eigenanalysis

September 30, 2014

1 Eigenanalysis in Python

```
In [122]: import numpy as np
          import numpy.linalg as la # contains the eig() fxn
          import pandas as pd
          from matplotlib import pyplot as plt

          %matplotlib inline
```

1.0.1 Creating and reshaping an array

```
In [123]: # creat a floating pt array
          A = np.array([2,1,2,3], dtype=np.float)
          A

Out[123]: array([ 2.,  1.,  2.,  3.])
```

You can reshape the array by setting the shape attribute.

```
In [124]: A.shape = 2,2
          A

Out[124]: array([[ 2.,  1.],
                 [ 2.,  3.]])
```

1.0.2 Calculating eigenvalues and eigenvectors

```
In [125]: evals, evecs = la.eig(A)

In [172]: evals

Out[172]: array([ 4.,  1.])

In [127]: # eigenvectors
          evecs

Out[127]: array([[-0.70710678, -0.4472136 ],
                 [ 0.70710678, -0.89442719]])
```

Note that (somewhat inconveniently) the `eig()` function does not necessarily return the eigenvalues and eigenvectors in sorted fashion. The Numpy documentation states that the normalized eigenvector corresponding to the eigenvalue `w[i]` is the column `v[:,i]`. Also note that the `eig()` function returns normalized eigenvectors (i.e. each eigenvector has length 1).

We can get sort the eigenvectors by their eigenvaluesby exploiting the `argsort` function

```
In [171]: ?np.argsort
```

```
In [173]: # figure sorting order based on eigenvalues
          column_order = list(np.argsort(evals))

In [130]: column_order

Out[130]: [0, 1]

In [175]: # need to reverse the column order because want from large to small
          column_order.reverse()
          column_order

Out[175]: [1, 0]

In [180]: # an alternate trick for reverse a numpy array
          column_order = np.argsort(evals)
          column_order = column_order[::-1]
          column_order

Out[180]: array([0, 1])
```

We then use the `take()` function to get the columns of `evec` in the order specified by `column_order`.

```
In [132]: evals = np.take(evals, column_order)
          evecs = np.take(evecs, column_order, axis=1)  # take along columns

In [133]: print "eigenvalues (sorted): "
          print evals
          print
          print "eigenvectors (sorted by eigenvalues): "
          print np.array2string(evecs, precision=3)

eigenvalues (sorted):
[ 4.  1.]

eigenvectors (sorted by eigenvalues):
[[-0.447 -0.707]
 [-0.894  0.707]]
```

1.0.3 Mathematical relationships involving eigenvectors and eigenvalues

Let's confirm some of the basic mathematical relationships we discussed in lecture.

First up, let's show that:

$$\mathbf{Av} = k\mathbf{v}$$

where $\boldsymbol{v}$ is an eigenvector and k is the corresponding eigenvalue.

```
In [134]: Av = np.dot(A, evecs)
          kv = np.dot(evecs, np.diag(evals))

In [135]: # tests near equality
          np.allclose(Av, kv)

Out[135]: True
```

We should be able to reconstruct the original matrix A from the eigevectors and eigenvalues:

$$\boldsymbol{A} = \boldsymbol{V}\boldsymbol{L}\boldsymbol{V}^{-1}$$

where $\boldsymbol{V}$ is the matrix of eigvectors of $\boldsymbol{A}$ and $\boldsymbol{L}$ is a diagonal matrix filled with the eigenvalues of $\boldsymbol{A}$.

```
In [136]: L = np.diag(evals)
          L
```

```
Out[136]: array([[ 4.,  0.],
                 [ 0.,  1.]])
```

```
In [137]: V = evecs
          Vinv = la.inv(V)
          Vinv
```

```
Out[137]: array([[-0.74535599, -0.74535599],
                 [-0.94280904,  0.47140452]])
```

```
In [138]: VLVinv = np.dot(V, np.dot(L, Vinv))
          VLVinv
```

```
Out[138]: array([[ 2.,  1.],
                 [ 2.,  3.]])
```

```
In [139]: np.allclose(A, VLVinv)
```

```
Out[139]: True
```

Let's check for orthonogonality of the eigenvectors:

```
In [140]: np.dot(V[:,0], V[:,1])
```

```
Out[140]: -0.31622776601683789
```

If the vectors were orthogonal we'd expected their dot product to be zero. What's going on? Review your lecture notes if your baffled by this.

Let's define another matrix, $\boldsymbol{B}$.

```
In [141]: B = np.array([2,2,2,3])
          B.shape = 2,2
          B
```

```
Out[141]: array([[2, 2],
                 [2, 3]])
```

```
In [142]: uB, vB = la.eig(B)
```

```
In [143]: # these eigenvectors *are* orthogonal!
          np.dot(vB[:,0], vB[:,1])
```

```
Out[143]: 0.0
```

1.0.4 Geometric representation of eigenvectors in $\mathbb{R}^2$

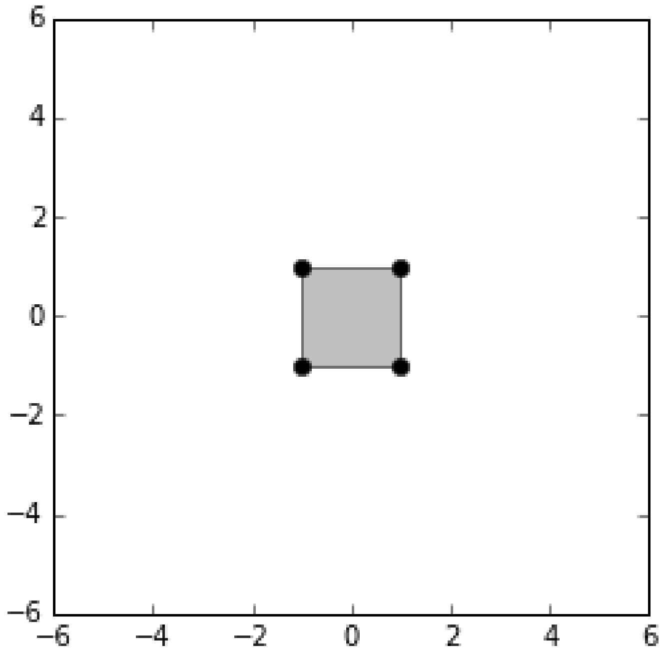
Since the matrix $\boldsymbol{A}$ and $\boldsymbol{B}$ above represent 2D linear transformations, we can visualize the effect of these transformations using points in the plane. We'll show how they distort a set of points that make up a square.

```
In [144]: pts = np.array([1,1,1,-1,-1,-1,-1,1])

          # by specifying -1,2 we're saying reshape so it fits a
          # 2 column matrix
          pts.shape = -1,2
          pts
```

```
Out[144]: array([[ 1,  1],
                 [ 1, -1],
                 [-1, -1],
                 [-1,  1]])
```

```
In [164]: from matplotlib import patches
          fig, ax = plt.subplots()
          polygon = patches.Polygon(pts, edgecolor='black', facecolor='grey', alpha=0.5)
          plt.plot(pts[:,0], pts[:,1], 'ko')
          ax.add_patch(polygon)
          ax.set_aspect('equal')
          plt.xlim(-6,6)
          plt.ylim(-6,6)
```

```
Out[164]: (-6, 6)
```

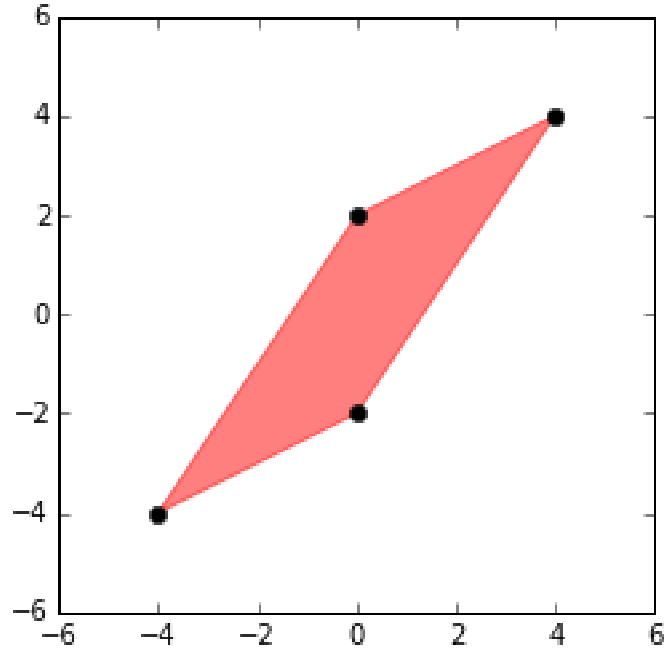



```
In [152]: ptsA = np.dot(pts, A)
          ptsA
```

```
Out[152]: array([[ 4.,  4.],
                 [ 0., -2.],
                 [-4., -4.],
                 [ 0.,  2.]])
```

```
In [153]: fig, ax = plt.subplots()
          polygon = patches.Polygon(ptsA, color='red', alpha=0.5)
          plt.plot(ptsA[:,0], ptsA[:,1], 'ko')
          ax.add_patch(polygon)
          ax.set_aspect('equal')
          plt.xlim(-6,6)
          plt.ylim(-6,6)
```

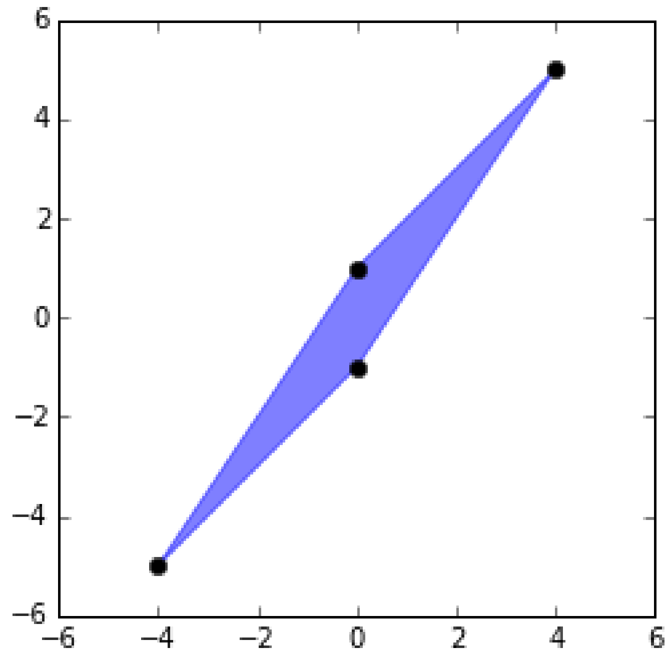
```
Out[153]: (-6, 6)
```




```
In [154]: ptsB = np.dot(pts, B)
```

```
In [161]: fig, ax = plt.subplots()
          polygon = patches.Polygon(ptsB, color='blue', alpha=0.5)
          plt.plot(ptsB[:,0], ptsB[:,1], 'ko')
          ax.add_patch(polygon)
          ax.set_aspect('equal')
          plt.xlim(-6,6)
          plt.ylim(-6,6)
```

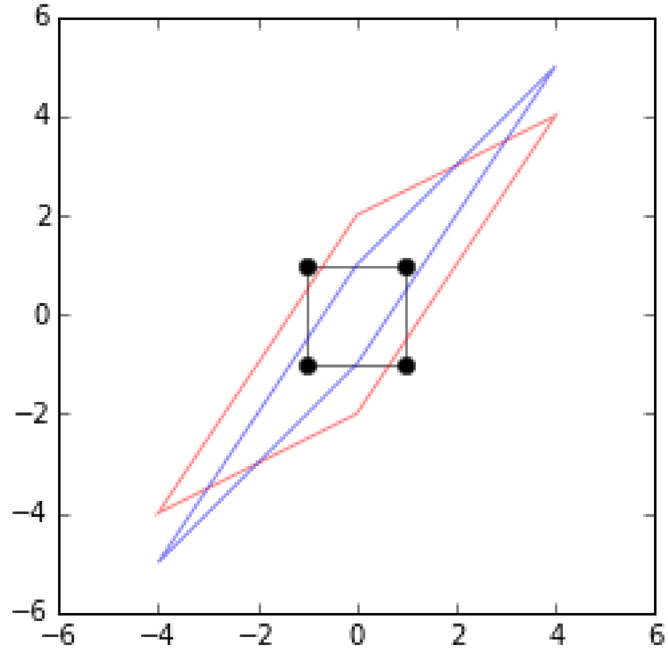
```
Out[161]: (-6, 6)
```



Let's combine the three plots.

In [170]:
```
fig, ax = plt.subplots()

polygon = patches.Polygon(pts, edgecolor='black', facecolor='None', alpha=0.5)
plt.plot(pts[:,0], pts[:,1], 'ko')
ax.add_patch(polygon)

polygonA = patches.Polygon(ptsA, edgecolor='red', facecolor='None', alpha=0.5)
ax.add_patch(polygonA)

polygonB = patches.Polygon(ptsB, edgecolor='blue', facecolor='None', alpha=0.5)
ax.add_patch(polygonB)

ax.set_aspect('equal')
plt.xlim(-6,6)
plt.ylim(-6,6)
```

Out[170]:
```
(-6, 6)
```


6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6