Lab-1

October 13, 2016

1 Notebook for Lab 1

Check basic arithmetic operations.

```
In [1]: 1+2

Out[1]: 3

In [2]: 2*3

Out[2]: 6

In [3]: 4/5

Out[3]: 0.8
```

What happened there? python does *integer* division by default (in python 2). Should make explicit that numbers are *real*:

```
In [4]: 4.0/5.0

Out[4]: 0.8
```

1.1 Variables

```
In [5]: two = 1 + 1

In [6]: two

Out[6]: 2
```

1.2 Vectors

Use `numpy` for all of these.

```
In [7]: import numpy

In [8]: x = numpy.array([1, 2])
        x.shape
```

```
Out[8]: (2,)

In [9]: y=x.T
        y.shape

Out[9]: (2,)
```

Unlike eg Matlab, `python` does not distinguish between row and column vectors.

1.3 Matrices

```
In [10]: A = numpy.array([[2, 3], [6, 5]])
         print(A)

[[2 3]
 [6 5]]


In [11]: print(len(A))
         print(A.size)
         print(A.shape)

2
4
(2, 2)


In [12]: numpy.dot(A, x)

Out[12]: array([ 8, 16])

In [13]: numpy.linalg.eig(A)

Out[13]: (array([-1.,  8.]), array([[-0.70710678, -0.4472136 ],
                 [ 0.70710678, -0.89442719]]))

In [14]: A = A.T
         print(A)

[[2 6]
 [3 5]]


In [15]: A**2

Out[15]: array([[ 4, 36],
                [ 9, 25]])

In [16]: numpy.dot(A, A)
```

```
Out[16]: array([[22, 42],
                [21, 43]])
```

```
In [17]: numpy.sin(x)
```

```
Out[17]: array([ 0.84147098,  0.90929743])
```

```
In [18]: l, v = numpy.linalg.eig(A)
         x = v[:,0]
```

```
In [19]: x
```

```
Out[19]: array([-0.89442719,  0.4472136 ])
```

```
In [20]: v
```

```
Out[20]: array([[-0.89442719, -0.70710678],
                [ 0.4472136 , -0.70710678]])
```

```
In [21]: import scipy.linalg
```

```
In [22]: scipy.linalg?
```

1.4 Plotting

This command is only needed in the notebook, although variants may be helpful in spyder

```
In [23]: %matplotlib inline
```

This is the plotting library `matplotlib`.

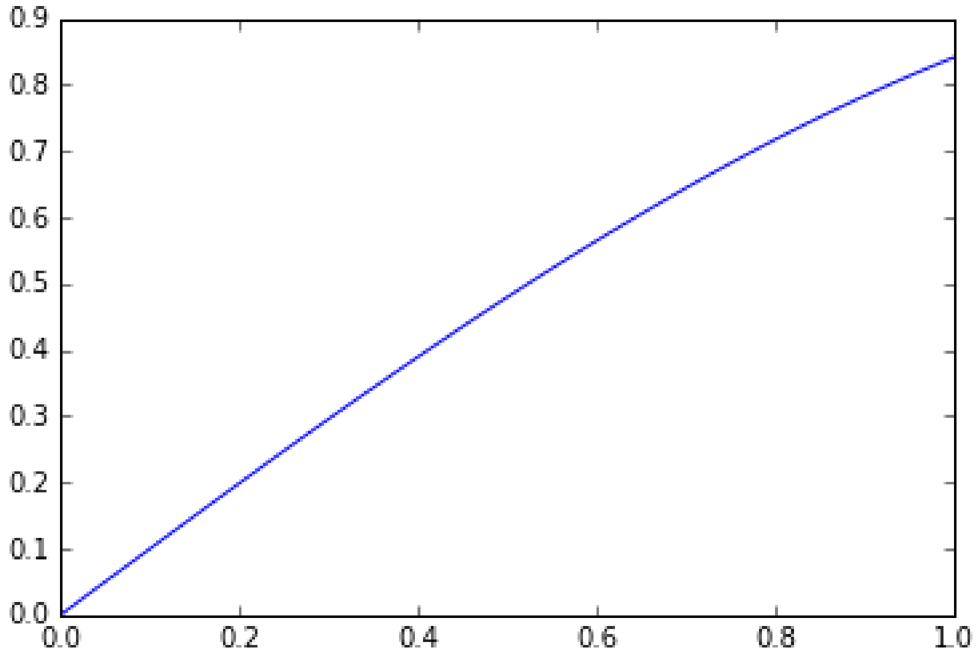
```
In [24]: from matplotlib import pyplot
```

```
In [25]: x = numpy.linspace(0.0, 1.0, 80)
         y = numpy.linspace(0.0, 2.0, 60)
```

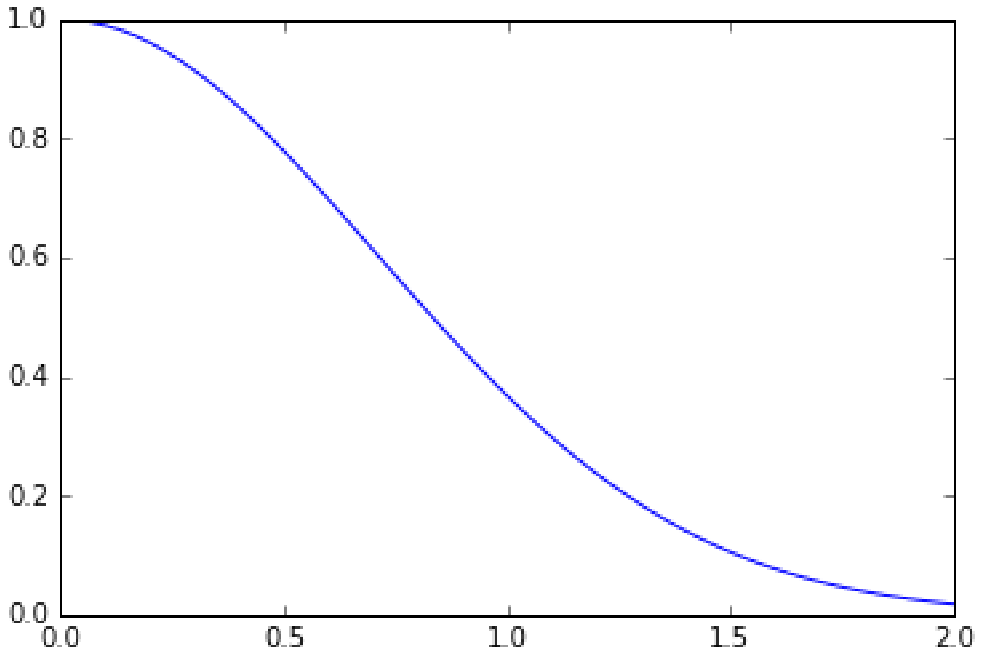
```
In [26]: s = numpy.sin(x)
         e = numpy.exp(-y**2)
```

```
In [27]: pyplot.plot(x, s)
```

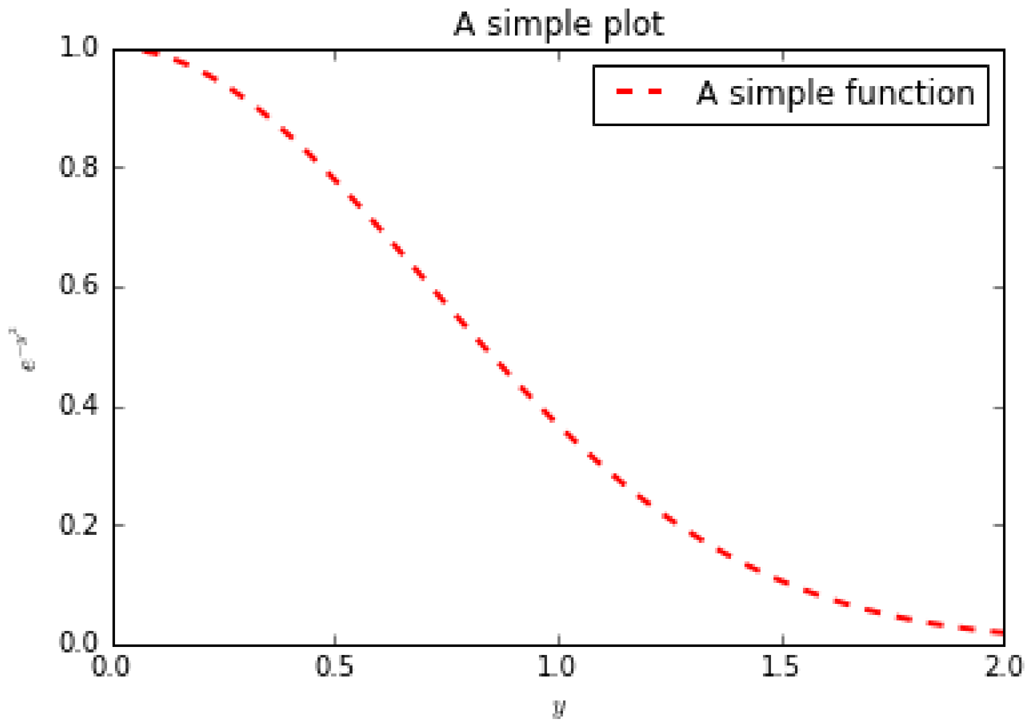
```
Out[27]: [<matplotlib.lines.Line2D at 0x10e44a828>]
```



```
In [28]: pyplot.plot(y,e)

Out[28]: [<matplotlib.lines.Line2D at 0x10e6680b8>]
```

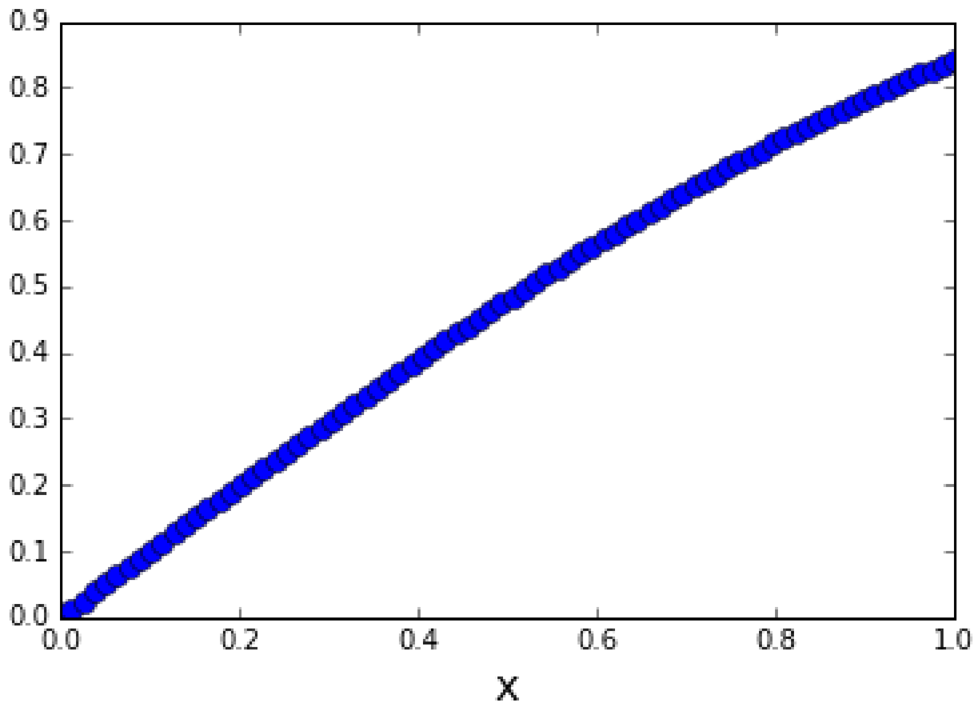


```
In [29]: pyplot.plot(y, e, 'r--', linewidth = 2, label = 'A simple function')
         pyplot.xlabel(r"$y$")
         pyplot.ylabel(r"$e^{-y^2}$")
         pyplot.title("A simple plot")
         pyplot.legend()

Out[29]: <matplotlib.legend.Legend at 0x10e6f7da0>
```



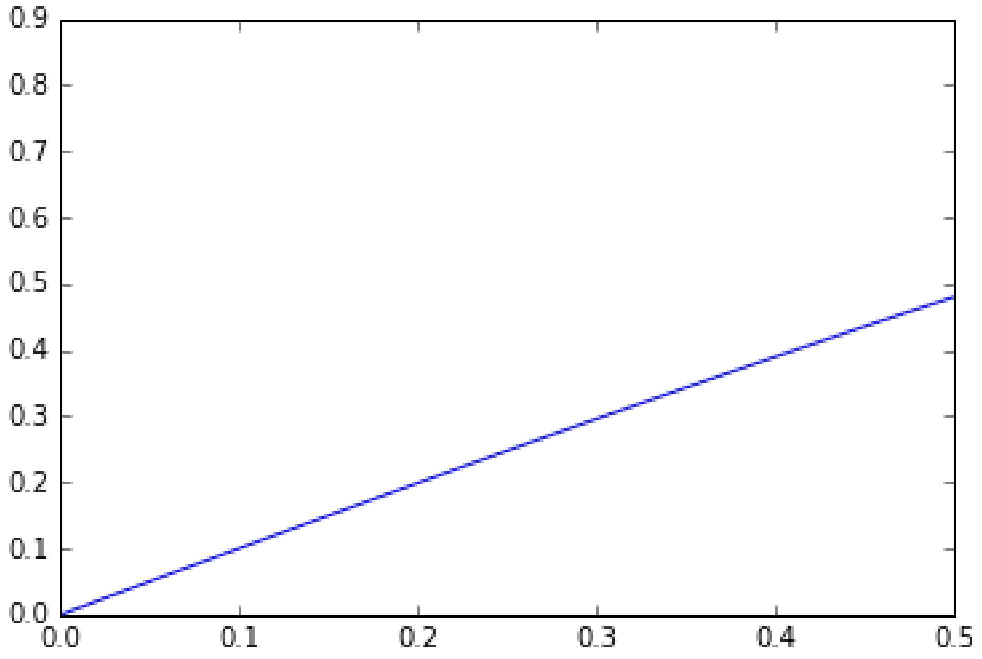
The plot above used LaTeX notation to typeset the mathematics; this need "raw strings" (hence the `r` in `r"..."`). LaTeX typeset mathematics between the `$` symbols. You could just use plain strings.

```
In [30]: pyplot.plot(x, s, marker = "o", markersize = 8)
         pyplot.xlabel("x", size=16)

Out[30]: <matplotlib.text.Text at 0x10edd5518>
```



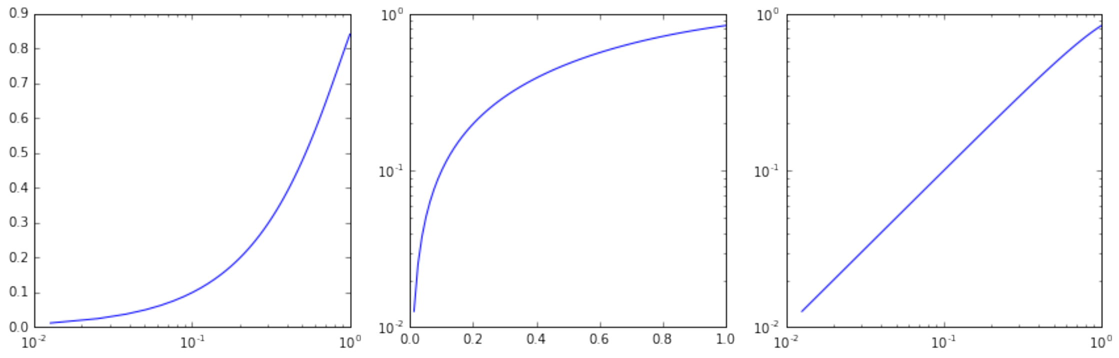
The above plot is not sensible, but does show how to change many of the aspects of the plot. If you want to do this for all plots, investigate the use of `rcParams`.

```
In [31]: pyplot.plot(x,s)
         pyplot.xlim(0.0, 0.5)

Out[31]: (0.0, 0.5)
```



Note that the y scale is not automatically reset here!

```
In [32]: fig = pyplot.figure(figsize = (12, 4))
         ax1 = fig.add_subplot(131)
         ax1.semilogx(x,s)
         ax2 = fig.add_subplot(132)
         ax2.semilogy(x,s)
         ax3 = fig.add_subplot(133)
         ax3.loglog(x,s)
         fig.tight_layout()
```

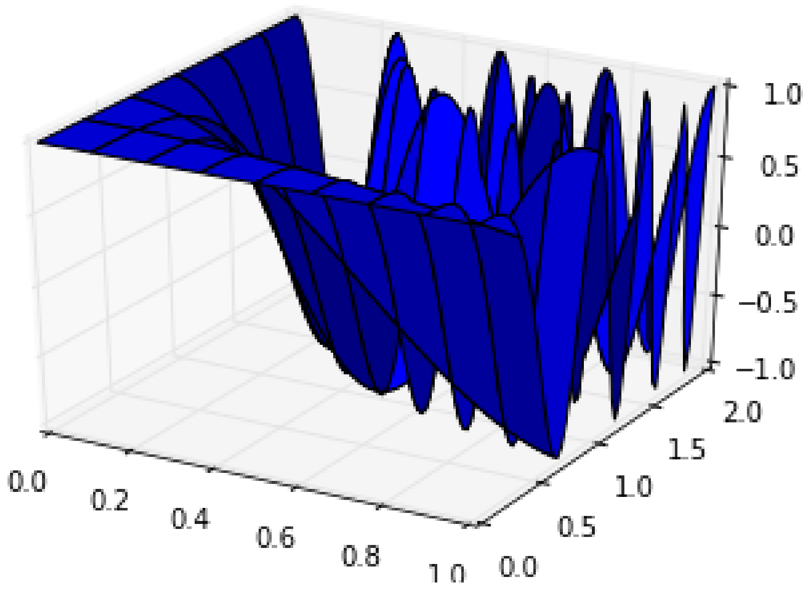


The use of subplots here is not needed to use the various different log plots; you can experiment with single plots of this type as well.

```
In [33]: X, Y = numpy.meshgrid(x, y)
         Z = numpy.cos(2.0 * numpy.pi * X * Y**2)
```

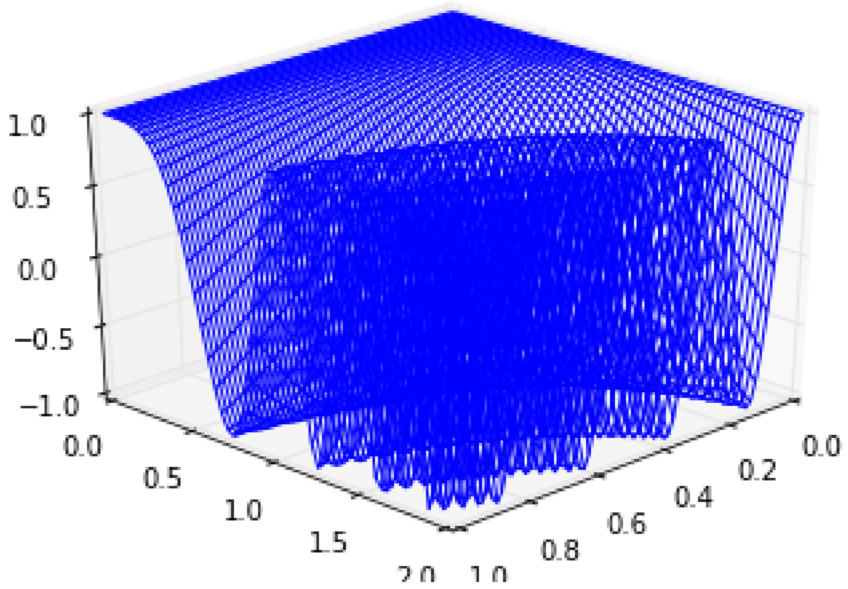
We could not build z directly from the vectors `x, y` as they have different sizes. The variables `X, Y` are instead matrices with the same size which can be used.

```
In [34]: from mpl_toolkits.mplot3d.axes3d import Axes3D

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

         ax.plot_surface(X, Y, Z)

Out[34]: <mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x10ef5bd30>
```



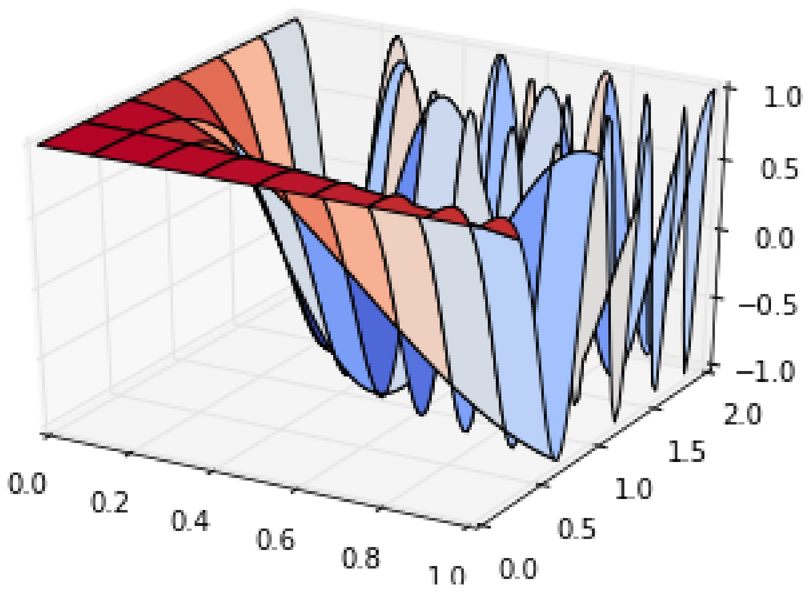
We need to import a new part of `matplotlib` (the first line). We also *need* to create the axis with a particular projection - there isn't a quick shortcut in this case.

```
In [35]: fig = pyplot.figure()
         ax = fig.add_subplot(111, projection='3d')

         ax.plot_wireframe(X, Y, Z)
         ax.view_init(30, 45)
```



Rotating the viewing angle to start.

```
In [36]: from matplotlib import cm

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

         ax.plot_surface(X, Y, Z, cmap = cm.coolwarm)

Out[36]: <mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x10ed23f98>
```

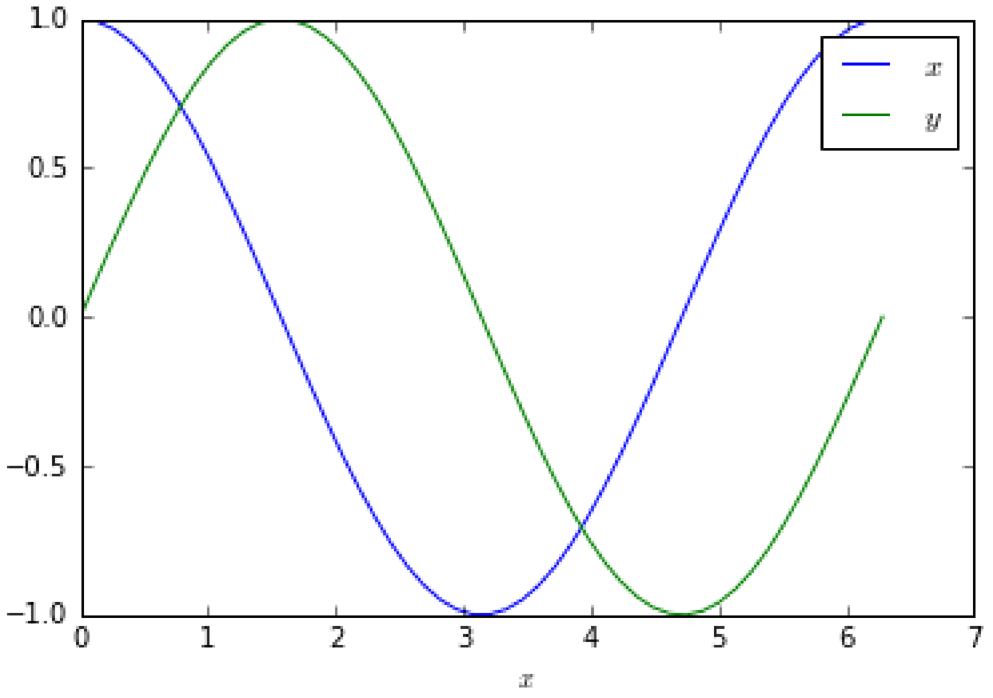


To change colormap we had to import another part of `matplotlib`, then pass the suitable argument.

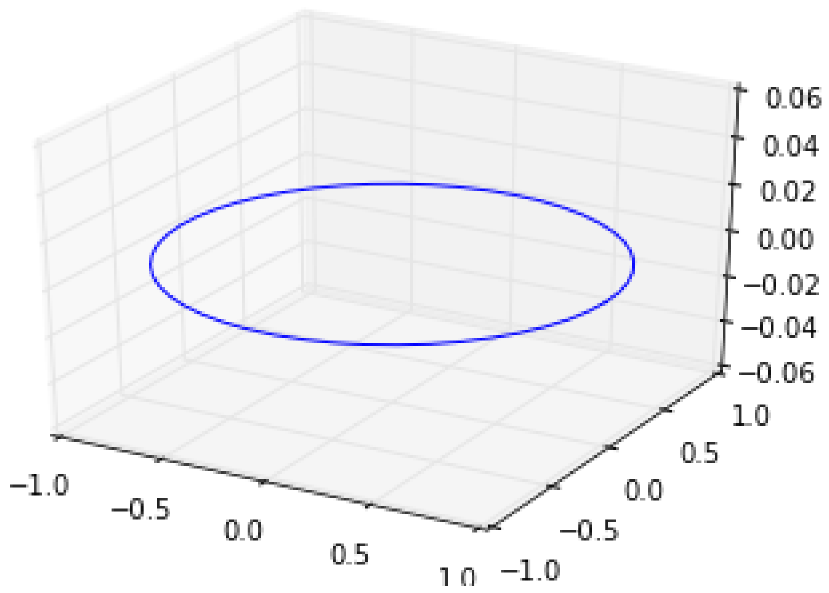
```
In [37]: theta = numpy.linspace(0, 2.0 * numpy.pi)
         x = numpy.cos(theta)
         y = numpy.sin(theta)
```

```
In [38]: pyplot.plot(theta, x, label=r"$x$")
         pyplot.plot(theta, y, label=r"$y$")
         pyplot.xlabel(r"$x$")
         pyplot.legend()
```

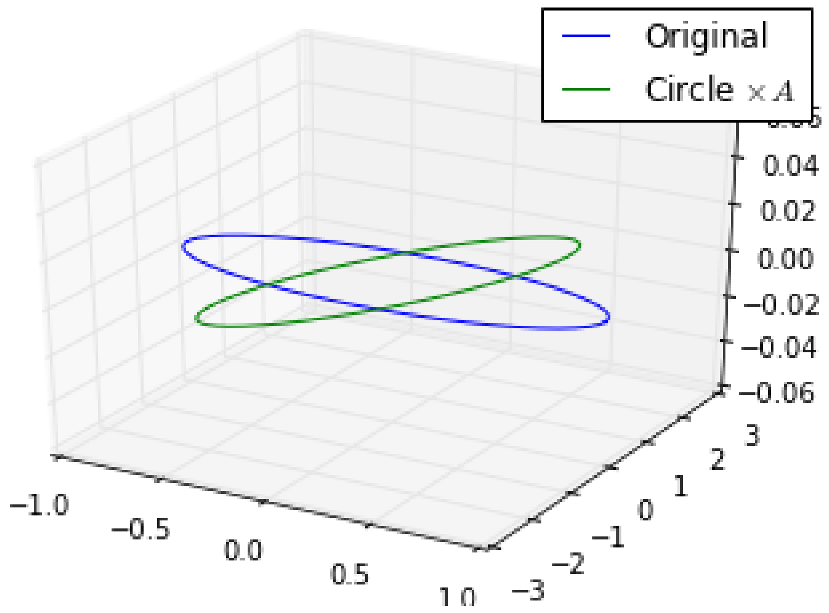
```
Out[38]: <matplotlib.legend.Legend at 0x1103d5f28>
```



```
In [39]: fig = pyplot.figure()
         ax = fig.add_subplot(111, projection='3d')
         ax.plot3D(x, y, 0);
```

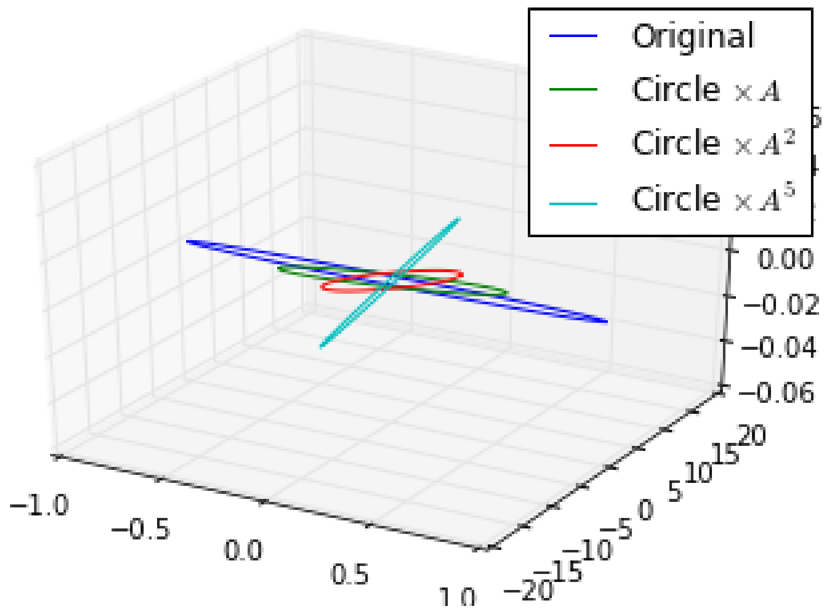


```
In [40]: A = numpy.random.randn(2,2)
         print(numpy.linalg.eig(A))

(array([-0.50979206, -1.48817901]), array([[ 0.44617681, -0.0108679 ],
       [-0.89494483,  0.99994094]]))


In [41]: x1 = x.copy()
         y1 = y.copy()
         r = numpy.vstack((x, y))
         for i in range(len(x)):
             r1 = numpy.dot(A, r[:,i])
             x1[i] = r1[0]
             y1[i] = r1[1]

In [42]: fig = pyplot.figure()
         ax = fig.add_subplot(111, projection='3d')
         ax.plot3D(x, y, 0, label = "Original")
         ax.plot3D(x1, y1, 0, label = r"Circle $\times A$")
         pyplot.legend();
```



```
In [43]: x2 = x.copy()
         y2 = y.copy()
         x5 = x.copy()
         y5 = y.copy()
         r = numpy.vstack((x, y))
         for i in range(len(x)):
             r2 = numpy.dot(numpy.linalg.matrix_power(A,2), r[:,i])
             r5 = numpy.dot(numpy.linalg.matrix_power(A,5), r[:,i])
             x2[i] = r2[0]
             y2[i] = r2[1]
             x5[i] = r5[0]
             y5[i] = r5[1]
```

```
In [44]: fig = pyplot.figure()
         ax = fig.add_subplot(111, projection='3d')
         ax.plot3D(x, y, 0, label = "Original")
         ax.plot3D(x1, y1, 0, label = r"Circle $\times A$")
         ax.plot3D(x2, y2, 0, label = r"Circle $\times A^2$")
         ax.plot3D(x5, y5, 0, label = r"Circle $\times A^5$")
         pyplot.legend();
```



The circles become ellipses. The semi-major axes align themselves with the eigenvectors. The size of the axis is multiplied by the appropriate eigenvalue each time: eigenvalues greater than one lead to the size diverging.