Multiple Shooting

1

Generated by Doxygen 1.8.13

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

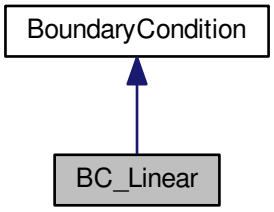
Class Documentation

3.1 BC_Linear Class Reference

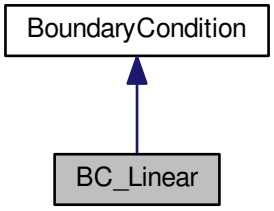
Boundary conditions for linear boundary value problems.

`#include <boundary.h>`

Inheritance diagram for BC_Linear:



Collaboration diagram for BC_Linear:



Public Member Functions

- BC_Linear (MatrixD2 _A, MatrixD2 _B, VectorD2 _c)

  Constructor.
- virtual VectorD2 **operator()** (const VectorD2 &u, const VectorD2 &v) override
- virtual MatrixD2 **diff_u** (const VectorD2 &, const VectorD2 &) override
- virtual MatrixD2 **diff_v** (const VectorD2 &, const VectorD2 &) override

3.1.1 Detailed Description

Boundary conditions for linear boundary value problems.

A linear BVP $u' = f(t,u)$ has boundary condition $r(u,v) = Au + Bv - c$, where A and B are quadratic, n^2-dimensional matrices.

On the boundary of the interval $I = [a,b]$, there holds $r(u(a),u(b)) = 0$. If $n = 2$, $A_1 y(a) = c_1$ and $B_2 y(b) = c_2$, we say the BVP is **separated**.

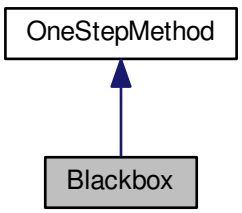
Most BVPs we consider are separated, for example the Thomas-Fermi or Troesch problems.

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

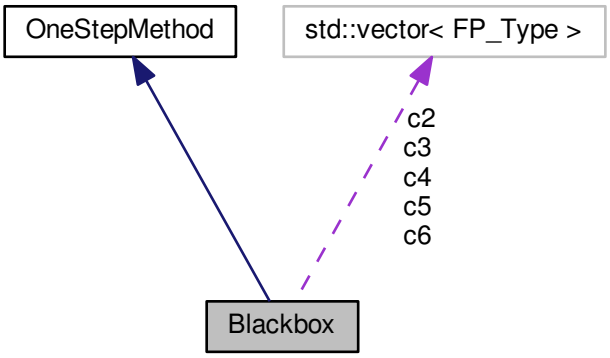
- bvp/boundary.h

3.2 Blackbox Class Reference

Inheritance diagram for Blackbox:



Collaboration diagram for Blackbox:



Public Member Functions

- virtual dealii::Vector< FP_Type > **increment_function** (const FP_Type t, const dealii::Vector< FP_Type > &y, const FP_Type h) override

Public Attributes

- const std::vector< FP_Type > **c2** = { 1./5, 1./5 }
- const std::vector< FP_Type > **c3** = { 3./10, 3./40, 9./40 }
- const std::vector< FP_Type > **c4** = { 4./5, 44./45, 56./15, 32./9 }
- const std::vector< FP_Type > **c5**
- const std::vector< FP_Type > **c6**
- const FP_Type **s1** = 35./384
- const FP_Type **s2** = 0.
- const FP_Type **s3** = 500./1113
- const FP_Type **s4** = 125./192
- const FP_Type **s5** = 2187./6784
- const FP_Type **s6** = 11./84

3.2.1 Member Data Documentation

3.2.1.1 c5

```
const std::vector<FP_Type> Blackbox::c5
```

Initial value:

```
= { 8./9, 19372./6561, 25360./2187,
                                    64448./6561, 212./729 }
```

3.2.1.2 c6

```
const std::vector<FP_Type> Blackbox::c6
```

Initial value:

```
= { 1., 9017./3168, 355./33,
                                    46732./5247, 49./176, 5103./18656 }
```

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

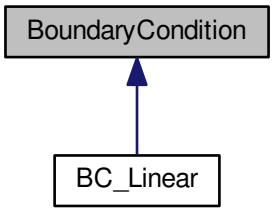
- test/test_runge_kutta.h

3.3 BoundaryCondition Class Reference

Abstract base class for boundary conditions $r(u, v)$ in boundary value problems.

```
#include <boundary.h>
```

Inheritance diagram for BoundaryCondition:



Public Member Functions

- virtual VectorD2 **operator()** (const VectorD2 &u, const VectorD2 &v)=0
- virtual MatrixD2 **diff_u** (const VectorD2 &u, const VectorD2 &v)=0
- virtual MatrixD2 **diff_v** (const VectorD2 &u, const VectorD2 &v)=0

3.3.1 Detailed Description

Abstract base class for boundary conditions $r(u, v)$ in boundary value problems.

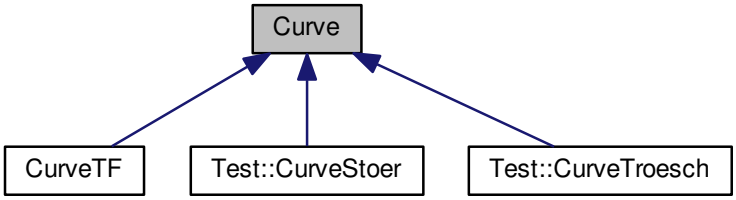
Supports evaluation and differentation over u and v.

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

- bvp/boundary.h

3.4 Curve Class Reference

Inheritance diagram for Curve:



Public Member Functions

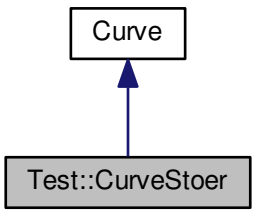
- **Curve** (size_t n)
- virtual VectorD2 **operator()** (FP_Type t)=0
- size_t **n_dim** () const

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

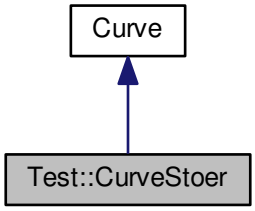
- lac/lac_types.h

3.5 Test::CurveStoer Class Reference

Inheritance diagram for Test::CurveStoer:



Collaboration diagram for Test::CurveStoer:



Public Member Functions

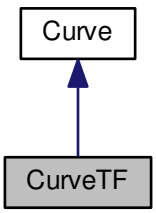
- VectorD2 **operator()** (FP_Type t)

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

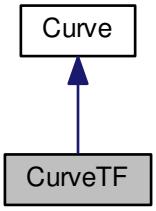
- test/test_bvp.h

3.6 CurveTF Class Reference

Inheritance diagram for CurveTF:



Collaboration diagram for CurveTF:



Public Member Functions

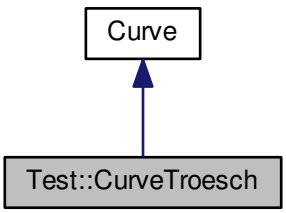
- VectorD2 **operator()** (FP_Type t)

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

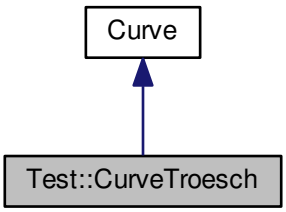
- multiple-shooting.cc

3.7 Test::CurveTroesch Class Reference

Inheritance diagram for Test::CurveTroesch:



Collaboration diagram for Test::CurveTroesch:



Public Member Functions

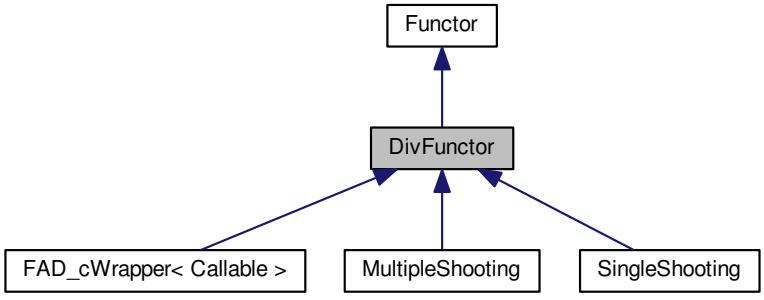
- VectorD2 **operator()** (FP_Type t)

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

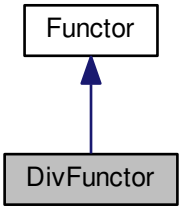
- test/test_bvp.h

3.8 DivFunctor Class Reference

Inheritance diagram for DivFunctor:



Collaboration diagram for DivFunctor:



Public Member Functions

- virtual MatrixD2 **diff** (const VectorD2 &x)=0

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

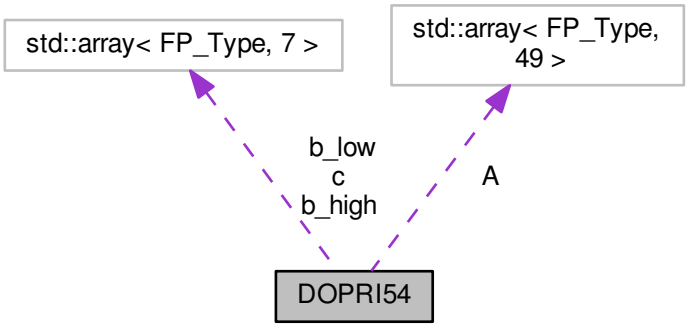
- lac/lac_types.h

3.9 DOPRI54 Struct Reference

Butcher Tableau for the Dormand-Prince method of order 5(4).

`#include <tableau.h>`

Collaboration diagram for DOPRI54:



Public Attributes

- const size_t **n** = 7
- const size_t **p** = 4
- const std::array< FP_Type, 7 > **c**
- const std::array< FP_Type, 49 > **A**
- const std::array< FP_Type, 7 > **b_high**
- const std::array< FP_Type, 7 > **b_low**

3.9.1 Detailed Description

Butcher Tableau for the Dormand-Prince method of order 5(4).

struct DOPRI54

3.9.2 Member Data Documentation

3.9.2.1 A

```
const std::array<FP_Type, 49> DOPRI54::A
```

Initial value:

```
= {
    0,            0,             0,            0,          0,              0,       0,
    1./5,         0,             0,            0,          0,              0,       0,
    3./40,        9./40,         0,            0,          0,              0,       0,
    44./45,       -56./15,       32./9,        0,          0,              0,       0,
    19372./6561,  -25360./2187,  64448./6561,  -212./729,  0,              0,       0,
    9017./3168,   -355./33,      46732./5247,  49./176,    -5103./18656,   0,       0,
    35./384,      0,             500./1113,    125./192,   -2187./6784,    11./84,  0
  }
```

3.9.2.2 b_high

```
const std::array<FP_Type, 7> DOPRI54::b_high
```

Initial value:

```
= {
    35./384, 0, 500./1113, 125./192, -2187./6784, 11./84, 0
  }
```

3.9.2.3 b_low

```
const std::array<FP_Type, 7> DOPRI54::b_low
```

Initial value:

```
= {
    5179./57600, 0, 7571./16695, 393./640, -92097./339200, 187./2100, 1./40
  }
```

3.9.2.4 c

```
const std::array<FP_Type, 7> DOPRI54::c
```

Initial value:

```
= {
    0, 1./5, 3./10, 4./5, 8./9, 1., 1.
  }
```

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

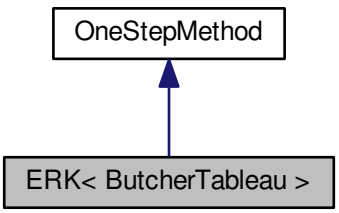
- ivp/tableau.h

3.10 ERK< ButcherTableau > Class Template Reference

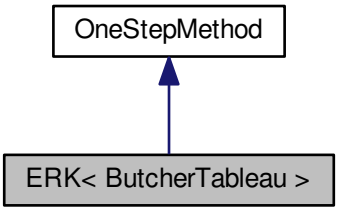
Solve an IVP of shape $u'(t) = f(t, u(t)), u(t_0) = u_0$ using an explicit Runge-Kutta method.

```
#include <runge_kutta.h>
```

Inheritance diagram for ERK< ButcherTableau >:



Collaboration diagram for ERK< ButcherTableau >:



Public Member Functions

- ERK (TimeFunctor &f, FP_Type t0, VectorD2 u0, bool var_eq=false, Curve *u=nullptr)

  ERK.
- VectorD2 **k_increment** (FP_Type t, const VectorD2 &y, FP_Type h, const VectorD2 &b)
- std::pair< VectorD2, MatrixD2 > **k_variational** (FP_Type t, const VectorD2 &y, FP_Type h, const VectorD2 &b, const MatrixD2 &Y)
- virtual VectorD2 **increment_function** (FP_Type t, const VectorD2 &y, FP_Type h) override
- virtual std::pair< VectorD2, MatrixD2 > **increment_variational** (FP_Type t, const VectorD2 &y, FP_Type h, const MatrixD2 &Y) override
- size_t **n_misfires** () const
- void iterate_with_ssc (FP_Type t_lim, FP_Type h0, FP_Type TOL, FP_Type C=2)

  iterate_with_ssc
- void **print_step_size** (std::ostream &out)

3.10.1 Detailed Description

template<typename ButcherTableau>
class ERK< ButcherTableau >

Solve an IVP of shape $u'(t) = f(t, u(t)), u(t_0) = u_0$ using an explicit Runge-Kutta method.

3.10.2 Constructor & Destructor Documentation

3.10.2.1 ERK()

```
template<typename ButcherTableau >
ERK< ButcherTableau >::ERK (
            TimeFunctor & f,
            FP_Type t0,
            VectorD2 u0,
            bool var_eq = false,
            Curve * u = nullptr )  [inline]
```

ERK.

This constructor follows `OneStepMethod`. The used Butcher tableau is specified via a template parameter. Whether a method is embedded (for step-size control) is determined by checking if lower-order weights are available in the Butcher tableau.

Only fixed order methods or methods of order $p+1, p$ are supported. See `iterate_with_ssc` for details.

3.10.3 Member Function Documentation

3.10.3.1 iterate_with_ssc()

```
template<typename ButcherTableau >
void ERK< ButcherTableau >::iterate_with_ssc (
            FP_Type t_lim,
            FP_Type h0,
            FP_Type TOL,
            FP_Type C = 2 )  [inline]
```

iterate_with_ssc

Parameters

t_lim	
h0	Initial time step width.
TOL	Threshold for the local error at each time step.
C	Threshold for the global error at each time step.

Iteration function with support for step-size control. In each step, two solutions are computed, for order $p+1$ and p respectively. Whether a step is accepted is then decided by computing an "optimal" step width. If the optimal width is smaller than the current value, the step is repeated using the new width.

More complex algorithms which choose method order dynamically (for example the KARP method for orders 1 to 5) are not implemented, but may be more suitable for discontinuous or rapidly changing ODEs. See the paper by Cash-Karp for details.

http://www.elegio.it/mc2/rk/doc/p201-cash-karp.pdf

When solving the variational equation, the step width is *only* determined by the initial-value problem. This approach may be improved by first solving the IVP with a *continuous* Runge-Kutta method. See the paper by L. Rández, 1990.

https://www.sciencedirect.com/science/article/pii/037704279290226N

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

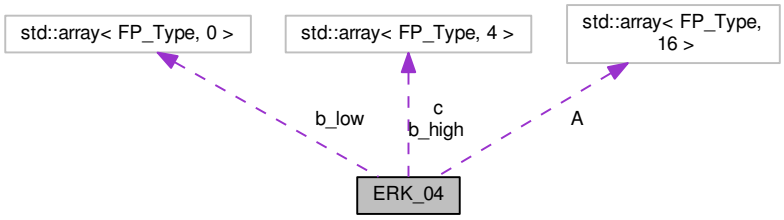
- ivp/runge_kutta.h

3.11 ERK_04 Struct Reference

Butcher Tableau for the classic Runge-Kutta method.

`#include <tableau.h>`

Collaboration diagram for ERK_04:



Public Attributes

- const size_t **n** = 4
- const size_t **p** = 4
- const std::array< FP_Type, 4 > **c**
- const std::array< FP_Type, 16 > **A**
- const std::array< FP_Type, 4 > **b_high**
- const std::array< FP_Type, 0 > **b_low** = {}

3.11.1 Detailed Description

Butcher Tableau for the classic Runge-Kutta method.

3.11.2 Member Data Documentation

3.11.2.1 A

```
const std::array<FP_Type, 16> ERK_04::A
```

Initial value:

```
= {
    0,    0,    0,    0,
    0.5,  0,    0,    0,
    0,    0.5,  0,    0,
    0,    0,    1.0,  0
  }
```

3.11.2.2 b_high

```
const std::array<FP_Type, 4> ERK_04::b_high
```

Initial value:

```
= {
    1./6, 2./6, 2./6, 1./6
  }
```

3.11.2.3 c

```
const std::array<FP_Type, 4> ERK_04::c
```

Initial value:

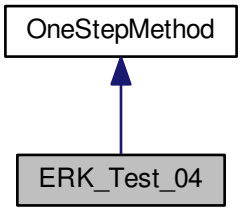
```
= {
    0, 0.5, 0.5, 1.0
  }
```

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

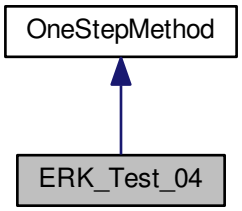
- ivp/tableau.h

3.12 ERK_Test_04 Class Reference

Inheritance diagram for ERK_Test_04:



Collaboration diagram for ERK_Test_04:



Public Member Functions

- virtual dealii::Vector< FP_Type > **increment_function** (FP_Type t, const dealii::Vector< FP_Type > &y, FP_Type h) override

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

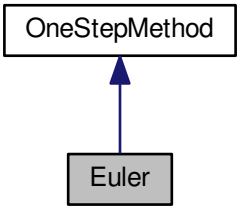
- test/test_runge_kutta.h

3.13 Euler Class Reference

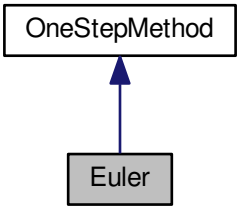
Classic Euler method of order 1.

```
#include <euler.h>
```

Inheritance diagram for Euler:



Collaboration diagram for Euler:



Additional Inherited Members

3.13.1 Detailed Description

Classic Euler method of order 1.

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

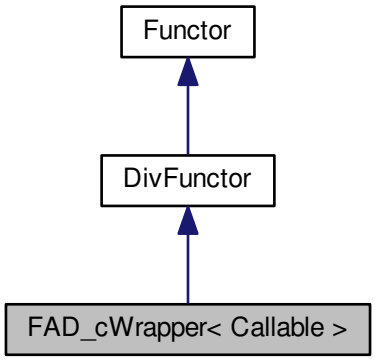
- ivp/euler.h

3.14 FAD_cWrapper< Callable > Class Template Reference

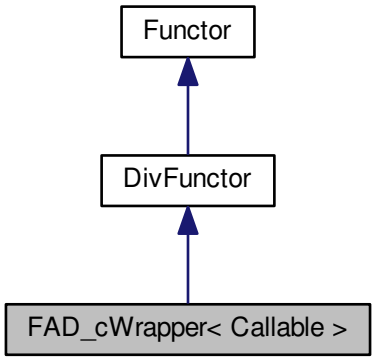
Class for equidimensional problems $f : \mathbb{R}^d \to \mathbb{R}^d$ using AD.

`#include <forward_ad.h>`

Inheritance diagram for FAD_cWrapper< Callable >:



Collaboration diagram for FAD_cWrapper< Callable >:



Public Member Functions

- **FAD_cWrapper** (Callable f, size_t dim, FP_Type _t=0)
- virtual VectorD2 **operator()** (const VectorD2 &u) override
- virtual MatrixD2 **diff** (const VectorD2 &u) override

3.14.1 Detailed Description

template<typename Callable>
class FAD_cWrapper< Callable >

Class for equidimensional problems $f : \mathbb{R}^d \to \mathbb{R}^d$ using AD.

Adapter for `DivFunctor`.

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

- base/forward_ad.h

3.15 FAD_Setup< Callable > Class Template Reference

Class representing the time-dependent function with n components $f : I \times \mathbb{R}^m \to \mathbb{R}^n$.

`#include <forward_ad.h>`

Public Member Functions

- **FAD_Setup** (Callable _f, size_t _m, size_t _n)
- **FAD_Setup** (Callable _f, size_t _d)
- void init (FP_Type t, const VectorD2 &u)

  Evaluate function (t, u) on AD variables. Results may be retrieved using the `value()` and `diff()` methods.
- VectorD2 value () const

  Return the value (t, u) as a `dealii` vector.
- MatrixD2 diff () const

  Return the partial derivatives $\frac{\partial f}{\partial u_1}, \cdots, \frac{\partial f}{\partial u_m}$ as a `dealii` matrix.

3.15.1 Detailed Description

template<typename Callable>
class FAD_Setup< Callable >

Class representing the time-dependent function with n components $f : I \times \mathbb{R}^m \to \mathbb{R}^n$.

Both evaluation and automatic differentation (using the Sacado package from the Trilinos library) are supported.

For general information, see SESS 2007, E. Phipps:

`https://software.sandia.gov/SESS/past_seminars/111307_Phipps.html`

3.15.2 Member Function Documentation

3.15.2.1 init()

```
template<typename Callable >
FAD_Setup< Callable >::init (
            FP_Type t,
            const VectorD2 & u )  [inline]
```

Evaluate function (t, u) on AD variables. Results may be retrieved using the `value()` and `diff()` methods.

$u_1, \cdots, u_m$ are set as independent variables.

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

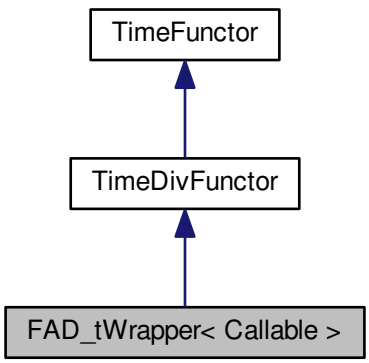
- base/forward_ad.h

3.16 FAD_tWrapper< Callable > Class Template Reference

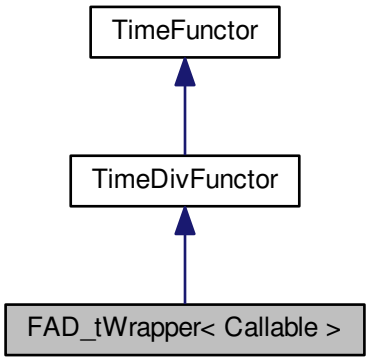
Class for equidimensional, time-dependent problems $f : I \times \mathbb{R}^d \to \mathbb{R}^d$ using AD.

```
#include <forward_ad.h>
```

Inheritance diagram for FAD_tWrapper< Callable >:



Collaboration diagram for FAD_tWrapper< Callable >:



Public Member Functions

- **FAD_tWrapper** (Callable f, size_t dim)
- virtual VectorD2 **operator()** (FP_Type t, const VectorD2 &u) override
- virtual MatrixD2 **diff** (FP_Type t, const VectorD2 &u) override

3.16.1 Detailed Description

template<typename Callable>
class FAD_tWrapper< Callable >

Class for equidimensional, time-dependent problems $f : I \times \mathbb{R}^d \to \mathbb{R}^d$ using AD.

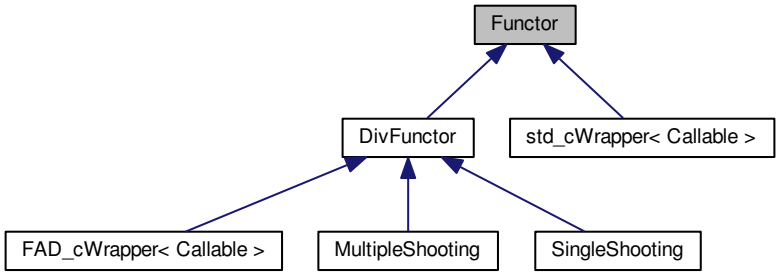
Adapter for `TimeDivFunctor`.

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

- base/forward_ad.h

3.17 Functor Class Reference

Inheritance diagram for Functor:



Public Member Functions

- **Functor** (size_t n)
- virtual VectorD2 **operator()** (const VectorD2 &x)=0
- size_t **n_dim** () const

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

- lac/lac_types.h

3.18 GnuPlot Class Reference

Plot output data with Gnuplot.

`#include <gnuplot.h>`

Public Member Functions

- **GnuPlot** (std::string _filename, std::ofstream &_output_file)
- void **plot_with_lines** (size_t dim, std::string style="lines", bool plot_3d=false)

3.18.1 Detailed Description

Plot output data with Gnuplot.

This class constructs a command line and passes it to gnuplot via `std::system`.

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

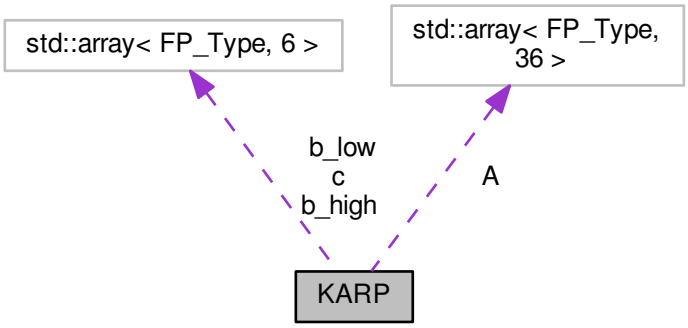
- base/gnuplot.h

3.19 KARP Struct Reference

Butcher Tableau for the Cash-Karp method of order 5(4).

`#include <tableau.h>`

Collaboration diagram for KARP:



Public Attributes

- const size_t **n** = 6
- const size_t **p** = 4
- const std::array< FP_Type, 6 > **c**
- const std::array< FP_Type, 36 > **A**
- const std::array< FP_Type, 6 > **b_high**
- const std::array< FP_Type, 6 > **b_low**

3.19.1 Detailed Description

Butcher Tableau for the Cash-Karp method of order 5(4).

Details on this method and its lower order variants are available in the original paper:

http://www.elegio.it/mc2/rk/doc/p201-cash-karp.pdf

3.19.2 Member Data Documentation

3.19.2.1 A

```
const std::array<FP_Type, 36> KARP::A
```

Initial value:

```
= {
    0,            0,         0,           0,              0,          0,
    1./5,         0,         0,           0,              0,          0,
    3./40,        9./40,     0,           0,              0,          0,
    3./10,        -9./10,    6./5,        0,              0,          0,
    -11./54,      5./2,      -70./27,     35./27,         0,          0,
    1631./55296,  175./512,  575./13824,  44275./110592,  253./4096,  0
  }
```

3.19.2.2 b_high

```
const std::array<FP_Type, 6> KARP::b_high
```

Initial value:

```
= {
    37./378, 0, 250./621, 125./594, 0, 512./1771
  }
```

3.19.2.3 b_low

```
const std::array<FP_Type, 6> KARP::b_low
```

Initial value:

```
= {
    2825./27648, 0, 18575./48384, 13525./55296, 277./14336, 1./4
  }
```

3.19.2.4 c

```
const std::array<FP_Type, 6> KARP::c
```

Initial value:

```
= {
    0, 1./5, 3./10, 3./5, 1, 7./8
  }
```

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

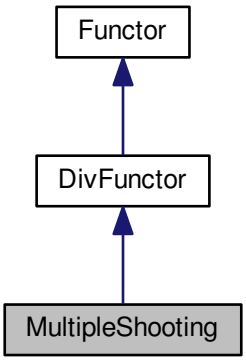
- ivp/tableau.h

3.20 MultipleShooting Class Reference

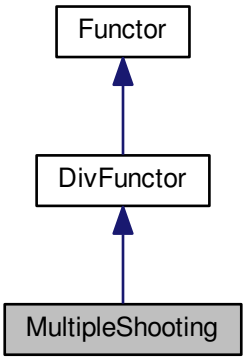
Multiple shooting method for boundary value problems. Represented as a differentiable function $F : \mathbb{R}^d \to \mathbb{R}^d$.

`#include <methods.h>`

Inheritance diagram for MultipleShooting:



Collaboration diagram for MultipleShooting:



Public Member Functions

- MultipleShooting (ShootingFunction &_M, std::vector< FP_Type > _t, BoundaryCondition &_r)
- virtual VectorD2 operator() (const VectorD2 &s) override

  Retrieve $F_1(s_1, s_2), \cdots, F_{m-1}(s_{m-1}, s_m), F_m(s_1, s_m)$ *for the vector* $s = (s_1, \cdots, s_m)$.
- virtual MatrixD2 diff (const VectorD2 &s) override

  Retrieve the matrix $DF(s)$ *for the vector* $s = (s_1, \cdots, s_m)$.

3.20.1 Detailed Description

Multiple shooting method for boundary value problems. Represented as a differentiable function $F : \mathbb{R}^d \to \mathbb{R}^d$.

See Stoer, Num. Math. 2, pp. 215.

3.20.2 Constructor & Destructor Documentation

3.20.2.1 MultipleShooting()

```
MultipleShooting::MultipleShooting (
            ShootingFunction & _M,
            std::vector< FP_Type > _t,
            BoundaryCondition & _r )  [inline]
```

Constructor. As `SingleShooting`, but a series of time points (interval subdivision) must be supplied.

3.20.3 Member Function Documentation

3.20.3.1 diff()

```
virtual MatrixD2 MultipleShooting::diff (
            const VectorD2 & s )  [inline], [override], [virtual]
```

Retrieve the matrix $DF(s)$ for the vector $s = (s_1, \cdots, s_m)$.

Blocks are implemented manually and through `dealii::FullMatrix::add`.

Implements DivFunctor.

3.20.3.2 operator()()

```
virtual VectorD2 MultipleShooting::operator() (
            const VectorD2 & s )  [inline], [override], [virtual]
```

Retrieve $F_1(s_1, s_2), \cdots, F_{m-1}(s_{m-1}, s_m), F_m(s_1, s_m)$ for the vector $s = (s_1, \cdots, s_m)$.

Vector "blocks" are implemented manually.

Implements Functor.

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

- bvp/methods.h

3.21 Newton< Callable > Class Template Reference

Newton method with globalization and rank-1 updates.

`#include <newton.h>`

Public Member Functions

- Newton (Callable _f, size_t _dim, FP_Type _TOL=1e-6, bool _ssc=true, size_t _ssc_lim=20)

  Constructor. Initialize the method with a function $f : \mathbb{R}^d \to \mathbb{R}^d$ of dimension d.
- void step (const MatrixD2 &J_inv, VectorD2 &x)

  Perform a Newton step.
- VectorD2 iterate (const VectorD2 &x0, size_t step_limit=25)

  Perform Newton steps until $\|f(s)\| < TOL$.
- VectorD2 iterate_broyden (const VectorD2 &x0, size_t skips=5, size_t step_limit=50)

  Use rank-1 updates during iteration.

3.21.1 Detailed Description

template<typename Callable>
class Newton< Callable >

Newton method with globalization and rank-1 updates.

This class implements the Newton method for finding the root of a non-linear equation $f : \mathbb{R}^d \to \mathbb{R}^d$. Step size control is available as globalization strategy (Def 4.2.3) in case a good intial guess of the root is not available.

3.21.2 Constructor & Destructor Documentation

3.21.2.1 Newton()

```
template<typename Callable >
Newton< Callable >::Newton (
            Callable _f,
            size_t _dim,
            FP_Type _TOL = 1e-6,
            bool _ssc = true,
            size_t _ssc_lim = 20 )  [inline]
```

Constructor. Initialize the method with a function $f : \mathbb{R}^d \to \mathbb{R}^d$ of dimension d.

The TOL parameter specifies when a root s is accepted. Step size control is activated by default; as the value j used within need not be bounded, the maximum may be set here (Remark 4.2.4).

3.21.3 Member Function Documentation

3.21.3.1 iterate()

```
template<typename Callable >
Newton< Callable >::iterate (
            const VectorD2 & x0,
            size_t step_limit = 25 )  [inline]
```

Perform Newton steps until $\|f(s)\| < TOL$.

As the Jacobian is computed in this function, we assume that f is differentiable, i.e. has an available `diff()` method.

The maximum amount of steps may be specified, defaulting to 25. In our context, exceeding this limit has indicated either a program error or an unsuitably chosen method. Should this occur, the function therefore exits with an exception.

To solve the resulting linear systems, LU decomposition is used via dealii and LAPACK. The Jacobian that results from the multiple shooting method is sparse, but of small dimension in the problems we consider. (In particular, the Thomas-Fermi problem with 20 subintervals results in a 40 x 40 Jacobian.)

3.21.3.2 iterate_broyden()

```
template<typename Callable >
Newton< Callable >::iterate_broyden (
            const VectorD2 & x0,
            size_t skips = 5,
            size_t step_limit = 50 )  [inline]
```

Use rank-1 updates during iteration.

The Jacobian is computed periodicially, as specified trough the `skips` parameter. This method converges slower, with the default step limit is chosen accordingly.

For **small problems** such as the Thomas-Fermi problem, the slow convergence rate was more expensive than computing the Jacobian in each step.

For **large problems**, the Sherman-Morrison formula may be used to update J^{-1} directly, instead of performing an LU decomposition. Due to lower relevance for small problems, this is not implemented here.

3.21.3.3 step()

```
template<typename Callable >
Newton< Callable >::step (
            const MatrixD2 & J_inv,
            VectorD2 & x )  [inline]
```

Perform a Newton step.

The Jacobian *inverse* is specifically taken as argument, to allow use of this function for both Newton and quasi-↩ Newton methods.

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

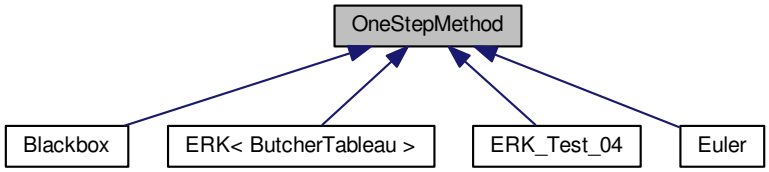
- algo/newton.h

3.22 OneStepMethod Class Reference

Solve an IVP of shape $u'(t) = f(t, u(t)), u(t_0) = u_0$ using a one-step method.

`#include <eos_method.h>`

Inheritance diagram for OneStepMethod:



Public Member Functions

- OneStepMethod (TimeFunctor &_f, FP_Type _t0, VectorD2 _u0, bool _var_eq=false, Curve *_u=nullptr)
  OneStepMethod.
- VectorD2 **approx** () const
- FP_Type **endpoint** () const
- size_t **n_steps** () const
- MatrixD2 **fund_matrix** () const
- void print (std::ostream &out=std::cout) const
  Print the approximate solution at each time step in a tabular format.
- bool sol_is_nan (const VectorD2 &y)
  Check if an a ::Vector element is NaN (`std::isnan`)
- void **reset** ()
- void **save_step** (const FP_Type &t, const VectorD2 &u)
- virtual VectorD2 **increment_function** (FP_Type, const VectorD2 &, FP_Type)
- virtual std::pair< VectorD2, MatrixD2 > **increment_variational** (FP_Type, const VectorD2 &, FP_Type, const MatrixD2 &)
- void **iterate_up_to** (FP_Type t_lim, FP_Type h, FP_Type C=2)

Friends

- class **Blackbox**
- class **Euler**
- class **ERK_Test_04**
- template<typename BTab >
  class **ERK**

3.22.1 Detailed Description

Solve an IVP of shape $u'(t) = f(t, u(t)), u(t_0) = u_0$ using a one-step method.

The common wrapped functionality includes collecting of intermediary computation results.

3.22.2 Constructor & Destructor Documentation

3.22.2.1 OneStepMethod()

```
OneStepMethod::OneStepMethod (
            TimeFunctor & _f,
            FP_Type _t0,
            VectorD2 _u0,
            bool _var_eq = false,
            Curve * _u = nullptr )  [inline]
```

OneStepMethod.

Parameters

_f	Right-hand side of the ODE.
_t0	Initial time value.
_u0	Initial value, $u(t_0) = u_0$.
_var_eq	If true, solve the variational equation.
_u	Exact solution of the IVP.

Solving the variational equation is done *simultaneously* with solving the IVP. In particular, the same step width is used for both problems. See the ERK documentation for further discussion.

A provided `exact` solution $u(t)$ may be used to verify the local error at each time step. The involved constant are specified to the iteration methods.

3.22.3 Member Function Documentation

3.22.3.1 print()

```
void OneStepMethod::print (
            std::ostream & out = std::cout ) const  [inline]
```

Print the approximate solution at each time step in a tabular format.

Parameters

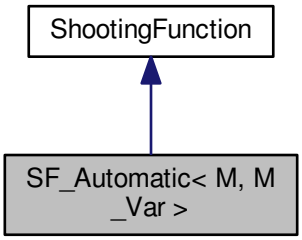
File	stream to print to, for example stdout or an output file.

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

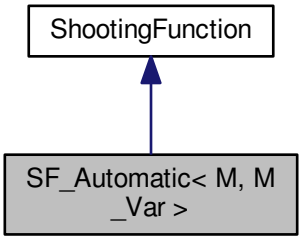
- ivp/eos_method.h

3.23 SF_Automatic< M, M_Var > Class Template Reference

Inheritance diagram for SF_Automatic< M, M_Var >:



Collaboration diagram for SF_Automatic< M, M_Var >:



Public Member Functions

- virtual VectorD2 solve_y (FP_Type t0, FP_Type t1, const VectorD2 &s) override

  Solve $y(t;t_0,s)$ *in* $t=t_1$.
- virtual std::pair< VectorD2, MatrixD2 > solve_Z (FP_Type t0, FP_Type t1, const VectorD2 &s) override

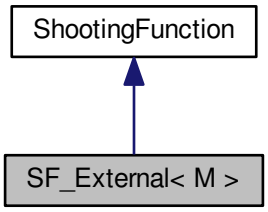
  Solve $D_s y(t;t_0,s)$ *in* $t=t_1$ *by automatic differentation.*

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

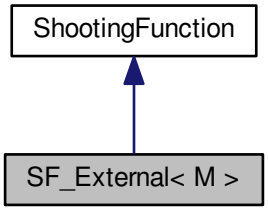
- bvp/shooting.h

3.24 SF_External< M > Class Template Reference

Inheritance diagram for SF_External< M >:



Collaboration diagram for SF_External< M >:



Public Member Functions

- virtual VectorD2 solve_y (FP_Type t0, FP_Type t1, const VectorD2 &s) override

  Solve $y(t;t_0,s)$ *in* $t=t_1$.
- virtual std::pair< VectorD2, MatrixD2 > solve_Z (FP_Type t0, FP_Type t1, const VectorD2 &s) override

  Solve $D_s y(t;t_0,s)$ *in* $t=t_1$ *by external differentation.*

3.24.1 Member Function Documentation

3.24.1.1 solve_Z()

```
template<typename M >
virtual std::pair<VectorD2, MatrixD2> SF_External< M >::solve_Z (
            FP_Type t0,
            FP_Type t1,
            const VectorD2 & s )  [inline], [override], [virtual]
```

Solve $D_s y(t; t_0, s)$ in $t = t_1$ by external differentation.

For the choice of TOL in the adaptive method and the constant eps, see Stoer, Num. Math. 2, pp. 192.

Implements ShootingFunction.

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

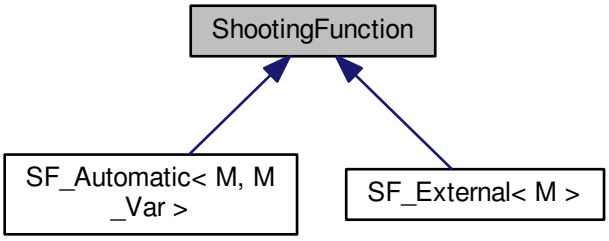
- bvp/shooting.h

3.25 ShootingFunction Class Reference

Integrate an IVP $u' = f(t, u)$ on a given time interval $[t_0, t_1]$ dependent on the initial value $s = u(t_0)$.

`#include <shooting.h>`

Inheritance diagram for ShootingFunction:



Public Member Functions

- ShootingFunction (TimeFunctor &_f, bool _ssc=true, FP_Type _h0=1e-1, FP_Type _TOL=1e-4)
  Constructor.
- size_t n_dim () const
  Return the dimension of $F(s)$.
- virtual VectorD2 solve_y (FP_Type t0, FP_Type t1, const VectorD2 &s)=0
  Solve $y(t; t_0, s)$ in $t = t_1$.
- virtual std::pair< VectorD2, MatrixD2 > solve_Z (FP_Type t0, FP_Type t1, const VectorD2 &s)=0
  Solve $D_s y(t; t_0, s)$ in $t = t_1$.

Friends

- template<typename M >
  class **SF_External**
- template<typename M , typename N >
  class **SF_Automatic**

3.25.1 Detailed Description

Integrate an IVP $u' = f(t, u)$ on a given time interval $[t_0, t_1]$ dependent on the initial value $s = u(t_0)$.

Notation: $f(t, u(t; s))$ or $y(t; t_0, s)$.

The partial derivatives $D_s f = \frac{\partial f}{\partial s}$ are computed approximatively (*external differentation*) or by solving the variational equation $Y' = \nabla_u f(t, u(t))Y$.

3.25.2 Constructor & Destructor Documentation

3.25.2.1 ShootingFunction()

```
ShootingFunction::ShootingFunction (
            TimeFunctor & _f,
            bool _ssc = true,
            FP_Type _h0 = 1e-1,
            FP_Type _TOL = 1e-4 )  [inline]
```

Constructor.

When disabling step-size control, the intial step width h_0 should be set to a smaller value, for example $1e-3$.

The appropriate value for TOL depends on the chosen method for differentiating F. For example, when computing DF with external differentation, F should be integrated as accurately as possible.

3.25.3 Member Function Documentation

3.25.3.1 solve_Z()

```
virtual std::pair<VectorD2, MatrixD2> ShootingFunction::solve_Z (
            FP_Type t0,
            FP_Type t1,
            const VectorD2 & s )  [pure virtual]
```

Solve $D_s y(t; t_0, s)$ in $t = t_1$.

As solving D_s typically involves solving f, a pair of solutions is returned.

Implemented in SF_Automatic< M, M_Var >, and SF_External< M >.

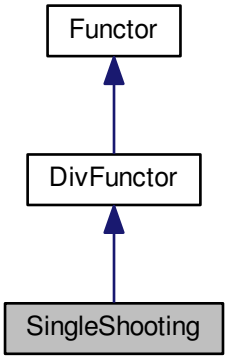
The documentation for this class was generated from the following file:

- bvp/shooting.h

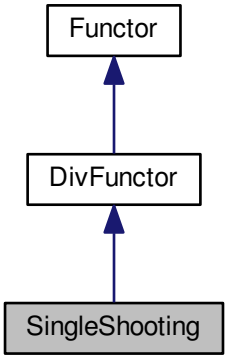
3.26 SingleShooting Class Reference

`#include <methods.h>`

Inheritance diagram for SingleShooting:



Collaboration diagram for SingleShooting:



Public Member Functions

- SingleShooting (ShootingFunction &_M, FP_Type _a, FP_Type _b, BoundaryCondition &_r)

  Constructor for the single shooting method. Accepts any valid boundary condition r on a time interval $[a, b]$.
- virtual VectorD2 operator() (const VectorD2 &s) override

  Retrieve $F(s)$ by solving the IVP $y(b; s)$.
- virtual MatrixD2 diff (const VectorD2 &s) override

  Retrieve $DF(s)$.

3.26.1 Detailed Description

Single shooting method for boundary value problems. Represented as a differentiable function $F : \mathbb{R}^d \to \mathbb{R}^d$.

See Stoer, Num. Math. 2, pp. 195.

3.26.2 Member Function Documentation

3.26.2.1 diff()

```
virtual MatrixD2 SingleShooting::diff (
            const VectorD2 & s )  [inline], [override], [virtual]
```

Retrieve $DF(s)$.

DF may be derived in several ways:

- Automatic differentation
- Internal differentation
- External differentation

Automatic and internal differentation imply solving the *variational equation*. See `ShootingFunction` for implemented methods.

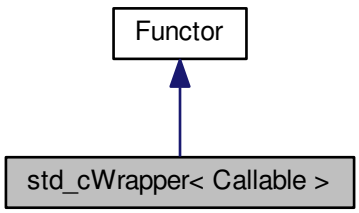
Implements DivFunctor.

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

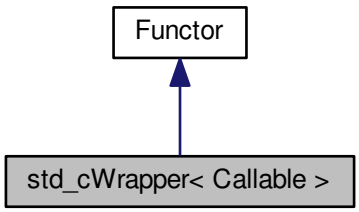
- bvp/methods.h

3.27 std_cWrapper< Callable > Class Template Reference

Inheritance diagram for std_cWrapper< Callable >:



Collaboration diagram for std_cWrapper< Callable >:



Public Member Functions

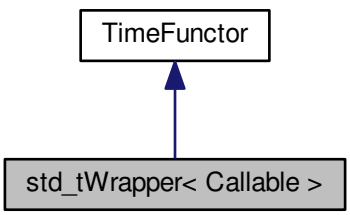
- **std_cWrapper** (Callable _f, size_t dim)
- virtual VectorD2 **operator()** (const VectorD2 &u) override

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

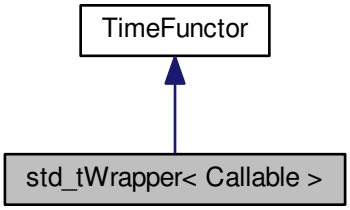
- lac/lac_types.h

3.28 std_tWrapper< Callable > Class Template Reference

Inheritance diagram for std_tWrapper< Callable >:



Collaboration diagram for std_tWrapper< Callable >:



Public Member Functions

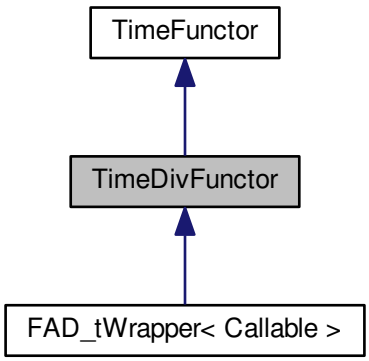
- **std_tWrapper** (Callable _f, size_t dim)
- virtual VectorD2 **operator()** (FP_Type t, const VectorD2 &u) override

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

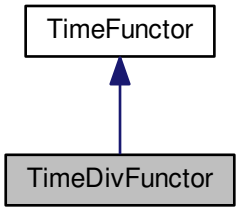
- lac/lac_types.h

3.29 TimeDivFunctor Class Reference

Inheritance diagram for TimeDivFunctor:



Collaboration diagram for TimeDivFunctor:



Public Member Functions

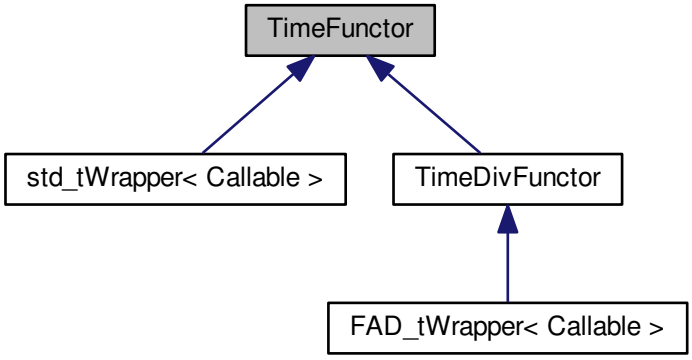
- virtual MatrixD2 **diff** (FP_Type t, const VectorD2 &u)=0

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

- lac/lac_types.h

3.30 TimeFunctor Class Reference

Inheritance diagram for TimeFunctor:



Public Member Functions

- **TimeFunctor** (size_t n)
- virtual VectorD2 **operator()** (FP_Type t, const VectorD2 &u)=0
- size_t **n_dim** () const

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

- lac/lac_types.h

Index