Introduction to Computational Physics

PHYS 250 (Autumn 2018) – Lecture 2

David Miller

Department of Physics and the Enrico Fermi Institute
University of Chicago

October 4, 2018

Outline

Computational Physics (PHYS 250)

Course Description PHYS 250 (link to Course Catalog)

This course introduces the use of computers in the physical sciences. After an introduction to programming basics, we cover numerical solutions to fundamental types of problems, including cellular automatons, artificial neural networks, computer simulations of complex systems, and finite element analysis. Additional topics may include an introduction to graphical programming, with applications to data acquisition and device control.

There are an infinite number of paths that we might follow and still not deviate from this description. I therefore would like to lay out some of the principles that will guide me, and us, in how we navigate through those many possibilities.

Outline

Version control

- The most important message of this slide is simple...**Use a software version control system for all of your code**
  - And that means now...not tomorrow or next week
  - Because if you wait until you need it, it will be too late

Version control

- The most important message of this slide is simple...**Use a software version control system for all of your code**
  - And that means now...not tomorrow or next week
  - Because if you wait until you need it, it will be too late

/Users/fizisist/Pictures/Random/Git-Logo-2Color.png

/Users/fizisist/Pictures/Ra

A brief history of version control

- **The first version control systems were designed to be used on large systems where everyone logged into the same machine**
  - They tracked code on the same filesystem where it lived (e.g., in a subdirectory)
  - SCCS and RCS are examples
- **Then client-server systems were developed, so that developers could work on their own machines**
  - Checking code into a central server to share and collaborate
  - CVS and SVN are examples
- **More recently distributed version control systems have arisen**
  - These are decentralised, so everyone has a complete copy of the repository
  - Gives a lot of freedom to developers to share and merge as they like, so liked very much by the open source community
  - git, mercurial and bit keeper are examples

git, ***GitHub***, & ***GitLab***

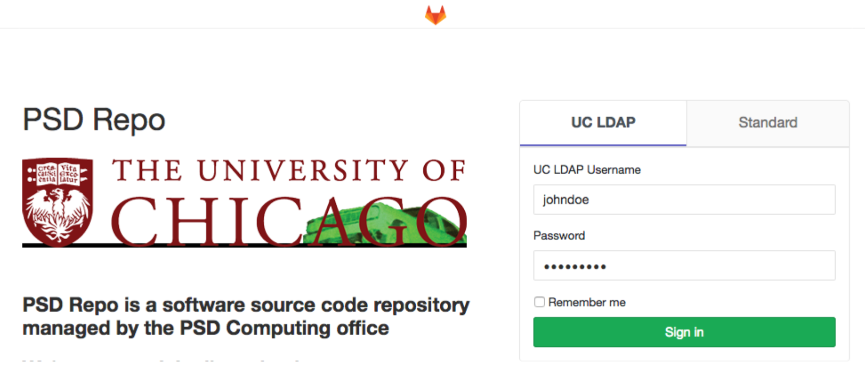
https://git-scm.com, *https://github.com*, *https://about.gitlab.com*

- **`git` is the most popular open source version control system**
  - can host huge projects (Linux Kernel, LHC experiment software, etc)
  - scales very well and it's extremely fast and powerful
  - very flexible (= complex)
- **Distributed version control systems (`git`) are great, but they're made even better by using a social coding site (`GitHub` or `GitLab`)**
- **These sites allow developers:**
  - browse code easily
  - compare different versions
  - take copies (a.k.a. fork)
  - offer patches back to upstream repositories
    - And discuss and review these patches before acceptance
  - even build websites
- **The best known social coding site is GitHub, but there are others, e.g. BitBucket and GitLab**
  - Familiarity with **`git`/`GitHub`/`GitLab`** will serve you well, trust me

GitHub & GitLab *resources*

GitHub is a free resource as long as your code remains public (you can pay for private repositories). The Physical Sciences Division (PSD) at UChicago hosts a **private** **GitLab** repository.

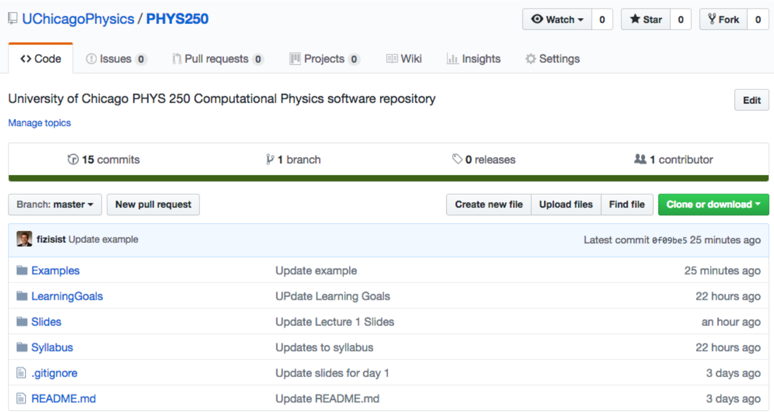
- https://psdcomputing.uchicago.edu/page/psd-repo



PHYS 250 **GitHub**

https://github.com/UChicagoPhysics/PHYS250

Course materials are hosted in the **GitHub** UChicagoPhysics repository



- Slides (e.g. *these!*), syllabi, learning goals, and code examples
- Stable versions will be cross-posted to Canvas as well.
- Homework submission will be done via **GitHub** (*instructions to come*)

Linux "shell"

- We will be using an interface to Linux called a "shell"
- It is a command-line interpreter : you type, it executes
- Two major options are `bash` (as in, smash) and `csh` (like "sea shell", modern version is "tcsh", "tea sea shell")
  - Only real difference: environment variables syntax
  - `bash: export X=value`
  - `csh: setenv X value`

Shell basics

Listing directory contents : `ls`, like "list"

Copy: `cp`

Where am I?: `pwd`, `cd`

Hello world!

Interactive in the python interpreter

From a script (containing the above print line):

Self-running script:

Lists (I)

In my opinion, python's great advantage is **list comprehension**.

List basics

Append elements

Concatenation

Removal of elements

Lists (II)

Element acces read/write

Test if an element is in a list (or not)

`for` and `while` loops

The `for` statement iterates through a collection, iterable object or generator function.

The `while` statement merely loops until a condition is `False`.

Iterate over list

Iterate using built-in `range` function

Iterate until a condition is met

Putting lists and loops together is amazing (and complex)

Filter one list into another (the "old" way)

List comprehension (the "pythonic" way)

where `filter` and `function` just perform "some" operation on the list elements. Basically, the syntax is:

and this replaces:

Useful list comprehension

Filter one list into another (the "old" way)

List comprehension (the "pythonic" way)

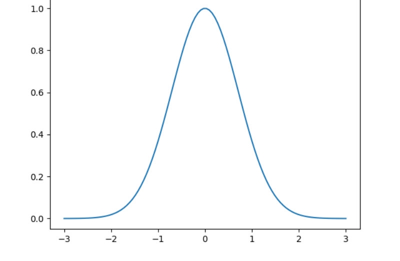
where `filter` and `function` just perform "some" operation on the list elements. Basically, the syntax is:

and this replaces:

Hello Gaussian!

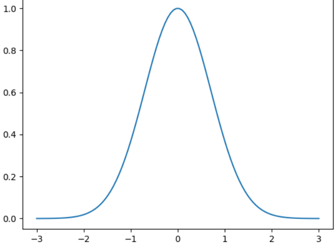
Basic but useful code example

Hello Gaussian!



Basic but useful code example

Hello Gaussian!



Basic but useful code example

But what about that `linspace` thingy? Google it! (numpy docs)

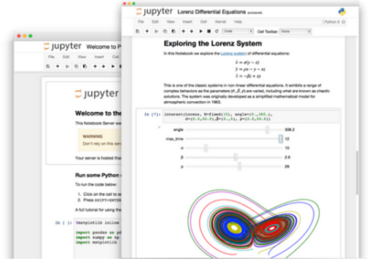
```
numpy.linspace(start, stop, num=50, endpoint=True,
retstep=False, dtype=None)
```

"Returns num evenly spaced samples, calculated over the interval [start, stop]."

Jupyter notebooks

Interactive, web-based, integrated code and documentation environment

We will be following-up with more technical practice with python, but I want to introduce you to the resources that we'll be using this quarter for many of our examples and projects: Jupyter notebooks.



The Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Try it in your browser | Install the Notebook

Language of choice

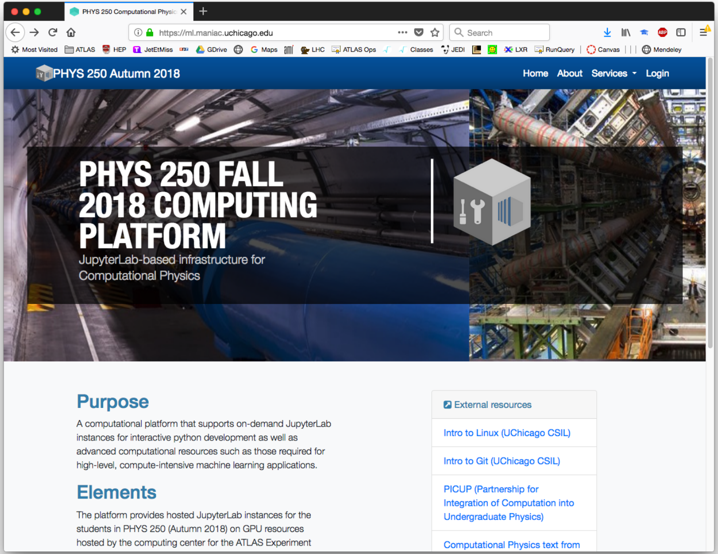
Share notebooks

Interactive output

Big data integration

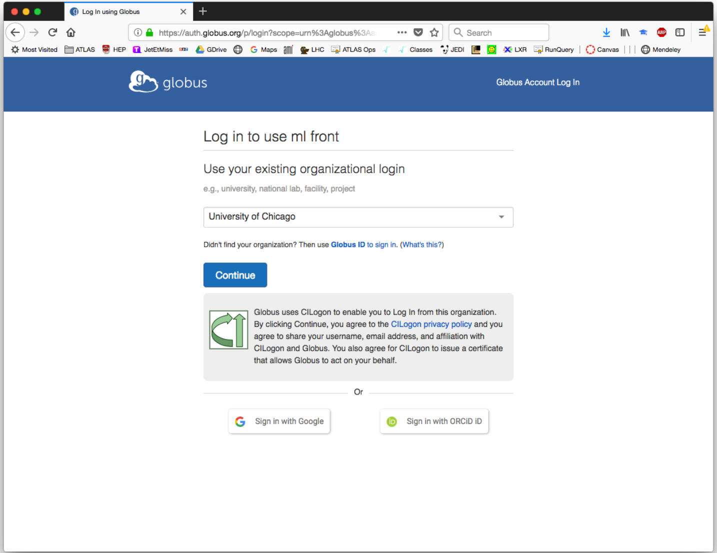
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



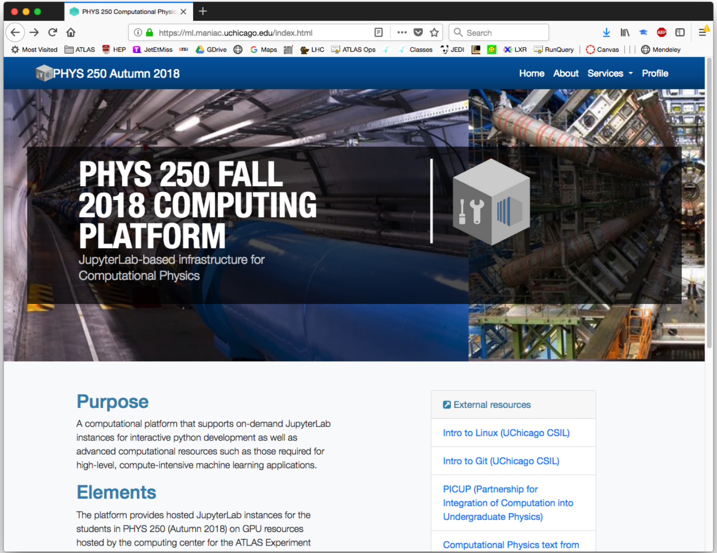
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



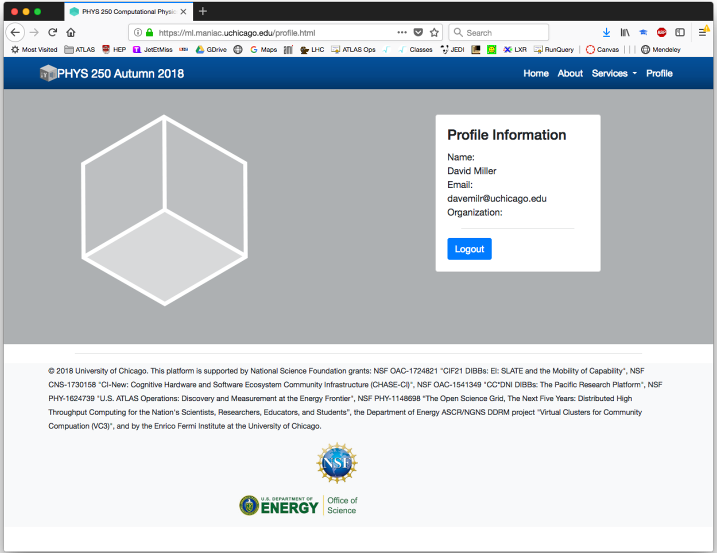
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



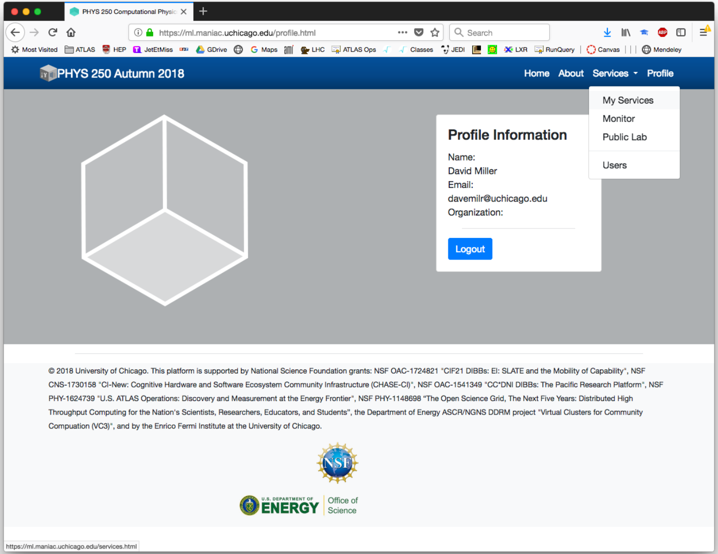
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



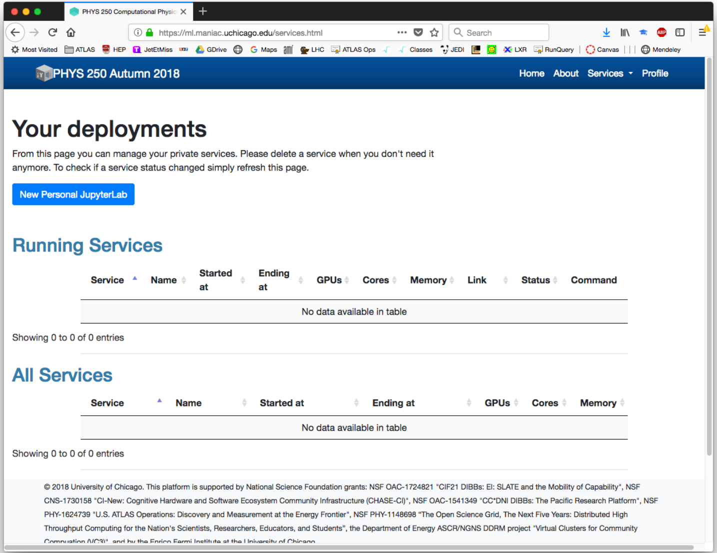
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



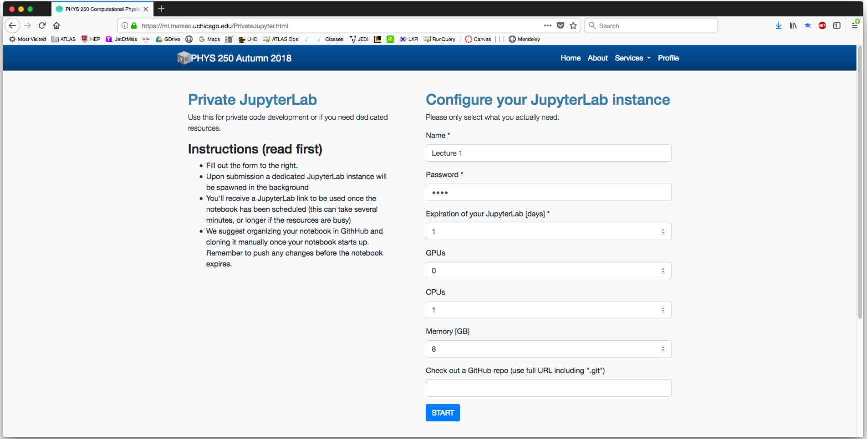
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



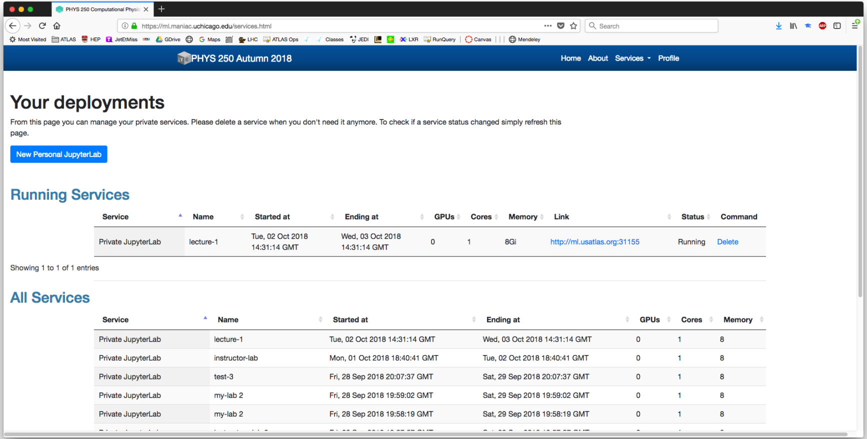
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



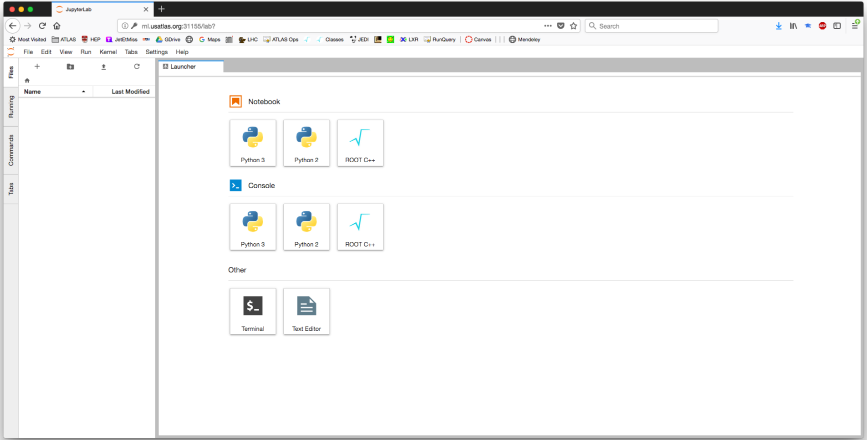
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



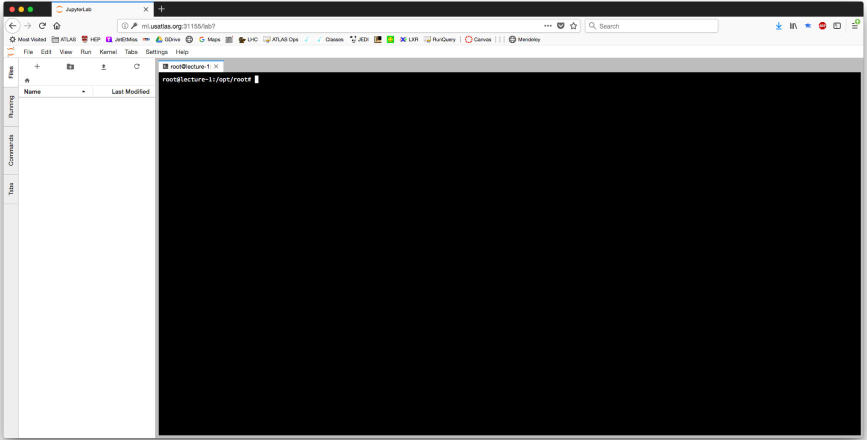
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



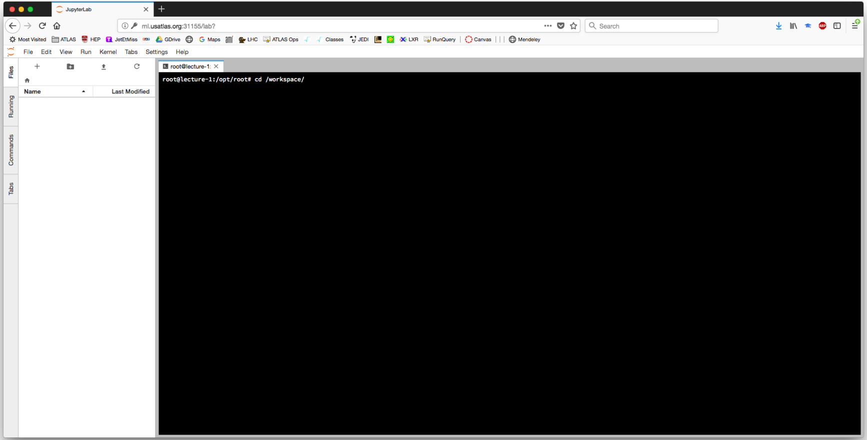
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



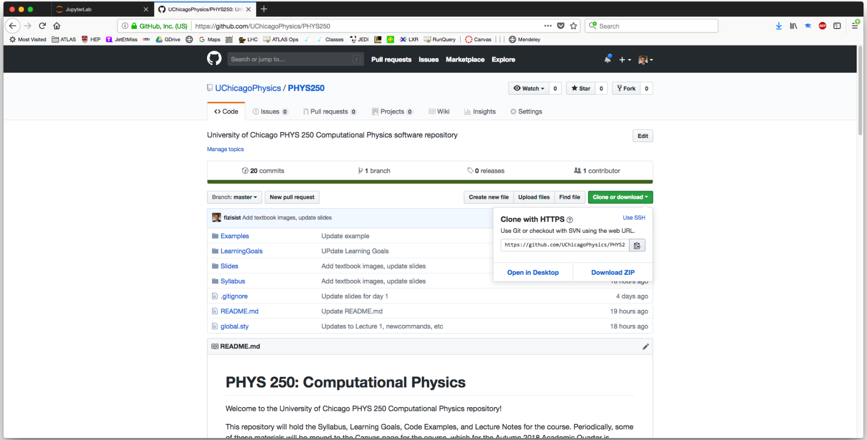
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



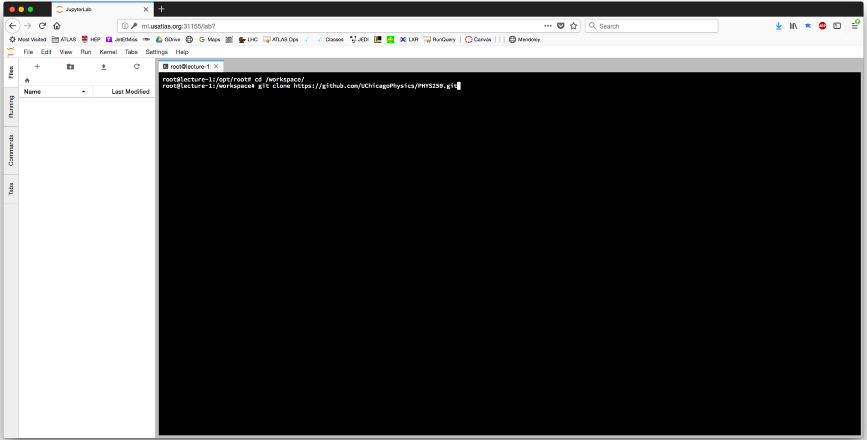
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



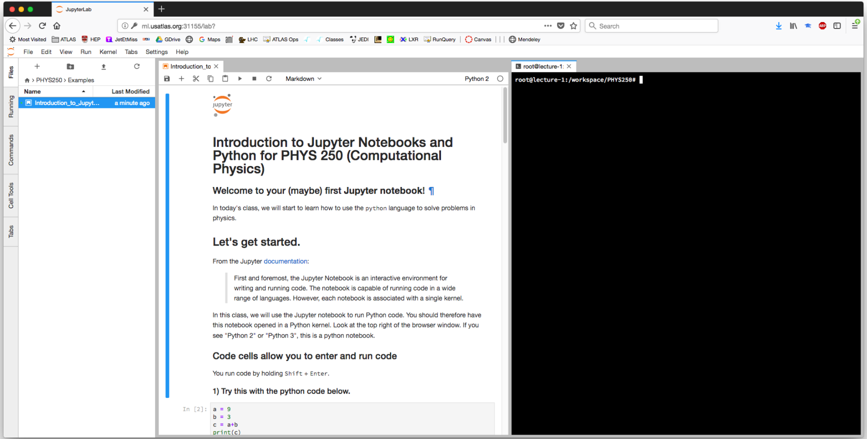
Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



Jupyter notebooks on the PHYS 250 computing platform

Built for machine learning applications running mostly in Jupyter.



Outline

Random Numbers

Random numbers may seem innocuous, but they underlie nearly everything you do electronically, rely on for security, or employ in the context of simulations and evaluation of models.

- **cryptography**
- **computer simulation**
  - the most well version of which is named after the gambling town, **Monte Carlo**
- **sampling**

As such, their use, their control, and the evaluation of just how random they are, is of paramount importance for computational physics.

And you thought lava lamps were a thing of the past

Cryptography relies on the ability to generate random numbers that are both unpredictable and secret. But **random** is a very malleable term.

True vs. Pseudo: it's more than semantics

The computers of today are, by design and by implementation, ***deterministic***. That means that an **algorithm cannot**, on its own and without input from an external source, provide a stream of 100% uncorrelated random sequences. Hence the division:

- **TRNGs: True** random number generators
  - generally use some physical process that is unpredictable, but often slow and requires specialized hardware
  - possibly combined with some compensation mechanism that might remove any bias in the process
  - examples include: lava lamps, quantum processes, radioactive decays, thermal noise
- **PRNGs: Pseudo-**random number generators
  - based on algorithms and, therefore, not truly *random*
  - do not require special hardware and therefore are very portable and fast
  - can be reproduced given initial conditions

We will, perhaps obviously, focus on PRNGs.

(but if you want to build a wall of lava lamps in KPTC, I will cheer you on)

Pseudo-random number generators (PRNGs)

The idea behind an algorithmic PRNG is to generate a sequence of numbers, $x_1, x_2, x_3...$ using a recurrence of the form

$$x_i = f(x_{i-1}, x_{i-2}, ..., x_{i-n}), \tag{1}$$

where n initial numbers ("seeds") are needed to begin the recurrence. The magic lies in the function, f, used, and the resulting *uniformity* and *correlation length* across some sequence of numbers.
Linear congruential generators (LCGs) are one of the oldest PRNGs and have the form

$$x_{i+1} = (ax_i + c) \mod m \tag{2}$$

IBM mainframes in the 60s had $a = 65539$, $c = 0$, and $m = 2^{31}$. This leads to

$$x_{i+2} = 6x_{i+1} - 9x_i \tag{3}$$

which, maybe you can tell, isn't great.

Python random number generator: Mersenne Twister

Python comes with a built-in (`stdlib`) random number generator that can be accessed with:
In addition, the `numpy` package provides an even more developed set of tools with `numpy.random`.
Both of these use the Mersenne Twister algorithm, which was developed in 1997 by Matsumoto and Nishimura and is a version of a generalized feedback shift register PRNG. The name comes from the fact that the period is given by a Mersenne prime

$$M_n = 2^n - 1, n \in \mathbb{N} \tag{4}$$

(In preparing this, I found out that the largest known prime number $2^{77,232,917}1$ is a Mersenne prime, found on December 26, 2017.)