

Threading B-Splines Through 2D Channels

1.0

Generated by Doxygen 1.8.10

Tue Jul 12 2016 17:17:38

Contents

1	The Channel2D Library Documentation	1
2	Introduction	3
3	Installing and compiling the library	7
4	The CurveBuilder class API	11
5	Using the library API	13
6	Examples and file formats	17
7	License	21
8	Acknowledgements	23
9	Module Index	25
9.1	Modules	25
10	Namespace Index	27
10.1	Namespace List	27
11	Hierarchical Index	29
11.1	Class Hierarchy	29
12	Class Index	31
12.1	Class List	31
13	File Index	33
13.1	File List	33
14	Module Documentation	35
14.1	Namespace channel.	35
14.1.1	Detailed Description	35

15 Namespace Documentation	37
15.1 channel Namespace Reference	37
15.1.1 Detailed Description	37
16 Class Documentation	39
16.1 channel::a3 Class Reference	39
16.1.1 Detailed Description	41
16.1.2 Member Function Documentation	41
16.1.2.1 a(const size_t i, const double u) const	41
16.1.2.2 a1(const double u) const	42
16.1.2.3 a1lower(const double u) const	42
16.1.2.4 a1upper(const double u) const	43
16.1.2.5 alower(const size_t i, const double u) const	44
16.1.2.6 aupper(const size_t i, const double u) const	45
16.1.2.7 degree() const	46
16.1.2.8 h(const double u) const	46
16.2 channel::Bound Class Reference	47
16.2.1 Detailed Description	48
16.2.2 Constructor & Destructor Documentation	48
16.2.2.1 Bound(const CONSTRAINTYPE type, const double value, const size_t row)	48
16.2.2.2 Bound(const Bound &b)	48
16.2.3 Member Function Documentation	49
16.2.3.1 get_row() const	49
16.2.3.2 get_type() const	49
16.2.3.3 get_value() const	49
16.3 channel::Coefficient Class Reference	50
16.3.1 Detailed Description	50
16.3.2 Constructor & Destructor Documentation	50
16.3.2.1 Coefficient(const size_t row, const size_t col, double value)	50
16.3.2.2 Coefficient(const Coefficient &c)	51
16.3.3 Member Function Documentation	51
16.3.3.1 get_col() const	51
16.3.3.2 get_row() const	51
16.3.3.3 get_value() const	52
16.4 channel::CurveBuilder Class Reference	52
16.4.1 Detailed Description	57
16.4.2 Constructor & Destructor Documentation	57

16.4.2.1	CurveBuilder(size_t np, size_t nc, bool closed, double *lx, double *ly, double *ux, double *uy)	57
16.4.2.2	CurveBuilder(const CurveBuilder &b)	58
16.4.3	Member Function Documentation	59
16.4.3.1	build(int &error)	59
16.4.3.2	compute_channel_corners_outside_sleeve_constraints(size_t &eqline)	60
16.4.3.3	compute_control_value_column_index(const size_t p, const size_t i, const size_t v) const	61
16.4.3.4	compute_correspondence_constraints(size_t &eqline)	62
16.4.3.5	compute_index_of_corner_barycentric_coordinate(const size_t i) const	64
16.4.3.6	compute_index_of_endpoint_barycentric_coordinate(const size_t i) const	64
16.4.3.7	compute_min_max_constraints(size_t &eqline)	65
16.4.3.8	compute_normal_to_csection(const size_t s, double &nx, double &ny) const	67
16.4.3.9	compute_normal_to_lower_envelope(const size_t s, double &nx, double &ny) const	67
16.4.3.10	compute_normal_to_upper_envelope(const size_t s, double &nx, double &ny) const	68
16.4.3.11	compute_second_difference_column_index(const size_t p, const size_t i, const size_t l, const size_t v) const	68
16.4.3.12	compute_sleeve_corners_in_channel_constraints(size_t &eqline)	69
16.4.3.13	compute_sleeve_inside_csegment_constraints(size_t &eqline)	72
16.4.3.14	evaluate_bounding_polynomial(const size_t j, const double t, double &lower, double &upper)	76
16.4.3.15	get_bound_of_ith_constraint(const size_t i) const	77
16.4.3.16	get_coefficient_identifier(const size_t i, const size_t j) const	77
16.4.3.17	get_coefficient_value(const size_t i, const size_t j) const	78
16.4.3.18	get_control_value(const size_t i, const size_t v) const	78
16.4.3.19	get_degree() const	79
16.4.3.20	get_lower_bound_on_second_difference_value(const size_t p, const size_t i, const size_t v) const	79
16.4.3.21	get_lp_solver_result_information(glp_prob *lp)	80
16.4.3.22	get_number_of_coefficients_in_the_ith_constraint(const size_t i) const	82
16.4.3.23	get_number_of_constraints() const	83
16.4.3.24	get_number_of_control_points() const	83
16.4.3.25	get_number_of_csegments() const	84
16.4.3.26	get_number_of_segments() const	84
16.4.3.27	get_solver_error_message(int error)	84
16.4.3.28	get_upper_bound_on_second_difference_value(const size_t p, const size_t i, const size_t v) const	85
16.4.3.29	insert_bound(const size_t eqline, const Bound::CONSTRAINTTYPE type, const double value)	86

16.4.3.30	<code>insert_coefficient(const size_t eqline, const size_t index, const double value)</code>	87
16.4.3.31	<code>insert_csegment_constraint(const size_t eqline, const double lower, const double upper, const size_t sdlo, const size_t sdup, const double normal)</code>	87
16.4.3.32	<code>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)</code>	88
16.4.3.33	<code>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)</code>	89
16.4.3.34	<code>insert_extreme_point_correspondence_constraint(const size_t eqline, const std::vector< size_t > &col, const std::vector< double > &val, const double rhs)</code>	90
16.4.3.35	<code>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)</code>	90
16.4.3.36	<code>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)</code>	92
16.4.3.37	<code>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)</code>	93
16.4.3.38	<code>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)</code>	94
16.4.3.39	<code>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)</code>	95
16.4.3.40	<code>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)</code>	96
16.4.3.41	<code>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)</code>	97
16.4.3.42	<code>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)</code>	98
16.4.3.43	<code>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)</code>	99
16.4.3.44	<code>insert_rhs_of_sleeve_inside_csegment_constraints(const size_t eqline, const size_t c, const std::vector< std::vector< double > > &ncsec)</code>	100
16.4.3.45	<code>is_curve_closed()</code>	100
16.4.3.46	<code>is_equality(const size_t i)</code>	100
16.4.3.47	<code>is_greater_than_or_equal_to(const size_t i)</code>	101
16.4.3.48	<code>is_less_than_or_equal_to(const size_t i)</code>	102

16.4.3.49	minimum_value() const	102
16.4.3.50	set_up_lp_constraints(glp_prob *lp) const	103
16.4.3.51	set_up_objective_function(glp_prob *lp) const	104
16.4.3.52	set_up_structural_variables(glp_prob *lp) const	105
16.4.3.53	solve_lp(const size_t rows, const size_t cols)	107
16.5	channel::ExceptionObject Class Reference	109
16.5.1	Detailed Description	111
16.5.2	Constructor & Destructor Documentation	111
16.5.2.1	ExceptionObject(const char *file, unsigned ln)	111
16.5.2.2	ExceptionObject(const char *file, unsigned int ln, const char *desc)	111
16.5.2.3	ExceptionObject(const char *file, unsigned ln, const char *desc, const char *loc)	111
16.5.2.4	ExceptionObject(const ExceptionObject &xpt)	112
16.5.3	Member Function Documentation	112
16.5.3.1	get_description() const	112
16.5.3.2	get_file() const	112
16.5.3.3	get_line() const	113
16.5.3.4	get_location() const	113
16.5.3.5	get_name_of_class() const	113
16.5.3.6	set_description(const std::string &s)	113
16.5.3.7	set_description(const char *s)	114
16.5.3.8	set_location(const std::string &s)	114
16.5.3.9	set_location(const char *s)	114
16.5.3.10	what() const	114
16.6	channel::TabulatedFunction Class Reference	115
16.6.1	Detailed Description	116
16.6.2	Member Function Documentation	116
16.6.2.1	a(const size_t i, const double u) const =0	116
16.6.2.2	alower(const size_t i, const double u) const =0	116
16.6.2.3	upper(const size_t i, const double u) const =0	117
16.6.2.4	degree() const =0	118
17	File Documentation	119
17.1	a3.hpp File Reference	119
17.1.1	Detailed Description	120
17.2	bound.hpp File Reference	121
17.2.1	Detailed Description	122
17.3	coefficient.hpp File Reference	122

17.3.1 Detailed Description	123
17.4 curvebuilder.cpp File Reference	124
17.4.1 Detailed Description	124
17.5 curvebuilder.hpp File Reference	125
17.5.1 Detailed Description	126
17.6 exceptionobject.hpp File Reference	127
17.6.1 Detailed Description	128
17.6.2 Macro Definition Documentation	129
17.6.2.1 treat_exception	129
17.7 main.cpp File Reference	129
17.7.1 Detailed Description	130
17.7.2 Function Documentation	131
17.7.2.1 main(int argc, char *argv[])	131
17.7.2.2 read_input(const string &fn, size_t &np, size_t &nc, bool &closed, double *&lx, double *&ly, double *&ux, double *&uy)	134
17.7.2.3 write_lp(const string &fn, const CurveBuilder &b)	136
17.7.2.4 write_solution(const string &fn, const CurveBuilder &b)	139
17.8 tabulatedfunction.hpp File Reference	140
17.8.1 Detailed Description	142
Index	143

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,

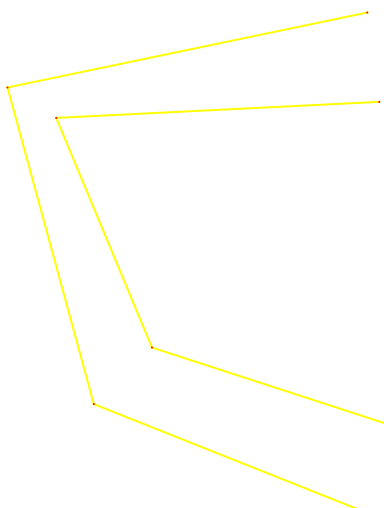


Figure 2.1: Example of an open channel

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,

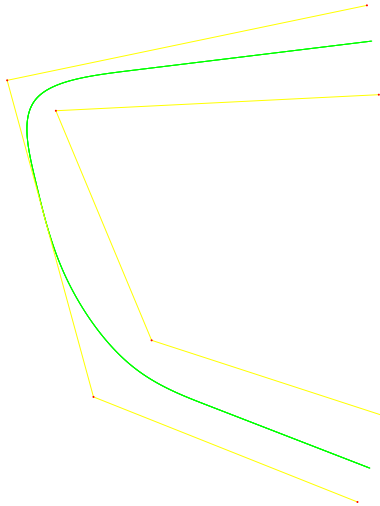


Figure 2.2: Example of a solution for the channel problem

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).

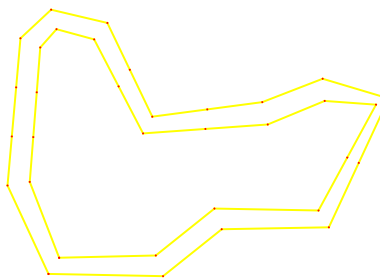


Figure 2.3: Example of a closed channel

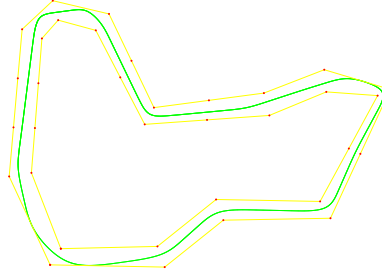


Figure 2.4: Example of a solution for the channel problem

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 $nc = 3$:

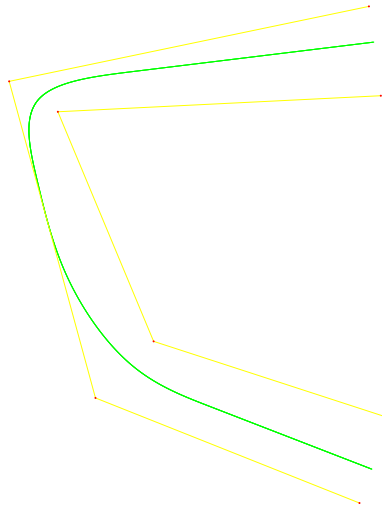


Figure 2.5: Example of a channel

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 $nc = 17$:

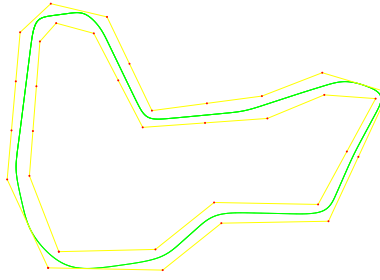


Figure 2.6: Example of a solution for the channel problem

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

The Channel2D library can be easily installed by downloading and unzipping the file `channel-2d.zip`. After doing that, one should see a directory named `channel-2d` with subdirectories `bin`, `data`, `doc`, `lib`, `scripts`, `src`, and `tst` inside. Subdirectory `scripts` contains an installation script, `install.sh`, that compiles the *Channel2D* library and an executable file that demonstrates how to use the library.

Before you execute the installation script `install.sh`, make sure you install the GNU [GLPK](#) in your computer. This toolkit contains the linear program solver used by the Channel2D library. If your computer runs Mac OSX, then you can install GLPK from `macports`. 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 library file `libglpk.a` are. In my computer, these files can be found in the following directories:

```
/opt/local/include
```

and

```
/opt/local/lib
```

Finally, you must edit two of my files named `Makefile`, so that you replace the directories above with the corresponding ones in your computer. The first `Makefile` is inside subdirectory `src`. Its content is:

```
CC = g++
AR = ar

#CFLAGS = -g -c -std=c++11 -Wall -pedantic
CFLAGS = -O2 -c -std=c++11 -Wall -pedantic

INC1 = .
INC2 = /opt/local/include

INCS = -I$(INC1) -I$(INC2)

OBJ = curvebuilder.o

LIB = libChannel2D.a

curvebuilder.o: $(INC1)/a3.hpp $(INC1)/tabulatedfunction.hpp \
                $(INC1)/bound.hpp $(INC1)/coefficient.hpp \
                $(INC1)/exceptionobject.hpp $(INC2)/glpk.h \
                $(INC1)/curvebuilder.hpp $(INC1)/curvebuilder.cpp
                $(CC) $(CFLAGS) $(INC1)/curvebuilder.cpp $(INCS)

all: $(OBJ)

lib:    $(INC1)/a3.hpp $(INC1)/tabulatedfunction.hpp \
```

```

$(INC1)/bound.hpp $(INC1)/coefficient.hpp \
$(INC1)/exceptionobject.hpp $(INC2)/glpk.h \
$(INC1)/curvebuilder.hpp $(INC1)/curvebuilder.cpp
$(AR) rc $(LIB) $(OBJ)
ranlib $(LIB)
mv $(LIB) ../lib

clean:
    rm -f *.o *~

realclean:
    rm -f *.o *~ ../lib/libChannel2D.a

```

Replace the line

```
INC2 = /opt/local/include
```

with

```
INC2 = path to the include directory of your computer where glpk.h is
```

Repeat the above step for the Makefile inside subdirectory `tst`. Its content is

```

CC = g++

#CFLAGS = -g -c -Wall -pedantic -std=c++11 -DDEBUGMODE
CFLAGS = -O2 -c -Wall -pedantic -std=c++11

#LFLAGS = -g
LFLAGS = -O2

INC1 = .
INC2 = ../src
INC3 = /opt/local/include

INCS = -I$(INC1) -I$(INC2) -I$(INC3)

LIB1 = ../lib
LIB2 = /opt/local/lib

LIBS = -L$(LIB1) -L$(LIB2) -lm -lChannel2D -lglpk

OBJS = main.o

bc2d: $(OBJS)
    $(CC) $(LFLAGS) $(OBJS) -o channel2d $(LIBS)
    mv channel2d ../bin/.

main.o: $(INC2)/a3.hpp $(INC2)/tabulatedfunction.hpp \
    $(INC2)/exceptionobject.hpp $(INC3)/glpk.h \
    $(INC2)/curvebuilder.hpp $(INC1)/main.cpp
    $(CC) $(CFLAGS) $(INC1)/main.cpp $(INCS)

clean:
    rm -fr *.o *~

realclean:
    rm -fr *.o *~ ../bin/channel2d

```

Replace the lines

```
INC3 = /opt/local/include
LIB2 = /opt/local/lib
```

with

```
INC3 = path to the include directory of your computer where glpk.h is
LIB2 = path to the lib directory of your computer where libglpk.a is
```


If you reach this point after executing the instructions above, then you are ready to compile the `Channel2D` library as well as a simple program to demonstrate how the library can be used. This is easy. The hard part is the installation of the GNU GLPK. If your computer runs Mac OSX or Linux, open a terminal, go to subdirectory `scripts`, and execute the script `install.sh`:

```
cd scripts
```

and

```
sh install.sh
```

If everything goes as expected in the compilation process, one should see the library `libChannel2D.a` inside subdirectory `lib`, and the executable `channel2d` inside subdirectory `bin`. Using this executable, we can run some examples of the channel problem, which are located in subdirectory `data/channels`. You find the details in section [The CurveBuilder class API](#).

I also left an XCode project file inside the `channel-2d` directory, but I currently have no Windows machine to create a .NET project file. So, if your computer runs Windows, you may have to create your own .NET project file by inspecting the two `Makefile` listed before. As you can see, they are both quite small. So, it should not be a problem to create your own .NET project file.

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

- Mac OSX 10.10.5 / GNU gcc 4.2.1 and clang-602.0.53

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. 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

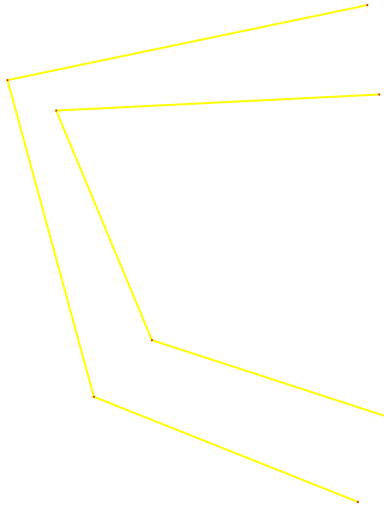


Figure 6.1: Example of a channel

and asks for a b-spline of degree 3 consisting of $n_s = 9$ 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 `ncp` lines. Each line specifies the pair of Cartesian coordinates of a control point. These coordinates are listed as follows:

$b_{0,x}, b_{0,y}$
 $b_{1,x}, b_{1,y}$
 \vdots
 $b_{n_{cp}-1,x}, b_{n_{cp}-1,y}$

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
```

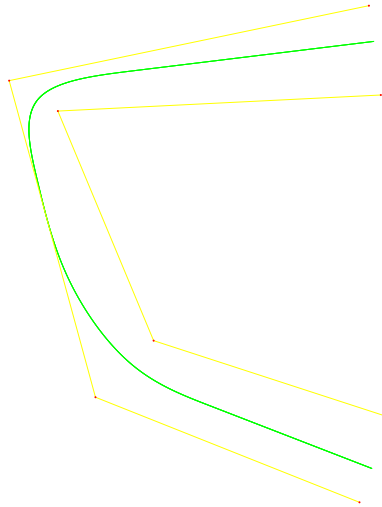


Figure 6.2: Example of a solution for the channel problem

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.	35
----------------------------	--------------------

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 [37](#)

Chapter 11

Hierarchical Index

11.1 Class Hierarchy

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

channel::Bound	47
channel::Coefficient	50
channel::CurveBuilder	52
exception	
channel::ExceptionObject	109
channel::TabulatedFunction	115
channel::a3	39

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	39
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	47
channel::Coefficient	This class represents a nonzero coefficient of an unknown of a constraint (inequality or equality) of a linear program instance	50
channel::CurveBuilder	This class provides methods for threading a cubic b-spline curve through a planar channel delimited by a pair of polygonal chains	52
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	109
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	115

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	119
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	121
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	122
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	124
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	125
exceptionobject.hpp	Definition of a class for handling exceptions	127
main.cpp	A simple program for testing the channel-2d library	129
tabulatedfunction.hpp	Definition of an abstract class for representing piecewise linear enclosures of certain polynomial functions of arbitrary degree	140

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.

Classes

- class [channel::a3](#)

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

- 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.

- 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.

- class [channel::Coefficient](#)

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

- class [channel::ExceptionObject](#)

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

- 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.

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

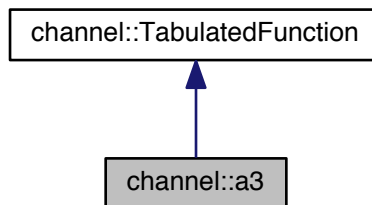
Class Documentation

16.1 channel::a3 Class Reference

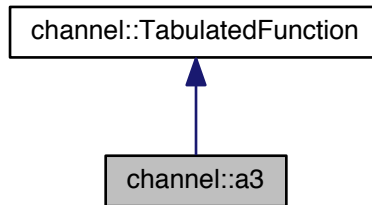
This class represents two-sided, piecewise linear enclosures for two polynomial functions of degree 3 in Bézier form.

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

Inheritance diagram for channel::a3:



Collaboration diagram for channel::a3:



Public Member Functions

- `a3 ()`
Creates an instance of this class.
- `virtual ~a3 ()`
Releases the memory held by an instance of this class.
- `virtual double alower (const size_t i, const double u) const throw (ExceptionObject)`
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 throw (ExceptionObject)`
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 throw (ExceptionObject)`
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`
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 slefes. In Proceedings of the
2002 St Malo conference on Curves and Surfaces, 2003.

Definition at line 73 of file a3.hpp.

16.1.2 Member Function Documentation

16.1.2.1 double channel::a3::a (const size_t *i*, const double *u*) const throw **ExceptionObject** [inline],[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>i</i>	Index of the i -th polynomial function.
<i>u</i>	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 223 of file a3.hpp.

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

```

228 {
229     if ( ( i != 1 ) && ( i != 2 ) ) {
230         std::stringstream ss( std::stringstream::in | std::stringstream::out );
231         ss << "Index of the polynomial function is out of range" ;
232         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
233     }
234
235     if ( ( u < 0 ) || ( u > 1 ) ) {
236         std::stringstream ss( std::stringstream::in | std::stringstream::out );
237         ss << "Parameter value must belong to the interval [0,1]" ;

```

```

238         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
239     }
240
241     return ( i == 1 ) ? a1( u ) : a1( 1 - u ) ;
242 }

```

16.1.2.2 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 329 of file a3.hpp.

Referenced by a().

```

330     {
331     #ifdef DEBUGMODE
332         assert( u >= 0 ) ;
333         assert( u <= 1 ) ;
334     #endif
335
336     return -u * ( 2 - u * ( 3 - u ) ) ;
337 }

```

16.1.2.3 double channel::a3::a1lower (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 279 of file a3.hpp.

References h().

Referenced by alower().

```

280     {
281     const double onethird = double( 1 ) / 3 ;
282
283     double res = _10 * h( u )
284                 + _11 * h( u - onethird )
285                 + _12 * h( u - 2 * onethird )
286                 + _13 * h( u - 1 ) ;
287
288     return res ;
289 }

```

16.1.2.4 `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 304 of file a3.hpp.

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

Referenced by [aupper\(\)](#).

```

305     {
306         const double onethird = double( 1 ) / 3 ;
307
308         double res = -10 * h( u - onethird ) - 8 * h( u - 2 * onethird ) ;
309
310         res *= ( double(1) / 27 ) ;
311
312         return res ;
313     }

```

16.1.2.5 `double channel::a3::alower (const size_t i , const double u) const throw ExceptionObject` `[inline]`,
`[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 147 of file a3.hpp.

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

```

152     {
153         if ( ( i != 1 ) && ( i != 2 ) ) {
154             std::stringstream ss( std::stringstream::in | std::stringstream::out ) ;
155             ss << "Index of the polynomial function is out of range" ;
156             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
157         }
158
159         if ( ( u < 0 ) || ( u > 1 ) ) {
160             std::stringstream ss( std::stringstream::in | std::stringstream::out ) ;
161             ss << "Parameter value must belong to the interval [0,1]" ;
162             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
163         }
164
165         return ( i == 1 ) ? alower( u ) : alower( 1 - u ) ;
166     }

```

16.1.2.6 `double channel::a3::aupper (const size_t i, const double u) const throw ExceptionObject` `[inline]`,
`[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 185 of file a3.hpp.

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

```

190     {
191         if ( ( i != 1 ) && ( i != 2 ) ) {
192             std::stringstream ss( std::stringstream::in | std::stringstream::out ) ;
193             ss << "Index of the polynomial function is out of range" ;
194             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
195         }
196
197         if ( ( u < 0 ) || ( u > 1 ) ) {
198             std::stringstream ss( std::stringstream::in | std::stringstream::out ) ;
199             ss << "Parameter value must belong to the interval [0,1]" ;
200             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
201         }
202
203         return ( i == 1 ) ? alupper( u ) : alupper( 1 - u ) ;
204     }

```

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

Returns the degree of tabulated functions.

Returns

The degree of the tabulated functions.

Implements [channel::TabulatedFunction](#).

Definition at line 253 of file a3.hpp.

```

254     {
255         return 3 ;
256     }

```

16.1.2.8 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

u	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 352 of file a3.hpp.

Referenced by a1lower(), and a1upper().

```

353     {
354         const double onethird = 1.0 / 3.0 ;
355
356         if ( u <= -onethird ) {
357             return 0 ;
358         }
359         else if ( u <= 0 ) {
360             return 3 * u + 1 ;
361         }
362         else if ( u <= onethird ) {
363             return 1 - 3 * u ;
364         }
365
366         return 0 ;
367     }

```

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.
- [Bound](#) (const [Bound](#) &b)
Creates an instance of this class from another instance of this class.
- [~Bound](#) ()
Releases the memory held by 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

- `CONSTRAINTYPE _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 58 of file `bound.hpp`.

16.2.2 Constructor & Destructor Documentation

16.2.2.1 `channel::Bound::Bound (const CONSTRAINTYPE 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 124 of file `bound.hpp`.

```

125     :
126     _ctype( type ) ,
127     _value( value ) ,
128     _row( row )
129     {
130     }
```

16.2.2.2 `channel::Bound::Bound (const Bound & b) [inline]`

Creates an instance of this class from another instance of this class.

Parameters

<i>b</i>	An instance of this class.
----------	----------------------------

Definition at line 142 of file `bound.hpp`.

```

143     :
144     _ctype( b._ctype ) ,
145     _value( b._value ) ,
146     _row( b._row )
147     {
148     }
```


16.2.3 Member Function Documentation

16.2.3.1 `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 204 of file bound.hpp.

References `_row`.

```
205     {  
206         return _row ;  
207     }
```

16.2.3.2 `CONSTRAINTTYPE 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 171 of file bound.hpp.

References `_ctype`.

```
172     {  
173         return _ctype ;  
174     }
```

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

Returns the value of this bound.

Returns

The value of this bound.

Definition at line 185 of file bound.hpp.

References `_value`.

```
186     {  
187         return _value ;  
188     }
```

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.
- [Coefficient](#) (const [Coefficient](#) &c)
Creates an instance of this class from another instance of this class.
- [~Coefficient](#) ()
Releases the memory held by 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 59 of file coefficient.hpp.

16.3.2 Constructor & Destructor Documentation

16.3.2.1 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 108 of file coefficient.hpp.

```

109      :
110      _row( row ) ,
111      _col( col ) ,
112      _value( value )
113      {
114      }
```

16.3.2.2 channel::Coefficient::Coefficient (const Coefficient & c) [inline]

Creates an instance of this class from another instance of this class.

Parameters

<i>c</i>	An instance of this class.
----------	----------------------------

Definition at line 126 of file coefficient.hpp.

```

127      :
128      _row( c._row ) ,
129      _col( c._col ) ,
130      _value( c._value )
131      {
132      }
```

16.3.3 Member Function Documentation**16.3.3.1 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 178 of file coefficient.hpp.

References `_col`.

```

179      {
180      return _col ;
181      }
```

16.3.3.2 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 159 of file coefficient.hpp.

References `_row`.

```
160     {
161         return _row ;
162     }
```

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

Returns the value of this coefficient.

Returns

The value of this coefficient.

Definition at line 192 of file coefficient.hpp.

References `_value`.

```
193     {
194         return _value ;
195     }
```

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

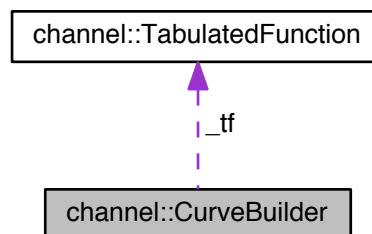
- [coefficient.hpp](#)

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) throw (ExceptionObject)
Creates an instance of this class.
- [CurveBuilder](#) (const [CurveBuilder](#) &b)
Clones an instance of this class.
- [~CurveBuilder](#) ()
Releases the memory held by 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 throw (ExceptionObject)
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 throw (ExceptionObject)
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 throw (ExceptionObject)
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 throw (ExceptionObject)
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 throw (ExceptionObject)
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 throw (ExceptionObject)
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 throw (ExceptionObject)
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 throw (ExceptionObject)
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 throw (ExceptionObject)
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 throw (ExceptionObject)
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 throw (ExceptionObject)

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::CONSTRAINTYPE 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 > > &nl, 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 > > &nl, 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 > > &nl, 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 80 of file curvebuilder.hpp.

16.4.2 Constructor & Destructor Documentation

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

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.

```

90 :
91   _np( np ) ,
92   _nc( nc ) ,
93   _closed ( closed )

```

```

94  {
95      if ( _closed ) {
96          if ( _np < 4 ) {
97              std::stringstream ss( std::stringstream::in | std::stringstream::out );
98              ss << "The number of curve segments must be at least 4 for a closed curve" ;
99              throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
100          }
101          if ( _nc < 3 ) {
102              std::stringstream ss( std::stringstream::in | std::stringstream::out );
103              ss << "The number of segments of a closed channel must be at least 3" ;
104              throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
105          }
106      }
107      else {
108          if ( _np < 1 ) {
109              std::stringstream ss( std::stringstream::in | std::stringstream::out );
110              ss << "The number of curve segments must be at least 1" ;
111              throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
112          }
113          if ( _nc < 1 ) {
114              std::stringstream ss( std::stringstream::in | std::stringstream::out );
115              ss << "The number of segments of an open channel must be at least 1" ;
116              throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
117          }
118      }
119
120      size_t nn = ( _closed ) ? _nc : ( _nc + 1 ) ;
121
122      _lxcoords.resize( nn ) ;
123      _lycoords.resize( nn ) ;
124      _uxcoords.resize( nn ) ;
125      _uycoords.resize( nn ) ;
126
127      for ( unsigned i = 0 ; i < nn ; i++ ) {
128          _lxcoords[ i ] = lx[ i ] ;
129          _lycoords[ i ] = ly[ i ] ;
130          _uxcoords[ i ] = ux[ i ] ;
131          _uycoords[ i ] = uy[ i ] ;
132      }
133
134      _tf = new a3() ;
135
136      _ofvalue = DBL_MAX ;
137
138      return ;
139  }

```

16.4.2.2 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 150 of file curvebuilder.cpp.

```

151  :
152      _np( b._np ) ,
153      _nc( b._nc ) ,
154      _closed( b._closed ) ,
155      _lxcoords( b._lxcoords ) ,
156      _lycoords( b._lycoords ) ,
157      _uxcoords( b._uxcoords ) ,
158      _uycoords( b._uycoords ) ,
159      _tf( b._tf ) ,
160      _coefficients( b._coefficients ) ,
161      _bounds( b._bounds ) ,
162      _ctrlpts( b._ctrlpts ) ,
163      _secdiff( b._secdiff ) ,
164      _ofvalue( b._ofvalue )
165  {
166  }

```

16.4.3 Member Function Documentation

16.4.3.1 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 195 of file curvebuilder.cpp.

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()`.

Referenced by `main()`.

```

196 {
197     // Compute the number of linear constraints (i.e., the number of
198     // rows of the matrix) of the linear program whose solution yields
199     // the curve.
200     size_t rows = (
201         ( 6 * ( _np + 1 ) )           // min-max
202         + ( ( _closed ) ? 8 : 4 )     // correspondence
203         + ( 2 * ( _nc - 1 ) )         // channel corners
204         + ( ( 3 * 4 * _np ) - 4 )     // sleeve corners
205         + ( 4 * ( _nc - 1 ) )         // sleeve in csegments
206     ) ;
207
208     // Compute the unknowns (i.e., the number of columns of the
209     // matrix) of the linear program whose solution yields the
210     // b-spline curve.
211     size_t cols = ( 6 * _np ) + 10 + ( ( _closed ) ? 1 : 2 ) + ( _nc - 1 ) ;
212
213     //
214     // Allocate memory for the array of coefficients and bounds.
215     //
216     _coefficients.resize( rows ) ;
217     _bounds.resize( rows ) ;
218
219     //
220     // Initialize the equation counter.
221     //
222     size_t eqline = 0 ;
223
224     //
225     // Compute the min-max constraints.
226     //
227     compute_min_max_constraints( eqline ) ;
228
229     //
230     // Compute the correspondence constraints.
231     //
232     compute_correspondence_constraints( eqline ) ;
233
234     //
235     // Compute channel corners outside sleeve constraints.
236     //
237     compute_channel_corners_outside_sleeve_constraints(
238     eqline ) ;
239
240     //
241     // Compute the sleeve corners in channel constraints.
242     //
243     compute_sleeve_corners_in_channel_constraints( eqline ) ;

```

```

243
244 //
245 // Compute the sleeve inside csegment constraints.
246 //
247 compute_sleeve_inside_csegment_constraints( eqline ) ;
248
249 //
250 // Solve the LP and get the solution.
251 //
252 error = solve_lp( rows , cols ) ;
253
254 return ( error == 0 ) ;
255 }

```

16.4.3.2 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 730 of file curvebuilder.cpp.

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()`.

```

731 {
732 //
733 // This restriction applies to channels with at least 3 c-sections
734 // only.
735 //
736 if ( _nc < 2 ) {
737     return ;
738 }
739
740 // For each inner corner of the given channel, compute a
741 // constraint that ensures that the channel corner is outside the
742 // sleeve.
743 const double NpOverNc = _np / double( _nc ) ;
744 const double onesixth = 1 / double( 6 ) ;
745
746 for ( size_t c = 1 ; c < _nc ; c++ ) {
747     //
748     // Find the parameter \e t corresponding to the \e c corner.
749     //
750     double t = ( c * NpOverNc ) + 3 ;
751
752     //
753     // Find the curve segment \e p containing point at parameter \e
754     // t.
755     size_t p = ( size_t ) floor( t - 3 ) ;
756
757     // Compute the column indices of the linear program matrix
758     // corresponding to the four control points defining the p-th
759     // segment of the b-spline curve.
760
761     std::vector< std::vector< size_t > > cp( 4 , std::vector< size_t >( 2 , 0 ) ) ;
762
763     for ( size_t i = 0 ; i < 4 ; i++ ) {
764         for ( size_t j = 0 ; j < 2 ; j++ ) {
765             cp[ i ][ j ] = compute_control_value_column_index( p , i , j )
766         }
767     }
768
769     //
770     // Compute the coefficients of the control points.
771     //
772     double s = t - floor( t ) ;

```

```

773     std::vector< double > coeffs( 4 ) ;
774
775     coeffs[ 0 ] = onesixth * ( 1 + s * ( -3 + s * ( 3 - s ) ) ) ;
776     coeffs[ 1 ] = onesixth * ( 4 + s * s * ( - 6 + 3 * s ) ) ;
777     coeffs[ 2 ] = onesixth * ( 1 + s * ( 3 + s * ( 3 - 3 * s ) ) ) ;
778     coeffs[ 3 ] = onesixth * ( s * s * s ) ;
779
780     // Get the LP matrix column index corresponding to the
781     // barycentric coordinate associated with the c-th corner of the
782     // channel.
783     size_t k = compute_index_of_corner_barycentric_coordinate
784     ( c ) ;
785     //
786     // Insert the constraints into the LP program.
787     //
788     for ( size_t i = 0 ; i < 4 ; i++ ) {
789         insert_coefficient( eqline , cp[ i ][ 0 ] , coeffs[ i ] ) ;
790         insert_coefficient( eqline + 1 , cp[ i ][ 1 ] , coeffs[ i ] ) ;
791     }
792
793     insert_coefficient( eqline , k , ( _lxcoords[ c ] -
794     _uxcoords[ c ] ) ) ;
795     insert_coefficient( eqline + 1 , k , ( _lycoords[ c ] -
796     _uycoords[ c ] ) ) ;
797
798     insert_bound( eqline , Bound::EQT , _lxcoords[ c ] ) ;
799     insert_bound( eqline + 1 , Bound::EQT , _lycoords[ c ] ) ;
800
801     //
802     // Increment equation counter.
803     //
804     eqline += 2 ;
805 }
806 return ;
807 }

```

16.4.3.3 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 1215 of file curvebuilder.cpp.

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()`.

```

1221 {
1222     #ifdef DEBUGMODE
1223         assert( p < _np ) ;
1224         assert( i <= 3 ) ;
1225         assert( v <= 1 ) ;

```

```

1226 #endif
1227
1228     return 2 * ( p + i ) + v ;
1229 }

```

16.4.3.4 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 406 of file curvebuilder.cpp.

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()`.

```

407 {
408     // Get the column index of the x- and y-coordinates of the first
409     // three control points of the b-spline to be threaded into the
410     // channel.
411
412     std::vector< size_t > strx( 4 ) ;
413
414     strx[ 0 ] = compute_control_value_column_index( 0 , 0 , 0 ) ;
415     strx[ 1 ] = compute_control_value_column_index( 0 , 1 , 0 ) ;
416     strx[ 2 ] = compute_control_value_column_index( 0 , 2 , 0 ) ;
417
418     // Get the column index of the barycentric coordinate defining
419     // the first endpoint of the b-spline with respect to the first
420     // channel points.
421
422     strx[ 3 ] = compute_index_of_endpoint_barycentric_coordinate
423     ( 0 ) ;
424
425     //
426     // Get the coefficients of the unknowns of the constraint.
427     //
428     std::vector< double > vals( 4 ) ;
429
430     vals[ 0 ] = double( 1 ) / double( 6 ) ;
431     vals[ 1 ] = double( 2 ) / double( 3 ) ;
432     vals[ 2 ] = vals[ 0 ] ;
433     vals[ 3 ] = _lxcoords[ 0 ] - _uxcoords[ 0 ] ;
434
435     //
436     // Constraint corresponding to first Cartesian coordinate.
437     //
438     insert_extreme_point_correspondence_constraint(
439         eqline ,
440         strx ,
441         vals ,
442         _lxcoords[ 0 ]
443     ) ;
444
445     ++eqline ; // increment the equation counter.
446
447     //
448     // Constraint corresponding to second Cartesian coordinate.
449     //
450     std::vector< size_t > stry( 4 ) ;
451
452     stry[ 0 ] = compute_control_value_column_index( 0 , 0 , 1 ) ;
453     stry[ 1 ] = compute_control_value_column_index( 0 , 1 , 1 ) ;
454     stry[ 2 ] = compute_control_value_column_index( 0 , 2 , 1 ) ;
455     stry[ 3 ] = strx[ 3 ] ;
456

```

```

457     vals[ 3 ] = _lycoords[ 0 ] - _uycoords[ 0 ] ;
458
459     insert_extreme_point_correspondence_constraint(
460         eqline ,
461         stry ,
462         vals ,
463         _lycoords[ 0 ]
464     ) ;
465
466     ++eqline ; // increment the equation counter.
467
468     // -----
469     //
470     // If the curve is closed, then the last three control points must
471     // match the first three control points. Otherwise, we must fix
472     // the position of the final of the curve, which differs from the
473     // starting one.
474     //
475     // -----
476
477     // Get the column index of the x- and y-coordinates of the last
478     // three control points of the b-spline to be threaded into the
479     // channel.
480
481     std::vector< size_t > endx( 4 ) ;
482
483     endx[ 0 ] = compute_control_value_column_index(
484         _np - 1 , 1 , 0 ) ;
485     endx[ 1 ] = compute_control_value_column_index(
486         _np - 1 , 2 , 0 ) ;
487     endx[ 2 ] = compute_control_value_column_index(
488         _np - 1 , 3 , 0 ) ;
489
490     std::vector< size_t > endy( 4 ) ;
491
492     endy[ 0 ] = compute_control_value_column_index(
493         _np - 1 , 1 , 1 ) ;
494     endy[ 1 ] = compute_control_value_column_index(
495         _np - 1 , 2 , 1 ) ;
496     endy[ 2 ] = compute_control_value_column_index(
497         _np - 1 , 3 , 1 ) ;
498
499     if ( _closed ) {
500         //
501         // Compute the equations that match the first three and last
502         // three control points of the b-spline: last is equal to third,
503         // ...
504         //
505         insert_periodic_correspondence_constraints(
506             eqline ,
507             strx ,
508             stry ,
509             endx ,
510             endy
511         ) ;
512
513         eqline += 6 ; // increment the equation counter.
514     }
515     else {
516         //
517         // Compute the equations determining the b-spline final point.
518         //
519         // Get the column index of the barycentric coordinate of the
520         // final point of the b-spline with respect to the final points
521         // of the channel.
522
523         endx[ 3 ] = compute_index_of_endpoint_barycentric_coordinate
524         ( 1 ) ;
525         vals[ 3 ] = _lxcoords[ _nc ] - _uxcoords[ _nc ] ;
526
527         //
528         // Constraint corresponding to first Cartesian coordinate.
529         //
530         insert_extreme_point_correspondence_constraint(
531             eqline ,
532             endx ,
533             vals ,
534             _lxcoords[ _nc ]
535         ) ;

```

```

531
532     endy[ 3 ] = endx[ 3 ] ;
533     vals[ 3 ] = _lycoords[ _nc ] - _uycoords[ _nc ] ;
534
535     ++eqline ; // increment the equation counter.
536
537     insert_extreme_point_correspondence_constraint(
538         eqline ,
539         endy ,
540         vals ,
541         _lycoords[ _nc ]
542     ) ;
543
544     ++eqline ; // increment the equation counter.
545 }
546
547 return ;
548 }

```

16.4.3.5 `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 1325 of file curvebuilder.cpp.

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

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

```

1326 {
1327 #ifdef DEBUGMODE
1328     assert( i > 0 ) ;
1329     assert( i < _nc ) ;
1330 #endif
1331
1332     size_t offset = ( 6 * _np ) + 10 + ( ( _closed ) ? 0 : 1 ) ;
1333
1334     return offset + i ;
1335 }
1336 }

```

16.4.3.6 `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 1293 of file curvebuilder.cpp.

References `_closed`, and `_np`.

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

```

1294 {
1295     #ifdef DEBUGMODE
1296     if ( _closed ) {
1297         assert( i == 0 ) ;
1298     }
1299     else {
1300         assert( i <= 1 ) ;
1301     }
1302 #endif
1303
1304     size_t offset = ( 6 * _np ) + 10 ;
1305
1306     return offset + i ;
1307 }
```

16.4.3.7 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 274 of file curvebuilder.cpp.

References `_np`, `compute_control_value_column_index()`, `compute_second_difference_column_index()`, and `insert_↔ min_max_constraints()`.

Referenced by `build()`.

```

275 {
276     //
277     // Obtain the min-max constraints for each second difference.
278     //
279
280     for ( size_t j = 1 ; j < 3 ; j++ ) {
281         for ( size_t v = 0 ; v < 2 ; v++ ) {
282             // Get the column indices of the lower bound and of the
283             // upper bound of the v-th coordinate of the j-th second
284             // difference.
285             size_t jl = compute_second_difference_column_index(
286                                     0 ,
287                                     j ,
288                                     0 ,
289                                     v
290                                 ) ;
291
292             size_t ju = compute_second_difference_column_index(
293                                     0 ,
294                                     j ,
295                                     1 ,
296                                     v
297                                 ) ;
298
299             // Get the column indices of the v-th coordinates that define
```

```

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

```

```

381             c3
382         ) ;
383
384         //
385         // Increment equation counter
386         //
387         eqline += 3 ;
388     }
389 }
390
391 return ;
392 }

```

16.4.3.8 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

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

Definition at line 1176 of file curvebuilder.cpp.

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

Referenced by `compute_sleeve_inside_csegment_constraints()`.

```

1182 {
1183     #ifdef DEBUGMODE
1184         assert( s <= _nc ) ;
1185     #endif
1186
1187     size_t t = ( _closed ) ? ( s % _nc ) : s ;
1188
1189     nx = _lycoords[ t ] - _uycoords[ t ] ;
1190     ny = _uxcoords[ t ] - _lxcoords[ t ] ;
1191
1192     return ;
1193 }

```

16.4.3.9 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

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

Definition at line 1100 of file curvebuilder.cpp.

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

Referenced by `compute_sleeve_corners_in_channel_constraints()`.

```

1106 {
1107     #ifdef DEBUGMODE
1108         assert( s < _nc ) ;
1109     #endif

```

```

1110
1111     size_t t = s + 1 ;
1112
1113     if ( _closed ) {
1114         t %= _nc ;
1115     }
1116
1117     nx = _lycoords[ s ] - _lycoords[ t ] ;
1118     ny = _lxcoords[ t ] - _lxcoords[ s ] ;
1119
1120     return ;
1121 }

```

16.4.3.10 `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

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

Definition at line 1138 of file curvebuilder.cpp.

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

Referenced by `compute_sleeve_corners_in_channel_constraints()`.

```

1144 {
1145     #ifdef DEBUGMODE
1146         assert( s < _nc ) ;
1147     #endif
1148
1149     size_t t = s + 1 ;
1150
1151     if ( _closed ) {
1152         t %= _nc ;
1153     }
1154
1155     nx = _uycoords[ t ] - _uycoords[ s ] ;
1156     ny = _uxcoords[ s ] - _uxcoords[ t ] ;
1157
1158     return ;
1159 }

```

16.4.3.11 `size_t channel::CurveBuilder::compute_second_difference_column_index (const size_t p, const size_t l, 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 1256 of file curvebuilder.cpp.

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()`.

```

1263 {
1264 #ifdef DEBUGMODE
1265     assert( p < _np );
1266     assert( i >= 1 );
1267     assert( i <= 2 );
1268     assert( l <= 1 );
1269     assert( v <= 1 );
1270 #endif
1271
1272     size_t offset = ( 2 * _np ) + 6 ;
1273
1274     return offset + ( 4 * ( p + i - 1 ) ) + ( 2 * l ) + v ;
1275 }
```

16.4.3.12 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

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

Definition at line 562 of file curvebuilder.cpp.

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()`.

```

563 {
564     //
565     // Compute outward normals to the line segments of the channel.
566     //
567     std::vector< std::vector< double > > nl( _nc , std::vector< double >( 2 , 0 ) ) ;
568     std::vector< std::vector< double > > nu( _nc , std::vector< double >( 2 , 0 ) ) ;
569
570     for ( size_t c = 0 ; c < _nc ; c++ ) {
571         compute_normal_to_lower_envelope( c , nl[ c ][ 0 ] , nl[ c ][ 1 ] ) ;
572         compute_normal_to_upper_envelope( c , nu[ c ][ 0 ] , nu[ c ][ 1 ] ) ;
573     }
574
575     // Each segment of the b-spline must be enclosed by a sleeve with
576     // four breakpoints, two of which are shared with the previous and
577     // next segment (if any). Each breakpoint is constrained to be
578     // bounded by a pair of parallel segments (lower and upper) of the
579     // channel.
580
581     const size_t lo = ( 3 * 3 ) + 1 ;
582     const size_t up = ( 3 * _np ) + 8 ;
583     const double NcOverNp = _nc / double( _np ) ;
584
585     for ( size_t u = lo ; u <= up ; u++ ) {
586         //
587         // Find the index of the channel segment corresponding to \e u.
588         //
589         double t = u / double( 3 ) ;
590         double s = t - floor( t ) ;
591         size_t p = ( size_t ) floor( t - 3 ) ;
592         size_t c = ( size_t ) ( ( t - 3 ) * NcOverNp ) ;
593
594         // Compute the column indices of the linear program matrix
595         // corresponding to the four control points defining the p-th
596         // segment of the b-spline curve.
597
598         std::vector< std::vector< size_t > > cp( 4 , std::vector< size_t >( 2 , 0 ) ) ;
599
600         for ( size_t i = 0 ; i < 4 ; i++ ) {
601             for ( size_t j = 0 ; j < 2 ; j++ ) {
602                 cp[ i ][ j ] = compute_control_value_column_index( p , i , j )
603             }
604         }
605
606         // Compute the column indices of the linear program matrix
607         // of the values of the second difference bounds associated with
608         // the p-th segment of the b-spline curve.
609
610         std::vector< std::vector< std::vector< size_t > > > sd(
611             2 ,
612             std::vector< std::vector< size_t > >
613             (
614                 2 ,
615                 std::vector< size_t >( 2 , 0 )
616             )
617         ) ;
618
619         for ( size_t j = 1 ; j < 3 ; j++ ) {
620             for ( size_t l = 0 ; l < 2 ; l++ ) {
621                 for ( size_t v = 0 ; v < 2 ; v++ ) {
622                     sd[ j - 1 ][ l ][ v ] = compute_second_difference_column_index
623                     ( p , j , l , v ) ;
624                 }
625             }
626
627             // -----
628             //
629             // Process nonlinear terms of Constraint (3a).
630             //
631             // -----
632
633             //
634             // Nonlinear terms of \f$\stackrel{\sim}{p}$\f$
635             //
636             insert_nonlinear_terms_of_epiece_point_lower_bound(
637                 eqline ,
638                 s ,
639                 c ,
640                 sd ,
641

```

```

642                                     nl ,
643                                     nu
644                                 ) ;
645
646 //
647 // Nonlinear terms of  $f \sim e^p f$ .
648 //
649
650 insert_nonlinear_terms_of_epiece_point_upper_bound(
651                                     eqline + 2 ,
652                                     s ,
653                                     c ,
654                                     sd ,
655                                     nl ,
656                                     nu
657                                 ) ;
658
659
660 // -----
661 //
662 // Process linear terms of Constraint (3a).
663 //
664 // -----
665
666 //
667 // Linear terms of  $f \sim e \sim p f$ 
668 //
669
670 insert_linear_terms_of_epiece_point_bounds(
671                                     eqline ,
672                                     s ,
673                                     t ,
674                                     p ,
675                                     c ,
676                                     cp ,
677                                     nl ,
678                                     nu
679                                 ) ;
680
681 //
682 // Linear terms of  $f \sim e \sim p f$ .
683 //
684
685 insert_linear_terms_of_epiece_point_bounds(
686                                     eqline + 2 ,
687                                     s ,
688                                     t ,
689                                     p ,
690                                     c ,
691                                     cp ,
692                                     nl ,
693                                     nu
694                                 ) ;
695
696 // -----
697 //
698 // Compute right-hand side of the constraints.
699 //
700 // -----
701
702 insert_rhs_of_sleeve_corners_in_channel_constraints
703 (
704                                     eqline ,
705                                     c ,
706                                     nl ,
707                                     nu
708                                 ) ;
709
710 //
711 // Increment equation counter.
712 //
713 eqline += 4 ;
714 }
715 return ;
716 }

```

16.4.3.13 `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 820 of file curvebuilder.cpp.

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()`.

```

821 {
822     //
823     // This restriction applies to channels with at least 3 c-sections
824     // only.
825     //
826     if ( _nc < 2 ) {
827         return ;
828     }
829
830     //
831     // Compute normals to the c-sections of the channel.
832     //
833
834     const size_t NumberOfCSections = ( _closed ) ? _nc : _nc + 1 ;
835
836     std::vector< std::vector< double > > ncsec(
837         NumberOfCSections ,
838         std::vector< double >( 2 , 0 )
839     ) ;
840
841     for ( size_t c = 0 ; c < NumberOfCSections ; c++ ) {
842         compute_normal_to_csection(
843             c ,
844             ncsec[ c ][ 0 ] ,
845             ncsec[ c ][ 1 ]
846         ) ;
847     }
848
849     // For each inner corner of the given channel, compute two
850     // constraints which ensure that the e-piece breakpoints
851     // immediately on the right (resp. left) of the corresponding
852     // c-section remain in the right (resp. left) c-segment of the
853     // channel.
854
855     const double NpOverNc = _np / double( _nc ) ;
856     const double onethird = 1 / double( 3 ) ;
857     const double twothird = 2 * onethird ;
858
859     for ( size_t c = 1 ; c < _nc ; c++ ) {
860         //
861         // Find the parameter \e t corresponding to the \e c corner.
862         //
863         double t = ( c * NpOverNc ) + 3 ;
864
865         //
866         // Find the curve segment \e p containing point at parameter \e
867         // t.
868         size_t p = ( size_t ) floor( t - 3 ) ;
869
870         // Compute the indices of the curve segments corresponding to
871         // the e-piece breakpoints immediately to the right and left of
872         // point \c p(t).
873
874         double s = t - floor( t ) ;
875
876         size_t p1 , p2 ;
877         double s1 , s2 ;
878
879         if ( s == 0 ) {
880             p1 = p - 1 ;
881             p2 = p ;
882             s1 = twothird ;
883             s2 = onethird ;
884         }
885         else if ( s < onethird ) {

```

```

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

```

```

965     for ( size_t i = 0 ; i < 4 ; i++ ) {
966         for ( size_t j = 0 ; j < 2 ; j++ ) {
967             cp1[ i ][ j ] = compute_control_value_column_index( p1 , i , j
    ) ;
968         }
969     }
970
971     // Compute the column indices of the LP matrix corresponding to
972     // the four control points defining the p2-th segment of the
973     // curve.
974
975     std::vector< std::vector< size_t > > cp2( 4 , std::vector< size_t >( 2 , 0 ) ) ;
976
977     for ( size_t i = 0 ; i < 4 ; i++ ) {
978         for ( size_t j = 0 ; j < 2 ; j++ ) {
979             cp2[ i ][ j ] = compute_control_value_column_index( p2 , i , j
    ) ;
980         }
981     }
982
983     // -----
984     //
985     // Process nonlinear terms of Constraint (3c).
986     //
987     // -----
988
989     //
990     // Nonlinear terms of  $f \stackrel{\sim}{\sim} \{e\}^p(s_1) f$ 
991     //
992     insert_nonlinear_terms_of_epiece_point_lower_bound(
993         eqline ,
994         s1 ,
995         c ,
996         sd1 ,
997         ncsec
998     ) ;
999
1000    //
1001    // Nonlinear terms of  $f \stackrel{\sim}{\sim} \{e\}^p(s_1) f$ .
1002    //
1003    insert_nonlinear_terms_of_epiece_point_upper_bound(
1004        eqline + 1 ,
1005        s1 ,
1006        c ,
1007        sd1 ,
1008        ncsec
1009    ) ;
1010
1011    //
1012    // Nonlinear terms of  $f \stackrel{\sim}{\sim} \{e\}^p(s_2) f$ 
1013    //
1014    insert_nonlinear_terms_of_epiece_point_lower_bound(
1015        eqline + 2 ,
1016        s2 ,
1017        c ,
1018        sd2 ,
1019        ncsec
1020    ) ;
1021
1022    //
1023    // Nonlinear terms of  $f \stackrel{\sim}{\sim} \{e\}^p(s_2) f$ .
1024    //
1025    insert_nonlinear_terms_of_epiece_point_upper_bound(
1026        eqline + 3 ,
1027        s2 ,
1028        c ,
1029        sd2 ,
1030        ncsec
1031    ) ;
1032
1033    // -----
1034    //
1035    // Process linear terms of Constraint (3c).
1036    //
1037    // -----
1038
1039    //
1040    // Linear terms of  $f \stackrel{\sim}{\sim} \{e\}^p(s_1) f$ 
1041    //
1042    insert_linear_terms_of_epiece_point_bounds(
1043        eqline ,

```

```

1044             s1 ,
1045             t1 ,
1046             p1 ,
1047             c ,
1048             cpl ,
1049             ncsec
1050         ) ;
1051
1052         //
1053         // Linear terms of \f$\stackrel{\sim}{e}^p( s_2 )\f$.
1054         //
1055         insert_linear_terms_of_epiece_point_bounds(
1056             eqline + 2 ,
1057             s2 ,
1058             t2 ,
1059             p2 ,
1060             c ,
1061             cp2 ,
1062             ncsec
1063         ) ;
1064
1065         // -----
1066         //
1067         // Compute right-hand side of the constraints.
1068         //
1069         // -----
1070
1071         insert_rhs_of_sleeve_inside_csegment_constraints(
1072             eqline ,
1073             c ,
1074             ncsec
1075         ) ;
1076
1077         // Increment equation counter.
1078         //
1079         eqline += 4 ;
1080     }
1081
1082     return ;
1083 }

```

16.4.3.14 `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 2087 of file curvebuilder.cpp.

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()`.

```

2093 {
2094     try {
2095         lower = _tf->alower( j , t ) ;
2096         upper = _tf->aupper( j , t ) ;
2097     }
2098     catch ( const ExceptionObject& xpt ) {
2099         treat_exception( xpt ) ;
2100         exit( EXIT_FAILURE ) ;
2101     }
2102 }

```

```

2103     return ;
2104 }

```

16.4.3.15 double channel::CurveBuilder::get_bound_of_ith_constraint (const size_t *i*) const throw ExceptionObject

[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 439 of file curvebuilder.hpp.

Referenced by write_lp().

```

440 {
441     if ( _coefficients.empty() ) {
442         std::stringstream ss( std::stringstream::in | std::stringstream::out );
443         ss << "No constraint has been created so far" ;
444         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
445     }
446
447     if ( i >= _coefficients.size() ) {
448         std::stringstream ss( std::stringstream::in | std::stringstream::out );
449         ss << "Constraint index is out of range" ;
450         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
451     }
452
453 #ifdef DEBUGMODE
454     assert( _bounds.size() == _coefficients.size() ) ;
455     assert( _bounds.size() > std::vector< std::vector< Bound > >::size_type( i ) ) ;
456 #endif
457
458     return _bounds[ i ].get_value() ;
459 }

```

16.4.3.16 size_t channel::CurveBuilder::get_coefficient_identifier (const size_t *i*, const size_t *j*) const throw ExceptionObject

[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 364 of file curvebuilder.hpp.

Referenced by write_lp().

```

365 {
366     if ( _coefficients.empty() ) {
367         std::stringstream ss( std::stringstream::in | std::stringstream::out );
368         ss << "No constraint has been created so far" ;
369         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
370     }
371
372     if ( i >= _coefficients.size() ) {
373         std::stringstream ss( std::stringstream::in | std::stringstream::out );
374         ss << "Constraint index is out of range" ;
375         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
376     }
377
378     if ( j >= _coefficients[ i ].size() ) {
379         std::stringstream ss( std::stringstream::in | std::stringstream::out );
380         ss << "Coefficient index is out of range" ;
381         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
382     }
383
384     return _coefficients[ i ][ j ].get_col() ;
385 }

```

16.4.3.17 `size_t channel::CurveBuilder::get_coefficient_value (const size_t i, const size_t j) const throw ExceptionObject`
`[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 402 of file curvebuilder.hpp.

Referenced by `write_lp()`.

```

403 {
404     if ( _coefficients.empty() ) {
405         std::stringstream ss( std::stringstream::in | std::stringstream::out );
406         ss << "No constraint has been created so far" ;
407         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
408     }
409
410     if ( i >= _coefficients.size() ) {
411         std::stringstream ss( std::stringstream::in | std::stringstream::out );
412         ss << "Constraint index is out of range" ;
413         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
414     }
415
416     if ( j >= _coefficients[ i ].size() ) {
417         std::stringstream ss( std::stringstream::in | std::stringstream::out );
418         ss << "Coefficient index is out of range" ;
419         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
420     }
421
422     return _coefficients[ i ][ j ].get_value() ;
423 }

```

16.4.3.18 `double channel::CurveBuilder::get_control_value (const size_t i, const size_t v) const throw ExceptionObject`
`[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 288 of file curvebuilder.hpp.

Referenced by write_solution().

```

293     {
294         if ( i >= ( _np + 3 ) ) {
295             std::stringstream ss( std::stringstream::in | std::stringstream::out );
296             ss << "Index of the control point is out of range" ;
297             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
298         }
299
300         if ( v >= 2 ) {
301             std::stringstream ss( std::stringstream::in | std::stringstream::out );
302             ss << "Index of the Cartesian coordinate is out of range" ;
303             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
304         }
305
306         if ( _ctrlpts.empty() ) {
307             std::stringstream ss( std::stringstream::in | std::stringstream::out );
308             ss << "Control points have not been computed" ;
309             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
310         }
311
312         size_t index = ( 2 * i ) + v ;
313
314         return _ctrlpts[ index ] ;
315     }

```

16.4.3.19 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 184 of file curvebuilder.hpp.

Referenced by get_number_of_control_points(), and write_lp().

```

185     {
186         return 3 ;
187     }

```

16.4.3.20 double channel::CurveBuilder::get_lower_bound_on_second_difference_value (const size_t p , const size_t i , const size_t v) const throw ExceptionObject [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 591 of file curvebuilder.hpp.

```

597 {
598     if ( p >= _np ) {
599         std::stringstream ss( std::stringstream::in | std::stringstream::out );
600         ss << "Index of the curve segment is out of range" ;
601         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
602     }
603
604     if ( ( i < 1 ) || ( i > 3 ) ) {
605         std::stringstream ss( std::stringstream::in | std::stringstream::out );
606         ss << "Index of the second difference vector is out of range" ;
607         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
608     }
609
610     if ( v >= 2 ) {
611         std::stringstream ss( std::stringstream::in | std::stringstream::out );
612         ss << "Index of the Cartesian coordinate is out of range" ;
613         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
614     }
615
616     if ( _secdiff.empty() ) {
617         std::stringstream ss( std::stringstream::in | std::stringstream::out );
618         ss << "Second differences have not been computed" ;
619         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
620     }
621
622     size_t index = ( 4 * 2 * p ) + ( 4 * ( i - 1 ) ) + v ;
623
624     return _secdiff[ index ] ;
625 }
```

16.4.3.21 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

lp	A pointer to the instance of the LP program.
------	--

Definition at line 2706 of file curvebuilder.cpp.

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

Referenced by `solve_lp()`.

```

2707 {
2708     //
2709     // Obtain the control points of the spline curve.
2710     //
2711     for ( size_t i = 0 ; i < 4 ; i++ ) {
2712         for ( size_t v = 0 ; v < 2 ; v++ ) {
2713             size_t c = compute_control_value_column_index(
2714                                     0 ,
```



```

2715                                     i ,
2716                                     v
2717                                 ) ;
2718     _ctrlpts.push_back(
2719         glp_get_col_prim(
2720             lp ,
2721             int( c ) + 1
2722         )
2723     ) ;
2724 }
2725 }
2726
2727 for ( size_t p = 1 ; p < _np ; p++ ) {
2728     for ( size_t v = 0 ; v < 2 ; v++ ) {
2729         size_t c = compute_control_value_column_index(
2730             p ,
2731             3 ,
2732             v
2733         ) ;
2734         _ctrlpts.push_back(
2735             glp_get_col_prim(
2736                 lp ,
2737                 int( c ) + 1
2738             )
2739         ) ;
2740     }
2741 }
2742
2743 //
2744 // Obtain the lower and upper bounds of the second differences.
2745 //
2746 for ( size_t i = 1 ; i < 3 ; i++ ) {
2747     for ( size_t l = 0 ; l < 2 ; l++ ) {
2748         for ( size_t v = 0 ; v < 2 ; v++ ) {
2749             size_t c = compute_second_difference_column_index(
2750                 0 ,
2751                 i ,
2752                 l ,
2753                 v
2754             ) ;
2755
2756             _secdiff.push_back(
2757                 glp_get_col_prim(
2758                     lp ,
2759                     int( c ) + 1
2760                 )
2761             ) ;
2762         }
2763     }
2764 }
2765
2766 for ( size_t p = 1 ; p < _np ; p++ ) {
2767     for ( size_t l = 0 ; l < 2 ; l++ ) {
2768         for ( size_t v = 0 ; v < 2 ; v++ ) {
2769             size_t c = compute_second_difference_column_index(
2770                 p ,
2771                 2 ,
2772                 l ,
2773                 v
2774             ) ;
2775
2776             _secdiff.push_back(
2777                 glp_get_col_prim(
2778                     lp ,
2779                     int( c ) + 1
2780                 )
2781             ) ;
2782         }
2783     }
2784 }
2785
2786 //
2787 // Obtain the minimum value of the objective function.
2788 //
2789 _ofvalue = glp_get_obj_val( lp ) ;
2790
2791 return ;
2792 }

```

16.4.3.22 `size_t channel::CurveBuilder::get_number_of_coefficients_in_the_ith_constraint (const size_t i) const throw
ExceptionObject) [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 330 of file curvebuilder.hpp.

Referenced by write_lp().

```

331     {
332         if ( _coefficients.empty() ) {
333             std::stringstream ss( std::stringstream::in | std::stringstream::out );
334             ss << "No constraint has been created so far" ;
335             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
336         }
337
338         if ( i >= _coefficients.size() ) {
339             std::stringstream ss( std::stringstream::in | std::stringstream::out );
340             ss << "Constraint index is out of range" ;
341             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
342         }
343
344         return _coefficients[ i ].size() ;
345     }

```

16.4.3.23 size_t channel::CurveBuilder::get_number_of_constraints () const throw ExceptionObject [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 260 of file curvebuilder.hpp.

Referenced by write_lp().

```

261     {
262         if ( _coefficients.empty() ) {
263             std::stringstream ss( std::stringstream::in | std::stringstream::out );
264             ss << "No constraint has been created so far" ;
265             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
266         }
267
268         return _coefficients.size() ;
269     }

```

16.4.3.24 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 242 of file curvebuilder.hpp.

References `get_degree()`.

Referenced by `write_solution()`.

```
243     {
244         return _np + get_degree() ;
245     }
```

16.4.3.25 `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 212 of file curvebuilder.hpp.

References `_nc`.

Referenced by `write_lp()`.

```
213     {
214         return _nc ;
215     }
```

16.4.3.26 `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 198 of file curvebuilder.hpp.

References `_np`.

Referenced by `write_lp()`.

```
199     {
200         return _np ;
201     }
```

16.4.3.27 `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 715 of file curvebuilder.hpp.

Referenced by main().

```

716     {
717         std::string message ;
718         switch ( error ) {
719             case GLP_EBADB :
720                 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." ;
721                 break ;
722             case GLP_ESING :
723                 message = "Unable to start the search because the basis matrix corresponding to the initial
basis is singular within the working precision." ;
724                 break ;
725             case GLP_ECOND :
726                 message = "Unable to start the search because the basis matrix corresponding to the initial
basis is ill-conditioned." ;
727                 break ;
728             case GLP_EBOUND :
729                 message = "Unable to start the search because some double-bounded variables have incorrect
bounds." ;
730                 break ;
731             case GLP_EFAIL :
732                 message = "The search was prematurely terminated due to the solver failure." ;
733                 break ;
734             case GLP_EOBJLL :
735                 message = "The search was prematurely terminated because the objective function being maximized
has reached its lower limit and continues decreasing." ;
736                 break ;
737             case GLP_EOBJUL :
738                 message = "The search was prematurely terminated because the objective function being minimized
has reached its upper limit and continues increasing." ;
739                 break ;
740             case GLP_EITLIM :
741                 message = "The search was prematurely terminated because the simplex iteration limit has been
exceeded." ;
742                 break ;
743             case GLP_ETMLIM :
744                 message = "The search was prematurely terminated because the time limit has been exceeded." ;
745                 break ;
746             case GLP_ENOPFS :
747                 message = "The LP problem instance has no primal feasible solution." ;
748                 break ;
749             case GLP_ENODFS :
750                 message = "The LP problem instance has no dual feasible solution." ;
751                 break ;
752             default :
753                 message = "Unknown reason." ;
754                 break ;
755         }
756         return message ;
757     }
758 }
```

16.4.3.28 double channel::CurveBuilder::get_upper_bound_on_second_difference_value (const size_t *p*, const size_t *i*, const size_t *v*) const throw ExceptionObject [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 648 of file curvebuilder.hpp.

```

654 {
655     if ( p >= _np ) {
656         std::stringstream ss( std::stringstream::in | std::stringstream::out );
657         ss << "Index of the curve is out of range" ;
658         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
659     }
660
661     if ( ( i < 1 ) || ( i > 3 ) ) {
662         std::stringstream ss( std::stringstream::in | std::stringstream::out );
663         ss << "Index of the second difference vector is out of range" ;
664         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
665     }
666
667     if ( v >= 2 ) {
668         std::stringstream ss( std::stringstream::in | std::stringstream::out );
669         ss << "Index of the Cartesian coordinate is out of range" ;
670         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
671     }
672
673     if ( _secdiff.empty() ) {
674         std::stringstream ss( std::stringstream::in | std::stringstream::out );
675         ss << "Second differences have not been computed" ;
676         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
677     }
678
679     size_t index = ( 4 * 2 * p ) + ( 4 * ( i - 1 ) ) + 2 + v ;
680
681     return _secdiff[ index ] ;
682 }
```

16.4.3.29 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>	Bound value (right-hand side of the constraint)

Definition at line 964 of file curvebuilder.hpp.

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().

```

969 {
970     _bounds[ eqline ] = Bound(
971         type ,
972         value ,
```

```

973             eqline
974         ) ;
975
976     return ;
977 }

```

16.4.3.30 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 934 of file curvebuilder.hpp.

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().

```

939     {
940         _coefficients[ eqline ].push_back(
941             Coefficient(
942                 eqline ,
943                 index ,
944                 value
945             )
946         ) ;
947
948     return ;
949 }

```

16.4.3.31 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 2133 of file curvebuilder.cpp.

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().

```

2141 {
2142     double temp ;
2143
2144     temp = lower * normal ;
2145     if ( temp != 0 ) {
2146         insert_coefficient( eqline , sdlo , temp ) ;
2147     }
2148
2149     temp = upper * normal ;
2150     if ( temp != 0 ) {
2151         insert_coefficient( eqline , sdup , temp ) ;
2152     }
2153
2154     return ;
2155 }

```

16.4.3.32 `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.
<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 2198 of file curvebuilder.cpp.

References `insert_coefficient()`.

```

2210 {
2211     double temp = c0 * normal ;
2212     if ( temp != 0 ) {
2213         insert_coefficient( eqline , b0 , temp ) ;
2214     }
2215
2216     temp = c1 * normal ;
2217     if ( temp != 0 ) {
2218         insert_coefficient( eqline , b1 , temp ) ;
2219     }
2220
2221     temp = c2 * normal ;
2222     if ( temp != 0 ) {

```



```

2223     insert_coefficient( eqline , b2 , temp ) ;
2224 }
2225
2226 temp = c3 * normal ;
2227 if ( temp != 0 ) {
2228     insert_coefficient( eqline , b3 , temp ) ;
2229 }
2230
2231 return ;
2232 }

```

16.4.3.33 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 2284 of file curvebuilder.cpp.

References insert_coefficient(), and insert_csegment_constraint().

```

2298 {
2299     insert_csegment_constraint(
2300         eqline ,
2301         c0 ,
2302         c1 ,
2303         c2 ,
2304         c3 ,
2305         b0 ,
2306         b1 ,
2307         b2 ,

```

```

2308             b3 ,
2309             normal
2310         ) ;
2311
2312     double temp = c4 * normal ;
2313     if ( temp != 0 ) {
2314         insert_coefficient( eqline , b4 , temp ) ;
2315     }
2316
2317     return ;
2318 }

```

16.4.3.34 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 1424 of file curvebuilder.cpp.

References insert_bound(), and insert_coefficient().

Referenced by compute_correspondence_constraints().

```

1430 {
1431     for ( size_t i = 0 ; i < 4 ; i++ ) {
1432         insert_coefficient( eqline , col[ i ] , val[ i ] ) ;
1433     }
1434
1435     insert_bound( eqline , Bound::EQT , rhs ) ;
1436
1437     return ;
1438 }

```

16.4.3.35 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>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>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 1850 of file curvebuilder.cpp.

References `insert_csegment_constraint()`.

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

```

1860 {
1861     //
1862     // The coefficients are the same for each Cartesian coordinate.
1863     //
1864     const double onesixth = double( 1 ) / double( 6 ) ;
1865
1866     // The upper and lower bounds on the e-piece points must be on
1867     // or above the lower envelope of the c-th c-segment of the
1868     // channel.
1869
1870     const double c0 = onesixth * ( 1 - s ) ;
1871     const double c1 = ( ( -2 + 3 * s ) * onesixth ) + ( p + 4 - t ) ;
1872     const double c2 = ( ( 1 - 3 * s ) * onesixth ) + ( t - p - 3 ) ;
1873     const double c3 = onesixth * s ;
1874
1875     for ( size_t v = 0 ; v < 2 ; v++ ) {
1876         //
1877         // Compute constraints for the v-th Cartesian coordinate.
1878         //
1879         insert_csegment_constraint(
1880             eqline ,
1881             c0 ,
1882             c1 ,
1883             c2 ,
1884             c3 ,
1885             cp[ 0 ][ v ] ,
1886             cp[ 1 ][ v ] ,
1887             cp[ 2 ][ v ] ,
1888             cp[ 3 ][ v ] ,
1889             nl[ c ][ v ]
1890         ) ;
1891
1892         // The upper and lower bounds on the e-piece points must be on
1893         // or below the upper envelope of the c-th c-segment of the
1894         // channel.
1895         insert_csegment_constraint(
1896             eqline + 1 ,
1897             c0 ,
1898             c1 ,
1899             c2 ,
1900             c3 ,
1901             cp[ 0 ][ v ] ,
1902             cp[ 1 ][ v ] ,
1903             cp[ 2 ][ v ] ,
1904             cp[ 3 ][ v ] ,
1905             nu[ c ][ v ]
1906         ) ;
1907     }
1908     return ;
1909 }

```

16.4.3.36 `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 1939 of file `curvebuilder.cpp`.

References `insert_csegment_constraint()`.

```

1948 {
1949     //
1950     // The coefficients are the same for each Cartesian coordinate.
1951     //
1952     const double onesixth = double( 1 ) / double( 6 ) ;
1953
1954     // The upper and lower bounds on the e-piece point must be either
1955     // on the left or on the right side of a c-section of the channel.
1956
1957     const double c0 = onesixth * ( 1 - s ) ;
1958     const double c1 = ( ( -2 + 3 * s ) * onesixth ) + ( p + 4 - t ) ;
1959     const double c2 = ( ( 1 - 3 * s ) * onesixth ) + ( t - p - 3 ) ;
1960     const double c3 = onesixth * s ;
1961
1962     for ( size_t v = 0 ; v < 2 ; v++ ) {
1963         //
1964         // Compute constraints for the v-th Cartesian coordinate.
1965         //
1966
1967         //
1968         // Lower bound --> Equation eqline
1969         //
1970         insert_csegment_constraint(
1971             eqline ,
1972             c0 ,
1973             c1 ,
1974             c2 ,
1975             c3 ,
1976             cp[ 0 ][ v ] ,
1977             cp[ 1 ][ v ] ,
1978             cp[ 2 ][ v ] ,
1979             cp[ 3 ][ v ] ,
1980             ncsec[ c ][ v ]
1981         ) ;
1982
1983         //
1984         // Upper bound --> Equation eqline + 1
1985         //
1986         insert_csegment_constraint(
1987             eqline + 1 ,
1988             c0 ,
1989             c1 ,
1990             c2 ,
1991             c3 ,
1992             cp[ 0 ][ v ] ,
1993             cp[ 1 ][ v ] ,

```

```

1994             cp[ 2 ][ v ] ,
1995             cp[ 3 ][ v ] ,
1996             ncsec[ c ][ v ]
1997         ) ;
1998     }
1999
2000     return ;
2001 }

```

16.4.3.37 `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 1361 of file curvebuilder.cpp.

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

Referenced by `compute_min_max_constraints()`.

```

1369 {
1370     // First min-max constraint: the upper bound of the second
1371     // difference must be greater than or equal to the value of the
1372     // second difference:
1373
1374     const double onesixth = double( 1 ) / double( 6 ) ;
1375
1376     insert_coefficient( eqline , up , 1 ) ;
1377     insert_coefficient( eqline , b0 , -1 * onesixth ) ;
1378     insert_coefficient( eqline , b1 , 2 * onesixth ) ;
1379     insert_coefficient( eqline , b2 , -1 * onesixth ) ;
1380
1381     insert_bound( eqline , Bound::GTE , 0 ) ;
1382
1383     // Second min-max constraint: the upper bound on the second
1384     // difference must be greater than or equal to zero (i.e., must be
1385     // non-negative).
1386
1387     insert_coefficient( eqline + 1 , up , 1 ) ;
1388
1389     insert_bound( eqline + 1 , Bound::GTE , 0 ) ;
1390
1391     // Third min-max constraint: the sum of the upper and lower bounds
1392     // of the second difference must be equal to the value the second
1393     // difference.
1394
1395     insert_coefficient( eqline + 2 , up , 1 ) ;
1396     insert_coefficient( eqline + 2 , lo , 1 ) ;
1397     insert_coefficient( eqline + 2 , b0 , -1 * onesixth ) ;
1398     insert_coefficient( eqline + 2 , b1 , 2 * onesixth ) ;
1399     insert_coefficient( eqline + 2 , b2 , -1 * onesixth ) ;
1400
1401     insert_bound( eqline + 2 , Bound::EQT , 0 ) ;
1402
1403     return ;
1404 }

```

16.4.3.38 `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 1514 of file curvebuilder.cpp.

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

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

```

1522 {
1523     // Insert into the matrix associated with the linear program (LP)
1524     // the coefficients of the second differences of the e-piece
1525     // breakpoint lower bound  $\{f\}_{\text{stackrel{e}{\sim}}^p\{f\}$  in constraint
1526     // (3a).
1527
1528     //
1529     // The computation is performed for each second difference j.
1530     //
1531
1532     for ( size_t j = 1 ; j < 3 ; j++ ) {
1533         //
1534         // Get lower and upper bounds for the special polynomial.
1535         //
1536         double dl ;
1537         double du ;
1538         evaluate_bounding_polynomial(
1539             j ,
1540             s ,
1541             du , // switch lower and upper bounds.
1542             dl , // switch lower and upper bounds.
1543         ) ;
1544
1545         //
1546         // The coefficients are the same for each Cartesian coordinate.
1547         //
1548
1549         for ( size_t v = 0 ; v < 2 ; v++ ) {
1550             // Point  $\{f\}_{\text{stackrel{e}{\sim}}^p( s ) \{f\}$  of the e-piece must
1551             // be above the lower envelope of the c-th c-segment of the
1552             // channel.
1553             insert_csegment_constraint(
1554                 eqline ,
1555                 dl ,
1556                 du ,
1557                 sd[ j - 1 ][ 0 ][ v ] ,
1558                 sd[ j - 1 ][ 1 ][ v ] ,
1559                 nl[ c ][ v ]
1560             ) ;
1561
1562             // Point  $\{f\}_{\text{stackrel{e}{\sim}}^p( s ) \{f\}$  of the e-piece must
1563             // be below the upper envelope of the c-th c-segment of the
1564             // channel.
1565             insert_csegment_constraint(

```

```

1566             eqline + 1 ,
1567             dl ,
1568             du ,
1569             sd[ j - 1 ][ 0 ][ v ] ,
1570             sd[ j - 1 ][ 1 ][ v ] ,
1571             nu[ c ][ v ]
1572         ) ;
1573     }
1574 }
1575
1576 return ;
1577 }

```

16.4.3.39 `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 1602 of file curvebuilder.cpp.

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

```

1609 {
1610     // Insert into the matrix associated with the linear program (LP)
1611     // the coefficients of the second differences of the e-piece
1612     // breakpoint lower bound \f$\stackrel{e}{\sim}^p \f$ in constraint
1613     // (3c).
1614
1615     //
1616     // The computation is performed for each second difference j.
1617     //
1618
1619     for ( size_t j = 1 ; j < 3 ; j++ ) {
1620         //
1621         // Get lower and upper bounds for the special polynomial.
1622         //
1623         double dl ;
1624         double du ;
1625         evaluate_bounding_polynomial(
1626             j ,
1627             s ,
1628             du , // switch lower and upper bounds.
1629             dl   // switch lower and upper bounds.
1630         ) ;
1631
1632         //
1633         // The coefficients are the same for each Cartesian coordinate.
1634         //
1635
1636         for ( size_t v = 0 ; v < 2 ; v++ ) {
1637             // Point \f$\stackrel{e}{\sim}^p ( s ) \f$ of the e-piece must
1638             // be either on the right side or on the left side of the
1639             // channel c-section.
1640             insert_csegment_constraint(
1641                 eqline ,
1642                 dl ,

```

```

1643         du ,
1644         sd[ j - 1 ][ 0 ][ v ] ,
1645         sd[ j - 1 ][ 1 ][ v ] ,
1646         ncsec[ c ][ v ]
1647     ) ;
1648 }
1649 }
1650
1651     return ;
1652 }

```

16.4.3.40 `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.
<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 1680 of file curvebuilder.cpp.

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

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

```

1688 {
1689     // Insert into the matrix associated with the linear program (LP)
1690     // the coefficients of the second differences of the e-piece
1691     // breakpoint lower bound  $f_{\text{stackrel{r}{l}}\{e\}^p}$  in constraint
1692     // (3a).
1693
1694     //
1695     // The computation is performed for each second difference j.
1696     //

```



```

1697
1698     for ( size_t j = 1 ; j < 3 ; j++ ) {
1699         //
1700         // Get lower and upper bounds for the special polynomial.
1701         //
1702         double dl ;
1703         double du ;
1704         evaluate_bounding_polynomial(
1705             j ,
1706             s ,
1707             dl ,    // DON't switch lower and upper bounds.
1708             du ,    // DON't switch lower and upper bounds.
1709             ) ;
1710
1711         //
1712         // The coefficients are the same for each Cartesian coordinate.
1713         //
1714
1715         for ( size_t v = 0 ; v < 2 ; v++ ) {
1716             // Point \f$\stackrel{e}{\sim}^p( s )$ of the e-piece must
1717             // be above the lower envelope of the c-th c-segment of the
1718             // channel.
1719             insert_csegment_constraint(
1720                 eqline ,
1721                 dl ,
1722                 du ,
1723                 sd[ j - 1 ][ 0 ][ v ] ,
1724                 sd[ j - 1 ][ 1 ][ v ] ,
1725                 nl[ c ][ v ]
1726             ) ;
1727
1728             // Point \f$\stackrel{e}{\sim}^p( s )$ of the e-piece must
1729             // be below the upper envelope of the c-th c-segment of the
1730             // channel.
1731             insert_csegment_constraint(
1732                 eqline + 1 ,
1733                 dl ,
1734                 du ,
1735                 sd[ j - 1 ][ 0 ][ v ] ,
1736                 sd[ j - 1 ][ 1 ][ v ] ,
1737                 nu[ c ][ v ]
1738             ) ;
1739         }
1740     }
1741
1742     return ;
1743 }

```

16.4.3.41 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 1768 of file curvebuilder.cpp.

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

```

1775 {
1776     // Insert into the matrix associated with the linear program (LP)
1777     // the coefficients of the second differences of the e-piece
1778     // breakpoint lower bound \f$\stackrel{\sim}{e}^p\{s\}$ in constraint
1779     // (3c).
1780
1781     //
1782     // The computation is performed for each second difference j.
1783     //
1784
1785     for ( size_t j = 1 ; j < 3 ; j++ ) {
1786         //
1787         // Get lower and upper bounds for the special polynomial.
1788         //
1789         double dl ;
1790         double du ;
1791         evaluate_bounding_polynomial(
1792             j ,
1793             s ,
1794             dl , // DON't switch lower and upper bounds.
1795             du   // DON't switch lower and upper bounds.
1796         ) ;
1797
1798         //
1799         // The coefficients are the same for each Cartesian coordinate.
1800         //
1801
1802         for ( size_t v = 0 ; v < 2 ; v++ ) {
1803             // Point \f$\stackrel{\sim}{e}^p( s )$ \f$ of the e-piece must
1804             // be either on the right side or on the left side of the
1805             // channel c-section.
1806             insert_csegment_constraint(
1807                 eqline ,
1808                 dl ,
1809                 du ,
1810                 sd[ j - 1 ][ 0 ][ v ] ,
1811                 sd[ j - 1 ][ 1 ][ v ] ,
1812                 ncsec[ c ][ v ]
1813             ) ;
1814         }
1815     }
1816
1817     return ;
1818 }

```

16.4.3.42 `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 1465 of file curvebuilder.cpp.

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

Referenced by `compute_correspondence_constraints()`.

```

1472 {
1473     for ( size_t j = 0 ; j < 3 ; j++ ) {
1474         insert_coefficient( eqline + 2 * j , strx[ j ] , 1 ) ;
1475         insert_coefficient( eqline + 2 * j , endx[ j ] , -1 ) ;
1476
1477         insert_bound( eqline + 2 * j , Bound::EQT , 0 ) ;
1478
1479         insert_coefficient( eqline + 2 * j + 1 , stry[ j ] , 1 ) ;
1480         insert_coefficient( eqline + 2 * j + 1 , endy[ j ] , -1 ) ;
1481
1482         insert_bound( eqline + 2 * j + 1 , Bound::EQT , 0 ) ;
1483     }
1484
1485     return ;
1486 }
```

16.4.3.43 `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 2025 of file curvebuilder.cpp.

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

Referenced by `compute_sleeve_corners_in_channel_constraints()`.

```

2031 {
2032     insert_bound( eqline , Bound::LTE , _lxcoords[ c ] * nl[ c ][ 0 ] +
2033         _lycoords[ c ] * nl[ c ][ 1 ] ) ;
2034     insert_bound( eqline + 1 , Bound::LTE , _uxcoords[ c ] * nu[ c ][ 0 ] +
2035         _uycoords[ c ] * nu[ c ][ 1 ] ) ;
2036
2037     insert_bound( eqline + 2 , Bound::LTE , _lxcoords[ c ] * nl[ c ][ 0 ] +
2038         _lycoords[ c ] * nl[ c ][ 1 ] ) ;
2039     insert_bound( eqline + 3 , Bound::LTE , _uxcoords[ c ] * nu[ c ][ 0 ] +
2040         _uycoords[ c ] * nu[ c ][ 1 ] ) ;
2041
2042     return ;
2043 }
```

16.4.3.44 `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 2060 of file `curvebuilder.cpp`.

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

Referenced by `compute_sleeve_inside_csegment_constraints()`.

```

2065 {
2066     insert_bound( eqline      , Bound::LTE , _lxcoords[ c ] * ncsec[ c ][ 0 ] +
    _lycoords[ c ] * ncsec[ c ][ 1 ] ) ;
2067     insert_bound( eqline + 1 , Bound::LTE , _uxcoords[ c ] * ncsec[ c ][ 0 ] +
    _uycoords[ c ] * ncsec[ c ][ 1 ] ) ;
2068
2069     insert_bound( eqline + 2 , Bound::GTE , _lxcoords[ c ] * ncsec[ c ][ 0 ] +
    _lycoords[ c ] * ncsec[ c ][ 1 ] ) ;
2070     insert_bound( eqline + 3 , Bound::GTE , _uxcoords[ c ] * ncsec[ c ][ 0 ] +
    _uycoords[ c ] * ncsec[ c ][ 1 ] ) ;
2071 }
```

16.4.3.45 `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 228 of file `curvebuilder.hpp`.

References `_closed`.

Referenced by `write_lp()`.

```

229 {
230     return _closed ;
231 }
```

16.4.3.46 `bool channel::CurveBuilder::is_equality (const size_t i) const throw ExceptionObject [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 475 of file curvebuilder.hpp.

Referenced by write_lp().

```

476 {
477     if ( _coefficients.empty() ) {
478         std::stringstream ss( std::stringstream::in | std::stringstream::out );
479         ss << "No constraint has been created so far" ;
480         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
481     }
482
483     if ( i >= _coefficients.size() ) {
484         std::stringstream ss( std::stringstream::in | std::stringstream::out );
485         ss << "Constraint index is out of range" ;
486         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
487     }
488
489 #ifdef DEBUGMODE
490     assert( _bounds.size() == _coefficients.size() ) ;
491     assert( _bounds.size() > i ) ;
492 #endif
493
494     return _bounds[ i ].get_type() == Bound::EQT ;
495 }
```

16.4.3.47 bool channel::CurveBuilder::is_greater_than_or_equal_to (const size_t *i*) const throw ExceptionObject)

[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 511 of file curvebuilder.hpp.

Referenced by write_lp().

```

512 {
513     if ( _coefficients.empty() ) {
514         std::stringstream ss( std::stringstream::in | std::stringstream::out );
515         ss << "No constraint has been created so far" ;
516         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
517     }
518
519     if ( i >= _coefficients.size() ) {
520         std::stringstream ss( std::stringstream::in | std::stringstream::out );
521         ss << "Constraint index is out of range" ;
522         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
523     }
524
525 #ifdef DEBUGMODE
526     assert( _bounds.size() == _coefficients.size() ) ;
```

```

527     assert( _bounds.size() > i ) ;
528 #endif
529
530     return _bounds[ i ].get_type() == Bound::GTE ;
531 }

```

16.4.3.48 bool channel::CurveBuilder::is_less_than_or_equal_to (const size_t i) const throw ExceptionObject) [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 548 of file curvebuilder.hpp.

Referenced by write_lp().

```

549 {
550     if ( _coefficients.empty() ) {
551         std::stringstream ss( std::stringstream::in | std::stringstream::out ) ;
552         ss << "No constraint has been created so far" ;
553         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
554     }
555
556     if ( i >= _coefficients.size() ) {
557         std::stringstream ss( std::stringstream::in | std::stringstream::out ) ;
558         ss << "Constraint index is out of range" ;
559         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
560     }
561
562 #ifdef DEBUGMODE
563     assert( _bounds.size() == _coefficients.size() ) ;
564     assert( _bounds.size() > i ) ;
565 #endif
566
567     return _bounds[ i ].get_type() == Bound::LTE ;
568 }

```

16.4.3.49 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 697 of file curvebuilder.hpp.

References _ofvalue.

```

698 {
699     return _ofvalue ;
700 }

```

16.4.3.50 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 2412 of file curvebuilder.cpp.

References `_bounds`, and `_coefficients`.

Referenced by `solve_lp()`.

```

2413 {
2414     /*
2415      * Set up the bounds on the constraints of the problem.
2416      */
2417
2418     for ( size_t j = 0 ; j < _bounds.size() ; j++ ) {
2419 #ifdef DEBUGMODE
2420         assert( j == _bounds[ j ].get_row() ) ;
2421 #endif
2422
2423         int i = int( _bounds[ j ].get_row() + 1 ) ;
2424
2425         std::stringstream ss ( std::stringstream::in | std::stringstream::out ) ;
2426         ss << "c" << i ;
2427         glp_set_row_name( lp , i , ss.str().c_str() ) ;
2428
2429         double val = _bounds[ j ].get_value() ;
2430         if ( _bounds[ j ].get_type() == Bound::LTE ) {
2431             glp_set_row_bnds( lp , i , GLP_UP , 0 , val ) ;
2432         }
2433         else if ( _bounds[ j ].get_type() == Bound::GTE ) {
2434             glp_set_row_bnds( lp , i , GLP_LO , val , 0 ) ;
2435         }
2436         else {
2437             glp_set_row_bnds( lp , i , GLP_FX , val , val ) ;
2438         }
2439     }
2440
2441     /*
2442      * Obtain the coefficients of the constraints of the problem.
2443      */
2444
2445     std::vector< int > ia ; ia.push_back( 0 ) ; // GLPK starts indexing array \e ia at 1
2446     std::vector< int > ja ; ja.push_back( 0 ) ; // GLPK starts indexing array \e ja at 1
2447     std::vector< double > ar ; ar.push_back( 0 ) ; // GLPK starts indexing array \e ar at 1
2448
2449     int h = 0 ;
2450     for ( size_t j = 0 ; j < _coefficients.size() ; j++ ) {
2451         for ( size_t k = 0 ; k < _coefficients[ j ].size() ; k++ ) {
2452 #ifdef DEBUGMODE
2453             assert( _coefficients[ j ][ k ].get_row() == j ) ;
2454 #endif
2455             ia.push_back( int( _coefficients[ j ][ k ].get_row() + 1 ) ) ;
2456             ja.push_back( int( _coefficients[ j ][ k ].get_col() + 1 ) ) ;
2457             ar.push_back( _coefficients[ j ][ k ].get_value() ) ;
2458             ++h ;
2459         }
2460     }
2461
2462     glp_load_matrix(
2463         lp ,
2464         h ,
2465         &ia[ 0 ] ,
2466         &ja[ 0 ] ,
2467         &ar[ 0 ]
2468     ) ;
2469
2470     return ;
2471 }
2472

```

16.4.3.51 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 2643 of file curvebuilder.cpp.

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

Referenced by `solve_lp()`.

```

2644 {
2645     //
2646     // Add the first two second difference bounds to the function.
2647     //
2648     for ( size_t i = 1 ; i < 3 ; i++ ) {
2649         for ( size_t l = 0 ; l < 2 ; l++ ) {
2650             for ( size_t v = 0 ; v < 2 ; v++ ) {
2651                 size_t c = compute_second_difference_column_index(
2652                                     0 ,
2653                                     i ,
2654                                     l ,
2655                                     v
2656                                 ) ;
2657
2658                 if ( l == 0 ) {
2659                     glp_set_obj_coef( lp , int( c ) + 1 , -1 ) ;
2660                 }
2661                 else {
2662                     glp_set_obj_coef( lp , int( c ) + 1 , 1 ) ;
2663                 }
2664             }
2665         }
2666     }
2667
2668     //
2669     // Add the remaining second difference bounds to the function.
2670     //
2671     for ( size_t p = 1 ; p < _np ; p++ ) {
2672         for ( size_t l = 0 ; l < 2 ; l++ ) {
2673             for ( size_t v = 0 ; v < 2 ; v++ ) {
2674                 size_t c = compute_second_difference_column_index(
2675                                     p ,
2676                                     2 ,
2677                                     l ,
2678                                     v
2679                                 ) ;
2680
2681                 if ( l == 0 ) {
2682                     glp_set_obj_coef( lp , int( c ) + 1 , -1 ) ;
2683                 }
2684                 else {
2685                     glp_set_obj_coef( lp , int( c ) + 1 , 1 ) ;
2686                 }
2687             }
2688         }
2689     }
2690
2691     return ;
2692 }

```

16.4.3.52 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

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

Definition at line 2486 of file curvebuilder.cpp.

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().

```

2487 {
2488     //
2489     // Set up bounds for the first two second differences.
2490     //
2491     for ( size_t i = 1 ; i <= 2 ; i++ ) {
2492         for ( size_t l = 0 ; l < 2 ; l++ ) {
2493             for ( size_t v = 0 ; v < 2 ; v++ ) {
2494                 size_t c = compute_second_difference_column_index(
2495                                     0 ,
2496                                     i ,
2497                                     l ,
2498                                     v
2499                                 ) ;
2500                 if ( l == 0 ) {
2501                     std::stringstream ss ( std::stringstream::in | std::stringstream::out ) ;
2502                     if ( v == 0 ) {
2503                         ss << "mx" << i ;
2504                     }
2505                     else {
2506                         ss << "my" << i ;
2507                     }
2508                     glp_set_col_name( lp , int( c ) + 1 , ss.str().c_str() ) ;
2509                     glp_set_col_bnds( lp , int( c ) + 1 , GLP_UP , 0 , 0 ) ;
2510                 }
2511                 else {
2512                     std::stringstream ss ( std::stringstream::in | std::stringstream::out ) ;
2513                     if ( v == 0 ) {
2514                         ss << "px" << i ;
2515                     }
2516                     else {
2517                         ss << "py" << i ;
2518                     }
2519                     glp_set_col_name( lp , int( c ) + 1 , ss.str().c_str() ) ;
2520                     glp_set_col_bnds( lp , int( c ) + 1 , GLP_LO , 0 , 0 ) ;
2521                 }
2522             }
2523         }
2524     }
2525
2526     //
2527     // Set up bounds for the remaining second differences.
2528     //
2529     for ( size_t p = 1 ; p < _np ; p++ ) {
2530         for ( size_t l = 0 ; l < 2 ; l++ ) {
2531             for ( size_t v = 0 ; v < 2 ; v++ ) {
2532                 size_t c = compute_second_difference_column_index(
2533                                     p ,
2534                                     2 ,
2535                                     l ,
2536                                     v
2537                                 ) ;
2538                 if ( l == 0 ) {
2539                     std::stringstream ss ( std::stringstream::in | std::stringstream::out ) ;
2540                     if ( v == 0 ) {
2541                         ss << "mx" << p + 2 ;
2542                     }
2543                     else {
2544                         ss << "my" << p + 2 ;
2545                     }
2546                     glp_set_col_name( lp , int( c ) + 1 , ss.str().c_str() ) ;
2547                     glp_set_col_bnds( lp , int( c ) + 1 , GLP_UP , 0 , 0 ) ;
2548                 }
2549                 else {
2550                     std::stringstream ss ( std::stringstream::in | std::stringstream::out ) ;
2551                     if ( v == 0 ) {
2552                         ss << "px" << p + 2 ;
2553                     }
2554                     else {
2555                         ss << "py" << p + 2 ;
2556                     }
2557                     glp_set_col_name( lp , int( c ) + 1 , ss.str().c_str() ) ;
2558                     glp_set_col_bnds( lp , int( c ) + 1 , GLP_LO , 0 , 0 ) ;
2559                 }
2560             }
2561         }
2562     }
2563
2564     //
2565     // Set up bounds for the first four control points.

```

```

2566 //
2567 for ( size_t i = 0 ; i < 4 ; i++ ) {
2568     for ( size_t v = 0 ; v < 2 ; v++ ) {
2569         size_t c = compute_control_value_column_index(
2570             0 ,
2571             i ,
2572             v
2573         ) ;
2574
2575         std::stringstream ss ( std::stringstream::in | std::stringstream::out ) ;
2576         if ( v == 0 ) {
2577             ss << "x" << i + 1 ;
2578         }
2579         else {
2580             ss << "y" << i + 1 ;
2581         }
2582         glp_set_col_name( lp , int( c ) + 1 , ss.str().c_str() ) ;
2583         glp_set_col_bnds( lp , int( c ) + 1 , GLP_FR , 0 , 0 ) ;
2584     }
2585 }
2586
2587 for ( size_t p = 1 ; p < _np ; p++ ) {
2588     for ( size_t v = 0 ; v < 2 ; v++ ) {
2589         size_t c = compute_control_value_column_index(
2590             p ,
2591             3 ,
2592             v
2593         ) ;
2594
2595         std::stringstream ss ( std::stringstream::in | std::stringstream::out ) ;
2596         if ( v == 0 ) {
2597             ss << "x" << p + 4 ;
2598         }
2599         else {
2600             ss << "y" << p + 4 ;
2601         }
2602         glp_set_col_name( lp , int( c ) + 1 , ss.str().c_str() ) ;
2603         glp_set_col_bnds( lp , int( c ) + 1 , GLP_FR , 0 , 0 ) ;
2604     }
2605 }
2606
2607 size_t s = compute_index_of_endpoint_barycentric_coordinate
( 0 ) ;
2608 std::stringstream ss ( std::stringstream::in | std::stringstream::out ) ;
2609 ss << "st" ;
2610 glp_set_col_name( lp , int( s ) + 1 , ss.str().c_str() ) ;
2611 glp_set_col_bnds( lp , int( s ) + 1 , GLP_DB , 0.40 , 0.60 ) ;
2612
2613 if ( !_closed ) {
2614     size_t e = compute_index_of_endpoint_barycentric_coordinate
( 1 ) ;
2615     std::stringstream ss2 ( std::stringstream::in | std::stringstream::out ) ;
2616     ss2 << "en" ;
2617     glp_set_col_name( lp , int( e ) + 1 , ss2.str().c_str() ) ;
2618     glp_set_col_bnds( lp , int( e ) + 1 , GLP_DB , 0.40 , 0.60 ) ;
2619 }
2620
2621 for ( size_t i = 1 ; i < _nc ; i++ ) {
2622     size_t corner_coord = compute_index_of_corner_barycentric_coordinate
( i ) ;
2623     std::stringstream ss ( std::stringstream::in | std::stringstream::out ) ;
2624     ss << "co" << i ;
2625     glp_set_col_name( lp , int( corner_coord ) + 1 , ss.str().c_str() ) ;
2626     glp_set_col_bnds( lp , int( corner_coord ) + 1 , GLP_DB , 0.40 , 0.60 ) ;
2627 }
2628
2629 return ;
2630 }

```

16.4.3.53 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 2334 of file curvebuilder.cpp.

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

Referenced by `build()`.

```

2338 {
2339     /*
2340      * Create the LP problem.
2341      */
2342     glp_prob* lp = glp_create_prob() ;
2343
2344     /*
2345      * Set up the number of constraints and structural variables.
2346      */
2347     glp_add_rows( lp , int( rows ) ) ;
2348     glp_add_cols( lp , int( cols ) ) ;
2349
2350     /*
2351      * Set the problem as a minimization one.
2352      */
2353     glp_set_obj_dir( lp , GLP_MIN ) ;
2354
2355     /*
2356      * Set up the constraints of the problem.
2357      */
2358     set_up_lp_constraints( lp ) ;
2359
2360     /*
2361      * Define bounds on the structural variables of the problem.
2362      */
2363     set_up_structural_variables( lp ) ;
2364
2365     /*
2366      * Define objective function.
2367      */
2368     set_up_objective_function( lp ) ;
2369
2370     /*
2371      * Set parameters of the solver.
2372      */
2373     glp_smcp param ;
2374     glp_init_smcp( &param ) ;
2375
2376     param.msg_lev = GLP_MSG_OFF ;
2377     param.presolve = GLP_ON ;
2378
2379     /*
2380      * Call the solver.
2381      */
2382
2383     int res = glp_simplex( lp , &param ) ;
2384
2385     if ( res == 0 ) {
2386         /*
2387          * Get the solver result information.
2388          */
2389         get_lp_solver_result_information( lp ) ;
2390     }
2391
2392     /*
2393      * Release memory held by the solver.
2394      */
2395     glp_delete_prob( lp ) ;
2396

```

```
2397     return res ;  
2398 }
```

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

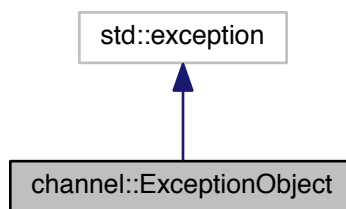
- [curvebuilder.hpp](#)
- [curvebuilder.cpp](#)

16.5 channel::ExceptionObject Class Reference

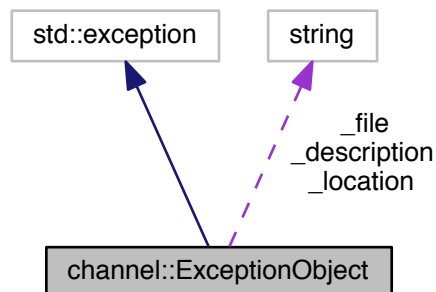
This class extends class *exception* of STL and provides us with a customized way of handling exceptions and showing error messages.

```
#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 In)
Creates an instance of this class.
- [ExceptionObject](#) (const char *file, unsigned int In, const char *desc)
Creates an instance of this class.
- [ExceptionObject](#) (const char *file, unsigned In, 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 76 of file `exceptionobject.hpp`.

16.5.2 Constructor & Destructor Documentation

16.5.2.1 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 126 of file `exceptionobject.hpp`.

```

127 :
128     _location( "Unknown" ) ,
129     _description( "Unknown" ) ,
130     _file( file ) ,
131     _line( ln )
132 {
133 }
```

16.5.2.2 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 149 of file `exceptionobject.hpp`.

```

150 :
151     _location( "Unknown" ) ,
152     _description( desc ) ,
153     _file( file ) ,
154     _line( ln )
155 {
156 }
```

16.5.2.3 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 174 of file exceptionobject.hpp.

```

175 :
176     _location( loc ) ,
177     _description( desc ) ,
178     _file( file ) ,
179     _line( ln )
180 {
181 }
```

16.5.2.4 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 192 of file exceptionobject.hpp.

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

```

192                                     : exception()
193 {
194     _location = xpt._location ;
195     _description = xpt._description ;
196     _file = xpt._file ;
197     _line = xpt._line ;
198 }
```

16.5.3 Member Function Documentation

16.5.3.1 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 322 of file exceptionobject.hpp.

```

323 {
324     return _description.c_str() ;
325 }
```

16.5.3.2 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 338 of file exceptionobject.hpp.

```
339     {  
340         return _file.c_str() ;  
341     }
```

16.5.3.3 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 352 of file exceptionobject.hpp.

References `_line`.

```
353     {  
354         return _line ;  
355     }
```

16.5.3.4 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 307 of file exceptionobject.hpp.

```
308     {  
309         return _location.c_str() ;  
310     }
```

16.5.3.5 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 237 of file exceptionobject.hpp.

```
238     {  
239         return "ExceptionObject" ;  
240     }
```

16.5.3.6 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 279 of file exceptionobject.hpp.

```

280     {
281         _description = s ;
282     }

```

16.5.3.7 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 293 of file exceptionobject.hpp.

```

294     {
295         _description = s ;
296     }

```

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

Assigns a location to this exception.

Parameters

s	A string containing the location.
----------	-----------------------------------

Definition at line 251 of file exceptionobject.hpp.

```

252     {
253         _location = s ;
254     }

```

16.5.3.9 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 265 of file exceptionobject.hpp.

```

266     {
267         _location = s ;
268     }

```

16.5.3.10 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 367 of file exceptionobject.hpp.

```

368     {
369         return _description.c_str() ;
370     }

```

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

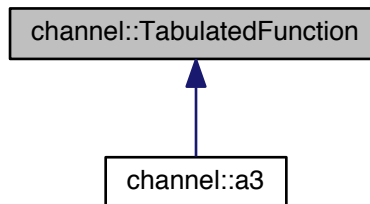
- [exceptionobject.hpp](#)

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 throw (ExceptionObject)
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 throw (ExceptionObject)
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 throw (ExceptionObject)
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 73 of file tabulatedfunction.hpp.

16.6.2 Member Function Documentation

16.6.2.1 `double channel::TabulatedFunction::a (const size_t i , const double u) const throw ExceptionObject` [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 `double channel::TabulatedFunction::alower (const size_t i , const double u) const throw ExceptionObject` [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 `double channel::TabulatedFunction::upper (const size_t i, const double u) const throw ExceptionObject` [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 `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

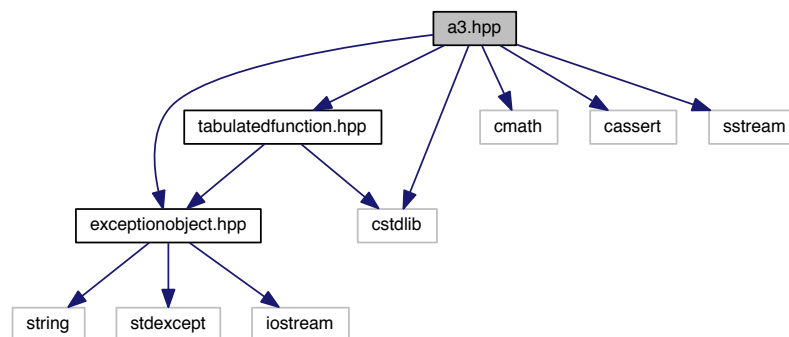
File Documentation

17.1 a3.hpp File Reference

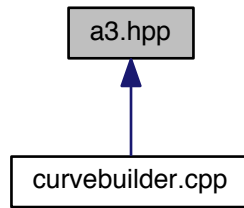
Definition of a class for representing piecewise linear enclosures of certain cubic polynomial functions in Bézier form.

```
#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

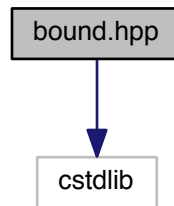
This program is distributed WITHOUT ANY WARRANTY, and it may be freely redistributed under the condition that the copyright notices are not removed, and no compensation is received. Private, research, and institutional use is free. Distribution of this code as part of a commercial system is permissible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

17.2 bound.hpp File Reference

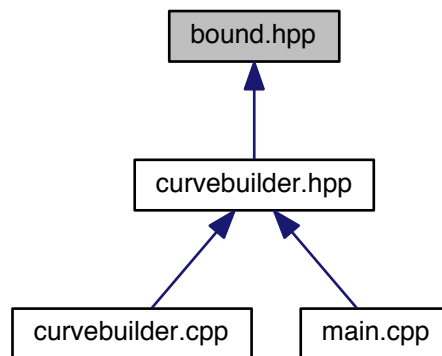
Definition of a class for representing the type of a linear constraint (i.e., equality or inequality) and its right-hand side: a real number.

```
#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

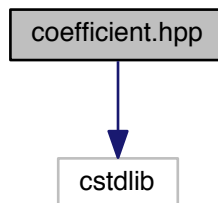
This program is distributed WITHOUT ANY WARRANTY, and it may be freely redistributed under the condition that the copyright notices are not removed, and no compensation is received. Private, research, and institutional use is free. Distribution of this code as part of a commercial system is permissible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

17.3 coefficient.hpp File Reference

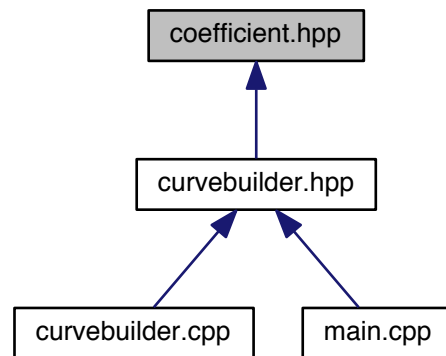
Definition of a class for representing a nonzero coefficient of an unknown of a constraint (inequality or equality) of a linear program instance.

```
#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

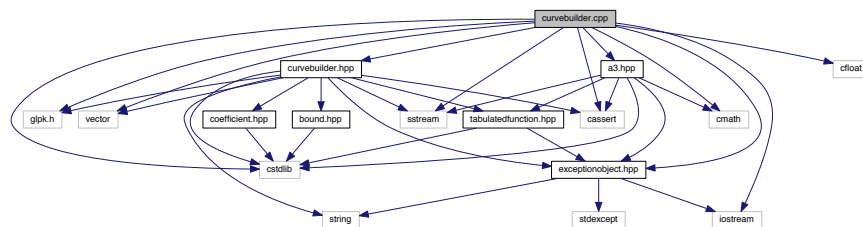
This program is distributed WITHOUT ANY WARRANTY, and it may be freely redistributed under the condition that the copyright notices are not removed, and no compensation is received. Private, research, and institutional use is free. Distribution of this code as part of a commercial system is permissible 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 <cfloat>
#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

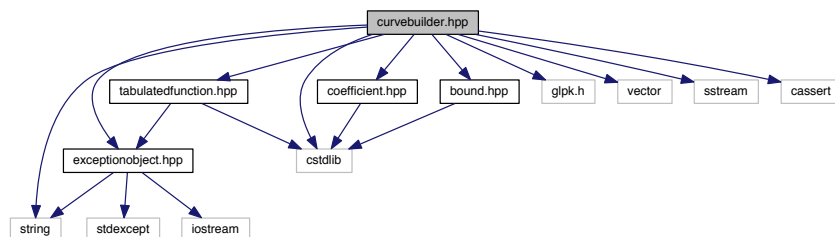
This program is distributed WITHOUT ANY WARRANTY, and it may be freely redistributed under the condition that the copyright notices are not removed, and no compensation is received. Private, research, and institutional use is free. Distribution of this code as part of a commercial system is permissible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

17.5 curvebuilder.hpp File Reference

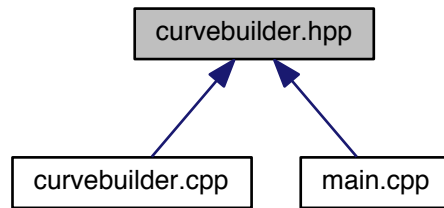
Definition of a class for threading a b-spline curve of degree 3 through a planar channel defined by a pair of polygonal chains.

```
#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

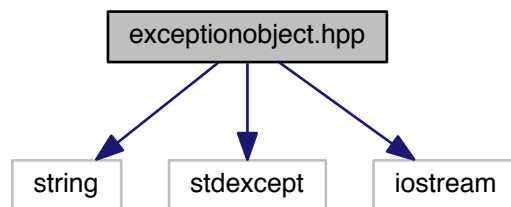
This program is distributed WITHOUT ANY WARRANTY, and it may be freely redistributed under the condition that the copyright notices are not removed, and no compensation is received. Private, research, and institutional use is free. Distribution of this code as part of a commercial system is permissible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

17.6 exceptionobject.hpp File Reference

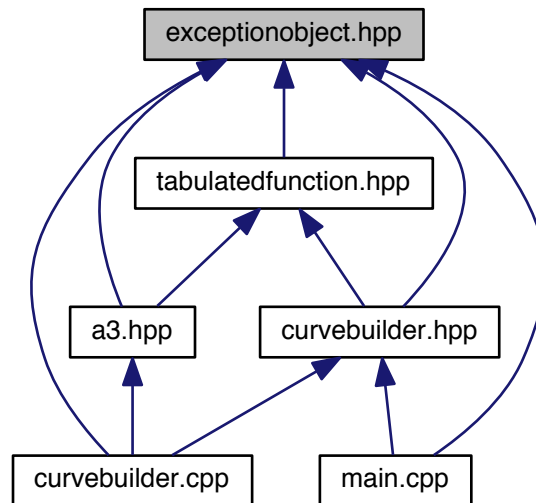
Definition of a class for handling exceptions.

```
#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 #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 42 of file exceptionobject.hpp.

Referenced by channel::CurveBuilder::evaluate_bounding_polynomial(), main(), write_lp(), and write_solution().

17.7 main.cpp File Reference

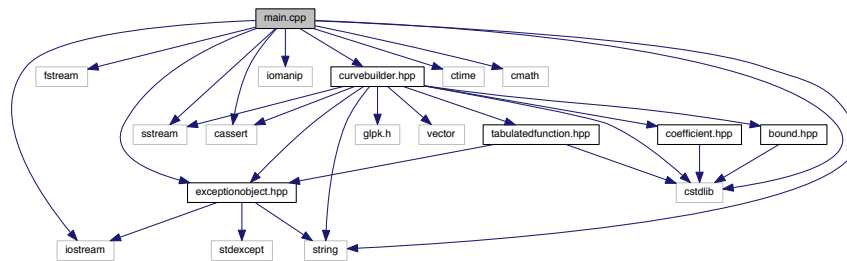
A simple program for testing the channel-2d library.

```

#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <cstdlib>
#include <iomanip>
#include <cassert>
#include <ctime>
#include <cmath>
#include "exceptionobject.hpp"
#include "curvebuilder.hpp"

```

Include dependency graph for main.cpp:



Functions

- void [read_input](#) (const string &fn, size_t &np, size_t &nc, bool &closed, double *&lx, double *&ly, double *&ux, double *&uy) throw (ExceptionObject)
Read in a file describing a polygonal channel.
- void [write_solution](#) (const string &fn, const [CurveBuilder](#) &b)
Write the control points of the b-spline curve to an output file.
- void [write_lp](#) (const string &fn, const [CurveBuilder](#) &b)
Write the instance of the linear program problem solved by this program in CPLEX format. The output file can be given to the gpsolve function of the GNU GLPK or to debug the assembly of the constraints.
- int [main](#) (int argc, char *argv[])
A simple program for testing the bc2d library.

17.7.1 Detailed Description

A simple program for testing the channel-2d library.

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.

17.7.2 Function Documentation**17.7.2.1** `int main (int argc, char * argv[])`

A simple program for testing the bc2d library.

Parameters

<i>argc</i>	The number of command-line arguments.
<i>argv</i>	An array with the command-line arguments.

Returns

An integer number.

Definition at line 125 of file main.cpp.

References `channel::CurveBuilder::build()`, `channel::CurveBuilder::get_solver_error_message()`, `read_input()`, `treat_exception`, `write_lp()`, and `write_solution()`.

```

125                                     {
126     //
127     // Check command-line arguments.
128     //
129
130     if ( ( argc != 3 ) && ( argc != 4 ) ) {
131         cerr << "Usage: "
132              << endl
133              << "\t\t channel2d arg1 arg2 [ arg3 ]"
134              << endl
135              << "\t\t arg1: name of the file describing the polygonal channel"
136              << endl
137              << "\t\t arg2: name of the output file describing the computed cubic b-spline curve"
138              << endl
139              << "\t\t arg3: name of an output file to store a CPLEX format definition of the LP solved by this
program (OPTIONAL)"
140              << endl
141              << endl ;
142         cerr.flush() ;
143
144         return EXIT_FAILURE ;
145     }
146
147     //
148     // Read in the input file.
149     //
150
151     clock_t start, end ;
152

```

```

153     cerr << endl
154         << "Reading file describing a polygonal channel..."
155         << endl ;
156     cerr.flush() ;
157
158     string fn1( argv[ 1 ] ) ;
159
160     size_t np ;
161     size_t nc ;
162     bool closed ;
163     double* lx ;
164     double* ly ;
165     double* ux ;
166     double* uy ;
167
168     start = clock() ;
169     try {
170         read_input( fn1 , np , nc , closed , lx , ly , ux , uy ) ;
171     }
172     catch ( const ExceptionObject& xpt ) {
173         treat_exception( xpt ) ;
174         exit( EXIT_FAILURE ) ;
175     }
176     end = clock() ;
177
178     cerr << ( (double) ( end - start ) ) / CLOCKS_PER_SEC
179         << " seconds."
180         << endl
181         << endl ;
182     cerr.flush() ;
183
184     //
185     // Compute a cubic b-spline curve that passes through the channel.
186     //
187
188     cerr << "Computing a cubic b-spline curve that passes through the channel... "
189         << endl ;
190     cerr.flush() ;
191
192     start = clock() ;
193     CurveBuilder* builder = 0 ;
194     try {
195         builder = new CurveBuilder(
196             np ,
197             nc ,
198             closed ,
199             &lx[ 0 ] ,
200             &ly[ 0 ] ,
201             &ux[ 0 ] ,
202             &uy[ 0 ]
203         ) ;
204     }
205     catch ( const ExceptionObject& xpt ) {
206         treat_exception( xpt ) ;
207         exit( EXIT_FAILURE ) ;
208     }
209
210     int error ;
211     bool res = builder->build( error ) ;
212     end = clock() ;
213
214     cerr << ( (double) ( end - start ) ) / CLOCKS_PER_SEC
215         << " seconds."
216         << endl
217         << endl ;
218     cerr.flush() ;
219
220     if ( res ) {
221         //
222         // Write the control points of the b-spline to a file.
223         //
224         cerr << "Writing out the control points of the b-spline to a file..."
225             << endl ;
226         cerr.flush() ;
227
228         start = clock() ;
229         string fn2( argv[ 2 ] ) ;
230         write_solution(
231             fn2 ,
232             *builder
233         ) ;

```

```

234     end = clock() ;
235
236     cerr << ( (double) ( end - start ) ) / CLOCKS_PER_SEC
237         << " seconds."
238         << endl
239         << endl ;
240     cerr.flush() ;
241 }
242 else {
243     //
244     // Print the error message returned by the LP solver.
245     //
246     cerr << endl
247         << "ATTENTION: "
248         << endl
249         << builder->get_solver_error_message( error )
250         << endl
251         << endl ;
252 }
253
254 //
255 // Generate a description of the linear program in CPLEX format.
256 //
257 if ( argc == 4 ) {
258     cerr << "Writing out a description of the linear program in CPLEX format..."
259         << endl ;
260     cerr.flush() ;
261
262     start = clock() ;
263     string fn3( argv[ 3 ] ) ;
264     write_lp(
265         fn3 ,
266         *builder
267     ) ;
268     end = clock() ;
269
270     cerr << ( (double) ( end - start ) ) / CLOCKS_PER_SEC
271         << " seconds."
272         << endl
273         << endl ;
274     cerr.flush() ;
275 }
276
277 //
278 // Release memory
279 //
280
281 cerr << "Releasing memory..."
282     << endl ;
283 cerr.flush() ;
284
285 start = clock() ;
286 if ( lx != 0 ) delete[ ] lx ;
287 if ( ly != 0 ) delete[ ] ly ;
288 if ( ux != 0 ) delete[ ] ux ;
289 if ( uy != 0 ) delete[ ] uy ;
290 if ( builder != 0 ) delete builder ;
291 end = clock() ;
292
293 cerr << ( (double) ( end - start ) ) / CLOCKS_PER_SEC
294 << " seconds."
295 << endl
296 << endl ;
297 cerr.flush() ;
298
299 //
300 // Done.
301 //
302
303 cerr << "Finished."
304     << endl
305     << endl
306     << endl ;
307 cerr.flush() ;
308
309 return EXIT_SUCCESS ;
310 }

```

17.7.2.2 `void read_input (const string & fn, size_t & np, size_t & nc, bool & closed, double *& lx, double *& ly, double *& ux, double *& uy) throw ExceptionObject)`

Read in a file describing a polygonal channel.

Parameters

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

Definition at line 333 of file main.cpp.

Referenced by main().

```

344 {
345     //
346     // Open the input file
347     //
348     std::ifstream in( fn.c_str() );
349
350     if ( in.is_open() ) {
351         //
352         // Read in the number of segments of the b-spline.
353         //
354         in >> np ;
355
356         //
357         // Read in the number of c-segments of the channel.
358         //
359         in >> nc ;
360
361         //
362         // Read in the flag indicating whether the channel is closed.
363         //
364         unsigned flag ;
365         in >> flag ;
366
367         if ( ( flag != 0 ) && ( flag != 1 ) ) {
368             std::stringstream ss( std::stringstream::in | std::stringstream::out );
369             ss << "Flag value indicating whether the channel is closed or open is invalid" ;
370             in.close() ;
371             throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
372         }
373
374         closed = ( flag == 1 ) ;
375
376         if ( closed ) {
377             if ( np < 4 ) {
378                 std::stringstream ss( std::stringstream::in | std::stringstream::out );
379                 ss << "The number of curve segments must be at least 4 for a closed curve" ;
380                 in.close() ;
381                 throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
382             }
383             if ( nc < 3 ) {
384                 std::stringstream ss( std::stringstream::in | std::stringstream::out );
385                 ss << "The number of segments of a closed channel must be at least 3" ;
386                 in.close() ;
387                 throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
388             }
389         }
390         else {
391             if ( np < 1 ) {
392                 std::stringstream ss( std::stringstream::in | std::stringstream::out );
393                 ss << "The number of curve segments must be at least 1" ;
394                 in.close() ;
395                 throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
396             }
397             if ( nc < 1 ) {
398                 std::stringstream ss( std::stringstream::in | std::stringstream::out );

```

```

399         ss << "The number of segments of an open channel must be at least 1" ;
400         in.close() ;
401         throw ExceptionObject( __FILE__ , __LINE__ , ss.str().c_str() ) ;
402     }
403 }
404
405 //
406 // Read in the channel vertex coordinates.
407 //
408 const size_t nn = ( closed ) ? nc : ( nc + 1 ) ;
409
410 lx = new double[ nn ] ;
411 ly = new double[ nn ] ;
412
413 for ( size_t i = 0 ; i < nn ; i++ ) {
414     //
415     // Read in the X and Y coordinates of the i-th vertex.
416     //
417     in >> lx[ i ] ;
418     in >> ly[ i ] ;
419 }
420
421 ux = new double[ nn ] ;
422 uy = new double[ nn ] ;
423
424 for ( size_t i = 0 ; i < nn ; i++ ) {
425     //
426     // Read in the X and Y coordinates of the i-th vertex.
427     //
428     in >> ux[ i ] ;
429     in >> uy[ i ] ;
430 }
431
432 //
433 // Close file
434 //
435
436 in.close() ;
437 }
438
439 return ;
440 }

```

17.7.2.3 void write_lp (const string & fn, const CurveBuilder & b)

Write the instance of the linear program problem solved by this program in CPLEX format. The output file can be given to the *gpsolve* function of the GNU GLPK or to debug the assembly of the constraints.

Parameters

<i>fn</i>	The name of the output file.
<i>b</i>	An instance of the spline curve builder.

Definition at line 518 of file main.cpp.

References `channel::CurveBuilder::get_bound_of_ith_constraint()`, `channel::CurveBuilder::get_coefficient_identfier()`, `channel::CurveBuilder::get_coefficient_value()`, `channel::CurveBuilder::get_degree()`, `channel::CurveBuilder::get_number_of_coefficients_in_the_ith_constraint()`, `channel::CurveBuilder::get_number_of_constraints()`, `channel::CurveBuilder::get_number_of_csegments()`, `channel::CurveBuilder::get_number_of_segments()`, `channel::CurveBuilder::is_curve_closed()`, `channel::CurveBuilder::is_equality()`, `channel::CurveBuilder::is_greater_than_or_equal_to()`, `channel::CurveBuilder::is_less_than_or_equal_to()`, and `treat_exception`.

Referenced by `main()`.

```

522 {
523     //
524     // Create the output file
525     //
526
527     const size_t np = b.get_number_of_segments() ;
528     const size_t dg = b.get_degree() ;

```



```

529
530 const size_t NumberOfControlPoints = np + dg ;
531 const size_t NumberOfSecondDifferences = ( np - 1 ) * ( dg - 2 ) + ( dg - 1 ) ;
532
533 std::ofstream ou( fn.c_str() ) ;
534
535 if ( ou.is_open() ) {
536     //
537     // Set the precision of the floating-point numbers.
538     //
539     ou << std::setprecision( 6 ) << std::fixed ;
540
541     //
542     // Write the objective function
543     //
544     ou << "Minimize"
545         << std::endl ;
546     ou << '\t'
547         << "obj: " ;
548
549     size_t j = 1 ;
550     for ( size_t i = 0 ; i < NumberOfSecondDifferences ; i++ ) {
551         ou << "-mx" << j << " " ;
552         ou << "-my" << j << " " ;
553         ou << "+px" << j << " " ;
554         ou << "+py" << j << " " ;
555         ++j ;
556     }
557
558     ou << std::endl ;
559
560     //
561     // Write the constraints
562     //
563     ou << "Subject To" << std::endl ;
564
565     try {
566         for( size_t i = 0 ; i < b.get_number_of_constraints() ; i++ ) {
567             //
568             // Write out the number of the constraint.
569             //
570             ou << '\t' << "c" << i + 1 << ": " ;
571
572             //
573             // Get the coefficients of the i-th constraint.
574             //
575             for( j = 0 ; j < b.get_number_of_coefficients_in_the_ith_constraint
576                 ( i ) ; ++j ) {
577                 //
578                 // Get the column index of the coefficient.
579                 //
580                 size_t col = b.get_coefficient_identifier( i , j ) ;
581
582                 //
583                 // Get the value of the coefficient.
584                 //
585                 double value = b.get_coefficient_value( i , j ) ;
586
587                 //
588                 // Compute the index of the curve piece associated with the
589                 // coefficient, and find the type of structural variable the
590                 // coefficient is.
591                 //
592                 if ( col < ( 2 * NumberOfControlPoints ) ) {
593                     if ( ( col % 2 ) == 0 ) {
594                         if ( value >= 0 ) {
595                             ou << "+" << value << "x" << ( col >> 1 ) + 1 << " " ;
596                         }
597                     }
598                     else {
599                         ou << value << "x" << ( col >> 1 ) + 1 << " " ;
600                     }
601                 }
602                 else {
603                     if ( value >= 0 ) {
604                         ou << "+" << value << "y" << ( col >> 1 ) + 1 << " " ;
605                     }
606                     else {
607

```

```

609         ou << value << "y" << ( col >> 1 ) + 1 << " " ;
610     }
611 }
612 }
613 else if ( col < ( ( 2 * NumberOfControlPoints ) + ( 4 * NumberOfSecondDifferences ) ) ) {
614     col -= ( 2 * NumberOfControlPoints ) ;
615     size_t ro = col % 4 ;
616     if ( ro == 0 ) {
617         if ( value >= 0 ) {
618             ou << "+" << value << "mx" << ( col / 4 ) + 1 << " " ;
619         }
620         else {
621             ou << value << "mx" << ( col / 4 ) + 1 << " " ;
622         }
623     }
624     else if ( ro == 1 ) {
625         if ( value >= 0 ) {
626             ou << "+" << value << "my" << ( col / 4 ) + 1 << " " ;
627         }
628         else {
629             ou << value << "my" << ( col / 4 ) + 1 << " " ;
630         }
631     }
632     else if ( ro == 2 ) {
633         if ( value >= 0 ) {
634             ou << "+" << value << "px" << ( col / 4 ) + 1 << " " ;
635         }
636         else {
637             ou << value << "px" << ( col / 4 ) + 1 << " " ;
638         }
639     }
640     else if ( ro == 3 ) {
641         if ( value >= 0 ) {
642             ou << "+" << value << "py" << ( col / 4 ) + 1 << " " ;
643         }
644         else {
645             ou << value << "py" << ( col / 4 ) + 1 << " " ;
646         }
647     }
648 }
649 else {
650     col -= ( ( 2 * NumberOfControlPoints ) + ( 4 * NumberOfSecondDifferences ) ) ;
651
652     if ( col == 0 ) {
653         if ( value >= 0 ) {
654             ou << "+" << value << "as" << " " ;
655         }
656         else if ( value < 0 ) {
657             ou << value << "as" << " " ;
658         }
659     }
660     else if ( ( col == 1 ) && !b.is_curve_closed() ) {
661         if ( value >= 0 ) {
662             ou << "+" << value << "ae" << " " ;
663         }
664         else if ( value < 0 ) {
665             ou << value << "ae" << " " ;
666         }
667     }
668     else if ( !b.is_curve_closed() ) {
669         if ( value >= 0 ) {
670             ou << "+" << value << "co" << ( col - 1 ) << " " ;
671         }
672         else if ( value < 0 ) {
673             ou << value << "co" << ( col - 1 ) << " " ;
674         }
675     }
676     else {
677         if ( value >= 0 ) {
678             ou << "+" << value << "co" << col << " " ;
679         }
680         else if ( value < 0 ) {
681             ou << value << "co" << col << " " ;
682         }
683     }
684 }
685 }
686
687 if ( b.is_equality( i ) ) {
688     ou << " = " ;
689 }

```

```

690         else if ( b.is_less_than_or_equal_to( i ) ) {
691             ou << " <= " ;
692         }
693         else {
694 #ifdef DEBUGMODE
695             assert( b.is_greater_than_or_equal_to( i ) ) ;
696 #endif
697             ou << " >= " ;
698         }
699         ou << b.get_bound_of_ith_constraint( i ) << std::endl ;
700     }
701 }
702 }
703 }
704 catch ( const ExceptionObject& xpt ) {
705     treat_exception( xpt ) ;
706     exit( EXIT_FAILURE ) ;
707 }
708 //
709 // Write the bounds
710 //
711 //
712 ou << "Bounds" << std::endl ;
713
714 for ( unsigned k = 0 ; k < NumberOfControlPoints ; k++ ) {
715     ou << '\t' << "x" << k + 1 << " free" << std::endl ;
716     ou << '\t' << "y" << k + 1 << " free" << std::endl ;
717 }
718
719 for ( unsigned k = 0 ; k < NumberOfSecondDifferences ; k++ ) {
720     ou << '\t' << "-inf <= mx" << k + 1 << " <= 0" << std::endl ;
721     ou << '\t' << "-inf <= my" << k + 1 << " <= 0" << std::endl ;
722     ou << '\t' << "0 <= px" << k + 1 << " <= +inf" << std::endl ;
723     ou << '\t' << "0 <= py" << k + 1 << " <= +inf" << std::endl ;
724 }
725
726 ou << '\t' << 0.40 << " <= as <= " << 0.60 << std::endl ;
727
728 if ( !b.is_curve_closed() ) {
729     ou << '\t' << 0.40 << " <= ae <= " << 0.60 << std::endl ;
730 }
731
732 const size_t NumberOfCSegments = b.get_number_of_csegments() ;
733
734 for ( unsigned k = 1 ; k < NumberOfCSegments ; k++ ) {
735     ou << '\t'
736     << 0.40
737     << " <= co"
738     << k
739     << " <= "
740     << 0.60
741     << std::endl ;
742 }
743
744 ou << "End" << std::endl ;
745
746 //
747 // Close file
748 //
749 ou.close() ;
750
751 }
752
753 return ;
754 }
755 }

```

17.7.2.4 void write_solution (const string & fn, const CurveBuilder & b)

Write the control points of the b-spline curve to an output file.

Parameters

<i>fn</i>	The name of the output file.
<i>b</i>	An instance of the b-spline curve builder.

Definition at line 453 of file main.cpp.

References `channel::CurveBuilder::get_control_value()`, `channel::CurveBuilder::get_number_of_control_points()`, and `treat_exception`.

Referenced by `main()`.

```

457 {
458     using std::endl ;
459
460     std::ofstream ou( fn.c_str() ) ;
461
462     if ( ou.is_open() ) {
463         //
464         // Set the precision of the floating-point numbers.
465         //
466
467         ou << std::setprecision( 6 ) << std::fixed ;
468
469         //
470         // Write out the number of control points and the degree of the b-spline.
471         //
472
473         size_t ncp = b.get_number_of_control_points() ;
474
475         ou << ncp
476            << '\t'
477            << 3
478            << endl ;
479
480         for ( size_t i = 0 ; i < ncp ; i++ ) {
481             double x ;
482             double y ;
483             try {
484                 x = b.get_control_value( i , 0 ) ;
485                 y = b.get_control_value( i , 1 ) ;
486             }
487             catch ( const ExceptionObject& xpt ) {
488                 treat_exception( xpt ) ;
489                 ou.close() ;
490                 exit( EXIT_FAILURE ) ;
491             }
492             ou << x << '\t' << y << endl ;
493         }
494
495         //
496         // Close file
497         //
498         ou.close() ;
499     }
500 }
501
502 return ;
503 }
```

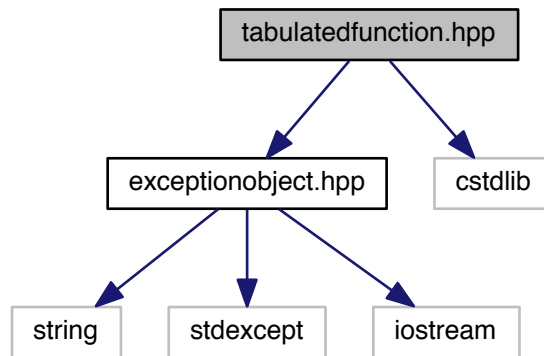
17.8 tabulatedfunction.hpp File Reference

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

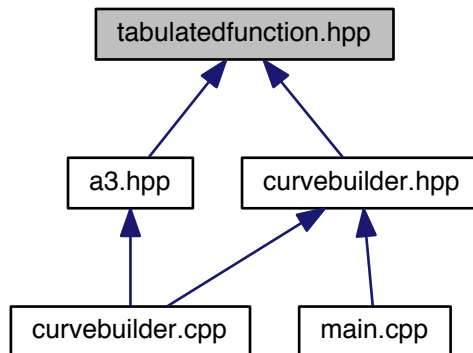
```

#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.

17.8.1 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.

Index

- a
 - channel::TabulatedFunction, 116
 - channel::a3, 41
- a1
 - channel::a3, 42
- a1lower
 - channel::a3, 42
- a1upper
 - channel::a3, 42
- a3.hpp, 119
- alower
 - channel::TabulatedFunction, 116
 - channel::a3, 44
- aupper
 - channel::TabulatedFunction, 116
 - channel::a3, 44
- Bound
 - channel::Bound, 48
- bound.hpp, 121
- build
 - channel::CurveBuilder, 59
- channel, 37
- channel::Bound, 47
 - Bound, 48
 - get_row, 49
 - get_type, 49
 - get_value, 49
- channel::Coefficient, 50
 - Coefficient, 50, 51
 - get_col, 51
 - get_row, 51
 - get_value, 52
- channel::CurveBuilder, 52
 - build, 59
 - compute_channel_corners_outside_sleeve_↔ constraints, 60
 - compute_control_value_column_index, 61
 - compute_correspondence_constraints, 62
 - compute_index_of_corner_barycentric_coordinate, 64
 - compute_index_of_endpoint_barycentric_coordinate, 64
 - compute_min_max_constraints, 65
 - compute_normal_to_csection, 67
 - compute_normal_to_lower_envelope, 67
 - compute_normal_to_upper_envelope, 68
 - compute_second_difference_column_index, 68
 - compute_sleeve_corners_in_channel_constraints, 69
 - compute_sleeve_inside_csegment_constraints, 71
 - CurveBuilder, 57, 58
 - evaluate_bounding_polynomial, 76
 - get_bound_of_ith_constraint, 77
 - get_coefficient_identifier, 77
 - get_coefficient_value, 78
 - get_control_value, 78
 - get_degree, 79
 - get_lower_bound_on_second_difference_value, 79
 - get_lp_solver_result_information, 80
 - get_number_of_coefficients_in_the_ith_constraint, 81
 - get_number_of_constraints, 83
 - get_number_of_control_points, 83
 - get_number_of_csegments, 84
 - get_number_of_segments, 84
 - get_solver_error_message, 84
 - get_upper_bound_on_second_difference_value, 85
 - insert_bound, 86
 - insert_coefficient, 87
 - insert_csegment_constraint, 87–89
 - insert_extreme_point_correspondence_constraint, 90
 - insert_linear_terms_of_epiece_point_bounds, 90, 91
 - insert_min_max_constraints, 93
 - insert_nonlinear_terms_of_epiece_point_lower_↔ bound, 93, 95
 - insert_nonlinear_terms_of_epiece_point_upper_↔ bound, 96, 97
 - insert_periodic_correspondence_constraints, 98
 - insert_rhs_of_sleeve_corners_in_channel_constraints, 99
 - insert_rhs_of_sleeve_inside_csegment_constraints, 99
 - is_curve_closed, 100
 - is_equality, 100
 - is_greater_than_or_equal_to, 101
 - is_less_than_or_equal_to, 102
 - minimum_value, 102
 - set_up_lp_constraints, 102

- set_up_objective_function, 104
 - set_up_structural_variables, 105
 - solve_lp, 107
- channel::ExceptionObject, 109
 - ExceptionObject, 111, 112
 - get_description, 112
 - get_file, 112
 - get_line, 113
 - get_location, 113
 - get_name_of_class, 113
 - set_description, 113, 114
 - set_location, 114
 - what, 114
- channel::TabulatedFunction, 115
 - a, 116
 - alower, 116
 - aupper, 116
 - degree, 118
- channel::a3, 39
 - a, 41
 - a1, 42
 - a1lower, 42
 - a1upper, 42
 - alower, 44
 - aupper, 44
 - degree, 46
 - h, 46
- Coefficient
 - channel::Coefficient, 50, 51
- coefficient.hpp, 122
- compute_channel_corners_outside_sleeve_constraints
 - channel::CurveBuilder, 60
- compute_control_value_column_index
 - channel::CurveBuilder, 61
- compute_correspondence_constraints
 - channel::CurveBuilder, 62
- compute_index_of_corner_barycentric_coordinate
 - channel::CurveBuilder, 64
- compute_index_of_endpoint_barycentric_coordinate
 - channel::CurveBuilder, 64
- compute_min_max_constraints
 - channel::CurveBuilder, 65
- compute_normal_to_csection
 - channel::CurveBuilder, 67
- compute_normal_to_lower_envelope
 - channel::CurveBuilder, 67
- compute_normal_to_upper_envelope
 - channel::CurveBuilder, 68
- compute_second_difference_column_index
 - channel::CurveBuilder, 68
- compute_sleeve_corners_in_channel_constraints
 - channel::CurveBuilder, 69
- compute_sleeve_inside_csegment_constraints
 - channel::CurveBuilder, 71
- CurveBuilder
 - channel::CurveBuilder, 57, 58
- curvebuilder.cpp, 124
- curvebuilder.hpp, 125
- degree
 - channel::TabulatedFunction, 118
 - channel::a3, 46
- evaluate_bounding_polynomial
 - channel::CurveBuilder, 76
- ExceptionObject
 - channel::ExceptionObject, 111, 112
- exceptionobject.hpp, 127
 - treat_exception, 129
- get_bound_of_ith_constraint
 - channel::CurveBuilder, 77
- get_coefficient_identifier
 - channel::CurveBuilder, 77
- get_coefficient_value
 - channel::CurveBuilder, 78
- get_col
 - channel::Coefficient, 51
- get_control_value
 - channel::CurveBuilder, 78
- get_degree
 - channel::CurveBuilder, 79
- get_description
 - channel::ExceptionObject, 112
- get_file
 - channel::ExceptionObject, 112
- get_line
 - channel::ExceptionObject, 113
- get_location
 - channel::ExceptionObject, 113
- get_lower_bound_on_second_difference_value
 - channel::CurveBuilder, 79
- get_lp_solver_result_information
 - channel::CurveBuilder, 80
- get_name_of_class
 - channel::ExceptionObject, 113
- get_number_of_coefficients_in_the_ith_constraint
 - channel::CurveBuilder, 81
- get_number_of_constraints
 - channel::CurveBuilder, 83
- get_number_of_control_points
 - channel::CurveBuilder, 83
- get_number_of_csegments
 - channel::CurveBuilder, 84
- get_number_of_segments
 - channel::CurveBuilder, 84
- get_row
 - channel::Bound, 49
 - channel::Coefficient, 51

- get_solver_error_message
 - channel::CurveBuilder, [84](#)
- get_type
 - channel::Bound, [49](#)
- get_upper_bound_on_second_difference_value
 - channel::CurveBuilder, [85](#)
- get_value
 - channel::Bound, [49](#)
 - channel::Coefficient, [52](#)
- h
 - channel::a3, [46](#)
- insert_bound
 - channel::CurveBuilder, [86](#)
- insert_coefficient
 - channel::CurveBuilder, [87](#)
- insert_csegment_constraint
 - channel::CurveBuilder, [87–89](#)
- insert_extreme_point_correspondence_constraint
 - channel::CurveBuilder, [90](#)
- insert_linear_terms_of_epiece_point_bounds
 - channel::CurveBuilder, [90, 91](#)
- insert_min_max_constraints
 - channel::CurveBuilder, [93](#)
- insert_nonlinear_terms_of_epiece_point_lower_bound
 - channel::CurveBuilder, [93, 95](#)
- insert_nonlinear_terms_of_epiece_point_upper_bound
 - channel::CurveBuilder, [96, 97](#)
- insert_periodic_correspondence_constraints
 - channel::CurveBuilder, [98](#)
- insert_rhs_of_sleeve_corners_in_channel_constraints
 - channel::CurveBuilder, [99](#)
- insert_rhs_of_sleeve_inside_csegment_constraints
 - channel::CurveBuilder, [99](#)
- is_curve_closed
 - channel::CurveBuilder, [100](#)
- is_equality
 - channel::CurveBuilder, [100](#)
- is_greater_than_or_equal_to
 - channel::CurveBuilder, [101](#)
- is_less_than_or_equal_to
 - channel::CurveBuilder, [102](#)
- main
 - main.cpp, [131](#)
- main.cpp, [129](#)
 - main, [131](#)
 - read_input, [133](#)
 - write_lp, [136](#)
 - write_solution, [139](#)
- minimum_value
 - channel::CurveBuilder, [102](#)
- Namespace channel., [35](#)
- read_input
 - main.cpp, [133](#)
- set_description
 - channel::ExceptionObject, [113, 114](#)
- set_location
 - channel::ExceptionObject, [114](#)
- set_up_lp_constraints
 - channel::CurveBuilder, [102](#)
- set_up_objective_function
 - channel::CurveBuilder, [104](#)
- set_up_structural_variables
 - channel::CurveBuilder, [105](#)
- solve_lp
 - channel::CurveBuilder, [107](#)
- tabulatedfunction.hpp, [140](#)
- treat_exception
 - exceptionobject.hpp, [129](#)
- what
 - channel::ExceptionObject, [114](#)
- write_lp
 - main.cpp, [136](#)
- write_solution
 - main.cpp, [139](#)