Computational Macroeconomics Exercises

Week 1

Willi Mutschler
willi@mutschler.eu

Version: August 16, 2022

Contents

1. What is Computational Macroeconomics

Broadly define the scope and research topics of "Computational Macroeconomics", sometimes referred to as "Numerical Methods in Economics". What are the challenges and approaches?

Readings

- Fernández-Villaverde, Rubio-Ramírez, and Schorfheide (2016, Part I)
- Judd (1998, Ch. 1)
- L. Maliar and S. Maliar (2014)
- Schmedders and Judd (2014)

2. DSGE Models: Definition, Key Challenges, And Basic Structure

1. Briefly define the term and key challenges of Dynamic Stochastic General Equilibrium (DSGE) models. What are DSGE models useful for?
2. Outline the common structure of a DSGE model. How do Neo-Classical, New-Classical and New-Keynesian models differ?
3. Comment whether or not the assumptions underlying DSGE models should be as realistic as possible. For example, a very common assumption is that all agents live forever.

Readings

- Fernández-Villaverde, Rubio-Ramírez, and Schorfheide (2016, Ch. 1)
- Torres (2013, Ch. 1)

3. Programming Language

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

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

4. 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)

5. ARMA(1,1) simulation

Consider the ARMA(1,1) model:

$$x_t - \theta x_{t-1} = \varepsilon_t - \phi \varepsilon_{t-1}$$

where $\varepsilon_t \sim N(0, 1)$.

1. Compute the non-stochastic steady-state with pen and paper, i.e. what is the value of x_t if $\varepsilon_t = 0$ for all t?
2. Write a Dynare mod file for this model:
   - x is the endogenous variable, ε the exogenous variabel, and θ and ϕ are the parameters.
   - Set $\theta = \phi = 0.4$.
   - Write either a `steady_state_model` or `initval` block and compute the steady-state.
   - Start at the non-stochastic steady-state and simulate 200 data points using a `shocks` block and the `stoch_simul` command. Drop the first 50 observations and plot both x as well as ε.
   - Try out different values for θ and ϕ. What do you notice?
3. Redo the exercise in MATLAB without using Dynare.

Hints:

- Don't forget to add Dynare's matlab path: `addpath C:/dynare/5.1/matlab` for Windows, `addpath /Applications/Dynare/5.1/matlab` for macOS, `addpath /opt/dynare/5.1/matlab` for Linux machines. If you installed Dynare to a different folder, adapt the path accordingly.
- When saving your Dynare script as a mod file, try to use a simple name. That is, don't use special characters or spaces in the filename, e.g. `arma(1,1).mod` as well as `arma 1 1.mod` are not valid filenames, whereas `arma.mod` or `arma_1_1.mod` are valid.
- It is advised to save the mod file directly on your hard drive and not on a cloud storage drive. If you do save it to a folder that is in sync with e.g. Dropbox, iCloud, Nextcloud, OneDrive (etc), please (temporarily) deactivate sync activities, to prevent sync conflicts and weird errors when running Dynare.

References

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

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. ISSN: 01651889. 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.

Fernández-Villaverde, Jesús, Juan F. Rubio-Ramírez, and Frank Schorfheide (2016). "Solution and Estimation Methods for DSGE Models". In: *Handbook of Macroeconomics*. Ed. by John B. Taylor and Harald Uhlig. Vol. A. Elsevier North-Holland, pp. 527–724. ISBN: 978-0-444-59469-3.

Judd, Kenneth L. (1998). *Numerical Methods in Economics*. Vol. 1. The MIT Press.

Maliar, Lilia and Serguei Maliar (2014). "Numerical Methods for Large-Scale Dynamic Economic Models". In: *Handbook of Computational Economics Volume 3*. Ed. by Karl Schmedders and Kenneth L. Judd. Elsevier, pp. 325–477. DOI: 10.1016/B978-0-444-52980-0.00007-4.

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*.

Schmedders, Karl and Kenneth L. Judd (2014). "Introduction for Volume 3 of the Handbook of Computational Economics". In: *Handbook of Computational Economics*. Vol. 3. Elsevier, pp. xv–xvii. ISBN: 978-0-444-52980-0. DOI: 10.1016/B978-0-444-52980-0.00020-7.

Torres, José L. (2013). *Introduction to Dynamic Macroeconomic General Equilibrium Models / José L. Torres, Department of Economics, University of Málaga*. Malaga, Spain: Vernon Press. ISBN: 978-1-62273-007-0.

A. Solutions

1 Solution to What is Computational Macroeconomics Computational 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 numerical methods (the application of numerical methods by means of algorithms). To this end, we will focus on the workhorse Dynamic Stochastic General Equilibrium (DSGE) model paradigm, and develop the numerical procedures and algorithms required to solve and simulate such models. Very important: this course is NOT ABOUT HOW TO BUILT structural models, but it IS ABOUT HOW TO SOLVE AND SIMULATE such models on a computer. This enables us to look at abstract macroeconomic concepts, for example:

- transmission channels of macroeconomic shocks in linearly vs non-linearly solved models
- the short- and long-run effects of a higher inflation target
- models with occasionally binding constraints (zero-lower-bound, irreversible investment)
- the effects of quantitative easing/tightening (possibly at the zero-lower-bound)
- models with precautionary savings and heterogenous agents
- fiscal multipliers, is there a difference between linearly vs. non-linearly solved models
- models with an increasing probability of a rare disaster (financial crises, pandemic, war)
- models with an actual disaster
- dynamics of the transition from dirty to green energy
- optimal monetary, fiscal and environmental policy

To study such phenomena we typically cannot use simplified and linearized pen-and-paper models anymore, but need to rely on a computer to solve and simulate such models. The computer basically becomes our lab in which we can conduct experiments by changing parameters or running counterfactual scenarios and policies. This enables us to not only describe but also quantify transmission channels and effects of different economic policies in order to provide sound and exact policy analysis and advice.

The computational implementation, however, is often cumbersome and challenging. Fortunately, modern programming languages make it easier for us to develop algorithms and toolboxes for dealing with structural models in macroeconomics. There is always a trade-off between model complexity and numerical method complexity. Sometimes it is easier to simplify slightly your model instead of inventing new numerical methods. Other times the specific question under study enforces you to develop new algorithms and approaches. In practice, however, most of the times you don't have to reinvent the wheel; but build on previous approaches, improve algorithms and adapt them in order to answer your research question. The challenge is often that the codes developed so far are quite model-specific and require a high degree of fine-tuning and thus a deep understanding of the underlying algorithms. Open-Source projects like Dynare aim to provide a model-independent and user-friendly approach; but it takes time for the development team of Dynare to catch up with the most recent developments in the literature.

Be warned: there is quite the investment one needs to undergo to study computational macroeconomics and there is a huge component of self-teaching involved, because most of us lack the required background in computational science, numerics and mathematics. In fact, most macroeconomists doing research in this area are more or less self-taught (I can attest to this for myself), and we are always trying to look across the pond at what the current developments are in computer science and numerical mathematics and whether or not these are useful to solve and simulate our structural models.

2 Solution to DSGE Models: Definition, Key Challenges, And Basic Structure

1. 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. In reality all macro variables are related to each other, either directly or indirectly, so there is no "cetribus paribus", but a dynamic stochastic general equilibrium system.

   - 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.

   Basic structure:

   $$E_t \left[f(y_{t+1}, y_t, y_{t-1}, u_t) \right] = 0$$

   where E_t is the expectation operator with information conditional up to and including period t, y_t is a vector of endogenous variables at time t, u_t a vector of exogenous shocks or random disturbances with proper density functions. $f(\cdot)$ is what we call economic theory.

   First key challenge: values of endogenous variables in a given period of time depend on future expected values. We need dynamic programming techniques to find the optimality conditions which define the economic behavior of the agents. The solution to this system is called a decision or policy function:

   $$y_t = g(y_{t-1}, u_t)$$

   describing optimal behavior of all agents given the current state of the world y_{t-1} and after observing current shocks u_t.

   Second key challenge: DSGE models cannot be solved analytically, except for some very simple and unrealistic examples. We have to resort to numerical methods and a computer to find an approximated solution. This is the focus of the course on `Computational Macroeconomics`.

   Once the theoretical model and solution is at hands, the next step is the
   third key challenge: application to the data. The usual procedure consists in the calibration of the parameters of the model using previous information or matching some key ratios or moments provided by the data. More recently, within the field of quantitative macroeconomics or macroeconometrics, researchers have applied formal statistical methods to estimate the parameters using maximum likelihood, Bayesian techniques, indirect inference, or a method of moments. This is the focus of the course on `Quantitative Macroeconomics`.

2. 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 $\hookrightarrow$ 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. $\hookrightarrow$ 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, e.g. current models feature a financial sector, international trade, research & development, etc.

3. Neoclassical or New-Classical models are basically the same terminology (unless you study economic history or really want to dive into the different school of thoughts). Basically, both approaches focus on microfoundations, the one more in a classical sense (focus on real rigidities) and the other one more in a Keynesian sense (focus on nominal rigidities). In principle this is already evident in the baseline RBC model and the baseline New-Keynesian model.
   - Canonical neoclassical model (aka RBC model): reduce economy to the interaction of just one (representative) consumer/household and just one (representative) firm. Representative household takes decisions in terms of how much to consume (or save) and how much time is devoted to work (or leisure). Representative firm decides how much it will produce. Equilibrium of the economy will be defined by a situation in which all decisions taken by all economic agents are compatible and feasible. One can show that business cycles can be generated by one special disturbance: total factor productivity or neutral technological shock; hence, model generates real business cycles without any nominal frictions. Of course, one can add real frictions like time-to-built or preference shocks.
   - New-Keynesian models have the same foundations as New-Classical general equilibrium models, but incorporate different types of rigidities in the economy. Whereas new classical DSGE models are constructed on the basis of a perfect competition environment, New-Keynesian models include additional elements to the basic model such as imperfect competitions, existence of adjustment costs in investment process, liquidity constraints or rigidities in the determination of prices and wages.

   Noth that the scale of DSGE models has grown over time with incorporation of a large number of features. To name a few: consumption habit formation, nominal and real rigidities, non-Ricardian agents, investment adjustment costs, investment-specific technological change, taxes, public spending, public capital, human capital, household production, imperfect competition, monetary union, steady-state unemployment etc. Moreover, New-Keynesian models have become the leading macroeconomic paradigm.

4. The degree of realism offered by an economic model is not a goal per se to be pursued by macroeconomists; typically we are focused on the model's **usefulness** in explaining macroeconomic reality. General strategy is the construction of formal structures through equations that reflect the interrelationships between the different economic variables. These simplified structures is what we call a model. The essential question is not that these theoretical constructions are realistic descriptions of the economy, but that they are able to explain the dynamics observed in the economy. Therefore, in my opinion, it is not correct to outright reject a model ex-ante just because it is based on assumptions that we believe are not realistic. Rather, the validations must be based on the usefulness of these models to explain reality, and whether they are more useful than other models. Of course, most of the times unrealistic assumptions will yield non-useful models: often, however, simplified assumptions that are a very rough approximation of reality

yield quite useful models. Either way, the DSGE model paradigm is EXTREMELY UP-FRONT with our assumptions and provide the EXACT model dynamics in terms of mathematical formulations that can be challenged, adapted and, ideally, improved.

Regarding the assumption that the lifetime of economic agents is assumed to be infinite: We know that the lifetime of consumers, businesses and governments is in fact finite. However, in most models this is a valid approximation, because for solving and simulating these models it is not important that agents actually live forever, but that they use the infinite time as **the reference period for taking economic decisions**. This is highly realistic! No government thinks it will cease to exist at some point in the future and no entrepreneur takes decisions based on the idea that the firm will go bankrupt sometime in the future. Granted, for consumers this is rather weak; however, we may think about families, dynasties or households rather than consumers, then the infinite time planning horizon assumption is a feasible one. Of course, to study the finite life cycle of an agent (school-work-retirement) or pension schemes, the so-called Overlapping Generations (OLG) framework is probably more adequate. But, we need the same methods and techniques to deal with OLG models as with New-Keynesian or RBC models, because all these models belong to the the same class, i.e. are all DSGE models.

3 Solution to Programming Language:

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 C++ 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. Moreover, looking at legacy code in 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. Nevertheless, in the last couple of years, researchers in Macroeconomics are really pushing Julia. Moreover, 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): Markdown and Latex

4 Solution to Quick Tour: MATLAB

../progs/matlab/quick_tour_matlab.m

```
% 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
x*y
x*y'
x'*y
y/x
x/y

% 8
x.*y
y./x

% 9
log(x)
exp(x)

% 10
any(sqrt(x)>=2)

% 11
a = sum(x)
b = sum(y.^2,'omitnan')

% 12
sum(x.*y.^2,'omitnan')

% 13
sum(x>0)

% 14
x.^y
x.^(1/y)
log(exp(y))
y*[-1,1]
x+[-1,0,1]
sum(y*[-1,1],1,'omitnan')

% 15
X = [1 4 7;
```

```
     2 5 8;
     3 6 9];

transpose(X)
X'
size(X)
det(X)

% 16
trace(X)
sum(diag(X))

% 17
X
X([1 5 9])=[7 8 9]
X

% 18
eigvalX = eig(X);
if all(eigvalX>0)
    fprintf('X is positive definite\n')
elseif all(eigvalX>=0)
    fprintf('X is positive semi-definite\n')
elseif all(eigvalX<0)
    fprintf('X is negative definite\n')
elseif all(eigvalX<=0)
    fprintf('X is negative semi-definite\n')
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')
```

5 Solution to ARMA(1,1) simulation

1. As for all t the shocks are zero, i.e. $\varepsilon_t = 0$, we get $x - \theta x = 0 \Leftrightarrow x = 0$, where x denotes the non-stochastic steady-state (we drop the time subscript in the original equation when computing the steady-state).

../progs/dynare/arma_1_1_dynare.mod

2.
```
% ========================================================================
% Declare endogenous variables
% ========================================================================
var
  x          ${X}$              (long_name='ARMA(1,1) Process')
;

% ========================================================================
% Declare exogenous variables
% ========================================================================
varexo
  e          ${\varepsilon}$ (long_name='White Noise Process')
;

% ========================================================================
% Declare parameters
% ========================================================================
parameters
  THETA   ${\theta}$   (long_name='AR parameter')
  PHI     ${\phi}$     (long_name='MA parameter')
;

% ========================================================================
% Calibrate parameter values
% ========================================================================
THETA = 0.4;
PHI   = 0.4;

% ========================================================================
% Model equations
% ========================================================================
model;
[name='ARMA(1,1)']
x - THETA*x(-1) = e - PHI*e(-1);
end;

% ========================================================================
% Steady State Model
% ========================================================================
steady_state_model;
x = 0;
end;

% ========================================================================
% Declare settings for shocks
% ========================================================================
shocks;
var e = 1;
end;

% ========================================================================
% Computations
% ========================================================================
steady;
check;

stoch_simul(order=1,periods=200,irf=0);
```

```
% stoch_simul stores simulated values for x in two places:
%   - a variable x in the workspace
%   - a variable endo_simul in the structure oo_
% the underlying shocks (that were drawn from the normal distribution) are in
%   - a variable exo_simul in the structure oo_

% ========================================================================
% Plotting
% ========================================================================
figure;
subplot(2,1,1);
plot(x(51:end));% get rid of the initial 50 periods
title('$x_t$','Interpreter','latex');
subplot(2,1,2);
plot(oo_.exo_simul(51:end));
title('$\varepsilon_t$','Interpreter','latex');

% Note that both x and exo_simul are equal whenever PHI=THETA
if PHI == THETA
    disp(isequal(x,oo_.exo_simul))
end
```

Some notes:

- we used MATLAB codes for plotting.
- whenever the parameters are equal to each other, the simulated series for x_t is always equal to the drawn shocks ε_t.
- for some parameter combinations, e.g. $\theta = 1.5$ and $\theta = 0.4$, Dynare prints an error `Blanchard & Kahn conditions are not satisfied: no stable equilibrium.` If you are familiar with time series models, this is due to the fact that this parameter combination provides a non-stationary process, i.e. the process explodes and does not return to the equilibrium/steady-state.

../progs/matlab/arma_1_1_matlab.m

```
% housekeeping
clear global; clearvars; close all; clc;

% run this script after "dynare arma_1_1_dynare" so we can compare stuff
dynare arma_1_1_dynare

% store stuff from Dynare
dyn_x = transpose(oo_.endo_simul(1,:));
dyn_e = oo_.exo_simul(:,1);
sample_size = options_.periods;
std_dev = sqrt(M_.Sigma_e); % this is the standard error declared in the shocks
    block
THETA = M_.params(ismember(M_.param_names,'THETA'));
PHI = M_.params(ismember(M_.param_names,'PHI'));

% get same shock series as in Dynare by using the same seed
set_dynare_seed('default');
e = std_dev*randn(sample_size,1); % randn draws standard normally distributed
    variables
if isequal(e,dyn_e)
    disp('shock series are equal')
else
    fprintf('shock series are not equal\n') % fprintf is more flexible than disp,
    the \n prints a new line
end

% simulate
x = zeros(sample_size,1); % pre-allocate storage
```

```
x(1) = THETA*0 + e(1) - PHI*0; % first period t=1
for t = 2:sample_size
    x(t) = THETA*x(t-1) + e(t) - PHI*e(t-1);
end

if isequal(x,dyn_x)
    fprintf('simulated series are the same\n')
else
    fprintf('simulated series are not the same, the maximum absolute deviation is
    %e:\n',norm(x-dyn_x))
end

% exploding paths
THETA = 1.5;
PHI = 0.4;
x = zeros(sample_size,1); % pre-allocate storage
x(1) = THETA*0 + e(1) - PHI*0; % first period t=1
for t = 2:sample_size
    x(t) = THETA*x(t-1) + e(t) - PHI*e(t-1);
end

figure;
subplot(2,1,1)
plot([1:50],x(1:50));
subplot(2,1,2)
plot([51:100],x(51:100));
sgtitle('exploding $x_t$','Interpreter','latex');
```