

Threading B-Splines Through 2D Channels

1.0

Generated by Doxygen 1.8.17

1 The Channel2D Library Documentation	1
2 Introduction	3
3 Installing and compiling the library	7
4 The CurveBuilder class API	9
5 Using the library API	11
6 Examples and file formats	15
7 License	19
8 Acknowledgements	21
9 Module Index	23
9.1 Modules	23
10 Namespace Index	25
10.1 Namespace List	25
11 Hierarchical Index	27
11.1 Class Hierarchy	27
12 Class Index	29
12.1 Class List	29
13 File Index	31
13.1 File List	31
14 Module Documentation	33
14.1 Namespace channel.	33
14.1.1 Detailed Description	33
15 Namespace Documentation	35
15.1 channel Namespace Reference	35
15.1.1 Detailed Description	35
16 Class Documentation	37
16.1 channel::a3 Class Reference	37
16.1.1 Detailed Description	38
16.1.2 Member Function Documentation	39
16.1.2.1 a()	39
16.1.2.2 a1()	39

16.1.2.3 a1lower()	40
16.1.2.4 a1upper()	41
16.1.2.5 alower()	41
16.1.2.6 aupper()	42
16.1.2.7 degree()	43
16.1.2.8 h()	43
16.2 channel::Bound Class Reference	44
16.2.1 Detailed Description	44
16.2.2 Constructor & Destructor Documentation	45
16.2.2.1 Bound()	45
16.2.3 Member Function Documentation	45
16.2.3.1 get_row()	45
16.2.3.2 get_type()	46
16.2.3.3 get_value()	46
16.3 channel::Coefficient Class Reference	46
16.3.1 Detailed Description	47
16.3.2 Constructor & Destructor Documentation	47
16.3.2.1 Coefficient()	47
16.3.3 Member Function Documentation	48
16.3.3.1 get_col()	48
16.3.3.2 get_row()	48
16.3.3.3 get_value()	49
16.4 channel::CurveBuilder Class Reference	49
16.4.1 Detailed Description	54
16.4.2 Constructor & Destructor Documentation	54
16.4.2.1 CurveBuilder() [1/2]	54
16.4.2.2 CurveBuilder() [2/2]	55
16.4.3 Member Function Documentation	56
16.4.3.1 build()	56
16.4.3.2 compute_channel_corners_outside_sleeve_constraints()	57
16.4.3.3 compute_control_value_column_index()	59
16.4.3.4 compute_correspondence_constraints()	59
16.4.3.5 compute_index_of_corner_barycentric_coordinate()	61
16.4.3.6 compute_index_of_endpoint_barycentric_coordinate()	62
16.4.3.7 compute_min_max_constraints()	63
16.4.3.8 compute_normal_to_csection()	65
16.4.3.9 compute_normal_to_lower_envelope()	65
16.4.3.10 compute_normal_to_upper_envelope()	66
16.4.3.11 compute_second_difference_column_index()	67

16.4.3.12 compute_sleeve_corners_in_channel_constraints()	68
16.4.3.13 compute_sleeve_inside_csegment_constraints()	70
16.4.3.14 evaluate_bounding_polynomial()	74
16.4.3.15 get_bound_of_ith_constraint()	75
16.4.3.16 get_coefficient_identifier()	75
16.4.3.17 get_coefficient_value()	76
16.4.3.18 get_control_value()	77
16.4.3.19 get_degree()	78
16.4.3.20 get_lower_bound_on_second_difference_value()	78
16.4.3.21 get_lp_solver_result_information()	79
16.4.3.22 get_number_of_coefficients_in_the_ith_constraint()	81
16.4.3.23 get_number_of_constraints()	81
16.4.3.24 get_number_of_control_points()	82
16.4.3.25 get_number_of_csegments()	82
16.4.3.26 get_number_of_segments()	82
16.4.3.27 get_solver_error_message()	82
16.4.3.28 get_upper_bound_on_second_difference_value()	83
16.4.3.29 insert_bound()	84
16.4.3.30 insert_coefficient()	85
16.4.3.31 insert_csegment_constraint() [1/3]	86
16.4.3.32 insert_csegment_constraint() [2/3]	87
16.4.3.33 insert_csegment_constraint() [3/3]	88
16.4.3.34 insert_extreme_point_correspondence_constraint()	89
16.4.3.35 insert_linear_terms_of_epiece_point_bounds() [1/2]	90
16.4.3.36 insert_linear_terms_of_epiece_point_bounds() [2/2]	91
16.4.3.37 insert_min_max_constraints()	92
16.4.3.38 insert_nonlinear_terms_of_epiece_point_lower_bound() [1/2]	93
16.4.3.39 insert_nonlinear_terms_of_epiece_point_lower_bound() [2/2]	94
16.4.3.40 insert_nonlinear_terms_of_epiece_point_upper_bound() [1/2]	96
16.4.3.41 insert_nonlinear_terms_of_epiece_point_upper_bound() [2/2]	97
16.4.3.42 insert_periodic_correspondence_constraints()	99
16.4.3.43 insert_rhs_of_sleeve_corners_in_channel_constraints()	100
16.4.3.44 insert_rhs_of_sleeve_inside_csegment_constraints()	100
16.4.3.45 is_curve_closed()	101
16.4.3.46 is_equality()	101
16.4.3.47 is_greater_than_or_equal_to()	102
16.4.3.48 is_less_than_or_equal_to()	103
16.4.3.49 minimum_value()	104
16.4.3.50 set_up_lp_constraints()	104

16.4.3.51 set_up_objective_function()	105
16.4.3.52 set_up_structural_variables()	106
16.4.3.53 solve_lp()	108
16.5 channel::ExceptionObject Class Reference	110
16.5.1 Detailed Description	112
16.5.2 Constructor & Destructor Documentation	112
16.5.2.1 ExceptionObject() [1/4]	112
16.5.2.2 ExceptionObject() [2/4]	112
16.5.2.3 ExceptionObject() [3/4]	113
16.5.2.4 ExceptionObject() [4/4]	113
16.5.3 Member Function Documentation	114
16.5.3.1 get_description()	114
16.5.3.2 get_file()	114
16.5.3.3 get_line()	115
16.5.3.4 get_location()	115
16.5.3.5 get_name_of_class()	115
16.5.3.6 set_description() [1/2]	115
16.5.3.7 set_description() [2/2]	116
16.5.3.8 set_location() [1/2]	116
16.5.3.9 set_location() [2/2]	117
16.5.3.10 what()	117
16.6 channel::TabulatedFunction Class Reference	118
16.6.1 Detailed Description	118
16.6.2 Member Function Documentation	119
16.6.2.1 a()	119
16.6.2.2 alower()	119
16.6.2.3 aupper()	120
16.6.2.4 degree()	120
17 File Documentation	123
17.1 a3.hpp File Reference	123
17.1.1 Detailed Description	124
17.2 bound.hpp File Reference	125
17.2.1 Detailed Description	126
17.3 coefficient.hpp File Reference	126
17.3.1 Detailed Description	127
17.4 curvebuilder.cpp File Reference	128
17.4.1 Detailed Description	129
17.5 curvebuilder.hpp File Reference	129

17.5.1 Detailed Description	131
17.6 exceptionobject.hpp File Reference	131
17.6.1 Detailed Description	133
17.6.2 Macro Definition Documentation	133
17.6.2.1 treat_exception	133
17.7 tabulatedfunction.hpp File Reference	134
17.7.1 Detailed Description	135

Chapter 1

The Channel2D Library Documentation

- [Introduction](#)
- [Installing and compiling the library](#)
- [The CurveBuilder class API](#)
- [Using the library API](#)
- [Examples and file formats](#)
- [License](#)
- [Acknowledgements](#)

Chapter 2

Introduction

The `channel2d` Library consists of a set of C++ classes for solving planar instances of the **channel problem**. A detailed description of the channel problem and its solution (slightly different from the one implemented in the `channel2d` library) can be found in the following papers:

- David Lutterkort and Jörg Peters. Smooth paths in a polygonal channel. Proceedings of the 15th Annual ACM Symposium on Computational Geometry (SoCG), Miami Beach, FL, USA, June 13-16, 1999. ([PS](#))
- Ashish Myles and Jörg Peters. Threading splines through 3d channels. *Computer-Aided Design (CAD)*, v. 37, n. 2, pp. 139-148, 2005. ([PDF](#))

I really encourage you to read both papers (at least Section 3 of the second paper from top to bottom) before you try to use the `channel2d` library, as the input file format requires some idea about the input values and unknowns of the problem.

For the 2D version of the channel problem, we are given a channel, which is a planar region delimited by two polygonal chains: the *lower* and *upper envelopes* of the channel. For instance,

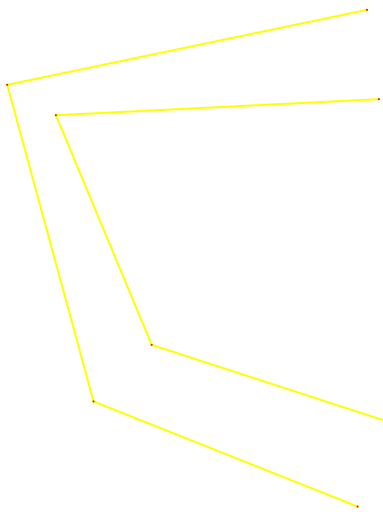


Figure 2.1 Example of an open channel

The two polygonal chains must have the same number of vertices (resp. edges). There is a one-to-one correspondence between the set of points (resp. edges) of the lower and upper envelopes. To be more precise, given the sequences of vertices (resp. edges) of the lower and upper envelopes, obtained by a *counterclockwise* traversal of the envelope, the i -th vertex (resp. edge) of the lower envelope is in correspondence with the i -th vertex (resp. edge) of the upper envelope. However, the two corresponding edges need not be parallel.

A solution for the problem is a C^k spline curve of a given degree d , with $k \geq 1$ and $d \geq 2$, which is entirely contained in the channel and whose endpoints belong to (distinct) extremities of the channel. For instance,

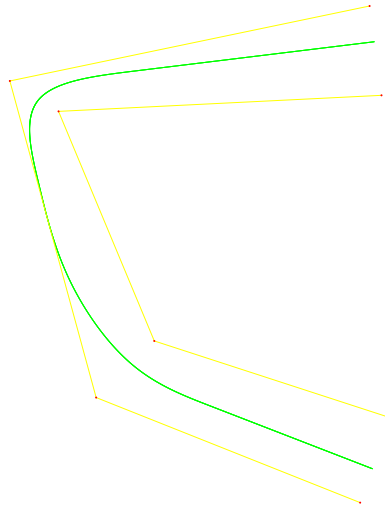


Figure 2.2 Example of a solution for the channel problem

The spline curve is shown in green and its control polygon is shown in blue. Myles and Peters devised a solution for the channel problem as a linear program whose constraints are responsible for keeping the spline inside the channel. In turn, the objective function can be tuned to influence on the geometry of the spline. In the `channel2d` library, the same objective function given by Myles and Peters' paper was adopted, which aims at minimizing the total curvature variation. This is done indirectly by defining a linear function based on the second differences of the Bézier coefficients of the curves that make up the spline.

The main differences between the solution implemented in the `channel2d` library and the one proposed by Myles and Peters are two-fold. First, the constraints of the linear program have been slightly modified, so that the resulting curve is C^2 rather than C^1 . Second, the resulting curve is always a cubic uniform b-spline curve, i.e., the degree d is fixed and equal to 3. In Myles and Peters' paper, the degree d of the curve is chosen by the user.

A channel can either be open (as in the previous example) or closed (as shown below).

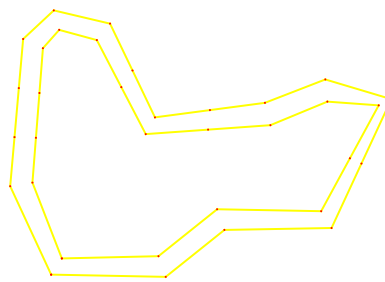


Figure 2.3 Example of a closed channel

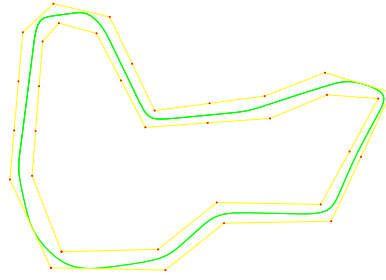


Figure 2.4 Example of a solution for the channel problem

To specify an instance of the channel problem, you must provide the Cartesian coordinates, $(lx_0, ly_0), \dots, (lx_n, ly_n)$ and $(ux_0, uy_0), \dots, (ux_n, uy_n)$, of the lower and upper envelopes, respectively, together with three parameter values: ns , nc , and *closed*. Parameter ns specifies the number of b-spline segments of the spline curve. Since the curve is a cubic b-spline, each b-spline curve segment is given by four consecutive control points. So, the number of control points of the curve is equal to $ns + 3$, and thus there is no need to specify the number of control points of the curve. Parameter nc specifies the number of *c-segments* of the channel. Each *c-segment* is given by a pair of corresponding edges of the lower and upper envelopes of the channel. The number of curve segments, ns , must be a multiple of the number of c-segment, nc . We have experimentally observed that choosing $ns = 3 \times nc$ is a good trade off between smoothness and number of control points of the curve. Note that the number of vertices in each envelope is $nc + 1$ if the channel is *open*, and equal to nc if the channel is *closed*.

For the first example of the channel problem I showed above, we have $ns = 9$ and $nc = 3$:

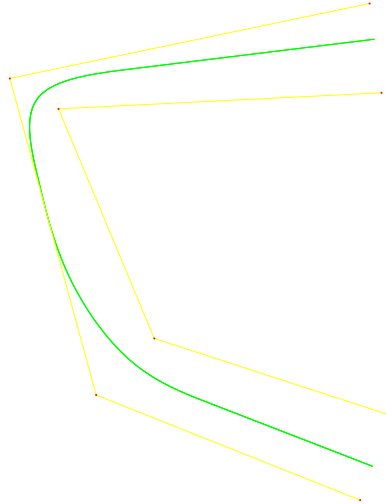


Figure 2.5 Example of a channel

That is, the spline consists of exactly $ns = 9$ curve segments. Starting from the first curve segment, each three consecutive curve segments are bounded (above and below) by the same pair of corresponding edges of the channel: an edge of the lower envelope and an edge of the upper envelope. Each envelope has exactly $nc = 3$ edges and $nc + 1 = 4$ vertices. The entire b-spline curve has $ns + 3 = 12$ control points.

For the second example of the channel problem I showed above, we have $ns = 51$ and $nc = 17$:

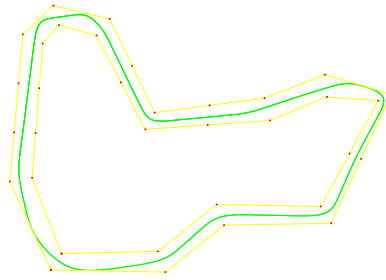


Figure 2.6 Example of a solution for the channel problem

That is, the spline consists of exactly $ns = 51$ curve segments. Starting from the first curve segment, each three consecutive curve segments are bounded (above and below) by the same pair of corresponding edges of the channel: an edge of the lower envelope and an edge of the upper envelope. Each envelope has exactly $nc = 17$ edges and $nc = 17$ vertices. The entire b-spline curve has $ns + 3 = 54$ control points.

It is worth mentioning that the channel problem may not have a solution if the value of ns is not large enough. In our experiments, letting $ns = 3 \times nc$ was sufficient to get a solution for all instances of the test dataset. But, if I had chosen $ns = 2 \times nc$, for instance, the code would not be able to build a few curves from the same dataset. If an instance of the channel problem has no solution, the main function of the `channel2d` library will show a message to indicate the infeasibility of the problem. In principle, the method could apply a midpoint subdivision to the curve and try to solve the problem again, but such an approach has not been implemented in the current version of the library. I also noted that the infeasibility of the problem is sometimes dictated by the channel geometry. As a rule of thumb, the lengths of any two consecutive c-segments of the channel should not differ by a factor greater than 2. This is even more critical when two consecutive c-segments meet at a sharp angle. I actually wrote code to refine channels, so that the lengths of any two consecutive c-segments of the channel do not differ by a factor greater than 2. This code has not been packaged together with the `channel2d` library code, but if you are interested in having a copy of it, please email me.

Chapter 3

Installing and compiling the library

To install the library `channel2d`, clone its source code from GitHub

```
git clone git@github.com:siqueirafm/channel2d.git
```

After doing so, you should see a directory named `channel2d` with subdirectories

```
bin data doc include lib scripts src LICENSE.md README.md
```

inside. Before you try to build code, though, you need to install the GNU package GLPK [GLPK](#) if it is not installed already. This package contains the linear program solver used by the `channel2d` library. If your computer runs Mac OSX, then you can install GLPK from `macports` or `homebrew`. If your computer is based on a Unix-like system, such as Linux, then you can follow the installation instructions in the GLPK documentation pages. If your computer runs Windows, then you may install GLPK by following the instructions you find [here](#). Once you have installed GNU GLPK in your computer, take note of the directories where the header file `glpk.h` and the lib file `libglpk.a` are. In my own computer (running on Linux), these files can be found in the following directories:

```
/usr/include
```

and

```
/usr/lib
```

At this point, you can build the library `Channel2D` using CMake. To that end, you need to assign values to the following three CMake variables:

- `CMAKE_INSTALL_PREFIX`
- `GLPK_INCLUDE_DIR`
- `GLPK_LIB_DIR`

The first variable should be assigned the absolute path to the folder of the library (i.e., the directory `channel2d`). The second variable should be assigned the absolute path to the header file `glpk.h` on your machine. Finally, the third variable should be assigned the absolute path to the header file `libglpk.a` on your machine. That's all!

Finally, open a terminal and type the following:

```
cmake -S . -B build -DGLPK_INCLUDE_DIR=<path to GLPK header file> -DGLPK_LIB_DIR=<path to GLPK lib file>
cmake --build build
cmake --install build <path to your directory channel2d>
```

If your machine runs Mac OSX, then you might want to replace the first command-line above to generate files of a XCode project:

```
cmake -G Xcode -S . -B build -DGLPK_INCLUDE_DIR=<path to GLPK header file> -DGLPK_LIB_DIR=<path to GLPK lib file>
```

If all goes well, then you should see the library file `Channel2D` inside subdirectory `lib`, and the executable `Channel2D-App` inside subdirectory `bin`, of your directory `channel2d`.

The current version of the library was successfully compiled and tested using the following operating system(s) / compiler(s).

- Ubuntu 20.04.4 LTS / GNU gcc version 9.4.0
- Mac OSX 11.6.1 / clang version 13.0.0

The `channel2d` library code is based on plain features of the C++ language. Apart from the GLPK functions, there is nothing that should prevent the code from being successfully compiled by any wide used and up-to-date C++ compiler that support C++ 11. However, if you face any problems, please feel free to contact me. Use the email address given inside the sources files of the library.

Chapter 4

The CurveBuilder class API

The main class of the `channel2d` library is `CurveBuilder`. To solve the channel problem, we first instantiate an object of this class using the class constructor:

```
CurveBuilder(
    size_t ns ,
    size_t nc ,
    bool closed ,
    double* lx ,
    double* ly ,
    double* ux ,
    double* uy
)
throw( ExceptionObject ) ;
```

as in

```
CurveBuilder* builder = 0 ;
try {
    builder = new CurveBuilder(
        np ,
        nc ,
        closed ,
        &lx[ 0 ] ,
        &ly[ 0 ] ,
        &ux[ 0 ] ,
        &uy[ 0 ]
    ) ;
}
catch ( const ExceptionObject& xpt ) {
    treat_exception( xpt ) ;
    exit( EXIT_FAILURE ) ;
}
```

Variables `ns` and `nc` hold the values of the parameters *ns* and *nc*, respectively, that we discussed in section [Introduction](#). Variable `closed` is boolean. If its value is `true`, then the channel is assumed to be closed. If its value is `false`, then the channel is assumed to be open. Variables `lx` and `ly` are two arrays of elements of type `double` that hold the *x* and *y* coordinates of the lower envelope of the channel. Likewise, variables `ux` and `uy` are two arrays hold of elements of type `double` that hold the *x* and *y* coordinates of the upper envelope of the channel. It is assumed that the vertices with coordinates $(lx[i], ly[i])$ and $(ux[i], uy[i])$ are corresponding vertices of the lower and upper envelopes, respectively. **IT IS VERY IMPORTANT** that the vertices are listed in the same order they are visited in a *counterclockwise traversal* of the envelopes (starting at one extreme of the channel). This is equivalent to walking along the edges of the envelopes from the "outside" of the channel in a counterclockwise direction. The reason for such a restriction is that my code must compute outward normals to the edges of the envelopes, and the direction of these normals matters! If the vertices are not given as they are found in a counterclockwise traversal of the envelope edges, the direction of the normals will be opposite to the correct one. As a result, the inequalities of the linear program will be incorrectly defined, which will prevent the solver from finding the correct optimal solution for the channel problem.

Once an instance of the channel problem is created, the next step is to find a solution for it. Class `CurveBuilder` offers the following method for solving the channel problem:

```
bool build( int& error ) ;
```

This method calls the GNU GLPK linear program (LP) solver to solve the instance of the channel problem defined by the constructor of the class `CurveBuilder`. If the solver finds a solution, `build` returns the logic value `true`. Otherwise, it returns the logic value `false`. In addition, the error code returned by the GLPK solver is stored in `error`. Using this error code, we can find out why the solver could not solve the problem. If the problem has been specified correctly (and if my code has no bug!), the fact that the solver cannot find a solution is mostly due to the infeasibility of the problem.

A typical call for `build()` is shown below:

```
int error ;
bool res = b.build( error ) ;
```

If the value of `res` is `true`, then we can recover the control points of the splines by invoking another function of class `CurveBuilder`:

```
double get_control_value( unsigned i , unsigned v ) const throw( ExceptionObject )
```

The above function has two input parameters: `i` and `v`. These parameters tell function `get_control_value` that we want the v -th coordinate of the i -th control point of the b-spline curve, i.e., $b_{i,v}$. Parameter `i` holds a value in the interval $[0, ns + 2]$. Parameter `v` holds the value 0 or 1, where 0 corresponds to the x coordinate and 1 corresponds to the y coordinate of $b_{i,v}$. The following piece of code prints out the coordinates of all control points of the spline found by the GNU GLPK solver:

```
for ( size_t i = 0 ; i < NumberOfControlPoints ; i++ ) {
    double x ;
    double y ;
    try {
        x = b.get_control_value( i , 0 ) ;
        y = b.get_control_value( i , 1 ) ;
    }
    catch ( const ExceptionObject& xpt ) {
        treat_exception( xpt ) ;
        ou.close() ;
        exit( EXIT_FAILURE ) ;
    }
}
```

The set of public methods of class `CurveBuilder` consists of many more functions. But, the ones presented here are enough to prescribe, solve, and obtain the solution of an instance of the channel problem. Section [Using the library API](#) describes a simple C++ program to read a file with the description of an instance of the channel problem, solve the problem using the functions I explained before, and then save the solution of the problem to an output file.

Chapter 5

Using the library API

I wrote a simple C++ program to show how to use the `channel2d` library to solve an instance of the channel problem. Here, I will examine and explain each line of the `main()` function of the program. You can find the program in the subdirectory `tst`. The program has only one file: `main.cpp`. Below are the header files included in `main.cpp`:

```
#include <iostream>           // std::cout, std::endl, std::cerr
#include <fstream>            // std::ifstream, std::ofstream
#include <sstream>            // std::stringstream
#include <string>             // std::string
#include <cstdlib>            // exit, EXIT_SUCCESS, EXIT_FAILURE, size_t
#include <iomanip>            // std::setprecision
#include <cassert>            // assert
#include <ctime>              // time, clock, CLOCKS_PER_SEC, clock_t
#include <cmath>              // fabs
#include "exceptionobject.hpp" // channel::ExceptionObject
#include "curvebuilder.hpp"    // channel::CurveBuilder
using std::cin ;
using std::cout ;
using std::cerr ;
using std::endl ;
using std::string ;
using channel::CurveBuilder ;
using channel::ExceptionObject ;
```

File `curvebuilder.hpp` contains the definition of class `CurveBuilder` and file `exceptionobject.hpp` contains the definition of a class, `ExceptionObject`, that I use to throw and treat exceptions in a more friendly way. The next lines check the command-line arguments and read an input file with the input values of an instance of the channel problem:

```
if ( ( argc != 3 ) && ( argc != 4 ) ) {
    cerr << "Usage: "
         << endl
         << "\t\t channel2d arg1 arg2 [ arg3 ]"
         << endl
         << "\t\t arg1: name of the file describing the polygonal channel"
         << endl
         << "\t\t arg2: name of the output file describing the computed cubic b-spline curve"
         << endl
         << "\t\t arg3: name of an output file to store a CPLEX format definition of the LP solved by this program (OPTIONAL)"
         << endl
         << endl ;
    cerr.flush() ;

    return EXIT_FAILURE ;
}
string fn1( argv[ 1 ] ) ;
size_t ns ;
size_t nc ;
bool closed ;
double* lx ;
double* ly ;
double* ux ;
double* uy ;
```

```

start = clock() ;
try {
    read_input( fn1 , ns , nc , closed , lx , ly , ux , uy ) ;
}
catch ( const ExceptionObject& xpt ) {
    treat_exception( xpt ) ;
    exit( EXIT_FAILURE ) ;
}

```

As we can see, the program requires two or three file names as command-line arguments. The first name refers to the file containing the input values of an instance of the channel problem. The second name refers to the file in which we want the program to write out the control points of the resulting spline curve, i.e., the solution of the channel problem. The third name is *optional* and refers to a file in which the program will store a description of the linear program corresponding to the instance of the channel problem given as input. I initially created this option as a way of debugging my code as needed. The description of the LP is given in CPLEX format, which is quite easy to read and look for mistakes. We can also give this description to any LP solver that takes in files in CPLEX format. The GNU GLPK itself is such a solver. We can use its `glpsol` function to solve an instance of a linear program written in CPLEX format. When I was done with the first version of the code, I thought it would be useful to leave the option of generating this file in the distributed version of the code.

After checking the number of input command-line arguments, the code reads in the input file using function `read_input()`. This function recovers the input values of the instance of the problem: `ns`, `nc`, `closed`, `lx`, `ly`, `ux`, and `uy`. I already talked about all these parameters. Observe that the memory occupied by the arrays `lx`, `ly`, `ux`, and `uy` is allocated inside function `read_input()`.

The next lines invoke the constructor of `CurveBuilder` to create the given instance of the channel problem:

```

CurveBuilder* builder = 0 ;
try {
    builder = new CurveBuilder(
        ns ,
        nc ,
        closed ,
        &lx[ 0 ] ,
        &ly[ 0 ] ,
        &ux[ 0 ] ,
        &uy[ 0 ]
    ) ;
}
catch ( const ExceptionObject& xpt ) {
    treat_exception( xpt ) ;
    exit( EXIT_FAILURE ) ;
}

```

Once the instance of the channel problem has been created, which is equivalent to saying that an object of class `CurveBuilder` has been instantiated, we can ask the object to solve the problem, which is done by invoking function `build()` (see section [The CurveBuilder class API](#)).

```

int error ;
bool res = builder->build( error ) ;

```

If this function returns `true`, the solver has found an optimal solution for the problem, and thus the code can recover the control points of the resulting spline. Otherwise, the code prints out a message to explain why the solver could not find a solution for the problem. This is done by examining the value of the variable `error` passed to function `build()`.

See below:

```

if ( res ) {
    string fn2( argv[ 2 ] ) ;
    write_solution( fn2 , *builder ) ;
}
else {
    cerr << endl
        << "ATTENTION: "
        << endl
        << builder->get_solver_error_message( error )
        << endl
        << endl ;
}

```

Function `get_solver_error_message()` from the API of class `CurveBuilder` is invoked when the solver cannot find a solution for the given instance of the channel problem. The GNU GLPK solver returns an error code that

allows us to know why the solver failed. When given this code, function `get_solver_error_message()` simply compares it with all error codes provided by the GLPK, and then returns a message explaining the meaning of the error code.

If a third file name is provided among the command-line arguments, then a description of the linear program corresponding to the given instance of the channel problem is written out to a file using the CPLEX format. As I mentioned before, such an output is only necessary if we want to verify whether my code was able to assemble the correct linear program. Another possible use for it is when the GNU GLPK solver is not able to find a solution. We can then give the linear program to another solver or to the `glpsol` function of the GNU GLPK to obtain more information on why the problem could not be solved. It might be the case that additional information can actually tell us the exact point of the channel that caused infeasibility of the problem.

```
if ( argc == 4 ) {
    string fn3( argv[ 3 ] );
    write_lp( fn3 , *builder );
}
```

The remaining of the `main()` function just releases memory:

```
if ( lx != 0 ) delete[] lx ;
if ( ly != 0 ) delete[] ly ;
if ( ux != 0 ) delete[] ux ;
if ( uy != 0 ) delete[] uy ;
if ( builder != 0 ) delete builder ;
return EXIT_SUCCESS ;
```

The auxiliary functions of the program are `read_input()`, `write_solution()`, and `write_lp()`. I will only comment on the code of the second function.

Function `write_solution()` must obtain the control points of the resulting spline in order to write them out to a file. This is done by invoking function `get_control_point()` of class `CurveBuilder` as explained in section [The CurveBuilder class API](#). Below is the body of `write_solution()`:

```
using std::endl ;
std::ofstream ou( fn.c_str() ) ;
if ( ou.is_open() ) {
    ou << std::setprecision( 6 ) << std::fixed ;
    const size_t NumberOfControlPoints = b.get_number_of_control_points() ;
    ou << NumberOfControlPoints
        << '\t'
        << 3
        << endl ;

    for ( size_t i = 0 ; i < NumberOfControlPoints ; i++ ) {
        double x ;
        double y ;
        try {
            x = b.get_control_value( i , 0 ) ;
            y = b.get_control_value( i , 1 ) ;
        }
        catch ( const ExceptionObject& xpt ) {
            treat_exception( xpt ) ;
            ou.close() ;
            exit( EXIT_FAILURE ) ;
        }
        ou << x << '\t' << y << endl ;
    }
    ou.close() ;
}
```


Chapter 6

Examples and file formats

To solve the channel problem using the `main()` function I described in Section [Using the library API](#), you must give the function a `.chn` file. This file must contain the complete information about one particular instance of the channel problem. The *first line* of the file contains the values of the input parameters

```
ns nc closed
```

in this order, where *ns* is the number of curve segments of the entire b-spline curve, *nc* is the number of c-segments of the channel, and *closed* is a flag to indicate whether the channel is open or closed. See section [Introduction](#) for a detailed description of the above parameters. After the first line, there are $2 \times nn$ lines, where $nn = nc + 1$ if the curve is open, and $nn = nc$ if the curve is closed. Each line contains the first and second Cartesian coordinates of a vertex of the lower envelope of the channel:

```
lx[0] ly[0]  
lx[1] ly[1]  
...  
lx[nn - 1] ly[nn - 1]
```

Recall that the coordinates must be given in the same order their corresponding vertices appear in a counterclockwise traversal of the "outside" of the lower envelope, from one extreme of the channel to the other. Right after the coordinates of the vertices of the lower envelope, the coordinates of the vertices of the upper envelope are listed using the same rules:

```
ux[0] uy[0]  
ux[1] uy[1]  
...  
ux[nn - 1] uy[nn - 1]
```

Recall also that $(lx[i], ly[i])$ and $(ux[i], uy[i])$ must be coordinates of the corresponding vertices of the lower and upper envelopes, respectively.

Here is an example of a typical `.chn` file:

```
9      3      0  
639.130835  36.518734  
632.034992  36.165892  
634.138728  31.121699  
639.338308  29.430348  
638.869165  38.481266
```

```

630.965008      36.834108
632.861272      29.878301
638.661692      27.569652

```

The above file describes the *open* channel

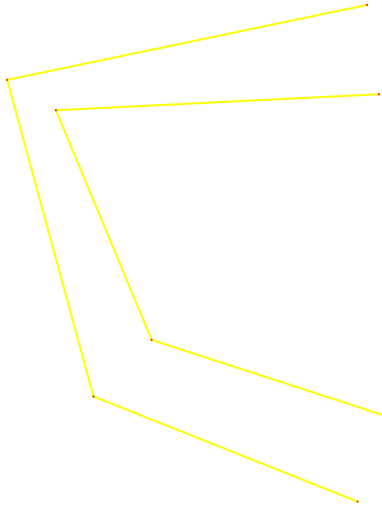


Figure 6.1 Example of a channel

and asks for a b-spline of degree 3 consisting of $ns = 9$ curve segments. Starting from the first segment, each three consecutive segments are delimited by one c-segment of the channel (i.e., by only one pair of edges). Each envelope of the channel has $nn = 4$ vertices, and the channel is open. Observe that $nn = nc + 1$. Function `read_input()` (see section [Using the library API](#)) reads in the input `.chn` file and obtains the values of ns , nc , nn , lx , ly , ux , and uy . Once the problem is solved, the program generates an output file with extension `.spl`. This file contains the Cartesian coordinates of the control points of the entire b-spline curve. The first line of a `.spl` file specifies the total number of control points and the degree of the spline (which is always 3 equal to), i.e.,

`n cp dg`

After the first line, there are `n cp` lines. Each line specifies the pair of Cartesian coordinates of a control point. These coordinates are listed as follows:

```

b0,x, b0,y
b1,x, b1,y
⋮
bn cp-1,x, bn cp-1,y

```

where $b_{n cp-1,x}$ and $b_{n cp-1,y}$ are the first and second Cartesian coordinates of the i -th control point of the p -th Bézier curve of resulting spline. Below, you find the `.spl` file corresponding to the solution of the instance of the channel problem described by the `.chn` file given above, as well as a plot of the spline and its control points:

```

12      3
641.603639      38.017630
638.973833      37.696253
636.344027      37.374876
633.714221      37.053499
631.084414      36.732122
631.590112      34.617083
632.095810      32.502043
633.577118      30.387004

```


635.362191	29.695979
637.147265	29.004955
638.932338	28.313930
640.717412	27.622906

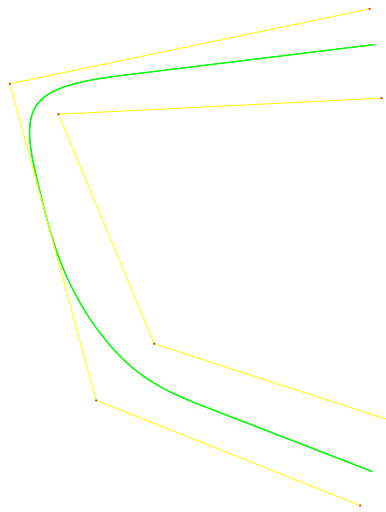


Figure 6.2 Example of a solution for the channel problem

You can find more examples of .chn files in the subdirectory `data/channels`. I wrote a script, `run.sh`, that executes `channel2d` on every input file in subdirectory `data/channels`, and then save the resulting .spl files in subdirectory `data/spcurves`. If your computer runs Mac OSX or a Unix-like system, then you can execute `run.sh`

```
sh run.sh
```

inside subdirectory `scripts`. I didn't provide any GUI to visualize the curves specified by the .spl files. If you decide to write your own .chn file to be tested by my program, execute the line below inside subdirectory `bin`, where the program `channel2d` should be located:

```
channel2d < your input CHN file > < your output SPL file >
```

If you want to see the instance of the linear program assembled by my program and solved by the GLPK solver, execute the line

```
channel2d < your input CHN file > < your output SPL file > < your output LP file >
```

When the execution ends, the third file stores a description of the instance of the linear program using the CPLEX language. Usually, we save such a file with the extension .lp. You can use the function `glpsol` of the GNU GLPK to solve the linear program written in CPLEX language. To that end, execute:

```
glpsol --lp < your LP file >
```

I am assuming that you installed GLPK in your computer and that the path to function `glpsol` is known. By executing `glpsol`, you can compare the solution given by this function with the solution produced by my code. They should be the same! If that is not the case, then I made a mistake when writing the code for generating the CPLEX description of the instance of the linear program that solves the channel problem.

Chapter 7

License

Copyright Notice

Copyright © 2016 Marcelo Siqueira. All rights reserved.

Terms and Conditions

This program is distributed WITHOUT ANY WARRANTY, and it may be freely redistributed under the condition that the copyright notices are not removed, and no compensation is received. Private, research, and institutional use is free. Distribution of this code as part of a commercial system is permissible ONLY BY DIRECT ARRANGEMENT WITH THE AUTHOR.

Chapter 8

Acknowledgements

I would like to acknowledge Dr. Jörg Peters for hosting me at CISE-UFL during my sabbatical year (2015-2016), and for patiently helped me understand the papers that underlie the `channel2d` library code.

Chapter 9

Module Index

9.1 Modules

Here is a list of all modules:

Namespace channel.	33
----------------------------	----

Chapter 10

Namespace Index

10.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

[channel](#)

The namespace `channel` contains the definition and implementation of a set of classes for threading a cubic b-spline curve into a given planar channel delimited by two polygonal chains [35](#)

Chapter 11

Hierarchical Index

11.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

channel::Bound	44
channel::Coefficient	46
channel::CurveBuilder	49
exception	
channel::ExceptionObject	110
channel::TabulatedFunction	118
channel::a3	37

Chapter 12

Class Index

12.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

channel::a3	This class represents two-sided, piecewise linear enclosures for two polynomial functions of degree 3 in Bézier form	37
channel::Bound	This class represents the type of a constraint (i.e., equality or inequality) and the value of its right-hand side: a real number	44
channel::Coefficient	This class represents a nonzero coefficient of an unknown of a constraint (inequality or equality) of a linear program instance	46
channel::CurveBuilder	This class provides methods for threading a cubic b-spline curve through a planar channel delimited by a pair of polygonal chains	49
channel::ExceptionObject	This class extends class <i>exception</i> of STL and provides us with a customized way of handling exceptions and showing error messages	110
channel::TabulatedFunction	This class represents two-sided, piecewise linear enclosures of a set of $(d - 1)$ polynomial functions of degree d in Bézier form. The enclosures must be made available by implementating a pure virtual method in derived classes	118

Chapter 13

File Index

13.1 File List

Here is a list of all documented files with brief descriptions:

a3.hpp	Definition of a class for representing piecewise linear enclosures of certain cubic polynomial functions in Bézier form	123
bound.hpp	Definition of a class for representing the type of a linear constraint (i.e., equality or inequality) and its right-hand side: a real number	125
coefficient.hpp	Definition of a class for representing a nonzero coefficient of an unknown of a constraint (inequality or equality) of a linear program instance	126
curvebuilder.cpp	Implementation of a class for threading a b-spline curve of degree 3 through a planar channel defined by a pair of polygonal chains	128
curvebuilder.hpp	Definition of a class for threading a b-spline curve of degree 3 through a planar channel defined by a pair of polygonal chains	129
exceptionobject.hpp	Definition of a class for handling exceptions	131
tabulatedfunction.hpp	Definition of an abstract class for representing piecewise linear enclosures of certain polynomial functions of arbitrary degree	134

Chapter 14

Module Documentation

14.1 Namespace channel.

Namespaces

- [channel](#)

The namespace channel contains the definition and implementation of a set of classes for threading a cubic b-spline curve into a given planar channel delimited by two polygonal chains.

14.1.1 Detailed Description

Chapter 15

Namespace Documentation

15.1 channel Namespace Reference

The namespace channel contains the definition and implementation of a set of classes for threading a cubic b-spline curve into a given planar channel delimited by two polygonal chains.

Classes

- class [a3](#)
This class represents two-sided, piecewise linear enclosures for two polynomial functions of degree 3 in Bézier form.
- class [Bound](#)
This class represents the type of a constraint (i.e., equality or inequality) and the value of its right-hand side: a real number.
- class [Coefficient](#)
This class represents a nonzero coefficient of an unknown of a constraint (inequality or equality) of a linear program instance.
- class [CurveBuilder](#)
This class provides methods for threading a cubic b-spline curve through a planar channel delimited by a pair of polygonal chains.
- class [ExceptionObject](#)
This class extends class exception of STL and provides us with a customized way of handling exceptions and showing error messages.
- class [TabulatedFunction](#)
This class represents two-sided, piecewise linear enclosures of a set of $(d - 1)$ polynomial functions of degree d in Bézier form. The enclosures must be made available by implementing a pure virtual method in derived classes.

15.1.1 Detailed Description

The namespace channel contains the definition and implementation of a set of classes for threading a cubic b-spline curve into a given planar channel delimited by two polygonal chains.

Chapter 16

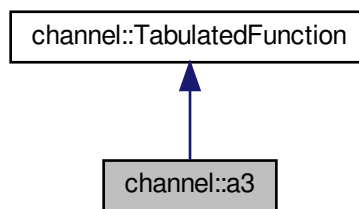
Class Documentation

16.1 channel::a3 Class Reference

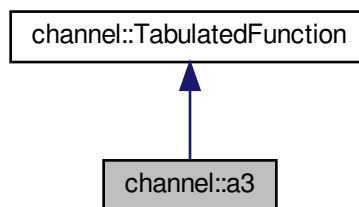
This class represents two-sided, piecewise linear enclosures for two polynomial functions of degree 3 in Bézier form.

```
#include <a3.hpp>
```

Inheritance diagram for channel::a3:



Collaboration diagram for channel::a3:



Public Member Functions

- `a3 ()`
Creates an instance of this class.
- `double alower (const size_t i, const double u) const` override
Evaluates the piecewise linear function corresponding to the lower enclosure of the i -th tabulated function at a point in $[0, 1]$.
- `double aupper (const size_t i, const double u) const` override
Evaluates the piecewise linear function corresponding to the upper enclosure of the i -th tabulated function at a point in $[0, 1]$.
- `double a (const size_t i, const double u) const` override
Computes the value of the i -th polynomial function a_i at a given point of the interval $[0, 1]$ of the real line.
- `unsigned degree () const` override
Returns the degree of tabulated functions.

Protected Member Functions

- `double a1lower (const double u) const`
Compute the image of a given point of the interval $[0, 1]$ under the lower enclosure of function a_1 .
- `double a1upper (const double u) const`
Compute the image of a given point of the interval $[0, 1]$ under the upper enclosure of function a_1 .
- `double a1 (const double u) const`
Computes the value of the cubic polynomial function a_1 at a given point of the interval $[0, 1]$ of the real line.
- `double h (const double u) const`
Computes the value of a piecewise linear hat function at a given point of the real line.

Protected Attributes

- `double _l0`
1st control value of the lower enclosure of the polynomial a_1 .
- `double _l1`
2nd control value of the lower enclosure of the polynomial a_1 .
- `double _l2`
3rd control value of the lower enclosure of the polynomial a_1 .
- `double _l3`
4th control value of the lower enclosure of the polynomial a_1 .

16.1.1 Detailed Description

This class represents two-sided, piecewise linear enclosures for two polynomial functions of degree 3 in Bézier form.

Attention

This class is based on the work described in

J. Peters and X. Wu.
On the optimality of piecewise linear max-norm
enclosures based on siefes. In Proceedings of the
2002 St Malo conference on Curves and Surfaces, 2003.

Definition at line 72 of file a3.hpp.

16.1.2 Member Function Documentation

16.1.2.1 a()

```
double channel::a3::a (
    const size_t i,
    const double u ) const [inline], [override], [virtual]
```

Computes the value of the i -th polynomial function a at a given point of the interval $[0, 1]$ of the real line.

Parameters

i	Index of the i -th polynomial function.
u	A parameter point in the interval $[0, 1]$.

Returns

The value of the i -th polynomial function a at a given point u of the interval $[0, 1]$ of the real line.

Implements [channel::TabulatedFunction](#).

Definition at line 207 of file a3.hpp.

```
212     {
213         if ( ( i != 1 ) && ( i != 2 ) ) {
214             std::stringstream ss( std::stringstream::in | std::stringstream::out );
215             ss << "Index of the polynomial function is out of range" ;
216             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
217         }
218
219         if ( ( u < 0 ) || ( u > 1 ) ) {
220             std::stringstream ss( std::stringstream::in | std::stringstream::out );
221             ss << "Parameter value must belong to the interval [0,1]" ;
222             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
223         }
224
225         return ( i == 1 ) ? a1( u ) : a1( 1 - u ) ;
226     }
```

References [a1\(\)](#).

16.1.2.2 a1()

```
double channel::a3::a1 (
    const double u ) const [inline], [protected]
```

Computes the value of the cubic polynomial function a_1 at a given point of the interval $[0, 1]$ of the real line.

Parameters

u	A parameter point in the interval $[0, 1]$.
-----	--

Returns

The value of the cubic polynomial function a_1 at a given point of the interval $[0, 1]$ of the real line.

Definition at line 313 of file a3.hpp.

```

314 {
315 #ifdef DEBUGMODE
316     assert( u >= 0 ) ;
317     assert( u <= 1 ) ;
318 #endif
319
320     return -u * ( 2 - u * ( 3 - u ) ) ;
321 }
```

Referenced by a().

16.1.2.3 a1lower()

```

double channel::a3::allower (
    const double u ) const [inline], [protected]
```

Compute the image of a given point of the interval $[0, 1]$ under the lower enclosure of function a_1 .

Parameters

u	A point in the interval $[0, 1]$.
-----	------------------------------------

Returns

The image of a given point of the interval $[0, 1]$ under the lower enclosure of function a_1 .

Definition at line 263 of file a3.hpp.

```

264 {
265     const double onethird = double( 1 ) / 3 ;
266
267     double res = _l0 * h( u )
268                 + _l1 * h( u - onethird )
269                 + _l2 * h( u - 2 * onethird )
270                 + _l3 * h( u - 1 ) ;
271
272     return res ;
273 }
```

References $_l0$, $_l1$, $_l2$, $_l3$, and h().

Referenced by alower().

16.1.2.4 a1upper()

```
double channel::a3::a1upper (
    const double u ) const [inline], [protected]
```

Compute the image of a given point of the interval $[0, 1]$ under the upper enclosure of function a_1 .

Parameters

u	A point in the interval $[0, 1]$.
-----	------------------------------------

Returns

The image of a given point of the interval $[0, 1]$ under the upper enclosure of function a_1 .

Definition at line 288 of file a3.hpp.

```
289 {
290     const double onethird = double( 1 ) / 3 ;
291
292     double res = -10 * h( u - onethird ) - 8 * h( u - 2 * onethird ) ;
293
294     res *= ( double(1) / 27 ) ;
295
296     return res ;
297 }
```

References `h()`.

Referenced by `aupper()`.

16.1.2.5 alower()

```
double channel::a3::alower (
    const size_t i,
    const double u ) const [inline], [override], [virtual]
```

Evaluates the piecewise linear function corresponding to the lower enclosure of the i -th tabulated function at a point in $[0, 1]$.

Parameters

i	The index of the i -th polynomial function.
u	A value in the interval $[0, 1]$.

Returns

The value of the piecewise linear function corresponding to the lower enclosure of the i -th tabulated function at a point in $[0, 1]$.

Implements [channel::TabulatedFunction](#).

Definition at line 133 of file a3.hpp.

```

138     {
139         if ( ( i != 1 ) && ( i != 2 ) ) {
140             std::stringstream ss( std::stringstream::in | std::stringstream::out );
141             ss << "Index of the polynomial function is out of range" ;
142             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
143         }
144
145         if ( ( u < 0 ) || ( u > 1 ) ) {
146             std::stringstream ss( std::stringstream::in | std::stringstream::out );
147             ss << "Parameter value must belong to the interval [0,1]" ;
148             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
149         }
150
151         return ( i == 1 ) ? a1lower( u ) : a1lower( 1 - u ) ;
152     }

```

References [a1lower\(\)](#).

16.1.2.6 aupper()

```

double channel::a3::aupper (
    const size_t i,
    const double u ) const [inline], [override], [virtual]

```

Evaluates the piecewise linear function corresponding to the upper enclosure of the i -th tabulated function at a point in $[0, 1]$.

Parameters

i	The index of the i -th polynomial function.
u	A value in the interval $[0, 1]$.

Returns

The value of the piecewise linear function corresponding to the upper enclosure of the i -th tabulated function at a point in $[0, 1]$.

Implements [channel::TabulatedFunction](#).

Definition at line 170 of file a3.hpp.

```

175     {
176         if ( ( i != 1 ) && ( i != 2 ) ) {
177             std::stringstream ss( std::stringstream::in | std::stringstream::out );
178             ss << "Index of the polynomial function is out of range" ;
179             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
180         }
181
182         if ( ( u < 0 ) || ( u > 1 ) ) {
183             std::stringstream ss( std::stringstream::in | std::stringstream::out );
184             ss << "Parameter value must belong to the interval [0,1]" ;
185             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
186         }
187
188         return ( i == 1 ) ? a1upper( u ) : a1upper( 1 - u ) ;
189     }

```

References [a1upper\(\)](#).

16.1.2.7 degree()

```
unsigned channel::a3::degree ( ) const [inline], [override], [virtual]
```

Returns the degree of tabulated functions.

Returns

The degree of the tabulated functions.

Implements [channel::TabulatedFunction](#).

Definition at line 237 of file a3.hpp.

```
238 {
239     return 3 ;
240 }
```

16.1.2.8 h()

```
double channel::a3::h (
    const double u ) const [inline], [protected]
```

Computes the value of a piecewise linear hat function at a given point of the real line.

Parameters

<i>u</i>	A parameter point of the real line.
----------	-------------------------------------

Returns

The value of a piecewise linear hat function at a given point of the real line.

Definition at line 336 of file a3.hpp.

```
337 {
338     const double onethird = 1.0 / 3.0 ;
339
340     if ( u <= -onethird ) {
341         return 0 ;
342     }
343     else if ( u <= 0 ) {
344         return 3 * u + 1 ;
345     }
346     else if ( u <= onethird ) {
347         return 1 - 3 * u ;
348     }
349
350     return 0 ;
351 }
```

Referenced by [a1lower\(\)](#), and [a1upper\(\)](#).

The documentation for this class was generated from the following file:

- [a3.hpp](#)

16.2 channel::Bound Class Reference

This class represents the type of a constraint (i.e., equality or inequality) and the value of its right-hand side: a real number.

```
#include <bound.hpp>
```

Public Types

- enum [CONSTRAINTTYPE](#) { **EQT**, **LTE**, **GTE** }

Defines a type for the type of a constraint.

Public Member Functions

- [Bound](#) ()
Creates an instance of this class.
- [Bound](#) (const [CONSTRAINTTYPE](#) type, const double value, const size_t row)
Creates an instance of this class.
- [CONSTRAINTTYPE](#) [get_type](#) () const
Returns the type of the constraint associated with this bound.
- double [get_value](#) () const
Returns the value of this bound.
- size_t [get_row](#) () const
Returns the identifier of the constraint associated with this bound. This identifier corresponds to the number of a row in the coefficient matrix associated with of a linear program instance.

Protected Attributes

- [CONSTRAINTTYPE](#) [_ctype](#)
The type of the constraint associated with this bound.
- double [_value](#)
The bound value.
- size_t [_row](#)
The identifier of the constraint associated with this bound.

16.2.1 Detailed Description

This class represents the type of a constraint (i.e., equality or inequality) and the value of its right-hand side: a real number.

Definition at line 57 of file bound.hpp.

16.2.2 Constructor & Destructor Documentation

16.2.2.1 Bound()

```
channel::Bound::Bound (
    const CONSTRAINTTYPE type,
    const double value,
    const size_t row ) [inline]
```

Creates an instance of this class.

Parameters

<i>type</i>	The type of the constraint associated with this bound.
<i>value</i>	The value of the bound.
<i>row</i>	The identifier of the constraint associated with this bound.

Definition at line 123 of file bound.hpp.

```
124 :
125     _ctype( type ) ,
126     _value( value ) ,
127     _row( row )
128 {
129 }
```

16.2.3 Member Function Documentation

16.2.3.1 get_row()

```
size_t channel::Bound::get_row ( ) const [inline]
```

Returns the identifier of the constraint associated with this bound. This identifier corresponds to the number of a row in the coefficient matrix associated with of a linear program instance.

Returns

The identifier of the constraint associated with this bound.

Definition at line 174 of file bound.hpp.

```
175 {
176     return _row ;
177 }
```

References `_row`.

16.2.3.2 `get_type()`

```
CONSTRAINTYPE channel::Bound::get_type ( ) const [inline]
```

Returns the type of the constraint associated with this bound.

Returns

The type of the constraint associated with this bound.

Definition at line 141 of file bound.hpp.

```
142     {  
143         return _ctype ;  
144     }
```

References `_ctype`.

16.2.3.3 `get_value()`

```
double channel::Bound::get_value ( ) const [inline]
```

Returns the value of this bound.

Returns

The value of this bound.

Definition at line 155 of file bound.hpp.

```
156     {  
157         return _value ;  
158     }
```

References `_value`.

The documentation for this class was generated from the following file:

- [bound.hpp](#)

16.3 `channel::Coefficient` Class Reference

This class represents a nonzero coefficient of an unknown of a constraint (inequality or equality) of a linear program instance.

```
#include <coefficient.hpp>
```

Public Member Functions

- [Coefficient](#) ()
Creates an instance of this class.
- [Coefficient](#) (const size_t row, const size_t col, double value)
Creates an instance of this class.
- size_t [get_row](#) () const
Returns the identifier of the constraint the coefficient is associated with. This identifier corresponds to the number of a row in the constraint coefficient matrix of a linear program.
- size_t [get_col](#) () const
Returns the identifier of the unknown multiplied by this coefficient in a constraint of a linear program instance. This identifier corresponds to the number of a column in the coefficient matrix of the linear program instance.
- double [get_value](#) () const
Returns the value of this coefficient.

Protected Attributes

- size_t [_row](#)
The identifier of the constraint this coefficient belongs to.
- size_t [_col](#)
The identifier of the unknown multiplied by this coefficient.
- double [_value](#)
The coefficient value.

16.3.1 Detailed Description

This class represents a nonzero coefficient of an unknown of a constraint (inequality or equality) of a linear program instance.

Definition at line 58 of file coefficient.hpp.

16.3.2 Constructor & Destructor Documentation

16.3.2.1 Coefficient()

```
channel::Coefficient::Coefficient (  
    const size_t row,  
    const size_t col,  
    double value ) [inline]
```

Creates an instance of this class.

Parameters

<i>row</i>	The identifier of the constraint this coefficient belongs to.
<i>col</i>	The identifier of the unknown multiplied by this coefficient.
<i>value</i>	The value of the coefficient.

Definition at line 107 of file coefficient.hpp.

```

108 :
109     _row( row ) ,
110     _col( col ) ,
111     _value( value )
112 {
113 }
```

16.3.3 Member Function Documentation**16.3.3.1 get_col()**

```
size_t channel::Coefficient::get_col ( ) const [inline]
```

Returns the identifier of the unknown multiplied by this coefficient in a constraint of a linear program instance. This identifier corresponds to the number of a column in the coefficient matrix of the linear program instance.

Returns

The identifier of the unknown multiplied by this coefficient in a constraint of a linear program instance.

Definition at line 148 of file coefficient.hpp.

```

149 {
150     return _col ;
151 }
```

References `_col`.

16.3.3.2 get_row()

```
size_t channel::Coefficient::get_row ( ) const [inline]
```

Returns the identifier of the constraint the coefficient is associated with. This identifier corresponds to the number of a row in the constraint coefficient matrix of a linear program.

Returns

The identifier of the constraint this coefficient is associated with.

Definition at line 129 of file coefficient.hpp.

```

130 {
131     return _row ;
132 }
```

References `_row`.

16.3.3.3 get_value()

```
double channel::Coefficient::get_value ( ) const [inline]
```

Returns the value of this coefficient.

Returns

The value of this coefficient.

Definition at line 162 of file coefficient.hpp.

```
163 {  
164     return _value ;  
165 }
```

References `_value`.

The documentation for this class was generated from the following file:

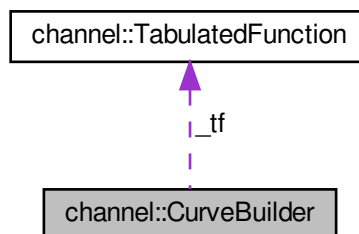
- [coefficient.hpp](#)

16.4 channel::CurveBuilder Class Reference

This class provides methods for threading a cubic b-spline curve through a planar channel delimited by a pair of polygonal chains.

```
#include <curvebuilder.hpp>
```

Collaboration diagram for channel::CurveBuilder:



Public Member Functions

- [CurveBuilder](#) (size_t np, size_t nc, bool closed, double *lx, double *ly, double *ux, double *uy)
Creates an instance of this class.
- [CurveBuilder](#) (const [CurveBuilder](#) &b)
Clones an instance of this class.
- bool [build](#) (int &error)
Solves the channel problem by solving a linear program.
- size_t [get_degree](#) () const
Returns the degree of the bspline curve.
- size_t [get_number_of_segments](#) () const
Returns the number of b-spline segments.
- size_t [get_number_of_csegments](#) () const
Returns the number of c-segments of the channel.
- bool [is_curve_closed](#) () const
Returns the logic value true if the b-spline curve is closed, and the logic value false otherwise.
- size_t [get_number_of_control_points](#) () const
Returns the number of control points of the b-spline.
- size_t [get_number_of_constraints](#) () const
Returns the number of constraints of the instance of the linear program corresponding to the channel problem solved by this class.
- double [get_control_value](#) (const size_t i, const size_t v) const
Returns the v-th coordinate of the i-th control point of the b-spline curve threaded into the channel.
- size_t [get_number_of_coefficients_in_the_ith_constraint](#) (const size_t i) const
Returns the number of coefficients of the i-th constraint of the instance of the linear program.
- size_t [get_coefficient_identifier](#) (const size_t i, const size_t j) const
Returns the index of the column that corresponds to the j-th coefficient of the i-th constraint in the matrix associated with the linear program (LP) instance.
- double [get_coefficient_value](#) (const size_t i, const size_t j) const
Returns the (i, j) entry of the matrix associated with the instance of the linear program.
- double [get_bound_of_ith_constraint](#) (const size_t i) const
Returns the real value on the right-hand side of the equality or inequality corresponding to the i-th constraint.
- bool [is_equality](#) (const size_t i) const
Returns the logic value true if the type of the i-th constraint is equality; otherwise, returns the logic value false.
- bool [is_greater_than_or_equal_to](#) (const size_t i) const
Returns the logic value true if the i-th constraint is an inequality of the type greater than or equal to; otherwise, returns the logic value false.
- bool [is_less_than_or_equal_to](#) (const size_t i) const
Returns the logic value true if the i-th constraint is an inequality of the type less than or equal to; otherwise, returns the logic value false.
- double [get_lower_bound_on_second_difference_value](#) (const size_t p, const size_t i, const size_t v) const
Returns the lower bound (found by the LP solver) on the v-th coordinate of the i-th second difference of the i-th curve segment of the b-spline curve threaded into the channel.
- double [get_upper_bound_on_second_difference_value](#) (const size_t p, const size_t i, const size_t v) const
Returns the upper bound (found by the LP solver) on the v-th coordinate of the i-th second difference vector of the p-th curve segment of the b-spline curve threaded into the channel.
- double [minimum_value](#) () const
Returns the optimal (minimum) value of the objective function of the instance of the channel problem as found by the LP solver.
- std::string [get_solver_error_message](#) (int error)
Returns the error message of the GLPK solver associated with a given error code.

Private Member Functions

- void [compute_min_max_constraints](#) (size_t &eqline)
Computes the equations defining the min-max constraints.
- void [compute_correspondence_constraints](#) (size_t &eqline)
Computes the equations defining the constraints on the location of the endpoints of the b-spline curve threaded into the channel.
- void [compute_sleeve_corners_in_channel_constraints](#) (size_t &eqline)
Computes the equations defining the constraints that ensure that the breakpoints of the sleeves are inside the channel.
- void [compute_channel_corners_outside_sleeve_constraints](#) (size_t &eqline)
Computes the equations defining the constraints that ensure that the corners of the channel are located on the boundary or outside the sleeve.
- void [compute_sleeve_inside_csegment_constraints](#) (size_t &eqline)
Computes the equations defining the constraints that ensure the bspline segments associated with a c-segment remain inside it.
- void [compute_normal_to_lower_envelope](#) (const size_t s, double &nx, double &ny) const
Computes an outward normal to the s-th line segment of the lower envelope of the channel.
- void [compute_normal_to_upper_envelope](#) (const size_t s, double &nx, double &ny) const
Computes an outward normal to the s-th line segment of the upper envelope of the channel.
- void [compute_normal_to_csection](#) (const size_t s, double &nx, double &ny) const
Computes a normal to the s-th c-section of the channel.
- size_t [compute_control_value_column_index](#) (const size_t p, const size_t i, const size_t v) const
Computes the index of the linear program matrix column corresponding to the x- or y-coordinate of the i-th control point of the p-th segment of the b-spline to be threaded into the channel.
- void [insert_coefficient](#) (const size_t eqline, const size_t index, const double value)
Assigns a value to the coefficient of an unknown of a given constraint of the linear program (LP). The unknown is identified by its corresponding column index in the associated matrix of the LP.
- void [insert_bound](#) (const size_t eqline, const [Bound::CONSTRAINTTYPE](#) type, const double value)
Assigns a real value to the right-hand side of a constraint (equality or inequality) of an instance of the linear program associated with the channel problem.
- size_t [compute_second_difference_column_index](#) (const size_t p, const size_t i, const size_t l, const size_t v) const
Computes the index of the linear program matrix column corresponding to the x- or y-coordinate of the l-th bound of the i-th second difference of the p-th segment of the b-spline to be threaded into the channel.
- size_t [compute_index_of_endpoint_barycentric_coordinate](#) (const size_t i) const
Computes the index of the linear program matrix column corresponding to the barycentric coordinate defining the i-th endpoint of the b-spline.
- size_t [compute_index_of_corner_barycentric_coordinate](#) (const size_t i) const
Computes the index of the linear program matrix column corresponding to the barycentric coordinate associated with a channel corner.
- void [insert_min_max_constraints](#) (const size_t eqline, const size_t lo, const size_t up, const size_t b0, const size_t b1, const size_t b2)
Inserts the coefficients of the equations defining the three min-max constraints into the matrix associated with the linear program (LP), and sets the right-hand side of the constraints as well.
- void [insert_extreme_point_correspondence_constraint](#) (const size_t eqline, const std::vector< size_t > &col, const std::vector< double > &val, const double rhs)
Inserts into the linear program (LP) matrix the coefficients of the unknowns and the right-hand side value of a constraint corresponding to the location of the starting or final point of the b-spline curve.
- void [insert_periodic_correspondence_constraints](#) (const size_t eqline, const std::vector< size_t > &strx, const std::vector< size_t > &stry, const std::vector< size_t > &endx, const std::vector< size_t > &endy)

Inserts into the linear program (LP) matrix the coefficients of the unknowns and the right-hand side values of the constraints that ensure that the first three control points are the same as the last three control points (in this order).

- void [insert_nonlinear_terms_of_epiece_point_lower_bound](#) (const size_t eqline, const double s, const size_t c, const std::vector< std::vector< std::vector< size_t > > > &sd, const std::vector< std::vector< double > > &nl, const std::vector< std::vector< double > > &nu)

Inserts the coefficients of the second difference terms of the equation defining lower bounds for the e-piece points into the matrix associated with an instance of the Linear Program (LP). The terms belong to the constraint that forces the e-piece points to be inside a certain c-section of the channel.

- void [insert_nonlinear_terms_of_epiece_point_lower_bound](#) (const size_t eqline, const double s, const size_t c, const std::vector< std::vector< std::vector< size_t > > > &sd, const std::vector< std::vector< double > > &ncsec)

Inserts the coefficients of the second difference terms of the equation defining lower bounds for the e-piece points into the matrix associated with an instance of the Linear Program (LP). The terms belong to the constraint that forces one e-piece point to be on the right or left side of a channel c-section.

- void [insert_nonlinear_terms_of_epiece_point_upper_bound](#) (const size_t eqline, const double s, const size_t c, const std::vector< std::vector< std::vector< size_t > > > &sd, const std::vector< std::vector< double > > &nl, const std::vector< std::vector< double > > &nu)

Inserts into the matrix associated with the Linear Program (LP) the coefficients of the lower and upper bounds of the second difference terms of the equation defining upper bounds for the e-piece points. These terms occur in the constraint that keep the sleeve inside a certain c-section of the channel.

- void [insert_nonlinear_terms_of_epiece_point_upper_bound](#) (const size_t eqline, const double s, const size_t c, const std::vector< std::vector< std::vector< size_t > > > &sd, const std::vector< std::vector< double > > &ncsec)

Inserts the coefficients of the second difference terms of the equation defining upper bounds for the e-piece points into the matrix associated with an instance of the Linear Program (LP). The terms belong to the constraint that forces one e-piece point to be on the right or left side of a channel c-section.

- void [insert_linear_terms_of_epiece_point_bounds](#) (const size_t eqline, const double s, const double t, const size_t p, const size_t c, const std::vector< std::vector< size_t > > &cp, const std::vector< std::vector< double > > &nl, const std::vector< std::vector< double > > &nu)

Inserts the coefficients of the linear terms of the equation defining lower and upper bounds for the e-piece points into the matrix associated with the Linear Program (LP). These terms occur in the constraint that enforces an e-piece point to stay inside channel.

- void [insert_linear_terms_of_epiece_point_bounds](#) (const size_t eqline, const double s, const double t, const size_t p, const size_t c, const std::vector< std::vector< size_t > > &cp, const std::vector< std::vector< double > > &ncsec)

Inserts the coefficients of the linear terms of the equation defining lower and upper bounds for the e-piece points into the matrix associated with the Linear Program (LP). These terms occur in the constraint that enforces an e-piece point to stay either on the right or on left side of a channel c-section.

- void [insert_rhs_of_sleeve_corners_in_channel_constraints](#) (const size_t eqline, const size_t c, const std::vector< std::vector< double > > &nl, const std::vector< std::vector< double > > &nu)

Inserts into the matrix associated with the Linear Program (LP) the right-hand side values of the constraints that enforce a sleeve point to stay inside a c-segment of the channel. The type of each constraint (equality or inequality: ==, >= or <=) is also set here.

- void [insert_rhs_of_sleeve_inside_csegment_constraints](#) (const size_t eqline, const size_t c, const std::vector< std::vector< double > > &ncsec)

Inserts into the matrix associated with the Linear Program (LP) the right-hand side values of the constraints that enforce one e-piece breakpoint to stay on the right side of a c-section of the channel, and another e-piece breakpoint to stay on the left side of the same c-section.

- void [evaluate_bounding_polynomial](#) (const size_t j, const double t, double &lower, double &upper)

Obtains a lower bound and an upper bound for the value of a precomputed, bounding polynomial at a given parameter value.

- void [insert_csegment_constraint](#) (const size_t eqline, const double lower, const double upper, const size_t sdlo, const size_t sdup, const double normal)

Inserts the coefficients of the lower and upper bounds of a constraint second difference term into the matrix associated with an instance of the linear program (LP). The term belongs to the equation defining the upper (or lower) bound of a point of an e-piece. The constraint ensures that the point lies on a specific side of the oriented supporting line of one of the four line segments delimiting a c-segment of the channel.

- void [insert_csegment_constraint](#) (const size_t eqline, const double c0, const double c1, const double c2, const double c3, const size_t b0, const size_t b1, const size_t b2, const size_t b3, const double normal)

Inserts the coefficients of the linear terms of the upper and lower bounds of an e-piece point equation into the matrix associated with an instance of the linear program (LP). The constraint ensures that the point of the e-piece lies inside a c-segment of the channel.

- void [insert_csegment_constraint](#) (const size_t eqline, const double c0, const double c1, const double c2, const double c3, const double c4, const size_t b0, const size_t b1, const size_t b2, const size_t b3, const size_t b4, const double normal)

Inserts the coefficients of the linear terms of the upper and lower bounds of an e-piece point equation into the matrix associated with an instance of the linear program (LP). This point belongs to an e-piece segment whose endpoints bound b-spline curve points in two distinct, but consecutive curve segments. The constraint ensures that the e-piece point lies inside a c-segment of the channel.

- int [solve_lp](#) (const size_t rows, const size_t cols)

Solves the linear program corresponding to the channel problem.

- void [set_up_lp_constraints](#) (glp_prob *lp) const

Assemble the matrix of constraints of the linear program, and define the type (equality or inequality) and bounds on the constraints.

- void [set_up_structural_variables](#) (glp_prob *lp) const

Define lower and/or upper bounds on the structural variables of the linear program corresponding to the channel problem.

- void [set_up_objective_function](#) (glp_prob *lp) const

Define the objective function of the linear program corresponding to the channel problem, which is a minimization problem.

- void [get_lp_solver_result_information](#) (glp_prob *lp)

Obtain the optimal values found by the LP solver for the structural values of the linear programming corresponding to the channel problem.

Private Attributes

- size_t [_np](#)

The number of b-spline segments per channel segment.

- size_t [_nc](#)

The number of segments of the channel.

- bool [_closed](#)

A flag to indicate whether the channel is closed.

- std::vector< double > [_lxcoords](#)

X coordinates of the lower polygonal chain of the channel.

- std::vector< double > [_lycoords](#)

Y coordinates of the lower polygonal chain of the channel.

- std::vector< double > [_uxcoords](#)

X coordinates of the upper polygonal chain of the channel.

- std::vector< double > [_uycoords](#)

Y coordinates of the upper polygonal chain of the channel.

- [TabulatedFunction](#) * [_tf](#)

A pointer to the lower and upper α functions.

- std::vector< std::vector< [Coefficient](#) > > [_coefficients](#)

Coefficients of the constraints of the linear program.

- `std::vector< Bound > _bounds`

Type of the constraints and their bounds.

- `std::vector< double > _ctrlpts`

X and Y coordinates of the control points of the resulting b-spline.

- `std::vector< double > _secdiff`

Lower and upper bounds on the second difference values.

- `double _ofvalue`

Optimal value (i.e., minimum) of the objective function.

16.4.1 Detailed Description

This class provides methods for threading a cubic b-spline curve through a planar channel delimited by a pair of polygonal chains.

Attention

This class is based on a particular case (i.e., planar and cubic curves) of the method described by Myles & Peters in

A. Myles and J. Peters, Threading splines through 3d channels Computer-Aided Design, 37(2), 139-148, 2005.

Definition at line 79 of file curvebuilder.hpp.

16.4.2 Constructor & Destructor Documentation

16.4.2.1 CurveBuilder() [1/2]

```
channel::CurveBuilder::CurveBuilder (
    size_t np,
    size_t nc,
    bool closed,
    double * lx,
    double * ly,
    double * ux,
    double * uy )
```

Creates an instance of this class.

Parameters

<i>np</i>	The number of b-spline segments.
<i>nc</i>	The number of c-segments of the channel.
<i>closed</i>	A flag to indicate whether the channel is closed.
<i>lx</i>	A pointer to an array with the x-coordinates of the lower envelope of the channel.
<i>ly</i>	A pointer to an array with the y-coordinates of the lower envelope of the channel.
<i>ux</i>	A pointer to an array with the x-coordinates of the upper envelope of the channel.
<i>uy</i>	A pointer to an array with the y-coordinates of the upper envelope of the channel.

Definition at line 80 of file curvebuilder.cpp.

```

89 :
90     _np( np ) ,
91     _nc( nc ) ,
92     _closed ( closed )
93 {
94     if ( _closed ) {
95         if ( _np < 4 ) {
96             std::stringstream ss( std::stringstream::in | std::stringstream::out ) ;
97             ss << "The number of curve segments must be at least 4 for a closed curve" ;
98             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
99         }
100         if ( _nc < 3 ) {
101             std::stringstream ss( std::stringstream::in | std::stringstream::out ) ;
102             ss << "The number of segments of a closed channel must be at least 3" ;
103             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
104         }
105     }
106     else {
107         if ( _np < 1 ) {
108             std::stringstream ss( std::stringstream::in | std::stringstream::out ) ;
109             ss << "The number of curve segments must be at least 1" ;
110             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
111         }
112         if ( _nc < 1 ) {
113             std::stringstream ss( std::stringstream::in | std::stringstream::out ) ;
114             ss << "The number of segments of an open channel must be at least 1" ;
115             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
116         }
117     }
118
119     size_t nn = ( _closed ) ? _nc : ( _nc + 1 ) ;
120
121     _lxcoords.resize( nn ) ;
122     _lycoords.resize( nn ) ;
123     _uxcoords.resize( nn ) ;
124     _uycoords.resize( nn ) ;
125
126     for ( unsigned i = 0 ; i < nn ; i++ ) {
127         _lxcoords[ i ] = lx[ i ] ;
128         _lycoords[ i ] = ly[ i ] ;
129         _uxcoords[ i ] = ux[ i ] ;
130         _uycoords[ i ] = uy[ i ] ;
131     }
132
133     _tf = new a3() ;
134     _ofvalue = DBL_MAX ;
135
136     return ;
137 }

```

References `_closed`, `_lxcoords`, `_lycoords`, `_nc`, `_np`, `_ofvalue`, `_tf`, `_uxcoords`, and `_uycoords`.

16.4.2.2 CurveBuilder() [2/2]

```

channel::CurveBuilder::CurveBuilder (
    const CurveBuilder & b )

```

Clones an instance of this class.

Parameters

<i>b</i>	A reference to another instance of this class.
----------	--

Definition at line 149 of file curvebuilder.cpp.

```

150 :
151     _np( b._np ) ,
152     _nc( b._nc ) ,
153     _closed( b._closed ) ,
154     _lxcoords( b._lxcoords ) ,
155     _lycoords( b._lycoords ) ,
156     _uxcoords( b._uxcoords ) ,
157     _uycoords( b._uycoords ) ,
158     _tf( b._tf ) ,
159     _coefficients( b._coefficients ) ,
160     _bounds( b._bounds ) ,
161     _ctrlpts( b._ctrlpts ) ,
162     _secdiff( b._secdiff ) ,
163     _ofvalue( b._ofvalue )
164 {
165 }

```

16.4.3 Member Function Documentation

16.4.3.1 build()

```

bool channel::CurveBuilder::build (
    int & error )

```

Solves the channel problem by solving a linear program.

Parameters

<i>error</i>	Code returned by the LP solver whenever a solution could not be found. If a solution is found, this parameter is ignored.
--------------	---

Returns

The logic value true if the LP solver is able to find an optimal solution for the channel problem; otherwise, the logic value false is returned.

Definition at line 182 of file curvebuilder.cpp.

```

183 {
184     // Compute the number of linear constraints (i.e., the number of
185     // rows of the matrix) of the linear program whose solution yields
186     // the curve.
187     size_t rows = (
188         ( 6 * ( _np + 1 ) )           // min-max
189         + ( ( _closed ) ? 8 : 4 )     // correspondence
190         + ( 2 * ( _nc - 1 ) )         // channel corners
191         + ( ( 3 * 4 * _np ) - 4 )     // sleeve corners
192         + ( 4 * ( _nc - 1 ) )         // sleeve in csegments
193     ) ;
194
195     // Compute the unknowns (i.e., the number of columns of the
196     // matrix) of the linear program whose solution yields the
197     // b-spline curve.
198     size_t cols = ( 6 * _np ) + 10 + ( ( _closed ) ? 1 : 2 ) + ( _nc - 1 ) ;
199
200     //
201     // Allocate memory for the array of coefficients and bounds.
202     //
203     _coefficients.resize( rows ) ;

```



```

204     _bounds.resize( rows ) ;
205
206     //
207     // Initialize the equation counter.
208     //
209     size_t eqline = 0 ;
210
211     //
212     // Compute the min-max constraints.
213     //
214     compute_min_max_constraints( eqline ) ;
215
216     //
217     // Compute the correspondence constraints.
218     //
219     compute_correspondence_constraints( eqline ) ;
220
221     //
222     // Compute channel corners outside sleeve constraints.
223     //
224     compute_channel_corners_outside_sleeve_constraints( eqline ) ;
225
226     //
227     // Compute the sleeve corners in channel constraints.
228     //
229     compute_sleeve_corners_in_channel_constraints( eqline ) ;
230
231     //
232     // Compute the sleeve inside csegment constraints.
233     //
234     compute_sleeve_inside_csegment_constraints( eqline ) ;
235
236     //
237     // Solve the LP and get the solution.
238     //
239     error = solve_lp( rows , cols ) ;
240
241     return ( error == 0 ) ;
242 }

```

References `_bounds`, `_closed`, `_coefficients`, `_nc`, `_np`, `compute_channel_corners_outside_sleeve_constraints()`, `compute_correspondence_constraints()`, `compute_min_max_constraints()`, `compute_sleeve_corners_in_channel_constraints()`, `compute_sleeve_inside_csegment_constraints()`, and `solve_lp()`.

16.4.3.2 compute_channel_corners_outside_sleeve_constraints()

```

void channel::CurveBuilder::compute_channel_corners_outside_sleeve_constraints (
    size_t & eqline ) [private]

```

Computes the equations defining the constraints that ensure that the corners of the channel are located on the boundary or outside the sleeve.

Parameters

<i>eqline</i>	A reference to the counter of equations.
---------------	--

Definition at line 717 of file `curvebuilder.cpp`.

```

718 {
719     //
720     // This restriction applies to channels with at least 3 c-sections
721     // only.
722     //
723     if ( _nc < 2 ) {
724         return ;

```

```

725     }
726
727     // For each inner corner of the given channel, compute a
728     // constraint that ensures that the channel corner is outside the
729     // sleeve.
730     const double NpOverNc = _np / double( _nc ) ;
731     const double onesixth = 1 / double( 6 ) ;
732
733     for ( size_t c = 1 ; c < _nc ; c++ ) {
734         //
735         // Find the parameter \e t corresponding to the \e c corner.
736         //
737         double t = ( c * NpOverNc ) + 3 ;
738
739         //
740         // Find the curve segment \e p containing point at parameter \e
741         // t.
742         size_t p = ( size_t ) floor( t - 3 ) ;
743
744         // Compute the column indices of the linear program matrix
745         // corresponding to the four control points defining the p-th
746         // segment of the b-spline curve.
747
748         std::vector< std::vector< size_t > > cp( 4 , std::vector< size_t >( 2 , 0 ) ) ;
749
750         for ( size_t i = 0 ; i < 4 ; i++ ) {
751             for ( size_t j = 0 ; j < 2 ; j++ ) {
752                 cp[ i ][ j ] = compute_control_value_column_index( p , i , j ) ;
753             }
754         }
755
756         //
757         // Compute the coefficients of the control points.
758         //
759         double s = t - floor( t ) ;
760         std::vector< double > coeffs( 4 ) ;
761
762         coeffs[ 0 ] = onesixth * ( 1 + s * ( -3 + s * ( 3 - s ) ) ) ;
763         coeffs[ 1 ] = onesixth * ( 4 + s * s * ( -6 + 3 * s ) ) ;
764         coeffs[ 2 ] = onesixth * ( 1 + s * ( 3 + s * ( 3 - 3 * s ) ) ) ;
765         coeffs[ 3 ] = onesixth * ( s * s * s ) ;
766
767         // Get the LP matrix column index corresponding to the
768         // barycentric coordinate associated with the c-th corner of the
769         // channel.
770         size_t k = compute_index_of_corner_barycentric_coordinate( c ) ;
771
772         //
773         // Insert the constraints into the LP program.
774         //
775         for ( size_t i = 0 ; i < 4 ; i++ ) {
776             insert_coefficient( eqline , cp[ i ][ 0 ] , coeffs[ i ] ) ;
777             insert_coefficient( eqline + 1 , cp[ i ][ 1 ] , coeffs[ i ] ) ;
778         }
779
780         insert_coefficient( eqline , k , ( _lxcoords[ c ] - _uxcoords[ c ] ) ) ;
781         insert_coefficient( eqline + 1 , k , ( _lycoords[ c ] - _uycoords[ c ] ) ) ;
782
783         insert_bound( eqline , Bound::EQT , _lxcoords[ c ] ) ;
784         insert_bound( eqline + 1 , Bound::EQT , _lycoords[ c ] ) ;
785
786         //
787         // Increment equation counter.
788         //
789         eqline += 2 ;
790     }
791
792     return ;
793 }

```

References `_lxcoords`, `_lycoords`, `_nc`, `_np`, `_uxcoords`, `_uycoords`, `compute_control_value_column_index()`, `compute_index_of_corner_barycentric_coordinate()`, `insert_bound()`, and `insert_coefficient()`.

Referenced by `build()`.

16.4.3.3 compute_control_value_column_index()

```
size_t channel::CurveBuilder::compute_control_value_column_index (
    const size_t p,
    const size_t i,
    const size_t v ) const [private]
```

Computes the index of the linear program matrix column corresponding to the x- or y-coordinate of the i-th control point of the p-th segment of the b-spline to be threaded into the channel.

Parameters

<i>p</i>	Index of the b-spline segment.
<i>i</i>	Index of a control point of the p-th b-spline segment.
<i>v</i>	Index of the x- or y-coordinate of the control point.

Returns

The index of the linear program matrix column corresponding to the x- or y-coordinate of the i-th control point of the p-th segment of the b-spline to be threaded into the channel.

Definition at line 1202 of file curvebuilder.cpp.

```
1208 {
1209 #ifdef DEBUGMODE
1210     assert( p < _np ) ;
1211     assert( i <= 3 ) ;
1212     assert( v <= 1 ) ;
1213 #endif
1214
1215     return 2 * ( p + i ) + v ;
1216 }
```

References `_np`.

Referenced by `compute_channel_corners_outside_sleeve_constraints()`, `compute_correspondence_constraints()`, `compute_min_max_constraints()`, `compute_sleeve_corners_in_channel_constraints()`, `compute_sleeve_inside_csegment_constraints()`, `get_lp_solver_result_information()`, and `set_up_structural_variables()`.

16.4.3.4 compute_correspondence_constraints()

```
void channel::CurveBuilder::compute_correspondence_constraints (
    size_t & eqline ) [private]
```

Computes the equations defining the constraints on the location of the endpoints of the b-spline curve threaded into the channel.

Parameters

<i>eqline</i>	A reference to the counter of equations.
---------------	--

Definition at line 393 of file curvebuilder.cpp.

```

394 {
395     // Get the column index of the x- and y-coordinates of the first
396     // three control points of the b-spline to be threaded into the
397     // channel.
398
399     std::vector< size_t > strx( 4 ) ;
400
401     strx[ 0 ] = compute_control_value_column_index( 0 , 0 , 0 ) ;
402     strx[ 1 ] = compute_control_value_column_index( 0 , 1 , 0 ) ;
403     strx[ 2 ] = compute_control_value_column_index( 0 , 2 , 0 ) ;
404
405     // Get the column index of the barycentric coordinate defining
406     // the first endpoint of the b-spline with respect to the first
407     // channel points.
408
409     strx[ 3 ] = compute_index_of_endpoint_barycentric_coordinate( 0 ) ;
410
411     //
412     // Get the coefficients of the unknowns of the constraint.
413     //
414     std::vector< double > vals( 4 ) ;
415
416     vals[ 0 ] = double( 1 ) / double( 6 ) ;
417     vals[ 1 ] = double( 2 ) / double( 3 ) ;
418     vals[ 2 ] = vals[ 0 ] ;
419     vals[ 3 ] = _lxcoords[ 0 ] - _uxcoords[ 0 ] ;
420
421     //
422     // Constraint corresponding to first Cartesian coordinate.
423     //
424     insert_extreme_point_correspondence_constraint(
425         eqline ,
426         strx ,
427         vals ,
428         _lxcoords[ 0 ]
429     ) ;
430
431     ++eqline ; // increment the equation counter.
432
433     //
434     // Constraint corresponding to second Cartesian coordinate.
435     //
436
437     std::vector< size_t > stry( 4 ) ;
438
439     stry[ 0 ] = compute_control_value_column_index( 0 , 0 , 1 ) ;
440     stry[ 1 ] = compute_control_value_column_index( 0 , 1 , 1 ) ;
441     stry[ 2 ] = compute_control_value_column_index( 0 , 2 , 1 ) ;
442     stry[ 3 ] = strx[ 3 ] ;
443
444     vals[ 3 ] = _lycoords[ 0 ] - _uycoords[ 0 ] ;
445
446     insert_extreme_point_correspondence_constraint(
447         eqline ,
448         stry ,
449         vals ,
450         _lycoords[ 0 ]
451     ) ;
452
453     ++eqline ; // increment the equation counter.
454
455     // -----
456     //
457     // If the curve is closed, then the last three control points must
458     // match the first three control points. Otherwise, we must fix
459     // the position of the final of the curve, which differs from the
460     // starting one.
461     //
462     // -----
463
464     // Get the column index of the x- and y-coordinates of the last
465     // three control points of the b-spline to be threaded into the
466     // channel.
467
468     std::vector< size_t > endx( 4 ) ;
469
470     endx[ 0 ] = compute_control_value_column_index( _np - 1 , 1 , 0 ) ;
471     endx[ 1 ] = compute_control_value_column_index( _np - 1 , 2 , 0 ) ;
472     endx[ 2 ] = compute_control_value_column_index( _np - 1 , 3 , 0 ) ;
473

```

```

474     std::vector< size_t > endy( 4 ) ;
475
476     endy[ 0 ] = compute_control_value_column_index( _np - 1 , 1 , 1 ) ;
477     endy[ 1 ] = compute_control_value_column_index( _np - 1 , 2 , 1 ) ;
478     endy[ 2 ] = compute_control_value_column_index( _np - 1 , 3 , 1 ) ;
479
480     if ( _closed ) {
481         //
482         // Compute the equations that match the first three and last
483         // three control points of the b-spline: last is equal to third,
484         // ...
485         //
486         insert_periodic_correspondence_constraints(
487             eqline ,
488             strx ,
489             stry ,
490             endx ,
491             endy
492         ) ;
493
494         eqline += 6 ; // increment the equation counter.
495     }
496     else {
497         //
498         // Compute the equations determining the b-spline final point.
499         //
500         // Get the column index of the barycentric coordinate of the
501         // final point of the b-spline with respect to the final points
502         // of the channel.
503         endx[ 3 ] = compute_index_of_endpoint_barycentric_coordinate( 1 ) ;
504         vals[ 3 ] = _lxcoords[ _nc ] - _uxcoords[ _nc ] ;
505
506         //
507         // Constraint corresponding to first Cartesian coordinate.
508         //
509         insert_extreme_point_correspondence_constraint(
510             eqline ,
511             endx ,
512             vals ,
513             _lxcoords[ _nc ]
514         ) ;
515
516         endy[ 3 ] = endx[ 3 ] ;
517         vals[ 3 ] = _lycoords[ _nc ] - _uycoords[ _nc ] ;
518
519         ++eqline ; // increment the equation counter.
520
521         insert_extreme_point_correspondence_constraint(
522             eqline ,
523             endy ,
524             vals ,
525             _lycoords[ _nc ]
526         ) ;
527
528         ++eqline ; // increment the equation counter.
529     }
530
531     return ;
532 }
533
534 }
535

```

References `_closed`, `_lxcoords`, `_lycoords`, `_nc`, `_np`, `_uxcoords`, `_uycoords`, `compute_control_value_column_index()`, `compute_index_of_endpoint_barycentric_coordinate()`, `insert_extreme_point_correspondence_constraint()`, and `insert_periodic_correspondence_constraints()`.

Referenced by `build()`.

16.4.3.5 compute_index_of_corner_barycentric_coordinate()

```

size_t channel::CurveBuilder::compute_index_of_corner_barycentric_coordinate (
    const size_t i ) const [private]

```

Computes the index of the linear program matrix column corresponding to the barycentric coordinate associated with a channel corner.

Parameters

<i>i</i>	Index of a channel corner.
----------	----------------------------

Returns

The index of the linear program matrix column corresponding to the barycentric coordinate associated with a channel corner.

Definition at line 1312 of file curvebuilder.cpp.

```

1313 {
1314     #ifdef DEBUGMODE
1315         assert( i > 0 );
1316         assert( i < _nc );
1317     #endif
1318
1319     size_t offset = ( 6 * _np ) + 10 + ( ( _closed ) ? 0 : 1 );
1320
1321     return offset + i ;
1322 }
1323
```

References `_closed`, `_nc`, and `_np`.

Referenced by `compute_channel_corners_outside_sleeve_constraints()`, and `set_up_structural_variables()`.

16.4.3.6 compute_index_of_endpoint_barycentric_coordinate()

```

size_t channel::CurveBuilder::compute_index_of_endpoint_barycentric_coordinate (
    const size_t i ) const [private]

```

Computes the index of the linear program matrix column corresponding to the barycentric coordinate defining the *i*-th endpoint of the b-spline.

Parameters

<i>i</i>	Index of the <i>i</i> -th barycentric coordinate.
----------	---

Returns

The index of the linear program matrix column corresponding to the barycentric coordinate defining the *i*-th endpoint of the b-spline.

Definition at line 1280 of file curvebuilder.cpp.

```

1281 {
1282     #ifdef DEBUGMODE
1283         if ( _closed ) {
1284             assert( i == 0 );

```

```

1285     }
1286     else {
1287         assert( i <= 1 ) ;
1288     }
1289 #endif
1290
1291     size_t offset = ( 6 * _np ) + 10 ;
1292
1293     return offset + i ;
1294 }

```

References `_closed`, and `_np`.

Referenced by `compute_correspondence_constraints()`, and `set_up_structural_variables()`.

16.4.3.7 compute_min_max_constraints()

```

void channel::CurveBuilder::compute_min_max_constraints (
    size_t & eqline ) [private]

```

Computes the equations defining the min-max constraints.

Parameters

<i>eqline</i>	A reference to the counter of equations.
---------------	--

Definition at line 261 of file `curvebuilder.cpp`.

```

262 {
263     //
264     // Obtain the min-max constraints for each second difference.
265     //
266
267     for ( size_t j = 1 ; j < 3 ; j++ ) {
268         for ( size_t v = 0 ; v < 2 ; v++ ) {
269             // Get the column indices of the lower bound and of the
270             // upper bound of the v-th coordinate of the j-th second
271             // difference.
272             size_t jl = compute_second_difference_column_index(
273                                     0 ,
274                                     j ,
275                                     0 ,
276                                     v
277                                 ) ;
278
279             size_t ju = compute_second_difference_column_index(
280                                     0 ,
281                                     j ,
282                                     1 ,
283                                     v
284                                 ) ;
285
286             // Get the column indices of the v-th coordinates that define
287             // the j-th second difference of the p-th curve segment.
288             size_t c1 = compute_control_value_column_index(
289                                     0 ,
290                                     j - 1 ,
291                                     v
292                                 ) ;
293
294             size_t c2 = compute_control_value_column_index(
295                                     0 ,
296                                     j ,
297                                     v
298                                 ) ;
299
300             size_t c3 = compute_control_value_column_index(

```

```

299             0 ,
300             j + 1 ,
301             v
302         ) ;
303
304     //
305     // Set the nonzero coefficients of the next three equations.
306     //
307     insert_min_max_constraints(
308         eqline ,
309         j1 ,
310         ju ,
311         c1 ,
312         c2 ,
313         c3
314     ) ;
315
316     //
317     // Increment equation counter
318     //
319     eqline += 3 ;
320 }
321 }
322
323 for ( size_t p = 1 ; p < _np ; p++ ) {
324     for ( size_t v = 0 ; v < 2 ; v++ ) {
325         // Get the column indices of the lower bound and of the upper
326         // bound of the v-th coordinate of the 2nd second difference.
327         size_t j1 = compute_second_difference_column_index(
328             p ,
329             2 ,
330             0 ,
331             v
332         ) ;
333
334         size_t ju = compute_second_difference_column_index(
335             p ,
336             2 ,
337             1 ,
338             v
339         ) ;
340
341         // Get the column indices of the v-th coordinates that define
342         // the i-th second difference of the p-th curve segment.
343         size_t c1 = compute_control_value_column_index(
344             p ,
345             1 ,
346             v
347         ) ;
348         size_t c2 = compute_control_value_column_index(
349             p ,
350             2 ,
351             v
352         ) ;
353         size_t c3 = compute_control_value_column_index(
354             p ,
355             3 ,
356             v
357         ) ;
358
359         //
360         // Set the nonzero coefficients of the next three equations.
361         //
362         insert_min_max_constraints(
363             eqline ,
364             j1 ,
365             ju ,
366             c1 ,
367             c2 ,
368             c3
369         ) ;
370
371         //
372         // Increment equation counter
373         //
374         eqline += 3 ;
375     }
376 }
377
378 return ;
379 }

```


References `_np`, `compute_control_value_column_index()`, `compute_second_difference_column_index()`, and `insert_min_max_constraints()`.

Referenced by `build()`.

16.4.3.8 compute_normal_to_csection()

```
void channel::CurveBuilder::compute_normal_to_csection (
    const size_t s,
    double & nx,
    double & ny ) const [private]
```

Computes a normal to the s -th c-section of the channel.

Parameters

<i>s</i>	Index of a c-section of the channel.
<i>nx</i>	A reference to the first Cartesian coordinate of the normal.
<i>ny</i>	A reference to the Second Cartesian coordinate of the normal.

Definition at line 1163 of file `curvebuilder.cpp`.

```
1169 {
1170     #ifdef DEBUGMODE
1171         assert( s <= _nc ) ;
1172     #endif
1173
1174     size_t t = ( _closed ) ? ( s % _nc ) : s ;
1175
1176     nx = _lycoords[ t ] - _uycoords[ t ] ;
1177     ny = _uxcoords[ t ] - _lxcoords[ t ] ;
1178
1179     return ;
1180 }
```

References `_closed`, `_lxcoords`, `_lycoords`, `_nc`, `_uxcoords`, and `_uycoords`.

Referenced by `compute_sleeve_inside_csegment_constraints()`.

16.4.3.9 compute_normal_to_lower_envelope()

```
void channel::CurveBuilder::compute_normal_to_lower_envelope (
    const size_t s,
    double & nx,
    double & ny ) const [private]
```

Computes an outward normal to the s -th line segment of the lower envelope of the channel.

Parameters

<i>s</i>	Index of a line segment of the lower channel envelope.
<i>nx</i>	A reference to the first Cartesian coordinate of the normal.
<i>ny</i>	A reference to the second Cartesian coordinate of the normal.

Definition at line 1087 of file curvebuilder.cpp.

```

1093 {
1094 #ifdef DEBUGMODE
1095     assert( s < _nc );
1096 #endif
1097
1098     size_t t = s + 1 ;
1099
1100     if ( _closed ) {
1101         t %= _nc ;
1102     }
1103
1104     nx = _lycoords[ s ] - _lycoords[ t ] ;
1105     ny = _lxcoords[ t ] - _lxcoords[ s ] ;
1106
1107     return ;
1108 }
```

References `_closed`, `_lxcoords`, `_lycoords`, and `_nc`.

Referenced by `compute_sleeve_corners_in_channel_constraints()`.

16.4.3.10 compute_normal_to_upper_envelope()

```

void channel::CurveBuilder::compute_normal_to_upper_envelope (
    const size_t s,
    double & nx,
    double & ny ) const [private]
```

Computes an outward normal to the *s*-th line segment of the upper envelope of the channel.

Parameters

<i>s</i>	Index of a line segment of the upper channel envelope.
<i>nx</i>	A reference to the first Cartesian coordinate of the normal.
<i>ny</i>	A reference to the second Cartesian coordinate of the normal.

Definition at line 1125 of file curvebuilder.cpp.

```

1131 {
1132 #ifdef DEBUGMODE
1133     assert( s < _nc );
1134 #endif
1135
1136     size_t t = s + 1 ;
1137
1138     if ( _closed ) {
1139         t %= _nc ;
1140     }
1141
1142     nx = _uycoords[ t ] - _uycoords[ s ] ;
```

```

1143     ny = _uxcoords[ s ] - _uxcoords[ t ] ;
1144
1145     return ;
1146 }

```

References `_closed`, `_nc`, `_uxcoords`, and `_uycoords`.

Referenced by `compute_sleeve_corners_in_channel_constraints()`.

16.4.3.11 compute_second_difference_column_index()

```

size_t channel::CurveBuilder::compute_second_difference_column_index (
    const size_t p,
    const size_t i,
    const size_t l,
    const size_t v ) const [private]

```

Computes the index of the linear program matrix column corresponding to the x - or y -coordinate of the l -th bound of the i -th second difference of the p -th segment of the b-spline to be threaded into the channel.

Computes the index of the linear program matrix column corresponding to the x - or y -coordinate of the l -th bound of the i -th second difference of the p -th segment of the b-spline to be threaded into the channel.

Parameters

p	Index of the b-spline segment.
i	Index of the second difference of the p -th b-spline segment.
l	Index of the l -th bound of the second difference (0 - lower bound; 1 - upper bound).
v	Index of the x - or y -coordinate of the second difference bound.

Returns

The index of the linear program matrix column corresponding to the x - or y -coordinate of the l -th bound of the i -th second difference of the p -th segment of the b-spline to be threaded into the channel.

Parameters

p	Index of the b-spline segment.
i	Index of the second difference of the p -th b-spline segment.
l	Index of the l -th bound of the second difference (0 - lower bound; 1 - upper bound).
v	Index of the x - or y -coordinate of the second difference bound.

Returns

The index of the linear program matrix column corresponding to the x - or y -coordinate of the l -th bound of the i -th second difference of the p -th segment of the b-spline to be threaded into the channel.

Definition at line 1243 of file curvebuilder.cpp.

```

1250 {
1251     #ifdef DEBUGMODE
1252         assert( p < _np ) ;
1253         assert( i >= 1 ) ;
1254         assert( i <= 2 ) ;
1255         assert( l <= 1 ) ;
1256         assert( v <= 1 ) ;
1257     #endif
1258
1259     size_t offset = ( 2 * _np ) + 6 ;
1260
1261     return offset + ( 4 * ( p + i - 1 ) ) + ( 2 * l ) + v ;
1262 }
```

References `_np`.

Referenced by `compute_min_max_constraints()`, `compute_sleeve_corners_in_channel_constraints()`, `compute_sleeve_inside_csegment_constraints()`, `get_lp_solver_result_information()`, `set_up_objective_function()`, and `set_up_structural_variables()`.

16.4.3.12 `compute_sleeve_corners_in_channel_constraints()`

```

void channel::CurveBuilder::compute_sleeve_corners_in_channel_constraints (
    size_t & eqline ) [private]
```

Computes the equations defining the constraints that ensure that the breakpoints of the sleeves are inside the channel.

Parameters

<code>eqline</code>	A reference to the counter of equations.
---------------------	--

Definition at line 549 of file curvebuilder.cpp.

```

550 {
551     //
552     // Compute outward normals to the line segments of the channel.
553     //
554     std::vector< std::vector< double > > nl( _nc , std::vector< double >( 2 , 0 ) ) ;
555     std::vector< std::vector< double > > nu( _nc , std::vector< double >( 2 , 0 ) ) ;
556
557     for ( size_t c = 0 ; c < _nc ; c++ ) {
558         compute_normal_to_lower_envelope( c , nl[ c ][ 0 ] , nl[ c ][ 1 ] ) ;
559         compute_normal_to_upper_envelope( c , nu[ c ][ 0 ] , nu[ c ][ 1 ] ) ;
560     }
561
562     // Each segment of the b-spline must be enclosed by a sleeve with
563     // four breakpoints, two of which are shared with the previous and
564     // next segment (if any). Each breakpoint is constrained to be
565     // bounded by a pair of parallel segments (lower and upper) of the
566     // channel.
567
568     const size_t lo = ( 3 * 3 ) + 1 ;
569     const size_t up = ( 3 * _np ) + 8 ;
570     const double NcOverNp = _nc / double( _np ) ;
571
572     for ( size_t u = lo ; u <= up ; u++ ) {
573         //
574         // Find the index of the channel segment corresponding to \e u.
575         //
576         double t = u / double( 3 ) ;
577         double s = t - floor( t ) ;
578         size_t p = ( size_t ) floor( t - 3 ) ;
579         size_t c = ( size_t ) ( ( t - 3 ) * NcOverNp ) ;
```

```

580
581 // Compute the column indices of the linear program matrix
582 // corresponding to the four control points defining the p-th
583 // segment of the b-spline curve.
584
585 std::vector< std::vector< size_t > > cp( 4 , std::vector< size_t >( 2 , 0 ) ) ;
586
587 for ( size_t i = 0 ; i < 4 ; i++ ) {
588     for ( size_t j = 0 ; j < 2 ; j++ ) {
589         cp[ i ][ j ] = compute_control_value_column_index( p , i , j ) ;
590     }
591 }
592
593 // Compute the column indices of the linear program matrix
594 // of the values of the second difference bounds associated with
595 // the p-th segment of the b-spline curve.
596
597 std::vector< std::vector< std::vector< size_t > > > sd(
598     2 ,
599     std::vector< std::vector< size_t > >
600     (
601         2 ,
602         std::vector< size_t >( 2 , 0 )
603     )
604 ) ;
605
606 for ( size_t j = 1 ; j < 3 ; j++ ) {
607     for ( size_t l = 0 ; l < 2 ; l++ ) {
608         for ( size_t v = 0 ; v < 2 ; v++ ) {
609             sd[ j - 1 ][ l ][ v ] = compute_second_difference_column_index( p , j , l , v ) ;
610         }
611     }
612 }
613
614 // -----
615 //
616 // Process nonlinear terms of Constraint (3a).
617 //
618 // -----
619
620 //
621 // Nonlinear terms of  $f \stackrel{e}{\sim} p f$ 
622 //
623
624 insert_nonlinear_terms_of_epiece_point_lower_bound(
625     eqline ,
626     s ,
627     c ,
628     sd ,
629     nl ,
630     nu
631 ) ;
632
633 //
634 // Nonlinear terms of  $f \stackrel{\sim}{e} p f$ .
635 //
636
637 insert_nonlinear_terms_of_epiece_point_upper_bound(
638     eqline + 2 ,
639     s ,
640     c ,
641     sd ,
642     nl ,
643     nu
644 ) ;
645
646
647 // -----
648 //
649 // Process linear terms of Constraint (3a).
650 //
651 // -----
652
653 //
654 // Linear terms of  $f \stackrel{e}{\sim} p f$ 
655 //
656
657 insert_linear_terms_of_epiece_point_bounds(
658     eqline ,
659     s ,
660     t ,

```

```

661                                     p ,
662                                     c ,
663                                     cp ,
664                                     nl ,
665                                     nu
666                                 ) ;
667
668                                //
669                                // Linear terms of \f$\stackrel{\sim}{e}^p\f$.
670                                //
671                                insert_linear_terms_of_epiece_point_bounds(
672                                eqline + 2 ,
673                                s ,
674                                t ,
675                                p ,
676                                c ,
677                                cp ,
678                                nl ,
679                                nu
680                                ) ;
681
682                                // -----
683                                //
684                                // Compute right-hand side of the constraints.
685                                //
686                                // -----
687
688                                insert_rhs_of_sleeve_corners_in_channel_constraints(
689                                eqline ,
690                                c ,
691                                nl ,
692                                nu
693                                ) ;
694
695                                //
696                                // Increment equation counter.
697                                //
698                                eqline += 4 ;
699                                }
700                                return ;
701                                }
702                                }
703                                }

```

References `_nc`, `_np`, `compute_control_value_column_index()`, `compute_normal_to_lower_envelope()`, `compute_normal_to_upper_envelope()`, `compute_second_difference_column_index()`, `insert_linear_terms_of_epiece_point_bounds()`, `insert_nonlinear_terms_of_epiece_point_lower_bound()`, `insert_nonlinear_terms_of_epiece_point_upper_bound()`, and `insert_rhs_of_sleeve_corners_in_channel_constraints()`.

Referenced by `build()`.

16.4.3.13 compute_sleeve_inside_csegment_constraints()

```

void channel::CurveBuilder::compute_sleeve_inside_csegment_constraints (
    size_t & eqline ) [private]

```

Computes the equations defining the constraints that ensure the bspline segments associated with a c-segment remain inside it.

Parameters

<i>eqline</i>	A reference to the counter of equations.
---------------	--

Definition at line 807 of file curvebuilder.cpp.

```

808 {
809     //
810     // This restriction applies to channels with at least 3 c-sections
811     // only.
812     //
813     if ( _nc < 2 ) {
814         return ;
815     }
816     //
817     // Compute normals to the c-sections of the channel.
818     //
819     const size_t NumberOfCSections = ( _closed ) ? _nc : _nc + 1 ;
820     std::vector< std::vector< double > > ncsec(
821         NumberOfCSections ,
822         std::vector< double >( 2 , 0 )
823     ) ;
824     for ( size_t c = 0 ; c < NumberOfCSections ; c++ ) {
825         compute_normal_to_csection(
826             c ,
827             ncsec[ c ][ 0 ] ,
828             ncsec[ c ][ 1 ]
829         ) ;
830     }
831     // For each inner corner of the given channel, compute two
832     // constraints which ensure that the e-piece breakpoints
833     // immediately on the right (resp. left) of the corresponding
834     // c-section remain in the right (resp. left) c-segment of the
835     // channel.
836     const double NpOverNc = _np / double( _nc ) ;
837     const double onethird = 1 / double( 3 ) ;
838     const double twothird = 2 * onethird ;
839     for ( size_t c = 1 ; c < _nc ; c++ ) {
840         //
841         // Find the parameter \e t corresponding to the \e c corner.
842         //
843         double t = ( c * NpOverNc ) + 3 ;
844         //
845         // Find the curve segment \e p containing point at parameter \e
846         // t.
847         size_t p = ( size_t ) floor( t - 3 ) ;
848         // Compute the indices of the curve segments corresponding to
849         // the e-piece breakpoints immediately to the right and left of
850         // point \c p(t).
851         double s = t - floor( t ) ;
852         size_t p1 , p2 ;
853         double s1 , s2 ;
854         if ( s == 0 ) {
855             p1 = p - 1 ;
856             p2 = p ;
857             s1 = twothird ;
858             s2 = onethird ;
859         }
860         else if ( s < onethird ) {
861             p1 = p ;
862             p2 = p ;
863             s1 = 0 ;
864             s2 = onethird ;
865         }
866         else if ( s < twothird ) {
867             p1 = p ;
868             p2 = p ;
869             s1 = onethird ;
870             s2 = twothird ;
871         }
872         else {
873             p1 = p ;
874             p2 = p ;
875             s1 = twothird ;
876             s2 = 0 ;
877         }
878     }
879 }

```

```

888     s2 = 1 ;
889 }
890
891 double t1 = p1 + 3 + s1 ;
892 double t2 = p2 + 3 + s2 ;
893
894 // Compute the column indices of the LP matrix corresponding to
895 // the second difference bounds associated with the p1-th
896 // segment.
897
898 std::vector< std::vector< std::vector< size_t > > > sd1(
899     2 ,
900     std::vector< std::vector< size_t > >
901     (
902         2 ,
903         std::vector< size_t >( 2 , 0 )
904     )
905 );
906
907 for ( size_t j = 1 ; j < 3 ; j++ ) {
908     for ( size_t l = 0 ; l < 2 ; l++ ) {
909         for ( size_t v = 0 ; v < 2 ; v++ ) {
910             sd1[ j - 1 ][ l ][ v ] = compute_second_difference_column_index(
911                                     p1 ,
912                                     j ,
913                                     l ,
914                                     v
915                                     ) ;
916         }
917     }
918 }
919
920 // Compute the column indices of the LP matrix corresponding to
921 // the second difference bounds associated with the p2-th
922 // segment.
923
924 std::vector< std::vector< std::vector< size_t > > > sd2(
925     2 ,
926     std::vector< std::vector< size_t > >
927     (
928         2 ,
929         std::vector< size_t >( 2 , 0 )
930     )
931 );
932
933 for ( size_t j = 1 ; j < 3 ; j++ ) {
934     for ( size_t l = 0 ; l < 2 ; l++ ) {
935         for ( size_t v = 0 ; v < 2 ; v++ ) {
936             sd2[ j - 1 ][ l ][ v ] = compute_second_difference_column_index(
937                                     p2 ,
938                                     j ,
939                                     l ,
940                                     v
941                                     ) ;
942         }
943     }
944 }
945
946 // Compute the column indices of the LP matrix corresponding to
947 // the four control points defining the p1-th segment of the
948 // curve.
949
950 std::vector< std::vector< size_t > > cp1( 4 , std::vector< size_t >( 2 , 0 ) ) ;
951
952 for ( size_t i = 0 ; i < 4 ; i++ ) {
953     for ( size_t j = 0 ; j < 2 ; j++ ) {
954         cp1[ i ][ j ] = compute_control_value_column_index( p1 , i , j ) ;
955     }
956 }
957
958 // Compute the column indices of the LP matrix corresponding to
959 // the four control points defining the p2-th segment of the
960 // curve.
961
962 std::vector< std::vector< size_t > > cp2( 4 , std::vector< size_t >( 2 , 0 ) ) ;
963
964 for ( size_t i = 0 ; i < 4 ; i++ ) {
965     for ( size_t j = 0 ; j < 2 ; j++ ) {
966         cp2[ i ][ j ] = compute_control_value_column_index( p2 , i , j ) ;
967     }
968 }

```



```

969
970 // -----
971 //
972 // Process nonlinear terms of Constraint (3c).
973 //
974 // -----
975
976 //
977 // Nonlinear terms of  $f \stackrel{\sim}{\sim} e^p(s_1) f$ 
978 //
979 insert_nonlinear_terms_of_epiece_point_lower_bound(
980     eqline ,
981     s1 ,
982     c ,
983     sd1 ,
984     ncsec
985 ) ;
986
987 //
988 // Nonlinear terms of  $f \stackrel{\sim}{\sim} e^p(s_1) f$ .
989 //
990 insert_nonlinear_terms_of_epiece_point_upper_bound(
991     eqline + 1 ,
992     s1 ,
993     c ,
994     sd1 ,
995     ncsec
996 ) ;
997
998 //
999 // Nonlinear terms of  $f \stackrel{\sim}{\sim} e^p(s_2) f$ 
1000 //
1001 insert_nonlinear_terms_of_epiece_point_lower_bound(
1002     eqline + 2 ,
1003     s2 ,
1004     c ,
1005     sd2 ,
1006     ncsec
1007 ) ;
1008
1009 //
1010 // Nonlinear terms of  $f \stackrel{\sim}{\sim} e^p(s_2) f$ .
1011 //
1012 insert_nonlinear_terms_of_epiece_point_upper_bound(
1013     eqline + 3 ,
1014     s2 ,
1015     c ,
1016     sd2 ,
1017     ncsec
1018 ) ;
1019
1020 // -----
1021 //
1022 // Process linear terms of Constraint (3c).
1023 //
1024 // -----
1025
1026 //
1027 // Linear terms of  $f \stackrel{\sim}{\sim} e^p(s_1) f$ 
1028 //
1029 insert_linear_terms_of_epiece_point_bounds(
1030     eqline ,
1031     s1 ,
1032     t1 ,
1033     p1 ,
1034     c ,
1035     cp1 ,
1036     ncsec
1037 ) ;
1038
1039 //
1040 // Linear terms of  $f \stackrel{\sim}{\sim} e^p(s_2) f$ .
1041 //
1042 insert_linear_terms_of_epiece_point_bounds(
1043     eqline + 2 ,
1044     s2 ,
1045     t2 ,
1046     p2 ,
1047     c ,
1048     cp2 ,
1049     ncsec

```

```

1050                                     ) ;
1051
1052     // -----
1053     //
1054     // Compute right-hand side of the constraints.
1055     //
1056     // -----
1057
1058     insert_rhs_of_sleeve_inside_csegment_constraints(
1059                                     eqline ,
1060                                     c ,
1061                                     ncsec
1062                                     ) ;
1063
1064     //
1065     // Increment equation counter.
1066     //
1067     eqline += 4 ;
1068 }
1069 return ;
1070 }

```

References `_closed`, `_nc`, `_np`, `compute_control_value_column_index()`, `compute_normal_to_csection()`, `compute_←
_second_difference_column_index()`, `insert_linear_terms_of_epiece_point_bounds()`, `insert_nonlinear_terms_of_←
epiece_point_lower_bound()`, `insert_nonlinear_terms_of_epiece_point_upper_bound()`, and `insert_rhs_of_sleeve_←
inside_csegment_constraints()`.

Referenced by `build()`.

16.4.3.14 evaluate_bounding_polynomial()

```

void channel::CurveBuilder::evaluate_bounding_polynomial (
    const size_t j,
    const double t,
    double & lower,
    double & upper ) [private]

```

Obtains a lower bound and an upper bound for the value of a precomputed, bounding polynomial at a given parameter value.

Parameters

<i>j</i>	An index for the precomputed, bounding polynomial.
<i>t</i>	A parameter value.
<i>lower</i>	A reference to the lower bound.
<i>upper</i>	A reference to the upper bound.

Definition at line 2074 of file `curvebuilder.cpp`.

```

2080 {
2081     try {
2082         lower = _tf->alower( j , t ) ;
2083         upper = _tf->aupper( j , t ) ;
2084     }
2085     catch ( const ExceptionObject& xpt ) {
2086         treat_exception( xpt ) ;
2087         exit( EXIT_FAILURE ) ;
2088     }
2089
2090     return ;

```

```
2091 }
```

References `_tf`, `channel::TabulatedFunction::alower()`, `channel::TabulatedFunction::aupper()`, and `treat_exception`.

Referenced by `insert_nonlinear_terms_of_epiece_point_lower_bound()`, and `insert_nonlinear_terms_of_epiece_point_upper_bound()`.

16.4.3.15 get_bound_of_ith_constraint()

```
double channel::CurveBuilder::get_bound_of_ith_constraint (
    const size_t i ) const [inline]
```

Returns the real value on the right-hand side of the equality or inequality corresponding to the i -th constraint.

Parameters

<i>i</i>	The index of a constraint.
----------	----------------------------

Returns

The real value on the right-hand side of the equality or inequality corresponding to the i -th constraint.

Definition at line 429 of file `curvebuilder.hpp`.

```
430 {
431     if ( _coefficients.empty() ) {
432         std::stringstream ss( std::stringstream::in | std::stringstream::out );
433         ss << "No constraint has been created so far" ;
434         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
435     }
436
437     if ( i >= _coefficients.size() ) {
438         std::stringstream ss( std::stringstream::in | std::stringstream::out );
439         ss << "Constraint index is out of range" ;
440         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
441     }
442
443 #ifdef DEBUGMODE
444     assert( _bounds.size() == _coefficients.size() ) ;
445     assert( _bounds.size() > std::vector< std::vector< Bound > >::size_type( i ) ) ;
446 #endif
447
448     return _bounds[ i ].get_value() ;
449 }
```

References `_bounds`, and `_coefficients`.

16.4.3.16 get_coefficient_identfier()

```
size_t channel::CurveBuilder::get_coefficient_identfier (
    const size_t i,
    const size_t j ) const [inline]
```

Returns the index of the column that corresponds to the j -th coefficient of the i -th constraint in the matrix associated with the linear program (LP) instance.

Parameters

<i>i</i>	The index of a constraint.
<i>j</i>	The <i>j</i> -th (nonzero) coefficient of the <i>i</i> -th constraint.

Returns

The index of the column that corresponds to the *j*-th coefficient of the *i*-th constraint in the matrix associated with the linear program (LP) instance.

Definition at line 354 of file curvebuilder.hpp.

```

355     {
356         if ( _coefficients.empty() ) {
357             std::stringstream ss( std::stringstream::in | std::stringstream::out );
358             ss << "No constraint has been created so far" ;
359             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
360         }
361
362         if ( i >= _coefficients.size() ) {
363             std::stringstream ss( std::stringstream::in | std::stringstream::out );
364             ss << "Constraint index is out of range" ;
365             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
366         }
367
368         if ( j >= _coefficients[ i ].size() ) {
369             std::stringstream ss( std::stringstream::in | std::stringstream::out );
370             ss << "Coefficient index is out of range" ;
371             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
372         }
373
374         return _coefficients[ i ][ j ].get_col() ;
375     }

```

References `_coefficients`.

16.4.3.17 `get_coefficient_value()`

```

size_t channel::CurveBuilder::get_coefficient_value (
    const size_t i,
    const size_t j ) const [inline]

```

Returns the (i, j) entry of the matrix associated with the instance of the linear program.

Parameters

<i>i</i>	The index of a constraint.
<i>j</i>	The <i>j</i> -th (nonzero) coefficient of the <i>i</i> -th constraint.

Returns

The (i, j) entry of the matrix associated with the instance of the linear program.

Definition at line 392 of file curvebuilder.hpp.

```

393 {
394     if ( _coefficients.empty() ) {
395         std::stringstream ss( std::stringstream::in | std::stringstream::out );
396         ss << "No constraint has been created so far" ;
397         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
398     }
399
400     if ( i >= _coefficients.size() ) {
401         std::stringstream ss( std::stringstream::in | std::stringstream::out );
402         ss << "Constraint index is out of range" ;
403         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
404     }
405
406     if ( j >= _coefficients[ i ].size() ) {
407         std::stringstream ss( std::stringstream::in | std::stringstream::out );
408         ss << "Coefficient index is out of range" ;
409         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
410     }
411
412     return _coefficients[ i ][ j ].get_value() ;
413 }

```

References `_coefficients`.

16.4.3.18 get_control_value()

```

double channel::CurveBuilder::get_control_value (
    const size_t i,
    const size_t v ) const [inline]

```

Returns the v -th coordinate of the i -th control point of the b-spline curve threaded into the channel.

Parameters

i	The index of the i -th control point of the b-spline curve.
v	The v -th Cartesian coordinate of the i -th control point of the b-spline curve.

Returns

The v -th coordinate of the i -th control point of the b-spline curve threaded into the channel.

Definition at line 278 of file `curvebuilder.hpp`.

```

283 {
284     if ( i >= ( _np + 3 ) ) {
285         std::stringstream ss( std::stringstream::in | std::stringstream::out );
286         ss << "Index of the control point is out of range" ;
287         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
288     }
289
290     if ( v >= 2 ) {
291         std::stringstream ss( std::stringstream::in | std::stringstream::out );
292         ss << "Index of the Cartesian coordinate is out of range" ;
293         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
294     }
295
296     if ( _ctrlpts.empty() ) {
297         std::stringstream ss( std::stringstream::in | std::stringstream::out );
298         ss << "Control points have not been computed" ;
299         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
300     }
301 }

```

```

302         size_t index = ( 2 * i ) + v ;
303
304         return _ctrlpts[ index ] ;
305     }

```

References `_ctrlpts`, and `_np`.

16.4.3.19 `get_degree()`

```
size_t channel::CurveBuilder::get_degree ( ) const [inline]
```

Returns the degree of the bspline curve.

Returns

The degree of the bspline curve.

Definition at line 174 of file `curvebuilder.hpp`.

```

175     {
176         return 3 ;
177     }

```

Referenced by `get_number_of_control_points()`.

16.4.3.20 `get_lower_bound_on_second_difference_value()`

```
double channel::CurveBuilder::get_lower_bound_on_second_difference_value (
    const size_t p,
    const size_t i,
    const size_t v ) const [inline]
```

Returns the lower bound (found by the LP solver) on the v -th coordinate of the i -th second difference of the i -th curve segment of the b-spline curve threaded into the channel.

Parameters

p	The index of a curve segment of the b-spline.
i	The index of the i -th second difference of the p -th curve segment of the b-spline.
v	The v -th Cartesian coordinate of the i -th control point of the p -th curve segment of the b-spline.

Returns

The lower bound (found by the LP solver) on the v -th coordinate of the i -th second difference vector of the p -th curve segment of the b-spline curve threaded into the channel.

Definition at line 581 of file curvebuilder.hpp.

```

587 {
588     if ( p >= _np ) {
589         std::stringstream ss( std::stringstream::in | std::stringstream::out );
590         ss << "Index of the curve segment is out of range" ;
591         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
592     }
593
594     if ( ( i < 1 ) || ( i > 3 ) ) {
595         std::stringstream ss( std::stringstream::in | std::stringstream::out );
596         ss << "Index of the second difference vector is out of range" ;
597         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
598     }
599
600     if ( v >= 2 ) {
601         std::stringstream ss( std::stringstream::in | std::stringstream::out );
602         ss << "Index of the Cartesian coordinate is out of range" ;
603         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
604     }
605
606     if ( _secdiff.empty() ) {
607         std::stringstream ss( std::stringstream::in | std::stringstream::out );
608         ss << "Second differences have not been computed" ;
609         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
610     }
611
612     size_t index = ( 4 * 2 * p ) + ( 4 * ( i - 1 ) ) + v ;
613
614     return _secdiff[ index ] ;
615 }

```

References `_np`, and `_secdiff`.

16.4.3.21 get_lp_solver_result_information()

```

void channel::CurveBuilder::get_lp_solver_result_information (
    glp_prob * lp ) [private]

```

Obtain the optimal values found by the LP solver for the structural values of the linear programming corresponding to the channel problem.

Parameters

<i>lp</i>	A pointer to the instance of the LP program.
-----------	--

Definition at line 2693 of file curvebuilder.cpp.

```

2694 {
2695     //
2696     // Obtain the control points of the spline curve.
2697     //
2698     for ( size_t i = 0 ; i < 4 ; i++ ) {
2699         for ( size_t v = 0 ; v < 2 ; v++ ) {
2700             size_t c = compute_control_value_column_index(
2701                 0 ,
2702                 i ,
2703                 v ,
2704                 ) ;
2705             _ctrlpts.push_back(
2706                 glp_get_col_prim(
2707                     lp ,
2708                     int( c ) + 1
2709                 )
2710             ) ;
2711         }
2712     }

```

```

2713
2714     for ( size_t p = 1 ; p < _np ; p++ ) {
2715         for ( size_t v = 0 ; v < 2 ; v++ ) {
2716             size_t c = compute_control_value_column_index(
2717                                     p ,
2718                                     3 ,
2719                                     v
2720                                 ) ;
2721             _ctrlpts.push_back(
2722                 glp_get_col_prim(
2723                     lp ,
2724                     int( c ) + 1
2725                 )
2726             ) ;
2727         }
2728     }
2729
2730     //
2731     // Obtain the lower and upper bounds of the second differences.
2732     //
2733     for ( size_t i = 1 ; i < 3 ; i++ ) {
2734         for ( size_t l = 0 ; l < 2 ; l++ ) {
2735             for ( size_t v = 0 ; v < 2 ; v++ ) {
2736                 size_t c = compute_second_difference_column_index(
2737                                     0 ,
2738                                     i ,
2739                                     l ,
2740                                     v
2741                                 ) ;
2742                 _secdiff.push_back(
2743                     glp_get_col_prim(
2744                         lp ,
2745                         int( c ) + 1
2746                     )
2747                 ) ;
2748             }
2749         }
2750     }
2751 }
2752
2753     for ( size_t p = 1 ; p < _np ; p++ ) {
2754         for ( size_t l = 0 ; l < 2 ; l++ ) {
2755             for ( size_t v = 0 ; v < 2 ; v++ ) {
2756                 size_t c = compute_second_difference_column_index(
2757                                     p ,
2758                                     2 ,
2759                                     l ,
2760                                     v
2761                                 ) ;
2762                 _secdiff.push_back(
2763                     glp_get_col_prim(
2764                         lp ,
2765                         int( c ) + 1
2766                     )
2767                 ) ;
2768             }
2769         }
2770     }
2771 }
2772
2773     //
2774     // Obtain the minimum value of the objective function.
2775     //
2776     _ofvalue = glp_get_obj_val( lp ) ;
2777
2778     return ;
2779 }

```

References `_ctrlpts`, `_np`, `_ofvalue`, `_secdiff`, `compute_control_value_column_index()`, and `compute_second_difference_column_index()`.

Referenced by `solve_lp()`.

16.4.3.22 get_number_of_coefficients_in_the_ith_constraint()

```
size_t channel::CurveBuilder::get_number_of_coefficients_in_the_ith_constraint (
    const size_t i ) const [inline]
```

Returns the number of coefficients of the i -th constraint of the instance of the linear program.

Parameters

<i>i</i>	The index of a constraint.
----------	----------------------------

Returns

The number of coefficients of the i -th constraint of the instance of the linear program.

Definition at line 320 of file curvebuilder.hpp.

```
321 {
322     if ( _coefficients.empty() ) {
323         std::stringstream ss( std::stringstream::in | std::stringstream::out );
324         ss << "No constraint has been created so far" ;
325         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
326     }
327
328     if ( i >= _coefficients.size() ) {
329         std::stringstream ss( std::stringstream::in | std::stringstream::out );
330         ss << "Constraint index is out of range" ;
331         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
332     }
333
334     return _coefficients[ i ].size() ;
335 }
```

References [_coefficients](#).

16.4.3.23 get_number_of_constraints()

```
size_t channel::CurveBuilder::get_number_of_constraints ( ) const [inline]
```

Returns the number of constraints of the instance of the linear program corresponding to the channel problem solved by this class.

Returns

The number of constraints of the instance of the linear program corresponding to the channel problem solved by this class.

Definition at line 250 of file curvebuilder.hpp.

```
251 {
252     if ( _coefficients.empty() ) {
253         std::stringstream ss( std::stringstream::in | std::stringstream::out );
254         ss << "No constraint has been created so far" ;
255         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
256     }
257
258     return _coefficients.size() ;
259 }
```

References [_coefficients](#).

16.4.3.24 get_number_of_control_points()

```
size_t channel::CurveBuilder::get_number_of_control_points ( ) const [inline]
```

Returns the number of control points of the b-spline.

Returns

The number of control points of the b-spline.

Definition at line 232 of file curvebuilder.hpp.

```
233 {  
234     return _np + get_degree() ;  
235 }
```

References `_np`, and `get_degree()`.

16.4.3.25 get_number_of_csegments()

```
size_t channel::CurveBuilder::get_number_of_csegments ( ) const [inline]
```

Returns the number of c-segments of the channel.

Returns

The number of c-segments of the channel.

Definition at line 202 of file curvebuilder.hpp.

```
203 {  
204     return _nc ;  
205 }
```

References `_nc`.

16.4.3.26 get_number_of_segments()

```
size_t channel::CurveBuilder::get_number_of_segments ( ) const [inline]
```

Returns the number of b-spline segments.

Returns

The number of b-spline segments.

Definition at line 188 of file curvebuilder.hpp.

```
189 {  
190     return _np ;  
191 }
```

References `_np`.

16.4.3.27 get_solver_error_message()

```
std::string channel::CurveBuilder::get_solver_error_message (  
    int error ) [inline]
```

Returns the error message of the GLPK solver associated with a given error code.

Parameters

<i>error</i>	Error code returned by the LP solver.
--------------	---------------------------------------

Returns

The error message of the GLPK solver associated with a given error code.

Definition at line 705 of file curvebuilder.hpp.

```

706     {
707         std::string message ;
708         switch ( error ) {
709             case GLP_EBADB :
710                 message = "Unable to start the search because the number of basic variables is not the same as
the number of rows in the problem object." ;
711                 break ;
712             case GLP_ESING :
713                 message = "Unable to start the search because the basis matrix corresponding to the initial basis
is singular within the working precision." ;
714                 break ;
715             case GLP_ECOND :
716                 message = "Unable to start the search because the basis matrix corresponding to the initial basis
is ill-conditioned." ;
717                 break ;
718             case GLP_EBOUND :
719                 message = "Unable to start the search because some double-bounded variables have incorrect
bounds." ;
720                 break ;
721             case GLP_EFAIL :
722                 message = "The search was prematurely terminated due to the solver failure." ;
723                 break ;
724             case GLP_EOBJLL :
725                 message = "The search was prematurely terminated because the objective function being maximized
has reached its lower limit and continues decreasing." ;
726                 break ;
727             case GLP_EOBJUL :
728                 message = "The search was prematurely terminated because the objective function being minimized
has reached its upper limit and continues increasing." ;
729                 break ;
730             case GLP_EITLIM :
731                 message = "The search was prematurely terminated because the simplex iteration limit has been
exceeded." ;
732                 break ;
733             case GLP_ETMLIM :
734                 message = "The search was prematurely terminated because the time limit has been exceeded." ;
735                 break ;
736             case GLP_ENOPFS :
737                 message = "The LP problem instance has no primal feasible solution." ;
738                 break ;
739             case GLP_ENODFS :
740                 message = "The LP problem instance has no dual feasible solution." ;
741                 break ;
742             default :
743                 message = "Unknown reason." ;
744                 break ;
745         }
746
747         return message ;
748     }

```

16.4.3.28 get_upper_bound_on_second_difference_value()

```

double channel::CurveBuilder::get_upper_bound_on_second_difference_value (
    const size_t p,
    const size_t i,
    const size_t v ) const [inline]

```

Returns the upper bound (found by the LP solver) on the v -th coordinate of the i -th second difference vector of the p -th curve segment of the b-spline curve threaded into the channel.

Parameters

p	The index of the curve segment of the b-spline.
i	The index of the i -th second difference of the p -th segment of the b-spline.
v	The v -th Cartesian coordinate of the i -th control point of the p -th curve segment of the b-spline.

Returns

The upper bound (found by the LP solver) on the v -th coordinate of the i -th second difference vector of the p -th curve segment of the b-spline curve threaded into the channel.

Definition at line 638 of file curvebuilder.hpp.

```

644     {
645         if ( p >= _np ) {
646             std::stringstream ss( std::stringstream::in | std::stringstream::out );
647             ss << "Index of the curve is out of range" ;
648             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
649         }
650
651         if ( ( i < 1 ) || ( i > 3 ) ) {
652             std::stringstream ss( std::stringstream::in | std::stringstream::out );
653             ss << "Index of the second difference vector is out of range" ;
654             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
655         }
656
657         if ( v >= 2 ) {
658             std::stringstream ss( std::stringstream::in | std::stringstream::out );
659             ss << "Index of the Cartesian coordinate is out of range" ;
660             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
661         }
662
663         if ( _secdiff.empty() ) {
664             std::stringstream ss( std::stringstream::in | std::stringstream::out );
665             ss << "Second differences have not been computed" ;
666             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
667         }
668
669         size_t index = ( 4 * 2 * p ) + ( 4 * ( i - 1 ) ) + 2 + v ;
670
671         return _secdiff[ index ] ;
672     }

```

References `_np`, and `_secdiff`.

16.4.3.29 insert_bound()

```

void channel::CurveBuilder::insert_bound (
    const size_t eqline,
    const Bound::CONSTRAINTTYPE type,
    const double value ) [inline], [private]

```

Assigns a real value to the right-hand side of a constraint (equality or inequality) of an instance of the linear program associated with the channel problem.

Parameters

<i>eqline</i>	Matrix row corresponding to the constraint.
<i>type</i>	Type of the bound (==, >= or <=).
<i>value</i>	<code>Bound</code> value (right-hand side of the constraint)

Definition at line 954 of file curvebuilder.hpp.

```

959     {
960         _bounds[ eqline ] = Bound(
961             type ,
962             value ,
963             eqline
964         ) ;
965
966         return ;
967     }

```

References `_bounds`.

Referenced by `compute_channel_corners_outside_sleeve_constraints()`, `insert_extreme_point_correspondence_constraint()`, `insert_min_max_constraints()`, `insert_periodic_correspondence_constraints()`, `insert_rhs_of_sleeve_corners_in_channel_constraints()`, and `insert_rhs_of_sleeve_inside_csegment_constraints()`.

16.4.3.30 insert_coefficient()

```

void channel::CurveBuilder::insert_coefficient (
    const size_t eqline,
    const size_t index,
    const double value ) [inline], [private]

```

Assigns a value to the coefficient of an unknown of a given constraint of the linear program (LP). The unknown is identified by its corresponding column index in the associated matrix of the LP.

Parameters

<i>eqline</i>	Matrix row corresponding to the constraint.
<i>index</i>	Matrix column index corresponding to the unknown.
<i>value</i>	Value to be assigned to the unknown coefficient.

Definition at line 924 of file curvebuilder.hpp.

```

929     {
930         _coefficients[ eqline ].push_back(
931             Coefficient(
932                 eqline ,
933                 index ,
934                 value
935             )
936         ) ;
937
938         return ;
939     }

```

References `_coefficients`.

Referenced by `compute_channel_corners_outside_sleeve_constraints()`, `insert_csegment_constraint()`, `insert_extreme_point_correspondence_constraint()`, `insert_min_max_constraints()`, and `insert_periodic_correspondence_constraints()`.

16.4.3.31 insert_csegment_constraint() [1/3]

```
void channel::CurveBuilder::insert_csegment_constraint (
    const size_t eqline,
    const double c0,
    const double c1,
    const double c2,
    const double c3,
    const double c4,
    const size_t b0,
    const size_t b1,
    const size_t b2,
    const size_t b3,
    const size_t b4,
    const double normal ) [private]
```

Inserts the coefficients of the linear terms of the upper and lower bounds of an e-piece point equation into the matrix associated with an instance of the linear program (LP). This point belongs to an e-piece segment whose endpoints bound b-spline curve points in two distinct, but consecutive curve segments. The constraint ensures that the e-piece point lies inside a c-segment of the channel.

Parameters

<i>eqline</i>	A counter for the number of constraints.
<i>c0</i>	Coefficient of the first control point of the b-spline segment containing the curve point associated with the right endpoint of the e-piece segment.
<i>c1</i>	Coefficient of the second control point of the b-spline segment containing the curve point associated with the right endpoint of the e-piece segment.
<i>c2</i>	Coefficient of the third control point of the b-spline segment containing the curve point associated with the right endpoint of the e-piece segment.
<i>c3</i>	Coefficient of the fourth control point of the b-spline segment containing the curve point associated with the right endpoint of the e-piece segment.
<i>c4</i>	Coefficient of the fourth control point of the b-spline segment containing the curve point associated with the left endpoint of the e-piece segment.
<i>b0</i>	Index of the LP matrix column corresponding to the first control point of the b-spline segment containing the curve point associated with the right endpoint of the e-piece segment.
<i>b1</i>	Index of the LP matrix column corresponding to the second control point of the b-spline segment containing the curve point associated with the right endpoint of the e-piece segment.
<i>b2</i>	Index of the LP matrix column corresponding to the third control point of the b-spline segment containing the curve point associated with the right endpoint of the e-piece segment.
<i>b3</i>	Index of the LP matrix column corresponding to the fourth control point of the b-spline segment containing the curve point associated with the right endpoint of the e-piece segment.
<i>b4</i>	Index of the LP matrix column corresponding to the fourth control point of the b-spline segment containing the curve point associated with the left endpoint of the e-piece segment.
<i>normal</i>	A normal to a supporting, oriented line of one of the four line segments delimiting a specific c-segment of the channel.

Definition at line 2271 of file curvebuilder.cpp.

```
2285 {
2286     insert_csegment_constraint (
2287         eqline ,
```

```

2288             c0 ,
2289             c1 ,
2290             c2 ,
2291             c3 ,
2292             b0 ,
2293             b1 ,
2294             b2 ,
2295             b3 ,
2296             normal
2297         ) ;
2298
2299         double temp = c4 * normal ;
2300         if ( temp != 0 ) {
2301             insert_coefficient( eqline , b4 , temp ) ;
2302         }
2303
2304         return ;
2305     }

```

References `insert_coefficient()`, and `insert_csegment_constraint()`.

16.4.3.32 insert_csegment_constraint() [2/3]

```

void channel::CurveBuilder::insert_csegment_constraint (
    const size_t eqline,
    const double c0,
    const double c1,
    const double c2,
    const double c3,
    const size_t b0,
    const size_t b1,
    const size_t b2,
    const size_t b3,
    const double normal ) [private]

```

Inserts the coefficients of the linear terms of the upper and lower bounds of an e-piece point equation into the matrix associated with an instance of the linear program (LP). The constraint ensures that the point of the e-piece lies inside a c-segment of the channel.

Parameters

<i>eqline</i>	A counter for the number of constraints.
<i>c0</i>	Coefficient of the first control point of the b-spline segment containing the curve point associated to the e-piece point.
<i>c1</i>	Coefficient of the second control point of the b-spline segment containing the curve point associated to the e-piece point.
<i>c2</i>	Coefficient of the third control point of the b-spline segment containing the curve point associated to the e-piece point.
<i>c3</i>	Coefficient of the fourth control point of the b-spline segment containing the curve point associated to the e-piece point.
<i>b0</i>	Index of the LP matrix column corresponding to the first control point of the b-spline segment containing the curve point associated to the e-piece point.
<i>b1</i>	Index of the LP matrix column corresponding to the second control point of the b-spline segment containing the curve point associated to the e-piece point.

Parameters

<i>b2</i>	Index of the LP matrix column corresponding to the third control point of the b-spline segment containing the curve point associated to the e-piece point.
<i>b3</i>	Index of the LP matrix column corresponding to the fourth control point of the b-spline segment containing the curve point associated to the e-piece point.
<i>normal</i>	A normal to a supporting, oriented line of one of the four line segments delimiting a specific c-segment of the channel.

Definition at line 2185 of file curvebuilder.cpp.

```

2197 {
2198     double temp = c0 * normal ;
2199     if ( temp != 0 ) {
2200         insert_coefficient( eqline , b0 , temp ) ;
2201     }
2202
2203     temp = c1 * normal ;
2204     if ( temp != 0 ) {
2205         insert_coefficient( eqline , b1 , temp ) ;
2206     }
2207
2208     temp = c2 * normal ;
2209     if ( temp != 0 ) {
2210         insert_coefficient( eqline , b2 , temp ) ;
2211     }
2212
2213     temp = c3 * normal ;
2214     if ( temp != 0 ) {
2215         insert_coefficient( eqline , b3 , temp ) ;
2216     }
2217
2218     return ;
2219 }
```

References `insert_coefficient()`.

16.4.3.33 `insert_csegment_constraint()` [3/3]

```

void channel::CurveBuilder::insert_csegment_constraint (
    const size_t eqline,
    const double lower,
    const double upper,
    const size_t sdlo,
    const size_t sdup,
    const double normal ) [private]
```

Inserts the coefficients of the lower and upper bounds of a constraint second difference term into the matrix associated with an instance of the linear program (LP). The term belongs to the equation defining the upper (or lower) bound of a point of an e-piece. The constraint ensures that the point lies on a specific side of the oriented supporting line of one of the four line segments delimiting a c-segment of the channel.

Parameters

<i>eqline</i>	A counter for the number of constraints.
<i>lower</i>	Coefficient of the second difference lower bound term.
<i>upper</i>	Coefficient of the second difference upper bound term.
<i>sdlo</i>	The index of the LP matrix column corresponding to the second difference lower bound term.
<i>sdup</i>	The index of the LP matrix column corresponding to the second difference upper bound term.
<i>normal</i>	A normal to a supporting, oriented line of one of the four line segments delimiting a specific c-segment of the channel.

Definition at line 2120 of file curvebuilder.cpp.

```

2128 {
2129     double temp ;
2130
2131     temp = lower * normal ;
2132     if ( temp != 0 ) {
2133         insert_coefficient( eqline , sdlo , temp ) ;
2134     }
2135
2136     temp = upper * normal ;
2137     if ( temp != 0 ) {
2138         insert_coefficient( eqline , sdup , temp ) ;
2139     }
2140
2141     return ;
2142 }
```

References `insert_coefficient()`.

Referenced by `insert_csegment_constraint()`, `insert_linear_terms_of_epiece_point_bounds()`, `insert_nonlinear_terms_of_epiece_point_lower_bound()`, and `insert_nonlinear_terms_of_epiece_point_upper_bound()`.

16.4.3.34 insert_extreme_point_correspondence_constraint()

```

void channel::CurveBuilder::insert_extreme_point_correspondence_constraint (
    const size_t eqline,
    const std::vector< size_t > & col,
    const std::vector< double > & val,
    const double rhs ) [private]
```

Inserts into the linear program (LP) matrix the coefficients of the unknowns and the right-hand side value of a constraint corresponding to the location of the starting or final point of the b-spline curve.

Parameters

<i>eqline</i>	A reference to the counter of equations.
<i>col</i>	An array with the LP matrix column indices corresponding to the unknowns of the correspondence constraint.
<i>val</i>	An array with the values corresponding to the unknowns of the correspondence constraint.
<i>rhs</i>	The right-hand side value of the constraint.

Definition at line 1411 of file curvebuilder.cpp.

```

1417 {
1418     for ( size_t i = 0 ; i < 4 ; i++ ) {
1419         insert_coefficient( eqline , col[ i ] , val[ i ] ) ;
1420     }
1421
1422     insert_bound( eqline , Bound::EQT , rhs ) ;
1423
1424     return ;
1425 }
```

References `insert_bound()`, and `insert_coefficient()`.

Referenced by `compute_correspondence_constraints()`.

16.4.3.35 insert_linear_terms_of_epiece_point_bounds() [1/2]

```
void channel::CurveBuilder::insert_linear_terms_of_epiece_point_bounds (
    const size_t eqline,
    const double s,
    const double t,
    const size_t p,
    const size_t c,
    const std::vector< std::vector< size_t > > & cp,
    const std::vector< std::vector< double > > & ncsec ) [private]
```

Inserts the coefficients of the linear terms of the equation defining lower and upper bounds for the e-piece points into the matrix associated with the Linear Program (LP). These terms occur in the constraint that enforces an e-piece point to stay either on the right or on left side of a channel c-section.

Parameters

<i>eqline</i>	A counter for the number of constraints.
<i>s</i>	A parameter value identifying a point on the e-piece.
<i>t</i>	A parameter value identifying the b-spline point that corresponds to the point on the e-piece at parameter <i>s</i> .
<i>p</i>	Index of the b-spline segment containing the b-spline point at parameter <i>t</i> .
<i>c</i>	An index identifying the c-segment the e-piece point belongs to.
<i>cp</i>	Array with the LP matrix column indices corresponding to the control points of the b-spline defining the <i>p</i> -th piece of the curve.
<i>ncsec</i>	Array of Cartesian coordinates of normals (pointing to the left) to the supporting lines of the c-sections of the channel.

Definition at line 1926 of file curvebuilder.cpp.

```
1935 {
1936     //
1937     // The coefficients are the same for each Cartesian coordinate.
1938     //
1939     const double onesixth = double( 1 ) / double( 6 ) ;
1940
1941     // The upper and lower bounds on the e-piece point must be either
1942     // on the left or on the right side of a c-section of the channel.
1943
1944     const double c0 = onesixth * ( 1 - s ) ;
1945     const double c1 = ( ( -2 + 3 * s ) * onesixth ) + ( p + 4 - t ) ;
1946     const double c2 = ( ( 1 - 3 * s ) * onesixth ) + ( t - p - 3 ) ;
1947     const double c3 = onesixth * s ;
1948
1949     for ( size_t v = 0 ; v < 2 ; v++ ) {
1950         //
1951         // Compute constraints for the v-th Cartesian coordinate.
1952         //
1953
1954         //
1955         // Lower bound --> Equation eqline
1956         //
1957         insert_csegment_constraint(
1958             eqline ,
1959             c0 ,
1960             c1 ,
1961             c2 ,
1962             c3 ,
1963             cp[ 0 ][ v ] ,
1964             cp[ 1 ][ v ] ,
1965             cp[ 2 ][ v ] ,
1966             cp[ 3 ][ v ] ,
1967             ncsec[ c ][ v ]
1968         ) ;
1969     }
```

```

1970      //
1971      // Upper bound --> Equation eqline + 1
1972      //
1973      insert_csegment_constraint(
1974          eqline + 1 ,
1975          c0 ,
1976          c1 ,
1977          c2 ,
1978          c3 ,
1979          cp[ 0 ][ v ] ,
1980          cp[ 1 ][ v ] ,
1981          cp[ 2 ][ v ] ,
1982          cp[ 3 ][ v ] ,
1983          ncsec[ c ][ v ]
1984      ) ;
1985  }
1986
1987  return ;
1988  }

```

References `insert_csegment_constraint()`.

16.4.3.36 insert_linear_terms_of_epiece_point_bounds() [2/2]

```

void channel::CurveBuilder::insert_linear_terms_of_epiece_point_bounds (
    const size_t eqline,
    const double s,
    const double t,
    const size_t p,
    const size_t c,
    const std::vector< std::vector< size_t > > & cp,
    const std::vector< std::vector< double > > & nl,
    const std::vector< std::vector< double > > & nu ) [private]

```

Inserts the coefficients of the linear terms of the equation defining lower and upper bounds for the e-piece points into the matrix associated with the Linear Program (LP). These terms occur in the constraint that enforces an e-piece point to stay inside channel.

Parameters

<i>eqline</i>	A counter for the number of constraints.
<i>s</i>	A parameter value identifying a point on the e-piece.
<i>t</i>	A parameter value identifying the b-spline point that corresponds to the point on the e-piece at parameter <i>t</i> .
<i>p</i>	Index of the b-spline segment containing the b-spline point at parameter <i>t</i> .
<i>c</i>	An index identifying the c-segment the e-piece point belongs to.
<i>cp</i>	Array with the LP matrix column indices corresponding to the control points of the b-spline defining the <i>p</i> -th piece of the curve.
<i>nl</i>	Array of Cartesian coordinates of outward normals to the supporting lines of the lower envelope segments of the channel.
<i>nu</i>	Array of Cartesian coordinates of outward normals to the supporting lines of the upper envelope segments of the channel.

Definition at line 1837 of file `curvebuilder.cpp`.

```
1847 {
```

```

1848 //
1849 // The coefficients are the same for each Cartesian coordinate.
1850 //
1851 const double onesixth = double( 1 ) / double( 6 ) ;
1852
1853 // The upper and lower bounds on the e-piece points must be on
1854 // or above the lower envelope of the c-th c-segment of the
1855 // channel.
1856
1857 const double c0 = onesixth * ( 1 - s ) ;
1858 const double c1 = ( ( -2 + 3 * s ) * onesixth ) + ( p + 4 - t ) ;
1859 const double c2 = ( ( 1 - 3 * s ) * onesixth ) + ( t - p - 3 ) ;
1860 const double c3 = onesixth * s ;
1861
1862 for ( size_t v = 0 ; v < 2 ; v++ ) {
1863 //
1864 // Compute constraints for the v-th Cartesian coordinate.
1865 //
1866 insert_csegment_constraint(
1867     eqline ,
1868     c0 ,
1869     c1 ,
1870     c2 ,
1871     c3 ,
1872     cp[ 0 ][ v ] ,
1873     cp[ 1 ][ v ] ,
1874     cp[ 2 ][ v ] ,
1875     cp[ 3 ][ v ] ,
1876     nl[ c ][ v ]
1877 ) ;
1878
1879 // The upper and lower bounds on the e-piece points must be on
1880 // or below the upper envelope of the c-th c-segment of the
1881 // channel.
1882 insert_csegment_constraint(
1883     eqline + 1 ,
1884     c0 ,
1885     c1 ,
1886     c2 ,
1887     c3 ,
1888     cp[ 0 ][ v ] ,
1889     cp[ 1 ][ v ] ,
1890     cp[ 2 ][ v ] ,
1891     cp[ 3 ][ v ] ,
1892     nu[ c ][ v ]
1893 ) ;
1894 }
1895
1896 return ;
1897 }

```

References `insert_csegment_constraint()`.

Referenced by `compute_sleeve_corners_in_channel_constraints()`, `compute_sleeve_inside_csegment_constraints()`, and `compute_sleeve_outside_csegment_constraints()`.

16.4.3.37 insert_min_max_constraints()

```

void channel::CurveBuilder::insert_min_max_constraints (
    const size_t eqline,
    const size_t lo,
    const size_t up,
    const size_t b0,
    const size_t b1,
    const size_t b2 ) [private]

```

Inserts the coefficients of the equations defining the three min-max constraints into the matrix associated with the linear program (LP), and sets the right-hand side of the constraints as well.

Parameters

<i>eqline</i>	A reference to the counter of equations.
<i>lo</i>	Column index of the lower bound for a second difference.
<i>up</i>	Column index of the upper bound for a second difference.
<i>b0</i>	Column index of the first control value defining the second difference.
<i>b1</i>	Column index of the second control value defining the second difference.
<i>b2</i>	Column index of the third control value defining the second difference.

Definition at line 1348 of file curvebuilder.cpp.

```

1356 {
1357     // First min-max constraint: the upper bound of the second
1358     // difference must be greater than or equal to the value of the
1359     // second difference:
1360
1361     const double onesixth = double( 1 ) / double( 6 ) ;
1362
1363     insert_coefficient( eqline , up ,          1 ) ;
1364     insert_coefficient( eqline , b0 , -1 * onesixth ) ;
1365     insert_coefficient( eqline , b1 ,  2 * onesixth ) ;
1366     insert_coefficient( eqline , b2 , -1 * onesixth ) ;
1367
1368     insert_bound( eqline , Bound::GTE , 0 ) ;
1369
1370     // Second min-max constraint: the upper bound on the second
1371     // difference must be greater than or equal to zero (i.e., must be
1372     // non-negative).
1373
1374     insert_coefficient( eqline + 1 , up , 1 ) ;
1375
1376     insert_bound( eqline + 1 , Bound::GTE , 0 ) ;
1377
1378     // Third min-max constraint: the sum of the upper and lower bounds
1379     // of the second difference must be equal to the value the second
1380     // difference.
1381
1382     insert_coefficient( eqline + 2 , up ,          1 ) ;
1383     insert_coefficient( eqline + 2 , lo ,          1 ) ;
1384     insert_coefficient( eqline + 2 , b0 , -1 * onesixth ) ;
1385     insert_coefficient( eqline + 2 , b1 ,  2 * onesixth ) ;
1386     insert_coefficient( eqline + 2 , b2 , -1 * onesixth ) ;
1387
1388     insert_bound( eqline + 2 , Bound::EQT , 0 ) ;
1389
1390     return ;
1391 }
```

References `insert_bound()`, and `insert_coefficient()`.

Referenced by `compute_min_max_constraints()`.

16.4.3.38 insert_nonlinear_terms_of_epiece_point_lower_bound() [1/2]

```

void channel::CurveBuilder::insert_nonlinear_terms_of_epiece_point_lower_bound (
    const size_t eqline,
    const double s,
    const size_t c,
    const std::vector< std::vector< std::vector< size_t > > > & sd,
    const std::vector< std::vector< double > > & ncsec ) [private]
```

Inserts the coefficients of the second difference terms of the equation defining lower bounds for the e-piece points into the matrix associated with an instance of the Linear Program (LP). The terms belong to the constraint that forces one e-piece point to be on the right or left side of a channel c-section.

Parameters

<i>eqline</i>	A counter for the number of constraints.
<i>s</i>	A parameter value identifying a point on the e-piece.
<i>c</i>	An index identifying a c-segment of the channel.
<i>sd</i>	Array with the LP matrix column indices corresponding to the lower and upper bounds on second differences occurring in the equation defining the e-piece points belonging to the c-segment.
<i>ncsec</i>	Array of Cartesian coordinates of normals (pointing to the left) to the supporting lines of the c-sections of the channel.

Definition at line 1589 of file curvebuilder.cpp.

```

1596 {
1597     // Insert into the matrix associated with the linear program (LP)
1598     // the coefficients of the second differences of the e-piece
1599     // breakpoint lower bound  $f_{\text{stackrel{e}{\sim}}p}$  in constraint
1600     // (3c).
1601
1602     //
1603     // The computation is performed for each second difference j.
1604     //
1605
1606     for ( size_t j = 1 ; j < 3 ; j++ ) {
1607         //
1608         // Get lower and upper bounds for the special polynomial.
1609         //
1610         double dl ;
1611         double du ;
1612         evaluate_bounding_polynomial(
1613             j ,
1614             s ,
1615             du , // switch lower and upper bounds.
1616             dl  // switch lower and upper bounds.
1617         ) ;
1618
1619         //
1620         // The coefficients are the same for each Cartesian coordinate.
1621         //
1622
1623         for ( size_t v = 0 ; v < 2 ; v++ ) {
1624             // Point  $f_{\text{stackrel{e}{\sim}}p}(s)$  of the e-piece must
1625             // be either on the right side or on the left side of the
1626             // channel c-section.
1627             insert_csegment_constraint(
1628                 eqline ,
1629                 dl ,
1630                 du ,
1631                 sd[ j - 1 ][ 0 ][ v ] ,
1632                 sd[ j - 1 ][ 1 ][ v ] ,
1633                 ncsec[ c ][ v ]
1634             ) ;
1635         }
1636     }
1637     return ;
1638 }

```

References `evaluate_bounding_polynomial()`, and `insert_csegment_constraint()`.

16.4.3.39 insert_nonlinear_terms_of_epiece_point_lower_bound() [2/2]

```

void channel::CurveBuilder::insert_nonlinear_terms_of_epiece_point_lower_bound (
    const size_t eqline,
    const double s,

```

```

const size_t c,
const std::vector< std::vector< std::vector< size_t > > > & sd,
const std::vector< std::vector< double > > & nl,
const std::vector< std::vector< double > > & nu ) [private]

```

Inserts the coefficients of the second difference terms of the equation defining lower bounds for the e-piece points into the matrix associated with an instance of the Linear Program (LP). The terms belong to the constraint that forces the e-piece points to be inside a certain c-section of the channel.

Parameters

<i>eqline</i>	A counter for the number of constraints.
<i>s</i>	A parameter value identifying a point on the e-piece.
<i>c</i>	An index identifying a c-segment of the channel.
<i>sd</i>	Array with the LP matrix column indices corresponding to the lower and upper bounds on second differences occurring in the equation defining the e-piece points belonging to the c-segment.
<i>nl</i>	Array of Cartesian coordinates of outward normals to the supporting lines of the lower envelope segments of the channel.
<i>nu</i>	Array of Cartesian coordinates of outward normals to the supporting lines of the upper envelope segments of the channel.

Definition at line 1501 of file curvebuilder.cpp.

```

1509 {
1510     // Insert into the matrix associated with the linear program (LP)
1511     // the coefficients of the second differences of the e-piece
1512     // breakpoint lower bound \f$\stackrel{e}{\sim}^p \f$ in constraint
1513     // (3a).
1514
1515     //
1516     // The computation is performed for each second difference j.
1517     //
1518
1519     for ( size_t j = 1 ; j < 3 ; j++ ) {
1520         //
1521         // Get lower and upper bounds for the special polynomial.
1522         //
1523         double dl ;
1524         double du ;
1525         evaluate_bounding_polynomial(
1526             j ,
1527             s ,
1528             du , // switch lower and upper bounds.
1529             dl , // switch lower and upper bounds.
1530         ) ;
1531
1532         //
1533         // The coefficients are the same for each Cartesian coordinate.
1534         //
1535
1536         for ( size_t v = 0 ; v < 2 ; v++ ) {
1537             // Point \f$\stackrel{e}{\sim}^p ( s ) \f$ of the e-piece must
1538             // be above the lower envelope of the c-th c-segment of the
1539             // channel.
1540             insert_csegment_constraint(
1541                 eqline ,
1542                 dl ,
1543                 du ,
1544                 sd[ j - 1 ][ 0 ][ v ] ,
1545                 sd[ j - 1 ][ 1 ][ v ] ,
1546                 nl[ c ][ v ]
1547             ) ;
1548
1549             // Point \f$\stackrel{e}{\sim}^p ( s ) \f$ of the e-piece must
1550             // be below the upper envelope of the c-th c-segment of the
1551             // channel.
1552             insert_csegment_constraint(
1553                 eqline + 1 ,

```

```

1554             dl ,
1555             du ,
1556             sd[ j - 1 ][ 0 ][ v ] ,
1557             sd[ j - 1 ][ 1 ][ v ] ,
1558             nu[ c ][ v ]
1559         ) ;
1560     }
1561 }
1562
1563     return ;
1564 }

```

References `evaluate_bounding_polynomial()`, and `insert_csegment_constraint()`.

Referenced by `compute_sleeve_corners_in_channel_constraints()`, and `compute_sleeve_inside_csegment_constraints()`.

16.4.3.40 insert_nonlinear_terms_of_epiece_point_upper_bound() [1/2]

```

void channel::CurveBuilder::insert_nonlinear_terms_of_epiece_point_upper_bound (
    const size_t eqline,
    const double s,
    const size_t c,
    const std::vector< std::vector< std::vector< size_t > > > & sd,
    const std::vector< std::vector< double > > & ncsec ) [private]

```

Inserts the coefficients of the second difference terms of the equation defining upper bounds for the e-piece points into the matrix associated with an instance of the Linear Program (LP). The terms belong to the constraint that forces one e-piece point to be on the right or left side of a channel c-section.

Parameters

<i>eqline</i>	A counter for the number of constraints.
<i>s</i>	A parameter value identifying a point on the e-piece.
<i>c</i>	An index identifying a c-segment of the channel.
<i>sd</i>	Array with the LP matrix column indices corresponding to the lower and upper bounds on second differences occurring in the equation defining the e-piece points belonging to the c-segment.
<i>ncsec</i>	Array of Cartesian coordinates of normals (pointing to the left) to the supporting lines of the c-sections of the channel.

Definition at line 1755 of file `curvebuilder.cpp`.

```

1762 {
1763     // Insert into the matrix associated with the linear program (LP)
1764     // the coefficients of the second differences of the e-piece
1765     // breakpoint lower bound \f$\stackrel{\sim}{e}^p \f$ in constraint
1766     // (3c).
1767
1768     //
1769     // The computation is performed for each second difference j.
1770     //
1771
1772     for ( size_t j = 1 ; j < 3 ; j++ ) {
1773         //
1774         // Get lower and upper bounds for the special polynomial.
1775         //
1776         double dl ;
1777         double du ;

```



```

1778     evaluate_bounding_polynomial(
1779         j ,
1780         s ,
1781         dl , // DON't switch lower and upper bounds.
1782         du , // DON't switch lower and upper bounds.
1783     ) ;
1784
1785     //
1786     // The coefficients are the same for each Cartesian coordinate.
1787     //
1788
1789     for ( size_t v = 0 ; v < 2 ; v++ ) {
1790         // Point  $\text{f}(\text{stackrel{\sim}{e}}^p(s))$  of the e-piece must
1791         // be either on the right side or on the left side of the
1792         // channel c-section.
1793         insert_csegment_constraint(
1794             eqline ,
1795             dl ,
1796             du ,
1797             sd[ j - 1 ][ 0 ][ v ] ,
1798             sd[ j - 1 ][ 1 ][ v ] ,
1799             ncsec[ c ][ v ]
1800         ) ;
1801     }
1802 }
1803
1804 return ;
1805 }

```

References `evaluate_bounding_polynomial()`, and `insert_csegment_constraint()`.

16.4.3.41 insert_nonlinear_terms_of_epiece_point_upper_bound() [2/2]

```

void channel::CurveBuilder::insert_nonlinear_terms_of_epiece_point_upper_bound (
    const size_t eqline,
    const double s,
    const size_t c,
    const std::vector< std::vector< std::vector< size_t > > > & sd,
    const std::vector< std::vector< double > > & nl,
    const std::vector< std::vector< double > > & nu ) [private]

```

Inserts into the matrix associated with the Linear Program (LP) the coefficients of the lower and upper bounds of the second difference terms of the equation defining upper bounds for the e-piece points. These terms occur in the constraint that keep the sleeve inside a certain c-section of the channel.

Inserts the coefficients of the second difference terms of the equation defining lower bounds for the e-piece points into the matrix associated with an instance of the Linear Program (LP). The terms belong to the constraint that forces the e-piece points to be inside a certain c-section of the channel.

Parameters

<i>eqline</i>	A counter for the number of constraints.
<i>s</i>	A parameter value identifying a point on the e-piece.
<i>c</i>	An index identifying a c-segment of the channel.
<i>sd</i>	Array with the LP matrix column indices corresponding to the lower and upper bounds on second differences occurring in the equation defining the points on the e-piece matched with the c-segment.
<i>nl</i>	Array of Cartesian coordinates of outward normals to the supporting lines of the lower envelope segments of the channel.

Parameters

<i>nu</i>	Array of Cartesian coordinates of outward normals to the supporting lines of the upper envelope segments of the channel.
<i>eqline</i>	A counter for the number of constraints.
<i>s</i>	A parameter value identifying a point on the e-piece.
<i>c</i>	An index identifying a c-segment of the channel.
<i>sd</i>	Array with the LP matrix column indices corresponding to the lower and upper bounds on second differences occurring in the equation defining the e-piece points belonging to the c-segment.
<i>nl</i>	Array of Cartesian coordinates of outward normals to the supporting lines of the lower envelope segments of the channel.
<i>nu</i>	Array of Cartesian coordinates of outward normals to the supporting lines of the upper envelope segments of the channel.

Definition at line 1667 of file curvebuilder.cpp.

```

1675 {
1676     // Insert into the matrix associated with the linear program (LP)
1677     // the coefficients of the second differences of the e-piece
1678     // breakpoint lower bound  $f_{\text{stackrel{e}{\sim}}^p(s)}$  in constraint
1679     // (3a).
1680
1681     //
1682     // The computation is performed for each second difference j.
1683     //
1684
1685     for ( size_t j = 1 ; j < 3 ; j++ ) {
1686         //
1687         // Get lower and upper bounds for the special polynomial.
1688         //
1689         double dl ;
1690         double du ;
1691         evaluate_bounding_polynomial(
1692             j ,
1693             s ,
1694             dl , // DON't switch lower and upper bounds.
1695             du , // DON't switch lower and upper bounds.
1696         ) ;
1697
1698         //
1699         // The coefficients are the same for each Cartesian coordinate.
1700         //
1701
1702         for ( size_t v = 0 ; v < 2 ; v++ ) {
1703             // Point  $f_{\text{stackrel{e}{\sim}}^p(s)}$  of the e-piece must
1704             // be above the lower envelope of the c-th c-segment of the
1705             // channel.
1706             insert_csegment_constraint(
1707                 eqline ,
1708                 dl ,
1709                 du ,
1710                 sd[ j - 1 ][ 0 ][ v ] ,
1711                 sd[ j - 1 ][ 1 ][ v ] ,
1712                 nl[ c ][ v ]
1713             ) ;
1714
1715             // Point  $f_{\text{stackrel{e}{\sim}}^p(s)}$  of the e-piece must
1716             // be below the upper envelope of the c-th c-segment of the
1717             // channel.
1718             insert_csegment_constraint(
1719                 eqline + 1 ,
1720                 dl ,
1721                 du ,
1722                 sd[ j - 1 ][ 0 ][ v ] ,
1723                 sd[ j - 1 ][ 1 ][ v ] ,
1724                 nu[ c ][ v ]
1725             ) ;
1726         }
1727     }
1728
1729     return ;
1730 }
```

References `evaluate_bounding_polynomial()`, and `insert_csegment_constraint()`.

Referenced by `compute_sleeve_corners_in_channel_constraints()`, and `compute_sleeve_inside_csegment_constraints()`.

16.4.3.42 insert_periodic_correspondence_constraints()

```
void channel::CurveBuilder::insert_periodic_correspondence_constraints (
    const size_t eqline,
    const std::vector< size_t > & strx,
    const std::vector< size_t > & stry,
    const std::vector< size_t > & endx,
    const std::vector< size_t > & endy ) [private]
```

Inserts into the linear program (LP) matrix the coefficients of the unknowns and the right-hand side values of the constraints that ensure that the first three control points are the same as the last three control points (in this order).

Parameters

<i>eqline</i>	A reference to the counter of equations.
<i>strx</i>	An array with the column indices of the LP matrix corresponding to the first Cartesian coordinates of the first three control points.
<i>stry</i>	An array with the column indices of the LP matrix corresponding to the second Cartesian coordinates of the first three control points.
<i>endx</i>	An array with the column indices of the LP matrix corresponding to the first Cartesian coordinates of the last three control points.
<i>endy</i>	An array with the column indices of the LP matrix corresponding to the second Cartesian coordinates of the last three control points.

Definition at line 1452 of file `curvebuilder.cpp`.

```
1459 {
1460     for ( size_t j = 0 ; j < 3 ; j++ ) {
1461         insert_coefficient( eqline + 2 * j , strx[ j ] , 1 ) ;
1462         insert_coefficient( eqline + 2 * j , endx[ j ] , -1 ) ;
1463
1464         insert_bound( eqline + 2 * j , Bound::EQT , 0 ) ;
1465
1466         insert_coefficient( eqline + 2 * j + 1 , stry[ j ] , 1 ) ;
1467         insert_coefficient( eqline + 2 * j + 1 , endy[ j ] , -1 ) ;
1468
1469         insert_bound( eqline + 2 * j + 1 , Bound::EQT , 0 ) ;
1470     }
1471     return ;
1472 }
1473 }
```

References `insert_bound()`, and `insert_coefficient()`.

Referenced by `compute_correspondence_constraints()`.

16.4.3.43 insert_rhs_of_sleeve_corners_in_channel_constraints()

```
void channel::CurveBuilder::insert_rhs_of_sleeve_corners_in_channel_constraints (
    const size_t eqline,
    const size_t c,
    const std::vector< std::vector< double > > & nl,
    const std::vector< std::vector< double > > & nu ) [private]
```

Inserts into the matrix associated with the Linear Program (LP) the right-hand side values of the constraints that enforce a sleeve point to stay inside a c-segment of the channel. The type of each constraint (equality or inequality: ==, >= or <=) is also set here.

Parameters

<i>eqline</i>	A counter for the number of constraints.
<i>c</i>	An index identifying the c-segment the e-piece point belongs to.
<i>nl</i>	Array of Cartesian coordinates of outward normals to the supporting lines of the lower envelope segments of the channel.
<i>nu</i>	Array of Cartesian coordinates of outward normals to the supporting lines of the upper envelope segments of the channel.

Definition at line 2012 of file curvebuilder.cpp.

```
2018 {
2019     insert_bound( eqline      , Bound::LTE , _lxcoords[ c ] * nl[ c ][ 0 ] + _lycoords[ c ] * nl[ c ][ 1 ] )
2020 ;     insert_bound( eqline + 1 , Bound::LTE , _uxcoords[ c ] * nu[ c ][ 0 ] + _uycoords[ c ] * nu[ c ][ 1 ] )
2021 ;
2022     insert_bound( eqline + 2 , Bound::LTE , _lxcoords[ c ] * nl[ c ][ 0 ] + _lycoords[ c ] * nl[ c ][ 1 ] )
2023 ;     insert_bound( eqline + 3 , Bound::LTE , _uxcoords[ c ] * nu[ c ][ 0 ] + _uycoords[ c ] * nu[ c ][ 1 ] )
2024 ;
2025     return ;
2026 }
```

References `_lxcoords`, `_lycoords`, `_uxcoords`, `_uycoords`, and `insert_bound()`.

Referenced by `compute_sleeve_corners_in_channel_constraints()`.

16.4.3.44 insert_rhs_of_sleeve_inside_csegment_constraints()

```
void channel::CurveBuilder::insert_rhs_of_sleeve_inside_csegment_constraints (
    const size_t eqline,
    const size_t c,
    const std::vector< std::vector< double > > & ncsec ) [private]
```

Inserts into the matrix associated with the Linear Program (LP) the right-hand side values of the constraints that enforce one e-piece breakpoint to stay on the right side of a c-section of the channel, and another e-piece breakpoint to stay on the left side of the same c-section.

Parameters

<i>eqline</i>	A counter for the number of constraints.
<i>c</i>	An index identifying the c-segment the e-piece points belongs to.
<i>ncsec</i>	Array of Cartesian coordinates of normals (pointing to the left) to the supporting lines of the c-sections of the channel.

Definition at line 2047 of file curvebuilder.cpp.

```

2052 {
2053     insert_bound( eqline      , Bound::LTE , _lxcoords[ c ] * ncsec[ c ][ 0 ] + _lycoords[ c ] * ncsec[ c ][
1 ] ) ;
2054     insert_bound( eqline + 1 , Bound::LTE , _uxcoords[ c ] * ncsec[ c ][ 0 ] + _uycoords[ c ] * ncsec[ c ][
1 ] ) ;
2055
2056     insert_bound( eqline + 2 , Bound::GTE , _lxcoords[ c ] * ncsec[ c ][ 0 ] + _lycoords[ c ] * ncsec[ c ][
1 ] ) ;
2057     insert_bound( eqline + 3 , Bound::GTE , _uxcoords[ c ] * ncsec[ c ][ 0 ] + _uycoords[ c ] * ncsec[ c ][
1 ] ) ;
2058 }
```

References `_lxcoords`, `_lycoords`, `_uxcoords`, `_uycoords`, and `insert_bound()`.

Referenced by `compute_sleeve_inside_csegment_constraints()`.

16.4.3.45 is_curve_closed()

```
bool channel::CurveBuilder::is_curve_closed ( ) const [inline]
```

Returns the logic value true if the b-spline curve is closed, and the logic value false otherwise.

Returns

The logic value true if the b-spline curve is closed, and the logic value false otherwise.

Definition at line 218 of file curvebuilder.hpp.

```

219 {
220     return _closed ;
221 }
```

References `_closed`.

16.4.3.46 is_equality()

```
bool channel::CurveBuilder::is_equality (
    const size_t i ) const [inline]
```

Returns the logic value true if the type of the i-th constraint is equality; otherwise, returns the logic value false.

Parameters

<i>i</i>	The index of a constraint.
----------	----------------------------

Returns

The logic value true if the type of the *i*-th constraint is equality; otherwise, the logic value false is returned.

Definition at line 465 of file curvebuilder.hpp.

```

466     {
467         if ( _coefficients.empty() ) {
468             std::stringstream ss( std::stringstream::in | std::stringstream::out );
469             ss << "No constraint has been created so far" ;
470             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
471         }
472
473         if ( i >= _coefficients.size() ) {
474             std::stringstream ss( std::stringstream::in | std::stringstream::out );
475             ss << "Constraint index is out of range" ;
476             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
477         }
478
479 #ifdef DEBUGMODE
480     assert( _bounds.size() == _coefficients.size() ) ;
481     assert( _bounds.size() > i ) ;
482 #endif
483
484     return _bounds[ i ].get_type() == Bound::EQT ;
485 }

```

References `_bounds`, and `_coefficients`.

16.4.3.47 `is_greater_than_or_equal_to()`

```

bool channel::CurveBuilder::is_greater_than_or_equal_to (
    const size_t i ) const [inline]

```

Returns the logic value true if the *i*-th constraint is an inequality of the type greater than or equal to; otherwise, returns the logic value false.

Parameters

<i>i</i>	The index of a constraint.
----------	----------------------------

Returns

The logic value true if the *i*-th constraint is an inequality of the type greater than or equal to; otherwise, the logic value false is returned.

Definition at line 501 of file curvebuilder.hpp.

```

502     {
503         if ( _coefficients.empty() ) {
504             std::stringstream ss( std::stringstream::in | std::stringstream::out );
505             ss << "No constraint has been created so far" ;

```

```

506         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
507     }
508
509     if ( i >= _coefficients.size() ) {
510         std::stringstream ss( std::stringstream::in | std::stringstream::out ) ;
511         ss << "Constraint index is out of range" ;
512         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
513     }
514
515 #ifdef DEBUGMODE
516     assert( _bounds.size() == _coefficients.size() ) ;
517     assert( _bounds.size() > i ) ;
518 #endif
519
520     return _bounds[ i ].get_type() == Bound::GTE ;
521 }

```

References `_bounds`, and `_coefficients`.

16.4.3.48 is_less_than_or_equal_to()

```

bool channel::CurveBuilder::is_less_than_or_equal_to (
    const size_t i ) const [inline]

```

Returns the logic value true if the i-th constraint is an inequality of the type less than or equal to; otherwise, returns the logic value false.

Parameters

<i>i</i>	The index of a constraint.
----------	----------------------------

Returns

The logic value true if the i-th constraint is an inequality of the type less than or equal to; otherwise, the logic value false is returned.

Definition at line 538 of file `curvebuilder.hpp`.

```

539 {
540     if ( _coefficients.empty() ) {
541         std::stringstream ss( std::stringstream::in | std::stringstream::out ) ;
542         ss << "No constraint has been created so far" ;
543         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
544     }
545
546     if ( i >= _coefficients.size() ) {
547         std::stringstream ss( std::stringstream::in | std::stringstream::out ) ;
548         ss << "Constraint index is out of range" ;
549         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
550     }
551
552 #ifdef DEBUGMODE
553     assert( _bounds.size() == _coefficients.size() ) ;
554     assert( _bounds.size() > i ) ;
555 #endif
556
557     return _bounds[ i ].get_type() == Bound::LTE ;
558 }

```

References `_bounds`, and `_coefficients`.

16.4.3.49 minimum_value()

```
double channel::CurveBuilder::minimum_value ( ) const [inline]
```

Returns the optimal (minimum) value of the objective function of the instance of the channel problem as found by the LP solver.

Returns

The optimal (minimum) value of the objective function of the instance of the channel problem as found by the LP solver.

Definition at line 687 of file curvebuilder.hpp.

```
688 {
689     return _ofvalue ;
690 }
```

References `_ofvalue`.

16.4.3.50 set_up_lp_constraints()

```
void channel::CurveBuilder::set_up_lp_constraints (
    glp_prob * lp ) const [private]
```

Assemble the matrix of constraints of the linear program, and define the type (equality or inequality) and bounds on the constraints.

Parameters

<i>lp</i>	A pointer to the instance of the LP program.
-----------	--

Definition at line 2399 of file curvebuilder.cpp.

```
2400 {
2401     /*
2402      * Set up the bounds on the constraints of the problem.
2403      */
2404
2405     for ( size_t j = 0 ; j < _bounds.size() ; j++ ) {
2406 #ifdef DEBUGMODE
2407         assert( j == _bounds[ j ].get_row() ) ;
2408 #endif
2409
2410         int i = int( _bounds[ j ].get_row() + 1 ) ;
2411
2412         std::stringstream ss ( std::stringstream::in | std::stringstream::out ) ;
2413         ss << "c" << i ;
2414         glp_set_row_name( lp , i , ss.str().c_str() ) ;
2415
2416         double val = _bounds[ j ].get_value() ;
2417         if ( _bounds[ j ].get_type() == Bound::LTE ) {
2418             glp_set_row_bnds( lp , i , GLP_UP , 0 , val ) ;
2419         }
2420         else if ( _bounds[ j ].get_type() == Bound::GTE ) {
2421             glp_set_row_bnds( lp , i , GLP_LO , val , 0 ) ;
2422         }
2423         else {
```



```

2424         glp_set_row_bnds( lp , i , GLP_FX , val , val ) ;
2425     }
2426 }
2427
2428
2429 /*
2430  * Obtain the coefficients of the constraints of the problem.
2431  */
2432
2433 std::vector< int      > ia ; ia.push_back( 0 ) ; // GLPK starts indexing array \e ia at 1
2434 std::vector< int      > ja ; ja.push_back( 0 ) ; // GLPK starts indexing array \e ja at 1
2435 std::vector< double > ar ; ar.push_back( 0 ) ; // GLPK starts indexing array \e ar at 1
2436
2437 int h = 0 ;
2438 for ( size_t j = 0 ; j < _coefficients.size() ; j++ ) {
2439     for ( size_t k = 0 ; k < _coefficients[ j ].size() ; k++ ) {
2440 #ifdef DEBUGMODE
2441         assert( _coefficients[ j ][ k ].get_row() == j ) ;
2442 #endif
2443         ia.push_back( int( _coefficients[ j ][ k ].get_row() + 1 ) ) ;
2444         ja.push_back( int( _coefficients[ j ][ k ].get_col() + 1 ) ) ;
2445         ar.push_back( _coefficients[ j ][ k ].get_value() ) ;
2446         ++h ;
2447     }
2448 }
2449
2450 glp_load_matrix(
2451     lp ,
2452     h ,
2453     &ia[ 0 ] ,
2454     &ja[ 0 ] ,
2455     &ar[ 0 ]
2456 ) ;
2457
2458 return ;
2459 }

```

References `_bounds`, and `_coefficients`.

Referenced by `solve_lp()`.

16.4.3.51 set_up_objective_function()

```

void channel::CurveBuilder::set_up_objective_function (
    glp_prob * lp ) const [private]

```

Define the objective function of the linear program corresponding to the channel problem, which is a minimization problem.

Parameters

<i>lp</i>	A pointer to the instance of the LP program.
-----------	--

Definition at line 2630 of file `curvebuilder.cpp`.

```

2631 {
2632     //
2633     // Add the first two second difference bounds to the function.
2634     //
2635     for ( size_t i = 1 ; i < 3 ; i++ ) {
2636         for ( size_t l = 0 ; l < 2 ; l++ ) {
2637             for ( size_t v = 0 ; v < 2 ; v++ ) {
2638                 size_t c = compute_second_difference_column_index(
2639                                     0 ,
2640                                     i ,

```

```

2641                                     1 ,
2642                                     v
2643                                     ) ;
2644
2645         if ( l == 0 ) {
2646             glp_set_obj_coef( lp , int( c ) + 1 , -1 ) ;
2647         }
2648         else {
2649             glp_set_obj_coef( lp , int( c ) + 1 , 1 ) ;
2650         }
2651     }
2652 }
2653 }
2654
2655 //
2656 // Add the remaining second difference bounds to the function.
2657 //
2658 for ( size_t p = 1 ; p < _np ; p++ ) {
2659     for ( size_t l = 0 ; l < 2 ; l++ ) {
2660         for ( size_t v = 0 ; v < 2 ; v++ ) {
2661             size_t c = compute_second_difference_column_index(
2662                                     p ,
2663                                     2 ,
2664                                     1 ,
2665                                     v
2666                                     ) ;
2667
2668             if ( l == 0 ) {
2669                 glp_set_obj_coef( lp , int( c ) + 1 , -1 ) ;
2670             }
2671             else {
2672                 glp_set_obj_coef( lp , int( c ) + 1 , 1 ) ;
2673             }
2674         }
2675     }
2676 }
2677
2678 return ;
2679 }

```

References `_np`, and `compute_second_difference_column_index()`.

Referenced by `solve_lp()`.

16.4.3.52 set_up_structural_variables()

```

void channel::CurveBuilder::set_up_structural_variables (
    glp_prob * lp ) const [private]

```

Define lower and/or upper bounds on the structural variables of the linear program corresponding to the channel problem.

Parameters

<code>lp</code>	A pointer to the instance of the LP program.
-----------------	--

Definition at line 2473 of file `curvebuilder.cpp`.

```

2474 {
2475     //
2476     // Set up bounds for the first two second differences.
2477     //
2478     for ( size_t i = 1 ; i <= 2 ; i++ ) {
2479         for ( size_t l = 0 ; l < 2 ; l++ ) {
2480             for ( size_t v = 0 ; v < 2 ; v++ ) {
2481                 size_t c = compute_second_difference_column_index(

```

```

2482                                     0 ,
2483                                     i ,
2484                                     l ,
2485                                     v
2486                                     ) ;
2487     if ( l == 0 ) {
2488         std::stringstream ss ( std::stringstream::in | std::stringstream::out ) ;
2489         if ( v == 0 ) {
2490             ss << "mx" << i ;
2491         }
2492         else {
2493             ss << "my" << i ;
2494         }
2495         glp_set_col_name( lp , int( c ) + 1 , ss.str().c_str() ) ;
2496         glp_set_col_bnds( lp , int( c ) + 1 , GLP_UP , 0 , 0 ) ;
2497     }
2498     else {
2499         std::stringstream ss ( std::stringstream::in | std::stringstream::out ) ;
2500         if ( v == 0 ) {
2501             ss << "px" << i ;
2502         }
2503         else {
2504             ss << "py" << i ;
2505         }
2506         glp_set_col_name( lp , int( c ) + 1 , ss.str().c_str() ) ;
2507         glp_set_col_bnds( lp , int( c ) + 1 , GLP_LO , 0 , 0 ) ;
2508     }
2509 }
2510 }
2511 }
2512
2513 //
2514 // Set up bounds for the remaining second differences.
2515 //
2516 for ( size_t p = 1 ; p < _np ; p++ ) {
2517     for ( size_t l = 0 ; l < 2 ; l++ ) {
2518         for ( size_t v = 0 ; v < 2 ; v++ ) {
2519             size_t c = compute_second_difference_column_index(
2520                                     p ,
2521                                     2 ,
2522                                     l ,
2523                                     v
2524                                     ) ;
2525             if ( l == 0 ) {
2526                 std::stringstream ss ( std::stringstream::in | std::stringstream::out ) ;
2527                 if ( v == 0 ) {
2528                     ss << "mx" << p + 2 ;
2529                 }
2530                 else {
2531                     ss << "my" << p + 2 ;
2532                 }
2533                 glp_set_col_name( lp , int( c ) + 1 , ss.str().c_str() ) ;
2534                 glp_set_col_bnds( lp , int( c ) + 1 , GLP_UP , 0 , 0 ) ;
2535             }
2536             else {
2537                 std::stringstream ss ( std::stringstream::in | std::stringstream::out ) ;
2538                 if ( v == 0 ) {
2539                     ss << "px" << p + 2 ;
2540                 }
2541                 else {
2542                     ss << "py" << p + 2 ;
2543                 }
2544                 glp_set_col_name( lp , int( c ) + 1 , ss.str().c_str() ) ;
2545                 glp_set_col_bnds( lp , int( c ) + 1 , GLP_LO , 0 , 0 ) ;
2546             }
2547         }
2548     }
2549 }
2550
2551 //
2552 // Set up bounds for the first four control points.
2553 //
2554 for ( size_t i = 0 ; i < 4 ; i++ ) {
2555     for ( size_t v = 0 ; v < 2 ; v++ ) {
2556         size_t c = compute_control_value_column_index(
2557                                     0 ,
2558                                     i ,
2559                                     v
2560                                     ) ;
2561
2562         std::stringstream ss ( std::stringstream::in | std::stringstream::out ) ;

```

```

2563         if ( v == 0 ) {
2564             ss << "x" << i + 1 ;
2565         }
2566         else {
2567             ss << "y" << i + 1 ;
2568         }
2569         glp_set_col_name( lp , int( c ) + 1 , ss.str().c_str() ) ;
2570         glp_set_col_bnds( lp , int( c ) + 1 , GLP_FR , 0 , 0 ) ;
2571     }
2572 }
2573
2574 for ( size_t p = 1 ; p < _np ; p++ ) {
2575     for ( size_t v = 0 ; v < 2 ; v++ ) {
2576         size_t c = compute_control_value_column_index(
2577             p ,
2578             3 ,
2579             v
2580         ) ;
2581
2582         std::stringstream ss ( std::stringstream::in | std::stringstream::out ) ;
2583         if ( v == 0 ) {
2584             ss << "x" << p + 4 ;
2585         }
2586         else {
2587             ss << "y" << p + 4 ;
2588         }
2589         glp_set_col_name( lp , int( c ) + 1 , ss.str().c_str() ) ;
2590         glp_set_col_bnds( lp , int( c ) + 1 , GLP_FR , 0 , 0 ) ;
2591     }
2592 }
2593
2594 size_t s = compute_index_of_endpoint_barycentric_coordinate( 0 ) ;
2595 std::stringstream ss ( std::stringstream::in | std::stringstream::out ) ;
2596 ss << "st" ;
2597 glp_set_col_name( lp , int( s ) + 1 , ss.str().c_str() ) ;
2598 glp_set_col_bnds( lp , int( s ) + 1 , GLP_DB , 0.40 , 0.60 ) ;
2599
2600 if ( !_closed ) {
2601     size_t e = compute_index_of_endpoint_barycentric_coordinate( 1 ) ;
2602     std::stringstream ss2 ( std::stringstream::in | std::stringstream::out ) ;
2603     ss2 << "en" ;
2604     glp_set_col_name( lp , int( e ) + 1 , ss2.str().c_str() ) ;
2605     glp_set_col_bnds( lp , int( e ) + 1 , GLP_DB , 0.40 , 0.60 ) ;
2606 }
2607
2608 for ( size_t i = 1 ; i < _nc ; i++ ) {
2609     size_t corner_coord = compute_index_of_corner_barycentric_coordinate( i ) ;
2610     std::stringstream ss ( std::stringstream::in | std::stringstream::out ) ;
2611     ss << "co" << i ;
2612     glp_set_col_name( lp , int( corner_coord ) + 1 , ss.str().c_str() ) ;
2613     glp_set_col_bnds( lp , int( corner_coord ) + 1 , GLP_DB , 0.40 , 0.60 ) ;
2614 }
2615
2616 return ;
2617 }

```

References `_closed`, `_nc`, `_np`, `compute_control_value_column_index()`, `compute_index_of_corner_barycentric_coordinate()`, `compute_index_of_endpoint_barycentric_coordinate()`, and `compute_second_difference_column_index()`.

Referenced by `solve_lp()`.

16.4.3.53 solve_lp()

```

int channel::CurveBuilder::solve_lp (
    const size_t rows,
    const size_t cols ) [private]

```

Solves the linear program corresponding to the channel problem.

Parameters

<i>rows</i>	The number of constraints of the linear program.
<i>cols</i>	The number of unknowns of the linear program.

Returns

The code returned by the LP solver to indicate the status of the computation of the solution of the linear program.

Definition at line 2321 of file curvebuilder.cpp.

```

2325 {
2326     /*
2327      * Create the LP problem.
2328      */
2329     glp_prob* lp = glp_create_prob() ;
2330
2331     /*
2332      * Set up the number of constraints and structural variables.
2333      */
2334     glp_add_rows( lp , int( rows ) ) ;
2335     glp_add_cols( lp , int( cols ) ) ;
2336
2337     /*
2338      * Set the problem as a minimization one.
2339      */
2340     glp_set_obj_dir( lp , GLP_MIN ) ;
2341
2342     /*
2343      * Set up the constraints of the problem.
2344      */
2345     set_up_lp_constraints( lp ) ;
2346
2347     /*
2348      * Define bounds on the structural variables of the problem.
2349      */
2350     set_up_structural_variables( lp ) ;
2351
2352     /*
2353      * Define objective function.
2354      */
2355     set_up_objective_function( lp ) ;
2356
2357     /*
2358      * Set parameters of the solver.
2359      */
2360     glp_smcp param ;
2361     glp_init_smcp( &param ) ;
2362
2363     param.msg_lev = GLP_MSG_OFF ;
2364     param.presolve = GLP_ON ;
2365
2366     /*
2367      * Call the solver.
2368      */
2369
2370     int res = glp_simplex( lp , &param ) ;
2371
2372     if ( res == 0 ) {
2373         /*
2374          * Get the solver result information.
2375          */
2376         get_lp_solver_result_information( lp ) ;
2377     }
2378
2379     /*
2380      * Release memory held by the solver.
2381      */
2382     glp_delete_prob( lp ) ;
2383
2384     return res ;
2385 }
```

References `get_lp_solver_result_information()`, `set_up_lp_constraints()`, `set_up_objective_function()`, and `set_up_structural_variables()`.

Referenced by `build()`.

The documentation for this class was generated from the following files:

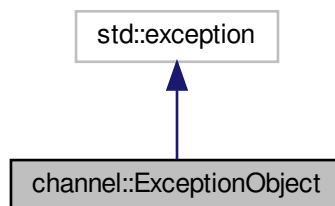
- [curvebuilder.hpp](#)
- [curvebuilder.cpp](#)

16.5 channel::ExceptionObject Class Reference

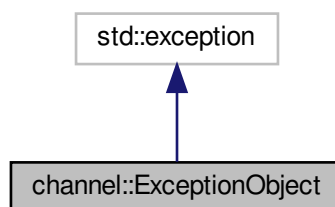
This class extends class *exception* of STL and provides us with a customized way of handling exceptions and showing error messages.

```
#include <exceptionobject.hpp>
```

Inheritance diagram for `channel::ExceptionObject`:



Collaboration diagram for `channel::ExceptionObject`:



Public Member Functions

- [ExceptionObject](#) ()
Creates an instance of this class.
- [ExceptionObject](#) (const char *file, unsigned ln)
Creates an instance of this class.
- [ExceptionObject](#) (const char *file, unsigned int ln, const char *desc)
Creates an instance of this class.
- [ExceptionObject](#) (const char *file, unsigned ln, const char *desc, const char *loc)
Creates an instance of this class.
- [ExceptionObject](#) (const [ExceptionObject](#) &xpt)
Clones an instance of this class.
- virtual [~ExceptionObject](#) () throw ()
Releases the memory held by an instance of this class.
- [ExceptionObject](#) & [operator=](#) (const [ExceptionObject](#) &xpt)
Overloads the assignment operator.
- virtual const char * [get_name_of_class](#) () const
Returns the name of this class.
- virtual void [set_location](#) (const std::string &s)
Assigns a location to this exception.
- virtual void [set_location](#) (const char *s)
Assigns a location to this exception.
- virtual void [set_description](#) (const std::string &s)
Assigns a description to this exception.
- virtual void [set_description](#) (const char *s)
Assigns a description to this exception.
- virtual const char * [get_location](#) () const
Returns the location where this exception occurs.
- virtual const char * [get_description](#) () const
Returns a description of the error that caused this exception.
- virtual const char * [get_file](#) () const
Returns the name of the file containing the line that caused the exception.
- virtual unsigned [get_line](#) () const
Returns the line that caused this exception.
- virtual const char * [what](#) () const throw ()
Returns a description of the error that caused this exception.

Protected Attributes

- std::string [_location](#)
Location of the error in the line that caused the exception.
- std::string [_description](#)
Description of the error.
- std::string [_file](#)
File where the error occurred.
- unsigned [_line](#)
Line of the file where the error occurred.

16.5.1 Detailed Description

This class extends class *exception* of STL and provides us with a customized way of handling exceptions and showing error messages.

Definition at line 75 of file `exceptionobject.hpp`.

16.5.2 Constructor & Destructor Documentation

16.5.2.1 ExceptionObject() [1/4]

```
channel::ExceptionObject::ExceptionObject (
    const char * file,
    unsigned ln ) [inline]
```

Creates an instance of this class.

Parameters

<i>file</i>	A pointer to the name of the file where the exception occurred.
<i>ln</i>	Number of the line containing the instruction that caused the exception.

Definition at line 125 of file `exceptionobject.hpp`.

```
126 :
127     _location( "Unknown" ) ,
128     _description( "Unknown" ) ,
129     _file( file ) ,
130     _line( ln )
131 {
132 }
```

16.5.2.2 ExceptionObject() [2/4]

```
channel::ExceptionObject::ExceptionObject (
    const char * file,
    unsigned int ln,
    const char * desc ) [inline]
```

Creates an instance of this class.

Parameters

<i>file</i>	A pointer to the name of the file where the exception occurred.
<i>ln</i>	Number of the line containing the instruction that caused the exception.
<i>desc</i>	A pointer to a description of the error that caused the exception.

Definition at line 148 of file exceptionobject.hpp.

```

149      :
150      _location( "Unknown" ) ,
151      _description( desc ) ,
152      _file( file ) ,
153      _line( ln )
154      {
155      }
```

16.5.2.3 ExceptionObject() [3/4]

```

channel::ExceptionObject::ExceptionObject (
    const char * file,
    unsigned ln,
    const char * desc,
    const char * loc ) [inline]
```

Creates an instance of this class.

Parameters

<i>file</i>	A pointer to the name of the file where the exception occurred.
<i>ln</i>	Number of the line containing the instruction that caused the exception.
<i>desc</i>	A pointer to a description of the error that caused the exception.
<i>loc</i>	A pointer to the location of the exception inside the line where it occurred.

Definition at line 173 of file exceptionobject.hpp.

```

174      :
175      _location( loc ) ,
176      _description( desc ) ,
177      _file( file ) ,
178      _line( ln )
179      {
180      }
```

16.5.2.4 ExceptionObject() [4/4]

```

channel::ExceptionObject::ExceptionObject (
    const ExceptionObject & xpt ) [inline]
```

Clones an instance of this class.

Parameters

<i>xpt</i>	A reference to another instance of this class.
------------	--

Definition at line 191 of file exceptionobject.hpp.

```

191      : exception()
```

```
192     {  
193         _location = xpt._location ;  
194         _description = xpt._description ;  
195         _file = xpt._file ;  
196         _line = xpt._line ;  
197     }
```

References `_description`, `_file`, `_line`, and `_location`.

16.5.3 Member Function Documentation

16.5.3.1 `get_description()`

```
const char * channel::ExceptionObject::get_description ( ) const [inline], [virtual]
```

Returns a description of the error that caused this exception.

Returns

A description of the error that caused this exception.

Definition at line 321 of file `exceptionobject.hpp`.

```
322     {  
323         return _description.c_str() ;  
324     }
```

References `_description`.

16.5.3.2 `get_file()`

```
const char * channel::ExceptionObject::get_file ( ) const [inline], [virtual]
```

Returns the name of the file containing the line that caused the exception.

Returns

The name of the file containing the line that caused the exception.

Definition at line 337 of file `exceptionobject.hpp`.

```
338     {  
339         return _file.c_str() ;  
340     }
```

References `_file`.

16.5.3.3 get_line()

```
unsigned channel::ExceptionObject::get_line ( ) const [inline], [virtual]
```

Returns the line that caused this exception.

Returns

The line that caused this exception.

Definition at line 351 of file exceptionobject.hpp.

```
352 {  
353     return _line ;  
354 }
```

References `_line`.

16.5.3.4 get_location()

```
const char * channel::ExceptionObject::get_location ( ) const [inline], [virtual]
```

Returns the location where this exception occurs.

Returns

The location where this exception occurs.

Definition at line 306 of file exceptionobject.hpp.

```
307 {  
308     return _location.c_str() ;  
309 }
```

References `_location`.

16.5.3.5 get_name_of_class()

```
const char * channel::ExceptionObject::get_name_of_class ( ) const [inline], [virtual]
```

Returns the name of this class.

Returns

The name of this class.

Definition at line 236 of file exceptionobject.hpp.

```
237 {  
238     return "ExceptionObject" ;  
239 }
```

16.5.3.6 set_description() [1/2]

```
void channel::ExceptionObject::set_description (  
    const char * s ) [inline], [virtual]
```

Assigns a description to this exception.

Parameters

s	A pointer to a string containing the description.
----------	---

Definition at line 292 of file exceptionobject.hpp.

```
293     {  
294         _description = s ;  
295     }
```

References `_description`.

16.5.3.7 set_description() [2/2]

```
void channel::ExceptionObject::set_description (  
    const std::string & s ) [inline], [virtual]
```

Assigns a description to this exception.

Parameters

s	A string containing the description.
----------	--------------------------------------

Definition at line 278 of file exceptionobject.hpp.

```
279     {  
280         _description = s ;  
281     }
```

References `_description`.

16.5.3.8 set_location() [1/2]

```
void channel::ExceptionObject::set_location (  
    const char * s ) [inline], [virtual]
```

Assigns a location to this exception.

Parameters

s	A pointer to a string containing the location.
----------	--

Definition at line 264 of file exceptionobject.hpp.

```
265     {  
266         _location = s ;  
267     }
```

References `_location`.

16.5.3.9 `set_location()` [2/2]

```
void channel::ExceptionObject::set_location (  
    const std::string & s ) [inline], [virtual]
```

Assigns a location to this exception.

Parameters

<code>s</code>	A string containing the location.
----------------	-----------------------------------

Definition at line 250 of file exceptionobject.hpp.

```
251     {  
252         _location = s ;  
253     }
```

References `_location`.

16.5.3.10 `what()`

```
const char * channel::ExceptionObject::what ( ) const throw ( ) [inline], [virtual]
```

Returns a description of the error that caused this exception.

Returns

A description of the error that caused this exception.

Definition at line 366 of file exceptionobject.hpp.

```
367     {  
368         return _description.c_str() ;  
369     }
```

References `_description`.

The documentation for this class was generated from the following file:

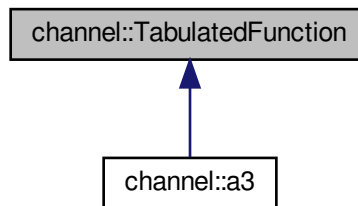
- [exceptionobject.hpp](#)

16.6 channel::TabulatedFunction Class Reference

This class represents two-sided, piecewise linear enclosures of a set of $(d - 1)$ polynomial functions of degree d in Bézier form. The enclosures must be made available by implementing a pure virtual method in derived classes.

```
#include <tabulatedfunction.hpp>
```

Inheritance diagram for channel::TabulatedFunction:



Public Member Functions

- [TabulatedFunction](#) ()
Creates an instance of this class.
- virtual [~TabulatedFunction](#) ()
Releases the memory held by an instance of this class.
- virtual double [alower](#) (const size_t i, const double u) const =0
Evaluates the piecewise linear function corresponding to the lower enclosure of the i -th tabulated function at a point in $[0, 1]$.
- virtual double [aupper](#) (const size_t i, const double u) const =0
Evaluates the piecewise linear function corresponding to the upper enclosure of the i -th tabulated function at a point in $[0, 1]$.
- virtual double [a](#) (const size_t i, const double u) const =0
Computes the value of the i -th polynomial function a at a given point of the interval $[0, 1]$ of the real line.
- virtual unsigned [degree](#) () const =0
Returns the degree of the tabulated functions.

16.6.1 Detailed Description

This class represents two-sided, piecewise linear enclosures of a set of $(d - 1)$ polynomial functions of degree d in Bézier form. The enclosures must be made available by implementing a pure virtual method in derived classes.

Attention

This class is based on several papers surveyed in

J. Peters.
Efficient one-sided linearization of spline geometry.
Proceeding of the 10th International Conference on
Mathematics of Surfaces, Leeds, UK, September 15-17,
2003, p. 297-319. (Lecture Notes in Computer
Science, volume 2768, Eds. M.J. Wilson and
R.R. Martin).

Definition at line 71 of file tabulatedfunction.hpp.

16.6.2 Member Function Documentation**16.6.2.1 a()**

```
double channel::TabulatedFunction::a (
    const size_t i,
    const double u ) const [pure virtual]
```

Computes the value of the i -th polynomial function a at a given point of the interval $[0, 1]$ of the real line.

Parameters

i	The index of the i -th polynomial function.
u	A parameter point in the interval $[0, 1]$.

Returns

The value of the i -th polynomial function a at a given point u of the interval $[0, 1]$ of the real line.

Implemented in [channel::a3](#).

16.6.2.2 alower()

```
double channel::TabulatedFunction::alower (
    const size_t i,
    const double u ) const [pure virtual]
```

Evaluates the piecewise linear function corresponding to the lower enclosure of the i -th tabulated function at a point in $[0, 1]$.

Parameters

i	The index of the i -th polynomial function.
u	A value in the interval $[0, 1]$.

Returns

The value of the piecewise linear function corresponding to the lower enclosure of the i -th tabulated function at a point in $[0, 1]$.

Implemented in [channel::a3](#).

Referenced by `channel::CurveBuilder::evaluate_bounding_polynomial()`.

16.6.2.3 upper()

```
double channel::TabulatedFunction::upper (
    const size_t i,
    const double u ) const [pure virtual]
```

Evaluates the piecewise linear function corresponding to the upper enclosure of the i -th tabulated function at a point in $[0, 1]$.

Parameters

i	The index of the i -th polynomial function.
u	A value in the interval $[0, 1]$.

Returns

The value of the piecewise linear function corresponding to the upper enclosure of the i -th tabulated function at a point in $[0, 1]$.

Implemented in [channel::a3](#).

Referenced by `channel::CurveBuilder::evaluate_bounding_polynomial()`.

16.6.2.4 degree()

```
unsigned channel::TabulatedFunction::degree ( ) const [pure virtual]
```

Returns the degree of the tabulated functions.

Returns

The degree of the tabulated functions.

Implemented in [channel::a3](#).

The documentation for this class was generated from the following file:

- [tabulatedfunction.hpp](#)

Chapter 17

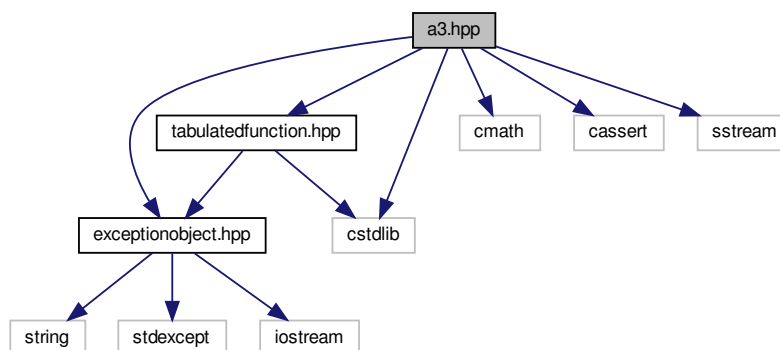
File Documentation

17.1 a3.hpp File Reference

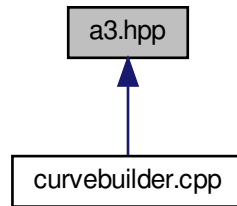
Definition of a class for representing piecewise linear enclosures of certain cubic polynomial functions in Bézier form.

```
#include "exceptionobject.hpp"  
#include "tabulatedfunction.hpp"  
#include <cmath>  
#include <cassert>  
#include <sstream>  
#include <cstdlib>
```

Include dependency graph for a3.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [channel::a3](#)

This class represents two-sided, piecewise linear enclosures for two polynomial functions of degree 3 in Bézier form.

Namespaces

- [channel](#)

The namespace channel contains the definition and implementation of a set of classes for threading a cubic b-spline curve into a given planar channel delimited by two polygonal chains.

17.1.1 Detailed Description

Definition of a class for representing piecewise linear enclosures of certain cubic polynomial functions in Bézier form.

Author

Marcelo Ferreira Siqueira
Universidade Federal do Rio Grande do Norte,
Departamento de Matemática,
mfsiqueira at mat (dot) ufrn (dot) br

Version

1.0

Date

March 2016

Attention

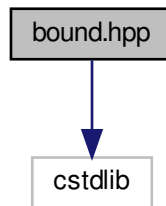
This program is distributed WITHOUT ANY WARRANTY, and it may be freely redistributed under the condition that the copyright notices are not removed, and no compensation is received. Private, research, and institutional use is free. Distribution of this code as part of a commercial system is permissible ONLY BY DIRECT ARRANGEMENT WITH THE AUTHOR.

17.2 bound.hpp File Reference

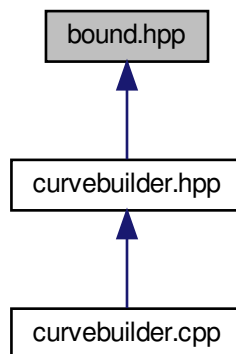
Definition of a class for representing the type of a linear constraint (i.e., equality or inequality) and its right-hand side: a real number.

```
#include <cstdlib>
```

Include dependency graph for bound.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [channel::Bound](#)

This class represents the type of a constraint (i.e., equality or inequality) and the value of its right-hand side: a real number.

Namespaces

- [channel](#)

The namespace `channel` contains the definition and implementation of a set of classes for threading a cubic b-spline curve into a given planar channel delimited by two polygonal chains.

17.2.1 Detailed Description

Definition of a class for representing the type of a linear constraint (i.e., equality or inequality) and its right-hand side: a real number.

Author

Marcelo Ferreira Siqueira
Universidade Federal do Rio Grande do Norte,
Departamento de Matemática,
mfsiqueira at mat (dot) ufrn (dot) br

Version

1.0

Date

March 2016

Attention

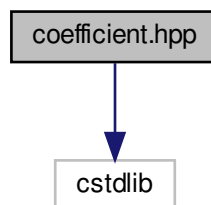
This program is distributed WITHOUT ANY WARRANTY, and it may be freely redistributed under the condition that the copyright notices are not removed, and no compensation is received. Private, research, and institutional use is free. Distribution of this code as part of a commercial system is permissible ONLY BY DIRECT ARRANGEMENT WITH THE AUTHOR.

17.3 coefficient.hpp File Reference

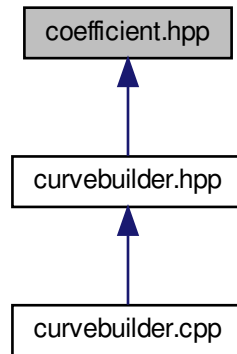
Definition of a class for representing a nonzero coefficient of an unknown of a constraint (inequality or equality) of a linear program instance.

```
#include <cstdlib>
```

Include dependency graph for `coefficient.hpp`:



This graph shows which files directly or indirectly include this file:



Classes

- class [channel::Coefficient](#)

This class represents a nonzero coefficient of an unknown of a constraint (inequality or equality) of a linear program instance.

Namespaces

- [channel](#)

The namespace channel contains the definition and implementation of a set of classes for threading a cubic b-spline curve into a given planar channel delimited by two polygonal chains.

17.3.1 Detailed Description

Definition of a class for representing a nonzero coefficient of an unknown of a constraint (inequality or equality) of a linear program instance.

Author

Marcelo Ferreira Siqueira
Universidade Federal do Rio Grande do Norte,
Departamento de Matemática,
mfsiqueira at mat (dot) ufrn (dot) br

Version

1.0

Date

March 2016

Attention

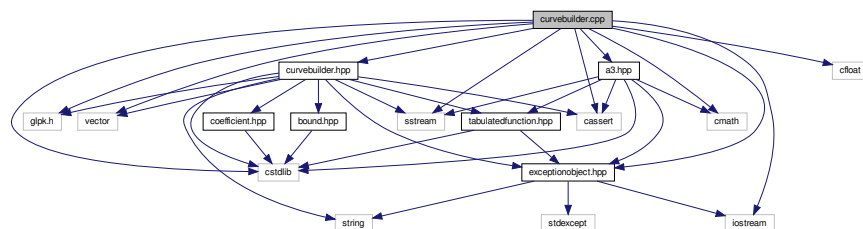
This program is distributed WITHOUT ANY WARRANTY, and it may be freely redistributed under the condition that the copyright notices are not removed, and no compensation is received. Private, research, and institutional use is free. Distribution of this code as part of a commercial system is permissible ONLY BY DIRECT ARRANGEMENT WITH THE AUTHOR.

17.4 curvebuilder.cpp File Reference

Implementation of a class for threading a b-spline curve of degree 3 through a planar channel defined by a pair of polygonal chains.

```
#include "curvebuilder.hpp"
#include "exceptionobject.hpp"
#include "a3.hpp"
#include "glpk.h"
#include <cmath>
#include <cassert>
#include <sstream>
#include <iostream>
#include <vector>
#include <cstdio>
#include <cstdlib>
```

Include dependency graph for curvebuilder.cpp:



Namespaces

- [channel](#)

The namespace channel contains the definition and implementation of a set of classes for threading a cubic b-spline curve into a given planar channel delimited by two polygonal chains.

17.4.1 Detailed Description

Implementation of a class for threading a b-spline curve of degree 3 through a planar channel defined by a pair of polygonal chains.

Author

Marcelo Ferreira Siqueira
Universidade Federal do Rio Grande do Norte,
Departamento de Matemática,
mfsiqueira at mat (dot) ufrn (dot) br

Version

1.0

Date

May 2016

Attention

This program is distributed WITHOUT ANY WARRANTY, and it may be freely redistributed under the condition that the copyright notices are not removed, and no compensation is received. Private, research, and institutional use is free. Distribution of this code as part of a commercial system is permissible ONLY BY DIRECT ARRANGEMENT WITH THE AUTHOR.

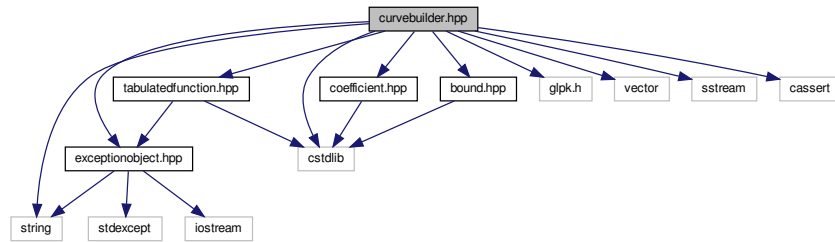
17.5 curvebuilder.hpp File Reference

Definition of a class for threading a b-spline curve of degree 3 through a planar channel defined by a pair of polygonal chains.

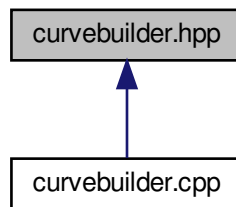
```
#include "exceptionobject.hpp"
#include "tabulatedfunction.hpp"
#include "coefficient.hpp"
#include "bound.hpp"
#include "glpk.h"
#include <vector>
#include <string>
#include <sstream>
#include <cassert>
```

```
#include <cstdlib>
```

Include dependency graph for curvebuilder.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [channel::CurveBuilder](#)

This class provides methods for threading a cubic b-spline curve through a planar channel delimited by a pair of polygonal chains.

Namespaces

- [channel](#)

The namespace channel contains the definition and implementation of a set of classes for threading a cubic b-spline curve into a given planar channel delimited by two polygonal chains.

17.5.1 Detailed Description

Definition of a class for threading a b-spline curve of degree 3 through a planar channel defined by a pair of polygonal chains.

Author

Marcelo Ferreira Siqueira
Universidade Federal do Rio Grande do Norte,
Departamento de Matemática,
mfsiqueira at mat (dot) ufrn (dot) br

Version

1.0

Date

May 2016

Attention

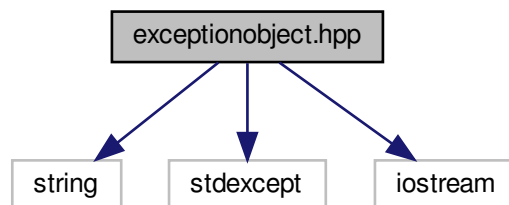
This program is distributed WITHOUT ANY WARRANTY, and it may be freely redistributed under the condition that the copyright notices are not removed, and no compensation is received. Private, research, and institutional use is free. Distribution of this code as part of a commercial system is permissible ONLY BY DIRECT ARRANGEMENT WITH THE AUTHOR.

17.6 exceptionobject.hpp File Reference

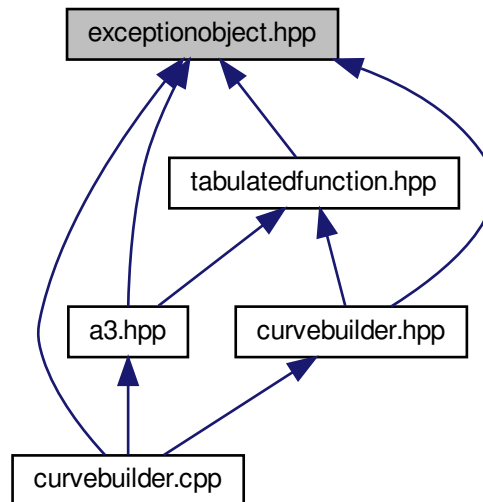
Definition of a class for handling exceptions.

```
#include <string>
#include <stdexcept>
#include <iostream>
```

Include dependency graph for exceptionobject.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [channel::ExceptionObject](#)

This class extends class exception of STL and provides us with a customized way of handling exceptions and showing error messages.

Namespaces

- [channel](#)

The namespace channel contains the definition and implementation of a set of classes for threading a cubic b-spline curve into a given planar channel delimited by two polygonal chains.

Macros

- #define [treat_exception\(e\)](#)

Prints out the description of the error that caused an exception as well as the file containing the instruction that threw the exception and the line of the instruction in the file.

17.6.1 Detailed Description

Definition of a class for handling exceptions.

Author

Marcelo Ferreira Siqueira
Universidade Federal do Rio Grande do Norte,
Departamento de Matemática,
mfsiqueira at mat (dot) ufrn (dot) br

Version

1.0

Date

March 2016

Attention

This program is distributed WITHOUT ANY WARRANTY, and it may be freely redistributed under the condition that the copyright notices are not removed, and no compensation is received. Private, research, and institutional use is free. Distribution of this code as part of a commercial system is permissible ONLY BY DIRECT ARRANGEMENT WITH THE AUTHOR.

17.6.2 Macro Definition Documentation

17.6.2.1 treat_exception

```
#define treat_exception(  
    e )
```

Value:

```
std::cerr << std::endl \  
    << "Exception: " << e.get_description() << std::endl \  
    << "File: "      << e.get_file()      << std::endl \  
    << "Line: "      << e.get_line()      << std::endl \  
    << std::endl ;
```

Prints out the description of the error that caused an exception as well as the file containing the instruction that threw the exception and the line of the instruction in the file.

Parameters

<i>e</i>	An exception.
----------	---------------

Definition at line 41 of file exceptionobject.hpp.

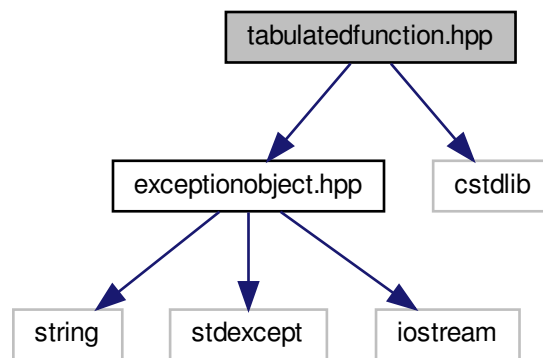
17.7 tabulatedfunction.hpp File Reference

Definition of an abstract class for representing piecewise linear enclosures of certain polynomial functions of arbitrary degree.

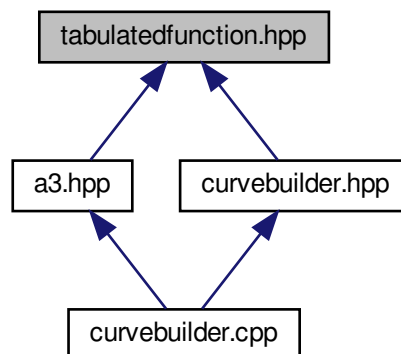
```
#include "exceptionobject.hpp"
```

```
#include <cstdlib>
```

Include dependency graph for tabulatedfunction.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [channel::TabulatedFunction](#)

This class represents two-sided, piecewise linear enclosures of a set of $(d - 1)$ polynomial functions of degree d in Bézier form. The enclosures must be made available by implementing a pure virtual method in derived classes.

Namespaces

- [channel](#)

The namespace channel contains the definition and implementation of a set of classes for threading a cubic b-spline curve into a given planar channel delimited by two polygonal chains.

17.7.1 Detailed Description

Definition of an abstract class for representing piecewise linear enclosures of certain polynomial functions of arbitrary degree.

Author

Marcelo Ferreira Siqueira
Universidade Federal do Rio Grande do Norte,
Departamento de Matemática,
mfsiqueira at mat (dot) ufrn (dot) br

Version

1.0

Date

March 2016

Attention

This program is distributed WITHOUT ANY WARRANTY, and it may be freely redistributed under the condition that the copyright notices are not removed, and no compensation is received. Private, research, and institutional use is free. Distribution of this code as part of a commercial system is permissible ONLY BY DIRECT ARRANGEMENT WITH THE AUTHOR.

