The Journal of Open Source Education

# CFD Python: the 12 steps to Navier-Stokes equations

**Lorena A. Barba**[1] **and Gilbert F. Forsyth**[1, 2]

**1** The George Washington University **2** Capital One

**DOI:** 10.21105/jose.00020

**Software**

- Review
- Repository
- Archive

**Submitted:** 01 July 2018
**Published:** 01 July 2018

**License**
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC-BY).

## Summary

The **CFD Python** learning module is a set of Jupyter notebooks, consisting of 12 "core" lessons, 3 "bonus" lessons, and a "lesson zero" as a quick intro to Python for numerical computing. This practical module takes students through 12 steps, one by one, at the end of which they will have programmed a solver for the two-dimensional Navier-Stokes equations, using finite differences. The steps are the following:

- Steps 1–4 are in one dimension: (i) linear convection with a step-function initial condition (IC) and appropriate boundary conditions (BC); with the same IC/BCs: (ii) nonlinear convection, and (iii) diffusion only; (iv) Burgers' equation, with a saw-tooth IC and periodic BCs.
- Steps 5–10 are in two dimensions: (v) linear convection with a square function IC and appropriate BCs; with the same IC/BCs: (vi) nonlinear convection, and (vii) diffusion only; (viii) Burgers' equation; (ix) Laplace equation, with zero IC and both Neumann and Dirichlet BCs; (x) Poisson equation in 2D.
- Steps 11–12 solve the Navier-Stokes equation in 2D: (xi) cavity flow; (xii) channel flow.

Guiding the students through these steps (without skipping any), they learn many valuable lessons. The incremental nature of the exercises means they get a sense of achievement at the end of each assignment, and they feel they are learning with low effort. As they progress, they naturally practice code re-use and they incrementally learn programming and plotting techniques. As they analyze their results, they learn about numerical diffusion, accuracy and convergence. In about four weeks, they become moderately proficient programmers and are motivated to start discussing more theoretical matters.

## Statement of need

Many university courses in computational fluid dynamics (CFD) follow a similar order of topics. As reflected in various textbooks, the course usually starts with the subject of interpolation, going on to numerical integration of ordinary differential equations, and continuing to standard material on partial differential equations. Students encounter theory of stability, order of convergence, and numerical diffusion, but they often do not gain much experience programming. This learning module places emphasis on practical experience with *programming* the solution to fundamental mathematical models that can represent fluid behavior. It is unique in its approach of taking a beginner in a step-by-step fashion to complete the solution of a fairly complex numerical problem: two-dimensional cavity flow via the Navier-Stokes equations, discretized with finite differences.

The "12 steps to Navier-Stokes" lessons have proved effectiveness. They were used in the classroom as part of a university course for four years in a row (Boston University, 2009 to 2013), guiding several dozen students to develop their Navier-Stokes solutions. Written as a set of Jupyter notebooks, the module was the backbone of an intensive tutorial held as part of the 2013 Latin American Symposium on High-Performance Computing in Mendoza, Argentina (http://ecar2013.hpclatam.org).

The module is complete and can be easily adopted by other instructors who wish to teach CFD usting a practical approach. An instructor can complement the lessons with brief presentations, class discussions, and readings. For example, after students experiment with different values of the parameters in the first two steps, they will encounter situations when the solution blows up, due to numerical instability. As they become perplexed of this behavior, the instructor can use the "bonus" lesson about the stability criterion of Courant-Friedrichs-Lewy (CFL), and complement with a brief lecture on the concepts of consistency, stability and convergence.

We know that instructors around the world have already adopted these lessons: many of them have written us with expressions of gratitude (or with improvement suggestions). The collection was translated to Spanish by a volunteer (Fran Navarro) in 2015 (https://github.com/franktoffel/CFDPython-ES), another user (Manuel Ramsaier) made screencast videos with Matlab versions of the lessons (https://youtu.be/QOeTk6C6dZI), and a professor in Singapore (Claus-Dieter Ohl) incorporates the lessons in his local CFD course (http://cav2012.sg/cdohl/CFD_course/). After Barba left Boston University, the instructor who took over the CFD course there continued to use the module. She writes: "The 12 steps are great for building a student's computational toolbox, building her understanding of numerical methods, and providing some benchmark solutions for validating full CFD simulations. I use these 12 steps in the beginning weeks of my graduate level CFD course every year. They help lay the perfect foundation for further analysis and use of available CFD tools." (Prof. Sheryl Grace, 2018.)

The lessons are also often mentioned in posts on sites like CFD Online, Quora, and others, and they are cited in a SciPy Conference paper (Ketcheson 2014) and a book (Rossant 2018). A portion of the module was translated to use the Devito package (a finite-difference code-generation tool based on SymPy), as part of their tutorials. Various translations of the lessons to other programming languages can be found online.

## Notes on instructional design

Based on the experience of the authors after developing the "CFD Python" learning module, and other modules following a similar approach, we adopted this basic design pattern for creating lessons using *computable content*:

1. Break it down into small steps
2. Chunk small steps into bigger steps
3. Add narrative and connect
4. Link out to documentation
5. Interleave easy exercises
6. Spice with challenge questions/tasks
7. Publish openly online

Regarding the inclusion of step-by-step, incremental code exposition, some critics are hesitant with "giving the solution" to the learners, and propose an alternative approach where the learners have to struggle and build their code solutions from scratch. When teaching novices, however, several studies show significant learning improvements when

using worked examples, versus problem-solving with no guidance. This is called the "worked-example effect," and it is a widely studied cognitive-load effect (Sweller 2006, Chen, Kalyuga, and Sweller (2015)). It applies particularly when teaching complex material to novice learners, resulting in heavy working-memory load: strong guidance is likely to result in enhanced performance, in this case.

## References

Chen, Ouhao, Slava Kalyuga, and John Sweller. 2015. "The Worked Example Effect, the Generation Effect, and Element Interactivity." *Journal of Educational Psychology* 107 (3). American Psychological Association:689. https://doi.org/10.1037/edu0000018.

Ketcheson, David I. 2014. "Teaching Numerical Methods with IPython Notebooks and Inquiry-Based Learning." In *Proceedings of the 13th Scipy Conferenece*.

Rossant, Cyrille. 2018. *IPython Interactive Computing and Visualization Cookbook: Over 100 Hands-on Recipes to Sharpen Your Skills in High-Performance Numerical Computing and Data Science in the Jupyter Notebook*. Packt Publishing Ltd.

Sweller, John. 2006. "The Worked Example Effect and Human Cognition." *Learning and Instruction*. Elsevier Science. https://doi.org/10.1016/j.learninstruc.2006.02.005.