Assignment 1 IM - HL7 Message Parsing

Gerardo Vitagliano - 2017214620
Martin Schlegel - 2017190694

1 HL7 Message

Il seguente documento analizza le prestazioni della funzione gauss(), definita nello script matlab gauss.m. Quest'ultima calcola la soluzione di un sistema lineare espresso nella forma matriciale tramite la matrice dei coefficienti A e il vettore dei termini noti b. Nel seguito verranno presentati i risultati dei test case effettuati facendo utilizzo del framework offerto da MATLAB. I casi di test hanno lo scopo di verificare l'efficacia della funzione implementata, la robustezza nelle condizioni in cui non siano state seguite le specifiche necessarie, l'accuratezza e l'errore relativo della soluzione ottenuta.

L'accuratezza della funzione è stata calcolata considerando l'errore relativo. Tale errore può essere ottenuto, a partire dalla funzione, utilizzando i risultati discendenti dal teorema di Wilkinson. In particolare, grazie a tale teorema è possibile calcolare il residuo della funzione e utilizzarlo per stimare l'errore relativo.

Nello specifico, per ottenere il residuo si utilizza il seguente risultato: (dal teorema di Wilkinson)

$$\text{residuo}=norm(b - Ax_c)/norm(A) * norm(x_c)$$

Dove per $norm(x)$ si definisce la norma-2 vettoriale. L'errore relativo verrà poi stimato tramite la formula:

$$\text{errore}=cond(A) * residuo \quad ,$$

in cui $cond(A)$ rappresenta il numero di condizionamento in base alla norma-2 della matrice A.

In appendice saranno riportati i codici degli script in **MATLAB** utilizzati per effettuare il testing.

1.1 Casi di test

Ciascun test è stato implementato come funzione all'interno del framework per i test case di MATLAB. Di seguito si analizzerà il codice di ciascuno di essi e se ne riporterà il risultato. I casi di test sono stati progettati ed eseguiti secondo la seguente tabella:

2 Parser

The implemented parser takes into account the structure of the message according to the HL7 standard 2.7. Due to the task requested, the parser does no full check over HL7 2.7 syntax, but instead takes into account only the specific case of a ORUˆR01 message. Moreover, it implements a logic to check the correct

use of segments, without further analyzing fields and subfields. In this section the MATLAB script code will be analyzed one chunk at a time, in order to explain in detail its behaviour.
The very first lines of code have the task of importing the message in the MATLAB environment and extract the relevant information to obtain a structured data containing the message. The input file is imported with the use of textscan: since the segments in the HL7 message are separated by a new line, the "Delimiter" option can be used to separate different text lines. This function returns a cell structure with just one cell, containing inside an Nx1 string matrix in which every row element corresponds to one of the N segment of the original message. This matrix is stored in the temporary variable 'res'.

```
[fileID,msg] = fopen('message.hl7');
inp = textscan(fileID,'%s','Delimiter','\n');
res=vertcat(inp{1,1});
fclose(fileID);
```

Code 1: Message import

Right after the segments are imported, the next task is separating fields contained in a segment. The desired output is a NxM matrix, where M is the maximum number of fields contained in a segment received. Because M is not known in advance, since segments can have a different and not fixed number of fields, there is the need of two nested for loops. The outer one will retrieve the fields for each segment, and transpose them in order to get a row vector, while the inner one, in case the number of fields of current segment is greater than the one in the previous ones, has the task to "pad" all the previous rows to fit them in the new dimensions of the matrix. At the end of this code, the variable 'field' will be an NxM string matrix, in which the (i,j) element is the j-th field of the i-th segment.

```
columns=1;
rows=length(res);
 for i=1:rows
    temp = strread(res{i,1},'%s','delimiter','|')'; %The output is transposed so to get every segment on a row
    if length(temp)>columns %Columns store the maximum number of fields found
        for j=1:i           %if a new segment has more field than the previous, the previous are padded with '\p'
            fields(j,columns+1:length(temp))={'\p'};  %to pad, the old
```

```
                value of columns is used until the new one
        end
         columns=length(temp);
    end
    fields(i,1:length(temp))=temp;
 end

end
```

Code 2: Message formatting

After the message is properly imported, the code follows checking the validity of the segments. The very first, simple checks are that the first segment is the proper header, MSH, and that the second is the Patient Identification. In fact, according to HL7 2.7 syntax, these two segments are mandatory, and should the message should respect the order.

```
if ~strcmp(fields{1,1},'MSH')
    disp('First segment is not MSH - invalid syntax');
    return
end
if ~strcmp(fields{2,1},'PID')
    disp('Second segment is not Patient ID - invalid syntax');
    return
end
```

Code 3: Validity of MSH and PID segments

The following check is to be sure the message sent actually contains an OBR segment, since being a Observation Request this segment is mandatory. However, given the possible flexibility in the message structure, there is no fixed position for an OBR segment: because of this a for loop is used to check all the segments. This loop starts from 3 since the two first fields, for the message to be valid, must be MSH and PID. In case at least one OBR segment is found, the variable validOBR is set to be true. Since every OBR can one or more associated OBX segments, after finding one OBR a set of check is performed in order to validate the presence and the correct ordering of the IDs for eventual OBX segments. The variable validOBX is at first set as true, since an OBX segment could not be present at all without this being an error, while is set to false in case an OBX segment is found with an invalid ID. Additional checks are present in order to consider situations in which the message ends with an OBR segment, and thus preventing an out-of-bound exception while trying to access a row index in overflow for the ’fields’ matrix.

```matlab
%There should be at least one OBR segment, if there are any
validOBR=false;
validOBX=true;
for i=3:rows
    if strcmp(fields{i,1},'OBR')
        validOBR=true;
        if (i+1<rows)                       %check if the message does not end with an OBR segment
            if strcmp(fields{i+1,1},'OBX')
                if(1~=str2num(fields{i+1,2}))   %If the first OBX is not with ID 1
                    validOBX=false;
                    break
                end
                j=2;
                if(i+j<rows)                    %Check if the message ends with an OBX
                    while strcmp(fields{i+j,1},'OBX') %check if subsequent OBX have correct ID
                        if (j~=str2num(fields{i+j,2}))
                            validOBX=false;
                            break
                        end
                    j=j+1;
                    end
                end
            end
        end
    end
end

if ~validOBR
    disp('There is not an OBR segment - invalid syntax');
    return
end

if ~validOBX
    disp('There is a problem with OBX segment indexes - invalid syntax');
    return
end
```

Code 4: Validity of OBR/OBX segments

Finally, given the correctness of segment syntax, the value of the fields is retrieved for an output to the user. This is done using the strsplit MATLAB function taking into account that values in fields are separated by the '^' character. Given the formatting of the field matrix, the needed information is in fixed position: in fact even though some fields are not presents, these will result in empty cells of the matrix. This formatting was chosen rightly in order to guarantee ease of value retrieval.

```
name=strsplit(fields{2,6},'^');
height=fields{5,6};
heightMeasure=strsplit(fields{5,7},'^');

weight=fields{6,6};
weightMeasure=strsplit(fields{6,7},'^');

heartRate=fields{7,6};
heartMeasure=strsplit(fields{7,7},'^');

sysPre=fields{8,6};
sysMeasure=strsplit(fields{8,7},'^');

dyaPre=fields{9,6};
dyaMeasure=strsplit(fields{9,7},'^');
```

Code 5: Information Retrieval

The last lines of code simply perform output of the information retrieved from the message.

```
fprintf('Information about the patient: %s %s\n' ,name{1},name{2});
fprintf('Details: \n');
fprintf('Weight: %s %s\n' ,weight,weightMeasure{2});
fprintf('Height: %s %s\n' ,height,heightMeasure{2});
fprintf('Systolic blood pressure: %s %s\n' ,sysPre,sysMeasure{2});
fprintf('Dyastolic blood pressure: %s %s\n' ,dyaPre,dyaMeasure{2});
fprintf('Mean blood pressure by ppg shown in figure\n');

y=str2double(strsplit(fields{13,6},'^'));
x=1:1000;
plot(x,y);
title('Results of PPG Examination for patient' name);
xlabel('Time ticks - seconds');
ylabel('PPG Values');
```

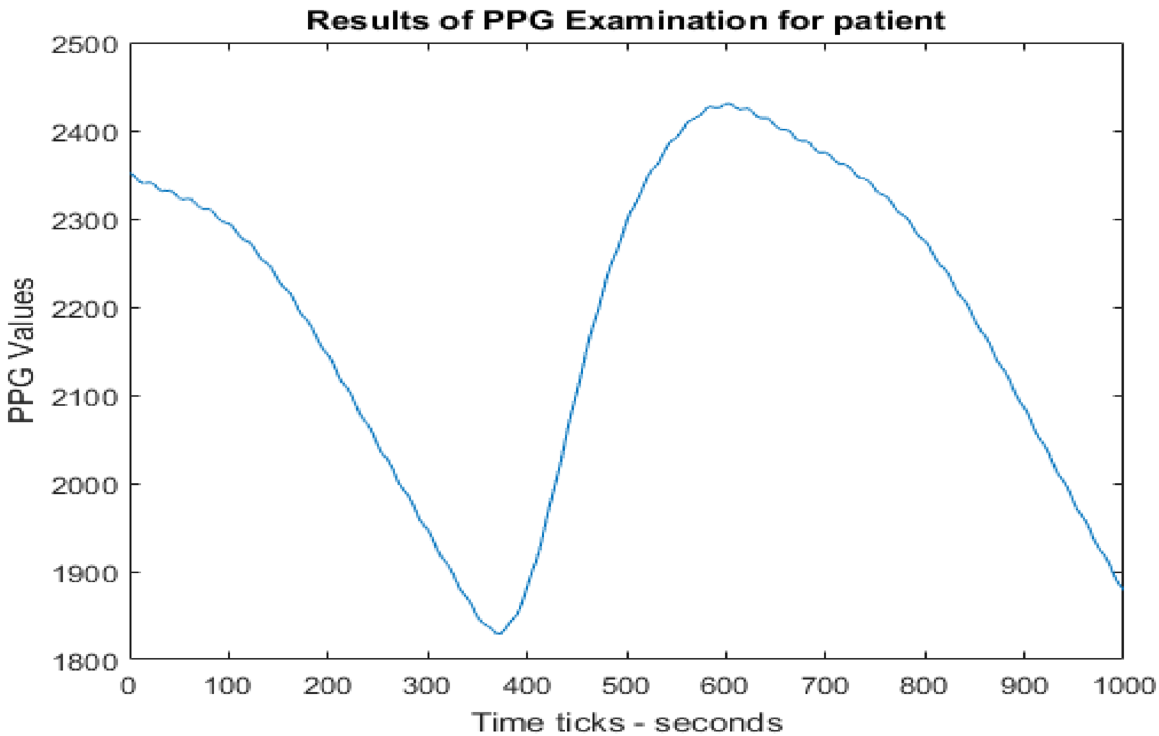
Code 6: Information Retrieval

2.1 Code output

Once run, the MATLAB script output is the following:

```
>> parser
    Information about the patient: Botija Anacleto
    Details:
    Weight: 123 kg
    Height: 176 cm
    Systolic blood pressure: 120 mmHg
    Dyastolic blood pressure: 88 mmHg
    Mean blood pressure by ppg shown in figure
```

Code 7: Parser Output



A Appendix: Script

A.1 parser.m

```
[fileID,msg] = fopen('message.hl7');
inp = textscan(fileID,'%s','Delimiter','\n');
res=vertcat(inp{1,1});
fclose(fileID);

columns=1;
rows=length(res);
 for i=1:rows
    temp = strread(res{i,1},'%s','delimiter','|')'; %The output is transposed so to get every segment on a row
    if length(temp)>columns %Columns store the maximum number of fields found
        for j=1:i            %if a new segment has more field than the previous, the previous are padded with '\p'
            fields(j,columns+1:length(temp))={'\p'};  %to pad, the old value of columns is used until the new one
        end
         columns=length(temp);
    end
    fields(i,1:length(temp))=temp;
 end


%checking validity
if ~strcmp(fields{1,1},'MSH')
    disp('First segment is not MSH - invalid syntax');
    return
end
if ~strcmp(fields{2,1},'PID')
    disp('Second segment is not Patient ID - invalid syntax');
    return
end


%There should be at least one OBR segment, if there are any
validOBR=false;
validOBX=true;
for i=3:rows
    if strcmp(fields{i,1},'OBR')
        validOBR=true;
        if (i+1<rows)                                %check if the message does not end with an OBR segment
```

```
            if strcmp(fields{i+1,1},'OBX')
                if(1~=str2num(fields{i+1,2}))   %If the first OBX is not with ID 1
                    validOBX=false;
                    break
                end
                j=2;
                if(i+j<rows)                    %Check if the message ends with an OBX
                    while strcmp(fields{i+j,1},'OBX') %check if subsequent OBX have correct ID
                        if (j~=str2num(fields{i+j,2}))
                            validOBX=false;
                            break
                        end
                    j=j+1;
                    end
                end
            end
        end
    end
end

if ~validOBR
    disp('There is not an OBR segment - invalid syntax');
    return
end

if ~validOBX
    disp('There is a problem with OBX segment indexes - invalid syntax');
    return
end
name=strsplit(fields{2,6},'^');
height=fields{5,6};
heightMeasure=strsplit(fields{5,7},'^');

weight=fields{6,6};
weightMeasure=strsplit(fields{6,7},'^');

heartRate=fields{7,6};
heartMeasure=strsplit(fields{7,7},'^');

sysPre=fields{8,6};
sysMeasure=strsplit(fields{8,7},'^');

dyaPre=fields{9,6};
dyaMeasure=strsplit(fields{9,7},'^');

```

```
fprintf('Information about the patient: %s %s\n' ,name{1},name{2});
fprintf('Details: \n');
fprintf('Weight: %s %s\n' ,weight,weightMeasure{2});
fprintf('Height: %s %s\n' ,height,heightMeasure{2});
fprintf('Systolic blood pressure: %s %s\n' ,sysPre,sysMeasure{2});
fprintf('Dyastolic blood pressure: %s %s\n' ,dyaPre,dyaMeasure{2});
fprintf('Mean blood pressure by ppg shown in figure\n');

y=str2double(strsplit(fields{13,6},'^'));
x=1:1000;
plot(x,y);
title('Results of PPG Examination for patient');
xlabel('Time ticks - seconds');
ylabel('PPG Values');
```