V-Matplotlib

December 7, 2014

1 V-Matplotlib

1.0.1 Index

- Simple Plot
- Colormap
- 3D
- Histogram
- Animation

```
In [1]: import matplotlib as mpl
        %pylab inline
```

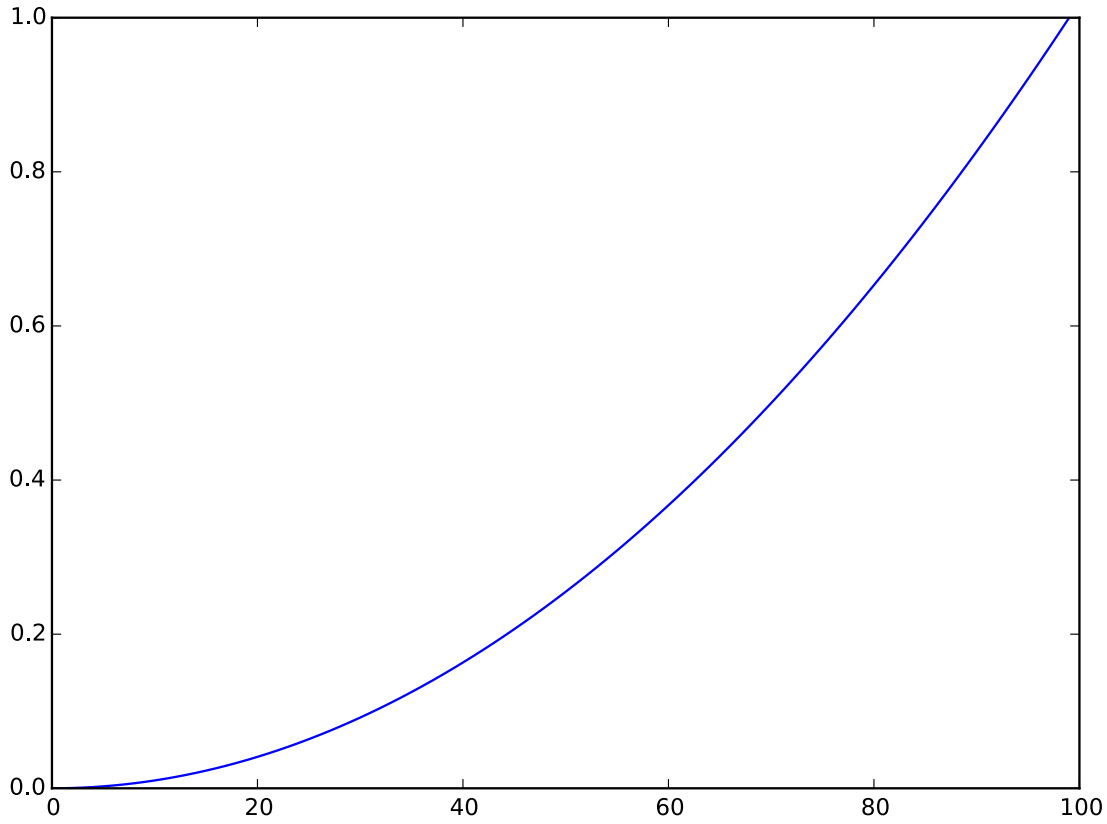
```
Populating the interactive namespace from numpy and matplotlib
```

Simple Plot

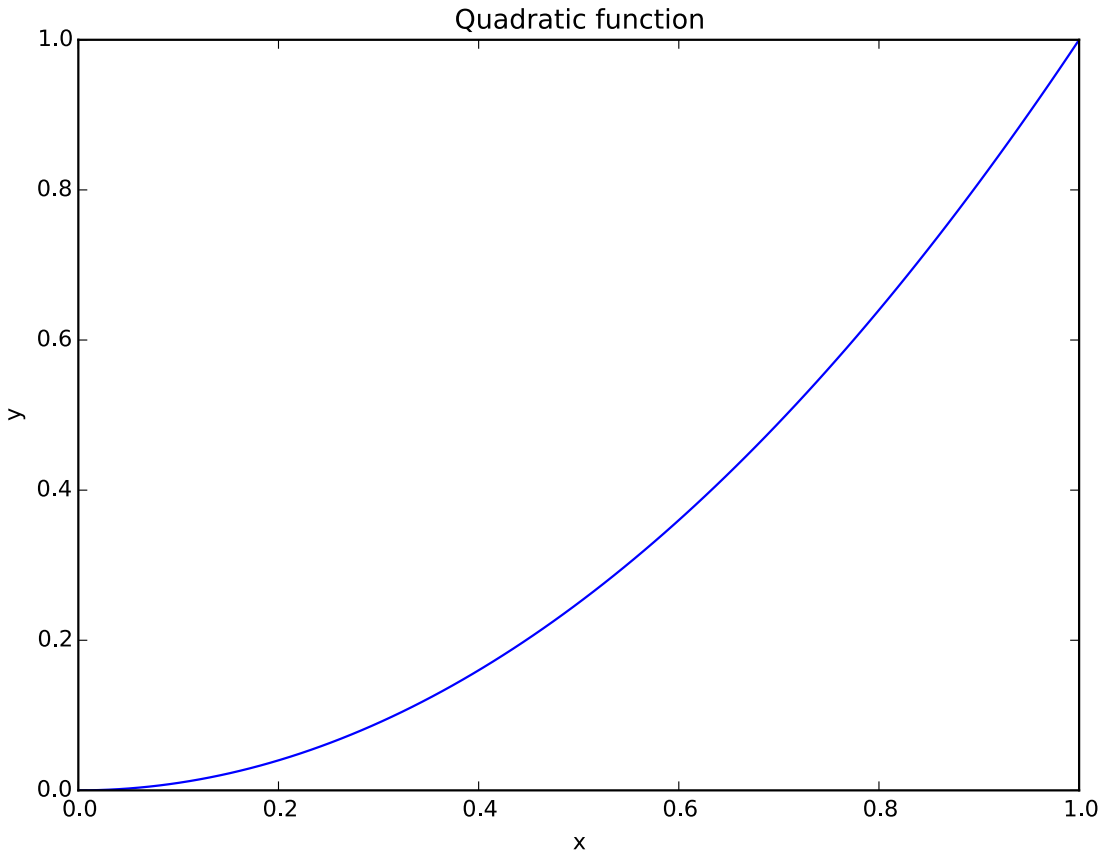
```
In [2]: x = linspace(0,1,100)
        y = x**2
```

```
In [3]: plot(y)
```

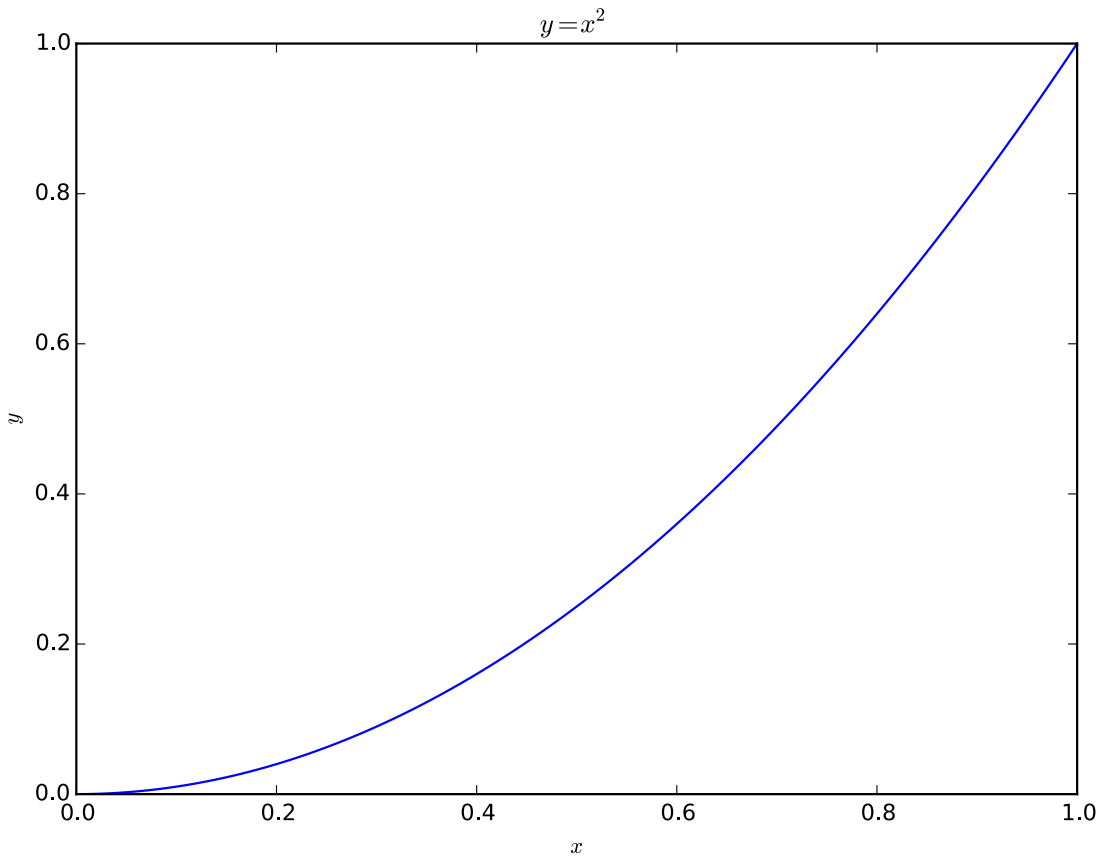
```
Out[3]: [<matplotlib.lines.Line2D at 0x7fce14bd80d0>]
```



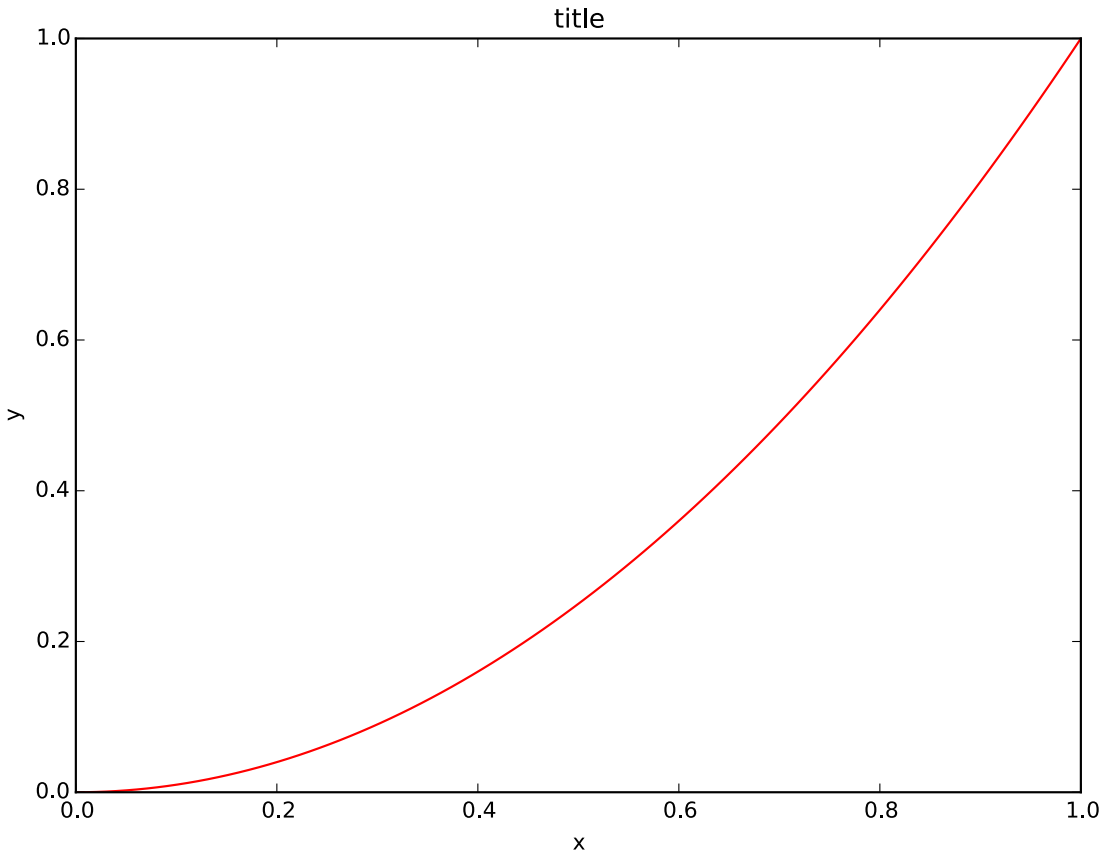

```
In [4]: figure()
        plot(x,y)
        xlabel('x')
        ylabel('y')
        title('Quadratic function')
        show()
```



In [5]:
```
figure()
plot(x,y)
xlabel('$x$')
ylabel('$y$')
title('$y=x^2$')
show()
```

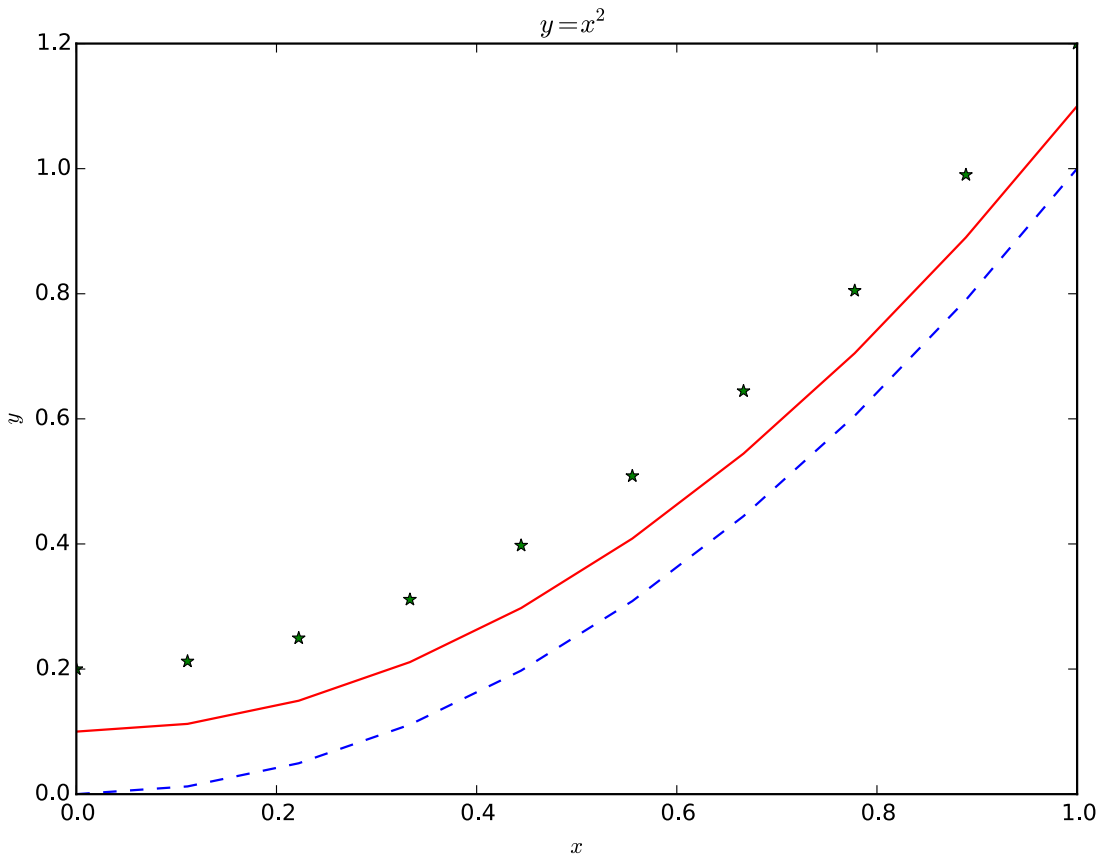


In [6]:
```
fig = plt.figure()

axes = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # left, bottom, width, height (range 0 to 1)
axes.plot(x, y, 'r')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('title');
```

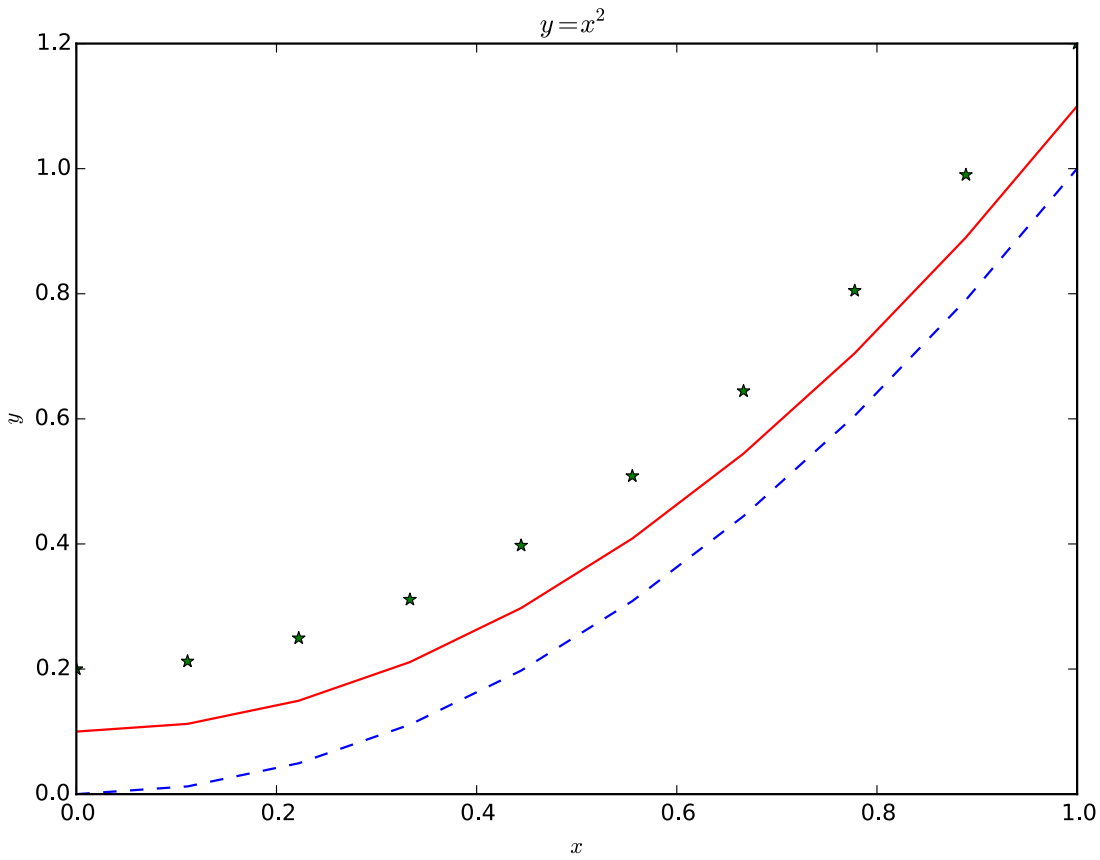


We can change the appearance, similar to MATLAB

```
In [7]: x = linspace(0,1,10)
        y1 = x**2
        y2 = x**2 + 0.1
        y3 = x**2 + 0.2

In [8]: figure()
        plot(x, y1, color='blue', linestyle='dashed')
        plot(x, y2, color='red')
        plot(x, y3, color='green', linestyle='.',marker='*')
        xlabel('$x$')
        ylabel('$y$')
        title('$y=x^2$')
        show()
```

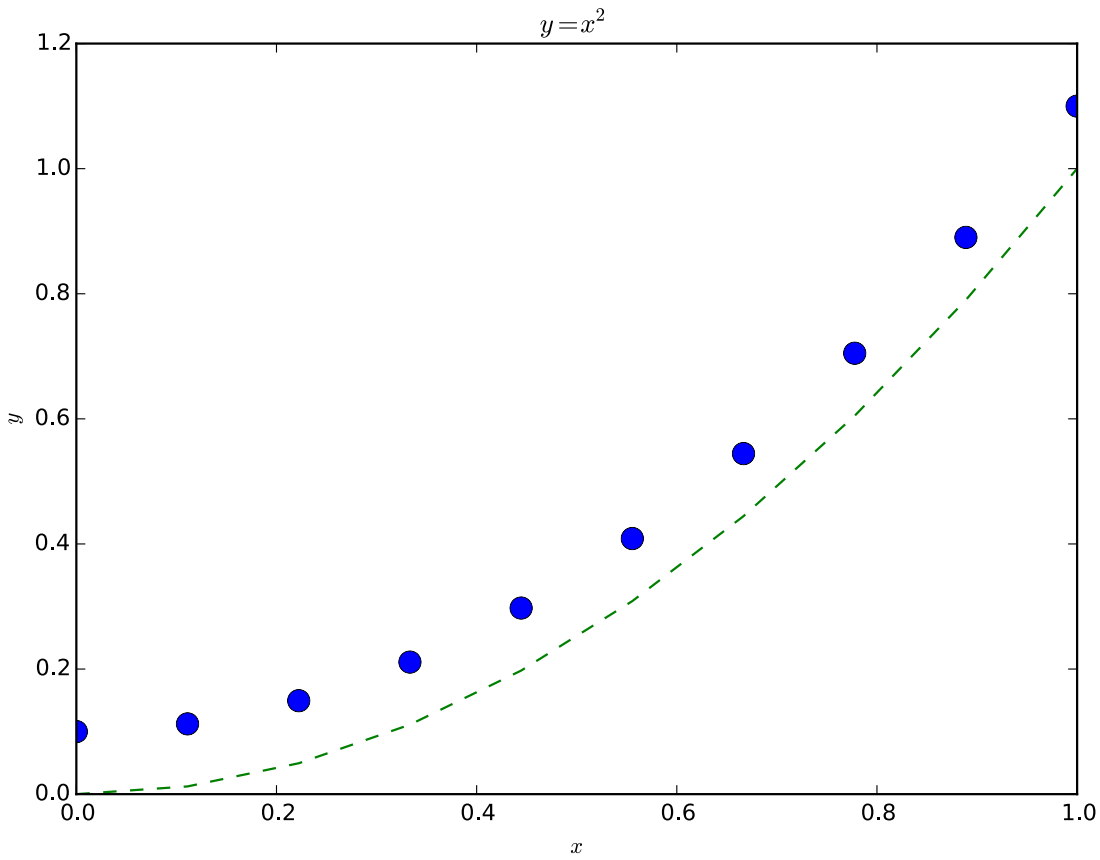


In [9]:
```
figure()
plot(x,y1,'--',x,y2,'r',x,y3,'*')
xlabel('$x$')
ylabel('$y$')
title('$y=x^2$')
show()
```



or more definition based

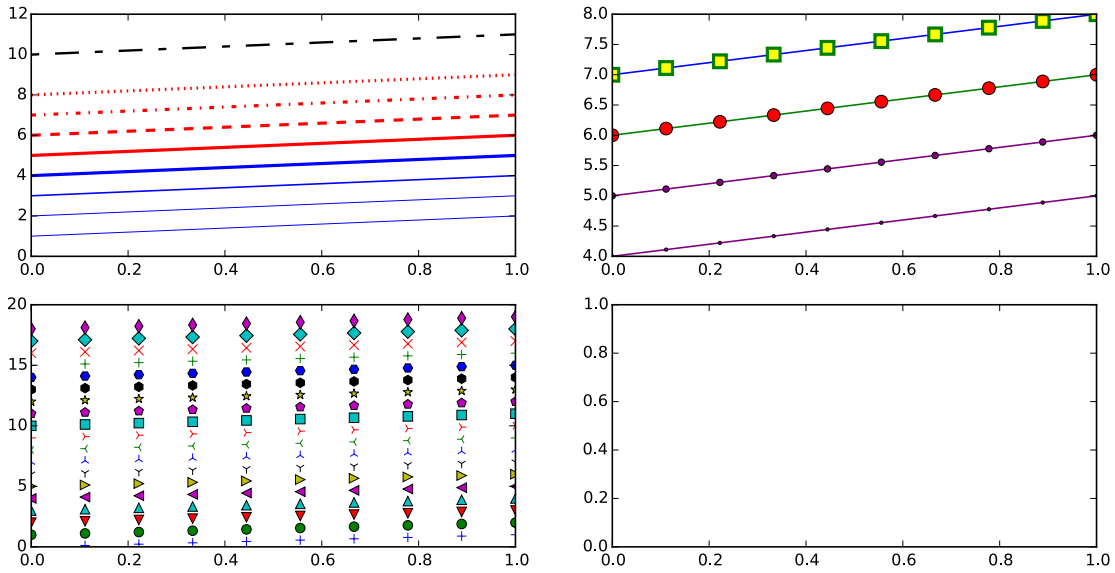
```
In [10]: figure()
         plot(x, y1, color='green', linestyle='dashed')
         plot(x, y2, color='blue', linestyle='.', marker='o', markerfacecolor='blue', markersize=10, al
         xlabel('$x$')
         ylabel('$y$')
         title('$y=x^2$')
         show()
```




```
In [11]: fig, ax = plt.subplots(2,2,figsize=(12,6))

         ax[0,0].plot(x, x+1, color="blue", linewidth=0.25)
         ax[0,0].plot(x, x+2, color="blue", linewidth=0.50)
         ax[0,0].plot(x, x+3, color="blue", linewidth=1.00)    # default
         ax[0,0].plot(x, x+4, color="blue", linewidth=2.00)

         # possible linestype options '-', '{', '-.', ':', '.', 'steps'
         ax[0,0].plot(x, x+5, color="red", lw=2, linestyle='-')
         ax[0,0].plot(x, x+6, color="red", lw=2, ls='--')
         ax[0,0].plot(x, x+7, color="red", lw=2, ls='-.')
         ax[0,0].plot(x, x+8, color="red", lw=2, ls=':')
         ax[0,0].plot(x, x+9, color="red", lw=2, ls='.')

         # custom
         line, = ax[0,0].plot(x, x+10, color="black", lw=1.50)
         line.set_dashes([5, 10, 15, 10]) # format: line length, space length, ...

         # Marker Symbols
         ax[1,0].plot(x, x, lw=2, ls='*', marker='+')
         ax[1,0].plot(x, x+1, lw=2, ls='*', marker='o')
         ax[1,0].plot(x, x+2, lw=2, ls='*', marker='v')
         ax[1,0].plot(x, x+3, lw=2, ls='*', marker='^')
```

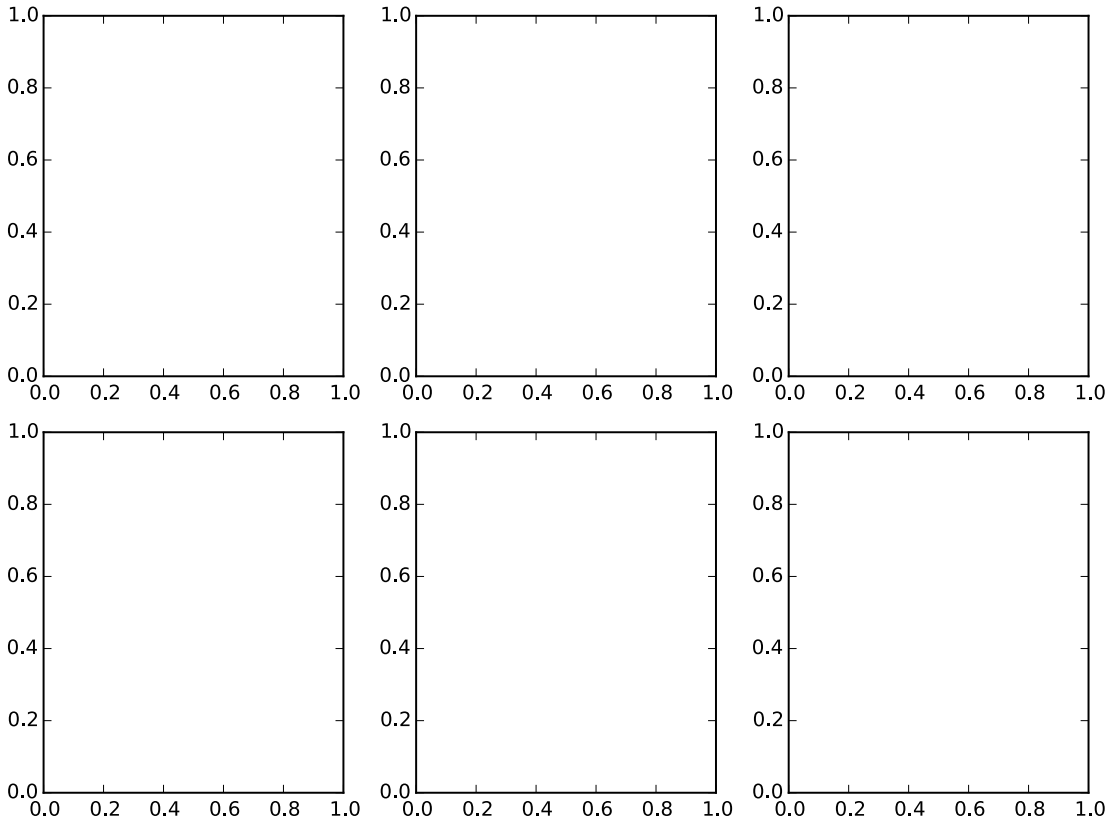
```
ax[1,0].plot(x, x+4, lw=2, ls='*', marker='<')
ax[1,0].plot(x, x+5, lw=2, ls='*', marker='>')
ax[1,0].plot(x, x+6, lw=2, ls='*', marker='1')
ax[1,0].plot(x, x+7, lw=2, ls='*', marker='2')
ax[1,0].plot(x, x+8, lw=2, ls='*', marker='3')
ax[1,0].plot(x, x+9, lw=2, ls='*', marker='4')
ax[1,0].plot(x, x+10, lw=2, ls='*', marker='s')
ax[1,0].plot(x, x+11, lw=2, ls='*', marker='p')
ax[1,0].plot(x, x+12, lw=2, ls='*', marker='*')
ax[1,0].plot(x, x+13, lw=2, ls='*', marker='h')
ax[1,0].plot(x, x+14, lw=2, ls='*', marker='H')
ax[1,0].plot(x, x+15, lw=2, ls='*', marker='+')
ax[1,0].plot(x, x+16, lw=2, ls='*', marker='x')
ax[1,0].plot(x, x+17, lw=2, ls='*', marker='D')
ax[1,0].plot(x, x+18, lw=2, ls='*', marker='d')
```

```
# marker size and color
ax[0,1].plot(x, x+4, color="purple", lw=1, ls='-', marker='o', markersize=2)
ax[0,1].plot(x, x+5, color="purple", lw=1, ls='-', marker='o', markersize=4)
ax[0,1].plot(x, x+6, color="green", lw=1, ls='-', marker='o', markersize=8, markerfacecolor="r
ax[0,1].plot(x, x+7, color="blue", lw=1, ls='-', marker='s', markersize=8,
        markerfacecolor="yellow", markeredgewidth=2, markeredgecolor="green");
```



Subplots

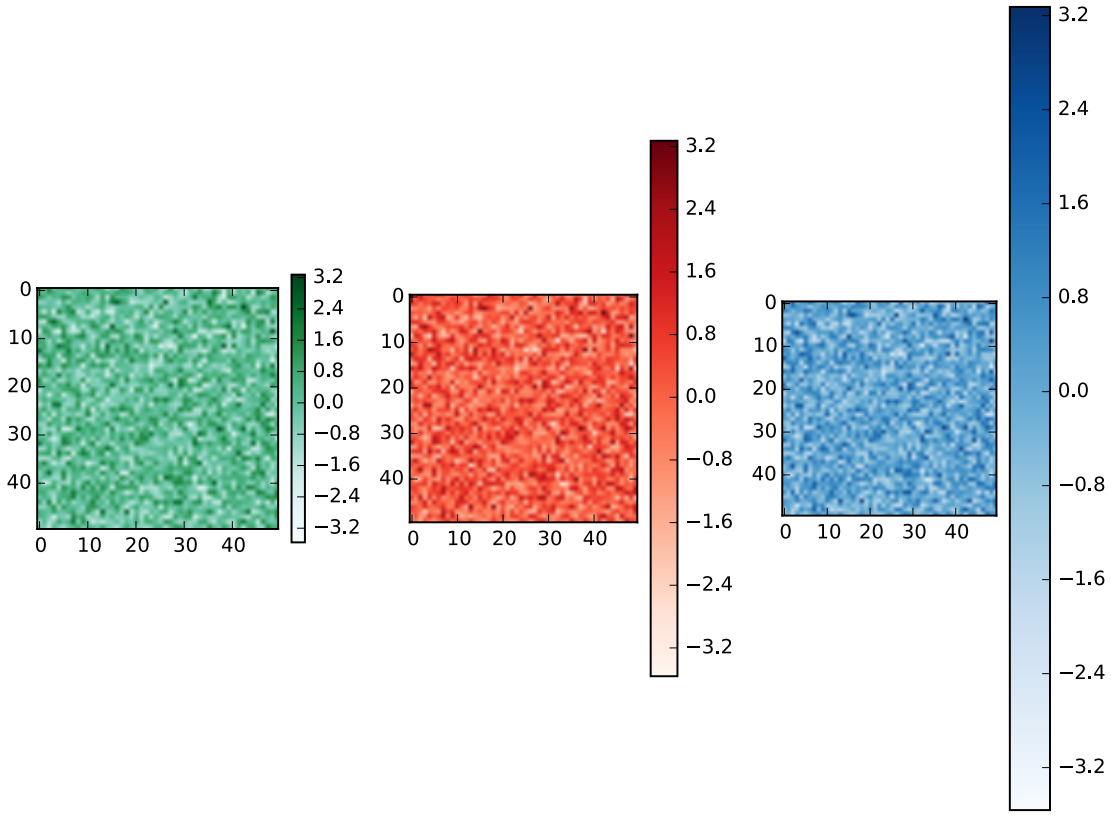
```
In [12]: fig, ax = plt.subplots(2, 3)
         fig.tight_layout()
```



1.0.2 Colormap

cmap with _r is a reversed one

```
In [13]: data = np.random.randn(50,50)
         fig, ax = subplots(1,3)

         plt1 = ax[0].imshow(data,cmap='BuGn')
         fig.colorbar(plt1,ax=ax[0],fraction=0.05)
         plt2 = ax[1].imshow(data,cmap='Reds')
         fig.colorbar(plt2,ax=ax[1],fraction=0.1)
         plt3 = ax[2].imshow(data,cmap='Blues')
         fig.colorbar(plt3,ax=ax[2])
         fig.tight_layout()
```



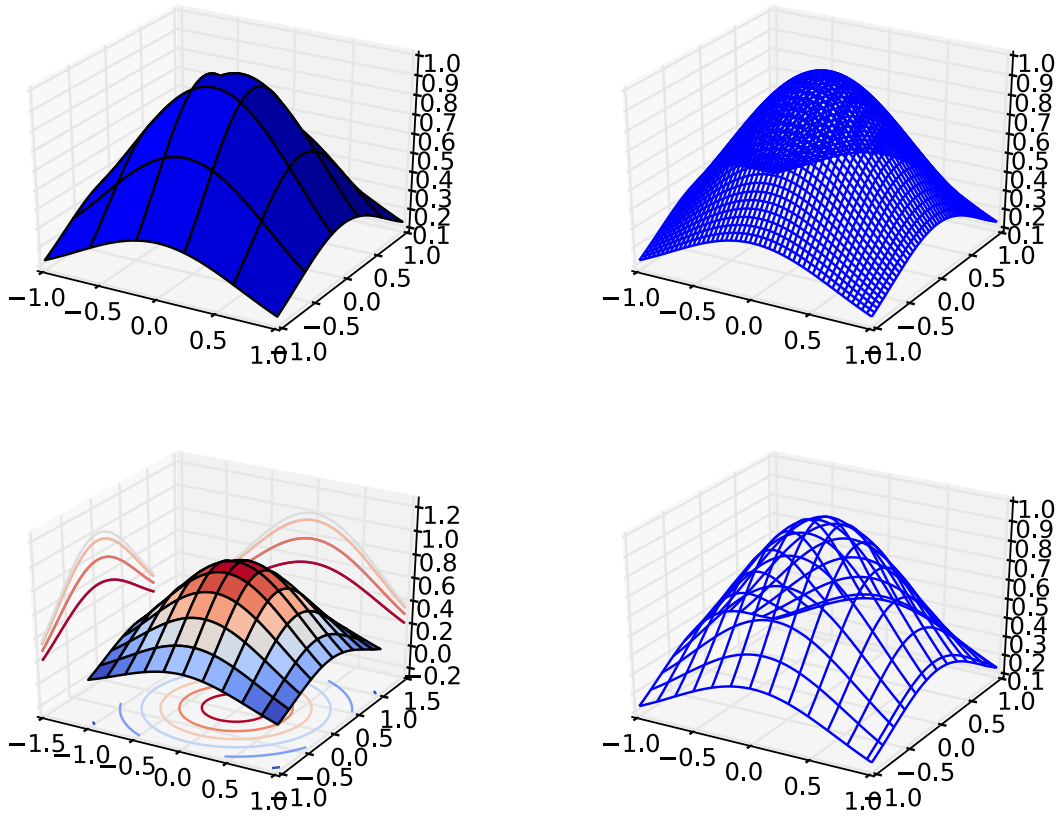
1.0.3 3D

```
In [14]: from mpl_toolkits.mplot3d import Axes3D
```

```
In [15]: x = linspace(-1,1,50)
         y = linspace(-1,1,50)
         X,Y = meshgrid(x,y)
         Z = exp(-X**2-Y**2)
```

```
In [16]: fig, ax = subplots(2,2,subplot_kw=dict(projection='3d'))
         ax[0,0].plot_surface(X,Y,Z)
         ax[0,1].plot_wireframe(X,Y,Z)
         ax[1,0].plot_surface(X,Y,Z,cmap=cm.coolwarm,rstride=5,cstride=5)
         cset = ax[1,0].contour(X, Y, Z, zdir='z', offset=-0.25, cmap=cm.coolwarm)
         cset = ax[1,0].contour(X, Y, Z, zdir='x', offset=-1.5, cmap=cm.coolwarm)
         cset = ax[1,0].contour(X, Y, Z, zdir='y', offset=1.5, cmap=cm.coolwarm)

         ax[1,0].set_xlim([-1.5, 1])
         ax[1,0].set_ylim([-1, 1.5])
         ax[1,0].set_zlim([-0.25, 1.25])

         ax[1,1].plot_wireframe(X,Y,Z,rstride=4,cstride=4)
```

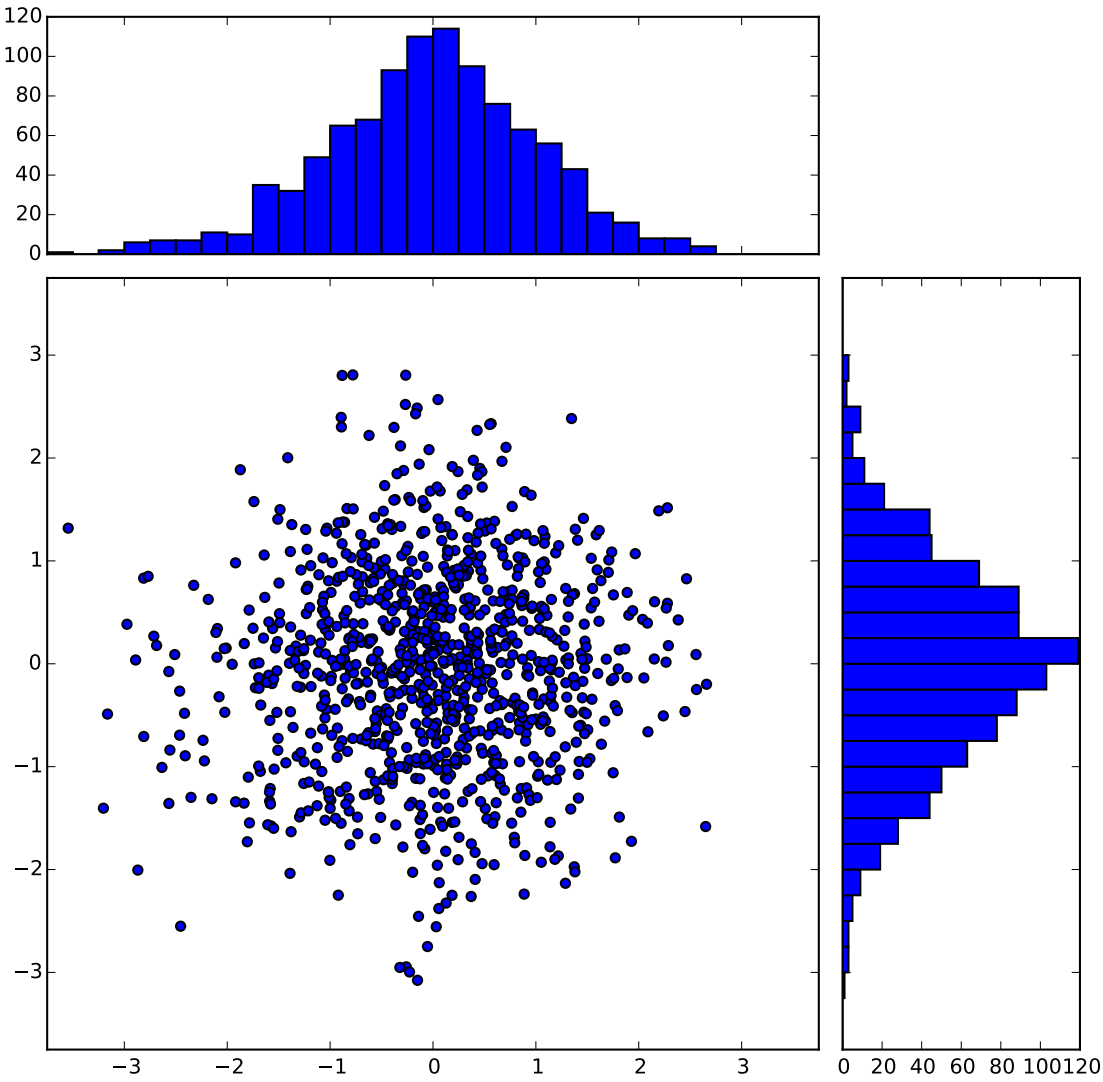
```
Out[16]: <mpl_toolkits.mplot3d.art3d.Line3DCollection at 0x7fce12c1ee10>
```



1.0.4 Histogram

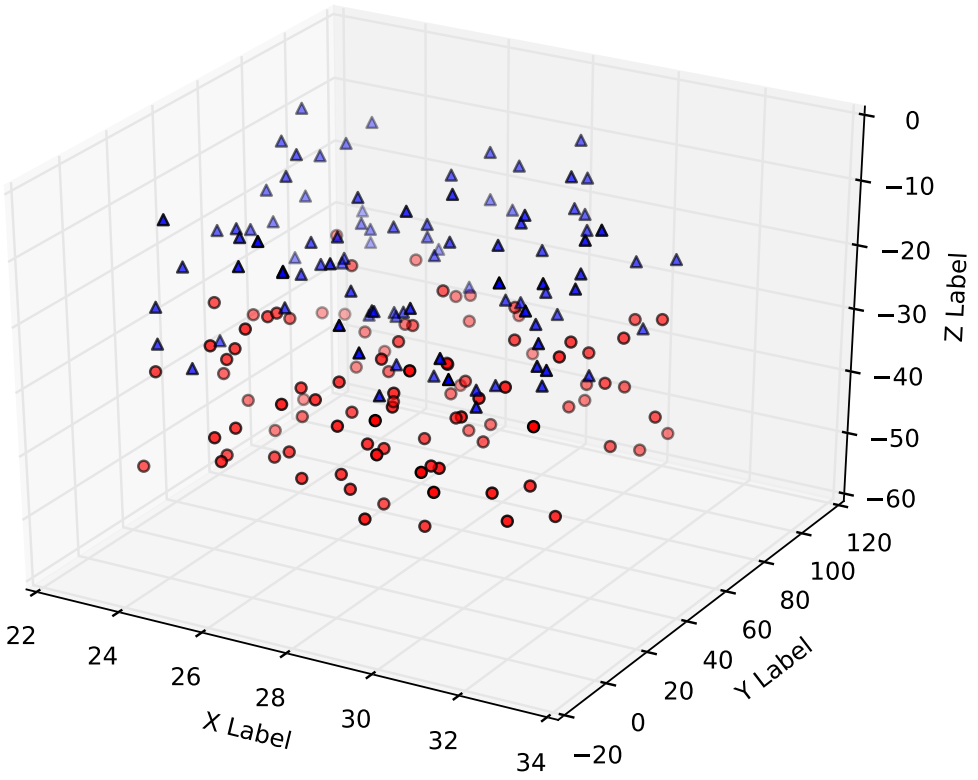
```
In [17]: import numpy as np
         import matplotlib.pyplot as plt
         from matplotlib.ticker import NullFormatter

         # the random data
         x = np.random.randn(1000)
         y = np.random.randn(1000)

         nullfmt   = NullFormatter()         # no labels

         # definitions for the axes
         left, width = 0.1, 0.65
         bottom, height = 0.1, 0.65
         bottom_h = left_h = left+width+0.02

         rect_scatter = [left, bottom, width, height]
         rect_histx = [left, bottom_h, width, 0.2]
         rect_histy = [left_h, bottom, 0.2, height]

         # start with a rectangular Figure
         plt.figure(1, figsize=(8,8))
```

```
axScatter = plt.axes(rect_scatter)
axHistx = plt.axes(rect_histx)
axHisty = plt.axes(rect_histy)

# no labels
axHistx.xaxis.set_major_formatter(nullfmt)
axHisty.yaxis.set_major_formatter(nullfmt)

# the scatter plot:
axScatter.scatter(x, y)

# now determine nice limits by hand:
binwidth = 0.25
xymax = np.max( [np.max(np.fabs(x)), np.max(np.fabs(y))] )
lim = ( int(xymax/binwidth) + 1) * binwidth

axScatter.set_xlim( (-lim, lim) )
axScatter.set_ylim( (-lim, lim) )

bins = np.arange(-lim, lim + binwidth, binwidth)
axHistx.hist(x, bins=bins)
axHisty.hist(y, bins=bins, orientation='horizontal')

axHistx.set_xlim( axScatter.get_xlim() )
axHisty.set_ylim( axScatter.get_ylim() )

plt.show()
```



we can run any example from the online gallery examples

```
In [18]: import os
         os.system('wget http://matplotlib.org/mpl_examples/mplot3d/scatter3d_demo.py')

         %run 'scatter3d_demo.py'
```



1.0.5 Animation

In [21]:
```
from scipy.integrate import odeint
from matplotlib import animation

g = 9.82; L = 0.5; m = 0.1

def dx(x, t):
    x1, x2, x3, x4 = x[0], x[1], x[2], x[3]

    dx1 = 6.0/(m*L**2) * (2 * x3 - 3 * cos(x1-x2) * x4)/(16 - 9 * cos(x1-x2)**2)
    dx2 = 6.0/(m*L**2) * (8 * x4 - 3 * cos(x1-x2) * x3)/(16 - 9 * cos(x1-x2)**2)
    dx3 = -0.5 * m * L**2 * ( dx1 * dx2 * sin(x1-x2) + 3 * (g/L) * sin(x1))
    dx4 = -0.5 * m * L**2 * (-dx1 * dx2 * sin(x1-x2) + (g/L) * sin(x2))
    return [dx1, dx2, dx3, dx4]

x0 = [pi/2, pi/2, 0, 0]  # initial state
t = linspace(0, 10, 250) # time coordinates
x = odeint(dx, x0, t)    # solve the ODE problem

fig, ax = plt.subplots(figsize=(5,5))

ax.set_ylim([-1.5, 0.5])
ax.set_xlim([1, -1])
```

```
t = linspace(0, 10, 250) # time coordinates


pendulum1, = ax.plot([], [], color="red", lw=2)
pendulum2, = ax.plot([], [], color="blue", lw=2)

def init():
    pendulum1.set_data([], [])
    pendulum2.set_data([], [])

def update(n):
    # n = frame counter
    # calculate the positions of the pendulums
    x1 = + L * sin(x[n, 0])
    y1 = - L * cos(x[n, 0])
    x2 = x1 + L * sin(x[n, 1])
    y2 = y1 - L * cos(x[n, 1])

    # update the line data
    pendulum1.set_data([0 ,x1], [0 ,y1])
    pendulum2.set_data([x1,x2], [y1,y2])

anim = animation.FuncAnimation(fig, update, init_func=init, frames=len(t), blit=True)
anim.save('../data/animation.mp4', fps=20)
plt.close(fig)
```

In [22]:
```
from IPython.display import HTML
from tempfile import NamedTemporaryFile
import shutil

WEBM_VIDEO_TAG = """<video controls>
 <source src="data:video/x-webm;base64,{0}" type="video/webm">
 Your browser does not support the video tag.
</video>"""

M4V_VIDEO_TAG = """<video controls>
 <source src="data:video/x-m4v;base64,{0}" type="video/mp4">
 Your browser does not support the video tag.
</video>"""

FPS = 20          # Frames per second in the generated movie

def anim_to_html(anim, filename=None):
    if not hasattr(anim, '_encoded_video'):
        with NamedTemporaryFile(suffix='.webm') as f:
            webm_writer = animation.FFMpegWriter(fps=FPS, codec="libvpx")  # you'll need libvp
            vpx_args = ["-quality", "good",     # many arguments are not needed in this example
                        "-cpu-used", "0",
                        "-b:v", "500k",
                        "-qmin", "10",
                        "-qmax", "42",
                        "-maxrate", "500k",
                        "-bufsize", "1000k",
                        "-threads", "4",
```

```
                                    "-vf", "scale=-1:240",
                                    "-codec:a", "libvorbis",
                                    "-b:a", "128k"]
                        anim.save(f.name, writer=webm_writer, extra_args=vpx_args)
                        if filename is not None:  # in case you want to keep a copy of the generated movie
                            shutil.copyfile(f.name, filename)
                        video = open(f.name, "rb").read()
                    anim._encoded_video = video.encode("base64")

                return WEBM_VIDEO_TAG.format(anim._encoded_video)

            def display_animation(anim, filename):
                plt.close(anim._fig)
                return HTML(anim_to_html(anim, filename))
```

In [24]:
```
display_animation(anim,filename='../data/animation.mp4')
```

Out[24]:
```
<IPython.core.display.HTML at 0x7fce11bc5a90>
```

In [25]:
```
%load_ext version_information
%version_information matplotlib
```

Out[25]:

Software	Version
Python	2.7.8 —Anaconda 2.1.0 (64-bit)— (default, Aug 21 2014, 18:22:21) [GCC 4.4.7 20120313 (Red Hat 4.4.7-1
IPython	2.3.1
OS	posix [linux2]
matplotlib	1.4.2
Fri Dec 05 10:17:37 2014 CET	

In [2]:
```
from IPython.core.display import HTML
def css_styling():
    styles = open("./styles/custom.css", "r").read()
    return HTML(styles)
css_styling()
```

Out[2]:
```
<IPython.core.display.HTML at 0x7f35f0194a90>
```

Back to top