IMMUNE TARGETING AND VIRAL FITNESS LANDSCAPES

BEN FULAN

1. Introduction

This project arose out of the search for a vaccine to combat HIV and the obstacles faced by this search. While it is possible to create a vaccine that induces an immune response against the virus, the effectiveness of this approach is limited by the ability of the virus to mutate into strains which the immune response would no longer recognize. The goal of this project is to produce an immune targeting scheme that can do the most damage to the virus given its ability to mutate.

To this end, a measure of the fitness of the virus is needed. This takes the form of a fitness landscape, which gives the virus's ability to replicate as a function of its amino acid sequence. In particular, this amino acid sequence can be described using a binary string, where '0' represents the natural ("wild type") form of the amino acid and '1' represents any of 19 possible mutants. This approach loses the ability to distinguish between different mutants but greatly reduces computational complexity. The fitness function can then be constructed from empirical data, as in [1]. This function assigns a fitness cost to each mutation and also contains terms to represent interactions between pairs of mutations, which can be either beneficial or harmful. For simplicity, higher-order interactions are not considered. Finally, we add a term to represent immune pressure, effectively forcing a mutation by imposing a high fitness cost to remaining in the wild type state. Thus the fitness function takes the form

$$E(\mathbf{z},\mathbf{b}) = \sum_{i=1}^{N} h_i z_i + \sum_{i=1}^{N}\sum_{j=i+1}^{N} J_{ij} z_i z_j + \sum_{i=1}^{N} b_i(1-z_i) \tag{1.1}$$

where $\mathbf{z}$ is the binary vector of length N representing the amino acid sequence, h_i is a positive real number that gives the fitness cost associated with a mutation at position i, J_{ij} is a real number that gives the additional cost (if positive) or benefit (if negative) associated with a pair of mutations at positions i and j, and $\mathbf{b}$ is the vector representing immune pressure, where b_i is taken to be $+\infty$ if pressure is applied at position i and 0 otherwise. Note that $E(\mathbf{z},\mathbf{b})$ is actually an energy function, where low energy corresponds to high fitness and vice versa.

2. Optimization Problem Statement

This energy function gives a measure of the viral fitness given a particular amino acid sequence and a particular set of positions targeted by immune pressure. Our goal is to create an algorithm that will determine the "best" set of positions to target given empirical data for $\mathbf{h}$ and $\mathbf{J}$. To do this, we need a function that measures the overall fitness of the virus across all possible amino acid sequences given a set of targeted positions. Such a function should give more weight to low-energy states given that the virus is more likely to be found there. One possibility is the partition function $Z(\mathbf{b}) = \sum_{\mathbf{z}} e^{-\beta E(\mathbf{z},\mathbf{b})}$, where β is a (positive real) parameter corresponding to the difficulty of mutation. In this paper, we will focus on the function $M(\mathbf{b}) = \min_{\mathbf{z}} E(\mathbf{z},\mathbf{b})$, which is simply the value of the lowest-energy state. This is a good function to choose because the virus tends to be found in the lowest-energy state it can attain. It is also much more computationally tractable than $Z(\mathbf{b})$, which requires computing each of the 2^N possible $\mathbf{z}$ states. In contrast, the lowest-energy amino acid sequence can be determined without any computation based on that fact that mutations always have a positive energy cost; this state is the one with mutations occurring at precisely

Date: September 21, 2014.

the positions where immune pressure is applied. So $M(\mathbf{b})$ is quite easy to compute given $\mathbf{b}$; the best set of positions to target is the set corresponding to the $\mathbf{b}$ vector which gives the largest value for $M(\mathbf{b})$.

3. Optimization Algorithm: Positions

In the above, it is assumed that only a limited number of positions can be targeted due to immune system constraints. We call this number t and assume t is known. The simplest approach to find the largest value of $M(\mathbf{b})$ is an exhaustive search of all possible $\mathbf{b}$ vectors with exactly t nonzero (i.e. 1) entries, of which there are $\binom{N}{t}$. This approach is well-suited to smaller values of N and t but quickly slows down as N and t increase. In these cases, a genetic algorithm was employed for faster computation. This algorithm works by computing the energies of all possible pairs of positions (i.e. treating t as 2), which is fast even if N is large, and producing a list of the k highest-energy pairs for a specified value of k. It then creates a list of all positional triples that contain a pair in the previous list and computes the k highest-energy triples in this list. This process is repeated until the computed tuples have t elements. This algorithm provides a significant increase in speed, and for sufficiently large values of k ($k > 20$) it virtually always arrives at the same solution as the exhaustive algorithm.

The only major drawback to this algorithm is that it does not accurately represent the capabilities of the immune system. In particular, it is not possible for the immune system to target specific positions in the virus's amino acid sequence as is assumed here. Instead, the immune system can only target epitopes, which are groups of 8-11 consecutive positions. Targeting an epitope forces the virus to mutate at any one of the positions in the epitope, but the virus, not the immune system, chooses which one. The next step is modify the algorithm to reflect this behavior.

4. Optimization Algorithm: Epitopes

In the epitope case, an exhaustive search of all possible solutions becomes more complicated. First, the lengths of each epitope must be specified; alternatively they can be allowed to vary from 8 to 11 for more generality. An epitope is determined by its length and starting position, so once the lengths are given, enumerating all possible t-tuples of epitopes is similar to enumerating all possible t-tuples of positions above. Finally, given a set of epitopes, the energy of each combination of positions within these epitopes is computed; the lowest-energy combination is the one the virus will choose. It is worth noting that if two or more epitopes overlap, a mutation in a position in the overlap allows an escape in each of the epitopes, so the total number of mutations forced may be less than t.

The exhaustive approach described here is much less effective than in the position case due to the increased number of computations required. The number of possible t-tuples of epitopes is slightly less than $\binom{N}{t}$ (since the last few positions in the amino acid sequence are not valid starting positions), but for each epitope tuple, the number of possible position combinations is on the order of 10^t, which severely slows down the algorithm as t increases. The genetic algorithm outlined above is somewhat effective at counteracting this. In this case, the genetic algorithm determines the k highest-energy pairs of epitopes, forms all possible triples of epitopes that contain one of these pairs, finds the k highest-energy such triples, and continues this process until reaching t-tuples. This approach is rather less effective than in the position case since the energy of the possible position combinations within a given tuple of epitopes must still be computed. However, it does provide a significant speed increase over the exhaustive search algorithm.

5. Results

With a variety of algorithms at our disposal, the next step was to compare the results with empirical data. In this paper, we concentrate on the HIV proteins p6 and especially p24 due to ease of computation (for p6) and availability of data (for p24). Note that the amino acid sequence for p6 contains 53 positions, whereas p6 has 231. First, we attempt to establish the consistency of our algorithm with observed behavior regarding the relative fitness of different epitopes. We do so by comparing the results of our algorithm with [1], which uses a Monte Carlo procedure to rank the fitness of 121 known epitopes in p24. The results are shown in Figure 1; note that the algorithm can produce ties if different epitopes have the same lowest-energy position. The Spearman's rank correlation coefficient between the ranks given by the algorithm and the Monte Carlo

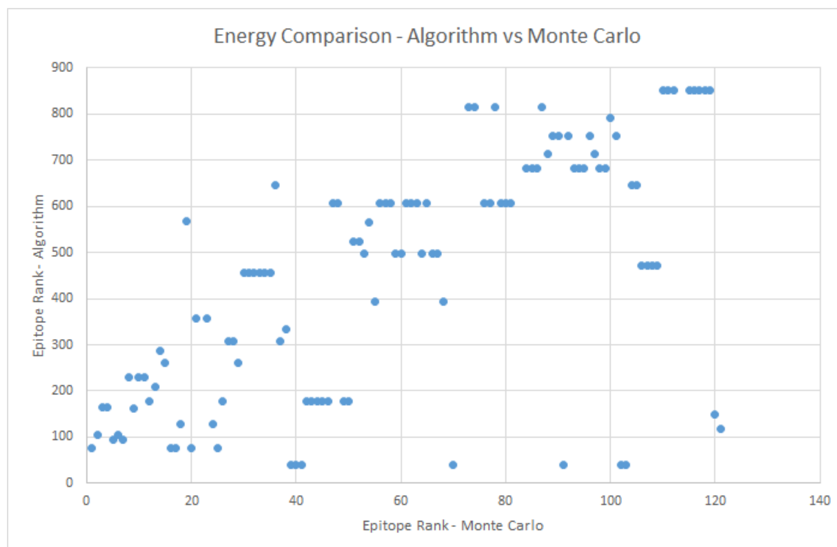

Figure 1. Comparison of the energy rankings of known epitopes using the algorithm given in this paper and a Monte Carlo procedure in [1].

ranks is 0.64946, indicating a high level of correlation between the results of the algorithm and the empirical results.

Running the algorithm on single epitopes in p6, the highest energy attainable is 2.89623, which ranks 38th out of the 53 p6 positions. For doubles, the highest energy is 4.67278, which ranks 765th out of 973 positional doubles, and for triples, the highest energy is 6.01269, which ranks 10004th out of 11252 positional triples. These results indicate that the immune system cannot effectively impede the fitness of the HIV virus in the p6 protein, so we will focus on p24 for the remainder of the paper. (Note that only epitopes of length 9 were considered for p6, which should not significantly affect the results.)

For p24 the results were much more promising. For single epitopes, the highest energy is 26.5836, which ranks 119th out of 222 positions. (The discrepancy with the 231 p24 positions mentioned earlier is due to the fact that 9 positions were considered unattainable in the empirical data.) For doubles, the highest energy is 118.5200, which ranks 52nd out of 6781 positional doubles. For triples, only length 9 epitopes were considered for faster computation; in this case the highest energy is 167.7121, which ranks 244th out of 107607 positional triples. In contrast to the p6 case, some of the highest-energy position combinations in p24 are capable of being targeted by the immune system.

The above algorithm considers all sequences of 8-11 consecutive positions as epitopes, but empirical observations only confirm certain epitopes to be valid targets for the immune system. As a result, it is necessary to determine whether the algorithm's strong performance in the p24 protein is maintained when restricting to only known epitopes; we use the list of 121 known epitopes in [1] to do this. For single epitopes, the highest energy among known epitopes is 18.9450, which ranks 7th out of all possible epitopes produced by the algorithm and 158th out of 222 positions. For epitope doubles, the highest energy among pairs of known epitopes is 79.0092, which ranks 16th out of all epitope pairs and 888th out of 6781 positional doubles. From these results we see that the immune system is somewhat less effective at harming the virus when its behavior

is restricted to targeting known epitopes, but it is still able to perform fairly well particularly in the case of epitope pairs.

6. Conclusions

In this project, we succeeded in developing an algorithm to assess the viability of epitopes and positions within the amino acid sequence of an HIV protein as targets for a potential vaccine. The results of the algorithm were shown to agree with empirical data for the most part, and these results were then used to identify the p24 protein as a promising target for a vaccine-induced immune response.

Further work could be done to improve the effectiveness of the algorithm. In particular, the speed of the algorithm could be improved in the epitope case to allow the study of larger combinations of epitopes. The primary factor that slows down the algorithm is the fact that it computes all possible combinations of positions within a given set of epitopes; if a faster approach could be found, perhaps along the same lines as the genetic algorithm described above, that could speed up the algorithm considerably. Another improvement that would allow the algorithm to more accurately reflect the behavior of the immune system is to generalize from the two-state Ising model to the multi-state Potts model. In this model, the algorithm would distinguish between each of the 19 mutant amino acids to allow for more accurate energy calculations that take the specific amino acids into account. This increased accuracy would come at the cost of speed, so further computational improvements would need to be made to the algorithm to allow it to run in a realistic amount of time.

Acknowledgments

The author would like to thank Andrew L. Ferguson for his guidance in this project. The author acknowledges support from National Science Foundation grant DMS 1345032 "MCTP: PI4: Program for Interdisciplinary and Industrial Internships at Illinois."

References

[1] Andrew L Ferguson, Jaclyn K Mann, Saleha Omarjee, Thumbi Ndungu, Bruce D Walker, and Arup K Chakraborty. Translating hiv sequences into quantitative fitness landscapes predicts viral vulnerabilities for rational immunogen design. *Immunity*, 38(3):606–617, 2013.

Appendix A: MaxMinEnergy.py, runs algorithm to find best epitopes/positions given h,J as input

```
1  from nChoosek import nCk
2  from hMaker import hMaker
3  from Energy import Energy2
4  import time
5
6  #Calculates highest-energy t-tuples of positions
7
8  def MaxMinEnergy(h,J,t):
9      N = len(h)
10     zlist = []
11     zcurrent = hMaker(t)
12     zlist.append(zcurrent[:])
13     #produces list of all possible t-tuples of positions
14     for i in range(nCk(N,t)-1):
15         count = 0
16         zcurrent[t-1] += 1
17         while zcurrent[t-1] > N:
18             count += 1
19             zcurrent[t-count-1] += 1
20             for j in range(count):
21                 zcurrent[t-count+j] = zcurrent[t-count-1]+j+1
```

```
22          zlist.append(zcurrent[:])
23      maxEnergy = 0
24      for z in zlist:
25          energy = Energy2(z,h,J)
26          if energy > maxEnergy:
27              maxZ = z[:]
28              maxEnergy = energy
29      #returns highest-energy position tuple and energy of this position tuple
30      return [maxEnergy,maxZ]
31
32  #Calculates highest-energy t-tuples of positions using genetic algorithm that considers top
        "elts" elements
33
34  def MaxMinEnergyGenetic(h,J,t,elts):
35      faket = 2
36      N = len(h)
37      zlist = []
38      #produces list of all possible pairs
39      for i in range(1,N+1):
40          for j in range(i+1,N+1):
41              zlist.append([i,j])
42      bestlist = []
43      #produces list of "elts" highest-energy triples
44      for z in zlist:
45          energy = Energy2(z,h,J)
46          lowerBound = -1
47          upperBound = len(bestlist)
48          #if list contains fewer elements than "elts", adds current element to appropriate
                place in list
49          if upperBound < elts:
50              while upperBound-lowerBound > 1:
51                  midpt = int(upperBound+lowerBound/2)
52                  if energy < bestlist[midpt][1]:
53                      lowerBound = midpt
54                  else:
55                      upperBound = midpt
56              bestlist = bestlist[:upperBound]+[[z,energy]]+bestlist[upperBound:]
57          #adds current element to appropriate place in list and removes lowest-energy element
58          elif energy > bestlist[elts-1][1]:
59              while upperBound-lowerBound > 1:
60                  midpt = int(upperBound+lowerBound/2)
61                  if energy < bestlist[midpt][1]:
62                      lowerBound = midpt
63                  else:
64                      upperBound = midpt
65              bestlist = bestlist[:upperBound]+[[z,energy]]+bestlist[upperBound:elts-1]
66      #repeats above process until number of positions reaches t
67      while faket < t:
68          faket += 1
69          zlist = []
70          for b in bestlist:
71              for i in range(1,N+1):
72                  if not (i in b[0]):
73                      zlist.append(b[0]+[i])
74          bestlist = []
75          for z in zlist:
76              energy = Energy2(z,h,J)
77              lowerBound = -1
```

```
78              upperBound = len(bestlist)
79              if upperBound < elts:
80                  while upperBound-lowerBound > 1:
81                      midpt = int(upperBound+lowerBound/2)
82                      if energy < bestlist[midpt][1]:
83                          lowerBound = midpt
84                      else:
85                          upperBound = midpt
86                  if (upperBound == len(bestlist) or set(z) != set(bestlist[upperBound][0]))
                        and (lowerBound == -1 or set(z) != set(bestlist[lowerBound][0])):
87                      bestlist = bestlist[:upperBound]+[[z,energy]]+bestlist[upperBound:]
88              elif energy > bestlist[elts-1][1]:
89                  while upperBound-lowerBound > 1:
90                      midpt = int(upperBound+lowerBound/2)
91                      if energy < bestlist[midpt][1]:
92                          lowerBound = midpt
93                      else:
94                          upperBound = midpt
95                  if (upperBound == len(bestlist) or set(z) != set(bestlist[upperBound][0]))
                        and (lowerBound == -1 or set(z) != set(bestlist[lowerBound][0])):
96                      bestlist = bestlist[:upperBound]+[[z,energy]]+bestlist[upperBound:elts
                            -1]
97      #returns highest-energy position tuple and energy of this position tuple
98      return bestlist[0]
99
100 #Calculates highest-energy t-tuple of epitopes of size "epitopeSize"
101
102 def MaxMinEnergyEpitopes(h,J,t,epitopeSize):
103     t1 = time.clock()
104     N = len(h)
105     bcurrent = hMaker(t)
106     blist = [bcurrent[:]]
107     #produces list of all t-tuples of epitope starting positions
108     for i in range(nCk(N-epitopeSize+1,t)-1):
109         count = 0
110         bcurrent[t-1] += 1
111         while bcurrent[t-1] > N-epitopeSize+1:
112             count += 1
113             bcurrent[t-count-1] += 1
114             for j in range(count):
115                 bcurrent[t-count+j] = bcurrent[t-count-1]+j+1
116         blist.append(bcurrent[:])
117     zminlist = []
118     minEnergylist = []
119     #for a given set of epitopes, finds lowest-energy set of positions
120     for b in blist:
121         z = b[:]
122         z[-1] -= 1
123         minEnergy = 1000000000
124         for j in range(epitopeSize**t):
125             z[-1] += 1
126             for k in range(1,t+1):
127                 if z[-k] == b[-k]+epitopeSize:
128                     z[-k] = b[-k]
129                     z[-k-1] += 1
130                 else:
131                     break
132             zadd = list(set(z[:]))
```

```
133                 energy = Energy2(zadd,h,J)
134                 if energy < minEnergy:
135                     minEnergy = energy
136                     zmin = zadd[:]
137             zminlist.append(zmin)
138             minEnergylist.append(minEnergy)
139         maxminE = max(minEnergylist)
140         maxminZ = zminlist[minEnergylist.index(maxminE)]
141         maxminB = blist[minEnergylist.index(maxminE)]
142         t2 = time.clock()
143         print('Elapsed time = '+str(t2-t1)+' seconds')
144         #returns highest-energy set of epitopes, lowest-energy set of positions in these
                epitopes, and energy of this set of positions
145         return [maxminE,maxminZ,maxminB]
146
147 #Calculates highest-energy t-tuple of epitopes of size "epitopeSize" using genetic algorithm
148
149 def MaxMinEnergyEpitopesGenetic(h,J,t,epitopeSize,elts):
150     #ztime = 0
151     t1 = time.clock()
152     faket = 2
153     N = len(h)
154     while True:
155         #first time through, generate all pairs of epitopes
156         if faket == 2:
157             blist = []
158             for i in range(1,N-epitopeSize+2):
159                 for j in range(i+1,N-epitopeSize+2):
160                     blist.append([i,j])
161             zminlist = []
162             minEnergylist = []
163             bminlist = []
164         #subsequent runs, generate all epitope tuples formed by adding an epitope to a tuple
                from previous list
165         else:
166             blist = []
167             for b in bminlist:
168                 for i in range(1,N-epitopeSize+2):
169                     if i not in b:
170                         flag = True
171                         for bprime in blist[:]:
172                             if set(b+[i]) == set(bprime):
173                                 flag = False
174                                 break
175                         if flag:
176                             blist.append(b+[i])
177             zminlist = []
178             minEnergylist = []
179             bminlist = []
180         #for a given set of epitopes, finds lowest-energy set of positions
181         for b in blist:
182             z = b[:]
183             z[-1] -= 1
184             minEnergy = 1000000000
185             #t5 = time.clock()
186             for i in range(epitopeSize**faket):
187                 z[-1] += 1
188                 for k in range(1,faket+1):
```

```
189                     if z[-k] == b[-k]+epitopeSize:
190                         z[-k] = b[-k]
191                         z[-k-1] += 1
192                     else:
193                         break
194                 zadd = list(set(z[:]))
195                 energy = Energy2(zadd,h,J)
196                 if energy < minEnergy:
197                     minEnergy = energy
198                     zmin = zadd[:]
199             #t6 = time.clock()
200             #ztime += (t6-t5)
201             lowerBound = -1
202             upperBound = len(zminlist)
203             #if list contains fewer elements than "elts", adds current element to
                    appropriate place in list
204             if upperBound < elts:
205                 while upperBound-lowerBound > 1:
206                     midpt = int(upperBound+lowerBound/2)
207                     if minEnergy < minEnergylist[midpt]:
208                         lowerBound = midpt
209                     else:
210                         upperBound = midpt
211                 zminlist = zminlist[:upperBound]+[zmin]+zminlist[upperBound:]
212                 minEnergylist = minEnergylist[:upperBound]+[minEnergy]+minEnergylist[
                        upperBound:]
213                 bminlist = bminlist[:upperBound]+[b]+bminlist[upperBound:]
214             #adds current element to appropriate place in list and removes lowest-energy
                    element
215             elif minEnergy > minEnergylist[elts-1]:
216                 while upperBound-lowerBound > 1:
217                     midpt = int(upperBound+lowerBound/2)
218                     if minEnergy < minEnergylist[midpt]:
219                         lowerBound = midpt
220                     else:
221                         upperBound = midpt
222                 zminlist = zminlist[:upperBound]+[zmin]+zminlist[upperBound:elts-1]
223                 minEnergylist = minEnergylist[:upperBound]+[minEnergy]+minEnergylist[
                        upperBound:elts-1]
224                 bminlist = bminlist[:upperBound]+[b]+bminlist[upperBound:elts-1]
225         #terminate if tuple size reaches t
226         if faket < t:
227             faket += 1
228         else:
229             break
230     t2 = time.clock()
231     print('Elapsed time = '+str(t2-t1)+' seconds')
232     #print('Time in z loops = '+str(ztime)+' seconds')
233     #returns highest-energy set of epitopes, lowest-energy set of positions in these
            epitopes, and energy of this set of positions
234     return [minEnergylist[0],zminlist[0],bminlist[0]]
```

Appendix B: MaxMinEnergyImport.py, imports h,J from file and runs algorithm to find best epitopes/positions

```
1 import time
2 from hMaker import hMaker
3 from Energy import Energy2
```

```
4  from nChoosek import nCk
5
6  #Calculates highest-energy t-tuple of epitopes of size "epitopeSize"
7
8  def MaxMinEnergyEpitopesImport(t,epitopeSize,pnum):
9      t1 = time.clock()
10     #import h,J from file
11     hfile = open('C:\\Users\\Ben\\Eclipse\\workspace\\PI4\\src\\h_J_Ising_final\\p'+str(pnum
           )+'\\h.dat')
12     hstring = hfile.read()
13     hfile.close()
14     h = []
15     for i in range(1,len(hstring)):
16         if hstring[i] != ' ' and hstring[i-1] == ' ':
17             numStart = i
18         elif hstring[i] == ' ' and hstring[i-1] != ' ':
19             h.append(float(hstring[numStart:i]))
20     h.append(float(hstring[numStart:]))
21     N = len(h)
22     Jfile = open('C:\\Users\\Ben\\Eclipse\\workspace\\PI4\\src\\h_J_Ising_final\\p'+str(pnum
           )+'\\J.dat')
23     Jstring = Jfile.read()
24     Jfile.close()
25     J = [[]]
26     Jrow = 0
27     Jcol = 0
28     for i in range(1,len(Jstring)):
29         if Jstring[i] != ' ' and Jstring[i-1] == ' ':
30             numStart = i
31         elif Jstring[i] == ' ' and Jstring[i-1] != ' ':
32             J[Jrow].append(float(Jstring[numStart:i]))
33             Jcol += 1
34             if Jcol >= N:
35                 Jrow += 1
36                 Jcol = 0
37                 J.append([])
38     J[Jrow].append(float(Jstring[numStart:]))
39     #produce list of t-tuples of epitopes
40     bcurrent = hMaker(t)
41     blist = [bcurrent[:]]
42     for i in range(nCk(N-epitopeSize+1,t)-1):
43         count = 0
44         bcurrent[t-1] += 1
45         while bcurrent[t-1] > N-epitopeSize+1:
46             count += 1
47             bcurrent[t-count-1] += 1
48             for j in range(count):
49                 bcurrent[t-count+j] = bcurrent[t-count-1]+j+1
50         blist.append(bcurrent[:])
51     zminlist = []
52     minEnergylist = []
53     #find lowest-energy tuples of positions within a given t-tuple of epitopes
54     for b in blist:
55         z = b[:]
56         z[-1] -= 1
57         minEnergy = 100000000000
58         for j in range(epitopeSize**t):
59             z[-1] += 1
```

```
60              for k in range(1,t+1):
61                  if z[-k] == b[-k]+epitopeSize:
62                      z[-k] = b[-k]
63                      z[-k-1] += 1
64                  else:
65                      break
66              zadd = list(set(z[:]))
67              energy = Energy2(zadd,h,J)
68              if energy < minEnergy:
69                  minEnergy = energy
70                  zmin = zadd[:]
71          if minEnergy < 1000000000:
72              zminlist.append(zmin)
73              minEnergylist.append(minEnergy)
74          else:
75              zminlist.append(0)
76              minEnergylist.append(0)
77      #sort lists by energy
78      maxminE = max(minEnergylist)
79      maxminZ = zminlist[minEnergylist.index(maxminE)]
80      maxminB = blist[minEnergylist.index(maxminE)]
81      t2 = time.clock()
82      print('Elapsed time = '+str(t2-t1)+' seconds')
83      #returns highest-energy set of epitopes, lowest-energy set of positions in these
            epitopes, and energy of this set of positions
84      return [maxminE,maxminZ,maxminB]
```

Appendix C: MaxMinEnergyAll.py, imports h,J from file and runs algorithm to produce ranked list of epitopes/positions

```
1  import time
2  from hMaker import hMaker
3  from nChoosek import nCk
4  from Energy import Energy2
5
6  #Produces list of t-tuples of positions sorted by energy
7
8  def MaxMinEnergyAll(t,pnum):
9      t1 = time.clock()
10     #imports h,J from file
11     hfile = open('C:\\Users\\Ben\\Dropbox\\Grad School\\PI4 2014\\h_J_Ising_final\\p'+str(
           pnum)+'\\h.dat')
12     hstring = hfile.read()
13     hfile.close()
14     h = []
15     for i in range(1,len(hstring)):
16         if hstring[i] != ' ' and hstring[i-1] == ' ':
17             numStart = i
18         elif hstring[i] == ' ' and hstring[i-1] != ' ':
19             h.append(float(hstring[numStart:i]))
20     h.append(float(hstring[numStart:]))
21     N = len(h)
22     Jfile = open('C:\\Users\\Ben\\Dropbox\\Grad School\\PI4 2014\\h_J_Ising_final\\p'+str(
           pnum)+'\\J.dat')
23     Jstring = Jfile.read()
24     Jfile.close()
25     J = [[]]
26     Jrow = 0
```

```
27      Jcol = 0
28      for i in range(1,len(Jstring)):
29          if Jstring[i] != ' ' and Jstring[i-1] == ' ':
30              numStart = i
31          elif Jstring[i] == ' ' and Jstring[i-1] != ' ':
32              J[Jrow].append(float(Jstring[numStart:i]))
33              Jcol += 1
34              if Jcol >= N:
35                  Jrow += 1
36                  Jcol = 0
37                  J.append([])
38      J[Jrow].append(float(Jstring[numStart:]))
39      zlist = []
40      Elist = []
41      zcurrent = hMaker(t)
42      zcurrent[-1] -= 1
43      #generates list of tuples of positions and calculates energy of each
44      for i in range(nCk(N,t)):
45          count = 0
46          zcurrent[t-1] += 1
47          while zcurrent[t-1] > N:
48              count += 1
49              zcurrent[t-count-1] += 1
50              for j in range(count):
51                  zcurrent[t-count+j] = zcurrent[t-count-1]+j+1
52          energy = Energy2(zcurrent,h,J)
53          if energy < 1000000000:
54              zlist.append(zcurrent[:])
55              Elist.append(energy)
56      #sort lists by energy
57      sortedElist = sorted(Elist,reverse = True)
58      sortedzlist = []
59      for E in sortedElist:
60          sortedzlist.append(zlist[Elist.index(E)])
61      t2 = time.clock()
62      print('Elapsed time = '+str(t2-t1)+' seconds')
63      #return list of tuples of positions, list of corresponding energies
64      return [sortedzlist,sortedElist]
65
66  #Produces list of "elts" highest-energy t-tuples of positions sorted by energy using genetic
        algorithm
67
68  def MaxMinEnergyGeneticAll(t,elts,pnum):
69      t1 = time.clock()
70      #import h,J from file
71      hfile = open('C:\\Users\\Ben\\Dropbox\\Grad School\\PI4 2014\\h_J_Ising_final\\p'+str(
            pnum)+'\\h.dat')
72      hstring = hfile.read()
73      hfile.close()
74      h = []
75      for i in range(1,len(hstring)):
76          if hstring[i] != ' ' and hstring[i-1] == ' ':
77              numStart = i
78          elif hstring[i] == ' ' and hstring[i-1] != ' ':
79              h.append(float(hstring[numStart:i]))
80      h.append(float(hstring[numStart:]))
81      N = len(h)
```

```
82      Jfile = open('C:\\Users\\Ben\\Dropbox\\Grad School\\PI4 2014\\h_J_Ising_final\\p'+str(
            pnum)+'\\J.dat')
83      Jstring = Jfile.read()
84      Jfile.close()
85      J = [[]]
86      Jrow = 0
87      Jcol = 0
88      for i in range(1,len(Jstring)):
89          if Jstring[i] != ' ' and Jstring[i-1] == ' ':
90              numStart = i
91          elif Jstring[i] == ' ' and Jstring[i-1] != ' ':
92              J[Jrow].append(float(Jstring[numStart:i]))
93              Jcol += 1
94              if Jcol >= N:
95                  Jrow += 1
96                  Jcol = 0
97                  J.append([])
98      J[Jrow].append(float(Jstring[numStart:]))
99      faket = 2
100     zlist = []
101     #generates all pairs of positions
102     for i in range(1,N+1):
103         for j in range(i+1,N+1):
104             zlist.append([i,j])
105     newzlist = []
106     Elist = []
107     #finds "elts" highest-energy pairs of positions
108     for z in zlist:
109         energy = Energy2(z,h,J)
110         if energy < 1000000000:
111             lowerBound = -1
112             upperBound = len(newzlist)
113             if upperBound < elts:
114                 while upperBound-lowerBound > 1:
115                     midpt = int(upperBound+lowerBound/2)
116                     if energy < Elist[midpt]:
117                         lowerBound = midpt
118                     else:
119                         upperBound = midpt
120                 newzlist = newzlist[:upperBound]+[z]+newzlist[upperBound:]
121                 Elist = Elist[:upperBound]+[energy]+Elist[upperBound:]
122             elif energy > Elist[elts-1]:
123                 while upperBound-lowerBound > 1:
124                     midpt = int(upperBound+lowerBound/2)
125                     if energy < Elist[midpt]:
126                         lowerBound = midpt
127                     else:
128                         upperBound = midpt
129                 newzlist = newzlist[:upperBound]+[z]+newzlist[upperBound:]
130                 Elist = Elist[:upperBound]+[energy]+Elist[upperBound:]
131     while faket < t:
132         faket += 1
133         zlist = []
134         #produces list of tuples formed by adding one position to a tuple in previous list
135         for z in newzlist:
136             for i in range(1,N+1):
137                 if not (i in z):
138                     zlist.append(z+[i])
```

```
139             newzlist = []
140             Elist = []
141             #finds "elts" highest-energy tuples of positions
142             for z in zlist:
143                 energy = Energy2(z,h,J)
144                 if energy < 1000000000:
145                     lowerBound = -1
146                     upperBound = len(newzlist)
147                     if upperBound < elts:
148                         while upperBound-lowerBound > 1:
149                             midpt = int(upperBound+lowerBound/2)
150                             if energy < Elist[midpt]:
151                                 lowerBound = midpt
152                             else:
153                                 upperBound = midpt
154                         if (upperBound == len(newzlist) or set(z) != set(newzlist[upperBound]))
                                and (lowerBound == -1 or set(z) != set(newzlist[lowerBound])):
155                             newzlist = newzlist[:upperBound]+[z]+newzlist[upperBound:]
156                             Elist = Elist[:upperBound]+[energy]+Elist[upperBound:]
157                     elif energy > Elist[elts-1]:
158                         while upperBound-lowerBound > 1:
159                             midpt = int(upperBound+lowerBound/2)
160                             if energy < Elist[midpt]:
161                                 lowerBound = midpt
162                             else:
163                                 upperBound = midpt
164                         if set(z) != set(newzlist[upperBound]) and (lowerBound == -1 or set(z)
                                != set(newzlist[lowerBound])):
165                             newzlist = newzlist[:upperBound]+[z]+newzlist[upperBound:elts-1]
166                             Elist = Elist[:upperBound]+[energy]+Elist[upperBound:elts-1]
167         #sort lists by energy
168         sortedElist = sorted(Elist,reverse = True)
169         sortedzlist = []
170         for E in sortedElist:
171             sortedzlist.append(newzlist[Elist.index(E)])
172         t2 = time.clock()
173         print('Elapsed time = '+str(t2-t1)+' seconds')
174         #return list of tuples of positions, list of corresponding energies
175         return [sortedzlist,sortedElist]
176
177     #Produces list of t-tuples of epitopes sorted by energy
178
179     def MaxMinEnergyEpitopesAll(t,epitopeSize,pnum):
180         t1 = time.clock()
181         #import h,J from file
182         hfile = open('C:\\Users\\Ben\\Dropbox\\Grad School\\PI4 2014\\h_J_Ising_final\\p'+str(
                pnum)+'\\h.dat')
183         hstring = hfile.read()
184         hfile.close()
185         h = []
186         for i in range(1,len(hstring)):
187             if hstring[i] != ' ' and hstring[i-1] == ' ':
188                 numStart = i
189             elif hstring[i] == ' ' and hstring[i-1] != ' ':
190                 h.append(float(hstring[numStart:i]))
191         h.append(float(hstring[numStart:]))
192         N = len(h)
```

```
193     Jfile = open('C:\\Users\\Ben\\Dropbox\\Grad School\\PI4 2014\\h_J_Ising_final\\p'+str(
            pnum)+'\\J.dat')
194     Jstring = Jfile.read()
195     Jfile.close()
196     J = [[]]
197     Jrow = 0
198     Jcol = 0
199     for i in range(1,len(Jstring)):
200         if Jstring[i] != ' ' and Jstring[i-1] == ' ':
201             numStart = i
202         elif Jstring[i] == ' ' and Jstring[i-1] != ' ':
203             J[Jrow].append(float(Jstring[numStart:i]))
204             Jcol += 1
205             if Jcol >= N:
206                 Jrow += 1
207                 Jcol = 0
208                 J.append([])
209     J[Jrow].append(float(Jstring[numStart:]))
210     bcurrent = hMaker(t)
211     blist = [bcurrent[:]]
212     #produce list of t-tuples of epitopes of size "epitopeSize"
213     for i in range(nCk(N-epitopeSize+1,t)-1):
214         count = 0
215         bcurrent[t-1] += 1
216         while bcurrent[t-1] > N-epitopeSize+1:
217             count += 1
218             bcurrent[t-count-1] += 1
219             for j in range(count):
220                 bcurrent[t-count+j] = bcurrent[t-count-1]+j+1
221         blist.append(bcurrent[:])
222     zminlist = []
223     minEnergylist = []
224     #find lowest-energy set of positions in a given t-tuple of epitopes
225     for b in blist[:]:
226         z = b[:]
227         z[-1] -= 1
228         minEnergy = 100000000000
229         for j in range(epitopeSize**t):
230             z[-1] += 1
231             for k in range(1,t+1):
232                 if z[-k] == b[-k]+epitopeSize:
233                     z[-k] = b[-k]
234                     z[-k-1] += 1
235                 else:
236                     break
237             zadd = list(set(z[:]))
238             energy = Energy2(zadd,h,J)
239             if energy < minEnergy:
240                 minEnergy = energy
241                 zmin = zadd[:]
242         if minEnergy < 1000000000:
243             zminlist.append(zmin)
244             minEnergylist.append(minEnergy)
245         else:
246             blist.remove(b)
247     #sort lists by energy
248     sortedElist = sorted(minEnergylist,reverse = True)
249     M = len(zminlist)
```

```
250     sortedzlist = [0]*M
251     sortedblist = [0]*M
252     for i in range(M):
253         j = sortedElist.index(minEnergylist[i])
254         while True:
255             if sortedblist[j] == 0:
256                 sortedblist[j] = blist[i]
257                 sortedzlist[j] = zminlist[i]
258                 break
259             else:
260                 j += 1
261     t2 = time.clock()
262     print('Elapsed time = '+str(t2-t1)+' seconds')
263     #return list of epitope tuples, list of lowest-energy position tuples within each
            epitopes tuple, list of corresponding energies
264     return [sortedblist,sortedElist,sortedzlist]
265 
266 #Produces list of t-tuples of variable length epitopes sorted by energy
267 
268 def MaxMinEnergyEpitopesVariableAll(t,pnum):
269     t1 = time.clock()
270     #import h,J from file
271     hfile = open('C:\\Users\\Ben\\Dropbox\\Grad School\\PI4 2014\\h_J_Ising_final\\p'+str(
            pnum)+'\\h.dat')
272     hstring = hfile.read()
273     hfile.close()
274     h = []
275     for i in range(1,len(hstring)):
276         if hstring[i] != ' ' and hstring[i-1] == ' ':
277             numStart = i
278         elif hstring[i] == ' ' and hstring[i-1] != ' ':
279             h.append(float(hstring[numStart:i]))
280     h.append(float(hstring[numStart:]))
281     N = len(h)
282     Jfile = open('C:\\Users\\Ben\\Dropbox\\Grad School\\PI4 2014\\h_J_Ising_final\\p'+str(
            pnum)+'\\J.dat')
283     Jstring = Jfile.read()
284     Jfile.close()
285     J = [[]]
286     Jrow = 0
287     Jcol = 0
288     for i in range(1,len(Jstring)):
289         if Jstring[i] != ' ' and Jstring[i-1] == ' ':
290             numStart = i
291         elif Jstring[i] == ' ' and Jstring[i-1] != ' ':
292             J[Jrow].append(float(Jstring[numStart:i]))
293             Jcol += 1
294             if Jcol >= N:
295                 Jrow += 1
296                 Jcol = 0
297                 J.append([])
298     J[Jrow].append(float(Jstring[numStart:]))
299 
300     #produce list of possible epitope lengths
301     epitopeSizeList = []
302     currentEpList = [8]*t
303     while currentEpList[0] < 12:
304         epitopeSizeList.append(currentEpList[:])
```

```
305             currentEpList[-1] += 1
306             for k in range(1,t):
307                 if currentEpList[-k] == 12:
308                     currentEpList[-k-1] += 1
309                     for l in range(1,k+1):
310                         currentEpList[-l] = currentEpList[-k-1]
311                 else:
312                     break
313
314         #produce list of t-tuples of epitopes
315         zminlist = []
316         minEnergylist = []
317         epitopeSizesBigList = []
318         bigBlist = []
319         for epitopeSizes in epitopeSizeList:
320             print("Currently running size "+str(epitopeSizes))
321             bcurrent = [1]*t
322             blist = []
323             while bcurrent[0] < N-epitopeSizes[0]+2:
324                 flag = True
325                 for i in range(t):
326                     for j in range(i+1,t):
327                         if epitopeSizes[i] == epitopeSizes[j] and bcurrent[i] < bcurrent[j]:
328                             flag = False
329                 if flag:
330                     blist.append(bcurrent[:])
331                 bcurrent[-1] += 1
332                 for k in range(1,t):
333                     if bcurrent[-k] == N-epitopeSizes[-k]+2:
334                         bcurrent[-k-1] += 1
335                         bcurrent[-k] = 1
336                     else:
337                         break
338
339             #for each t-tuple of epitopes, find lowest-energy tuple of positions
340             print("length of blist = "+str(len(blist)))
341             for b in blist[:]:
342                 z = b[:]
343                 minEnergy = 100000000000
344                 while z[0] < b[0]+epitopeSizes[0]:
345                     zadd = list(set(z[:]))
346                     energy = Energy2(zadd,h,J)
347                     if energy < minEnergy:
348                         minEnergy = energy
349                         zmin = zadd[:]
350                     z[-1] += 1
351                     for k in range(1,t):
352                         if z[-k] == b[-k]+epitopeSizes[-k]:
353                             z[-k] = b[-k]
354                             z[-k-1] += 1
355                         else:
356                             break
357                 if minEnergy < 1000000000:
358                     zminlist.append(zmin[:])
359                     minEnergylist.append(minEnergy)
360                     epitopeSizesBigList.append(epitopeSizes)
361                     bigBlist.append(b)
362
```

```
363      #sort lists by energy
364      sortedElist = sorted(minEnergylist,reverse = True)
365      M = len(zminlist)
366      sortedzlist = [0]*M
367      sortedblist = [0]*M
368      sortedEplist = [0]*M
369      for i in range(M):
370          if i == M//2:
371              print("Sorting 50% done")
372          j = sortedElist.index(minEnergylist[i])
373          while True:
374              if sortedblist[j] == 0:
375                  sortedblist[j] = bigBlist[i]
376                  sortedzlist[j] = zminlist[i]
377                  sortedEplist[j] = epitopeSizesBigList[i]
378                  break
379              else:
380                  j += 1
381      print("Sorting 100% done")
382      finalblist = []
383      #form list of epitopes by combining starting positions and lengths
384      for i in range(M):
385          finalblist.append([])
386          for j in range(t):
387              finalblist[i].append([sortedblist[i][j],sortedblist[i][j]+sortedEplist[i][j]-1])
388      t2 = time.clock()
389      print("Final list complete")
390      newfile = open('C:\\Users\\Ben\\Dropbox\\Grad School\\PI4 2014\\variablePairs.txt','w')
391      newfile.write(str(finalblist)+'\n')
392      newfile.write('\n')
393      newfile.write(str(sortedElist)+'\n')
394      newfile.write('\n')
395      newfile.write(str(sortedzlist)+'\n')
396      newfile.close()
397      print('Elapsed time = '+str(t2-t1)+' seconds')
398
399  #Produces list of "elts" highest-energy t-tuples of epitopes sorted by energy using genetic
         algorithm
400
401  def MaxMinEnergyEpitopesGeneticAll(t,epitopeSize,elts,pnum):
402      t1 = time.clock()
403      #import h,J from file
404      hfile = open('C:\\Users\\Ben\\Dropbox\\Grad School\\PI4 2014\\h_J_Ising_final\\p'+str(
             pnum)+'\\h.dat')
405      hstring = hfile.read()
406      hfile.close()
407      h = []
408      for i in range(1,len(hstring)):
409          if hstring[i] != ' ' and hstring[i-1] == ' ':
410              numStart = i
411          elif hstring[i] == ' ' and hstring[i-1] != ' ':
412              h.append(float(hstring[numStart:i]))
413      h.append(float(hstring[numStart:]))
414      N = len(h)
415      Jfile = open('C:\\Users\\Ben\\Dropbox\\Grad School\\PI4 2014\\h_J_Ising_final\\p'+str(
             pnum)+'\\J.dat')
416      Jstring = Jfile.read()
417      Jfile.close()
```

```
418     J = [[]]
419     Jrow = 0
420     Jcol = 0
421     for i in range(1,len(Jstring)):
422         if Jstring[i] != ' ' and Jstring[i-1] == ' ':
423             numStart = i
424         elif Jstring[i] == ' ' and Jstring[i-1] != ' ':
425             J[Jrow].append(float(Jstring[numStart:i]))
426             Jcol += 1
427             if Jcol >= N:
428                 Jrow += 1
429                 Jcol = 0
430                 J.append([])
431     J[Jrow].append(float(Jstring[numStart:]))
432     faket = 2
433     while True:
434         #generates all pairs of epitopes
435         if faket == 2:
436             blist = []
437             for i in range(1,N-epitopeSize+2):
438                 for j in range(i+1,N-epitopeSize+2):
439                     blist.append([i,j])
440             zminlist = []
441             minEnergylist = []
442             bminlist = []
443         else:
444             blist = []
445             #produces list of tuples formed by adding one epitopes to a tuple in previous
                    list
446             for b in bminlist:
447                 for i in range(1,N-epitopeSize+2):
448                     if i not in b:
449                         flag = True
450                         for bprime in blist[:]:
451                             if set(b+[i]) == set(bprime):
452                                 flag = False
453                                 break
454                         if flag:
455                             blist.append(b+[i])
456             zminlist = []
457             minEnergylist = []
458             bminlist = []
459         #for each t-tuple of epitopes, find lowest-energy tuple of positions
460         for b in blist:
461             z = b[:]
462             z[-1] -= 1
463             minEnergy = 1000000000
464             #t5 = time.clock()
465             for i in range(epitopeSize**faket):
466                 z[-1] += 1
467                 for k in range(1,faket+1):
468                     if z[-k] == b[-k]+epitopeSize:
469                         z[-k] = b[-k]
470                         z[-k-1] += 1
471                     else:
472                         break
473                 zadd = list(set(z[:]))
474                 energy = Energy2(zadd,h,J)
```

```
475                     if energy < minEnergy:
476                         minEnergy = energy
477                         zmin = zadd[:]
478                 #t6 = time.clock()
479                 #ztime += (t6-t5)
480                 if minEnergy < 1000000000:
481                     lowerBound = -1
482                     upperBound = len(zminlist)
483                     if upperBound < elts:
484                         while upperBound-lowerBound > 1:
485                             midpt = int(upperBound+lowerBound/2)
486                             if minEnergy < minEnergylist[midpt]:
487                                 lowerBound = midpt
488                             else:
489                                 upperBound = midpt
490                         zminlist = zminlist[:upperBound]+[zmin]+zminlist[upperBound:]
491                         minEnergylist = minEnergylist[:upperBound]+[minEnergy]+minEnergylist[
                                upperBound:]
492                         bminlist = bminlist[:upperBound]+[b]+bminlist[upperBound:]
493                     elif minEnergy > minEnergylist[elts-1]:
494                         while upperBound-lowerBound > 1:
495                             midpt = int(upperBound+lowerBound/2)
496                             if minEnergy < minEnergylist[midpt]:
497                                 lowerBound = midpt
498                             else:
499                                 upperBound = midpt
500                         zminlist = zminlist[:upperBound]+[zmin]+zminlist[upperBound:elts-1]
501                         minEnergylist = minEnergylist[:upperBound]+[minEnergy]+minEnergylist[
                                upperBound:elts-1]
502                         bminlist = bminlist[:upperBound]+[b]+bminlist[upperBound:elts-1]
503         if faket < t:
504             faket += 1
505         else:
506             break
507     t2 = time.clock()
508     print('Elapsed time = '+str(t2-t1)+' seconds')
509     #return list of epitope tuples, list of lowest-energy position tuples within each
            epitopes tuple, list of corresponding energies
510     return [bminlist,minEnergylist,zminlist]
```

Appendix D: MaxMinEnergyRandom.py, produces random h,J and runs algorithm to find best epitopes/positions

```
1   import random
2   import MaxMinEnergy
3
4   #Calculates highest-energy t-tuple of epitopes with random h,J, runs "trials" times
5
6   def MaxMinEnergyRandom(N,t,epitopeSize,elts,trials,Genetic = False):
7       for ct in range(trials):
8           h = []
9           J = []
10          for i in range(N):
11              h.append(random.uniform(0,10))
12              J.append([0]*N)
13              for j in range(i+1,N):
14                  J[i][j] = random.gauss(0,1)
15          if Genetic:
```

```
16                print(MaxMinEnergy.MaxMinEnergyEpitopesGenetic(h, J, t, epitopeSize, elts))
17            else:
18                print(MaxMinEnergy.MaxMinEnergyEpitopes(h, J, t, epitopeSize))
19
20 #Tests if genetic algorithm gives same result as exhaustive algorithm with random h,J, runs
       "trials" times
21
22 def MaxMinEnergyRandomTester(N,t,epitopeSize,elts,trials):
23     successCount = 0
24     for ct in range(trials):
25         h = []
26         J = []
27         for i in range(N):
28             h.append(random.uniform(0,10))
29             J.append([0]*N)
30             for j in range(i+1,N):
31                 J[i][j] = random.gauss(0,1)
32         [a,b] = [MaxMinEnergy.MaxMinEnergyEpitopes(h, J, t, epitopeSize), MaxMinEnergy.
               MaxMinEnergyEpitopesGenetic(h, J, t, epitopeSize,elts)]
33         if abs(a[0] - b[0]) < 10**(-9):
34             print('Success!')
35             print(a)
36             successCount += 1
37         else:
38             print('Failure!')
39             print('Actual:',a)
40             print('Consecutive:',b)
41     print('Number of successes = '+str(successCount))
42     print('Number of failures = '+str(trials-successCount))
```

Appendix E: Energy.py, calculates energy for a given amino acid sequence

```
1  def E(h,z,b,J):
2      Evalue = 0
3      for i in range(0,len(z)):
4          Evalue = Evalue+h[i]*z[i]+b[i]*(1-z[i])
5          for j in range(len(z)):
6              if z[i] == z[j] == 1:
7                  Evalue = Evalue+J[i][j]
8      return Evalue
9
10 #evaluates energy given z,h,J
11
12 def Energy2(z,h,J):
13     energy = 0
14     for zi in z:
15         energy += h[zi-1]
16         for zj in z:
17             energy += J[zi-1][zj-1]
18     return energy
```

Appendix F: nChooseK.py, calculates binomial coefficient $\binom{n}{k}$

```
1  from operator import mul
2  from fractions import Fraction
3  from functools import reduce
4
5  #returns the binomial coefficient n choose k
6
```

```
7 def nCk(n,k):
8     return int( reduce(mul, (Fraction(n-i, i+1) for i in range(k)), 1) )
```

Appendix G: hMaker.py, produces a vector 1,2,...,N

```
1 #produces list [1,2,...,N]
2
3 def hMaker(N):
4     h = []
5     for i in range(N):
6         h.append(i+1)
7     return h
```

Department of Mathematics, University of Illinois at Urbana–Champaign, Urbana, IL 61801, USA