In [1]:
```
%matplotlib notebook
import scipy
import scipy.sparse as sparse
import scipy.sparse.linalg as spalin
import numpy as np
from scipy import linalg
import matplotlib.pyplot as plt

np.random.seed(5)
```

This is the list of functions I'm going to use. I would usually have these in a separate "module".

Build lists of x,y,z Pauli matrices on each site. Write general code for constructing two-body and one-body terms in Hamiltonians.
Obtain the level statistics distributions from the spectrum. Contruct projectors onto "diagonal" symmetry sectors

In [2]:
```
def gen_s0sxsysz(L):
    sx = sparse.csr_matrix([[0., 1.],[1., 0.]])
    sy = sparse.csr_matrix([[0.,-1j],[1j,0.]])
    sz = sparse.csr_matrix([[1., 0],[0, -1.]])
    s0_list =[]
    sx_list = []
    sy_list = []
    sz_list = []
    I = sparse.csr_matrix(np.eye(2**L))
    for i_site in range(L):
        if i_site==0:
            X=sx
            Y=sy
            Z=sz
        else:
            X= sparse.csr_matrix(np.eye(2))
            Y= sparse.csr_matrix(np.eye(2))
            Z= sparse.csr_matrix(np.eye(2))

        for j_site in range(1,L):
            if j_site==i_site:
                X=sparse.kron(X,sx, 'csr')
                Y=sparse.kron(Y,sy, 'csr')
                Z=sparse.kron(Z,sz, 'csr')
            else:
                X=sparse.kron(X,np.eye(2),'csr')
                Y=sparse.kron(Y,np.eye(2),'csr')
                Z=sparse.kron(Z,np.eye(2),'csr')
        sx_list.append(X)
        sy_list.append(Y)
        sz_list.append(Z)
        s0_list.append(I)

    return s0_list, sx_list,sy_list,sz_list


def gen_op_total(op_list):
    L = len(op_list)
    tot = op_list[0]
    for i in range(1,L):
        tot = tot + op_list[i]
    return tot

def gen_onsite_field(op_list, h_list):
    L= len(op_list)
    H = h_list[0]*op_list[0]
    for i in range(1,L):
        H = H + h_list[i]*op_list[i]
    return H

# generates \sum_i O_i O_{i+k} type interactions
def gen_interaction_kdist(op_list, op_list2=[],k=1, J_list=[], bc='obc'
):
    L= len(op_list)

    if op_list2 ==[]:
```

```
        op_list2=op_list
    H = sparse.csr_matrix(op_list[0].shape)
    if J_list == []:
        J_list =[1]*L
    Lmax = L if bc == 'pbc' else L-k
    for i in range(Lmax):
        H = H+ J_list[i]*op_list[i]*op_list2[np.mod(i+k,L)]
    return H

def gen_nn_int(op_list, J_list=[], bc='obc'):
    return gen_interaction_kdist(op_list,op_list, 1, J_list, bc)

def LevelStatistics(energySpec, ret=True):
    delta = energySpec[1:] -energySpec[0:-1]
    r = list(map(lambda x,y: min(x,y)*1.0/max(x,y), delta[1:], delta[0:-
1]))
    if ret==True:
        return np.array(delta), np.array(r), np.mean(r)
    return np.mean(r)

def gen_diagprojector(symvec, symval):
    ind = np.where(symvec==float(symval))
    dim = np.size(ind)
    P = sparse.lil_matrix((dim,len(symvec)))
    for j in range(dim):
        P[j,ind[0][j]] = 1.0
    return P

### !!!! Assumes real eigenvectors!!!!!! #### (Computation is faster if
 you know your eigenvectors are real)
def gen_diagonal_ME(op, evecs):
    return np.sum(evecs*np.dot(op, evecs),0)
```

Random Matrix Level Statistics

2x2 random matrices and the Wigner surmise

Let's start by looking at the distribution of level spacings for a 2x2 real, symmetric Hamiltonian with matrix elements drawn from the Gaussian ensemble ("GOE = Gaussian Orthogonal Ensemble").

$$H = \begin{pmatrix} h_{11} & h_{12} \\ h_{12} & h_{22} \end{pmatrix}.$$

where h_{11}, h_{12}, h_{22} are drawn randomly and independently from the normal distribution. The eigenvalues for this Hamiltonian are

$$E_{1,2} = \frac{h_{11}+h_{22}}{2} \pm \frac{1}{2}\sqrt{(h_{11}-h_{22})^2 + 4h_{12}^2}.$$

You can easily derive (try this using the steps outlined in lecture!) that the distribution of level spacings $s = E_1 - E_2 = \sqrt{(h_{11}-h_{22})^2 + 4h_{12}^2}$ agrees with the predicted Wigner surmise (1950s):

$$P_{GOE}(s) = \frac{1}{2}\pi s e^{-\frac{1}{4}\pi s^2}.$$

Note that I'll use s and Δ interchangeably to refer to the level spacings in the spectrum: $\Delta_n = s_n = E_n - E_{n-1}$. This distribution goes to zero as $s \to 0$, reflecting the "level repulsion" characteristic of chaotic systems. Indeed, as the expression for s shows, even if h_{11} and h_{22} are degenerate, the eigenvalues are not degenerate because of the off-diagonal coupling h_{12}. That is, two levels that try to become degenerate are still "repelled" away from each other due to the coupling h_{12}.

A random matrix (we'll consider larger random matrices momentarily) is a good model for a chaotic system since, when viewed as a Hamiltonian, it reflects a system with all degrees of coupled to each other in a completely random fashion. Remarkably, local thermalizing spin 1/2 local Hamiltonians (which may have no random parameters at all) also show GOE level statistics! When written as a matrix, these local Hamiltonian models are extremely sparse. So the fact that their eigenspectra show the same statistics as dense random matrices isn't obvious a priori. We're only now (2018) starting to prove how random matrix statistics arises in such models in a few specific cases (Work by Prosen group and Chalker group).

By contrast, if the off-diagonal terms in the Hamiltonian $h_{12} = h_{21}$ are forced to be zero, say due to symmetry reasons, then the eigenvalues (equal to the two diagonal entries) are uncorrelated from each each other and their spacing follows a Poisson distribution characteristic of the intervals between uncorrelated random events. The two levels can freely pass through each other and there is no level repulsion. Which means level spacings can be zero with high probability. Similar Poisson statistics are found in integrable systems with extensively many conservation laws (since these extra symmetries prevent generic mixing between spectra), and in MBL systems with extensively many "emergent" conservation laws -- the l-bits.

$$P_{Poisson}(s) = e^{-s}.$$

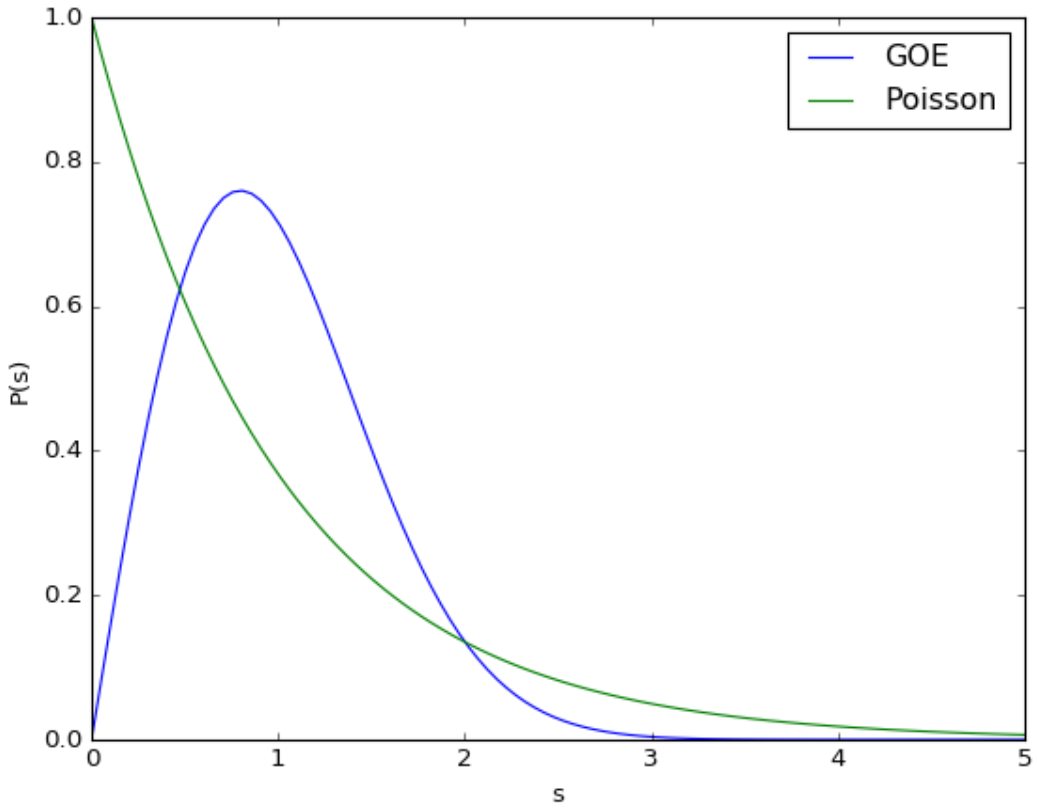
Note that both the GOE and Poisson distributions are normalized so that the mean level spacing is 1:

$$\langle s \rangle = \int ds\ s\ P(s) = 1$$

This normalization is important to arrive at a universal distribution for level spacings. Of course, if we scale all entries of our matrix by a factor of two, the spacing will also be scaled up by a factor of two. So this normalization is done to get rid of such non-universal effects. When we consider larger NxN random matrices

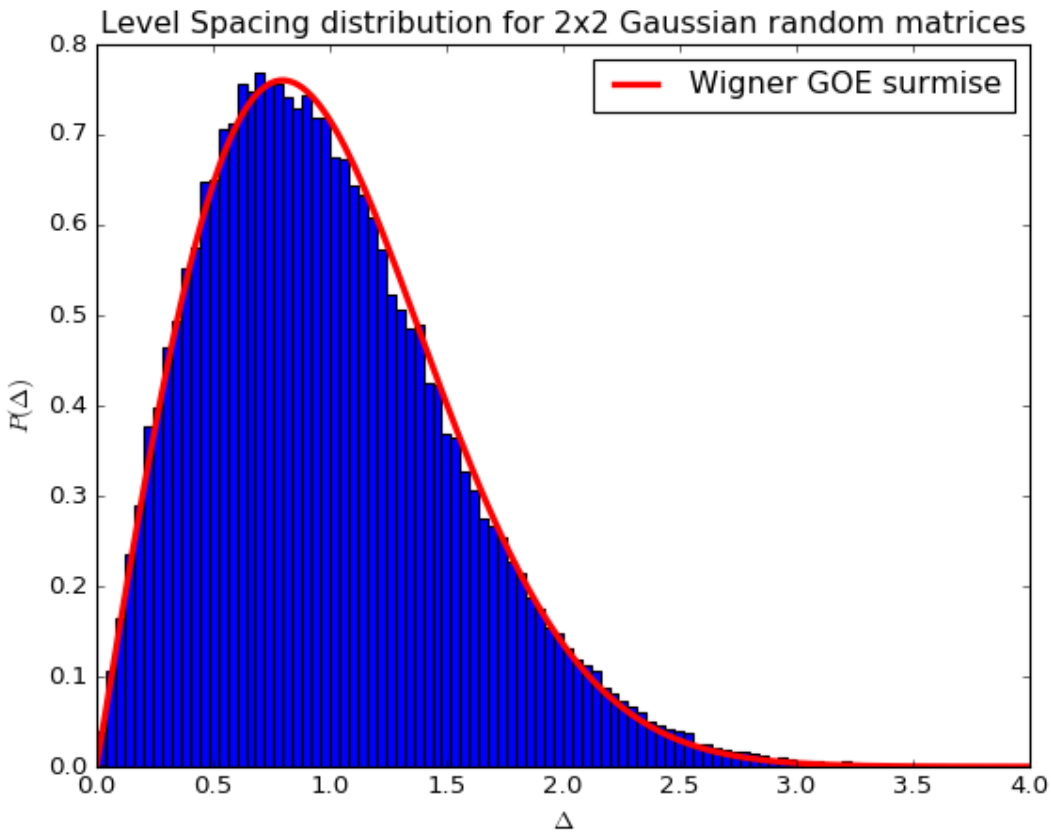
(below), the spacings will depend on N so we will again have to adjust the spectrum to have $\langle s \rangle = 1$. Such adjustments are called "spectral unfolding".

In [3]:
```
## Plot GOE and Poisson distributions
slist = np.linspace(0,5,100)
GOE = 0.5*np.pi*slist*np.exp(-0.25*np.pi*slist*slist)
Poisson = np.exp(-slist)
plt.figure()
plt.plot(slist, GOE, label='GOE' )
plt.plot(slist, Poisson, label='Poisson')
_=plt.legend()
_=plt.xlabel('s')
_=plt.ylabel('P(s)')
```



In [4]:
```
# Verify Wigner surmise for 2x2 random matrices

niter = 100000
delta = np.zeros(niter)
for n in range(niter):
    randlist = np.random.normal(size = 3)
    M = np.zeros((2,2))
    M[0,0] = randlist[0]
    M[1,1] = randlist[1]
    M[0,1] = M[1,0]= randlist[2]

    evals = linalg.eigvalsh(M)
    delta[n] = evals[1]-evals[0]
```

In [5]:
```
plt.figure()
_=plt.hist(delta/np.mean(delta), bins=100, normed=True)  #I divide by th
e mean to have avg. spacing = 1.

x = np.linspace(0, 4,200)
GOE = 0.5*np.pi*x*np.exp(-0.25*np.pi*x*x) ### Wigner surmise
_=plt.plot(x,GOE, 'r', lw=3, label = 'Wigner GOE surmise')
_=plt.xlabel(r'$\Delta$')
_=plt.ylabel(r'$P(\Delta)$')
_=plt.title('Level Spacing distribution for 2x2 Gaussian random matrice
s' )
_=plt.legend()
```



NxN random matrices

Now let us do this for a large NxN random matrices, again drawn from the GOE ensemble. Since the level spacings will depend on N so we will have to adjust the spectrum to normalize the mean level spacing. I'll consider two different spectral unfolding procedures below for normalizing the mean level spacing to one: a "naive" procedure and a more sophisticated one (which is standard in the literature). The naive method simply normalizes s_i by $\langle s \rangle$. But this doesn't work so well because the density of states is not uniform -- thus, spacings near $E = 0$ where the spectrum is the densest (and the level spacing smallest) should have been normalized differently from spacings near the edges of the spectrum. A standard technique for more properly unfolding the spectrum obtains the "staircase function" $N(E)$ as the number of levels upto energy E. This has a smooth part $N_{\rm sm}$ and a fluctuating part. The unfolded spectrum is defined as $\epsilon_i = N_{\rm sm}(E_i)$. On average, the local density of states for this unfolded spectrum is approximately one.

Finally, I should mention that there are many different approaches for spectral unfolding (which can be quite ad hoc) and there are papers detailing how various aspects of the spectrum can be quite sensitive to particular unfolding techniques. Thus, for looking at level spacing distributions, it's safest to switch to a different measure of level spacings, the r-ratio, defined by Oganesyan and Huse which precludes the need for unfolding by considering a dimensionless ratio of adjacent level spacings: $r_n = \frac{\min\{\Delta_n, \Delta_{n-1}\}}{\max\{\Delta_n, \Delta_{n-1}\}}$. Notice that $r \in [0, 1]$ so all dependences on system sizes have been removed. Atas et. al. (PRL 2013) have derived the corresponding GOE/Poisson distributions for the r-ratio and these take the form:

$$P_{GOE}(r) = \frac{27}{4} \frac{r + r^2}{(1 + r + r^2)^{5/2}}$$

$$P_{Poisson}(r) = \frac{2}{(1+r)^2}$$

Below, I'll show the level spacing distributions using all three techniques: "naive", staircase unfolded, and r-ratio.

In [6]:
```
## Constructing a symmetric random matrix efficiently
a= np.random.normal(size=(4,4))
np.set_printoptions(precision=2)
print(a)
print('\n')
print(np.tril(a))
print('\n')
print(np.tril(a,-1).T)
print('\n')
print(np.tril(a)+np.tril(a,-1).T)
```

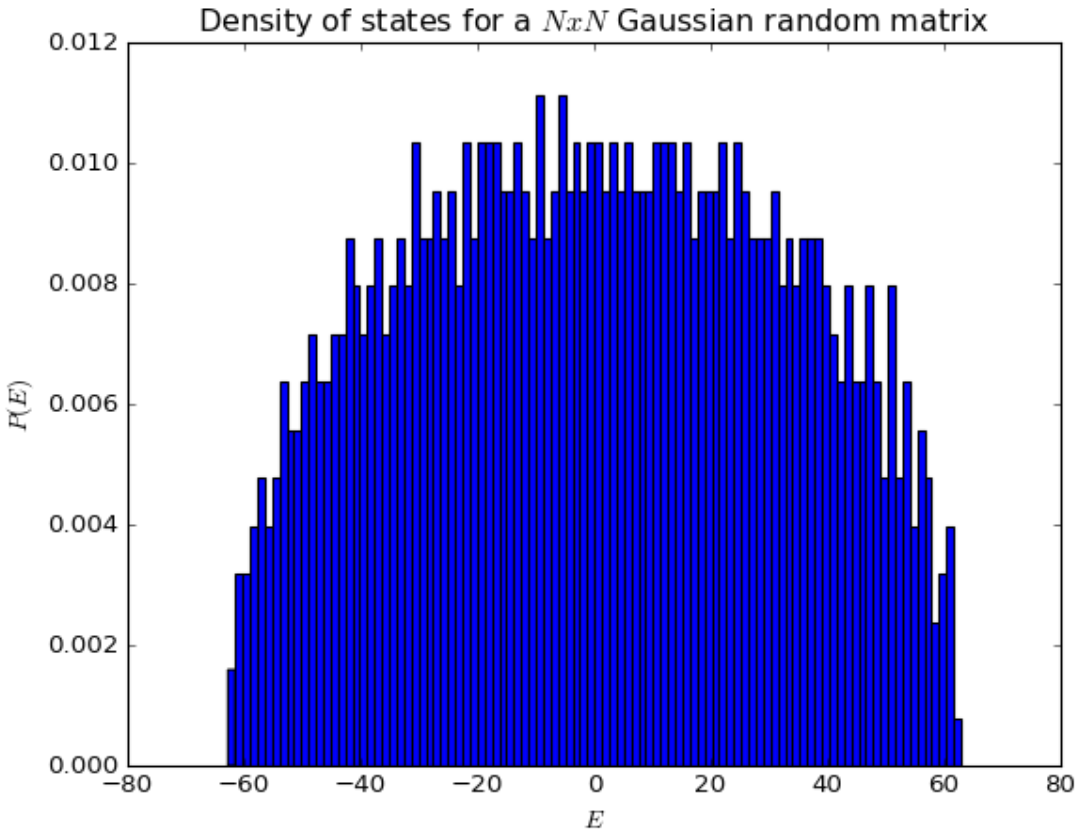
```
[[ 0.58  0.6  -0.52 -1.25]
 [ 0.06 -0.63 -0.   -1.23]
 [ 0.04 -0.37 -0.14  0.72]
 [ 1.46 -1.24 -0.9  -0.3 ]]


[[ 0.58  0.    0.    0.  ]
 [ 0.06 -0.63  0.    0.  ]
 [ 0.04 -0.37 -0.14  0.  ]
 [ 1.46 -1.24 -0.9  -0.3 ]]


[[ 0.    0.06  0.04  1.46]
 [ 0.    0.   -0.37 -1.24]
 [ 0.    0.    0.   -0.9 ]
 [ 0.    0.    0.    0.  ]]


[[ 0.58  0.06  0.04  1.46]
 [ 0.06 -0.63 -0.37 -1.24]
 [ 0.04 -0.37 -0.14 -0.9 ]
 [ 1.46 -1.24 -0.9  -0.3 ]]
```

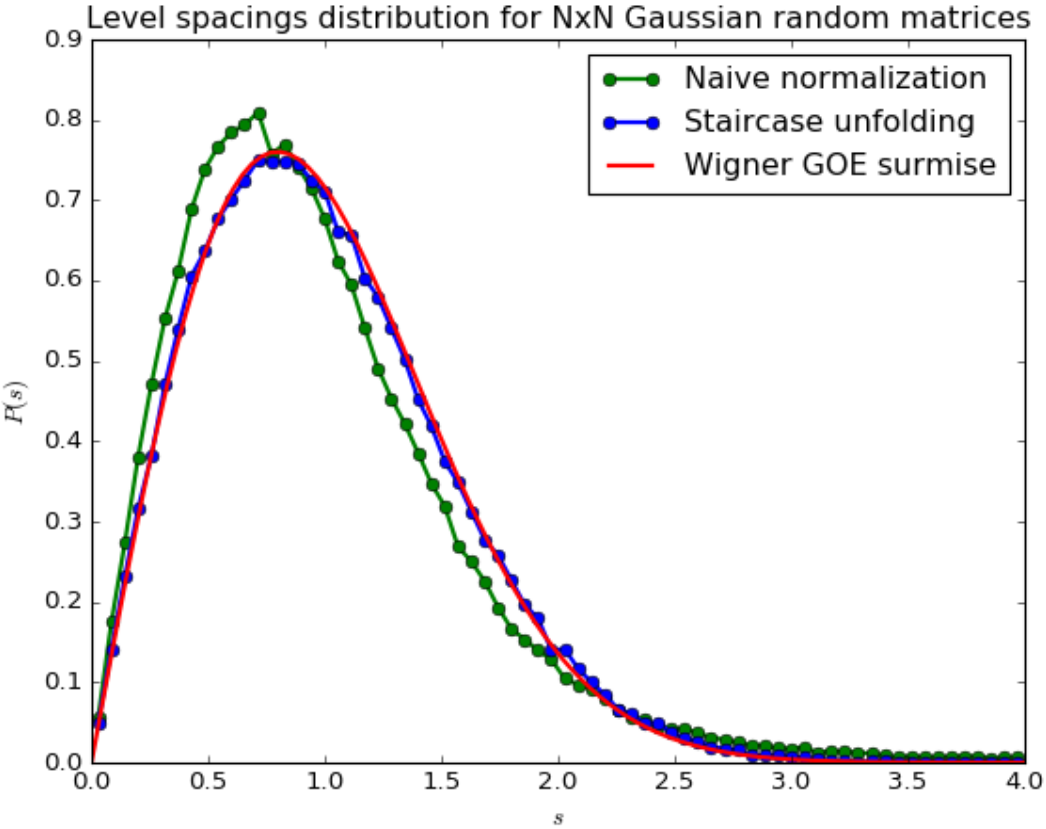
In [7]:
```
## Illustration of unfolding using staircase function
N = 50
a = np.random.normal(size=(N,N))
M = np.tril(a) + np.tril(a,-1).T
evals = linalg.eigvalsh(M)

Nsmooth = np.poly1d(np.polyfit(x=evals, y = range(len(evals)), deg=5))

plt.figure()
plt.step(evals,range(len(evals)), '.-', label='$N(E_i)$', where = 'mid')
plt.plot(evals,Nsmooth(evals), 'r-', label='$N_{sm}(E_i)$')
plt.xlabel(r'$E$')
plt.ylabel(r'$N(E)$')
plt.legend(loc=2)
_=plt.ylim(-1, N)
```

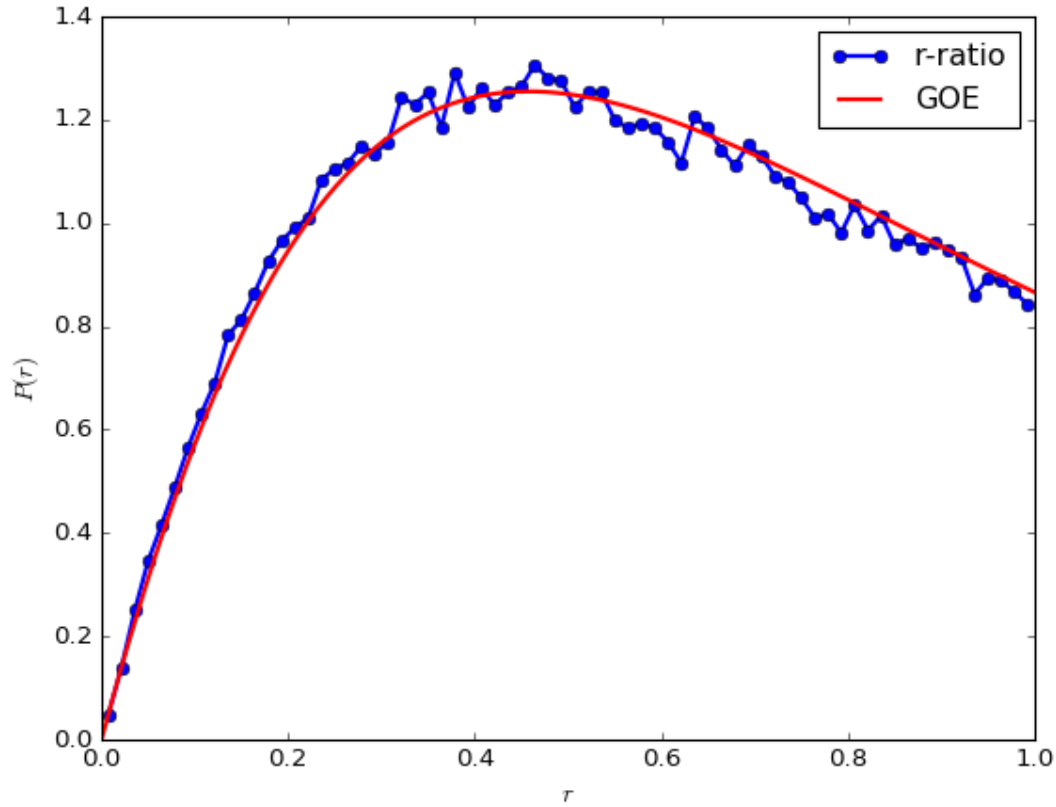
In [6]:
```
## Obtain spectra -- takes a few minutes to run

niter = 100 # number of disorder iterations
N = 1000 #size of random matrix
deltalist = np.zeros((niter, N-1))
deltalist_unfold = np.zeros((niter, N-1))
rlist = np.zeros((niter, N-2))

for n in range(niter):

    #construct matrix
    a = np.random.normal(size=(N,N))
    M = np.tril(a) + np.tril(a,-1).T
    evals = linalg.eigvalsh(M)

    # get the unfolded spectrum
    Nsmooth = np.poly1d(np.polyfit(x=evals, y = range(N), deg=5))
    evalssmooth = Nsmooth(evals)

    # get the level spacings (naively normalized, staircase unfolded, r-
ratio)
    deltalist[n] = evals[1:]-evals[0:-1]
    deltalist[n] = deltalist[n]/np.mean(deltalist[n])  # Naive
    deltalist_unfold[n] = evalssmooth[1:]-evalssmooth[0:-1] # Staircase
    _, rlist[n],_ = LevelStatistics(evals) #r-ratio
```

In [7]:
```
# obtain probability distributions for the three level-spacing measures
nbins = 70
bslist = np.zeros((3,niter, nbins+1))
hslist = np.zeros((3,niter, nbins))

for n in range(niter):
    hslist[0,n],bslist[0,n],  = np.histogram(deltalist[n], bins=nbins, r
ange=(0,4), normed=True)
    hslist[1,n],bslist[1,n],  = np.histogram(deltalist_unfold, bins=nbin
s, range=(0,4), normed=True)
    hslist[2,n],bslist[2,n],  = np.histogram(rlist[n], bins=nbins, range
 = (0,1), normed=True)
```

In [8]:
```
_=plt.figure()
_=plt.hist(evals, bins=100, normed=True)
_=plt.xlabel(r'$E$')
_=plt.ylabel(r'$P(E)$')
_=plt.title(r'Density of states for a $NxN$ Gaussian random matrix')

_=plt.figure()
x = np.linspace(0, 4,100)
GOE = 0.5*np.pi*x*np.exp(-0.25*np.pi*x*x) ### Wigner surmise
bslistmean =np.mean(bslist,1)
hslistmean =np.mean(hslist,1)
_=plt.plot(0.5*(bslistmean[0,1:]+bslistmean[0,0:-1]),hslistmean[0], 'go-
',lw=2,label = 'Naive normalization')
_=plt.plot(0.5*(bslistmean[1,1:]+bslistmean[1,0:-1]),hslistmean[1], 'bo-
',lw=2,label = 'Staircase unfolding')
_=plt.plot(x,GOE, 'r', lw=2, label='Wigner GOE surmise')
_=plt.xlabel(r'$s$')
_=plt.ylabel(r'$P(s)$')
_=plt.title('Level spacings distribution for NxN Gaussian random matrice
s' )
_=plt.xlim([0,4])
_=plt.legend()


_=plt.figure()
_=plt.plot(0.5*(bslistmean[2,1:]+bslistmean[2,0:-1]),hslistmean[2], 'o-'
,lw=2,label = 'r-ratio')
xarr=np.linspace(0,1,100)
Poi = 2/(1+xarr)**2
GOE = (27/4.)*(xarr+xarr**2)/(1+xarr+xarr**2)**(5/2.)
_=plt.xlabel(r'$r$')
_=plt.ylabel(r'$P(r)$')
_=plt.plot(xarr, GOE, 'r', lw=2,label='GOE')
_=plt.legend()
```


Density of states for a NxN Gaussian random matrix
0.012
0.010
0.008
0.006
0.004
0.002
0.000
$P(E)$
−80
−60
−40
−20
0
20
40
60
80
E


Level spacings distribution for NxN Gaussian random matrices
Naive normalization
Staircase unfolding
Wigner GOE surmise
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
$P(s)$
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
s



Note that the density of states has the famous "semicircle distribution" for random matrices.

While the level spacing distribution for the "naive" normalization shows the same qualitative features as the Wigner surmise (level repulsion etc.), there are quantitative deviations due to the poor normalization procedure which doesn't take into account variations in the density of states. On the other hand, the spacing distribution for the properly unfolded spectrum using the "staircase" unfolding procedure agrees almost exactly with the Wigner surmise. Finally, the r-ratio distribution which obviates the need for unfolding again agrees well with its prediced form.

Level Statistics across the MBL Transition

We now switch to studying the r-ratio across the MBL transition.

Pal Huse Model

$H = \sum_i S_i^x S_{i+1}^x + S_i^y S_{i+1}^y + S_i^z S_{i+1}^z + h_i S_i^z$

$S_i^x = \frac{1}{2}\sigma_i^x$ are spin 1/2 operators on site i.

The fields h_i are drawn uniformly from a random distribution of width W. This model has a transition from a thermalizing phase to an MBL phase at W~4.

In [10]:
```
L=2
W = 10.0

s0, x,y,z = gen_s0sxsysz(L)
Hint = 0.25*(gen_nn_int(x) + gen_nn_int(y) + gen_nn_int(z))  #0.25 is fo
r the conversion from Pauli's to spin 1/2s.

hlist = np.random.uniform(-W/2.0, W/2.0, L) #The factor of two is for th
e conversion from Pauli's to spin 1/2s.
H = Hint + gen_onsite_field(hlist, z)
```

Do a basic sanity check that the Hamiltonian you've constructed looks right on a small system

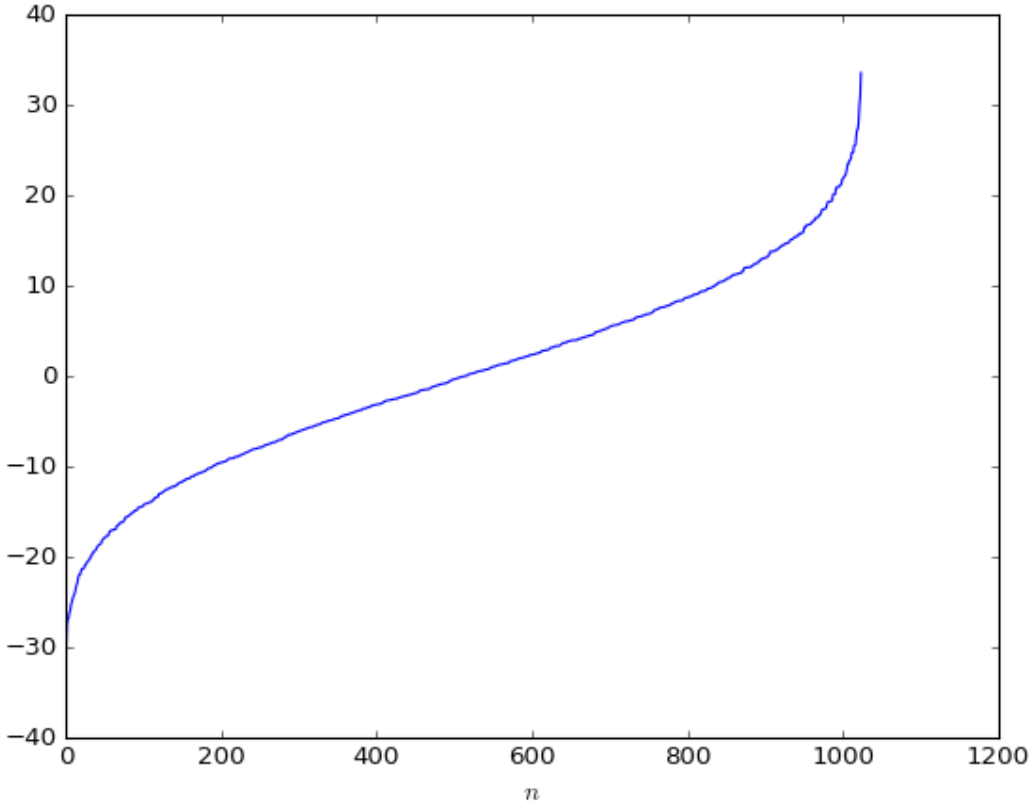
In [11]:
```
np.round(np.real(H.toarray()),2)
```

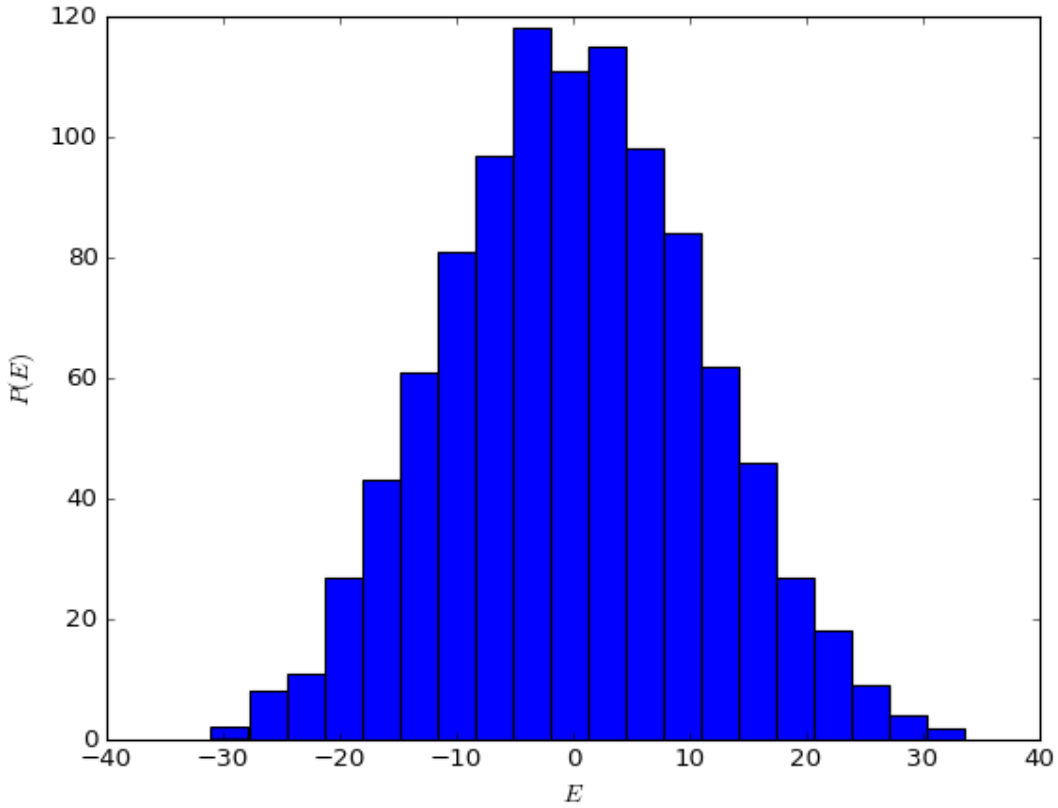
Out[11]:
```
array([[ 0.59,  0.  ,  0.  ,  0.  ],
       [ 0.  ,  7.75,  0.5 ,  0.  ],
       [ 0.  ,  0.5 , -8.25,  0.  ],
       [ 0.  ,  0.  ,  0.  , -0.09]])
```

Diagonalize the Hamiltonian and look at the spectrum

In [12]:
```
L=10
W = 10.0

s0, x,y,z = gen_s0sxsysz(L)
Hint = 0.25*(gen_nn_int(x) + gen_nn_int(y) + gen_nn_int(z))  #0.25 is fo
r the conversion from Pauli's to spin 1/2s.

hlist = np.random.uniform(-W/2.0, W/2.0, L) #The factor of two is for th
e conversion from Pauli's to spin 1/2s.
H = Hint + gen_onsite_field(hlist, z)

evals, evecs = linalg.eigh(np.real(H.toarray()))
```

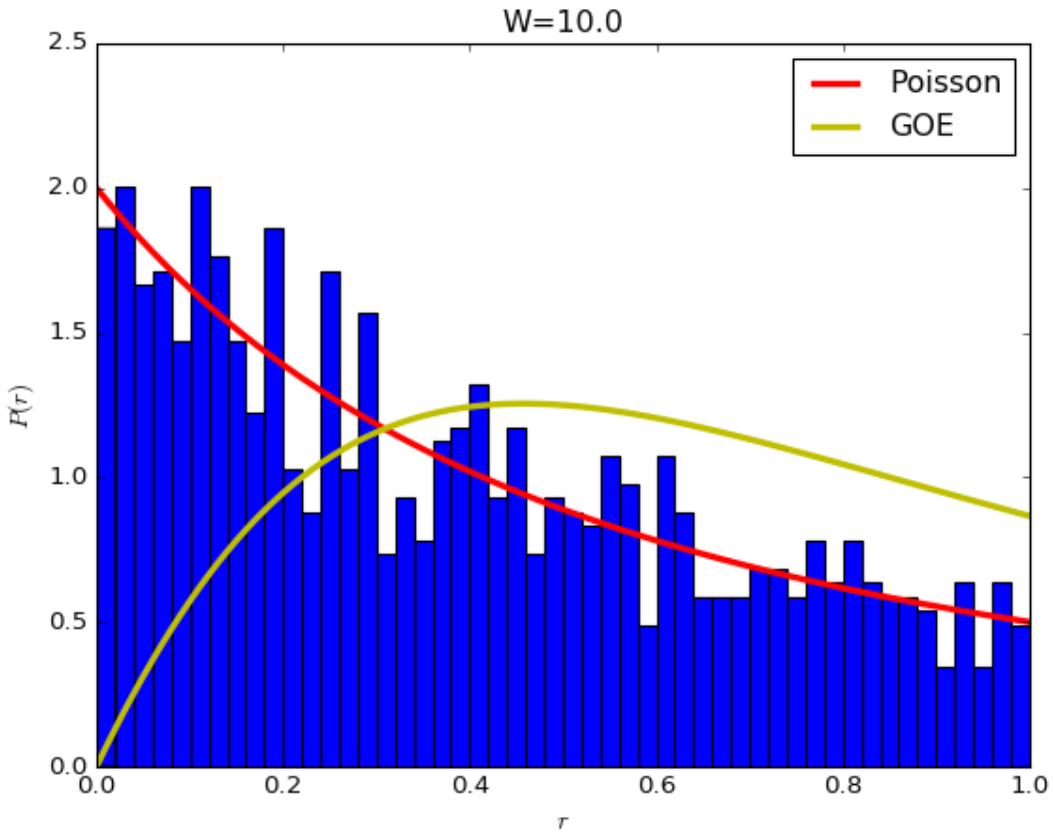
In [13]:
```
plt.figure()
_=plt.plot(evals)
_=plt.ylabel(r'$E_\n$')
_=plt.xlabel(r'$n$')
plt.figure()
_=plt.hist(evals, bins=20)
_=plt.ylabel(r'$P(E)$')
_=plt.xlabel(r'$E$')
```


40
30
20
10
0
−10
−20
−30
−40
0
200
400
600
800
1000
1200
n

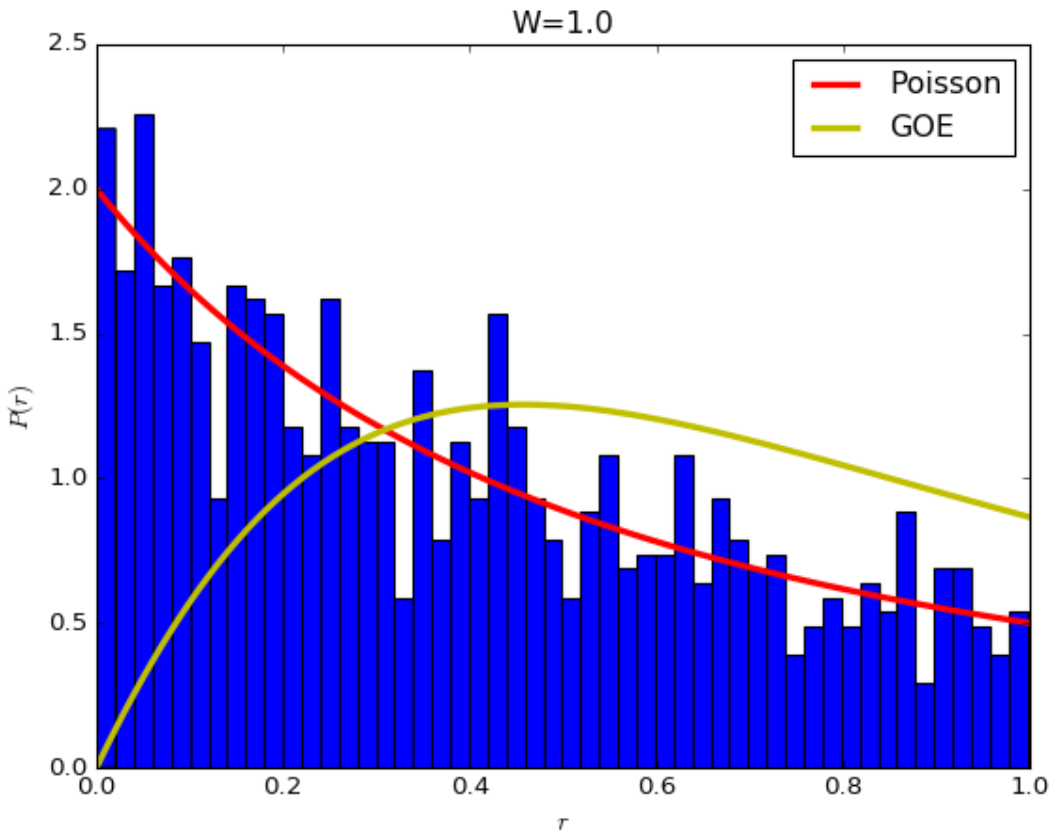

120
100
80
60
40
20
0
$P(E)$
−40
−30
−20
−10
0
10
20
30
40
E

Note that for this sparse local Hamiltonian with only two-body interactions, the density of states looks like a Gaussian instead of a semicircle. Thus, while RMT predicts many features of the spectrum (like level spacing distribution), there are differnces.

In [14]:
```
# Look at the r-ratio distribution in the MBL phase. This should look Po
isson

W = 10.0
hlist = np.random.uniform(-W/2.0, W/2.0, L) #The factor of two is for th
e conversion from Pauli's to spin 1/2s.
H = Hint + gen_onsite_field(hlist, z)
evals, evecs = linalg.eigh(np.real(H.toarray()))


delta,rlist,r = LevelStatistics(evals)
_=plt.figure()
_=plt.hist(rlist, normed=True, bins=50)
plt.xlabel('$r$')
plt.ylabel('$P(r)$')
plt.title('W=%.1f'%W)
xarr=np.linspace(0,1,100)
Poi = 2/(1+xarr)**2
GOE = (27/4.)*(xarr+xarr**2)/(1+xarr+xarr**2)**(5/2.)
_=plt.plot(xarr,Poi, 'r', lw='3', label='Poisson')
_=plt.plot(xarr,GOE, 'y', lw='3', label='GOE')
_=plt.legend()
```



In [15]:
```
# Now look at the r-ratio distribution for a W in the thermalizing phas
e. This should look GOE


W = 1.0
hlist = np.random.uniform(-W/2.0, W/2.0, L) #The factor of two is for th
e conversion from Pauli's to spin 1/2s.
H = Hint + gen_onsite_field(hlist, z)
evals, evecs = linalg.eigh(np.real(H.toarray()))

delta,rlist,r = LevelStatistics(evals)
_=plt.figure()
_=plt.hist(rlist, normed=True, bins=50)
plt.xlabel('$r$')
plt.ylabel('$P(r)$')
plt.title('W=%.1f'%W)
xarr=np.linspace(0,1,100)
Poi = 2/(1+xarr)**2
GOE = (27/4.)*(xarr+xarr**2)/(1+xarr+xarr**2)**(5/2.)
_=plt.plot(xarr,Poi, 'r', lw='3', label='Poisson')
_=plt.plot(xarr,GOE, 'y', lw='3', label='GOE')
_=plt.legend()
```

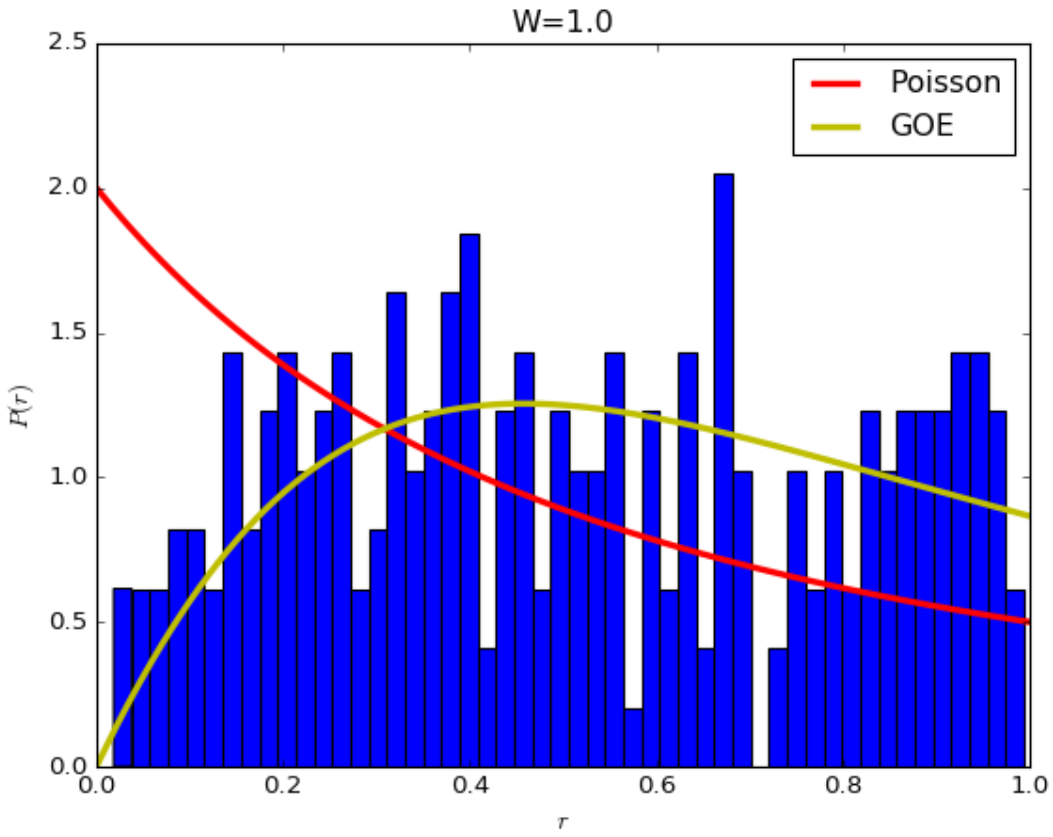


What's going on?! Why do the distributions look Poisson in both case? The system conserves S^z_{tot} and the level statistics must be computed in a single symmetry sector! Otherwise the lack of level repulsion between difference sectors of S^z_{tot} gives Poisson distributions.

In [16]:
```
L=10
s0, x,y,z = gen_s0sxsysz(L)
sztot = gen_op_total(z)
val=0
P = gen_diagprojector(sztot.diagonal(), val) ## Projects onto the sector
 with Sztot = 0
print(P.shape)
print(P.toarray())
```

```
(252, 1024)
[[ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 ...,
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]]
```

In [17]:
```
W = 1.0
Hint = 0.25*(gen_nn_int(x) + gen_nn_int(y) + gen_nn_int(z))  #0.25 is fo
r the conversion from Pauli's to spin 1/2s.
hlist = np.random.uniform(-W/2.0, W/2.0, L) #The factor of two is for th
e conversion from Pauli's to spin 1/2s.
H = Hint + gen_onsite_field(hlist, z)
evals, evecs = linalg.eigh(np.real((P*H*(P.T)).toarray()))

delta,rlist,r = LevelStatistics(evals)
_=plt.figure()
_=plt.hist(rlist, normed=True, bins=50)
plt.xlabel('$r$')
plt.ylabel('$P(r)$')
plt.title('W=%.1f'%W)
xarr=np.linspace(0,1,100)
Poi = 2/(1+xarr)**2
GOE = (27/4.)*(xarr+xarr**2)/(1+xarr+xarr**2)**(5/2.)
_=plt.plot(xarr,Poi, 'r', lw='3', label='Poisson')
_=plt.plot(xarr,GOE, 'y', lw='3', label='GOE')
_=plt.legend()
```



Great! Taking the symmetry into account makes the distribution look GOE as expected. Notice the level repulsion since the distribution goes to zero as $r \to 0$.

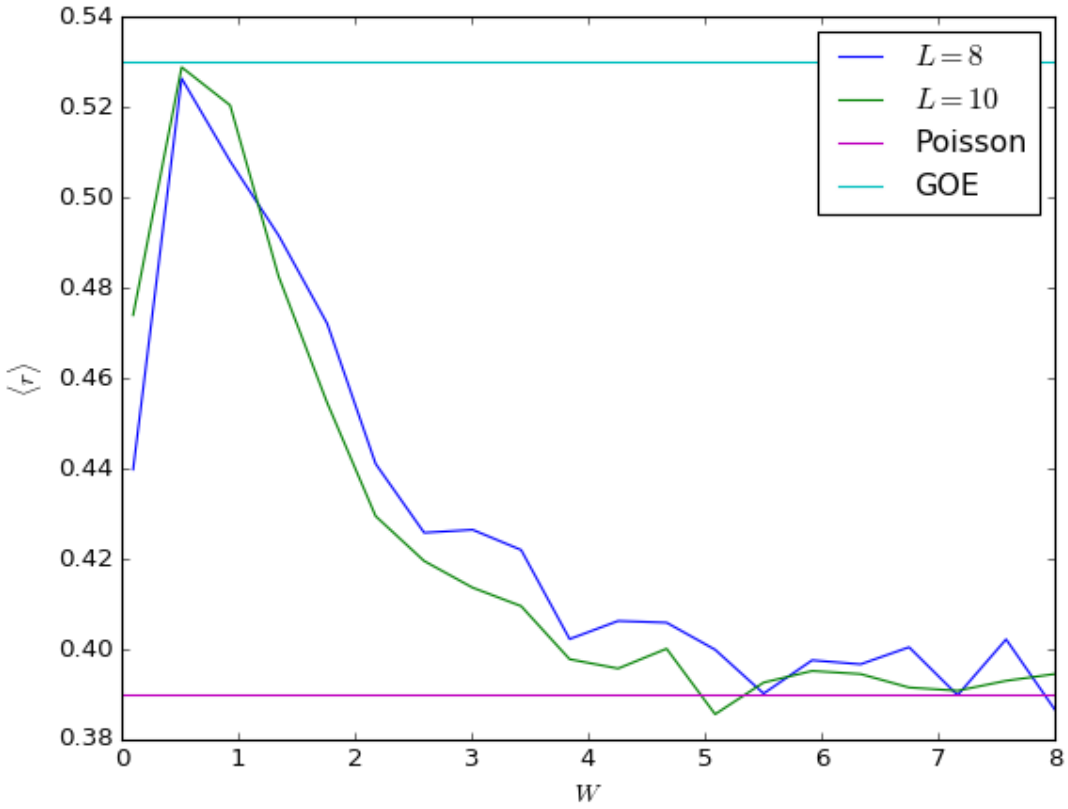
To see a crossover from the MBL phase to the thermalizing phase, it's often convenient to use a single number characterizing these distributions instead of looking at the full distributions. Note that $\langle r \rangle = 0.53$ in for the GOE distribution, and $\langle r \rangle = 0.39$ for the Poisson distribution. This is a single dimensionless number, so we should see $\langle r \rangle approach 0.53/0.39 with increasing system size in the thermal/MBL phases respectively. The location of the crossing in the curves for different system sizes thus serves as an estimate of the transition. See Figure 3 in Pal and Huse 2010 (https://arxiv.org/pdf/1010.1992.pdf (https://arxiv.org/pdf/1010.1992.pdf)).

Exercises

1. Make a plot of $\langle r \rangle$ vs. W to see the crossover in level statistics at the MBL transition
2. Can you do this at a few different system sizes to see the crossover sharpening? (don't go beyond L=10 to avoid over-burdening the cluster)
3. Can you disorder average over a few separate realizations and make the curves smoother?
4. What's happenening near W=0? Can you think of a way to "fix" this?

In [18]:
```
## Takes several minutes
Llist = [8,10]
Wnum=20
niter = 100
Wlist = np.linspace(0.1, 8, Wnum)
ravglist = np.zeros((len(Llist),niter,Wnum))

for l in range(len(Llist)):
    L=Llist[l]
    s0, x,y,z = gen_s0sxsysz(L)
    sztot = gen_op_total(z)
    val=0
    P = gen_diagprojector(sztot.diagonal(), val)
    Hint = 0.25*(gen_nn_int(x) + gen_nn_int(y) + gen_nn_int(z))  #0.25 i
s for the conversion from Pauli's to spin 1/2s.
    dim = P.shape[0]
    for n in range(niter):
        for w in range(len(Wlist)):
            W= Wlist[w]
            hlist = np.random.uniform(-W/2.0, W/2.0, L)
            H = Hint + gen_onsite_field(hlist, z)
            evals = linalg.eigvalsh(np.real((P*H*(P.T)).toarray()))
            _, _, ravglist[l,n,w] = LevelStatistics(evals[int(dim/3):int
(2*dim/3)]) #Use only middle third of spectrum
```

In [19]:
```
plt.figure()
for l in range(len(Llist)):
    plt.plot(Wlist, np.mean(ravglist[l],0), label = r'$L=%d$'%Llist[l])
_=plt.axhline(y=0.39, color='m', label=r'Poisson')
_=plt.axhline(y=0.53, color='c', label=r'GOE')
plt.legend()
_=plt.xlabel(r'$W$')
_=plt.ylabel(r'$\langle r \rangle$')
```

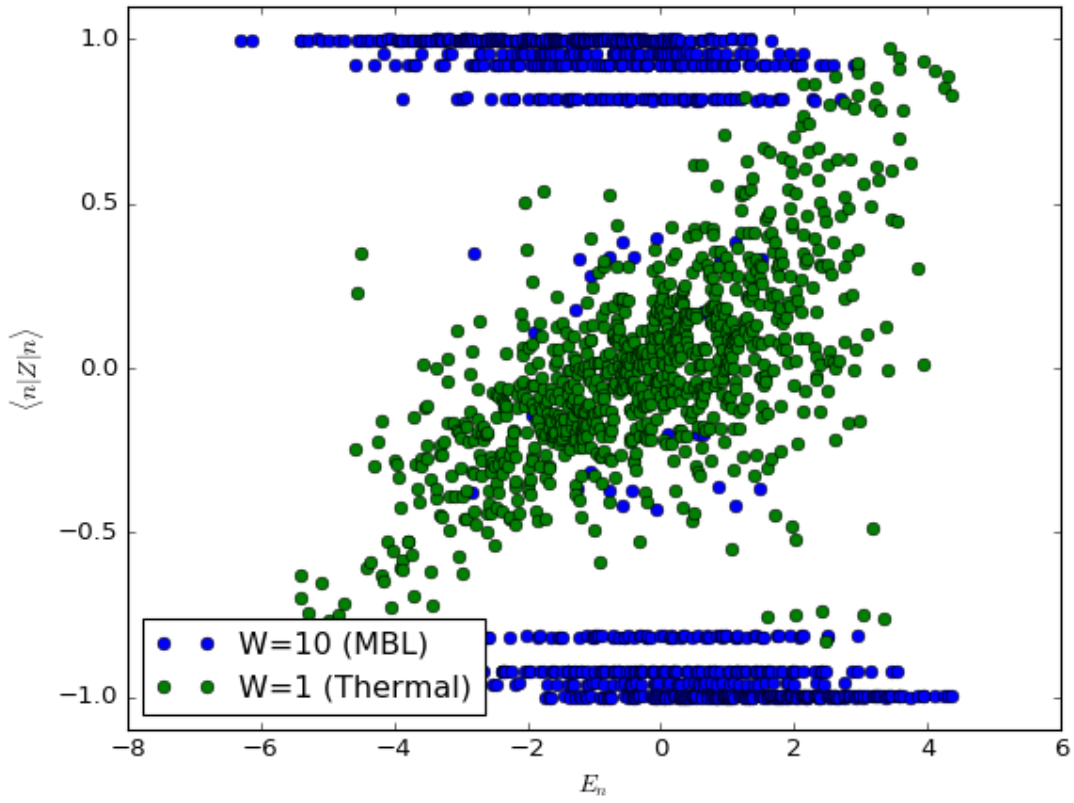


The data is too noisy and the system sizes too small to see a clear sharpening of the transition, but we do see the curves crossing near W~2.

Matrix Elements and ETH

In the MBL phase, expectation values of local observables evaluated in eigenstates fluctuate wildly from eigenstate to eigenstate at the same energy density, while these vary smoothly with energy in the thermal phase.

In [22]:
```
L=12
s0, x,y,z = gen_s0sxsysz(L)
sztot = gen_op_total(z)
val=0
P = gen_diagprojector(sztot.diagonal(), val)
Hint = 0.25*(gen_nn_int(x) + gen_nn_int(y) + gen_nn_int(z))  #0.25 is fo
r the conversion from Pauli's to spin 1/2s.


W = 10.0
hlist = np.random.uniform(-W/2.0, W/2.0, L) #The factor of two is for th
e conversion from Pauli's to spin 1/2s.
H = Hint + gen_onsite_field(hlist, z)
evals, evecs = linalg.eigh(np.real((P*H*(P.T)).toarray()))
zop = P*z[int(L/2)]*(P.T)
ZME_MBL = gen_diagonal_ME(zop.toarray(), evecs)

W = 1.0
hlist = np.random.uniform(-W/2.0, W/2.0, L) #The factor of two is for th
e conversion from Pauli's to spin 1/2s.
H = Hint + gen_onsite_field(hlist, z)
evals, evecs = linalg.eigh(np.real((P*H*(P.T)).toarray()))
zop = P*z[int(L/2)]*(P.T)
ZME_th = gen_diagonal_ME(zop.toarray(), evecs)


_=plt.figure()
_=plt.plot(evals, ZME_MBL, 'o', label = 'W=10 (MBL)')
_=plt.plot(evals, ZME_th, 'o', label = 'W=1 (Thermal)')
_=plt.legend(loc=3)
_=plt.ylabel(r'$\langle n|Z| n\rangle$')
_=plt.xlabel(r'$E_n$')
_=plt.ylim(-1.1, 1.1)
```



Exercise

How do we extend these calculations to do real-time dynamics? Can you see the decay in Imbalance starting from certain initial states (like |10101010>) as in the cold-atom experiments?