Electron transport in disordered environments: Anderson localization

We calculate the ways electrons move through dirty materials.

Start this notebook with "ipython notebook --pylab" (without the final 'inline').

Our Hamiltonian has hopping of strength -tHop between neighboring sites on a one-dimensional chain of atoms, and random on-site energies picked from a Gaussian normal distribution of width W and mean zero.

In []:
```
from matplotlib import animation

def Hamiltonian(tHop, W, N=100, seed = 6572):
    """Routine to calculate Hamiltonian"""
    # Set seed for random number generator
    random.seed(seed)
   # Generate random energies
    E = random.normal(scale=..., size=...)
    # Start with matrix of zeros
    Ham = zeros((N,N))
    # Fill diagonal and off diagonal entries: stop just before end
    for n in range(N-1):
        Ham[n,n] = ...
        Ham[n,n+1] = ...
        Ham[n+1,n] = ...
    # Add last diagonal entry
    Ham[N-1,N-1] = ...
    return Ham

# Test for small matrix: should be zero except along diagonal (random) and near
est neighbor off-diagonal (negative tHop)
HamTest = Hamiltonian(1.0,0.1,5)
print HamTest

Ham_t01 = Hamiltonian(1.0,0.1,100)
Ham_t1 = Hamiltonian(1.0,1.0,100)
```

Define the propagator \(K(i,j,t) = \exp(-i H t)\). Here use the routine scipy.linalg.expm. Define \(\psi_n(t) = K(1,n,t)\).

In []:
```
from scipy.linalg import expm

def K(t,Ham):
    """ Returns propagator exp(-i Ham t) """
    # Hint: Use expm from scipy.linalg
```

```
    return expm(-1.0j ...)

# Test with a small matrix
print "K = \n", K(1.0,HamTest)

#Define psi(t)
def psi(t,Ham):
    """ Return K(t,i=0,j). """
    # Hint: What row or column of K corresponds to i=0?
    return ...

# Test routine for psi
print "\n psi = \n", psi(1.0,HamTest)

# Test that sum(psi* psi) = 1
dot(...,...)

# Test by plotting real and imaginary parts of psi at t=1 for HamTest
n_index = range(len(HamTest))
pylab.plot(n_index, psi(1.0,HamTest).real, n_index, psi(1.0,HamTest).imag)
```

Animate \(\psi(t)\). Did you start the notebook without the 'inline' operator? Capture the plot at tMax = 10 after it finishes.

In []:
```
Ham = Ham_t01
tMax = 10.
dt = 0.1

N = len(Ham)
n_index = arange(N)
fig = figure()
ax = axes( xlim=(0,N-1), ylim = (-1.,1.) )
psi0 = psi(0.0,Ham)
realLine, imagLine = ax.plot(n_index, psi0.real, n_index, psi0.imag)
legend([realLine,imagLine],['Real psi', 'Imag psi'])

def animate(n_frame):
    t = n_frame * dt
    psi_t = ...
    realLine.set_data(n_index, psi_t.real)
    imagLine.set_data(n_index, ...)
    return realLine, imagLine

anim = animation.FuncAnimation(fig, animate, frames=int(tMax/dt), interval=2, r
epeat=False)
show()
```

Make a routine that calculates the current

In []:
```
def J(psi, tHop=1.):
    """ Given psi and hopping matrix element tHop, returns current
        J_n = J(n -> n+1) = (tHop / i) (psi*_n psi_(n+1) - psi_n psi*_(n+1)) ""
"
    # Hint: use slicing.
    # For debugging, return both real and imaginary parts; then return real par
t once the imaginary part is zero.
    return (...) * (conjugate(psi)[...] * psi[...] - ...).real
```

Animate the current, first for \(W=0.1\) and then for \(W=1.0\).

In []:
```
Ham = Ham_t01; tHop = 0.1
# Ham = Ham_t1; tHop = 1.0
tMax = 100.
dt = 0.1

N = len(Ham)
n_index = arange(N-1)
fig = figure()
ax = axes( xlim=(0,N-2), ylim = (-0.3,0.3) )
J0 = J(psi(0.0,Ham),tHop)
J_Line, = ax.plot(n_index, J0)

def animate(n_frame):
    t = n_frame * dt
    psi_t = ...
    J_t = ...
    J_Line.set_data(n_index, J_t)
    return J_Line

anim = animation.FuncAnimation(fig, animate, frames=int(tMax/dt), interval=2, r
epeat=False)
show()
```

Calculate eigenvalues and eigenvectors of Hamiltonian. Plot the eigenvector with lowest energy, and the one with energy closest to zero.

In []:
```
from scipy.linalg import eigh, eig
# Ham = Ham_t01
Ham = Ham_t1
vals, vecs = eigh(Ham)
figure()
# Warning: the eigenvector for eigenvalue n is vecs[:,n].
plot(..., label='Lowest energy')
# Hint: use argmin on abs(vals) to find which has the energy closest to zero
nZer = ...
plot(..., label='Closest to zero energy')
```

```
legend()
# Hint: Did you plot vecs[:,n], rather than vecs[n]?
```