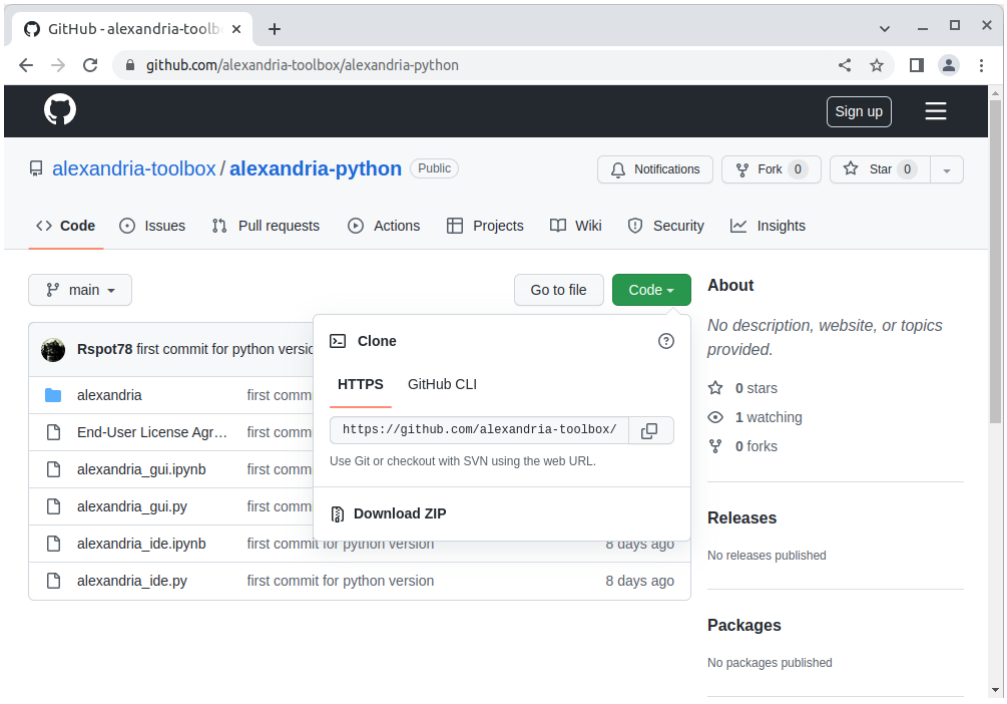


Figure 1.14: The Github repo for the Python version of the toolbox

In both cases, unzip the ZIP archive to obtain a folder named "alexandria-python". This is your local install folder that contains all the programmes to run Alexandria. This folder also constitutes the basis of

your project folder (see section 3.1), so you can rename it the way you want to match your project name and move it to any directory you wish. For instance, you may rename the folder "my_project" and place it in D:\my_project (Figure 1.15).

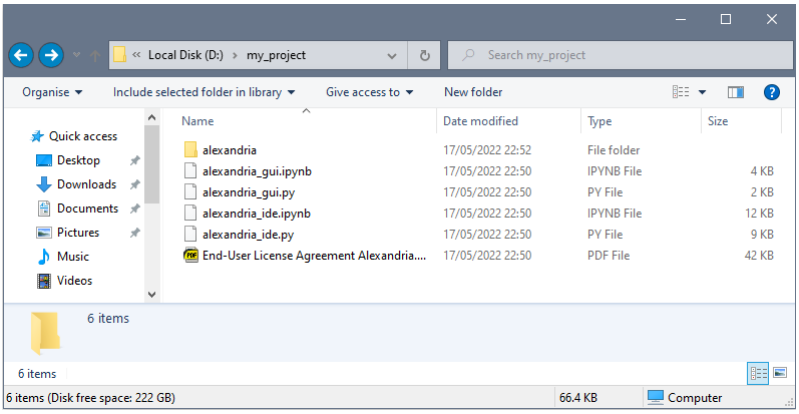


Figure 1.15: The local install folder after renaming

1.6 Alexandria: permanent installation

To install Alexandria permanently, the most convenient option consists in using PyPi. PyPi is a web repository of softwares for the Python programming language. It permits easy installation of third-party projects on personal computers. To install Alexandria from PyPi, first open a terminal. If your operating system is Windows, open the conda terminal (refer to section 1.3). If your operating system is Linux/macOS, open a regular terminal (refer to section 1.4). Then execute the command (Figure 1.16):

```
pip install alexandria-python
```

This will install alexandria on your computer. Note that you need an internet connection to proceed to the installation from PyPi. Also, at any time, you may uninstall Alexandria by using the command:

```
pip uninstall alexandria-python
```

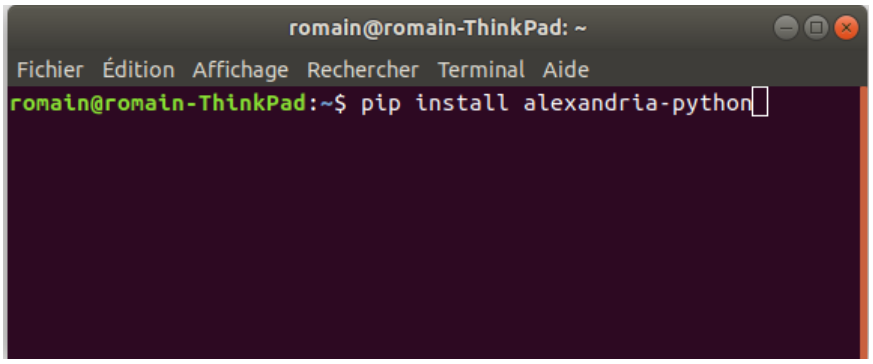


Figure 1.16: Installing Alexandria with pip

Once Alexandria is installed on your system, you can move to the creation of your project folder (section 3.1).

CHAPTER 2

Installing Alexandria, Matlab edition

2.1 Matlab: an overview

Matlab is a numerical software and programming language developed by the firm MathWorks. The name is an abbreviation of "Matrix Laboratory", stressing the primary orientation of the language towards linear algebra applications. Matlab was initially developed by Cleve Moler, a math professor at the University of New Mexico, as a hobby for his students. The programme was initially a simple matrix calculator distributed for free in universities. As popularity started growing, the programme got developed further and was first released as a commercial product in 1984. As of today, Matlab has evolved to include many features beyond linear algebra, including numerical optimization, symbolic algebra, statistical applications, algorithms and graphical visualisation. In 2020, Mathworks claims more than 4 million Matlab users worldwide, mostly from the fields of engineering, science, and economics.

There are several reasons explaining the popularity of Matlab. First, as a software specialised in mathematical applications, Matlab's syntax proves simpler than that of general languages like Python. In fact, its syntax is close to mathematical writing, making the language especially attractive for users with a scientific background. Developing and testing in Matlab is also considerably faster than with other languages, thanks to its simple and concise syntax.

Second, Matlab is powerful. It benefits from – litterally – decades of development, and its routines are highly optimized. Matlab can prove several times faster than Python in numerical applications, which represents a strong asset for computationally intensive programmes.

Third, Matlab benefits from Simulink, a graphical programming environment for modeling, simulating and analyzing multidomain dynamical systems. Simulink is widely used in the scientific industry and explains much of Matlab's success within the engineering community.

For these reasons Matlab has been extensively used in diverse fields of engineering ranging from signal processing, image treatment and control systems to algorithmic finance, computational biology and econometrics. Recently, Matlab has also tried to capitalize on the recent rise of data science, but it has been facing fierce competition from open source languages such as Python.

On the downside, it should be noted that as a commercial product, Matlab requires the purchase of a license which can prove quite expensive. A standard professional license costs $2150. For individual users, Matlab proposes a Home license for $149. Students may benefit from the cheapest option with a student license costing $49 (these prices may vary depending on which country you live). For more details on licensing, you may consult the Mathworks webpage on pricing and licensing.

2.2 Matlab installation on Windows

Make sure you own a valid Matlab licence and have a Mathworks account activated before you initiate the installation of Matlab. Once this is done, the first step consists in downloading the Matlab installer from

the Mathworks download webpage. Enter your account credentials and complete the steps to obtain the installation programme.

To initiate the installation, double-click on the setup.exe icon within the installation programme. This will open the installation navigator (Figure 2.1). Follow the steps to install Matlab, register your license and create a desktop shortcut. Once this is done, Matlab's installation should be complete. To start Matlab, simply double-click on the desktop shortcut.

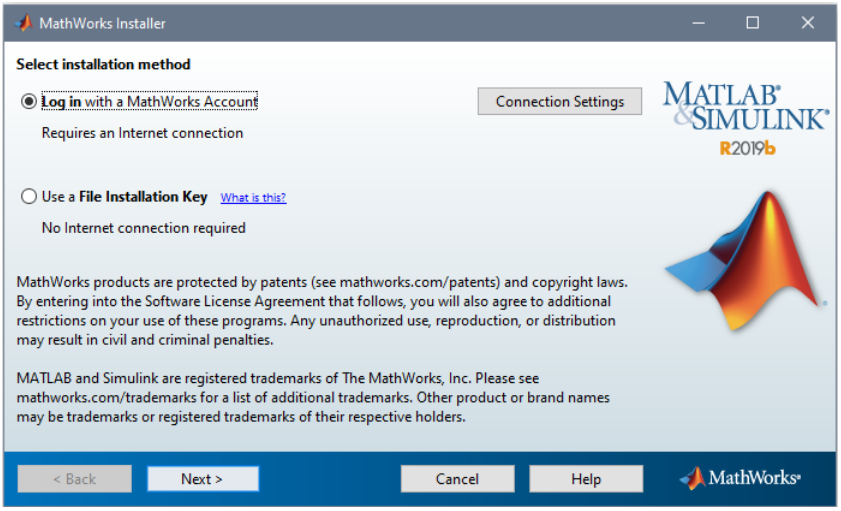


Figure 2.1: The Matlab installation navigator

If you experience issues or wish furher details about the installation process, you may consult the Mathworks help page on installation, or the Mathworks video on Windows installation.

2.3 Matlab installation on Linux/macOS

Installing Matlab on Unix operating systems can be tricky, and unfortunately Mathworks does not provide much official installation support. For Linux, Matlab is only available for recent editions of Ubuntu, Debian, Redhat and Suse. On macOS, Matlab can be installed on macOS Catalina, macOS Mojave and macOS High Sierra. Before you start the installation, make sure you own a valid Matlab licence and have a Mathworks account activated. Then the first step consists in downloading the Matlab installer from the Mathworks download webpage. Enter your account credentials and complete the steps to obtain the installation programme.

If you are on Linux, once the download is over, open a terminal (refer to section 1.4 if you are unfamiliar with the terminal) and navigate to the folder containing the download. So for instance if the path to the folder containing the installation file is "~/Downloads", execute the command:

```
cd ~/Downloads
```

Then initiate the installation by executing the command:

```
sudo ./install
```

You may be prompted to enter your username and password. This will open the installation navigator (Figure 2.1). Follow the steps to install Matlab and register your license. Once this is done, Matlab's installation should be complete. To start Matlab, you may then execute the following command in a terminal:

```
matlab
```

If this fails, it means that Matlab did not generate the required symbolic links during the installation. In this case, Matlab must be started by specifying the full path to the bin folder within the installation directory. For instance, if Matlab is installed in /usr/local/MATLAB/R2020b, then it should be started with the command:

```
/usr/local/MATLAB/R2020b/bin/matlab
```

If you experience issues during the installation process, you may find some additional information on the Mathworks help page on installation, on this Mathworks forum and on this Linux community webpage. If you experience issues to start Matlab, you may consult the Mathworks help page on Matlab start for Linux.

If you are on macOS, once the download is over, the installation file should come as a zip archive. Once extracted, you should obtain a file called InstallForMacOSX. Double-click it to initiate the installation. This will open the installation navigator (Figure 2.1). Follow the steps to install Matlab and register your license. Once this is done, Matlab's installation should be complete. To start Matlab, you may either double-click the Matlab icon in your Matlab installation folder, or start it from the terminal by specifying the full path to the bin folder within the installation directory. For instance, if Matlab is installed in /Applications/MATLAB/R2020b, then it should be started with the terminal command:

```
/Applications/MATLAB/R2020b/bin/matlab
```

If you experience issues during the installation process, you may find some additional information on the Mathworks help page on installation, or on this support web page. If you experience issues to start Matlab, you may consult the Mathworks help page on Matlab start for macOS.

2.4 Alexandria: local installation

There are two options to install Alexandria: a local install that consists in creating a local copy of the folder containing the programmes for each of your projects, or a permanent install that installs the programme on your system once for all. For beginners, it may be easier to use the local installation as it is straightforward and more intuitive (one copy of the application is created for each project).

To proceed to a local install, you first need to recover the folder containing the Alexandria programmes. There are two possibilities to do so. You can go to the Alexandria website, navigate to Downloads on the left menu, and in the Toolbox section click on the link for the Matlab edition of the software (Figure 2.2). This will donwload a zip file containing the toolbox programme folder.

Alternatively, you can visit the Github page of the project, click on the alexandria-matlab repo (it is a public repository), then click on (Figure 2.3):

```
Code -> Download ZIP
```

In both cases, unzip the ZIP archive to obtain a folder named "alexandria-matlab". This is your local install folder that contains all the programmes to run Alexandria. This folder also constitutes the basis of your project folder (see section 3.2), so you can rename it the way you want to match your project name and move it to any directory you wish. For instance, you may rename the folder "my_project" and place it in D:\my_project (Figure 2.4).

Figure 2.2: The Downloads page of the Alexandria website

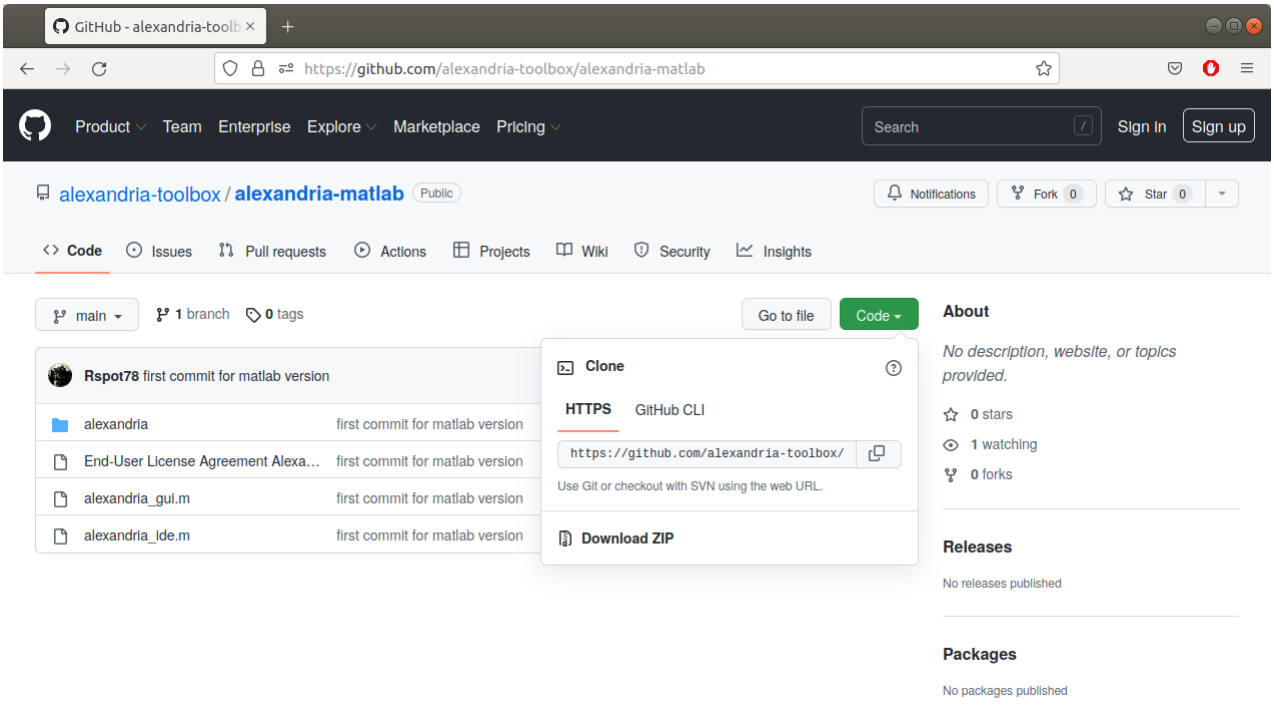


Figure 2.3: The Github repo for the Matlab version of the toolbox

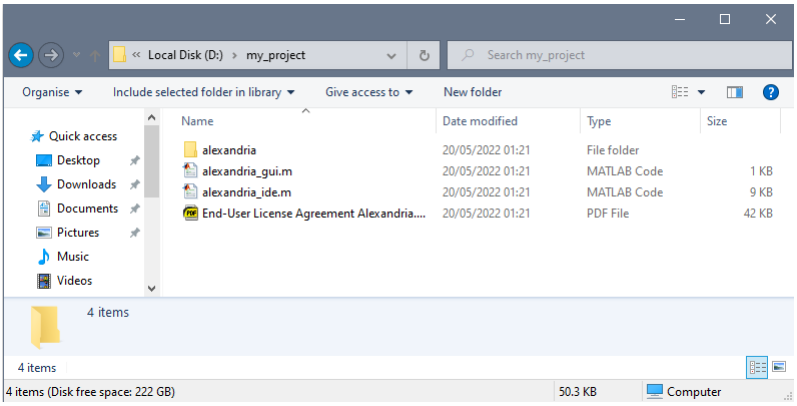


Figure 2.4: The local install folder after renaming

2.5 Alexandria: permanent installation

Installing Alexandria permanently on Matlab is easy. Start Matlab, then in the top menu bar select "Add-Ons", then "Get Add-Ons" (Figure 2.5).

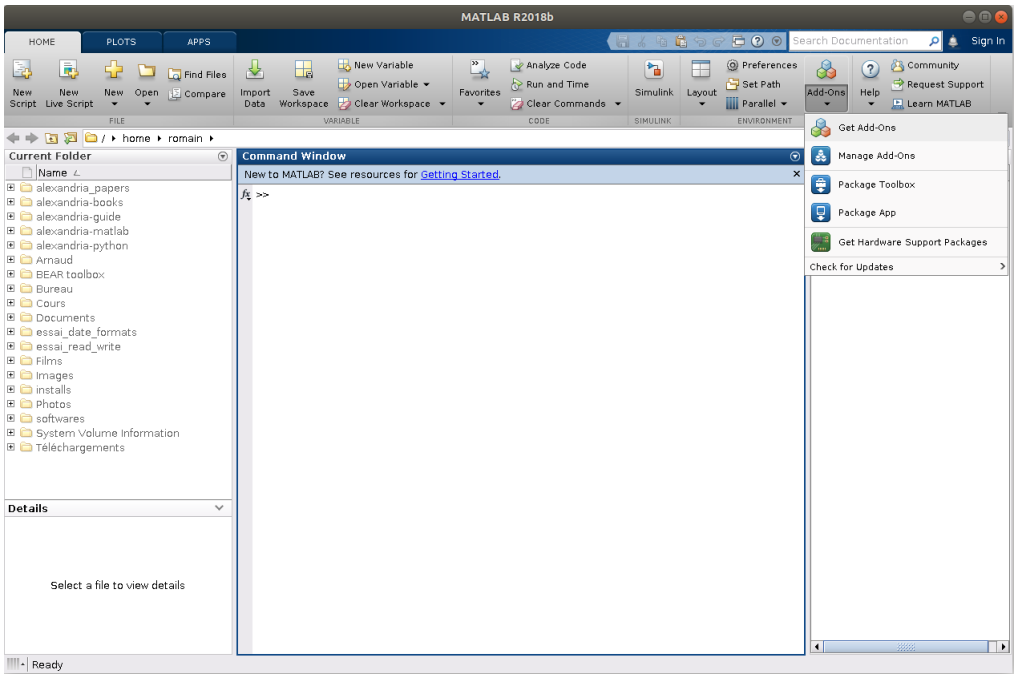


Figure 2.5: Selecting Matlab Add-Ons

This opens the Add-On Explorer. In the top search bar, search for "Alexandria". Select the first choice: Alexandria, by Romain Legrand. Finally, click on the "Add" button on the right to complete the installation (Figure 2.6). This requires a valid Mathworks account. Once the procedure is over, Alexandria is installed permanently on Matlab. At anytime you may uninstall Alexandria by selecting "Add-Ons" in the top menu bar, then "Manage Add-Ons", then select the rollmenu (triple dots) of Alexandria and select "uninstall".

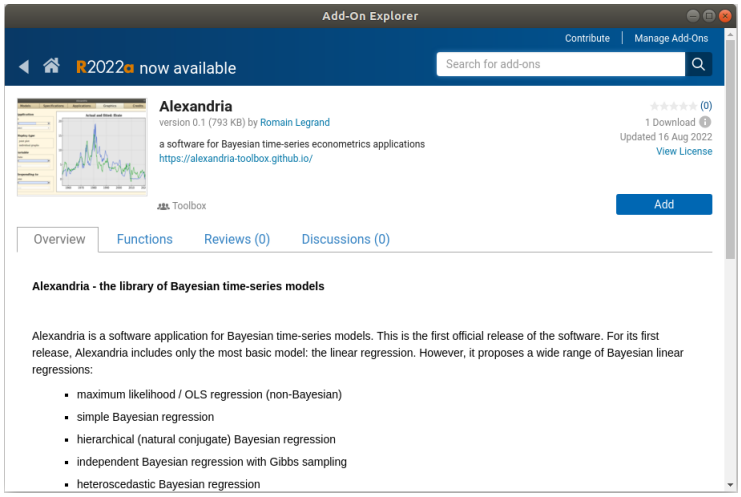


Figure 2.6: Installing the Alexandria toolbox

With Alexandria installed on your system, you can move to the creation of your project folder (section 3.2).

CHAPTER 3

Preparing the project

3.1 Creating the project folder: Python edition

If you opted for a local installation of Alexandria, your project folder is already initiated. It should look like Figure 3.1:

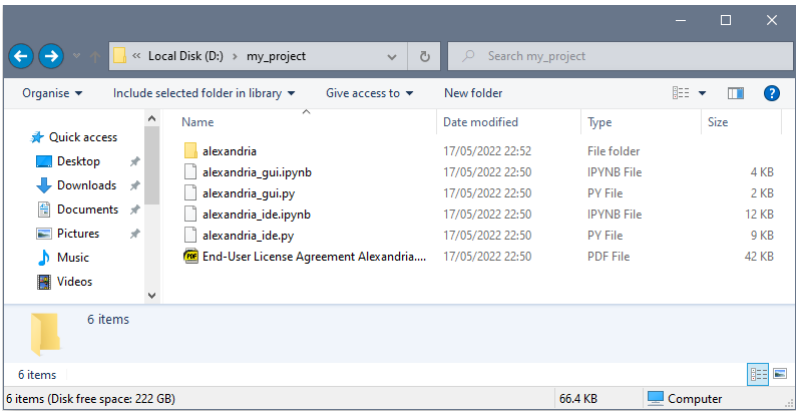
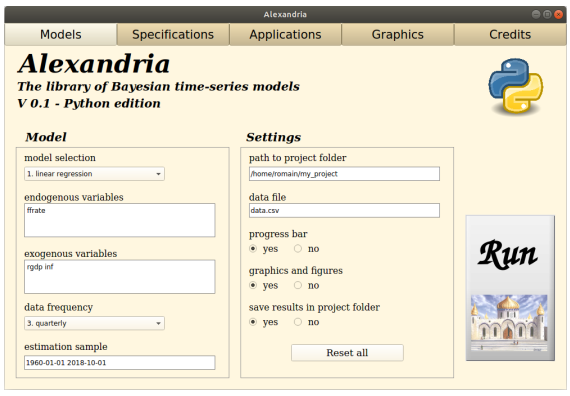


Figure 3.1: Initial project folder with local installation

Make sure you read and agree with the End-User License Agreement. You can then delete the file if you wish (note that are still bound by the Agreement if you do so). Do not delete or modify the alexandria folder and its contents as it contains the software programmes.

What to do with the four remaining files depends on how you plan to use Alexandria. Alexandria can be run either from a graphical user interface (Figure 3.2, panel (a)), or from an integrated development environment (Figure 3.2, panel (b)).



(a) Alexandria GUI

(b) Alexandria IDE

Figure 3.2: Graphical User Interface and Integrated Development Environment

The graphical user interface (GUI) is a simple graphical window in which the user inputs the model parameters. It is recommended for users with limited programming experience. The integrated development environment (IDE) is a python script in which the user enters manually the model information directly as code. It is recommended for users with some programming experience. Additional information on the GUI and IDE can be found in chapters 4 and 5.

If you plan to use the GUI, then keep either the file alexandria_gui.ipynb for a use in Jupyter Notebook, or the file alexandria_gui.py for a use in Spyder. If you plan instead to use the IDE, then keep either the file alexandria_ide.ipynb for a use in Jupyter Notebook, or the file alexandria_ide.py for a use in Spyder. Whatever your choice, the other three files can be deleted. So for instance if you plan to use Alexandria with the GUI in Jupyter notebook (the recommended choice for novice users), your project folder will look like Figure 3.3:

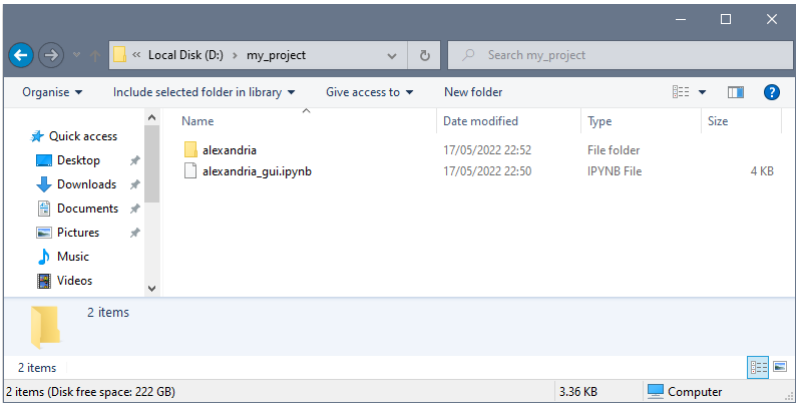


Figure 3.3: Updated project folder with local installation

If you opted for a permanent installation of Alexandria, you don't have a project folder yet. So, create a new folder that will contain your project, place it in any convenient directory and call it as you wish. It is not necessary to copy all the programmes for the software as for a local install since they are permanently installed on your system. It is however recommended for convenience that you include a copy of one of the four files alexandria_gui.ipynb, alexandria_gui.py, alexandria_ide.py or alexandria_ide.ipynb in your project folder, depending on how you plan to use Alexandria. Please refer to section 1.5 to see how you can download these files.

So assume for instance that you create a project folder called my_project and place it in D:\my_project. You then create in it a copy of the file alexandria_gui.ipynb to use Alexandria with the GUI. Your initial project folder should then look like Figure 3.4:

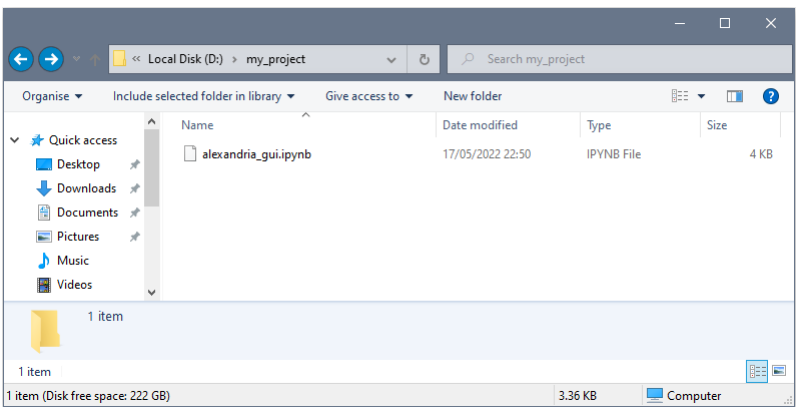


Figure 3.4: Project folder with permanent installation

3.2 Creating the project folder: Matlab edition

If you opted for a local installation of Alexandria, your project folder is already initiated. It should look like Figure 3.5:

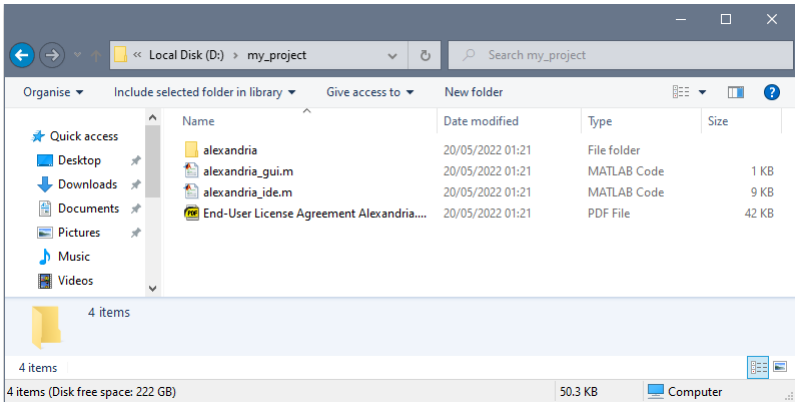
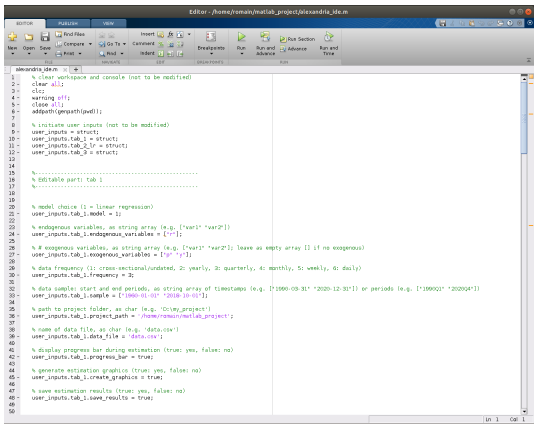


Figure 3.5: Initial project folder with local installation

Make sure you read and agree with the End-User License Agreement. You can then delete the file if you wish (note that are still bound by the Agreement if you do so). Do not delete or modify the alexandria folder and its contents as it contains the software programmes.

What to do with the two remaining files depends on how you plan to use Alexandria. Alexandria can be run either from a graphical user interface (Figure 3.2, panel (a)), or from an integrated development environment (Figure 3.2, panel (b)).

(a) Alexandria GUI



(b) Alexandria IDE

Figure 3.6: Graphical User Interface and Integrated Development Environment

The graphical user interface (GUI) is a simple graphical window in which the user inputs the model parameters. It is recommended for users with limited programming experience. The integrated development environment (IDE) is a python script in which the user enters manually the model information directly as code. It is recommended for users with some programming experience. Additional information on the GUI and IDE can be found in chapters 4 and 5.

If you plan to use the GUI, then keep the file alexandria_gui.m. If you plan instead to use the IDE, keep the file alexandria_ide.m. Whatever your choice, the other file can be deleted. So for instance if you plan

to use Alexandria with the GUI (the recommended choice for novice users), your project folder will look like Figure 3.7:

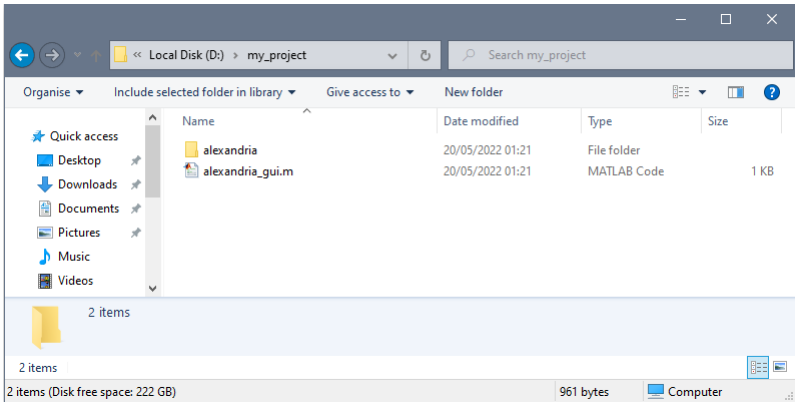


Figure 3.7: Updated project folder with local installation

If you opted for a permanent installation of Alexandria, you don't have a project folder yet. So, create a new folder that will contain your project, place it in any convenient directory and call it as you wish. It is not necessary to copy all the programmes for the software as for a local install since they are permanently installed on your system. You may however include a copy of either alexandria_gui.m or alexandria_ide.m for convenience, though there are ways to work without these files completely. Please refer to section 2.4 to see how you can download these files, and to sections 4.2 and 5.2 for precisions on how to use Alexandria without having to copy these files.

So assume for instance that you create a project folder called my_project and place it in D:\my_project. You then create in it a copy of the file alexandria_gui.m to use Alexandria with the GUI. Your initial project folder should then look like Figure 3.8:

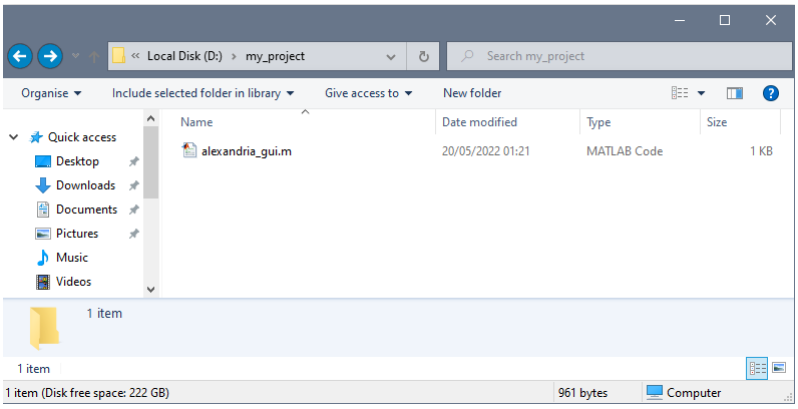


Figure 3.8: Project folder with permanent installation

3.3 Creating the base data file for the model

Once your project folder is initiated, the next step consists in creating the data file that constitutes the base input of your project. This base dataset must be placed directly in your project folder, or, if you want to avoid to have many inputs files in the project folder, in some created subfolder that you may e.g. call inputs. If you place the data file in an input subfolder, your project folder may look like Figure 3.9:

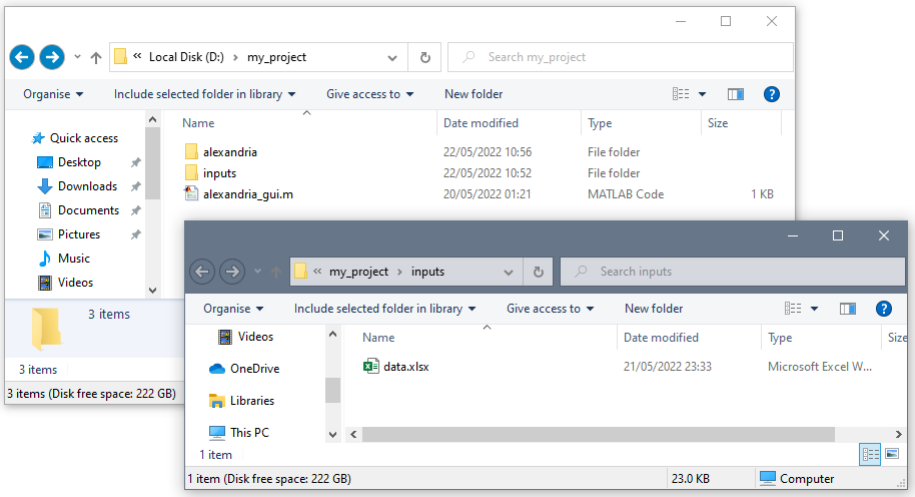
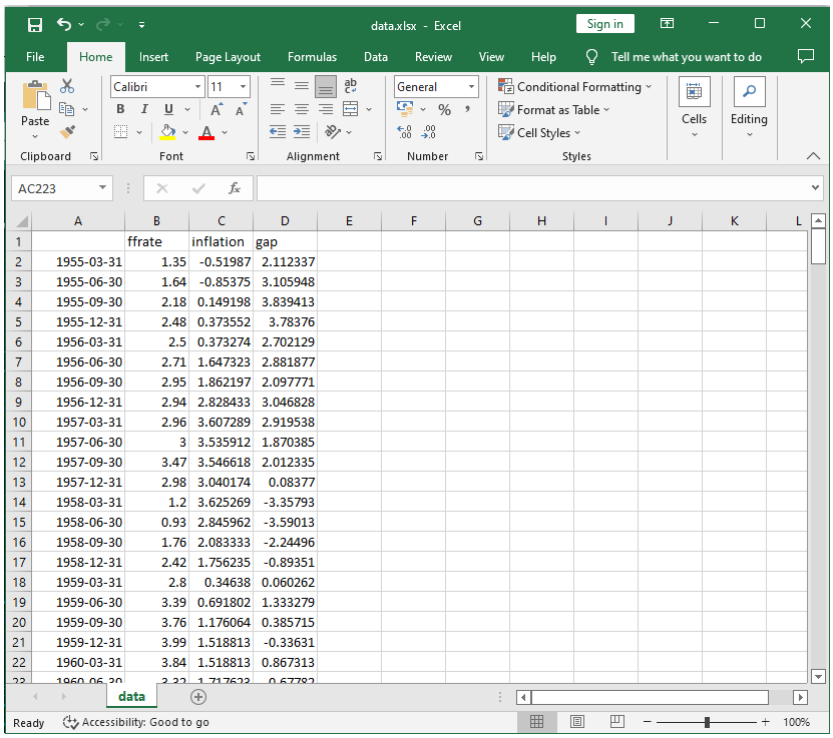


Figure 3.9: Project folder with data file

The data file must contain all the variables that are involved in the estimation of your model: endogenous and exogenous variables, but also other variables that enter model estimation. This includes heteroscedastic variables for the heteroscedastic Bayesian regression, and the proxys for the Bayesian proxy-SVAR. Do not include constants or trends in the data file, as Alexandra handles these automatically (see section 4.4).

Alexandria accepts three types of files: Excel spreadsheets with either the .xls or .xlsx file extension, and also .csv files which can be easily obtained from any open source spreadsheet such as LibreOffice Calc. A typical dataset will look like Figure 3.10:



	A	B	C	D
1		ffrate	inflation	gap
2	1955-03-31	1.35	-0.51987	2.112337
3	1955-06-30	1.64	-0.85375	3.105948
4	1955-09-30	2.18	0.149198	3.839413
5	1955-12-31	2.48	0.373552	3.78376
6	1956-03-31	2.5	0.373274	2.702129
7	1956-06-30	2.71	1.647323	2.881877
8	1956-09-30	2.95	1.862197	2.097771
9	1956-12-31	2.94	2.828433	3.046828
10	1957-03-31	2.96	3.607289	2.919538
11	1957-06-30	3	3.535912	1.870385
12	1957-09-30	3.47	3.546618	2.012335
13	1957-12-31	2.98	3.040174	0.08377
14	1958-03-31	1.2	3.625269	-3.35793
15	1958-06-30	0.93	2.845962	-3.59013
16	1958-09-30	1.76	2.083333	-2.24496
17	1958-12-31	2.42	1.756235	-0.89351
18	1959-03-31	2.8	0.34638	0.060262
19	1959-06-30	3.39	0.691802	1.333279
20	1959-09-30	3.76	1.176064	0.385715
21	1959-12-31	3.99	1.518813	-0.33631
22	1960-03-31	3.84	1.518813	0.867313

Figure 3.10: Project spreadsheet in Excel format

The spreadsheet is organised in a simple way: the first column contains the dates labels, while the first row

is used for the names of the variables. The rest of the speadsheet contains the numerical values. Consider now these elements in details.

- **Variable names**

The first row of the spreasdheet contains the labels of the variables. It must start in cell B1 and expand on the right, as in Figure 3.9. Names can contain letters, digits and underscores, but no spaces or special characters. They cannot start with a digit. Also, it is recommended to keep names short (less than 15 characters) to avoid minor display issues with the toolbox.

- **Date labels**

The first column of the spreadsheet contains the date labels. It must start in cell A2 and develop downward, as in Figure 3.9. Alexandria accepts six different date formats: annual, quarterly, monthly, weekly, daily, and if none of these applies, a cross-section/undated format. Except for the undated format, the dates can be expressed either in international date format (as in Figure 3.9), or in periodic format. Precisely, the different formats must be specified as follows:

annual data: dates can be specified either in international date format of the form yyyy-mm-dd, of in periodic format of the form yyyy. For instance:
international date format: 2000-12-31, 2001-12-31, 2002-12-31...
periodic format: 2000, 2001, 2002...
It does not matter which day is chosen within the year for the international date format, as long as there is only one observation per year.

quarterly data: dates can be specified either in international date format of the form yyyy-mm-dd, of in periodic format of the form yyyy + Q + quarter. For instance:
international date format: 2000-03-31, 2000-06-31, 2000-09-31...
periodic format: 2000Q1, 2000Q2, 2000Q3...
The format is case-sensitive so that 2000Q1 is valid, while 2000q1 isn't. It does not matter which day is chosen within the quarter for the international date format, as long as there is only one observation per quarter.

monthly data: dates can be specified either in international date format of the form yyyy-mm-dd, of in periodic format of the form yyyy + M + month. For instance:
international date format: 2000-01-31, 2000-02-28, 2000-03-31...
periodic format: 2000M1, 2000M2, 2000M3...
The format is case-sensitive so that 2000M1 is valid, while 2000m1 isn't. It does not matter which day is chosen within the month for the international date format, as long as there is only one observation per month.

weekly data: dates can be specified either in international date format of the form yyyy-mm-dd, of in periodic format of the form yyyy + W + week. For instance:
international date format: 2000-01-07, 2000-01-14, 2000-01-21...
periodic format: 2000W1, 2000W2, 2000W3...
The format is case-sensitive so that 2000W1 is valid, while 2000w1 isn't. It does not matter which day is chosen within the week for the international date format, as long as there is only one observation per week. Also, any number of weeks is acceptable for each year as long as it is at most 53.

daily data: dates can be specified either in international date format of the form yyyy-mm-dd, of in periodic format of the form yyyy + D + day. For instance:
international date format: 2000-01-03, 2000-01-04, 2000-01-05...
periodic format: 2000D3, 2000D4, 2000D5...
The format is case-sensitive so that 2000D1 is valid, while 2000d1 isn't. Alexandria uses calendar days,

not business days. hence December 31, 2000 is 2000D366, not 2000D261. Your dataset doesn't need to include all the days in the year, and any day can be missing. This will typically happen for instance if you consider business days only so that weekends are ignored.

cross-sectional/undated data: observations must be labelled as integer values. For instance:
1, 2, 3, 4 ...
The list does not have to start at 1, though that is probably the option that makes most sense.

For consistency reasons Alexandria always works with end-of-period dates. If the dataset does not follow this convention, Alexandria will automatically offset the dates to match this format. So for instance if you use quarterly data and uses the date 1960-01-01 for the first quarter of 1960 (as in Figure 3.10), Alexandria will shift it to 1960-03-31. This mostly impacts the spreadsheet/graphical outputs of the toolbox and has no consequence on estimation.

- **Data**

The remainder of the spreadsheet contains the data itself. The dataset is a rectangular array corresponding to the dates for the rows and to the variables for the columns, as shown in Figure 3.10. The data has to be numerical and may not include missing values.

3.4 Other datafiles: forecasts

If you run forecasts as an application for your model, you may need to provide additional data in a separate file. To do so, create a new spreadsheet and give it the name you want (e.g. data_forecast). Similar to the data file, the prediction data file can be of type .xlsx, .xls, or .csv. Also, you may place the file either in the project folder directly, or in subfolder if convenient. Assuming you place the file in a subfolder called inputs, your project folder looks like Figure 3.11:

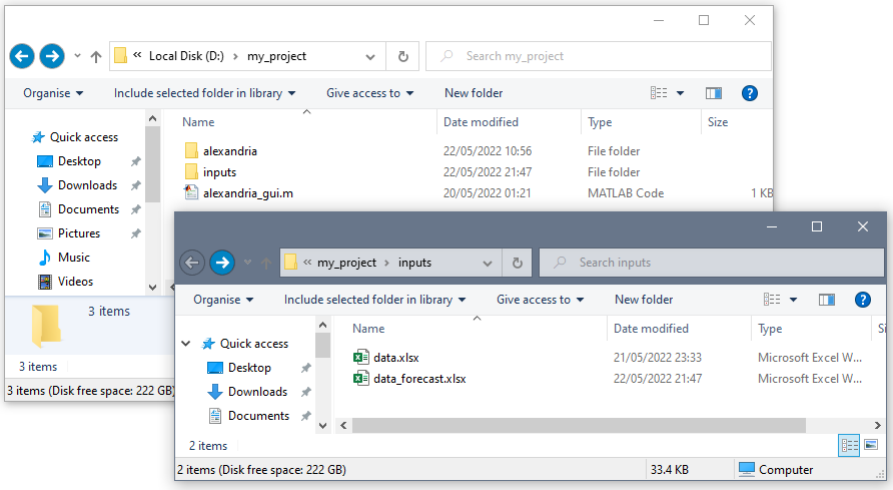


Figure 3.11: Forecast datafile in project folder

A typical forecast data file is depicted in Figure 3.12. The file is overall similar to the data file introduced in section 3.3. One difference is that the left columns does not have to provide dates as Alexandria generates prediction dates automatically. Instead, use integer values for the periods, as shown in the figure.

What must be in the file depends on the class of model being estimated.

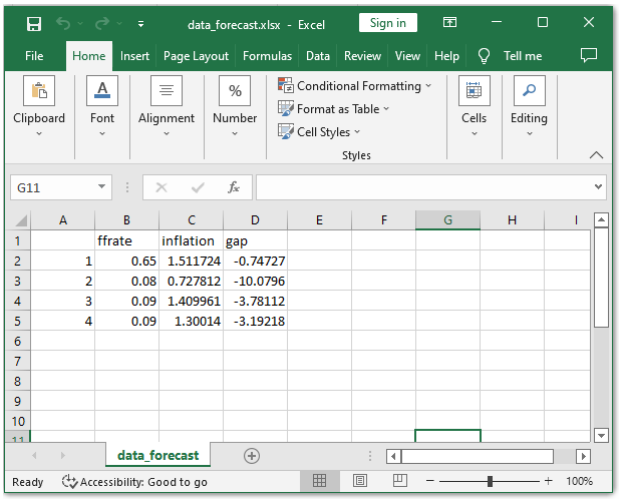


Figure 3.12: Example forecast datafile

- **Linear regression**

If the model you estimate is a linear regression, providing the forecast data file is mandatory. The file must contain the exogenous variables since they are necessary to produce the forecasts (except for constants and trends: these are handled automatically). Values for the endogenous variable may or may not be provided, depending on whether forecast evaluation criteria is selected. If yes, endogenous values are required to be used as counterfactuals for the computation of forecast evaluation criteria.

For instance, assume the data in Figure 3.12 is that of a linear regression with "ffrate" as the endogenous variable and "inflation" and "gap" as the exogenous variables. Then the latter two variables must be provided to generate the forecasts. "ffrate" is only necessary if forecast evaluation criteria is selected.

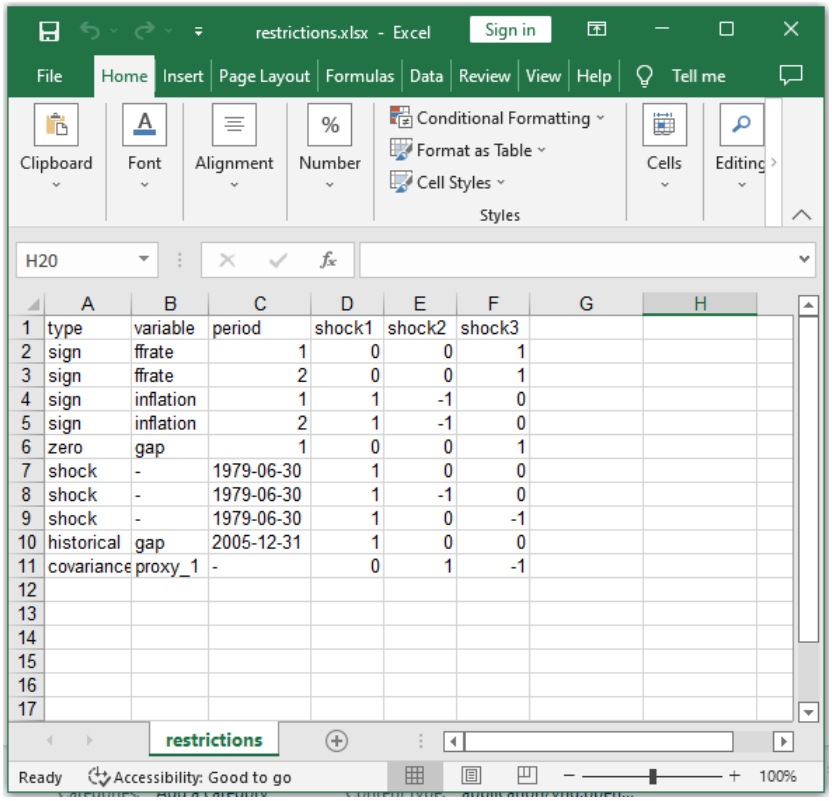
- **Vector autoregression**

If the model you estimate is a vector autoregression, the forecast data file is not necessarily mandatory. It can be provided if forecast evaluation criteria are selected. In this case, values for all the endogenous variables must be provided as counterfactuals. If there are any exogenous variables (other than constant and trends), it is possible to provide values for them which will then be used as predictors. If no value is provided, the model will build its forecast by replicating the last exogenous sample value by default. If neither forecast evaluation criteria nor exogenous variables are involved in the forecast, it is acceptable not to provide a forecast file at all.

For instance, assume the data in Figure 3.12 is that of a vector autoregression with "ffrate" and "inflation" as the endogenous variables, while "gap" is used as an additional exogenous regressor. If forecast evaluation criteria is selected, values must be provided for the first two variables. Since "gap" is an exogenous, the provided values will be used as exogenous predictors for the forecasts. It would be possible not to provide values for "gap", in which case the final sample value would be replicated for the forecasts.

3.5 Other datafiles: structural identification by restrictions

If you select a Bayesian VAR model with a structural identification by restrictions (see section 4.5), you must provide a file that details these restrictions. The file can be of type .xlsx, .xls, or .csv, and may be placed in the project folder or any convenient subfolder. A typical restriction file is depicted in Figure 3.13.



	A	B	C	D	E	F	G	H
1	type	variable	period	shock1	shock2	shock3		
2	sign	ffrate	1	0	0	1		
3	sign	ffrate	2	0	0	1		
4	sign	inflation	1	1	-1	0		
5	sign	inflation	2	1	-1	0		
6	zero	gap	1	0	0	1		
7	shock	-	1979-06-30	1	0	0		
8	shock	-	1979-06-30	1	-1	0		
9	shock	-	1979-06-30	1	0	-1		
10	historical	gap	2005-12-31	1	0	0		
11	covariance	proxy_1	-	0	1	-1		
12								
13								
14								
15								
16								
17								

Figure 3.13: Example restriction datafile

The table has a specific structure that must be complied with. The first three columns must be labelled as “type”, “variable” and “period”. Then there must be one additional column per structural shock in the model, i.e. per endogenous variable. These columns must be labeled as “shock1”, “shock2” ... Here for instance the model is a VAR with three endogenous variables: “ffrate”, “inflation” and “gap”. There are thus three structural shocks, resulting in three additional columns named “shock1”, “shock2” and “shock3”.

In the table, each line corresponds to one restriction. The first column “type” specifies the type of restriction. Alexandria allows for five types of restrictions: sign restrictions on impulse response function (“sign”), zero restrictions on impulse response function (“zero”), narrative restrictions on shocks (“shock”), narrative restrictions on historical decomposition (“historical”), and covariance restrictions (“covariance”, for proxy-SVAR only).

The second column “variable” reports the name of the variable to which the restriction applies. The third column “period” reports the period to which the restriction applies. The period can be either an integer that represents the IRF period (for “sign” and “zero”), or a sample period (for “shock” and “historical”). It can be left as an hyphen for covariance restrictions as it is ignored.

The subsequent columns (“shock1”, “shock2” ...) specify the shocks involved in the restrictions. They can contain only 0, 1 or -1, and for each row there can only be one (for sign restrictions) or two (for magnitude restrictions) non-zero entries. A single 1 indicates a positivity restriction, while a single -1 is used for negativity restrictions. A combination of 1 and -1 indicates a magnitude restriction, the restriction being that the shock with coefficient 1 has the largest magnitude.

Consider Figure 3.13, line 1. The type is sign which sets a restriction on impulse response functions. The restrictions applies to variable ffrate over period 1 of impulse response functions. There is a single 1 on shock3 so that means a positivity restriction: the response of ffrate to the third shock at impact of impulse response function is restricted to be positive. The second line carries a similar restriction, but applicable to period 2 of impulse response function.

Line 3 and 4 also set restrictions on impulse response functions, but this time on the variable inflation. Also, there is combination of 1 and -1 in the shock columns, which indicates a magnitude restriction. The two constraints read: The response of inflation to the first structural shock is larger than the response to the second shock, over the first two periods of impulse reponse functions.

Line 6 is a zero restriction on the variable gap, at period 1 of impulse response functions, on the third shock.

Line 7 is a narrative restriction on shocks, applicable at period 1979-06-30. There is a single 1 on the shocks columns, so it is a positivity restriction: shock 1 is restricted to be positive at period 1979-06-30.

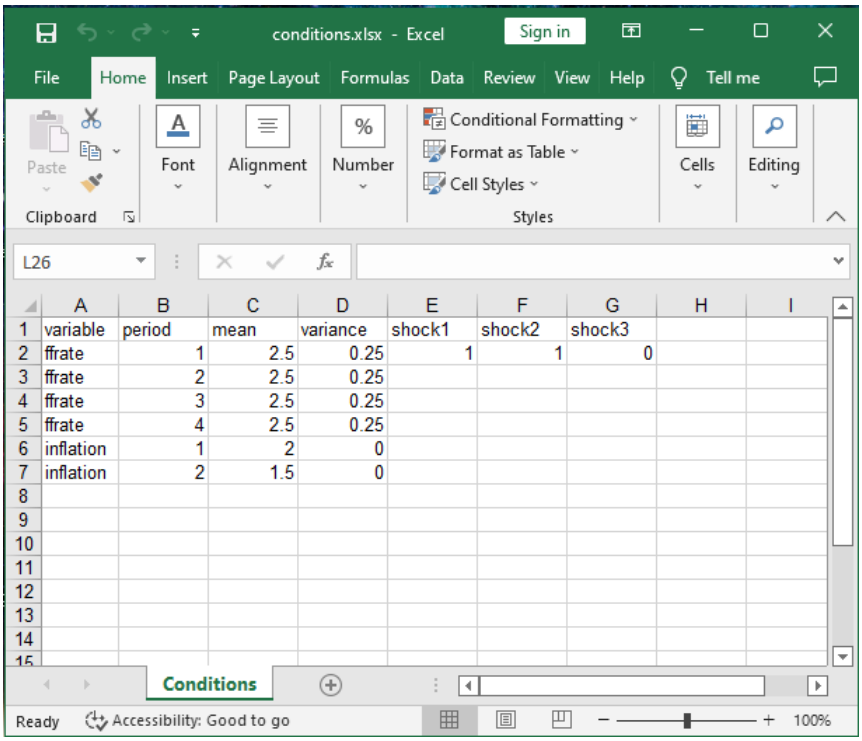
Line 8 and 9 set narrative restrictions on shocks, but this time specify magnitude restrictions. This reads: shock 1 is larger than both shock 2 and shock 3 at period 1979-06-30.

Line 10 is a narrative restriction on historical decomposition, applicable to the variable gap at period 2005-12-31. It says that the shock 1 component of historical decomposition at this period must be positive.

Line 11 is a covariance restriction. It applies only if the model is a proxy-SVAR, so assume here a proxy-SVAR model with the same three endogenous variables as before, plus two proxys: proxy_1 and proxy_2. Remember that the 2 proxys relate to the last 2 shocks, so that proxy_1 relates to shock 2 while proxy_2 relates to shock 3. Hence the magnitude restriction here says that the covariance of shock 2 with proxy_1 is larger than its covariance with shock 3. The restriction guarantees that the proxy approximates its own shock better than the other shocks.

3.6 Other datafiles: conditional forecasts

If you select a Bayesian VAR model with conditional forecasts (see section 4.6), you must provide a file that details the conditions. The file can be of type .xlsx, .xls, or .csv, and may be placed in the project folder or any convenient subfolder. A typical conditional forecast file is depicted in Figure 3.14.



	A	B	C	D	E	F	G	H	I
1	variable	period	mean	variance	shock1	shock2	shock3		
2	ffrate	1	2.5	0.25	1	1	0		
3	ffrate	2	2.5	0.25					
4	ffrate	3	2.5	0.25					
5	ffrate	4	2.5	0.25					
6	inflation	1	2	0					
7	inflation	2	1.5	0					
8									
9									
10									
11									
12									
13									
14									
15									

Figure 3.14: Example conditional forecast datafile

The table has a specific structure that must be complied with. The first four columns must be labelled as “variable”, “period”, “mean” and “variance”. There must then be one additional column per structural shock in the model, i.e. per endogenous variable. The columns must be labeled as “shock1”, “shock2”... Here for instance the model is a VAR with three endogenous variables: “ffrate”, “inflation” and “gap”. There are thus three structural shocks, hence three additional columns labeled “shock1”, “shock2” and “shock3”.

In the table, each line corresponds to one condition. The first column “variable” specifies the variable to which the condition applies. The second column “period” reports the forecast period to which the condition applies. The third column “mean” gives the mean of the condition, while the fourth column “variance” sets its variance. This fairly straightforward. The first line for instance sets a condition on the variable ffrate at the first forecast period. The condition takes a mean of 2.5 and a variance of 0.25. This corresponds to a soft condition.

Line 6 represents a condition on inflation also applying to the first forecast period. It has a mean of 2 and a variance of 0, which represents a hard condition: the condition holds exactly. Note that a variance of 0 is acceptable for conditions.

The shock columns are relevant only if you run structural conditional forecasts. In this case, they define which shocks are generating the conditions. A shock with a value of 1 generates the condition, a shock with a value of 0 does not. So for an all-shock setup put a 1 in each column, while for a single generating shock use a single 1 entry. In Figure 3.14 the setup involves two generating shocks: the first and second shocks.

Note that the set of generating shocks must be the same for all conditions, hence only the first row of the shock columns needs be filled, as in Figure 3.14. Also, the columns must appear, even if choose the agnostic conditional forecast approach. In this case, simply leave all entries as 0.

3.7 Other datafiles: constrained coefficients

If you select a Bayesian VAR model with constrained coefficients (see section 4.5), you must provide a file that details these constraints. The file can be of type .xlsx, .xls, or .csv, and may be placed in the project folder or any convenient subfolder. A typical constrained coefficients file is depicted in Figure 3.15.

The table has a specific structure that must be complied with. It must have five columns labelled as “variable”, “responding_to”, “lag”, “mean” and “variance”. In the table, each line corresponds to one constraint. The first three columns define the coefficient to which the constraint applies. In this example the model is a simple VAR with three endogenous variables: “ffrate”, “inflation” and “gap”. Line 3 in Figure 3.15 thus sets a constraint on the response of the variable ffrate to the first lag of the variable inflation.

Note that the column “responding_to” can accept names of endogenous variables, but also exogenous variables like “constant”, “trend” and “quadratic trend” (which are reserved names for these 3 elements that are automatically handled), or any other exogenous variable you may include in the model. Since these variables are not lagged, the column “lag” is set to 0 (see line 2 in Figure 3.15).

Also, you may use the special value -1 for the column lag. In this case, the constraint will automatically apply to all the lags of the model. For instance, line 4 implies a constraint on the response of the variable inflation to all the lags of gap.

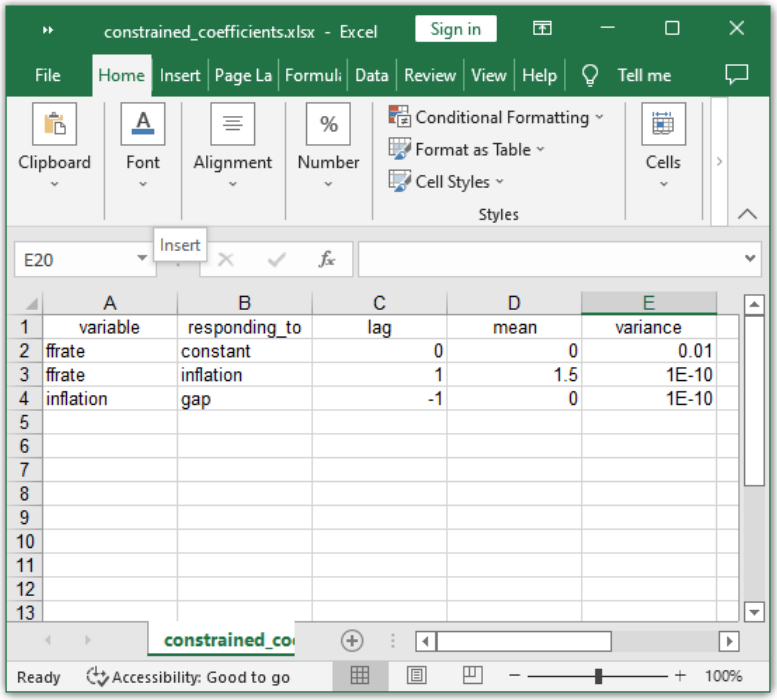


Figure 3.15: Example constrained coefficients datafile

The columns "mean" and "variance" are used to specify the prior mean and variance of the constrained coefficient. It is usually the case that ones want to set variances close to 0 to force the distribution of the coefficient to become a single degenerate value at the specified mean, but the setup is flexible and lets you choose whatever variance you desire. Note that the variance cannot be strictly 0. It must be positive, though it can take values that are arbitrarily close to 0, as in lines 3 and 4 of the example.

3.8 Other datafiles: long-run prior

If you select a Bayesian VAR model with long-run prior (see section 4.5), you must provide a file that details the prior. The file can be of type .xlsx, .xls, or .csv, and may be placed in the project folder or any convenient subfolder. A typical long-run prior file is depicted in Figure 3.16.

The long-run prior table follows a simple structure. It must have as many rows and columns as the number of endogenous variables. The names of the rows and columns have to be those of the endogenous variables. This defines a square matrix for which you can enter the coefficients you like.

Figure 3.16 displays a long-run prior matrix for a simple VAR with three endogenous variables: "ffrate", "inflation" and "gap". The line and column labels replicate the name of the variables, as needed. In this example the long-run prior uses a simple matrix with values 0, 1 and -1 only, but for your own applications any numerical values can be used.

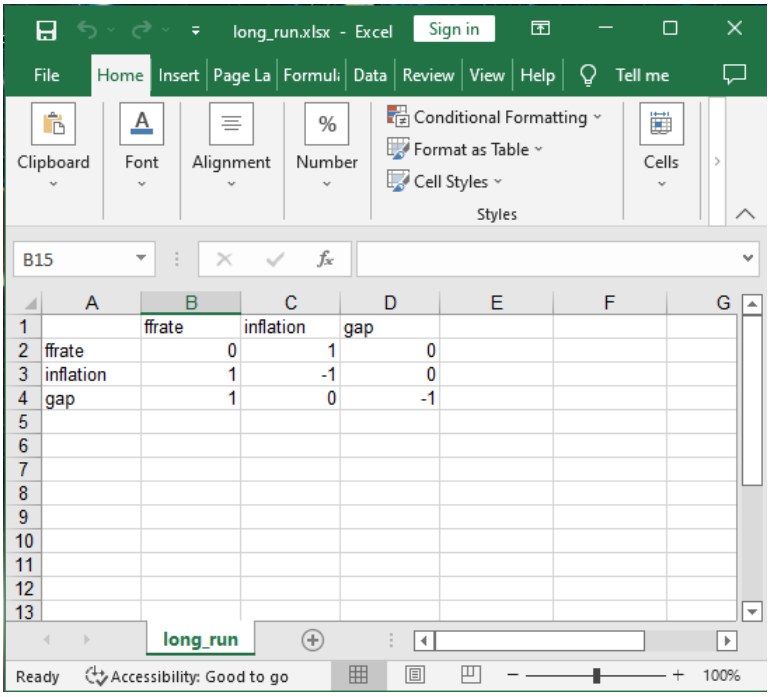


Figure 3.16: Example long-run prior datafile

CHAPTER 4

Running Alexandria from the Graphical user Interface

4.1 Launching the interface: Python edition

If you have opted for a local installation of Alexandria, your project folder must contain either the file alexandria_gui.ipynb (for a use in Jupyter Notebook) or the file alexandria_gui.py (for a use in Spyder). If you want to use Alexandria in Jupyter, start Jupyter Notebook, and in the explorer navigate to the project folder, then click on the file alexandria_gui.ipynb. This will open the notebook in a web browser. To start the Graphical Interface, move to the top menu bar (Figure 4.1) and click on:

```
Kernel -> Restart & Run All
```

This will start the Graphical User Interface.

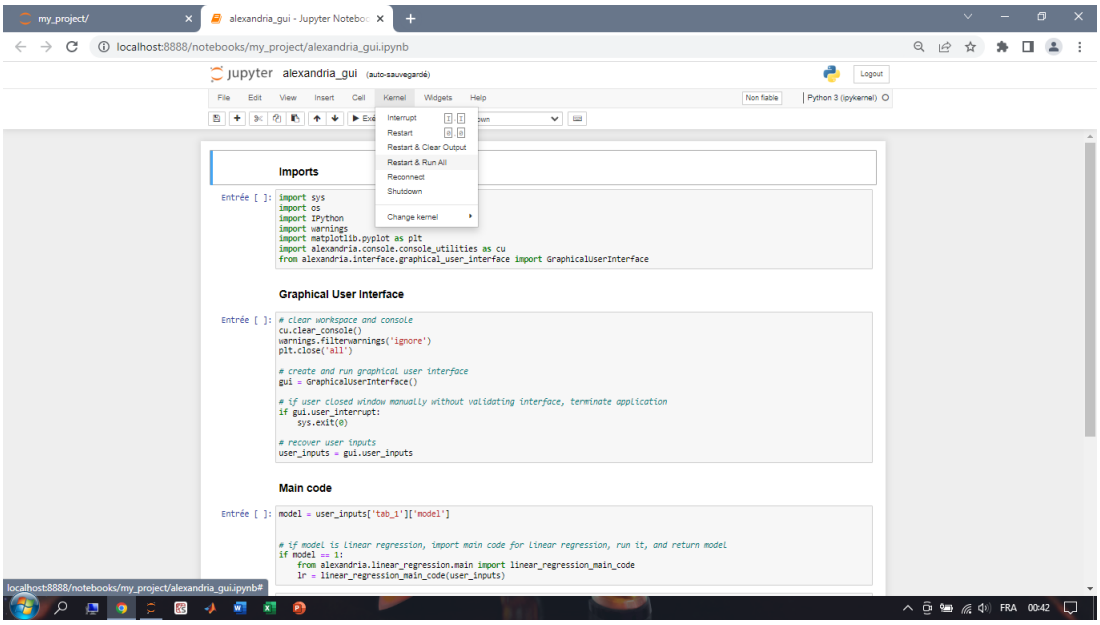


Figure 4.1: Starting the Graphical Interface from Jupyter Notebook

If instead you prefer to start Alexandria from a simple python script, open Spyder, then on the top menu bar click:

```
File -> Open
```

This will open a navigator. Navigate to your project folder and click the file alexandria_gui.py. This opens the python script in your Spyder window. To start the Graphical Interface, go again for the top menu bar of Spyder (Figure 4.2) and click:

```
Run -> Run
```

This will start the Graphical User Interface.

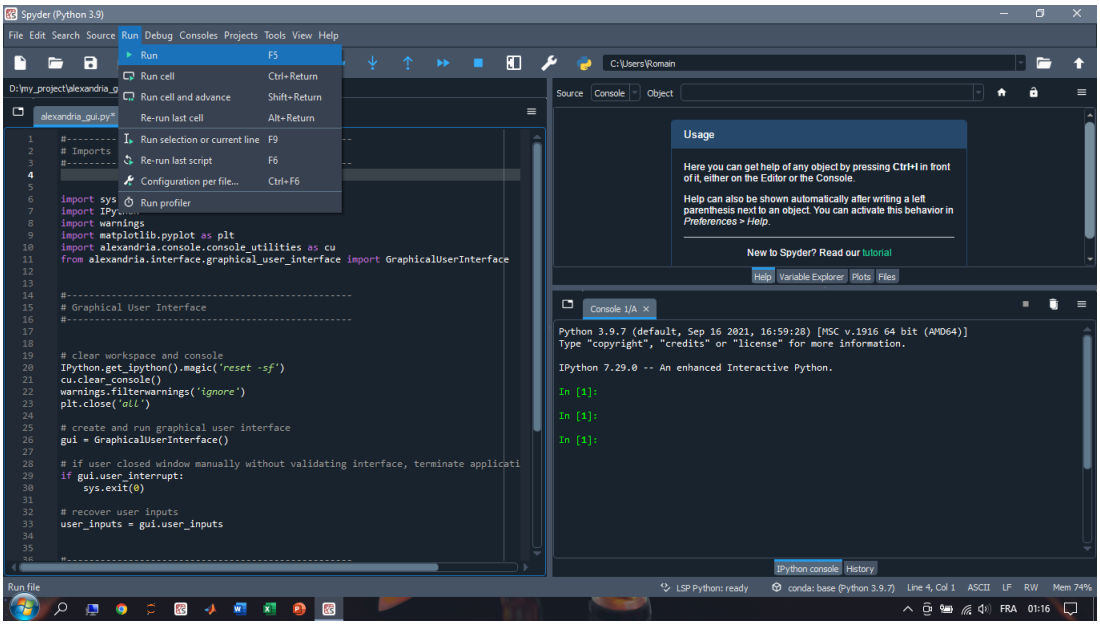


Figure 4.2: Starting the Graphical Interface from Spyder

If you opted for a permanent installation, it is recommended that you proceed the same way. However, in this case there exists an alternative way to start the Graphical Interface without using any of the files. You may just execute the following command in Jupyter or Spyder:

```
from alexandria import alexandria_gui
```

Importing the module will also execute it, and so the command is equivalent to running the script alexandria_gui.py.

4.2 Launching the interface: Matlab edition

If you have opted for a local installation of Alexandria, your project folder should normally contain the file alexandria_gui.m. If that is the case, start Matlab, then on the current folder explorer (usually located on the left side of the Matlab window) navigate to your project file and double-click on alexandria_gui.m. This should open the file in a new window. To start the Graphical Interface, move to the top menu bar (Figure 4.3) and click on:

```
Run -> Run:  alexandria_gui
```

This will start the Graphical User Interface. You may also simply click on the green triangle arrow above the Run button to run the script directly.

If you opted for a permanent installation, you can proceed the same way. However, it is also possible to start the interface without any script in this case. Simply execute the following command in the Matlab console:

```
alexandria_gui
```

This will launch the Graphical User Interface.

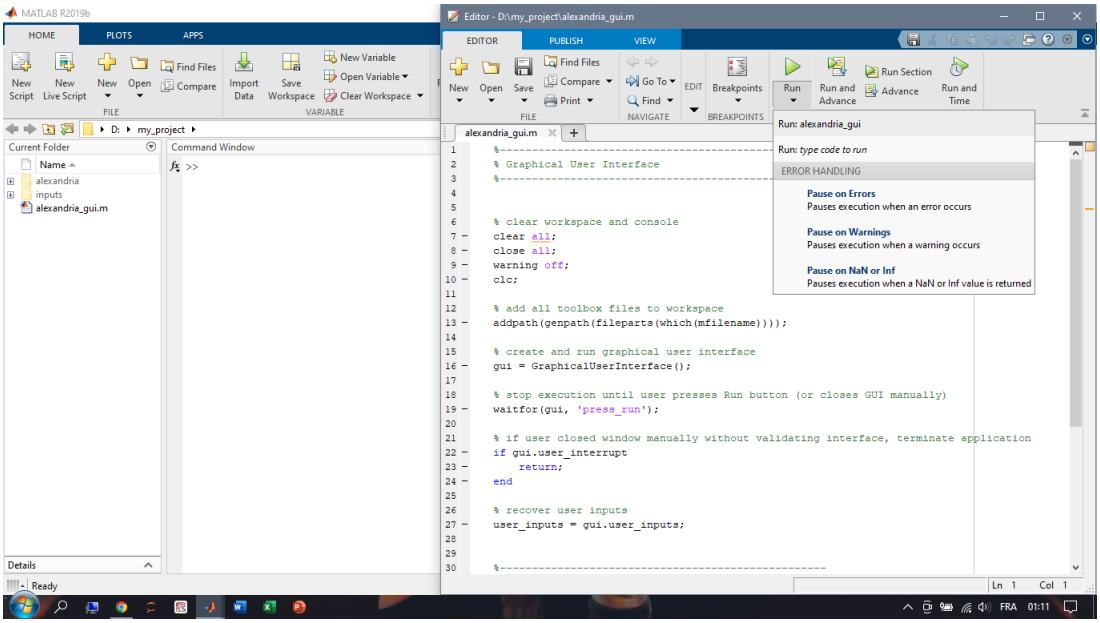


Figure 4.3: Starting the Graphical Interface from Matlab

4.3 Interface: tab 1

The first tab of the Graphical User Interface is depicted in Figure 4.4. The figure shows the interface for the Python edition of Alexandria, but the Matlab version is absolutely similar. Tab 1 is used for general model specification, data source and estimation options.

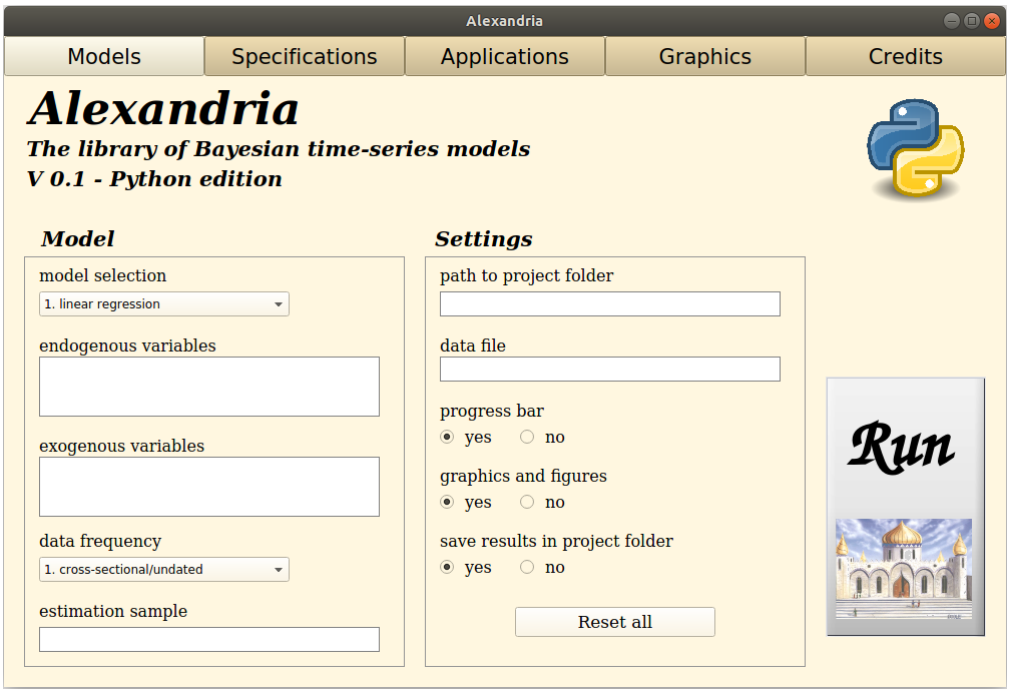


Figure 4.4: Tab 1 of the Graphical User Interface

Tab 1 specifies the following elements:

■ **Model, model selection**: the general model you want to estimate. You can currently choose from linear regression or vector autoregression. Other models will be availble as the toolbox develops.

■ **Model, endogenous variables**: the list of endogenous variables for the model, separated by a space.
for instance: endo1 endo2 endo3

■ **Model, exogenous variables**: the list of exogenous variables for the model, separated by a space.
for instance: exo1 exo2 exo3

■ **Model, data frequency**: the dataset frequency. It must correspond to the frequency of the data in the base data file (see section 3.3). The six frequencies available are: annual, quarterly, monthly, weekly, daily, and cross-sectional/undated.

■ **Model, estimation sample**: the start and end dates of the estimation sample, separated by a space. The date format must be consistent with that of the the base data file (see section 3.3).
for instance, for quarterly data in international date format: 1960-01-01 2018-10-01
for instance, for quarterly data in periodic format: 1960Q1 2018Q4

■ **Settings, path to project folder**: the path to your project folder. The path must be consistent with your OS, and thus use the correct separator (slash or backslash).
for instance, on Windows: D:\my_project
for instance, on Linux: /home/user/my_project

■ **Settings, data file**: name of the base datafile in your project folder, with the relevant extension (xls, xlsx or csv).
for instance: data.xls
tip: for a better organisation of the project folder, it is possible to create a subfolder that will contain the base data file, possibly along with other input files. In this case, simply specify the data file as a path with the subfolder containing the file.
for instance, with a subfolder called "inputs": inputs\data.xls

■ **Settings, progress bar**: if yes, a progress bar is displayed as the model is being estimated. The progress bar is a useful indicator, but it increases the computational cost of the programme.

■ **Settings, graphics and figures**: if yes, estimation plots are produced after estimation is complete. The graphics are saved in a subfolder called "graphics", and displayed in tab 4 of the interface. Please refer to section 6.4 for additional details. The plots constitute very useful outputs, but can take time to produce for large models.

■ **Settings, save results in project folder**: if yes, a number of estimation outputs are saved in text and csv format in a subfolder called "results". The files contain information about the model settings, coefficients and applications. Please refer to section 6.3 for additional details.

■ **Settings, Reset all**: by default, Alexandria saves your interface inputs and propose them again in subsequent runs. Pressing the Reset all buttons deletes all previous inputs and resets the interface to default values.

■ **Run button**: press this button to start the estimation. This should not be pressed if tabs 2 and 3 have not been completed. It can however constitute a useful shortcut if for instance you run the same model again and don't modify your previous settings.

4.4 Interface: tab 2, linear regression

Figure 4.5 depicts tab 2 of the Graphical User Interface when the selected model is the linear regression.

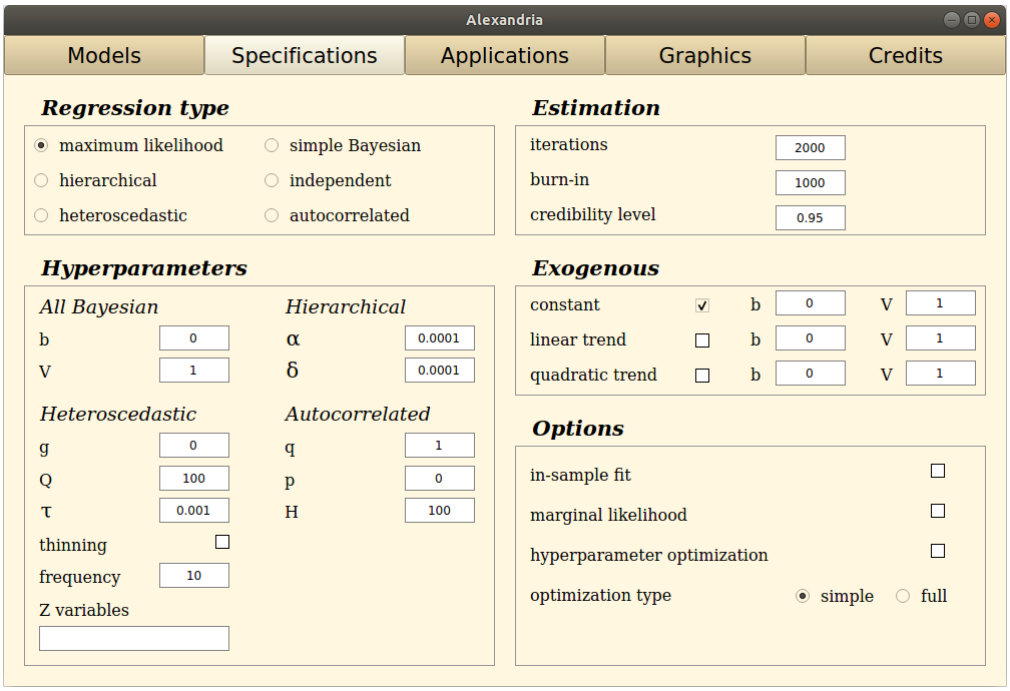


Figure 4.5: Tab 2 of the Graphical User Interface, linear regression

Tab 2 deals with model-specific information and prior. It contains the following elements:

■ **Regression type**: the specific linear regression model you want to estimate. Alexandria proposes six different linear regression models: maximum likelihood, simple Bayesian, hierarchical, independent, heteroscedastic and autocorrelated. Please consult the textbook, sections 9.1 to 9.6, for additional details on the different models.

■ **Estimation, iterations**: the number of iterations for the MCMC algorithms, after the burn-in iterations are completed. Applies only to models using MCMC methods.
for instance: 5000

■ **Estimation, burn-in**: the number of burn-in iterations for the MCMC algorithms. Applies only to models using MCMC methods.
for instance: 2000

■ **Estimation, credibility level**: the credibility level used for the calculation of credibility intervals for the model estimates. Must be comprised between 0 and 1.
for instance: 0.95

■ **Hyperparameters, b**: the prior mean on the regression coefficients β. Can be a single value, in which case the same prior mean applies to all coefficients. Else, must be a list of values separated by a space, one value for each exogenous variable. No values should be provided for potential constants and trends which are handled separately (see below).
for instance: 0, or: 0.2 0.4

■ **Hyperparameters, V**: the prior variance on the regression coefficients β. Can be a single value, in which case the same prior variance applies to all coefficients. Else, must be a list of values separated by

a space, one value for each exogenous variable. No values should be provided for potential constants and trends which are handled separately (see below).
for instance: 1, or: 0.5 1

■ **Hyperparameters, α**: the prior shape on the residual variance σ. Must be a positive scalar.
for instance: 0.0001

■ **Hyperparameters, δ**: the prior scale on the residual variance σ. Must be a positive scalar.
for instance: 0.0001

■ **Hyperparameters, g**: the prior mean on the heteroscedastic coefficients γ. Can be a single value, in which case the same prior mean applies to all coefficients. Else, must be a list of values separated by a space, one value for each heteroscedastic variable.
for instance: 0, or 0 0

■ **Hyperparameters, Q**: the prior variance on the heteroscedastic coefficients γ. Can be a single value, in which case the same prior variance applies to all coefficients. Else, must be a list of values separated by a space, one value for each heteroscedastic variable.
for instance: 100, or 100 100

■ **Hyperparameters, τ**: the variance of the random walk kernel for the Metropolis-Hastings algorithm. Should be set (by trial and error) to generate an acceptance rate of 20%-30%.
for instance: 0.001

■ **Hyperparameters, thinning**: if selected, thinning is applied to the Metropolis-Hastings algorithm, which helps to reduce the number of repeated values and produces a finer distribution. This is useful but generates additional calculations (see next parameter).

■ **Hyperparameters, frequency**: thinning frequency: if set to n, only 1 out of n iterations will be retained, the other iterations being discarded. This mutliplies by n the total number of iterations in order to maintain constant the final number of draws. It can then prove very costly for large n.
for instance: 10

■ **Hyperparameters, Z variables**: the list of heteroscedastic variables. These regressors must be found in the same base data file as the base model regressors. Can be the same variables as the model exogenous variables, but other variables can be used if relevant.
for instance: htr1 htr2 htr3

■ **Hyperparameters, q**: the order of autocorrelation for the autocorrelated regression. Must be some integer equal of larger than 1.
for instance: 2

■ **Hyperparameters, p**: the prior mean on the autocorrelated coefficients ϕ. Can be a single value, in which case the same prior mean applies to all coefficients. Else, must be a list of values separated by a space, one value for each lag of autocorrelation.
for instance: 0 or 0.9 0

■ **Hyperparameters, H**: the prior variance on the autocorrelated coefficients γ. Can be a single value, in which case the same prior variance applies to all coefficients. Else, must be a list of values separated by a space, one value for each lag of autocorrelation.
for instance: 100 or 100 100

■ **Exogenous, constant**: if selected, adds a constant to the linear regression.

■ **Exogenous, constant, b**: the prior mean on the constant. Must be a scalar.
for instance: 0

■ **Exogenous, constant, V**: the prior variance on the constant. Must be a positive scalar.
for instance: 100

■ **Exogenous, linear trend**: if selected, adds a linear trend to the linear regression.

■ **Exogenous, linear trend, b**: the prior mean on the linear trend. Must be a scalar.
for instance: 0

■ **Exogenous, linear trend, V**: the prior variance on the linear trend. Must be a positive scalar.
for instance: 100

■ **Exogenous, quadratic trend**: if selected, adds a quadratic trend to the linear regression.

■ **Exogenous, quadratic trend, b**: the prior mean on the quadratic trend. Must be a scalar.
for instance: 0

■ **Exogenous, quadratic trend, V**: the prior variance on the quadratic trend. Must be a positive scalar.
for instance: 100

■ **Options, in-sample fit**: if selected, calculates the in-sample fit of the model.

■ **Options, marginal likelihood**: if selected, calculates the marginal likelihood for the model and displays the value in the console output of the model estimation.

■ **Options, hyperparameter optimization**: if selected, runs a numerical optimization procedure to find the hyperparameter values V and δ that maximize the marginal likelihood, and use these values for the priors. Applicable only to the simple Bayesian and hierarchical regressions.

■ **Options, optimization type**: if simple, a common prior variance is assumed for all the β coefficients, and the optimizer will search for a single optimal scalar value for V. If full, a different prior variance is assumed for the β coefficients, so that the optimizer will try to find a different optimal value for each V coefficient.

4.5 Interface: tab 2, vector autoregression

Figure 4.6 depicts tab 2 of the Graphical User Interface when the selected model is the vector autoregression.

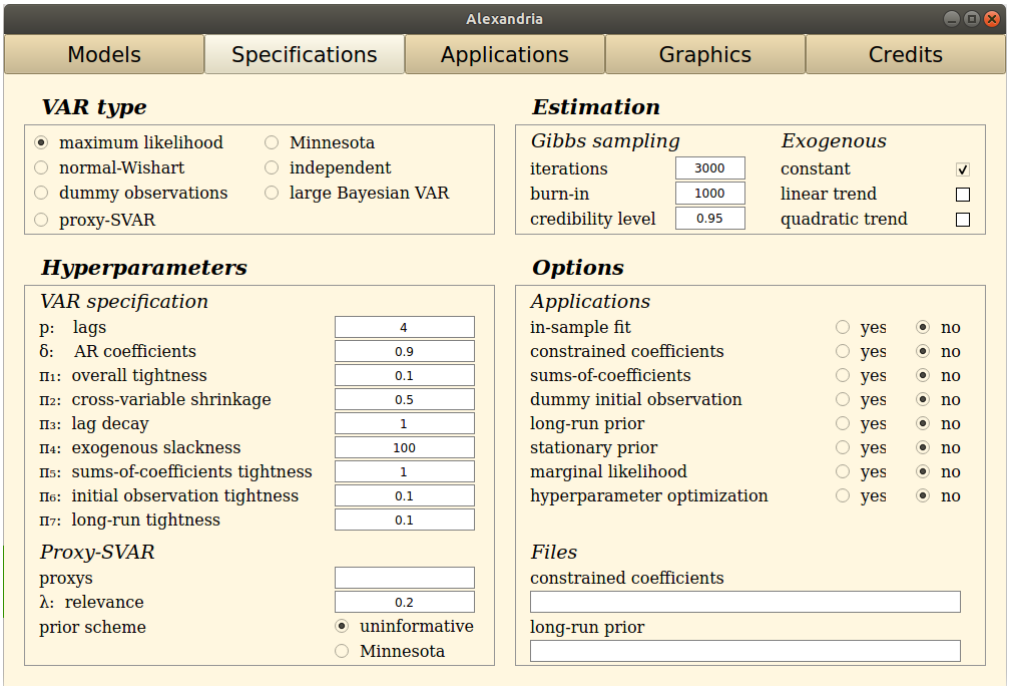


Figure 4.6: Tab 2 of the Graphical User Interface, linear regression

Tab 2 deals with model-specific information and prior. It contains the following elements:

■ **VAR type**: the specific VAR model you want to estimate. Alexandria proposes six different vector autoregression models: maximum likelihood, Minnesota, normal-Wishart, independent, dummy observations, large Bayesian VAR, and Bayesian proxy-SVAR. Please consult the textbook, sections 11.1 to 11.6 and 14.5 for additional details on the different models.

■ **Estimation, iterations**: the number of iterations for the MCMC algorithms, after the burn-in iterations are completed. Applies to all models except maximum likelihood VAR.
for instance: 5000

■ **Estimation, burn-in**: the number of burn-in iterations for the MCMC algorithms. Applies to all models except maximum likelihood VAR.
for instance: 2000

■ **Estimation, credibility level**: the credibility level used for the calculation of credibility intervals for the model estimates. Must be comprised between 0 and 1.
for instance: 0.6

■ **Estimation, constant**: if selected, adds a constant to the VAR model.

■ **Estimation, linear trend**: if selected, adds a linear trend to the VAR model.

■ **Estimation, quadratic trend**: if selected, adds a quadratic trend to the VAR model.

■ **Hyperparameters, p: lags**: number of lags in the VAR model.
for instance: 6

■ **Hyperparameters, delta: AR coefficients**: prior mean for the Minnesota prior. Either a single scalar (same prior mean for all equations), or list of scalars separated by a space, one value for each equation.
for instance: 0.9
for instance: 0.8 0.7 0.9 (for a VAR with three endogenous variables)

■ **Hyperparameters, pi1: overall tightness**: overall tightness for the prior variance scheme of the Minnesota prior. Must be a positive scalar.
for instance: 0.01

■ **Hyperparameters, pi2: cross-variable shrinkage**: cross-variable shrinkage for the prior variance scheme of the Minnesota prior. Must be a positive scalar.
for instance: 0.2

■ **Hyperparameters, pi3: lag decay**: lag decay for the prior variance scheme of the Minnesota prior. Must be a positive scalar.
for instance: 2

■ **Hyperparameters, pi4: exogenous slackness**: exogenous slackness for the prior variance scheme of the Minnesota prior. Must be a positive scalar.
for instance: 1000

■ **Hyperparameters, pi5: sums-of-coefficients tightness**: sums-of-coefficients tightness for the prior variance scheme of the Minnesota prior. Must be a positive scalar.
for instance: 0.1

■ **Hyperparameters, pi6: initial observation tightness**: initial observation tightness for the prior variance scheme of the Minnesota prior. Must be a positive scalar.
for instance: 0.001

■ **Hyperparameters, pi7: long-run tightness**: long-run tightness for the prior variance scheme of the Minnesota prior. Must be a positive scalar.
for instance: 0.001

■ **Hyperparameters, proxys**: list of proxy variables for the model, separated by a space. Only applicable to the proxy-SVAR.
for instance: proxy_1 proxy_2

■ **Hyperparameters, lambda: relevance**: relevance value for the proxy-SVAR. Must be a scalar between 0 and 1.
for instance: 0.1

■ **Hyperparameters, prior scheme**: choice of prior for the proxy-SVAR. Can be either uninformative, or Minnesota.

■ **Options, in-sample fit**: if selected, calculates the in-sample fit of the model.

■ **Options, constrained coefficients**: if selected, applies constrained coefficients to the model. Only applicable to Minnesota, independent, and large BVAR priors. Requires a valid constrained coefficients file (see section 3.7 for additional details).

■ **Options, sums-of-coefficients**: if selected, applies the sums-of-coefficients dummy extension to the model. Applicable to all models except maximum likelihood VAR and proxy-SVAR.

■ **Options, dummy initial observation**: if selected, applies the dummy initial observation extension to the model. Applicable to all models except maximum likelihood VAR and proxy-SVAR.

■ **Options, long-run prior**: if selected, applies the long-run prior dummy extension to the model. Applicable to all models except maximum likelihood VAR and proxy-SVAR. Requires a valid long-run prior file (see section 3.8 for additional details).

■ **Options, stationary prior**: if selected, applies stationary prior to the model. Applicable to all models except maximum likelihood VAR and proxy-SVAR.

■ **Options, marginal likelihood**: if selected, calculates the marginal likelihood for the model and displays the value in the console output of the model estimation. Only applies to Minnesota, normal-Wishart and independent priors.

■ **Options, hyperparameter optimization**: if selected, runs a numerical optimization procedure to find the hyperparameter values δ, π_1, π_2, π_3, and π_4 that maximize the marginal likelihood, and use these values for the priors. Applicable only to the Minnesota and normal-Wishart priors.

■ **Options, constrained coefficients file**: name of the datafile for constrained coefficients in your project folder, with the relevant extension (xls, xlsx or csv). If placed on a subfolder, simply specify the data file as a path with the subfolder containing the file.
for instance: constrained_coefficients.xlsx
for instance: inputs\constrained_coefficients.xlsx (for file in a project subfolder called inputs)

■ **Options, long-run prior file**: name of the datafile for long-run prior in your project folder, with the relevant extension (xls, xlsx or csv). If placed on a subfolder, simply specify the data file as a path with the subfolder containing the file.
for instance: long_run.xlsx
for instance: inputs\long_run.xlsx (for file in a project subfolder called inputs)

4.6 Interface: tab 3

Tab 3 deals with model applications. It is depicted in Figure 4.7. Not all applications are available for all models, in particular for the linear regressions.

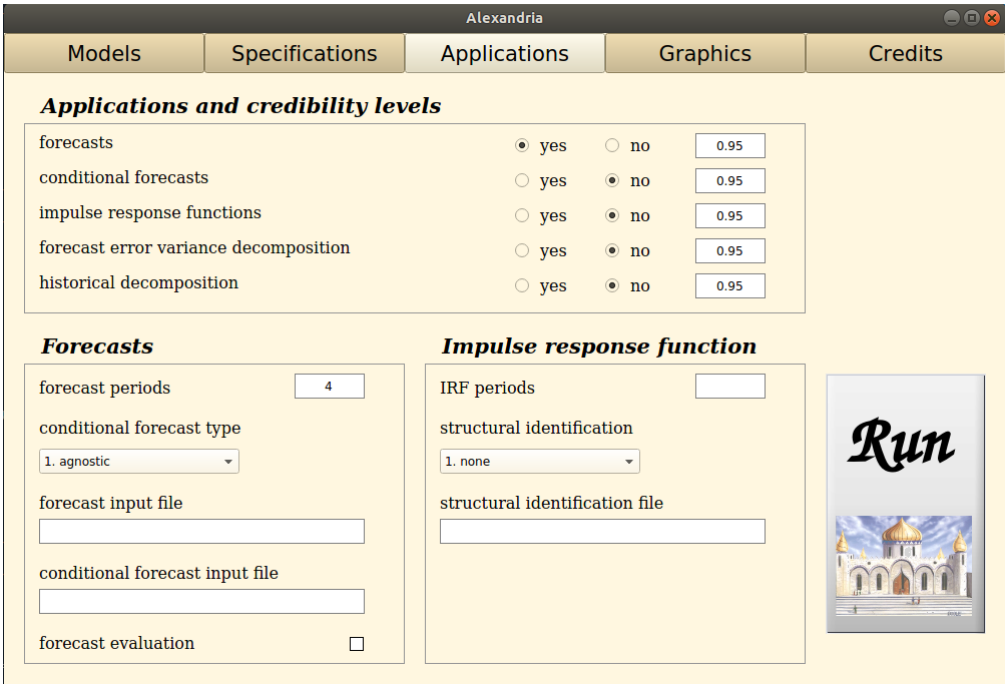


Figure 4.7: Tab 3 of the Graphical User Interface

Tab 3 contains the following elements:

■ **Application and credibility levels, forecasts**: if yes, forecasts are estimated for the model. This may require a valid input file. See section 3.4 for additional details. Applicable to both linear regression and VAR models.

■ **Application and credibility levels, forecasts, credibility level**: credibility level for the computation of credibility bands around the forecasts. Must be a scalar between 0 and 1.
for instance: 0.95

■ **Application and credibility levels, conditional forecasts**: if yes, conditional forecasts are estimated for the model. This requires a valid input file. See section 3.6 for additional details. Applicable to Bayesian VAR models only.

■ **Application and credibility levels, conditional forecasts, credibility level**: credibility level for the computation of credibility bands around the conditional forecasts. Must be a scalar between 0 and 1.
for instance: 0.95

■ **Application and credibility levels, impulse response function**: if yes, impulse response functions are estimated for the model. Applicable to VAR models only.

■ **Application and credibility levels, impulse response function, credibility level**: credibility level for the computation of credibility bands around the impulse response function. Must be a scalar between 0 and 1.
for instance: 0.95

■ **Application and credibility levels, forecast error variance decomposition**: if yes, forecast error variance decomposition is estimated for the model. Applicable to VAR models only.

■ **Application and credibility levels, forecast error variance decomposition, credibility level**: credibility level for the computation of credibility bands around the forecast error variance decomposition. Must be a scalar between 0 and 1.
for instance: 0.95

■ **Application and credibility levels, historical decomposition**: if yes, historical decomposition is estimated for the model. Applicable to VAR models only.

■ **Application and credibility levels, historical decomposition, credibility level**: credibility level for the computation of credibility bands around the historical decomposition. Must be a scalar between 0 and 1.
for instance: 0.95

■ **Forecasts, forecast periods**: number of forecast periods. Only applicable to VAR models.
for instance: 8

■ **Forecasts, conditional forecast type**: can be either agnostic, or structural shocks. Only applicable to Bayesian VAR models if conditional forecasts is selected.

■ **Forecasts, forecast input file**: name of the datafile for forecasts in your project folder, with the relevant extension (xls, xlsx or csv). If placed on a subfolder, simply specify the data file as a path with the subfolder containing the file.
for instance: forecasts.xlsx
for instance: inputs\forecasts.xlsx (for file in a project subfolder called inputs)

■ **Forecasts, conditional forecast input file**: name of the datafile for conditional forecasts in your project folder, with the relevant extension (xls, xlsx or csv). If placed on a subfolder, simply specify the data file as a path with the subfolder containing the file.
for instance: conditional_forecasts.xlsx
for instance: inputs\conditional_forecasts.xlsx (for file in a project subfolder called inputs)

■ **Forecasts, forecast evaluation**: if selected, calculates forecast evaluation criteria for the model predictions and displays them in the estimation output. Requires valid counterfactual values in the forecast data file (see section 3.4 for additional details).

■ **Impulse response function, IRF periods**: number of impulse response function periods. Only applicable to VAR models.
for instance: 30

■ **Impulse response function, structural identification**: can be none, Cholesky, triangular or restrictions. Restrictions cannot apply to maximum likelihood VAR models. Proxy SVAR models only accept none (if no additional restrictions) or restrictions. If set to restrictions, requires a valid restriction file (see section 3.5 for additional details).

■ **Impulse response function, structural identification file**: name of the datafile for structural identification in your project folder, with the relevant extension (xls, xlsx or csv). If placed on a subfolder, simply specify the data file as a path with the subfolder containing the file.
for instance: structural_identification.xlsx
for instance: inputs\structural_identification.xlsx (for file in a project subfolder called inputs)

■ **Run button**: press this button to start the estimation. This should be pressed only when tabs 1, 2 and 3 have all been completed.

4.7 Interface: tab 4

Tab 4 deals with the graphical outputs of the toolbox. It is depicted in Figure 4.8. Prior to model estimation, no graphics are available and thus the interface only displays the image "No graphic to display yet". This tab becomes useful only once model estimation is completed. Please refer to section 6.3 for additional details on how to navigate this tab once graphics are produced.

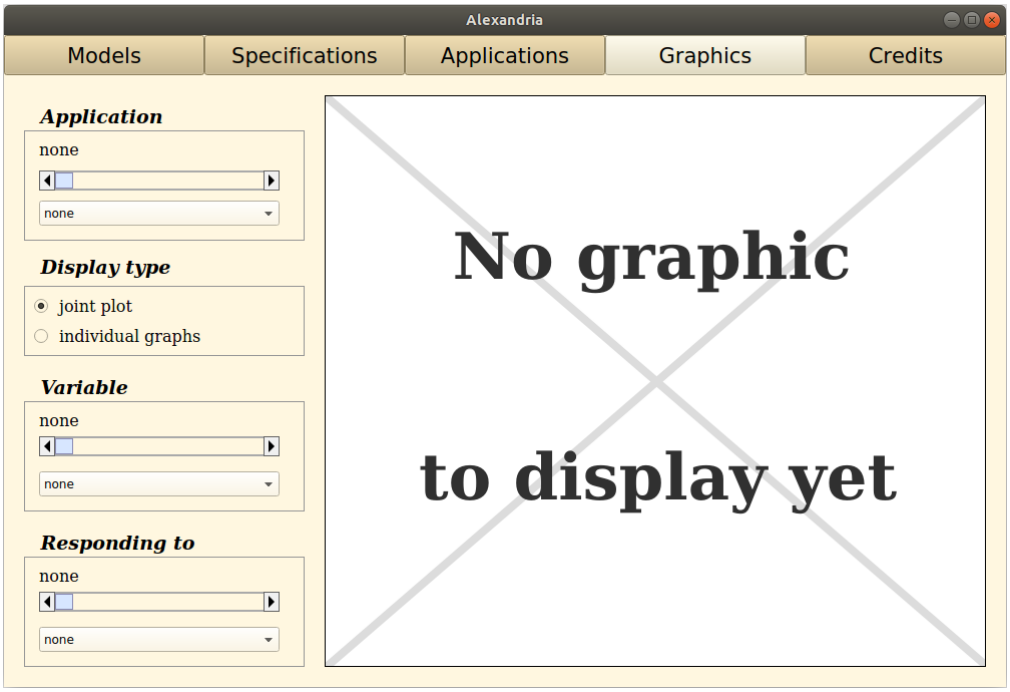


Figure 4.8: Tab 4 of the Graphical User Interface

4.8 Interface: tab 5

Tab 5 of the Graphical User Interface is depicted in Figure 4.9. It does not offer any interaction but offers general information about the toolbox and related social media.

Figure 4.9: Tab 5 of the Graphical User Interface

CHAPTER 5

Alternative ways to run Alexandria

5.1 Running Alexandria from the Integrated Development Environment: Python edition

If you don't want to navigate the Graphical User Interface every time you re-estimate a model, you can use a more straightfoward way to run the toolbox: the Integrated Development Environment (IDE). Fundamentally, the IDE is a simple Python script that is equivalent to the Graphical user Interface but replaces the interface navigation with direct Python code. Running Alexandria from the IDE then only requires to run the script from Jupyter or Spyder, which can prove much faster than using the Graphical User Interface.

For the Python edition of Alexandria, using the IDE requires that you place in your project folder either the file alexandria_ide.ipynb (for a use in Jupyter Notebook) or the file alexandria_ide.py (for a use in Spyder). It is recommended that you install Alexandria permanently if you want to use the IDE so that you can simply copy and paste the file alexandria_ide.ipynb / alexandria_ide.py in any new project folder you create. Otherwise, you need to proceed to a full local installation, as described in section 1.5.

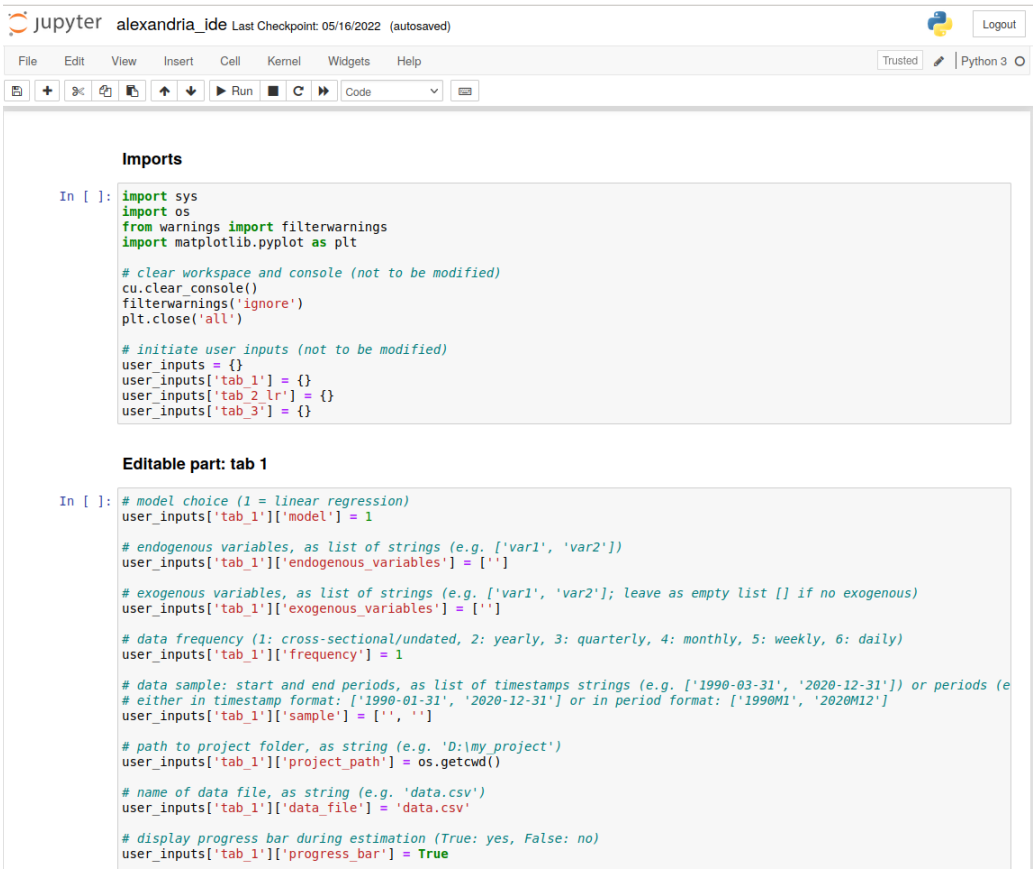


Figure 5.1: Alexandria IDE on Jupyter Notebook

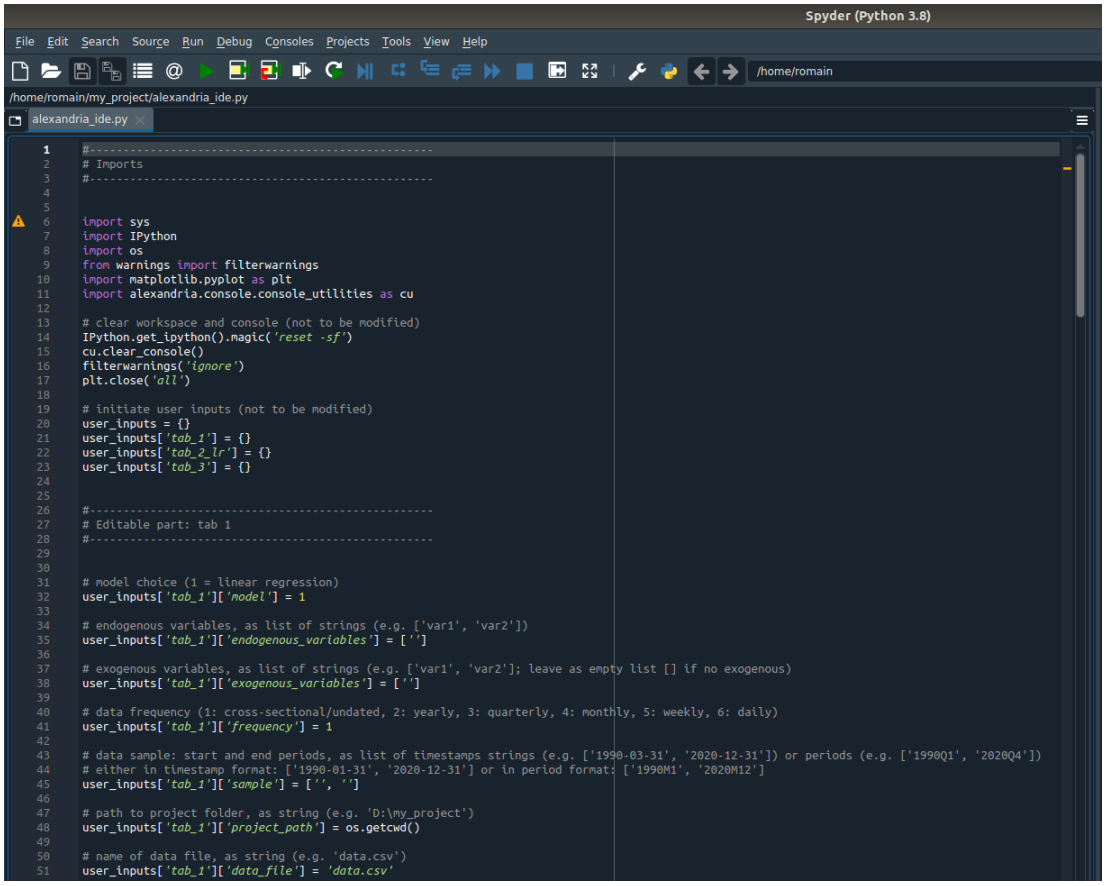


Figure 5.2: Alexandria IDE on Spyder

The file alexandria_ide.ipynb is shown in Figure 5.1, while alexandria_ide.py is depicted in Figure 5.2. As you can see they appear as simple Python scripts, with some parts editable. All you need to do is to edit the editable parts of the script to provide the relevant information about your project. The correspondance with the elements of the Graphical user Interface should be straightforward, but if needed you may refer to chapter 4 for additional details.

In terms of format, the comments provided before each editable lines will guide you to edit the part correctly. Note that even though the format is indicated for each item, using the IDE requires minimal familiarity with Python programming and its syntax. Note also that the code cannot check for improper inputs, so that any misspecified entry will result in an exception during the programme.

Once the file is properly edited, starting the toolbox only requires to execute the script: if needed, refer to section 4.1.

5.2 Running Alexandria from the Integrated Development Environment: Matlab edition

If you don't want to navigate the Graphical User Interface every time you re-estimate a model, you can use a more straightfoward way to run the toolbox: the Integrated Development Environment (IDE). Fundamentally, the IDE is a simple Matlab script that is equivalent to the Graphical user Interface but replaces the interface navigation with direct Matlab code. Running Alexandria from the IDE then only requires to run the script from your Matlab console, which can prove musch faster than using the Graphical User Interface.

For the Matlab edition of Alexandria, there are two different ways to use the IDE. If you installed Alexandria locally, using the IDE requires that you place in your project folder the file alexandria_ide.m. If you installed Alexandria permanently you can proceed the same way, but you can also use the file alexandria_ide.m that is permanently installed on your Matlab. To edit the file, simply execute the following command in the Matlab console:

```
open alexandria_ide
```

You can then work with the opened file as if it was a local alexandria_ide.m file placed in your project folder.

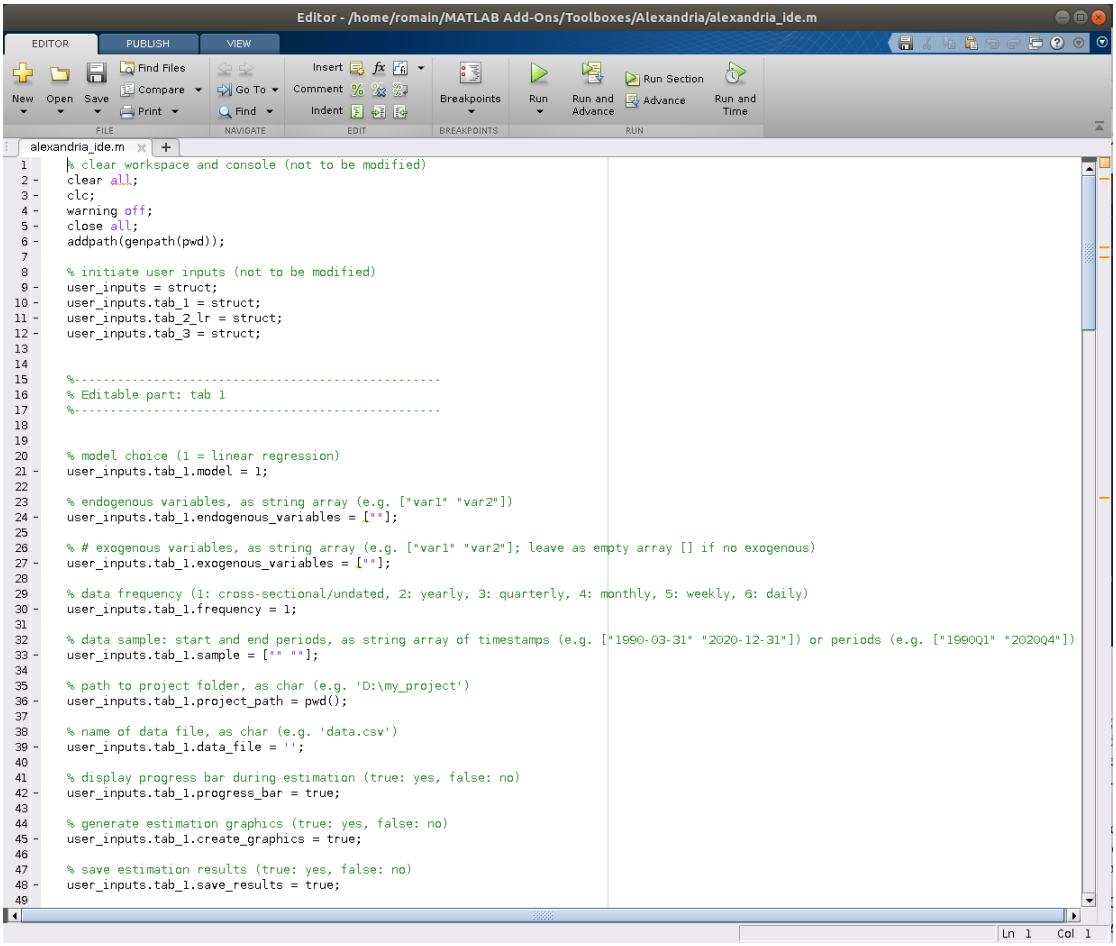


Figure 5.3: Alexandria IDE on Jupyter Notebook

The file alexandria_ide.m is shown in Figure 5.3. As you can see, it appears as a simple Matlab script, with some parts editable. All you need to do is to edit the editable parts of the script to provide the relevant information about your project. The correspondance with the elements of the Graphical user Interface should be straightforward, but if needed you may refer to chapter 4 for additional details.

In terms of format, the comments provided before each editable lines will guide you to edit the part correctly. Note that even though the format is indicated for each item, using the IDE requires minimal familiarity with Matlab programming and its syntax. Note also that the code cannot check for improper inputs, so that any misspecified entry will result in an error during the programme.

Once the file is properly edited, starting the toolbox only requires to execute the script: if needed, refer to section 4.2.

5.3 Running Alexandria on the fly: Python edition

The Graphical User Interface and Integrated Developement Environments provide convenient ways to run your models, but they may lack flexibility. For instance, experienced computer or data scientists may want to integrate the model estimation within some larger programme or application. In this case, the GUI and IDE are not suitable. Fortunately, Alexandria also offers the possibility to call the models on the fly, as you would do for instance with libraries such as Scikit-learn or statsmodels. In this case, you can directly create model objects, call their methods and recover their attributes, as you would with any object.

The following piece of code provides a simple example of the use of Alexandria on the fly (here to estimate a simple normal-Wishart Bayesian VAR with the IS-LM dataset developed in section 13.6 and 14.6 of the textbook):

```
# imports: normal-Wishart Bayesian VAR, sample datasets, console results,
# and graphics
from alexandria import NormalWishartBayesianVar
from alexandria import DataSets
from alexandria import Results
from alexandria import Graphics

# load IS-LM dataset, keep only first four columns with economic data
ds = DataSets()
islm_data = ds.load_islm()[:,:4]

# create and train Bayesian VAR with default settings
var = NormalWishartBayesianVar(endogenous = islm_data)
var.estimate()

# display console outputs for the model
res = Results(var)
res.make_estimation_summary()
res.show_estimation_summary()

# estimate forecasts for the next 4 periods, 60% credibility level
forecast_estimates = var.forecast(4, 0.6)

# create graphics of predictions
gp = Graphics(var)
gp.forecast_graphics(show=True, save=False)
```

Let's take a closer look at the code. The first block of code deals with imports, importing a number of classes from Alexandria. The first line imports the NormalWishartBayesianVar class, which is the model we are going to estimate. The next line imports the Datasets class from which we will recover the IS-LM toy dataset. The third line imports the Results class that will be used to display the model estimation output in the console. The last line imports the Graphics class which is used to create application graphics.

The second block creates an instance of the Datasets class, then load the IS-LM dataset. Only the first four columns are used as the last two columns contain proxys which are no interest here.

The third block creates an instance of the NormalWishartBayesianVar class, with the IS-LM dataset used as minimal input. The estimate methods then trains the model.

The fourth block creates an instance of the Results class, used to create an estimation summary of the model. The make_estimation_summary() method creates the summary and stores it as attribute, while the show_estimation_summary() method displays it in the console.

The last block creates an instance of the Graphics class, used to create application graphics for the model, then generates and displays plots for the forecasts.

This simple example illustrates the flexibility of Alexandria on the fly. Basically, every single feature produced from the Graphical user Interface version of Alexandria can be replicated on the fly, using relevant classes and methods.

These classes are fully documented in the subsequent chapters of the guide. Chapter 7 provides the details on the different model classes available in Alexandria, including linear regressions and vector autoregressions. Chapter 8 details the Results class, which creates console outputs. Chapter 9 details the Graphics class, which creates application graphics for all Alexandria models. Chapter 10 finally introduces the Datasets class, which is used to load toy datasets for research and replication purposes.

As an alternative to reading the guide, you can use the help function for the class. For instance, for the normal-Wishart Bayesian VAR, you can first import the module as in the example, then execute the command:

```
help(NormalWishartBayesianVar)
```

This will display the help of the class in the console (Figure 5.4), detailing the class constructor, methods and attributes.

Two remarks to conclude. First, it is recommended that you proceed to a permanent installation if you inted to use Alexandria on the fly. This will save the hastle of creating local copies of the toolbox everytime you want to estimate a model. Second, a use on the fly provides additional flexibility but requires good knowledge of Python programming, and in particular some familiarity with the object-oriented paradigm. Novice users might prefer the GUI and IDE for an comfortable experience.

```
Entrée [2]: help(NormalWishartBayesianVar)

Help on class NormalWishartBayesianVar in module alexandria.vector_autoregression.normal_wishart_bayesian_var:

class NormalWishartBayesianVar(alexandria.vector_autoregression.vector_autoregression.VectorAutoRegression, alexandr
ia.vector_autoregression.bayesian_var.BayesianVar)
 |  NormalWishartBayesianVar(endogenous, exogenous=[], structural_identification=2, restriction_table=[], lags=4, co
nstant=True, trend=False, quadratic_trend=False, ar_coefficients=0.95, pi1=0.1, pi3=1, pi4=100, pi5=1, pi6=0.1, pi7=
0.1, sums_of_coefficients=False, dummy_initial_observation=False, long_run_prior=False, long_run_table=[], hyperpara
meter_optimization=False, stationary_prior=False, credibility_level=0.95, iterations=2000, verbose=False)
 |
 |  Normal-Wishart vector autoregression, developed in section 11.3
 |
 |  Parameters:
 |  -----------
 |  endogenous : ndarray of size (n_obs,n)
 |      endogenous variables, defined in (4.11.1)
 |
 |  exogenous : ndarray of size (n_obs,m), default = []
 |      exogenous variables, defined in (4.11.1)
 |
 |  structural_identification : int, default = 2
 |      structural identification scheme, as defined in section 13.2
 |      1 = none, 2 = Cholesky, 3 = triangular, 4 = restrictions
 |
 |  restriction_table : ndarray
 |      numerical matrix of restrictions for structural identification
 |
 |  lags : int, default = 4
 |      number of lags, defined in (4.11.1)
 |
 |  constant : bool, default = True
 |      if True, an intercept is included in the VAR model exogenous
 |
 |  trend : bool, default = False
 |      if True, a linear trend is included in the VAR model exogenous
 |
 |  quadratic_trend : bool, default = False
 |      if True, a quadratic trend is included in the VAR model exogenous
 |
 |  ar_coefficients : float or ndarray of size (n_endo,1), default = 0.95
 |      prior mean delta for AR coefficients, defined in (4.11.16)
 |
```

Figure 5.4: The help function for the NormalWishartBayesianVar class

5.4 Running Alexandria on the fly: Matlab edition

The Graphical User Interface and Integrated Developement Environments provide convenient ways to run your models, but they may lack flexibility. For instance, experienced computer or data scientists may want to integrate the model estimation within some larger programme or application. In this case, the GUI and IDE are not suitable. Fortunately, Alexandria also offers the possibility to call the models on the fly, as you would do for instance with libraries such as Scikit-learn or statsmodels. In this case, you can directly create model objects, call their methods and recover their attributes, as you would with any object.

The following piece of code provides a simple example of the use of Alexandria on the fly (here to estimate a simple normal-Wishart Bayesian VAR with the IS-LM dataset developed in section 13.6 and 14.6 of the textbook):

```
% load IS-LM dataset, keep only first four columns with economic data
ds = DataSets();
islm_data = ds.load_islm();
islm_data = islm_data(:,1:4);

% create and train Bayesian VAR with default settings
var = NormalWishartBayesianVar(islm_data);
var.estimate();

% display console outputs for the model
res = Results(var);
res.make_estimation_summary();
res.show_estimation_summary();

% estimate forecasts for the next 4 periods, 60% credibility level
forecast_estimates = var.forecast(4, 0.6);

% create graphics of predictions
gp = Graphics(var);
gp.forecast_graphics(true, false);
```

Let's take a closer look at the code. The first block of code creates an instance of the Datasets class, then load the IS-LM dataset. Only the first four columns are used as the last two columns contain proxys which are no interest here.

The second block creates an instance of the NormalWishartBayesianVar class, with the IS-LM dataset used as minimal input. The estimate methods then trains the model.

The third block creates an instance of the Results class, used to create an estimation summary of the model. The make_estimation_summary() method creates the summary and stores it as attribute, while the show_estimation_summary() method displays it in the console.

The last block creates an instance of the Graphics class, used to create application graphics for the model, then generates and displays plots for the forecasts.

This simple example illustrates the flexibility of Alexandria on the fly. Basically, every single feature produced from the Graphical user Interface version of Alexandria can be replicated on the fly, using relevant classes and methods.

These classes are fully documented in the subsequent chapters of the guide. Chapter 7 provides the details on the different model classes available in Alexandria, including linear regressions and vector autoregressions. Chapter 8 details the Results class, which creates console outputs. Chapter 9 details the Graphics class, which creates application graphics for all Alexandria models. Chapter 10 finally introduces the Datasets class, which is used to load toy datasets for research and replication purposes.

As an alternative to reading the guide, you can use the help function for the class. For instance, for the normal-Wishart Bayesian VAR, you can execute the command:

```
help NormalWishartBayesianVar
```

This will display the help of the class in the console (Figure 5.5), detailing the class constructor, methods and attributes.

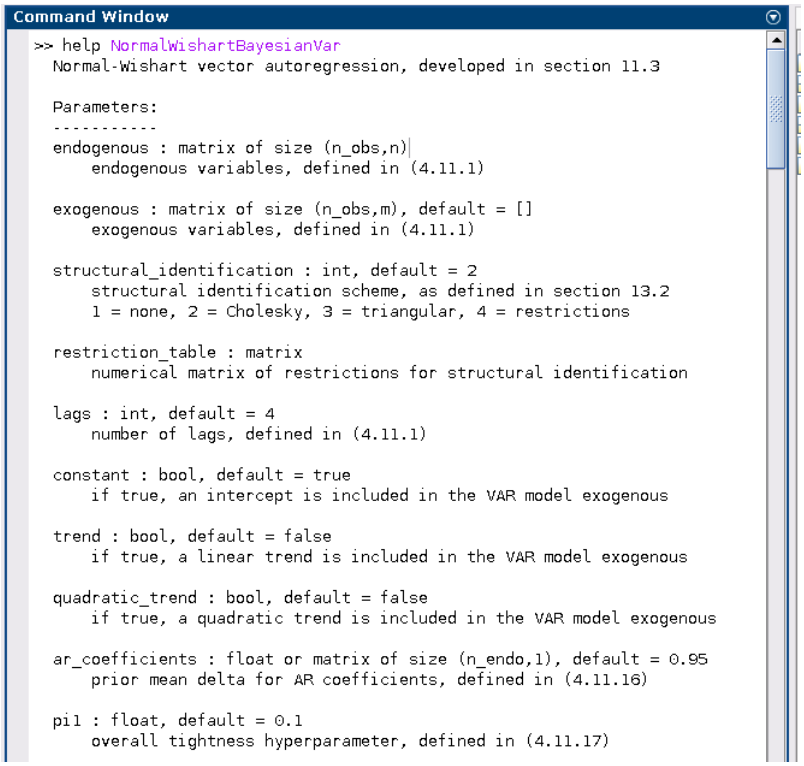


Figure 5.5: The help function for the NormalWishartBayesianVar class

Two remarks to conclude. First, it is recommended that you proceed to a permanent installation if you inted to use Alexandria on the fly. This will save the hastle of creating local copies of the toolbox everytime you want to estimate a model. Second, a use on the fly provides additional flexibility but requires good knowledge of Matlab programming, and in particular some familiarity with the object-oriented paradigm. Novice users might prefer the GUI and IDE for a comfortable experience.

5.5 Running Alexandria on the fly: forecast table input

Section 3.4 details how to create the input table used for forecasts. If you use Alexandria on the fly, these tables must be converted as numerical inputs for the forecast() and forecast_evaluation() methods. As a reminder, the forecast input table is provided in Figure 5.6.

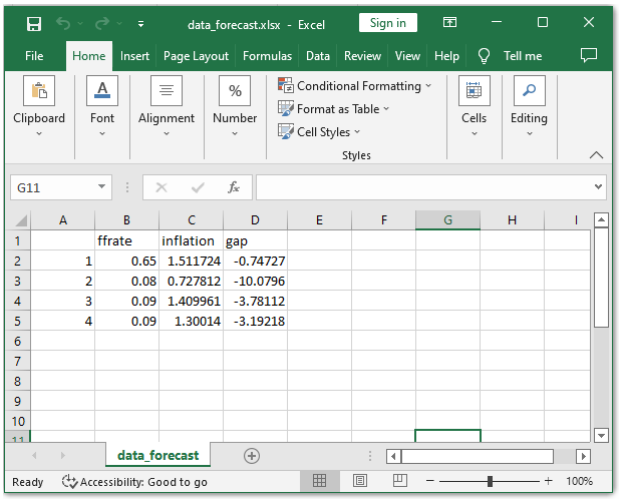


Figure 5.6: Example forecast datafile

■ **Linear regression**: Assume you estimate a linear regression where ffrate is the explained variable and inflation and gap are the regressors. The forecast() method has the following syntax:

```
forecast(X_hat, credibility_level)
```

where X_hat is the predictor matrix for the forecasts, and credibility_level is a float between 0 and 1. X_hat obtains in a straightforward way from the forecast table by taking the regressor columns. For instance, using the data from Figure 5.6, X_hat is given by the following matrix:

```
In [3]: X_hat
Out[3]:
array([[  1.511724,  -0.74727 ],
       [  0.727812, -10.0796  ],
       [  1.409961,  -3.78112 ],
       [  1.30014 ,  -3.19218 ]])
```

Figure 5.7: X_hat matrix for the forecast() method

As for the forecast_evaluation() method, it has the following syntax:

```
forecast_evaluation(y)
```

where y is the vector of counterfactuals used to compare with the forecasts. Using the data from Figure 5.6, y is given by the following matrix:

```
y
array([0.65, 0.08, 0.09, 0.09])
```

Figure 5.8: y matrix for the forecast_evaluation() method

■ **Vector autoregression**: Assume you estimate a vector autoregression where ffrate and inflation as the endogenous variables and gap as an additional exogenous variable. The forecast() method has the following syntax:

```
forecast(h, credibility_level, Z_p=[])
```

where h is the number of forecast periods, credibility_level is a float between 0 and 1, and Z_p is an optional input for additional exogenous variables. In this case, gap can be used for Z_p, which gives the following matrix:

```
In [9]: X_hat
Out[9]:
array([[ -0.74727],
       [-10.0796 ],
       [ -3.78112],
       [ -3.19218]])
```

Figure 5.9: X_hat matrix for the VAR forecast() method

As for the forecast_evaluation() method, it has the following syntax:

```
forecast_evaluation(Y)
```

where Y is the matrix of counterfactuals used to compare with the forecasts. Using the data from Figure 5.6, Y is given by the following matrix:

```
In [11]: Y
Out[11]:
array([[0.65    , 1.511724],
       [0.08    , 0.727812],
       [0.09    , 1.409961],
       [0.09    , 1.30014 ]])
```

Figure 5.10: y matrix for the VAR forecast_evaluation() method

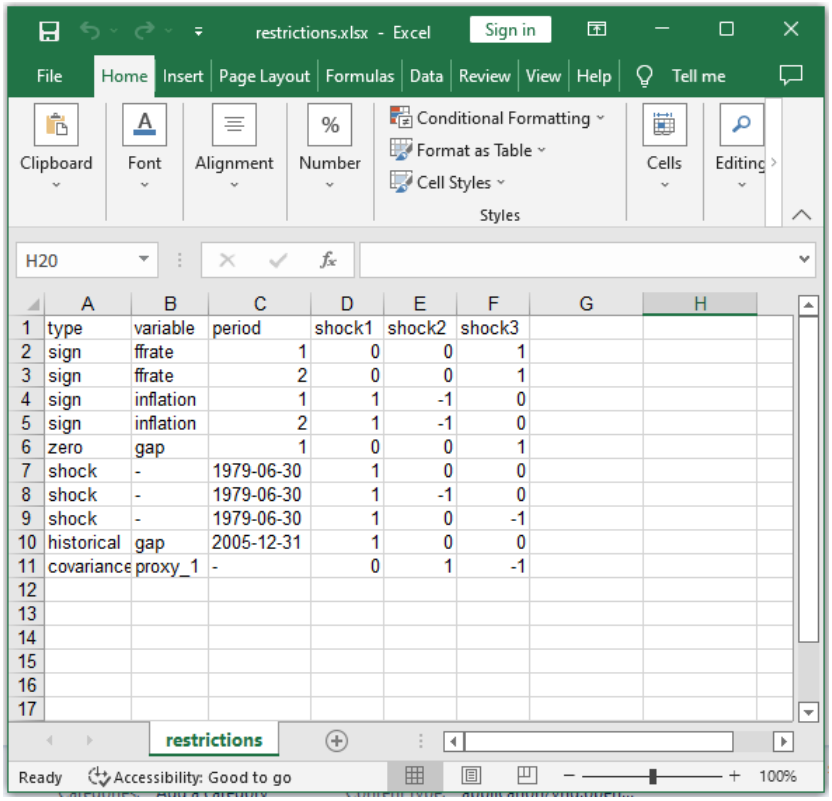
5.6 Running Alexandria on the fly: restrictions table input

Section 3.5 details how to create the input table used for structural identification by restrictions. If you use Alexandria on the fly, these tables must be converted as numerical inputs for the restriction_table argument of all Bayesian VAR models whenever the structural_identification argument is set to 4 (structural identification by restriction). As a reminder, the forecast input table is provided in Figure 5.11.

Assume the VAR model has three endogenous variables: ffrate, inflation, and gap. To convert the table into a numerical matrix, apply the following principles.

■ **"type" column**: "zero" restrictions are assigned the value 1; "sign" restrictions are assigned the value 2; "shock" restrictions are assigned the value 3; "historical" restrictions are assigned the value 4; "covariance" restrictions are assigned the value 5.

■ **"variable" column**: convert the variable to its rank in the list of endogenous variables. Here ffrate is assigned the value 1; inflation is assigned the value 2; and gap is assigned the value 3. For proxys, the same logic applies. In the case of a "shock" restriction, there is no variable so set the value to 0.



	A	B	C	D	E	F	G	H
1	type	variable	period	shock1	shock2	shock3		
2	sign	ffrate	1	0	0	1		
3	sign	ffrate	2	0	0	1		
4	sign	inflation	1	1	-1	0		
5	sign	inflation	2	1	-1	0		
6	zero	gap	1	0	0	1		
7	shock	-	1979-06-30	1	0	0		
8	shock	-	1979-06-30	1	-1	0		
9	shock	-	1979-06-30	1	0	-1		
10	historical	gap	2005-12-31	1	0	0		
11	covariance	proxy_1	-	0	1	-1		
12								
13								
14								
15								
16								
17								

Figure 5.11: Example restriction datafile

■ **"period" column**: for numeric periods, keep the value as it is. For "shock" and "historical" restrictions, use the period number in the sample, regardless of the lags. So for instance if "19979-06-30" is period 134 of your sample, use 134. For covariance restrictions, set the value to 0.

■ **"shocks" columns**: just leave these columns unchanged

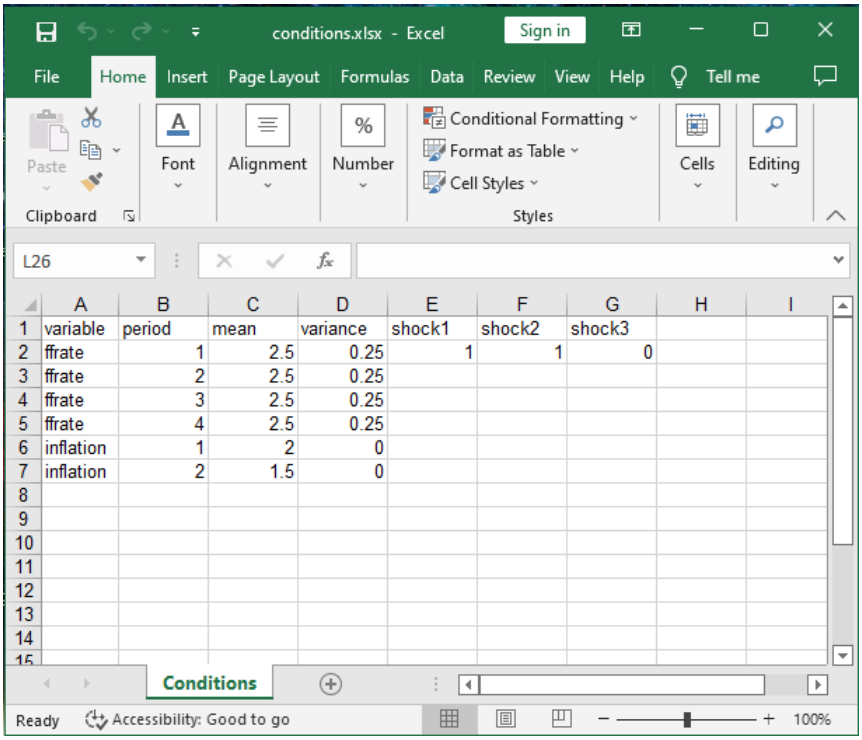
With these principles, the table in Figure 5.11 is converted into the following matrix:

```
In [13]: restrictions
Out[13]:
array([[  2,   1,   1,   0,   0,   1],
       [  2,   1,   2,   0,   0,   1],
       [  2,   2,   1,   1,  -1,   0],
       [  2,   2,   2,   1,  -1,   0],
       [  1,   3,   1,   0,   0,   1],
       [  3,   0,  53,   1,   0,   0],
       [  3,   0,  53,   1,  -1,   0],
       [  3,   0,  53,   1,   0,  -1],
       [  4,   3, 142,   1,   0,   0],
       [  5,   1,   0,   0,   1,  -1]])
```

Figure 5.12: Restriction matrix

5.7 Running Alexandria on the fly: conditional forecasts table input

Section 3.6 details how to create the input table used for conditional forecasts. If you use Alexandria on the fly, this table must be converted as numerical inputs for the conditional_forecast method. As a reminder, the forecast input table is provided in Figure 5.13.



	A	B	C	D	E	F	G	H	I
1	variable	period	mean	variance	shock1	shock2	shock3		
2	ffrate	1	2.5	0.25	1	1	0		
3	ffrate	2	2.5	0.25					
4	ffrate	3	2.5	0.25					
5	ffrate	4	2.5	0.25					
6	inflation	1	2	0					
7	inflation	2	1.5	0					
8									
9									
10									
11									
12									
13									
14									
15									

Figure 5.13: Example conditional forecast datafile

The conditional_forecast() method has the following syntax:

```
conditional_forecast(h, credibility_level, conditions, shocks,
                     conditional_forecast_type, Z_p=[])
```

where h is the number of forecast periods, credibility_level is a float between 0 and 1, conditions is the matrix defining the conditions, shocks is the vector defining the generating shocks (in case of structural conditional forecasts), and conditional_forecast_type is the type of conditional forecasts (agnostic or structual). Z_p is an optional input for additional exogenous variables.

Assume the VAR model has three endogenous variables: ffrate, inflation, and gap. Consider first the conditions matrix. The matrix obtains from the first four columns of the table. All the entries are already numeric save for the variable column that must be converted into numbers. To do that, assign to each variable its rank in the endogenous list. So ffrate is assigned the value 1; inflation is assigned the value 2; and gap is assigned the value 3. This yields the following matrix:

```
In [15]: restrictions
Out[15]:
array([[1.  , 1.  , 2.5 , 0.25],
       [1.  , 2.  , 2.5 , 0.25],
       [1.  , 3.  , 2.5 , 0.25],
       [1.  , 4.  , 2.5 , 0.25],
       [2.  , 1.  , 2.  , 0.  ],
       [2.  , 2.  , 1.5 , 0.  ]])
```

Figure 5.14: conditions matrix

For structural conditional forecasts, a shock matrix must also be supplemented (use shocks = [] for agnostic conditional forecasts). In this case, use the first line of the shock columns and convert them as a vector. For the example in Figure 5.13, this yields:

```
In [17]: shocks
Out[17]: array([1, 1, 0])
```

Figure 5.15: shock matrix

5.8 Running Alexandria on the fly: constrained coefficients table input

Section 3.7 details how to create the input table used for constrained coefficients. If you use Alexandria on the fly, this table must be converted as numerical inputs for the constrained_coefficients_table argument of the Bayesian VAR models whenever the constrained_coefficients argument is set to True. As a reminder, the constrained coefficient table is provided in Figure 5.16.

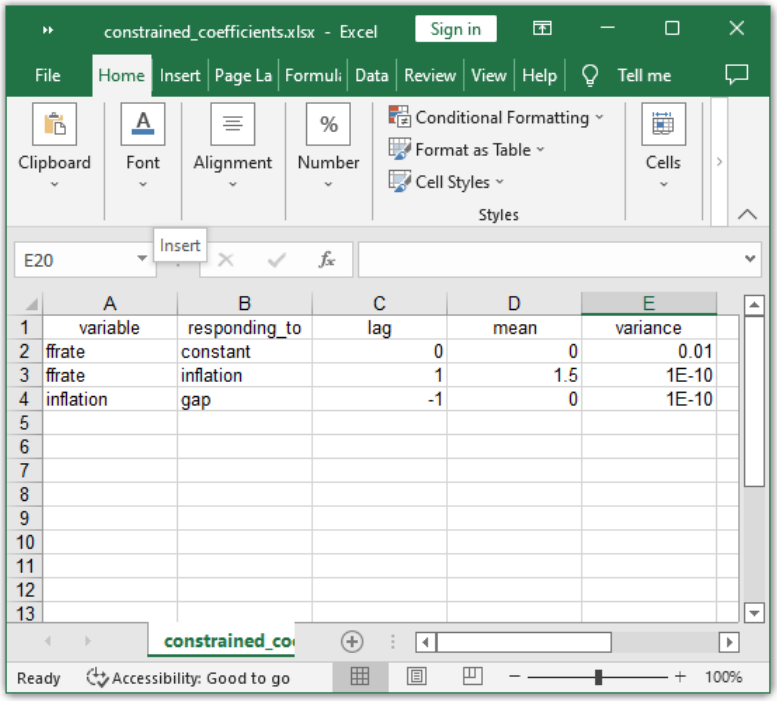


Figure 5.16: Example constrained coefficients datafile

Assume the VAR model has three endogenous variables: ffrate, inflation, and gap. To convert the table into a numerical matrix, simply convert the variables to integer values. For the endogenous variables, use the rank in the list of endogenous variables. So for instance ffrate is assigned value 1, inflation is assigned value 2, and gap is assigned value 3. If there are additional exogenous regressors, use negative values instead, so that the first exogenous variable would be assigned value -1, the second exogenous would be assigned value -2, and so on. Alexandria automatically manages constants, linear trends and quadratic trends. To assign constraints on these variable, use respectively the reserved values 0.1, 0.2 and 0.3. For the columns other than "variable" and "responding_to", simply keep the same values as the table.

Applying these principles, the table in Figure 5.16 is converted to:

```
In [6]: coefficients
Out[6]:
array([[1.000, 0.100, 0.000, 0.000, 0.010],
       [1.000, 2.000, 1.000, 1.500, 0.000],
       [2.000, 3.000, -1.000, 0.000, 0.000]])
```

Figure 5.17: constrained coefficient matrix

5.9 Running Alexandria on the fly: Long-run prior table input

Section 3.8 details how to create the input table used for the long-run prior. If you use Alexandria on the fly, this table must be converted as numerical inputs for the long_run_table argument of the Bayesian VAR models whenever the long_run_prior argument is set to True. As a reminder, the long-run prior table is provided in Figure 5.18.

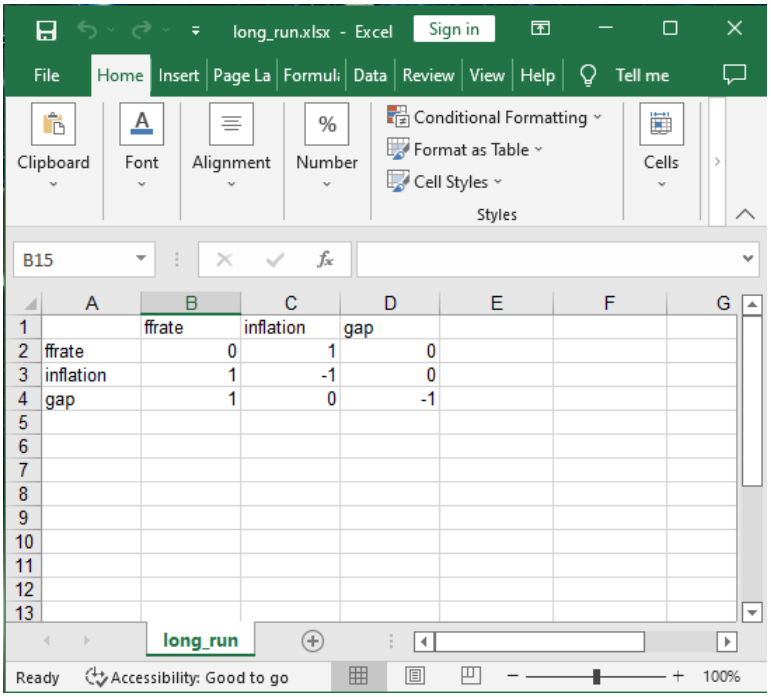


Figure 5.18: Example long-run prior datafile

The conversion to a numerical matrix is trivial: simply convert the table as it is, taking its numerical values. For the example table in Figure 5.18, the equivalent matrix is given by:

```
In [8]: long_run
Out[8]:
array([[ 0,  1,  0],
       [ 1, -1,  0],
       [ 1,  0, -1]])
```

Figure 5.19: long-run prior matrix

CHAPTER 6

Alexandria outputs

Once your model is succesfully estimated, Alexandria will produce a number of estimation outputs. If you selected the option 'save results in project folder" in the GUI/IDE, Alexandria will create a "results" folder within your project folder and will store in it a number of output files. If you selected the option "graphics and figures"in the GUI/IDE, Alexandria will create a "graphics" folder and save in it a copy of all the plots resulting from model estimation.

6.1 Console outputs: linear regression

The first estimation output produced by Alexandria is a console output that summarizes model estimation. A typical output for linear regression is displayed in Figure 6.1.

The output comprises several parts. The top part (green frame) shows the progress bars used to visualize the progression of the different estimation algorithms in real time. It appears only if the progress bar option is selected in the GUI.

The second part (red frame) displays the model estimates for each parameter, including a point estimate and, if applicable, the standard deviations and credibility bands.

The third part exhibits a number of in-sample fit critera, including traditional frequentist criteria (sum of squared residuals, R^2 and adjusted R^2), but also the Bayesian log10 marginal likelihood. This part is displayed only if the in-sample fit/marginal likelihood options are selected in the GUI.

The final part (purple frame) displays a number of out-of-sample forecast evaluation criteria. Here also traditional frequentist criteria (RMSE, MAE, MAPE, Theil's U and bias) are proposed along with specific Bayesian criteria (CRPS and log score). This part is only displayed if the forecast evaluation option is selected in the GUI and proper counterfactual values have been supplied in the forecast data file.

If the option "save results in project folder" is selected in the GUI, a copy of this console output will be saved in the file "results.txt" in your "results" folder. Also, a second file named "settings.txt" will be generated, which recapitulates all the inputs and options of your model.

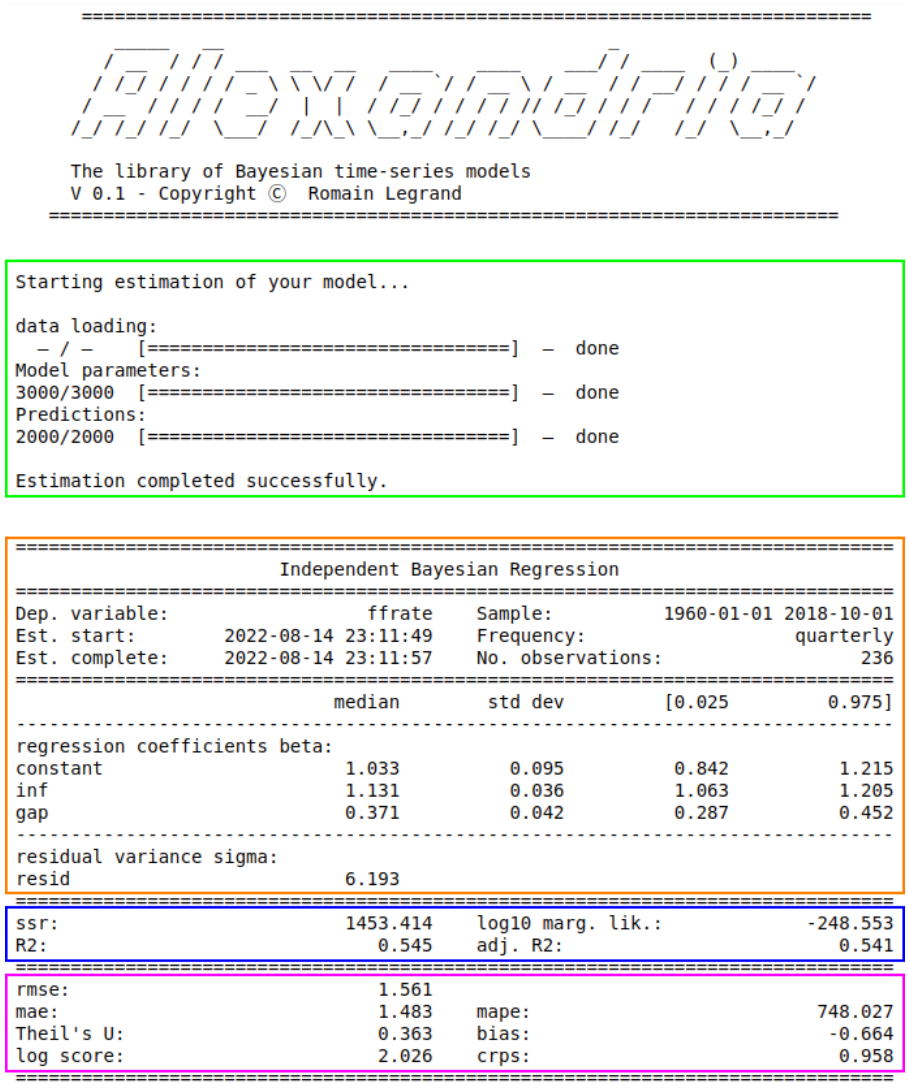


Figure 6.1: Alexandria IDE on Jupyter Notebook

6.2 Console outputs: vector autoregression

The console output of the VAR models is relatively similar to the console output of the linear regression. A typical VAR output is displayed in Figure 6.2.

The output comprises several parts. The top part (green frame) shows the progress bars used to visualize the progression of the different estimation algorithms in real time. It appears only if the progress bar option is selected in the GUI.

The second part (red frame) displays the model estimates for each parameter, including a point estimate and, if applicable, the standard deviations and credibility bands. For a VAR model, the output displays the model equations in a row.

The third part (purple frame) exhibits the variance-covariance matrix for the reduced-form VAR.

This is a minimal output. Other elements may appear depending on the selected options, for instance structural identification matrices, in-sample evaluation criteria, and so on.

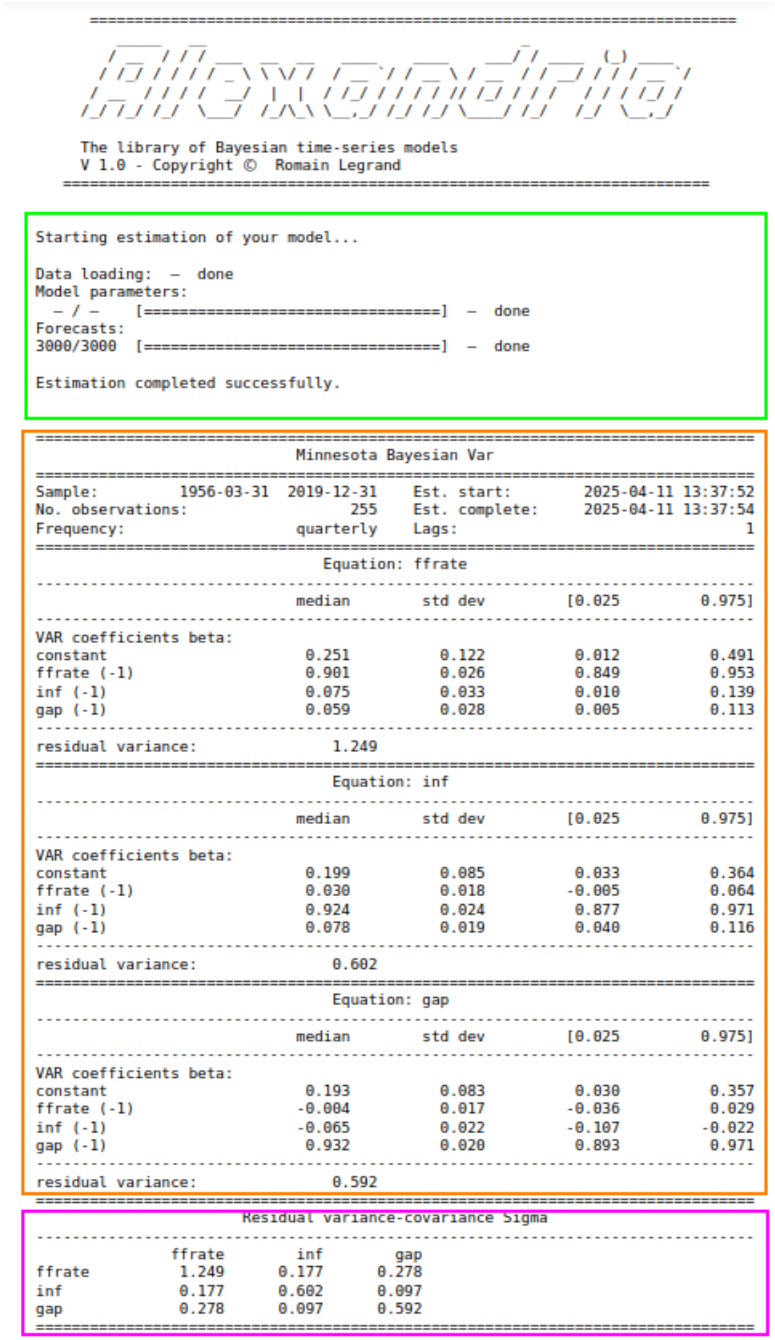


Figure 6.2: Alexandria IDE on Jupyter Notebook

6.3 Graphics outputs

The second type of outputs produced by Alexandria is a set of plots summarizing your model estimates and applications. For convenience, these plots are automatically displayed in tab 4 of the Graphical User Interface, as a simple navigator. The navigator is displayed in Figure 6.3:

The "Application" panel (green frame) allows you to navigate across the different applications. Many applications are available, including "fit", "residuals", "shocks", "steady state", "forecasts", "IRF", "conditional forecasts", "FEVD", and "HD". To switch from one application to another you can either use the slider, or the rollmenu located below it.

The next three panels are designed to provide a smooth navigation across the different graphics. By default Alexandria proposes joint graphics ("Display type" panel in orange frame) , were all plots for a given application are displayed simultaneously, like in Figure 6.3.

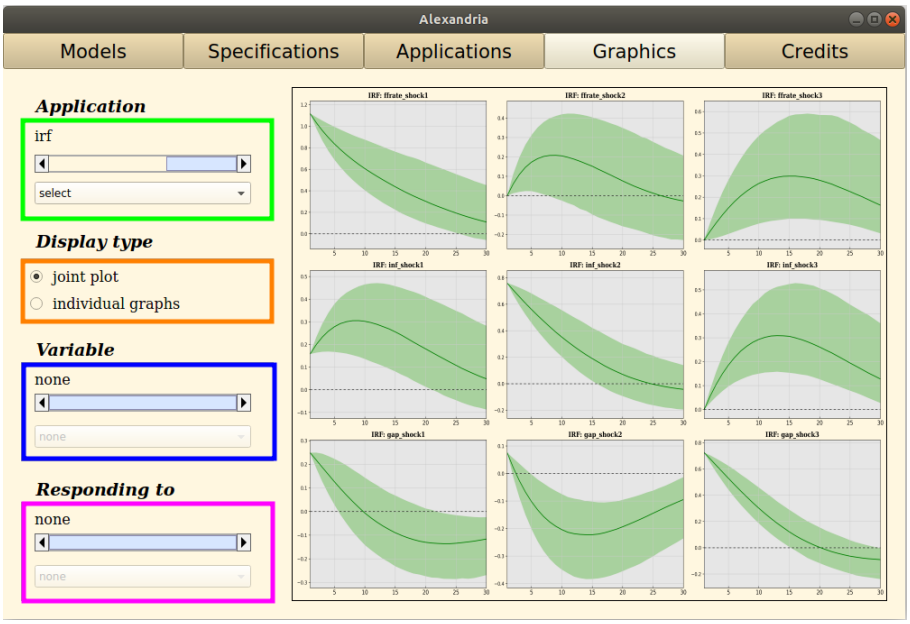


Figure 6.3: Tab 4 of the GUI as a graphical navigator

When there are too many elements to display however, vizualization may become poor. For this reason, the navigator also proposes to select "individual graphs" in the orange panel. You can then set navigate over the "Variable" panel (in blue frame), and "Responding to" panel (in purple frame) to view single elements. See for instance Figure 6.4. For both panels you can either use the slider or the rollmenu located below it.

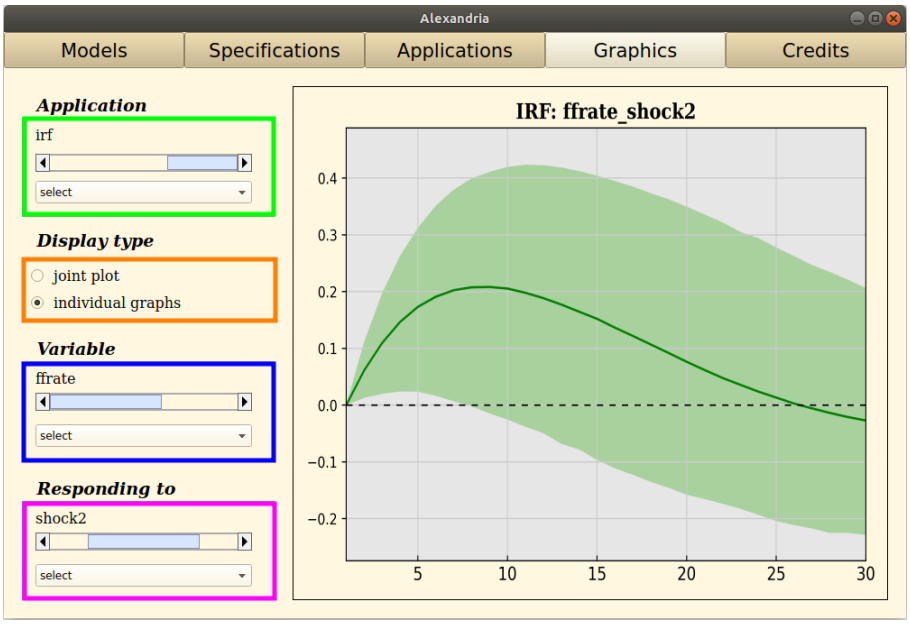


Figure 6.4: Tab 4 of the GUI as a graphical navigator, single element

6.4 File outputs

Alexandria produces a number of estimation output files in the "results" folder of the project. By default, it always saves a copy of the model estimation inputs, along with the console output summary. In addition, it can also save posterior estimates for the applications that have been selected. For instance, if you selected the "forecasts" application, Alexandria will produce a file "forecasts.csv" that recapitulates the prediction estimates. The file is fairly straightforward: it provides a summary of the posterior estimates of the forecasts, indicating for each prediction its lower bound, median and upper bound. An example of such file for the linear regression is displayed in Figure 6.5:

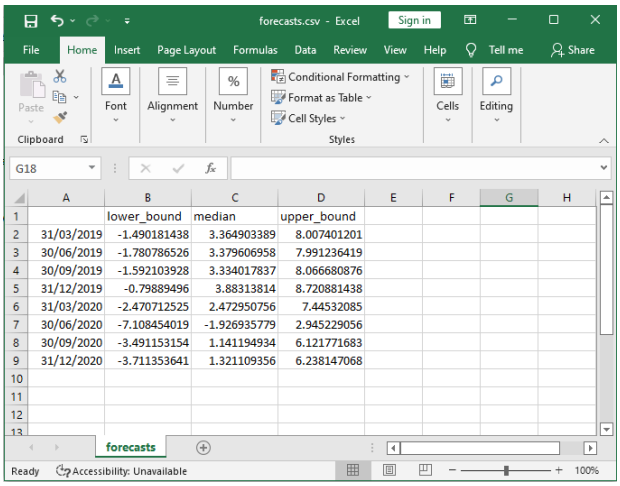


Figure 6.5: Tab 4 of the GUI as a graphical navigator

So far the applications for which Alexandria can record the posterior estimates and save them in csv files are: in-sample fit, forecasts, conditional forecats, impulse response functions, forecast error variance decomposition, and historical decomposition.

CHAPTER 7

Alexandria models - documentation

This chapter proposes an exhaustive documentation for the classes corresponding to the different models proposed in Alexandria. The documentation is provided for the Python version of Alexandria, but it is immediately applicable in a similar way to the Matlab version.

Note also that the documentation can be accessed directly from the console, by using the help function. For instance, to obtain the documentation on the MaximumLikelihoodRegression class, you may simply execute the Python command:

```
help(MaximumLikelihoodRegression)
```

On Matlab, equivalently:

```
help MaximumLikelihoodRegression
```

This will print on the console the same information as that developed in the incoming sections.

7.1 Linear regression: MaximumLikelihoodRegression

A simple maximum likelihood (OLS) regression, described in section 9.1.

Class:

alexandria.linear_regression.MaximumLikelihoodRegression(endogenous, exogenous, constant = True, trend = False, quadratic_trend = False, credibility_level = 0.95, verbose = False)

Parameters:

parameter	description
endogenous	ndarray of shape (n_obs,): endogenous variable, defined in (3.9.1)
exogenous	ndarray of shape (n_obs,n_regressors): exogenous variables, defined in (3.9.1)
constant	bool, default = True: if True, an intercept is included in the regression
trend	bool, default = False: if True, a linear trend is included in the regression
quadratic_trend	bool, default = False: if True, a quadratic trend is included in the regression
credibility_level	float, default = 0.95: credibility level (between 0 and 1)
verbose	bool, default = False: if True, displays a progress bar

Attributes:

attribute	description
endogenous	ndarray of shape (n_obs,): endogenous variable, defined in (3.9.1)
exogenous	ndarray of shape (n_obs,n_regressors): exogenous variables, defined in (3.9.1)
constant	bool: if True, an intercept is included in the regression
trend	bool: if True, a linear trend is included in the regression
quadratic_trend	bool: if True, a quadratic trend is included in the regression
credibility_level	float: credibility level (between 0 and 1)
verbose	bool: if True, displays a progress bar
y	ndarray of shape (n,): endogenous variable, defined in (3.9.3)
X	ndarray of shape (n,k): exogenous variables, defined in (3.9.3)
n	int: number of observations, defined in (3.9.1)
k	int: dimension of beta, defined in (3.9.1)
beta	ndarray of shape (k,): regression coefficients
sigma	float: residual variance, defined in (3.9.1)
beta_estimates	ndarray of shape (k,4): estimates for beta column 1: point estimate; column 2: interval lower bound; column 3: interval upper bound; column 4: standard deviation
sigma	float: residual variance, defined in (3.9.1)
X_hat	ndarray of shape (m,k): predictors for the model
m	int: number of predicted observations, defined in (3.10.1)
forecasts_estimates	ndarray of shape (m,3): estimates for predictions column 1: point estimate; column 2: interval lower bound; column 3: interval upper bound
fitted_estimates	ndarray of shape (n,): posterior estimates (median) for in sample-fit
residual_estimates	ndarray of shape (n,): posterior estimates (median) for residuals
insample_evaluation	dict: in-sample fit evaluation (SSR, R2, adj-R2)
forecast_evaluation_criteria	dict: out-of-sample forecast evaluation (RMSE, MAE, ...)

Methods:

■ **estimate()**
trains the model, produces estimates for parameters β and σ.
parameters:
- none
returns:
- none

■ **insample_fit()**
generates in-sample fit and residuals along with evaluation criteria.
parameters:
- none
returns:
- none

■ **forecast(X_hat, credibility_level)**
predictions for the linear regression model, using (3.10.1) and (3.10.2).
parameters:
- X_hat: ndarray of shape (m,k), predictors for the model
- credibility_level: float, credibility level for predictions (between 0 and 1)
returns:
- forecasts_estimates: ndarray of shape (m, 3), posterior estimates for predictions
column 1: median; column 2: credibility interval lower bound;
column 3: credibility interval upper bound

■ **forecast_evaluation(y)**
forecast evaluation criteria for the linear regression model.
parameters:
- y: ndarray of shape (m,), array of realised values for forecast evaluation
returns:
- none

Example (Python):

```
# imports
from alexandria import MaximumLikelihoodRegression
from alexandria import Data_sets

# load Taylor dataset, split as train/test
ds = Data_sets()
taylor_data = ds.load_taylor()
y_train, X_train = taylor_data[:198,0], taylor_data[:198,1:]
y_test, X_test = taylor_data[198:,0], taylor_data[198:,1:]

# create and train regression
mlr = MaximumLikelihoodRegression(endogenous=y_train, exogenous=X_train,
constant=True)
mlr.estimate()

# get predictions on test sample, run forecast evaluation, display RMSE
estimates_forecasts = mlr.forecast(X_test, 0.95)
mlr.forecast_evaluation(y_test)
print('RMSE on test sample : '
+ str(round(mlr.forecast_evaluation_criteria['rmse'], 2)))

'RMSE on test sample: 3.21'
```

Example (Matlab):

```
% load Taylor dataset, split as train/test
ds = Data_sets();
taylor_data = ds.load_taylor();
y_train = taylor_data(1:198,1); X_train = taylor_data(1:198,2:end);
y_test = taylor_data(198:end,1); X_test = taylor_data(198:end,2:end);

% create and train regression
mlr = MaximumLikelihoodRegression(y_train, X_train, 'constant', true);
mlr.estimate();

% get predictions on test sample, run forecast evaluation, display RMSE
estimates_forecasts = mlr.forecast(X_test, 0.95);
mlr.forecast_evaluation(y_test);
disp(['RMSE on test sample: ' ...
num2str(round(mlr.forecast_evaluation_criteria.rmse, 2))]);

'RMSE on test sample: 3.21'
```

7.2 Linear regression: SimpleBayesianRegression

A simple Bayesian regression, described in section 9.2.

Class:

alexandria.linear_regression.SimpleBayesianRegression(endogenous, exogenous, constant = True, trend = False, quadratic_trend = False, b_exogenous = 0, V_exogenous = 1, b_constant = 0, V_constant = 1, b_trend = 0, V_trend = 1, b_quadratic_trend = 0, V_quadratic_trend = 1, hyperparameter_optimization = False, optimization_type = 1, credibility_level = 0.95, verbose = False)

Parameters:

parameter	description
endogenous	ndarray of shape (n_obs,): endogenous variable, defined in (3.9.3)
exogenous	ndarray of shape (n_obs,n_regressors): exogenous variables, defined in (3.9.3)
constant	bool, default = True: if True, an intercept is included in the regression
trend	bool, default = False: if True, a linear trend is included in the regression
quadratic_trend	bool, default = False: if True, a quadratic trend is included in the regression
b_exogenous	float or ndarray of shape (n_regressors,), default = 0: prior mean for regressors
V_exogenous	float or ndarray of shape (n_regressors,), default = 1: prior variance for regressors
b_constant	float, default = 0: prior mean for constant term
V_constant	float, default = 1: prior variance for constant (positive)
b_trend	float, default = 0: prior mean for trend
V_trend	float, default = 1: prior variance for trend (positive)
b_quadratic_trend	float, default = 0: prior mean for quadratic trend
V_quadratic_trend	float, default = 1: prior variance for quadratic trend (positive)
hyperparameter_optimization	bool, default = False: if True, conducts hyperparameter optimization
optimization_type	int, default = 1: if 1, simple optimzation; if 2: full optimization
credibility_level	float, default = 0.95: credibility level (between 0 and 1)
verbose	bool, default = False: if True, displays a progress bar

Attributes:

attribute	description
endogenous	ndarray of shape (n_obs,): endogenous variable, defined in (3.9.1)
exogenous	ndarray of shape (n_obs,n_regressors): exogenous variables, defined in (3.9.1)
constant	bool: if True, an intercept is included in the regression
trend	bool: if True, a linear trend is included in the regression
quadratic_trend	bool: if True, a quadratic trend is included in the regression
b_exogenous	float or ndarray of shape (n_regressors,): prior mean for regressors
V_exogenous	float or ndarray of shape (n_regressors,): prior variance for regressors
b_constant	float: prior mean for constant term
V_constant	float: prior variance for constant (positive)
b_trend	float: prior mean for trend
V_trend	float: prior variance for trend (positive)
b_quadratic_trend	float: prior mean for quadratic trend
V_quadratic_trend	float: prior variance for quadratic trend (positive)
b	ndarray of shape (k,): prior mean, defined in (3.9.10)
V	ndarray of shape (k,k): prior variance, defined in (3.9.10)
credibility_level	float: credibility level (between 0 and 1)
verbose	bool: if True, displays a progress bar
y	ndarray of shape (n,): endogenous variable, defined in (3.9.3)
X	ndarray of shape (n,k): exogenous variables, defined in (3.9.3)
n	int: number of observations, defined in (3.9.1)
k	int: dimension of beta, defined in (3.9.1)
sigma	float: residual variance, defined in (3.9.1)
b_bar	ndarray of shape (k,): posterior mean, defined in (3.9.14)
V_bar	ndarray of shape (k,k): posterior variance, defined in (3.9.14)
beta_estimates	ndarray of shape (k,4): estimates for beta column 1: point estimate; column 2: interval lower bound; column 3: interval upper bound; column 4: standard deviation
X_hat	ndarray of shape (m,k): predictors for the model
m	int: number of predicted observations, defined in (3.10.1)
forecasts_estimates	ndarray of shape (m,3): estimates for predictions column 1: point estimate; column 2: interval lower bound; column 3: interval upper bound
fitted_estimates	ndarray of shape (n,): posterior estimates (median) for in sample-fit
residual_estimates	ndarray of shape (n,): posterior estimates (median) for residuals
insample_evaluation	dict: in-sample fit evaluation (SSR, R2, adj-R2)
forecast_evaluation_criteria	dict: out-of-sample forecast evaluation (RMSE, MAE, ...)
m_y	float: log10 marginal likelihood

Methods:

■ **estimate**()
trains the model, produces estimates for parameters β and σ
parameters:
- none
returns:
- none

■ **insample_fit()**
generates in-sample fit and residuals along with evaluation criteria.
parameters:
- none
returns:
- none

■ **forecast(X_hat, credibility_level)**
predictions for the linear regression model, using (3.10.1) and (3.10.2).
parameters:
- X_hat: ndarray of shape (m,k), predictors for the model
- credibility_level: float, credibility level for predictions (between 0 and 1)
returns:
- forecasts_estimates: ndarray of shape (m, 3), posterior estimates for predictions
column 1: median; column 2: credibility interval lower bound;
column 3: credibility interval upper bound

■ **forecast_evaluation(y)**
forecast evaluation criteria for the linear regression model.
parameters:
- y: ndarray of shape (m,), array of realised values for forecast evaluation
returns:
- none

■ **marginal_likelihood()**
log10 marginal likelihood, defined in (3.10.20).
parameters:
- none
returns:
- m_y: float, log 10 marginal likelihood value.

Example (Python):

```
# imports
from alexandria import SimpleBayesianRegression
from alexandria import Data_sets
import numpy as np

# load Taylor dataset, split as train/test
ds = Data_sets()
taylor_data = ds.load_taylor()
y_train, X_train = taylor_data[:198,0], taylor_data[:198,1:]
y_test, X_test = taylor_data[198:,0], taylor_data[198:,1:]

# set prior mean and prior variance for the model
b = np.array([1.5, 0.5])
b_const = 1
V = np.array([0.01, 0.0025])
V_const = 0.01

# create and train regression
sbr = SimpleBayesianRegression(endogenous=y_train, exogenous=X_train,
constant=True, b_exogenous=b, V_exogenous=V, b_constant=b_const, V_constant=V_const)
sbr.estimate()
```

```
# get predictions on test sample, run forecast evaluation, display log score
estimates_forecasts = sbr.forecast(X_test, 0.95)
sbr.forecast_evaluation(y_test)
print('log score on test sample : '
+ str(round(sbr.forecast_evaluation_criteria['log_score'], 2)))
```

```
'log score on test sample : 2.17'
```

Example (Matlab):

```
% load Taylor dataset, split as train/test
ds = Data_sets();
taylor_data = ds.load_taylor();
y_train = taylor_data(1:198,1); X_train = taylor_data(1:198,2:end);
y_test = taylor_data(198:end,1); X_test = taylor_data(198:end,2:end);

% set prior mean and prior variance for the model
b = [1.5, 0.5]';
b_const = 1;
V = [0.01, 0.0025]';
V_const = 0.01;

% create and train regression
sbr = SimpleBayesianRegression(y_train, X_train, 'constant', true, 'b_exogenous', b, ...
'V_exogenous', V, 'b_constant', b_const, 'V_constant', V_const);
sbr.estimate();

% get predictions on test sample, run forecast evaluation, display log score
estimates_forecasts = sbr.forecast(X_test, 0.95);
sbr.forecast_evaluation(y_test);
disp(['log score on test sample: ' ...
num2str(round(sbr.forecast_evaluation_criteria.log_score, 2))]);
```

```
'log score on test sample : 2.17'
```

7.3 Linear regression: HierarchicalBayesianRegression

A hierarchical Bayesian regression, described in section 9.3.

Class:

alexandria.linear_regression.HierarchicalBayesianRegression(endogenous, exogenous, constant = True, trend = False, quadratic_trend = False, b_exogenous = 0, V_exogenous = 1, b_constant = 0, V_constant = 1, b_trend = 0, V_trend = 1, b_quadratic_trend = 0, V_quadratic_trend = 1, alpha = 1e-4, delta = 1e-4, hyperparameter_optimization = False, optimization_type = 1, credibility_level = 0.95, verbose = False)

Parameters:

parameter	description
endogenous	ndarray of shape (n_obs,): endogenous variable, defined in (3.9.3)
exogenous	ndarray of shape (n_obs,n_regressors): exogenous variables, defined in (3.9.3)
constant	bool, default = True: if True, an intercept is included in the regression
trend	bool, default = False: if True, a linear trend is included in the regression
quadratic_trend	bool, default = False: if True, a quadratic trend is included in the regression
b_exogenous	float or ndarray of shape (n_regressors,), default = 0: prior mean for regressors
V_exogenous	float or ndarray of shape (n_regressors,), default = 1: prior variance for regressors
b_constant	float, default = 0: prior mean for constant term
V_constant	float, default = 1: prior variance for constant (positive)
b_trend	float, default = 0: prior mean for trend
V_trend	float, default = 1: prior variance for trend (positive)
b_quadratic_trend	float, default = 0: prior mean for quadratic trend
V_quadratic_trend	float, default = 1: prior variance for quadratic trend (positive)
hyperparameter_optimization	bool, default = False: if True, conducts hyperparameter optimization
optimization_type	int, default = 1: if 1, simple optimzation; if 2: full optimization
alpha	float, default = 1e-4: prior shape, defined in (3.9.21)
delta	float, default = 1e-4: prior scale, defined in (3.9.21)
credibility_level	float, default = 0.95: credibility level (between 0 and 1)
verbose	bool, default = False: if True, displays a progress bar

Attributes:

attribute	description
endogenous	ndarray of shape (n_obs,): endogenous variable, defined in (3.9.1)
exogenous	ndarray of shape (n_obs,n_regressors): exogenous variables, defined in (3.9.1)
constant	bool: if True, an intercept is included in the regression
trend	bool: if True, a linear trend is included in the regression
quadratic_trend	bool: if True, a quadratic trend is included in the regression
b_exogenous	float or ndarray of shape (n_regressors,): prior mean for regressors
V_exogenous	float or ndarray of shape (n_regressors,): prior variance for regressors
b_constant	float: prior mean for constant term
V_constant	float: prior variance for constant (positive)
b_trend	float: prior mean for trend
V_trend	float: prior variance for trend (positive)
b_quadratic_trend	float: prior mean for quadratic trend
V_quadratic_trend	float: prior variance for quadratic trend (positive)
b	ndarray of shape (k,): prior mean, defined in (3.9.10)
V	ndarray of shape (k,k): prior variance, defined in (3.9.10)
alpha	float, default = 1e-4: prior shape, defined in (3.9.21)
delta	float, default = 1e-4: prior scale, defined in (3.9.21)
credibility_level	float: credibility level (between 0 and 1)
verbose	bool: if True, displays a progress bar
y	ndarray of shape (n,): endogenous variable, defined in (3.9.3)
X	ndarray of shape (n,k): exogenous variables, defined in (3.9.3)
n	int: number of observations, defined in (3.9.1)
k	int: dimension of beta, defined in (3.9.1)
b_bar	ndarray of shape (k,): posterior mean, defined in (3.9.14)
V_bar	ndarray of shape (k,k): posterior variance, defined in (3.9.14)
alpha_bar	float: posterior shape, defined in (3.9.24)
delta_bar	float: posterior scale, defined in (3.9.24)
location	ndarray of shape (k,): location for the student posterior of beta, defined in (3.9.28)
scale	ndarray of shape (k,k): scale for the student posterior of beta, defined in (3.9.28)
df	float: degrees of freedom for the student posterior of beta, defined in (3.9.28)
beta_estimates	ndarray of shape (k,4): estimates for beta column 1: point estimate; column 2: interval lower bound; column 3: interval upper bound; column 4: standard deviation
sigma_estimates	float: posterior estimate for sigma
X_hat	ndarray of shape (m,k): predictors for the model
m	int: number of predicted observations, defined in (3.10.1)
forecasts_estimates	ndarray of shape (m,3): estimates for predictions column 1: point estimate; column 2: interval lower bound; column 3: interval upper bound
fitted_estimates	ndarray of shape (n,): posterior estimates (median) for in sample-fit
residual_estimates	ndarray of shape (n,): posterior estimates (median) for residuals
insample_evaluation	dict: in-sample fit evaluation (SSR, R2, adj-R2)
forecast_evaluation_criteria	dict: out-of-sample forecast evaluation (RMSE, MAE, ...)
m_y	float: log10 marginal likelihood

Methods:

■ **estimate()**
trains the model, produces estimates for parameters β and σ
parameters:
- none
returns:
- none

■ **insample_fit()**
generates in-sample fit and residuals along with evaluation criteria.
parameters:
- none
returns:
- none

■ **forecast(X_hat, credibility_level)**
predictions for the linear regression model, using (3.10.1) and (3.10.2).
parameters:
- X_hat: ndarray of shape (m,k), predictors for the model
- credibility_level: float, credibility level for predictions (between 0 and 1)
returns:
- forecasts_estimates: ndarray of shape (m, 3), posterior estimates for predictions
column 1: median; column 2: credibility interval lower bound;
column 3: credibility interval upper bound

■ **forecast_evaluation(y)**
forecast evaluation criteria for the linear regression model.
parameters:
- y: ndarray of shape (m,), array of realised values for forecast evaluation
returns:
- none

■ **marginal_likelihood()**
log10 marginal likelihood, defined in (3.10.20).
parameters:
- none
returns:
- m_y: float, log 10 marginal likelihood value.

Example (Python):

```
# imports
from alexandria import HierarchicalBayesianRegression
from alexandria import Data_sets
import numpy as np

# load Taylor dataset, split as train/test
ds = Data_sets()
taylor_data = ds.load_taylor()
y_train, X_train = taylor_data[:198,0], taylor_data[:198,1:]
y_test, X_test = taylor_data[198:,0], taylor_data[198:,1:]
```

```
# set prior mean and prior variance for the model
b = np.array([1.5, 0.5])
b_const = 1
V = np.array([0.01, 0.0025])
V_const = 0.01

# create and train regression
hbr = HierarchicalBayesianRegression(endogenous=y_train, exogenous=X_train,
constant=True, b_exogenous=b, V_exogenous=V, b_constant=b_const, V_constant=V_const)
hbr.estimate()

# get predictions on test sample, run forecast evaluation, display log score
estimates_forecasts = hbr.forecast(X_test, 0.95)
hbr.forecast_evaluation(y_test)
print('log score on test sample : '
+ str(round(hbr.forecast_evaluation_criteria['log_score'], 2)))

'log score on test sample : 2.33'
```

Example (Matlab):

```
% load Taylor dataset, split as train/test
ds = Data_sets();
taylor_data = ds.load_taylor();
y_train = taylor_data(1:198,1); X_train = taylor_data(1:198,2:end);
y_test = taylor_data(198:end,1); X_test = taylor_data(198:end,2:end);

% set prior mean and prior variance for the model
b = [1.5, 0.5]';
b_const = 1;
V = [0.01, 0.0025]';
V_const = 0.01;

% create and train regression
hbr = HierarchicalBayesianRegression(y_train, X_train, 'constant', true, 'b_exogenous', b, ..
'V_exogenous', V, 'b_constant', b_const, 'V_constant', V_const);
hbr.estimate();

% get predictions on test sample, run forecast evaluation, display log score
estimates_forecasts = hbr.forecast(X_test, 0.95);
hbr.forecast_evaluation(y_test);
disp(['log score on test sample: ' ...
num2str(round(hbr.forecast_evaluation_criteria.log_score, 2))]);

'log score on test sample : 2.34'
```

7.4 Linear regression: IndependentBayesianRegression

A independent Bayesian regression with Gibbs sampling, described in section 9.4.

Class:

alexandria.linear_regression.IndependentBayesianRegression(endogenous, exogenous, constant = True, trend = False, quadratic_trend = False, b_exogenous = 0, V_exogenous = 1, b_constant = 0, V_constant = 1, b_trend = 0, V_trend = 1, b_quadratic_trend = 0, V_quadratic_trend = 1, alpha = 1e-4, delta = 1e-4, iterations = 2000, burn = 1000, credibility_level = 0.95, verbose = False)

Parameters:

parameter	description
endogenous	ndarray of shape (n_obs,): endogenous variable, defined in (3.9.3)
exogenous	ndarray of shape (n_obs,n_regressors): exogenous variables, defined in (3.9.3)
constant	bool, default = True: if True, an intercept is included in the regression
trend	bool, default = False: if True, a linear trend is included in the regression
quadratic_trend	bool, default = False: if True, a quadratic trend is included in the regression
b_exogenous	float or ndarray of shape (n_regressors,), default = 0: prior mean for regressors
V_exogenous	float or ndarray of shape (n_regressors,), default = 1: prior variance for regressors
b_constant	float, default = 0: prior mean for constant term
V_constant	float, default = 1: prior variance for constant (positive)
b_trend	float, default = 0: prior mean for trend
V_trend	float, default = 1: prior variance for trend (positive)
b_quadratic_trend	float, default = 0: prior mean for quadratic trend
V_quadratic_trend	float, default = 1: prior variance for quadratic trend (positive)
alpha	float, default = 1e-4: prior shape, defined in (3.9.21)
delta	float, default = 1e-4: prior scale, defined in (3.9.21)
iterations	int, default = 2000: post burn-in iterations for MCMC algorithm
burn	int, default = 1000: burn-in iterations for MCMC algorithm
credibility_level	float, default = 0.95: credibility level (between 0 and 1)
verbose	bool, default = False: if True, displays a progress bar

Attributes:

attribute	description
endogenous	ndarray of shape (n_obs,): endogenous variable, defined in (3.9.1)
exogenous	ndarray of shape (n_obs,n_regressors): exogenous variables, defined in (3.9.1)
constant	bool: if True, an intercept is included in the regression
trend	bool: if True, a linear trend is included in the regression
quadratic_trend	bool: if True, a quadratic trend is included in the regression
b_exogenous	float or ndarray of shape (n_regressors,): prior mean for regressors
V_exogenous	float or ndarray of shape (n_regressors,): prior variance for regressors
b_constant	float: prior mean for constant term
V_constant	float: prior variance for constant (positive)
b_trend	float: prior mean for trend
V_trend	float: prior variance for trend (positive)
b_quadratic_trend	float: prior mean for quadratic trend
V_quadratic_trend	float: prior variance for quadratic trend (positive)
b	ndarray of shape (k,): prior mean, defined in (3.9.10)
V	ndarray of shape (k,k): prior variance, defined in (3.9.10)
alpha	float, default = 1e-4: prior shape, defined in (3.9.21)
delta	float, default = 1e-4: prior scale, defined in (3.9.21)
iterations	int, default = 2000: post burn-in iterations for MCMC algorithm
burn	int, default = 1000: burn-in iterations for MCMC algorithm
credibility_level	float: credibility level (between 0 and 1)
verbose	bool: if True, displays a progress bar
y	ndarray of shape (n,): endogenous variable, defined in (3.9.3)
X	ndarray of shape (n,k): exogenous variables, defined in (3.9.3)
n	int: number of observations, defined in (3.9.1)
k	int: dimension of beta, defined in (3.9.1)
alpha_bar	float: posterior shape, defined in (3.9.24)
mcmc_beta	matrix of size (k, iterations): storage of mcmc values for beta
mcmc_sigma	vector of size (iterations,): storage of mcmc values for sigma
beta_estimates	ndarray of shape (k,4): estimates for beta column 1: point estimate; column 2: interval lower bound; column 3: interval upper bound; column 4: standard deviation
sigma_estimates	float: posterior estimate for sigma
X_hat	ndarray of shape (m,k): predictors for the model
m	int: number of predicted observations, defined in (3.10.1)
mcmc_forecasts	matrix of size (m, iterations): storage of mcmc values for forecasts
forecasts_estimates	ndarray of shape (m,3): estimates for predictions column 1: point estimate; column 2: interval lower bound; column 3: interval upper bound
fitted_estimates	ndarray of shape (n,): posterior estimates (median) for in sample-fit
residual_estimates	ndarray of shape (n,): posterior estimates (median) for residuals
insample_evaluation	dict: in-sample fit evaluation (SSR, R2, adj-R2)
forecast_evaluation_criteria	dict: out-of-sample forecast evaluation (RMSE, MAE, ...)
m_y	float: log10 marginal likelihood

Methods:

■ **estimate()**
trains the model, produces estimates for parameters β and σ
parameters:
- none
returns:
- none

■ **insample_fit()**
generates in-sample fit and residuals along with evaluation criteria.
parameters:
- none
returns:
- none

■ **forecast(X_hat, credibility_level)**
predictions for the linear regression model, using (3.10.1) and (3.10.2).
parameters:
- X_hat: ndarray of shape (m,k), predictors for the model
- credibility_level: float, credibility level for predictions (between 0 and 1)
returns:
- forecasts_estimates: ndarray of shape (m, 3), posterior estimates for predictions
column 1: median; column 2: credibility interval lower bound;
column 3: credibility interval upper bound

■ **forecast_evaluation(y)**
forecast evaluation criteria for the linear regression model.
parameters:
- y: ndarray of shape (m,), array of realised values for forecast evaluation
returns:
- none

■ **marginal_likelihood()**
log10 marginal likelihood, defined in (3.10.20).
parameters:
- none
returns:
- m_y: float, log 10 marginal likelihood value.

Example (Python):

```
# imports
from alexandria import IndependentBayesianRegression
from alexandria import Data_sets
import numpy as np

# load Taylor dataset, split as train/test
ds = Data_sets()
taylor_data = ds.load_taylor()
y_train, X_train = taylor_data[:198,0], taylor_data[:198,1:]
y_test, X_test = taylor_data[198:,0], taylor_data[198:,1:]
```

```
# set prior mean and prior variance for the model
b = np.array([1.5, 0.5])
b_const = 1
V = np.array([0.01, 0.0025])
V_const = 0.01

# create and train regression
ibr = IndependentBayesianRegression(endogenous=y_train, exogenous=X_train,
constant=True, b_exogenous=b, V_exogenous=V, b_constant=b_const, V_constant=V_const)
ibr.estimate()

# get predictions on test sample, run forecast evaluation, display log score
estimates_forecasts = ibr.forecast(X_test, 0.95)
ibr.forecast_evaluation(y_test)
print('log score on test sample : '
+ str(round(ibr.forecast_evaluation_criteria['log_score'], 2)))

'log score on test sample : 2.18'
```

Example (Matlab):

```
% load Taylor dataset, split as train/test
ds = Data_sets();
taylor_data = ds.load_taylor();
y_train = taylor_data(1:198,1); X_train = taylor_data(1:198,2:end);
y_test = taylor_data(198:end,1); X_test = taylor_data(198:end,2:end);

% set prior mean and prior variance for the model
b = [1.5, 0.5]';
b_const = 1;
V = [0.01, 0.0025]';
V_const = 0.01;

% create and train regression
ibr = IndependentBayesianRegression(y_train, X_train, 'constant', true, 'b_exogenous', b, ...
'V_exogenous', V, 'b_constant', b_const, 'V_constant', V_const);
ibr.estimate();

% get predictions on test sample, run forecast evaluation, display log score
estimates_forecasts = ibr.forecast(X_test, 0.95);
ibr.forecast_evaluation(y_test);
disp(['log score on test sample: ' ...
num2str(round(ibr.forecast_evaluation_criteria.log_score, 2))]);

'log score on test sample : 2.18'
```

7.5 Linear regression: HeteroscedasticBayesianRegressionn

A Bayesian regression with heteroscedastic disturbances, described in section 9.5.

Class:

alexandria.linear_regression.HeteroscedasticBayesianRegression(endogenous, exogenous, heteroscedastic = None, constant = True, trend = False, quadratic_trend = False, b_exogenous = 0, V_exogenous = 1, b_constant = 0, V_constant = 1, b_trend = 0, V_trend = 1, b_quadratic_trend = 0, V_quadratic_trend = 1, alpha = 1e-4, g = 0, Q = 100, tau = 0.001, delta = 1e-4, iterations = 2000, burn = 1000, thinning = False, thinning_frequency = 10, credibility_level = 0.95, verbose = False)

Parameters:

parameter	description
endogenous	ndarray of shape (n_obs,): endogenous variable, defined in (3.9.3)
exogenous	ndarray of shape (n_obs,n_regressors): exogenous variables, defined in (3.9.3)
heteroscedastic	ndarray of shape (n_obs,h), default = exogenous: heteroscedasticity variables, defined in (3.9.37)
constant	bool, default = True: if True, an intercept is included in the regression
trend	bool, default = False: if True, a linear trend is included in the regression
quadratic_trend	bool, default = False: if True, a quadratic trend is included in the regression
b_exogenous	float or ndarray of shape (n_regressors,), default = 0: prior mean for regressors
V_exogenous	float or ndarray of shape (n_regressors,), default = 1: prior variance for regressors
b_constant	float, default = 0: prior mean for constant term
V_constant	float, default = 1: prior variance for constant (positive)
b_trend	float, default = 0: prior mean for trend
V_trend	float, default = 1: prior variance for trend (positive)
b_quadratic_trend	float, default = 0: prior mean for quadratic trend
V_quadratic_trend	float, default = 1: prior variance for quadratic trend (positive)
alpha	float, default = 1e-4: prior shape, defined in (3.9.21)
delta	float, default = 1e-4: prior scale, defined in (3.9.21)
g	float or ndarray of shape (h,), default = 0: prior mean, defined in (3.9.43)
Q	float or ndarray of shape (h,), default = 100: prior variance, defined in (3.9.43)
tau	float, default = 0.001: variance of the random walk shock, defined in (3.9.50)
iterations	int, default = 2000: post burn-in iterations for MCMC algorithm
burn	int, default = 1000: burn-in iterations for MCMC algorithm
thinning	bool, default = False: if True, thinning is applied to posterior draws from MCMC algorithm
thinning_frequency	int, default = 10: if thinning is True, retains only one out of so many draws from MCMC algorithm
credibility_level	float, default = 0.95: credibility level (between 0 and 1)
verbose	bool, default = False: if True, displays a progress bar

Attributes:

attribute	description
endogenous	ndarray of shape (n_obs,): endogenous variable, defined in (3.9.1)
exogenous	ndarray of shape (n_obs,n_regressors): exogenous variables, defined in (3.9.1)
heteroscedastic	ndarray of shape (n_obs,h), default = exogenous: heteroscedasticity variables, defined in (3.9.37)
constant	bool: if True, an intercept is included in the regression
trend	bool: if True, a linear trend is included in the regression
quadratic_trend	bool: if True, a quadratic trend is included in the regression
b_exogenous	float or ndarray of shape (n_regressors,): prior mean for regressors
V_exogenous	float or ndarray of shape (n_regressors,): prior variance for regressors
b_constant	float: prior mean for constant term
V_constant	float: prior variance for constant (positive)
b_trend	float: prior mean for trend
V_trend	float: prior variance for trend (positive)
b_quadratic_trend	float: prior mean for quadratic trend
V_quadratic_trend	float: prior variance for quadratic trend (positive)
b	ndarray of shape (k,): prior mean, defined in (3.9.10)
V	ndarray of shape (k,k): prior variance, defined in (3.9.10)
alpha	float, default = 1e-4: prior shape, defined in (3.9.21)
delta	float, default = 1e-4: prior scale, defined in (3.9.21)
g	ndarray of shape (h,): prior mean, defined in (3.9.43)
Q	ndarray of shape (h,h): prior variance, defined in (3.9.43)
tau	float: variance of the random walk shock, defined in (3.9.50)
iterations	int: post burn-in iterations for MCMC algorithm
burn	int: burn-in iterations for MCMC algorithm
thinning	bool: if True, thinning is applied to posterior draws from MCMC algorithm
thinning_frequency	int: if thinning is True, retains only one out of so many draws from MCMC algorithm
credibility_level	float: credibility level (between 0 and 1)
verbose	bool: if True, displays a progress bar
y	ndarray of shape (n,): endogenous variable, defined in (3.9.3)
X	ndarray of shape (n,k): exogenous variables, defined in (3.9.3)
Z	ndarray of shape (n,h): heteroscedasticity variables, defined in (3.9.39)
n	int: number of observations, defined in (3.9.1)
k	int: dimension of beta, defined in (3.9.1)
h	int: dimension of gamma, defined in (3.9.37)
alpha_bar	float: posterior shape, defined in (3.9.24)
mcmc_beta	matrix of size (k, iterations): storage of mcmc values for beta
mcmc_sigma	vector of size (iterations,): storage of mcmc values for sigma
mcmc_gamma	ndarray of shape (h,iterations): storage of mcmc values for gamma
beta_estimates	ndarray of shape (k,4): estimates for beta column 1: point estimate; column 2: interval lower bound; column 3: interval upper bound; column 4: standard deviation
sigma_estimates	float: posterior estimate for sigma
gamma_estimates	ndarray of shape (h,3): posterior estimates for gamma column 1: median; column 2: interval lower bound; column 3: interval upper bound
X_hat	ndarray of shape (m,k): predictors for the model
Z_hat	ndarray of shape (m,h): heteroscedasticity predictors for the model
m	int: number of predicted observations, defined in (3.10.1)
mcmc_forecasts	matrix of size (m, iterations): storage of mcmc values for forecasts
forecasts_estimates	ndarray of shape (m,3): estimates for predictions column 1: point estimate; column 2: interval lower bound; column 3: interval upper bound
fitted_estimates	ndarray of shape (n,): posterior estimates (median) for in sample-fit
residual_estimates	ndarray of shape (n,): posterior estimates (median) for residuals
insample_evaluation	dict: in-sample fit evaluation (SSR, R2, adj-R2)
forecast_evaluation_criteria	dict: out-of-sample forecast evaluation (RMSE, MAE, ...)
m_y	float: log10 marginal likelihood

Methods:

■ **estimate()**
trains the model, produces estimates for parameters β, σ and γ
parameters:
- none
returns:
- none

■ **insample_fit()**
generates in-sample fit and residuals along with evaluation criteria.
parameters:
- none
returns:
- none

■ **forecast(X_hat, credibility_level)**
predictions for the linear regression model, using (3.10.1) and (3.10.2).
parameters:
- X_hat: ndarray of shape (m,k), predictors for the model
- credibility_level: float, credibility level for predictions (between 0 and 1)
returns:
- forecasts_estimates: ndarray of shape (m, 3), posterior estimates for predictions
column 1: median; column 2: credibility interval lower bound;
column 3: credibility interval upper bound

■ **forecast_evaluation(y)**
forecast evaluation criteria for the linear regression model.
parameters:
- y: ndarray of shape (m,), array of realised values for forecast evaluation
returns:
- none

■ **marginal_likelihood()**
log10 marginal likelihood, defined in (3.10.20).
parameters:
- none
returns:
- m_y: float, log 10 marginal likelihood value.

Example (Python):

```
# imports
from alexandria import HeteroscedasticBayesianRegression
from alexandria import Data_sets
import numpy as np

# load Taylor dataset, split as train/test
ds = Data_sets()
taylor_data = ds.load_taylor()
y_train, X_train = taylor_data[:198,0], taylor_data[:198,1:]
y_test, X_test = taylor_data[198:,0], taylor_data[198:,1:]
```

```
# set prior mean and prior variance for the model
b = np.array([1.5, 0.5])
b_const = 1
V = np.array([0.01, 0.0025])
V_const = 0.01

# create and train regression
hbr = HeteroscedasticBayesianRegression(endogenous=y_train, exogenous=X_train,
constant=True, b_exogenous=b, V_exogenous=V, b_constant=b_const, V_constant=V_const)
hbr.estimate()

# get predictions on test sample, run forecast evaluation, display log score
estimates_forecasts = hbr.forecast(X_test, 0.95)
hbr.forecast_evaluation(y_test)
print('log score on test sample : '
+ str(round(hbr.forecast_evaluation_criteria['log_score'], 2)))

'log score on test sample : 2.14'
```

Example (Matlab):

```
% load Taylor dataset, split as train/test
ds = Data_sets();
taylor_data = ds.load_taylor();
y_train = taylor_data(1:198,1); X_train = taylor_data(1:198,2:end);
y_test = taylor_data(198:end,1); X_test = taylor_data(198:end,2:end);

% set prior mean and prior variance for the model
b = [1.5, 0.5]';
b_const = 1;
V = [0.01, 0.0025]';
V_const = 0.01;

% create and train regression
hbr = HeteroscedasticBayesianRegression(y_train, X_train, 'constant', true, 'b_exogenous', b,
'V_exogenous', V, 'b_constant', b_const, 'V_constant', V_const);
hbr.estimate();

% get predictions on test sample, run forecast evaluation, display log score
estimates_forecasts = hbr.forecast(X_test, 0.95);
hbr.forecast_evaluation(y_test);
disp(['log score on test sample: ' ...
num2str(round(hbr.forecast_evaluation_criteria.log_score, 2))]);

'log score on test sample : 2.14'
```

7.6 Linear regression: AutocorrelatedBayesianRegression

A Bayesian regression with autocorrelated disturbances, described in section 9.6.

Class:

alexandria.linear_regression.AutocorrelatedBayesianRegression(endogenous, exogenous, q = 1, constant = True, trend = False, quadratic_trend = False, b_exogenous = 0, V_exogenous = 1, b_constant = 0, V_constant = 1, b_trend = 0, V_trend = 1, b_quadratic_trend = 0, V_quadratic_trend = 1, alpha = 1e-4, delta = 1e-4, p = 0, H = 100, iterations = 2000, burn = 1000, credibility_level = 0.95, verbose = False)

Parameters:

parameter	description
endogenous	ndarray of shape (n_obs,): endogenous variable, defined in (3.9.3)
exogenous	ndarray of shape (n_obs,n_regressors): exogenous variables, defined in (3.9.3)
q	int, default = 1: order of autocorrelation (number of residual lags)
constant	bool, default = True: if True, an intercept is included in the regression
trend	bool, default = False: if True, a linear trend is included in the regression
quadratic_trend	bool, default = False: if True, a quadratic trend is included in the regression
b_exogenous	float or ndarray of shape (n_regressors,), default = 0: prior mean for regressors
V_exogenous	float or ndarray of shape (n_regressors,), default = 1: prior variance for regressors
b_constant	float, default = 0: prior mean for constant term
V_constant	float, default = 1: prior variance for constant (positive)
b_trend	float, default = 0: prior mean for trend
V_trend	float, default = 1: prior variance for trend (positive)
b_quadratic_trend	float, default = 0: prior mean for quadratic trend
V_quadratic_trend	float, default = 1: prior variance for quadratic trend (positive)
alpha	float, default = 1e-4: prior shape, defined in (3.9.21)
delta	float, default = 1e-4: prior scale, defined in (3.9.21)
p	float or ndarray of shape (q,), default = 0: prior mean, defined in (3.9.62)
H	float or ndarray of shape (q,), default = 100: prior variance, defined in (3.9.62)
iterations	int, default = 2000: post burn-in iterations for MCMC algorithm
burn	int, default = 1000: burn-in iterations for MCMC algorithm
credibility_level	float, default = 0.95: credibility level (between 0 and 1)
verbose	bool, default = False: if True, displays a progress bar

Attributes:

attribute	description
endogenous	ndarray of shape (n_obs,): endogenous variable, defined in (3.9.1)
exogenous	ndarray of shape (n_obs,n_regressors): exogenous variables, defined in (3.9.1)
q	int: order of autocorrelation (number of residual lags)
constant	bool: if True, an intercept is included in the regression
trend	bool: if True, a linear trend is included in the regression
quadratic_trend	bool: if True, a quadratic trend is included in the regression
b_exogenous	ndarray of shape (n_regressors,): prior mean for regressors
V_exogenous	ndarray of shape (n_regressors,): prior variance for regressors
b_constant	float: prior mean for constant term
V_constant	float: prior variance for constant (positive)
b_trend	float: prior mean for trend
V_trend	float: prior variance for trend (positive)
b_quadratic_trend	float: prior mean for quadratic trend
V_quadratic_trend	float: prior variance for quadratic trend (positive)
b	ndarray of shape (k,): prior mean, defined in (3.9.10)
V	ndarray of shape (k,k): prior variance, defined in (3.9.10)
alpha	float: prior shape, defined in (3.9.21)
delta	float: prior scale, defined in (3.9.21)
p	ndarray of shape (q,): prior mean, defined in (3.9.62)
H	ndarray of shape (q,q): prior variance, defined in (3.9.62)
iterations	int: post burn-in iterations for MCMC algorithm
burn	int: burn-in iterations for MCMC algorithm
credibility_level	float: credibility level (between 0 and 1)
verbose	bool: if True, displays a progress bar
y	ndarray of shape (n,): endogenous variable, defined in (3.9.3)
X	ndarray of shape (n,k): exogenous variables, defined in (3.9.3)
T	int: number of observations, defined in (3.9.52)
k	int: dimension of beta, defined in (3.9.1)
alpha_bar	float: posterior shape, defined in (3.9.24)
mcmc_beta	matrix of size (k, iterations): storage of mcmc values for beta
mcmc_sigma	vector of size (iterations,): storage of mcmc values for sigma
mcmc_phi	ndarray of shape (q,iterations): storage of mcmc values for phi
beta_estimates	ndarray of shape (k,4): estimates for beta column 1: point estimate; column 2: interval lower bound; column 3: interval upper bound; column 4: standard deviation
sigma_estimates	float: posterior estimate for sigma
phi_estimates	ndarray of shape (q,3): posterior estimates for phi column 1: median; column 2: interval lower bound; column 3: interval upper bound
X_hat	ndarray of shape (m,k): predictors for the model
m	int: number of predicted observations, defined in (3.10.1)
mcmc_forecasts	matrix of size (m, iterations): storage of mcmc values for forecasts
forecasts_estimates	ndarray of shape (m,3): estimates for predictions column 1: point estimate; column 2: interval lower bound; column 3: interval upper bound
fitted_estimates	ndarray of shape (n,): posterior estimates (median) for in sample-fit
residual_estimates	ndarray of shape (n,): posterior estimates (median) for residuals
insample_evaluation	dict: in-sample fit evaluation (SSR, R2, adj-R2)
forecast_evaluation_criteria	dict: out-of-sample forecast evaluation (RMSE, MAE, ...)
m_y	float: log10 marginal likelihood

Methods:

■ **estimate()**
trains the model, produces estimates for parameters β, σ and γ
parameters:
- none
returns:
- none

■ **insample_fit()**
generates in-sample fit and residuals along with evaluation criteria.
parameters:
- none
returns:
- none

■ **forecast(X_hat, credibility_level)**
predictions for the linear regression model, using (3.10.1) and (3.10.2).
parameters:
- X_hat: ndarray of shape (m,k), predictors for the model
- credibility_level: float, credibility level for predictions (between 0 and 1)
returns:
- forecasts_estimates: ndarray of shape (m, 3), posterior estimates for predictions
column 1: median; column 2: credibility interval lower bound;
column 3: credibility interval upper bound

■ **forecast_evaluation(y)**
forecast evaluation criteria for the linear regression model.
parameters:
- y: ndarray of shape (m,), array of realised values for forecast evaluation
returns:
- none

■ **marginal_likelihood()**
log10 marginal likelihood, defined in (3.10.20).
parameters:
- none
returns:
- m_y: float, log 10 marginal likelihood value.

Example (Python):

```
# imports
from alexandria import AutocorrelatedBayesianRegression
from alexandria import Data_sets
import numpy as np

# load Taylor dataset, split as train/test
ds = Data_sets()
taylor_data = ds.load_taylor()
y_train, X_train = taylor_data[:198,0], taylor_data[:198,1:]
y_test, X_test = taylor_data[198:,0], taylor_data[198:,1:]
```

```
# set prior mean and prior variance for the model
b = np.array([1.5, 0.5])
b_const = 1
V = np.array([0.01, 0.0025])
V_const = 0.01

# create and train regression
abr = AutocorrelatedBayesianRegression(endogenous=y_train, exogenous=X_train,
constant=True, b_exogenous=b, V_exogenous=V, b_constant=b_const, V_constant=V_const)
abr.estimate()

# get predictions on test sample, run forecast evaluation, display log score
estimates_forecasts = abr.forecast(X_test, 0.95)
abr.forecast_evaluation(y_test)
print('log score on test sample : '
+ str(round(abr.forecast_evaluation_criteria['log_score'], 2)))

'log score on test sample : 2.12'
```

Example (Matlab):

```
% load Taylor dataset, split as train/test
ds = Data_sets();
taylor_data = ds.load_taylor();
y_train = taylor_data(1:198,1); X_train = taylor_data(1:198,2:end);
y_test = taylor_data(198:end,1); X_test = taylor_data(198:end,2:end);

% set prior mean and prior variance for the model
b = [1.5, 0.5]';
b_const = 1;
V = [0.01, 0.0025]';
V_const = 0.01;

% create and train regression
abr = AutocorrelatedBayesianRegression(y_train, X_train, 'constant', true, 'b_exogenous', b,
'V_exogenous', V, 'b_constant', b_const, 'V_constant', V_const);
abr.estimate();

% get predictions on test sample, run forecast evaluation, display log score
estimates_forecasts = abr.forecast(X_test, 0.95);
abr.forecast_evaluation(y_test);
disp(['log score on test sample: ' ...
num2str(round(abr.forecast_evaluation_criteria.log_score, 2))]);

'log score on test sample : 2.12'
```

7.7 Vector autoregression: MaximumLikelihoodVar

A maximum likelihood vector autoregression, described in section 11.1.

Class:

alexandria.vector_autoregression.MaximumLikelihoodVar(endogenous, exogenous = [], structural_identification = 2, lags = 4, constant = True, trend = False, quadratic_trend = False, credibility_level = 0.95, verbose = False)

Parameters:

parameter	description
endogenous	ndarray of shape (n_obs,n): endogenous variables, defined in (4.11.1)
exogenous	ndarray of shape (n_obs,m), default = []: exogenous variables, defined in (4.11.1)
structural_identification	int, default = 2: structural identification scheme, as defined in section 13.2
lags	int, default = 4: number of lags, defined in (4.11.1)
constant	bool, default = True: if True, an intercept is included in the VAR model exogenous
trend	bool, default = False: if True, a linear trend is included in the VAR model exogenous
quadratic_trend	bool, default = False: if True, a quadratic trend is included in the VAR model exogenous
credibility_level	float, default = 0.95; VAR model credibility level (between 0 and 1)
verbose	bool, default = False: if True, displays a progress bar

Attributes:

attribute	description
endogenous	ndarray of shape (n_obs,n): endogenous variables, defined in (4.11.1)
exogenous	ndarray of shape (n_obs,m), default = []: exogenous variables, defined in (4.11.1)
structural_identification	int, default = 2: structural identification scheme, as defined in section 13.2 1 = none; 2 = Cholesky; 3 = triangular
lags	int, default = 4: number of lags, defined in (4.11.1)
constant	bool, default = True: if True, an intercept is included in the VAR model exogenous
trend	bool, default = False: if True, a linear trend is included in the VAR model exogenous
quadratic_trend	bool, default = False: if True, a quadratic trend is included in the VAR model exogenous
credibility_level	float, default = 0.95; VAR model credibility level (between 0 and 1)
verbose	bool, default = False: if True, displays a progress bar
B	ndarray of size (k,n): matrix of VAR coefficients, defined in (4.11.2)
Sigma	ndarray of size (n,n): variance-covariance ndarray of VAR residuals, defined in (4.11.1)
beta_estimates	ndarray of size (k,n,4): estimates of VAR coefficients page 1: median, page 2: st dev; page 3: lower bound, page 4: upper bound
Sigma_estimates	ndarray of size (n,n): estimates of variance-covariance ndarray of VAR residuals
H	ndarray of size (n,n): structural identification ndarray, defined in (4.13.5)
Gamma	ndarray of size (n,) : diagonal of structural shock variance ndarray, defined in section 13.2
Gamma_estimates	ndarray of size (n,): estimates of structural shock variance ndarray, defined in section 13.2
steady_state_estimates	ndarray of size (T,n,3); estimates of steady-state, defined in (4.12.30)
fitted_estimates	ndarray of size (T,n,3): estimates of in-sample fit, defined in (4.11.2)
residual_estimates	ndarray of size (T,n,3): estimates of in-sample residuals, defined in (4.11.2)
structural_shocks_ estimates	ndarray of size (T,n,3): estimates of in-sample structural shocks, defined in definition 13.1
insample_evaluation	dict: in-sample evaluation criteria, defined in (4.13.15)-(4.13.17)
forecast_estimates	ndarray of size (f_periods,n,3): forecast estimates, defined in (4.13.12) and (4.13.13) page 1: median, page 2: lower bound, page 3: upper bound

Attributes:

attribute	description
forecast_evaluation_ criteria	dict: forecast evaluation criteria, defined in (4.13.18)-(4.13.19)
irf_estimates	ndarray of size (n,n,irf_periods,3): estimates of impulse response function, defined in (4.13.1) or (4.13.9) page 1: median, page 2: lower bound, page 3: upper bound
exo_irf_estimates	ndarray of size (n,m,irf_periods,3): estimates of exogenous impulse response function, if any exogenous variable page 1: median, page 2: lower bound, page 3: upper bound
fevd_estimates	ndarray of size (n,n,fevd_periods,3): estimates of forecast error variance decomposition, defined in (4.13.31) page 1: median, page 2: lower bound, page 3: upper bound
hd_estimates	ndarray of size (n,n,T,3): estimates of historical decomposition, defined in (4.13.35) page 1: median, page 2: lower bound, page 3: upper bound
Y	ndarray of size (T,n): ndarray of in-sample endogenous variables, defined in (4.11.3)
Z	ndarray of size (T,m): ndarray of in-sample endogenous variables, defined in (4.11.3)
X	ndarray of size (T,k): ndarray of exogenous and lagged regressors, defined in (4.11.3)
n	int: number of endogenous variables, defined in (4.11.1)
m	int: number of exogenous variables, defined in (4.11.1)
p	int: number of lags, defined in (4.11.1)
T	int: number of sample periods, defined in (4.11.1)
k	int: number of VAR coefficients in each equation, defined in (4.11.1)
q	int: total number of VAR coefficients, defined in (4.11.1)

Methods:

■ **estimate()**
trains the model and produces posterior estimates for both reduced-form VAR and SVAR parameters
parameters:
- none
returns:
- none

■ **insample_fit()**
generates in-sample fit, residuals, structural shocks and steady-state along with evaluation criteria.
parameters:
- none
returns:
- none

■ **forecast(h, credibility_level, Z_p=[])**
predictions for the vector autoregression model, using (4.13.12).
parameters:
- h: int, number of forecast periods
- credibility_level: float, credibility level for predictions (between 0 and 1)
- Z_p: empty list or ndarray of shape (h,n_exo), matrix of exogenous predictors (except constant and trends)
returns:
- forecasts_estimates: ndarray of shape (h, n, 3), posterior estimates for predictions
page 1: median; page 2: credibility interval lower bound;
page 3: credibility interval upper bound

■ **forecast_evaluation(Y)**
forecast evaluation criteria for the vector autoregression model.
parameters:
- Y: ndarray of shape (h,n), array of realised values for forecast evaluation
returns:
- none

■ **impulse_response_function(h, credibility_level)**
impulse response function for the vector autoregression model, using (4.13.9).
parameters:
- h: int, number of IRF periods
- credibility_level: float, credibility level for predictions (between 0 and 1)
returns:
- irf_estimates: ndarray of shape (n, n, h, 3), posterior estimates for impulse response function
first 3 dimensions are variable, shock, period;
4th dimension is median, lower bound, upper bound

■ **forecast_error_variance_decomposition(h, credibility_level)**
forecast error variance decomposition for the vector autoregression model, using (4.13.31).
parameters:
- h: int, number of FEVD periods
- credibility_level: float, credibility level for predictions (between 0 and 1)
returns:
- fevd_estimates: ndarray of shape (n, n, h, 3), posterior estimates for impulse response function
first 3 dimensions are variable, shock, period;
4th dimension is median, lower bound, upper bound

■ **historical_decomposition(credibility_level)**
historical decomposition for the vector autoregression model, using (4.13.36).
parameters:
- credibility_level: float, credibility level for predictions (between 0 and 1)
returns:
- hd_estimates: ndarray of shape (n, n, T, 3), posterior estimates for impulse response function
first 3 dimensions are variable, shock, period;
4th dimension is median, lower bound, upper bound

Example (Python):

```
# imports
from alexandria import MaximumLikelihoodVar
from alexandria import DataSets
from alexandria import Results
from alexandria import Graphics

# load IS-LM dataset, keep only first four columns
ds = DataSets()
islm_data = ds.load_islm()[:,:4]

# create and train Bayesian VAR with Cholesky as structural identification
var = MaximumLikelihoodVar(endogenous = islm_data, structural_identification = 2)
var.estimate()

# display console outputs for the model
res = Results(var)
res.make_estimation_summary()
res.show_estimation_summary()

# estimate impulse response function for 30 periods, 60% credibility level
irf_estimates = var.impulse_response_function(30, 0.6)

# create graphics of predictions and display them
gp = Graphics(var)
gp.irf_graphics(show=True, save=False)
```

Example (Matlab):

```
% load IS-LM dataset, keep only first four columns
ds = DataSets();
islm_data = ds.load_islm();
islm_data = islm_data(:,1:4);

% create and train Bayesian VAR with Cholesky as structural identification
var = MaximumLikelihoodVar(islm_data, 'structural_identification', 2);
var.estimate();

% display console outputs for the model
res = Results(var);
res.make_estimation_summary();
res.show_estimation_summary();

% estimate impulse response function for 30 periods, 60% credibility level
irf_estimates = var.impulse_response_function(30, 0.6);

% create graphics of predictions and display them
gp = Graphics(var);
gp.irf_graphics(true, false);
```

7.8 Vector autoregression: MinnesotaBayesianVar

A vector autoregression using the Minnesota prior, described in section 11.2.

Class:

alexandria.vector_autoregression.MinnesotaBayesianVar(endogenous, exogenous = [], structural_identification = 2, restriction_table = [], lags = 4, constant = True, trend = False, quadratic_trend = False, ar_coefficients = 0.95, pi1 = 0.1, pi2 = 0.5, pi3 = 1, pi4 = 100, pi5 = 1, pi6 = 0.1, pi7 = 0.1, constrained_coefficients = False, constrained_coefficients_table = [], sums_of_coefficients = False, dummy_initial_observation = False, long_run_prior = False, long_run_table = [], hyperparameter_optimization = False, stationary_prior = False, credibility_level = 0.95, iterations = 2000, verbose = False)

Parameters:

parameter	description
endogenous	ndarray of shape (n_obs,n): endogenous variables, defined in (4.11.1)
exogenous	ndarray of shape (n_obs,m), default = []: exogenous variables, defined in (4.11.1)
structural_identification	int, default = 2: structural identification scheme, as defined in section 13.2
restriction_table	ndarray, default = []: numerical matrix of restrictions for structural identification
lags	int, default = 4: number of lags, defined in (4.11.1)
constant	bool, default = True: if True, an intercept is included in the VAR model exogenous
trend	bool, default = False: if True, a linear trend is included in the VAR model exogenous
quadratic_trend	bool, default = False: if True, a quadratic trend is included in the VAR model exogenous
ar_coefficients	float or ndarray of size (n_endo,1), default = 0.95: prior mean delta for AR coefficients, defined in (4.11.16)
pi1	float, default = 0.1: overall tightness hyperparameter, defined in (4.11.17)
pi2	float, default = 0.5: cross-variable shrinkage hyperparameter, defined in (4.11.18)
pi3	float, default = 1: lag decay hyperparameter, defined in (4.11.17)
pi4	float, default = 100: exogenous slackness hyperparameter, defined in (4.11.19)
pi5	float, default = 1: sums-of-coefficients hyperparameter, defined in (4.12.6)
pi6	float, default = 0.1: initial observation hyperparameter, defined in (4.12.10)
pi7	float, default = 0.1: long-run hyperparameter, defined in (4.12.16)
constrained_coefficients	bool, default = False: if True, applies constrained coefficients, as defined in section 12.1
constrained_coefficients _table	ndarray, default = []: numerical matrix of constrained coefficients
sums_of_coefficients	bool, default = False: if True, applies sums-of-coefficients, as defined in section 12.2
dummy_initial _observation	bool, default = False: if True, applies dummy initial observation, as defined in section 12.2
long_run_prior	bool, default = False: if True, applies long-run prior, as defined in section 12.2
long_run_table	ndarray, default = []: numerical matrix of long-run prior
hyperparameter _optimization	bool, default = False: if True, applies hyperparameter optimization by marginal likelihood
stationary_prior	bool, default = False: if True, applies stationary prior, as defined in section 12.4
iterations	int, default = 2000: number of Gibbs sampler replications
credibility_level	float, default = 0.95; VAR model credibility level (between 0 and 1)
verbose	bool, default = False: if True, displays a progress bar

Attributes:

attribute	description
endogenous	ndarray of shape (n_obs,n): endogenous variables, defined in (4.11.1)
exogenous	ndarray of shape (n_obs,m), default = []: exogenous variables, defined in (4.11.1)
structural_identification	int, default = 2: structural identification scheme, as defined in section 13.2
restriction_table	ndarray, default = []: numerical matrix of restrictions for structural identification
lags	int, default = 4: number of lags, defined in (4.11.1)
constant	bool, default = True: if True, an intercept is included in the VAR model exogenous
trend	bool, default = False: if True, a linear trend is included in the VAR model exogenous
quadratic_trend	bool, default = False: if True, a quadratic trend is included in the VAR model exogenous
ar_coefficients	float or ndarray of size (n_endo,1), default = 0.95: prior mean delta for AR coefficients, defined in (4.11.16)
pi1	float, default = 0.1: overall tightness hyperparameter, defined in (4.11.17)
pi2	float, default = 0.5: cross-variable shrinkage hyperparameter, defined in (4.11.18)
pi3	float, default = 1: lag decay hyperparameter, defined in (4.11.17)
pi4	float, default = 100: exogenous slackness hyperparameter, defined in (4.11.19)
pi5	float, default = 1: sums-of-coefficients hyperparameter, defined in (4.12.6)
pi6	float, default = 0.1: initial observation hyperparameter, defined in (4.12.10)
pi7	float, default = 0.1: long-run hyperparameter, defined in (4.12.16)
constrained_coefficients	bool, default = False: if True, applies constrained coefficients, as defined in section 12.1
constrained_coefficients _table	ndarray, default = []: numerical matrix of constrained coefficients
sums_of_coefficients	bool, default = False: if True, applies sums-of-coefficients, as defined in section 12.2
dummy_initial _observation	bool, default = False: if True, applies dummy initial observation, as defined in section 12.2
long_run_prior	bool, default = False: if True, applies long-run prior, as defined in section 12.2
J	ndarray of size (n,n); matrix of long-run prior coefficients, defined in (4.12.15)
hyperparameter _optimization	bool, default = False: if True, applies hyperparameter optimization by marginal likelihood
stationary_prior	bool, default = False: if True, applies stationary prior, as defined in section 12.4
iterations	int, default = 2000: number of Gibbs sampler replications
credibility_level	float, default = 0.95; VAR model credibility level (between 0 and 1)
verbose	bool, default = False: if True, displays a progress bar
Sigma	ndarray of size (n,n): variance-covariance matrix of VAR residuals, defined in (4.11.1)
b_bar	ndarray of size (q,1): posterior mean of VAR coefficients, defined in (4.11.15)
V_bar	ndarray of size (q,q): posterior mean of VAR coefficients, defined in (4.11.15)
beta_estimates	ndarray of size (k,n,3): estimates of VAR coefficients page 1: median, page 2: st dev, page 3: lower bound, page 4: upper bound
Sigma_estimates	ndarray of size (n,n): estimates of variance-covariance matrix of VAR residuals
mcmc_beta	ndarray of size (k,n,iterations): MCMC values of VAR coefficients
mcmc_Sigma	ndarray of size (n,n,iterations): MCMC values of residual variance-covariance matrix
m_y	float: log 10 marginal likelihood, defined in (4.12.21)
b	ndarray of size (q,1): prior mean of VAR coefficients, defined in (4.11.16)
V	ndarray of size (q,q): prior mean of VAR coefficients, defined in (4.11.20)
mcmc_H	ndarray of size (n,n,iterations): MCMC values of structural identification matrix, defined in (4.13.5)
mcmc_Gamma	ndarray of size (iterations,n): MCMC values of structural shock variance matrix, defined in definition 13.1
Y	ndarray of size (T,n): matrix of in-sample endogenous variables, defined in (4.11.3)
Z	ndarray of size (T,m): matrix of in-sample endogenous variables, defined in (4.11.3)
X	ndarray of size (T,k): matrix of exogenous and lagged regressors, defined in (4.11.3)
n	int: number of endogenous variables, defined in (4.11.1)
m	int: number of exogenous variables, defined in (4.11.1)
p	int: number of lags, defined in (4.11.1)
T	int: number of sample periods, defined in (4.11.1)

Attributes:

attribute	description
k	int: number of VAR coefficients in each equation, defined in (4.11.1)
q	int: total number of VAR coefficients, defined in (4.11.1)
delta	ndarray of size (n,1): prior mean delta for AR coefficients, defined in (4.11.16)
s	ndarray of size (n,1): individual AR models residual variance, defined in (4.11.18)
Y_sum	ndarray of size (n,n): sums-of-coefficients Y matrix, defined in (4.12.6)
X_sum	ndarray of size (n,k): sums-of-coefficients X matrix, defined in (4.12.6)
Y_obs	ndarray of size (1,n): dummy initial observation Y matrix, defined in (4.12.10)
X_obs	ndarray of size (1,k): dummy initial observation X matrix, defined in (4.12.10)
Y_lrp	ndarray of size (1,n): long run prior Y matrix, defined in (4.12.16)
X_lrp	ndarray of size (1,k): long run prior X matrix, defined in (4.12.16)
Y_d	ndarray of size (T_d,n): full Y matrix combining sample data and dummy observations, defined in (4.11.62)
X_d	ndarray of size (T_d,k): full X matrix combining sample data and dummy observations, defined in (4.11.62)
T_d	int: total number of observations combining sample data and dummy observations, defined in (4.11.62)
steady_state_estimates	ndarray of size (T,n,3): estimates of steady-state, defined in (4.12.30)
fitted_estimates	ndarray of size (T,n,3): estimates of in-sample fit, defined in (4.11.2)
residual_estimates	ndarray of size (T,n,3): estimates of in-sample residuals, defined in (4.11.2)
structural_shocks _estimates	ndarray of size (T,n,3): estimates of in-sample structural shocks, defined in definition 13.1
insample_evaluation	dict: in-sample evaluation criteria, defined in (4.13.15)-(4.13.17)
mcmc_structural_shocks	ndarray of size (T,n,iterations): MCMC values of structural shocks
mcmc_forecasts	ndarray of size (f_periods,n,iterations): MCMC values of forecasts
forecast_estimates	ndarray of size (f_periods,n,3): forecast estimates, defined in (4.13.12) and (4.13.13) page 1: median, page 2: lower bound, page 3: upper bound
forecast_evaluation _criteria	dict: forecast evaluation criteria, defined in (4.13.18)-(4.13.21)
mcmc_irf	ndarray of size (n,n,irf_periods,iterations): MCMC values of impulse response function, defined in section 13.1
mcmc_irf_exo	ndarray of size (n,m,irf_periods,iterations): MCMC values of exogenous impulse response function
mcmc_structural_irf	ndarray of size (n,n,irf_periods,iterations): MCMC values of structural impulse response function, defined in section 13.2
irf_estimates	ndarray of size (n,n,irf_periods,3): posterior estimates of impulse response function, defined in section 13.1 - 13.2 page 1: median, page 2: lower bound, page 3: upper bound
exo_irf_estimates	ndarray of size (n,m,irf_periods,3): posterior estimates of exogenous impulse response function, if any exogenous variable page 1: median, page 2: lower bound, page 3: upper bound
mcmc_fevd	ndarray of size (n,n,fevd_periods,iterations): MCMC values of forecast error variance decompositions, defined in section 13.4
fevd_estimates	ndarray of size (n,n,fevd_periods,3): posterior estimates of forecast error variance decomposition, defined in section 13.4 page 1: median, page 2: lower bound, page 3: upper bound
mcmc_hd	ndarray of size (n,n,T,iterations): MCMC values of historical decompositions, defined in section 13.5
hd_estimates	ndarray of size (n,n,T,3): posterior estimates of historical decomposition, defined in section 13.5 page 1: median, page 2: lower bound, page 3: upper bound
mcmc_conditional _forecasts	ndarray of size (f_periods,n,iterations): MCMC values of conditional forecasts, defined in section 14.1
conditional_forecast _estimates	ndarray of size (f_periods,n,3): posterior estimates of conditional forecast, defined in section 14.1 page 1: median, page 2: lower bound, page 3: upper bound
H_estimates	ndarray of size (n,n): posterior estimates of structural matrix, defined in section 13.2
Gamma_estimates	ndarray of size (1,n): estimates of structural shock variance matrix, defined in section 13.2

Methods:

■ **estimate()**
trains the model and produces posterior estimates for both reduced-form VAR and SVAR parameters
parameters:
- none
returns:
- none

■ **marginal_likelihood()**
log10 marginal likelihood, defined in (4.12.21).
parameters:
- none
returns:
- m_y: float, log 10 marginal likelihood value.

■ **insample_fit()**
generates in-sample fit, residuals, structural shocks and steady-state along with evaluation criteria.
parameters:
- none
returns:
- none

■ **forecast(h, credibility_level, Z_p=[])**
predictions for the vector autoregression model, using (4.13.12).
parameters:
- h: int, number of forecast periods
- credibility_level: float, credibility level for predictions (between 0 and 1)
- Z_p: empty list or ndarray of shape (h,n_exo), matrix of exogenous predictors (except constant and trends)
returns:
- forecasts_estimates: ndarray of shape (h, n, 3), posterior estimates for predictions
page 1: median; page 2: credibility interval lower bound;
page 3: credibility interval upper bound

■ **forecast_evaluation(Y)**
forecast evaluation criteria for the vector autoregression model.
parameters:
- Y: ndarray of shape (h,n), array of realised values for forecast evaluation
returns:
- none

■ **impulse_response_function(h, credibility_level)**
impulse response function for the vector autoregression model, using (4.13.9).
parameters:
- h: int, number of IRF periods
- credibility_level: float, credibility level for predictions (between 0 and 1)
returns:
- irf_estimates: ndarray of shape (n, n, h, 3), posterior estimates for impulse response function
first 3 dimensions are variable, shock, period;
4th dimension is median, lower bound, upper bound

■ **forecast_error_variance_decomposition(h, credibility_level)**
forecast error variance decomposition for the vector autoregression model, using (4.13.31).
parameters:
- h: int, number of FEVD periods
- credibility_level: float, credibility level for predictions (between 0 and 1)
returns:
- fevd_estimates: ndarray of shape (n, n, h, 3), posterior estimates for impulse response function
first 3 dimensions are variable, shock, period;
4th dimension is median, lower bound, upper bound

■ **historical_decomposition(credibility_level)**
historical decomposition for the vector autoregression model, using (4.13.36).
parameters:
- credibility_level: float, credibility level for predictions (between 0 and 1)
returns:
- hd_estimates: ndarray of shape (n, n, T, 3), posterior estimates for impulse response function
first 3 dimensions are variable, shock, period;
4th dimension is median, lower bound, upper bound

■ **conditional_forecast(h, credibility_level, conditions, shocks, conditional_forecast_type, Z_p=[])**
conditional forecasts for the Bayesian VAR model, using algorithms 14.1 and 14.2.
parameters:
- h: int, number of forecast periods
- credibility_level: float, credibility level for predictions (between 0 and 1)
- conditions: ndarray of shape (n_conditions,4)
- shocks: empty list or ndarray of shape (n,), vector defining shocks generating the conditions; should be empty if conditional_forecast_type = 1
- conditional_forecast_type : int, conditional forecast type (1 = agnostic, 2 = structural)
- Z_p: empty list or ndarray of shape (h,n_exo), matrix of exogenous predictors (except constant and trends)
returns:
- conditional_forecasts_estimates: ndarray of shape (h, n, 3), posterior estimates for predictions
page 1: median; page 2: credibility interval lower bound;
page 3: credibility interval upper bound

Example (Python):

```
# imports
from alexandria import MinnesotaBayesianVar
from alexandria import DataSets
from alexandria import Results
from alexandria import Graphics
import numpy as np

# load IS-LM dataset, keep only first four columns
ds = DataSets()
islm_data = ds.load_islm()[:,:4]

# create and train Bayesian VAR with Cholesky as structural identification
var = MinnesotaBayesianVar(islm_data, structural_identification = 2)
var.estimate()

# display console outputs for the model
res = Results(var)
res.make_estimation_summary()
res.show_estimation_summary()

# estimate conditional forecasts: short-term rate is 2.7% over next 2 periods
conditions = np.array([[3,1,2.7,1e-10],[3,2,2.7,1e-10]])
forecasts = var.conditional_forecast(4, 0.6, conditions, [], 1, Z_p = [])

# create graphics of predictions and display them
gp = Graphics(var)
gp.conditional_forecast_graphics(show=True, save=False)
```

Example (Matlab):

```
% load IS-LM dataset, keep only first four columns
ds = DataSets();
islm_data = ds.load_islm();
islm_data = islm_data(:,1:4);

% create and train Bayesian VAR with Cholesky as structural identification
var = MinnesotaBayesianVar(islm_data, 'structural_identification', 2);
var.estimate();

% display console outputs for the model
res = Results(var);
res.make_estimation_summary();
res.show_estimation_summary();

% estimate conditional forecasts: short-term rate is 2.7% over next 2 periods
conditions = [3 1 2.7 1e-10;3 2 2.7 1e-10];
forecasts = var.conditional_forecast(4, 0.6, conditions, [], 1, Z_p = []);

% create graphics of predictions and display them
gp = Graphics(var);
gp.conditional_forecast_graphics(true, false);
```

7.9 Vector autoregression: NormalWishartBayesianVar

A vector autoregression using the normal-Wishart prior, described in section 11.3.

Class:

alexandria.vector_autoregression.NormalWishartBayesianVar(endogenous, exogenous = [], structural_identification = 2, restriction_table = [], lags = 4, constant = True, trend = False, quadratic_trend = False, ar_coefficients = 0.95, pi1 = 0.1, pi3 = 1, pi4 = 100, pi5 = 1, pi6 = 0.1, pi7 = 0.1, sums_of_coefficients = False, dummy_initial_observation = False, long_run_prior = False, long_run_table = [], hyperparameter_optimization = False, stationary_prior = False, credibility_level = 0.95, iterations = 2000, verbose = False)

Parameters:

parameter	description
endogenous	ndarray of shape (n_obs,n): endogenous variables, defined in (4.11.1)
exogenous	ndarray of shape (n_obs,m), default = []: exogenous variables, defined in (4.11.1)
structural_identification	int, default = 2: structural identification scheme, as defined in section 13.2
restriction_table	ndarray, default = []: numerical matrix of restrictions for structural identification
lags	int, default = 4: number of lags, defined in (4.11.1)
constant	bool, default = True: if True, an intercept is included in the VAR model exogenous
trend	bool, default = False: if True, a linear trend is included in the VAR model exogenous
quadratic_trend	bool, default = False: if True, a quadratic trend is included in the VAR model exogenous
ar_coefficients	float or ndarray of size (n_endo,1), default = 0.95: prior mean delta for AR coefficients, defined in (4.11.16)
pi1	float, default = 0.1: overall tightness hyperparameter, defined in (4.11.17)
pi3	float, default = 1: lag decay hyperparameter, defined in (4.11.17)
pi4	float, default = 100: exogenous slackness hyperparameter, defined in (4.11.19)
pi5	float, default = 1: sums-of-coefficients hyperparameter, defined in (4.12.6)
pi6	float, default = 0.1: initial observation hyperparameter, defined in (4.12.10)
pi7	float, default = 0.1: long-run hyperparameter, defined in (4.12.16)
sums_of_coefficients	bool, default = False: if True, applies sums-of-coefficients, as defined in section 12.2
dummy_initial _observation	bool, default = False: if True, applies dummy initial observation, as defined in section 12.2
long_run_prior	bool, default = False: if True, applies long-run prior, as defined in section 12.2
long_run_table	ndarray, default = []: numerical matrix of long-run prior
hyperparameter _optimization	bool, default = False: if True, applies hyperparameter optimization by marginal likelihood
stationary_prior	bool, default = False: if True, applies stationary prior, as defined in section 12.4
credibility_level	float, default = 0.95; VAR model credibility level (between 0 and 1)
iterations	int, default = 2000: number of Gibbs sampler replications
verbose	bool, default = False: if True, displays a progress bar

Attributes:

attribute	description
endogenous	ndarray of shape (n_obs,n): endogenous variables, defined in (4.11.1)
exogenous	ndarray of shape (n_obs,m), default = []: exogenous variables, defined in (4.11.1)
structural_identification	int, default = 2: structural identification scheme, as defined in section 13.2
restriction_table	ndarray, default = []: numerical matrix of restrictions for structural identification
lags	int, default = 4: number of lags, defined in (4.11.1)
constant	bool, default = True: if True, an intercept is included in the VAR model exogenous
trend	bool, default = False: if True, a linear trend is included in the VAR model exogenous
quadratic_trend	bool, default = False: if True, a quadratic trend is included in the VAR model exogenous
ar_coefficients	float or ndarray of size (n_endo,1), default = 0.95: prior mean delta for AR coefficients, defined in (4.11.16)
pi1	float, default = 0.1: overall tightness hyperparameter, defined in (4.11.17)
pi3	float, default = 1: lag decay hyperparameter, defined in (4.11.17)
pi4	float, default = 100: exogenous slackness hyperparameter, defined in (4.11.19)
pi5	float, default = 1: sums-of-coefficients hyperparameter, defined in (4.12.6)
pi6	float, default = 0.1: initial observation hyperparameter, defined in (4.12.10)
pi7	float, default = 0.1: long-run hyperparameter, defined in (4.12.16)
sums_of_coefficients	bool, default = False: if True, applies sums-of-coefficients, as defined in section 12.2
dummy_initial _observation	bool, default = False: if True, applies dummy initial observation, as defined in section 12.2
long_run_prior	bool, default = False: if True, applies long-run prior, as defined in section 12.2
J	ndarray of size (n,n); matrix of long-run prior coefficients, defined in (4.12.15)
hyperparameter _optimization	bool, default = False: if True, applies hyperparameter optimization by marginal likelihood
stationary_prior	bool, default = False: if True, applies stationary prior, as defined in section 12.4
iterations	int, default = 2000: number of Gibbs sampler replications
credibility_level	float, default = 0.95; VAR model credibility level (between 0 and 1)
verbose	bool, default = False: if True, displays a progress bar
B	ndarray of size (k,n): prior mean of VAR coefficients, defined in (4.11.24)
W	ndarray of size (k,k): prior mean of VAR coefficients, defined in (4.11.20)
alpha	float: prior degrees of freedom, defined in (4.11.29)
S	ndarray of size (n,n): prior scale matrix, defined in (4.11.29)
B_bar	ndarray of size (k,n): posterior mean of VAR coefficients, defined in (4.11.33)
W_bar	ndarray of size (k,k): posterior mean of VAR coefficients, defined in (4.11.33)
alpha_bar	float: posterior degrees of freedom, defined in (4.11.33)
S_bar	ndarray of size (n,n): posterior scale matrix, defined in (4.11.33)
alpha_hat	float: posterior degrees of freedom, defined in (4.11.38)
S_hat	ndarray of size (n,n): posterior scale matrix, defined in (4.11.38)
beta_estimates	ndarray of size (k,n,3): estimates of VAR coefficients page 1: median, page 2: st dev, page 3: lower bound, page 4: upper bound
Sigma_estimates	ndarray of size (n,n): estimates of variance-covariance matrix of VAR residuals
mcmc_beta	ndarray of size (k,n,iterations): MCMC values of VAR coefficients
mcmc_Sigma	ndarray of size (n,n,iterations): MCMC values of residual variance-covariance matrix
m_y	float: log 10 marginal likelihood, defined in (4.12.21)
mcmc_H	ndarray of size (n,n,iterations): MCMC values of structural identification matrix, defined in (4.13.5)
mcmc_Gamma	ndarray of size (iterations,n): MCMC values of structural shock variance matrix, defined in definition 13.1
Y	ndarray of size (T,n): matrix of in-sample endogenous variables, defined in (4.11.3)
Z	ndarray of size (T,m): matrix of in-sample endogenous variables, defined in (4.11.3)
X	ndarray of size (T,k): matrix of exogenous and lagged regressors, defined in (4.11.3)
n	int: number of endogenous variables, defined in (4.11.1)
m	int: number of exogenous variables, defined in (4.11.1)
p	int: number of lags, defined in (4.11.1)
T	int: number of sample periods, defined in (4.11.1)

Attributes:

attribute	description
k	int: number of VAR coefficients in each equation, defined in (4.11.1)
q	int: total number of VAR coefficients, defined in (4.11.1)
delta	ndarray of size (n,1): prior mean delta for AR coefficients, defined in (4.11.16)
s	ndarray of size (n,1): individual AR models residual variance, defined in (4.11.18)
Y_sum	ndarray of size (n,n): sums-of-coefficients Y matrix, defined in (4.12.6)
X_sum	ndarray of size (n,k): sums-of-coefficients X matrix, defined in (4.12.6)
Y_obs	ndarray of size (1,n): dummy initial observation Y matrix, defined in (4.12.10)
X_obs	ndarray of size (1,k): dummy initial observation X matrix, defined in (4.12.10)
Y_lrp	ndarray of size (1,n): long run prior Y matrix, defined in (4.12.16)
X_lrp	ndarray of size (1,k): long run prior X matrix, defined in (4.12.16)
Y_d	ndarray of size (T_d,n): full Y matrix combining sample data and dummy observations, defined in (4.11.62)
X_d	ndarray of size (T_d,k): full X matrix combining sample data and dummy observations, defined in (4.11.62)
T_d	int: total number of observations combining sample data and dummy observations, defined in (4.11.62)
steady_state_estimates	ndarray of size (T,n,3): estimates of steady-state, defined in (4.12.30)
fitted_estimates	ndarray of size (T,n,3): estimates of in-sample fit, defined in (4.11.2)
residual_estimates	ndarray of size (T,n,3): estimates of in-sample residuals, defined in (4.11.2)
structural_shocks _estimates	ndarray of size (T,n,3): estimates of in-sample structural shocks, defined in definition 13.1
insample_evaluation	dict: in-sample evaluation criteria, defined in (4.13.15)-(4.13.17)
mcmc_structural_shocks	ndarray of size (T,n,iterations): MCMC values of structural shocks
mcmc_forecasts	ndarray of size (f_periods,n,iterations): MCMC values of forecasts
forecast_estimates	ndarray of size (f_periods,n,3): forecast estimates, defined in (4.13.12) and (4.13.13) page 1: median, page 2: lower bound, page 3: upper bound
forecast_evaluation _criteria	dict: forecast evaluation criteria, defined in (4.13.18)-(4.13.21)
mcmc_irf	ndarray of size (n,n,irf_periods,iterations): MCMC values of impulse response function, defined in section 13.1
mcmc_irf_exo	ndarray of size (n,m,irf_periods,iterations): MCMC values of exogenous impulse response function
mcmc_structural_irf	ndarray of size (n,n,irf_periods,iterations): MCMC values of structural impulse response function, defined in section 13.2
irf_estimates	ndarray of size (n,n,irf_periods,3): posterior estimates of impulse response function, defined in section 13.1 - 13.2 page 1: median, page 2: lower bound, page 3: upper bound
exo_irf_estimates	ndarray of size (n,m,irf_periods,3): posterior estimates of exogenous impulse response function, if any exogenous variable page 1: median, page 2: lower bound, page 3: upper bound
mcmc_fevd	ndarray of size (n,n,fevd_periods,iterations): MCMC values of forecast error variance decompositions, defined in section 13.4
fevd_estimates	ndarray of size (n,n,fevd_periods,3): posterior estimates of forecast error variance decomposition, defined in section 13.4 page 1: median, page 2: lower bound, page 3: upper bound
mcmc_hd	ndarray of size (n,n,T,iterations): MCMC values of historical decompositions, defined in section 13.5
hd_estimates	ndarray of size (n,n,T,3): posterior estimates of historical decomposition, defined in section 13.5 page 1: median, page 2: lower bound, page 3: upper bound
mcmc_conditional _forecasts	ndarray of size (f_periods,n,iterations): MCMC values of conditional forecasts, defined in section 14.1
conditional_forecast _estimates	ndarray of size (f_periods,n,3): posterior estimates of conditional forecast, defined in section 14.1 page 1: median, page 2: lower bound, page 3: upper bound
H_estimates	ndarray of size (n,n): posterior estimates of structural matrix, defined in section 13.2
Gamma_estimates	ndarray of size (1,n): estimates of structural shock variance matrix, defined in section 13.2

Methods:

■ **estimate()**
trains the model and produces posterior estimates for both reduced-form VAR and SVAR parameters
parameters:
- none
returns:
- none

■ **marginal_likelihood()**
log10 marginal likelihood, defined in (4.12.24).
parameters:
- none
returns:
- m_y: float, log 10 marginal likelihood value.

■ **insample_fit()**
generates in-sample fit, residuals, structural shocks and steady-state along with evaluation criteria.
parameters:
- none
returns:
- none

■ **forecast(h, credibility_level, Z_p=[])**
predictions for the vector autoregression model, using (4.13.12).
parameters:
- h: int, number of forecast periods
- credibility_level: float, credibility level for predictions (between 0 and 1)
- Z_p: empty list or ndarray of shape (h,n_exo), matrix of exogenous predictors (except constant and trends)
returns:
- forecasts_estimates: ndarray of shape (h, n, 3), posterior estimates for predictions
page 1: median; page 2: credibility interval lower bound;
page 3: credibility interval upper bound

■ **forecast_evaluation(Y)**
forecast evaluation criteria for the vector autoregression model.
parameters:
- Y: ndarray of shape (h,n), array of realised values for forecast evaluation
returns:
- none

■ **impulse_response_function(h, credibility_level)**
impulse response function for the vector autoregression model, using (4.13.9).
parameters:
- h: int, number of IRF periods
- credibility_level: float, credibility level for predictions (between 0 and 1)
returns:
- irf_estimates: ndarray of shape (n, n, h, 3), posterior estimates for impulse response function
first 3 dimensions are variable, shock, period;
4th dimension is median, lower bound, upper bound

■ **forecast_error_variance_decomposition(h, credibility_level)**
forecast error variance decomposition for the vector autoregression model, using (4.13.31).
parameters:
- h: int, number of FEVD periods
- credibility_level: float, credibility level for predictions (between 0 and 1)
returns:
- fevd_estimates: ndarray of shape (n, n, h, 3), posterior estimates for impulse response function
first 3 dimensions are variable, shock, period;
4th dimension is median, lower bound, upper bound

■ **historical_decomposition(credibility_level)**
historical decomposition for the vector autoregression model, using (4.13.36).
parameters:
- credibility_level: float, credibility level for predictions (between 0 and 1)
returns:
- hd_estimates: ndarray of shape (n, n, T, 3), posterior estimates for impulse response function
first 3 dimensions are variable, shock, period;
4th dimension is median, lower bound, upper bound

■ **conditional_forecast(h, credibility_level, conditions, shocks, conditional_forecast_type, Z_p=[])**
conditional forecasts for the Bayesian VAR model, using algorithms 14.1 and 14.2.
parameters:
- h: int, number of forecast periods
- credibility_level: float, credibility level for predictions (between 0 and 1)
- conditions: ndarray of shape (n_conditions,4)
- shocks: empty list or ndarray of shape (n,), vector defining shocks generating the conditions; should be empty if conditional_forecast_type = 1
- conditional_forecast_type : int, conditional forecast type (1 = agnostic, 2 = structural)
- Z_p: empty list or ndarray of shape (h,n_exo), matrix of exogenous predictors (except constant and trends)
returns:
- conditional_forecasts_estimates: ndarray of shape (h, n, 3), posterior estimates for predictions
page 1: median; page 2: credibility interval lower bound;
page 3: credibility interval upper bound

Example (Python):

```
# imports
from alexandria import NormalWishartBayesianVar
from alexandria import DataSets
from alexandria import Results
from alexandria import Graphics
import numpy as np

# load IS-LM dataset, keep only first four columns
ds = DataSets()
islm_data = ds.load_islm()[:,:4]

# create and train Bayesian VAR with Cholesky as structural identification
var = NormalWishartBayesianVar(islm_data, structural_identification = 2)
var.estimate()

# display console outputs for the model
res = Results(var)
res.make_estimation_summary()
res.show_estimation_summary()

# estimate conditional forecasts: short-term rate is 2.7% over next 2 periods
conditions = np.array([[3,1,2.7,1e-10],[3,2,2.7,1e-10]])
forecasts = var.conditional_forecast(4, 0.6, conditions, [], 1, Z_p = [])

# create graphics of predictions and display them
gp = Graphics(var)
gp.conditional_forecast_graphics(show=True, save=False)
```

Example (Matlab):

```
% load IS-LM dataset, keep only first four columns
ds = DataSets();
islm_data = ds.load_islm();
islm_data = islm_data(:,1:4);

% create and train Bayesian VAR with Cholesky as structural identification
var = NormalWishartBayesianVar(islm_data, 'structural_identification', 2);
var.estimate();

% display console outputs for the model
res = Results(var);
res.make_estimation_summary();
res.show_estimation_summary();

% estimate conditional forecasts: short-term rate is 2.7% over next 2 periods
conditions = [3 1 2.7 1e-10;3 2 2.7 1e-10];
forecasts = var.conditional_forecast(4, 0.6, conditions, [], 1, Z_p = []);

% create graphics of predictions and display them
gp = Graphics(var);
gp.conditional_forecast_graphics(true, false);
```

7.10 Vector autoregression: IndependentBayesianVar

A vector autoregression using the independent prior, described in section 11.4.

Class:

alexandria.vector_autoregression.IndependentBayesianVar(endogenous, exogenous = [], structural_identification = 2, restriction_table = [], lags = 4, constant = True, trend = False, quadratic_trend = False, ar_coefficients = 0.95, pi1 = 0.1, pi2 = 0.5, pi3 = 1, pi4 = 100, pi5 = 1, pi6 = 0.1, pi7 = 0.1, constrained_coefficients = False, constrained_coefficients_table = [], sums_of_coefficients = False, dummy_initial_observation = False, long_run_prior = False, long_run_table = [], stationary_prior = False, credibility_level = 0.95, iterations = 2000, burnin = 1000, verbose = False)

Parameters:

parameter	description
endogenous	ndarray of shape (n_obs,n): endogenous variables, defined in (4.11.1)
exogenous	ndarray of shape (n_obs,m), default = []: exogenous variables, defined in (4.11.1)
structural_identification	int, default = 2: structural identification scheme, as defined in section 13.2
restriction_table	ndarray, default = []: numerical matrix of restrictions for structural identification
lags	int, default = 4: number of lags, defined in (4.11.1)
constant	bool, default = True: if True, an intercept is included in the VAR model exogenous
trend	bool, default = False: if True, a linear trend is included in the VAR model exogenous
quadratic_trend	bool, default = False: if True, a quadratic trend is included in the VAR model exogenous
ar_coefficients	float or ndarray of size (n_endo,1), default = 0.95: prior mean delta for AR coefficients, defined in (4.11.16)
pi1	float, default = 0.1: overall tightness hyperparameter, defined in (4.11.17)
pi2	float, default = 0.5: cross-variable shrinkage hyperparameter, defined in (4.11.18)
pi3	float, default = 1: lag decay hyperparameter, defined in (4.11.17)
pi4	float, default = 100: exogenous slackness hyperparameter, defined in (4.11.19)
pi5	float, default = 1: sums-of-coefficients hyperparameter, defined in (4.12.6)
pi6	float, default = 0.1: initial observation hyperparameter, defined in (4.12.10)
pi7	float, default = 0.1: long-run hyperparameter, defined in (4.12.16)
constrained_coefficients	bool, default = False: if True, applies constrained coefficients, as defined in section 12.1
constrained_coefficients_table	ndarray, default = []: numerical matrix of constrained coefficients
sums_of_coefficients	bool, default = False: if True, applies sums-of-coefficients, as defined in section 12.2
dummy_initial_observation	bool, default = False: if True, applies dummy initial observation, as defined in section 12.2
long_run_prior	bool, default = False: if True, applies long-run prior, as defined in section 12.2
long_run_table	ndarray, default = []: numerical matrix of long-run prior
stationary_prior	bool, default = False: if True, applies stationary prior, as defined in section 12.4
credibility_level	float, default = 0.95; VAR model credibility level (between 0 and 1)
iterations	int, default = 2000: number of Gibbs sampler replications
burnin	int, default = 1000: number of Gibbs sampler burn-in replications
verbose	bool, default = False: if True, displays a progress bar

Attributes:

attribute	description
endogenous	ndarray of shape (n_obs,n): endogenous variables, defined in (4.11.1)
exogenous	ndarray of shape (n_obs,m), default = []: exogenous variables, defined in (4.11.1)
structural_identification	int, default = 2: structural identification scheme, as defined in section 13.2
restriction_table	ndarray, default = []: numerical matrix of restrictions for structural identification
lags	int, default = 4: number of lags, defined in (4.11.1)
constant	bool, default = True: if True, an intercept is included in the VAR model exogenous
trend	bool, default = False: if True, a linear trend is included in the VAR model exogenous
quadratic_trend	bool, default = False: if True, a quadratic trend is included in the VAR model exogenous
ar_coefficients	float or ndarray of size (n_endo,1), default = 0.95: prior mean delta for AR coefficients, defined in (4.11.16)
pi1	float, default = 0.1: overall tightness hyperparameter, defined in (4.11.17)
pi2	float, default = 0.5: cross-variable shrinkage hyperparameter, defined in (4.11.18)
pi3	float, default = 1: lag decay hyperparameter, defined in (4.11.17)
pi4	float, default = 100: exogenous slackness hyperparameter, defined in (4.11.19)
pi5	float, default = 1: sums-of-coefficients hyperparameter, defined in (4.12.6)
pi6	float, default = 0.1: initial observation hyperparameter, defined in (4.12.10)
pi7	float, default = 0.1: long-run hyperparameter, defined in (4.12.16)
constrained_coefficients	bool, default = False: if True, applies constrained coefficients, as defined in section 12.1
constrained_coefficients _table	ndarray, default = []: numerical matrix of constrained coefficients
sums_of_coefficients	bool, default = False: if True, applies sums-of-coefficients, as defined in section 12.2
dummy_initial _observation	bool, default = False: if True, applies dummy initial observation, as defined in section 12.2
long_run_prior	bool, default = False: if True, applies long-run prior, as defined in section 12.2
J	ndarray of size (n,n); matrix of long-run prior coefficients, defined in (4.12.15)
stationary_prior	bool, default = False: if True, applies stationary prior, as defined in section 12.4
credibility_level	float, default = 0.95: VAR model credibility level (between 0 and 1)
iterations	int, default = 2000: number of Gibbs sampler replications
burnin	int: number of Gibbs sampler burn-in replications
verbose	bool, default = False: if True, displays a progress bar
alpha	float: prior degrees of freedom, defined in (4.11.29)
S	ndarray of size (n,n): prior scale matrix, defined in (4.11.29)
alpha_bar	float: posterior degrees of freedom, defined in (4.11.33)
mcmc_beta	ndarray of size (k,n,iterations): MCMC values of VAR coefficients
mcmc_Sigma	ndarray of size (n,n,iterations): MCMC values of residual variance-covariance matrix
beta_estimates	ndarray of size (k,n,3): estimates of VAR coefficients page 1: median, page 2: st dev, page 3: lower bound, page 4: upper bound
Sigma_estimates	ndarray of size (n,n): estimates of variance-covariance matrix of VAR residuals
m_y	float: log 10 marginal likelihood, defined in (4.12.21)
b	ndarray of size (q,1): prior mean of VAR coefficients, defined in (4.11.16)
V	ndarray of size (q,q): prior mean of VAR coefficients, defined in (4.11.20)
mcmc_H	ndarray of size (n,n,iterations): MCMC values of structural identification matrix, defined in (4.13.5)
mcmc_Gamma	ndarray of size (iterations,n): MCMC values of structural shock variance matrix, defined in definition 13.1
Y	ndarray of size (T,n): matrix of in-sample endogenous variables, defined in (4.11.3)
Z	ndarray of size (T,m): matrix of in-sample endogenous variables, defined in (4.11.3)
X	ndarray of size (T,k): matrix of exogenous and lagged regressors, defined in (4.11.3)
n	int: number of endogenous variables, defined in (4.11.1)
m	int: number of exogenous variables, defined in (4.11.1)
p	int: number of lags, defined in (4.11.1)
T	int: number of sample periods, defined in (4.11.1)

Attributes:

attribute	description
k	int: number of VAR coefficients in each equation, defined in (4.11.1)
q	int: total number of VAR coefficients, defined in (4.11.1)
delta	ndarray of size (n,1): prior mean delta for AR coefficients, defined in (4.11.16)
s	ndarray of size (n,1): individual AR models residual variance, defined in (4.11.18)
Y_sum	ndarray of size (n,n): sums-of-coefficients Y matrix, defined in (4.12.6)
X_sum	ndarray of size (n,k): sums-of-coefficients X matrix, defined in (4.12.6)
Y_obs	ndarray of size (1,n): dummy initial observation Y matrix, defined in (4.12.10)
X_obs	ndarray of size (1,k): dummy initial observation X matrix, defined in (4.12.10)
Y_lrp	ndarray of size (1,n): long run prior Y matrix, defined in (4.12.16)
X_lrp	ndarray of size (1,k): long run prior X matrix, defined in (4.12.16)
Y_d	ndarray of size (T_d,n): full Y matrix combining sample data and dummy observations, defined in (4.11.62)
X_d	ndarray of size (T_d,k): full X matrix combining sample data and dummy observations, defined in (4.11.62)
T_d	int: total number of observations combining sample data and dummy observations, defined in (4.11.62)
steady_state_estimates	ndarray of size (T,n,3): estimates of steady-state, defined in (4.12.30)
fitted_estimates	ndarray of size (T,n,3): estimates of in-sample fit, defined in (4.11.2)
residual_estimates	ndarray of size (T,n,3): estimates of in-sample residuals, defined in (4.11.2)
structural_shocks _estimates	ndarray of size (T,n,3): estimates of in-sample structural shocks, defined in definition 13.1
insample_evaluation	dict: in-sample evaluation criteria, defined in (4.13.15)-(4.13.17)
mcmc_structural_shocks	ndarray of size (T,n,iterations): MCMC values of structural shocks
mcmc_forecasts	ndarray of size (f_periods,n,iterations): MCMC values of forecasts
forecast_estimates	ndarray of size (f_periods,n,3): forecast estimates, defined in (4.13.12) and (4.13.13) page 1: median, page 2: lower bound, page 3: upper bound
forecast_evaluation _criteria	dict: forecast evaluation criteria, defined in (4.13.18)-(4.13.21)
mcmc_irf	ndarray of size (n,n,irf_periods,iterations): MCMC values of impulse response function, defined in section 13.1
mcmc_irf_exo	ndarray of size (n,m,irf_periods,iterations): MCMC values of exogenous impulse response function
mcmc_structural_irf	ndarray of size (n,n,irf_periods,iterations): MCMC values of structural impulse response function, defined in section 13.2
irf_estimates	ndarray of size (n,n,irf_periods,3): posterior estimates of impulse response function, defined in section 13.1 - 13.2 page 1: median, page 2: lower bound, page 3: upper bound
exo_irf_estimates	ndarray of size (n,m,irf_periods,3): posterior estimates of exogenous impulse response function, if any exogenous variable page 1: median, page 2: lower bound, page 3: upper bound
mcmc_fevd	ndarray of size (n,n,fevd_periods,iterations): MCMC values of forecast error variance decompositions, defined in section 13.4
fevd_estimates	ndarray of size (n,n,fevd_periods,3): posterior estimates of forecast error variance decomposition, defined in section 13.4 page 1: median, page 2: lower bound, page 3: upper bound
mcmc_hd	ndarray of size (n,n,T,iterations): MCMC values of historical decompositions, defined in section 13.5
hd_estimates	ndarray of size (n,n,T,3): posterior estimates of historical decomposition, defined in section 13.5 page 1: median, page 2: lower bound, page 3: upper bound
mcmc_conditional _forecasts	ndarray of size (f_periods,n,iterations): MCMC values of conditional forecasts, defined in section 14.1
conditional_forecast _estimates	ndarray of size (f_periods,n,3): posterior estimates of conditional forecast, defined in section 14.1 page 1: median, page 2: lower bound, page 3: upper bound
H_estimates	ndarray of size (n,n): posterior estimates of structural matrix, defined in section 13.2
Gamma_estimates	ndarray of size (1,n): estimates of structural shock variance matrix, defined in section 13.2

```
# imports
from alexandria import IndependentBayesianVar
from alexandria import DataSets
from alexandria import Results
from alexandria import Graphics
import numpy as np

# load IS-LM dataset, keep only first four columns
ds = DataSets()
islm_data = ds.load_islm()[:,:4]

# create and train Bayesian VAR with Cholesky as structural identification
var = IndependentBayesianVar(islm_data, structural_identification = 2)
var.estimate()

# display console outputs for the model
res = Results(var)
res.make_estimation_summary()
res.show_estimation_summary()

# estimate conditional forecasts: short-term rate is 2.7% over next 2 periods
conditions = np.array([[3,1,2.7,1e-10],[3,2,2.7,1e-10]])
forecasts = var.conditional_forecast(4, 0.6, conditions, [], 1, Z_p = [])

# create graphics of predictions and display them
gp = Graphics(var)
gp.conditional_forecast_graphics(show=True, save=False)
```

Example (Matlab):

```
% load IS-LM dataset, keep only first four columns
ds = DataSets();
islm_data = ds.load_islm();
islm_data = islm_data(:,1:4);

% create and train Bayesian VAR with Cholesky as structural identification
var = IndependentBayesianVar(islm_data, 'structural_identification', 2);
var.estimate();

% display console outputs for the model
res = Results(var);
res.make_estimation_summary();
res.show_estimation_summary();

% estimate conditional forecasts: short-term rate is 2.7% over next 2 periods
conditions = [3 1 2.7 1e-10;3 2 2.7 1e-10];
forecasts = var.conditional_forecast(4, 0.6, conditions, [], 1, Z_p = []);

% create graphics of predictions and display them
gp = Graphics(var);
gp.conditional_forecast_graphics(true, false);
```

7.11 Vector autoregression: DummyObservationBayesianVar

A vector autoregression using the dummy observation prior, described in section 11.5.

Class:

alexandria.vector_autoregression.DummyObservationBayesianVar(endogenous, exogenous = [], structural_identification = 2, restriction_table = [], lags = 4, constant = True, trend = False, quadratic_trend = False, ar_coefficients = 0.95, pi1 = 0.1, pi3 = 1, pi4 = 100, pi5 = 1, pi6 = 0.1, pi7 = 0.1, sums_of_coefficients = False, dummy_initial_observation = False, long_run_prior = False, long_run_table = [], stationary_prior = False, credibility_level = 0.95, iterations = 2000, verbose = False)

Parameters:

parameter	description
endogenous	ndarray of shape (n_obs,n): endogenous variables, defined in (4.11.1)
exogenous	ndarray of shape (n_obs,m), default = []: exogenous variables, defined in (4.11.1)
structural_identification	int, default = 2: structural identification scheme, as defined in section 13.2
restriction_table	ndarray, default = []: numerical matrix of restrictions for structural identification
lags	int, default = 4: number of lags, defined in (4.11.1)
constant	bool, default = True: if True, an intercept is included in the VAR model exogenous
trend	bool, default = False: if True, a linear trend is included in the VAR model exogenous
quadratic_trend	bool, default = False: if True, a quadratic trend is included in the VAR model exogenous
ar_coefficients	float or ndarray of size (n_endo,1), default = 0.95: prior mean delta for AR coefficients, defined in (4.11.16)
pi1	float, default = 0.1: overall tightness hyperparameter, defined in (4.11.17)
pi3	float, default = 1: lag decay hyperparameter, defined in (4.11.17)
pi4	float, default = 100: exogenous slackness hyperparameter, defined in (4.11.19)
pi5	float, default = 1: sums-of-coefficients hyperparameter, defined in (4.12.6)
pi6	float, default = 0.1: initial observation hyperparameter, defined in (4.12.10)
pi7	float, default = 0.1: long-run hyperparameter, defined in (4.12.16)
sums_of_coefficients	bool, default = False: if True, applies sums-of-coefficients, as defined in section 12.2
dummy_initial _observation	bool, default = False: if True, applies dummy initial observation, as defined in section 12.2
long_run_prior	bool, default = False: if True, applies long-run prior, as defined in section 12.2
long_run_table	ndarray, default = []: numerical matrix of long-run prior
stationary_prior	bool, default = False: if True, applies stationary prior, as defined in section 12.4
credibility_level	float, default = 0.95; VAR model credibility level (between 0 and 1)
iterations	int, default = 2000: number of Gibbs sampler replications
verbose	bool, default = False: if True, displays a progress bar

Attributes:

attribute	description
endogenous	ndarray of shape (n_obs,n): endogenous variables, defined in (4.11.1)
exogenous	ndarray of shape (n_obs,m), default = []: exogenous variables, defined in (4.11.1)
structural_identification	int, default = 2: structural identification scheme, as defined in section 13.2
restriction_table	ndarray, default = []: numerical matrix of restrictions for structural identification
lags	int, default = 4: number of lags, defined in (4.11.1)
constant	bool, default = True: if True, an intercept is included in the VAR model exogenous
trend	bool, default = False: if True, a linear trend is included in the VAR model exogenous
quadratic_trend	bool, default = False: if True, a quadratic trend is included in the VAR model exogenous
ar_coefficients	float or ndarray of size (n_endo,1), default = 0.95: prior mean delta for AR coefficients, defined in (4.11.16)
pi1	float, default = 0.1: overall tightness hyperparameter, defined in (4.11.17)
pi3	float, default = 1: lag decay hyperparameter, defined in (4.11.17)
pi4	float, default = 100: exogenous slackness hyperparameter, defined in (4.11.19)
pi5	float, default = 1: sums-of-coefficients hyperparameter, defined in (4.12.6)
pi6	float, default = 0.1: initial observation hyperparameter, defined in (4.12.10)
pi7	float, default = 0.1: long-run hyperparameter, defined in (4.12.16)
sums_of_coefficients	bool, default = False: if True, applies sums-of-coefficients, as defined in section 12.2
dummy_initial _observation	bool, default = False: if True, applies dummy initial observation, as defined in section 12.2
long_run_prior	bool, default = False: if True, applies long-run prior, as defined in section 12.2
J	ndarray of size (n,n); matrix of long-run prior coefficients, defined in (4.12.15)
stationary_prior	bool, default = False: if True, applies stationary prior, as defined in section 12.4
iterations	int, default = 2000: number of Gibbs sampler replications
credibility_level	float, default = 0.95; VAR model credibility level (between 0 and 1)
verbose	bool, default = False: if True, displays a progress bar
Y_dum	ndarray of size (n,n): dummy observation prior Y matrix, defined in (4.12.6)
X_dum	ndarray of size (n,k): dummy observation prior X matrix, defined in (4.12.6)
B_hat	ndarray of size (k,n): posterior mean of VAR coefficients, defined in (4.11.50)
W_hat	ndarray of size (k,k): posterior mean of VAR coefficients, defined in (4.11.50)
alpha_hat	float: posterior degrees of freedom, defined in (4.11.50)
S_hat	ndarray of size (n,n): posterior scale matrix, defined in (4.11.50)
alpha_tilde	float: posterior degrees of freedom, defined in (4.11.53)
S_tilde	ndarray of size (n,n): posterior scale matrix, defined in (4.11.53)
beta_estimates	ndarray of size (k,n,3): estimates of VAR coefficients page 1: median, page 2: st dev, page 3: lower bound, page 4: upper bound
Sigma_estimates	ndarray of size (n,n): estimates of variance-covariance matrix of VAR residuals
mcmc_beta	ndarray of size (k,n,iterations): MCMC values of VAR coefficients
mcmc_Sigma	ndarray of size (n,n,iterations): MCMC values of residual variance-covariance matrix
m_y	float: log 10 marginal likelihood, defined in (4.12.21)
mcmc_H	ndarray of size (n,n,iterations): MCMC values of structural identification matrix, defined in (4.13.5)
mcmc_Gamma	ndarray of size (iterations,n): MCMC values of structural shock variance matrix, defined in definition 13.1
Y	ndarray of size (T,n): matrix of in-sample endogenous variables, defined in (4.11.3)
Z	ndarray of size (T,m): matrix of in-sample endogenous variables, defined in (4.11.3)
X	ndarray of size (T,k): matrix of exogenous and lagged regressors, defined in (4.11.3)
n	int: number of endogenous variables, defined in (4.11.1)
m	int: number of exogenous variables, defined in (4.11.1)
p	int: number of lags, defined in (4.11.1)
T	int: number of sample periods, defined in (4.11.1)

Attributes:

attribute	description
k	int: number of VAR coefficients in each equation, defined in (4.11.1)
q	int: total number of VAR coefficients, defined in (4.11.1)
delta	ndarray of size (n,1): prior mean delta for AR coefficients, defined in (4.11.16)
s	ndarray of size (n,1): individual AR models residual variance, defined in (4.11.18)
Y_sum	ndarray of size (n,n): sums-of-coefficients Y matrix, defined in (4.12.6)
X_sum	ndarray of size (n,k): sums-of-coefficients X matrix, defined in (4.12.6)
Y_obs	ndarray of size (1,n): dummy initial observation Y matrix, defined in (4.12.10)
X_obs	ndarray of size (1,k): dummy initial observation X matrix, defined in (4.12.10)
Y_lrp	ndarray of size (1,n): long run prior Y matrix, defined in (4.12.16)
X_lrp	ndarray of size (1,k): long run prior X matrix, defined in (4.12.16)
Y_d	ndarray of size (T_d,n): full Y matrix combining sample data and dummy observations, defined in (4.11.62)
X_d	ndarray of size (T_d,k): full X matrix combining sample data and dummy observations, defined in (4.11.62)
T_d	int: total number of observations combining sample data and dummy observations, defined in (4.11.62)
steady_state_estimates	ndarray of size (T,n,3): estimates of steady-state, defined in (4.12.30)
fitted_estimates	ndarray of size (T,n,3): estimates of in-sample fit, defined in (4.11.2)
residual_estimates	ndarray of size (T,n,3): estimates of in-sample residuals, defined in (4.11.2)
structural_shocks _estimates	ndarray of size (T,n,3): estimates of in-sample structural shocks, defined in definition 13.1
insample_evaluation	dict: in-sample evaluation criteria, defined in (4.13.15)-(4.13.17)
mcmc_structural_shocks	ndarray of size (T,n,iterations): MCMC values of structural shocks
mcmc_forecasts	ndarray of size (f_periods,n,iterations): MCMC values of forecasts
forecast_estimates	ndarray of size (f_periods,n,3): forecast estimates, defined in (4.13.12) and (4.13.13) page 1: median, page 2: lower bound, page 3: upper bound
forecast_evaluation _criteria	dict: forecast evaluation criteria, defined in (4.13.18)-(4.13.21)
mcmc_irf	ndarray of size (n,n,irf_periods,iterations): MCMC values of impulse response function, defined in section 13.1
mcmc_irf_exo	ndarray of size (n,m,irf_periods,iterations): MCMC values of exogenous impulse response function
mcmc_structural_irf	ndarray of size (n,n,irf_periods,iterations): MCMC values of structural impulse response function, defined in section 13.2
irf_estimates	ndarray of size (n,n,irf_periods,3): posterior estimates of impulse response function, defined in section 13.1 - 13.2 page 1: median, page 2: lower bound, page 3: upper bound
exo_irf_estimates	ndarray of size (n,m,irf_periods,3): posterior estimates of exogenous impulse response function, if any exogenous variable page 1: median, page 2: lower bound, page 3: upper bound
mcmc_fevd	ndarray of size (n,n,fevd_periods,iterations): MCMC values of forecast error variance decompositions, defined in section 13.4
fevd_estimates	ndarray of size (n,n,fevd_periods,3): posterior estimates of forecast error variance decomposition, defined in section 13.4 page 1: median, page 2: lower bound, page 3: upper bound
mcmc_hd	ndarray of size (n,n,T,iterations): MCMC values of historical decompositions, defined in section 13.5
hd_estimates	ndarray of size (n,n,T,3): posterior estimates of historical decomposition, defined in section 13.5 page 1: median, page 2: lower bound, page 3: upper bound
mcmc_conditional _forecasts	ndarray of size (f_periods,n,iterations): MCMC values of conditional forecasts, defined in section 14.1
conditional_forecast _estimates	ndarray of size (f_periods,n,3): posterior estimates of conditional forecast, defined in section 14.1 page 1: median, page 2: lower bound, page 3: upper bound
H_estimates	ndarray of size (n,n): posterior estimates of structural matrix, defined in section 13.2
Gamma_estimates	ndarray of size (1,n): estimates of structural shock variance matrix, defined in section 13.2

Methods:

■ **estimate()**
trains the model and produces posterior estimates for both reduced-form VAR and SVAR parameters
parameters:
- none
returns:
- none

■ **insample_fit()**
generates in-sample fit, residuals, structural shocks and steady-state along with evaluation criteria.
parameters:
- none
returns:
- none

■ **forecast(h, credibility_level, Z_p=[])**
predictions for the vector autoregression model, using (4.13.12).
parameters:
- h: int, number of forecast periods
- credibility_level: float, credibility level for predictions (between 0 and 1)
- Z_p: empty list or ndarray of shape (h,n_exo), matrix of exogenous predictors (except constant and trends)
returns:
- forecasts_estimates: ndarray of shape (h, n, 3), posterior estimates for predictions
page 1: median; page 2: credibility interval lower bound;
page 3: credibility interval upper bound

■ **forecast_evaluation(Y)**
forecast evaluation criteria for the vector autoregression model.
parameters:
- Y: ndarray of shape (h,n), array of realised values for forecast evaluation
returns:
- none

■ **impulse_response_function(h, credibility_level)**
impulse response function for the vector autoregression model, using (4.13.9).
parameters:
- h: int, number of IRF periods
- credibility_level: float, credibility level for predictions (between 0 and 1)
returns:
- irf_estimates: ndarray of shape (n, n, h, 3), posterior estimates for impulse response function
first 3 dimensions are variable, shock, period;
4th dimension is median, lower bound, upper bound

■ **forecast_error_variance_decomposition(h, credibility_level)**
forecast error variance decomposition for the vector autoregression model, using (4.13.31).
parameters:
- h: int, number of FEVD periods
- credibility_level: float, credibility level for predictions (between 0 and 1)
returns:
- fevd_estimates: ndarray of shape (n, n, h, 3), posterior estimates for impulse response function
first 3 dimensions are variable, shock, period;
4th dimension is median, lower bound, upper bound

■ **historical_decomposition(credibility_level)**
historical decomposition for the vector autoregression model, using (4.13.36).
parameters:
- credibility_level: float, credibility level for predictions (between 0 and 1)
returns:
- hd_estimates: ndarray of shape (n, n, T, 3), posterior estimates for impulse response function
first 3 dimensions are variable, shock, period;
4th dimension is median, lower bound, upper bound

■ **conditional_forecast(h, credibility_level, conditions, shocks, conditional_forecast_type, Z_p=[])**
conditional forecasts for the Bayesian VAR model, using algorithms 14.1 and 14.2.
parameters:
- h: int, number of forecast periods
- credibility_level: float, credibility level for predictions (between 0 and 1)
- conditions: ndarray of shape (n_conditions,4)
- shocks: empty list or ndarray of shape (n,), vector defining shocks generating the conditions; should be empty if conditional_forecast_type = 1
- conditional_forecast_type : int, conditional forecast type (1 = agnostic, 2 = structural)
- Z_p: empty list or ndarray of shape (h,n_exo), matrix of exogenous predictors (except constant and trends)
returns:
- conditional_forecasts_estimates: ndarray of shape (h, n, 3), posterior estimates for predictions
page 1: median; page 2: credibility interval lower bound;
page 3: credibility interval upper bound

Example (Python):

```
# imports
from alexandria import DummyObservationBayesianVar
from alexandria import DataSets
from alexandria import Results
from alexandria import Graphics
import numpy as np

# load IS-LM dataset, keep only first four columns
ds = DataSets()
islm_data = ds.load_islm()[:,:4]

# create and train Bayesian VAR with Cholesky as structural identification
var = DummyObservationBayesianVar(islm_data, structural_identification = 2)
var.estimate()

# display console outputs for the model
res = Results(var)
res.make_estimation_summary()
res.show_estimation_summary()

# estimate conditional forecasts: short-term rate is 2.7% over next 2 periods
conditions = np.array([[3,1,2.7,1e-10],[3,2,2.7,1e-10]])
forecasts = var.conditional_forecast(4, 0.6, conditions, [], 1, Z_p = [])

# create graphics of predictions and display them
gp = Graphics(var)
gp.conditional_forecast_graphics(show=True, save=False)
```

Example (Matlab):

```
% load IS-LM dataset, keep only first four columns
ds = DataSets();
islm_data = ds.load_islm();
islm_data = islm_data(:,1:4);

% create and train Bayesian VAR with Cholesky as structural identification
var = DummyObservationBayesianVar(islm_data, 'structural_identification', 2);
var.estimate();

% display console outputs for the model
res = Results(var);
res.make_estimation_summary();
res.show_estimation_summary();

% estimate conditional forecasts: short-term rate is 2.7% over next 2 periods
conditions = [3 1 2.7 1e-10;3 2 2.7 1e-10];
forecasts = var.conditional_forecast(4, 0.6, conditions, [], 1, Z_p = []);

% create graphics of predictions and display them
gp = Graphics(var);
gp.conditional_forecast_graphics(true, false);
```

7.12 Vector autoregression: LargeBayesianVar

A vector autoregression using the large Bayesian VAR prior, described in section 11.6.

Class:

alexandria.vector_autoregression.LargeBayesianVar(endogenous, exogenous = [], structural_identification = 2, restriction_table = [], lags = 4, constant = True, trend = False, quadratic_trend = False, ar_coefficients = 0.95, pi1 = 0.1, pi2 = 0.5, pi3 = 1, pi4 = 100, pi5 = 1, pi6 = 0.1, pi7 = 0.1, constrained_coefficients = False, constrained_coefficients_table = [], sums_of_coefficients = False, dummy_initial_observation = False, long_run_prior = False, long_run_table = [], stationary_prior = False, credibility_level = 0.95, iterations = 2000, burnin = 1000, verbose = False)

Parameters:

parameter	description
endogenous	ndarray of shape (n_obs,n): endogenous variables, defined in (4.11.1)
exogenous	ndarray of shape (n_obs,m), default = []: exogenous variables, defined in (4.11.1)
structural_identification	int, default = 2: structural identification scheme, as defined in section 13.2
restriction_table	ndarray, default = []: numerical matrix of restrictions for structural identification
lags	int, default = 4: number of lags, defined in (4.11.1)
constant	bool, default = True: if True, an intercept is included in the VAR model exogenous
trend	bool, default = False: if True, a linear trend is included in the VAR model exogenous
quadratic_trend	bool, default = False: if True, a quadratic trend is included in the VAR model exogenous
ar_coefficients	float or ndarray of size (n_endo,1), default = 0.95: prior mean delta for AR coefficients, defined in (4.11.16)
pi1	float, default = 0.1: overall tightness hyperparameter, defined in (4.11.17)
pi2	float, default = 0.5: cross-variable shrinkage hyperparameter, defined in (4.11.18)
pi3	float, default = 1: lag decay hyperparameter, defined in (4.11.17)
pi4	float, default = 100: exogenous slackness hyperparameter, defined in (4.11.19)
pi5	float, default = 1: sums-of-coefficients hyperparameter, defined in (4.12.6)
pi6	float, default = 0.1: initial observation hyperparameter, defined in (4.12.10)
pi7	float, default = 0.1: long-run hyperparameter, defined in (4.12.16)
constrained_coefficients	bool, default = False: if True, applies constrained coefficients, as defined in section 12.1
constrained_coefficients_table	ndarray, default = []: numerical matrix of constrained coefficients
sums_of_coefficients	bool, default = False: if True, applies sums-of-coefficients, as defined in section 12.2
dummy_initial_observation	bool, default = False: if True, applies dummy initial observation, as defined in section 12.2
long_run_prior	bool, default = False: if True, applies long-run prior, as defined in section 12.2
long_run_table	ndarray, default = []: numerical matrix of long-run prior
stationary_prior	bool, default = False: if True, applies stationary prior, as defined in section 12.4
credibility_level	float, default = 0.95; VAR model credibility level (between 0 and 1)
iterations	int, default = 2000: number of Gibbs sampler replications
burnin	int, default = 1000: number of Gibbs sampler burn-in replications
verbose	bool, default = False: if True, displays a progress bar

Attributes:

attribute	description
endogenous	ndarray of shape (n_obs,n): endogenous variables, defined in (4.11.1)
exogenous	ndarray of shape (n_obs,m), default = []: exogenous variables, defined in (4.11.1)
structural_identification	int, default = 2: structural identification scheme, as defined in section 13.2
restriction_table	ndarray, default = []: numerical matrix of restrictions for structural identification
lags	int, default = 4: number of lags, defined in (4.11.1)
constant	bool, default = True: if True, an intercept is included in the VAR model exogenous
trend	bool, default = False: if True, a linear trend is included in the VAR model exogenous
quadratic_trend	bool, default = False: if True, a quadratic trend is included in the VAR model exogenous
ar_coefficients	float or ndarray of size (n_endo,1), default = 0.95: prior mean delta for AR coefficients, defined in (4.11.16)
pi1	float, default = 0.1: overall tightness hyperparameter, defined in (4.11.17)
pi2	float, default = 0.5: cross-variable shrinkage hyperparameter, defined in (4.11.18)
pi3	float, default = 1: lag decay hyperparameter, defined in (4.11.17)
pi4	float, default = 100: exogenous slackness hyperparameter, defined in (4.11.19)
pi5	float, default = 1: sums-of-coefficients hyperparameter, defined in (4.12.6)
pi6	float, default = 0.1: initial observation hyperparameter, defined in (4.12.10)
pi7	float, default = 0.1: long-run hyperparameter, defined in (4.12.16)
constrained_coefficients	bool, default = False: if True, applies constrained coefficients, as defined in section 12.1
constrained_coefficients _table	ndarray, default = []: numerical matrix of constrained coefficients
sums_of_coefficients	bool, default = False: if True, applies sums-of-coefficients, as defined in section 12.2
dummy_initial _observation	bool, default = False: if True, applies dummy initial observation, as defined in section 12.2
long_run_prior	bool, default = False: if True, applies long-run prior, as defined in section 12.2
J	ndarray of size (n,n); matrix of long-run prior coefficients, defined in (4.12.15)
stationary_prior	bool, default = False: if True, applies stationary prior, as defined in section 12.4
credibility_level	float, default = 0.95: VAR model credibility level (between 0 and 1)
iterations	int, default = 2000: number of Gibbs sampler replications
burnin	int: number of Gibbs sampler burn-in replications
verbose	bool, default = False: if True, displays a progress bar
alpha_bar	float: posterior degrees of freedom, defined in (4.11.79)
mcmc_beta	ndarray of size (k,n,iterations): MCMC values of VAR coefficients
mcmc_Sigma	ndarray of size (n,n,iterations): MCMC values of residual variance-covariance matrix
beta_estimates	ndarray of size (k,n,3): estimates of VAR coefficients page 1: median, page 2: st dev, page 3: lower bound, page 4: upper bound
Sigma_estimates	ndarray of size (n,n): estimates of variance-covariance matrix of VAR residuals
b	ndarray of size (q,1): prior mean of VAR coefficients, defined in (4.11.16)
V	ndarray of size (q,q): prior mean of VAR coefficients, defined in (4.11.20)
mcmc_H	ndarray of size (n,n,iterations): MCMC values of structural identification matrix, defined in (4.13.5)
mcmc_Gamma	ndarray of size (iterations,n): MCMC values of structural shock variance matrix, defined in definition 13.1
Y	ndarray of size (T,n): matrix of in-sample endogenous variables, defined in (4.11.3)
Z	ndarray of size (T,m): matrix of in-sample endogenous variables, defined in (4.11.3)
X	ndarray of size (T,k): matrix of exogenous and lagged regressors, defined in (4.11.3)
n	int: number of endogenous variables, defined in (4.11.1)
m	int: number of exogenous variables, defined in (4.11.1)
p	int: number of lags, defined in (4.11.1)
T	int: number of sample periods, defined in (4.11.1)

Attributes:

attribute	description
k	int: number of VAR coefficients in each equation, defined in (4.11.1)
q	int: total number of VAR coefficients, defined in (4.11.1)
delta	ndarray of size (n,1): prior mean delta for AR coefficients, defined in (4.11.16)
s	ndarray of size (n,1): individual AR models residual variance, defined in (4.11.18)
Y_sum	ndarray of size (n,n): sums-of-coefficients Y matrix, defined in (4.12.6)
X_sum	ndarray of size (n,k): sums-of-coefficients X matrix, defined in (4.12.6)
Y_obs	ndarray of size (1,n): dummy initial observation Y matrix, defined in (4.12.10)
X_obs	ndarray of size (1,k): dummy initial observation X matrix, defined in (4.12.10)
Y_lrp	ndarray of size (1,n): long run prior Y matrix, defined in (4.12.16)
X_lrp	ndarray of size (1,k): long run prior X matrix, defined in (4.12.16)
Y_d	ndarray of size (T_d,n): full Y matrix combining sample data and dummy observations, defined in (4.11.62)
X_d	ndarray of size (T_d,k): full X matrix combining sample data and dummy observations, defined in (4.11.62)
T_d	int: total number of observations combining sample data and dummy observations, defined in (4.11.62)
steady_state_estimates	ndarray of size (T,n,3): estimates of steady-state, defined in (4.12.30)
fitted_estimates	ndarray of size (T,n,3): estimates of in-sample fit, defined in (4.11.2)
residual_estimates	ndarray of size (T,n,3): estimates of in-sample residuals, defined in (4.11.2)
structural_shocks _estimates	ndarray of size (T,n,3): estimates of in-sample structural shocks, defined in definition 13.1
insample_evaluation	dict: in-sample evaluation criteria, defined in (4.13.15)-(4.13.17)
mcmc_structural_shocks	ndarray of size (T,n,iterations): MCMC values of structural shocks
mcmc_forecasts	ndarray of size (f_periods,n,iterations): MCMC values of forecasts
forecast_estimates	ndarray of size (f_periods,n,3): forecast estimates, defined in (4.13.12) and (4.13.13) page 1: median, page 2: lower bound, page 3: upper bound
forecast_evaluation _criteria	dict: forecast evaluation criteria, defined in (4.13.18)-(4.13.21)
mcmc_irf	ndarray of size (n,n,irf_periods,iterations): MCMC values of impulse response function, defined in section 13.1
mcmc_irf_exo	ndarray of size (n,m,irf_periods,iterations): MCMC values of exogenous impulse response function
mcmc_structural_irf	ndarray of size (n,n,irf_periods,iterations): MCMC values of structural impulse response function, defined in section 13.2
irf_estimates	ndarray of size (n,n,irf_periods,3): posterior estimates of impulse response function, defined in section 13.1 - 13.2 page 1: median, page 2: lower bound, page 3: upper bound
exo_irf_estimates	ndarray of size (n,m,irf_periods,3): posterior estimates of exogenous impulse response function, if any exogenous variable page 1: median, page 2: lower bound, page 3: upper bound
mcmc_fevd	ndarray of size (n,n,fevd_periods,iterations): MCMC values of forecast error variance decompositions, defined in section 13.4
fevd_estimates	ndarray of size (n,n,fevd_periods,3): posterior estimates of forecast error variance decomposition, defined in section 13.4 page 1: median, page 2: lower bound, page 3: upper bound
mcmc_hd	ndarray of size (n,n,T,iterations): MCMC values of historical decompositions, defined in section 13.5
hd_estimates	ndarray of size (n,n,T,3): posterior estimates of historical decomposition, defined in section 13.5 page 1: median, page 2: lower bound, page 3: upper bound
mcmc_conditional _forecasts	ndarray of size (f_periods,n,iterations): MCMC values of conditional forecasts, defined in section 14.1
conditional_forecast _estimates	ndarray of size (f_periods,n,3): posterior estimates of conditional forecast, defined in section 14.1 page 1: median, page 2: lower bound, page 3: upper bound
H_estimates	ndarray of size (n,n): posterior estimates of structural matrix, defined in section 13.2
Gamma_estimates	ndarray of size (1,n): estimates of structural shock variance matrix, defined in section 13.2

Example (Python):

```python
# imports
from alexandria import LargeBayesianVar
from alexandria import DataSets
from alexandria import Results
from alexandria import Graphics
import numpy as np

# load IS-LM dataset, keep only first four columns
ds = DataSets()
islm_data = ds.load_islm()[:,:4]

# create and train Bayesian VAR with Cholesky as structural identification
var = LargeBayesianVar(islm_data, structural_identification = 2)
var.estimate()

# display console outputs for the model
res = Results(var)
res.make_estimation_summary()
res.show_estimation_summary()

# estimate conditional forecasts: short-term rate is 2.7% over next 2 periods
conditions = np.array([[3,1,2.7,1e-10],[3,2,2.7,1e-10]])
forecasts = var.conditional_forecast(4, 0.6, conditions, [], 1, Z_p = [])

# create graphics of predictions and display them
gp = Graphics(var)
gp.conditional_forecast_graphics(show=True, save=False)
```

Example (Matlab):

```matlab
% load IS-LM dataset, keep only first four columns
ds = DataSets();
islm_data = ds.load_islm();
islm_data = islm_data(:,1:4);

% create and train Bayesian VAR with Cholesky as structural identification
var = LargeBayesianVar(islm_data, 'structural_identification', 2);
var.estimate();

% display console outputs for the model
res = Results(var);
res.make_estimation_summary();
res.show_estimation_summary();

% estimate conditional forecasts: short-term rate is 2.7% over next 2 periods
conditions = [3 1 2.7 1e-10;3 2 2.7 1e-10];
forecasts = var.conditional_forecast(4, 0.6, conditions, [], 1, Z_p = []);

% create graphics of predictions and display them
gp = Graphics(var);
gp.conditional_forecast_graphics(true, false);
```

7.13 Vector autoregression: BayesianProxySvar

A Bayesian proxy-SVAR, described in section 14.5.

Class:

alexandria.vector_autoregression.BayesianProxySvar(endogenous, proxys, exogenous = [], structural_identification = 1, restriction_table = [], lamda = 0.2, proxy_prior = 1, lags = 4, constant = True, trend = False, quadratic_trend = False, ar_coefficients = 0.95, pi1 = 0.1, pi3 = 1, pi4 = 100, credibility_level = 0.95, iterations = 2000, burnin = 1000, verbose = False)

Parameters:

parameter	description
endogenous	ndarray of size (n_obs,n_endogenous): endogenous variables, defined in (4.11.1)
proxys	ndarray of size (n_obs,n_proxys): proxy variables, defined in (4.14.42)
exogenous	ndarray of size (n_obs,n_exogenous), default = []: exogenous variables, defined in (4.11.1)
structural_identification	int, default = 1: structural identification scheme, additional to proxy-SVAR 1 = none, 4 = restrictions
restriction_table	ndarray: numerical matrix of restrictions for structural identification
lamda	float, default = 0.2: relevance parameter, defined in (4.14.54)
proxy_prior	int, default = 1: prior scheme for normal-generalized-normal prior 1 = uninformative, 2 = Minnesota
lags	int, default = 4: number of lags, defined in (4.11.1)
constant	bool, default = True: if True, an intercept is included in the VAR model exogenous
trend	bool, default = False: if True, a linear trend is included in the VAR model exogenous
quadratic_trend	bool, default = False: if True, a quadratic trend is included in the VAR model exogenous
ar_coefficients	float or ndarray of size (n_endo,1), default = 0.95: prior mean delta for AR coefficients, defined in (4.11.16)
pi1	float, default = 0.1: overall tightness hyperparameter, defined in (4.11.17)
pi3	float, default = 1: lag decay hyperparameter, defined in (4.11.17)
pi4	float, default = 100: exogenous slackness hyperparameter, defined in (4.11.19)
credibility_level	float, default = 0.95: VAR model credibility level (between 0 and 1)
iterations	int, default = 2000: number of Gibbs sampler replications
burnin	int, default = 1000: number of Gibbs sampler burn-in replications
verbose	bool, default = False: if True, displays a progress bar

Attributes:

attribute	description
endogenous	ndarray of shape (n_obs,n): endogenous variables, defined in (4.11.1)
proxys	ndarray of size (n_obs,n_proxys): proxy variables, defined in (4.14.42)
exogenous	ndarray of shape (n_obs,m), default = []: exogenous variables, defined in (4.11.1)
structural_identification	int, default = 2: structural identification scheme, as defined in section 13.2
restriction_table	ndarray, default = []: numerical matrix of restrictions for structural identification
lags	int, default = 4: number of lags, defined in (4.11.1)
constant	bool, default = True: if True, an intercept is included in the VAR model exogenous
trend	bool, default = False: if True, a linear trend is included in the VAR model exogenous
quadratic_trend	bool, default = False: if True, a quadratic trend is included in the VAR model exogenous
ar_coefficients	float or ndarray of size (n_endo,1), default = 0.95: prior mean delta for AR coefficients, defined in (4.11.16)
pi1	float, default = 0.1: overall tightness hyperparameter, defined in (4.11.17)
pi3	float, default = 1: lag decay hyperparameter, defined in (4.11.17)
pi4	float, default = 100: exogenous slackness hyperparameter, defined in (4.11.19)
credibility_level	float, default = 0.95; VAR model credibility level (between 0 and 1)
iterations	int, default = 2000: number of Gibbs sampler replications
burnin	int: number of Gibbs sampler burn-in replications
verbose	bool, default = False: if True, displays a progress bar
R	ndarray of size (T,h): in-sample matrix of proxy variables, defined in (4.14.42)
Y_bar	ndarray of size (T,n+h): in-sample matrix of endogenous and proxy regressors, defined in (4.14.49)
X_bar	ndarray of size (T,m+(n+h)*p): in-sample matrix of full regressors, defined in (4.14.49)
h	int: number of proxy variables
n_bar	int: number of endogenous and proxy variables (n+h)
k_bar	int: total number of proxy-SVAR regressors (m+(n+h)*p)
alpha	float: prior degrees of freedom, defined in (4.14.50)
inv_W	ndarray of size (k_bar,k_bar): prior variance of VAR coefficients, defined in (4.14.50)
B	ndarray of size (k_bar,n_bar): prior mean of VAR coefficients, defined in (4.14.50)
S	ndarray of size (n_bar,n_bar): prior scale matrix, defined in (4.14.50)
alpha_bar	float: posterior degrees of freedom, defined in (4.11.33)
W_bar	ndarray of size (k_bar,k_bar): posterior variance of VAR coefficients, defined in (4.14.50)
B_bar	ndarray of size (k_bar,n_bar): posterior mean of VAR coefficients, defined in (4.14.50)
S_bar	ndarray of size (n_bar,n_bar): posterior scale matrix, defined in (4.14.50)
mcmc_beta	ndarray of size (k,n,iterations): MCMC values of VAR coefficients
mcmc_Sigma	ndarray of size (n,n,iterations): MCMC values of residual variance-covariance matrix
mcmc_V	ndarray of size (h,h,iterations): MCMC values of covariance matrix V, defined in (4.14.46)
mcmc_min_eigenvalue	ndarray of size (iterations,): MCMC values of minimum eigenvalue, defined in (4.14.54)
beta_estimates	ndarray of size (k,n,3): estimates of VAR coefficients page 1: median, page 2: st dev, page 3: lower bound, page 4: upper bound
Sigma_estimates	ndarray of size (n,n): estimates of variance-covariance matrix of VAR residuals
V_estimates	ndarray of size (h,h): posterior estimates of covariance matrix V, defined in (4.14.46)
min_eigenvalue _estimates	float: posterior estimate of minimum eigenvalue, defined in (4.14.54)
mcmc_H	ndarray of size (n,n,iterations): MCMC values of structural identification matrix, defined in (4.13.5)
mcmc_Gamma	ndarray of size (iterations,n): MCMC values of structural shock variance matrix, defined in definition 13.1
Y	ndarray of size (T,n): matrix of in-sample endogenous variables, defined in (4.11.3)
Z	ndarray of size (T,m): matrix of in-sample endogenous variables, defined in (4.11.3)
X	ndarray of size (T,k): matrix of exogenous and lagged regressors, defined in (4.11.3)
n	int: number of endogenous variables, defined in (4.11.1)
m	int: number of exogenous variables, defined in (4.11.1)
p	int: number of lags, defined in (4.11.1)
T	int: number of sample periods, defined in (4.11.1)

Attributes:

attribute	description
k	int: number of VAR coefficients in each equation, defined in (4.11.1)
q	int: total number of VAR coefficients, defined in (4.11.1)
delta	ndarray of size (n,1): prior mean delta for AR coefficients, defined in (4.11.16)
s	ndarray of size (n,1): individual AR models residual variance, defined in (4.11.18)
steady_state_estimates	ndarray of size (T,n,3): estimates of steady-state, defined in (4.12.30)
fitted_estimates	ndarray of size (T,n,3): estimates of in-sample fit, defined in (4.11.2)
residual_estimates	ndarray of size (T,n,3): estimates of in-sample residuals, defined in (4.11.2)
structural_shocks _estimates	ndarray of size (T,n,3): estimates of in-sample structural shocks, defined in definition 13.1
insample_evaluation	dict: in-sample evaluation criteria, defined in (4.13.15)-(4.13.17)
mcmc_structural_shocks	ndarray of size (T,n,iterations): MCMC values of structural shocks
mcmc_forecasts	ndarray of size (f_periods,n,iterations): MCMC values of forecasts
forecast_estimates	ndarray of size (f_periods,n,3): forecast estimates, defined in (4.13.12) and (4.13.13) page 1: median, page 2: lower bound, page 3: upper bound
forecast_evaluation _criteria	dict: forecast evaluation criteria, defined in (4.13.18)-(4.13.21)
mcmc_irf	ndarray of size (n,n,irf_periods,iterations): MCMC values of impulse response function, defined in section 13.1
mcmc_irf_exo	ndarray of size (n,m,irf_periods,iterations): MCMC values of exogenous impulse response function
mcmc_structural_irf	ndarray of size (n,n,irf_periods,iterations): MCMC values of structural impulse response function, defined in section 13.2
irf_estimates	ndarray of size (n,n,irf_periods,3): posterior estimates of impulse response function, defined in section 13.1 - 13.2 page 1: median, page 2: lower bound, page 3: upper bound
exo_irf_estimates	ndarray of size (n,m,irf_periods,3): posterior estimates of exogenous impulse response function, if any exogenous variable page 1: median, page 2: lower bound, page 3: upper bound
mcmc_fevd	ndarray of size (n,n,fevd_periods,iterations): MCMC values of forecast error variance decompositions, defined in section 13.4
fevd_estimates	ndarray of size (n,n,fevd_periods,3): posterior estimates of forecast error variance decomposition, defined in section 13.4 page 1: median, page 2: lower bound, page 3: upper bound
mcmc_hd	ndarray of size (n,n,T,iterations): MCMC values of historical decompositions, defined in section 13.5
hd_estimates	ndarray of size (n,n,T,3): posterior estimates of historical decomposition, defined in section 13.5 page 1: median, page 2: lower bound, page 3: upper bound
mcmc_conditional _forecasts	ndarray of size (f_periods,n,iterations): MCMC values of conditional forecasts, defined in section 14.1
conditional_forecast _estimates	ndarray of size (f_periods,n,3): posterior estimates of conditional forecast, defined in section 14.1 page 1: median, page 2: lower bound, page 3: upper bound
H_estimates	ndarray of size (n,n): posterior estimates of structural matrix, defined in section 13.2
Gamma_estimates	ndarray of size (1,n): estimates of structural shock variance matrix, defined in section 13.2

Methods:

■ **estimate()**
trains the model and produces posterior estimates for both reduced-form VAR and SVAR parameters
parameters:
- none
returns:
- none

■ **insample_fit()**
generates in-sample fit, residuals, structural shocks and steady-state along with evaluation criteria.
parameters:
- none
returns:
- none

■ **forecast(h, credibility_level, Z_p=[])**
predictions for the vector autoregression model, using (4.13.12).
parameters:
- h: int, number of forecast periods
- credibility_level: float, credibility level for predictions (between 0 and 1)
- Z_p: empty list or ndarray of shape (h,n_exo), matrix of exogenous predictors (except constant and trends)
returns:
- forecasts_estimates: ndarray of shape (h, n, 3), posterior estimates for predictions
page 1: median; page 2: credibility interval lower bound;
page 3: credibility interval upper bound

■ **forecast_evaluation(Y)**
forecast evaluation criteria for the vector autoregression model.
parameters:
- Y: ndarray of shape (h,n), array of realised values for forecast evaluation
returns:
- none

■ **impulse_response_function(h, credibility_level)**
impulse response function for the vector autoregression model, using (4.13.9).
parameters:
- h: int, number of IRF periods
- credibility_level: float, credibility level for predictions (between 0 and 1)
returns:
- irf_estimates: ndarray of shape (n, n, h, 3), posterior estimates for impulse response function
first 3 dimensions are variable, shock, period;
4th dimension is median, lower bound, upper bound

■ **forecast_error_variance_decomposition(h, credibility_level)**
forecast error variance decomposition for the vector autoregression model, using (4.13.31).
parameters:
- h: int, number of FEVD periods
- credibility_level: float, credibility level for predictions (between 0 and 1)
returns:
- fevd_estimates: ndarray of shape (n, n, h, 3), posterior estimates for impulse response function
first 3 dimensions are variable, shock, period;
4th dimension is median, lower bound, upper bound

■ **historical_decomposition(credibility_level)**
historical decomposition for the vector autoregression model, using (4.13.36).
parameters:
- credibility_level: float, credibility level for predictions (between 0 and 1)
returns:
- hd_estimates: ndarray of shape (n, n, T, 3), posterior estimates for impulse response function
first 3 dimensions are variable, shock, period;
4th dimension is median, lower bound, upper bound

■ **conditional_forecast(h, credibility_level, conditions, shocks, conditional_forecast_type, Z_p=[])**
conditional forecasts for the Bayesian VAR model, using algorithms 14.1 and 14.2.
parameters:
- h: int, number of forecast periods
- credibility_level: float, credibility level for predictions (between 0 and 1)
- conditions: ndarray of shape (n_conditions,4)
- shocks: empty list or ndarray of shape (n,), vector defining shocks generating the conditions; should be empty if conditional_forecast_type = 1
- conditional_forecast_type : int, conditional forecast type (1 = agnostic, 2 = structural)
- Z_p: empty list or ndarray of shape (h,n_exo), matrix of exogenous predictors (except constant and trends)
returns:
- conditional_forecasts_estimates: ndarray of shape (h, n, 3), posterior estimates for predictions
page 1: median; page 2: credibility interval lower bound;
page 3: credibility interval upper bound

Example (Python):

```python
# imports
from alexandria import BayesianProxySvar
from alexandria import DataSets
from alexandria import Results
from alexandria import Graphics
import numpy as np

# load IS-LM dataset, keep all columns
ds = DataSets()
islm_endogenous = ds.load_islm()[:,:4]
islm_proxys = ds.load_islm()[:,4:]

# matrix of covariance restrictions: each proxy is positively correlated with its shock
restrictions = np.array([[5,1,0,0,0,1,0],[5,2,0,0,0,0,1]])

# create and train Bayesian proxy-SVAR with covariance restrictions
var = BayesianProxySvar(islm_endogenous, islm_proxys, structural_identification = 4, restrict
var.estimate()

# display console outputs for the model
res = Results(var)
res.make_estimation_summary()
res.show_estimation_summary()

# estimate conditional forecasts: short-term rate is 2.7% over next 2 periods
conditions = np.array([[3,1,2.7,1e-10],[3,2,2.7,1e-10]])
forecasts = var.conditional_forecast(4, 0.6, conditions, [], 1, Z_p = [])

# create graphics of predictions and display them
gp = Graphics(var)
gp.conditional_forecast_graphics(show=True, save=False)
```

Example (Matlab):

```matlab
% load IS-LM dataset, keep all columns
ds = DataSets();
islm_data = ds.load_islm();
islm_endogenous = islm_data(:,1:4);
islm_proxys = islm_data(:,5:end);

% matrix of covariance restrictions: each proxy is positively correlated with its shock
restrictions = [5 1 0 0 0 1 0; 5 2 0 0 0 0 1];

% create and train Bayesian proxy-SVAR with covariance restrictions
var = BayesianProxySvar(islm_endogenous, islm_proxys, 'structural_identification', 4, 'restri
var.estimate();

% display console outputs for the model
res = Results(var);
res.make_estimation_summary();
res.show_estimation_summary();
```

```
% estimate conditional forecasts: short-term rate is 2.7% over next 2 periods
conditions = [3 1 2.7 1e-10;3 2 2.7 1e-10];
forecasts = var.conditional_forecast(4, 0.6, conditions, [], 1, Z_p = []);

% create graphics of predictions and display them
gp = Graphics(var);
gp.conditional_forecast_graphics(true, false);
```

CHAPTER 8

Alexandria results - documentation

Alexandria proposes the Results class as a convenience tool to create, vizualize and save summaries of inputs, model estimation and model applications. The class is especially useful if you use Alexandria on the fly as it permits to create the same estimation outputs as the ones you would obtain by using the Graphical user Interface. This permits to obtain nicely formatted information at a glance, avoiding the hastle to interpret your model attributes manually.

8.1 Examples

The Results class performs three main functions: saving your model inputs for later replication; producing estimation summaries to oversee model quality; and creating application summaries for easy use in e.g. reports or research works.

```
==========================================================================
   _____   __                              _                 
  / __  / / /___  __  __  ____   ____  ___/ / ____  (_) ____ 
 / /_/ / / / / _ \ \ \/ / / __ `/ / __ \/ __  / / __/ / / / __ `/
/ __  / / / /  __/  |  | / /_/ / / / / / /_/ / / /   / / / /_/ / 
/_/ /_/ /_/ \___/  /_/\_\ \__,_/ /_/ /_/ \____/ /_/   /_/  \__,_/  

The library of Bayesian time-series models
V 1.0 - Copyright © Romain Legrand
==========================================================================


Estimation date:  2025-04-14 00:49:22


Model
---------

selected model: vector autoregression
endogenous variables: ffrate, inflation, gap
exogenous variables: none
data frequency: quarterly
estimation sample: 1956-03-31 2019-12-31
path to project folder: /home/romain/alexandria-matlab-in-progress
data file: data/data.csv
progress bar: yes
create graphics: yes
save_results: yes


Specification
-----------------

VAR type: Minnesota Bayesian VAR
iterations: 2000
credibility level: 0.6
constant: yes
trend: no
quadratic trend: no
lags: 2
AR coefficients: 0.9
pi1 (overall tightness): 0.1
pi2 (cross-variable shrinkage): 0.5
pi3 (lag decay): 1
pi4 (exogenous slackness): 100
pi5 (sums-of-coefficients tightness): 1
pi6 (initial observation tightness): 0.001
pi7 (long-run tightness): 0.1
in-sample fit: no
constrained coefficients: no
```

Figure 8.1: Input summary

Figure 8.1 illustrates the kind of input summary produced by the Result class. The file reports the user inputs for a given estimated model, following the oragnization of the Graphical User Interface. The "model" part of the report contains the information of tab 1 of the GUI (the "Models" tab). The "Specification" part contains the information of tab 2 of the GUI (the "Specifications" tab), while the "Applications" part (not visible on the figure) contains the information of tab 3 of the GUI (the "Applications" tab). The information is comprehensive and should permit to replicate a model exactly in the future, should you need it.

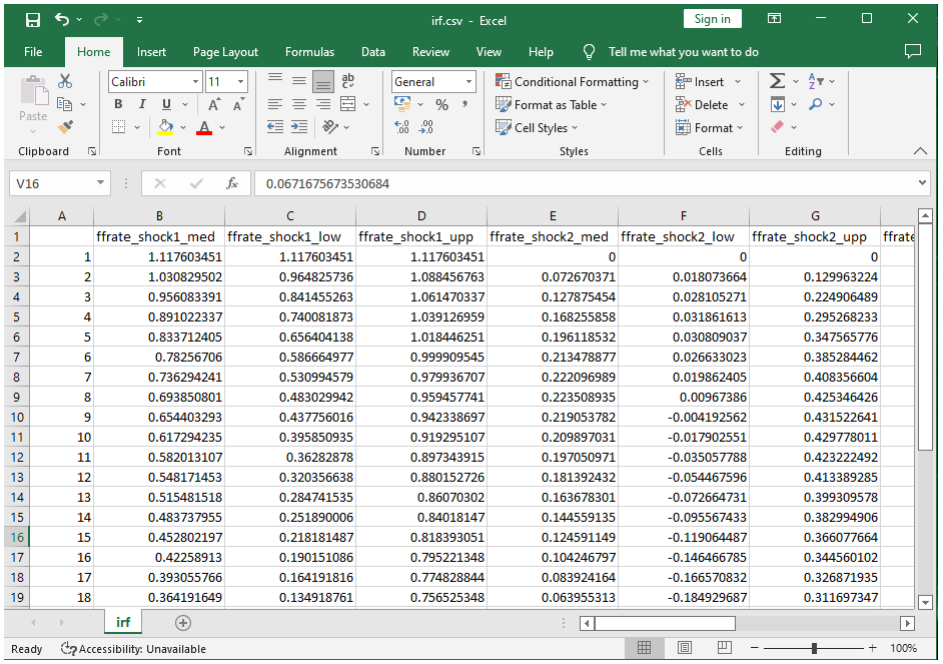
Figure 8.2 shows (a small part of) the kind of estimation summary produced by the Result class. For a given model, it creates a nicely formatted report summarizing the principle features of the model. For a vector autoregression for instance, the report will include a first block with general sample information; then a block of model coefficients, equation by equation; a block of summary of the residual variance, structural shock variance and structural identification matrices; and a final block with forecast evaluation criteria. While the summary is not fully comprehensive, it provides a broad overview of the model and permits an analysis of the model at a glance.

```
The library of Bayesian time-series models
V 1.0 - Copyright © Romain Legrand

================================================================================
                          Normal-Wishart Bayesian Var
================================================================================
Sample:          1956-03-31  2019-12-31    Est. start:        2025-04-14 10:33:39
No. observations:                   254    Est. complete:     2025-04-14 10:33:40
Frequency:                    quarterly    Lags:                                2
================================================================================
                                Equation: ffrate
--------------------------------------------------------------------------------
                              median        std dev        [0.025         0.975]
--------------------------------------------------------------------------------
VAR coefficients beta:
constant                       0.218          0.125        -0.028          0.463
ffrate (-1)                    0.875          0.042         0.793          0.956
ffrate (-2)                    0.025          0.037        -0.047          0.097
inf (-1)                       0.024          0.057        -0.089          0.137
inf (-2)                       0.065          0.052        -0.038          0.168
gap (-1)                       0.151          0.057         0.039          0.264
gap (-2)                      -0.078          0.053        -0.183          0.026
--------------------------------------------------------------------------------
residual variance:                1.239
================================================================================
                                Equation: inf
--------------------------------------------------------------------------------
                              median        std dev        [0.025         0.975]
--------------------------------------------------------------------------------
VAR coefficients beta:
constant                       0.198          0.087         0.028          0.368
ffrate (-1)                    0.070          0.029         0.014          0.127
ffrate (-2)                   -0.043          0.026        -0.093          0.008
inf (-1)                       0.953          0.040         0.875          1.031
inf (-2)                      -0.027          0.036        -0.099          0.044
gap (-1)                       0.077          0.040        -0.001          0.155
gap (-2)                       0.003          0.037        -0.070          0.076
--------------------------------------------------------------------------------
residual variance:                0.596
================================================================================
```

Figure 8.2: Estimation summary

Application summaries finally permit to record and save summaries of your model applications. This is handy for e.g. later use in speadsheets, or use to produce economic reports. Figure 8.3 shows an example application summary file for impulse response functions. The file reports the response of each variable to each shock, detailing the point estimates, credibility interval lower bound and credibility interval upper bound. Similar files can be generated for all model applications.



irf.csv - Excel

	A	B	C	D	E	F	G	
1		ffrate_shock1_med	ffrate_shock1_low	ffrate_shock1_upp	ffrate_shock2_med	ffrate_shock2_low	ffrate_shock2_upp	ffrate
2	1	1.117603451	1.117603451	1.117603451	0	0	0	
3	2	1.030829502	0.964825736	1.088456763	0.072670371	0.018073664	0.129963224	
4	3	0.956083391	0.841455263	1.061470337	0.127875454	0.028105271	0.224906489	
5	4	0.891022337	0.740081873	1.039126959	0.168255858	0.031861613	0.295268233	
6	5	0.833712405	0.656404138	1.018446251	0.196118532	0.030809037	0.347565776	
7	6	0.78256706	0.586664977	0.999909545	0.213478877	0.026633023	0.385284462	
8	7	0.736294241	0.530994579	0.979936707	0.222096989	0.019862405	0.408356604	
9	8	0.693850801	0.483029942	0.959457741	0.223508935	0.00967386	0.425346426	
10	9	0.654403293	0.437756016	0.942338697	0.219053782	-0.004192562	0.431522641	
11	10	0.617294235	0.395850935	0.919295107	0.209897031	-0.017902551	0.429778011	
12	11	0.582013107	0.36282878	0.897343915	0.197050971	-0.035057788	0.423222492	
13	12	0.548171453	0.320356638	0.880152726	0.181392432	-0.054467596	0.413389285	
14	13	0.515481518	0.284741535	0.86070302	0.163678301	-0.072664731	0.399309578	
15	14	0.483737955	0.251890006	0.84018147	0.144559135	-0.095567433	0.382994906	
16	15	0.452802197	0.218181487	0.818393051	0.124591149	-0.119064487	0.366077664	
17	16	0.42258913	0.190151086	0.795221348	0.104246797	-0.146466785	0.344560102	
18	17	0.393055766	0.164191816	0.774828844	0.083924164	-0.166570832	0.326871935	
19	18	0.364191649	0.134918761	0.756525348	0.063955313	-0.184929687	0.311697347	

Figure 8.3: Application summary

8.2 Class arguments

To use the Results class, you first need to create an instance of it. You can then use the different methods of the class to generate, vizualize and saves the different results outputs. The class takes two arguments when it is instantiated. A mandatory element: the estimated model; and an optional argument: the complementary_information dictionary (in Python) or structure (in Matlab).

The mandatory element for the Results class is the model from which you want to extract the result outputs. So for instance assume you just estimated a Normal-Wishart Bayesian VAR on the fly with the command:

```
var = NormalWishartBayesianVar(data)
```

The model "var" can then be used as an argument to initiate the Results class:

```
res = Results(var)
```

The object "res" is the instance of the Results class, which can then be used to run the methods of the class. For instance, to create an estimation summary of the VAR model, use:

```
res.make_estimation_summary()
```

To display the summary in the console, use then:

```
res.show_estimation_summary()
```

The console output is displayed in Figure 8.4:

```
================================================================================
                        Normal-Wishart Bayesian Var
================================================================================
Sample:                                 –    Est. start:                      –
No. observations:                     254    Est. complete:     2025-04-14 02:26:01
Frequency:                              –    Lags:                            2
================================================================================
                                  Equation: y1
--------------------------------------------------------------------------------
                           median        std dev         [0.025         0.975]
--------------------------------------------------------------------------------
VAR coefficients beta:
constant                    0.218          0.125         -0.028          0.463
y1 (-1)                     0.875          0.042          0.793          0.956
y1 (-2)                     0.025          0.037         -0.047          0.097
y2 (-1)                     0.024          0.057         -0.089          0.137
y2 (-2)                     0.065          0.052         -0.038          0.168
y3 (-1)                     0.151          0.057          0.039          0.264
y3 (-2)                    -0.078          0.053         -0.183          0.026
--------------------------------------------------------------------------------
residual variance:              1.239    shock variance:                   1.000
================================================================================
                                  Equation: y2
--------------------------------------------------------------------------------
                           median        std dev         [0.025         0.975]
--------------------------------------------------------------------------------
VAR coefficients beta:
constant                    0.198          0.087          0.028          0.368
y1 (-1)                     0.070          0.029          0.014          0.127
y1 (-2)                    -0.043          0.026         -0.093          0.008
y2 (-1)                     0.953          0.040          0.875          1.031
y2 (-2)                    -0.027          0.036         -0.099          0.044
y3 (-1)                     0.077          0.040         -0.001          0.155
y3 (-2)                     0.003          0.037         -0.070          0.076
--------------------------------------------------------------------------------
residual variance:              0.596    shock variance:                   1.000
================================================================================
```

Figure 8.4: Estimation summary with "var" as only class argument

Notice the difference with the output in Figure 8.2. Some information is missing in the output, such as the sample dates and the data name of the variables (switched to "y1", "y2" and "y3" by default). This is because the Results class has been instantiated only with the model as argument, and the latter does not contain this information. To obtain a complete output, one may want to include the optional argument: the "complementary_information" dictionary/structure. For instance, in Figure 8.4, four elements are missing: the sample dates, the data frequency, the estimation start date, and the variable names.

For Python we can create the following dictionary:

```
complementary_information = {}
complementary_information['sample_start'] = '1956-03-01'
complementary_information['sample_end'] = '2019-12-31'
complementary_information['frequency'] = 'quarterly'
complementary_information['estimation_start'] = '2025-04-14 10:33:39'
complementary_information['estimation_end'] = '2025-04-14 10:33:40'
complementary_information['endogenous_variables'] = ['ffrate','inf','gap']
```

Equivalently, in Matlab:

```
complementary_information = struct;
complementary_information.sample_start = '1956-03-01';
complementary_information.sample_end = '2019-12-31';
complementary_information.frequency = 'quarterly';
complementary_information.estimation_start = '2025-04-14 10:33:39';
complementary_information.estimation_end = '2025-04-14 10:33:40';
complementary_information.endogenous_variables = ["ffrate" "inf" "gap"];
```

Then, create the class instance with:

```
res = Results(var, complementary information)
```

Calling again the make_estimation_summary() and show_estimation_summary() methods will now produce Figure 8.2 instead of Figure 8.4.

The complementary_information element can take the following arguments. All of them are optional, i.e. you can include and omit any of them.

complementary_information:

argument	description
estimation_start	str: date at which model estimation starts
estimation_end	str: date at which model estimation completes
sample_start	str: sample start date
sample_end	str: sample end date
project_path	str: path to project folder
data_file	str: name of data file
frequency	str: sample data frequency
progress_bar	str: if 'True', indicates use of progress bar
create_graphics	str: if 'True', indicates creation of graphics
save_results	str: if 'True', indicates use of progress bar
forecast	str: if 'True', indicates computation of forecasts
forecast_credibility	str: credibility level for forecasts
conditional_forecast	str: if 'True', indicates computation of conditional forecasts
conditional_forecast_credibility	str: credibility level for conditional forecasts
irf	str: if 'True', indicates computation of impulse response function
irf_credibility	str: credibility level for impulse response function
fevd	str: if 'True', indicates computation of forecast error variance decomposition
fevd_credibility	str: credibility level for forecast error variance decomposition
hd	str: if 'True', indicates computation of historical decomposition
hd_credibility	str: credibility level for historical decomposition
forecast_periods	str: number of forecast periods
conditional_forecast_type	str: type of conditional forecasts
forecast_file	str: name of forecast data file
conditional_forecast_file	str: name of conditional forecast data file
forecast_evaluation	str: if 'True', indicates computation of forecast evaluation criteria
irf_periods	str: number of impulse response function periods
structural_identification	str: type of structural identification
structural_identification_file	str: name of structural identification data file

The following arguments are specific to the linear regression models:

complementary_information:

argument	description
endogenous_variables	str list: name of regression endogenous variable
exogenous_variables	str list: name of regression exogenous variables
dates	datetime list: list of in-sample dates
heteroscedastic_variables	str list: name of regression heteroscedastic variables
insample_fit	str: if 'True', indicates computation of in-sample fit applications
marginal_likelihood	str: if 'True', indicates computation of marginal likelihood
hyperparameter_optimization	str: if 'True', indicates use of hyperparameter optimization
optimization_type	str: type of hyperparameter optimization

The following arguments are specific to the vector autoregression models:

complementary_information:

argument	description
endogenous_variables	str list: name of VAR endogenous variables
exogenous_variables	str list: name of VAR exogenous variables
dates	datetime list: list of in-sample dates
forecast_dates	datetime list: list of forecast dates
conditional_forecast_dates	datetime list: list of conditional forecast dates
proxy_variables	str list: name of proxy variables for proxy-SVAR
insample_fit	str: if 'True', indicates computation of in-sample fit applications
constrained_coefficients	str: if 'True', indicates use of constrained coefficients
sums_of_coefficients	str: if 'True', indicates use of sums-of-coefficients
initial_observation	str: if 'True', indicates use of dummy initial observation
long_run	str: if 'True', indicates use of long-run prior
stationary	str: if 'True', indicates use of stationary prior
marginal_likelihood	str: if 'True', indicates computation of marginal likelihood
hyperparameter_optimization	str: if 'True', indicates use of hyperparameter optimization
coefficients_file	str: if 'True', indicates use of hyperparameter optimization
optimization_type	str: type of hyperparameter optimization
coefficients_file	str: name of constrained coefficient data file
long_run_file	str: name of long-run prior data file

8.3 Class methods

The Results class has the following methods:

■ **make_input_summary()**
computes input summary for the model, stored as "input_summary" attribute.
parameters:
- none
returns:
- none

■ **show_input_summary()**
display input_summary attribute in console.
parameters:
- none
returns:
- none

■ **save_input_summary(path)**
saves input_summary attribute on disk as txt file.
parameters:
- path: str, full path to folder where file is saved
returns:
- none

■ **make_estimation_summary()**
computes estimation summary for the model, stored as "estimation_summary" attribute.
parameters:
- none
returns:
- none

■ **show_estimation_summary()**
display estimation_summary attribute in console.
parameters:
- none
returns:
- none

■ **save_estimation_summary(path)**
saves estimation_summary attribute on disk as txt file.
parameters:
- path: str, full path to folder where file is saved
returns:
- none

■ **make_application_summary()**
computes application summary for the model, stored as "application_summary" attribute.
parameters:
- none
returns:
- none

■ **save_application_summary(path)**
saves application_summary attribute on disk as a series of csv files, one for each application.
parameters:
- path: str, full path to folder where file is saved
returns:
- none

8.4 A complete example

This final section illustrates the use of the Results class in a small Alexandria project on the fly. The example is provided in python code, but adaptation to Matlab is straightforward. Figure 8.5 shows the small piece of code:

Entrée [ ]:
```
# imports
from alexandria import NormalWishartBayesianVar
from alexandria import DataSets
from alexandria import Results
from datetime import datetime
import pandas as pd
```

Entrée [ ]:
```
# load data
ds = DataSets()
data = ds.load_islm()[:,:3]
```

Entrée [ ]:
```
# create and train model, compute impulse response function
var = NormalWishartBayesianVar(data, lags=2, structural_identification=2)
estimation_start = datetime.now()
var.estimate()
estimation_end = datetime.now()
irf, _ = var.impulse_response_function(20, 0.6)
```

Entrée [ ]:
```
# Results class: complementary information
complementary_information = {}
complementary_information['endogenous_variables'] = ['gdp','m3','rate']
complementary_information['sample_start'] = '1974-03-31'
complementary_information['sample_end'] = '2024-12-31'
complementary_information['estimation_start'] = estimation_start.strftime('%Y-%m-%d %H:%M:%S')
complementary_information['estimation_end'] = estimation_end.strftime('%Y-%m-%d %H:%M:%S')
complementary_information['frequency'] = 'quarterly'
complementary_information['irf'] = 'True'
complementary_information['irf_credibility'] = '0.6'
complementary_information['irf_periods'] = '20'
complementary_information['structural_identification'] = 'Cholesky'
```

Entrée [ ]:
```
# Results class: creation
res = Results(var, complementary_information)
```

Entrée [ ]:
```
# Results class: input summary
res.make_input_summary()
res.show_input_summary()
res.save_input_summary('D:\my_project')
```

Entrée [ ]:
```
# Results class: estimation summary
res.make_estimation_summary()
res.show_estimation_summary()
res.save_estimation_summary('D:\my_project')
```

Entrée [ ]:
```
# Results class: application summary
res.make_application_summary()
res.save_application_summary('D:\my_project')
```

Figure 8.5: Example code for the results class

The code is fairly straightforward: it imports the necessary modules, then loads the IS-LM toy dataset. It trains a Normal-Wishart Bayesian VAR, and computes impulse response functions. The next block creates the complementary_information dictionary, adding relevant information for the estimated model and its applications. The "res" object is then created form the Results class.

The three input summary methods are then used to generate, display and save the input summary. The console output is shown in Figure 8.6.

The next part of the code uses the three estimation summary to generate, display and save the estimation summary for the estimated model. The console output is shown in Figure 8.6.

The final part of the code uses the two application summary to generate and save the application summaries for the estimated model. The outcome for the impulse response function can be seen in Figure 8.3.

```
Estimation date:  2025-04-14 14:50:03


Model
---------

selected model: vector autoregression
endogenous variables: gdp, m3, rate
exogenous variables: none
data frequency: quarterly
estimation sample: 1974-03-31 2024-12-31
path to project folder: –
data file: –
progress bar: –
create graphics: –
save_results: –


Specification
----------------

VAR type: Normal-Wishart Bayesian VAR
iterations: 2000
credibility level: 0.95
constant: yes
trend: no
quadratic trend: no
lags: 2
AR coefficients: 0.95
pi1 (overall tightness): 0.1
pi3 (lag decay): 1
pi4 (exogenous slackness): 100
pi5 (sums-of-coefficients tightness): 1
pi6 (initial observation tightness): 0.1
pi7 (long-run tightness): 0.1
in-sample fit: –
sums-of-coefficients: no
dummy initial observation: no
long-run prior: no
stationary prior: no
marginal likelihood: –
hyperparameter optimization: no
long-run prior file: –


Applications
---------

forecast: –
credibility level, forecasts: –
conditional forecast: –
credibility level, conditional forecasts: –
impulse response function: True
credibility level, impulse response function: 0.6
forecast error variance decomposition: –
credibility level, forecast error variance decomposition: –
historical decomposition: –
credibility level, historical decomposition: –
forecast periods: –
conditional forecast type: –
forecast file: –
conditional forecast file: –
forecast evaluation: –
IRF periods: 20
structural identification: Cholesky
structural identification file: –
```

Figure 8.6: Console output for input methods

```
================================================================================
                        Normal-Wishart Bayesian Var
================================================================================
Sample:         1974-03-31  2024-12-31    Est. start:        2025-04-14 16:43:09
No. observations:                  202    Est. complete:     2025-04-14 16:43:09
Frequency:                   quarterly    Lags:                              2
================================================================================
                                Equation: gdp
--------------------------------------------------------------------------------
                            median        std dev         [0.025         0.975]
--------------------------------------------------------------------------------
VAR coefficients beta:
constant                     0.280          0.272         -0.253          0.814
gdp (-1)                     0.747          0.051          0.646          0.848
gdp (-2)                    -0.037          0.040         -0.115          0.041
m3 (-1)                      0.174          0.089         -0.001          0.349
m3 (-2)                     -0.119          0.086         -0.287          0.049
rate (-1)                    0.077          0.128         -0.173          0.327
rate (-2)                   -0.104          0.124         -0.347          0.140
--------------------------------------------------------------------------------
residual variance:            2.965    shock variance:                    1.000
================================================================================
                                Equation: m3
--------------------------------------------------------------------------------
                            median        std dev         [0.025         0.975]
--------------------------------------------------------------------------------
VAR coefficients beta:
constant                     0.219          0.126         -0.029          0.468
gdp (-1)                     0.020          0.024         -0.027          0.067
gdp (-2)                     0.009          0.019         -0.028          0.045
m3 (-1)                      1.111          0.042          1.029          1.192
m3 (-2)                     -0.169          0.040         -0.247         -0.091
rate (-1)                    0.020          0.059         -0.096          0.136
rate (-2)                   -0.002          0.058         -0.116          0.111
--------------------------------------------------------------------------------
residual variance:            0.641    shock variance:                    1.000
================================================================================
                                Equation: rate
--------------------------------------------------------------------------------
                            median        std dev         [0.025         0.975]
--------------------------------------------------------------------------------
VAR coefficients beta:
constant                    -0.103          0.085         -0.270          0.065
gdp (-1)                     0.045          0.016          0.014          0.077
gdp (-2)                     0.004          0.012         -0.020          0.029
m3 (-1)                      0.025          0.028         -0.030          0.079
m3 (-2)                     -0.007          0.027         -0.060          0.045
rate (-1)                    1.110          0.040          1.032          1.189
rate (-2)                   -0.136          0.039         -0.213         -0.060
--------------------------------------------------------------------------------
residual variance:            0.291    shock variance:                    1.000
================================================================================
                        Residual variance-covariance Sigma
--------------------------------------------------------------------------------
                gdp         m3       rate
gdp           2.965     -0.239      0.130
m3           -0.239      0.641      0.004
rate          0.130      0.004      0.291
--------------------------------------------------------------------------------
                  Structural shocks variance-covariance Gamma
--------------------------------------------------------------------------------
                gdp         m3       rate
gdp           1.000      0.000      0.000
m3            0.000      1.000      0.000
rate          0.000      0.000      1.000
--------------------------------------------------------------------------------
                        Structural identification matrix H
--------------------------------------------------------------------------------
                gdp         m3       rate
gdp           1.715      0.000      0.000
m3           -0.137      0.785      0.000
rate          0.074      0.018      0.530
================================================================================
```

Figure 8.7: Console output for estimation methods

CHAPTER 9

Alexandria graphics - documentation

Alexandria proposes the Graphics class as a convenience tool to create, vizualize and save graphics realised from your model applications. The class is especially useful if you use Alexandria on the fly as it permits to create the same graphics as the ones you would obtain by using the Graphical user Interface. This saves the effort of creating your own graphics from model attributes and outputs.

9.1 Example

The Graphics class makes it easy to generate both individual and joint plots of your model applications. Figure 9.1 illustrates the kind of input summary produced by the Result class, here for a simple forecast application.

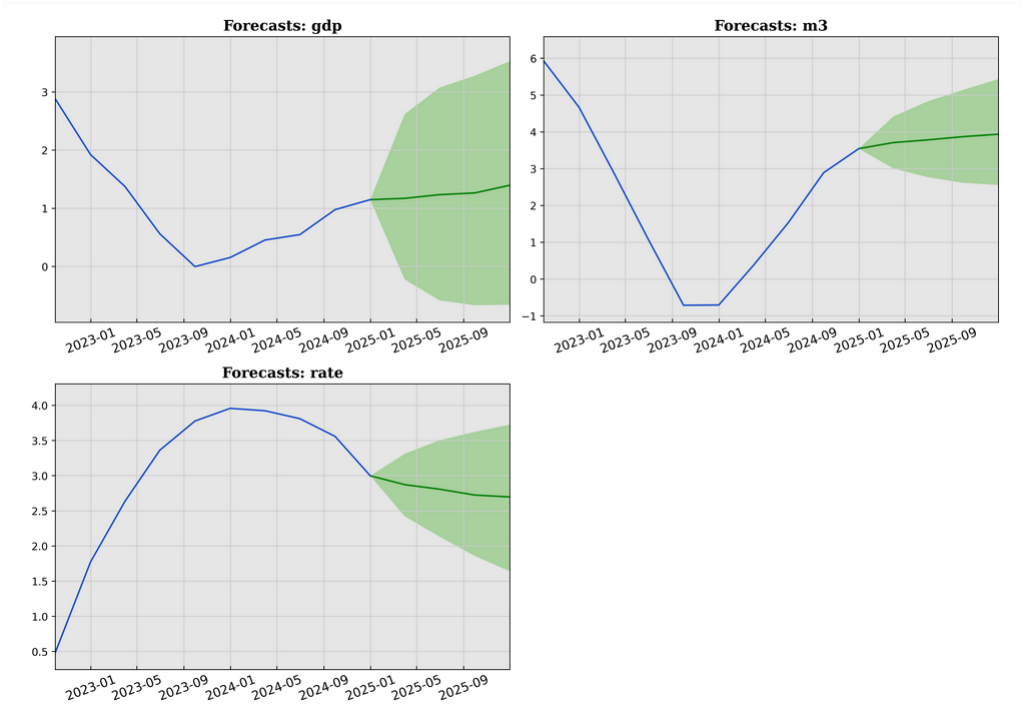


Figure 9.1: Example from the Graphics class

9.2 Class arguments

To use the Graphics class, you first need to create an instance of it. You can then use the different methods of the class to generate, vizualize and saves the different graphics outputs. The class takes four arguments when it is instantiated. A mandatory element: the estimated model. And three optional arguments: the complementary_information dictionary (in Python) or structure (in Matlab); the path argument, a string providing the path to your image folder; and the clear_folder argument, a boolean that clears your project folderthe contents of your image folder if set to True.

The mandatory element for the Graphics class is the model from which you want to generate the graphics. So for instance assume you just estimated a Normal-Wishart Bayesian VAR on the fly with the command:

```
var = NormalWishartBayesianVar(data)
```

From this model forecasts have been estimated:

```
forecasts = var.forecasts(4, 0.6)
```

The model "var" can then be used as an argument to initiate the Graphics class:

```
gpc = Graphics(var)
```

The object "gpc" is the instance of the Graphics class, which can then be used to run the methods of the class. For instance, to create graphics for the forecasts, use:

```
gpc.forecast_graphics(show = True, save = False)
```

The console output is displayed in Figure 9.2:

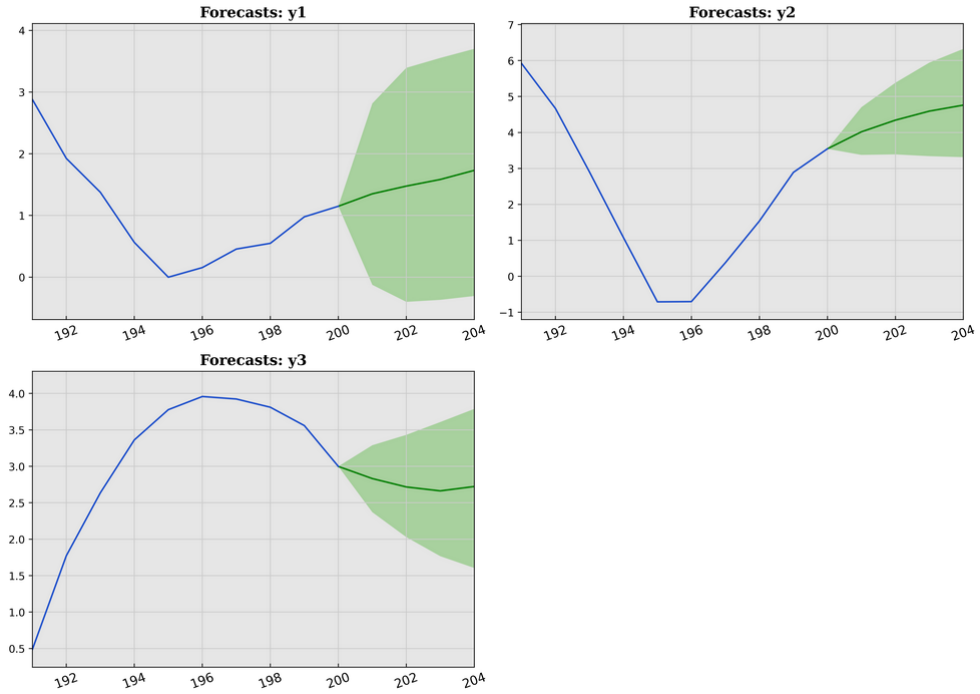


Figure 9.2: Example from the Graphics class

Notice the difference with the output in Figure 9.1. Some information is missing in the graphics, such as the sample dates (replaced by period number) and the data name of the variables (switched to "y1", "y2" and "y3" by default). This is because the Graphics class has been instantiated only with the model

as argument, and the latter does not contain this information. To obtain a complete output, one may want to include the optional argument: the "complementary_information" dictionary/structure. For instance, in Figure 9.2, three elements are missing: the sample dates, the forecast dates, and the variable names.

For Python we can create the following dictionary:

```
complementary_information = {}
complementary_information['endogenous_variables'] = ['ffrate','inf','gap']
complementary_information['dates'] = pd.date_range('1974-03-31',
                                     '2024-12-31', freq = 'Q')
complementary_information['forecast_dates'] = pd.date_range('2025-03-31',
                                     '2025-12-31',freq = 'Q')
```

Equivalently, in Matlab:

```
complementary_information = struct;
complementary_information.endogenous_variables = ["ffrate" "inf" "gap"];
complementary_information.dates =
                  datetime(1974,03,31):calmonths(3):datetime(2024,12,31);
complementary_information.forecast_dates =
                 datetime(2025,03,31):calmonths(3):datetime(2025,12,31);
```

Then, create the class instance with:

```
gpc = Graphics(var, complementary information)
```

Calling again the forecast_graphics() method will now produce Figure Figure 9.1 instead of 9.2.

The arguments accepted by the complementary_information element depends on the model estimated. All of them are optional, i.e. you can include and omit any of them.

The following arguments are specific to the linear regression models:

complementary_information:

argument	description
endogenous_variables	str list: name of regression endogenous variable
exogenous_variables	str list: name of regression exogenous variables
dates	datetime list: list of in-sample dates
forecast_dates	datetime list: list of forecast dates
y_p	ndarray: vector of realised values for forecast evaluation

The following arguments are specific to the vector autoregression models:

complementary_information:

argument	description
endogenous_variables	str list: name of VAR endogenous variables
exogenous_variables	str list: name of VAR exogenous variables
dates	datetime list: list of in-sample dates
forecast_dates	datetime list: list of forecast dates
conditional_forecast_dates	datetime list: list of conditional forecast dates
Y_p	ndarray: matrix of realised values for forecast evaluation
shocks	str list: name of structural shocks

The Graphics class takes two additional optional arguments. The first is the path argument. It is a string providing the path to the folder were images will be saved if you use the save option of the class methods. If you do not provide a value, the default value will be the current working directory.

The final argument is the clear_folder argument. It is a boolean set to False by default. If set to True, the contents of the folder designated by the "path" argument will be deleted when the Graphics object is created. This is handy if you want to make sure that you will not have in your project images from former estimations. However it is strongly recommended that you leave the argument to False if you are not sure that you want to clear the folder contents at each new use of the Graphics class.

9.3 Class methods

The Graphics class has the following methods:

■ **insample_fit_graphics(show, save)**
computes graphics for in-sample applications, including fitted, residuals, shocks and steady-state.
parameters:
- show: boolean, displays image if set to True.
- save: boolean, saves images in "path" folder if set to True.
returns:
- none

■ **forecast_graphics(show, save)**
computes graphics for forecasts.
parameters:
- show: boolean, displays image if set to True.
- save: boolean, saves images in "path" folder if set to True.
returns:
- none

■ **conditional_forecast_graphics(show, save)**
computes graphics for conditional forecasts.
parameters:
- show: boolean, displays image if set to True.
- save: boolean, saves images in "path" folder if set to True.
returns:
- none

- **irf_graphics(show, save)**
computes graphics for impulse response function.
parameters:
- show: boolean, displays image if set to True.
- save: boolean, saves images in "path" folder if set to True.
returns:
- none

- **fevd_graphics(show, save)**
computes graphics for forecast error variance decomposition.
parameters:
- show: boolean, displays image if set to True.
- save: boolean, saves images in "path" folder if set to True.
returns:
- none

- **hd_graphics(show, save)**
computes graphics for historical decomposition.
parameters:
- show: boolean, displays image if set to True.
- save: boolean, saves images in "path" folder if set to True.
returns:
- none

9.4 A complete example

This final section illustrates the use of the Graphics class in a small Alexandria project on the fly. The example is provided in python code, but adaptation to Matlab is straightforward. Figure 9.3 shows the small piece of code:

The code is fairly straightforward: it imports the necessary modules, then loads the IS-LM toy dataset. It trains a Normal-Wishart Bayesian VAR, and computes two applications: forecasts, and impulse response functions. The next block creates the complementary_information dictionary, adding relevant information for the estimated model and its applications. The "gpc" object is then created form the Graphics class.

The next block uses the forecast_graphics() method with show = True and save = False. This generates both individual and joint plots of forecasts for the model, displays them, but does not save them on the disk. The output for the joint plot is given by Figure 9.2.

The final block of the code uses the irf_graphics() method with show = True and save = False. The output for the joint plot is given by Figure 9.4.

Entrée [ ]:
```
# imports
from alexandria import NormalWishartBayesianVar
from alexandria import DataSets
from alexandria import Graphics
from datetime import datetime
import pandas as pd
```

Entrée [ ]:
```
# load data
ds = DataSets()
data = ds.load_islm()[:,:3]
```

Entrée [ ]:
```
# create and train model, compute impulse response function
var = NormalWishartBayesianVar(data, lags=2, structural_identification=2)
estimation_start = datetime.now()
var.estimate()
estimation_end = datetime.now()
forecasts = var.forecast(4, 0.6)
irf, _ = var.impulse_response_function(20, 0.6)
```

Entrée [ ]:
```
# Results class: complementary information
complementary_information = {}
complementary_information['endogenous_variables'] = ['gdp','m3','rate']
complementary_information['dates'] = pd.date_range('1974-03-31','2024-12-31',freq = 'Q')
complementary_information['forecast_dates'] = pd.date_range('2025-03-31','2025-12-31',freq = 'Q')
complementary_information['shocks'] = ['supply','money','demand']
```

Entrée [ ]:
```
# Graphics class: creation
gpc = Graphics(model = var, complementary_information = complementary_information, \
               path = 'D:\my_project', clear_folder = False)
```

Entrée [ ]:
```
# Graphics class: forecasts
gpc.forecast_graphics(show = True, save = False)
```

Entrée [ ]:
```
# Graphics class: forecasts
gpc.irf_graphics(show = True, save = False)
```

Figure 9.3: Example code for the graphics class

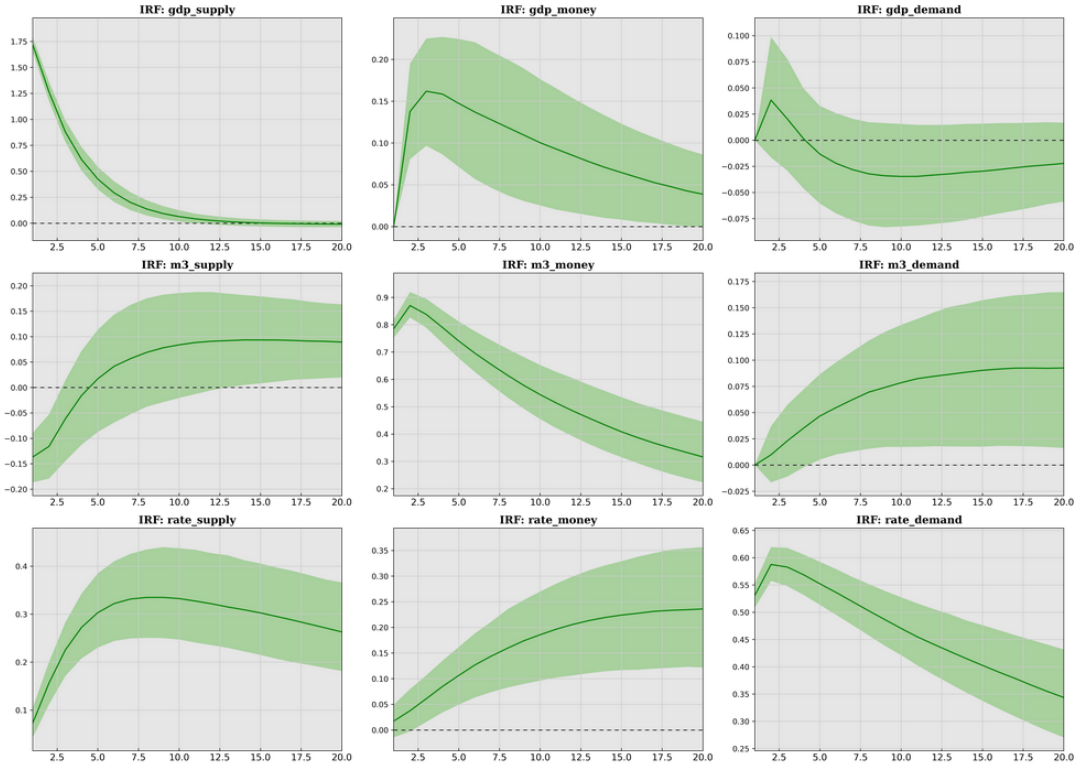


Figure 9.4: Example of impulse response function graphics

CHAPTER 10

Alexandria datasets - documentation

Alexandria comes with a few pre-built datasets that can be loaded directly into the working space without external files. These are mostly toy datasets proposed for pedagogical purposes, but they constitute interesting economic data sources on their own.

10.1 The Taylor rule dataset

Description:

A dataset for the estimation of a Taylor rule for the Unite States.

Characteristics:

observations	264
frequency	quarterly
start date	1955q1
end date	2020q4
variables	3
variable description	**ffrate**: federal funds rate
	inf: year-to-year inflation
	gap: output gap, as percentage deviation from potential output

This is the dataset used in sections 9.8 and 10.4 of the textbook. It can be called with the following functions:

Example (Python):

```
# imports: required to use the load functions
from alexandria import DataSets

# load Taylor dataset as raw numpy array, with numeric data only
ds = DataSets()
taylor_data = ds.load_taylor()

# or load Taylor dataset as pandas dataframe, with dates and variable names
taylor_table = ds.load_taylor_table()
```

Example (Matlab):

```
% load Taylor dataset as regular matrix, with numeric data only
ds = DataSets();
taylor_data = ds.load_taylor();

% load Taylor dataset as Matlab table, with dates and variable names
taylor_table = ds.load_taylor_table();
```

10.2 The IS-LM dataset

Description:

A dataset for a simple IS-LM model for the Euro area.

Characteristics:

observations	204
frequency	quarterly
start date	1974q1
end date	2024q4
variables	6
variable description	**gdp**: year-to-year real GDP growth rate **m3**: year-to-year growth of broad money M3 **rate**: 3-month interest rate **cpi**: year-to-year growth of harmonized CPI **proxy_supply**: supply shock proxy, as quarterly change in commodity prices **proxy_demand**: real demand shock proxy, as quarterly change in consumer opinion surveys

This is the dataset used in sections 13.6 and 14.6 of the textbook. It can be called with the following functions:

Example (Python):

```
# imports: required to use the load functions
from alexandria import DataSets

# load IS-LM dataset as raw numpy array, with numeric data only
ds = DataSets()
islm_data = ds.load_islm()

# or IS-LM dataset as pandas dataframe, with dates and variable names
islm_table = ds.load_islm_table()
```

Example (Matlab):

```
% load IS-LM dataset as regular matrix, with numeric data only
ds = DataSets();
islm_data = ds.load_taylor();

% load IS-LM dataset as Matlab table, with dates and variable names
islm_table = ds.load_taylor_table();
```