9

June 30, 2016

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
% matplotlib inline
```

1 9.1 Simulated Annealing

In [8]:
```
#set initial state s randomly
N = 6
s = np.sign(np.random.rand(N)-.5)

#w arbitrary, but symmetrically and with zero diagonal
w = np.random.rand(N,N)*2-1
for i in range(N):
    w[i,i] = 0
    for j in range(i):
        w[j,i] = w[i,j]
print(s)
```

```
[-1. -1.  1.  1.  1.  1.]
```

In [3]:
```
#the cost (\energy")
def E(s, w):
    total = 0
    for i in range(len(s)):
        for j in range(len(s)):
            total = total + w[i,j]*s[i]*s[j]
    return -.5 * total

def Ei(s, w, i):
    total = 0
    for j in range(len(s)):
        total = total + w[i,j]*s[i]*s[j]
    return -.5 * total

#The probability that the network is in a state s with energy E(s) is given by
def P(s, w, b):
    #print("P =",(1.0/Z(s,w,b)),"*",np.exp(-b * E(s,w)))
    #print("Z =",Z(s,w,b),"->",(1.0/Z(s,w,b)))
    #print("E =",E(s,w))
    return (1.0/Z(s,w,b)) * np.exp(-b * E(s,w))

#the partition function Z guarantees P (s) to be a valid probability mass function
#is given as the sum over all possible configurations
```

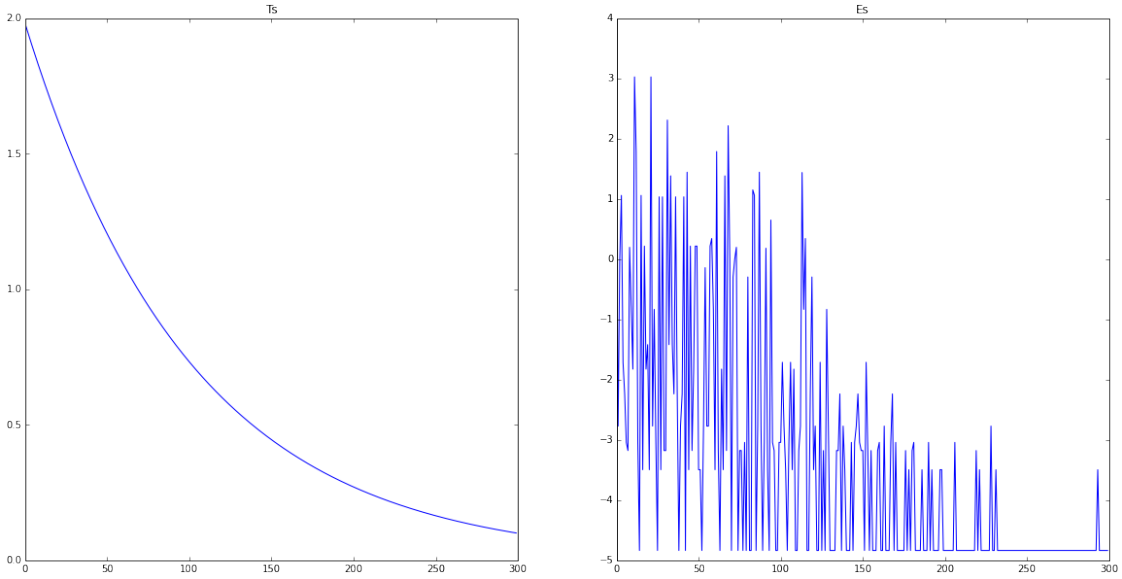
```
        def Z(s, w, b):
            total = 0
            for i in range(len(s)):
                total = total + np.exp( -b * E(s, w))
            return total
```

In [4]:
```
def SimulatedAnnealing(s, w):
    # Initialization
    mys = s.copy()
    myw = w.copy()

    #set β0,τ>1
    b0 = .5
    tau = 1.01

    #tmax
    tmax = 300

    #number of times to repeat
    #try both M=1 and M=500
    M = 500

    # Optimization
    Ts = np.zeros(tmax)
    Es = np.zeros(tmax)
    bt = b0
    #for each iteration t = 0, ..., tmax
    for t in range(tmax):

        #repeat the following M times (state update loop):
        for m in range(M):
            #select node i randomly
            i = np.random.randint(0, N)

            #determine the energies for the two options of state si and compute their differenc
            sci = mys.copy()
            sfi = mys.copy()
            sfi[i] = sfi[i] * -1
            Ec = Ei(sci, myw, i)
            Ef = -Ec
            #Ef = E(sfi, myw)
            dE = Ef - Ec

            #Change to new state with some probability
            #flip state si with probability
            Pf = 1.0 / (1 + np.exp(bt*dE))
            if np.random.rand() < Pf:
                mys[i] = mys[i] * -1
            #print(Pf)
            #print(Pf*bt)

        #increase β
        bt = tau * bt
```

```
            #For plotting
            Ts[t] = 1.0/bt
            Es[t] = E(mys, myw)
        return s, Ts, Es
    snew, Ts, Es = SimulatedAnnealing(s, w)
```

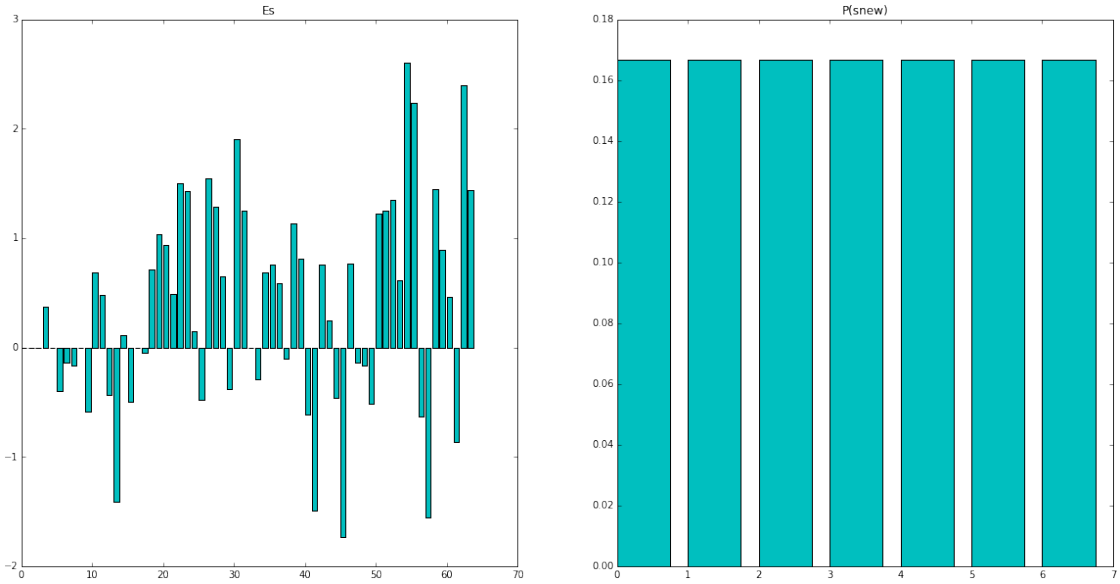
2 Plotting

```
In [5]: f, axarr = plt.subplots(1, 2, figsize=(20,10))
        axarr[0].plot(range(len(Ts)), Ts)
        axarr[0].set_title('Ts')
        axarr[1].plot(range(len(Es)), Es)
        axarr[1].set_title('Es')
        plt.show()
```




```
In [7]: def MakeAllS(curs, depth):
            if depth == 0:
                return curs
            else:
                reso = MakeAllS(curs, depth-1)
                resn = []
                for i in range(len(reso)):
                    resn.append(reso[i]+[0])
                    resn.append(reso[i]+[1])
                return resn

        allEs = [E(sc,w) for sc in MakeAllS([[0],[1]], N-1)]

        ind = np.arange(len(np.arange(len(allEs))))  # the x locations for the groups
        width = .75       # the width of the bars

        bs = [110, 11, 1.1, 1.01, 1.001, 1.0001, 1.00001]
```

```
cb = bs[4]
Pbs = [P(s, w, myb) for myb in bs]

f, axarr = plt.subplots(1, 2, figsize=(20,10))
axarr[0].bar(ind, allEs, width, color='c')
axarr[0].set_title('Es')
axarr[1].bar(np.arange(len(np.arange(len(Pbs)))), Pbs, width, color='c')
axarr[1].set_title('P(snew)')
plt.show()
```



3 9.2 Mean-Field Annealing

In [9]:
```
#set initial state s randomly
N = 6
s = np.sign(np.random.rand(N)-.5)

#w from above
```

In [10]:
```
#The cost (energy) function remains the same
def E(s, w):
    total = 0
    for i in range(len(s)):
        for j in range(len(s)):
            total = total + w[i,j]*s[i]*s[j]
    return -.5 * total

#The approximated probability of a state s is now given by
def Q(s, e, b):
    total=0
    for j in range(len(s)):
        total += e[j]*s[j]
    return (1.0/Z(s,w,b)) * np.exp(-b * total)
```

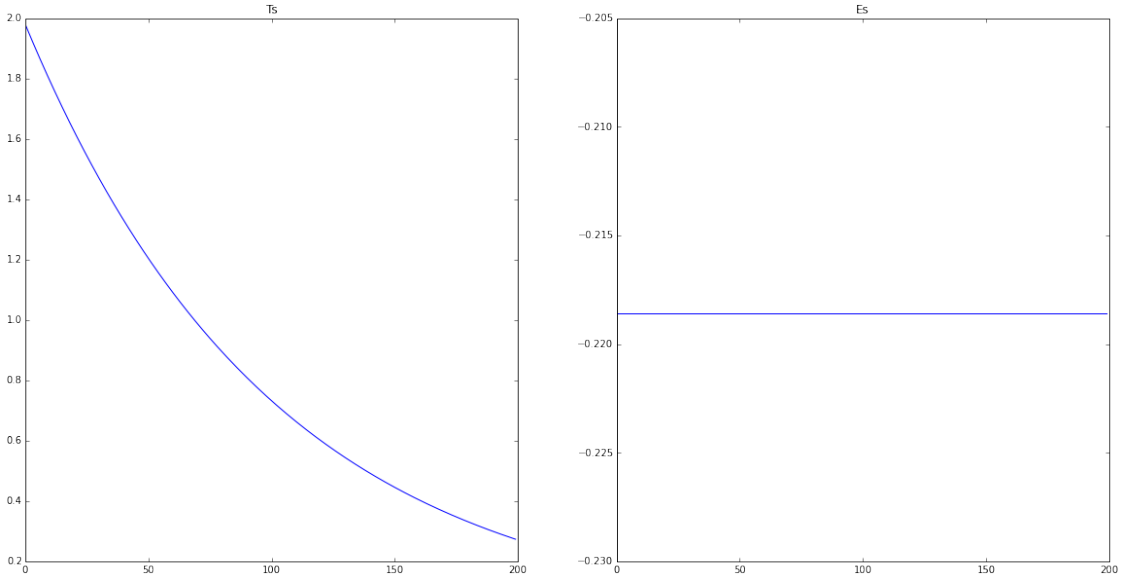
```
        #the partition function Z guarantees P (s) to be a valid probability mass function
        #is given as the sum over all possible configurations
        def Z(s, w, b):
            total = 0
            for i in range(len(s)):
                total = total + np.exp( -b * E(s, w))
            return total

        def e(s, w):
            mye = np.zeros(len(s))
            for i in range(len(s)):
                mye[i] = ei(s, w, i)
            return mye

        def ei(s, w, i):
            myei = 0
            for j in range(len(s)):
                myei = myei + w[i,j]*s[j]
            return -1 * myei
```

In [11]:
```
        def MeanFieldAnnealing(s, w):
            # Initialization
            mys = s.copy()
            myw = w.copy()

            #β0 small enough, τ > 1
            b0 = .5
            tau = 1.01

            #set tmax, ε small enough
            tmax = 200
            epsilon = .00001

            #number of times to repeat
            #try both M=1 and M=500
            M = 500

            # Optimization
            Ts = np.zeros(tmax)
            Es = np.zeros(tmax)
            bt = b0
            eold = np.zeros(len(s))
            enew = eold
            iteration = 0
            maxiterations = 500

            #for each iteration t = 0, ..., tmax
            for t in range(tmax):
                eold = enew
                enew = e(s,w)

                #repeat the following until convergence, i.e., |enew − eold| < ε
```

```
        while np.absolute( np.min(enew - eold) ) > epsilon and iteration < maxiterations:
            #for i=1,...,N
            for i in range(N):
                #compute mean-field
                enew[i] = ei(s, w, i)

                #update the state
                s[i] = np.tanh(-bt * enew[i])

            iteration = iteration + 1

        #if iteration < maxiterations:
        #    print("converged at", iteration)

        #increase β
        bt = tau * bt

        #For plotting
        Ts[t] = 1.0/bt
        Es[t] = E(mys, myw)
    return s, Ts, Es
snew, Ts, Es = MeanFieldAnnealing(s, w)
```

4 Plotting

In [12]:
```
f, axarr = plt.subplots(1, 2, figsize=(20,10))
axarr[0].plot(range(len(Ts)), Ts)
axarr[0].set_title('Ts')
axarr[1].plot(range(len(Es)), Es)
axarr[1].set_title('Es')
plt.show()
```



In [ ]: