NMPB08 – PROPAGATION

Programmer's Guide

Loiodice Christellle - Van Maercke Dirk
GEOMOD / CSTB Grenoble
19/07/2011

Table of Contents

1 Getting started 4

1.1 Using the PropagationNMPB.dll 4

1.2 Create and delete the data structure 4

1.3 Passing in the geometry of the propagation path 5

1.4 Getting the results 6

1.5 Mutlitple sources, receivers and paths 7

1.6 Impedance jumps and relief 8

1.7 Barriers, screens and buildings 10

1.8 Reflections and lateral diffractions 12

1.9 Embankments 13

1.10 Errors 14

1.11 Utility functions 14

1.12 Options and advanced use 16

2 Worked out examples 18

2.1 Example 1: Straight line 18

2.2 Example 2: with a thin screen (vertical diffraction) 20

2.3 Example 3: with a thick screen 22

2.4 Example 4: reflection with a vertical obstacle 23

2.5 Example 5: embankment 24

1 Getting started

1.1 Using the PropagationNMPB.dll

In order to use the PropagationNMPB.dll in your software you must:

- include the PropagationNMPB.h header file in your code
- make calls to the functions defined in the header file
- link your application against the PropagationNMPB.dll library
- copy the PropagationNMPB.dll file in a directory where it is accessible from your application executable code ; the preferred option being to place the DLL file in the same directory as the executable code.

The first part of this manual provides a short overview of the functionalities implemented in the library and how to use these. The second part contains the detailed description of all functions, datastructures and constants that make up the the public programming interface (API) of the library. The third part contains the reference to all internal functions and data structures used inside the library; this part is intended to be used by programmers that need to maintain, to modify, to adapt or to optimise the library.

1.2 Create and delete the data structure

The PropagationNMPB calculation engine maintains an internal data structure in which are stored a complete description of the acoustical problem, a set of options and all intermediate and final results.

The first task of the host program is to allocate and to initialize this data structure using the `NMPB08_CreatePath function`. When done with the calculation the host program might want to free the internal data structure using `NMPB08_DeletePath`.

The functions and arguments for creating and cleaning up the internal data structure are:

```
void* PathID = NMPB08_CreatePath (void) ;

void NMPB08_DeletePath (void* PathID) ;
```

The `NMPB08_CreatePath` function will create and return the address of a new internal data structure. The calling program should use this value as the first argument in all other calls to functions in the PropagationNMPB calculation engine. The calling program should treat the value returned by `NMPB08_CreatePath` as a “handle” to a hidden data structure and make no assumptions about the structure or contents of this structure.

As a side effect, a call to NMPB08_CreatePath will initialize default values for:

- The frequency range is set to the default range defined in the NF S31-133 standard, i.e. 18 third octave bands, ranging from 100 to 5000 Hz,
- The sound speed is set to c = 340 m/s
- Air absorption is initialised to the predefined values given in the NF S31-133 standard.

Alternatively, the `NMPB08_CreatePathEx` function can be called to initialize the data structure with a user-defined frequency (see part 2).

To clean up the memory storage used by the **PropagationNMPB** module, the host program calls the `NMPB08_DeletePath` function. The void* argument must be a valid handle returned by the `NMPB08_CreatePath` function. After calling the `NMPB08_DeletePath` the handle becomes invalid and the host program must not use the handle in any call to another function inside the library.

Important: in order to keep the relations between frequencies, wavelengths, sound speed and propagation distances coherent, all input data must use MKS units. All distances, heights and other geometrical parameters must be entered in meters. If the host program uses different units (cm, inches, km, mph,…) these must be converted to MKS units before calling the **PropagationNMPB** API functions.

1.3 Passing in the geometry of the propagation path

A propation plane is a vertical plane connecting the source position to the receiver. In case of direct propagation from the source to the receiver, the propagation plane consists of a single plane. In case of reflections from obstacles or in case of diffraction around vertical edges of obstacles, the propagation plane consists of a set of connected planes.

Propagation within a propagation plane is entirely defined by the intersection of the (set of) vertical plane(s) with the ground and with all man-made or natural obstacles. This intersection is a single, albeit broken, line in 3D.

Your application passes the geometry of the propagation plane by creating an ordered set of points in the 3D space, using the `NMPB08_ExtendPath` function call. Each part (segment) of the intersection of the propagation plane with the underlying 3D model must be assigned an acoustical property. In the NMPB-2008 method, as in the ISO 9613-2 method, this acoustical property is coded as an equivalent absorption coefficient "G", i.e. a value between 0 (acoustically hard surfaces) and 1 (acoustically soft surfaces).

Note that in the NMPB-2008 method, the G-value of the first point defined the nature of the ground below the source and is used to calculate the source-specific correction term for G'_{trajet}. I.e. G should be equal to 0 in case of a road source and equal to 1 for a railway source on a ballast bed. For the next points, the G value passed to the call corresponds to the nature of the segment from the previous position to the new position.

The first point in the propagation plane corresponds to the foot print of the receiver on the ground (or on a roof in case the receiver is located above a building) and the last point to the foot point of the source. The height of the source and the receiver are set independently of the

segments describing the cross-section by means of the `NMPB08_SetSourceHeight` and `NMPB08_SetSourceHeight` functions.

The simplest propagation plane corresponds to only two points, one ground impedance value and one source and receiver heights, as illustrated in the following code segment:

```
#include "PropagationNMPB08.h"
// create the internal data structure
PathID pathNMPB = NMPB08_CreatePath () ;
// footprint of source
Position3D posSource;
posSource.x = 0;
posSource.y = 0;
posSource.z = 0;
// footprint of receiver
Position3D posReceiver;
posReceiver.x = 20;
posReceiver.y = 0;
posReceiver.z = 0;
// clear the interal geometry buffer
NMPB08_ClearPath (pathNMPB);
// pass in source position (hard ground)
NMPB08_ExtendPath (pathNMPB, &posSource, 0.0);
// pass in the receiver position (soft ground)
NMPB08_ExtendPath (pathNMPB, &posReceiver, 1.0);
// Set source and receiver height :
NMPB08_SetSourceHeight (pathNMPB, 0.05);
NMPB08_SetReceiverHeight (pathNMPB, 4.50);
// Calculate attenuation :
int err = NMPB08_DoCalculation (pathNMPB);
if(err == 0)
{
   // get results...
}
// done, cleanup memory :
NMPB08_DeletePath (pathNMPB) ;
```

After completing the definition of the propagation path, your application calls the `NMPB08_DoCalculation` function in order to perform the calculations. The function returns 0 if the calculation was performed without problems or an error code if appropriate.

1.4 Getting the results

To get the result, your application calls the `NMPB08_GetAttH` and `NMPB08_GetAttF` functions. `NMPB08_GetAttH` returns the calculation results in *homogeneous* conditions; `NMPB08_GetAttF` returns the calculation results for *moderate downward-refracting* conditions. The results are expressed in dB. Positive values indicate a decrease in sound level, positive value an increase.

The `NMPB08_GetAttH` and `NMPB08_GetAttF` functions return pointers to internal arrays of values. The size of these arrays matches the actual frequency range as setup in the calculation engine. Your may query the internal frequency range through the `NMPB08_GetNbFrequencies` and `NMPB08_GetFrequencies` functions.

Example:

```
int err = NMPB08_DoCalculation (pathNMPB);
if(err == 0)
{
    // Get size of frequency range :
    int nbFreq = NMPB08_GetNbFrequencies (pathNMPB);
    // Get frequency range :
    double const* freq = NMPB08_GetFrequencies (pathNMPB);
    // Get homogeneous conditions values :
    double const* attH = NMPB08_GetAttH (pathNMPB);
    // Get favorable conditions values :
    double const* attF = NMPB08_GetAttF (pathNMPB);
    // do something with the attenuation values...
    for (int index = 0 ; index < nbFreq ; index++)
    {
         cout << freq[index] << attH[i] << attF[i]
    }
}
```

1.5 Mutlitple sources, receivers and paths

The **PropagationNMPB** module stores the list of segment points as part of its internal data structure. In order to reuse the same handle for the calculation of multiple paths, the `NMPB08_ClearPath` function must be called to clear this list before the start of a new propagation plane. The `NMPB08_ExtendPath`, and `NMPB08_ExtendPathExt` functions add new vertex point at the end of the internal list.

The height of the source and the receiver are set independently of the segments describing the propagation plane. This allows your application to re-use the propagation plane in calculations that differ only in source and receiver height. Different source heights are required e.g. in case of a railway source. Multiple receiver heights allow optimised calculation of a set of receivers located on a single vertical line (i.e. sharing a single foot point).

Example:

```
double sourceHeights[3] = { 0.00, 0.50, 2.00 } ;
double receiverHeights[3] = { 1.50, 4.50, 7.50 } ;
// loop over receiver heights
for (int i = 0 ; i < 3 ; i++)
{
    // set receiver height
    NMPB08_SetReceiverHeight (pathNMPB, receiverHeight[i]);
    // loop over source heights
    for (int j = 0 ; j < 3 ; j++)
    {
```

```
            // set source height
            NMPB08_SetSourceHeight (pathNMPB, 0.05);
            // do the calculation and get results
            int err = NMPB08_DoCalculation (pathNMPB);
            if(err == 0)
            {
                  // Get attenuation values
                  double const* attH = NMPB08_GetAttH (pathNMPB);
                  double const* attF = NMPB08_GetAttF (pathNMPB);
                  // Do something...
            }
      }
}
```

1.6 Impedance jumps and relief

In the first example we assumed propagation over a soft surface, but a source on a hard platform. The transition from hard to soft ground is handled automatically in the NMPB-2008 method through the calculation of the G'_{trajet} correction. It may however be preferable to model impedance jumps explicitely. This is easily done by introducing a third vertex in the profile of the propagation path, as shown in the next example.

```
Position3D pos ;
// clear the path data
NMPB08_ClearPath () ;
// pass in footprint of the source position (hard ground)
pos.x = 0;
pos.y = 0;
pos.z = 0;
NMPB08_ExtendPath (pathNMPB, &pos, 0.0);
// hard surface from source till border of platform
pos.x = 10;
pos.y = 0;
pos.z = 0;
NMPB08_ExtendPath (pathNMPB, &pos, 0.0);
// soft surface from border to receiver
pos.x = 20;
pos.y = 0;
pos.z = 0;
NMPB08_ExtendPath (pathNMPB, &pos, 0.0);
// Set source and receiver height :
NMPB08_SetSourceHeight (pathNMPB, 0.05);
NMPB08_SetReceiverHeight (pathNMPB, 4.50);
```

Similarly, the altitude of the terrain can be passed directly to the `NMPB08_ExtendPath` function, as in the case of an elevated road platform:

```
Position3D pos ;
// clear the path data
NMPB08_ClearPath () ;
// pass in footprint of the source position (hard ground)
```

```
pos.x = 0;
pos.y = 0;
pos.z = 4;
NMPB08_ExtendPath (pathNMPB, &pos, 0.0);
// hard surface from source till border of platform
pos.x = 10;
pos.y = 0;
pos.z = 4;
NMPB08_ExtendPath (pathNMPB, &pos, 0.0);
// embankment, slope 2/3, soft ground
pos.x = 6;
pos.y = 0;
pos.z = 0;
NMPB08_ExtendPath (pathNMPB, &pos, 1.0);
// flat (soft) ground from embankment to receiver
pos.x = 20;
pos.y = 0;
pos.z = 0;
NMPB08_ExtendPath (pathNMPB, &pos, 1.0);
```

Different parts of the ground are connected by means of a vertical sustaining wall. This may be modelled by means of two points in the propagation plane, one exactly above the next:

```
Position3D pos ;
// clear the path data
NMPB08_ClearPath () ;
// footprint of the source position (hard ground)
pos.x = 0;
pos.y = 0;
pos.z = -4;
NMPB08_ExtendPath (pathNMPB, &pos, 0.0);
// hard surface from source till border of platform
pos.x = 10;
pos.y = 0;
pos.z = -4;
NMPB08_ExtendPath (pathNMPB, &pos, 0.0);
// vertical wall to natural terrain
pos.x = 10;
pos.y = 0;
pos.z = 0;
NMPB08_ExtendPath (pathNMPB, &pos, 0.0);
// flat ground from embankment to receiver
pos.x = 20;
pos.y = 0;
pos.z = 0;
NMPB08_ExtendPath (pathNMPB, &pos, 1.0);
```

Note that the vertical wall is characterised by a G value that will be ignored by the NMPB-2008 calculation scheme (because of the weighting function applied in the calculation of G_{trajet}). However, if the wall were slightly tilted away from the source, its value would enter the calculation.

1.7 Barriers, screens and buildings

As for the case of relief, man-made obstacles in the propagation plane are defined as segments that define the intersection of the propagation plane with the volumes created by the obstacles.

As an example, consider the excess attenuation for road, above a flat terrain with a building in between the source and the receiver. At 10m from the source, there is an 8 m-high building. To the right of the building, the ground impedance value changes to soft. The receiver is situated at 50m from the source. As above, we will create separare segments for the vertical walls and for the roof the building.

```
Position3D pos;
// clear the path buffer
NMPB08_ClearPath () ;
// the source position
pos.x = 0;
pos.y = 0;
pos.z = 0;
NMPB08_ExtendPath (pathNMPB, &pos, 0.0);
// hard ground to the building
pos.x = 10;
pos.y = 0;
pos.z = 0;
NMPB08_ExtendPath (pathNMPB, &pos, 0.0);
// vertical wall, 8 meter high
pos.x = 10;
pos.y = 0;
pos.z = 8;
NMPB08_ExtendPath (pathNMPB, &pos, 0.0);
//roof at z = 8m
pos.x = 20;
pos.y = 0;
pos.z = 8;
NMPB08_ExtendPath (pathNMPB, &pos, 0.0);
// vertical wall, down to z = 0
pos.x = 20;
pos.y = 0;
pos.z = 0;
NMPB08_ExtendPath (pathNMPB, &pos, 0.0);
// flat soft ground till receiver
pos.x = 50;
pos.y = 0;
pos.z = 0;
NMPB08_ExtendPath (pathNMPB, &pos, 1.0);
```

Important note: whether a segment point will give way to the calculation of attenuation due to diffraction or not is determinded automatically inside the calculation scheme, based on the geometrical construction of the Fermat path linking the source to the receiver respecting the constraints imposed by the shape of the boundary. This implies that the geometrical detection of diffracting eges is entirely automatic and does not require the host software to provide any

additional information. As a consequence, the libray does not offer the possibility to avoid and/or force calculation of a diffraction term from any of the segment points defining the boundary. Also, there is no means to distinguish diffraction caused by natural and/or man-made obstacles.

Thin barriers can be created in the same way by means of two vertical segments and no roof segment in between then. E.g. if the building were replaced by a thin barrier, 3m high, at 10m from the source, the code would become:

```
Position3D pos;
// clear the path buffer
NMPB08_ClearPath () ;
// the source position
pos.x = 0;
pos.y = 0;
pos.z = 0;
NMPB08_ExtendPath (pathNMPB, &pos, 0.0);
// hard ground to the barrier
pos.x = 10;
pos.y = 0;
pos.z = 0;
NMPB08_ExtendPath (pathNMPB, &pos, 0.0);
// the barrier, 3m high, is modelled by means of 3 segments
// left side of the barrier
pos.x = 10;
pos.y = 0;
pos.z = 3;
NMPB08_ExtendPath (pathNMPB, &pos, 0.0);
// right side of the barrier
pos.x = 10;
pos.y = 0;
pos.z = 0;
NMPB08_ExtendPath (pathNMPB, &pos, 0.0);
// flat soft ground till receiver
pos.x = 50;
pos.y = 0;
pos.z = 0;
NMPB08_ExtendPath (pathNMPB, &pos, 1.0);
```

However, it is equally possible to create the thin barrier by means of an "extension" element. Extension elements are created using the `NMPB08_ExtendPathExt` function that allows passing additional information to the calculation engine. The Propogation library includes many different extensions and each extension is identified by a unique code. For the creation of a thin barrier at a specified location, we use the `ETScreen_NMPB` extension code.

The next example is formally equal the previous one:

```
NMPB08_ClearPath () ;
// the source position
pos.x = 0;
pos.y = 0;
pos.z = 0;
```

```
NMPB08_ExtendPath (pathNMPB, &pos, 0.0);
// hard ground to the barrier at x = 10m
pos.x = 10;
pos.y = 0;
pos.z = 0;
// define the extension record, passing in the type and
// the height of the barrier
ExtensionNMPB ext;
ext.type = ETScreen_NMPB;
ext.height = 3.0;
// this call will create both the ground segment from z = 0
// to z = 10 (with G = 0.0) and a thin 3m high screen above
// end point of the segment (i.e. at z = 10m)
NMPB08_ExtendPathExt (pathNMPB, &pos, 0.0, &ext);
// flat soft ground till receiver
pos.x = 50;
pos.y = 0;
pos.z = 0;
NMPB08_ExtendPath (pathNMPB, &pos, 1.0);
```

Note: the use of the ETScreen_NMPB extension does not imply that a diffraction term will be accounted for in the calculation of the path (see note above); i.e. the ETScreen_NMPB extension defines a geometrical feature of the boundary beneath the propagation path whose effects on the calculations are solely defined by the geometrical (and physical) properties of the extra segments created by the extension record. As a consequence, both methods for creating a barrier (i.e. by means of a set of segments or by means of an extension) will lead to strictly identical acoustical results.

1.8 Reflections and lateral diffractions

Extension records can be used to indicate that the propagation plane contains reflections from vertical obstacles. The corresponding extension type is `ETReflection_NMPB`, the information to be passed to the library includes:

- the height of the reflecting obstacle
- a table of absorption coefficients of the surface (this table should have the same number of elements as there are frequencies defines in the calculation engine).

The reflecting obstacle is positioned at the end point of the segment that is it attached to.

As an example, consider the same situation as in the previous section but for a path including a reflection on a screen at the opposite side of the road. Note that the receiver has been moved to y = 70m and that it is assumed that the barriers are parallel to the (Oy) axis, so that the propagation plane makes an angle of 45 degrees with both screens.

```
// absorption values
double alphaSpec[] =
{0.1, 0.1, 0.2, 0.2, 0.3, 0.3, 0.4, 0.4, 0.5,
```

```
 0.5, 0.6, 0.6, 0.7, 0.7, 0.8, 0.8, 0.9, 0.9};
// clear the geometry buffer
NMPB08_ClearPath () ;
// data structures
Position3D    pos ;
ExtensionNMPB ext;
// the source position at x = 0m, y = 0m
pos.x = 0;
pos.y = 0;
pos.z = 0;
NMPB08_ExtendPath (pathNMPB, &pos, 0.0);
// reflection from the barrier at x = -10m, y = 10m
pos.x = -10;
pos.y = 10;
pos.z = 0;
ext.type = ETReflection_NMPB;
ext.height = 3.0 ;
ext.alphaArray = alphaSpec;
// diffraction over the barrier at x = 10m, y = 30m
pos.x = 10;
pos.y = 30;
pos.z = 0;
ext.type = ETScreen_NMPB;
ext.height = 3.0;
ext.alphaArray = 0;
NMPB08_ExtendPathExt (pathNMPB, &pos, 0.0, &ext);
// flat soft ground till receiver at x = 50m, y = 70m.
pos.x = 50;
pos.y = 70;
pos.z = 0;
NMPB08_ExtendPath (pathNMPB, &pos, 1.0);
```

Diffraction around vertical edges is encoded in the same way using the `ETSideDiffraction_NMPB` extension type. Note that the following limitations apply to laterally diffracted paths:

- The path can contain at most one diffracting obstacle, either a the border of a thin barrier or two borders belonging to the same thick obstacle.
- Any path may not contain either diffraction by a vertical edge or diffraction any number of top diffractions but combinations of lateral and top diffractions are not allowed.

1.9 Embankments

Optionnaly, the PropagationNMPB calculation engine will apply the corrections for the presence of embankments as prescribed in the NF S31-133 standard. In order to apply the corrections, the host software must:

- enable the CHECK_EMBANKMENT option (see below, by default, the option is disabled),

- pass extra information about the nature of the source (i.e. the correction only applies to a road source) and the position of the border of the road platform,
- indicate the segment that will be used as the candidate for the correction,
- indicate the angle of the propagation plane with the line source.

All these informations are passed as extension records. See below for a worked out example.

1.10 Errors

Most of the functions defined in the NMPB-2008 library return error codes. Error codes are of type `ErrorType` and enumeration in the header file:

```
ERRNone = 0           : no error catched
ERRNullPath = 10      : the PathID is NULL
ERRNoPoint = 11       : there is no point in the path
ERROnePoint = 12      : there is only one point in the path
ERRSideDiff = 13      : There are more than 2 side diffractions in the path
ERREmbankment = 14    : There are more than 1 embankment in the path
ERRFrequency = 20     : The frequency equals 0
ERRAttCoeffFrequency = 21   : frequency not found in array containing
                              attenuation coefficient
ERRAngle = 30         : angle not found in array containing favorable conditions
                        probabilities for the angles
ERRProbability = 40   : probability value > 1
ERRDivZero = 50       : division by 0
ERRSqrtNegative = 51  : square root of a negative number
ERRScreenAbsorption = 60    : screen absorption > 1
ERRUnknown = 100:     : unknown error catched
```

1.11 Utility functions

The PropagationNMPB library provides a set of utility functions that can be used outside the scope of a propagation path. These functions do not require a handle to an internal data structure and operate solely on application defined data.

The `NMPB08_SumLevels` function implements the energetic sum of sound levels ; it can be usefull to sum over frequency bands or to sum levels over different propagation paths. The function is defined as :

```
double NMPB08_SumLevels(int n, double const* levels);
```

The `NMPB08_GetFavorableConditionProbability` function can be used to obtain the percent of occurrence of favorable conditions for a given source-receiver direction.

```
double NMPB08_GetFavorableConditionProbability
           (Position3D const* source,
```

```
                    Position3D const* receiver,
                    int nbAngles, double const* fcpAngles,
                    double angleNorth);
```

The calling program must indicate the direction of North in the current coordinate system and pass a table of percentage of favorable per angular sector. The direction of north is defined as the angle of the north direction with the X-axis of the local coordinate system and measured positive in counter-clockwise sense. E.g. if the direction of north coincides with the positive Y axis, pass in a value of 90 degrees. The table of favorable conditions shall comply with those given in Annexe B of the NF S31-133 standard : the values in the table correspond to the angles of 20, 40, 60, … 360 degrees with respect to north and the angles are measured clockwise.

The `NMPB08_LTsound` function can be used to calculate the long-term sound level from the given sound levels under homogeneous and favorable conditions and the frequency of occurrence of favorable probability.

```
double NMPB08_LTsound(double Lp_h, double Lp_f, double p) ;
```

The `NMPB08_CalculateLeqLT` function calculates the specific sound level for a single propagation path. Given the sound power of the source, the attenuations under downward-refraction and homogeneous conditions and the frequency of occurrence of favorable propagation conditions, the function calculates the specific noise levels $L_{eq,H}$, $L_{eq,F}$ and $L_{eq,LT}$. All calculations are done in frequency bands. The calling application is responsible for allocating the arrays required for the storage of the calculated values.

```
int NMPB08_CalculateLeqLT (int nbFreq,
                           double* Lw,
                           double const* attH, double const* attF,
                           double fcp,
                           double* LeqH, double* LeqF, double* LeqLT);
```

Example…

```
// the direction of north is 20 degrees to the left of the (Oy) axis
double alphaN = 110;
// table of frequency of occurrence of favorable conditions
int nbAngles = 18;
double angleParray[] =
   {30, 28, 26, 25, 27, 28, 30, 32, 34,
    35, 36, 35, 34, 32, 32, 32, 32, 32};
// source levels for each frequency in third octave band
double Lw[] =
   {53.1, 54.1, 56.1, 59.1, 61.1, 64.1,
    66.1, 69.1, 69.1, 72.1, 73.1, 72.1,
    70.1, 67.1, 64.1, 62.1, 59.1, 57.1};
// allocate memory to store the sound levels
int nbFrequencies = NMPB08_GetNbFrequencies (pathNMPB);
double* Leq_H  = new double [nbFrequencies];
double* Leq_F  = new double [nbFrequencies];
double* Leq_LT = new double [nbFrequencies];
// loop over paths paths
for (...)
```

```
{
    // create path geometry
    // set source and receiver heights
    // do the calculation and get the results
    NMPB08_DoCalculation (pathNMPB);
    // Get attenuation values
    double const* attH = NMPB08_GetAttH (pathNMPB);
    double const* attF = NMPB08_GetAttF (pathNMPB);
    // get occurrence of favorable conditions
    // for this source and receveiver
    double pfc = NMPB08_GetFavorableConditionProbability
                 (&posSource, &posReceiver,
                  nbAngles, angleParray, alphaN);
    // calculate levels
    for (int i = 0 ; i < nbFrequencies ; i++)
    {
          Leq_H[i]  = Lw[i] - attH[i];
          Leq_F[i]  = Lw[i] - attF[i];
          Leq_LT[i] = NMPB08_LTsound (Leq_H[i], Leq_F[i], pfc);
    }
    // sum over frequencies in order to obtain dB(A) values
    double LeqA_H  = NMPB08_SumLevels(nbFrequencies, Leq_H);
    double LeqA_F  = NMPB08_SumLevels(nbFrequencies, Leq_F);
    double LeqA_LT = NMPB08_SumLevels(nbFrequencies, Leq_LT);
    // store the results
    // ...
}
```

1.12 Options and advanced use

Advanced features of the NMPB08 library can be activated (or inhibited) by means of options. Options are set or cleared using the `NMPB08_SetOption` function. The following options can be set of unset:

- EXCLUDE_ADIV — do not include geometrical divergence in the calculated attenuation values
- EXCLUDE_AATM — do not include atmospheric absorption in the calculated attenuation values
- TRACE_DETAILS — print out all calculation details on console
- CHECK_EMBANKMENT — perform corrections for embankments

NB : EXCLUDE_ADIV and EXCLUDE_AATM shall not be activated in an NMPB2008 compliant calculation.

As an example, a software developer might decide to use the NMPB-2008 library to calculate the excess attenuation in the propagation plane without any corrections for air absorption, for geometrical divergence and for lateral diffraction and reflections, because all these are already implemented in this software. In this case, the software will most likely use the NMPB08 library

in 2D mode (i.e. positions will be given by means of DZ coordinates in a deployed propagation plane).

Example:

```
NMPB08_SetOption(pathNMPB, EXCLUDE_ADIV, true);
NMPB08_SetOption(pathNMPB, EXCLUDE_AATM, true);

// assume the host defines the propagation path trhough
// an array of "impact" structures containing d, z and g values

NMPB08_ClearPath (pathNMPB) ;
for (int i = 0 ; i < impact.size() ; ++i)
{
     Position3D pos ;
     pos.x = impact[i].d ;
     pos.y = 0 ;
     pos.z = impact[i].z ;
     NMPB08_ExtendPath (pathNMPB, &pos, impact[i].g);
}
```

Note that DZ coordinates are easily obtained from 3D path coordinates by means of :

```
impact[0].d = 0 ;
for (int i = 1 ; i < impact.size() ; ++i)
{
     double x = impact[i].x – impact[i-1].x ;
     double y = impact[i].y – impact[i-1].y ;
     impact[i].d = impact[i-1].d + sqrt (x * x + y * y) ;
}
```

2 Worked out examples

2.1 Example 1: Straight line

In this example we calculate the excess attenuation for road, above a flat terrain. The source is situated at 5 cm above a road surface (impedance value = 0). After 6 m, the ground impedance value changes to 1. The receiver is situated at 20m from the source, 5m above local ground.

The code below shows how the problem is solved using the **PropagationNMPB** module.

```
// Path filling
PathID pathNMPB = NMPB08_CreatePath () ;
// P1
Position3D posSegment1;
posSegment1.x = 0;
posSegment1.y = 0;
posSegment1.z = 0;
// P2
Position3D posSegment2;
posSegment2.x = 6;
posSegment2.y = 0;
posSegment2.z = 0;
// P3
Position3D posSegment3;
posSegment3.x = 20;
posSegment3.y = 0;
posSegment3.z = 0;

// Source position
NMPB08_ExtendPath (pathNMPB, &posSegment1, 0);

// Changing impedance position (g = 0 between P1 and P2)
NMPB08_ExtendPath (pathNMPB, &posSegment2, 0);

// Receiver position (g = 1 between P2 and P3)
NMPB08_ExtendPath (pathNMPB, &posSegment3, 1);

// Set source and receiver height :
NMPB08_SetSourceHeight (pathNMPB, 0.05);
NMPB08_SetReceiverHeight (pathNMPB, 5);

// Calculate attenuation :
int err = NMPB08_DoCalculation (pathNMPB);
if(err == 0)
{
   // Get homogeneous conditions values :
   double const* attH = NMPB08_GetAttH (pathNMPB);
   // Get favorable conditions values :
```

```
    double const* attF = NMPB08_GetAttF (pathNMPB);

    // do something with the attenuation values...

  }

  // done, cleanup memory :
  NMPB08_DeletePath (pathNMPB) ;
```

The attenuation data can be used with utility functions:

Code that can be put in place of the "`// do something with the attenuation values...`" (Must be used before calling `NMPB08_DeletePath (pathNMPB)`)

```
  // the direction of north is 20 degrees to the left of the (Oy) axis
  double alphaN = 110;
  // favorable conditions probability according to directions:
  int nbAngles = 18;
  double angleParray[] = {30, 28, 26, 25, 27, 28, 30, 32, 34, 35,
  36, 35, 34, 32, 32, 32, 32, 32};
  // source sound levels for each frequency in third octave band
  double sourceSoundsArray[] = {53.1, 54.1, 56.1, 59.1, 61.1, 64.1,
     66.1, 69.1, 69.1, 72.1, 73.1, 72.1,
     70.1, 67.1, 64.1, 62.1, 59.1, 57.1};

  // get favorable probability in these conditions :
  double favourableProbability =
NMPB08_GetFavorableConditionProbability(&posSegment1, &posSegment3,
nbAngles, angleParray, alphaN);

  // get receiver sounds for each frequency :
  int nbFrequencies = NMPB08_GetNbFrequencies(pathNMPB);
  double* receiverSounds_h = new double[nbFrequencies];
  double* receiverSounds_f = new double[nbFrequencies];
  double* totalPathSoundLevel = new double[nbFrequencies];
  for (int i = 0 ; i < nbFrequencies ; i++)
  {
     receiverSounds_h[i] = sourceSoundsArray[i] - attH[i];
     receiverSounds_f[i] = sourceSoundsArray[i] - attF[i];

     totalPathSoundLevel[i] = NMPB08_LTsound(receiverSounds_h[i],
receiverSounds_f[i], favourableProbability);
  }

  // sum on frequencies :
  // get the sound level sum in homogeneous conditions :
  double Leq_h_sum = NMPB08_SumLevels(nbFrequencies, receiverSounds_h);
  // get the sound level sum in favorable conditions :
  double Leq_f_sum = NMPB08_SumLevels(nbFrequencies, receiverSounds_f);
  // get the total sound level sum
```

```
double LeqLT_sum = NMPB08_SumLevels(nbFrequencies,totalPathSoundLevel);

// do something with the sound level values...
```

2.2 Example 2: with a thin screen (vertical diffraction)

In this example we calculate the excess attenuation for road, above a flat terrain with a screen. The source is situated at 5 cm above a road surface (impedance value = 0). After 6 m, there is a 5m height screen and the ground impedance value changes to 1. The receiver is situated at 20m from the source, 5m above local ground.

The code below shows how the problem is solved using the **PropagationNMPB** module. There are two possibilities to fill the path :

- First possibility :

```
// Path filling
   PathID pathNMPB = NMPB08_CreatePath () ;
   // P1
   Position3D posSegment1;
   posSegment1.x = 0;
   posSegment1.y = 0;
   posSegment1.z = 0;
   // P2
   Position3D posSegment2a;
   posSegment2a.x = 6;
   posSegment2a.y = 0;
   posSegment2a.z = 0;
   Position3D posSegment2b;
   posSegment2b.x = 6;
   posSegment2b.y = 0;
   posSegment2b.z = 5;
   Position3D posSegment2c;
   posSegment2c.x = 6.01;
   posSegment2c.y = 0;
   posSegment2c.z = 0;
   // P3
   Position3D posSegment3;
   posSegment3.x = 20;
   posSegment3.y = 0;
   posSegment3.z = 0;

   // Source position
   NMPB08_ExtendPath (pathNMPB, &posSegment1, 0);

   // Changing impedance position (g = 0 between P1 and P2) and
screen item
   NMPB08_ExtendPath (pathNMPB, &posSegment2a, 0);
   NMPB08_ExtendPath (pathNMPB, &posSegment2b, 0);
   NMPB08_ExtendPath (pathNMPB, &posSegment2c, 0);

   // Receiver position (g = 1 between P2 and P3)
```

```
   NMPB08_ExtendPath (pathNMPB, &posSegment3, 1);

   // Set source and receiver height :
   NMPB08_SetSourceHeight (pathNMPB, 0.05);
   NMPB08_SetReceiverHeight (pathNMPB, 5);

   // Calculate attenuation :
   int err = NMPB08_DoCalculation (pathNMPB);
   if(err == 0)
   {

        // Get homogeneous conditions values :
        double const* attH = NMPB08_GetAttH (pathNMPB);
        // Get favorable conditions values :
        double const* attF = NMPB08_GetAttF (pathNMPB);

        // do something with the attenuation values...

   }

   // done, cleanup memory :
   NMPB08_DeletePath (pathNMPB) ;
```

- Second possibility :

```
// Path filling
   PathID pathNMPB = NMPB08_CreatePath () ;
   // P1
   Position3D posSegment1;
   posSegment1.x = 0;
   posSegment1.y = 0;
   posSegment1.z = 0;
   // P2
   Position3D posSegment2a;
   posSegment2a.x = 6;
   posSegment2a.y = 0;
   posSegment2a.z = 0;
   // P3
   Position3D posSegment3;
   posSegment3.x = 20;
   posSegment3.y = 0;
   posSegment3.z = 0;

   // Source position
   NMPB08_ExtendPath (pathNMPB, &posSegment1, 0);

   // screen item
   ExtensionNMPB ext;
   ext.type = ETScreen_NMPB;
   ext.height = 5;
   NMPB08_ExtendPathExt (pathNMPB, &posSegment2a, 0, &ext);
```

```
      // Receiver position (g = 1 between P2 and P3)
      NMPB08_ExtendPath (pathNMPB, &posSegment3, 1);

      // Set source and receiver height :
      NMPB08_SetSourceHeight (pathNMPB, 0.05);
      NMPB08_SetReceiverHeight (pathNMPB, 5);
```

2.3 Example 3: with a thick screen

In this example we calculate the excess attenuation for road, above a flat terrain with a screen. The source is situated at 5 cm above a road surface (impedance value = 0). After 10 m, there is an 8 m-high building and the ground impedance value changes to 1. The receiver is situated at 50m from the source, 5m above local ground.

The code below shows how the problem is solved using the **PropagationNMPB** module.

```
// Path filling
      PathID pathNMPB = NMPB08_CreatePath () ;
      // P1
      Position3D posSegment1;
      posSegment1.x = 0;
      posSegment1.y = 0;
      posSegment1.z = 0;
      // P2
      Position3D posSegment2a;
      posSegment2a.x = 10;
      posSegment2a.y = 0;
      posSegment2a.z = 0;
      Position3D posSegment2b;
      posSegment2b.x = 10;
      posSegment2b.y = 0;
      posSegment2b.z = 8;
      Position3D posSegment2c;
      posSegment2c.x = 20;
      posSegment2c.y = 0;
      posSegment2c.z = 8;
      Position3D posSegment2d;
      posSegment2d.x = 20;
      posSegment2d.y = 0;
      posSegment2d.z = 0;
      // P3
      Position3D posSegment3;
      posSegment3.x = 50;
      posSegment3.y = 0;
      posSegment3.z = 0;

      // Source position
      NMPB08_ExtendPath (pathNMPB, &posSegment1, 0);

      // Changing impedance position (g = 0 between P1 and P2) and
screen item
```

```
NMPB08_ExtendPath (pathNMPB, &posSegment2a, 0);
NMPB08_ExtendPath (pathNMPB, &posSegment2b, 0);
NMPB08_ExtendPath (pathNMPB, &posSegment2c, 0);
NMPB08_ExtendPath (pathNMPB, &posSegment2d, 0);

// Receiver position (g = 1 between P2 and P3)
NMPB08_ExtendPath (pathNMPB, &posSegment3, 1);

// Set source and receiver height :
NMPB08_SetSourceHeight (pathNMPB, 0.05);
NMPB08_SetReceiverHeight (pathNMPB, 5);

// Calculate attenuation :
int err = NMPB08_DoCalculation (pathNMPB);
if(err == 0)
{

      // Get homogeneous conditions values :
      double const* attH = NMPB08_GetAttH (pathNMPB);
      // Get favorable conditions values :
      double const* attF = NMPB08_GetAttF (pathNMPB);

      // do something with the attenuation values...

}

// done, cleanup memory :
NMPB08_DeletePath (pathNMPB) ;
```

2.4 Example 4: reflection with a vertical obstacle

In this example we calculate the excess attenuation for road, above a flat terrain, with a vertical obstacle. The source is situated at 5m above a road surface (impedance value = 0). After 25 m, on the left, there is a reflection on a vertical obstacle. The receiver is situated at 50m from the source, 5m above local ground.

The code below shows how the problem is solved using the **PropagationNMPB** module.

```
// Path filling
PathID pathNMPB = NMPB08_CreatePath () ;
// P1
Position3D posSegment1;
posSegment1.x = 0;
posSegment1.y = 10;
posSegment1.z = 0;
// P2
Position3D posSegment2;
posSegment2.x = 25;
posSegment2.y = 15;
posSegment2.z = 0;
// P3
```

```
Position3D posSegment3;
posSegment3.x = 50;
posSegment3.y = 10;
posSegment3.z = 0;

// Source position
NMPB08_ExtendPath (pathNMPB, &posSegment1, 0);

// Reflection
ExtensionNMPB ext;
Ext.type = ETReflection_NMPB;
Ext.height = 5;
double alphaSpec[] = {0.1, 0.1, 0.2, 0.2, 0.3, 0.3, 0.4, 0.4,
0.5, 0.5, 0.6, 0.6, 0.7, 0.7, 0.8, 0.8, 0.9, 0.9};
Ext.alphaArray = alphaSpec;

NMPB08_ExtendPathExt (pathNMPB, &posSegment2, 0, &ext);

// Receiver position
NMPB08_ExtendPath (pathNMPB, &posSegment3, 0);

// Set source and receiver height :
NMPB08_SetSourceHeight (pathNMPB, 5);
NMPB08_SetReceiverHeight (pathNMPB, 5);

int err = NMPB08_DoCalculation (pathNMPB);
if(err == 0)
{

      // Get homogeneous conditions values :
      double const* attH = NMPB08_GetAttH (pathNMPB);
      // Get favorable conditions values :
      double const* attF = NMPB08_GetAttF (pathNMPB);

      // do something with the attenuation values...

}

// done, cleanup memory :
NMPB08_DeletePath (pathNMPB) ;
```

2.5 Example 5: embankment

In this example we calculate the excess attenuation for road, above a terrain with embankment. The source is situated at 0.05m above a road surface (impedance value = 0). Until 6 m, there is a hard platform; then an embankment slope, 6m width, 4m high. The receiver is situated at 40m from the source, 5m above local ground.

The code below shows how the problem is solved using the **PropagationNMPB** module.

```
// Path filling
PathID pathNMPB = NMPB08_CreatePath () ;

// set embankment option
NMPB08_SetOption(pathNMPB, CHECK_EMBANKMENT, true);

// P1
Position3D posSegment1;
posSegment1.x = 0;
posSegment1.y = 10;
posSegment1.z = 0;
// P2
Position3D posSegment2;
posSegment2.x = 6;
posSegment2.y = 10;
posSegment2.z = 0;
// P3
Position3D posSegment3;
posSegment3.x = 12;
posSegment3.y = 10;
posSegment3.z = 4;
// P4
Position3D posSegment4;
posSegment4.x = 40;
posSegment4.y = 10;
posSegment4.z = 4;


// Source position
ExtensionNMPB ext1;
ext1.Type = ETRoadSource_NMPB ;
ext1. cosTheta = 1;
NMPB08_ExtendPathEx (pathNMPB, &posSegment1, 0, &ext1);

// Hard platform
ExtensionNMPB* ext2;
ext2.Type = ETPlatform_NMPB;
ext2. cosTheta = 1;
NMPB08_ExtendPathEx (pathNMPB, &posSegment2, 0, &ext2);

// Embankment
ExtensionNMPB ext3;
ext3.Type = ETEmbankment_NMPB;
ext3. cosTheta = 1;
NMPB08_ExtendPathEx (pathNMPB, &posSegment3, 0, &ext3);

// Receiver position
NMPB08_ExtendPath (pathNMPB, &posSegment4, 0);

// Set source and receiver height :
NMPB08_SetSourceHeight (pathNMPB, 0.05);
NMPB08_SetReceiverHeight (pathNMPB, 5);
```

```
int err = NMPB08_DoCalculation (pathNMPB);
if(err == 0)
{

      // Get homogeneous conditions values :
      double const* attH = NMPB08_GetAttH (pathNMPB);
      // Get favorable conditions values :
      double const* attF = NMPB08_GetAttF (pathNMPB);

      // do something with the attenuation values...

}

// done, cleanup memory :
NMPB08_DeletePath (pathNMPB) ;
```