
python-for-scientists Documentation

Release 0.1

Zachary Sailer

Apr 20, 2019

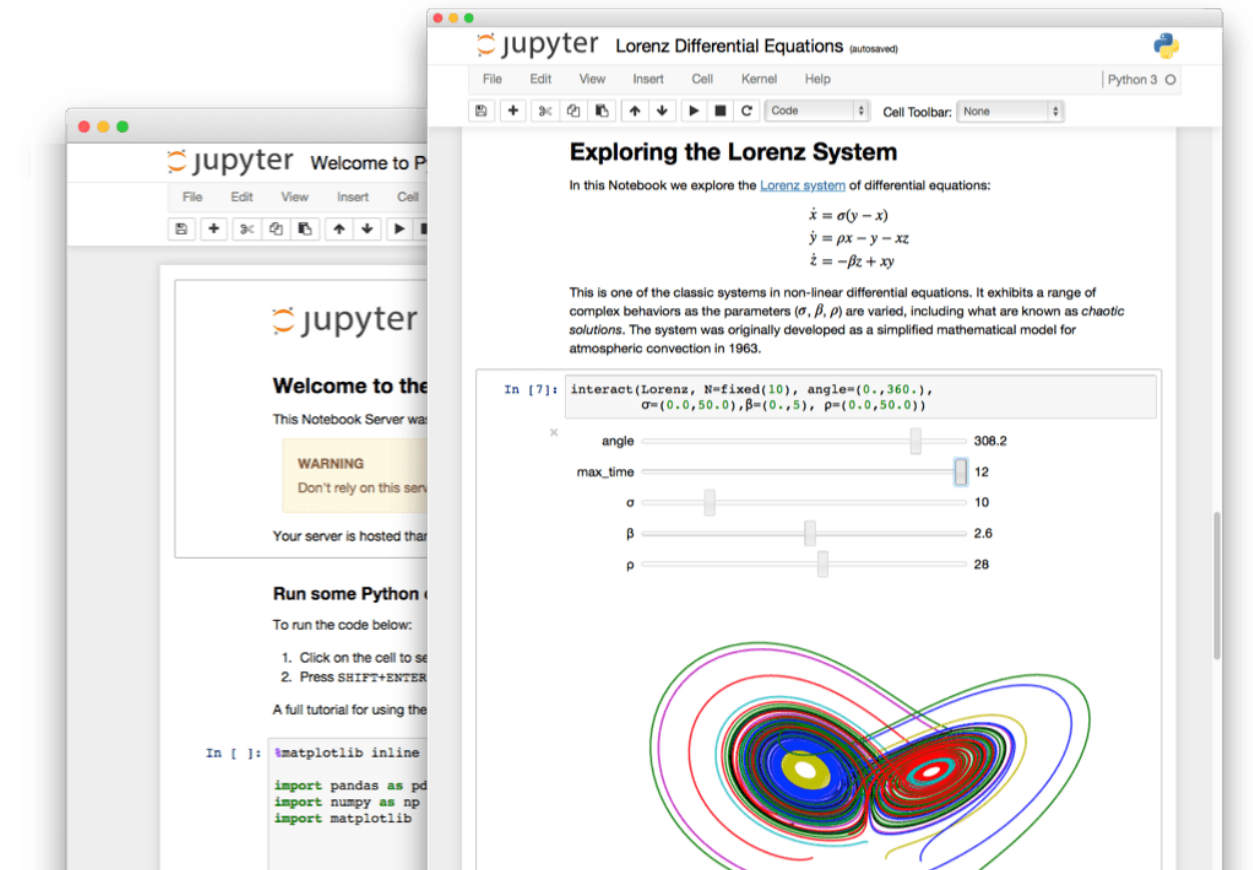
Setting up Scientific Python

1	4 steps to Python	3
2	Virtual Environments	7
3	ipython Kernels	11
4	Frequently Asked Questions	13
5	Plotting Packages	15
6	Core Scientific Packages	17
7	Advanced Scientific Packages	19
8	Was this page helpful?	25

For many scientists, the open-source nature of Python is intimidating. They'd like to use Python and Jupyter notebooks, but they don't know how to begin. Proprietary software like Matlab and Mathematica comes pre-packaged and ready out-of-the-box, but Python is less straightforward. Our goal is to provide a quick guide to help scientists get started.

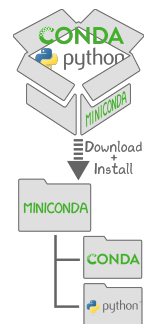
4 steps to Python for scientists:

1. Install Python using Miniconda
2. Install the Jupyter Notebook
3. Install core scientific packages
4. Run the Jupyter Notebook



1. *Install Python using Miniconda*
2. *Install the Jupyter Notebook*
3. *Install core scientific packages*
4. *Run the Jupyter Notebook*

1.1 1. Install Python using Miniconda



We recommend [Miniconda](#) (*even if you have Python already installed*).

Miniconda installers contains Python and the [Conda](#) package manager. [Installers](#) exist for Windows, Mac, and Linux. Download the installer that matches your operating system and follow the directions to install miniconda on your computer. Except in very special circumstances, we recommend installing Python 3. ([Why?](#))

Once installed, Miniconda becomes a folder that contains everything Python and Conda related. If anything ever goes wrong with your Python setup, you can always remove the Miniconda folder and start over.

Conda will be your main tool for installing Python packages moving forward. It installs packages through a mechanism known as *conda-recipes*. (It's important to note, it can do **'many other'** things as well.) Conda also manages package updates and creates virtual environments.

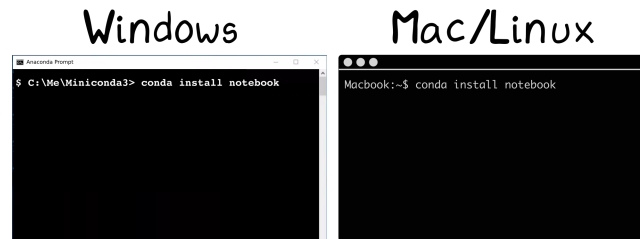
pip is another tool for installing Python packages (and comes with Miniconda). If a conda-recipe does not exist for a package, you can try installing using *pip*.

1.2 2. Install the Jupyter Notebook.

To install new Python packages using Conda, you'll need to use a [command line](#).

On Windows, use the new **Anaconda prompt** application installed by Miniconda.

On Mac and Linux, use the **Terminal** application.

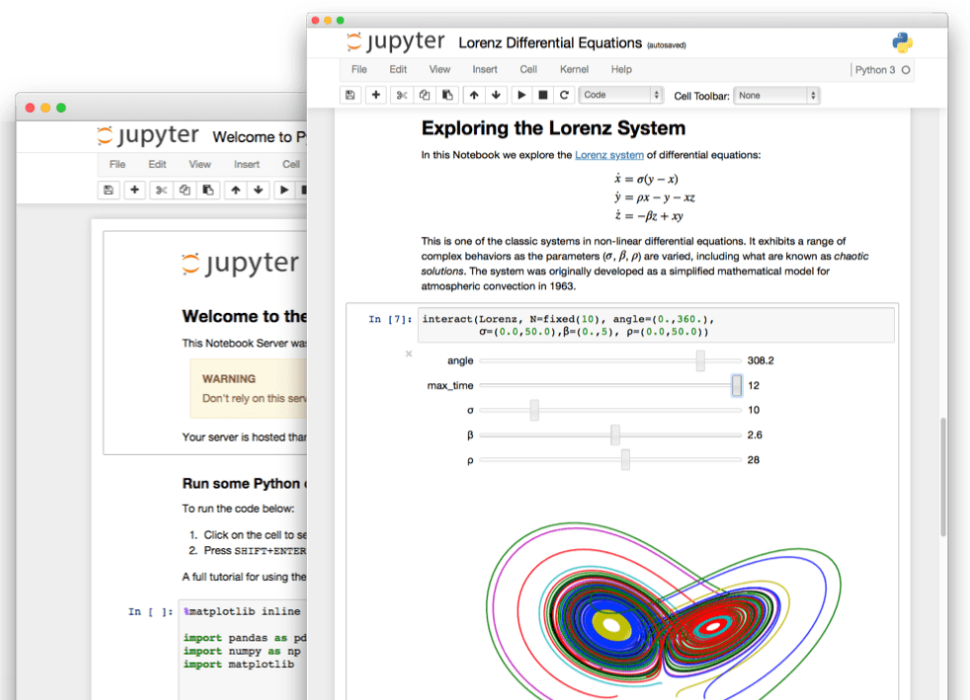


In general, you'll type `conda install some_package_name` to install new packages. If conda cannot find the package, try using **pip**.

```
# Installing with conda
> conda install some_package_name

# Installing with pip
> pip install some_package_name
```

1.3 3. Install core scientific packages



Before you run Jupyter, you'll want to install some core scientific packages. These packages allow you to produce awesome notebooks like the one shown to the right.

- [Matplotlib](#) is Python's most popular plotting package.
- [Numpy](#) and [SciPy](#) are Python's fundamental packages for scientific computing.

```
> conda install numpy scipy matplotlib
```

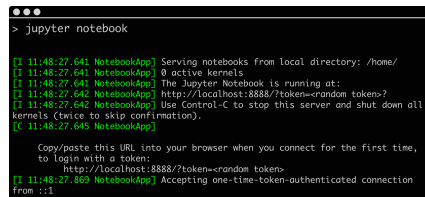
A more comprehensive list of Python packages for scientific computing can be found [here](#).

1.4 4. Run the Jupyter Notebook

Open Jupyter Notebooks using the command line again (*why?*). Run the following command:

```
> jupyter notebook
```

Your output on the command line will look something like this:

A terminal window with a black background and green text. The prompt is '> jupyter notebook'. The output shows the Jupyter Notebook application starting, serving notebooks from the local directory, and providing a URL to access the notebook in a browser. The URL is 'http://localhost:8888/?token=random token'. The output also includes instructions to use Control-C to stop the server and shut down all kernels, and to copy/paste the URL into a browser for the first time.

```
> jupyter notebook
[I 11:48:27.641 NotebookApp] Serving notebooks from local directory: /home/
[I 11:48:27.641 NotebookApp] 0 active kernels
[I 11:48:27.641 NotebookApp] The Jupyter Notebook is running at:
[I 11:48:27.642 NotebookApp] http://localhost:8888/?token=random token?
[I 11:48:27.642 NotebookApp] Use Control-C to stop this server and shut down all
kernels (twice to skip confirmation).
[C 11:48:27.645 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
http://localhost:8888/?token=random token
[I 11:48:27.869 NotebookApp] Accepting one-time-token-authenticated connection
from ::1
```

And the notebook application will launch in a browser window.

1. *Default Conda Environment*
2. *Creating a Python 2.7 Environment*
3. *Creating a TensorFlow Environment*
4. *Creating and using a Development Environment*

You may be wondering:

What is a virtual environment?

Why would you want a virtual environment?

2.1 1. Default Conda Environment

Here is a graphical representation of what you have when you start out using conda:



This is your default python environment. It uses python 3.6 and has any packages that we've installed using either the *pip* or *conda* commands.

2.2 2. Creating a Python 2.7 Environment

This is all well and good, but what if you need to use python 2.7 for a particular application or problem? This is an excellent opportunity to use a **virtual environment** in conda. A virtual environment creates a copy of your miniconda environment with a specific python version and only the packages you want.

This is how you create a virtual environment using conda:

```
conda create -n python2 python=2.7 matplotlib pandas
```

The field after `-n` is the name of your environment, the `python=` flag is where you specify your python version, and you can add package names that you already have installed in your default miniconda.

Here is the result of creating our python 2.7 virtual environment:



In order to use this environment you will have to activate it:

```
# Old conda
source activate python2

# New conda
conda activate python2
```

And when you want to switch back to your default:

```
# Old conda
source deactivate

# New conda
conda deactivate
```

2.3 3. Creating a TensorFlow Environment

Now say that you want to install TensorFlow, but you don't want to accidentally kill your default python by installing it or you want to make sure that you can easily uninstall it later. A virtual environment is great for this purpose too.

```
conda create -n Tensorflow python=3.6 numpy
```

Now activate the environment

```
conda activate Tensorflow
```

Then install tensorflow

```
pip install tensorflow
```

Now you're available environments will look like this:



2.4 4. Creating and using a Development Environment

One more reason that you might want a virtual environment is for developing your own packages. Say you've got a package called `"test"` that you want to test out as you develop it. Create a virtual environment with the packages you need and then install your package with pip in editable mode.


```
# Create the environment
conda create -n Test python=3.6 pandas matplotlib

# Activate this new environment
conda activate Test

# Then install your local package
pip install -e /path/to/your/package/test
```

Now your available environments will include your test development environment.



2.5 What is a virtual environment?

A virtual environment is a self-contained version of Python and specified packages. When you switch to a different virtual environment conda points to that python installation and installed packages. A package installed globally but not in that virtual environment won't show up.

2.6 Why would you want a virtual environment?

Virtual environments are a good way to protect yourself. Say you accidentally install or delete something, if you're in a virtual environment you can delete it and start over without reinstalling Python.

CHAPTER 3

ipython Kernels

1. *Going from virtual environments to kernels*
2. *Making an ipython kernel*

3.1 1. Going from virtual environments to kernels

Virtual environments are a nice way to compartmentalize your coding environment, but you can't make use of them in a jupyter notebook automatically.

What you need to do is create a kernel that is associated with each virtual environment. *What is a kernel?*

The cool thing is that once you create a kernel you don't have to change virtual environments to use them in the jupyter notebook!

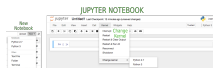
3.2 2. Making an ipython kernel

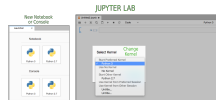
```
# Activate your virtual environment
source activate Python-2.7

# Install ipykernel
pip install ipykernel

# Create your ipykernel
python -m ipykernel install --user --name Python-2.7 --display-name "Python 2.7"
```

Now this kernel can be used in a jupyter notebook or jupyter lab without having to activate the associated virtual environment.





And this reflects our available virtual environments that we set up before.



3.3 What is a kernel?

A kernel is the engine that actually runs your code. Using Jupyter you can have a kernel for each virtual environment and even kernels for languages other than Python.

Frequently Asked Questions

4.1 How do I upgrade a python package that is already installed?

Sometimes you need to upgrade to a new version of a package that is already installed in your python environment.

To upgrade a package in your python environment, just use conda! For example to upgrade the scipy package library run the following:

```
conda update scipy
```

4.2 Should I install Python 2 or 3?

Python 3. Most scientific Python libraries now support Python 3 and it's a better language overall. Do not install Python 2 unless you absolutely must use it for some core dependency in your daily work.

4.3 What if my operating system already comes with Python installed?

It's best not to mess with the native Python on your machine. On Macs, many programs depend on that native Python. If you break it, you can break your Mac.

4.4 What is the difference between conda and pip?

conda— is command line tool that downloads conda-recipes from a repository hosted by [Continuum Analytics](#). One strength is that conda-recipes can be written for any programming language (not just Python). This is extremely useful for projects like Jupyter who weave Javascript tools with Python backends. Conda is quickly becoming a widely used

installer for this reason. Further, developers at Continuum are working out ways to make **pip** and **conda** work together seamlessly.

pip— a command line tool that downloads Python packages from PyPI, Python’s Packaging Index. It is designed to manage Python packages only.

conda is a younger than pip, so many Python packages don’t have conda-recipes but exist on PyPI.

4.5 Why do I need to launch Jupyter from the command line?

The answer is a bit complicated. Jupyter is web application frontend (written in Javascript) that executes code from a Python server backend. The command line is used to launch the server backend and open internet ports for the frontend. This frontend is rendered using a browser (like Google Chrome or Firefox). The command line interface makes this process transparent. You see the server start, then a browser window opens. The command line also provides flexibility for advanced users.

If you’d like a Jupyter-like Desktop application, it exists. It’s called **nteract**. You can open Jupyter notebooks without the command-line. [It can be found here!](#)

4.6 Why is Matplotlib a source of friction among computational scientists?

Plotting Packages

The Python visualization landscape is fairly diverse. We'll list a few of the major plotting packages here:

- *Matplotlib* — Python's most used plotting library.
- *Seaborn* — Statistical plotting functions and better matplotlib aesthetics.
- *Pandas.plot* — Declarative plotting directly from a Pandas DataFrame.
- *Altair* — Declarative plotting powered by Vega.
- *pdvega* — Vega-lite plots from Pandas DataFrames.

5.1 Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.

Why all the negativity around Matplotlib?

[Docs](#) | [Source](#) | **Gallery**

5.2 Seaborn

Seaborn is a library for making attractive and informative statistical graphics in Python. It is built on top of matplotlib and tightly integrated with the PyData stack, including support for numpy and pandas data structures and statistical routines from scipy and statsmodels.

[Docs](#) | [Source](#) | **Gallery**

5.3 Pandas.plot

Pandas provides a **ggplot** like API for creating plots from Pandas DataFrames. This module is built on top of Matplotlib.

[Docs](#) | [Source](#) | [Gallery](#)

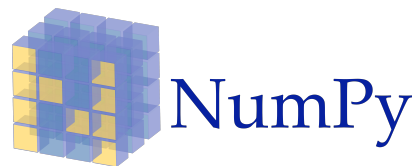
5.4 Altair

5.5 pdvega

CHAPTER 6

Core Scientific Packages

This section introduces you to the core numerical libraries in Python. These libraries are essential for scientific computing in a Python environment.



NumPy introduces arrays to python. Numpy arrays are an essential tool for scientific computing in Python. Arrays are an efficient way to perform computations on large datasets. A wide variety of functions for manipulating arrays and performing linear algebra calculations are included in NumPy.

To install NumPy in your python environment simply run:

```
conda install numpy
```



SciPy contains a broad range of useful tools for scientific computing including optimization functions, mathematical transforms, distance calculations, statistical tools, and image processing applications.

To install SciPy in your python environment simply run:

```
conda install scipy
```



Pandas brings the feel of spreadsheets to your python computing environment! Pandas provides powerful, easy-to-use data structures and analysis tools to help you handle your datasets. Pandas makes importing, navigating, and manipulating datasets easy!

To install pandas in your python environment simply run:

```
conda install pandas
```

Advanced Scientific Packages

This page introduces you to a set of powerful Python libraries for advanced numerical computing. Python has libraries for machine learning, model fitting, statistics, network calculations, and much more! Here we highlight the following important scientific libraries:

- scikit-learn — diverse machine learning tools
- scikit-image — image analysis
- LmFit — nonlinear fitting
- Networkx — network analysis
- Biopython — bioinformatics tools
- emcee — Bayesian MCMC
- PyMC3 — Probabilistic programming
- StatsModels — statistics
- Astropy — astrophysics tools
- Cython — simple C extensions
- Numba — just in time compiling for Python
- SymPy — executes symbolic math operations
- TensorFlow — deep learning
- Theano — deep learning
- Keras — deep learning



scikit-learn is a powerful Python library for machine learning. It contains a wide array of useful tools for classification, regression, clustering, dimensionality reduction, and much more. Implementations are available for most standard machine learning algorithms, such as Support Vector Machines and Decision Trees. If you're planning to do machine learning in Python, you'll want to install scikit-learn.

To install this library in your Python environment simply run:

```
conda install scikit-learn
```



scikit-image is a very useful Python library for handling image data. It's a powerful tool for image processing that allows users to identify objects, filter images, manipulate color channels, and a wide variety of other tasks.

To install this library in your Python environment simply run:

```
conda install scikit-image
```

LMFIT

LmFit is a great tool for non-linear curve fitting. It uses SciPy under-the-hood, but offers a better interface. Specifically, it offers more control when estimating model parameters. If you're going to be fitting complex data in Python, we suggest LmFit.

To install this library in your Python environment simply run:

```
conda install lmfit
```

NetworkX

Networkx offers a simple interface for managing network data. It introduces a Graph datatype that is easy and intuitive to use. It also provides various algorithms for analyzing and plotting networks.

To install this library in your Python environment simply run:

```
conda install networkx
```



Biopython is a library built for biological computation. It contains a wide array of bioinformatics tools for handling sequence data, parsing files, searching databases, performing population genetics calculations, and much more.

To install this library in your Python environment simply run:

```
conda install biopython
```



emcee is a Python library for efficiently estimating probability distributions. It uses an efficient MCMC sampling strategy that is often used to approximate posterior distributions in Bayes Theorem. If you're looking to implement a Bayesian fit in your analysis we recommend using emcee.

To install this library in your Python environment simply run:

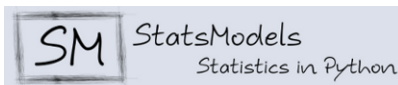
```
conda install emcee
```



PyMC3 is another useful tool for implementing Bayesian inference in your analyses. PyMC3 is a versatile probabilistic programming framework that allows users to define probabilistic models directly in Python. Under the hood it uses a variety of clever tricks to make computations faster.

To install this library in your Python environment simply run:

```
conda install pymc3
```



StatsModels is a versatile statistical environment for Python. It allows users to perform a wide array of statistical tests and analyses. Various regressions are available for model fitting. It also includes tools for plotting and nonparametric statistics. If you'll be implementing a lot of statistics in Python, StatsModels will likely be useful.

To install this library in your Python environment simply run:

```
conda install statsmodels
```



If you're an astrophysicist looking to use Python for your analyses, Astropy is for you. This library implements a range of methods, models, and statistics that are useful for astrophysical data.

To install this library in your Python environment simply run:

```
conda install astropy
```



Cython is another library aimed at speeding up Python code. Users can write Python code and quickly translate it to a C extension.

To install this library in your Python environment simply run:

```
conda install cython
```



Numba is a library designed to help you speed up your Python calculations. It achieves this goal using a just-in-time compiler, which gives Python code speed that is comparable in performance to C. Numba is easy to use. Python functions can be wrapped with a simple decorator that results in increased speed.

To install this library in your Python environment simply run:

```
conda install numba
```



SymPy is a library for doing symbolic math. You can compute integrals, derivatives, algebraic manipulations, etc. Think Mathematica in Python.

To install this library in your Python environment simply run:

```
conda install sympy
```



TensorFlow is a versatile library designed for implementations of deep learning algorithms. If you're looking to use deep neural networks on your data, for example a large-scale image classification problem, then TensorFlow will likely be useful.

To install this library in your Python environment simply run:

```
conda install tensorflow
```

theano

Theano is a library geared toward efficient computations on multidimensional arrays. It also supports implementation of code on GPUs. Theano is useful for implementing deep learning in Python.

To install this library in your Python environment simply run:

```
conda install theano
```



Keras is another Python library for machine learning using neural networks. It is capable of interacting with other machine learning libraries, including TensorFlow and Theano. Keras runs on CPUs and GPUs and is designed for fast implementation of neural networks.

To install this library in your Python environment simply run:

```
conda install keras
```


CHAPTER 8

Was this page helpful?

Let us know by visiting [this page](#) and giving it a “thumbs up”.

Is this webpage helpful? #4

Open Zsailer opened this issue 27 days ago · 0 comments



Zsailer commented 27 days ago • edited ▾

Owner



Is this webpage helpful?

Please give us a thumbs up if you found this page useful! Click on the smiley face in the top right corner of this box!

If you have another topic that you think we should add, feel free to open a new [issue](#)!

Please feel free to open an [issue](#) if:

1. You have a topic you'd like us to add or,
2. You find problems in our instructions.