CEN/TC 227

Date: 2016-03

Cen/wd 2016

CEN/TC 227

Road and airfield surface characteristics — Test methods — Part 5: Determination of longitudinal unevenness indices

Document type: European Standard
Document subtype:
Document stage: Preparation
Document language: E

Contents

Page

Foreword

This document (prEN 13036-5:2006) has been prepared by Technical Committee CEN/TC 227 “Road materials”, the secretariat of which is held by DIN.

This document is a working document.

Introduction

Through road/vehicle dynamic interaction and vehicle vibration, the road profile evenness affects safety (tyre contact forces), ride quality, energy consumption, vehicle wear as well as pavement and road durability. The road profile evenness is consequently key information for road maintenance-management-systems and performance control.

The purpose of this document is to provide a standard practice for calculating and reporting estimates of road evenness from digitized longitudinal profiles.

This practice covers the mathematical processing of longitudinal profile measurements to produce evenness statistics (indices) covering the wavelength range 0.5 to 50 meter. These wavelengths cover most situations for cars*.The practice describes the calculation procedure for the International Roughness Index (IRI), wave band analysis (Root Mean Square (RMS) and Longitudinal Profile Variance (LPV)) and the Weighted Longitudinal Profile (WLP).

The purpose of the practice is to ensure that when applying one of the possible procedures, exactly the same steps are carried out, with the aim of facilitating the comparison of evenness measurements carried out with different profiling instruments in European countries.

NOTE: As a control of the implementation of calculation of the evenness indices three longitudinal profiles are available including the “true values”. They can be found at www.erpug.org in the directory *reference profiles*.

** For higher speed, e.g. on airport runways or highways longer wavelengths may also be important.*

1 Scope

This European standard specifies the mathematical processing of digitized longitudinal profile measurements to produce evenness indices. The document describes the calculation procedure for the International Roughness Index (IRI), Root Mean Square (RMS) and Longitudinal Profile Variance (LPV) from three separate wavelength bands and the σWLP and ΔWLP from the Weighted Longitudinal Profile (WLP).

The purpose of this document is to provide a standard practice for calculating and reporting estimates of road evenness from digitized longitudinal profiles. Other aims with the standard are to facilitate the comparison of evenness measurement results carried out with different profiling instruments in European countries.

The evenness range covered in this standard is defined as the wavelength range 0.5 to 50 metres. It should be noted that both shorter and longer wavelengths can also influence the driving comfort but those are not covered in this standard.

The quantified evenness indices derived from the standard are useful support for pavement management systems. The output can also be used for type approval and performance control of new and old pavements. The indices can be used on rigid, flexible and gravel road surfaces.

The standard does not define from what position on the road the profile should be obtained.

The derived indices are portable in the sense that they can be obtained from longitudinal profiles measured with a variety of instruments.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

EN 13036-6, *Road and airfield surface characteristics Test methods Part 6: Measurement of transverse and longitudinal profiles in the evenness and megatexture wavelength ranges*.

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1
profile
is the intersection between the surface of the pavement and the plane which contains both the vertical of the measured pavement and the line of travel of the measuring instrument; when the measuring instrument travels in a curve the line of travel is the tangent to that curve, when travelling in a straight line it is this line. In this plane, a point of the profile can be adequately described by its coordinates x (abscissa) and z (elevation), in any orthonormal reference system (X, Z), where Z is parallel to the aforementioned vertical.

3.2
profilometer
is an instrument to measure and collect profiles representing evenness from e.g. roads.

3.3
spatial sampling interval
is the absolute value of the difference of abscissa between two adjacent points of the digitised longitudinal profile line. This definition assumes that the distance measured by the profilometer, which is usually related to the curvilinear abscissa, is close enough to the abscissa in the mathematical sense.

3.4
longitudinal road profile
is one of the profiles obtained when the measuring instrument travels in the same direction as the traffic. Usually one of the profiles measured in the wheel tracks is used.

3.5
longitudinal evenness
is the deviation of the longitudinal profile from a straight line in a defined wavelength range e.g. 0.5 m to 50 m.

3.6
raw profile
is the profile given by a profilometer when measuring a longitudinal road profile. Characteristics of the raw profile depend on the profilometer used.

3.7
pre-processed profile (re-sampled and filtered profile)
is obtained by applying the resampling and filtering procedures. The pre-processing procedure aims to harmonize the profile provided by various devices and is recommended in the case of benchmark use.

3.8
wavelength
is the distance between two identical adjacent points in a wave. It is typically measured between two easily identifiable points, such as two adjacent crests in a waveform. While wavelengths can be calculated for many types of waves, they are most accurately measured in sinusoidal waves, which have a smooth and repetitive oscillation. The wavelength of a sinusoidal wave is the spatial period of the wave, the distance over which the wave's shape repeats. In most cases the profile can be adequately described as a sum of sine functions, when this is possible one such sine function is

$$A \times \sin\left(\frac{2\pi}{\lambda}(x - x_0)\right)$$ **Equation 3-A**

Where

λ is the wavelength of the sine in metres (m);

A is the amplitude of the sine in metres (m);

x is the abscissa of the current point, in metres (m);

x_0 is the phase of the sine, in metres (m).

3.9
spatial frequency
is the reciprocal of a wavelength in cycles per metre. The spatial frequency N defines the number of waves, of wavelength λ, per metre:

$$N = \frac{1}{\lambda}$$ **Equation 3-B**

3.10
standard reference sampling interval
is the spatial sampling interval which must be used when performing the spectral analyses. The minimum interval depends on the wavelength, which should be analysed.

3.11
measuring track or position
is the selected intersection, of all possible profiles along the transverse direction, to be used.

3.12
profile measurement length
is the length of an uninterrupted profile measurement. It expresses the length over which the profilometer continuously and accurately digitises and records the profile (from point B to C in Figure 1). Most profilometers need to run for some minimum distance before and after the profile they are to measure, these starting (from point A to B in Figure 1) and ending phases (from point C to D in Figure 1) should not be included in the profile measurement length.

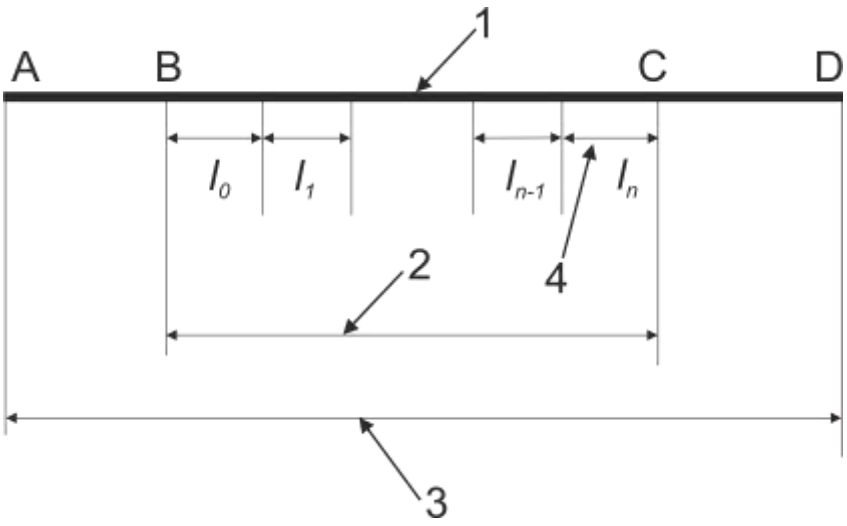

Figure 1 — Profile lengths definitions

Key

1 road surface

2 B to C, profile measurement length

3 A to D, overall profilometer route

4 $l_0 \ldots l_n$, evaluation or reporting length

3.13
evaluation or reporting length
is the measurements made over the profile measurement length which are often analysed using shorter parts or samples (key 4, l_0 to l_n in Figure 1) to allow for a more precise description of the measured profile. The evaluation or reporting length is the length of such a sample.

NOTE *In the case of consecutive samples such as l_0 and l_1 in Figure 1, over the profile measurement length, the word "segment" is used.*

4 Symbols and abbreviations

B is the base used for IRI calculation in metres (m). It is the length over which the IRI calculation is performed (or reporting length using the terminology of this document)

L denotes the measurement length

N is the spatial frequency, in cycles per metre: $N = \frac{1}{\lambda}$; N is usually called a wave number

x_i is the abscissa of the sampled point i, in metres (m)

z_i is the elevation of the profile determined at the sampling point i, in metres or millimetres

δ_x is the spatial sampling interval for the digitisation of the profile, in metres or millimetres

λ is the wavelength, in metres (m)

Δ sample interval

IRI is the International Roughness Index

WB is the wave band index calculated by using root mean square analysis applied to the pre-processed profile elevations for the wave band W, in metres

SW is the Root Mean Square value for the short wavelength band

MW is the Root Mean Square value for the medium wavelength band

LW is the Root Mean Square value for the long wavelength band

w is the waviness of the reference spectrum used for the WLP

WLP is the Weighted Longitudinal Profile

σWLP is the standard deviation of the WLP

ΔWLP is the range of variation of the WLP.

5 Calculation of evenness indices

A profile can be obtained starting from any lateral position in the lane (3.4). This standard doesn't specify which of those profiles that should be used, only how to calculate any of the four evenness indices specified.

The calculation of evenness indices, involves the following steps:

- the measurement of the profile, which includes identification of the start and end points of the profile measurement length and possible events (e.g. roundabouts, speed bumps, milestones, etc);
- if the purpose is benchmarking the profile should be resampled with a 0.05 m step. It is essential to consider the method of resampling to avoid erroneous energy in the signal.
- filtering and resampling of the raw profile, the output of which is a pre-processed profile, for the calculation of indices;
- the calculation of one or more indices;
- the creation of a report.

Pre-processing including resampling and filtering is essential in the case of wave band analysis and is recommended to homogenize the profiles and facilitate comparisons. For benchmark purposes a resampling procedure according to the resample function in Matlab® is needed. For other use a linear model could be used. Depending on purpose there are four possible evenness indices groups to choose from. The first is the calculation of the International Roughness Index, IRI, described in Annex A that should be used in international comparison and benchmark tests, second is calculating evenness-energy in three different wavelength bands using the bi-octave processing (RMS) described in Annex C. The third is an alternative method to calculate evenness-energy in three wave band bands using profile variance (LPV) in Annex D. Finally the fourth method is to calculate the range of evenness-variation and deviation using the weighted longitudinal profile (WLP) described in Annex E. In Annex B an example code to calculate IRI is presented. The general process on calculation of evenness indices is illustrated in Figure 2.

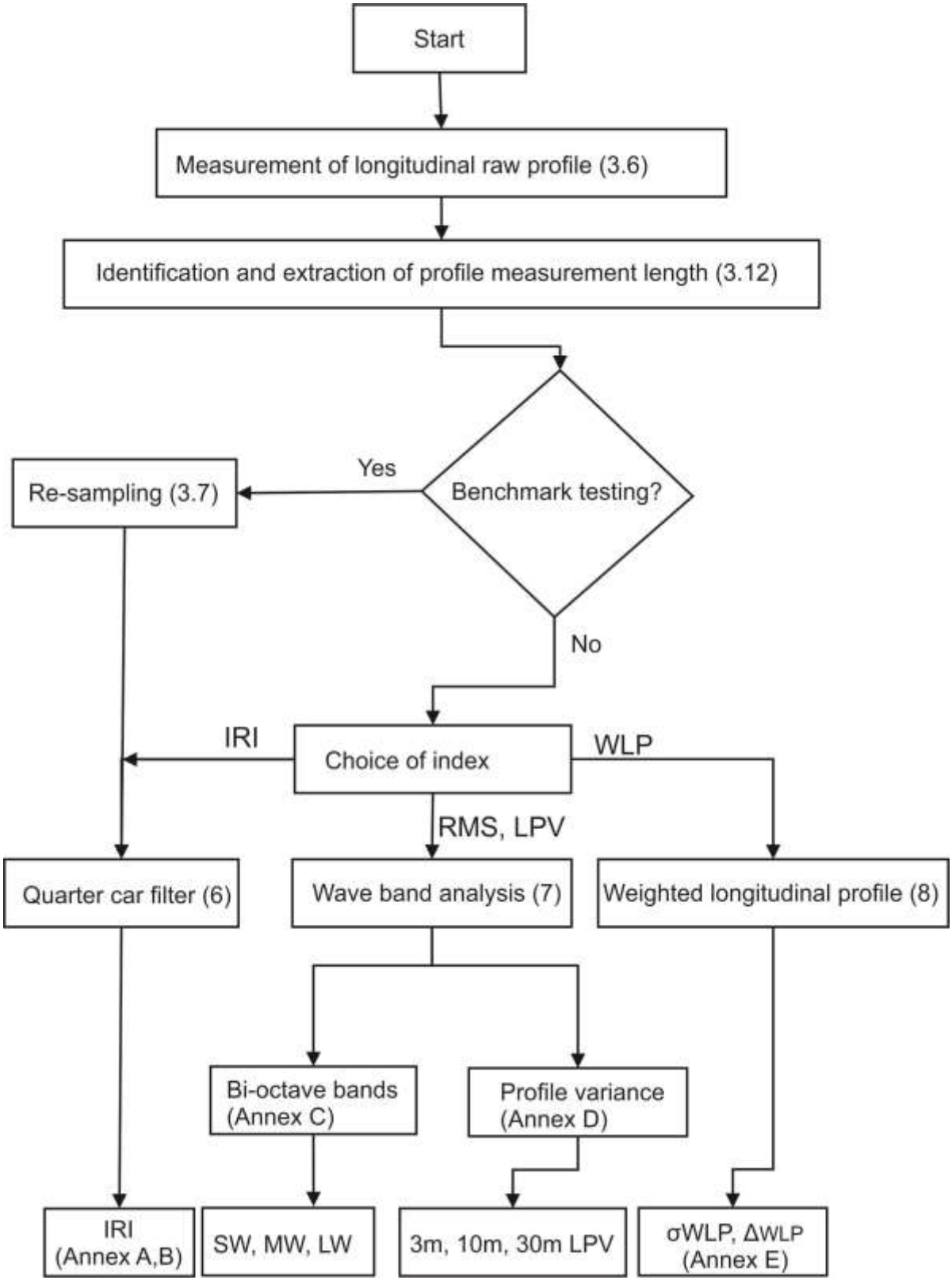

Figure 2 Overview of the calculation process of indices (references to chapter in paranthesis**)**

6 International Roughness Index (IRI)

6.1 General

The IRI is an index computed from a longitudinal road profile measurement using a virtual response type system, quarter-car simulation, running at a speed of 80 km/h, see Figure 3. The quarter-car simulation applied on the digitised road profile calculates the accumulated suspension motions divided by the distance travelled. The description of the IRI calculation is based on [4]:

- IRI is computed from a single longitudinal road profile. The sampling interval should be no larger than 125 mm for accurate calculations. The required vertical sensor resolution depends on the evenness level, with finer resolution being needed for smooth roads. A vertical sensor resolution of 0.5 mm is suitable for all conditions;
- The profile is assumed to have a constant slope between sampled elevation points;
- The profile is smoothed with a moving average whose base length is 250 mm;
- The smoothed profile is filtered using a quarter-car simulation, with specific parameter values, at a simulated speed of 80 km/h;
- The simulated suspension motion is linearly accumulated and divided by the length of the profile to yield IRI. Thus, IRI has units of slope, such as millimetres per meter or metres per kilometre;

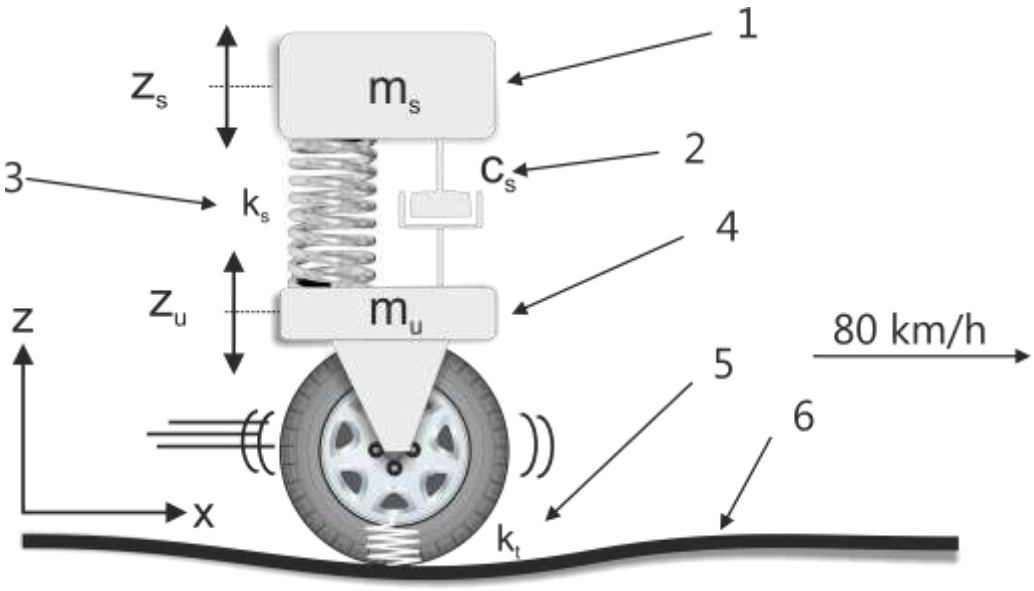

Figure 3 Quarter car (virtual response type system)

Key

1	sprung mass m_s	4	unsprung mass m_u
2	suspension damping rate c_s	5	tyre spring rate k_t
3	suspension spring rate k_s	6	longitudinal profile $Z(x)$

More details on the method of IRI calculation and sample code are given in Annex A and Annex B.

6.2 Representation of the obtained results

The IRI can be calculated with a different evaluation length L. For a more detailed analysis of the road profile data one may wish to vary the evaluation length L. Therefore, this should be indicated as the first sub index e.g.

IRI_{20} if the length L = 20 m. For the purpose of European benchmarking 100 m evaluation length is recommended and should be denoted IRI_{100}.

Variations in IRI parameters, (Figure 3) or the simulation speed are beyond the scope of this standard. If however for whatever reason the Quarter Car Model is calculated with other parameters than those mentioned, it should not be denoted as IRI.

7 Wave band analysis

7.1 General

In order to perform wave band analysis, the pre-processed profile is split into different wave band limited profiles using filters, see Figure 4. The wave bands are generally selected to represent different features of ride quality. For example, applying a short wavelength filter can result in an index that reflects the presence of small undulations in the road surface, which may be more significant at lower speeds. In contrast, applying a longer wavelength filter can result in an index that reflects the presence of long wavelength undulations that would be most likely to affect ride quality at higher speeds.

The definition of the wave bands used as well as the characteristics of the filters used to obtain band limited signals, from the original longitudinal profile must be reported. How indices are derived from the band limited signals must also be defined.

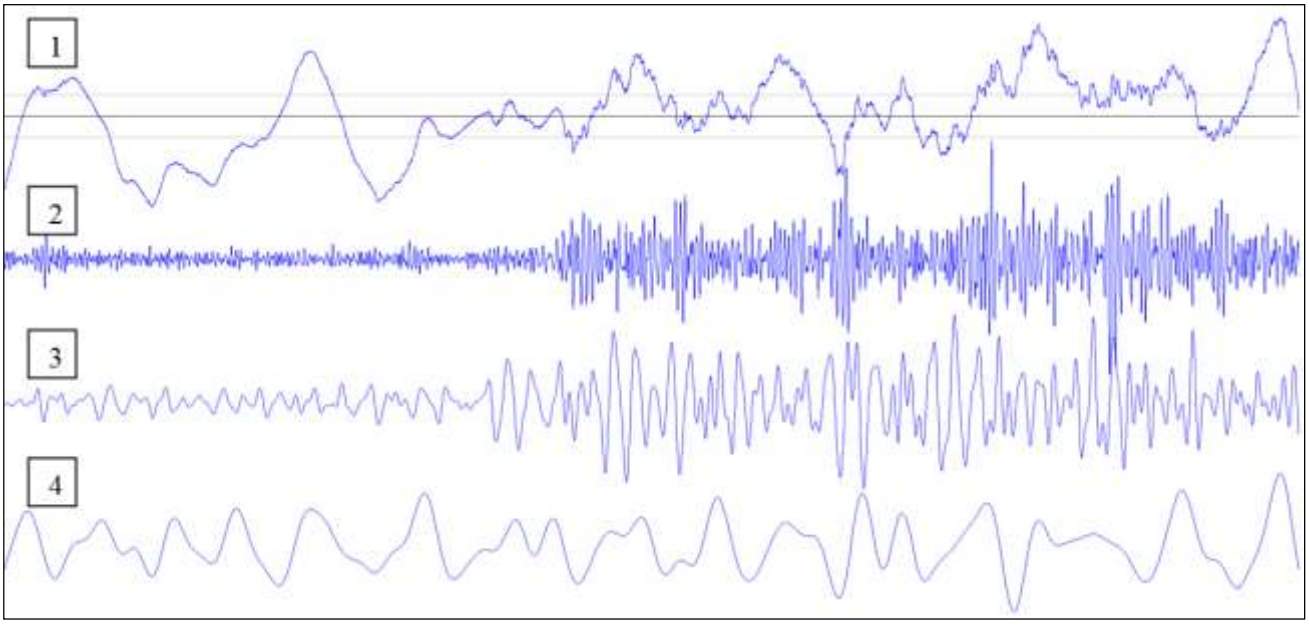

Figure 4 Wave band splitting

Key

1 pre-processed profile
2 short wavelength filtered profile
3 medium wavelength filtered profile
4 long wavelength filtered profile

The filters applied to the longitudinal profile are generally either

- Band-pass filters, where wavelengths outside of a defined range are attenuated; or
- High-pass filters, where wavelengths greater than a defined wavelength are attenuated.

The filters used to break the original longitudinal profile into the previously defined bands, should be carefully chosen in order to introduce as little distortion as possible in the filtered signals, a common technique in that view is to use digital forward and reverse filtering associated with measured section extending beyond the profile which is to be assessed.

7.2 Wave band indices

In order to characterise the different wave band filtered profiles two methods including indices are specified:

- Root Mean Square(RMS) over bi-octave bands (see Annex C)
- Longitudinal profile variance (LPV) over selected wavelengths (see Annex D).

8 Weighted Longitudinal Profile (WLP) analysis

8.1 General

The Weighted Longitudinal Profile (WLP) is the longitudinal pavement profile which has been weighted by a weighting function in the frequency domain. The weighting function enhances small wavelengths and decreases large wavelengths in such a way that their respective power contents become measurable later by the same scale in the spatial domain. Following the weighting of the spectrum, the WLP is calculated by carrying out the following steps:

- filtering the weighted signal in octave bands,
- multiplying the octave band-filtered signals by pre-factors (which take into account their respective power distributions to the total power content), and,
- adding up the signals to give the WLP.

The WLP is characterized by the standard deviation, σWLP, and the range of variation, ΔWLP.

8.2 Prerequisites

The WLP calculation is based on the pre-processed profile. The pre-processing involves:

- De-trending the raw profile using a linear regression to remove the offset and trend of the signal.
- Resampling by applying a constant interval of 0.1 m, using a linear interpolation algorithm
- Pre-filtering by applying a high pass filter to attenuate wavelengths in excess of 100m. The filter shall be such that the amplitude of wavelengths greater than 150m are attenuated by at least 50%. The filter should not distort the phase of any profile features with wavelengths shorter than 100m.

9 Quality control

As a support for controlling the implementation of the unevenness indices calculation, three longitudinal profiles are available. The profiles represent low, medium and high unevenness. A true value for each is also available. The longitudinal profiles are 1000 meter long divided into 0.125 meter steps and have the vertical unit millimetre. They can be downloaded from www.erpug.org under the directory *reference profiles*. The true IRI value is calculated with dx=0.125 m and presented as an average per 20 meter.

10 Reporting

The calculated evenness index should be presented per an agreed section length. In the case of international benchmarking this should be 100 meter. This could be presented as a sub-index described in chapter 3.12. It is recommended to include the following administrative information:

- Geographic position and information about the measured object
- The lateral position of longitudinal profile, e.g. right or left wheel path
- Distance measured
- Date and time
- Deviating conditions that could affect the result

The following information must be included regarding class of the profilometer used, as specified in EN 13036-6:

- the vertical sensor resolution
- travelled distance accuracy
- sampling interval

- larger wavelength cut-off
- reporting sampling interval
- the presentation length used

Finally any deviation from this standard, 13036-5 should be documented and reported.

11 Bibliography

[1] Sayers, M.W., T.D. Gillespie, and Querioz.C.A.V., International Experiment to Establish Correlations and Standard Calibration Methods for Road Roughness Measurements. World Bank Technical Paper 45. World Bank, Washington, D.C., 1986

[2] Sayers, M.W., T.D. Gillespie, and W.D., Paterson Guidelines for the conduct and calibration of Road Roughness Measurements. World Bank Technical Paper 46. World Bank, Washington, D.C., 1986

[3] Karamihas S.M., and T.D. Gillespie et al. “Guidelines for Longitudinal Pavement Profile Measurement”, TRB - NCHRP Report 434, Washington D.C., 1999,

[4] Sayers, M.W., “On the calculation of International Roughness Index from Longitudinal Road Profile” Transportation Research Record 1501, Transportation Research Board, pp. 1-12”, Washington D.C., 1995

[5] ASTM, “Computing International Roughness Index of Roads from Longitudinal Profile measurements”, E1926-98, ASTM Standards

[6] Descornet, G: “Inventory of High-Speed Longitudinal and Transverse Road Evenness Measuring Equipment in Europe”, BRRC editor. FEHRL Technical Note 1999/01, Brussels 1999.

[7] Willet M, Magnusson G., Ferne B., “FILTER- Theoretical study of Indices”, FEHRL Technical Note 2000/2

[8] de Witt, Kempkens E, Sjögren L., Ducros D.M.,”The FILTER Experiment”, FEHRL Technical Note 1999/2

[9] Ducros D.M.,Petkovic L., Descornet G., Berlémont B., Alonso M., Yanguas S., Jendryka W., Andrén P, “FILTER Experiment – Longitudinal Analyses”, Final Report 2001/1, FEHRL

[10] International Experiment to Harmonize Longitudinal and Transverse Profile Measurement and Reporting Procedure (The EVEN Project), PIARC Technical report 01.07.B, 2002

[11] Oppenheim A.V., Schafer R. W., “Digital Signal Processing”, Prentice Hall inc. editor, Englewood Cliffs, New Jersey, USA, 1975

[12] Feuer A., Goodwin G.C., “Sampling in Digital Signal Processing and Control, Birkhäuser editor, Boston, USA, 1996

[13] Ueckermann A., Steinauer B., “The Weighted Longitudinal Profile”, Road Materials and Pavement Design 9:2 (2008), 135 – 157

[14] Ueckermann, A.; "Ein geometrisch basiertes Verfahren zur Lokalisierung und Bewertung einzelner, periodischer und regelloser Unebenheiten im Straßenlängsprofil"; Aachener Mitteilungen Straßenwesen, Erd- und Tunnelbau; Lehrstuhl für Straßenwesen, Erd- und Tunnelbau und Institut für Straßenwesen der RWTH Aachen; Number 44; 2004

Annex A

Calculation of the IRI

The IRI calculation

The IRI calculation includes two distinct filters: a moving average and a quarter-car model.

Moving Average Filter

The moving average filter was included for two reasons; to simulate the enveloping behaviour of pneumatic tyres on motor vehicles and to reduce the sensitivity of the IRI algorithm to the sample interval, Δ. For a profile that has been sampled at Δ, a moving average smoothing filter is defined by the summation (Equation A.1):

$$h_{ps}(i) = \frac{1}{k}\sum_{j=1}^{i+k-1} h_p(j)$$

$$k = \max\left[1, \text{nint}\left(\frac{L_B}{\Delta}\right)\right]$$

Equation A.1

where:
h_p = profile height;
h_{ps} = smoothed profile height;
max = maximum of two arguments;
nint = nearest integer, and
L_B = moving average base length, 250 mm.

The quarter-car model parameters defining IRI

The IRI is calculated using the quarter-car model. It includes the major dynamic effects that determine how unevenness causes vibrations in a road vehicle. The masses, springs, and dampers are defined by the following parameters:

c_s = suspension damping rate

k_s = suspension spring rate

k_t = tyre spring rate

m_s = sprung mass (portion of vehicle body mass supported by one wheel)

m_u = unsprung mass (mass of wheel, tyre, and half of axle/suspension)

To simplify the equations, the parameters are normalized by the sprung mass, m_s. The following values for the normalized parameters define the IRI set:

$c = c_s/m_s = 6.0$ $\quad k_1 = k_t/m_s = 653$

$k_2 = k_s/m_s = 63.3$ $\quad \mu = m_u/m_s = 0.15$

Equation A.2

The quarter-car model is described by four first-order ordinary differential equations that can be written in matrix form (Equation A.3).

$$\dot{\mathbf{x}} = \mathbf{A}\mathrm{x} + \mathbf{B}h_{ps}$$ **Equation A.3**

where the **x**, **A**, and **B** arrays are defined as follows:

$$\mathbf{x} = \begin{bmatrix} z_s & \dot{z}_s & z_u & \dot{z}_u \end{bmatrix}^T$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -k_2 & -c & k_2 & c \\ 0 & 0 & 0 & 1 \\ \frac{k_2}{\mu} & \frac{c}{\mu} & -\frac{k_1 + k_2}{\mu} & -\frac{c}{\mu} \end{bmatrix}$$ **Equation A.4**

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & \frac{k_1}{\mu} \end{bmatrix}^T$$

where

h_{ps} = smoothed profile elevation,

z_s = height (vertical position) of the sprung mass,

z_u = height (vertical position) of the unsprung mass, and

x = array of state variables (i.e. the variable that completely describe the stat of the simulated system).

Time derivatives are indicated with a dot (e.g. $\dot{z}_s$). Time is calculated from the longitudinal distance and the simulated speed V, which is defined as 80 km/h.

The IRI is an accumulation of the simulated motion between the sprung and the unsprung masses in the quarter-car model, normalized by the length L of the profile:

$$IRI = \frac{1}{L}\int_0^{L/V} \left|\dot{z}_s - \dot{z}_u\right| dt$$ **Equation A.5**

Solving the differential equation

To solve differential equations such as Equation A.3, one should know the initial values of the state variables at the starting time. In the standard IRI simulation, the initial values influence the quarter car response for about 20 m. In order to estimate the initial values, z_s and z_u should be set to match the height of the first profile point and $\dot{z}_s$ and $\dot{z}_u$ should be set to match the average change in profile height per second, at the simulation speed over the first 11 m of profile. The initialization of the simulation should start on the profile data acquired at least 20 m before the start of the section to be evaluated. At the start of the test site, the IRI accumulation (5) shall start.

The quarter-car dynamics and the initialization method together make up a type of calculation whose generic name is "initial value problem" or "integration of ordinary differential equations" for which several solving methods are available. The accuracy of these methods may vary, for which this standard prescribes one method which is referred to in Sayers (1995) as the “recommended algorithm”. The algorithm is described in the following paragraphs. A computer code suitable for numerical computing programs like Matlab® or GNU Octave is included in the end of this Annex B. The original FORTRAN computer code can be found in the referenced article.

For a set of linear equations such as Equation A.1, the total response at point *i* is the sum of the free response (no input) of the system to its state at a previous point *i*-1, plus the forced response to an input over the interval between points *i*-1 and *i*. In the case that the input is a constant, the closed-form solution is known:

$$\mathbf{x}_i = e^{\mathbf{A}\Delta/V}\mathbf{x}_{i-1} + \mathbf{A}^{-1}\left(e^{\mathbf{A}\Delta/V} - \mathbf{I}\right)\mathbf{B}u \qquad \textbf{Equation A.6}$$

where:

e **=** the base of the natural logarithms

I = 4 x 4 identity matrix, and

u = input that is constant over the interval i-1 to i.

The exponential of the matrix can expressed as a Taylor series expansion:

$$e^{\mathbf{A}\Delta/V} = \mathbf{I} + \sum_{i=1}^{N} \frac{\mathbf{A}^i\left(\Delta/V\right)^i}{i!} \qquad \textbf{Equation A.7}$$

where N is a number large enough that the elements of the state transition matrix are correct to within the precision of the computer (typically N is about 10). Equation A.6 is exact to the extent that the input *u* is actually constant over the interval from point i-1 to i. The slope between two points of the profile is constant. This means that to obtain the best accuracy, the assumed constant input *u* in equation 6 should be profile slope. Then, Equation A.6 is the solution for the differential equation:

$$\dot{\mathbf{x}} = \mathbf{A}\mathrm{x} + \mathbf{B}s_{ps} \qquad \textbf{Equation A.8}$$

where s_{ps} is the smoothed profile slope. However, replacing h_{ps} with s_{ps} implies that the array **x** is redefined as

$$\mathbf{x} = \begin{bmatrix} s_s & \dot{s}_s & s_u & \dot{s}_u \end{bmatrix}^T \qquad \textbf{Equation A.9}$$

where s_s and s_u are filtered slope variable associated with the sprung and unsprung masses.

The IRI definition of equation A.5 can now be rewritten as:

$$IRI = \frac{1}{n}\sum_{i=1}^{n}\left|s_{s,i} - s_{u,i}\right| \qquad \textbf{Equation A.10}$$

Where *n* = number of profile samples

The smoothed profile slope, based on a 250 mm interval, is computed as follows:

$$s_{ps,i} = \frac{h_{p,i+k} - h_{p,i}}{k\Delta} \qquad \textbf{Equation A.11}$$

where *k* was defined in Equation A.1.

The IRI definition of Equation A.10 can be rewritten in matrix form as follows:

$$IRI = \frac{1}{n}\sum_{i=1}^{n}\left|\mathbf{C}\mathbf{x}\right| \qquad \textbf{Equation A.12}$$

where

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & -1 & 0 \end{bmatrix}$$

Equation A.13

To initialize the algorithm, the elements of the **x** array for *i* = 1 are set as

$$\mathbf{x}_1 = \begin{bmatrix} \left(h_{p,L_0/\Delta} - h_{p,1}\right)/L_0 & 0 & \left(h_{p,L_0/\Delta} - h_{p,1}\right)/L_0 & 0 \end{bmatrix}$$

Equation A.14

where L_0= 11 m.

Annex B
(informative)

Example code for IRI calculation

The Matlab® code below will compute the IRI. The `irivec` output will have the same length and sample distance as the input profile. `irivec` can later be averaged to a longer presentation length. The code has been verified with the FORTRAN code from the World Bank report [2] and gives the same result to numerical noise.

```
function irivec = iri(prof, dx)
%IRI Calculate the IRI from a height input profile.
%   irivec = iri(prof, dx)
%   'prof' is given in millimeters and 'dx' in meters.

% Set parameters.
base    = 0.25;
xinit   = 11;

% Physical Constants Of Quarter Car.
K1      = 653.0;     % kt/ms
K2      = 63.3;      % ks/ms
U       = 0.15;      % mu/ms
C       = 6.0;       % cs/ms
V       = 80/3.6;    % speed

% The wheel-road contact moving average distance.
ibase   = max(round(base/dx), 1);

% Initialize simulation variables.
ilead   = min(round(xinit/dx) + 1, length(prof));
X       = [(prof(ilead) - prof(1))/(dx*ilead); 0; (prof(ilead) - prof(1))/(dx*ilead); 0];

% The slope profile.
sprof   = (prof((ibase + 1):(length(prof))) - prof(1:(length(prof) - ibase)))./(dx*ibase);

% State Space Matrix 'A'.
A       = [    -0,      1,            0,    0;
               -K2,    -C,           K2,    C;
               -0,     -0,            0,    1;
               +K2/U, C/U, -(K2 + K1)/U, -C/U];

% State Space Matrix 'B'.
B       = [0; 0; 0; K1/U];

% Calculate the State Transition matrix 'ST' and the Partial Response vector 'PR'.
ST      = expm(A*dx/V);
PR      = inv(A)*(ST - eye(4))*B;

% Initialize the IRI vector.
irivec  = zeros(length(sprof), 1);

% Calculate the IRI.
for i = 1:length(sprof)
  % Calculate the new state vector.
  XN            = ST*X + PR*sprof(i);
  X             = XN;
  irivec(i)     = abs(X(3) - X(1));
end
```

Annex C
(informative)

Wave band analysis using bi-octave bands and RMS

This process can be applied to each reporting length or preferably to the whole profile measurement length in order to limit filtering induced edge effects. A C++ program, designed to perform the filtering of the profile, is presented later in this appendix.

All the filter descriptions follow the usual signal processing conventions [11], [12], where a filter response is characterised by its normalised transfer function defined as:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b(1) + b(2)z^{-1} + b(3)z^{-2} + \ldots + b(n+1)z^{-n}}{a(1) + a(2)z^{-1} + a(3)z^{-2} + \ldots + a(n+1)z^{-n}} \text{ with } a(1) = 1$$

Equation C.1

The following steps are applied to the raw profile (P):

Resampling and filtering

The profile (P) is de-trended using a linear regression, to remove the offset and trend of the signal creating a de-trended profile (Pd). The profile (Pd) is resampled and pre-filtered by applying a resampling interval of 0.05 m, using a linear interpolation algorithm to obtain Prs. A high-pass filtered profile Php is obtained by applying an IIR (Infinite Impulse Response) general high pass filter limited to λ=100 m (Butterworth 4th order) to the profile Prs, including the use of a forward and reverse filtering to prevent phase problems. The high pass filter should be applied to the measured Longitudinal Profile to attenuate wavelengths in excess of 100 m. The filter shall be such that the amplitude of wavelengths greater than 150 m are attenuated by at least 50%. The filter should not distort the phase of any profile features with wavelengths shorter than 100 m.

Three bands filtering to the profile

The short wave band, Psw

The short wave band, Psw is specified as, 0.707 < λ < 2.828 m and is obtained by the following steps:

- First a Psw1 is obtained using an IIR (Infinite Impulse Response) low pass filter limited to λ=0.707 m (Butterworth 8th order) to the profile*.
- Next, to obtain Psw2 an IIR (Infinite Impulse Response) low pass filter limited to λ=2.828 m (Butterworth 8th order) to the profile*.

The short wave band Psw is defined as the difference between Psw1 and Psw2

The medium wave band, Pmw

The medium wave band, Pmw is specified as, 2.828 < λ < 11.312 m and is obtained by the following steps:

- First Pmw1 is obtained by using an IIR (Infinite Impulse Response) low pass filter limited to λ=2.828 m (Butterworth 7th order) to the profile*.
- Next, to obtain Pmw2 an IIR (Infinite Impulse Response) low pass filter limited to λ=11.312 m (Butterworth 7th order) to the profile*.

The medium wave band Pmw is defined as the difference between Pmw1 and Pmw2.

The long wave band, Plw

The long wave band, Plw is specified as the long waves, 11.312 < λ < 45.248 m.

- First Plw1 is obtained by using an IIR (Infinite Impulse Response) low pass filter limited to λ=11.312 m (Butterworth 6th order) to the profile*.
- Next, to obtain Plw2 an IIR (Infinite Impulse Response) low pass filter limited to λ=45.248 m (Butterworth 6th order) to the profile*.

The long wave band Plw is defined as the difference between Plw1 and Plw2.

* a forward and reverse filter procedure should be used to prevent phase problems.

Calculation of indices and reporting length

Finally the energy of filtered profiles should be calculated and presented as defined below:

- each 20 m for Psw
- each 100 m for Pmw
- each 200 m for Plw

Filter parameter specifications

Octave filters (according to ANSI-S1-11-2004-07-27 and ISO 8608)

Octave filters are constant relative band filters defined by

$$\frac{f_h}{f_l} = 2$$ **Equation C.2**

where

f_h is the high cut-off spatial frequency at – 3 dB, in cycles per metre;

f_l is the low cut-off spatial frequency at – 3 dB, in cycles per metre.

The central spatial frequency of such a filter is defined as $f_c = \sqrt{f_l \ f_h}$.

Bi-octave band

The bi-octave bands used for the determination of band limited evenness indices group together two consecutive octaves. The bandwidths given here are defined by ANSI-S1-11-2004-07-27 and ISO 8608. These bands are defined below:

- SW, short wave band grouping together the octaves having a central spatial frequency of 1 cycle per metre and 0.5 cycle per metre;
- MW, medium wave band grouping together the octaves having a central spatial frequency of 0.25 cycle per metre and 0.125 cycle per metre;

- LW, long wave band grouping together the octaves having a central spatial frequency of 0.0625 cycle per metre and 0.0312 cycle per metre.

Table 1 summarises these values showing wavelengths as well as spatial frequencies.

Table 1: Characteristics of the filters

Band	Spatial filtering in wavelengths		Spatial frequencies in cycles per metre
	Bi-octave m	**Octave** m	
Short waves (SW)			
low limit	0.7071	0.7071	1.4142
		1	1
central value	1.4142	1.4142	0.7071
		2	0.5
high limit	2.828	2.828	0.3536
Medium waves (MW)			
low limit	2.828	2.828	0.3536
		4	0.25
central value	5.656	5.656	0.1768
		8	0.125
high value	11.312	11.312	0.0884
Long waves (LW)			
low limit	11.312	11.312	0.0884
		16	0.0625
central value	22.624	22.624	0.0442
		32	0.0312
high values	45.248	45.248	0.0221

Filter coefficients

The coefficients can be directly applied, or being calculated using the code described in section “C++ code for calculation of energy profils filtered by wave band” below. These coefficients assume that the sampling interval is 0.05m. For each filter, the coefficients are given in the following tables (17 decimal digits are valid):

- general high pass filter limited to λ=100 m (Butterworth 4th order)

Numerator	Denominator
0.99590372213842415	1
-3.9836148885536966	-3.9917906244144712
5.9754223328305454	5.9754055533867305
-3.9836148885536966	-3.9754391526444168
0.99590372213842415	0.99182422376916757

- low pass filter limited to λ=0.707 m (Butterworth 8th order)

Numerator	Denominator
$2.14529801577021*10^{-6}$	1
$17.16238412616169*10^{-6}$	-5.7242226208025686
$60.06834444156591*10^{-6}$	14.58145031328104
$120.13668888313182*10^{-6}$	-21.537527317629245
$150.17086110391477*10^{-6}$	20.139632615427132
$120.13668888313182*10^{-6}$	-12.191803302114604
$60.06834444156591*10^{-6}$	4.6608025372476556
$17.16238412616169*10^{-6}$	-1.0278051611001193
$2.14529801577021*10^{-6}$	0.1000221319827452

- low pass filter limited to λ=2.828 m (Butterworth 8th order)

Numerator	Denominator
$68.70180553*10^{-12}$	1
$549.61444423*10^{-12}$	-7.4306085596572169
$1923.65055482*10^{-12}$	24.175130708312153
$3847.30110964*10^{-12}$	-44.978754448661576
$4809.12638705*10^{-12}$	52.3418378405241
$3847.30110964*10^{-12}$	-39.0107816193456
$1923.65055482*10^{-12}$	18.184703074245409
$549.61444423*10^{-12}$	-4.8471638604738629
$68.70180553*10^{-12}$	0.56563688264425616

- low pass filter limited to λ=2.828 m (Butterworth 7th order)

Numerator	Denominator
$1.2797761878*10^{-9}$	1
$8.95843331461*10^{-9}$	-6.5008074238777498

$26.87529994383*10^{-9}$	18.128343066119079
$44.79216657305*10^{-9}$	-28.11014538461913
$44.79216657305*10^{-9}$	26.175397507756664
$26.87529994383*10^{-9}$	-14.63648503523595
$8.95843331461*10^{-9}$	4.5504954916545053
$1.2797761878*10^{-9}$	-0.60679805798606612

- low pass filter limited to λ=11.312 m (Butterworth 7th order)

Numerator	Denominator
$93.55624*10^{-15}$	1
$654.89367*10^{-15}$	-6.8751933778374745
$1964.68102*10^{-15}$	20.258931436785097
$3274.46837*10^{-15}$	-33.166446092752317
$3274.46837*10^{-15}$	32.580346116749226
$1964.68102*10^{-15}$	-19.203775331356002
$654.89367*10^{-15}$	6.2887997685528836
$93.55624*10^{-15}$	-0.88266252012943813

- low pass filter limited to λ=11.312 m (Butterworth 6th order)

Numerator	Denominator
$6.79745149*10^{-12}$	1
$40.78470894*10^{-12}$	-5.8926970878872194
$101.96177236*10^{-12}$	14.469227251620786
$135.94902981*10^{-12}$	-18.949743931136801
$101.96177236*10^{-12}$	13.960843524977586
$40.78470894*10^{-12}$	-5.4858787617673777
$6.79745149*10^{-12}$	0.89824900462806312

- low pass filter limited to λ=45.248 m (Butterworth 6th order)

Numerator	Denominator
$1.71738*10^{-15}$	1
$10.30426*10^{-15}$	-5.9731741134704492
$25.76064*10^{-15}$	14.866230144781298
$34.34752*10^{-15}$	-19.733176391096528
$25.76064*10^{-15}$	14.733889456574856
$10.30426*10^{-15}$	-5.8672997515712346
$1.71738*10^{-15}$	0.97353065478216805

Wave band filter response curves

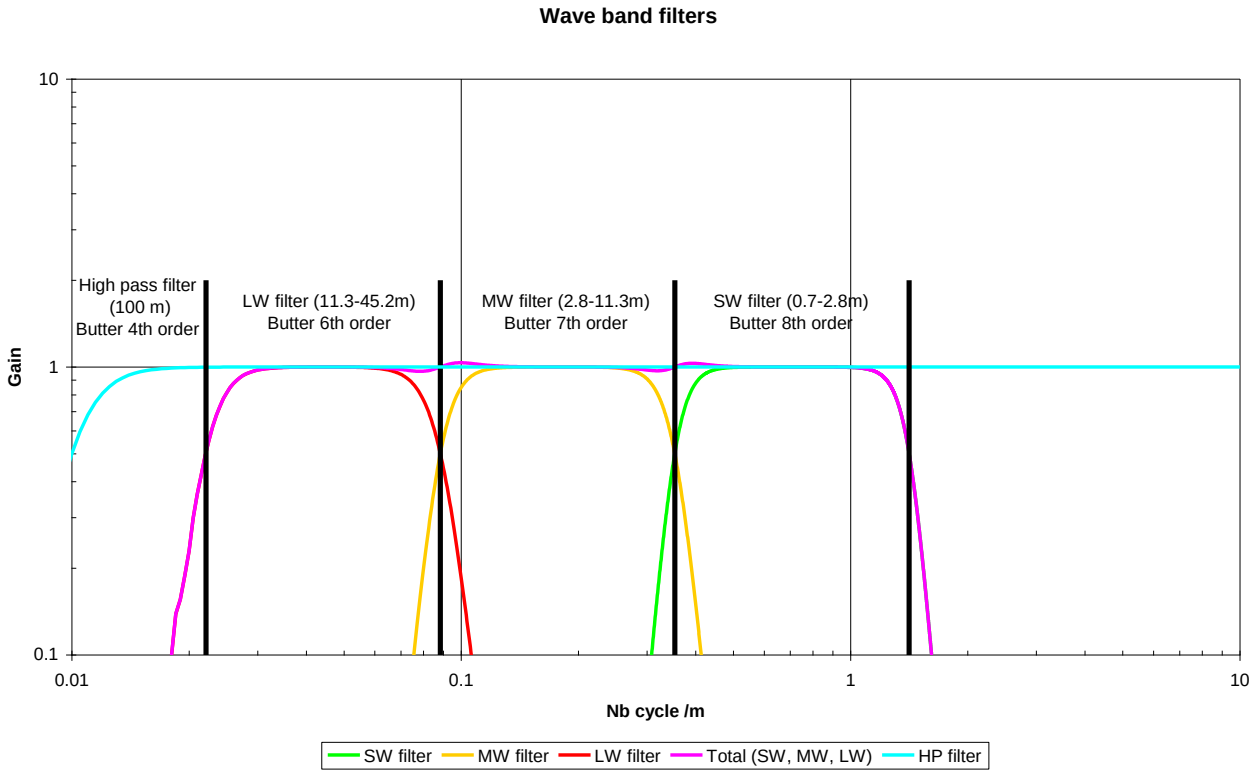

Figure 6: Response curves for the short waveband, medium waveband and long waveband filters

Root Mean Square value of the pre-processed profile elevations per wave band

The Root Mean Square value per wave band of the pre-processed profile elevations over a section S_{ij}, where i and j are respectively the indices of the first and last point of the profile to be considered, is defined as:

$$WB_{ij} = \sqrt{\frac{1}{j-i+1}\sum_{k=i}^{k=j} z_{wb,k}^2}$$ **Equation C.3**

The index per wave band WB_{ij} can determined for each of the three wave band limited profiles z_{wb}, by applying the above formula to the digital files obtained by filtering the pre-processed profile using the bi-octave filters described above. For example, for the short waveband range (0.7071 m to 2.828 m), the index, SWij is defined as:

$$SW_{ij} = \sqrt{\frac{1}{j-i+1}\sum_{k=i}^{k=j} z_{sw,k}^2}$$ **Equation C.4**

For instance $SW_{1024, 2048}$, is the short waveband Root Mean Square of the pre-processed profile elevations for the reporting length containing profile samples 1024 to 2048.

The significance and values of the different WB_{ij} depend on the reporting length, in a non-linear way, it is strongly recommended to use the following values:

- 100 m and at least 20 m for the SW_{ij} values,
- 100 m and at least 50 m for the MW_{ij} values,
- at least 100 m for the LW_{ij} values.

C++ code for calculation of energy profils filtered by wave band

Filtre_Butterworth.h

```
#if !defined(AFX_FILTRE_BUTTERWORTH_H__297458A3_B069_4CA1_9675_B1D2A618399F__INCLUDED_)
#define AFX_FILTRE_BUTTERWORTH_H__297458A3_B069_4CA1_9675_B1D2A618399F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CFiltre_Butterworth
{
public:
	CFiltre_Butterworth();
	virtual ~CFiltre_Butterworth();
	void Coefficients(double *A,double *B);
	int Filtrage(int ordre,double frequence_coupure,int np,double *x,double *y);

protected :
	double *A,*B;

	double *binomial_mult( int n, double *p );
	double *trinomial_mult( int n, double *b, double *c );
	double *dcof_bwlp( int n, double fcf );
	double *dcof_bwhp( int n, double fcf );
	double *dcof_bwbp( int n, double f1f, double f2f );
	double *dcof_bwbs( int n, double f1f, double f2f );
	int *ccof_bwlp( int n );
	int *ccof_bwhp( int n );
	int *ccof_bwbp( int n );
	double *ccof_bwbs( int n, double f1f, double f2f );
	double sf_bwlp( int n, double fcf );
	double sf_bwhp( int n, double fcf );
```

```
        double sf_bwbp( int n, double f1f, double f2f );
        double sf_bwbs( int n, double f1f, double f2f );
        void filter(int ord, double *a, double *b, int np, double *x, double *y);
};

#endif // !defined(AFX_FILTRE_BUTTERWORTH_H__297458A3_B069_4CA1_9675_B1D2A618399F__INCLUDED_)
```

Filtre_Butterworth.cpp

```
#include "stdafx.h"
#include "Filtre_Butterworth.h"
#include <math.h>

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

#define M_PI 3.1415926535898

CFiltre_Butterworth::CFiltre_Butterworth()
{
        A=NULL;
        B=NULL;
}

CFiltre_Butterworth::~CFiltre_Butterworth()
{
        if (A) delete [] A;
        if (B) delete [] B;
}

void CFiltre_Butterworth::Coefficients(double *CA,double *CB)
{
        CA=A;
        CB=B;
}

int CFiltre_Butterworth::Filtrage(int ordre,double frequence_coupure,int np,double *x,double *y)
{
   int i,*ccof;
   double sf,*aux;

        aux = new double[np];
        for (i=0;i<np;i++) aux[i]=x[i];
   A = dcof_bwlp(ordre,frequence_coupure);
   if (A==NULL) return 0;
   ccof = ccof_bwlp(ordre);
   if (ccof==NULL) return 0;
   sf = sf_bwlp(ordre,frequence_coupure);
        B = new double[ordre+1];
        for (i=0;i<=ordre;i++) B[i]=(double)ccof[i]*sf;

        filter(ordre,A,B,np,aux,y);
        /* NOW y=filter(b,a,x);*/
        /* reverse the series for FILTFILT */
        for (i=0;i<np;i++) aux[i]=y[np-i-1];
        /* do FILTER again */
        filter(ordre,A,B,np,aux,y);
        /* reverse the series back */
        for (i=0;i<np;i++) aux[i]=y[np-i-1];
        for (i=0;i<np;i++) y[i]=aux[i];
        delete [] aux;
        delete [] ccof;
        return 1;
}

/*********************************************************************
  binomial_mult - multiplies a series of binomials together and returns
  the coefficients of the resulting polynomial.
```

```
                The multiplication has the following form:

                (x+p[0])*(x+p[1])*...*(x+p[n-1])

                The p[i] coefficients are assumed to be complex and are passed to the
                function as a pointer to an array of doubles of length 2n.

                The resulting polynomial has the following form:

                x^n + a[0]*x^n-1 + a[1]*x^n-2 + ... +a[n-2]*x + a[n-1]

  The a[i] coefficients can in general be complex but should in most
  cases turn out to be real. The a[i] coefficients are returned by the
  function as a pointer to an array of doubles of length 2n. Storage
  for the array is allocated by the function and should be freed by the
  calling program when no longer needed.

  Function arguments:

  n  -  The number of binomials to multiply
  p  -  Pointer to an array of doubles where p[2i] (i=0...n-1) is
        assumed to be the real part of the coefficient of the ith binomial
        and p[2i+1] is assumed to be the imaginary part. The overall size
        of the array is then 2n.
*/

double *CFiltre_Butterworth::binomial_mult( int n, double *p )
{
    int i, j;
    double *a;

    a = (double *)calloc( 2 * n, sizeof(double) );
    if( a == NULL ) return( NULL );

    for( i = 0; i < n; ++i )
    {
        for( j = i; j > 0; --j )
        {
            a[2*j] += p[2*i] * a[2*(j-1)] - p[2*i+1] * a[2*(j-1)+1];
            a[2*j+1] += p[2*i] * a[2*(j-1)+1] + p[2*i+1] * a[2*(j-1)];
        }
        a[0] += p[2*i];
        a[1] += p[2*i+1];
    }
    return( a );
}


/*****************************************************************************
  trinomial_mult - multiplies a series of trinomials together and returns
  the coefficients of the resulting polynomial.

  The multiplication has the following form:

  (x^2 + b[0]x + c[0])*(x^2 + b[1]x + c[1])*...*(x^2 + b[n-1]x + c[n-1])

  The b[i] and c[i] coefficients are assumed to be complex and are passed
  to the function as a pointers to arrays of doubles of length 2n. The real
  part of the coefficients are stored in the even numbered elements of the
  array and the imaginary parts are stored in the odd numbered elements.

  The resulting polynomial has the following form:

  x^2n + a[0]*x^2n-1 + a[1]*x^2n-2 + ... +a[2n-2]*x + a[2n-1]

  The a[i] coefficients can in general be complex but should in most cases
  turn out to be real. The a[i] coefficients are returned by the function as
  a pointer to an array of doubles of length 4n. The real and imaginary
  parts are stored, respectively, in the even and odd elements of the array.
  Storage for the array is allocated by the function and should be freed by
  the calling program when no longer needed.

  Function arguments:
```

```
  n  -  The number of trinomials to multiply
  b  -  Pointer to an array of doubles of length 2n.
  c  -  Pointer to an array of doubles of length 2n.
*/

double *CFiltre_Butterworth::trinomial_mult( int n, double *b, double *c )
{
   int i, j;
   double *a;

        a = (double *)calloc( 4 * n, sizeof(double) );
        if( a == NULL ) return( NULL );

        a[2] = c[0];
        a[3] = c[1];
        a[0] = b[0];
        a[1] = b[1];

        for( i = 1; i < n; ++i )
        {
            a[2*(2*i+1)]   += c[2*i]*a[2*(2*i-1)]   - c[2*i+1]*a[2*(2*i-1)+1];
            a[2*(2*i+1)+1] += c[2*i]*a[2*(2*i-1)+1] + c[2*i+1]*a[2*(2*i-1)];

            for( j = 2*i; j > 1; --j )
            {
               a[2*j]   += b[2*i] * a[2*(j-1)]   - b[2*i+1] * a[2*(j-1)+1] +
                    c[2*i] * a[2*(j-2)]   - c[2*i+1] * a[2*(j-2)+1];
               a[2*j+1] += b[2*i] * a[2*(j-1)+1] + b[2*i+1] * a[2*(j-1)] +
                    c[2*i] * a[2*(j-2)+1] + c[2*i+1] * a[2*(j-2)];
            }

            a[2] += b[2*i] * a[0] - b[2*i+1] * a[1] + c[2*i];
            a[3] += b[2*i] * a[1] + b[2*i+1] * a[0] + c[2*i+1];
            a[0] += b[2*i];
            a[1] += b[2*i+1];
        }

        return( a );
    }


    /*************************************************************************
      dcof_bwlp - calculates the d coefficients for a butterworth lowpass
      filter. The coefficients are returned as an array of doubles.

    */

    double *CFiltre_Butterworth::dcof_bwlp( int n, double fcf )
    {
       int k;            // loop variables
       double theta;     // M_PI * fcf / 2.0
       double st;        // sine of theta
       double ct;        // cosine of theta
       double parg;      // pole angle
       double sparg;     // sine of the pole angle
       double cparg;     // cosine of the pole angle
       double a;         // workspace variable
       double *rcof;     // binomial coefficients
       double *dcof;     // dk coefficients

       rcof = (double *)calloc( 2 * n, sizeof(double) );
       if( rcof == NULL ) return( NULL );

       theta = M_PI * fcf;
       st = sin(theta);
       ct = cos(theta);

       for( k = 0; k < n; ++k )
       {
            parg = M_PI * (double)(2*k+1)/(double)(2*n);
            sparg = sin(parg);
            cparg = cos(parg);
            a = 1.0 + st*sparg;
            rcof[2*k] = -ct/a;
```

```
                rcof[2*k+1] = -st*cparg/a;
            }

            dcof = binomial_mult( n, rcof );
            free( rcof );

            dcof[1] = dcof[0];
            dcof[0] = 1.0;
            for( k = 3; k <= n; ++k )
                dcof[k] = dcof[2*k-2];
    return( dcof );
}

/*************************************************************************
  dcof_bwhp - calculates the d coefficients for a butterworth highpass
  filter. The coefficients are returned as an array of doubles.

*/

double *CFiltre_Butterworth::dcof_bwhp( int n, double fcf )
{
    return( dcof_bwlp( n, fcf ) );
}


/*************************************************************************
  dcof_bwbp - calculates the d coefficients for a butterworth bandpass
  filter. The coefficients are returned as an array of doubles.

*/

double *CFiltre_Butterworth::dcof_bwbp( int n, double f1f, double f2f )
{
    int k;            // loop variables
    double theta;     // M_PI * (f2f - f1f) / 2.0
    double cp;        // cosine of phi
    double st;        // sine of theta
    double ct;        // cosine of theta
    double s2t;       // sine of 2*theta
    double c2t;       // cosine 0f 2*theta
    double *rcof;     // z^-2 coefficients
    double *tcof;     // z^-1 coefficients
    double *dcof;     // dk coefficients
    double parg;      // pole angle
    double sparg;     // sine of pole angle
    double cparg;     // cosine of pole angle
    double a;         // workspace variables

    cp = cos(M_PI * (f2f + f1f) / 2.0);
    theta = M_PI * (f2f - f1f) / 2.0;
    st = sin(theta);
    ct = cos(theta);
    s2t = 2.0*st*ct;        // sine of 2*theta
    c2t = 2.0*ct*ct - 1.0;  // cosine of 2*theta

    rcof = (double *)calloc( 2 * n, sizeof(double) );
    tcof = (double *)calloc( 2 * n, sizeof(double) );

    for( k = 0; k < n; ++k )
    {
        parg = M_PI * (double)(2*k+1)/(double)(2*n);
        sparg = sin(parg);
        cparg = cos(parg);
        a = 1.0 + s2t*sparg;
        rcof[2*k] = c2t/a;
        rcof[2*k+1] = s2t*cparg/a;
        tcof[2*k] = -2.0*cp*(ct+st*sparg)/a;
        tcof[2*k+1] = -2.0*cp*st*cparg/a;
    }

    dcof = trinomial_mult( n, tcof, rcof );
    free( tcof );
    free( rcof );
```

```
    dcof[1] = dcof[0];
    dcof[0] = 1.0;
    for( k = 3; k <= 2*n; ++k )
        dcof[k] = dcof[2*k-2];
    return( dcof );
}

/*************************************************************************
  dcof_bwbs - calculates the d coefficients for a butterworth bandstop
  filter. The coefficients are returned as an array of doubles.

                */

                double *CFiltre_Butterworth::dcof_bwbs( int n, double f1f, double f2f )
                {
                    int k;            // loop variables
                    double theta;     // M_PI * (f2f - f1f) / 2.0
                    double cp;        // cosine of phi
                    double st;        // sine of theta
                    double ct;        // cosine of theta
                    double s2t;       // sine of 2*theta
                    double c2t;       // cosine 0f 2*theta
                    double *rcof;     // z^-2 coefficients
                    double *tcof;     // z^-1 coefficients
                    double *dcof;     // dk coefficients
                    double parg;      // pole angle
                    double sparg;     // sine of pole angle
                    double cparg;     // cosine of pole angle
                    double a;         // workspace variables

                    cp = cos(M_PI * (f2f + f1f) / 2.0);
                    theta = M_PI * (f2f - f1f) / 2.0;
                    st = sin(theta);
                    ct = cos(theta);
                    s2t = 2.0*st*ct;        // sine of 2*theta
                    c2t = 2.0*ct*ct - 1.0;  // cosine 0f 2*theta

                    rcof = (double *)calloc( 2 * n, sizeof(double) );
                    tcof = (double *)calloc( 2 * n, sizeof(double) );

                    for( k = 0; k < n; ++k )
                    {
                        parg = M_PI * (double)(2*k+1)/(double)(2*n);
                        sparg = sin(parg);
                        cparg = cos(parg);
                        a = 1.0 + s2t*sparg;
                        rcof[2*k] = c2t/a;
                        rcof[2*k+1] = -s2t*cparg/a;
                        tcof[2*k] = -2.0*cp*(ct+st*sparg)/a;
                        tcof[2*k+1] = 2.0*cp*st*cparg/a;
                    }

                    dcof = trinomial_mult( n, tcof, rcof );
                    free( tcof );
                    free( rcof );

                    dcof[1] = dcof[0];
                    dcof[0] = 1.0;
                    for( k = 3; k <= 2*n; ++k )
                        dcof[k] = dcof[2*k-2];
                    return( dcof );
                }

                /*************************************************************************
                  ccof_bwlp - calculates the c coefficients for a butterworth lowpass
                  filter. The coefficients are returned as an array of integers.

                */

                int *CFiltre_Butterworth::ccof_bwlp( int n )
                {
                    int *ccof;
                    int m;
                    int i;
```

```
                ccof = (int *)calloc( n+1, sizeof(int) );
                if( ccof == NULL ) return( NULL );

                ccof[0] = 1;
                ccof[1] = n;
                m = n/2;
                for( i=2; i <= m; ++i)
                {
                   ccof[i] = (n-i+1)*ccof[i-1]/i;
      ccof[n-i]= ccof[i];
   }
   ccof[n-1] = n;
   ccof[n] = 1;

   return( ccof );
}

/*************************************************************************
  ccof_bwhp - calculates the c coefficients for a butterworth highpass
  filter. The coefficients are returned as an array of integers.

*/

int *CFiltre_Butterworth::ccof_bwhp( int n )
{
   int *ccof;
   int i;

   ccof = ccof_bwlp( n );
   if( ccof == NULL ) return( NULL );

   for( i = 0; i <= n; ++i)
      if( i % 2 ) ccof[i] = -ccof[i];

   return( ccof );
}

/*************************************************************************
  ccof_bwbp - calculates the c coefficients for a butterworth bandpass
  filter. The coefficients are returned as an array of integers.

*/

int *CFiltre_Butterworth::ccof_bwbp( int n )
{
   int *tcof;
   int *ccof;
   int i;

   ccof = (int *)calloc( 2*n+1, sizeof(int) );
   if( ccof == NULL ) return( NULL );

   tcof = ccof_bwhp(n);
   if( tcof == NULL ) return( NULL );

   for( i = 0; i < n; ++i)
   {
      ccof[2*i] = tcof[i];
      ccof[2*i+1] = 0;
   }
   ccof[2*n] = tcof[n];

   free( tcof );
   return( ccof );
}

/*************************************************************************
  ccof_bwbs - calculates the c coefficients for a butterworth bandstop
  filter. The coefficients are returned as an array of integers.

*/

double *CFiltre_Butterworth::ccof_bwbs( int n, double f1f, double f2f )
```

```
{
   double alpha;
   double *ccof;
   int i, j;

   alpha = -2.0 * cos(M_PI * (f2f + f1f) / 2.0) / cos(M_PI * (f2f - f1f) / 2.0);

   ccof = (double *)calloc( 2*n+1, sizeof(double) );

   ccof[0] = 1.0;

                  ccof[2] = 1.0;
                  ccof[1] = alpha;

                  for( i = 1; i < n; ++i )
                  {
                      ccof[2*i+2] += ccof[2*i];
                      for( j = 2*i; j > 1; --j )
                          ccof[j+1] += alpha * ccof[j] + ccof[j-1];

                      ccof[2] += alpha * ccof[1] + 1.0;
                      ccof[1] += alpha;
                  }

                  return( ccof );
              }

              /*************************************************************************
                sf_bwlp - calculates the scaling factor for a butterworth lowpass filter.
                The scaling factor is what the c coefficients must be multiplied by so
                that the filter response has a maximum value of 1.

              */

              double CFiltre_Butterworth::sf_bwlp( int n, double fcf )
              {
                  int m, k;        // loop variables
                  double omega;    // M_PI * fcf
                  double fomega;   // function of omega
                  double parg0;    // zeroth pole angle
                  double sf;       // scaling factor

                  omega = M_PI * fcf;
                  fomega = sin(omega);
                  parg0 = M_PI / (double)(2*n);

                  m = n / 2;
                  sf = 1.0;
                  for( k = 0; k < n/2; ++k )
                      sf *= 1.0 + fomega * sin((double)(2*k+1)*parg0);

                  fomega = sin(omega / 2.0);

                  if( n % 2 ) sf *= fomega + cos(omega / 2.0);
                  sf = pow( fomega, n ) / sf;

                  return(sf);
              }

              /*************************************************************************
                sf_bwhp - calculates the scaling factor for a butterworth highpass filter.
                The scaling factor is what the c coefficients must be multiplied by so
                that the filter response has a maximum value of 1.

              */

              double CFiltre_Butterworth::sf_bwhp( int n, double fcf )
              {
                  int m, k;        // loop variables
                  double omega;    // M_PI * fcf
                  double fomega;   // function of omega
                  double parg0;    // zeroth pole angle
                  double sf;       // scaling factor
```

```
                omega = M_PI * fcf;
                fomega = sin(omega);
                parg0 = M_PI / (double)(2*n);

                m = n / 2;
                sf = 1.0;
                for( k = 0; k < n/2; ++k )
                   sf *= 1.0 + fomega * sin((double)(2*k+1)*parg0);

                fomega = cos(omega / 2.0);

   if( n % 2 ) sf *= fomega + sin(omega / 2.0);
   sf = pow( fomega, n ) / sf;

   return(sf);
}

/*****************************************************************************
 sf_bwbp - calculates the scaling factor for a butterworth bandpass filter.
 The scaling factor is what the c coefficients must be multiplied by so
 that the filter response has a maximum value of 1.

*/

double CFiltre_Butterworth::sf_bwbp( int n, double f1f, double f2f )
{
   int k;          // loop variables
   double ctt;     // cotangent of theta
   double sfr, sfi;  // real and imaginary parts of the scaling factor
   double parg;    // pole angle
   double sparg;   // sine of pole angle
   double cparg;   // cosine of pole angle
   double a, b, c;  // workspace variables

   ctt = 1.0 / tan(M_PI * (f2f - f1f) / 2.0);
   sfr = 1.0;
   sfi = 0.0;

   for( k = 0; k < n; ++k )
   {
      parg = M_PI * (double)(2*k+1)/(double)(2*n);
      sparg = ctt + sin(parg);
      cparg = cos(parg);
      a = (sfr + sfi)*(sparg - cparg);
      b = sfr * sparg;
      c = -sfi * cparg;
      sfr = b - c;
      sfi = a - b - c;
   }

   return( 1.0 / sfr );
}

//  sf_bwbs - calculates the scaling factor for a butterworth bandstop filter.
//  The scaling factor is what the c coefficients must be multiplied by so
//  that the filter response has a maximum value of 1.

double CFiltre_Butterworth::sf_bwbs( int n, double f1f, double f2f )
{
   int k;          // loop variables
   double tt;      // tangent of theta
   double sfr, sfi;  // real and imaginary parts of the scaling factor
   double parg;    // pole angle
   double sparg;   // sine of pole angle
   double cparg;   // cosine of pole angle
   double a, b, c;  // workspace variables

   tt = tan(M_PI * (f2f - f1f) / 2.0);
   sfr = 1.0;
   sfi = 0.0;

   for( k = 0; k < n; ++k )
   {
      parg = M_PI * (double)(2*k+1)/(double)(2*n);
```

```
        sparg = tt + sin(parg);
        cparg = cos(parg);
        a = (sfr + sfi)*(sparg - cparg);
        b = sfr * sparg;
        c = -sfi * cparg;
        sfr = b - c;
        sfi = a - b - c;
    }

    return( 1.0 / sfr );
            }


            void CFiltre_Butterworth::filter(int ord, double *a, double *b, int np, double *x, double *y)
            {
                int i,j;
                y[0]=b[0]*x[0];
                for (i=1;i<ord+1;i++)
                {
               y[i]=0.0;
               for (j=0;j<i+1;j++) y[i]=y[i]+b[j]*x[i-j];
               for (j=0;j<i;j++) y[i]=y[i]-a[j+1]*y[i-j-1];
                }

                /* end of initial part */

                for (i=ord+1;i<np+1;i++)
                {
                        y[i]=0.0;
               for (j=0;j<ord+1;j++) y[i]=y[i]+b[j]*x[i-j];
               for (j=0;j<ord;j++) y[i]=y[i]-a[j+1]*y[i-j-1];
```

Calcul.cpp

```
            void Calcul_BO(double Fc1,double Fc2,int ordre,double dx,double *Profil,int N,double *y1,double *y2,double pas,
                                double *Energie,int &NP)
            {
                        int i,j,n;
                double mean;
                CFiltre_Butterworth Butterworth;

                double Fsx=1.0/dx;
                Butterworth.Filtrage(ordre,Fc1/(Fsx/2.0),N,Profil,y1);
                mean=Calcul_XMOY(y1,0,N-1);
                for (i=0;i<N;i++) y1[i]-=mean;
                Butterworth.Filtrage(ordre,Fc2/(Fsx/2.0),N,Profil,y2);
                mean=Calcul_XMOY(y2,0,N-1);
                for (i=0;i<N;i++) y2[i]-=mean;
                for (i=0;i<N;i++) y1[i]=1000.0*(y1[i]-y2[i]);

                NP=0;
                int di=(int)(pas/dx+0.0001);
                // recalcul de n pour avoir un multiple de di
                n=N/di*di;
                for (i=0;i<n;i+=di)
                {
                        Energie[NP]=0.0;
                        int n=i+di;
                        if (n>N) n=N;
                        for (j=i;j<n;j++)
                        {
                                Energie[NP]+=y1[j]*y1[j];
                        }
                        Energie[NP]=sqrt(Energie[NP]/(n-i+1));
                        NP++;
                }
            }
```

```
	void Calcul_EBO_(double dx,double *Profil,int N,double *EPO,int &NBPO,double *EMO,int &NBMO,double *EGO,int &NBGO)
	{
		double *y1,*y2;
		int i;

		y1=new double[N+1];
		y2=new double[N+1];

		for (i=0;i<N;i++) Profil[i]/=1000.0;
	Calcul_BO(1.0/0.707,1.0/2.828,8,dx,Profil,N,y1,y2,20.0,EPO,NBPO);
	Calcul_BO(1.0/2.828,1.0/11.312,7,dx,Profil,N,y1,y2,100.0,EMO,NBMO);
	Calcul_BO(1.0/11.312,1.0/45.248,6,dx,Profil,N,y1,y2,200.0,EGO,NBGO);

	delete [] y1;
	delete [] y2;
}
```

Main Program

```
int N; // point number
double dx; sampling step;
double *EPO,*EMO,*EGO;
	int NBPO,NBMO,NBGO;

	// Calcul des NBPO,NBMO,NBGO et allocation
	NBPO=(int)(N*dx/20.0+1.0001);
	NBMO=(int)(N*dx/100.0+1.0001);
	NBGO=(int)(N*dx/200.0+1.0001);
	VPO = new double[NBPO+1];
	VMO = new double[NBMO+1];
	VGO = new double[NBGO+1];
	NPO = new double[NBPO+1];
	EPO = new double[NBPO+1];
	NMO = new double[NBMO+1];
	EMO = new double[NBMO+1];
	NGO = new double[NBGO+1];
	EGO = new double[NBGO+1];

	Calcul_EBO(dx,Profil,N,EPO,NBPO,EMO,NBMO,EGO,NBGO);
```

Annex D
(informative)

Wave band analysis using LPV over selected wavelengths

Overview of Longitudinal Profile Variance (LPV) calculation

Three values of Enhanced Longitudinal Profile Variance shall be reported for each reporting length. The calculation of each value of Enhanced Longitudinal Profile Variance can be summarised as follows:

- The raw longitudinal profile is filtered using a high pass filter that attenuates frequencies below a predetermined cut-off.
- The Enhanced Longitudinal Profile Variance is calculated from the filtered profile over the reporting length.

Pre-processing

The longitudinal separation of measured points shall not exceed 25mm and the recorded profile points shall be averaged over 100mm.

A filter should be applied to the measured Longitudinal Profile to attenuate wavelengths in excess of 100 m. The filter shall be such that the amplitude of wavelengths greater than 150 m are attenuated by at least 50%. The filter should not distort the phase of any profile features with wavelengths shorter than 100 m.

Regardless of the data collection method employed, Enhanced Longitudinal Profile Variance requires a run-in of 200 m of longitudinal profile measurements so that the Enhanced Longitudinal Profile Variances can be calculated for the entire Survey length of interest. This run in is necessary to fully satisfy the requirements of the filters applied in the calculation of Enhanced Longitudinal Profile Variance. A similar length run-out is also required at the end of the survey.

Filtering

Filtering of the data is achieved by firstly calculating the “filter coefficients”. These are then applied to the profile to generate a new data set that represents the filtered profile.

Calculation of filter coefficients

The filter is defined by a set of m plus one coefficients, which must be calculated before the filter can be applied. The values of the coefficients are determined by the values used to define the filter and are not dependent on the profile data to be processed. The filter coefficients for the pass-band filter are calculated in four stages, as described in the following sections.

Calculation of the width of the filter

The width of the filter, m, is determined by:

$$m = R / (f_L * \Delta) \qquad \textbf{Equation D.1}$$

Where:

Δ is the interval between profile points

R is the “Filter Order”, which has a default value of 3.

The calculated value of m shall be rounded up to the next, even integer.

$1/f_L$ is expressed in the same units as Δ and is defined in Table 1 for the 3 parameters.

Table 1: Specification for enhanced profile variance filters

	3m LPV	10m LPV	30m LPV
Frequency	f_L= 0.3333 m^{-1}	f_L=0.1000 m^{-1}	f_L=0.0333 m^{-1}
Filter Type	High Pass	High Pass	High Pass

Calculation of high-pass coefficients

There are m+1 high-pass coefficients, bhp_i. The values of bhp_i are initially determined by:

$$bhp_i = H_i * \mathrm{sinc}(2 * \pi * i * f_L * \Delta) \qquad \text{for } i = -m/2, -m/2+1, \ldots, m/2$$ **Equation D.2**

Where:

H_i are the coefficients of a Hamming window, given by:

$H_i = 0.54 – 0.46 * \cos(2 * pi * (i + m/2) / m)$

$\mathrm{sinc}(x) = \sin(x) / x$ if $x \neq 0$ OR $\mathrm{sinc}(x) = 1$ if $x=0$

$\pi = 3.14159$

f_L is the lower frequency limit of the bi-octave filter, expressed in the same units as Δ

and the trigonometric functions are defined such that the arguments are in radians

The coefficients, bhp_I, are then normalised as:

$$bhp_i = blp_i / n \qquad \text{for I} = -m/2, -m/2+1, \ldots, m/2$$

Where: $n = \sum_{i=-m/2}^{m/2} bhp_i$ **Equation D.3**

Application of the coefficients to filter the profile data

The value of the filtered profile height, z_i', at each position, i, is obtained by multiplying the profile heights between $z_{(i-m/2)}$ and $z_{(i+m/2)}$ by the corresponding m+1 filter coefficients and the summing the resulting products. Hence:

$$z_i' = \sum_{j=-\frac{m}{2}}^{\frac{m}{2}} z_{i+j} * bhp_j$$ **Equation D.4**

It can be inferred from the above definition that the calculation of the filtered profile cannot be performed within a distance of m/2 points of the start of the measured profile or m/2 points of the end of the measured profile.

If the filtering method was applied separately over each 10m length there will be an inevitable loss of information. For the range of frequencies to be filtered using texture profile the total length “lost” at the start and end of each 10m length will be less than 1m, which may not have a severe effect on the measurement of noise or fretting (although it would be desirable if this could be avoided by evaluating the filter over the whole of the Survey length). However, for the calculation of enhanced variance the effect will be severe and therefore the filter shall be applied over the whole of the Survey such that the points are only lost at the beginning and end of the Survey.

Enhanced variance

The enhanced variance is calculated as:

$$Enhanced\_Variance = \frac{1}{N}\sum_{j=1}^{N}\left(z_j\right)^2$$ **Equation D.5**

Where:

N is the number of filtered profile points within each reporting length (10m default)

z_j is the height of filtered profile point j in mm

Annex E
(informative)

Calculation of the WLP

The WLP calculation

Pre-processing

The WLP calculation is based on the pre-processed profile z with a length of 204.8 m, of which only the part in the centre is the WLP reference length (e.g. 50 or 100 m). The profiles are resampled at a fixed spatial sampling interval (standard: δx = 0.1 m (see chapter 8.2)).

Fourier transformation

The profile z(i) is Fourier transformed. The complex digital Fourier transform of z(i) is:

$$\underline{z}(k) = \sum_{i=1}^{2048} z(i) \cdot e^{-j\frac{2\pi\cdot(i-1)\cdot(k-1)}{2048}} \qquad \text{for } 0 < k \le 2048. \qquad \textbf{Equation E.1}$$

Limiting the wavelength range

For the assessment of unevenness it is necessary to limit the signals to the desired wavelength range. The following equations apply:

$$L(k) = \frac{1}{\sqrt{1+\left[(k-1)\frac{\Lambda_L}{\Lambda s}\right]^{2n}}} \quad ; \quad H(k) = 1 - \frac{1}{\sqrt{1+\left[(k-1)\frac{\Lambda_U}{\Lambda s}\right]^{2n}}} \qquad \text{for } 0 < k \le 1025, \qquad \textbf{Equation E.2a}$$

$$L(k) = \frac{1}{\sqrt{1+\left[(2049-k)\frac{\Lambda_L}{\Lambda s}\right]^{2n}}} \; ; \quad H(k) = 1 - \frac{1}{\sqrt{1+\left[(2049-k)\frac{\Lambda_U}{\Lambda s}\right]^{2n}}} \qquad \text{for } 1025 < k \le 2048 \qquad \textbf{Equation E.2b}$$

n is the order of the filter and set to 4. Λ_s is the maximum wavelength of the Fourier Transform (204.8 m), Λ_L the lower, and Λ_U the upper cut-off wavelength. For road profiles 0.5 m and 50 m apply respectively.

The filtered **complex Fourier Transform is:** $\underline{Z}(k) = L(k) \cdot H(k) \cdot \underline{z}(k)$ **.** **Equation E.3**

Weighting of the fourier transform

In a third step the filtered Fourier Transform is multiplied by a weighting function (*W*). *W(k)* is defined as:

$$W(k) = \sqrt{\frac{\left[(k-1)\frac{\Lambda_U}{\Lambda_S}\right]^{w-1}}{2^{\frac{w-1}{2}} - 2^{-\frac{w-1}{2}}}} \qquad \text{for } 0 < k \le 1025 \qquad \textbf{Equation E.4a}$$

$$W(k) = \sqrt{\frac{\left[(2049-k)\frac{\Lambda_U}{\Lambda_S}\right]^{w-1}}{2^{\frac{w-1}{2}} - 2^{-\frac{w-1}{2}}}} \qquad \text{for } 1025 < k \le 2048 \qquad \textbf{Equation E.4b}$$

w is the waviness of the reference spectrum set to 2.6.

The weighted filtered **complex Fourier Transform is:** $\underline{Z_W}(k) = W(k) \cdot \underline{Z}(k)$**.** **Equation E.5**

Octave-band filtering

The **weighted filtered** complex Fourier spectrum $\underline{Z_W}(k)$ is divided into 10 successive individual octave bands $\underline{Z_{WO,m}}(k)$ with 0<m≤10. The following Butterworth filters are applied to calculate the output of octave band m:

$$L_m(k) = \frac{1}{\sqrt{1+\left[(k-1)\cdot 2^{-m}\right]^{2n}}} \quad ; \quad H_m(k) = 1 - \frac{1}{\sqrt{1+\left[(k-1)\cdot 2^{-(m-1.5)}\right]^{2n}}} \qquad \text{for } 0 < k \le 1025,$$

Equation E.6a

$$L_m(k) = \frac{1}{\sqrt{1+\left[(2049-k)\cdot 2^{-m}\right]^{2n}}} \; ; \quad H_m(k) = 1 - \frac{1}{\sqrt{1+\left[(2049-k)\cdot 2^{-(m-1.5)}\right]^{2n}}} \qquad \text{for } 1025 < k \le 2048,$$

Equation E.6b

n is the order of the filter and set to 4. The output of octave band m is calculated as

$$\underline{Z_{WO,m}}(k) = L_m(k) \cdot H_m(k) \cdot \underline{Z_W}(k) \qquad \textbf{Equation E.7}$$

Inverse fourier transform

Each of the 10 individual octave-band filtered Fourier transforms is transformed back into space domain by the inverse Fourier transform:

$$z_{WO,m}(i) = \frac{1}{2048} \sum_{k=1}^{2048} \underline{Z_{WO,m}}(k) \cdot e^{j\frac{2\pi\cdot(i-1)\cdot(k-1)}{2048}} \qquad \textbf{Equation E.8}$$

The result of the individual inverse FT is 10 **octave-band filtered, weighted** profiles.

Calculating the WLP

Before calculating the WLP the 10 octave-band filtered weighted profiles are shortened from 2048 samples to the WLP reference length by symmetrically cutting off the data in front and back. For a reference length of 100 m (1024 samples) 524 samples in the front and 524 samples in the back have to be cut off. The Weighted Longitudinal Profile (WLP) then is calculated by following formula:

$$WLP(i) = \sum_{m=1}^{10} \frac{\sigma_m}{\sigma_{tot}} \cdot z_{WO,m}(i) \qquad \textbf{Equation E.9}$$

with

$z_{WO,\ m}$:= m[th] octave-band filtered profile according to equation (E.8), with $0 < m \leq 10$

σ_m := standard deviation of $z_{WO,\ m}$

σ_{tot} := standard deviation of z_W, the sum of the 10 octave-filtered profiles, $z_W(i) = \sum_{m=1}^{10} z_{WO,\ m}(i)$.

The standard deviations are calculated as follows:

$$\sigma_m = \sqrt{\frac{n\sum_{i=1}^{n}(z_{WO,\ m}\ (i))^2 - (\sum_{i=1}^{n} z_{WO,\ m}(i))^2}{n^2}}$$

Equation E.10a

$$\sigma_{tot} = \sqrt{\frac{n\sum_{i=1}^{n}(z_W\ (i))^2 - (\sum_{i=1}^{n} z_W\ (i))^2}{n^2}}$$

Equation E.10b

with *n* := number of samples according to the reference length. The reference length used for the WLP should be indicatedas the sub index, e.g. WLP_{100} if the reference length is 100 m.

Characterizing the WLP

The Weighted Longitudinal Profile is characterized by the range and the standard deviation based on the reporting (evaluation) length. The recommended reporting length is the reference length. The range and standard deviation is calculated as follows:

- range $\Delta_{WLP} = \max(WLP) - \min(WLP)$

- and standard deviation $\sigma_{WLP} = \sqrt{\frac{n\sum_{i=1}^{n}(WLP\ (i))^2 - (\sum_{i=1}^{n} WLP\ (i))^2}{n^2}}$

Equation E.11

With *n*:=number of samples within the reporting (evaluation) length.

Example code for WLP calculation

This Matlab® code will compute the WLP for the standard configuration (reference length=evaluation/reporting length).

```
function [WLP, delta_WLP, sigma_WLP] = wlp(x,np_eval,waviness,Lmin,Lmax)
   % Input
   %    x: profile containing np = 2048 samples
   %    np_eval: number of samples according to the reference length (e.g. 1000)
   %    waviness: slope of the WLP reference Power Spectral Density
   %    Lmin, Lmax: lower and upper cutoff wavelength for WLP calculation (i.e. 0.5 m; 50 m)
   % Output
   %    WLP:  WLP contains np_eval data centered at middle of x
   %    delta_WLP: range of WLP based on evaluation length (here: same as reference length)
   %    sigma_WLP: standard deviation of WLP based on evaluation length (here: same as ref. length)

   %%% Set parameters
   np = 2048;                  % number of samples
   sample_int = 0.1;             % sampling interval [m]
   nbands = 10;                % number of octave bands
   lambda_s = np*sample_int;      % maximal wavelength FFT  [m]
   lambda_ratio_L = Lmin/lambda_s;
   lambda_ratio_U = Lmax/lambda_s;

   %%% Evaluate low-pass filter
   l = bw_lp(np, lambda_ratio_L, 4);

   %%% evaluate high-pass filter
   h = bw_hp(np, lambda_ratio_U, 4);

   %%% evaluate weighting function
   w = weight_ft(np, lambda_ratio_U, waviness);

%--------------------------------------------------------------------------

   %%% Fourier transformation
   z = fft(x(:));

   %%% Limiting the wavelength range
   Z = z.*l.*h;

   %%% Weighting of Fourier transform
   Z_w = w.*Z;

   %%% Octave-band filtering
   for i = 1:nbands
      Li = bw_lp(np, 2^-i, 4);
      Hi = bw_hp(np, 2^-(i-1.5), 4);
      Z_wo(:, i) = Li.*Hi.*Z_w;
   end

   %%% Inverse Fourier transform
   z_wo = real(ifft(Z_wo));

   %%% Weighted Longitudinal Profile calculation
   WLP = wlp_comp(z_wo, np_eval);

   %%% Range and Standard Deviation of WLP (here: based on reference length)
   delta_WLP = max(WLP)-min(WLP);
   sigma_WLP = std(WLP,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% local functions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Butterworth high-pass filter

function H = bw_hp(np, lambda_ratio, n)
% Input
%    np: number of samples for FFT (here: np=2048)
%        must be a power of 2
%    lambda_ratio: ratio of cut-off wavelength and maximum wavelength in
%                  Fourier spectrum
```

```
%    n: order of Butterworth high-pass filter (e.g. n=4)
% Output
%    H: evaluated Butterworth high-pass filter (double, np x 1)

   % compute first half of highpass (array index 1:(np/2+1))
   index = 0:(np/2);
   H = 1 - 1./sqrt(1+(index*lambda_ratio).^(2*n))';
   % mirror function at array index (np/2+1)
   H = [H(1:(np/2+1)); H(np/2:-1:2)];

%%% Butterworth low-pass filter

function L = bw_lp(np, lambda_ratio, n)
% Input
%    np: number of samples for FFT (here: np=2048)
%        must be a power of 2
%    lambda_ratio: ratio of cut-off wavelength and maximum wavelength in
%                  Fourier spectrum
%    n: order of Butterworth low-pass filter (e.g. n=4)
% Output
%    H: evaluated Butterworth low-pass filter (double, np x 1)

   % compute first half of lowpass (array index 1:(np/2+1))
   index = 0:(np/2);
   L = 1./sqrt(1+(index*lambda_ratio).^(2*n))';

   % mirror function at array index (np/2+1)
   L = [L(1:(np/2+1)); L(np/2:-1:2)];

%%% Weighting of FT

function W = weight_ft(np, lambda_ratio, w)
% Input
%    np: number of samples for FFT (here: np=2048)
%        must be a power of 2
%    lambda_ratio: ratio of cut-off wavelength and maximum wavelength in
%                  Fourier spectrum
%    w: waviness of the signal spectrum (e.g. w=2.5)
% Output
%    W: weighting vector (double, np x 1)

   % compute first half of weighting function (array index 1:(np/2+1))
   index = 0:(np/2);
   tmp = (w-1)/2;
   num = (index*lambda_ratio).^(w-1);
   denom = 2^tmp - 2^(-tmp);
   W = sqrt(num/denom)';
   % mirror function at array index (np/2+1)
   W = [W(1:(np/2+1)); W(np/2:-1:2)];

%%% Weighted Longitudinal Profile calculation

function WLP = wlp_comp(z_wo, np_eval)
% Input
%    z_wo: octave band filtered weighted profiles
%          (double, np x nbands)
% Output
%    WLP: weighted longitudinal profile (double, np_eval x 1)

   np = size(z_wo, 1);
   shift = round((np-np_eval)/2);

   % symmetrically cut off data
   z_wo_eval = z_wo((shift+1):(shift+np_eval),:);

   % compute standard deviation of sum of octave-filtered profiles
   sigma_tot = std(sum(z_wo_eval,2), 1);
   % compute standard deviation of each octave-filtered profile
   sigma = std(z_wo_eval, 1);

   WLP = sum(z_wo_eval.*repmat(sigma,np_eval,1),2)/sigma_tot;
```