# BEng Biomedical Engineering
# Computer Programming

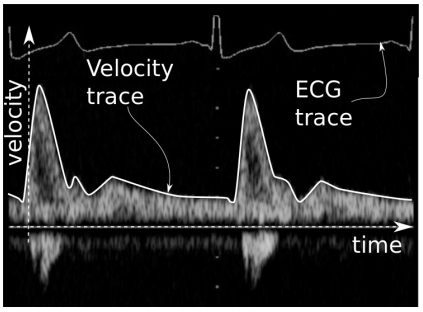
Coursework 3 - Take-Home Assignment

**Objective**

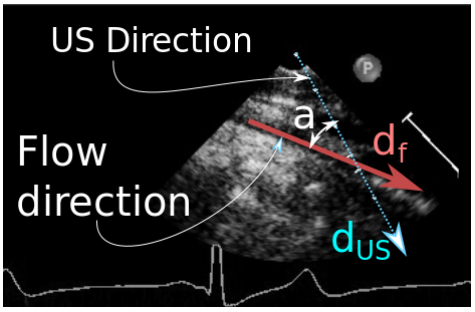
To gain practical experience of computer programming and design using MatLab.

**Introduction**

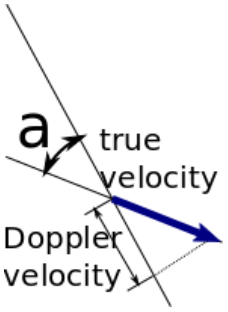
Doppler ultrasound is widely used in clinical practice to measure blood velocity in peripheral vessels, like the carotid artery. Doppler ultrasound estimates blood flow velocity by measuring the change in frequency of reflected sound waves due to the Doppler effect. An example of a Doppler measurement using ultrasound is shown in Fig. 1a.




(a) Pulsed Wave Doppler trace. The image shows the velocity measured over two consecutive cardiac cycles.




(b) Ultrasound image of a carotid artery. Blood velocity direction ($\mathbf{d}_f$) and ultrasound beam direction ($\mathbf{d}_{US}$) are highlighted with red and white arrows respectively. The angle between the two vectors is $a$.




(c) Schematic representation of the angle-dependent Doppler measurement.

Figure 1: Blood velocity measurement on a carotid artery using Doppler ultrasound.

Doppler measurements have one important limitation: the measured velocity is not the true velocity, but only a projection of the true velocity along the direction of propagation of the ultrasound wave, which can be represented as a vector $\mathbf{d}_{US}$. As illustrated in Fig. 1b, the direction of the blood flow $\mathbf{d}_f$ (another vector) is generally at an angle $a$ with $\mathbf{d}_{US}$. The resulting Doppler velocity is then the magnitude of the true velocity projected along the ultrasound direction, as shown in Fig. 1c. The relationship between the magnitude of the measured Doppler velocity $v_{Doppler}$ and the true velocity $v_t$ is:

$$v_{Doppler} = v_t \, \mathbf{d}_f \cdot \mathbf{d}_{US} \tag{1}$$

where $\cdot$ represents the *dot product* (or *inner product*) of the two *unit* vectors (i.e. with length 1).

The aim of this exercise is to design and implement a MatLab program that compensates for the angle-dependency in Doppler velocity measurements and displays the corrected velocity to the user.

**Instructions**

The file *velocity.csv* (available from the KEATS system via the link *Coursework 3 flow file*) contains velocity data for one patient, acquired over several cardiac cycles.

The file contains two columns of data containing the time (in the first column) and blood flow velocity (in the second column) measurements for the patient. The time and flow values in each row are separated by a comma, i.e. the file is in *comma separated values* format. Time is given in milliseconds (ms), and velocity is given in centimeters per second (cm/s).

The file *anatomicImage.png* contains a B-Mode ultrasound image (like the image in Fig. 1b) where the carotid artery and the ultrasound beam directions are visible. The size of each pixel in the image is equivalent to $0.25 \times 0.25$ mm.

You are required to write code to read the files in the format described above and then to analyse the Doppler measured velocity as described below.

- First read in the time and blood velocity data, and the input image and store them in appropriate variable(s). (*Hint: see MatLab documentation for the command* `imread`).
- Then your program should interactively determine the flow direction first, and the ultrasound beam direction next. Each of these directions can be found as follows:
  1. Display the input image. One option is to use the MatLab function `imagesc()` and set the colours to greyscale with the command `colormap(gray);`
  2. Prompt the user with a point-picking tool to pick two points defining the required direction. You can use the MatLab function `impoly` for this purpose (see the documentation for details). This function allows the user to pick as many points as desired on an image. The function returns when the user double-clicks the last point. An example usage of this function is:

     ```
     pickedPoints = impoly;
     ```

     and after the picking has finished, the picked points are returned as a 2D array where each row is a point.
  3. Calculate the direction from the picked points.
  4. Normalise the vector to unit length (i.e. length 1).
- The two directions $d_{US}$ and $d_f$ should then be used to compute the true velocity using Equation (1) above.
- Produce a figure contain the following visualisations:
  - A plot of the original Doppler-measured velocity $v_{Doppler}$ and the calculated true velocity $v_t$ over time.
  - A plot of the difference velocity, $\Delta v = v_t - v_{Doppler}$.

  All plots should be suitably annotated.

**Expected output**

The expected output for your program when run on the provided data file is shown in Fig. 2.

Note that you should write your program in such a way that it will work for any input file with the same format as the *velocity.csv* file and with any number of lines. Your code will be tested against a different file from the one provided.

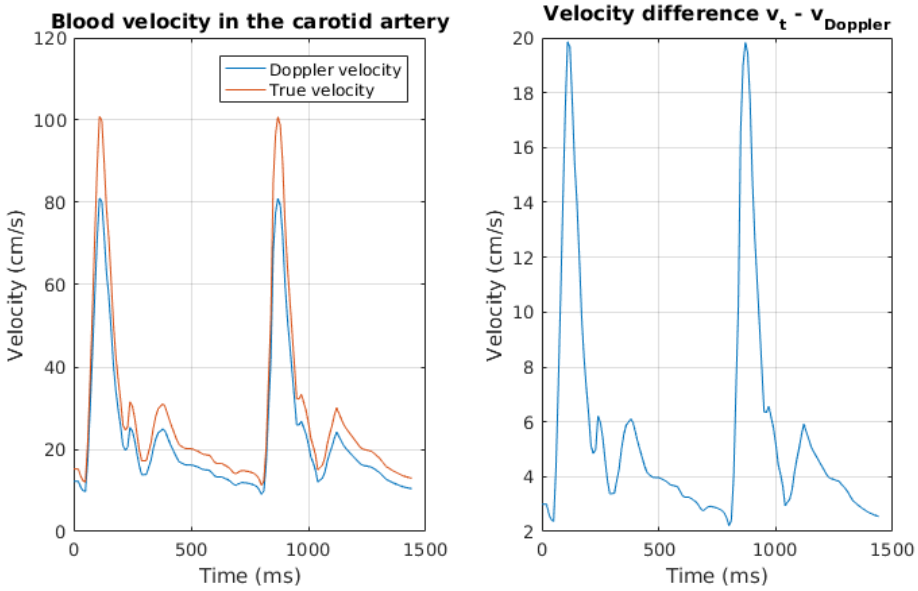



Figure 2: Expected output of the program when run on the provided data files.

### General advice

Before starting any coding, you should produce a structured design for your program, using structure charts and/or pseudocode. You should then apply an incremental development approach to develop your code.

There are many ways that this exercise can be tackled and a good approach is to break the tasks that you need to do down so that they can be implemented in multiple functions that are called by a single main function or script.

### Reporting Requirements

You should submit MatLab files that meet as many of the requirements as possible. Try to use variable/function naming, comments and indentation to make your program as easy to understand as possible.

You should also submit a short written report detailing the design of your program. This report should contain structure charts and/or pseudocode for your program design, together with a short explanation of why you chose to design the program in this way. To help you in producing this report, a sample model report will be made available to you through the KEATS system. The report should not need to be longer than 3 pages of A4, and can be shorter.

Submission will be via the KEATS system. The submission point will only allow you to upload a single file so if you have multiple files you should combine all files into a single *zip* file. Name your file *cw3-YEAR-SURNAME-FIRSTNAME.zip*, replacing *YEAR*, *SURNAME* and *FIRSTNAME* with your personal details (e.g. *cw3-1-GOMEZ-ALBERTO.zip*).

The hand-in date is **30 Nov 2016, 5 pm**. Late submissions (within 24 hours of this deadline) will be accepted but will be capped at the module pass mark (i.e. 40%).

If your program does not meet all requirements then please submit what you have written by the deadline.

**Assessment**

Your coursework will be marked on a number of factors:

- Does the program work? Does it meet all requirements? (60%)
- Program design, i.e. structure charts and/or pseudocode (20%)
- Appropriate use of MatLab language features, e.g. control structures, functions, etc. (10%)
- Use of comments, indentation and variable/function names to make code easy to understand (10%)

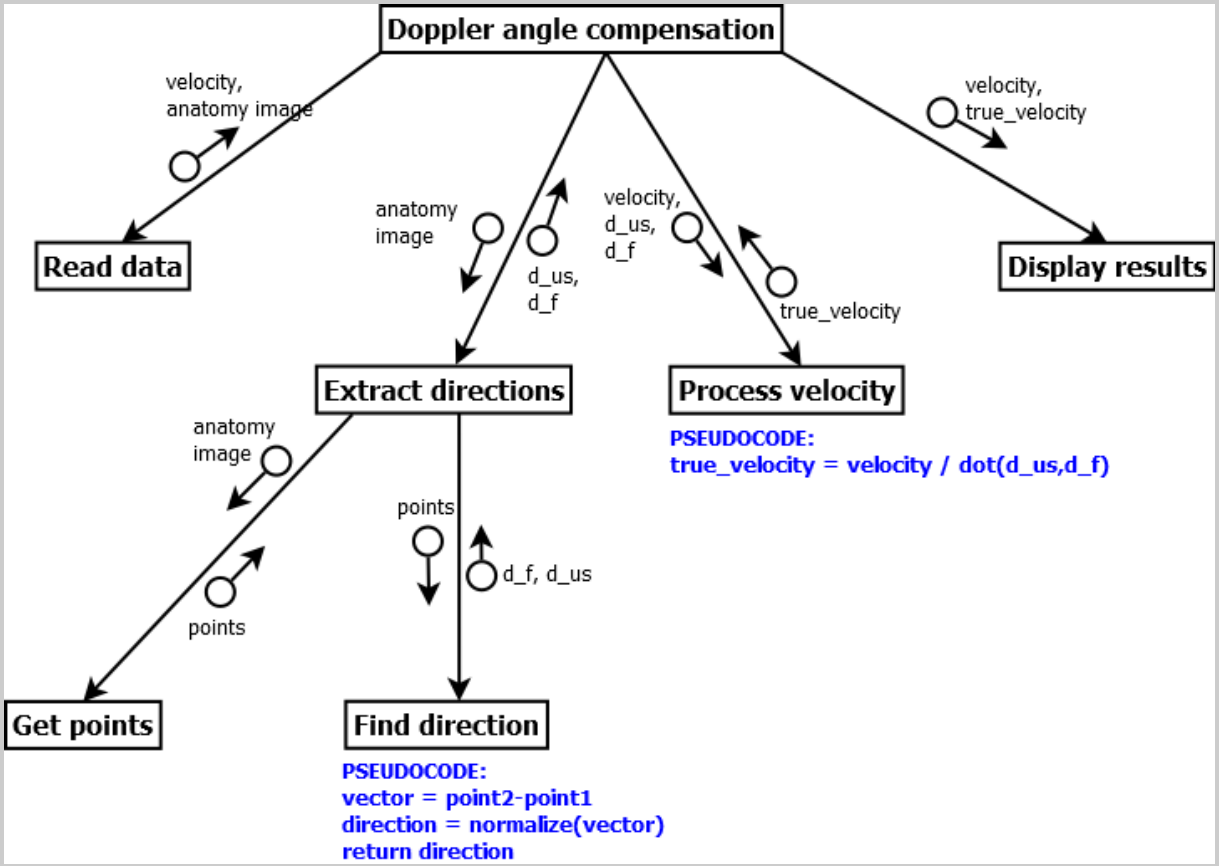
The overall mark for this coursework will make up 10% of your total mark for this module.

**This is an individual assignment. You are not permitted to work together with any other student.**

If you have any questions about this coursework please contact Alberto Gomez (alberto.gomez@kcl.ac.uk).

**Solution:**

Structure chart:




Listings:

*DopplerAngleCompensation.m*:

```
clear; close all;

% Ask user to provide filename, so code can be tested with any file
filename = input('Please enter the *.csv file name for the Doppler velocity: ...
    ','s');
```

```
% Read data into a variable with two columns of numbers, first column
% contains the times, second column contains the velocity values.
velocity = dlmread(filename);
anatomy_image = imread('anatomicImage.png');

% find US directions
d_f = findDirection(anatomy_image);
d_us = findDirection(anatomy_image);

% process velocity
true_velocity = processVelocity(velocity(:,2), d_us, d_f);

% display
figure;
subplot(1,2,1)
plot(velocity(:,1), velocity(:,2))
hold on;
plot(velocity(:,1), true_velocity)
hold off;
legend('Doppler velocity','True velocity')
xlabel('Time (ms)')
ylabel('Velocity (cm/s)')
title('Blood velocity in the carotid artery')
grid on;
%
subplot(1,2,2)
plot(velocity(:,1), true_velocity-velocity(:,2))
xlabel('Time (ms)')
ylabel('Velocity (cm/s)')
title('Velocity difference v_t - v_{Doppler}')
grid on;
```

*processVelocity.m*:

```
function true_velocity = processVelocity( velocity, d_us, d_f )
% Calculate the true velocity which, given a flow direction and
% a direction of propagation of the ultrasound produced the input
% Doppler velocity
% Usage:
%  true_velocity = processVelocity( velocity, d_us, d_f )
%  velocity: input velocity (1D array)
%  d_us: unit vector with the direction of propagation of the ultrasounds
%  d_f: unit vector with the direction of the velocity
%  d: unit vector indicating the direction of the structure in the image

true_velocity = velocity/dot(d_us, d_f);


end
```

*findDirection.m*:

```
function d = findDirection( anatomy_image )
% Find direction of a structure in the image
% by clicking 2 points which define this direction
% Usage:
%  d = findDirection( anatomy_image )
%  anatomy_image: input image of anatomy
%  d: unit vector indicating the direction of the structure in the image

% create a new, empty figure
fig = figure;
% display the input image
imagesc(anatomy_image);
```

```
    % use a grayscale colormap
    colormap(gray);
    % scale x and y axis consistently
    axis equal;
    % request the user to draw a line
    h = impoly;
    points = h.getPosition;
    close(fig);
    % calculate the vector that joins the two first points
    d = points(2,:)-points(1,:);
    % normalise the vector to unit norm
    d = d /norm(d);


    end
```