

PhD Coding Lab Syllabus

Christian SCHLUTER





(<https://christianschluter.github.io>)

“As you start to run R code, you’re likely to run into problems. Don’t worry — it happens to everyone. I have been writing R code for years, and every day I still write code that doesn’t work!” - R for Data Science (R4DS)

1 Outline

In the **Coding** or Programming **Lab** we develop the playbook of the modern applied and quantitatively minded economist whose ambition is to go beyond classic linear regression analysis. Many empirical research articles in the best journals feature these days a mix of economic modelling, explorative data analysis, and estimation strategies that exploit directly the structure of the initial model. The first objective of this module is to enable students to appreciate better such research designs, which requires a certain familiarity with what have become standard computational methods (such as the **recursive methods** of dynamic programming). Developing a working knowledge of these techniques could then lead the student to apply them in the context of their own research, in order to go beyond the constraints imposed by classic regression analysis.

We will focus first and foremost on *algorithms* developed in concrete economic settings (e.g. labour supply, job search), so that students can implement these in the language of their choice. For practical purposes, my illustrations are in ,  or Julia .

2 A preliminary list of topics

2.1 Topic 1: Iterative root-finding algorithms

We will commence with **iterative root-finding algorithms** for non-linear problems, such as the **Newton–Raphson** method, and apply the implementation to an economic problem of static labour supply. Generically, to find the root of a univariate smooth function f , $f(\theta) = 0$, the updating scheme is $\theta_{t+1} = \theta_t - f(\theta_t)/f'(\theta_t)$. Versions of Newton–Raphson are also used in off-the-shelf Maximum

Likelihood estimation and GLMs (and their Iteratively Reweighted Least Squares implementations) such as probits in the form of so-called Fisher's scoring or the BHHH algorithm. Generically, we have an updating scheme of the form

$$\theta_{t+1} = \theta_t + \rho_t \mathbf{d}_t(\theta)$$

where θ denotes the parameter estimate, the preset ρ_t captures the learning rate, and $\mathbf{d}_t(\theta)$ indicates the descent direction. Several Machine Learning approaches use Gradient Descent, a closely linked updating scheme. Here the objective is to minimise a loss function $R(\theta)$, and the scheme is $\theta_{t+1} = \theta_t - \rho \nabla R(\theta_t)$ where the preset parameter ρ is the learning rate and ∇R denotes the gradient of the loss function. If the data set is too large, sampling (or so-called mini-batching) leads to the Stochastic Gradient Descent method.

Applications: Static labour supply

2.2 Topic 2: Introduction to Dynamic Programming

The second topic is dynamic optimisation, which will be done in the modern recursive way called **Dynamic Programming** (DP) based on the Bellman equation. For finite horizon problems, such as *how to eat cake*, we will proceed by **backward induction**. For infinite horizon problems, such as *stochastic growth* or *search models of the labour market*, we will implement **value function iterations**. To speed up convergence, we will also examine **policy iterations**.

To be more specific (and to define the jargon), we are interested in the value function V of the sequential dynamic infinite horizon optimisation problem:

$$V(x_0) = \sup_{\{x_{t+1}\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t F(x_t, x_{t+1}) = \sup_{\{x_{t+1}\}_{t=0}^{\infty}} F(x_0, x_1) + \beta V(x_1)$$

subject to $x_{t+1} \in \Gamma(x_t)$ for all $t \geq 0$, and x_0 given, where $x_t \in X$. The payoff function F depends on the state variable x_t , and x_{t+1} is the control variable, which also becomes the state variable in the next period. For instance, take a plain-vanilla growth model: the pay-off function is the utility function, the state variable is the current capital stock, and the consumer has to decide optimally between consumption and investment, trading-off consumption today and consumption tomorrow. In a model of job search, the searcher needs to decide whether to accept the current job offer today, or continue search in the next period.

The Bellman equation turns the sequence problem into a functional equation, $V(s) = \sup_{a \in \Gamma(s)} \{F(s, a) + \beta V(s')\}$. Here s refers to the current state, s' to the state in the next period, and a to the action (choice). The right-hand-side of the Bellman equation suggests an iterative scheme

$$V^{(n+1)}(s) = \arg \max_a F(s, a) + \beta V^{(n)}(s')$$

which converges to a unique solution because we have a contraction mapping. The optimal action (when the state is s) is called the policy $\pi(s) = \arg \max_a F(s, a) + \beta V(s')$. The faster policy iterations algorithm flip-flops between evaluation of the value function for a given policy, and improving the policy function.

Applications: Dynamic labour supply, deterministic growth, stochastic growth, job search. If time permits, we will look at Chetty, R (2008, JPE), “Moral Hazard versus Liquidity and Optimal Unemployment Insurance”, who uses a finite-horizon McCall job search model with search effort.

2.3 Topic 3: Introduction to Reinforcement Learning

If time permits we will look into recent advances in **Reinforcement Learning** (RL) where the objective is to learn the value and policy functions of the dynamic programming problem without having to know and specify the full model. The typical approach is to combine assessing the optimality of the current policy (“exploitation”) with exploring the parameter space randomly (“exploration”), usually over a long episode, and then over many episodes. DeepMind’s AlphaGo is one example of the spectacular recent successes of RL.

To fix ideas, call the right-hand-side of the Bellman equation now the state-action value function $Q(s, a)$, and denote by q^* the true optimum of the DP problem. The choice of an action a is called greedy if it maximises Q , and ϵ -greedy if we randomly act with small probability ϵ (thereby allowing exploration). The following algorithm considers transitions from state–action pair to state–action pair, i.e. the quintuple of events $(s_t, a_t, R_{t+1}, s_{t+1}, a_{t+1})$, and is therefore called SARSA:

SARSA / TD control for estimating $Q = q^*$

Initialise $Q(s, a)$ and s .

Loop for each step of episode:

 choose action a from s using policy derived from Q (ϵ -greedy)

 observe return/pay-off R and s'



 choose action a' from s' using policy derived from Q (ϵ -greedy)



$Q(s, a) \leftarrow Q(s, a) + \alpha(R + \beta Q(s', a') - Q(s, a))$


$s \leftarrow s', a \leftarrow a'$

Applications: Stochastic growth.

3 Resources

We take a polyglot approach to coding. The algorithms are developed in pseudo-code, and students can implement these in the language of their choice. The interpreted languages such as  and  perform similarly in the problems studied here, while Julia can be much faster because of its just-in-time compilation.

Scientific programming: QuantEcon (<https://quantecon.org/>) is a fantastic resource, and features many topics at different levels of complexity. Although the implementations are in either  and Julia, the algorithms are easily translated into any other language such as .

: If you are beginner, you might wish to consult R4DS (<https://r4ds.had.co.nz/>). RStudio education (<https://education.rstudio.com/learn/>) offers (free) learning tracks for beginners, intermediates, and experts. DataCamp (<https://www.datacamp.com/>) also offers many training videos.

🔗: If you are a beginner, you might wish to consult Jake VanderPlas' Python Data Science Handbook (<https://jakevdp.github.io/PythonDataScienceHandbook/>). QuantEcon (<https://quantecon.org/>) has also several introductions.

Julia : If you are a beginner, you might wish to consult QuantEcon (<https://quantecon.org/>).

📄: For my notebooks, I typically use Rmarkdown, which is a very nice native markdown 📄 implementation in RStudio, see the webbook (<https://bookdown.org/yihui/rmarkdown/>), which can also be made to work with 🔗 using the `reticulate` package. Alternatively, I use the Jupyter (<https://jupyterlab.readthedocs.io/en/stable/user/notebook.html>) ecosystem to develop my 🔗 code. Jupyter also has a R kernel (<https://github.com/IRkernel/IRkernel>), as well as a Julia kernel.

Further reading. The leading textbook for Reinforcement Learning (<http://incompleteideas.net/book/RLbook2020.pdf>) is the eponymous title by Sutton and Barto. If you want to learn more about the methods of Topic 1, the (web)book Introduction to Statistical Learning (<https://www.statlearning.com/>) is very insightful.