Muyao Wu

April 26, 2017

HW2

Question 1 (output at the very end) I bolded the answer:

In [18]:

```
#python locAL.py <seq file> -m <match> -s <mismatch> -d <indel> -a
#python locAL.py testseqs.txt -m +1 -s -1 -d -1 -a

import sys, getopt, numpy

arguments = ["locAL", "p1seqs.txt", "-m", "1", "-s","-10", "-d", "-1", "-a"]


file = arguments[1]
matchScore = int(arguments[3])
mismatchScore =  int(arguments[5])
indel = int(arguments[7])
findA = False
if '-a' in arguments:
      findA = True
      #print('found -a')

print ('Number of arguments:', len(arguments), 'arguments.')
print ('Argument List:', str(arguments))
print ('file:', str(file))
print ('matchScore:', str(matchScore))
print ('mismatchScore:', str(mismatchScore))
print ('indel:', str(indel))
print ('findA:', str(findA))
```

```
Number of arguments: 9 arguments.
Argument List: ['locAL', 'p1seqs.txt', '-m', '1', '-s', '-10', '-d', '-1', '-
a']
file: p1seqs.txt
matchScore: 1
mismatchScore: -10
```

```
indel: -1
findA: True
```

In [19]:

```python
data = open(file, "r")


check = False
counter = 0
seq1 = []
seq2 = []
for line in data:
    if 'seq' in line:
        check = True
    elif check == True and counter == 0:
        for char in line:
            if char != '\n':
                seq1.append(char)

        counter +=1
    elif check == True and counter == 1:
        for char in line:
            if char != '\n':
                seq2.append(char)
```

In [ ]:

In [21]:

```python
#to find the max score

#initialize all them

#vert = insertion
vert = numpy.empty((len(seq2)+1, len(seq1)+1))
vert[:] = 0

d = 1
while d < len(vert):
    vert[d][0] = indel*d
    d+=1
```

```
d=1
while d < len(vert[0]):
      vert[0][d] = -float("inf")
      d+=1

vert [0][0] = 0


#hori = deletion
hori = numpy.empty((len(seq2)+1, len(seq1)+1))
hori[:] = 0
d=1
while d < len(hori[0]):
      hori[0][d] = indel*d
      d+=1

d=1
while d < len(hori):
      hori[d][0] = -float("inf")
      d+=1
hori[0][0]=0

#diag = match or mismatch
diag = numpy.empty((len(seq2)+1, len(seq1)+1))
diag[:] = 0
d=1
while d < len(diag[0]):
      diag[0][d] = indel*d
      d+=1
d=1
while d < len(diag):
      diag[d][0] = indel*d
      d+=1
diag [0][0] = 0

#score for scorekeeping
score = numpy.empty((len(seq2)+1, len(seq1)+1))
```

```
score[:] = numpy.NAN

d=1
while d < len(score[0]):
      score[0][d] = 0
      d+=1
d=1
while d < len(score):
      score[d][0] = 0
      d+=1
score [0][0] = 0

#1 = Vert, 2 = Horiz, 3 = diag

dire = numpy.empty((len(seq2)+1, len(seq1)+1))
d=1
while d < len(dire[0]):
      dire[0][d] = 2
      d+=1
d=1
while d < len(dire):
      dire[d][0] = 1
      d+=1

print ("Vertical")
print (vert)

print ("Horizontal")
print (hori)

print ("Diagonal")
print (diag)

print ("Score")
print (score)

print ("Direction")
print (dire)
```

```
#run the script until we good
```

```
Vertical
[[    0.   -inf   -inf ...,   -inf   -inf   -inf]
 [   -1.     0.     0. ...,     0.     0.     0.]
 [   -2.     0.     0. ...,     0.     0.     0.]
 ...,
 [ -998.     0.     0. ...,     0.     0.     0.]
 [ -999.     0.     0. ...,     0.     0.     0.]
 [-1000.     0.     0. ...,     0.     0.     0.]]
Horizontal
[[    0.    -1.    -2. ...,  -998.  -999. -1000.]
 [  -inf     0.     0. ...,     0.     0.     0.]
 [  -inf     0.     0. ...,     0.     0.     0.]
 ...,
 [  -inf     0.     0. ...,     0.     0.     0.]
 [  -inf     0.     0. ...,     0.     0.     0.]
 [  -inf     0.     0. ...,     0.     0.     0.]]
Diagonal
[[    0.    -1.    -2. ...,  -998.  -999. -1000.]
 [   -1.     0.     0. ...,     0.     0.     0.]
 [   -2.     0.     0. ...,     0.     0.     0.]
 ...,
 [ -998.     0.     0. ...,     0.     0.     0.]
 [ -999.     0.     0. ...,     0.     0.     0.]
 [-1000.     0.     0. ...,     0.     0.     0.]]
Score
[[  0.   0.   0. ...,   0.   0.   0.]
 [  0.  nan  nan ...,  nan  nan  nan]
 [  0.  nan  nan ...,  nan  nan  nan]
 ...,
 [  0.  nan  nan ...,  nan  nan  nan]
 [  0.  nan  nan ...,  nan  nan  nan]
 [  0.  nan  nan ...,  nan  nan  nan]]
Direction
[[  0.   2.   2. ...,   2.   2.   2.]
```

```
 [   1.   nan   nan ...,   nan   nan   nan]
 [   1.   nan   nan ...,   nan   nan   nan]
 ...,
 [   1.   nan   nan ...,   nan   nan   nan]
 [   1.   nan   nan ...,   nan   nan   nan]
 [   1.   nan   nan ...,   nan   nan   nan]]
```

In [ ]:

In [24]:

```
# here we going to loop through the whole thing and go from top left to
bottom right


#let's make a variable to keep track of the biggest score value:
maxScore = 0
bestLoc = (0,0)
# we want to iterate 1-10 in the 3 matrices. This is the nested for loop
i = 1
while i < len(diag):
    j=1
    while j < len(diag[i]):
        #print ('current i and j: ', i , ' ', j)
        #we gotta manipulate each matrix we're working with

        #Vertical
        a = vert[i-1][j]
        b = diag[i-1][j]

        if a>=b:
            vert[i][j] = a + indel
        elif b>=a:
            vert[i][j] = b + indel


        #Horizontal
        a = hori[i][j-1]
        b = diag[i][j-1]
        if a>=b:
            hori[i][j] = a + indel
        elif b>=a:
            hori[i][j] = b + indel

        #diag


        a = vert[i][j]
        b = hori[i][j]

        #print((seq1[j-1],seq2[i-1]))

        if(int(seq1[j-1]==seq2[i-1]) ==0):
            cScore = mismatchScore
        else:
            cScore = matchScore

        c = diag[i-1][j-1] + cScore


        if a>=b and a>=c:
            dire[i][j] = "1"
            diag[i][j]=a
        if b>=a and b>=c:
            dire[i][j] = "2"
            diag[i][j]=b
        if c>=a and c>=b:
            dire[i][j] = "3"
            diag[i][j]=c
        if diag[i][j]<0:
            dire[i][j] = 0

        if(diag[i][j] >= maxScore):
            maxScore = diag[i][j]
            bestLoc = (i,j)

        j+=1
```

```
    i+=1

print('vertical: ')
print(vert)
print('horizontal: ')
print(hori)
print('diagonal: ')
print(diag)

print('Directional: ')
print(dire)


print('best: ', maxScore)
print(bestLoc)


# run this if -a is on
#reset directional borders to zero:
d=1
while d < len(dire[0]):
      dire[0][d] = 0
      d+=1
d=1
while d < len(dire):
      dire[d][0] = 0
      d+=1


# let's write a function to find the local alignment


ali1 = ""
#print(seq1)
#print(seq1[bestLoc[1]-1])

#remember seq2 is the y value but is presented first in the coordinates
```

```
ali2 = ""
#print(seq2)
#print(seq2[bestLoc[0]-1])




k = 0
current = bestLoc

while k == 0:
    #print ("currentLoc: ", bestLoc, " currentDire ",dire[bestLoc])

    #on zero we stop

    if dire[bestLoc] == 0:
        print("stopped at: ", bestLoc)
        k=1
    #on 1 we go up. so i changes but j stays the same
    elif dire[bestLoc] == 1:
        bestLoc = (bestLoc[0]-1, bestLoc[1])
        ali1 = ali1 + "-"
        ali2 = ali2 + str(seq2[bestLoc[0]])
    #on 2 we go left so j changes but i stays constant
    elif dire[bestLoc] == 2:
        bestLoc = (bestLoc[0], bestLoc[1]-1)
        ali2 = ali2 + "-"
        ali1 = ali1 = ali1 + str(seq1[bestLoc[1]])
    #on 3 both change, yay!
    elif dire[bestLoc] == 3:
        bestLoc = (bestLoc[0]-1, bestLoc[1]-1)
        ali1 = ali1 + str(seq1[bestLoc[1]])
        ali2 = ali2 + str(seq2[bestLoc[0]])


print('Best Score: ', maxScore)
print ('Length: ', len(ali1))
print (ali1[::-1])
print (ali2[::-1])
```

```
vertical:
[[  0.00000000e+00             -inf             -inf ...,             -inf
              -inf             -inf]
 [ -1.00000000e+00  -2.00000000e+00  -3.00000000e+00 ...,  -9.99000000e+02
   -1.00000000e+03  -1.00100000e+03]
 [ -2.00000000e+00   0.00000000e+00  -1.00000000e+00 ...,  -9.97000000e+02
   -9.98000000e+02  -9.99000000e+02]
 ...,
 [ -9.98000000e+02  -9.96000000e+02  -9.94000000e+02 ...,   1.01000000e+02
    1.03000000e+02   1.02000000e+02]
 [ -9.99000000e+02  -9.97000000e+02  -9.95000000e+02 ...,   1.00000000e+02
    1.02000000e+02   1.04000000e+02]
 [ -1.00000000e+03  -9.98000000e+02  -9.96000000e+02 ...,   1.02000000e+02
    1.01000000e+02   1.03000000e+02]]
horizontal:
[[  0.00000000e+00  -1.00000000e+00  -2.00000000e+00 ...,  -9.98000000e+02
   -9.99000000e+02  -1.00000000e+03]
 [            -inf  -2.00000000e+00   0.00000000e+00 ...,  -9.96000000e+02
   -9.97000000e+02  -9.98000000e+02]
 [            -inf  -3.00000000e+00  -1.00000000e+00 ...,  -9.94000000e+02
   -9.95000000e+02  -9.96000000e+02]
 ...,
 [            -inf  -9.99000000e+02  -9.97000000e+02 ...,   1.01000000e+02
    1.00000000e+02   1.02000000e+02]
 [            -inf  -1.00000000e+03  -9.98000000e+02 ...,   1.00000000e+02
    1.02000000e+02   1.01000000e+02]
 [            -inf  -1.00100000e+03  -9.99000000e+02 ...,   1.02000000e+02
    1.01000000e+02   1.00000000e+02]]
diagonal:
[[    0.    -1.    -2. ...,  -998.  -999. -1000.]
 [   -1.     1.     0. ...,  -996.  -997.  -998.]
 [   -2.     0.     2. ...,  -994.  -995.  -996.]
 ...,
 [ -998.  -996.  -994. ...,   101.   103.   105.]
 [ -999.  -997.  -995. ...,   103.   102.   104.]
 [-1000.  -998.  -996. ...,   102.   101.   103.]]
```

```
Directional:
[[ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  3.  2. ...,  0.  0.  0.]
 [ 0.  1.  3. ...,  0.  0.  0.]
 ...,
 [ 0.  0.  0. ...,  2.  1.  3.]
 [ 0.  0.  0. ...,  3.  2.  1.]
 [ 0.  0.  0. ...,  2.  2.  3.]]
best:  112.0
(986, 993)
stopped at:  (17, 32)
Best Score:  119.0
Length:  1247
CCTA-AAACCACTCC——GCAGAA--AAAG--AATA--AG--GCCAAAACACGACTAAAATCGAAAGAC-
ATGACAAGTAAACGAGAAAAGAAA-A-ATA-AA-CGACATACACACTTGTAGGA--A-A--A-ATAA-GAAA-A-
AGGGGGAGACGAAGCAAAGA-AAGGGCAGCTAACCCT-CA-A---GGA-AGAACCAGACA-GAATAAGA--A-
AA---ACCCGAAA-GCCACC-AAA-TGAAA-G-GAC-AATAACACCTAA-GAGCAA--AAT--CAATAA--A--A-
CACCGATCCTC-----C---GAGGAT-AACCA-AGA-GAGACCTAAGAACGAC-A--AG-AAACCAATG--A--A-
A-GA-AA--AAG---AA-A--ATGGA-CATCAGAACGA-CTTAGAA-TGCTGGGAA-AA-AGAAAAATT------
ATAAACGAA-G-G-A-TG-G---G--CATAAATTG-G--AC-GAAG-C-C-A-AGAGATAG-GC-CGA--G--
ATAAAACGGAGAACAATAAG--GGAGAC-C--AT-G-GAGAGC--AAAC--CAACCGCAA-CAAA--TAA-A-
GGGGGGGACAAA-AACAAGACCAAC--CC-AAA-C-TGT-CA-G----A-CA-G-GA-A-GAGC-AATAAC--
CAAGACA-GAA-GAA-G-AAACAGGA--GACAAACA-AC--AT--AA-TA--TA-AGA-GCA-
CCTAGCTAACAAAAAAGA---CCAGCAAACGGATTAAGA-AGAT--AAAGAAA----AC---G--T----AA-
AGAA-C-A-GTC-AAG---GAACAAGCGA----TAATAA-ATG-CAGG-G-AAAAAATGG-G--GA-CAG-
ACGAAG--GAAACAACCA-G-AAATAATCTA-ACGCATCGCAGAAGATGACACTGCGA--GAA-AATACGAGCCGT-
ATACGACAC-A-AAAC--C--G--GGAA-TAA-A-GA-AA---AAAACCATACC-CAA-AA-AGA-ACA-AC-
GCGA-AAGATGAAACGCTCC--C-AAC--TC-G----G-A---TGAG-CAAAGCCGCCAG--GCCAAAA-
AAGAGAACCA--GAGC-AG-AGCGA--AGCTATGG-GT-A--GAAA-AC---ACCCTAAGCGCGGGTAGTAGA-
GACGAAAA-A-TAA-AAAC-AGGC-TGAC-C-
CGAACATAAGAGCCCACACAAGTAGAAGAACGGAAAGAAAACGAAAAGA

CC-AGAAACCA-TCCTAAA--AGAAGGAAAGCAAATAGGAGAA--CAAAACA--A-T--AATC-AAAG-CGAT----
A-TAAA-G-G---AGAAACATA-ACAACCG-C-TACAC-C----A--ATCACACCACA-AAGGAAAGATA-----
AG-C---GCAAAGAG-A----AG-T-ACCCTGCATACCT--AC--AACCA-A-AT-AA-AAGAGGAGAACTGA---
GAAACGCCACCAAAAC--AAACGTGACG-AT-A-A-CTAATGA--AACGAATGA-AA-AAGGAGGAT-A--GA-
CCTCAAATTCAAAGA-GATGAA-CAT-GAC-AG--CTAA-AA-GACAACGAGCAAA--AATGCTAGGAGAC-
ATAACCAAGCTAAAGACCA-GGACC--C--AACGACC---GAAC-GC----AAGAAT-G-AAAATTAGCCCCA-
AAA--AACGCGCAC-GAGAAAGAAC-TAAA--GAGCCACA-AAGACACAATAGA-A-AGTGCTCGACGGACA-
AAAA---A-AA-AA-AAGAA-GAGACACAAATAGA-A-A-CAAAAACAA-AA--G-AAGCAAACG-AATAT------
```

```
GACAAAG-A-AA-ACCAACTACCAAAAGCA-GTACATGACACATCATGC-ACAC-AGCGAA-AACAACAA-A-
ATGAAC-AACGAAAACA--ACC-A-AAA-AGACGAATCGAAC-AGG-AGAGAGG-ATCC---C---
CAAAAAAGAGGGCC--C-AAC----TAAGACA-ATGCAAAGAAACGCGACAAAGCCTCGCCAACAGAATCAAC--
CAAAGCATGAAC-AGC-ACTTTTAA-AACATGT--GGCGC-------GGCGTCGAGCAGTACG--GTT----CAA--
ATGCAAA-AAT-TACA---A----A-AA-A-GACA-TGC-ACTGAAC----C---CCGTAA-A-GA-
ACGAGAAACTTCAAGAAGGAAG-AAGAC-ATAAGCCAAAA--A-ACCA-AATAATAGACACAG-CT-CGAG-A-A--
AAA-GC-CCAACAAACGA-CAGAAAAGAAGGG-GAGT--AAG--G--AGAAG-CAAAAC---AGAA-CAGGG-
GCGAGAA-CGACT-GC---GGAGTAATCGAAAGACATGA--C-AA------GTA--A-ACGA-
GAAAAGATTAATAAACGA--CAT-ACACA--AAC--AA-A----A-ACAA-TAG-A-AAC-GAAA-AAAA--AAAA-
A
```

Question 2:

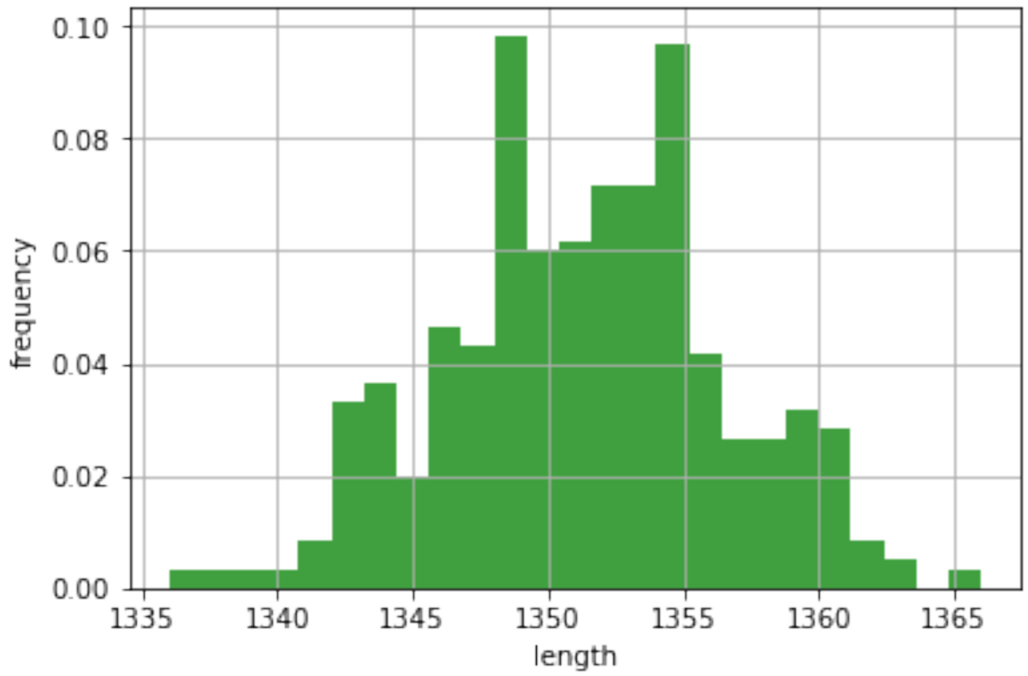
```
#now let's make a nice random DNA generator
#imports

import random



#inputs

numberSeq = 1000
sizeSeq = 1000


seqs = []


seqCt = 0

#nucleotide counts
aCt=0
tCt=0
cCt=0
gCt=0
```

```
while seqCt < numberSeq:
    nucCt = 0
    currentSeq = ""
    while nucCt < sizeSeq:
        r = random.random()
        if r<(1/4):
            currentSeq = currentSeq + "A"
            aCt+=1
        elif r <(1/2):
            currentSeq = currentSeq + "T"
            tCt+=1
        elif r <(3/4):
            currentSeq = currentSeq + "C"
            cCt+=1
        else:
            currentSeq = currentSeq + "G"
            gCt+=1
        nucCt += 1


    seqs.append(currentSeq)
    seqCt+= 1

print (seqs)
print ("Nuceotide freq: A: ", aCt, " T: ",tCt, " C: ", cCt, " G: ", gCt)


randDNAcount = 0
p1 = []
p2 = []
while randDNAcount < (len(seqs)/2):

    p2.append(getLocAL(seqs[randDNAcount], seqs[len(seqs)-1-randDNAcount], 1, -30, -20))
    print(randDNAcount)
    randDNAcount+=1


    import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
plt.xlabel('length')
plt.ylabel('frequency')
plt.grid(True)


n, bins, patches = plt.hist(p2,25, normed=1, facecolor='green', alpha=0.75)
```

```
Nuceotide freq: A:  250001  T:  249952  C:  249990  G:  250057
```

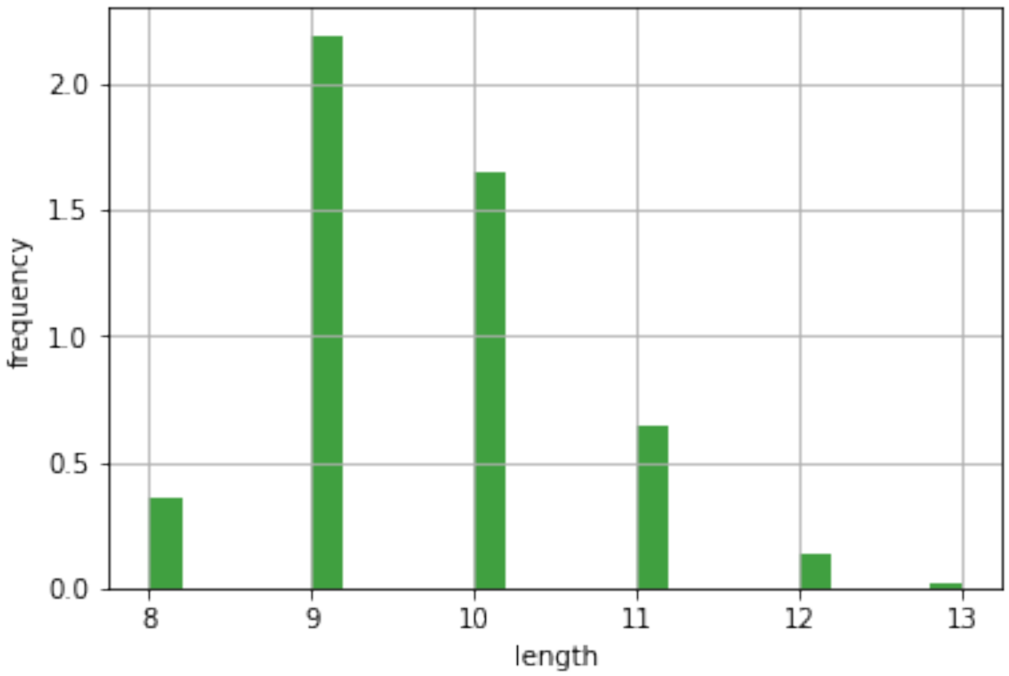
seems random enough. there's a more or less equal amout of all 4 bases.

Running the two parameters through, i find that with -20 mismatch and 0 indel penalty we have a pretty natural looking set that peaks out around 1348 to 1355



the -30 indel penalty, however, leads to a strict length of 1000 in every run.

and then i realized: oh wait, i set the indel to +20 instead. I'm stupid: running it again at -20:

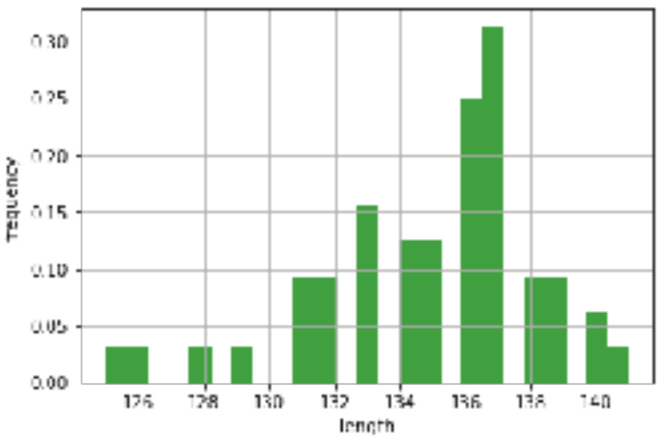


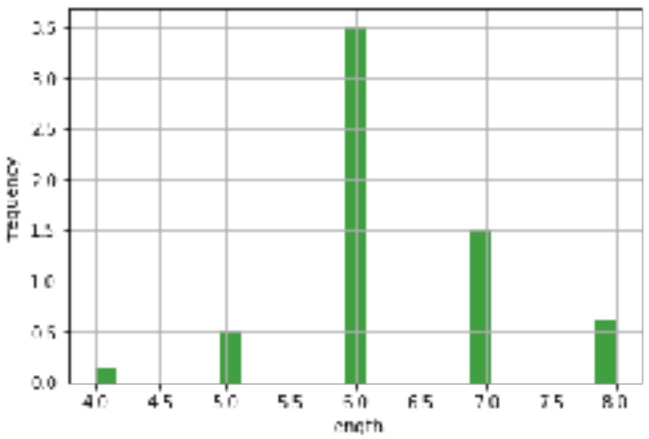
That makes much more sense. suddenly we're seeing a lot more zeroes with the occasional 1s and 2s and very few 3s and 4s.

The lengths of the optimal local alignments are very different. This is because with no indel penalty, the alignments can just spread out indefinitely until a matching nucleotide is found, and then move on. whereas with a high indel penalty, we're effectively forced to find exact matches between two random dna strands: something a lot less likely.

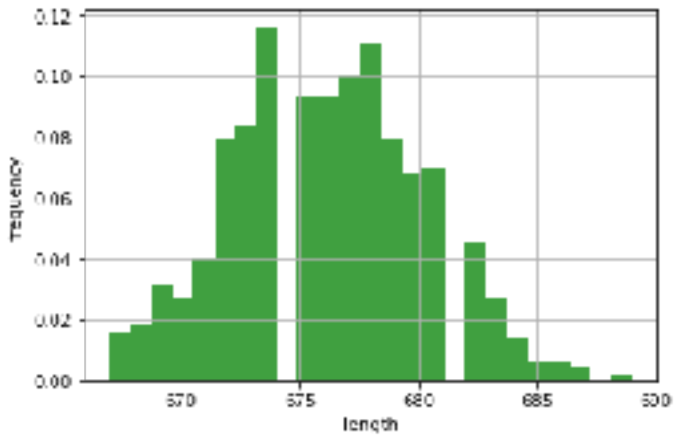
trying out different values all give very similar results. With lp1 being around 1.35*n while lp2 peaks around log(n)*3. In general other histograms looked like this with varying lengths:

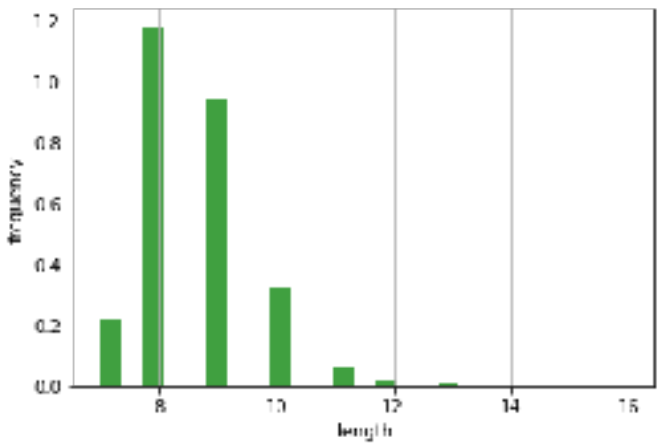
at 100:





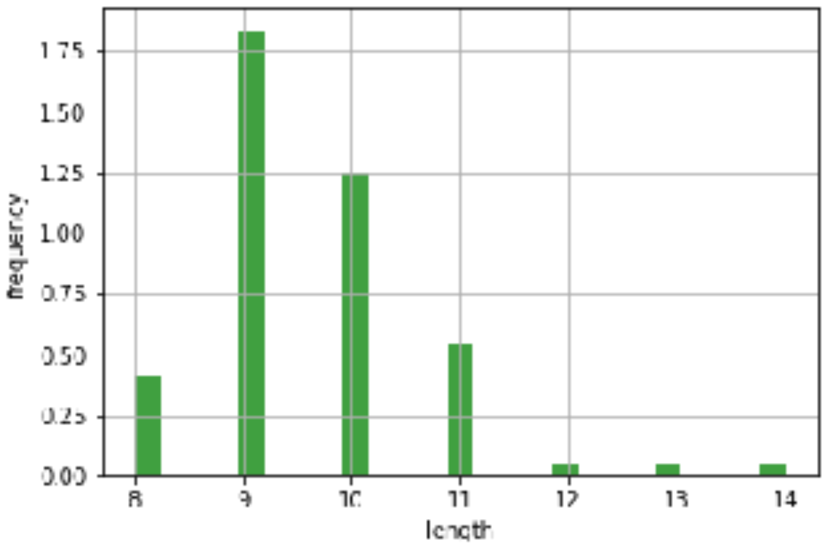
at 500:



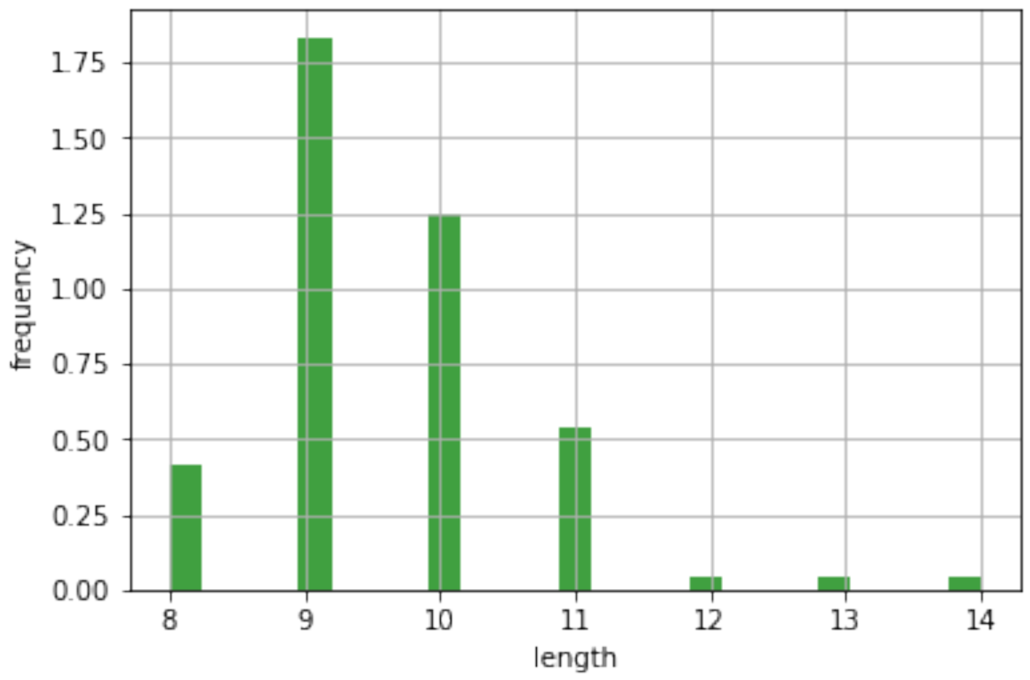


question 3:

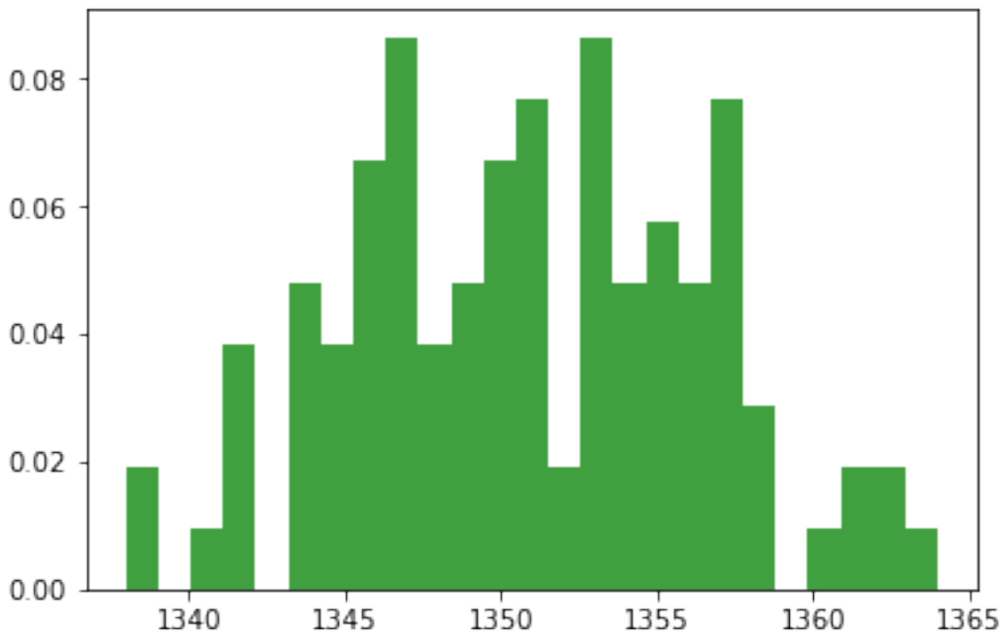
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
plt.xlabel('length')
plt.ylabel('frequency')
plt.grid(True)

n, bins, patches = plt.hist(p1,25, normed=1, facecolor='green', alpha=0.75)
```



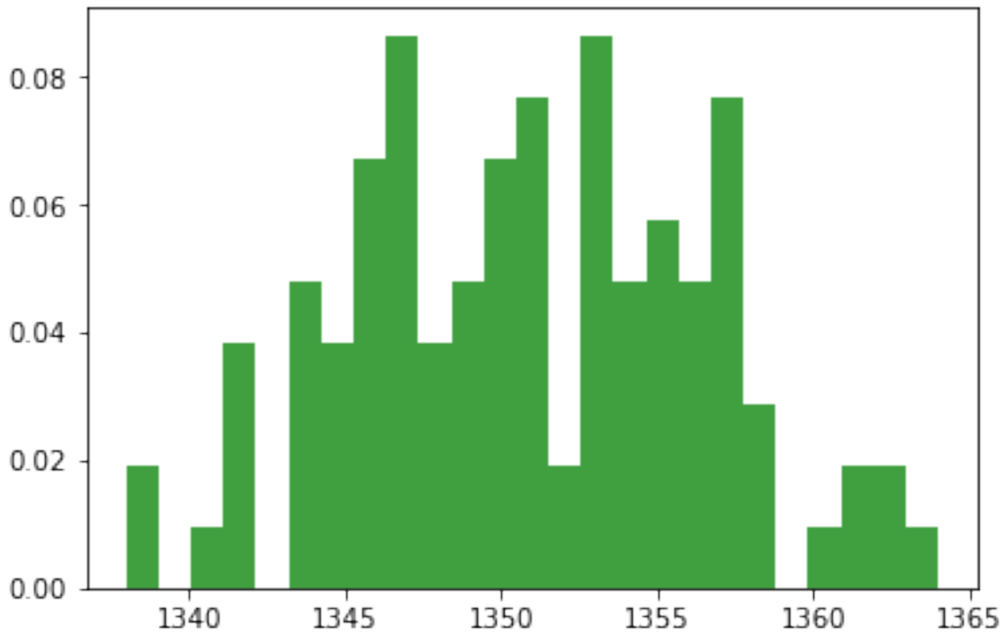
```
n, bins, patches = plt.hist(p2,25, normed=1, facecolor='green', alpha=0.75)
```



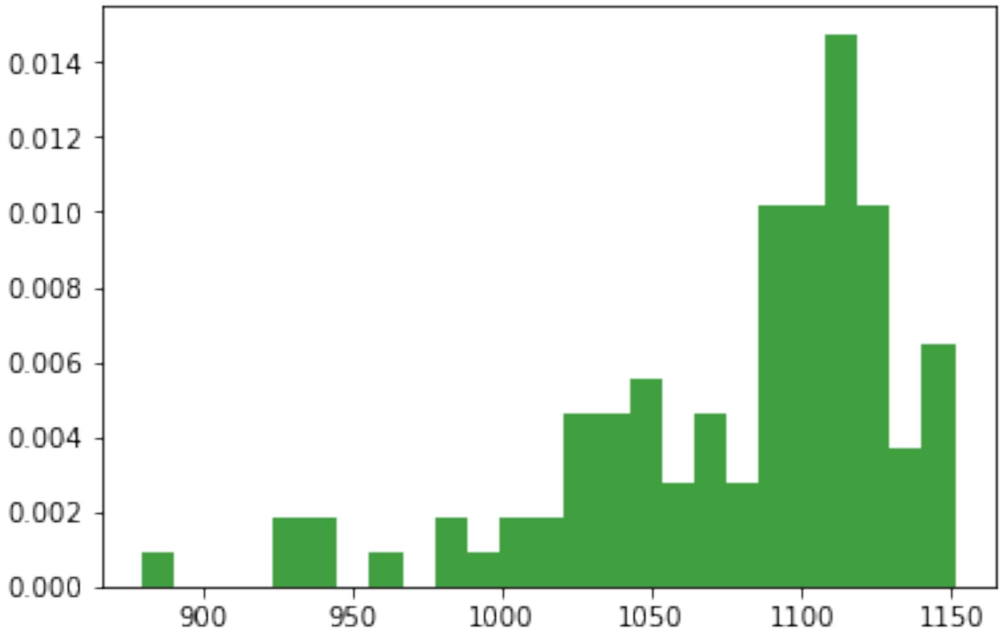
```
n, bins, patches = plt.hist(p3,25, normed=1, facecolor='green', alpha=0.75)
```



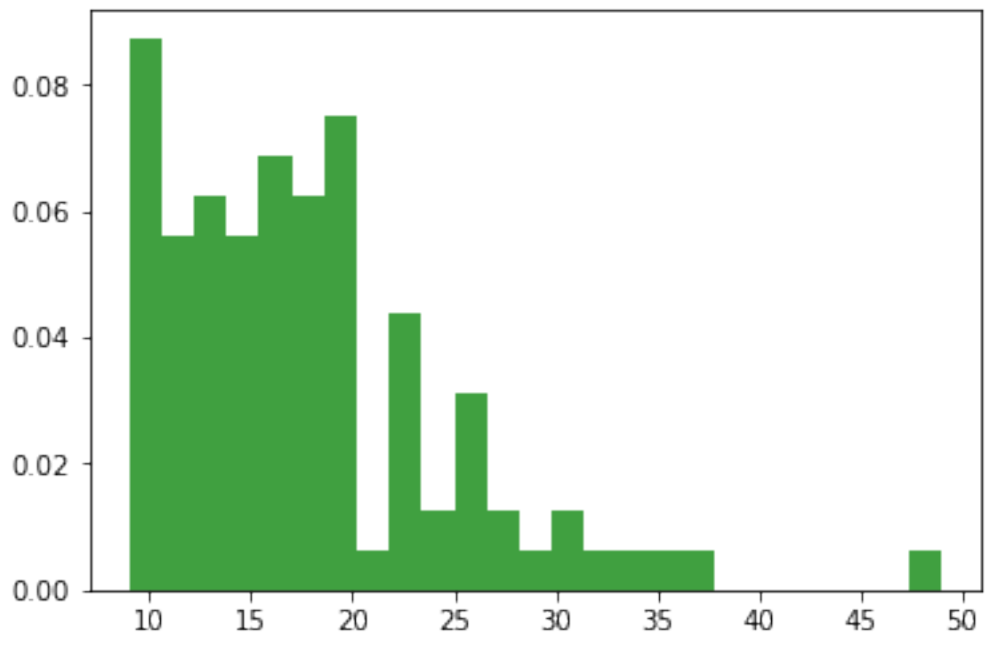
```
n, bins, patches = plt.hist(p4,25, normed=1, facecolor='green', alpha=0.75)
```



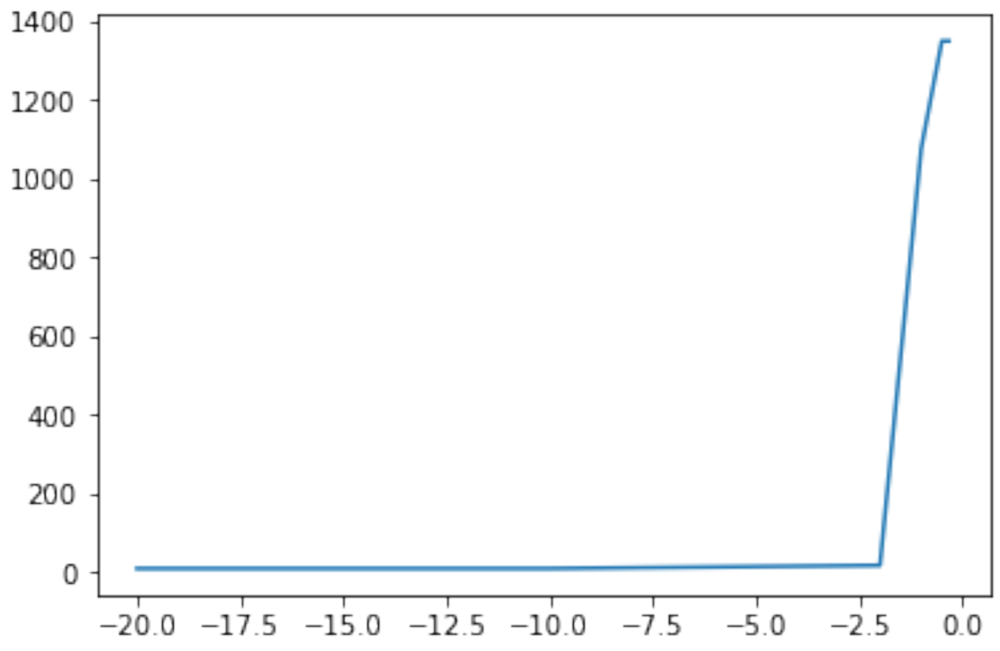
```
n, bins, patches = plt.hist(p5,25, normed=1, facecolor='green', alpha=0.75)
```



```
n, bins, patches = plt.hist(p6,25, normed=1, facecolor='green', alpha=0.75)
```



```
code used:
randDNAcount = 0
p1 = []
p2 = []
p3 = []
p4 = []
while randDNAcount < (len(seqs)/2):
    p1.append(getLocAL(seqs[randDNAcount], seqs[len(seqs)-1-randDNAcount], 1,
-20, -20))
    p2.append(getLocAL(seqs[randDNAcount], seqs[len(seqs)-1-randDNAcount], 1,
-10, -10))
    p3.append(getLocAL(seqs[randDNAcount], seqs[len(seqs)-1-randDNAcount], 1,
-.5, -.5))
    p4.append(getLocAL(seqs[randDNAcount], seqs[len(seqs)-1-randDNAcount], 1,
-.33, -.33))
    print(randDNAcount)
    randDNAcount+=1
randDNAcount = 0
p5 = []
p6 = []

while randDNAcount < (len(seqs)/2):
    p5.append(getLocAL(seqs[randDNAcount], seqs[len(seqs)-1-randDNAcount], 1,
-1, -1))
    p6.append(getLocAL(seqs[randDNAcount], seqs[len(seqs)-1-randDNAcount], 1,
-2, -2))
    print(randDNAcount)
    randDNAcount+=1

means: to plot:

means =
[numpy.mean(p1),numpy.mean(p2),numpy.mean(p6),numpy.mean(p5),numpy.mean(p3),n
umpy.mean(p4)]
plt.plot([-20, -10, -2, -1, -.5,-.33], means)
x axis is the mismatch = indel score
y axis is the mean length.
```



```
question 4:


#python locAL.py <seq file> -m <match> -s <mismatch> -d <indel> -a
#python locAL.py testseqs.txt -m +1 -s -1 -d -1 -a

import sys, getopt, numpy
```

```
arguments = ["locAL", "p4seqs.txt", "-m", "1", "-s","-2", "-d", "-2", "-a"]


file = arguments[1]
matchScore = int(arguments[3])
mismatchScore =  int(arguments[5])
indel = int(arguments[7])
findA = False
if '-a' in arguments:
    findA = True
    #print('found -a')

print ('Number of arguments:', len(arguments), 'arguments.')
print ('Argument List:', str(arguments))
print ('file:', str(file))
print ('matchScore:', str(matchScore))
print ('mismatchScore:', str(mismatchScore))
print ('indel:', str(indel))
print ('findA:', str(findA))


# In[10]:

data = open(file, "r")


check = False
counter = 0
seq1 = []
seq2 = []
for line in data:
    if 'seq' in line:
        check = True
    elif check == True and counter == 0:
        for char in line:
            if char != '\n':
                seq1.append(char)

        counter +=1
    elif check == True and counter == 1:
        for char in line:
            if char != '\n':
                seq2.append(char)
# In[ ]:




# In[11]:

#to find the max score

#initialize all them

#vert = insertion
vert = numpy.empty((len(seq2)+1, len(seq1)+1))
vert[:] = 0

d = 1
while d < len(vert):
    vert[d][0] = indel*d
    d+=1

d=1
while d < len(vert[0]):
    vert[0][d] = -float("inf")
    d+=1

vert [0][0] = 0



#hori = deletion
hori = numpy.empty((len(seq2)+1, len(seq1)+1))
hori[:] = 0
d=1
while d < len(hori[0]):
    hori[0][d] = indel*d
    d+=1
```

```
d=1
while d < len(hori):
    hori[d][0] = -float("inf")
    d+=1
hori[0][0]=0

#diag = match or mismatch
diag = numpy.empty((len(seq2)+1, len(seq1)+1))
diag[:] = 0
d=1
while d < len(diag[0]):
    diag[0][d] = indel*d
    d+=1
d=1
while d < len(diag):
    diag[d][0] = indel*d
    d+=1
diag [0][0] = 0



#1 = Vert, 2 = Horiz, 3 = diag

dire = numpy.empty((len(seq2)+1, len(seq1)+1))
d=1
while d < len(dire[0]):
      dire[0][d] = 2
      d+=1
d=1
while d < len(dire):
      dire[d][0] = 1
      d+=1

print ("Vertical")
print (vert)

print ("Horizontal")
print (hori)

print ("Diagonal")
print (diag)


print ("Direction")
print (dire)


#run the script until we good


# In[ ]:

print(dire.shape)


# In[ ]:


# here we going to loop through the whole thing and go from top left to
bottom right


#let's make a variable to keep track of the biggest score value:
maxScore = 0
bestLoc = (0,0)
# we want to iterate 1-10 in the 3 matrices. This is the nested for loop
i = 1
while i < len(diag):
    if i%100==0:
        print('filling in:', i)
    j=1
    while j < len(diag[i]):
        #print ('current i and j: ', i , ' ', j)
        #we gotta manipulate each matrix we're working with

        #Vertical
        a = vert[i-1][j]
        b = diag[i-1][j]

        if a>=b:
```

```
    vert[i][j] = a + indel
elif b>=a:
    vert[i][j] = b + indel



#Horizontal
a = hori[i][j-1]
b = diag[i][j-1]

if a>=b:
    hori[i][j] = a + indel
elif b>=a:
    hori[i][j] = b + indel

#diag



a = vert[i][j]
b = hori[i][j]

#print((seq1[j-1],seq2[i-1]))

if(int(seq1[j-1]==seq2[i-1]) ==0):
    cScore = mismatchScore
else:
    cScore = matchScore

c = diag[i-1][j-1] + cScore



if a>=b and a>=c:
    dire[i][j] = "1"
    diag[i][j]=a
if b>=a and b>=c:
    dire[i][j] = "2"
    diag[i][j]=b
if c>=a and c>=b:
    dire[i][j] = "3"
    diag[i][j]=c
if diag[i][j]<0:
```

```
                dire[i][j] = 0
                diag[i][j] = 0

            if(diag[i][j] >= maxScore):
                maxScore = diag[i][j]
                bestLoc = (i,j)
            j+=1

        i+=1

print('vertical: ')
print(vert)
print('horizontal: ')
print(hori)
print('diagonal: ')
print(diag)

print('Directional: ')
print(dire)


print('best: ', maxScore)
print(bestLoc)



# run this if -a is on
#reset directional borders to zero:
d=1
while d < len(dire[0]):
      dire[0][d] = 0
      d+=1
d=1
while d < len(dire):
      dire[d][0] = 0
      d+=1



# let's write a function to find the local alignment
```

26

```
ali1 = ""
#print(seq1)
#print(seq1[bestLoc[1]-1])

#remember seq2 is the y value but is presented first in the coordinates
ali2 = ""
#print(seq2)
#print(seq2[bestLoc[0]-1])




k = 0
current = bestLoc
count = 0
count2 = 0
while k == 0:
    #print ("currentLoc: ", bestLoc, " currentDire ",dire[bestLoc])

    #on zero we stop
    count +=1
    if count >=10:
        print ('backtracking: ', count2)
        count2+=1
        count = 0

    if dire[bestLoc] == 0:
        print("stopped at: ", bestLoc)
        k=1
    #on 1 we go up. so i changes but j stays the same
    elif dire[bestLoc] == 1:
        bestLoc = (bestLoc[0]-1, bestLoc[1])
        ali1 = ali1 + "-"
        ali2 = ali2 + str(seq2[bestLoc[0]])
    #on 2 we go left so j changes but i stays constant
    elif dire[bestLoc] == 2:
        bestLoc = (bestLoc[0], bestLoc[1]-1)
        ali2 = ali2 + "-"
        ali1 = ali1 = ali1 + str(seq1[bestLoc[1]])
    #on 3 both change, yay!
    elif dire[bestLoc] == 3:
        bestLoc = (bestLoc[0]-1, bestLoc[1]-1)
        ali1 = ali1 + str(seq1[bestLoc[1]])
        ali2 = ali2 + str(seq2[bestLoc[0]])


print('Best Score: ', maxScore)
print ('Length: ', len(ali1))
print (ali1[::-1])
print (ali2[::-1])
```

```
stopped at:  (3590, 21171)
Best Score:  51.0
Length:  57
GACCTCATCCCGGATTAGGATACTTCACGCTTACGAACTCTCAGGGAC--AGTTCCG
GACCTCATCCCGGATTAGGATACTTCACGCTTACGAACTCTCAGGGACAGAGTTCCG
```

5.Lolrip

6.
I used mostly python on Jupyter. Some more memory intensive computations are done on terminal to avoid chrome. I had some help from Dominik Stec and the TAs. This HW took around 30 hours.