```
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
import itertools
import copy

#fuction that returns all permutations of a vector with only zeros and ones
def perm(total, ones):
    output = []
    for indices in itertools.combinations(range(total), ones):
        vector = [0] * total
        for index in indices:
            vector[index] = 1
        output.append(vector)
    return output

#function that gives the XXZ hamiltonian with the addition of a defect of magnitude epsilon on site k
def hamiltonian(basis, delta, epsilon, k):
    m = len(basis)
    n = len(basis[0])
    H = np.zeros((m,m))
    for i in range(m):
        if basis[i][k] == 1:
            H[i][i] += epsilon/2
        else:
            H[i][i] -= epsilon/2
        for j in range(n-1):
            if basis[i][j] == basis[i][j+1]:
                H[i][i] += delta/4
            else:
                H[i][i] -= delta/4
                swapped = copy.copy(basis[i])
                swapped[j], swapped[j+1] = swapped[j+1], swapped[j]
                H[i][basis.index(swapped)] += .5
    return H

def level_spacing(evals, fc, group_size): # fc = fraction of eigenvalues cut off in unfolding
    D = len(evals)
    cut = int(.5*fc*D) #number of eigenvalues cut off each end
    cut_evals = evals[cut:(D-cut)]
    spacings_size = group_size - 1 #number of spacings in each group
    Dcut = len(cut_evals)
    if Dcut % spacings_size == 0:
        groups = Dcut/spacings_size - 1 #number of groups
    else:
        groups = Dcut/spacings_size
    avgs = [(cut_evals[(i+1)*spacings_size] - cut_evals[i*spacings_size])/spacings_size for i in range(groups)]
    scaled_spacings = [(cut_evals[i+1]-cut_evals[i])/avgs[i/spacings_size] for i in range(groups*spacings_size)]
    return scaled_spacings

def wigner_dyson(s):
    return .5*np.pi*s*np.exp(-.25*np.pi*s**2)

vec_wd = np.vectorize(wigner_dyson)
```

In [2]:

```
kappa = np.empty(9)
eta = np.empty(9)

site_basis = perm(16,8)
H = hamiltonian(site_basis, .48, 0, 0)
evals = np.linalg.eigvalsh(H)
```

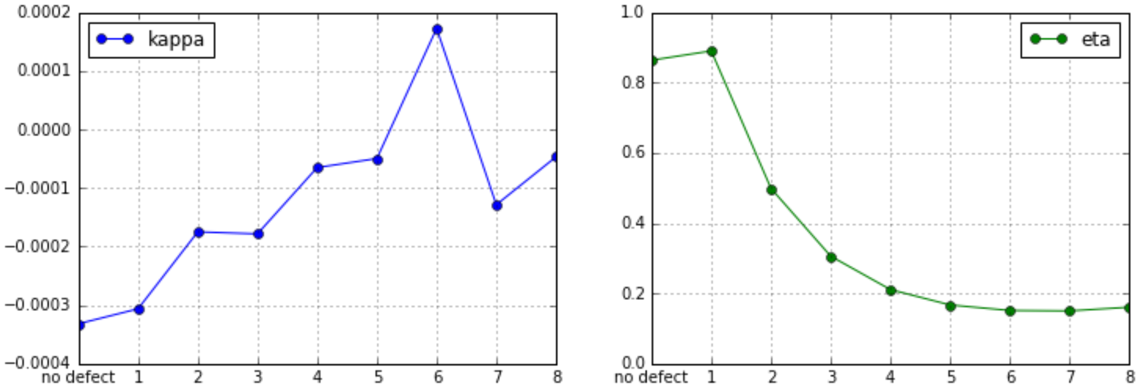
```
spacings = level_spacing(evals, .1, 11)
hist, bin_edges = np.histogram(spacings, 100, normed = True)
dbin = bin_edges[1] - bin_edges[0]
bincenters = bin_edges[:-1] + .5*dbin
wd = vec_wd(bincenters)
kappa[0] = np.sum(hist - wd)/np.sum(wd)

poisson = np.exp(-bincenters)
s0_index = np.argwhere(np.diff(np.sign(wd - poisson)) != 0)[0,0]
s0 = bincenters[s0_index]
srange = np.linspace(0, s0, 10000)
ds = srange[1] - srange[0]
Ps = np.empty(10000)
j = 1
for l in range(10000):
    while srange[l] > bin_edges[j]:
        j += 1
    Ps[l] = hist[j-1]
wd_dense = vec_wd(srange)
poisson_dense = np.exp(-srange)
eta[0] = np.sum(Ps - wd_dense)*ds/np.sum(poisson_dense - wd_dense)/ds

for i in range(8):
    print i
    H = hamiltonian(site_basis, .48, .1, i)
    evals = np.linalg.eigvalsh(H)
    spacings = level_spacing(evals, .1, 11)
    hist, bin_edges = np.histogram(spacings, 100, normed = True)
    dbin = bin_edges[1] - bin_edges[0]
    bincenters = bin_edges[:-1] + .5*dbin
    wd = vec_wd(bincenters)
    kappa[i+1] = np.sum(hist - wd)/np.sum(wd)

    poisson = np.exp(-bincenters)
    s0_index = np.argwhere(np.diff(np.sign(wd - poisson)) != 0)[0,0]
    s0 = bincenters[s0_index]
    srange = np.linspace(0, s0, 10000)
    ds = srange[1] - srange[0]
    Ps = np.empty(10000)
    j = 1
    for l in range(10000):
        while srange[l] > bin_edges[j]:
            j += 1
        Ps[l] = hist[j-1]
    wd_dense = vec_wd(srange)
    poisson_dense = np.exp(-srange)
    eta[i+1] = np.sum(Ps - wd_dense)*ds/np.sum(poisson_dense - wd_dense)/ds
```

```
0
1
2
3
4
5
6
7
```

In [29]:

```
x
```

```
fig, ax = plt.subplots(1,2, sharex = True, figsize = (12,4))

ax[0].plot(range(9), kappa, 'b-o', label = 'kappa')

ax[0].legend(loc = 0)

ax[0].grid()

ax[0].axes.set_xticklabels(['no defect'] + range(1,9))

ax[1].plot(range(9), eta, 'g-o', label = 'eta')

ax[1].set_ylim(0,1)

ax[1].legend()

ax[1].grid()

plt.show()
```



In [ ]:

In [ ]:

In [ ]:

```
x
```

In [ ]: