HW 2

Mathematics 127
Mathematical and Computational Methods in Molecular Biology

Fall 2002
UC Berkeley, CA

Nasser M. Abbasi

Fall 2002 Compiled on August 2, 2022 at 8:08am [public]

Contents

1 Problems

Problem Set 2 (due Thursday September 26)
MATH 127: Mathematical and Computational Methods in Molecular Biology

Please work on the starred problem alone.

Problem 1

Estimate the number of nucleosomes used for supercoiling in human chromosome 17.

Problem 2*

Show (in full detail) that the directed writhe of a knot is invariant under Reidemeister moves of type 2 and 3.

Problem 3

Given two sequences of length n, and a scoring scheme $1, -1, -2$ (for match, mismatch gap), let the score of the optimal global alignment be G and the optimal local alignment be L.

a) Prove that $L \geq G$, and find an example where strict equality holds.

b) What is the maximum value of $L - G$?

Problem 4

If the Jones polynomial of a knot is $X(L)$ what is the Jones polynomial of the mirror image of the knot?

Problem 5

Compute the Jones polynomial of the trefoil knot. Then show that the trefoil knot and its mirror image are not equivalent (use problem 4).

Problem 6 (optional)

Given sequences of lengths n and m what is the maximum number of optimal alignments (with the same score) that can result from a scoring scheme of 1 for a match, a for a mismatch and b for a gap.

Problem 7 (optional)

a) Implement the Needleman-Wunsch algorithm where the parameters are an input.

b) Use the scoring scheme of 1 for a match, and 0 for a mismatch and gap to find the average length of the longest increasing subsequence in a permutation of length n by simulation.

2 Problem 1

Problem 1
Math 127
Nasser Abbasi

+10

total : [illegible]

49/50

Link number is an invariant.

For relaxed DNA for chromosom 17 (97 million bases) From NCBI web page.

$$L_K = \frac{97\times 10^6}{10.5} + W_r$$

number of Bases per one Full turn

but $W_r = 0$ for relaxed DNA (because no supercoiling)

So $$\boxed{L_K = \frac{97\times 10^6}{10.5}}$$ (round to an integer since L_K must be an integer)

when supercoiling, number of bases per turn is 10 instead of 10.5.

So $$L_K = T_w + W_r$$

so $$\frac{97\times 10^6}{10.5} = \frac{97\times 10^6}{10} + W_r$$

so $W_r \approx -461904$

so there are approx $\boxed{461{,}904 \text{ nucleosomes}}$.

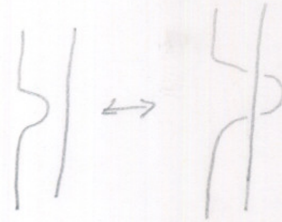
To confirm There are 200 bp per nucleosomes (146 bp wrapped around 4 histons, and 56 linking DNA)

so $$\frac{97\times 10^6}{200} = 485{,}500$$ (close to above calculations. This propably means not all the chromosom is constructed to have nucleosomes every where?) ok.

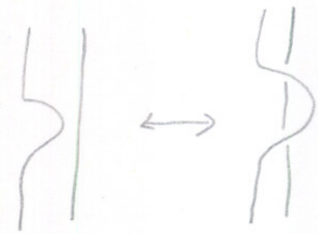
problem 2
HW 2 +0
Nassar Abbasi

4 cases here | 4 cases here.

move type2 is

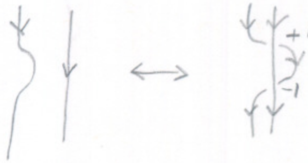


OR



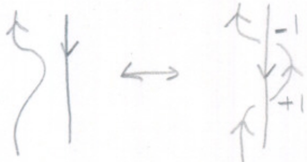
to show directed writhe is invariant, I calculate writhe before and after the move. I have to look at 4 cases per each subtype.

wr = ϕ



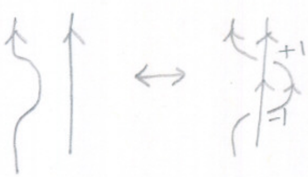
wr = +1 -1 = 0

wr=0

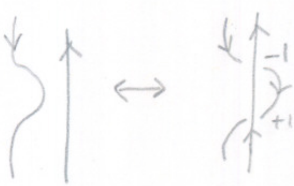


wr = -1 +1 = 0

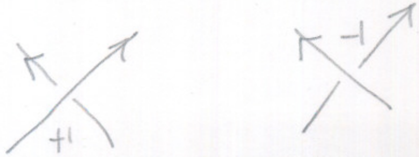
wr=0



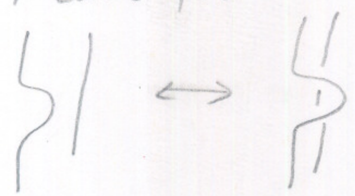
wr = +1-1 = 0



wr = -1+1 = 0



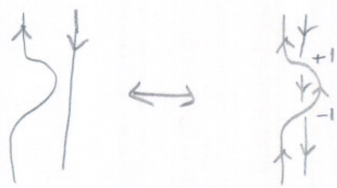
now look at the 4 cases for



wr=0

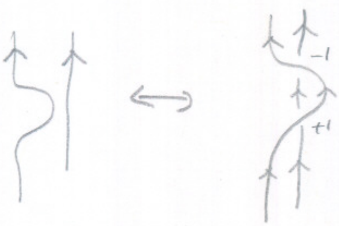
wr = $-1+1=0$

wr=0



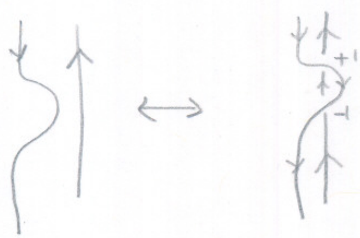
wr = $+1-1=0$

wr=0



wr = $-1+1=0$

wr=0

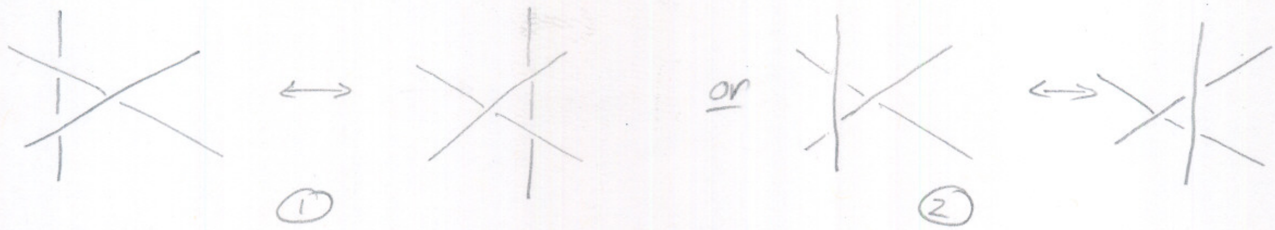


wr = $+1-1=0$

hence, under move type 2, directed writhe is invariant.

now I look at move type 3 →

move type 3 is



looking at ① now:
possible combinations are 8

w=1

wr=1

w=-3

wr=-3

wr=1

wr=1

wr=1

wr=1

wr=1

wr=1

wr=1

wr=1

wr=1

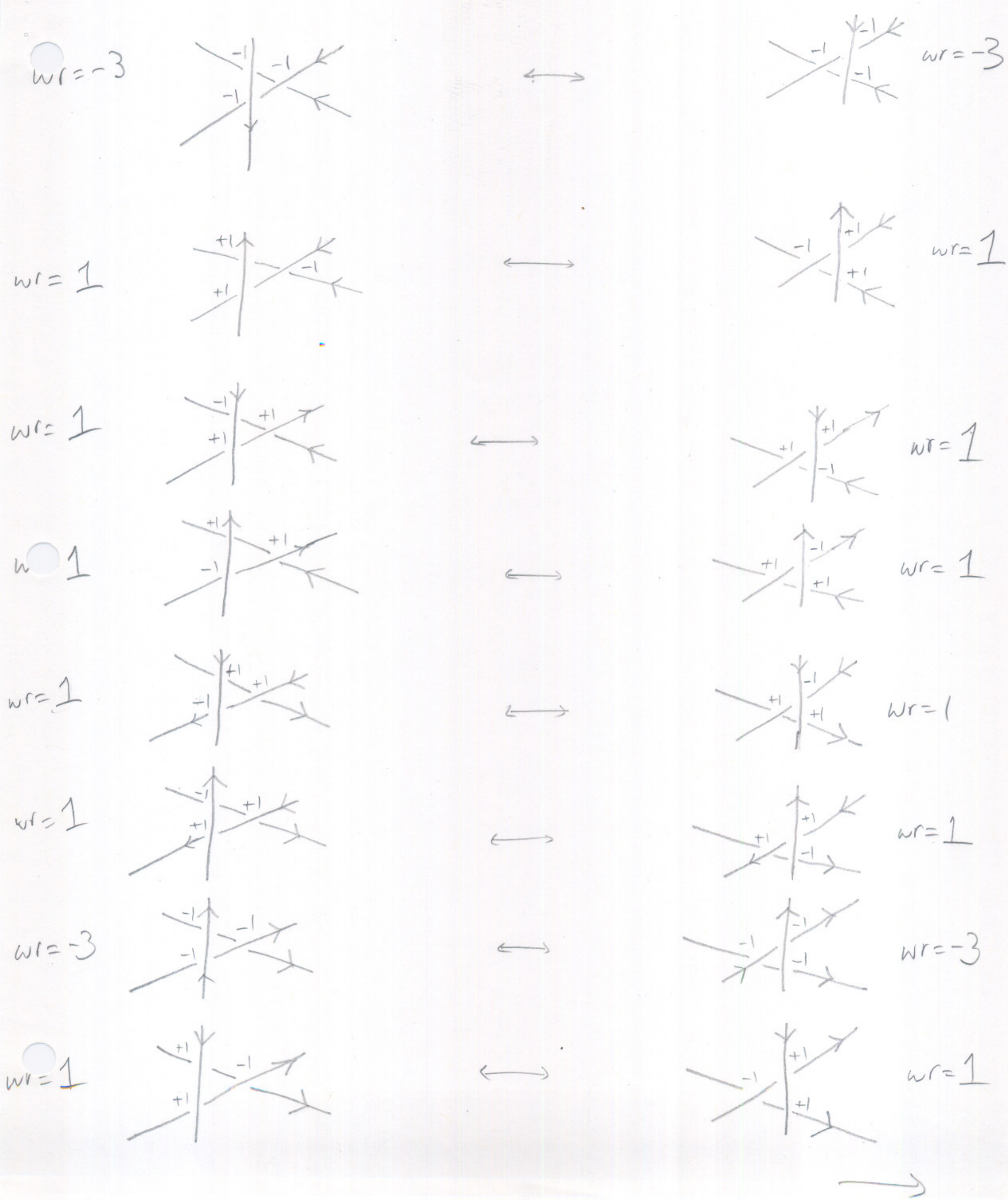
wr=1

wr=-3

wr=-3

→

so move type 3 part ① is invariant. now look at part ②



so this shows that "1" move type 3, part (2) and

so I showed for type 2 and 3 that directed writhe is invariant. QED

3 Problem 2

problem 3
HW 2
Nasser Abbasi ~~[illegible]~~ 8

The score G is the sum of the scores at each cell in the matrix along the global alignment path.
The score L is the sum of the scores at each cell in the matrix over the local alignment path. there is one global alignment path in the matrix, but many Local alignments paths.

Now, a Local alignment path will stop when we hit on a cell in the matrix that have score of ϕ. so local alignment score must be ≥ 0 all the time to continue over that path.

let the sequences be $a_1 a_2 \dots a_n$, $b_1 b_2 \dots b_n$.
There are these extreme cases

Case 1 $a_i = b_i$ for all $i = 1..n$ $\Rightarrow L = n$ and $G = n$ (since a match score $=1$)
So in this case $L = G$.

Case 2 $a_i \neq b_i$ for all $i = 1..n$ $\Rightarrow L = \phi$ and $G = -n$
this is because L score must be ≥ 0 by definition.

Case 3
$a_1 a_2 \dots a_n$ - - - - - - -
- - - - - - - $b_1 b_2 b_3 \dots b_n$ $\Rightarrow L = \phi$ and $G = 2n \times (-2)$
$= -4n$
since Gap has -2 score.

all other possible alignments must a G score greater than $-4n$ and less than n, and must have an L score greater than zero and less than n.

$\longrightarrow$

this means that $L \geq G$.

A case for strict eqvality is when $a_i = b_i$ $\quad i = 1 .. n$

(b) maximum value of $L - G$ is when L is max and G is minimum. this is $-4n$ for this case

$a_1 a_2 a_3 \ldots\ldots a_n$ — — — — — — — — ← gaps.

— — — — — — — — $b_1 b_2 b_3 \ldots\ldots b_n$

↑ gaps.

or somethis as

$a_1 - a_2 - a_3 - a_4 - a_n$

$- b_1 - b_2 - b_3 - b_4 - b_n$ $\Rightarrow$ $L = \phi$, $G = -4n$

↑ gap.

[illegible] , 2 $L - G = \frac{8}{5} n$

Dr: while thinking about this, I came with this representation which I found usefull.

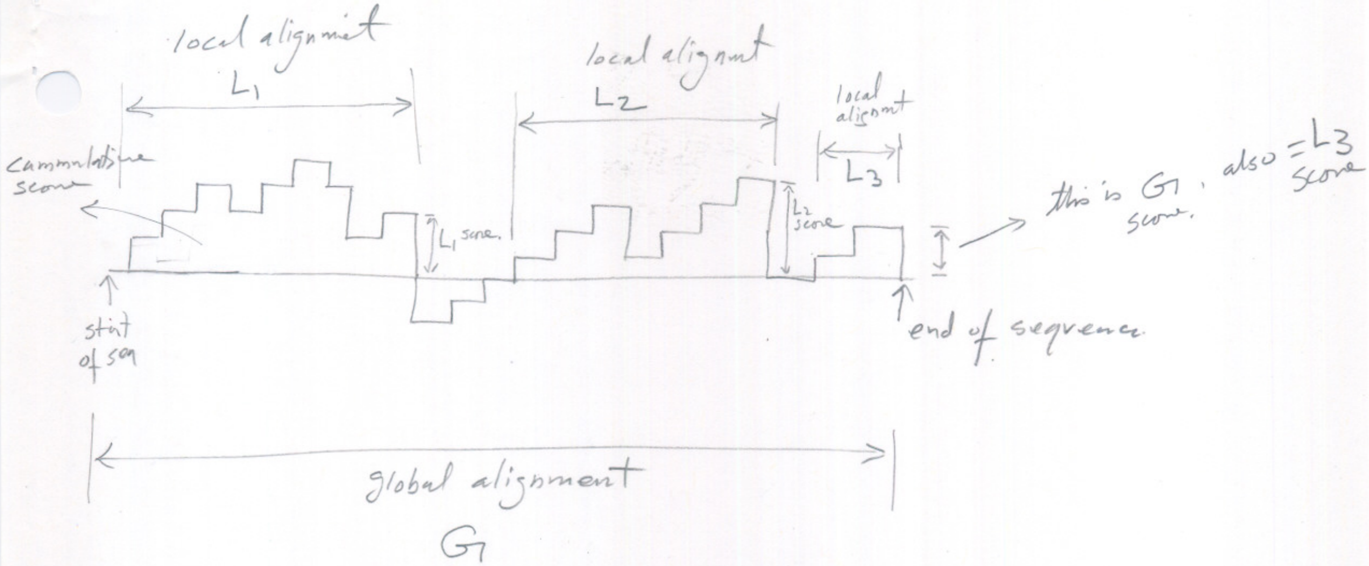
draw the cummulative score along the path as a stair steps.

local alignmemt paths are the steps 'over' the zero level.

the best local alignment is the continous steps with the area over the zero level. the L score is the height of the Last step.

The G score is the height of the last step that ends at the end of the sequence

⟶



From this diagram I see that $G \leq L_{max}$.

since if $G = L_3$, and L_3 happened to be the largest L of all L's, then $G = L$. otherwise L_1 or L_2 are larger than L_3, and hence G must be less than L_1 and L_2 as well.

This shows that $G \leq L$ also.

4 Problem 3

Did not do.

5 Problem 4

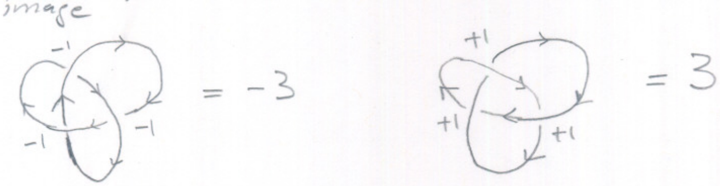
Problem 4
HW 2
Nasser Abbasi MATH 127 +10

if Jones polynomial is $X(L)$, what is the Jones polynomial of the mirror image of the Knot ?

$$X(L) = \left(-A^{3}\right)^{-w(L)} \langle L \rangle$$

where $w(L)$ is the directed writhe of projection of knot.

to see the effect, let me look at trefoil knot and its mirror image



So for mirror image we have directed wirth of different sign. what about $\langle L \rangle$ of image ?

looking at bracket polynomial rules

rule 1 $\langle O \rangle = 1$ ⟶ not affected.

rule 2 $\langle \times \rangle = A \langle)(\rangle + A^{-1} \langle \asymp \rangle$

For mirror image

$\langle \times \rangle = A \langle \asymp \rangle + A^{-1} \langle)(\rangle$

rule 3 $\langle L \cup O \rangle = C \langle L \rangle$ ⟶ not affected.

So only rule 2 can change the $\langle L \rangle$ polynomial between Knot and its mirror image.

I see that, from rule 2, $A \to A^{-1}$ in the mirror image. This means exponent changes sign ⟶

so, for bracket polynomial $\langle L \rangle$ in A, the mirror image will have each exponent of A having opposit sign.

For example, if $\langle L \rangle = A^{-5} + A^{6}$, then $\langle L \rangle' = A^{5} + A^{-6}$ where $\langle L \rangle'$ is the bracket polynomial for the mirror image.

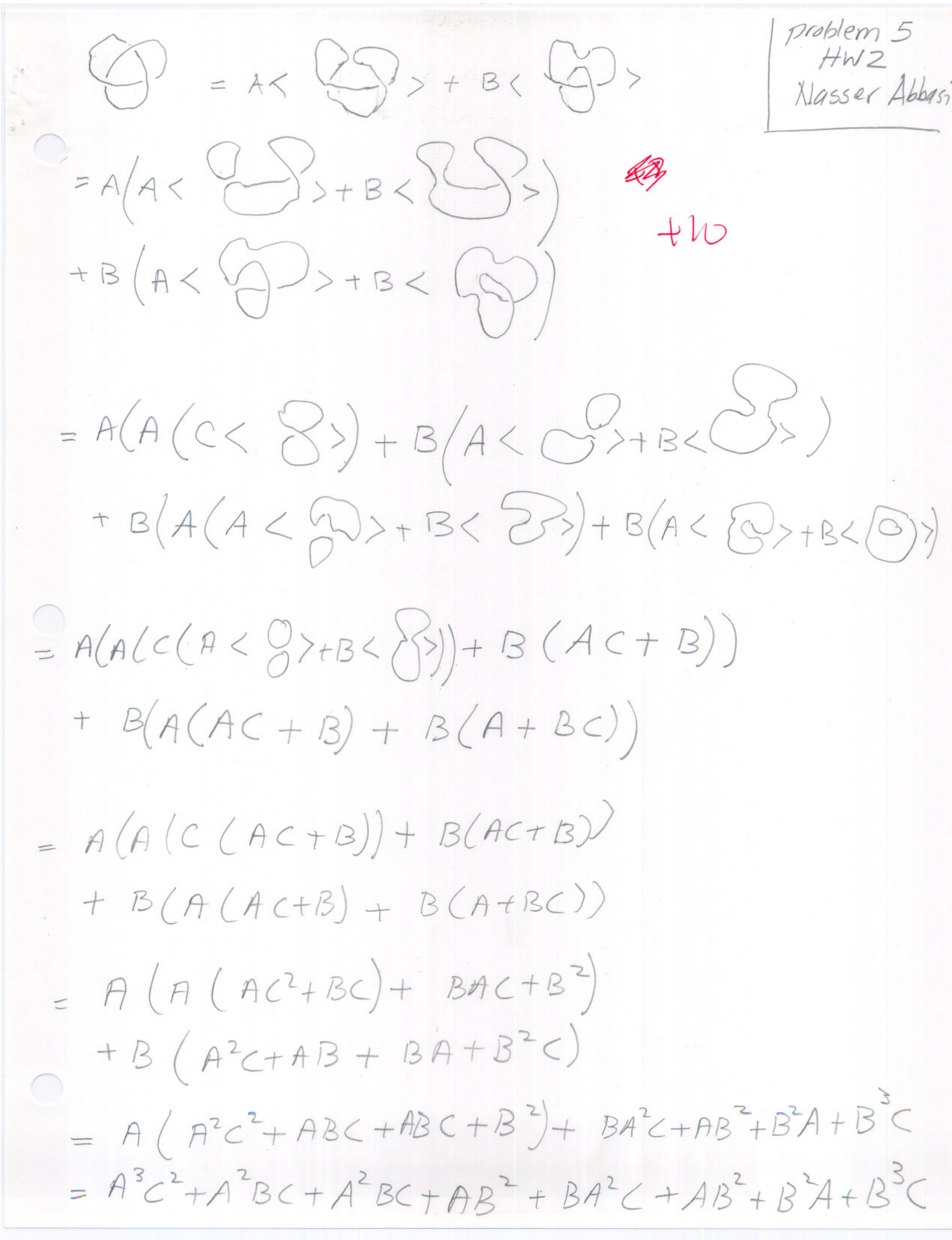
<u>to summarise</u>

if $X\langle L \rangle = (-A^{3})^{-\omega(L)} \langle L \rangle$

Then $X\langle L \rangle' = (-A^{3})^{\omega(L)} \langle L \rangle'$

when $\omega(L)$ is directed withe of L projection, and $\langle L \rangle'$ is the same as $\langle L \rangle$ except exponents of A are of opposit sign.

6 Problem 5


problem 5
HW2
Nasser Abbasi

+10

$+ B(A(AC+B) + B(A+BC))$

$= A(A(C(AC+B)) + B(AC+B))$
$+ B(A(AC+B) + B(A+BC))$

$= A(A(AC^2+BC) + BAC+B^2)$
$+ B(A^2C+AB + BA+B^2C)$

$= A(A^2C^2+ABC+ABC+B^2) + BA^2C+AB^2+B^2A+B^3C$

$= A^3C^2+A^2BC+A^2BC+AB^2+BA^2C+AB^2+B^2A+B^3C$

7 Problem 6

HW #2
problem 6 (optional)
Nasser Abbasi

For $n=1$, $m=1$
a_1, b_1 the possible alignments are

①	②	③
a_1 —	— a_1	a_1
— b_1	b_1 —	b_1

i.e. 3 alignments

score of ① is $-2b$
score of ② is $-2b$
score of ③ is 1 or $-a$

so for $n=1$, $m=1$ max number of optimal alignments with same score is $\boxed{1}$

for $n=2$, $m=1$

$a_1\ a_2$	$a_1\ a_2$	— a_1 a_2	a_1 — a_2	$a_1\ a_2$ —
b_1 —	— b_1	b_1 — —	— b_1 —	— — b_1
score $= 1-2b$ or $-a-2b$	$-2b-a$ or $-2b+1$	$-2b-2b-2b = -6b$	$-2b-2b-2b = -6b$	$-2b-2b-2b = -6b$

so in this case we have 2 alignments with score $1-2b$, and two alignments with score $-(2b+a)$ and 3 alignments with score $-6b$.
so here max is $\boxed{3}$

Need to generalize this recursively for n, m.
Sorry, no time to complete. hard problem!

8 Problem 7

Problem 7
HW2
MATH 127
By Nasser Abbasi
Sept 26, 2002.

Part (a):

Implemented needleman-Wunsch algorithm. Source code is below (also in floppy attached).

To test, I used the example given in the lecture to verify the output is correct.
When running the program, it formats the output showing the sequences and the scoring matrix.

```
K» help nma_problem_7_part_a

 function  R=nma_problen7_part_a(s1,s2,match,mismatch,gap)

  solves problem 7, HW 1 for MATH 127
  by Nasser Abbasi
  sept 25, 2002.

  implements Needleman-Wunsch algorithm

  INPUT
   s1: is one sequence.  example S1=['g' 'a' 't' 'c' 'g'];
   s2: is the second sequence.
   match:  score for matching bases. such as 1
   mismatch: score for mismatch. such as -1
   gap: penatly factor for gaps. such as -2

   NOTE:  S1 is the row sequence at the top, and S2 is column sequence at left.
```

This is an example:

```
K» S1=['G' 'A' 'T' 'T'];
K» S2=['G' 'A' 'T' 'A' 'C' 'G' 'T'];
K» match=1;
K» gap=-2;
K» mismatch=-1;
K»
K» nma_problem_7_part_a(S1,S2,match,mismatch,gap)
```

		G	A	T	T
	0	-2	-4	-6	-8
G	-2	1	-1	-3	-5
A	-4	-1	2	0	-2
T	-6	-3	0	3	1
A	-8	-5	-2	1	2
C	-10	-7	-4	-1	0
G	-12	-9	-6	-3	-2
T	-14	-11	-8	-5	-2

```
K»
```

This is another example:

```
K» S1=['T' 'T' 'T' 'C' 'G' 'T' 'A' 'G' 'T' 'T'];
K» S2=['T' 'T' 'C' 'C' 'G' 'A' 'A' 'G' 'C' 'T'];
K» nma_problem_7_part_a(S1,S2,match,mismatch,gap)
```

		T	T	T	C	G	T	A	G	T	T
	0	-2	-4	-6	-8	-10	-12	-14	-16	-18	-20
T	-2	1	-1	-3	-5	-7	-9	-11	-13	-15	-17
T	-4	-1	2	0	-2	-4	-6	-8	-10	-12	-14
C	-6	-3	0	1	1	-1	-3	-5	-7	-9	-11
C	-8	-5	-2	-1	2	0	-2	-4	-6	-8	-10
G	-10	-7	-4	-3	0	3	1	-1	-3	-5	-7
A	-12	-9	-6	-5	-2	1	2	2	0	-2	-4
A	-14	-11	-8	-7	-4	-1	0	3	1	-1	-3
G	-16	-13	-10	-9	-6	-3	-2	1	4	2	0
C	-18	-15	-12	-11	-8	-5	-4	-1	2	3	1
T	-20	-17	-14	-11	-10	-7	-4	-3	0	3	4

```
function  R=nma_problem7_part_a(s1,s2,match,mismatch,gap)
%function  R=nma_problen7_part_a(s1,s2,match,mismatch,gap)
%
% solves problem 7, HW 1 for MATH 127
% by Nasser Abbasi
% sept 25, 2002.
%
% implements Needleman-Wunsch algorithm
%
% INPUT
%  s1: is one sequence.  example S1=['g' 'a' 't' 'c' 'g'];
%  s2: is the second sequence.
%  match:  score for matching bases. such as 1
%  mismatch: score for mismatch. such as -1
%  gap: penatly factor for gaps. such as -2
%
%  NOTE:  S1 is the row sequence at the top, and S2 is column sequence at left.
%
```

```
% reserve space for the score matrix.
nRow=length(s2)+1;
nCol=length(s1)+1;

v=zeros(nRow,nCol);

%
% for needleman, set the boundary condition to gap penalites
%

for(i=2:size(v,2))
   v(1,i)=v(1,i-1)+gap;
end

for(i=2:size(v,1))
   v(i,1)=v(i-1,1)+gap;
end

for n=1:length(s2)
   nn= n+1;
   for m=1:length(s1)
      mm= m+1;
      if  s2(n) == s1(m)
         diagonal= match;
      else
         diagonal = mismatch;
      end

      diagonal   = diagonal + v(nn-1,mm-1);
      upScore    = v(nn-1,mm)+gap;
      leftScore  = v(nn,mm-1)+gap;

      v(nn,mm) = max([upScore, leftScore, diagonal]); %, 0]);
   end
end

% print the score matrix
fprintf('\t\t');
for(i=1:length(s1))
   fprintf('%c\t',s1(i));
end
fprintf('\n');

for(i=1:nRow)

   if(i==1)
      fprintf('\t');
   else
      fprintf('%c\t',s2(i-1));
   end

   for(j=1:nCol)
      fprintf('%d\t',v(i,j));
   end
   fprintf('\n');
end
```

Problem 7
HW2
MATH 127
By Nasser Abbasi
Sept 26, 2002.

Part (b)

In this part, the input is 'n' which is the length of the sequence. I'll use the MATLAB function 'perms' to generate all permutations of length n (which will be n! many). Then for each permutation, will use global alignment, then look at the score in the bottom right corner of the matrix. This gives me the length of the longest increasing subsequence for this one permutation. I add all these lengths and divide by n! to get the average.

I implemented this in the function nma_problem_7_part_b.m
» help nma_problem_7_part_b

```
 function  R=nma_problen7_part_b(n,match,mismatch,gap)

  solves problem 7 part b, HW 1 for MATH 127
  by Nasser Abbasi
  sept 25, 2002.

  find the average length of the longest increasing subsequence
  in a permutation of length n.

  INPUT
   n : the length of the sequence.
   match:  score for matching bases. such as 1
   mismatch: score for mismatch. such as -1
   gap: penatly factor for gaps. such as –2
```

Example runs:

```
» nma_problem_7_part_b(2,match,mismatch,gap)
Average length of longest increasing subsequence in perms of length 2 is 1.500000

» nma_problem_7_part_b(3,match,mismatch,gap)
Average length of longest increasing subsequence in perms of length 3 is 2.000000

» nma_problem_7_part_b(4,match,mismatch,gap)
Average length of longest increasing subsequence in perms of length 4 is 2.416667

» nma_problem_7_part_b(5,match,mismatch,gap)
Average length of longest increasing subsequence in perms of length 5 is 2.791667
```

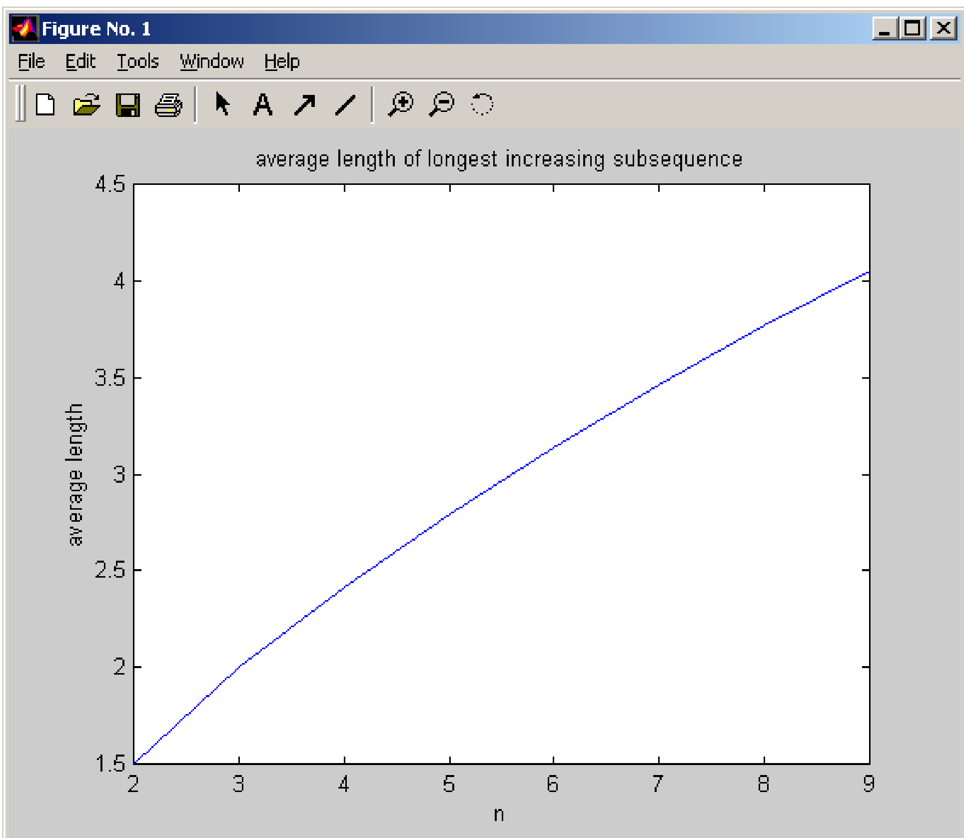
» nma_problem_7_part_b(6,match,mismatch,gap)
Average length of longest increasing subsequence in perms of length 6 is 3.140278

» nma_problem_7_part_b(7,match,mismatch,gap)
Average length of longest increasing subsequence in perms of length 7 is 3.465278

» nma_problem_7_part_b(8,match,mismatch,gap)
Average length of longest increasing subsequence in perms of length 8 is 3.770337

» nma_problem_7_part_b(9,match,mismatch,gap)
Average length of longest increasing subsequence in perms of length 9 is 4.059350

Plotting average length as function of n using MATLAB gives this:



```
function  R=nma_problem7_part_b(n,match,mismatch,gap)
%function  R=nma_problen7_part_b(n,match,mismatch,gap)
%
% solves problem 7 part b, HW 1 for MATH 127
% by Nasser Abbasi
% sept 25, 2002.
%
% find the average length of the longest increasing subsequence
% in a permutation of length n.
%
% INPUT
%  n : the length of the sequence.
%  match:  score for matching bases. such as 1
%  mismatch: score for mismatch. such as -1
%  gap: penatly factor for gaps. such as -2
%
%

thePerms = perms(1:n);
totLen=0;

for(k=1:size(thePerms,1))
   totLen = totLen +
getLengthOfLongestSubSequence(thePerms(k,:),match,mismatch,gap);
end

av=totLen/factorial(n);

fprintf('Average length of longest increasing subsequence in perms of
length %d is %f\m',...
   n,av);
```

```
%%%%%%%%%%%%%%%%%%%%
%
%
%%%%%%%%%%%%%%%%%%%%
function len=getLengthOfLongestSubSequence(seq,match,mismatch,gap)

s2=1:length(seq);
s1=seq;

%
% reserve space for the score matrix.
nRow=length(s2)+1;
nCol=length(s1)+1;

v=zeros(nRow,nCol);

%
% for needleman, set the boundary condition to gap penalites
%

for(i=2:size(v,2))
   v(1,i)=v(1,i-1)+gap;
end

for(i=2:size(v,1))
   v(i,1)=v(i-1,1)+gap;
end

for n=1:length(s2)
   nn= n+1;
   for m=1:length(s1)
      mm= m+1;
      if  s2(n) == s1(m)
         diagonal= match;
      else
         diagonal = mismatch;
      end

      diagonal   = diagonal + v(nn-1,mm-1);
      upScore    = v(nn-1,mm)+gap;
      leftScore  = v(nn,mm-1)+gap;

      v(nn,mm) = max([upScore, leftScore, diagonal]); %, 0]);
   end
end

len = v(end,end);
```

8.1 Problem 7 source code

```
function  nma_alignment_main()


h0= nma_alignment_GUI;

nma_alignment_callbacks('init',h0);
```

```
function  R=nma_problem7_part_a(s1,s2,match,mismatch,gap)
%function  R=nma_problen7_part_a(s1,s2,match,mismatch,gap)
%
% solves problem 7, HW 1 for MATH 127
% by Nasser Abbasi
% sept 25, 2002.
%
% implements Needleman-Wunsch algorithm
%
% INPUT
%  s1: is one sequence.  example S1=['g' 'a' 't' 'c' 'g'];
%  s2: is the second sequence.
%  match:  score for matching bases. such as 1
%  mismatch: score for mismatch. such as -1
%  gap: penatly factor for gaps. such as -2
%
%  NOTE:  S1 is the row sequence at the top, and S2 is column sequence at left.
%

%
% reserve space for the score matrix.
nRow=length(s2)+1;
nCol=length(s1)+1;

S=zeros(nRow,nCol);  %setup space for scoring matrix.

%
% for needleman, set the boundary condition to gap penalites
%

for(i=2:size(S,2))
    S(1,i)=S(1,i-1)+gap;
end

for(i=2:size(S,1))
    S(i,1)=S(i-1,1)+gap;
end
```

```
for n=1:length(s2)
    nn= n+1;
    for m=1:length(s1)
        mm= m+1;
        if  s2(n) == s1(m)
            diagonal= match;
        else
            diagonal = mismatch;
        end

        diagonal   = diagonal + S(nn-1,mm-1);
        upScore    = S(nn-1,mm)+gap;
        leftScore  = S(nn,mm-1)+gap;

        S(nn,mm) = max([upScore, leftScore, diagonal]); %, 0]);
    end
end

% print the score matrix
fprintf('\t\t');
for(i=1:length(s1))
    fprintf('%c\t',s1(i));
end
fprintf('\n');

for(i=1:nRow)

    if(i==1)
        fprintf('\t');
    else
        fprintf('%c\t',s2(i-1));
    end

    for(j=1:nCol)
        fprintf('%d\t',S(i,j));
    end
    fprintf('\n');
end


%print the alignment
doAlignment(S,s1,s2);


%%%%%%%%%%%%%5
```

```
%
%
%%%%%%%%%%%%%%%%%%%
function doAlignment(S,topSeq,leftSeq)

[nRow,nCol]=size(S);
i=nRow;
j=nCol;
A=zeros(2*(nRow+nCol-2),2)';  %create space for the alignment.
k=0;

top=topSeq(j-1);
btm=leftSeq(i-1);

while(1)
    k=k+1;
%     if(i==1 | j==1)
%         if(i==1)
%             btm='-';
%             while(j>=i)
%                 top=topSeq(j);
%                 A(1,k)=top;
%                 A(2,k)=btm;
%                 j=j-1;
%                 k=k+1;
%             end
%         else
%             top='-';
%             while(i>=1)
%                 btm=leftSeq(i);
%                 A(1,k)=top;
%                 A(2,k)=btm;
%                 k=k+1;
%                 i=i-1;
%             end
%         end
%         break;
%     end

    [newi,newj]=maxParent(S,i,j);

    if(newi==i) %same row
        top=topSeq(j-1);
        btm='-';
    else
        if(newj==j) %same column
            top='-';
```

```
            btm=leftSeq(i-1);
        else
            top=topSeq(j-1);
            btm=leftSeq(i-1);
        end
    end
    A(1,k)=top;
    A(2,k)=btm;

    if(newi==1 | newj==1)
        break;
    end

    i=newi;
    j=newj;
end

for(i=k:-1:1)
    fprintf('%c',A(1,i));
end
fprintf('\n');
for(i=k:-1:1)
    if(isequal(A(1,i),A(2,i)))
       fprintf('|');
   else
       fprintf(' ');
   end
end
fprintf('\n');
for(i=k:-1:1)
    fprintf('%c',A(2,i));
end




%%%%%%%%%%%%%%%%%%
%
%
%%%%%%%%%%%%%%%%%%%%%%%
function [newi,newj]=maxParent(S,i,j)

[nRow,nCol]=size(S);
if(i==1 & j==1)
    newi=i;
    newj=j;
else
```

```
    if(j==1)
        newi=i-1;
        newj=j;
    else
        if(i==1)
            newj=j-1;
            newi=i;
        else
            if(S(i,j-1) > S(i-1,j-1))
                if(S(i,j-1)>S(i-1,j))
                    newi=i;
                    newj=j-1;
                else
                    newi=i-1;
                    newj=j;
                end
            else
                if(S(i-1,j-1)>=S(i-1,j))
                    newi=i-1;
                    newj=j-1;
                else
                    newi=i-1;
                    newj=j;
                end
            end
        end
    end
end
```

```
function  R=nma_problem7_part_b(n,match,mismatch,gap)
%function  R=nma_problen7_part_b(n,match,mismatch,gap)
%
% solves problem 7 part b, HW 1 for MATH 127
% by Nasser Abbasi
% sept 25, 2002.
%
% find the average length of the longest increasing subsequence
% in a permutation of length n.
%
% INPUT
%  n : the length of the sequence.
%  match:  score for matching bases. such as 1
%  mismatch: score for mismatch. such as -1
%  gap: penatly factor for gaps. such as -2
%
%
```

```
thePerms = perms(1:n);
totLen=0;

for(k=1:size(thePerms,1))
   totLen = totLen + getLengthOfLongestSubSequence(thePerms(k,:),match,mismatch,gap);
end

av=totLen/factorial(n);

fprintf('Average length of longest increasing subsequence in perms of length %d is %f\m',...
   n,av);

%%%%%%%%%%%%%%%%%%%%
%
%
%%%%%%%%%%%%%%%%%%%%
function len=getLengthOfLongestSubSequence(seq,match,mismatch,gap)

s2=1:length(seq);
s1=seq;

%
% reserve space for the score matrix.
nRow=length(s2)+1;
nCol=length(s1)+1;

v=zeros(nRow,nCol);

%
% for needleman, set the boundary condition to gap penalites
%

for(i=2:size(v,2))
   v(1,i)=v(1,i-1)+gap;
end

for(i=2:size(v,1))
   v(i,1)=v(i-1,1)+gap;
end

for n=1:length(s2)
   nn= n+1;
   for m=1:length(s1)
      mm= m+1;
      if  s2(n) == s1(m)
         diagonal= match;
```

```
      else
         diagonal = mismatch;
      end

      diagonal   = diagonal + v(nn-1,mm-1);
      upScore    = v(nn-1,mm)+gap;
      leftScore  = v(nn,mm-1)+gap;

      v(nn,mm) = max([upScore, leftScore, diagonal]); %, 0]);
   end
end

len = v(end,end);
```