20.320 Problem Set 3 Solutions
September 25, 2009

#

General Instructions:

1. You are expected to state all your assumptions and provide step-by-step solutions to the numerical problems. Unless indicated otherwise, the computational problems may be solved using Python/MATLAB or hand-solved showing all calculations. Both the results of any calculations and the corresponding code must be printed and attached to the solutions. **For ease of grading (and in order to receive partial credit), your code must be well organized and thoroughly commented, with meaningful variable names.**

2. You will need to submit the solutions to each problem to a separate mail box, so please prepare your answers appropriately. Staples the pages for each question separately and make sure your name appears on each set of pages. (The problems will get sent to different graders, which should allow us to get the graded problem set back to you more quickly.)

3. Submit your completed problem set to the marked box mounted on the wall of the fourth floor hallway between buildings 8 and 16.

4. The problem sets are due at noon on Friday, October 2nd. There will be no extensions of deadlines for any problem sets in 20.320. Late submissions will not be accepted.

5. Please review the information about acceptable forms of collaboration, which was provided on the first day of class and follow the guidelines carefully.

20.320 Problem Set 3
Question 3

#

In class we discussed the role of GCSF in binding both bone marrow precursors and neutrophils. The interaction of interest is between the GCSF and the GCSF receptor (GR). Given this table from *Layton et al. JBC 274:25 pp.17445-17451* answer the following questions.

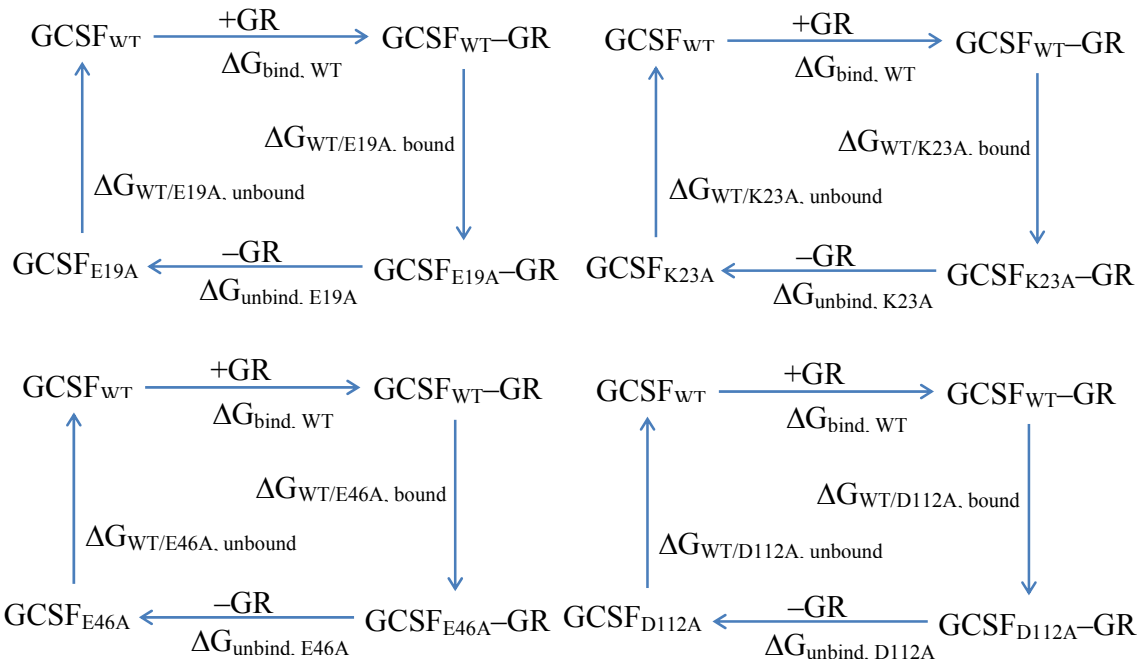
TABLE II
Binding of G-CSF mutants to Ba/F3 cells expressing WT-GR or (R288A)GR

G-CSF mutant	Receptor			
	WT-GR		(R288A)GR	
	K_d nM[a]	Mut/WT[b]	K_d nM[a]	Mut/WT[b]
WT	0.045 ± 0.008	1.0	0.37 ± 0.03[c]	1.0
E19A	0.050 ± 0.004	1.1	0.29 ± 0.03	0.78
K23A	0.077 ± 0.015	1.7	0.95 ± 0.11	2.5
E46A	0.076 ± 0.003	1.7	3.32 ± 0.86[c]	8.9
D112A	0.060 ± 0.003	1.3	4.06 ± 0.85	10.9

[a] Data are mean ± range of two assays, including data shown in Fig. 4.
[b] Ratio of K_d for mutant G-CSF/WT G-CSF.
[c] Data are mean ± S.D. of three assays, including data shown in Fig. 4.

This question will focus mainly on GCSF variants with wild-type GCSF receptor.

a) Draw out the four thermodynamic cycles for different GCSF mutants binding to the wild-type GR. Be sure to label the ligand and receptors along with each ΔG correctly.



#

b) Given 100 pM concentration of WT-GR, calculate the fractional site saturation assuming an excess of ligand for wild-type GCSF and each GCSF mutant.

Given a system where ligand concentration is in excess, we can ignore ligand depletion and use the formula discussed in lecture. For wild-type GCSF:

$$y_{\mathrm{WT}} = \frac{[L_0]}{[L_0] + K_d} = \frac{0.1\ \mathrm{nM}}{0.1\ \mathrm{nM} + 0.045\ \mathrm{nM}} = 0.69$$

For the other GCSF mutants, the only parameter that changes is the K_d for GCSF-GR binding. We can repeat this calculation for each GCSF mutant and obtain the following fractional saturation values.

$$y_{\mathrm{E19A}} = 0.67$$
$$y_{\mathrm{K23A}} = 0.56$$
$$y_{\mathrm{E46A}} = 0.57$$
$$y_{\mathrm{D112A}} = 0.63$$

c) Compute the ΔΔGs between all mutants (6 total ΔΔGs) at normal body conditions (37°C and 1atm pressure).

ΔΔG represents the difference in binding energies when comparing two different mutants of a ligand or a receptor. To compute a ΔΔG, we simply compute each individual ΔG and subtract them. Recall that $\Delta G = RT \ln K_d$.

To calculate the ΔΔG comparing the free energies of binding GR to the E19A and K23A mutants of GCSF:

$$\Delta\Delta G^\circ_{\mathrm{K23A-E19A}} = \Delta G^\circ_{\mathrm{GR-K23A}} - \Delta G^\circ_{\mathrm{GR-E19A}}$$
$$= -(0.00199\ \mathrm{kcal/mol\text{-}K})(310\ \mathrm{K})\left[\ln(0.077\times10^{-9}\ \mathrm{M}) - \ln(0.050\times10^{-9}\ \mathrm{M})\right]$$
$$= -0.266\ \mathrm{kcal/mol}$$

Similarly, for comparing other pairs of mutant GCSF:

$$\Delta\Delta G^\circ_{\mathrm{E46A-E19A}} = -0.258\ \mathrm{kcal/mol}$$
$$\Delta\Delta G^\circ_{\mathrm{D112A-E19A}} = -0.112\ \mathrm{kcal/mol}$$
$$\Delta\Delta G^\circ_{\mathrm{E46A-K23A}} = 0.00806\ \mathrm{kcal/mol}$$
$$\Delta\Delta G^\circ_{\mathrm{D112A-K23A}} = 0.154\ \mathrm{kcal/mol}$$
$$\Delta\Delta G^\circ_{\mathrm{D112A-E46A}} = 0.146\ \mathrm{kcal/mol}$$

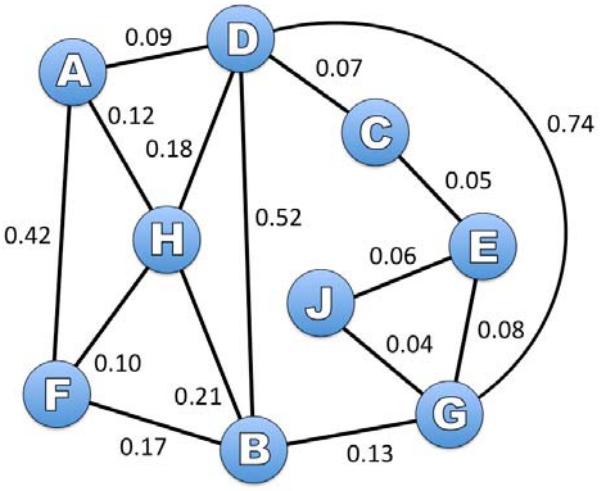
d) Suppose we wanted to look at WT and E46A GCSF variants with WT-GR and R288A-GR. Draw out the double-mutant cycle. Be sure to label the ligand and receptors along with the ΔGs/ ΔΔGs correctly. (note: you can draw it as a cube, or simplify it, but it must contain all of the components)

TIFFANY

20.320 Problem Set 3
Question 3

#

In lecture, we discussed the use of graphs for representing biomolecular interactions. In a protein-protein interaction graph the 'nodes' represent proteins. Two nodes are connected by an edge if there is evidence that the two proteins interact. Consider the protein-protein interaction (PPI) network graph shown below. Protein nodes are colored blue and denoted by a one-letter character. Each edge is associated with a probability of interaction as shown.



To answer the questions below, you will need the NetworkX Python package, which is installed on Athena. If you wish to try this on your personal computer, the NetworkX zip file can be downloaded from *http://pypi.python.org/pypi/networkx/*. The folder networkx contained in the zip file will need to be placed in the same folder as the starter code provided.

a) As you can observe from the PPI network graph provided, there may be multiple 'paths' to reach any node from any other node of the network. Assume that each path represents a potential signal transduction pathway. The "length" of a path between two nodes is defined as the sum of the edge weights along that path. If the edge weights are set to the negative log of the probabilities ($w_{ij} = -\log_{10} p_{ij}$) show that the shortest path will be the one with the maximum joint probability.

The distance along a path (i.e. the path length) is defined as the sum of edge weights along that path. From the problem statement, we know that an edge weight is defined as the negative log of the probability of moving from one node to another. In other words:

$$\text{Path length} = \sum_{\text{edge}_{ij} \in \text{ path}} -\log_{10}(P_{ij}),$$ and by the rules of logs: $$\text{Path length} = -\log_{10}\left[\prod_{\text{edge}_{ij} \in \text{ path}} (P_{ij})\right]$$

Since $-\log(n) = \log\left(\frac{1}{n}\right)$, a higher joint probability will result in a smaller path length.

b) Write a small piece of python code to accept the given graph as input and calculate the shortest path matrix (SPM) as output. The SPM is a matrix-representation format that is convenient to analyze PPI networks. Each element of the SPM represents the joint probability of the set of protein-protein interactions along the shortest path between two nodes as in the following equation:

#

$$\mathrm{SPM}(i,j) = \max_{P \in \text{ paths from } i \text{ to } j} \left(\Pi_{e_{k \in P}} prob_k \right)$$

where the product is over all edges *k* in *P*, a path from *i* to *j* and $prob_k$ is the probability of edge *k*. You may assume that proteins of this network do not self-interact – that is no protein has any interaction with itself (0 probability). The website contains some code to help you get started with this problem called `networkstarter.py` You will have to submit your fully commented python code to receive credit for this problem.

(Hint: The `NetworkX` module contains a function for computing the shortest distance between two nodes. Run `networkx.dijkstra_path_length(graph,start,stop)` to compute the sum of the edge weights along the shortest path between nodes `start` and `stop`. Remember to use weights equal to $-\log_{10} p_{ij}$ as defined in part (a).

```
## Solution file for Question 2 of Problem Set 3
## September 20, 2009
## 20.320

from networkx import *
from math import *

##Create the graph bionet shown in problem 2 with each number
##representing the -log10(probability of interactions)

bionet=Graph()
bionet.add_nodes_from(["A","B","C","D","E","F","G","H","J"])
bionet.add_edge("A","D",weight=(-log10(0.09)))
bionet.add_edge("A","F",weight=(-log10(0.42)))
bionet.add_edge("A","H",weight=(-log10(0.12)))
bionet.add_edge("B","H",weight=(-log10(0.21)))
bionet.add_edge("B","F",weight=(-log10(0.17)))
bionet.add_edge("B","D",weight=(-log10(0.52)))
bionet.add_edge("B","G",weight=(-log10(0.13)))
bionet.add_edge("C","D",weight=(-log10(0.07)))
bionet.add_edge("C","E",weight=(-log10(0.05)))
bionet.add_edge("D","G",weight=(-log10(0.74)))
bionet.add_edge("D","H",weight=(-log10(0.18)))
bionet.add_edge("E","J",weight=(-log10(0.06)))
bionet.add_edge("E","G",weight=(-log10(0.08)))
bionet.add_edge("F","H",weight=(-log10(0.10)))
bionet.add_edge("G","J",weight=(-log10(0.04)))

##You must now use the above graph to generate the SPM for all protein
##interactions. Your code must be commented and contain meaningful variable
##names for it to be graded

SPM = []  ##Initializes SPM matrix

##Goes through each protein and calculate the distance to all other proteins
for row in bionet:
    row_temp = [] ## Initializes list that will serve as a row in the SPM
    row_temp.append(" "+row+" ") ##Adds title as first entry in each row
    for column in bionet:
```

```
#=============================================================================
        SP=dijkstra_path_length(bionet,row,column) ##Determine shortest path
        prob=str(round(10**(-1*SP),2))##Converts shortest path to probability
        if prob=="1.0":  ##Sets probability of self interactions to 0
            prob="0  "
        if len(prob)==3: ##Makes all cells in final table the same width
            prob=prob+" "
        row_temp.append(prob) ##Adds probability value to current row in
                              table
    SPM.append(row_temp) ##Adds row to final matrix

##Prints a title for each column of the matrix
column_headers=["   "]
for row in bionet:
    column_headers.append("  "+row+" ")
print column_headers

##Print each row of the SPM
for row in range(len(SPM)):
    print SPM[row]
```

c) Given that there are indeed multiple paths between most pairs of nodes, do you suppose that the SPM is the best indicator of protein-protein interactions in biological networks? Why or why not?

Recall from Part A that the shortest path between two nodes in a network is one of maximum joint probability. Calculating joint probability involves a series of “AND” statements (i.e. to go from A to B to C: A interacting with B AND B interacting with C) where individual probabilities are multiplied together. This leaves the joint probability calculation quite sensitive to experimental variation. Not only are biological networks difficult to probe experimentally, they are also dynamic, meaning that measurements collected reflect the state of the network under a specific set of conditions. This means that the SPM for a particular network is very sensitive to variations in the data and the experimental conditions used.

d) Another approach to analysis of PPI networks is to consider the multiple interaction paths between proteins of the network. Consider nodes A and G of the PPI network shown above and answer the following questions:

   i. List all the different paths (along with the corresponding probabilities of interaction) between nodes A and G, assuming that no path can be more than 6 nodes long, i.e. if the path includes more than 6 nodes you do not need to account for that particular path. You may represent the paths between nodes as a chain of letters the order of which gives the order of progression through the nodes (i.e. ACG means A to C to G). Further assume that no path can include any node more than once, i.e. that is paths such as ABFCADG are unacceptable since A occurs more than once.

#

Path	Joint Probability
ADG	0.0666
ADHFBG	0.000035802
ADHBG	0.00044226
ADBG	0.006084
ADCEG	0.0000252
ADCEJG	0.000000756
AFBG	0.009282
AFBDG	0.02747472
AFHBG	0.0011466
AFHDG	0.0055944
AFBHDG	0.001997201
AFHDBG	0.000511056
AFHBDG	0.003393936
AHBG	0.003276
AHBDG	0.00969696
AHDG	0.015984
AHDBG	0.00146016
AHDCEG	0.000006048
AHFBG	0.0002652
AHFBDG	0.000784992

ii. Write an equation for the effective probability that at least one active protein-protein interaction path exists between two nodes in the network in terms of the probability of the edges. Note that prob(at least one active path exists) = 1 - prob(none of the paths are active). You may assume that each path is independent of other paths in the network.

Prob(at least one active path exists) = 1-prob(none of the paths are active)
Prob(path *n* is inactive) = 1 – prob(path *n* is active)
Prob(none of the paths are active) = Prob(path *n* is inactive) * Prob(path *m* is inactive) * …
Therefore Prob(none of the paths are active) = [1 – prob(*n*)]*[1 – prob(*m*)]*…
Therefore Prob(at least one active path exists) =
1 – ([1 – prob(*n*)]*[1 – prob(*m*)]*…)

#

iii. Apply the equation you derived above to calculate the probability that at least one active path exists between nodes A and G of our network.

Path	Joint Probability	1-[Joint Probability]
ADG	0.0666	0.9334
ADHFBG	0.000035802	0.999964198
ADHBG	0.00044226	0.99955774
ADBG	0.006084	0.993916
ADCEG	0.0000252	0.9999748
ADCEJG	0.000000756	0.999999244
AFBG	0.009282	0.990718
AFBDG	0.02747472	0.97252528
AFHBG	0.0011466	0.9988534
AFHDG	0.0055944	0.9944056
AFBHDG	0.001997201	0.998002799
AFHDBG	0.000511056	0.999488944
AFHBDG	0.003393936	0.996606064
AHBG	0.003276	0.996724
AHBDG	0.00969696	0.99030304
AHDG	0.015984	0.984016
AHDBG	0.00146016	0.99853984
AHDCEG	0.000006048	0.999993952
AHFBG	0.0002652	0.9997348
AHFBDG	0.000784992	0.999215008

Therefore, the probability that at least one path is active is: 1 – (0.9334 x 0.999964198 x 0.99955774 x …), or 0.145.

#

e) Let us now consider a medical application to this biological network analysis problem. Suppose that the interaction between protein nodes A and G of the provided PPI network is key to the progression of breast cancer. Identify a single node that, when inactivated, would maximally reduce the probability of any connection between A and G. For this problem, use the probability equations from question 2d and assume that when a protein is deactivated, the corresponding node and all its interaction edges are removed from the provided PPI network.

Since removing a node means that all edges involving that node are removed from the network, we can calculate a new probability of interaction. We first remove the node and its associated edges from the network and then repeat the final calculation from Part d) with the 1-[Joint Probability] values of the remaining edges.

Node Removed	Probability
B	0.086687044
C	0.145299998
D	0.013921405
E	0.145299998
F	0.10066385
H	0.106165407
J	0.145326705

Removing Node D causes roughly a 10-fold decrease in connections between A and G, and may be the most promising target for therapeutic intervention.

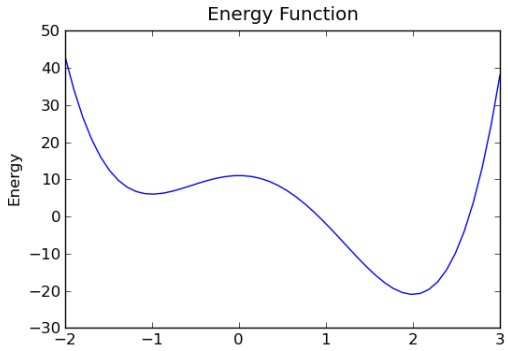
20.320 Problem Set 3
Question 3

#

In class, we discussed the Metropolis algorithm, which uses the Boltzmann distribution to sample states in order to find an absolute energy minimum. For this question, we will use the principles of the metropolis algorithm to find the absolute minimum of a polynomial function. To start, download the python file PolyEnergetics.py from the Course website.

The function `poly_energy()` in PolyEnergetics.py takes a single number as input and returns the value of the function $3x^4-4x^3-12x^2+11$ for that number.

a) Plot this polynomial function for *x* between -2 and 3. What are the minima of this function?




```
The minima of this function are:
6.00001802001  and  -20.9999639479
```

b) What minimum would you find if you implemented a gradient descent search starting at *x* = -2? What is the drawback in using gradient descent searches for energy minimization?

Implementing a gradient descent search starting at *x* = -2 would only yield the first minimum, which is about y = 6. Since gradient descent searches only accept new values of *x* when moving results in a decrease in *y*, they can only find the first minimum they come to. This way, they cannot discriminate between a local minimum and a global minimum.

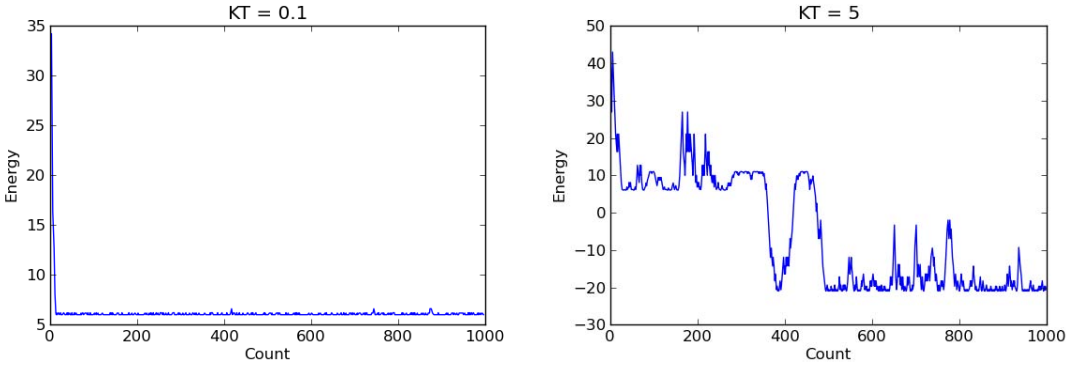
c) Complete the code `run_metropolis()` in PolyEnergetics.py to implement metropolis algorithm criteria to sample states (an x-value and it's `poly_energy` value) with the following specifications:

- The only input should be a float corresponding to a value of KT.
- The search should start at an x-value of -2.0
- Select the next *x*-value to test by generating a random number between 0 and 1. If the number is less than 0.5, decrease x by 0.1. Otherwise, increase x by 0.1.
- Decide whether or not to move to the test state based on the Metropolis criteria discussed in class.
- The function should run for 1000 cycles.
- The function should return a list of the energy values at the end of each cycle through the algorithm (i.e. 1000 entries per run).

#

See attached code.

d) Plot the list of energy values encountered during a search vs. cycle number at a KT of 0.1 and 5. Compare the behavior of the algorithm for these two KT values.



With higher values of KT, the algorithm accepts unfavorable increases in energy more often. Since the probability of accepting a higher energy state is exp(-ΔE/KT), whether the algorithm accepts a higher energy state depends on the energy difference and on KT. For a given ΔE, as KT increases, the probability of accepting a higher energy state increases, explaining the spikes seen as the algorithm searches for a minimum in the KT = 5 plot. With a low value of KT, the probability of accepting the higher state is close to 0, explaining the quick decent to the first local minimum the algorithm finds.

e) Run the `run_metropolis()` function 1000 times at a KT of 0.7, 2.0, and 5.0. How often does the function get within 0.1 of the global minimum at some point during the search? Explain your results.

The program outputs the following results, with a KT = 5 reaching the approximate global minimum the most often. With higher values of KT, the metropolis algorithm is more likely to accept increases in energy that come with moving *x* values, therefore enabling the algorithm to transit across the energy barrier between minima. Physically, this corresponds to a protein folding at a higher temperature. Higher temperatures mean that the protein has more energy as it folds, resulting in more conformational states being accessible and increasing the likelihood that the protein could cross small barriers in the folding funnel to reach the thermodynamically favored conformation.

```
KT = 0.7 ; # of Successes = 19
KT = 2 ; # of Successes = 494
KT = 5 ; # of Successes = 731
```

```
##############################################################################
## 20.320 Problem Set 3
## Question 3 Solution

## Jim Abshire
## September 22, 2009

import pylab
import time
import numpy as np
import PolyEnergetics

## Part A: Plot energy function and find minima
x = np.linspace(-2, 3, 1000) ## Create array of x values from x = -2 to +3
y = np.array([])
for i in x :        ## Calculate the y values corresponding to each x value
    y = np.append(y, PolyEnergetics.poly_energy(i))
pylab.plot(x, y)
pylab.title("Energy Function")
pylab.ylabel("Energy")
min1 = min(y[0:len(y)/2])   # Find first local minimum
min2 = min(y[len(y)/2:len(y)]) # Find second local minimum
print "The minima of this function are: "
print min1, " and ", min2

globalmin = min(y)  # Store global minimum for future reference


## Part D: Plot states accepted over 1000 iterations for KT = 0.1 and KT = 5
for KT in [0.1, 5] :
    energies = PolyEnergetics.run_metropolis(KT)
    x = range(0, len(energies))
    pylab.figure()
    if KT == 0.1 :
        pylab.title("KT = 0.1")
    else : pylab.title("KT = 5")
    pylab.xlabel("Count")
    pylab.ylabel("Energy")
    pylab.plot(x, energies)

## Part E: Compute # of times algorithm reaches minimum within 0.1 of global
##         minimum for KT = 0.7, KT = 2, and KT = 5
for KT in [0.7, 2, 5] :
    count = 0
    for repeats in range(0, 1000) :     # Run algorithm 1000 times
        energies = PolyEnergetics.run_metropolis(KT)
        if (min(energies) < (globalmin + 0.1)) :    # Increment counter if within 0.1
of global minimum
            count += 1
    print "KT =", KT, "; # of Successes =", count

pylab.show()
```

```
#=============================================================================#
def run_metropolis(KT):
    import numpy as np
    energies_encountered = []
    cycles = range(0, 1000)
    S = -2.0                # Starting x value
    E = poly_energy(S)      # Starting y value
    for i in cycles :       # Iterate 1000 times
        stateshifter = np.random.rand()     # Generate random number: 0-1
        if stateshifter < 0.5 :             # Half the time, decrease S by 0.1
            S_test = S - 0.1
        else : S_test = S + 0.1             # Other half the time, increase S by 0.1
        E_test = poly_energy(S_test)        # See what happens to the energy at this
                                              new S value
        if E_test < E :                     #   If this move decreased energy, accept
                                              move right away
            E = E_test
            S = S_test
            energies_encountered = np.append(energies_encountered, E)
        else :                              #   If the energy increased after the
                                              move:
            P_boltz = np.exp(-(E_test - E)/KT)  # Generate probability of accepting
                                                  based on a Boltzmann distribution
            stateshifter = np.random.rand()     # If a random number is between 0 and
                                                  the probability calculated above:
            if stateshifter < P_boltz :         #   accept the new state
                E = E_test
                S = S_test
            energies_encountered = np.append(energies_encountered, E)

    return energies_encountered
```

20.320 Analysis of Biomolecular and Cellular Systems
Fall 2012