Introduction to Scientific Computing
Report Practical : Sequence Alignment
Denis Garabajiu S4142551
Constantin Cainarean S4142152
Mar 20, 2021

PART 1 Basic Needleman-Wunsch algorithm

In this part of the practical our task was to implement the Needleman-Wunsch algorithm from the reader. We have followed the instructions and the pseudo-code described in the reader in order to implement the algorithm in MATLAB. First of all, we start with reading the input from a .txt file. For this we were given the code in the file *read_seqs.m*. In order to be able to record the result in the matrix D we need to allocate enough space for it, as the result of our program is a (n + 1) × (m + 1)- dimensional matrix containing the costs of optimal alignment. This can be done with the following line of code :

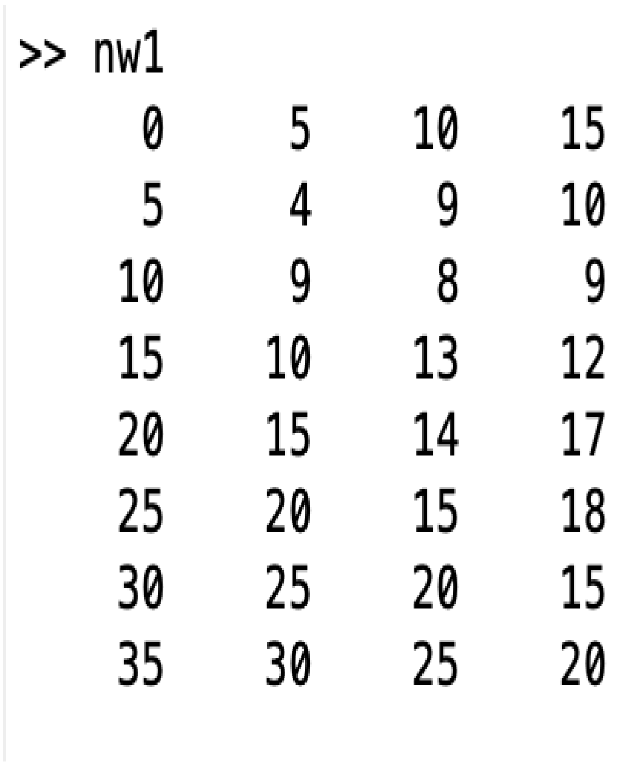
```
D = zeros(len_s + 1, len_t + 1);
```

where *len_s* is the length of the first line of input and *len_t* of the second. After that we can start to implement the algorithm itself. One thing we want to mention is the indexing in MATLAB. As we know MATLAB has indexing of arrays beginning from 1 instead of 0, this is the reason why in the two for-loops from the beggining of the algorithm we are running from 1 to the length of the line + 1. Following the same reason we have adapted the pseudo-code to the language requirements so we have added the value **1** inside the matrix positioning and the value **i - 1** in the evaluation of the matrix, so the two for-loops look like this :

```
for i = 1 : (len_s + 1)
   D(i,1) = ((i - 1) * g);
end

for j = 1 : (len_t + 1)
   D(1,j) = (j - 1) * g;
end
```

The main part of the algorithm is a nested for-loop. Both of the following loops start at the index 2 and end at the length **n + 1** and **m + 1**, again this is due to the specific indexing of the MATLAB. Next we are implementing the Match, Delete, Insert operations themselves. We have a helper *function w* that is used to find the position in the scoring matrix for the Match operation. This function is detecting if the current position is on the diagonal of the matrix or not, if so it returns *p (cost of a match)* or otherwise it returns *q (cost of the mismatch)*. Then we are computing the minimum of these three operations and assign it to the current position in the resulting matrix. Following the assignment instructions we have changed the input file to the *nw_test1.txt* and we have set the values of the cost of the match (p), the cost of the mismatch (q) and the gap penalty (g) to **p = 0 ; q = 4 ; g = 5**. The code described above can be found under the file *nw1.m*. Running the program with the input mentioned above we get



```
>> nw1
     0     5    10    15
     5     4     9    10
    10     9     8     9
    15    10    13    12
    20    15    14    17
    25    20    15    18
    30    25    20    15
    35    30    25    20
```

the following result :
The output matrix matches exactly the matrix from the page 26 of the reader, therefore we conclude that our algorithm is working properly, as intended.

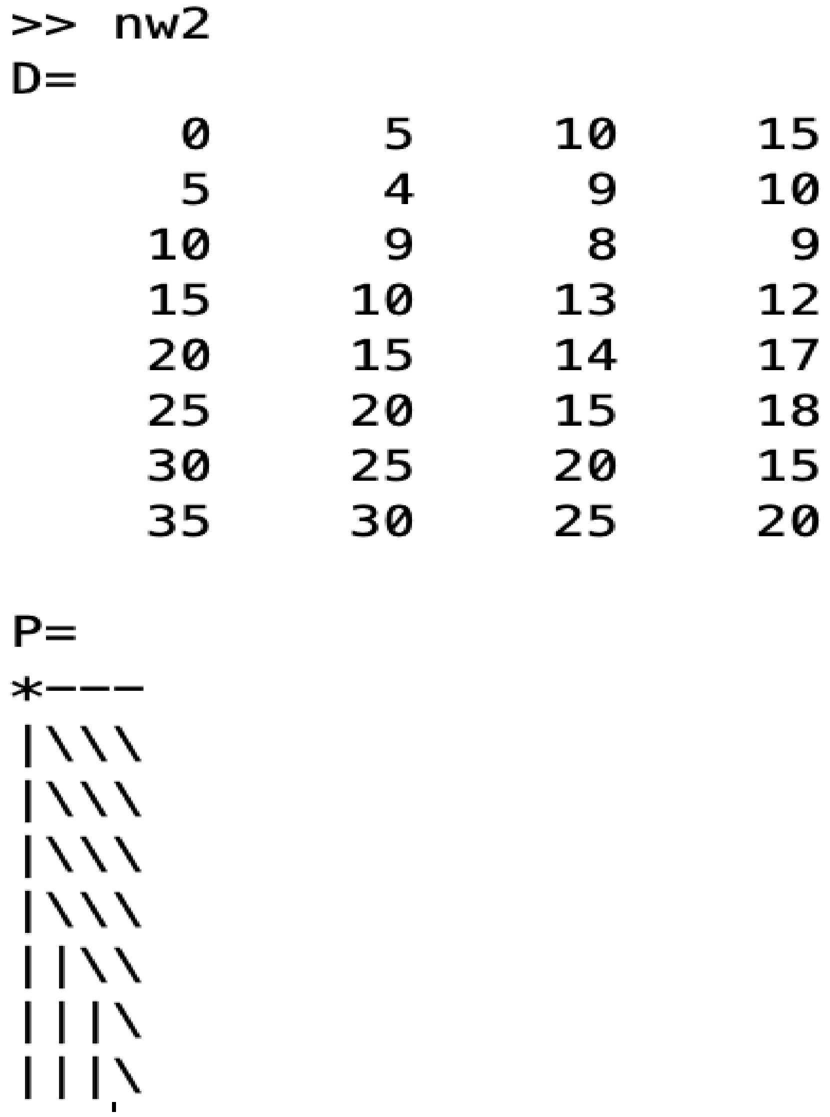
PART 2 Needleman-Wunsch algorithm with predecessors

In this part of the practical we are required to compute the predecessor matrix. Our first step is again to initialize the resulting matrix P, which will hold the ouput, therefore we create a matrix of the dimensions *len_s x len_t* and initialize it with the character ′*′, this is done with the help of the function *repmat*.

In the first two for-loops of the algorithm we add the characters of the cells in the first row and column. To implement the later required functionality we have created a function *predecessor* which is going to compute which of the operation has resulted in the minimal value, and output the character that corresponds to it. Match will be North with the sign ′|′, Insert is west with the sign **′-′** and delete is the North-West with the sign ′\′. Therefore we assign to the current position in matrix P the sign resulting from the function *predecessor*.

```
% This function is used to compute the sign that indicates
% the direction of the predecessor
function sign = predecessor(match, delete, insert, minimal)
   if (delete == minimal)
      sign = '|';
   end
   if (insert == minimal)
      sign = '-';
   end
   if (match == minimal)
      sign = '\';
   end
end
```

As requested in the assignment we have saved this modified version of the algorithm in the file with the name *nw2.m* and we have run it with the parameters used in the previous part. Our result was :



```
>> nw2
D=
     0     5    10    15
     5     4     9    10
    10     9     8     9
    15    10    13    12
    20    15    14    17
    25    20    15    18
    30    25    20    15
    35    30    25    20

P=
*---
|\\\
|\\\
|\\\
|\\\
||\\
|||\
|||\
```

Comparing this output with the provided in the assignment and also the one from the p.132 in the reader. This assignment limits our number of predecessors to one with one optimal alignment, therefore the P matrix doesn't show all the arrows of the predecessors. We have concluded that our algorithm works properly and as intended.

PART 3 Needleman-Wunsch algorithm with optimal alignment

In this part of the practical our task was to make the algorithm from the previous part more complex , such that also the optimal alignment is determined. For the sake of readability we decided to use different notation for the variables described in the assignment as *s_al, t_al, l_al*. We use the variables *s_string, t_string and line_string* respectively. Our first step is to initialize this strings as empty, because elements will be added during the iterations. Also we have created two variables *row* and *column* that start from the values **len_s + 1** and **len_t + 1** respectively. This is due to the fact that the predecessor matrix will be parsed from the last cell to the first one (1,1).

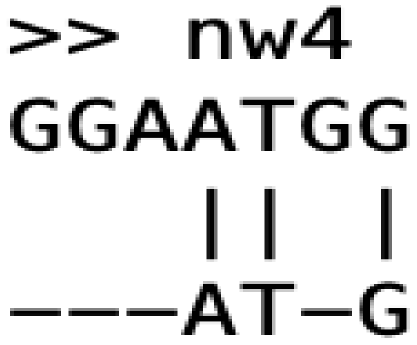
Our main computation occurs in the while-loop that contains four if-statements. The first if statement is when the predecessor is equal to " |". In this case we add the value s(row - 1) to the s_string and also a gap to the t_string. In this scenarion we do not have anything, not a match or a mismatch, so to the line_string we do not add anything, leaving an empty space. Since the direction of our predecessor is North, we decrement only the north variable.

In the second if-statement when the predecessor equals to "-" the result is completely opposite to the first case. We add t(column - 1) to t_string, and a gap to the s_string. Same empty space is added to line_string. And in this case, since we have West direction, we decrement the column variable.

In the third if-statement we have that the predecessor is equal to "\". This is a more complex scenario, where two outcomes are possible. If the predecessor from the North-West side is not a match, we add s(row - 1) to s_string and t(column - 1) to the t_string. In case of a match we add the same values to the strings, but to the line_string we add also the character "|" meaning that there is a match between the strings.

The fourth if-statement is to skip the default "*" sign, decrementing both variables row and column.

Following the instructions of the assignment we have tested the above described algorithm on the file *nw_test1.txt* and we have obtained the following result :



```
>> nw4
GGAATGG
   || |
---AT-G
```

.

Analyzing the output we came to the conclusion that our algorithm is working correctly, since the output is a right representation of an optimal alignment and also it coincides to the one shown in the assignment.
In the last point of the assignment, we were provided with the file *print_to_file.m* which contained code that would help us to open the file *nw3-output.txt* and fill it with the outputs from the previous parts, such as the matrix D, the matrix P, and the alignment. Having a look inside this text file, we again came to the conclusion that every part of the practical is working perfectly fine. We have included all of the above mentioned features in the code of the file *nw3.m*.

Appendix: the skeleton of our scripts

nw1.m

```
in=fopen('nw_test1.txt');
s=fgetl (in);
t=fgetl (in);
fclose (in);
len_s = length(s) ;
len_t = length(t);

% Test values from the assignment
p = 0;
q = 4;
g = 5;

% Allocating enough memory for the resulting matrix
D = zeros(len_s + 1, len_t + 1);

% Needleman-Wunsch Algorithm (9.1 in Reader)

for i = 1 : (len_s + 1)
   D(i,1) = ((i - 1) * g);
end

for j = 1 : (len_t + 1)
   D(1,j) = (j - 1) * g;
end

for i = 2 : (len_s + 1)
   for j = 2 : (len_t + 1)
      match = D(i - 1, j - 1) + w(s(1, i - 1), t(1,j - 1), p, q);
      delete = D(i - 1, j) + g;
      insert = D(i, j - 1) + g;
      D(i,j) = min([match, delete, insert]);
   end
end

disp(D); % Displays the resulting matrix D

% Function to compute the output value of the scoring matrix.
function val = w(a, b, p, q)
   if (a == b)
      val = p;
   else
      val = q;
   end
end
```

nw2.m

```
in=fopen('nw_test1.txt');
s=fgetl (in);
```

```
t=fgetl (in);
fclose (in);
len_s = length ( s ) ;
len_t = length(t);

% Test values from the assignment
p = 0;
q = 4;
g = 5;

% Allocating enough memory for the resulting and predecessor matrix
D = zeros(len_s + 1, len_t + 1);
P = repmat('*',[len_s + 1, len_t + 1]);

% Needleman-Wunsch Algorithm (9.1 in Reader)
for i = 1 : (len_s + 1)
   D(i,1) = ((i - 1) * g);
   if (i ~= 1 )
      P(i,1) = '|';
   end
end

for j = 1 : (len_t + 1)
   D(1,j) = (j - 1) * g;
   if (j ~= 1 )
      P(1,j) = '-';
   end
end

for i = 2 : (len_s + 1)
   for j = 2 : (len_t + 1)
      match = D(i - 1, j - 1) + w(s(1, i - 1), t(1,j - 1), p, q);
      delete = D(i - 1, j) + g;
      insert = D(i, j - 1) + g;
      minimal = min([match, delete, insert]);
      D(i,j) = minimal;
      P(i,j) = predecessor(match, delete, insert, minimal);
      % P - Matrix that contains characters that indicate the direction
         of
      % the predecessors
   end
end

% Display the matrices with the layout described in the assignment
fprintf("D=\n");
disp(D);
fprintf("P=\n");
disp(P);

% Function to compute the output value of the scoring matrix.
function x = w(a, b, p, q)
   if (a == b)
      x = p;
   else
      x = q;
   end
end

% This function is used to compute the sign that indicates
% the direction of the predecessor
function sign = predecessor(match, delete, insert, minimal)
   if (delete == minimal)
      sign = '|';
   end
   if (insert == minimal)
      sign = '-';
   end
   if (match == minimal)
      sign = '\';
   end
end
```

nw3.m

```
in=fopen('nw_test1.txt');
s=fgetl (in);
t=fgetl (in);
fclose (in);
len_s = length(s) ;
len_t = length(t);

% For our commodity and overall readability we decided to chage t_al,
    s_al and l_al into the fpllowing variables.
s_string = '';
t_string = '';
line_string = '';
row = len_s + 1;
column = len_t + 1;

% Test values from the assignment
p = 0;
q = 4;
g = 5;

% Allocating enough memory for the resulting and predecessor matrix
D = zeros(len_s + 1, len_t + 1);
P = repmat('*',[len_s + 1, len_t + 1]);

% Needleman-Wunsch Algorithm (9.1 in Reader)
for i = 1 : (len_s + 1)
   D(i,1) = ((i - 1) * g);
   if (i ~= 1 )
      P(i,1) = '|';
   end
end
```

```
for j = 1 : (len_t + 1)
   D(1,j) = (j - 1) * g;
   if (j ~= 1 )
      P(1,j) = '-';
   end
end

for i = 2 : (len_s + 1)
   for j = 2 : (len_t + 1)
      match = D(i - 1, j - 1) + w(s(1, i - 1), t(1,j - 1), p, q);
      delete = D(i - 1, j) + g;
      insert = D(i, j - 1) + g;
      minimal = min([match, delete, insert]);
      D(i,j) = minimal;
      P(i,j) = predecessor(match, delete, insert, minimal);
      % P - Matrix that contains characters that indicate the direction
          of
      % the predecessors
   end
end

while (row > 0 && column > 0)

   if (P(row,column) == '|')
      line_string = [' ', line_string];
      t_string = ['-', t_string];
      s_string = [s(row - 1), s_string];
      row = row - 1;
   end

    if (P(row,column) == '-')
      line_string = [' ', line_string];
      t_string = [t(column - 1), t_string];
      s_string = ['-', s_string];
      column = column - 1;
   end

   if (P(row,column) == '\')
      if (s(row - 1) ~= t(column - 1)) % Mismatch
         line_string = [' ', line_string];
         t_string = [t(column - 1), t_string];
         s_string = [s(row - 1), s_string];
      else
         line_string = ['|', line_string]; % Match
         t_string = [t(column - 1), t_string];
         s_string = [s(row - 1), s_string];
      end
      row = row - 1;
      column = column - 1;
   end

    if (P(row,column) == '*')
      row = row - 1;
      column = column - 1;
   end
end

% Displaying the Alignment in the console
disp(s_string);
disp(line_string);
disp(t_string);

% Writing the output of all the parts (matrix D, matrix P, Alignment)
    into a single file
output=fopen('nw3output.txt', 'w');
fprintf(output,'Name: <Cainarean Constantin(s4142152) && Denis
    Garabajiu(s4142551) >\n');
fprintf(output,'IBC, Practical 4 \n\n');

  fprintf(output,'\n\nString s:\n');
  for i=1:length(s)
    fprintf(output,'%s',s(i));
  end
  fprintf(output,'\n\nString t:\n');
  for i=1:length(t)
    fprintf(output,'%s',t(i));
  end

  fprintf(output,'\n\nMatrix D:\n\n');
  for i = 1:len_s + 1
     for j = 1:len_t + 1
        fprintf(output, '%4d', D(i,j));
     end

     fprintf(output,'\n');
  end

  fprintf(output,'\n\nMatrix P:\n\n');
  for i = 1:len_s + 1
     for j = 1:len_t + 1
        fprintf(output, '%2c', P(i,j));
     end
     fprintf(output,'\n');
  end

  fprintf(output,'\n\nAlignment:\n\n');

  for i = 1:length(s_string)
     fprintf(output, '%2c', s_string(i));
  end

  fprintf(output,'\n');

  for i = 1:length(line_string)
     fprintf(output, '%2c', line_string(i));
  end
  fprintf(output,'\n');

  for i = 1:length(t_string)
     fprintf(output, '%2c', t_string(i));
  end
  fclose(output);                    % close file


% Function to compute the output value of the scoring matrix.
function x = w(a, b, p, q)
   if (a == b)
      x = p;
   else
      x = q;
   end
end

% This function is used to compute the sign that indicates
% the direction of the predecessor
function sign = predecessor(match, delete, insert, minimal)
   if (delete == minimal)
      sign = '|';
   end
   if (insert == minimal)
      sign = '-';
   end
   if (match == minimal)
      sign = '\';
   end
end
```

Work Distribution

In the following practical we have collaborated through a video-call. We consider that this assignment was done together in collaboration so both of us have participated 50% in the realisation of this practical.