Quantitative Macroeconomics

Winter 2023/24

Week 1

Willi Mutschler
willi@mutschler.eu

Version: 1.0
Latest version available on: GitHub

Contents

1. What is Quantitative Macroeconomics

Broadly define the scope and research topics of "Quantitative Macroeconomics", sometimes referred to as "Macroeconometrics". What are

- **S**tructural **V**ector **A**uto**R**egressive models?
- **D**ynamic **S**tochastic **G**eneral **E**quilibrium models?
- the challenges and approaches in estimating SVAR and DSGE models?

Readings

- Cantore et al. (2013)
- Christiano, Eichenbaum, and Trabandt (2018)
- Fernández-Villaverde and Rubio-Ramírez (2010)
- Guerrón-Quintana and Nason (2013)
- Herbst and Schorfheide (2016, Ch. 1)
- Kilian (2013)
- Kilian and Lütkepohl (2017, Ch. 1, Ch. 6)
- Schorfheide (2010)

2. Programming Languages

1. Name some popular programming languages in Macroeconomics. Which are general purpose, which are domain specific?
2. What are the differences between compiled and interpreted languages?
3. What is important when choosing a programming language for scientific computing in Macroeconomics? Which programming language(s) would you choose?

Readings

- Aguirre and Danielsson (2020)
- Aruoba and Fernández-Villaverde (2015) (note the Update available at `https://www.sas.upenn.edu/~jesusfv/Update_March_23_2018.pdf`)

3. Quick Tour: MATLAB

Install the most recent version of MATLAB with the following Toolboxes: Econometrics Toolbox, Global Optimization Toolbox, Optimization Toolbox, Parallel Computing Toolbox, Statistics and Machine Learning Toolbox, Symbolic Math Toolbox. Open a new script and do the following: [1]

1. Define the column vectors
$$x = (-1, 0, 1, 4, 9, 2, 1, 4.5, 1.1, -0.9)' \qquad y = (1, 1, 2, 2, 3, 3, 4, 4, 5, \text{nan})'$$
2. Check if both vectors have the same length using either `length()` or `size()`.
3. Perform the following logical operations:
$$x < y \qquad x < 0 \qquad x + 3 \geq 0 \qquad y < 0$$
4. Check if all elements in x satisfy both $x + 3 \geq 0$ and $y > 0$.
5. Check if all elements in x satisfy either $x + 3 \geq 0$ or $y > 0$.
6. Check if at least one element of y is greater than 0.
7. Compute $x + y$, xy, xy', $x'y$, y/x, and x/y.
8. Compute the element-wise product and division of x and y.
9. Compute $\ln(x)$ and e^x.
10. Use `any` to check if the vector x contains elements satisfying $\sqrt{x} \geq 2$.
11. Compute $a = \sum_{i=1}^{10} x_i$ and $b = \sum_{i=1}^{10} y_i^2$. Omit the nan in y when computing the sum.
12. Compute $\sum_{i=1}^{10} x_i y_i^2$. Omit the nan in y when computing the sum.
13. Count the number of elements of $x > 0$.
14. Predict what the following commands will return:
```
x.^y   x.^(1/y)   log(exp(y))   y*[-1,1]   x+[-1,0,1]   sum(y*[-1,1],1,'omitnan')
```
15. Define the matrix $X = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$. Print the transpose, dimensions and determinant of X.
16. Compute the trace of X (i.e. the sum of its diagonal elements).
17. Change the diagonal elements of X to [7,8,9].
18. Compute the eigenvalues of (the new) X. Display a message if X is positive or negative definite.
19. Invert X and compute the eigenvalues of X^{-1}.
20. Define the column vector $a = (1, 3, 2)'$ and compute `a'*X`, `a'.*X`, and `X*a`.
21. Compute the quadratic form $a'Xa$.

[1] If you don't have any previous programming experience, I would highly recommend to go through Brandimarte (2006, Appendix A), Miranda and Fackler (2002, Appendix B), and Pfeifer (2017).

22. Define the matrices $I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, $Y = \begin{bmatrix} 1 & 4 & 7 & 1 & 0 & 0 \\ 2 & 5 & 8 & 0 & 1 & 0 \\ 3 & 6 & 9 & 0 & 0 & 1 \end{bmatrix}$ and $Z = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$.

23. Generate the vectors

$$x_1 = (1, 2, 3, \ldots, 9), \quad x_2 = (0, 1, 0, 1, 0, 1, 0, 1), \quad x_3 = (1, 1, 1, 1, 1, 1, 1, 1)$$
$$x_4 = (-1, 1, -1, 1, -1, 1), \quad x_5 = (1980, 1985, 1990, \ldots, 2010), \quad x_6 = (0, 0.01, 0.02, \ldots, 0.99, 1)$$

using sequence operator `:` or `repmat`.

24. Generate a grid of $n = 500$ equidistant points on the interval $[-\pi, \pi]$ using `linspace`.

25. Compare `1:10+1`, `(1:10)+1` and `1:(10+1)`.

26. Define the following column vectors

$$x = \begin{pmatrix} 1 & 1.1 & 9 & 8 & 1 & 4 & 4 & 1 \end{pmatrix}, \quad y = \begin{pmatrix} 1 & 2 & 3 & 4 & 4 & 3 & 2 & \text{NaN} \end{pmatrix}'$$
$$z = \begin{pmatrix} true & true & false & false & true & false & false & false \end{pmatrix}'$$

27. Predict what the following commands will return (and then check if you are right):
`x(2:5)`, `x(4:end-2)`, `x([1 5 8])`, `x(repmat(1:3,1,4))`,
`y(z)`, `y(~z)`, `y(x>2)`, `y(x==1)`,
`x(~isnan(y))`, `y(~isnan(y))`

28. Indexing is not only used to read certain elements of a vector but also to change them. Execute `x2 = x` to make a copy of `x`. Change all elements of `x2` that have the value 4 to the value -4. Print `x2`.

29. Change all elements of `x2` that have the value 1 to a missing value (`nan`). Print `x2`.

30. Execute `x2(z) = []`. Print `x2`.

31. Define the matrix $M = \begin{pmatrix} 1 & 5 & 9 & 12 & 8 & 4 \\ 2 & 6 & 10 & 11 & 7 & 3 \\ 3 & 7 & 11 & 10 & 6 & 2 \\ 4 & 8 & 12 & 9 & 5 & 1 \end{pmatrix}$ using the : operator and the `reshape` command.

32. Predict what the following commands will return (and then check if you are right):
`M(1,3)`, `M(:,5)`, `M(2,:)`, `M(2:3,3:4)`, `M(2:4,4)`, `M(M>5)`, `M(:,M(1,:)<=5)`, `M(M(:,2)>6,:)`,
`M(M(:,2)>6,4:6)`

33. Print all rows of M where column 5 is at least three times larger than column 6.

34. Count the number of elements of M that are larger than 7.

35. Count the number of elements of M in row 2 that are smaller than their neighbors in row 1.

36. Count the number of elements of M that are larger than their left neighbor.

Readings

- Brandimarte (2006, Appendix A).
- Miranda and Fackler (2002, Appendix B).
- Pfeifer (2017)

4. Quick Tour: git with GitKraken

1. What is *git*? What is *GitKraken*? What is *GitHub*?

2. How does a typical workflow in Quantitative Macroeconomics look like? How can a version control system like git support this workflow?

3. If you haven't already, sign up for a free account on GitHub: `https://github.com/signup`

4. If you haven't already, sign up for the GitHub Student Developer Pack `https://education.github.com/pack`. If you add your university email address, the decision process is very fast (usually within a day).

5. Install GitKraken for your operating system `https://www.gitkraken.com/download` and connect it with GitHub. Go to preferences → Integrations → GitHub and click *Generate SSH key and add to GitHub*. Checkout the other preferences. For instance, if you are on Windows make sure that under Editor the *End of Line Character* is set to *LF*.[2]

6. Create a local repository on your computer called *Quant-Macro* by selecting `Start a local repo` in GitKraken.

7. With your favorite text editor open the `README.md` file inside the folder and add the following:

   ```
   # Repository for Quantitative Macroeconomics
   In this repository I will practice estimating SVAR and DSGE models.
   The folder contains examples and codes developed in the lecture.
   I also don't get git and find this really cumbersome! Dropbox, iCloud, Nextcloud, and OneDrive is so much better!
   ```

   Save the file and have a look into GitKraken what happened in your repository.

8. Explain the *git model* of staging, committing and pushing.

9. Stage and commit all the changes you made so far to the repository.

10. What is a *good commit*?

11. *Soft Reset* your current status (so-called HEAD) to the initial commit. Re-commit everything except the last line in the README.md file without actually changing the file in your text editor. Put the remaining changes (i.e. the last line into the stash, which is a place for work-in-progress).

12. Push your local repository to GitHub. Visit the website https://github.com/[username]/Quant-Macro to see what happened.

13. What are *git branches*? Create a branch called *latex-exam-template* and checkout the branch.

14. While on your *latex-exam-template* branch, use your favorite text editor to create the three files `templateExamSolution.tex`, `templateExamBiblio.bib`, and `templateMatlabExample.m` given in the Appendix. Typically, when copy and pasting from a pdf you will run into characters (like minus signs and empty space) not correctly pasted into your text editor, so you will need to check everything manually. Once you are happy, commit only three files using the message "Created latex template for exam solutions".

15. Compile the Latex files using your favorite Latex GUI or via the command line:

[2] If you are on Windows and working with people who are not (or vice-versa), you'll probably run into line-ending issues at some point. This is because Windows uses both a carriage-return character and a linefeed character for newlines in its files, whereas macOS and Linux systems use only the linefeed character. This is a subtle but incredibly annoying fact of cross-platform work; many editors on Windows silently replace existing LF-style line endings with CRLF, or insert both line-ending characters when the user hits the enter key. Therefore, it is advised to tell git to convert CRLF to LF on commit but not the other way around.

```
pdflatex templateExamSolution
biber templateExamSolution
pdflatex templateExamSolution
pdflatex templateExamSolution
```

Open the pdf to make sure the file compiled correctly.

16. Like in the previous exercise we often have some auxiliary files created by a program which we do not care about. We can tell *git* to ignore individual files or patterns of files. In GitKraken, simply right-click on the file you want to ignore and select how you want to ignore it. Do so for the just created auxiliary Latex files, but also for the generated pdf (as it is binary). Have a look into the newly created `.gitignore` file and commit it.[3]

17. Create a Pull Request to GitHub of your develop branch to your main branch. Go to GitHub and accept this. See what happens in the repository.

18. Switch back to your main branch and pull the merged changes. Then click Pop to add the changes you had in your stash back on top of the current branch. Either decide to discard the last line in the README.md file or commit it and schedule a meeting with me to discuss your issues.

19. Once you got accepted to the GitHub Student Developer Pack, activate GitKraken to get the free PRO license.

Readings

- `https://www.gitkraken.com/learn/git`: highly recommended (particularly the videos)
- Chacon (2014): extensive book on git

Useful urls

- `https://www.sas.upenn.edu/~jesusfv/Chapter_HPC_5_Git.pdf`
- `https://github.com/jaredgars/LEAP`
- `https://luispfonseca.com/files/slides_git.pdf`
- `https://www.frankpinter.com/notes/git-for-economists-presentation.pdf`
- `https://github.com/fditraglia/git-for-economists`
- `https://matteosostero.com/files/slides_git.pdf`

[3] GitHub also has a collection of useful gitignore files for various programming languages at `https://github.com/github/gitignore`.

References

Aguirre, Alvaro and Jon Danielsson (2020). *Which Programming Language Is Best for Economic Research: Julia, Matlab, Python or R?* Blog. URL: https://voxeu.org/article/which-programming-language-best-economic-research.

Aruoba, S. Borağan and Jesús Fernández-Villaverde (2015). "A Comparison of Programming Languages in Macroeconomics". In: *Journal of Economic Dynamics and Control* 58, pp. 265–273. DOI: 10.1016/j.jedc.2015.05.009.

Brandimarte, Paolo (2006). *Numerical Methods in Finance and Economics: A MATLAB-based Introduction*. 2nd ed. Statistics in Practice. Hoboken, N.J: Wiley Interscience. ISBN: 978-0-471-74503-7.

Cantore, Cristiano et al. (2013). "The Science and Art of DSGE Modelling: I – Construction and Bayesian Estimation". In: *Handbook of Research Methods and Applications in Empirical Macroeconomics*. Edward Elgar Publishing, pp. 411–440. ISBN: 978-0-85793-102-3. DOI: 10.4337/9780857931023.00026.

Chacon, Scott (2014). *Pro Git*. Second edition. The Expert's Voice in Software Development. New York, NY: Apress. ISBN: 978-1-4842-0077-3.

Christiano, Lawrence J., Martin S. Eichenbaum, and Mathias Trabandt (2018). "On DSGE Models". In: *Journal of Economic Perspectives* 32.3, pp. 113–140. DOI: 10.1257/jep.32.3.113.

Fernández-Villaverde, Jesús and Juan F. Rubio-Ramírez (2010). "Structural Vector Autoregressions". In: *Macroeconometrics and Time Series Analysis*. Ed. by Steven N. Durlauf and Lawrence E. Blume. London: Palgrave Macmillan UK, pp. 303–307. ISBN: 978-0-230-23885-5 978-0-230-28083-0. URL: https://doi.org/10.1057/9780230280830_33.

Guerrón-Quintana, Pablo A. and James M. Nason (2013). "Bayesian Estimation of DSGE Models". In: *Handbook of Research Methods and Applications in Empirical Macroeconomics*. Edward Elgar Publishing, pp. 486–512. ISBN: 978-0-85793-102-3. URL: https://doi.org/10.4337/9780857931023.00029.

Herbst, Edward and Frank Schorfheide (2016). *Bayesian Estimation of DSGE Models*. The Econometric and Tinbergen Institutes Lectures. Princeton University Press. ISBN: 978-0-691-16108-2.

Kilian, Lutz (2013). "Structural Vector Autoregressions". In: *Handbook of Research Methods and Applications in Empirical Macroeconomics*. Ed. by Steven N. Hashimzade and Michael Thornton. Edward Elgar Publishing, pp. 515–554. URL: https://doi.org/10.4337/9780857931023.00031.

Kilian, Lutz and Helmut Lütkepohl (2017). *Structural Vector Autoregressive Analysis*. Themes in Modern Econometrics. Cambridge: Cambridge University Press. ISBN: 978-1-107-19657-5. URL: https://doi.org/10.1017/9781108164818.

Lucas, Robert E. (Jan. 1976). "Econometric Policy Evaluation: A Critique". In: *Carnegie-Rochester Conference Series on Public Policy* 1, pp. 19–46. DOI: 10.1016/S0167-2231(76)80003-6.

Miranda, Mario Javier and Paul L. Fackler (2002). *Applied Computational Economics and Finance*. Cambridge, Mass. London: MIT. ISBN: 978-0-262-63309-3.

Pfeifer, Johannes (2017). *MATLAB Handout*. URL: https://sites.google.com/site/pfeiferecon/dynare.

Schorfheide, Frank (2010). "Bayesian Methods in Macroeconometrics". In: *Macroeconometrics and Time Series Analysis*. Ed. by Steven N. Durlauf and Lawrence E. Blume. London: Palgrave Macmillan UK, pp. 28–34. ISBN: 978-0-230-23885-5 978-0-230-28083-0. URL: https://doi.org/10.1057/9780230280830_3.

Sims, Christopher A. (Jan. 1980). "Macroeconomics and Reality". In: *Econometrica* 48.1, p. 1. DOI: 10.2307/1912017.

A. Exam template files for LaTeX

progs/latex/templateExamSolution.tex

```
% !TeX encoding = UTF-8
% !TeX spellcheck = en_US
\documentclass[a4paper]{scrartcl}
\usepackage[T1]{fontenc}
%\usepackage[utf8]{inputenc}
\usepackage[english]{babel} \usepackage[bottom=2.5cm,top=2.0cm,left=2.0cm,right=2.0cm]{geometry}
\usepackage{amssymb,amsmath,amsfonts}
\usepackage{lmodern}
\usepackage{graphicx}
\usepackage{csquotes}
\usepackage[usenames,dvipsnames]{xcolor}
\definecolor{mygreen}{rgb}{0,0.4,0}
\definecolor{mygray}{rgb}{0.2,0.2,0.2}
\usepackage[numbered,framed]{matlab-prettifier}
\usepackage[backend=biber,style=authoryear]{biblatex}
\addbibresource{templateExamBiblio.bib}

\begin{document}
\title{Quantitative Macroeconomics\\Midterm Exam}
\author{Willi Mutschler\\Student ID: 123\\willi@mutschler.eu}
\date{Version: \today}
\maketitle\thispagestyle{empty}

\newpage
\tableofcontents\thispagestyle{empty}\newpage \setcounter{page}{1}

\section{Exercise 1}\label{sec:introduction}
This is a \LaTeX template you might find useful to hand in your exam.

\section{Tables}
Table \ref{tbl:1} is an example of a table.
\begin{table}[h!]
  \centering
  \begin{tabular}{|l|c|r|}
  \hline
  \multicolumn{3}{|c|}{Country List} \\
  \hline
  Country Name or Area Name& ISO ALPHA 2 Code &ISO ALPHA 3 \\ \hline
  Albania &AL & ALB \\
  Algeria &DZ & DZA \\
  American Samoa & AS & ASM \\
  Angola & AO & AGO \\
  \hline
  \end{tabular}
  \caption{This is the caption for the example table.} \label{tbl:1}
\end{table}

\section{Figures}
Table \ref{fig:1} is an example of a figure, remove the draft option to actually print it.
\begin{figure}[t!]\centering
  \includegraphics[draft,width=0.5\textwidth]{mycoolplot.pdf}
  \caption{This is the caption for the example figure.}
  \label{fig:1}
\end{figure}

\section{Math}
\begin{equation}
e^{\pi i} + 1 = 0\label{eq:euler}
\end{equation}

The beautiful equation \eqref{eq:euler} is known as the Euler equation. We can also align equations:
\begin{align*}
x&=y           &  w &=z              &  a&=b+c\\
2x&=-y & 3w&=\frac{1}{2}z & -4 + 5x&=2+y & w+2&=-1+w &
a&=b\\
ab&=cb
\end{align*}
Inline math works like this $x_t=A x_{t-1}+ \varepsilon_t$ where $\varepsilon_t \overset{iid}\sim \underbrace{N(\
     underset{3 \times 1}{\boldsymbol{0}},\boldsymbol{\Sigma})}_{\text{normally distributed}}$.
We can also break long lines:
```

```
\begin{multline}
p(x) = 3x^6 + 14x^5y + 590x^4y^2 + 19x^3y^3
\\
— 12x^2y^4 — 12xy^5 + 2y^6 — a^3b^3 \label{eq:long}
\end{multline}
Equation \eqref{eq:long} is a long equation.
Or group and center lines:
\begin{gather*}
y_t = C x_{t}\\
x_t = A x_{t—1} + Bu_t
\end{gather*}

\section{Displaying code}
Use \texttt{lstlisting} to display code directly:
\begin{lstlisting}[
style = Matlab—editor, basicstyle = \mlttfamily,
]
x = reshape(eye(3,3),3*3,1);
dlyap(A,RHS);
\end{lstlisting}
or load it from a file with \texttt{lstinputlisting}
\lstinputlisting[
        style = Matlab—editor,
        basicstyle = \mlttfamily,
        title=\lstname,
  ]{templateMatlabExample.m}
Note that the pretty formatting for MATLAB is achieved by loading the package \texttt{matlab—prettifier}.

\section{Citations}\label{sec:citations}
Examples how to do citations:
\begin{itemize}
  \item \textcite{Sims_1980_MacroeconomicsReality} shows that SVAR models can be used to study the transmission
       channel of monetary policy shocks.
  \item The book \textcite{Herbst.Schorfheide_2016_BayesianEstimationDSGE} emphasizes that DSGE models are usually
       estimated by Bayesian methods.
  \item Identification plays an important role in Quantitative Macroeconomics \parencite{Kilian_2013_
       StructuralVectorAutoregressions,Mutschler_2022_QuantitativeMacroeconomics}.
\end{itemize}
Now let's print the bibliography.
\printbibliography

\end{document}
```

progs/latex/templateExamBiblio.bib

```
@book{Herbst.Schorfheide_2016_BayesianEstimationDSGE,
  title = {Bayesian {{Estimation}} of {{DSGE Models}}},
  author = {Herbst, Edward and Schorfheide, Frank},
  year = {2016},
  series = {The {{Econometric}} and {{Tinbergen Institutes Lectures}}},
  publisher = {{Princeton University Press}},
  isbn = {978—0—691—16108—2},
}

@incollection{Kilian_2013_StructuralVectorAutoregressions,
  title = {Structural Vector Autoregressions},
  booktitle = {Handbook of {{Research Methods}} and {{Applications}} in {{Empirical Macroeconomics}}},
  author = {Kilian, Lutz},
  editor = {Hashimzade, Steven N. and Thornton, Michael},
  year = {2013},
  pages = {515——554},
  publisher = {{Edward Elgar Publishing}},
  doi = {10.4337/9780857931023.00031},
}

@misc{Mutschler_2022_QuantitativeMacroeconomics, title = {Quantitative Macroeconomics},
  author = {Mutschler, Willi},
  year = {2022},
  publisher = {GitHub},
  journal = {GitHub repository},
  howpublished = {\url{https://github.com/wmutschl/Quantitative—Macroeconomics}},
  commit = {0cfbd4fa7f002d783dd8eed1616e48042043176e}
}

@article{Sims_1980_MacroeconomicsReality, title = {Macroeconomics and {{Reality}}},
```

```
  author = {Sims, Christopher A.},
  year = {1980},
  journal = {Econometrica}, volume = {48},
  number = {1——48},
  pages = {1},
  doi = {10.2307/1912017},
}
```

progs/matlab/templateMatlabExample.m

```
x = reshape(eye(3,3),3*3,1);
dlyap(A,RHS);
```

B. Solutions

1 Solution to What is Quantitative Macroeconomics Quantitative macroeconomics combines (i) modern theoretical macroeconomics (the study of aggregated variables such as economic growth, unemployment and inflation by means of structural macroeconomic models) with (ii) state-of-the-art econometric methods (the application of formal statistical methods in empirical economics). To this end, we will focus on the two workhorse model frameworks, namely SVAR and DSGE, and develop the numerical procedures and algorithms required to estimate such models. Very important: this course is **NOT ABOUT HOW TO BUILT** structural models, but it is **ABOUT HOW TO ESTIMATE** such models using real data.

This enables us to look at abstract macroeconomic concepts, for example:

- Is uncertainty a source of business cycles or an endogenous response to them, and does the type of uncertainty matter? What are the effects on the real price of oil and on macroeconomic aggregates of oil supply shocks, shocks to the global demand for all industrial commodities, or demand shocks that are specific to the crude oil market? That is, we want to understand the propagation of shocks by means of an **impulse response analysis**. In other words, we want to **quantify** the dynamic effect of identified and economic meaningful shocks over time. For example:

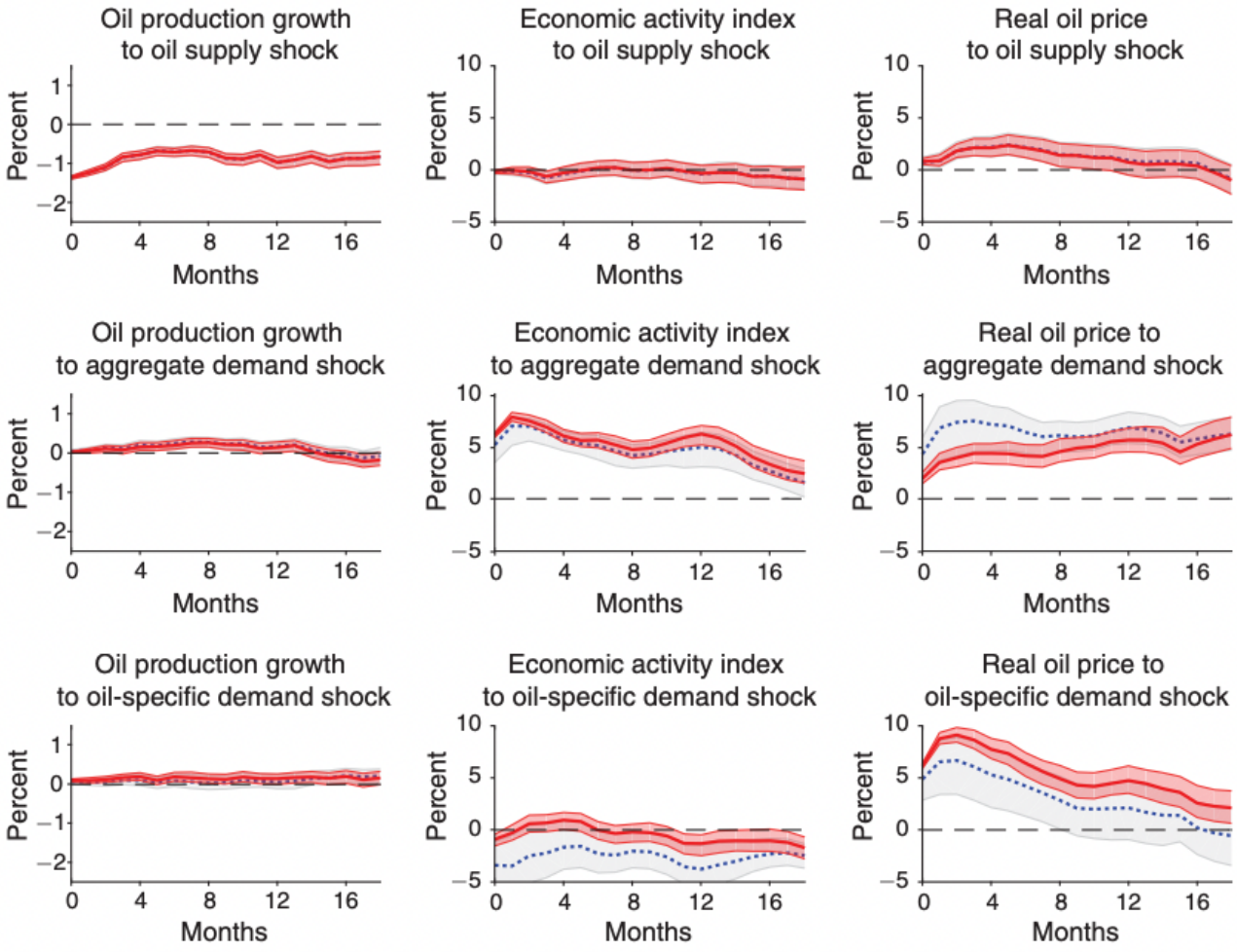
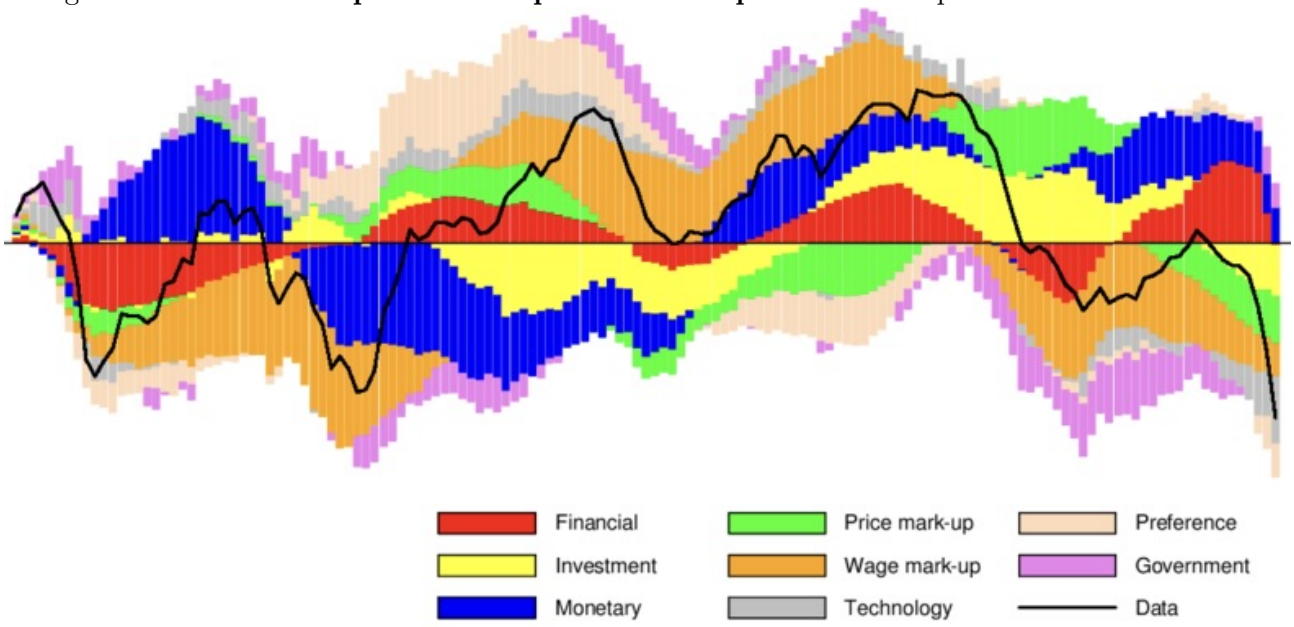


FIGURE 2. IRFS WITH AND WITHOUT NARRATIVE SIGN RESTRICTIONS

- How important are monetary policy shocks as opposed to other shocks for movements in aggregate output? That is, we want to identify sources of fluctuations by means of a **variance decomposition**. In other words, we want to **quantify** how important a shock is on explaining the variation in the endogenous variables **on average**. For example:

	Shocks				
	Monetary	Technology	Preference	Price markup	Wage markup
Output growth	4.8%	22.3%	57.1%	8.0%	7.1%
Inflation	27.1%	6.1%	36.3%	13.7%	14.7%
Nominal rate	5.0%	0.4%	72.3%	9.8%	11.8%
Hours	0.4%	0.8%	70.0%	17.6%	9.6%
Wage	0.1%	0.1%	73.6%	12.0%	12.8%

- Has monetary policy changed in the early 1980s? Has it caused the 1982 recession? That is, we want to study how much a given structural shock explains the historically observed fluctuations in aggregate variables at every given point in time by means of a **historical decomposition**. In other words, we want to **quantify** how important a shock was in driving the behavior of the endogenous variables in a **specific time period in the past**. For example:



- To what extent would the financial crises have deepened if monetary policy had not responded to the output contraction at all? That is, we want to study **counterfactual policies**.
- How will inflation, growth and unemployment evolve over the next eight quarters? That is, we want to predict the values of variables in the future by means of a **forecast**.

Traditional Cowles Commission approach The prevailing econometric framework was initiated and developed in the 1950s to 1970s by the Cowles Commission by inventing many things such as indirect least squares, instrumental variable methods, the full and the limited information maximum likelihood methods. Broadly speaking they developed the theory and toolkits (based on linear regressions) for estimating large-scale system of equations that have a certain structure that enables one to estimate these simultaneous equations one-by-one. The IS-LM or AS-AD models are examples of such systems, where we have some underlying ad-hoc assumptions (e.g. consumption is always equal to a fraction of current output) that give us a reduced-form from which we can compute things like fiscal policy multipliers. This reduced-form also enables us to estimate these parameters. Of course, these models are useful for gaining intuition, but are quite limited for actual **quantitative** policy advise and analysis of how the whole economy responds to particular shocks or policies. Nevertheless, these models were (and still are[4]) empirically somewhat successful when measured in terms of forecast quality. However, as the ultimate goal in macroeconomics is to gain policy insight and to actually understand the

[4]More elaborated versions of such models are still in use at central banks and policy institutions under the label of semi-structural models.

transmission channels, this approach came under heavy attack in the mid 1970s by (i) Lucas (1976) and (ii) Sims (1980):

- Sims critique: many restrictions that are used to identify behavioral equations in these models are inconsistent with dynamic macroeconomic theories → (structural) vector autoregressions ((S)VAR)
- Lucas critique: models are unreliable for policy analysis because they are unable to predict effects of policy regime changes on the expectation formation of economic agents in a coherent manner → development of micro-founded structural models is necessary

SVAR approach Pioneered by Christopher Sims (1980). The basic idea is to model all endogenous variables JOINTLY rather than one equation at a time. This breaks with the traditional so-called Cowles Commission approach that relies on ad-hoc assumptions to do equation-by-equation estimation, which in essence generates implausible exogeneity assumptions (think about estimating the IS-LM model). The underlying statistical framework is the vector autoregressive model, i.e. we do multivariate linear regressions of a vector of observables on its own lags and (possibly) other variables such as a constant, trends or exogenous variables. Importantly, we have cross-equation constraints that are jointly modeled and estimated. This is the VAR framework.

Now adding the **S** to **VAR** means to add structure on the cross-equation restrictions, in order to differentiate between correlation and causation. The basic idea is to decompose the forecast errors (i.e. the residuals) into so-called structural shocks that are mutually uncorrelated and have an economic interpretation (supply shocks, demand shocks, risk shocks, monetary policy shocks, rare disaster shocks, etc.). More precisely, instead of given every parameter in the VAR model a behavioral interpretation, we make explicit identifying assumptions to isolate estimates of policy and/or private agent's behavior and its effects on the economy while keeping the model free of many additional restrictive assumptions. In principle, this enables us to analyze the causal effects of these structural shocks on the model variables by all kinds of objectives. But of course, it all depends on the validity of the identifying assumptions. Typically, these stem from institutional knowledge, economic theory or other extraneous constraints on the model responses. Most of the recent literature is primarily concerned with finding new and more compelling identifying assumptions (or methods), in order to consolidate or challenge known results. The upside of the SVAR approach is that ideally, with very few assumptions about the structure of the economy, we get meaningful and, above all, data-based results.

DSGE approach DSGE models use modern macroeconomic theory to explain and predict co-movements of aggregate time series. DSGE models start from what we call the micro-foundations of macroeconomics (i.e. to be consistent with the underlying behavior of economic agents), with a heart based on the rational expectation forward-looking economic behavior of agents. The dynamic equilibrium is the result from the combination of economic decisions taken by all economic agents. For example the following agents or sectors are commonly included:

- Households: benefit from private consumption, leisure and possibly other things like money holdings or state services; subject to a budget constraint in which they finance their expenditures via (utility-reducing) work, renting capital and buying (government) bonds ↪ maximization of utility
- Firms produce a variety of products with the help of rented equipment (capital) and labor. They (possibly) have market power over their product and are responsible for the design, manufacture and price of their products. ↪ cost minimization or profit maximization
- Monetary policy follows a feedback rule, so-called Taylor rule, for instance: nominal interest rate reacts to deviations of the current (or lagged) inflation rate from its target and of current output from potential output

- Fiscal policy (the government) collects taxes from households and companies in order to finance government expenditures (possibly utility-enhancing) and government investment (possibly productivity-enhancing). In addition, the government can issue debt securities and might face a probability of sovereign default.
- There is no limitation, i.e. you can also add other sectors (financial, trade, R&D, etc).

We then mathematically solve these problems under uncertainty, because we add stochastic processes to the system of equations.

- General Equilibrium (GE): equations must always hold.
  Short-run: decisions, quantities and prices adjust such that equations are full-filled.
  Long-run: steady-state, i.e. a condition or situation where variables do not change their value (e.g. balanced-growth path where the rate of growth is constant).
- Stochastic (S): disturbances (or shocks) make the system deviate from its steady-state, we get business cycles or, more general, a data-generating process
- Dynamic (D): Agents are forward-looking and solve intertemporal optimization problems. When a disturbance hits the economy, macroeconomic variables do not return to equilibrium instantaneously, but change very slowly over time, producing complex reactions. Furthermore, some decisions like investment or saving only make sense in a dynamic context. We can analyze and quantify the effects after (i) a temporary shock: how does the economy return to its steady-state, or (ii) a permanent shock: how does the economy move to a new steady-state.

DSGE models are intensively used in policy making e.g. by central banks they play a central role in providing an analytical foundation for the decision making. They have become the standard modelling framework in modern macroeconomic research. They often serve as a macroeconomic laboratory that allows to analyze how economic agents respond to changes in their environment as all variables are determined simultaneously and endogenously given a sound micro-founded theoretical setting. Real business cycle (RBC) models, New Keynesian models, and asset pricing models all belong to this very general modelling class.

Estimating such models is a very challenging task as one has to first find a reduced-form suitable for estimation. The upside, however, is that ideally, with suitable solution and estimation techniques, we have a clear theoretical foundation for our empirical results that enable clear understanding of transmissions channels and policy advise.

Challenges in Quantitative Macroeconomics

- Macroeconomic time series are indicative of nonlinearities, non-Gaussianity and time-varying volatility.
- Model specification: How to select the "right" model, how to cope with misspecification?
- SVAR
  - many free parameters to estimate, need plausible identifying restrictions
  - sensitive to the identification restrictions used
  - risk of specification search: researchers look for reasonable answers or puzzles under the cloak of formal statistical inference
- DSGE
  - trade-off between theoretical coherence and empirical fit
  - heavily aggregated and stylized, require a lot of modelling assumptions
  - solving DSGE models requires knowledge in computational eocnomics

- computational implementation is in both cases often cumbersome and challenging

Despite the challenges and current developments, SVAR and DSGE models continue to be the fundamental tools in current macroeconomic analysis. Be warned: there is quite the investment one needs to undergo to study quantitative macroeconomics and there is a huge component of self-teaching involved, because most of us lack the required background in econometrics, mathematics, numerics and statistics. In fact, most macroeconomists doing research in this area are more or less self-taught (I can attest to this for myself), or come from a related field.

2 Solution to Programming Languages:

1. General purpose: C/C++, Fortran, Python, Excel. Domain-specific: MATLAB, Julia, R, Mathematica, EViews.

2. Every program is a set of instructions, say to add two numbers. Compilers and interpreters take human-readable code and convert it to computer-readable machine code. In a compiled language, the target machine directly translates the program. In an interpreted language, the source code is not directly translated by the target machine. Instead, a different program, aka the interpreter, reads and executes the code. Some modern languages like Python can have both compiled and interpreted implementations, but for simplicity's sake it is useful to keep in mind the distinction.

   Compiled languages like Fortran, C or C++ are usually fastest, more efficient and more powerful, but they are harder to learn and harder to code in. They also require a build step, i.e. they need to be compiled. Interpreted languages like Python, R, Mathematica, MATLAB, R or Julia are slower, but easier to learn and faster to code in. Interpreters run through a program line by line and execute each command. Interpreted languages tend to be very similar in the syntax, but differ in best practices and concepts.

   Interpreted languages were once significantly slower than compiled languages. But, with the development of just-in-time (JiT) compilation, that gap is shrinking. MATLAB and Julia are two very prominent examples that make use of JiT compilation, that is they combine both worlds.

   You can also make use of e.g. Fortran or C++ code in MATLAB, R, Python or Julia; that is, write very CPU-intensive tasks in a compiled language and use them in an interpreted language.

3. Learning a programming language is a huge investment; however, once one has knowledge of one, learning another one tends to be easier as they are based on similar principles. Try to stick with popular choices as the choice of learning resources and communities that help you learn this language are wider spread, i.e. googling for help is much easier for Python than for Fortran. Often the project you are working on dictates which programming language you should use. The general purpose languages can be used in many non-scientific applications, so your investment might payoff in very different fields in the end.

   In scientific computing, particularly in Macroeconomics, we are often faced with CPU intensive problems and need to prototype models and methods quickly. An interpreted language like MATLAB or Julia that does just-in-time compilation is therefore best suited for such tasks. Moreover, having some basic knowledge in C++ is advisable to write computational intensive tasks in a compiled language and reuse this as e.g. so-called MEX files in MATLAB. However, the main determining factor is by looking at legacy code of the last 20-30 years of research done in quantitative and computational Macroeconomics, we see that most was and still is conducted in MATLAB, whereas highly intensive tasks were programmed in Fortran. So keep in mind, that you need to understand this legacy codebase. In the last couple of years, researchers in Macroeconomics are really pushing Julia. New developments like Machine Learning require you to invest in Python. For writing scientific reports and papers you should get familiar with Latex and Markdown.

   Another issue to consider is the license, cost and support of the language maintainers. Most programming languages are free and open-source, others like MATLAB are proprietary and are quite expensive (free and open-source clones like Octave tend to be very slow unfortunately). Regardless of the license, having a good governance structure, i.e. a board, cooperation or company driving the development of the language, is very important for the sustainability of the language and for your investment in a computer language.

   Lastly, and very importantly, have a look at the toolset available for the languages. Which Integrated Development Environment (IDE) do you like best? Which code editor do you prefer? How good are the debugging capabilities of your chosen environment. Things like syntax

highlighting, smart indentation, code linting, comparison tools, handling of workspace, etc. are very important. Some languages like MATLAB bring their own IDE in one big package and it works very well. Others like Julia, Python or C++ can be neatly integrated in a variety of environments; in fact Visual Studio Code has become the leading editor and environment for many languages, but of course there are many other great choices depending on your needs and preferences.

So which computer languages should you devote your time into, if you are interested in computational or quantitative macroeconomics?

Here is my opinionated advice:

- Default languages (excellent knowledge): Julia and MATLAB
- Data analysis and Machine Learning (advanced knowledge): R and Python
- Heavy tasks (basic knowledge): C++ and Fortran
- Scientific writing (advanced knowledge): Latex and Markdown

3 Solution to Quick Tour: MATLAB

progs/matlab/quickTourMatlab.m

```
% ------------------------------------------------------------------------
% Illustration of MATLAB's basic matrix operations, logical operators, and
% indexing.
% ------------------------------------------------------------------------
% Willi Mutschler (willi@mutschler.eu)
% Version: October 19, 2023
% ------------------------------------------------------------------------

% 1)
x = [-1; 0; 1; 4; 9; 2; 1; 4.5; 1.1; -0.9];
y = [1, 1, 2, 2, 3, 3, 4, 4, 5, nan]';

% 2
length(x)==length(y)
size(x,1)==size(y,1)

% 3
x < y
x < 0
x+3 >= 0
y < 0

% 4
all(x+3>=0) && all(y>0)

% 5
all(x+3>=0) || all(y>0)

% 6
any(y>0)

% 7
x+y
try
    x*y
catch
    disp(lasterr)
end
x*y'
x'*y
y/x
x/y

% 8
x.*y
y./x

% 9
log(x)
exp(x)
```

```
51
52  % 10
53  any(sqrt(x)>=2)
54
55  % 11
56  a = sum(x)
57  b = sum(y.^2,'omitnan')
58
59  % 12
60  sum(x.*y.^2,'omitnan')
61
62  % 13
63  sum(x>0)
64
65  % 14
66  x.^y
67  x.^(1/y)
68  log(exp(y))
69  y*[-1,1]
70  x+[-1,0,1]
71  sum(y*[-1,1],1,'omitnan')
72
73  % 15
74  X = [1 4 7;
75       2 5 8;
76       3 6 9];
77
78  transpose(X)
79  X'
80  size(X)
81  det(X)
82
83  % 16
84  trace(X)
85  sum(diag(X))
86
87  % 17
88  X
89  X([1 5 9])=[7 8 9]
90  X
91
92  % 18
93  eigvalX = eig(X);
94  if all(eigvalX>0)
95      fprintf('X is positive definite\n')
96  elseif all(eigvalX>=0)
97      fprintf('X is positive semi-definite\n')
98  elseif all(eigvalX<0)
99      fprintf('X is negative definite\n')
100 elseif all(eigvalX<=0)
101     fprintf('X is negative semi-definite\n')
102 else
```

```
    fprintf('X is neither positive nor negative (semi-)definite\n')
end

% 19
invX = inv(X);
eig(invX)
1./eigvalX % these are the same eigenvalues

% 20
a = [1;3;2];
a'*X
a'.*X
X*a

% 21
a'*X*a

% 22
I = eye(3);
X = reshape(1:9,3,3);
Y = [X I]
Z = [X;I]

% 23
x1 = 1:9
x2 = repmat([0 1],1,4)
x3 = repmat(1,1,8)
x4 = repmat([-1 1],1,3)
x5 = 1980:5:2010
x6 = 0:0.01:1

% 24
linspace(-pi,pi,500)

% 25
1:10+1
(1:10)+1
1:(10+1)

% 26
x = [1 1.1 9 7 1 4 4 1]';
y = [1 2 3 4 4 3 2 nan]';
z = [true true false false true false false false];

% 27
x(2:5)
x(4:end-2)
x([1 5 8])
x(repmat(1:3,1,4))
y(z)
y(~z)
y(x>2)
```

```
y(x==1)
x(~isnan(y))
y(~isnan(y))

% 28
x2 = x;
x2(x2==4) = -4;
x2

% 29
x2(x2==1) = nan;
x2

% 30
x2(z) = [];
x2

% 31
M = reshape([1:12 12:-1:1],4,6);

% 32
M(1,3)
M(:,5)
M(2,:)
M(2:3,3:4)
M(2:4,4)
M(M>5)
M(:,M(1,:)<=5)
M(M(:,2)>6,:)
M(M(:,2)>6,4:6)

% 33
find(M(:,5)>3*M(:,6))

% 34
sum(M>7,'all')

% 35
sum(M(2,:) < M(1,:))

% 36
sum( M(:,2:end) > M(:,1:end-1) ,'all')
```

4 Solution to Quick Tour: git with GitKraken

1. *git* is a version control system, i.e. a way to track changes to code, text, documents, data etc. It let's you go back and forth between many different versions of the same file, and see a list of the differences. Collaboration becomes (technically) very easy and straightforward as people can work on different files or different versions of the same file simultaneously and afterwards merge their changes.

   git is the most popular version control system invented in 2005 to track the development of the worldwide largest open-source project: the Linux Kernel. It is a **command line tool**, and at some point you should learn the commands on the Command Line Interface (CLI). However, there are many graphical user interface (GUI) programs that make getting started with Git much easier and integrate seamlessly with online collaboration platforms such as GitHub or GitLab. Therefore, in this exercise I will focus on such a tool called GitKraken, which I use daily and highly recommend.[5] GitKraken offers a free trial of their paid license, but also offers a free version for use on publicly-hosted repositories (which is fine for our purposes as we will mostly do stuff locally anyway). Additionally, GitKraken also offers the Pro license FREE to students and teachers through the GitHub Student Developer Pack (`https://education.github.com/pack`).

   While git and GitKraken are tools that you install on your computer, GitHub is an online platform that provides a nice visual interface to help you manage your version-controlled projects remotely. It is the largest git repository hosting service and has become by far the largest open-source collaboration site. Another important online platform is Gitlab as you can also host that on your computer or server. For individuals gitea offers yet another way to host a stripped down version of GitHub or Gitlab. I personally have accounts on GitHub (`https://github.com/wmutschl`) and Gitlab (`https://gitlab.com/wmutschl`), but also use self-hosted versions of Gitlab (`https://git.dynare.org/wmutschl`) and gitea (`https://git.mutschler.eu`) to mirror my projects.

2. A typical workflow looks like this:
   - retrieve data and prepare it for estimation purposes
   - select a model framework, decide on certain hyperparameters and modeling choices and then run an estimation
   - prepare tables, graphs and reports

   All of these tasks heavily rely on coding, i.e. putting text into some files that are then evaluated by software that actually performs the tasks. Moreover, we will see that estimating macroeconomic models requires a lot of trial and error and accordingly those files constantly change and need to be adapted. *git* enables you to track these changes as it gives you an organized revision history. So you can experiment with your codes, make changes to a project and always keep the ability to go back and fourth between changes. So stop naming files like *2022-10-17-master-thesis-v2-final-now-really-final.tex* and let *git* do its magic for you by simply tracking the file *thesis.tex* with all of its revision history.

3. Follow the instructions provided in the links or get in touch if you are struggling with the installation.

4. Follow the instructions provided in the links or get in touch if you are struggling with the installation.

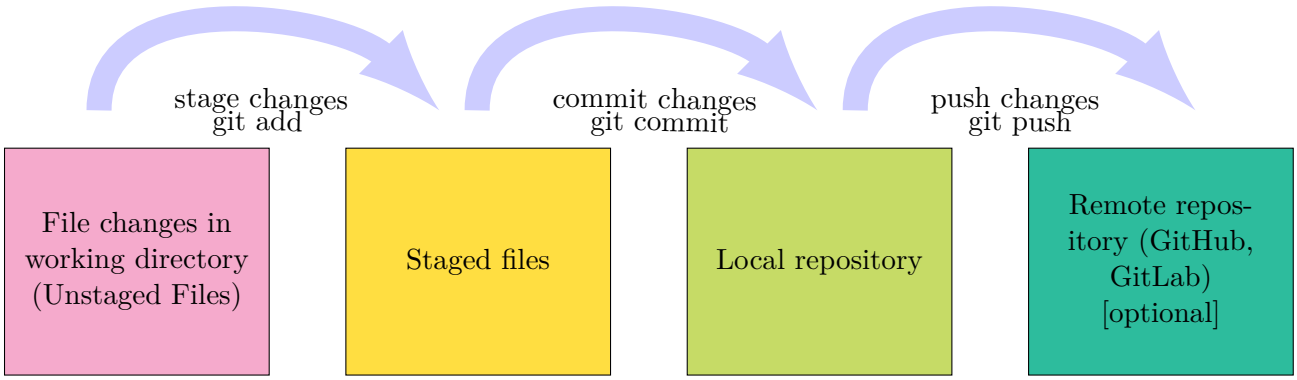
5. Follow the instructions provided in the links or get in touch if you are struggling with the installation.

[5] Other GUI programs work very similarly, see `https://en.wikipedia.org/wiki/Comparison_of_Git_GUIs` for a comparison of features. I also recommend the built-in git functionality of Visual Studio Code.

6. In GitKraken: Open a new Tab, click *Start a local repo*, then on the *Init* register select *Local Only* and fill out the details. Note that GitKraken automatically creates a first commit with a `README.md` file. Inside every repository there is a hidden folder `.git`. It contains everything done by *git*, so all the changes you will ever do. Never delete this folder! Also putting a repository on a cloud storage folder might damage this folder, so best practice is to use a local folder on the disk. We will cover how to push the repository to a so-called remote which works basically like syncing, but much more robust and git-ier.

7. Now the benefits of using a GUI like GitKraken become evident, as our changes are displayed in the *Unstaged Files* area and by clicking on the file we get a really pretty side-by-side comparison of all the changes. We can now decide which lines we want to `stage` and `commit`.

8. The git model looks like the following diagram:



   You do your work in your **working directory**. On the **stage** you collect all the changes that you want to save. This is very powerful because sometimes it is just individual lines of code or text that you want to keep track of and not the whole file. Once you've tracked all the changes that you want to combine, it is time to collect these changes into a **commit**. A commit is a permanent snapshot of the files that git tracks stored in the .git directory. It is associated with a unique identifier (hash). In other words, a commit is like a snapshot in time; you can always revert back to this and see what changes were made compared to any other commit. On your local repository (i.e. on your local machine) you now have a nice versioned history. However, if you want to collaborate with others or sync your repository to a specialized cloud provider you need to push these changes to a so-called remote repository, typically on GitHub, GitLab, but any folder that you can access via remotely might serve as a remote repository.

9. Click on the file and select `Stage File` or add each line by clicking on the plus or minus signs left to each line. Once you are happy with the file, click on the X to close the file-comparison window. We now don't see any unstaged files and can proceed to write a commit message and then click on the big green button.

10. A *good commit* typically does one discrete task or change only. For example, you added a variable to the regression specification in the code, in the output and in the report. Or you changed the name of a variable and treat it properly across multiple scripts. This enables you to make meaningful commit messages like *Add year dummies to regression specification* and you thus end up with a well organized repository. This workflow needs some practice and everyone is slightly different with regards to this. Nevertheless, try to combine changes to certain meaningful smaller tasks and provide good commit messages. In my experience, having ten tiny commits is always preferable to one large commit. Your future self and collaborators will thank you!

    The question to what you should include in your commits, is also a matter of choice and preference. Definitely your script files of codes, latex and text files. Data is also sometimes given

as csv files which are basically just text files. Binary files (like Excel sheets, Word documents, Power Point slides) are a bit tricky to handle, as you can't see the differences between versions in git. It depends on the specific needs whether one should commit these files as well (e.g. for Excel files with data this obviously makes sense), but I usually don't do this. Note that GitHub doesn't allow files larger than 100 MB or projects with total size larger than 1 GB. There is also a way to deal with large binary files called `Git Large File Storage (LFS)`, but we won't need this.

11. Right click on the initial commit and select *Reset main to this commit - Soft*. Click on the file in the staged files section and remove the last line from the stage. Re-commit your stage by providing a meaningful commit message and hitting the green button. Click on Stash to put the remaining changes into the stash.

12. Simply click on Push and add the remote. On the left Panel click on REMOTE to see the current remote (usually named origin). Note that you can add several remotes (say from different people) and compare the commits. Remotes are also a nice backup of your codes.

13. Branches are arguably the most powerful part of *git*. By default you have a **main** branch, but what if you want to do some experiments, re-write an estimation function from scratch, work on a new feature, etc? You could copy the whole folder and start working there or you use git and create a branch and make the changes there. You can switch between branches, make commits to any branch, move them around, etc. If your experiment doesn't work out, simply delete the branch. If your experiments work out, commit them and merge them into the main branch. Sometimes there will be conflicts which one needs to sort out, but using GUI tools like GitKraken makes this very easy as you have a pretty side-by-side comparison of changes. Branches are arguably the most powerful part of *git* especially for our purposes as research is a highly nonlinear process, and this way of doing version control is much more similar to how we actually work than the very linear way that other cloud storage providers do version control. Branches are also extremely powerful for collaboration as different people can work on the same thing at the same time.

    Select a so-called parent commit, where you want to create a new branch. Note that this doesn't have to be the latest commit. Click on the button *Branch* and name it according to the exercise. On the left panel, click on LOCAL to see an overview of all your branches.

14. Create, copy and paste the three files into your repository. Check for pasting errors and then *Stage all changes* and commit them.

15. Run the commands and solve any errors you might get from latex.

16. Follow the instructions in the exercise. Note that there is a difference between "Ignore" and "Ignore and Stop Tracking". "Ignore" simply adds the file(type) to the `.gitignore` file so that new files with that name/type/whatever are not tracked. To "Ignore and Stop Tracking" means to remove the file(s) from git version control: they will no longer be in the repo (as of the commit that performs the "stop tracking"). Basically, use "Ignore and Stop Tracking" if the file(s) you are ignoring never should have been in the repo in the first place.

17. Make sure you are on the correct branch *latex-exam-template* and push this branch to GitHub. Either right click on the commit or go to the left panel, click on PULL REQUESTS and on the green plus sign that appears. Select the *latex-exam-template* as the FROM REPO branch and *main* as the TO REPO branch. Enter a Title and Description and click on the green button. Have a look in GitHub ar the pull request. As there are no conflicts merge it and go back to GitKraken to see what happens in your repository. You might need to "fetch origin" by right clicking on the origin remote.

18. Double click on your local main branch and then click on pull, which fast forwards your repo to the merged changes. Then click on Pop to get the WIP codes which were stored on main. Right click on the README.md file in the *Unstaged Files* area and select *Discard changes*.