

CHEMTABLE

Generated by Doxygen 1.6.1

Thu Jan 15 16:41:15 2015

Contents

1	CHEMTABLE GENERATOR - Beta Version	1
1.1	INTRODUCTION	1
1.2	BETA DIRECTORY CONTENTS	1
1.3	INSTALLATION	2
1.4	RUNNING THE PROGRAM	3
1.5	TESTING	5
1.6	DETAILED DOCUMENTATION	5
2	Namespace Index	7
2.1	Namespace List	7
3	Class Index	9
3.1	Class Hierarchy	9
4	Class Index	11
4.1	Class List	11
5	File Index	13
5.1	File List	13
6	Namespace Documentation	15
6.1	chemtable_io Namespace Reference	15
6.1.1	Detailed Description	16
6.2	combinations Namespace Reference	18
6.2.1	Detailed Description	18
6.2.2	Function Documentation	18
6.2.2.1	combination_mat	18
6.2.2.2	combos	18
6.2.2.3	numcoms	18
6.2.2.4	totnumcoms	18

6.3	findprogvar Namespace Reference	19
6.3.1	Detailed Description	19
6.3.2	Function Documentation	19
6.3.2.1	findC	19
6.4	iofuncs Namespace Reference	20
6.4.1	Detailed Description	20
6.4.2	Function Documentation	20
6.4.2.1	get_progvar	20
6.4.2.2	read_input	20
7	Class Documentation	21
7.1	AdvancedLNM Class Reference	21
7.1.1	Constructor & Destructor Documentation	21
7.1.1.1	AdvancedLNM	21
7.1.2	Member Function Documentation	22
7.1.2.1	LeastNonMonotonic	22
7.2	BetaPDF Class Reference	23
7.2.1	Detailed Description	23
7.2.2	Member Function Documentation	23
7.2.2.1	pdfVal	23
7.3	BruteSort Class Reference	25
7.3.1	Constructor & Destructor Documentation	25
7.3.1.1	BruteSort	25
7.3.2	Member Function Documentation	25
7.3.2.1	sort_data	25
7.4	BubbleSort Class Reference	27
7.4.1	Constructor & Destructor Documentation	27
7.4.1.1	BubbleSort	27
7.4.2	Member Function Documentation	27
7.4.2.1	sort_data	27
7.5	CompVec Class Reference	29
7.5.1	Detailed Description	29
7.6	CubicInterp Class Reference	30
7.6.1	Member Function Documentation	30
7.6.1.1	Interp	30
7.7	DeltaPDF Class Reference	32
7.7.1	Detailed Description	32

7.7.2	Member Function Documentation	32
7.7.2.1	pdfVal	32
7.8	EndPointSlope Class Reference	34
7.8.1	Constructor & Destructor Documentation	34
7.8.1.1	EndPointSlope	34
7.8.2	Member Function Documentation	34
7.8.2.1	MostMonotonic	34
7.9	GLQuad Class Reference	36
7.9.1	Detailed Description	36
7.9.2	Constructor & Destructor Documentation	36
7.9.2.1	GLQuad	36
7.9.3	Member Function Documentation	36
7.9.3.1	integrate	36
7.10	HermiteInterp Class Reference	38
7.10.1	Member Function Documentation	38
7.10.1.1	Interp	38
7.11	Integrator Class Reference	40
7.12	Interpolator Class Reference	41
7.13	LeastNonMono Class Reference	42
7.14	LinInterp Class Reference	43
7.14.1	Member Function Documentation	43
7.14.1.1	Interp	43
7.15	LinRegression Class Reference	45
7.15.1	Constructor & Destructor Documentation	45
7.15.1.1	LinRegression	45
7.15.2	Member Function Documentation	45
7.15.2.1	MostMonotonic	45
7.16	Matrix Class Reference	46
7.17	Matrix3D Class Reference	47
7.18	Matrix4D Class Reference	48
7.19	MaxSlope Class Reference	49
7.20	MonoCheck Class Reference	50
7.20.1	Constructor & Destructor Documentation	50
7.20.1.1	MonoCheck	50
7.20.2	Member Function Documentation	50
7.20.2.1	CheckStrictMonotonicity	50

7.21	PDF Class Reference	51
7.22	iofuncs::ProcFile Class Reference	52
7.22.1	Detailed Description	52
7.22.2	Member Function Documentation	52
7.22.2.1	__init__	52
7.22.2.2	gettitles	52
7.22.2.3	interpolate	52
7.23	SequenceGen Class Reference	53
7.23.1	Detailed Description	53
7.24	SimpleLNM Class Reference	54
7.24.1	Constructor & Destructor Documentation	54
7.24.1.1	SimpleLNM	54
7.24.2	Member Function Documentation	54
7.24.2.1	LeastNonMonotonic	54
7.25	Simpson Class Reference	56
7.25.1	Detailed Description	56
7.25.2	Member Function Documentation	56
7.25.2.1	integrate	56
7.26	Sorting Class Reference	58
7.27	StandardSort Class Reference	59
7.27.1	Constructor & Destructor Documentation	59
7.27.1.1	StandardSort	59
7.27.2	Member Function Documentation	59
7.27.2.1	sort_data	59
7.28	Trapz Class Reference	61
7.28.1	Detailed Description	61
7.28.2	Member Function Documentation	61
7.28.2.1	integrate	61
8	File Documentation	63
8.1	src/convolute.cc File Reference	63
8.1.1	Detailed Description	63
8.1.2	Function Documentation	63
8.1.2.1	convVal	63
8.2	src/fittogrid.cc File Reference	64
8.2.1	Detailed Description	64
8.2.2	Function Documentation	64

8.2.2.1	fittogrid	64
---------	---------------------------	----

Chapter 1

CHEMTABLE GENERATOR - Beta Version

Emmet Cleary, Daniel Floryan, Jeffry Lew, Bruce Perry, Emre Turkoz
APC524 - Fall 2014

1.1 INTRODUCTION

This file includes instructions for building and running the chemtable generator software. This software processes .kg data files produced by FlameMaster, producing a table of chemical source terms (chemtable) and various plots for visualizing the data. The program is run by executing a single Python script (chemtable_io.py) as described below.

1.2 BETA DIRECTORY CONTENTS

This directory contains the following files:

- README
- Makefile - generates C++ code
- Makefile.in - called by Makefile
- Doxyfile - generates documentation
- chemtable_io.py - Python script which executes the program
- chemtable_inputs - stores user inputs for chemtable_io.py

The contents of the subdirectories are:

- src - all C++ source code including .cc, .hm and .i (SWIG) files

- python - contains Python files with helper functions for chemtable_io.py
 - obj - object files are stored here after building
 - mod - SWIG-generated Python modules are stored here after building
 - alglib - source code for the external library, AlgLib v. 3.9cpp
 - test - tests for classes and functions used in the program
 - data - sample data sets to be processed by the program
 - profiling - results of profiling studies
 - output - generated when chemtable_io.py is run, stores program outputs
 - doc - contains Doxygen generated documentation
-

1.3 INSTALLATION

The user interface for this version of the software (and a few helper functions) is written in Python and is ready to use. However, the Python code also contains calls to C++ functions (connected through SWIG) which must be compiled and wrapped before the software can be run, as described below. Building the program generates a variety of shared libraries containing the C++ functions which can be called from Python.

NOTE: when building the program the compiler may issue warnings due to AlgLib and SWIG, but the functions the original code does not generate warnings.

Building the program (compiling and wrapping C++ functions):

- From the current (home) directory, type "make".
- "make cleanall" removes all files generated by make.
- "make cleanlib" removes all external library object files.
- "make clean" removes all compiled source code/modules.

External libraries/Modules/Files needed (*denotes files included with this software):

- Python v. 2.6
- *AlgLib v. 3.9.0 C++ version (for glquad.cc and hermiteinterp.cc)
- SWIG
- Matplotlib
- Numpy
- *Numpy.i for SWIG

Generated shared libraries:

- _convolute.so

- `_fittogrid.so`
- `_integrator.so`
- `_interpolator.so`
- `_leastnonmono.so`
- `_matrix3d.so`
- `_matrix4d.so`
- `_matrix.so`
- `_maxslope.so`
- `_monocheck.so`
- `_pdf.so`
- `_sorting.so`

Capabilities of shared library functions:

- Monotonicity checks and slope tests
- [Sorting](#)
- Probability Density Functions (PDFs)
- Convolution with delta or beta PDFs
- Grid fitting
- Interpolation

1.4 RUNNING THE PROGRAM

After building, the chemtable generation program is run by executing a single Python script (`chemtable_io.py`) as described below. The program (`fittogrid` function) has been parallelized using OpenMP and can be run on multiple cores by changing the appropriate value in the input file.

Command: `./chemtable_io.py` OR `python chemtable_io.py`

Inputs:

- `'chemtable_inputs'` text file (must be in same directory as `chemtable_io.py`)
- `'*.kg'` datafiles (in directory as specified in `chemtable_inputs` file)

Outputs (written to `/output/` directory):

- text data file (name specified in `chemtable_inputs` file) containing tabulated source terms as a function of `Cmean`, `Zmean`, and `Zvar` (4 columns of data)
- `'CvsTemp.pdf'` plot of the chosen progress variable vs. temperature to verify monotonicity

- 'contour_zvar_XXX.pdf' contour plots of the chemical source term vs. Cmean and Zmean for up to 10 values of Zvar
- Several status messages, including the identity of the best progress variable, are printed to the terminal

DETAILED INPUT FILE DESCRIPTION:

Inputs are specified with the following syntax: <inputname:><inputvalue1><inputvalue2><inputvalue3>...

Each input name must appear on only one line. A value for every input must be specified unless it has a DEFAULT value specified. The table below lists the possible inputs and indicates which have default values. Setting 'extrapolate in fittogrid' to 'yes' populates all elements of the final chemtable, but may greatly increase run time. If this option is set to 'no' then values which would require extrapolation are set to -1.

INPUT: DEFAULT <NOTES>

- data file directory: data <can specify a path, eg data/C2H4>
- test species: Y-H2O Y-H2 Y-CO2
- output file name: data_output
- plot all progress variables: yes <options: yes, no>
- skip progress variable optimization: no <options: yes, no>
- extrapolate in fittogrid: no <options: yes, no>
- number of threads: 1 <integer>
- Zpdf: [none] <options: delta, beta>
- sort method: bubble <other options: standard, brute>
- interp method: linear <other options: hermite, cubic>
- least nonmonotonic check: simple <other options: advanced>
- max slope test: linear regression <other options: endpointslope>
- integrator: trapezoid <other options: glquad, simpson>
- StoichMassFrac: 0.055
- glq Number of Nodes: 50 <integer, only required when using glquad inter.>
- length Cgrid: 20 <integer>
- Zvar_max: [none] <integer, only required for beta [PDF](#)>
- Zvar_grid: [none] <integer, only required for beta [PDF](#)>
- Zmean_grid: Z <if Z, set to be same as the Zgrid in Flamelet files, otherwise, an integer>

DETAILED DATAFILE DESCRIPTION:

The program is designed specifically to run on .kg datafiles. The first line of the datafiles is expected to be blank and is ignored. The following requirements exist for processing the datafiles:

- All files in the data file directory specified by the user must be .kg files

- No file may be repeated.
- All files must have the same column headers and the same number of rows of data.
- 2nd line contains column headers, 3rd line and on contain data
- Column headers for production rate should be: 'ProdRate<SPECIES> [kg/m³s]'

Two sets of sample data files are included:

- data/C2H4: full output from FlameMaster for a C2H4 flame
- data/C2H4truncated: a subset of the above selected to be a "well-behaved" test case

1.5 TESTING

Several test functions for both the Python and C++ portions of the code are available in the /test/ directory. All tests are run using Python unittests (C++ functions are tested through their SWIG wrappers). The following test functions are available:

combinations_test.py findprogvar_test.py iofuncs_test.py sorting_test.py maxslope_test.py convolute_test.py integrator_test.py pdf_test.py monotonic_test.py leastnonmono_test.py interpolator_test.py fittogrid_test.py

Command: python XXXX_test.py (runs individual tests)

In addition, there is a Bash script "test_all" which can be used to run all tests with one command.

Command: ./test_all (runs all tests)

1.6 DETAILED DOCUMENTATION

For detailed documentation of all classes and functions used by this program, see the Doxygen-generated HTML documentation.

Chapter 2

Namespace Index

2.1 Namespace List

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

chemtable_io (Executable Python script for chemtable generation, including calling findprogvar to determine the best progress variable)	15
combinations (Module containing functions necessary for creating matrices for calculating all possible combinations of elements of a vector/matrix)	18
findprogvar (Package containing findC, a function which selects the best progress variable based on the given data files)	19
iofuncs (Module containing text file processing functions used by other python scripts)	20

Chapter 3

Class Index

3.1 Class Hierarchy

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

CompVec	29
Integrator	40
GLQuad	36
Simpson	56
Trapz	61
Interpolator	41
CubicInterp	30
HermiteInterp	38
LinInterp	43
LeastNonMono	42
AdvancedLNM	21
SimpleLNM	54
Matrix	46
Matrix3D	47
Matrix4D	48
MaxSlope	49
EndPointSlope	34
LinRegression	45
MonoCheck	50
PDF	51
BetaPDF	23
DeltaPDF	32
iofuncs::ProcFile	52
SequenceGen	53
Sorting	58
BruteSort	25
BubbleSort	27
StandardSort	59

Chapter 4

Class Index

4.1 Class List

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

AdvancedLNM	21
BetaPDF (Evaluates beta PDF and stores values in a Matrix3D object)	23
BruteSort	25
BubbleSort	27
CompVec (Comparator for the standard sorting algorithm)	29
CubicInterp	30
DeltaPDF (Evaluates delta PDF and stores values in a Matrix3D object)	32
EndPointSlope	34
GLQuad (Calculates integral using Gauss-Legendre quadrature)	36
HermiteInterp	38
Integrator	40
Interpolator	41
LeastNonMono	42
LinInterp	43
LinRegression	45
Matrix	46
Matrix3D	47
Matrix4D	48
MaxSlope	49
MonoCheck	50
PDF	51
iofuncs::ProcFile	52
SequenceGen (Sequence generator for the standard sorting algorithm)	53
SimpleLNM	54
Simpson (Calculates integral using Simpson's rule)	56
Sorting	58
StandardSort	59
Trapz (Calculates integral using the trapezoidal method)	61

Chapter 5

File Index

5.1 File List

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

src/advancedlnm.h	??
src/betaPDF.h	??
src/brutesort.h	??
src/bubblesort.h	??
src/convolute.cc	63
src/convolute.h	??
src/cubicinterp.h	??
src/deltaPDF.h	??
src/endpointslope.h	??
src/fittogrid.cc	64
src/fittogrid.h	??
src/glquad.h	??
src/hermiteinterp.h	??
src/integrator.h	??
src/interpolator.h	??
src/leastnonmono.h	??
src/lininterp.h	??
src/linregression.h	??
src/matrix.h	??
src/matrix3d.h	??
src/matrix4d.h	??
src/maxslope.h	??
src/monocheck.h	??
src/pdf.h	??
src/simplelnm.h	??
src/simpson.h	??
src/sorting.h	??
src/standardsort.h	??
src/trapz.h	??

Chapter 6

Namespace Documentation

6.1 chemtable_io Namespace Reference

Executable Python script for chemtable generation, including calling [findprogvar](#) to determine the best progress variable.

Variables

- tuple **fin1** = open('chemtable_inputs')
- list **inputs** = [line.strip().split('\t') for line in fin1]
- tuple **datafiledir** = iof.read_input("data file directory:", inputs, default=['data'])
- tuple **datafiles** = glob.glob("".join(["".join(datafiledir), "/*.kg"])))
- tuple **testspecies** = iof.read_input("test species:", inputs, minargs=0, default=["Y-CO2", "Y-CO", "Y-H2O"])
- dictionary **options** = { }
- list **bestC** = []
- tuple **nofiles** = len(datafiles)
- tuple **filesmatC** = fpv.findC(datafiles, testspecies, bestC, options)
- tuple **sorter** = sorting.BubbleSort(filesmatC)
- tuple **Z** = np.genfromtxt(datafiles[0], unpack=False, skip_header=2, delimiter = "\t", usecols = 0)
- **Zmean** = Z
- list **Zpdf** = options["Zpdf"]
- list **Zvar_grid** = [1]
- list **Zvar_max** = [0]
- tuple **Zvar** = np.linspace(0, float(Zvar_max[0]), int(Zvar_grid[0]))
- tuple **d** = pdf.DeltaPDF(Zmean)
- tuple **ZPoints** = len(Z)
- tuple **ZvarPoints** = len(Zvar)
- tuple **ZmeanPoints** = len(Zmean)
- tuple **pdfValM** = matrix3d.Matrix3D(ZvarPoints, ZmeanPoints, ZPoints)
- tuple **pdfValReturn** = d.pdfVal(Z, pdfValM)
- tuple **dataobj** = iof.ProcFile(datafiles[0])
- tuple **titles** = dataobj.gettitles()
- tuple **rxn_rate_locs** = range(len(bestC[0]))
- int **locflag** = 0

- tuple **species** = list(bestC[1][ii])
- list **speciesprodrate** = species[2:]
- list **integ** = options["Integrator"]
- tuple **Intgr** = integrator.Trapz()
- tuple **NumberNodes** = iof.read_input("glq Number of Nodes:", inputs, minargs=0, default=[50])
- list **convolutedC** = [0]
- list **convolutedST** = [0]
- int **maxC** = 0
- int **maxRate** = 0
- list **file** = datafiles[int(filesmatC.GetVal(kk,1))]
- tuple **massfracs** = np.genfromtxt(file, unpack=False, skip_header=2, delimiter = "\t", usecols = bestC[0])
- tuple **rxnrates** = np.genfromtxt(file, unpack=False, skip_header=2, delimiter = "\t", usecols = rxn_rate_locs)
- tuple **progvar** = np.zeros(ZPoints)
- tuple **rxnRates** = np.zeros(ZPoints)
- tuple **ConvReturn** = convolute.convVal_func(Z, progvar, pdfValM, convolutedC[kk], Intgr)
- int **dim1** = 2
- **dim2** = ZmeanPoints
- **dim3** = ZvarPoints
- **dim4** = nofiles
- tuple **lcgrid** = int(options["LCgrid"][0])
- tuple **cgrid** = np.linspace(0.0, maxC*1.5, lcgrid)
- list **interpmethod** = options["InterpMethod"]
- tuple **interp** = interpolator.LinInterp()
- tuple **datain** = matrix4d.Matrix4D(dim1, dim2, dim3, dim4)
- tuple **dataout** = matrix3d.Matrix3D(dim2, dim3, lcgrid)
- int **extrap** = 1
- tuple **f2gflag** = fittogrid.fittogrid_func(datain, cgrid, interp, dataout, int(options["nothreads"][0]), extrap)
- tuple **FinalData** = np.zeros((lcgrid, dim2, dim3))
- tuple **f** = open("".join(["output/", options["OutputFile"][0]]), 'w')
- int **noplots** = 10
- tuple **plots** = range(noplots)
- tuple **i** = int((j+1)*ZvarPoints/(noplots+2))
- tuple **levels** = np.linspace(-1, maxRate, 50)
- tuple **CS** = plt.contourf(X, Y, FinalData[:, :, i], levels)
- tuple **grid** = pow(10, np.floor(np.log10(maxRate))-1)

6.1.1 Detailed Description

Executable Python script for chemtable generation, including calling [findprogvar](#) to determine the best progress variable. Inputs:

- chemtable_inputs: text file containing user options. Each option is written on it's own line, and if multiple values are required they should be seperated by tabs. Details of the chemtable_inputs file and it's required contents can be found in the README.
- .kg (FlameMaster output) data files: in a directory specified by the user in chemtable_inputs. All files must be unique.

Outputs:

- /output/textfile (name specified by user in chemtable_inputs): 4 columns of data, representing Cmean, Zmean, Zvar, and the Chemical Source Term
- /output/contour_zvar_XXX.pdf *10: 10 contour plots of chemical source term vs. Zmean and Cmean at the values of Zvar specified in the filenames. If the user specifies taht less than 10 values of Zvar should be calculated, then that number of contour plots is generates
- /output/CvsTemp.pdf: plot of the best progress variable (and others if the user desries) vs. Temperature

Note: this Python script relies on C++ functions connected through SWIG, which must generate the following modules:

- matrix, matrix3D, matrix4D
- pdf
- integrator
- convolute
- sorting
- interpolator
- fittogrid

6.2 combinations Namespace Reference

Module containing functions necessary for creating matrices for calculating all possible combinations of elements of a vector/matrix.

Functions

- def [numcoms](#)
- def [totnumcoms](#)
- def [combos](#)
- def [combination_mat](#)

6.2.1 Detailed Description

Module containing functions necessary for creating matrices for calculating all possible combinations of elements of a vector/matrix.

6.2.2 Function Documentation

6.2.2.1 def combinations::combination_mat (*matrix*)

returns a combinations matrix A such that for a row vector x:

x*A gives a row vector y which contains all possible combinations of the elements of x. For example, for x = [x1 x2 x3], then x*A gives:
[x1 x2 x3 x1+x2 x1+x3 x2+x3 x1+x2+x3]

6.2.2.2 def combinations::combos (*k*, *matrix*)

returns a combinations matrix A such that for a row vector x:

x*A gives a row vector y which contains all possible combinations of k elements of x. For example, for x = [x1 x2 x3], k=2, then
x*A gives: [x1+x2 x1+x3 x2+x3]

6.2.2.3 def combinations::numcoms (*n*, *k*)

Returns the number of ways k elements can be selected from a set of size n

6.2.2.4 def combinations::totnumcoms (*n*)

Returns the number of ways elements can be selected from a set of size n in any group size from 1 to n.

6.3 findprogvar Namespace Reference

Package containing findC, a function which selects the best progress variable based on the given data files.

Functions

- def `findC`

Determines the best progress variable for the given data files.

6.3.1 Detailed Description

Package containing findC, a function which selects the best progress variable based on the given data files.

6.3.2 Function Documentation

6.3.2.1 `def findprogvar::findC (datafiles, testspecies, bestC, options)`

Determines the best progress variable for the given data files. This function returns a matrix containing the following: (row1) stoich prog var (row2) file indices (row3) stoich Temps, sorted by row1

This function also produces plots of the progress variables: output/CvsTemp.pdf. Note that a directory "output" must exist in the directory from which this function is called.

The user can skip optimization by specifying options["SkipProgVar"] = 'yes'

Note: this Python script relies on C++ functions connected through SWIG, which must generate the following modules:

- matrix
- sorting
- interpolator
- monochk
- maxslope
- least nonmono

Parameters:

datafiles vector of strings specifying file names or paths for the desired data files. Note that the data files should be .kg files produced by FlameMaster

testspecies vector of strings specifying the species to be considered in the progress variable

bestC vector containing information about the best progress variable (output)

options Python dictionary filled with user specified options for the program. Requires options["sort method"], options["StoichMassFrac"], options["InterpMethod"], options["MaxSlopeTest"], options["PlotAllC"], options["lnmcheck"], and options["SkipProgVar"] to be specified

6.4 iofuncs Namespace Reference

Module containing text file processing functions used by other python scripts.

Classes

- class [ProcFile](#)

Functions

- def [read_input](#)
Function that extracts input options from a text file.
- def [get_progvar](#)
Function that creates a vector containing information about a progress variable.

6.4.1 Detailed Description

Module containing text file processing functions used by other python scripts.

6.4.2 Function Documentation

6.4.2.1 def iofuncs::get_progvar (progvarvec, testspecies, locs, index = 1)

Function that creates a vector containing information about a progress variable. returns [progvarlocsFULL, progvarnames, index]

progvarnames: strings of species for a given PROGVARVEC, eg [0, 1, 0, 0, 1] --> ['Y-CH4', 'Y-O2'] for TESTSPECIES = ['Y-CO', 'Y-CH4', 'Y-H2', 'Y-CO2', 'Y-O2']

progvarlocsFull: a vector containing all elements of LOCS corresponding to non-zero elements of PROGVARVEC, eg [2, 5] for LOCS = [1 2 3 4 5] in the above example.

index: INDEX unchanged

6.4.2.2 def iofuncs::read_input (inputname, inputsarray, minargs = 1, default = None)

Function that extracts input options from a text file. Searches for a row in the text file beginning with the string specified by INPUTNAME. Raises an error if this string appears on multiple lines.

Writes the rest of that line (delimited by tabs) as a vector of strings into INPUTSARRAY.

Raises an error if fewer than MINARGS tab delimited elements are found on the input line.

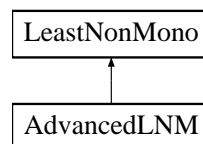
If INPUTNAME is not found, writes returns DEFAULT to INPUTSARRAY, otherwise raises an error.

Chapter 7

Class Documentation

7.1 AdvancedLNM Class Reference

Inheritance diagram for AdvancedLNM::



Public Member Functions

- [AdvancedLNM](#) (const [Matrix](#) &progVar)
Constructor.
- [~AdvancedLNM](#) ()
Destructor.
- int [LeastNonMonotonic](#) (int *monoAry, const int ncols, const int col)
Method to find least monotonic progress variable.

7.1.1 Constructor & Destructor Documentation

7.1.1.1 AdvancedLNM::AdvancedLNM (const Matrix & progVar)

Constructor. [AdvancedLNM](#) is a class that determines the least non-monotonic progress variable with respect to temperature (or another specified column). It determines the progress variable with the smallest percentage of non-unique (one-to-one) points and selects it as the least non-monotonic progress variable.

If two or more progress variables share the smallest percentage, then the progress variable with the greatest magnitude slope (by endpoints) is selected.

7.1.2 Member Function Documentation

7.1.2.1 `int AdvancedLNM::LeastNonMonotonic (int * monoAry, const int ncols, const int col)` `[virtual]`

Method to find least monotonic progress variable. `LeastNonMonotonic` calculates the percentage of non-unique points. The input array `monoAry` will initially be filled with 0s since all progress variables are non-monotonic. This method will select the least non-monotonic and change its value in `monoAry` to 1. `col` is the reference column.

INPUTS:

```
int *monoAry      array containing integer flags that denote the monotonicity of candidate progress variabl
const int ncols   number of columns of monoAry
const int col     the reference column
```

OUTPUT:

```
int              flag specifying whether or not the function succeeded
                  = 0: success
                  != 0: something went wrong
```

Implements [LeastNonMono](#).

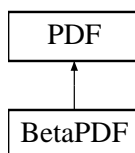
The documentation for this class was generated from the following files:

- `src/advancedlnm.h`
- `src/advancedlnm.cc`

7.2 BetaPDF Class Reference

Evaluates beta [PDF](#) and stores values in a [Matrix3D](#) object.

#include <betaPDF.h> Inheritance diagram for BetaPDF::



Public Member Functions

- [BetaPDF](#) (const double *Zmean, const int ZmeanPoints, const double *Zvar, const int ZvarPoints)
Constructor.
- [~BetaPDF](#) ()
Destructor.
- int [pdfVal](#) (const double *Z, const int ZPoints, [Matrix3D](#) *pdfValM)
Main routine that generates the [PDF](#) values.

7.2.1 Detailed Description

Evaluates beta [PDF](#) and stores values in a [Matrix3D](#) object.

7.2.2 Member Function Documentation

7.2.2.1 int BetaPDF::pdfVal (const double * Z, const int ZPoints, Matrix3D * pdfValM) [virtual]

Main routine that generates the [PDF](#) values. The Beta [PDF](#) uses statistics (means and variances) to generate a [PDF](#). The [PDF](#) values are stored in a [Matrix3D](#) object: dim1 is the variance, dim2 is the mean, and dim3 are the data points.

```

INPUTS:
const double* Z           double array containing mixture fraction values coming from the files

const int ZPoints         number of mixture fraction values in the Z array

Matrix3D* pdfValM         the Matrix3D type container that stores the PDF values

OUTPUT:

int                       flag specifying whether or not the function succeeded
                           = 0: success
!= 0: something went wrong
  
```

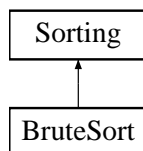
Implements [PDF](#).

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

- `src/betaPDF.h`
- `src/betaPDF.cc`

7.3 BruteSort Class Reference

Inheritance diagram for BruteSort::



Public Member Functions

- [BruteSort](#) ([Matrix](#) *data)
Constructor.
- [~BruteSort](#) ()
Destructor.
- [int sort_data](#) ()
Main function that sorts the given data.
- [void SetRefColNum](#) (int num)
Set the reference column number.

7.3.1 Constructor & Destructor Documentation

7.3.1.1 BruteSort::BruteSort ([Matrix](#) * data)

Constructor. This algorithm goes over the each entry (rows times) at the selected column, finds the maximum entry, records its index. This is repeated rows times until the indices of the entries are obtained in order. Then each column of the matrix is reordered using these indices.

7.3.2 Member Function Documentation

7.3.2.1 [int BruteSort::sort_data](#) () [[virtual](#)]

Main function that sorts the given data. The algorithm processes the reference column and sorts it using the brute sort approach.

INPUT:

There are no inputs. The data to be sorted is already passed via the constructor

OUTPUT:

```

int          flag specifying whether or not the function succeeded
              = 0: success
              != 0: something went wrong
  
```

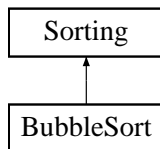
Implements [Sorting](#).

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

- `src/brutesort.h`
- `src/brutesort.cc`

7.4 BubbleSort Class Reference

Inheritance diagram for BubbleSort::



Public Member Functions

- [BubbleSort](#) ([Matrix](#) *data)
- [~BubbleSort](#) ()
Destructor.
- [int sort_data](#) ()
Main sorting body.
- [void SetRefColNum](#) (int num)
Set the reference column number and extract the data of the reference column to the container refColumn_.

7.4.1 Constructor & Destructor Documentation

7.4.1.1 BubbleSort::BubbleSort ([Matrix](#) * *data*)

The constructor duplicates the data from the matrix pointer to datacopy_ object. It also generates the array containing the indices to be used during sorting.

7.4.2 Member Function Documentation

7.4.2.1 [int BubbleSort::sort_data](#) () [[virtual](#)]

Main sorting body. This function processes the reference column with the bubble sorting algorithm.

Details of the bubble sort algorithm can be found from the following link:
http://en.wikipedia.org/wiki/Bubble_sort

INPUT

There are no inputs. The data to be sorted is already passed via the constructor

OUTPUT:

```
int          flag specifying whether or not the function succeeded
              = 0: success
              != 0: something went wrong
```

Implements [Sorting](#).

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

- `src/bubblesort.h`
- `src/bubblesort.cc`

7.5 CompVec Class Reference

Comparator for the standard sorting algorithm.

Public Member Functions

- **CompVec** (double *arr)
- bool **operator()** (size_t i, size_t j)

7.5.1 Detailed Description

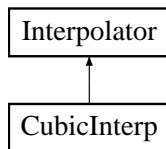
Comparator for the standard sorting algorithm.

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

- src/standardsort.cc

7.6 CubicInterp Class Reference

Inheritance diagram for CubicInterp::



Public Member Functions

- [CubicInterp](#) ()
Constructor.
- [~CubicInterp](#) ()
Destructor.
- [int Interp](#) (const [Matrix](#) *matin, int col, double ival, double *vecout, int cols)
Cubic spline interpolation function.

7.6.1 Member Function Documentation

7.6.1.1 `int CubicInterp::Interp (const Matrix * matin, int col, double ival, double * vecout, int cols) [virtual]`

Cubic spline interpolation function. This function takes in a 2D matrix of data and interpolates an entire row from it using a cubic spline interpolator. Each column of the matrix is treated as a variable, with a specified column being the independent variable. The input data is assumed to be sorted by the independent variable.

INPUTS:

<code>const Matrix *matin</code>	pointer to a Matrix object. This is the input data.
<code>int col</code>	integer specifying which column of the input Matrix is the independent variable
<code>double ival</code>	value at which to interpolate
<code>double *vecout</code>	pointer to an array which contains the interpolated row. This array has the same number of columns as the input Matrix.
<code>int cols</code>	number of columns of matin/vecout

OUTPUTS:

<code>int</code>	flag specifying whether or not the function succeeded = 0: success = 1: extrapolation attempted
------------------	---

Implements [Interpolator](#).

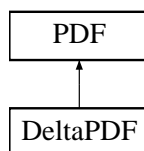
The documentation for this class was generated from the following files:

- src/cubicinterp.h
- src/cubicinterp.cc

7.7 DeltaPDF Class Reference

Evaluates delta [PDF](#) and stores values in a [Matrix3D](#) object.

`#include <deltaPDF.h>`Inheritance diagram for DeltaPDF::



Public Member Functions

- [DeltaPDF](#) (const double *Zmean, const int ZmeanPoints)
Constructor.
- [~DeltaPDF](#) ()
Destructor.
- int [pdfVal](#) (const double *Z, const int ZPoints, [Matrix3D](#) *pdfValM)

7.7.1 Detailed Description

Evaluates delta [PDF](#) and stores values in a [Matrix3D](#) object.

7.7.2 Member Function Documentation

7.7.2.1 int DeltaPDF::pdfVal (const double * Z, const int ZPoints, [Matrix3D](#) * pdfValM) [virtual]

The Delta [PDF](#) uses statistics (means) to generate a [PDF](#). The [PDF](#) values are stored in a [Matrix3D](#) object: dim1 is the variance, dim2 is the mean, and dim3 are the data points.

INPUTS:

const double *Z	double array containing mixture fraction values coming from the files
const int ZPoints	number of mixture fraction values in the Z array
Matrix3D * pdfValm	the Matrix3D type container that stores the PDF values

OUTPUT:

int	flag specifying whether or not the function succeeded
= 0: success	
!= 0: something went wrong	

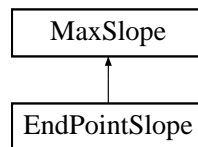
Implements [PDF](#).

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

- `src/deltaPDF.h`
- `src/deltaPDF.cc`

7.8 EndPointSlope Class Reference

Inheritance diagram for EndPointSlope::



Public Member Functions

- `EndPointSlope` (const `Matrix` &progVar)
Constructor.
- `~EndPointSlope` ()
Destructor.
- `int MostMonotonic` (int *monoAry, const int ncols, const int col)
Method to find the most monotonic progress variable.

7.8.1 Constructor & Destructor Documentation

7.8.1.1 EndPointSlope::EndPointSlope (const Matrix & progVar)

Constructor. `EndPointSlope` is a class that determines the most monotonic progress variable with respect to temperature (or another specified column). It calculates the slope from the first and last endpoints of each progress variable column and selects the progress variable with the largest magnitude slope.

The slope is given by $\frac{C_i(N) - C_i(1)}{N}$

7.8.2 Member Function Documentation

7.8.2.1 int EndPointSlope::MostMonotonic (int * monoAry, const int ncols, const int col) [virtual]

Method to find the most monotonic progress variable. `MostMonotonic` calculates the slope of the best linear approximation for each progress variable which is strictly increasing or strictly decreasing. The output array `monoAry` must be of length `ncols`, where each cell holds a value of 3 if `C` is strictly monotonic and has the largest slope, 2 if `C` is strictly monotonic but does not have the largest slope, and 0 for non-monotonic `C`. `col` is the reference column.

INPUTS:

<code>int *monoAry</code>	array containing integer flags that denote the monotonicity of candidate progress variables
<code>const int ncols</code>	number of columns of <code>monoAry</code>
<code>const int col</code>	the reference column

OUTPUT:

```
int                flag specifying whether or not the function succeeded
                  = 0: success
    != 0: something went wrong
```

Implements [MaxSlope](#).

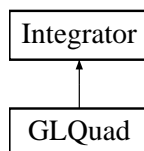
The documentation for this class was generated from the following files:

- src/endpointslope.h
- src/endpointslope.cc

7.9 GLQuad Class Reference

Calculates integral using Gauss-Legendre quadrature.

`#include <glquad.h>`Inheritance diagram for GLQuad::



Public Member Functions

- [GLQuad](#) (int Nodes)
Constructor.
- [~GLQuad](#) ()
Destructor.
- double [integrate](#) (const double *integrand, const double *Z, const int ZPoints)
Integration using Gauss-Legendre quadrature.

7.9.1 Detailed Description

Calculates integral using Gauss-Legendre quadrature. [GLQuad](#) takes in an array (the integrand) and returns the integral of that array using Gauss-Legendre quadrature. Abscissa are calculated using the external library AlgLib.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 GLQuad::GLQuad (int Nodes)

Constructor. Coordinates of abscissas are calculated in this function.

INPUTS:

const int Nodes number of abscissa

OUTPUT:

No particular output objects. Abscissas and weights are stored in `x_` and `w_` objects, respectively.

7.9.3 Member Function Documentation

7.9.3.1 double GLQuad::integrate (const double * *integrand*, const double * *Z*, const int *ZPoints*) [virtual]

Integration using Gauss-Legendre quadrature. Gauss-Legendre quadrature is applied to integrate a given integrand over a given double array, *Z*. Abscissas and weights are created during the execution of the con-

structor. The default number of nodes is 20, the number of nodes for the quadrature can be modified from the input file

INPUTS:

const double *integrand array that contains function values to be integrated

const double *Z array that contains the mixture fraction values which the integrand w

const int ZPoints number of values, size of the integrand and Z containers

OUTPUT:

double result of the integration

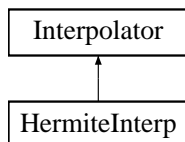
Implements [Integrator](#).

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

- src/glquad.h
- src/glquad.cc

7.10 HermiteInterp Class Reference

Inheritance diagram for HermiteInterp::



Public Member Functions

- [HermiteInterp \(\)](#)
Constructor.
- [~HermiteInterp \(\)](#)
Destructor.
- [int Interp \(const \[Matrix\]\(#\) *matin, int col, double ival, double *vecout, int cols\)](#)
Hermite spline interpolation function.

7.10.1 Member Function Documentation

7.10.1.1 `int HermiteInterp::Interp (const Matrix * matin, int col, double ival, double * vecout, int cols) [virtual]`

Hermite spline interpolation function. This function takes in a 2D matrix of data and interpolates an entire row from it using a hermite spline interpolator. Each column of the matrix is treated as a variable, with a specified column being the independent variable. The input data is assumed to be sorted

INPUTS:

<code>const Matrix *matin</code>	pointer to a Matrix object. This is the input data.
<code>int col</code>	integer specifying which column of the input Matrix is the independent variable
<code>double ival</code>	value at which to interpolate
<code>double *vecout</code>	pointer to an array which contains the interpolated row. This array has the same number of columns as the input Matrix.
<code>int cols</code>	number of columns of <i>matin</i> / <i>vecout</i>

OUTPUTS:

<code>int</code>	flag specifying whether or not the function succeeded
<code>= 0:</code>	success
<code>= 1:</code>	extrapolation attempted

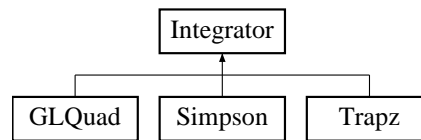
Implements [Interpolator](#).

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

- `src/hermiteinterp.h`
- `src/hermiteinterp.cc`

7.11 Integrator Class Reference

Inheritance diagram for Integrator::



Public Member Functions

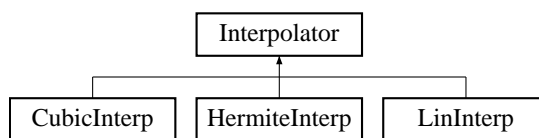
- virtual double [integrate](#) (const double *integrand, const double *Z, const int ZPoints)=0
Virtual function to be inherited by each integration algorithm to integrate the given data set.

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

- src/integrator.h

7.12 Interpolator Class Reference

Inheritance diagram for Interpolator::



Public Member Functions

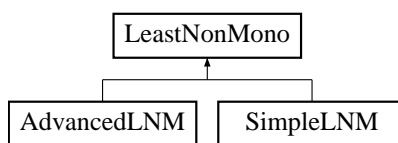
- virtual int [Interp](#) (const [Matrix](#) *matin, int col, double ival, double *vecout, int cols)=0
Virtual function to be inherited by each interpolation algorithm to interpolate the given data.

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

- src/interpolator.h

7.13 LeastNonMono Class Reference

Inheritance diagram for LeastNonMono::



Public Member Functions

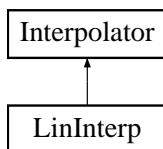
- virtual int [LeastNonMonotonic](#) (int *monoAry, const int ncols, const int col)=0
Virtual function to be inherited by each monotonicity cheking algorithm to determine the least non-monotonic progress variable.

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

- src/leastnonmono.h

7.14 LinInterp Class Reference

Inheritance diagram for LinInterp::



Public Member Functions

- [LinInterp \(\)](#)
Constructor.
- [~LinInterp \(\)](#)
Destructor.
- [int Interp \(const \[Matrix\]\(#\) *matin, int col, double ival, double *vecout, int cols\)](#)
Linear interpolation function.

7.14.1 Member Function Documentation

7.14.1.1 `int LinInterp::Interp (const Matrix * matin, int col, double ival, double * vecout, int cols)` [[virtual](#)]

Linear interpolation function. This function takes in a 2D matrix of data and interpolates an entire row from it using a linear interpolator. Each column of the matrix is treated as a variable, with a specified column being the independent variable. The input data is not assumed to be sorted.

INPUTS:

<code>const Matrix *matin</code>	pointer to a Matrix object. This is the input data.
<code>int col</code>	integer specifying which column of the input Matrix is the independent variable
<code>double ival</code>	value at which to interpolate
<code>double *vecout</code>	pointer to an array which contains the interpolated row. This array has the same number of columns as the input Matrix .
<code>int cols</code>	number of columns of <i>matin</i> / <i>vecout</i>

OUTPUTS:

`int` flag specifying whether or not the function succeeded
 = 0: success
 = 1: extrapolation attempted

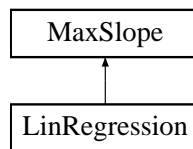
Implements [Interpolator](#).

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

- `src/lininterp.h`
- `src/lininterp.cc`

7.15 LinRegression Class Reference

Inheritance diagram for LinRegression::



Public Member Functions

- [LinRegression](#) (const [Matrix](#) &progVar)
Constructor.
- [~LinRegression](#) ()
Destructor.
- int [MostMonotonic](#) (int *monoAry, const int ncols, const int col)
Method to find the most monotonic progress variable.

7.15.1 Constructor & Destructor Documentation

7.15.1.1 LinRegression::LinRegression (const Matrix & progVar)

Constructor. [LinRegression](#) is a class that determines the most monotonic progress variable with respect to temperature (or another specified column). It calculates the slope of the best linear approximation for each progress variable and selects the largest magnitude.

The slope is given by $\{\sum_{i=1}^N (C_i - C_{ave})(T_i - T_{ave})\} / \{\sum_{i=1}^N (T_i - T_{ave})^2\}$

7.15.2 Member Function Documentation

7.15.2.1 int LinRegression::MostMonotonic (int * monoAry, const int ncols, const int col) [virtual]

Method to find the most monotonic progress variable. MostMonotonic calculates the slope of the best linear approximation for each progress variable which is strictly increasing or strictly decreasing. The output array monoAry must be of length ncols, where each cell holds a value of 3 if C is strictly monotonic and has the largest slope, 2 if C is strictly monotonic but does not have the largest slope, and 0 for non-monotonic C. col is the reference column.

Implements [MaxSlope](#).

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

- src/linregression.h
- src/linregression.cc

7.16 Matrix Class Reference

Public Member Functions

- [Matrix](#) (int rows, int cols)
Constructor.
- [~Matrix](#) ()
Destructor.
- double [GetVal](#) (int i, int j) const
Get the value at a specified index.
- void [SetVal](#) (int i, int j, double val)
Set the value at a specific location.
- int [GetNumRows](#) () const
Return the number of rows.
- int [GetNumCols](#) () const
Return the number of columns.
- int [GetCol](#) (int j, double *colAry) const
Return an array containing column j.

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

- src/matrix.h
- src/matrix.cc

7.17 Matrix3D Class Reference

Public Member Functions

- [Matrix3D](#) (int dim1, int dim2, int dim3)
Constructor.
- [~Matrix3D](#) ()
Destructor.
- double [GetVal](#) (int i, int j, int k) const
Get the value at a specified index.
- void [SetVal](#) (int i, int j, int k, double vol)
Set the value at a specified index.
- int [GetNumDim1](#) () const
Return dim1.
- int [GetNumDim2](#) () const
Return dim2.
- int [GetNumDim3](#) () const
Return dim3.

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

- src/matrix3d.h
- src/matrix3d.cc

7.18 Matrix4D Class Reference

Public Member Functions

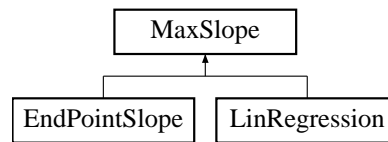
- [Matrix4D](#) (int dim1, int dim2, int dim3, int dim4)
Constructor.
- [~Matrix4D](#) ()
Destructor.
- double [GetVal](#) (int i, int j, int k, int l) const
Get the value at a specified index.
- void [SetVal](#) (int i, int j, int k, int l, double val)
Set the value at a specified index.
- int [GetNumDim1](#) () const
Return dim1.
- int [GetNumDim2](#) () const
Return dim2.
- int [GetNumDim3](#) () const
Return dim3.
- int [GetNumDim4](#) () const
Return dim4.

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

- src/matrix4d.h
- src/matrix4d.cc

7.19 MaxSlope Class Reference

Inheritance diagram for MaxSlope::



Public Member Functions

- virtual int [MostMonotonic](#) (int *monoAry, const int ncols, const int col)=0
Virtual function to be inherited by each monotonicity checking algorithm to determine the most monotonic progress variable.

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

- `src/maxslope.h`

7.20 MonoCheck Class Reference

Public Member Functions

- [MonoCheck](#) (const [Matrix](#) &progVar)
Constructor.
- [~MonoCheck](#) ()
Destructor.
- int [CheckStrictMonotonicity](#) (int *monoAry, const int ncols, int col)
Method to check whether each progress variable is strictly monotonic.

7.20.1 Constructor & Destructor Documentation

7.20.1.1 MonoCheck::MonoCheck (const Matrix & progVar)

Constructor. monochek is a class which checks whether a specified progress variable is strictly increasing or strictly decreasing with respect to temperature (or another specified column). That is, $C(T1) < C(T2)$ or $C(T1) > C(T2)$ for $T1 < T2$, where $T1$ and $T2$ are any two temperatures and C is the progress variable.

7.20.2 Member Function Documentation

7.20.2.1 int MonoCheck::CheckStrictMonotonicity (int * monoAry, const int ncols, int col)

Method to check whether each progress variable is strictly monotonic. CheckStrictMonotonicity checks the monotonicity of each column (AKA progress variable "C") in progVar with respect to column "col". The output array monoAry must be of length ncols_, where each cell holds a value of 3 if C is strictly increasing or strictly decreasing and 0 otherwise.

INPUTS:

```
int *monoAry      array containing integer flags that denote the monotonicity of candidate progress vari
const int ncols   number of columns of monoAry
const int col     the reference column
```

OUTPUT:

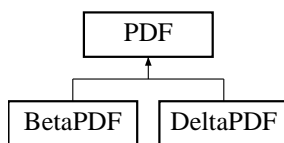
```
int              flag specifying whether or not the function succeeded
                  = 0: success
                  != 0: something went wrong
```

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

- src/monochek.h
- src/monochek.cc

7.21 PDF Class Reference

Inheritance diagram for PDF::



Public Member Functions

- virtual int **pdfVal** (const double *Z, const int ZPoints, [Matrix3D](#) *pdfValM)=0

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

- src/pdf.h

7.22 iofuncs::ProcFile Class Reference

Public Member Functions

- def [__init__](#)
The Constructor.
- def [gettitles](#)
Function that returns a vector containing the column headers in the 2nd row of the data file.
- def [interpolate](#)
Interpolate function extracts/interpolates a row of data in the data file at the specified value in the first column.

7.22.1 Detailed Description

Class for processing .kg datafiles, including extracting column headers and interpolating a row of data.

7.22.2 Member Function Documentation

7.22.2.1 def iofuncs::ProcFile::__init__ (self, sfile)

The Constructor. SFILE: a string containing the name of the data file to be processed or path to that file

7.22.2.2 def iofuncs::ProcFile::gettitles (self)

Function that returns a vector containing the column headers in the 2nd row of the data file. Ignores 1st row, returns contents of 2nd row as elements of a vector (assuming datafile is tab delimited).

7.22.2.3 def iofuncs::ProcFile::interpolate (self, inputvars, locs, datavec, interpval = 0.27, interpmethod = 'linear')

Interpolate function extracts/interpolates a row of data in the data file at the specified value in the first column. Read the columns of the data file with headers matching the strings in the INPUTVARS vector.

Write the column numbers corresponding to these column headers into LOCS vector.

Interpolate to find values of each column corresponding to INTERPVAL in the 1st column, using interpolation method specified by the INTERPMETHOD string.

Write interpolated values to DATAVEC.

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

- python/iofuncs.py

7.23 SequenceGen Class Reference

Sequence generator for the standard sorting algorithm.

Public Member Functions

- **SequenceGen** (int start=0)
- int **operator()** ()

7.23.1 Detailed Description

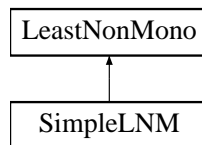
Sequence generator for the standard sorting algorithm.

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

- src/standardsort.cc

7.24 SimpleLNM Class Reference

Inheritance diagram for SimpleLNM::



Public Member Functions

- **SimpleLNM** (const **Matrix** &progVar)
Constructor.
- **~SimpleLNM** ()
Destructor.
- **LeastNonMonotonic** (int *monoAry, const int ncols, const int col)
Method to find the least non monotonic progress variable.

7.24.1 Constructor & Destructor Documentation

7.24.1.1 SimpleLNM::SimpleLNM (const Matrix & progVar)

Constructor. **SimpleLNM** is a class that determines the least non-monotonic progress variable with respect to temperature (or another specified column). It determines the percentage which the progress variable is strictly increasing and the percentage which the progress variable is strictly decreasing. The larger percentage not only determines whether the progress variable is increasing or decreasing, but will also be compared with the percentages of other progress variables. The progress variable with the largest percentage of increasing or decreasing values is the least non-monotonic progress variable.

If two or more progress variables share the highest percentage, then the progress variable with the greatest magnitude slope (by endpoints) is selected. Progress variables with neither increasing nor decreasing data are considered strongly non-monotonic.

7.24.2 Member Function Documentation

7.24.2.1 int SimpleLNM::LeastNonMonotonic (int * monoAry, const int ncols, const int col) [virtual]

Method to find the least non monotonic progress variable. LeastNonMonotonic calculates how much each progress variable is strictly increasing and strictly decreasing. The input array monoAry will initially be filled with 0s since all progress variables are non-monotonic. This method will select the least non-monotonic and change its value in monoAry to 1. col is the reference column.

INPUTS:

int *monoAry array containing integer flags that denote the monotonicity of candidate slope variables

```
const int ncols    number of columns of monoAry
```

```
const int col      the reference column
```

OUTPUT:

```
int                flag specifying whether or not the function succeeded
                   = 0: success
    != 0: something went wrong
```

Implements [LeastNonMono](#).

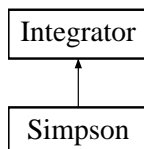
The documentation for this class was generated from the following files:

- src/simplelnm.h
- src/simplelnm.cc

7.25 Simpson Class Reference

Calculates integral using Simpson's rule.

`#include <simpson.h>`Inheritance diagram for Simpson::



Public Member Functions

- [Simpson \(\)](#)
Constructor.
- [~Simpson \(\)](#)
Destructor.
- `double integrate (const double *integrand, const double *Z, const int ZPoints)`
Main function that integrates a given data set using Simpson's Rule.

7.25.1 Detailed Description

Calculates integral using Simpson's rule. [Simpson](#) takes in an array (the integrand) and returns the integral of that array using Simpson's rule.

7.25.2 Member Function Documentation

7.25.2.1 `double Simpson::integrate (const double * integrand, const double * Z, const int ZPoints) [virtual]`

Main function that integrates a given data set using Simpson's Rule. Simpson's rule is applied to integrate a given integrand over a given double array, *Z*.

INPUTS:

`const double *integrand` array that contains function values to be integrated

`const double *Z` array that contains the mixture fraction values which the integrand will be integrated over

`const int ZPoints` number of values, size of the integrand and Z arrays

OUTPUT:

`double` result of the integration

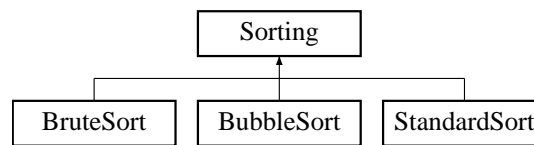
Implements [Integrator](#).

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

- `src/simpson.h`
- `src/simpson.cc`

7.26 Sorting Class Reference

Inheritance diagram for Sorting::



Public Member Functions

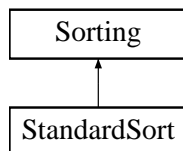
- virtual int [sort_data](#) ()=0
Virtual function to be inherited by each sorting algorithm to sort the give data.
- virtual void [SetRefColNum](#) (int num)
Setting the reference column according to which the data will be sorted.

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

- src/sorting.h

7.27 StandardSort Class Reference

Inheritance diagram for StandardSort::



Public Member Functions

- [StandardSort](#) ([Matrix](#) *data)
Constructor.
- [~StandardSort](#) ()
Destructor.
- [int sort_data](#) ()
Main function that sorts the given data.
- [void SetRefColNum](#) (int num)
Set the reference column number.

7.27.1 Constructor & Destructor Documentation

7.27.1.1 StandardSort::StandardSort ([Matrix](#) * data)

Constructor. The data to be sorted is passed to the constructor. A duplicate of the data is produced to be later at this stage

7.27.2 Member Function Documentation

7.27.2.1 [int StandardSort::sort_data](#) () [[virtual](#)]

Main function that sorts the given data. The algorithm sends the reference column to the standard sorting operator that is embedded into the C++ standard library

INPUT

There are no inputs. The data to be sorted is already passed via the constructor

OUTPUT:

```

int          flag specifying whether or not the function succeeded
              = 0: success
              != 0: something went wrong
  
```

Implements [Sorting](#).

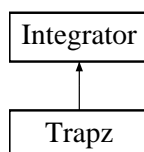
The documentation for this class was generated from the following files:

- `src/standardsort.h`
- `src/standardsort.cc`

7.28 Trapz Class Reference

Calculates integral using the trapezoidal method.

`#include <trapz.h>`Inheritance diagram for Trapz::



Public Member Functions

- [Trapz \(\)](#)
Constructor.
- [~Trapz \(\)](#)
Destructor.
- `double integrate (const double *integrand, const double *Z, const int ZPoints)`
Main function that integrates a given data set using Trapezoidal Rule.

7.28.1 Detailed Description

Calculates integral using the trapezoidal method. [Trapz](#) takes in an array (the integrand) and returns the integral of that array using the trapezoidal method.

7.28.2 Member Function Documentation

7.28.2.1 `double Trapz::integrate (const double * integrand, const double * Z, const int ZPoints) [virtual]`

Main function that integrates a given data set using Trapezoidal Rule. Trapezoidal rule is applied to integrate a given integrand over a given double array, Z.

INPUTS:

<code>const double *integrand</code>	array that contains function values to be integrated
<code>const double *Z</code>	array that contains the mixture fraction values which the integrand will be integrated over
<code>const int ZPoints</code>	number of values, size of the integrand and Z containers

OUTPUT:

<code>double</code>	result of the integration
---------------------	---------------------------

Implements [Integrator](#).

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

- src/trapz.h
- src/trapz.cc

Chapter 8

File Documentation

8.1 src/convolute.cc File Reference

```
#include "convolute.h"
#include "math.h"
```

Functions

- int [convVal](#) (double *Z, double *data, [Matrix3D](#) *pdfValues, [Matrix](#) *postConvVal, [Integrator](#) *integr)
Convolution function.

8.1.1 Detailed Description

8.1.2 Function Documentation

8.1.2.1 int convVal (double * Z, double * data, Matrix3D * pdfValues, Matrix * postConvVal, Integrator * integr)

Convolution function. Convolutates date and pdf.

8.2 src/fittogrid.cc File Reference

```
#include "fittogrid.h"
```

Functions

- `int fittogrid (const Matrix4D *datain, const double *cgrid, Interpolator *interp, Matrix3D *dataout, int nthreads, int ex)`
Function which fits data to a grid.

8.2.1 Detailed Description

8.2.2 Function Documentation

8.2.2.1 `int fittogrid (const Matrix4D *datain, const double *cgrid, Interpolator *interp, Matrix3D *dataout, int nthreads, int ex)`

Function which fits data to a grid. This function takes in a 4D matrix and fits the data onto a specified grid.

INPUTS:

<code>Matrix4D *datain</code>	input data stored as a 4D matrix. The structure is (mean, z_{\sim} , z_v , file), where mean has dimension 2 and contains w_{\sim} and c_{\sim} . datain can be thought of as two 3D matrices with structure (z_{\sim} , z_v , file) containing values of w_{\sim} and c_{\sim} , respectively.
<code>const double *cgrid</code>	pointer to an array which contains the values of c_{\sim} at which to interpolate
<code>Interpolator *interp</code>	pointer to an Interpolator object
<code>Matrix3D *dataout</code>	output data stored as a 3D matrix. The structure is (z_{\sim} , z_v , cgrid), with the numbers being interpolated values of w_{\sim}
<code>int numthreads</code>	integer specifying the number of threads to be used (1 = serial, >1 = parallel)
<code>int ex</code>	integer specifying whether or not to extrapolate (0 = no, 1 = yes)

OUTPUTS:

`int` flag specifying whether or not extrapolation was necessary:
 = 0: no extrapolation
 = 1: extrapolation performed

Index

- [__init__](#)
 - [iofuncs::ProcFile, 52](#)
- [AdvancedLNM, 21](#)
 - [AdvancedLNM, 21](#)
 - [LeastNonMonotonic, 22](#)
- [BetaPDF, 23](#)
 - [pdfVal, 23](#)
- [BruteSort, 25](#)
 - [BruteSort, 25](#)
 - [sort_data, 25](#)
- [BubbleSort, 27](#)
 - [BubbleSort, 27](#)
 - [sort_data, 27](#)
- [CheckStrictMonotonicity](#)
 - [MonoCheck, 50](#)
- [chemtable_io, 15](#)
- [combination_mat](#)
 - [combinations, 18](#)
- [combinations, 18](#)
 - [combination_mat, 18](#)
 - [combos, 18](#)
 - [numcoms, 18](#)
 - [totnumcoms, 18](#)
- [combos](#)
 - [combinations, 18](#)
- [CompVec, 29](#)
- [convolute.cc](#)
 - [convVal, 63](#)
- [convVal](#)
 - [convolute.cc, 63](#)
- [CubicInterp, 30](#)
 - [Interp, 30](#)
- [DeltaPDF, 32](#)
 - [pdfVal, 32](#)
- [EndPointSlope, 34](#)
 - [EndPointSlope, 34](#)
 - [MostMonotonic, 34](#)
- [findC](#)
 - [findprogvar, 19](#)
- [findprogvar, 19](#)
- [findC, 19](#)
- [fittogrid](#)
 - [fittogrid.cc, 64](#)
- [fittogrid.cc](#)
 - [fittogrid, 64](#)
- [get_progvar](#)
 - [iofuncs, 20](#)
- [gettittles](#)
 - [iofuncs::ProcFile, 52](#)
- [GLQuad, 36](#)
 - [GLQuad, 36](#)
 - [integrate, 36](#)
- [HermiteInterp, 38](#)
 - [Interp, 38](#)
- [integrate](#)
 - [GLQuad, 36](#)
 - [Simpson, 56](#)
 - [Trapz, 61](#)
- [Integrator, 40](#)
- [Interp](#)
 - [CubicInterp, 30](#)
 - [HermiteInterp, 38](#)
 - [LinInterp, 43](#)
- [interpolate](#)
 - [iofuncs::ProcFile, 52](#)
- [Interpolator, 41](#)
- [iofuncs, 20](#)
 - [get_progvar, 20](#)
 - [read_input, 20](#)
- [iofuncs::ProcFile, 52](#)
 - [__init__, 52](#)
 - [gettittles, 52](#)
 - [interpolate, 52](#)
- [LeastNonMono, 42](#)
- [LeastNonMonotonic](#)
 - [AdvancedLNM, 22](#)
 - [SimpleLNM, 54](#)
- [LinInterp, 43](#)
 - [Interp, 43](#)
- [LinRegression, 45](#)
 - [LinRegression, 45](#)
 - [MostMonotonic, 45](#)

Matrix, [46](#)
Matrix3D, [47](#)
Matrix4D, [48](#)
MaxSlope, [49](#)
MonoCheck, [50](#)
 CheckStrictMonotonicity, [50](#)
 MonoCheck, [50](#)
MostMonotonic
 EndPointSlope, [34](#)
 LinRegression, [45](#)

numcoms
 combinations, [18](#)

PDF, [51](#)
pdfVal
 BetaPDF, [23](#)
 DeltaPDF, [32](#)

read_input
 iofuncs, [20](#)

SequenceGen, [53](#)
SimpleLNM, [54](#)
 LeastNonMonotonic, [54](#)
 SimpleLNM, [54](#)
Simpson, [56](#)
 integrate, [56](#)
sort_data
 BruteSort, [25](#)
 BubbleSort, [27](#)
 StandardSort, [59](#)
Sorting, [58](#)
src/convolute.cc, [63](#)
src/fittogrid.cc, [64](#)
StandardSort, [59](#)
 sort_data, [59](#)
 StandardSort, [59](#)

totnumcoms
 combinations, [18](#)
Trapz, [61](#)
 integrate, [61](#)