Appendix C

Matlab codes

This appendix is a compilation of all the codes that were written in MATLAB for the analysis of raw data measured with the homemade Fourier transform spectrometer. The functions are introduced in alphabetical order.

C.1 Function `centerTheBurst.m`

```
function [interferogram_centered] = centerTheBurst(interferogram)
% DESCRIPTION : centerTheBurst is a function that finds the
%               centerburst of the interferogram and places
%               it at the middle of the data.
%
% INPUT  : - interferogram is a vector containing the values of the
%            measured interferogram.
%
% OUTPUT : - interferogram_centered is a vector containing the values
%            of the centered interferogram. The vector length N is
%            the largest power of two that is smaller than or equal
%            to the length of interferogram. The maximum is placed at
%            index 2^(N-1) or 2^(N-1)+1 depending on the weight of
%            the 64 first values on both sides of the interferogram
%            maximum.

[~,index] = max(interferogram);
half_length_new = pow2(nextpow2(min(index,length(interferogram)-(
    index+1))))/2;
interferogram_short = interferogram(index-64:index+64);

weight = (-64:1:64)';
center_gravity = sum(abs(interferogram_short).*weight);

if center_gravity > 0
    interferogram_centered = interferogram(index-half_length_new:
        index+half_length_new-1);
elseif center_gravity < 0
    interferogram_centered = interferogram(index-half_length_new+1:
        index+half_length_new);
```

```
end

end
```

C.2 Function `filterTheReferenceSignal.m`

```
function signal_filtered = filterTheReferenceSignal(signal)
% Description : This function allows for digitilally filtering
%               the signal using a Butterworth filter (order 20).
%
% INPUT  : - signal is the He:Ne signal to be filtered
%
% OUTPUT : - signal_filtered is the filtered signal. It has the same
%            length than signal.

[b,a] = butter(20,[0.1 0.9],'bandpass');
signal_filtered = filter(b,a,signal);
signal_filtered = signal_filtered(21:end);
end
```

C.3 Function `filterTheSignal.m`

```
function signal_filtered = filterTheSignal(signal,wavelength_max,
    wavelength_min)
% DESCRIPTION : This function digitilally filters the signal
%               using a Butterworth filter (order 5).
%
% INPUTS : - signal is the signal to be filtered
%          - wavelength_max is the maximum wavelength in nm of the
%            bandpass filter
%          - wavelength_min is the minimum wavelength in nm of the
%            bandpass filter
%
% OUTPUT : - signal_filtered is the filtered signal. It has the same
%            length than signal.


% cutoff frequencies
w1 = 632.8/wavelength_max; % low frequency
w2 = 632.8/wavelength_min; % high frequency

[b,a] = butter(5,[w1 w2],'bandpass');
fvtool(b,a); % check the magnitude response of the filter

signal_filtered = filter(b,a,signal);
signal_filtered = signal_filtered(21:end);
end
```

C.4 Function `findTheZeros.m`

```
function [index_zero_crossing] = findTheZeros(position,signal)
% DESCRIPTION : Compute an estimation of the zero-crossing locations
%               of the reference laser. They are obtained by fitting
%               the signal on an even number of points on both sides
%               of the zero-crossing and by finding the roots of this
%               polynomial.
%
% INPUTS  :     - position is an ordered vector containing the index
%                 of the values in signal
%               - signal corresponds to the interferogram of the
%                 reference laser which is sampled at constant
%                 intervals of times. (The dc component has already
%                 been removed, i.e. the mean of the signal is
%                 equal to zero)
%
% OUTPUTS :     - index_zero_crossing is a vector containing the
%                 indices at which signal is vanishing. These
%                 indices are real numbers at which the interferogram
%                 of the other source must be interpolated.

warning('off','all');

% Interpolation of the signal. The amount of data is increased
% by a factor 8 (Griffiths (2002)).
signal = interp(signal, 8);

% The first 8 values are not considerated because they were checked
% in the previous set of data on which the function has already been
% applied.
% The 7 last values are not considerated since they only contain
% extrapolated values resulting from the application of
% the function interp.
location = find(diff(sign(signal(9:end-7)))) + 8;

% number of points for the curve fitting.
% /!\ data can only take even values that are smaller or equal to 16
data = 10;

% Initialization of variables and vectors necessary for the loop :
k = 0;
index_zero_crossing = zeros(length(signal),1);

for j = 1:length(location)
    % Definition of the position vector for the cubic polynomial fit.
    % An even number of points on both sides of the zero-crossing is
    % considered.
    % -The position vector is multiplied by 8 using the interp
    %  function, the last index before the zero-crossing is given
    %  by 1+floor(location(j)/8).
```

```
    % -There are ((mod(location(j),8)-1) intervals of 0.125 between
    %  the last position point and the point before the zero-crossing
    position2 = (position(1+floor(location(j)/8)) + 0.125*((mod(
        location(j),8)-1))+ 0.125*((-data/2+1):(data/2)))';

    % Definition of the signal vector for the cubic polynomial.
    % An even number of points on both sides of the zero-crossing
    % are considered.
    signal2 = signal((location(j)-data/2+1):(location(j)+data/2));

    % Determine the polynomial that best fits the data
    polynomial = polyfit(position2,signal2,3);

    % Find the 3 roots of the cubic polynomial.
    racine = roots(polynomial);
    % The root that is considered as a zero-crossing is the one which
    % is the nearest to the two values located at each side of the
    % detected zero-crossing.
    [~,index] = min((racine-position2(data/2)).^2 + (racine-position2
        (data/2+1)).^2);

    % The zero-crossing is saved in the output vector
    k = k+1;
    index_zero_crossing(k) = racine(index);

end

% Suppress the zeros remaining at the end of the vector. A value was
% assigned to the k first points.
index_zero_crossing = index_zero_crossing(1:k);

warning('on','all');
end
```

C.5 Function `getTheSpectrum.m`

```
function [wavenumber,spectrum] = getTheSpectrum(interferogram)
% DESCRIPTION : This function computes the normalized irradiance from
%               the interferogram. The phase is corrected by
%               multiplying the spectrum by exp(-phase).
%
% INPUT  : -interferogram is a vector containing the values of the
%           centered interferogram, OPD is constant between two
%           consecutive data points.
%
% OUTPUT : -wavenumber is a vector containing the wavenumber values
%           associated to spectrum
%          -spectrum is the normalized spectrum associated to the
%           interferogram.
```

```
% Fourier transform
fourier_transform = fft(interferogram);

% Suppression of the remaining dc term
fourier_transform(1) = 0;

% phase correction
jj = sqrt(-1);
fourier_transform = fourier_transform.*exp(-jj*angle(
    fourier_transform));

% definition of the wavenumber axis
N = length(fourier_transform);
cst = 2/632.8e-7;
wavenumber = (cst*(0:(N/2-1))/N)';

spectrum = real(fourier_transform(1:(N/2)))/max(real(
    fourier_transform(1:(N/2))));

end
```