Random Matrix Theory (Sethna exercise 1.6)

(See also http://pages.physics.cornell.edu/~myers/teaching/ComputationalMethods/ComputerExercises/RandomMatrixTheory/RandomMatrixTheory.html)

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

Create routine for generating a Gaussian Orthogonal Ensemble of Size x Size matrices

In [ ]:
```
def GOE(size, number):
    """
    Creates an array of size x size matrices in the GOE ensemble, of length 'number'
    Use 'scipy.random.standard_normal((N,S,S)) to generate N SxS random matrices drawn
    from a Gaussian ('normal') distribution of zero mean and standard deviation one.
    Add each matrix to its transpose. When given a three-index array M_nij (i.e., a list of
N matrices),
    M_nij.transpose((0,2,1)) will leave the first ('0') index alone, and swap the second and
 third (2->1 and 1->2).
    """
    matrices =
    return matrices + matrices.transpose(...)
```

Define a routine that finds the central eigenvalue difference

In [ ]:
```
def CenterEigenvalueDifferences(ensemble):
    """
    For each matrix in 'ensemble', calculate the difference between the two center eigenvalu
es.
     * You can find the size of the matrices using 'len'
     * scipy.linalg.eigvalsh(m) gives the eigenvalues of a symmetric matrix m
     * This is a nice place to use 'list comprehensions'. In python you can make a list like
            blah = [f(b) for b in bloo]
     * which will apply the function f (say scipy.linalg.eigvalsh) to each element of a list
 bloo (say a list of GOE matrices).
     * Do this twice, the second time taking e[size/2]-e[size/2-1] for e in eigs.
     * (Note that the arrays in python start at zero, so for an even-sized matrix size/2 is
the element
     *  just past half-way, and size/2-1 just before half-way)
    """
    ...
```

Make a figure showing the normalized histogram (pylab.hist(blah, normed->True)) of the differences. Compare with the Wigner surmise. (First make an array of differences. Then find the average. Then blah should be diffs/diffAve; use 50 bins.

In [ ]:
```
diffs = ...;
diffAve = ...
pylab.hist(..., ..., bins=50);
s = scipy.arange(0.0,3.5,0.01);
pylab.plot(s, "WignerSurmise"(s))
```

Create a routine that generates symmetric matrices of random +-1

In [ ]:
```
def symmetrize(m):
    """
    Takes a matrix m, and replaces the elements below the diagonal with the elements above t
he diagonal.
    """
    for i in range(len(m)):
        for j in ...:
            if i>j:
                m[i][j] = ...
    return m

def PM1(size, number):
    """
    Creates an array of size x size symmetric matrices filled with +-1 at random.
    Note that scipy.random.randint(2, size = (number,size,size)) will give you a matrix fill
ed at random
    with whole numbers less than 2 -- that is, 0's and 1's. Multiply the array by two and su
btract one.
    Note that Python subtracts integers from arrays by subtracting it from each array elemen
t.
    You can then use symmetrize to return symmetric matrices.
    """
    matrices = scipy.random.randint(2, size=(number,size,size)) * 2. - 1.
    return [symmetrize(m) for m in matrices]
```

Make a figure showing the normalized histogram (pylab.hist(blah, normed->True)) of the eigenvalue differences for matrices of +-1. Compare with the Wigner surmise.

In [ ]:
```
diffs = ...
...
```