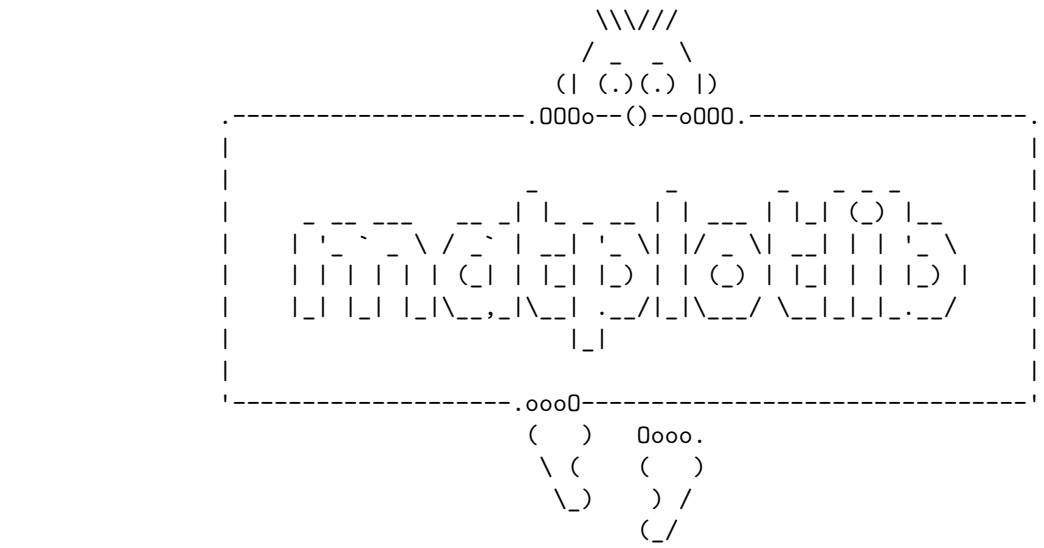


Codeblock class: shebang

```
http://www.google.com/search?q=shebang+line

runs:
> <fname>.shebang {im_opt} <fname>.{im_fmt}

class->cmd
  shebang -> shebang
```

Metadata options

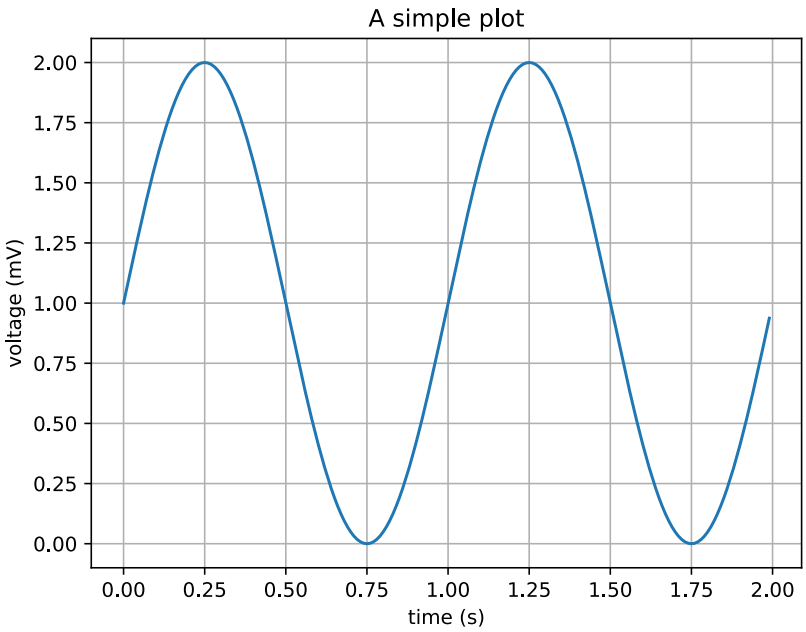
```
imagine.im_out: img,fcb
imagine.shebang.im_fmt: svg
imagine.shebang.im_log: 4
```

Notes

- sudo -H pip3 install matplotlib numpy

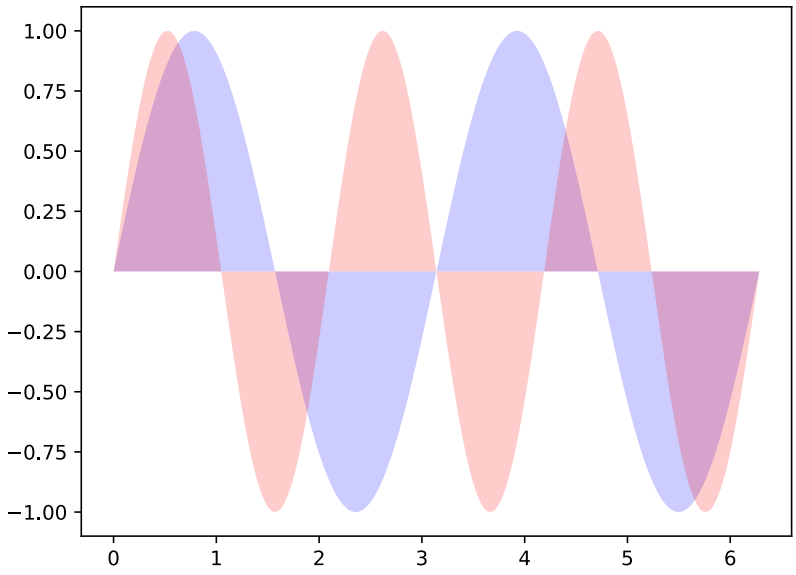
Matplotlib

Agg



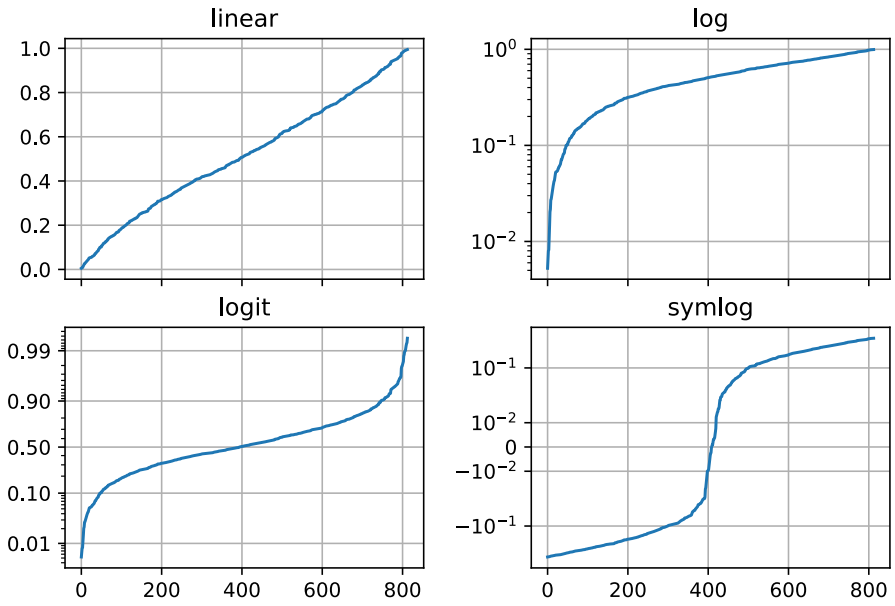

```shebang
#!/usr/bin/env python

import sys
import numpy as np
import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as plt

t = np.arange(0.0, 2.0, 0.01)
s = 1 + np.sin(2*np.pi*t)
plt.plot(t, s)

plt.xlabel('time (s)')
plt.ylabel('voltage (mV)')
plt.title('A simple plot')
plt.grid(True)
plt.savefig(sys.argv[-1])
```

Fill with alpha




```shebang
#!/usr/bin/env python

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

x = np.linspace(0, 2 * np.pi, 500)
y1 = np.sin(2 * x)
y2 = np.sin(3 * x)

fig, ax = plt.subplots()
ax.fill(x, y1, 'b', x, y2, 'r', alpha=0.2)
fig.savefig(sys.argv[-1])
```

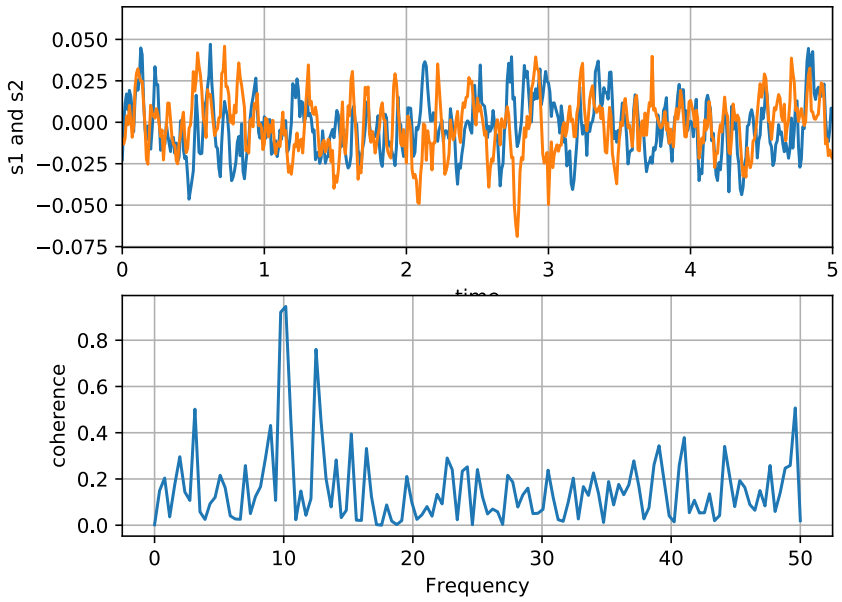
Axis scale transformations




```shebang
#!/usr/bin/env python

import sys
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter

np.random.seed(1)
# make up some data in the interval ]0, 1[
y = np.random.normal(loc=0.5, scale=0.4, size=1000)
y = y[(y > 0) & (y < 1)]
y.sort()
x = np.arange(len(y))

# plot with various axes scales
fig, axs = plt.subplots(2, 2, sharex=True)
fig.subplots_adjust(left=0.08, right=0.98, wspace=0.3)

# linear
ax = axs[0, 0]
```

```
ax.plot(x, y)
ax.set_yscale('linear')
ax.set_title('linear')
ax.grid(True)


# log
ax = axs[0, 1]
ax.plot(x, y)
ax.set_yscale('log')
ax.set_title('log')
ax.grid(True)


# symmetric log
ax = axs[1, 1]
ax.plot(x, y - y.mean())
ax.set_yscale('symlog', linthreshy=0.02)
ax.set_title('symlog')
ax.grid(True)

# logit
ax = axs[1, 0]
ax.plot(x, y)
ax.set_yscale('logit')
ax.set_title('logit')
ax.grid(True)
ax.yaxis.set_minor_formatter(NullFormatter())

fig.savefig(sys.argv[-1])
```

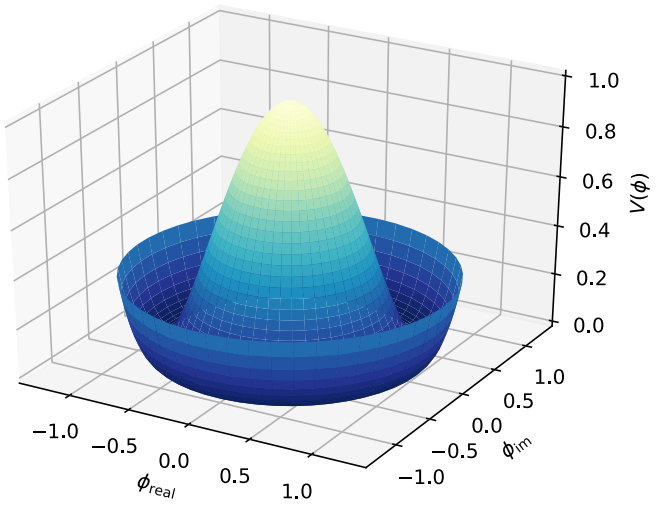
Coherence of two signals




```shebang
#!/usr/bin/env python

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

plt.subplots_adjust(wspace=0.5)                 # space the subplots

dt = 0.01
t = np.arange(0, 30, dt)
nse1 = np.random.randn(len(t))                  # white noise 1
nse2 = np.random.randn(len(t))                  # white noise 2
r = np.exp(-t/0.05)

cnse1 = np.convolve(nse1, r, mode='same')*dt    # colored noise 1
cnse2 = np.convolve(nse2, r, mode='same')*dt    # colored noise 2

# two signals with a coherent part and a random part
s1 = 0.01*np.sin(2*np.pi*10*t) + cnse1
s2 = 0.01*np.sin(2*np.pi*10*t) + cnse2
```

```
plt.subplot(211)
plt.plot(t, s1, t, s2)
plt.xlim(0, 5)
plt.xlabel('time')
plt.ylabel('s1 and s2')
plt.grid(True)

plt.subplot(212)
cxy, f = plt.cohere(s1, s2, 256, 1./dt)
plt.ylabel('coherence')
plt.savefig(sys.argv[-1])
```

3D image




```shebang
#!/usr/bin/env python
import sys
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import pyplot as plt
import numpy as np


fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Create the mesh in polar coordinates and compute corresponding Z.
r = np.linspace(0, 1.25, 50)
p = np.linspace(0, 2*np.pi, 50)
R, P = np.meshgrid(r, p)
Z = ((R**2 - 1)**2)

# Express the mesh in the cartesian system.
X, Y = R*np.cos(P), R*np.sin(P)

# Plot the surface.
```

```
ax.plot_surface(X, Y, Z, cmap=plt.cm.YlGnBu_r)

# Tweak the limits and add latex math labels.
ax.set_zlim(0, 1)
ax.set_xlabel(r'$\phi_\mathrm{real}$')
ax.set_ylabel(r'$\phi_\mathrm{im}$')
ax.set_zlabel(r'$V(\phi)$')

plt.savefig(sys.argv[-1])
```

Documentation

See matplotlib's website

docstring

```
This is an object-oriented plotting library.

A procedural interface is provided by the companion pyplot module,
which may be imported directly, e.g.::

    import matplotlib.pyplot as plt

or using ipython::

    ipython

at your terminal, followed by::

    In [1]: %matplotlib
    In [2]: import matplotlib.pyplot as plt

at the ipython shell prompt.

For the most part, direct use of the object-oriented library is
encouraged when programming; pyplot is primarily for working
interactively.  The
exceptions are the pyplot commands :func:`~matplotlib.pyplot.figure`,
:func:`~matplotlib.pyplot.subplot`,
:func:`~matplotlib.pyplot.subplots`, and
:func:`~pyplot.savefig`, which can greatly simplify scripting.
```

```
Modules include:

    :mod:`matplotlib.axes`
        defines the :class:`~matplotlib.axes.Axes` class.  Most pyplot
        commands are wrappers for :class:`~matplotlib.axes.Axes`
        methods.  The axes module is the highest level of OO access to
        the library.

    :mod:`matplotlib.figure`
        defines the :class:`~matplotlib.figure.Figure` class.

    :mod:`matplotlib.artist`
        defines the :class:`~matplotlib.artist.Artist` base class for
        all classes that draw things.

    :mod:`matplotlib.lines`
        defines the :class:`~matplotlib.lines.Line2D` class for
        drawing lines and markers

    :mod:`matplotlib.patches`
        defines classes for drawing polygons

    :mod:`matplotlib.text`
        defines the :class:`~matplotlib.text.Text`,
        :class:`~matplotlib.text.TextWithDash`, and
        :class:`~matplotlib.text.Annotate` classes

    :mod:`matplotlib.image`
        defines the :class:`~matplotlib.image.AxesImage` and
        :class:`~matplotlib.image.FigureImage` classes

    :mod:`matplotlib.collections`
        classes for efficient drawing of groups of lines or polygons

    :mod:`matplotlib.colors`
        classes for interpreting color specifications and for making
        colormaps

    :mod:`matplotlib.cm`
        colormaps and the :class:`~matplotlib.image.ScalarMappable`
        mixin class for providing color mapping functionality to other
        classes

    :mod:`matplotlib.ticker`
        classes for calculating tick mark locations and for formatting
```

```
        tick labels

    :mod:`matplotlib.backends`
        a subpackage with modules for various gui libraries and output
        formats

The base matplotlib namespace includes:

    :data:`~matplotlib.rcParams`
        a global dictionary of default configuration settings.  It is
        initialized by code which may be overridden by a matplotlibrc
        file.

    :func:`~matplotlib.rc`
        a function for setting groups of rcParams values

    :func:`~matplotlib.use`
        a function for setting the matplotlib backend.  If used, this
        function must be called immediately after importing matplotlib
        for the first time.  In particular, it must be called
        **before** importing pyplot (if pyplot is imported).

matplotlib was initially written by John D. Hunter (1968-2012) and is now
developed and maintained by a host of others.

Occasionally the internal documentation (python docstrings) will refer
to MATLAB&reg;, a registered trademark of The MathWorks, Inc.
```