

Module Interface Specification for Glass-BR

Jingwei Huang

June 3, 2018

Contents

1	Introduction	6
2	Notation	6
3	Module Hierarchy	6
4	MIS of the Input Format Module	7
4.1	Module Name: Input Format	7
4.2	Uses	7
4.2.1	Imported Constants	7
4.2.2	Imported Variables	7
4.2.3	Imported Data Types	7
4.2.4	Imported Access Programs	7
4.3	Interface Syntax	7
4.3.1	Exported Constants	7
4.3.2	Exported Variables	7
4.3.3	Exported Data Types	7
4.3.4	Exported Access Programs	7
4.4	Interface Semantics	8
4.4.1	Environment Variables	8
4.4.2	State Variables	8
4.4.3	State Invariant	8
4.4.4	Assumption	8
4.4.5	Access Program Semantics	8
4.4.6	Considerations	8

5	MIS of the Input Parameters Module	9
5.1	Module Name: Input Parameters	9
5.2	Uses	9
5.3	Interface Syntax	9
5.3.1	Exported Constants	9
5.3.2	Exported Variables	9
5.3.3	Exported Data Types	9
5.3.4	Exported Access Programs	10
5.4	Interface Semantics	10
5.4.1	Environment Variables	10
5.4.2	State Variables	11
5.4.3	State Invariant	11
5.4.4	Assumption	11
5.4.5	Access Program Semantics	11
5.4.6	Considerations	11
6	MIS of the Input Constraints Module	12
6.1	Module Name: Input Constraints	12
6.2	Uses	12
6.2.1	Imported Constants	12
6.2.2	Imported Variables	12
6.2.3	Imported Data Types	12
6.2.4	Imported Access Programs	12
6.3	Interface Syntax	12
6.3.1	Exported Constants	12
6.3.2	Exported Variables	12
6.3.3	Exported Data Types	12
6.3.4	Exported Access Programs	13
6.4	Interface Semantics	13
6.4.1	Environment Variables	13
6.4.2	State Variables	13
6.4.3	State Invariant	13
6.4.4	Assumption	13
6.4.5	Access Program Semantics	13
6.4.6	Considerations	14

7	MIS of Output Format Module	14
7.1	Module Name: Output Format	14
7.2	Uses	14
7.2.1	Imported Constants	14
7.2.2	Imported Variables	14
7.2.3	Imported Data Types	14
7.2.4	Imported Access Programs	15
7.3	Interface Syntax	15
7.3.1	Exported Constants	15
7.3.2	Exported Variables	15
7.3.3	Exported Data Types	15
7.3.4	Exported Access Programs	15
7.4	Interface Semantics	15
7.4.1	Environment Variables	15
7.4.2	State Variables	15
7.4.3	State Invariant	15
7.4.4	Assumption	16
7.4.5	Access Program Semantics	16
8	MIS of Derived Values Module	16
8.1	Module Name: Derived Values	16
8.2	Uses	16
8.2.1	Imported Constants	16
8.2.2	Imported Variables	16
8.2.3	Imported Data Types	16
8.2.4	Imported Access Programs	16
8.3	Interface Syntax	16
8.3.1	Exported Constants	16
8.3.2	Exported Variables	17
8.3.3	Exported Data Types	17
8.3.4	Exported Access Programs	17
8.4	Interface Semantics	17
8.4.1	Environment Variables	17
8.4.2	State Variables	17
8.4.3	State Invariant	17
8.4.4	Assumption	17
8.4.5	Access Program Semantics	17

9	MIS of Calculations Module	18
9.1	Module Name: Calculations	18
9.2	Uses	18
9.2.1	Imported Constants	18
9.2.2	Imported Variables	18
9.2.3	Imported Data Types	18
9.2.4	Imported Access Programs	18
9.3	Interface Syntax	19
9.3.1	Exported Constants	19
9.3.2	Exported Variables	19
9.3.3	Exported Data Types	19
9.3.4	Exported Access Programs	19
9.4	Interface Semantics	19
9.4.1	Environment Variables	19
9.4.2	State Variables	20
9.4.3	State Invariant	20
9.4.4	Assumption	20
9.4.5	Access Program Semantics	20
10	MIS of the Control Module	21
10.1	Module Name: Control	21
10.2	Uses	21
10.2.1	Imported Constants	21
10.2.2	Imported Variables	21
10.2.3	Imported Data Types	21
10.2.4	Imported Access Programs	21
10.3	Interface Syntax	22
10.3.1	Exported Constants	22
10.3.2	Exported Variables	22
10.3.3	Exported Data Types	22
10.3.4	Exported Access Programs	22
10.4	Interface Semantics	22
10.4.1	Environment Variables	22
10.4.2	State Variables	22
10.4.3	State Invariant	22
10.4.4	Assumption	22
10.4.5	Access Program Semantics	23

11 MIS of the Interpolation Data Module	23
11.1 Module Name: Interpolation Data	23
11.2 Uses	23
11.2.1 Imported Constants	23
11.2.2 Imported Variables	23
11.2.3 Imported Data Types	23
11.2.4 Imported Access Programs	23
11.3 Interface Syntax	23
11.3.1 Exported Constants	23
11.3.2 Exported Variables	24
11.3.3 Exported Data Types	24
11.3.4 Exported Access Programs	24
11.4 Interface Semantics	24
11.4.1 Environment Variables	24
11.4.2 State Variables	24
11.4.3 State Invariant	24
11.4.4 Assumption	24
11.4.5 Access Program Semantics	24
12 MIS of the Interpolation Module	25
12.1 Module Name: Interpolation	25
12.2 Uses	25
12.3 Interface Syntax	25
12.3.1 Exported Constants	25
12.3.2 Exported Variables	25
12.3.3 Exported Data Types	25
12.3.4 Exported Access Programs	26
12.4 Interface Semantics	26
12.4.1 Environment Variables	26
12.4.2 State Variables	26
12.4.3 State Invariant	26
12.4.4 Assumption	26
12.4.5 Access Program Semantics	27
12.4.6 Local Functions	27
12.4.7 Local Data Types	27
12.4.8 Considerations	27

1 Introduction

The following document details the Module Interface Specifications for the implemented modules in a program Glass-BR. It is intended to ease navigation through the program for design and maintenance purposes. Complementary documents include the [System Requirement Specifications](#) (SRS) and [Module Guide](#) (MG).

2 Notation

Glass-BR uses three primitive data types: Boolean, Integer and Float. Glass-BR also uses derived data types: Array, String, Tuple and File. These data types are summarized in the following table. The table lists the name of the data type, its notation in this document, and a description of the data type.

Data Type	Notation	Description
Boolean	<i>boolean</i>	an element of {true, false}
Integer	<i>integer</i>	a number without a fractional component in $(-\infty, \infty)$
Float	<i>float</i>	any number in $(-\infty, \infty)$
Array	<i>array</i>	a collection of data items of the same kind
String	<i>string</i>	a varying length array of characters
Tuple	<i>tuple</i>	a collection of elements of possibly different types
File	<i>FILE*</i>	a pointer to an input or output file

3 Module Hierarchy

To view the Module Hierarchy, see section 3 of the [MG](#).

4 MIS of the Input Format Module

4.1 Module Name: Input Format

4.2 Uses

4.2.1 Imported Constants

None

4.2.2 Imported Variables

params

4.2.3 Imported Data Types

Param: tuple

4.2.4 Imported Access Programs

None

4.3 Interface Syntax

4.3.1 Exported Constants

None

4.3.2 Exported Variables

None

4.3.3 Exported Data Types

None

4.3.4 Exported Access Programs

Routine Name	In	Out	Exceptions
<i>s_get_input</i>	FILE *, tuple	-	badFile \vee badFormat

4.4 Interface Semantics

4.4.1 Environment Variables

filename: FILE * (input file)

4.4.2 State Variables

params.a, *params.b*, *params.t*, *params.w*, *params.tnt*, *sd_x*, *sd_y*, *sd_z*,

params.pb_{tol}: float

params.gt: string

params.sdvect: tuple of (*sd_x*: float, *sd_y*: float, *sd_z*: float)

4.4.3 State Invariant

None

4.4.4 Assumption

The user input values are properly constrained and the data structure Param has been initialized.

4.4.5 Access Program Semantics

- *s_get_input(filename, params)*:

Transition: *params.a*, *params.b*, *params.t*, *params.gt*,
params.w, *params.tnt*, *sd_x*, *sd_y*, *sd_z*, *params.sdvect*,
params.pb_{tol} := *a*, *b*, *t*, *gt*, *w*, *tnt*, *sd_x*, *sd_y*, *sd_z*, (*sd_x*,
sd_y, *sd_z*), *pb_{tol}* stored in the input file

Exceptions: *exc* :=
(error reading file \Rightarrow badFile
| inconsistent input format \Rightarrow badFormat)

4.4.6 Considerations

The data type of *params.t* has been mutated from String to Float.

5 MIS of the Input Parameters Module

5.1 Module Name: Input Parameters

5.2 Uses

N/A

5.3 Interface Syntax

5.3.1 Exported Constants

```
#define params.E  $7.17 \times 10^7$   
#define params.t_d 3  
#define params.m 7  
#define params.k  $2.86 \times 10^{-53}$   
#define params.lsf 1
```

5.3.2 Exported Variables

params

5.3.3 Exported Data Types

Param: tuple

The parameters for this structure correspond to the state variables of this module, listed in section [5.4.2](#).

5.3.4 Exported Access Programs

Routine Name	In	Out	Exceptions
<i>s_params.a</i>	float	-	-
<i>g_params.a</i>	-	float	-
<i>s_params.b</i>	float	-	-
<i>g_params.b</i>	-	float	-
<i>s_params.t</i>	string	-	-
<i>g_params.t</i>	-	string	-
<i>s_params.gt</i>	string	-	-
<i>g_params.gt</i>	-	string	-
<i>s_params.w</i>	float	-	-
<i>g_params.w</i>	-	float	-
<i>s_params.tnt</i>	float	-	-
<i>g_params.tnt</i>	-	float	-
<i>s_params.sdvect</i>	tuple	-	-
<i>g_params.sdvect</i>	-	tuple	-
<i>s_params.pb_{tol}</i>	float	-	-
<i>g_params.pb_{tol}</i>	-	float	-
<i>s_params.asprat</i>	float	-	-
<i>g_params.asprat</i>	-	float	-
<i>s_params.sd</i>	float	-	-
<i>g_params.sd</i>	-	float	-
<i>s_params.h</i>	float	-	-
<i>g_params.h</i>	-	float	-
<i>s_params.gt_f</i>	float	-	-
<i>g_params.gt_f</i>	-	float	-
<i>s_params.ld_f</i>	float	-	-
<i>g_params.ld_f</i>	-	float	-
<i>s_params.wtnt</i>	float	-	-
<i>g_params.wtnt</i>	-	float	-

5.4 Interface Semantics

5.4.1 Environment Variables

None

5.4.2 State Variables

$a, b, w, tnt, pb_{tol}, asprat, sd, h, gtf, ldf, wtnt$: float
 t, gt : string
 $sdvect$: tuple of (sd_x : float, sd_y : float, sd_z : float)

5.4.3 State Invariant

None

5.4.4 Assumption

Before a get function is used, the necessary set functions have been called.

5.4.5 Access Program Semantics

- $s_params.*$:

Transition: $params.a, params.b, params.t, params.gt,$
 $params.w, params.tnt, params.sdvect, params.pb_{tol},$
 $params.asprat, params.sd, params.h, params.gtf,$
 $params.ldf, params.wtnt := 0.0, 0.0, "2.5", "AN",$
 $0.0, 0.0, (0.0, 0.0, 0.0), 0.0, 0.0, 0.0, 0.0, 0.0, 0.0$

Exceptions: none

- $g_params.*$:

Output: $out := a, b, t, gt, w, tnt, sdvect, pb_{tol}, asprat, sd, h,$
 $gtf, ldf, wtnt$ stored in the data structure Param.

Exceptions: none

5.4.6 Considerations

Param is a data structure designed to store the input information entered by the **Input Format Module** and **Derived Values Module**.

6 MIS of the Input Constraints Module

6.1 Module Name: Input Constraints

6.2 Uses

6.2.1 Imported Constants

None

6.2.2 Imported Variables

params

6.2.3 Imported Data Types

Param: tuple

6.2.4 Imported Access Programs

None

6.3 Interface Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Variables

None

6.3.3 Exported Data Types

None

6.3.4 Exported Access Programs

Routine Name	In	Out	Exceptions
<i>s_check_constraints</i>	tuple	-	badLength \vee badWidth \vee badAspectRatio \vee badThickness \vee badTNT \vee badWTNT \vee badSD

6.4 Interface Semantics

6.4.1 Environment Variables

scn : the terminal screen

6.4.2 State Variables

params.a, *params.b*, *params.asprat*, *params.t*, *params.tnt*, *params.wtnt*,
params.sd: float

6.4.3 State Invariant

$params.a > 0$
 $params.b > 0$
 $params.asprat \geq 1 \wedge params.asprat \leq 5$
 $params.t \in \{2.5, 2.7, 3.0, 4.0, 5.0, 6.0, 8.0, 10.0, 12.0, 16.0, 19.0, 22.0\}$
 $params.tnt > 0$
 $params.wtnt \geq 4.5 \wedge params.wtnt \leq 910$
 $params.sd \geq 6 \wedge params.sd \leq 130$

6.4.4 Assumption

The function `get_input` in the **Input Format Module** has been called and the data in the input file have been stored in the data structure `Param`.

6.4.5 Access Program Semantics

- *s_check_constraints(params)*:

Transition: display the exceptions on *scn*
Exceptions: *exc* :=
 $(params.a \leq 0 \vee params.b \leq 0 \Rightarrow \text{badLength or badWidth}$
 $| params.asprat < 1 \vee params.asprat > 5 \Rightarrow \text{badAspectRatio}$
 $| params.t \notin \{2.5, 2.7, 3.0, 4.0, 5.0, 8.0, 10.0, 12.0, 16.0, 19.0, 22.0\} \Rightarrow \text{badThickness}$
 $| params.tnt \leq 0 \Rightarrow \text{badTNT}$
 $| params.wtnt < 4.5 \vee params.wtnt > 910 \Rightarrow \text{badWTNT}$
 $| params.sd < 6 \vee params.sd > 130 \Rightarrow \text{badSD})$

6.4.6 Considerations

The data type of *params.t* has been mutated from String to Float.

7 MIS of Output Format Module

7.1 Module Name: Output Format

7.2 Uses

7.2.1 Imported Constants

None

7.2.2 Imported Variables

params, q, j, q_{tol}, pb, lr, nfl, is_safe1, is_safe2, safe

7.2.3 Imported Data Types

Param: tuple

q, j, q_{tol}, pb, lr, nfl: float

is_safe1, is_safe2: boolean

safe: string

7.2.4 Imported Access Programs

None

7.3 Interface Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Variables

None

7.3.3 Exported Data Types

None

7.3.4 Exported Access Programs

Routine Name	In	Out	Exceptions
<i>s_display_output</i>	string, float, float, float, float, float, float, boolean, boolean, string	-	badPath

7.4 Interface Semantics

7.4.1 Environment Variables

filename: FILE * (output file)

7.4.2 State Variables

None

7.4.3 State Invariant

None

7.4.4 Assumption

The functions in the **Calculations Module** have been called and the values for the imported variables have been calculated.

7.4.5 Access Program Semantics

- *s_display_output(filename, q, j, \hat{q}_{tol} , pb, lr, nrl, is_safe1, is_safe2, safe, params)*:

Transition: display the outputs in the output file *filename*

Exceptions: *exc* := error writing file \Rightarrow badPath

8 MIS of Derived Values Module

8.1 Module Name: Derived Values

8.2 Uses

8.2.1 Imported Constants

```
#define params.td 3  
#define params.m 7
```

8.2.2 Imported Variables

params

8.2.3 Imported Data Types

Param: tuple

8.2.4 Imported Access Programs

None

8.3 Interface Syntax

8.3.1 Exported Constants

```
#define params.ldf 0.2696493494752911
```


8.3.2 Exported Variables

params

8.3.3 Exported Data Types

Param: tuple

8.3.4 Exported Access Programs

Routine Name	In	Out	Exceptions
<i>sg_derived_params</i>	tuple	tuple	badFormat \vee notIndustrialStandard \vee wrongGlassType

8.4 Interface Semantics

8.4.1 Environment Variables

scn: the terminal screen

8.4.2 State Variables

params.asprat, *params.sd*, *params.ldf*, *params.wtnt*, *params.h*,
params.gtf: float

8.4.3 State Invariant

None

8.4.4 Assumption

The function `get_input` in the **Input Format Module** has been called and the data in the input file have been stored in the data structure `Param`.

8.4.5 Access Program Semantics

- *sg_derived_params(params)*:

Transition (display exceptions on *scn*
& Output: | *out* := *params.asprat*, *params.sd*, *params.ldf*,
params.wtnt, *params.h*, *params.gtf* calculated using the functions defined in the **SRS**)
Exceptions: *exc* :=
(inconsistent format \Rightarrow badFormat
| *params.t* \notin {2.5, 2.7, 3.0, 4.0, 5.0, 6.0, 8.0, 10.0, 12.0,
16.0, 19.0, 22.0} \Rightarrow notIndustrialStandard
| *params.gtf* \notin {"AN", "an", "HS", "hs", "FT", "ft"}
 \Rightarrow wrongGlassType)

9 MIS of Calculations Module

9.1 Module Name: Calculations

9.2 Uses

9.2.1 Imported Constants

```
#define E 7.17  $\times$  107
#define m 7
#define k 2.86  $\times$  10-53
#define ldf 0.2696493494752911
#define lsf 1
```

9.2.2 Imported Variables

params

9.2.3 Imported Data Types

Param: tuple

9.2.4 Imported Access Programs

Uses **Interpolation Module** Imports interp

9.3 Interface Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Variables

$q, j, \hat{q}_{tot}, pb, lr, nfl, is_safe1, is_safe2, safe$

9.3.3 Exported Data Types

$q, j, \hat{q}_{tot}, pb, lr, nfl$: float
 is_safe1, is_safe2 : boolean
 $safe$: string

9.3.4 Exported Access Programs

Routine Name	In	Out	Exceptions
g_calc_q	array of floats, array of floats, array of floats, tuple	float	badFormat
g_calc_j	array of floats, array of floats, array of floats, float, tuple	float, float	badFormat
g_calc_pb	float, tuple	float	-
g_calc_lr	float, tuple	float, float	-
g_is_safe	float, float, float, tuple	boolean, boolean, string	-

9.4 Interface Semantics

9.4.1 Environment Variables

None

9.4.2 State Variables

$q, j, \hat{q}_{tol}, pb, lr, nfl$: float
 $w_array, data_sd, data_q, j_array, data_asprat, data_qstar$: array of floats
 is_safe1, is_safe2 : boolean
 $safe$: string

9.4.3 State Invariant

None

9.4.4 Assumption

The `get_input` function in the **Input Format Module** and the `derived_params` function in the **Derived Values Module** have been called and the data in the input file as well as the derived values have been stored in the data structure `Param`. The **Interpolation Module** has been successfully implemented.

9.4.5 Access Program Semantics

- $g_calc_q(w_array, data_sd, data_q, params)$:
Output: $out := q$ calculated using interpolation
Exceptions: $exc := badFormat$
- $g_calc_j(j_array, data_asprat, data_qstar, q, params)$:
Output: $out := j, \hat{q}_{tol}$ calculated using interpolation and the functions defined in the **SRS**
Exceptions: $exc := badFormat$
- $g_calc_pb(j, params)$:
Output: $out := pb$ calculated using the functions defined in the **SRS**
Exceptions: none
- $g_calc_lr(\hat{q}_{tol}, params)$:
Output: $out := lr, nfl$ calculated using the functions defined in the **SRS**
Exceptions: none

- $g_is_safe(pb, lr, q, params)$:

Output: $out :=$
 $(pb < params.pb_{tol} \Rightarrow is_safe1 := True \mid pb \geq$
 $params.pb_{tol} \Rightarrow is_safe1 := False)$
 $\mid (lr > q \Rightarrow is_safe2 := True \mid lr \leq q \Rightarrow is_safe2 :=$
 $False)$
 $\mid (is_safe1 == True \wedge is_safe2 == True \Rightarrow safe :=$
 ‘For the given input parameters, the glass is consid-
 ered safe’ $\mid is_safe1 == False \vee is_safe2 == False$
 $\Rightarrow safe :=$ ‘For the given input parameters, the glass
 is NOT considered safe’)

Exceptions: none

10 MIS of the Control Module

10.1 Module Name: Control

10.2 Uses

10.2.1 Imported Constants

None

10.2.2 Imported Variables

None

10.2.3 Imported Data Types

None

10.2.4 Imported Access Programs

Uses **Input Parameters Module** Imports param
 Uses **Input Format Module** Imports inputFormat
 Uses **Derived Values Module** Imports derivedValues
 Uses **Input Constraints Module** Imports checkConstraints
 Uses **Interpolation Data Module** Imports readTable

Uses **Calculations Module Imports** calculations
 Uses **Output Format Module Imports** outputFormat

10.3 Interface Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Variables

None

10.3.3 Exported Data Types

None

10.3.4 Exported Access Programs

Routine Name	In	Out	Exceptions
<i>s_main</i>	FILE *	-	badFile \vee badPath \vee badFormat

10.4 Interface Semantics

10.4.1 Environment Variables

filename: FILE * (input file and output file)
scn: the terminal screen

10.4.2 State Variables

None

10.4.3 State Invariant

None

10.4.4 Assumption

The imported modules have been successfully implemented.

10.4.5 Access Program Semantics

- *s_main(filename)*:

Transition: (calculate the outputs
| display the results in the output file *filename*
| display the message “Main has been executed and
the results have been written to ‘outputfile’” on *scn*)

Exceptions: *exc* :=
(error reading file \Rightarrow badFile
| error writing file \Rightarrow badPath
| inconsistent input format \Rightarrow badFormat)

11 MIS of the Interpolation Data Module

11.1 Module Name: Interpolation Data

11.2 Uses

11.2.1 Imported Constants

None

11.2.2 Imported Variables

None

11.2.3 Imported Data Types

None

11.2.4 Imported Access Programs

None

11.3 Interface Syntax

11.3.1 Exported Constants

None

11.3.2 Exported Variables

num_col, *array1*, *array2*

11.3.3 Exported Data Types

num_col, *array1*, *array2*: array of floats

11.3.4 Exported Access Programs

Routine Name	In	Out	Exceptions
<i>g_read_table</i>	FILE *	array of floats, array of floats, array of floats	badFile \vee badFormat

11.4 Interface Semantics

11.4.1 Environment Variables

filename: FILE * (input file)

11.4.2 State Variables

num_col, *array1*, *array2*: array of floats

11.4.3 State Invariant

None

11.4.4 Assumption

The user input values are properly constrained.

11.4.5 Access Program Semantics

- *g_read_table(filename)*:

Output: *out* := *num_col*, *array1*, *array2*
Exceptions: *exc* :=
(error reading file \Rightarrow badFile
| inconsistent input format \Rightarrow badFormat)

12 MIS of the Interpolation Module

12.1 Module Name: Interpolation

12.2 Uses

N/A

12.3 Interface Syntax

12.3.1 Exported Constants

None

12.3.2 Exported Variables

*y*₀, *idx*, *jdx*, *kdx*, *num_interp1*, *num_interp2*, *interp_value*

12.3.3 Exported Data Types

*y*₀, *interp_value*: float

idx, *jdx*, *kdx*, *num_interp1*, *num_interp2*: integer

12.3.4 Exported Access Programs

Routine Name	In	Out	Exceptions
<i>g_lin_interp</i>	float, float, float, float, float	float	-
<i>g_find_bounds</i>	array of floats, array of floats, float, float	integer, integer, integer, integer, integer	-
<i>g_interp</i>	integer, integer, integer, integer, integer, array of floats, array of floats, array of floats, float, float	float	-

12.4 Interface Semantics

12.4.1 Environment Variables

None

12.4.2 State Variables

$y_0, y_1, y_2, x_1, x_2, input_param, value1, value2, interp_value$: float
 $idx, jdx, kdx, num_interp1, num_interp2$: integer
 $data1, data2, data3$: array of floats

12.4.3 State Invariant

$idx \geq 0$
 $jdx \geq 0$
 $kdx \geq 0$
 $num_interp1 \in \{0, 1\}$
 $num_interp2 \in \{0, 1, 2, 3\}$

12.4.4 Assumption

None

12.4.5 Access Program Semantics

- $g_lin_interp(y_1, y_2, x_1, x_2, input_param)$:
Output: $out := y_0$ calculated using the linear interpolation algorithm
Exceptions: none
- $g_find_bounds(data1, data2, value1, value2)$:
Output: $out := idx, jdx, kdx, num_interp1, num_interp2$
Exceptions: none
- $g_interp(idx, jdx, kdx, num_interp1, num_interp2, data1, data2, data3, value1, value2)$:
Output: $out := interp_value$
Exceptions: none

12.4.6 Local Functions

- $g_proper_index(index1, index2, data, value)$:
Output: $out := index1$
Exceptions: none

12.4.7 Local Data Types

Imported: $index1, index2$: integer
 $data$: array of floats
 $value$: float

Exported: $index1$: integer

12.4.8 Considerations

The local function finds the proper values for indices (jdx, kdx) in the function `find_bounds`.