NumPy User Guide

Release 1.11.1

Written by Joel Frederico

February 02, 2016

Contents

This section is devoted to a user manual with explanations of typical use of various modules.

CHAPTER 1

Introduction

This is the documentation for a Python package designed to make scientific data analysis easier. The main objective is to create a collection of interconnected methods frequently needed to visualize and analyze data using Numpy, Scipy, Matplotlib, and PyQt.

1.1 Prerequisites

1.1.1 Python 3

`scisalt` works with Python 3 and up, which should be installed via apt-get on *nix, Macports on Apple machines, or downloaded from https://www.python.org/downloads/.

1.1.2 HDF5 and h5py

While technically `scisalt` only depends on h5py, `h5py` depends on HDF5. `h5py` can be installed via pip:

```
pip install h5py
```

However, HDF5 may need to be installed at the system level via apt-get or Macports.

It is possible to download or compile HDF5 and h5py from source:

```
* https://www.hdfgroup.org/HDF5/release/obtain5.html
* https://pypi.python.org/pypi/h5py
```

1.1.3 NumPy and SciPy

`scisalt` depends on NumPy and SciPy to manipulate data.

NumPy has dependencies such as BLAS, LAPACK, and ATLAS, which makes downloading building form source is difficult. Installation via apt-get or Macports is highly recommended in order to handle these dependencies. It is possible to download or to build from source.

SciPy is similar to NumPy, and it is usually easiest to install the two at the same time - they have nearly identical install methods, to the point that at times it is difficult to tell the two apart. It is easiest to follow the install instructions.

1.1.4 PyQt4

`scisalt` requires PyQt4, which has dependencies of its own. Installation via apt-get or Macports is highly recommended in order to handle these dependencies.

It is possible, although difficult, to install from source, including dependencies. If you would like to run on a *nix machine without access to apt-get, you may have to compile from source. It is possible to build against Qt 5.

1.2 Installation

There are a few ways to install `scisalt`. If you are unsure or want something more reliable (and also updated less frequently), install from PyPI.

1.2.1 From PyPI

You can install the most recent SciSalt version using pip:

```
sudo pip install scisalt
```

This will install `scisalt` in your Python installation's site-packages directory.

The PyPI site is found at https://pypi.python.org/pypi/scisalt.

1.2.2 From the tarball release

1. Download the most recent tarball from the download page
2. Unpack the tarball
3. `sudo python setup.py install`

Note that you have to have setuptools installed.

This will install `scisalt` into your Python installation's site-packages directory.

1.2.3 Installing the development version

1. Install git (available through Linux's apt-get and Macports as well)
2. `git clone git@github.com:joelfrederico/SciSalt.git`
3. `cd SciSalt`
4. `python setup.py develop`

Note that you have to have setuptools installed.

CHAPTER 2

Visualization

There are several modules that are involved with making GUIs and plots. `scisalt.qt` is concerned with making QT GUIs, mostly for user interaction. `scisalt.matplotlib` assists with all types of plotting.

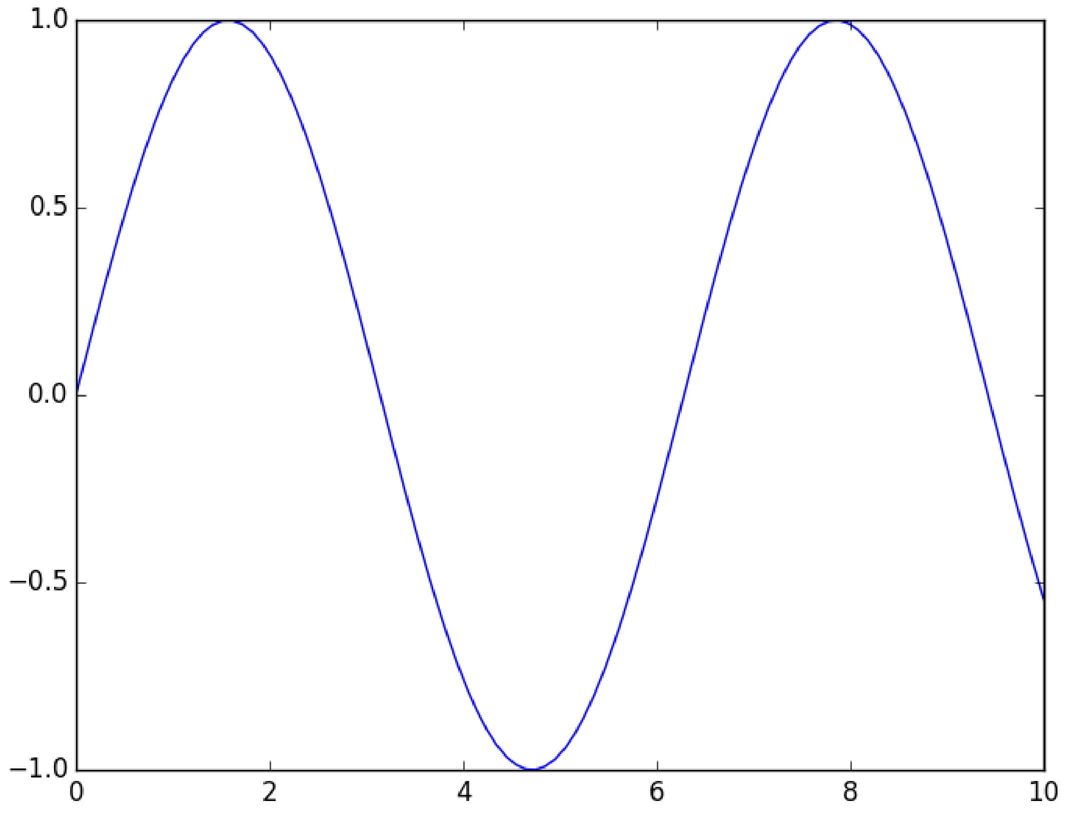
2.1 Matplotlib

Matplotlib is dedicated to making scientific plots. It can also be used for some user interfaces, and is much more friendly than QT.

2.1.1 Introduction

Plotting images can be done in two ways. Matlab style is familiar to Matlab users, but relies on the software figuring out what you mean. For instance, you can create a plot like so:

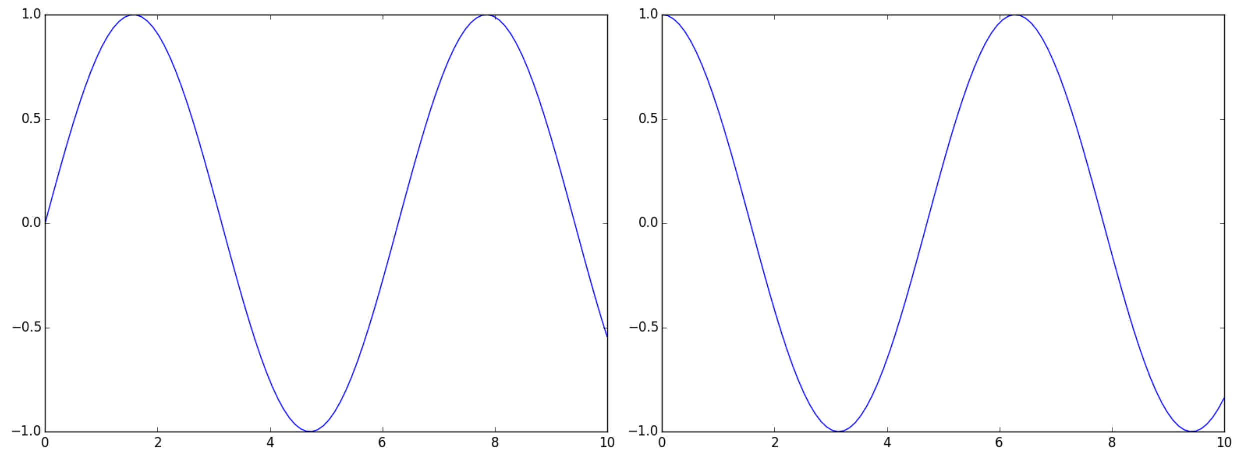
```
import matplotlib.pyplot as plt
import numpy as np

# Create data to plot
x = np.linspace(0, 10, 100)
y = np.sin(x)

# Plot data
plt.plot(x, y)

# Show figure
plt.show()
```



This certainly works, but what if you want to manipulate the figure, the axes, etc.? You have to access the figure and axes objects, and Matplotlib will only access the figure and axis with focus. It's much more Pythonic, and powerful, to access the figures and axes themselves, and things especially become easier to align if you use GridSpec:

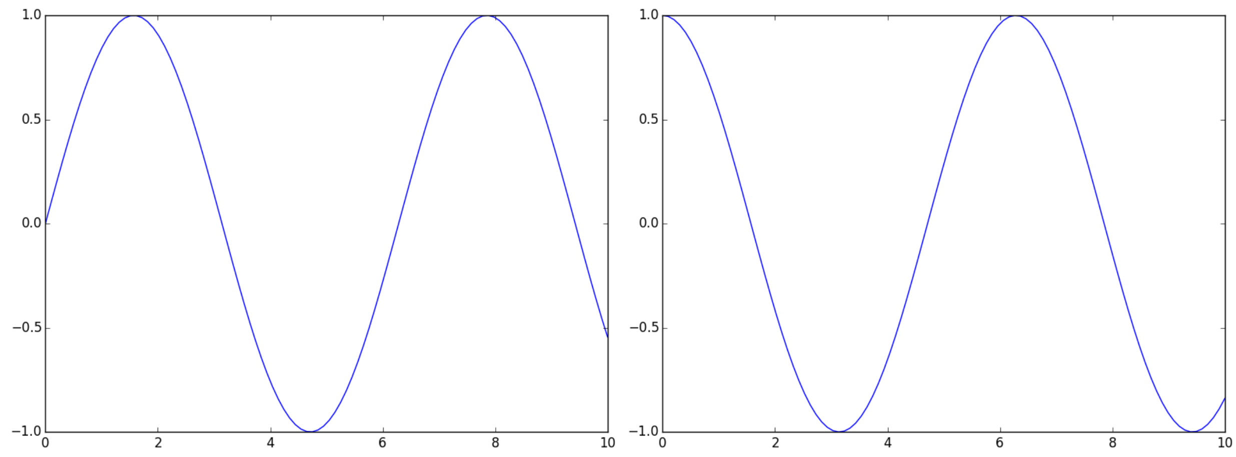
```
import matplotlib.gridspec as gridspec
import matplotlib.pyplot as plt
import numpy as np

# Create data to plot
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

# Create a grid
gs = gridspec.GridSpec(1, 2)

# Create a figure
fig = plt.figure(figsize=(16, 6))

# Create axes
ax1 = fig.add_subplot(gs[0, 0])
ax2 = fig.add_subplot(gs[0, 1])

# Plot data
```

```
ax1.plot(x, y1)
ax2.plot(x, y2)

# Rearrange figure to use all space
fig.tight_layout()

# Show figure
plt.show()
```



While this is great, Pythonic, and very clear as to what is happening, it is also cumbersome! In the course of making plots, this happens a lot. That's exactly why this module exists. It is much simpler to do:

```
import scisalt.matplotlib as sm
import matplotlib.pyplot as plt
import numpy as np

# Create data to plot
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

# Set up figure with axes
fig, ax = sm.setup_axes(rows=1, cols=2, figsize=(16, 6))

# Plot data
ax[0, 0].plot(x, y1)
ax[0, 1].plot(x, y2)

fig.tight_layout()

plt.show()
```



2.1.2 Recommended Setup

In order to make it easier, several convenience functions have been added. Each type of matplotlib plot has its own quirks, so there are convenience functions for each. It is highly recommended to create a matplotlibrc file with a few defaults:

```
backend : Qt4Agg
text.latex.preamble : \usepackage{booktabs},\usepackage{color},\usepackage{siunitx}

figure.figsize : 8, 6
figure.autolayout : True
image.cmap : viridis
image.interpolation : nearest

image.origin : 'lower'
```

This file accomplishes the following:

- Make sure the backend is Qt4Agg
- Use latex in figures, with the *booktabs*, *color*, and *siunitx* packages enabled
- Set the default figure to be 8” wide and 6” high
- Figures automatically resize their contents to use the most of the space available
- If using SciSalt, use the viridis color map by default
- Make the image origin to the lower left

2.1.3 Creating Axes

Basic

Every plot relies on first creating a figure and its axes. In order to make this process less cumbersome, two helper functions can be used. The most common usage will be to create a `figure` with `axes` laid out in a grid:

```
>>> fig, ax = scisalt.matplotlib.setup_axes(rows=2, cols=3)
>>> ax.shape
(2, 3)
```

Advanced

It is most powerful to create a `figure` and a `gridspec` using `scisalt.matplotlib.setup_figure`:

```
>>> fig, gs = scisalt.matplotlib.setup_figure(rows=2, cols=3)
```

This creates a figure and a gridspec instance in one step, which you can then use to make custom divisions of the figure:

```
>>> ax1 = fig.add_subplot(gs[0:2, 0])
>>> ax2 = fig.add_subplot(gs[0, 1:3])
>>> ax3 = fig.add_subplot(gs[1:3, 1:3])
```

2.1.4 Plotting

Plotting can also be cumbersome. Typically, plots need to be added. Matplotlib also has quirks. For example, images don't work the way physicists might expect, with our convention of the origin at the bottom left corner, positive axes proceeding up and to the right. Then labels need to be added, and the figure tightened up. All told, setting up a figure can take a lot of time.

Images

Images generally start as arrays. For this example, create a 2-D array to use:

```
>>> x = np.linspace(0, 10, 100)
>>> y = np.linspace(0, 10, 100)
>>> mesh = np.meshgrid(x, y)
>>> img = np.sin(mesh[0] * np.cos(mesh[1])
```

Plotting an image with matplotlib puts the axes in the upper right corner, with positive y-values down instead of up. This is undesirable from a physics perspective, and fixing this problem every time is cumbersome. `scisalt.matplotlib.imshow` solves this problem (and scales the window and image to look good):

```
>>> fig, ax, im = scisalt.matplotlib.imshow(img)
```

Or alternatively:

```
>>> fig, ax = scisalt.matplotlib.setup_axes()
>>> im = scisalt.matplotlib.imshow(img, ax=ax)
```

Note: If you would like to create an array of images that share the same axes, look at ImageGrid.