

ANETO library

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>ANETO Library Documentation</b>	<b>1</b>
1.1	The ANETO Library . . . . .	1
1.2	Using the Library . . . . .	1
1.3	Quick Test . . . . .	2
1.4	Full Test . . . . .	2
1.5	OpenMP Support . . . . .	3
1.6	Documentation . . . . .	3
1.7	Licence . . . . .	3
<b>2</b>	<b>Namespace Index</b>	<b>5</b>
2.1	Namespace List . . . . .	5
<b>3</b>	<b>Class Index</b>	<b>7</b>
3.1	Class List . . . . .	7
<b>4</b>	<b>Namespace Documentation</b>	<b>9</b>
4.1	aneto Namespace Reference . . . . .	9
4.1.1	Detailed Description . . . . .	9
4.1.2	Enumeration Type Documentation . . . . .	10
4.1.2.1	COEFF_OPTION . . . . .	10
4.1.2.2	DERGRID_OPTION . . . . .	10
4.1.2.3	GRID_OPTION . . . . .	10
4.1.2.4	INTEG_OPTION . . . . .	10
4.1.2.5	JAC_OPTION . . . . .	11
4.1.2.6	TRANSF_OPTION . . . . .	11

<b>5</b>	<b>Class Documentation</b>	<b>13</b>
5.1	aneto::multidomain< T > Class Template Reference	13
5.1.1	Detailed Description	14
5.1.2	Constructor & Destructor Documentation	14
5.1.2.1	multidomain()	14
5.1.2.2	multidomain(int p_domain_number, int p_domain_points, T left_bndry=0, T right_bndry=+1, int number_threads=+1, DERGRID_OPTION dergrid_option=DERGRID_OPTION::SINGLE, GRID_OPTION grid_option=GRID_OPTION::UNIFORM, T *boundaries=NULL)	15
5.1.3	Member Function Documentation	15
5.1.3.1	comp_derivative(T *func, T *der, JAC_OPTION deriv_option=JAC_OPTION::PHYSICAL)	15
5.1.3.2	comp_derivative_domain(T *func, T *der, int dom, JAC_OPTION deriv_option=JAC_OPTION::PHYSICAL)	15
5.1.3.3	comp_integral(T *func, T *integ, INTEG_OPTION in_option=INTEG_OPTION::LEFT_BC, const T bndry_cond=0, const T &xp_bc=0)	16
5.1.3.4	dom_number() const	16
5.1.3.5	dom_points() const	16
5.1.3.6	dX_dxp(int i) const	16
5.1.3.7	dX_dxp(int dom, int k) const	17
5.1.3.8	dxp_dX(int i) const	17
5.1.3.9	dxp_dX(int dom, int k) const	17
5.1.3.10	get_dom(const int id) const	17
5.1.3.11	get_dom_coord(T x_i) const	17
5.1.3.12	get_id(const int dom, const int poin) const	18
5.1.3.13	get_l_bndry(int dom) const	18
5.1.3.14	get_l_glob_bndry()	18
5.1.3.15	get_point(const int id) const	18
5.1.3.16	get_pointer_dom(T *pointer_a0, int dom)	18
5.1.3.17	get_r_bndry(int dom)	19
5.1.3.18	get_r_glob_bndry()	19
5.1.3.19	get_spectral_coeffs(T *func, T *coeffs)	19
5.1.3.20	get_spectral_coeffs(T *func, T *coeffs, int dom)	19

5.1.3.21	<code>get_spectral_domain(int num_thread)</code>	19
5.1.3.22	<code>get_X(T x_i)</code>	20
5.1.3.23	<code>get_xp(int dom, T X)</code>	20
5.1.3.24	<code>initialise(int p_domain_number, int p_domain_points, T left_bndry=0, T right_bndry=+1, int number_threads=+1, DERGRID_OPTION dergrid_option=DERGRID_OPTION::SINGLE, GRID_OPTION grid_option=GRID_OPTION::UNIFORM, T *boundaries=NULL)</code>	20
5.1.3.25	<code>interpolate(T *function, T x_value, bool use_spec_stored=false)</code>	21
5.1.3.26	<code>number_points() const</code>	21
5.1.3.27	<code>root_finder_decreasing(T f_xr, T *func)</code>	21
5.1.3.28	<code>root_finder_general(T f_xr, T *func, int domain)</code>	21
5.1.3.29	<code>root_finder_increasing(T f_xr, T *func)</code>	22
5.1.3.30	<code>xp(int i) const</code>	22
5.1.3.31	<code>xp(int dom, int k) const</code>	22
5.2	<code>aneto::spectral_domain&lt; T &gt; Class Template Reference</code>	23
5.2.1	Detailed Description	23
5.2.2	Constructor & Destructor Documentation	24
5.2.2.1	<code>spectral_domain()</code>	24
5.2.2.2	<code>spectral_domain(int po, TRANSF_OPTION transf_op=TRANSF_OPTION::DEFAULT, FAULT)</code>	24
5.2.2.3	<code>~spectral_domain()</code>	24
5.2.3	Member Function Documentation	24
5.2.3.1	<code>cheb_transf(T *in, T *out)</code>	24
5.2.3.2	<code>clean()</code>	25
5.2.3.3	<code>compute_1st_der(T *func, T *der, JAC_OPTION der_option=JAC_OPTION::SPECTRAL, T *dxdX=NULL)</code>	25
5.2.3.4	<code>compute_integral(T *func, T *Integ, INTEG_OPTION i_option=INTEG_OPTION::LEFT_BC, T bndry_cond=0, JAC_OPTION jac_option=JAC_OPTION::SPECTRAL, T *dxdX=NULL)</code>	25
5.2.3.5	<code>compute_spectral_func_der(T *func, JAC_OPTION der_option=JAC_OPTION::SPECTRAL, T *dxdX=NULL, COEFF_OPTION coeff_op=COEFF_OPTION::D1)</code>	26
5.2.3.6	<code>get_d2(T X)</code>	26
5.2.3.7	<code>get_der(T X)</code>	26
5.2.3.8	<code>get_fun(T X)</code>	26
5.2.3.9	<code>get_points()</code>	27
5.2.3.10	<code>get_spectral_coeffs(T *func, T *coeffs)</code>	27
5.2.3.11	<code>get_X(int k)</code>	27
5.2.3.12	<code>initialise(int po, TRANSF_OPTION transf_op=TRANSF_OPTION::DEFAULT)</code>	27
5.2.3.13	<code>interpolate(T X, T *func)</code>	28
5.2.3.14	<code>interpolate_with_spec_coeffs(T X, T *coeffs)</code>	28
5.2.3.15	<code>pi()</code>	28
5.2.3.16	<code>root_finder(T f_Xr, T *func)</code>	28



# Chapter 1

## ANETO Library Documentation

### 1.1 The ANETO Library

Project homepage: <https://github.com/DSantosO/anetolib>

Copyright (c) 2017 Daniel Santos-Olivan and Carlos F. Sopena

**ANETO** is under GNU General Public License ("GPL").

The **ANETO** (Arbitrary precision solvEr with pseudo-specTral methOds) library's main purpose is to provide a tool to perform simple one-dimensional problems or evolution ones with the Method of lines with Arbitrary Precision using Pseudo-Spectral Collocation methods (PSC).

The exponential convergence of spectral methods makes them the best option to go beyond the standard double precision (64-bits). For this precision, maximum accuracy is usually reached with a very low number of discretization points so it is not that computational expensive to go further.

The classes implemented here can be used with arbitrary data types allowing us to control the accuracy of our numerical computations until the level we need / can computationally afford.

A paper with a basic review of PSC methods and showing some of the possibilities of the library will be published soon.

For comments, questions or suggestions about new functionalities, the user can contact [anetolib@gmail.com](mailto:anetolib@gmail.com)

The authors thank Lluís Guesà and Víctor Martín at ICE (IEEC-CSIC) for suggestions and improvements in the code of the library.

### 1.2 Using the Library

All the functionalities of this library are implemented as template classes in the header files included in the folder **aneto/** so in order to be used in your program is enough to tell the compiler to include this folder or to copy them into an existing include folder.

The ANETO library needs the standard **C++** library and the **Eigen3** library for Linear Algebra that can be downloaded from <http://eigen.tuxfamily.org/>

## 1.3 Quick Test

For a quick test of the library, the script **tests.sh** can be used. Just:

```
./tests-sh build
./tests-sh run
```

If any of the previous fails, check the next section for a full option building.

## 1.4 Full Test

The current library includes some tests that can be execute in order to check the correct behaviour of the library or to be used as an use examples.

The first thing is to compile the test executable. Located in the **ANETO** folder, this can be done as:

```
cd build/
cmake ../.
make tests
```

The examples use standard types and **MPFR C++** a wrap-up created by Pavel Holoborodko (<http://www.holoborodko.com/pavel/mpfr/>). MPFR C++ is included with the examples but **GNU's MPFR** library (<http://www.mpfr.org>) is needed. This should be available in the repositories of most GNU/Linux distributions and can be also downloaded from its website. If it is not available an error will show it during the cmake building process.

**Boost C++** libraries (<http://www.boost.org>) are also used and need to be installed. If any of the boost modules needed are not available an error will show it during the cmake building process.

The test also check some of the functions with the non-standard quadruple precision 'quadmath'. The use of this library can be disable by adding '-DQUAD=OFF' as an cmake argument.

In addition, some functions are also checked with the QD library types `dd_real` (32 digits) and `qd_real` (64 digits). The headers and the static library needs to be installed in a folder available for the compiler. The library can be downloaded at <http://crd-legacy.lbl.gov/~dhbailey/mpdist/>. The use of this library can be disable by adding '-DQD=OFF' as an cmake argument.

Once the tests are compiled they can check most of the functions of the library and can be executed just by:

```
./tests
```

For a complete output of the tests the verbose option '-v' can be enabled.

This option will generate files for some of the tests.

If a single test is wanted it also can be executed with the option 't'.

The following are available:

```
./tests -t singledom_derivatives
./tests -t singledom_integral_comparison
./tests -t singledom_integral_LR
./tests -t multidom_derivative_dual
./tests -t multidom_integral_comparison
./tests -t multidom_integral_LR
./tests -t singledom_spectral_fft
./tests -t root_finding
./tests -t interpolation
```



Also, only singledomain or multidomain test can be executed by:

```
./tests -t singledom
./tests -t multidom
```

Aside from the predefined one, the user can also change the number of domains/points for checking different grid configuration. '-D' change the domains '-N' change the points per domain and '-S' change the points for single domain tests. In some tests, the user can also change the digits precision with '-p'. All of this options only will be visible in the verbose mode and will not impact in the default checks.

## 1.5 OpenMP Support

The parallel support via OpenMP (<http://www.openmp.org/>) is yet experimental. Now it is available in the multidomain for the integral and normal derivative but it is not assured a perfect behaviour. If it present any problem or one just one to execute the library sequentially it can be compiled without OpenMP adding -DOMP=OFF in the test cmake or adding directly -DOMP=0 as compilation option.

## 1.6 Documentation

The manual in pdf can be found in the **doc/** folder.

In addition, the full documentation of **ANETO** library can be generated by the script

```
./documentation
```

For a proper compilation, it needs Doxygen generator ([www.doxygen.org](http://www.doxygen.org)). By default, a HTML version and a LaTeX pdf will be generated, so **pdflatex** and the proper packages needs to be installed.

Doxygen options can be changed in the configuration file 'Doxfile'.

## 1.7 Licence

ANETO C++ Library - Arbitrary precision solvEr with pseudo-spectRal MethOds Project homepage: <https://github.com/DSantosO/anetolib> Copyright (c) 2017 Daniel Santos-Olivan and Carlos F. Sopena

ANETO is under GNU General Public License ("GPL").

GNU General Public License ("GPL") copyright permissions statement: This file is part of ANETO

ANETO is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

ANETO is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.



## Chapter 2

# Namespace Index

### 2.1 Namespace List

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

[aneto](#)

The **aneto** module contain all the tools to generate a grid and perform the needed operations . 9



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">aneto::multidomain&lt; T &gt;</a>	
Template of the multidomain structure defined a set of <a href="#">spectral_domain</a>	13
<a href="#">aneto::spectral_domain&lt; T &gt;</a>	
Template of a single spectral domain defined in $X = [-1,+1]$	23



## Chapter 4

# Namespace Documentation

### 4.1 aneto Namespace Reference

The **aneto** module contain all the tools to generate a grid and perform the needed operations.

#### Classes

- class [multidomain](#)  
*Template of the multidomain structure defined a set of [spectral\\_domain](#).*
- class [spectral\\_domain](#)  
*Template of a single spectral domain defined in  $X = [-1, +1]$ .*

#### Enumerations

- enum [GRID\\_OPTION](#) { [GRID\\_OPTION::UNIFORM](#), [GRID\\_OPTION::DEF\\_BNDS](#) }
- enum [DERGRID\\_OPTION](#) { [DERGRID\\_OPTION::SINGLE](#), [DERGRID\\_OPTION::DUAL](#) }
- enum [JAC\\_OPTION](#) { [JAC\\_OPTION::SPECTRAL](#), [JAC\\_OPTION::PHYSICAL](#) }
- enum [COEFF\\_OPTION](#) { [COEFF\\_OPTION::F](#), [COEFF\\_OPTION::D1](#), [COEFF\\_OPTION::D2](#) }
- enum [INTEG\\_OPTION](#) { [INTEG\\_OPTION::LEFT\\_BC](#), [INTEG\\_OPTION::RIGHT\\_BC](#), [INTEG\\_OPTION::CUSTOM\\_BC](#) }
- enum [TRANSF\\_OPTION](#) { [TRANSF\\_OPTION::MATRIX](#), [TRANSF\\_OPTION::FFT](#), [TRANSF\\_OPTION::DEFAULT](#) }

#### 4.1.1 Detailed Description

The **aneto** module contain all the tools to generate a grid and perform the needed operations.

The grid can be generated using a single grid with the class [spectral\\_domain](#) or a set of domains using the class [multidomain](#).

## 4.1.2 Enumeration Type Documentation

### 4.1.2.1 enum `aneto::COEFF_OPTION` [`strong`]

Option that controls how many levels of spectral coefficients will be stored.

Enumerator

- F*** We stored the spectral coefficients of the function.
- D1*** We stored the spectral coefficients of the function and the first derivative.
- D2*** We stored the spectral coefficients of the function and the first and second derivatives.

### 4.1.2.2 enum `aneto::DERGRID_OPTION` [`strong`]

Derivatives can be computed using one set of pseudospectral domains or two of them. In the latest, the domains are centred in the nodes of the original domain. The final derivatives are computed using a mixed of both grids that reduces notably the error near the boundaries of the domains.

Enumerator

- SINGLE*** Derivatives are computed in one domain.
- DUAL*** Derivatives are computed in two overlapping domains. This reduces the error near the domain boundaries

### 4.1.2.3 enum `aneto::GRID_OPTION` [`strong`]

Mode that select the the position of the internal domains of the grid.

Enumerator

- UNIFORM*** All domains have the same size.
- DEF\_BOUNDS*** User defined boundaries for the domains.

### 4.1.2.4 enum `aneto::INTEG_OPTION` [`strong`]

Possible options for imposing a boundary condition (BC) in the computation of the multidomain integral. Exact details can be seen in `comp_integral`. Be aware that in the [spectral\\_domain](#) function, the `CUSTOM_BC` is not accepted.

Enumerator

- LEFT\_BC*** Impose the BC in the left global boundary.
- RIGHT\_BC*** Impose the BC in the right global boundary.
- CUSTOM\_BC*** Impose the BC in an arbitrary point of the multidomain.



#### 4.1.2.5 enum aneto::JAC\_OPTION [strong]

Jacobian Option for the derivative / integral. It allows the operations to be performed in the coordinate of the spectral domain or in the one of the multidomain.

Enumerator

**SPECTRAL** The derivative is computed respect to the spectral coordinate ( $X = [-1, +1]$ ).

**PHYSICAL** The derivative is computed respect to the physical coordinate (multidomain).

#### 4.1.2.6 enum aneto::TRANSF\_OPTION [strong]

Possible options for making the transformation between the collocation points and the spectral representation. It can be done by matrix multiplication ( $\sim N^2$ ) or by FFT ( $\sim N \log(N)$ ) optimized for number of points  $2^i$  with  $i$  integer. The default option will select FFT if  $N = 2^i$  and matrix multiplication otherwise.

Enumerator

**MATRIX** Transformation from matrix multiplication.

**FFT** Transformation through FFT.

**DEFAULT** Select FFT option if  $N$  is a power of 2 ( $N = 2^i$ ) and MATRIX otherwise.



## Chapter 5

# Class Documentation

### 5.1 aneto::multidomain< T > Class Template Reference

Template of the multidomain structure defined a set of [spectral\\_domain](#).

```
#include <multidomain.hpp>
```

#### Public Member Functions

- [multidomain](#) ()
- [multidomain](#) (int p\_domain\_number, int p\_domain\_points, T left\_bndry=0, T right\_bndry=+1, int number\_threads=+1, [DERGRID\\_OPTION](#) dergrid\_option=[DERGRID\\_OPTION::SINGLE](#), [GRID\\_OPTION](#) grid\_option=[GRID\\_OPTION::UNIFORM](#), T \*boundaries=NULL)
- void [initialise](#) (int p\_domain\_number, int p\_domain\_points, T left\_bndry=0, T right\_bndry=+1, int number\_threads=+1, [DERGRID\\_OPTION](#) dergrid\_option=[DERGRID\\_OPTION::SINGLE](#), [GRID\\_OPTION](#) grid\_option=[GRID\\_OPTION::UNIFORM](#), T \*boundaries=NULL)
- int [number\\_points](#) () const
- int [dom\\_points](#) () const
- int [dom\\_number](#) () const
- T [xp](#) (int i) const
- T [xp](#) (int dom, int k) const
- T [dxp\\_dX](#) (int i) const
- T [dxp\\_dX](#) (int dom, int k) const
- T [dX\\_dxp](#) (int i) const
- T [dX\\_dxp](#) (int dom, int k) const
- T [get\\_l\\_bndry](#) (int dom) const
- T [get\\_r\\_bndry](#) (int dom)
- T [get\\_l\\_glob\\_bndry](#) ()
- T [get\\_r\\_glob\\_bndry](#) ()
- int [get\\_id](#) (const int dom, const int poin) const
- int [get\\_dom](#) (const int id) const
- int [get\\_point](#) (const int id) const
- T \* [get\\_pointer\\_dom](#) (T \*pointer\_a0, int dom)
- int [get\\_dom\\_coord](#) (T x\_i) const
- T [get\\_X](#) (T x\_i)
- T [get\\_xp](#) (int dom, T X)
- T [interpolate](#) (T \*function, T x\_value, bool use\_spec\_stored=false)

- [T root\\_finder\\_increasing](#) (T f\_xr, T \*func)
- [T root\\_finder\\_decreasing](#) (T f\_xr, T \*func)
- [T root\\_finder\\_general](#) (T f\_xr, T \*func, int domain)
- void [comp\\_integral](#) (T \*func, T \*integ, [INTEG\\_OPTION](#) in\_option=[INTEG\\_OPTION::LEFT\\_BC](#), const T bndry\_cond=0, const T &xp\_bc=0)
- void [comp\\_derivative](#) (T \*func, T \*der, [JAC\\_OPTION](#) deriv\_option=[JAC\\_OPTION::PHYSICAL](#))
- void [comp\\_derivative\\_domain](#) (T \*func, T \*der, int dom, [JAC\\_OPTION](#) deriv\_option=[JAC\\_OPTION::PHYSICAL](#))
- void [get\\_spectral\\_coeffs](#) (T \*func, T \*coeffs)
- void [get\\_spectral\\_coeffs](#) (T \*func, T \*coeffs, int dom)
- [spectral\\_domain](#)< T > & [get\\_spectral\\_domain](#) (int num\_thread)

### 5.1.1 Detailed Description

```
template<typename T>
class aneto::multidomain< T >
```

Template of the multidomain structure defined a set of [spectral\\_domain](#).

Class that create a multidomain structure with all the individual domains being defined with the [spectral\\_domain](#) class.

The multidomain can be created directly with all the options or just created empty and be initialised afterwards. Once it is initialised the structure can not be modify but can be cleaned up and initialised again.

The size of the grid needs to be specify at the moment of initialisation.

The position of the internal domains can be specified as an option that can take the value [GRID\\_OPTION::UNIFORM](#) for constant size domains and also [GRID\\_OPTION::DEF\\_BNDS](#) if we want to specify the position of the boundaries.

In the initialisation also can be choosen the mode of performing derivatives. [DERGRID\\_OPTION::SINGLE](#) will compute normal derivatives and [DERGRID\\_OPTION::DUAL](#) will create a dual grid structure that can increase the accuracy of the derivatives near the domain boundaries in two orders of magnitude.

The number\_threads option for select to do the operations in parallel is not yet supported and only admit the default value of one.

The template can be used with any general type where the usual operations + - \* / are defined.

Also a cos / acos functions are required for the specific type.

#### Template Parameters

<i>T</i>	type to be used for the spectral grid
----------	---------------------------------------

### 5.1.2 Constructor & Destructor Documentation

5.1.2.1 `template<typename T> aneto::multidomain< T>::multidomain ( ) [inline]`

Void constructor.

A multidomain is created empty. It needs to be initialised in order to be used.

```
5.1.2.2 template<typename T > aneto::multidomain< T >::multidomain ( int p_domain_number, int p_domain_points,
T left_bndry = 0, T right_bndry = +1, int number_threads = +1, DERGRID_OPTION dergrid_option =
DERGRID_OPTION::SINGLE, GRID_OPTION grid_option = GRID_OPTION::UNIFORM, T * boundaries =
NULL ) [inline]
```

Constructor.

It creates a multidomain with D domains and N+1 grid points each.

#### Parameters

<i>p_domain_number</i>	Number of domains
<i>p_domain_points</i>	Number of points in each domain (p_domain_points + 1)
<i>left_bndry</i>	left global boundary. Default 0.
<i>right_bndry</i>	right global boundary. Default +1.
<i>number_threads</i>	Number of OpenMP threads. Warning! now, only supports one.
<i>dergrid_option</i>	Option for derivatives. <a href="#">DERGRID_OPTION::SINGLE</a> for normal multidomain. <a href="#">DERGRID_OPTION::DUAL</a> for a dual grid.
<i>grid_option</i>	Option for the grid structure. <a href="#">GRID_OPTION::UNIFORM</a> : Equally distributed domains <a href="#">GRID_OPTION::DEF_BNDS</a> : Internal boundaries can be defined by the user.
<i>boundaries</i>	Array with the boundaries of the domains. It suppose to have D-1 components starting with the boundary between the first and second domains, not including the global boundaries. If the boundaries are not ordered correctly the run will be aborted. Can be left empty or passing a NULL points if it is not needed.

### 5.1.3 Member Function Documentation

```
5.1.3.1 template<typename T > void aneto::multidomain< T >::comp_derivative ( T * func, T * der, JAC_OPTION
deriv_option = JAC_OPTION::PHYSICAL ) [inline]
```

This function compute the derivative of the whole multidomain respect to the spectral coordinates or to the multidomain ones.

#### Parameters

<i>func</i>	Array with the value of the function in collocation points
<i>der</i>	Array where it will be stored the value of the derivative in the collocation points.
<i>deriv_option</i>	Compute the derivative respect to multidomain coordinates or to spectral ones. <a href="#">JAC_OPTION::PHYSICAL</a> respect to multidomain. Default. <a href="#">JAC_OPTION::SPECTRAL</a> respect to the spectral coordinate of the domains.

```
5.1.3.2 template<typename T > void aneto::multidomain< T >::comp_derivative_domain ( T * func, T * der, int dom,
JAC_OPTION deriv_option = JAC_OPTION::PHYSICAL ) [inline]
```

This function compute the derivative of a single domain respect to the spectral coordinates or to the multidomain ones.

## Parameters

<i>func</i>	Array with the value of the function in collocation points
<i>der</i>	Array where it will be stored the value of the derivative in the collocation points.
<i>dom</i>	Domain where to compute the derivative.
<i>deriv_option</i>	Compute the derivative respect to multidomain coordinates or to spectral ones. <a href="#">JAC_OPTION::PHYSICAL</a> respect to multidomain. Default. <a href="#">JAC_OPTION::SPECTRAL</a> respect to the spectral coordinate of the domains.

5.1.3.3 `template<typename T> void aneto::multidomain< T >::comp_integral ( T * func, T * integ, INTEG_OPTION in_option = INTEG_OPTION::LEFT_BC, const T bdry_cond = 0, const T & xp_bc = 0 )` [inline]

This function compute the following integrals:

$$I_L(x) = BC + \int_{x_L}^x f(x) dx,$$

$$I_R(x) = BC + \int_x^{x_R} f(x) dx$$

$$I_C(x) = BC + \int_{x_0}^x f(x) dx$$

depending if *integ\_option* is fixed to LEFT\_BC, RIGHT\_BC or CUSTOM\_BC. The term BC represents the boundary condition we impose in one of the boundaries.

For getting the integral in the physical coordinates we need to include the jacobian between the physical and pseudospectral coordinates.

## Parameters

<i>func</i>	Array with the value of the function in collocation points
<i>integ</i>	Array where it will be stored the value of the integral in the collocation points.
<i>in_option</i>	Integral option: from the left, from the right or from a custom point.
<i>bdry_cond</i>	Value we impose in the boundary. Default value to zero.
<i>xp_bc</i>	Value of x0 for CUSTOM_BC. Default value to zero.

5.1.3.4 `template<typename T> int aneto::multidomain< T >::dom_number ( ) const` [inline]

Returns the number of domains.

5.1.3.5 `template<typename T> int aneto::multidomain< T >::dom_points ( ) const` [inline]

Returns the number of points per domain.

5.1.3.6 `template<typename T> T aneto::multidomain< T >::dX_dxp ( int i ) const` [inline]

Returns the value of the multidomain jacobian (inverse) in the point k fo the domain dom.

## Parameters

<i>i</i>	the global index
----------	------------------

5.1.3.7 `template<typename T> T aneto::multidomain< T>::dX_dxp ( int dom, int k ) const` `[inline]`

Returns the value of the multidomain jacobian (inverse) in the point k fo the domain dom.

## Parameters

<i>dom</i>	the domain
<i>k</i>	the index in the domain

5.1.3.8 `template<typename T> T aneto::multidomain< T>::dxp_dX ( int i ) const` `[inline]`

Returns the value of the multidomain jacobian in the point k fo the domain dom.

## Parameters

<i>i</i>	the global index
----------	------------------

5.1.3.9 `template<typename T> T aneto::multidomain< T>::dxp_dX ( int dom, int k ) const` `[inline]`

Returns the value of the multidomain jacobian in the point k fo the domain dom.

## Parameters

<i>dom</i>	the domain
<i>k</i>	the index in the domain

5.1.3.10 `template<typename T> int aneto::multidomain< T>::get_dom ( const int id ) const` `[inline]`

Returns the domain that correspond with a certain global index.  
The input is not checked.

## Parameters

<i>id</i>	index of the array.
-----------	---------------------

5.1.3.11 `template<typename T> int aneto::multidomain< T>::get_dom_coord ( T x_i ) const` `[inline]`

Returns the domain to which correspond a given coordinate.  
The method will fail if the given coordinate is outside the grid.

## Parameters

$x \leftrightarrow$	coordinate of the grid.
$\_ \leftrightarrow$	
$i$	

**5.1.3.12** `template<typename T> int aneto::multidomain< T >::get_id ( const int dom, const int poin ) const`  
`[inline]`

Returns the index array of the point *i* of the domain *a*.  
The input is not checked.

## Parameters

<i>dom</i>	the domain
<i>poin</i>	the point

**5.1.3.13** `template<typename T> T aneto::multidomain< T >::get_l_bndry ( int dom ) const` `[inline]`

Returns the left boundary of a domain.

## Parameters

<i>dom</i>	the domain
------------	------------

**5.1.3.14** `template<typename T> T aneto::multidomain< T >::get_l_glob_bndry ( )` `[inline]`

Returns the left global boundary.

**5.1.3.15** `template<typename T> int aneto::multidomain< T >::get_point ( const int id ) const` `[inline]`

Returns number of point in the spectral grid that correspond with a certain global index.  
The input is not checked.

## Parameters

<i>id</i>	index of the array.
-----------	---------------------

**5.1.3.16** `template<typename T> T* aneto::multidomain< T >::get_pointer_dom ( T* pointer_a0, int dom )`  
`[inline]`

Returns the index array of the point *i* of the domain *a*.  
The input is not checked.



## Parameters

<i>dom</i>	the domain
<i>poin</i>	the point

5.1.3.17 `template<typename T> T aneto::multidomain< T >::get_r_bndry ( int dom ) [inline]`

Returns the right boundary of a domain.

## Parameters

<i>dom</i>	the domain
------------	------------

5.1.3.18 `template<typename T> T aneto::multidomain< T >::get_r_glob_bndry ( ) [inline]`

Returns the right global boundary.

5.1.3.19 `template<typename T> void aneto::multidomain< T >::get_spectral_coeffs ( T * func, T * coeffs ) [inline]`

This function computes the spectral coefficients of a function and stores it in a given array

## Parameters

<i>func</i>	array with the value of the function in collocation points.
<i>coeffs</i>	array for storing the spectral coefficients.

5.1.3.20 `template<typename T> void aneto::multidomain< T >::get_spectral_coeffs ( T * func, T * coeffs, int dom ) [inline]`

This function computes the spectral coefficients of a function in a given domain and stores it in a an array

## Parameters

<i>func</i>	array with the value of the function in collocation points.
<i>coeffs</i>	array for storing the spectral coefficients. It is assumed to be of the size of the spectral_doman N+1. If you want to to stored in a multidomain array, use get_pointer_dom.
<i>dom</i>	Domain where the spectral coefficients will be computed.

5.1.3.21 `template<typename T> spectral_domain<T>& aneto::multidomain< T >::get_spectral_domain ( int num_thread ) [inline]`

This function returns the pointer to one of the spectral domains used in the multidomain to perform operations with it. The multidomain class uses a spectral\_domain<T> object for every thread

## Parameters

<i>num_thread</i>	index of the <a href="#">spectral_domain</a> required. Need to be less than the number of threads used by the object. If it is greater it will return
-------------------	---

5.1.3.22 `template<typename T> T aneto::multidomain< T >::get_X( T x_i ) [inline]`

Return the spectral coordinate in the the domain that the given coordinate is in.  
The method will fail if the given coordinate is outside the grid.

## Parameters

$x \leftrightarrow$	coordinate of the grid.
$\_ \leftrightarrow$	
$i$	

5.1.3.23 `template<typename T> T aneto::multidomain< T >::get_xp( int dom, T X ) [inline]`

Grid coordinate that corresponds with the spectral coordinate X of the domain dom.

## Parameters

<i>dom</i>	Domain we are interested in.
<i>X</i>	Spectral coordinate.

5.1.3.24 `template<typename T> void aneto::multidomain< T >::initialise ( int p_domain_number, int p_domain_points, T left_bndry = 0, T right_bndry = +1, int number_threads = +1, DERGRID_OPTION dergrid_option = DERGRID_OPTION::SINGLE, GRID_OPTION grid_option = GRID_OPTION::UNIFORM, T * boundaries = NULL ) [inline]`

It initialises the multidomain with D domains and N grid points each. The exact range of the coordinate can be chosen.

## Parameters

<i>p_domain_number</i>	Number of domains
<i>p_domain_points</i>	Number of points in each domain (p_domain_points + 1)
<i>left_bndry</i>	left global boundary. Default 0.
<i>right_bndry</i>	right global boundary. Default +1.
<i>number_threads</i>	Number of OpenMP threads. Warning! now, only supports one.
<i>dergrid_option</i>	Option for derivatives. <a href="#">DERGRID_OPTION::SINGLE</a> for normal multidomain. <a href="#">DERGRID_OPTION::DUAL</a> for a dual grid.
<i>grid_option</i>	Option for the grid structure. <a href="#">GRID_OPTION::UNIFORM</a> : Equally distributed domains <a href="#">GRID_OPTION::DEF_BNDS</a> : Internal boundaries can be defined by the user.
<i>boundaries</i>	Array with the boundaries of the domains. It suppose to have D-1 components starting with the boundary between the first and second domains, not including the global boundaries. If the boundaries are not ordered correctly the run will be aborted. Can be left empty or passing a NULL points if it is not needed.

**5.1.3.25** `template<typename T> T aneto::multidomain< T >::interpolate ( T * function, T x_value, bool use_spec_stored = false ) [inline]`

Compute the value of a function in an arbitrary function through spectral interpolation. Careful with the `use_spec_stored` option because the class doesn't control if the stored values correspond with the same function or domain of the current call.

#### Parameters

<i>function</i>	Function we want to interpolate.
<i>x_value</i>	Coordinate where we want the function value.
<i>use_spec_stored</i>	If activated, the interpolation will use the stored values from a previous interpolation.

**5.1.3.26** `template<typename T> int aneto::multidomain< T >::number_points ( ) const [inline]`

Returns the total number of points.

`total = domain_number * (domain_points + 1);`

**5.1.3.27** `template<typename T> T aneto::multidomain< T >::root_finder_decreasing ( T f_xr, T * func ) [inline]`

It finds the coordinate where the function  $f(x)$  has a given value.

$$f(f_x r) - f_x r = 0;$$

It requires that the function  $f(x)$  is monotonically decreasing. In this case the root, if it exists, is unique. If there is no root in the whole multidomain, the function will return an error.

#### Parameters

<i>f_xr</i>	Value of the function we
<i>func</i>	Value of the function in the collocation points.

#### Returns

**5.1.3.28** `template<typename T> T aneto::multidomain< T >::root_finder_general ( T f_xr, T * func, int domain ) [inline]`

It finds the coordinate where the function  $f(x)$  has a given value in a given domain.

$$f(f_x r) - f_x r = 0;$$

If the function has no root in this domain, the function will return the closest value. If the function has several roots, the function will return the first it finds.

## Parameters

<i>f_xr</i>	Value of the function we
<i>func</i>	Value of the function in the collocation points.
<i>domain</i>	Domain where the function look for the root.

## Returns

5.1.3.29 `template<typename T> T aneto::multidomain<T>::root_finder_increasing( T f_xr, T* func ) [inline]`

It find the coordinate where the function  $f(x)$  have a given value in a given domain.

$$f(f_xr) - f_xr = 0;$$

It requires that the function  $f(x)$  is monotonically decreasing. In this case the root, if exists is unique. If there is no root in the whole multidomain, the function will rise an error.

## Parameters

<i>f_xr</i>	Value of the function we
<i>func</i>	Value of the function in the collocation points.

## Returns

5.1.3.30 `template<typename T> T aneto::multidomain<T>::xp( int i ) const [inline]`

Returns the value of the multidomain coordinate in the collocation point  $i$ .

## Parameters

<i>i</i>	the global index
----------	------------------

5.1.3.31 `template<typename T> T aneto::multidomain<T>::xp( int dom, int k ) const [inline]`

Returns the value of the multidomain coordinate in the collocation point  $i$ .

## Parameters

<i>dom</i>	the domain
<i>k</i>	the index in the domain

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

- /home/santos/grav\_local/anetolib/aneto/multidomain.hpp

## 5.2 aneto::spectral\_domain< T > Class Template Reference

Template of a single spectral domain defined in  $X = [-1, +1]$ .

```
#include <spectral_domain.hpp>
```

### Public Member Functions

- [spectral\\_domain](#) ()
- [spectral\\_domain](#) (int po, [TRANSF\\_OPTION](#) transf\_op=[TRANSF\\_OPTION::DEFAULT](#))
- [~spectral\\_domain](#) ()
- void [initialise](#) (int po, [TRANSF\\_OPTION](#) transf\_op=[TRANSF\\_OPTION::DEFAULT](#))
- void [clean](#) ()
- int [get\\_points](#) ()
- T [get\\_X](#) (int k)
- void [cheb\\_transf](#) (T \*in, T \*out)
- void [compute\\_integral](#) (T \*func, T \*Integ, [INTEG\\_OPTION](#) i\_option=[INTEG\\_OPTION::LEFT\\_BC](#), T bndry↵\_cond=0, [JAC\\_OPTION](#) jac\_option=[JAC\\_OPTION::SPECTRAL](#), T \*dxp\_dX=NULL)
- void [compute\\_1st\\_der](#) (T \*func, T \*der, [JAC\\_OPTION](#) der\_option=[JAC\\_OPTION::SPECTRAL](#), T \*dxp↵\_dX=NULL)
- T [interpolate](#) (T X, T \*func)
- void [get\\_spectral\\_coeffs](#) (T \*func, T \*coeffs)
- T [interpolate\\_with\\_spec\\_coeffs](#) (T X, T \*coeffs)
- void [compute\\_spectral\\_func\\_der](#) (T \*func, [JAC\\_OPTION](#) der\_option=[JAC\\_OPTION::SPECTRAL](#), T \*dxp↵\_dX=NULL, [COEFF\\_OPTION](#) coeff\_op=[COEFF\\_OPTION::D1](#))
- T [get\\_fun](#) (T X)
- T [get\\_der](#) (T X)
- T [get\\_d2](#) (T X)
- T [root\\_finder](#) (T f\_Xr, T \*func)
- T [pi](#) ()

### 5.2.1 Detailed Description

```
template<class T>
class aneto::spectral_domain< T >
```

Template of a single spectral domain defined in  $X = [-1, +1]$ .

The spectral domain is generated following the Lobatto-Chebyshev points:

$$X_k = -\cos(k\pi/N);$$

Notice that the grid is defined in the range  $[-1, 1]$ . Also the number of points is  $N + 1$  (from 0 to  $N$ ) but for convention  $N$  will be denoted as grid points.

The template can be used with any general type that represent real numbers where the usual operations  $+$   $-$   $*$   $/$  are defined.

Also a  $\cos$  /  $\arccos$  functions are required for the specific type.

## Template Parameters

<i>T</i>	type to be used for the spectral grid
----------	---------------------------------------

## 5.2.2 Constructor &amp; Destructor Documentation

5.2.2.1 `template<class T> aneto::spectral_domain< T >::spectral_domain ( ) [inline]`

Void constructor.

A spectral grid is created empty. It needs to be initialised in order to be used.

5.2.2.2 `template<class T> aneto::spectral_domain< T >::spectral_domain ( int po, TRANSF_OPTION transf_op = TRANSF_OPTION::DEFAULT ) [inline]`

General Constructor.

It creates a Lobatto-Chebyshev grid of *po*+1 points. Once created can not be modified until is clean out or deleted

## Parameters

<i>po</i>	Number of points ( <i>po</i> + 1)
<i>transf_op</i>	Possible options for the Chebyshev tranformation. <a href="#">TRANSF_OPTION::DEFAULT</a> is enable by default.

5.2.2.3 `template<class T> aneto::spectral_domain< T >::~~spectral_domain ( ) [inline]`

Destructor.

The used memory is freed.

## 5.2.3 Member Function Documentation

5.2.3.1 `template<class T> void aneto::spectral_domain< T >::cheb_transf ( T * in, T * out ) [inline]`

Chebyshev Transformation.

A Real Fourier Transformation (Discrete Cosine) that gives us the spectral coefficients of the function. The exact normalisation is:

$$\hat{f}_i = \sum_{j=0}^N \frac{2}{N m_j} \cos(ij\pi/N)$$

## Parameters

<i>in</i>	array with collocation points of the function.
<i>out</i>	array with the spectral coefficients.

5.2.3.2 `template<class T> void aneto::spectral_domain< T >::clean ( ) [inline]`

Cleaning the instance.

It frees the memory of the spectral grid. Once it is clean, can be initialised again.

5.2.3.3 `template<class T> void aneto::spectral_domain< T >::compute_1st_der ( T * func, T * der, JAC_OPTION der_option = JAC_OPTION::SPECTRAL, T * dxp_dX = NULL ) [inline]`

Compute the first derivative.

This function compute spectral derivative of a function.

By default compute the derivative respect to the spectral coordinate X. If the physical option is enable, the Jacobian of the transformation needs to be included.

If the collocation points of the function or the derivative are stored inside a bigger array the index to the first element can be included. This is convenient in the multidomain case.

#### Parameters

<i>func</i>	array with the value of the function in collocation points.
<i>der</i>	array where it will be stored the value of the integral in the collocation points.
<i>der_option</i>	option for spectral or physical derivative.
<i>dxp_dX</i>	array with the Jacobian values in the collocation points.

5.2.3.4 `template<class T> void aneto::spectral_domain< T >::compute_integral ( T * func, T * Integ, INTEG_OPTION i_option = INTEG_OPTION::LEFT_BC, T bndry_cond = 0, JAC_OPTION jac_option = JAC_OPTION::SPECTRAL, T * dxp_dX = NULL ) [inline]`

This function compute the following expression:

$$I_L(xp) = BC + \int_{-1}^X f(X) \frac{dxp}{dX} dX$$

or

$$I_R(xp) = BC + \int_X^{+1} f(X) \frac{dxp}{dX} dX$$

depending if i\_option is fixed to [INTEG\\_OPTION::LEFT\\_BC](#) or [INTEG\\_OPTION::RIGHT\\_BC](#) The term BC represents the boundary condition we impose in one of the boundaries. Notice that the option [INTEG\\_OPTION::CUSTOM\\_BC](#) is not allowed in this function.

For getting the integral in the physical coordinates we need to include the Jacobian between the physical and pseudospectral coordinates.

#### Parameters

<i>func</i>	array with the value of the function in collocation points
<i>Integ</i>	array where it will be stored the value of the integral in the collocation points.
<i>i_option</i>	Integral option from the left or from the right
<i>dxp_dX</i>	array with the Jacobian values in the collocation points
<i>bndry_cond</i>	value we impose in the boundary. Default value to zero.
<i>jac_option</i>	Use the spectral coordinate or including the Jacobian.
<i>dxp_dX</i>	Jacobian of the transformation between the spectral coordinate and a physical one. If it is not needed, NULL can be provided as parameter.

```
5.2.3.5  template<class T> void aneto::spectral_domain< T >::compute_spectral_func_der ( T * func,
      JAC_OPTION der_option = JAC_OPTION::SPECTRAL, T * dxp_dX = NULL, COEFF_OPTION coeff_op =
      COEFF_OPTION::D1 ) [inline]
```

This function computes the spectral coefficients and stores them internally to be used when the functions [get\\_fun\(\)](#), [get\\_der\(\)](#), [get\\_d2\(\)](#) are called. They are also used if interpolate function is called with use\_spec\_stored enabled.

An option can be included to select if you want to stored just the coeffs of the function (F), the function and the first derivative (D1) or the function and the two first derivatives (D2).

#### Parameters

<i>func</i>	array with the value of the function in collocation points.
<i>der_option</i>	option for spectral or physical derivative.
<i>dxp_dX</i>	array with the Jacobian values in the collocation points.
<i>coeff_op</i>	option for selecting which coeffs to stored.

```
5.2.3.6  template<class T> T aneto::spectral_domain< T >::get_d2 ( T X ) [inline]
```

Compute the value of the second derivative function at any point of the domain using they stored spectral coefficients.

#### Parameters

<i>X</i>	coordinate.
----------	-------------

#### Returns

interpolated value of the second derivative.

```
5.2.3.7  template<class T> T aneto::spectral_domain< T >::get_der ( T X ) [inline]
```

Compute the value of the derivative function at any point of the domain using they stored spectral coefficients.

#### Parameters

<i>X</i>	coordinate.
----------	-------------

#### Returns

interpolated value of the derivative.

```
5.2.3.8  template<class T> T aneto::spectral_domain< T >::get_fun ( T X ) [inline]
```

Compute the value of the function at any point of the domain using they stored spectral coefficients.



## Parameters

<i>X</i>	coordinate.
----------	-------------

## Returns

interpolated value of the function.

**5.2.3.9** `template<class T> int aneto::spectral_domain< T >::get_points ( ) [inline]`

Return the number of points of the spectral grid. Notice that according with our convention, the grid have N+1 points and this function returns N.

**5.2.3.10** `template<class T> void aneto::spectral_domain< T >::get_spectral_coeffs ( T * func, T * coeffs ) [inline]`

This function computes the spectral coefficients of a function and stores it in a given array. It does not stored them inside the [spectral\\_domain](#).

## Parameters

<i>func</i>	array with the value of the function in collocation points.
<i>coeffs</i>	array for storing the spectral coefficients.

**5.2.3.11** `template<class T> T aneto::spectral_domain< T >::get_X ( int k ) [inline]`

Return the spectral coordinate of point *k* Be careful! The index is not checked.

## Parameters

<i>k</i>	point of the grid.
----------	--------------------

**5.2.3.12** `template<class T> void aneto::spectral_domain< T >::initialise ( int po, TRANSF_OPTION transf_op = TRANSF_OPTION::DEFAULT ) [inline]`

## Spectral Grid Initialiser

It initialises a Lobatto-Chebyshev grid of *po*+1 points. If the object was in use it will abort the program.

## Parameters

<i>po</i>	Number of points ( <i>po</i> + 1)
<i>transf_op</i>	Possible options for the Chebyshev tranformation. <a href="#">TRANSF_OPTION::DEFAULT</a> is enable by default.

### 5.2.3.13 `template<class T> T aneto::spectral_domain< T >::interpolate ( T X, T * func ) [inline]`

This function computes the value of a function in any point of the domain. through the spectral representation. The spectral coefficients computed here are stored and can be used later calling the function `get_X`.

#### Parameters

<code>X</code>	spectral coordinate where the function will be evaluated.
<code>func</code>	array with the value of the function in collocation points.

### 5.2.3.14 `template<class T> T aneto::spectral_domain< T >::interpolate_with_spec_coeffs ( T X, T * coeffs ) [inline]`

This function compute the value of a function in any point of the domain. It uses a previously computed spectral coefficients that needs to be passed as parameter.

#### Parameters

<code>X</code>	spectral coordinate where the function will be evaluated.
<code>coeffs</code>	spectral coefficients of the function already computed.

### 5.2.3.15 `template<class T> T aneto::spectral_domain< T >::pi ( ) [inline]`

Returns Pi to the accuracy used by the object.

### 5.2.3.16 `template<class T> T aneto::spectral_domain< T >::root_finder ( T f_Xr, T * func ) [inline]`

It find the coordinate where the function  $f(X)$  have a given value.

$$f(f_X r) - f_X r = 0;$$

If the function have no root in the grid, the function will return the closest value. If the function have several roots, the function will return the first it finds.

#### Parameters

<code>f_Xr</code>	Value of the function we
<code>func</code>	Value of the function in the collocation points.

#### Returns

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

- `/home/santos/grav_local/anetolib/aneto/spectral_domain.hpp`

# Index

~spectral\_domain  
  aneto::spectral\_domain, [24](#)

aneto, [9](#)  
  COEFF\_OPTION, [10](#)  
  CUSTOM\_BC, [10](#)  
  D1, [10](#)  
  D2, [10](#)  
  DEF\_BNDS, [10](#)  
  DEFAULT, [11](#)  
  DERGRID\_OPTION, [10](#)  
  DUAL, [10](#)  
  F, [10](#)  
  FFT, [11](#)  
  GRID\_OPTION, [10](#)  
  INTEG\_OPTION, [10](#)  
  JAC\_OPTION, [10](#)  
  LEFT\_BC, [10](#)  
  MATRIX, [11](#)  
  PHYSICAL, [11](#)  
  RIGHT\_BC, [10](#)  
  SINGLE, [10](#)  
  SPECTRAL, [11](#)  
  TRANSF\_OPTION, [11](#)  
  UNIFORM, [10](#)

aneto::multidomain  
  comp\_derivative, [15](#)  
  comp\_derivative\_domain, [15](#)  
  comp\_integral, [16](#)  
  dX\_dxp, [16](#), [17](#)  
  dom\_number, [16](#)  
  dom\_points, [16](#)  
  dxp\_dX, [17](#)  
  get\_dom, [17](#)  
  get\_dom\_coord, [17](#)  
  get\_id, [18](#)  
  get\_l\_bndry, [18](#)  
  get\_l\_glob\_bndry, [18](#)  
  get\_point, [18](#)  
  get\_pointer\_dom, [18](#)  
  get\_r\_bndry, [19](#)  
  get\_r\_glob\_bndry, [19](#)  
  get\_spectral\_coeffs, [19](#)  
  get\_spectral\_domain, [19](#)  
  get\_X, [20](#)  
  get\_xp, [20](#)  
  initialise, [20](#)  
  interpolate, [21](#)  
  multidomain, [14](#)  
  number\_points, [21](#)

  root\_finder\_decreasing, [21](#)  
  root\_finder\_general, [21](#)  
  root\_finder\_increasing, [22](#)  
  xp, [22](#)

aneto::multidomain< T >, [13](#)  
aneto::spectral\_domain  
  ~spectral\_domain, [24](#)  
  cheb\_transf, [24](#)  
  clean, [25](#)  
  compute\_1st\_der, [25](#)  
  compute\_integral, [25](#)  
  compute\_spectral\_func\_der, [26](#)  
  get\_d2, [26](#)  
  get\_der, [26](#)  
  get\_fun, [26](#)  
  get\_points, [27](#)  
  get\_spectral\_coeffs, [27](#)  
  get\_X, [27](#)  
  initialise, [27](#)  
  interpolate, [27](#)  
  interpolate\_with\_spec\_coeffs, [28](#)  
  pi, [28](#)  
  root\_finder, [28](#)  
  spectral\_domain, [24](#)  
aneto::spectral\_domain< T >, [23](#)

COEFF\_OPTION  
  aneto, [10](#)  
CUSTOM\_BC  
  aneto, [10](#)  
cheb\_transf  
  aneto::spectral\_domain, [24](#)  
clean  
  aneto::spectral\_domain, [25](#)  
comp\_derivative  
  aneto::multidomain, [15](#)  
comp\_derivative\_domain  
  aneto::multidomain, [15](#)  
comp\_integral  
  aneto::multidomain, [16](#)  
compute\_1st\_der  
  aneto::spectral\_domain, [25](#)  
compute\_integral  
  aneto::spectral\_domain, [25](#)  
compute\_spectral\_func\_der  
  aneto::spectral\_domain, [26](#)

D1  
  aneto, [10](#)  
D2

- aneto, 10
- DEF\_BOUNDS
  - aneto, 10
- DEFAULT
  - aneto, 11
- DERGRID\_OPTION
  - aneto, 10
- DUAL
  - aneto, 10
- dX\_dxp
  - aneto::multidomain, 16, 17
- dom\_number
  - aneto::multidomain, 16
- dom\_points
  - aneto::multidomain, 16
- dxp\_dX
  - aneto::multidomain, 17
- F
  - aneto, 10
- FFT
  - aneto, 11
- GRID\_OPTION
  - aneto, 10
- get\_d2
  - aneto::spectral\_domain, 26
- get\_der
  - aneto::spectral\_domain, 26
- get\_dom
  - aneto::multidomain, 17
- get\_dom\_coord
  - aneto::multidomain, 17
- get\_fun
  - aneto::spectral\_domain, 26
- get\_id
  - aneto::multidomain, 18
- get\_l\_bndry
  - aneto::multidomain, 18
- get\_l\_glob\_bndry
  - aneto::multidomain, 18
- get\_point
  - aneto::multidomain, 18
- get\_pointer\_dom
  - aneto::multidomain, 18
- get\_points
  - aneto::spectral\_domain, 27
- get\_r\_bndry
  - aneto::multidomain, 19
- get\_r\_glob\_bndry
  - aneto::multidomain, 19
- get\_spectral\_coeffs
  - aneto::multidomain, 19
  - aneto::spectral\_domain, 27
- get\_spectral\_domain
  - aneto::multidomain, 19
- get\_X
  - aneto::multidomain, 20
  - aneto::spectral\_domain, 27
- get\_xp
  - aneto::multidomain, 20
- INTEG\_OPTION
  - aneto, 10
- initialise
  - aneto::multidomain, 20
  - aneto::spectral\_domain, 27
- interpolate
  - aneto::multidomain, 21
  - aneto::spectral\_domain, 27
- interpolate\_with\_spec\_coeffs
  - aneto::spectral\_domain, 28
- JAC\_OPTION
  - aneto, 10
- LEFT\_BC
  - aneto, 10
- MATRIX
  - aneto, 11
- multidomain
  - aneto::multidomain, 14
- number\_points
  - aneto::multidomain, 21
- PHYSICAL
  - aneto, 11
- pi
  - aneto::spectral\_domain, 28
- RIGHT\_BC
  - aneto, 10
- root\_finder
  - aneto::spectral\_domain, 28
- root\_finder\_decreasing
  - aneto::multidomain, 21
- root\_finder\_general
  - aneto::multidomain, 21
- root\_finder\_increasing
  - aneto::multidomain, 22
- SINGLE
  - aneto, 10
- SPECTRAL
  - aneto, 11
- spectral\_domain
  - aneto::spectral\_domain, 24
- TRANSF\_OPTION
  - aneto, 11
- UNIFORM
  - aneto, 10
- xp
  - aneto::multidomain, 22