Boost.Numeric.Odeint

Karsten Ahnert

Mario Mulansky

Copyright © 2009-2012 Karsten Ahnert and Mario Mulansky

Distributed under the Boost Software License, Version 1.0. (See accompanying file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)

Table of Contents

Getting started 3
    Overview 3
    Usage, Compilation, Headers 7
    Short Example 8
Tutorial 10
    Harmonic oscillator 10
    Solar system 13
    Chaotic systems and Lyapunov exponents 17
    Stiff systems 20
    Complex state types 22
    Lattice systems 23
    Ensembles of oscillators 25
    Using boost::units 28
    Using matrices as state types 30
    Using arbitrary precision floating point types 32
    Self expanding lattices 33
    Using CUDA (or OpenMP, TBB, ...) via Thrust 36
    Using OpenCL via VexCL 45
    Parallel computation with OpenMP and MPI 47
    All examples 54
odeint in detail 58
    Steppers 58
    Generation functions 75
    Integrate functions 77
    Iterators and Ranges 79
    State types, algebras and operations 83
    Using boost::ref 92
    Using boost::range 93
    Binding member functions 95
Concepts 97
    System 97
    Second Order System 97
    Symplectic System 98
    Simple Symplectic System 99
    Implicit System 100
    Stepper 100
    Error Stepper 102
    Controlled Stepper 104
    Dense Output Stepper 106
    State Algebra Operations 107
    State Wrapper 111
Literature 113

render X

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Acknowledgments 114
odeint Reference 115
    Header <boost/numeric/odeint/integrate/integrate.hpp> 115
    Header <boost/numeric/odeint/integrate/integrate_adaptive.hpp> 116
    Header <boost/numeric/odeint/integrate/integrate_const.hpp> 118
    Header <boost/numeric/odeint/integrate/integrate_n_steps.hpp> 119
    Header <boost/numeric/odeint/integrate/integrate_times.hpp> 120
    Header <boost/numeric/odeint/iterator/adaptive_iterator.hpp> 121
    Header <boost/numeric/odeint/iterator/adaptive_time_iterator.hpp> 124
    Header <boost/numeric/odeint/iterator/const_step_iterator.hpp> 127
    Header <boost/numeric/odeint/iterator/const_step_time_iterator.hpp> 130
    Header <boost/numeric/odeint/iterator/n_step_iterator.hpp> 133
    Header <boost/numeric/odeint/iterator/n_step_time_iterator.hpp> 136
    Header <boost/numeric/odeint/iterator/times_iterator.hpp> 139
    Header <boost/numeric/odeint/iterator/times_time_iterator.hpp> 142
    Header <boost/numeric/odeint/stepper/adams_bashforth.hpp> 145
    Header <boost/numeric/odeint/stepper/adams_bashforth_moulton.hpp> 150
    Header <boost/numeric/odeint/stepper/adams_moulton.hpp> 154
    Header <boost/numeric/odeint/stepper/bulirsch_stoer.hpp> 157
    Header <boost/numeric/odeint/stepper/bulirsch_stoer_dense_out.hpp> 162
    Header <boost/numeric/odeint/stepper/controlled_runge_kutta.hpp> 168
    Header <boost/numeric/odeint/stepper/controlled_step_result.hpp> 180
    Header <boost/numeric/odeint/stepper/dense_output_runge_kutta.hpp> 180
    Header <boost/numeric/odeint/stepper/euler.hpp> 188
    Header <boost/numeric/odeint/stepper/explicit_error_generic_rk.hpp> 190
    Header <boost/numeric/odeint/stepper/explicit_generic_rk.hpp> 194
    Header <boost/numeric/odeint/stepper/implicit_euler.hpp> 196
    Header <boost/numeric/odeint/stepper/modified_midpoint.hpp> 198
    Header <boost/numeric/odeint/stepper/rosenbrock4.hpp> 202
    Header <boost/numeric/odeint/stepper/rosenbrock4_controller.hpp> 206
    Header <boost/numeric/odeint/stepper/rosenbrock4_dense_output.hpp> 208
    Header <boost/numeric/odeint/stepper/runge_kutta4.hpp> 211
    Header <boost/numeric/odeint/stepper/runge_kutta4_classic.hpp> 213
    Header <boost/numeric/odeint/stepper/runge_kutta_cash_karp54.hpp> 216
    Header <boost/numeric/odeint/stepper/runge_kutta_cash_karp54_classic.hpp> 218
    Header <boost/numeric/odeint/stepper/runge_kutta_dopri5.hpp> 221
    Header <boost/numeric/odeint/stepper/runge_kutta_fehlberg78.hpp> 225
    Header <boost/numeric/odeint/stepper/stepper_categories.hpp> 228
    Header <boost/numeric/odeint/stepper/symplectic_euler.hpp> 233
    Header <boost/numeric/odeint/stepper/symplectic_rkn_sb3a_m4_mclachlan.hpp> 234
    Header <boost/numeric/odeint/stepper/symplectic_rkn_sb3a_mclachlan.hpp> 236
    Header <boost/numeric/odeint/stepper/velocity_verlet.hpp> 238
Indexes 244

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Getting started

Overview

odeint is a library for solving initial value problems (IVP) of ordinary differential equations. Mathematically, these problems are formulated as follows:

$x'(t) = f(x,t), x(0) = x0$.

x and f can be vectors and the solution is some function $x(t)$ fulfilling both equations above. In the following we will refer to $x'(t)$ also `dxdt` which is also our notation for the derivative in the source code.

Ordinary differential equations occur nearly everywhere in natural sciences. For example, the whole Newtonian mechanics are described by second order differential equations. Be sure, you will find them in every discipline. They also occur if partial differential equations (PDEs) are discretized. Then, a system of coupled ordinary differential occurs, sometimes also referred as lattices ODEs.

Numerical approximations for the solution $x(t)$ are calculated iteratively. The easiest algorithm is the Euler scheme, where starting at $x(0)$ one finds $x(dt) = x(0) + dt\ f(x(0),0)$. Now one can use $x(dt)$ and obtain $x(2dt)$ in a similar way and so on. The Euler method is of order 1, that means the error at each step is ~ dt^2. This is, of course, not very satisfying, which is why the Euler method is rarely used for real life problems and serves just as illustrative example.

The main focus of odeint is to provide numerical methods implemented in a way where the algorithm is completely independent on the data structure used to represent the state x. In doing so, odeint is applicable for a broad variety of situations and it can be used with many other libraries. Besides the usual case where the state is defined as a `std::vector` or a `boost::array`, we provide native support for the following libraries:

- Boost.uBLAS
- Thrust, making odeint naturally running on CUDA devices
- gsl_vector for compatibility with the many numerical function in the GSL
- Boost.Range
- Boost.Fusion (the state type can be a fusion vector)
- Boost.Units
- Intel Math Kernel Library for maximum performance
- VexCL for OpenCL
- Boost.Graph (still experimentally)

In odeint, the following algorithms are implemented:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 1. Stepper Algorithms

Algorithm	**Class**	**Concept**	**System Concept**	**Order**	**Error Estimation**	**Dense Output**	**Internal state**	**Remarks**
Explicit Euler	`euler`	Dense Output Stepper	System	1	No	Yes	No	Very simple, only for demonstrating purpose
Modified Midpoint	`modified_midpoint`	Stepper	System	configurable (2)	No	No	No	Used in Bulirsch-Stoer implementation
Runge-Kutta 4	`runge_kutta4`	Stepper	System	4	No	No	No	The classical Runge-Kutta scheme, good general scheme without error control
Cash-Karp	`runge_kutta_cash_karp54`	Error Stepper	System	5	Yes (4)	No	No	Good general scheme with error estimation, to be used in controlled_error_stepper
Dormand-Prince 5	`runge_kutta_dopri5`	Error Stepper	System	5	Yes (4)	Yes	Yes	Standard method with error control and dense output, to be used in controlled_error_stepper and in dense_output_controlled_explicit_fsal.
Fehlberg 78	`runge_kutta_fehlberg78`	Error Stepper	System	8	Yes (7)	No	No	Good high order method with error estimation, to be used in controlled_error_stepper.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Algorithm	Class	Concept	System Concept	Order	Error Estimation	Dense Output	Internal state	Remarks
Adams Bashforth	`adams_bashforth`	Stepper	System	configurable	No	No	Yes	Multistep method
Adams Moulton	`adams_moulton`	Stepper	System	configurable	No	No	Yes	Multistep method
Adams Bashforth Moulton	`adams_bashforth_moulton`	Stepper	System	configurable	No	No	Yes	Combined multistep method
Controlled Runge-Kutta	`controlled_runge_kutta`	Controlled Stepper	System	depends	Yes	No	depends	Error control for Error Stepper. Requires an Error Stepper from above. Order depends on the given ErrorStepper
Dense Output Runge-Kutta	`dense_output_runge_kutta`	Dense Output Stepper	System	depends	No	Yes	Yes	Dense output for Stepper and Error Stepper from above if they provide dense output functionality (like `euler` and `runge_kutta_dopri5`) Order depends on the given stepper.
Bulirsch-Stoer	`bulirsch_stoer`	Controlled Stepper	System	variable	Yes	No	No	Stepper with step size and order control. Very good if high precision is required.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Algorithm	Class	Concept	System Concept	Order	Error Estimation	Dense Output	Internal state	Remarks
Bulirsch-Stoer Dense Output	`bulirsch_stoer_dense_out`	Dense Output Stepper	System	variable	Yes	Yes	No	Stepper with step size and order control as well as dense output. Very good if high precision and dense output is required.
Implicit Euler	`implicit_euler`	Stepper	Implicit System	1	No	No	No	Basic implicit routine. Requires the Jacobian. Works only with Boost.uBLAS vectors as state types.
Rosenbrock 4	`rosenbrock4`	Error Stepper	Implicit System	4	Yes	Yes	No	Good for stiff systems. Works only with Boost.uBLAS vectors as state types.
Controlled Rosenbrock 4	`rosenbrock4_controller`	Controlled Stepper	Implicit System	4	Yes	Yes	No	Rosenbrock 4 with error control. Works only with Boost.uBLAS vectors as state types.
Dense Output Rosenbrock 4	`rosenbrock4_dense_output`	Dense Output Stepper	Implicit System	4	Yes	Yes	No	Controlled Rosenbrock 4 with dense output. Works only with Boost.uBLAS vectors as state types.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Algorithm	Class	Concept	System Concept	Order	Error Estimation	Dense Output	Internal state	Remarks
Symplectic Euler	`symplectic_euler`	Stepper	Symplectic System Simple Symplectic System	1	No	No	No	Basic symplectic solver for separable Hamiltonian system
Symplectic RKN McLachlan	`symplectic_rkn_sb3a_mclachlan`	Stepper	Symplectic System Simple Symplectic System	4	No	No	No	Symplectic solver for separable Hamiltonian system with 6 stages and order 4.
Symplectic RKN McLachlan	`symplectic_rkn_sb3a_m4_mclachlan`	Stepper	Symplectic System Simple Symplectic System	4	No	No	No	Symplectic solver with 5 stages and order 4, can be used with arbitrary precision types.
Velocity Verlet	`velocity_verlet`	Stepper	Second Order System	1	No	No	Yes	Velocity verlet method suitable for molecular dynamics simulation.

Usage, Compilation, Headers

odeint is a header-only library, no linking against pre-compiled code is required. It can be included by

```
#include <boost/numeric/odeint.hpp>
```

which includes all headers of the library. All functions and classes from odeint live in the namespace

```
using namespace boost::numeric::odeint;
```

It is also possible to include only parts of the library. This is the recommended way since it saves a lot of compilation time.

- `#include <boost/numeric/odeint/stepper/XYZ.hpp>` - the include path for all steppers, XYZ is a placeholder for a stepper.
- `#include <boost/numeric/odeint/algebra/XYZ.hpp>` - all algebras.
- `#include <boost/numeric/odeint/util/XYZ.hpp>` - the utility functions like `is_resizeable`, `same_size`, or `resize`.
- `#include <boost/numeric/odeint/integrate/XYZ.hpp>` - the integrate routines.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- `#include <boost/numeric/odeint/iterator/XYZ.hpp>` - the range and iterator functions.
- `#include <boost/numeric/odeint/external/XYZ.hpp>` - any binders to external libraries.

Short Example

Imaging, you want to numerically integrate a harmonic oscillator with friction. The equations of motion are given by $x'' = -x + \gamma x'$. Odeint only deals with first order ODEs that have no higher derivatives than x' involved. However, any higher order ODE can be transformed to a system of first order ODEs by introducing the new variables $q=x$ and $p=x'$ such that $w=(q,p)$. To apply numerical integration one first has to design the right hand side of the equation $w' = f(w) = (p,-q+\gamma p)$:

```
/* The type of container used to hold the state vector */
typedef std::vector< double > state_type;

const double gam = 0.15;

/* The rhs of x' = f(x) */
void harmonic_oscillator( const state_type &x , state_type &dxdt , const double /* t */ )
{
    dxdt[0] = x[1];
    dxdt[1] = -x[0] - gam*x[1];
}
```

Here we chose `vector<double>` as the state type, but others are also possible, for example `boost::array<double,2>`. odeint is designed in such a way that you can easily use your own state types. Next, the ODE is defined which is in this case a simple function calculating $f(x)$. The parameter signature of this function is crucial: the integration methods will always call them in the form `f(x, dxdt, t)` (there are exceptions for some special routines). So, even if there is no explicit time dependence, one has to define `t` as a function parameter.

Now, we have to define the initial state from which the integration should start:

```
state_type x(2);
x[0] = 1.0; // start at x=1.0, p=0.0
x[1] = 0.0;
```

For the integration itself we'll use the `integrate` function, which is a convenient way to get quick results. It is based on the error-controlled `runge_kutta54_cash_karp` stepper (5th order) and uses adaptive step-size.

```
size_t steps = integrate( harmonic_oscillator ,
        x , 0.0 , 10.0 , 0.1 );
```

The integrate function expects as parameters the rhs of the ode as defined above, the initial state `x`, the start-and end-time of the integration as well as the initial time step=size. Note, that `integrate` uses an adaptive step-size during the integration steps so the time points will not be equally spaced. The integration returns the number of steps that were applied and updates x which is set to the approximate solution of the ODE at the end of integration.

It is also possible to represent the ode system as a class. The rhs must then be implemented as a functor - a class with an overloaded function call operator:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
/* The rhs of x' = f(x) defined as a class */
class harm_osc {

    double m_gam;

public:
    harm_osc( double gam ) : m_gam(gam) { }

    void operator() ( const state_type &x , state_type &dxdt , const double /* t */ )
    {
        dxdt[0] = x[1];
        dxdt[1] = -x[0] - m_gam*x[1];
    }
};
```

which can be used via

```
harm_osc ho(0.15);
steps = integrate( ho ,
        x , 0.0 , 10.0 , 0.1 );
```

In order to observe the solution during the integration steps all you have to do is to provide a reasonable observer. An example is

```
struct push_back_state_and_time
{
    std::vector< state_type >& m_states;
    std::vector< double >& m_times;

    push_back_state_and_time( std::vector< state_type > &states , std::vector< double > &times )
    : m_states( states ) , m_times( times ) { }

    void operator()( const state_type &x , double t )
    {
        m_states.push_back( x );
        m_times.push_back( t );
    }
};
```

which stores the intermediate steps in a container. Note, the argument structure of the ()-operator: odeint calls the observer exactly in this way, providing the current state and time. Now, you only have to pass this container to the integration function:

```
vector<state_type> x_vec;
vector<double> times;

steps = integrate( harmonic_oscillator ,
        x , 0.0 , 10.0 , 0.1 ,
        push_back_state_and_time( x_vec , times ) );

/* output */
for( size_t i=0; i<=steps; i++ )
{
    cout << times[i] << '\t' << x_vec[i][0] << '\t' << x_vec[i][1] << '\n';
}
```

That is all. You can use functional libraries like Boost.Lambda or Boost.Phoenix to ease the creation of observer functions.

The full cpp file for this example can be found here: harmonic_oscillator.cpp

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Tutorial

Harmonic oscillator

Define the ODE

First of all, you have to specify the data type that represents a state x of your system. Mathematically, this usually is an n-dimensional vector with real numbers or complex numbers as scalar objects. For odeint the most natural way is to use `vector< double >` or `vector< complex< double > >` to represent the system state. However, odeint can deal with other container types as well, e.g. `boost::array< double , N >`, as long as it fulfills some requirements defined below.

To integrate a differential equation numerically, one also has to define the rhs of the equation $x' = f(x)$. In odeint you supply this function in terms of an object that implements the ()-operator with a certain parameter structure. Hence, the straightforward way would be to just define a function, e.g:

```
/* The type of container used to hold the state vector */
typedef std::vector< double > state_type;

const double gam = 0.15;

/* The rhs of x' = f(x) */
void harmonic_oscillator( const state_type &x , state_type &dxdt , const double /* t */ )
{
    dxdt[0] = x[1];
    dxdt[1] = -x[0] - gam*x[1];
}
```

The parameters of the function must follow the example above where `x` is the current state, here a two-component vector containing position q and momentum p of the oscillator, `dxdt` is the derivative x' and should be filled by the function with $f(x)$, and `t` is the current time. Note that in this example t is not required to calculate f, however odeint expects the function signature to have exactly three parameters (there are exception, discussed later).

A more sophisticated approach is to implement the system as a class where the rhs function is defined as the ()-operator of the class with the same parameter structure as above:

```
/* The rhs of x' = f(x) defined as a class */
class harm_osc {

    double m_gam;

public:
    harm_osc( double gam ) : m_gam(gam) { }

    void operator() ( const state_type &x , state_type &dxdt , const double /* t */ )
    {
        dxdt[0] = x[1];
        dxdt[1] = -x[0] - m_gam*x[1];
    }
};
```

odeint can deal with instances of such classes instead of pure functions which allows for cleaner code.

Stepper Types

Numerical integration works iteratively, that means you start at a state $x(t)$ and perform a time-step of length dt to obtain the approximate state $x(t+dt)$. There exist many different methods to perform such a time-step each of which has a certain order q. If the order

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

of a method is q than it is accurate up to term $\sim dt^q$ that means the error in x made by such a step is $\sim dt^{q+1}$. odeint provides several steppers of different orders, see Stepper overview.

Some of steppers in the table above are special: Some need the Jacobian of the ODE, others are constructed for special ODE-systems like Hamiltonian systems. We will show typical examples and use-cases in this tutorial and which kind of steppers should be applied.

Integration with Constant Step Size

The basic stepper just performs one time-step and doesn't give you any information about the error that was made (except that you know it is of order $q+1$). Such steppers are used with constant step size that should be chosen small enough to have reasonable small errors. However, you should apply some sort of validity check of your results (like observing conserved quantities) because you have no other control of the error. The following example defines a basic stepper based on the classical Runge-Kutta scheme of 4th order. The declaration of the stepper requires the state type as template parameter. The integration can now be done by using the `integrate_const( Stepper, System, state, start_time, end_time, step_size )` function from odeint:

```
runge_kutta4< state_type > stepper;
integrate_const( stepper , harmonic_oscillator , x , 0.0 , 10.0 , 0.01 );
```

This call integrates the system defined by `harmonic_oscillator` using the RK4 method from $t=0$ to 10 with a step-size $dt=0.01$ and the initial condition given in `x`. The result, $x(t=10)$ is stored in `x` (in-place). Each stepper defines a `do_step` method which can also be used directly. So, you write down the above example as

```
const double dt = 0.01;
for( double t=0.0 ; t<10.0 ; t+= dt )
    stepper.do_step( harmonic_oscillator , x , t , dt );
```

Tip

If you have a C++11 enabled compiler you can easily use lambdas to create the system function :

```
{
runge_kutta4< state_type > stepper;
integrate_const( stepper , []( const state_type &x , state_type &dxdt , double t ) {
        dxdt[0] = x[1]; dxdt[1] = -x[0] - gam*x[1]; }
    , x , 0.0 , 10.0 , 0.01 );
}
```

Integration with Adaptive Step Size

To improve the numerical results and additionally minimize the computational effort, the application of a step size control is advisable. Step size control is realized via stepper algorithms that additionally provide an error estimation of the applied step. odeint provides a number of such **ErrorSteppers** and we will show their usage on the example of `explicit_error_rk54_ck` - a 5th order Runge-Kutta method with 4th order error estimation and coefficients introduced by Cash and Karp.

```
typedef runge_kutta_cash_karp54< state_type > error_stepper_type;
```

Given the error stepper, one still needs an instance that checks the error and adjusts the step size accordingly. In odeint, this is done by **ControlledSteppers**. For the `runge_kutta_cash_karp54` stepper a `controlled_runge_kutta` stepper exists which can be used via

```
typedef controlled_runge_kutta< error_stepper_type > controlled_stepper_type;
controlled_stepper_type controlled_stepper;
integrate_adaptive( controlled_stepper , harmonic_oscillator , x , 0.0 , 10.0 , 0.01 );
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

As above, this integrates the system defined by `harmonic_oscillator`, but now using an adaptive step size method based on the Runge-Kutta Cash-Karp 54 scheme from $t=0$ to 10 with an initial step size of $dt=0.01$ (will be adjusted) and the initial condition given in x. The result, $x(t=10)$, will also be stored in x (in-place).

In the above example an error stepper is nested in a controlled stepper. This is a nice technique; however one drawback is that one always needs to define both steppers. One could also write the instantiation of the controlled stepper into the call of the integrate function but a complete knowledge of the underlying stepper types is still necessary. Another point is, that the error tolerances for the step size control are not easily included into the controlled stepper. Both issues can be solved by using `make_controlled`:

```
integrate_adaptive( make_controlled< error_stepper_type >( 1.0e-10 , 1.0e-6 ) ,
                    harmonic_oscillator , x , 0.0 , 10.0 , 0.01 );
```

`make_controlled` can be used with many of the steppers of odeint. The first parameter is the absolute error tolerance *eps_abs* and the second is the relative error tolerance *eps_rel* which is used during the integration. The template parameter determines from which error stepper a controlled stepper should be instantiated. An alternative syntax of `make_controlled` is

```
integrate_adaptive( make_controlled( 1.0e-10 , 1.0e-6 , error_stepper_type() ) ,
                    harmonic_oscillator , x , 0.0 , 10.0 , 0.01 );
```

For the Runge-Kutta controller the error made during one step is compared with $eps\_abs + eps\_rel * (a_x * |x| + a_{dxdt} * dt * |dxdt|)$. If the error is smaller than this value the current step is accepted, otherwise it is rejected and the step size is decreased. Note, that the step size is also increased if the error gets too small compared to the rhs of the above relation. The full instantiation of the `controlled_runge_kutta` with all parameters is therefore

```
double abs_err = 1.0e-10 , rel_err = 1.0e-6 , a_x = 1.0 , a_dxdt = 1.0;
controlled_stepper_type controlled_stepper(
    default_error_checker< double , range_algebra , default_opera↲
tions >( abs_err , rel_err , a_x , a_dxdt ) );
integrate_adaptive( controlled_stepper , harmonic_oscillator , x , 0.0 , 10.0 , 0.01 );
```

When using `make_controlled` the parameter a_x and a_{dxdt} are used with their standard values of 1.

In the tables below, one can find all steppers which are working with `make_controlled` and `make_dense_output` which is the analog for the dense output steppers.

Table 2. Generation functions make_controlled(abs_error , rel_error , stepper)

Stepper	Result of make_controlled	Remarks
`runge_kutta_cash_karp54`	`controlled_runge_kutta< runge_kutta_cash_karp54 , default_error_checker<...> >`	$a_x=1$, $a_{dxdt}=1$
`runge_kutta_fehlberg78`	`controlled_runge_kutta< runge_kutta_fehlberg78 , default_error_checker<...> >`	$a_x=1$, $a_{dxdt}=1$
`runge_kutta_dopri5`	`controlled_runge_kutta< runge_kutta_dopri5 , default_error_checker<...> >`	$a_x=1$, $a_{dxdt}=1$
`rosenbrock4`	`rosenbrock4_controlled< rosenbrock4 >`	-

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 3. Generation functions make_dense_output(abs_error , rel_error , stepper)

Stepper	Result of make_dense_output	Remarks
`runge_kutta_dopri5`	`dense_output_runge_kutta< controlled_runge_kutta< runge_kutta_dopri5 , default_error_checker<...> > >`	$a_x=1$, $a_{dxdt}=1$
`rosenbrock4`	`rosenbrock4_dense_output< rosenbrock4_controller< rosenbrock4 > >`	-

When using `make_controlled` or `make_dense_output` one should be aware which exact type is used and how the step size control works.

Using iterators

odeint supports iterators for solving ODEs. That is, you instantiate a pair of iterators and instead of using the integrate routines with an appropriate observer you put the iterators in one of the algorithm from the C++ standard library or from Boost.Range. An example is

```
std::for_each( make_const_step_time_iterator_begin( stepper , harmonic_oscillat↲
or, x , 0.0 , 0.1 , 10.0 ) ,
               make_const_step_time_iterator_end( stepper , harmonic_oscillator, x ) ,
               []( std::pair< const state_type & , const double & > x ) {
                   cout << x.second << " " << x.first[0] << " " << x.first[1] << "\n"; } );
```

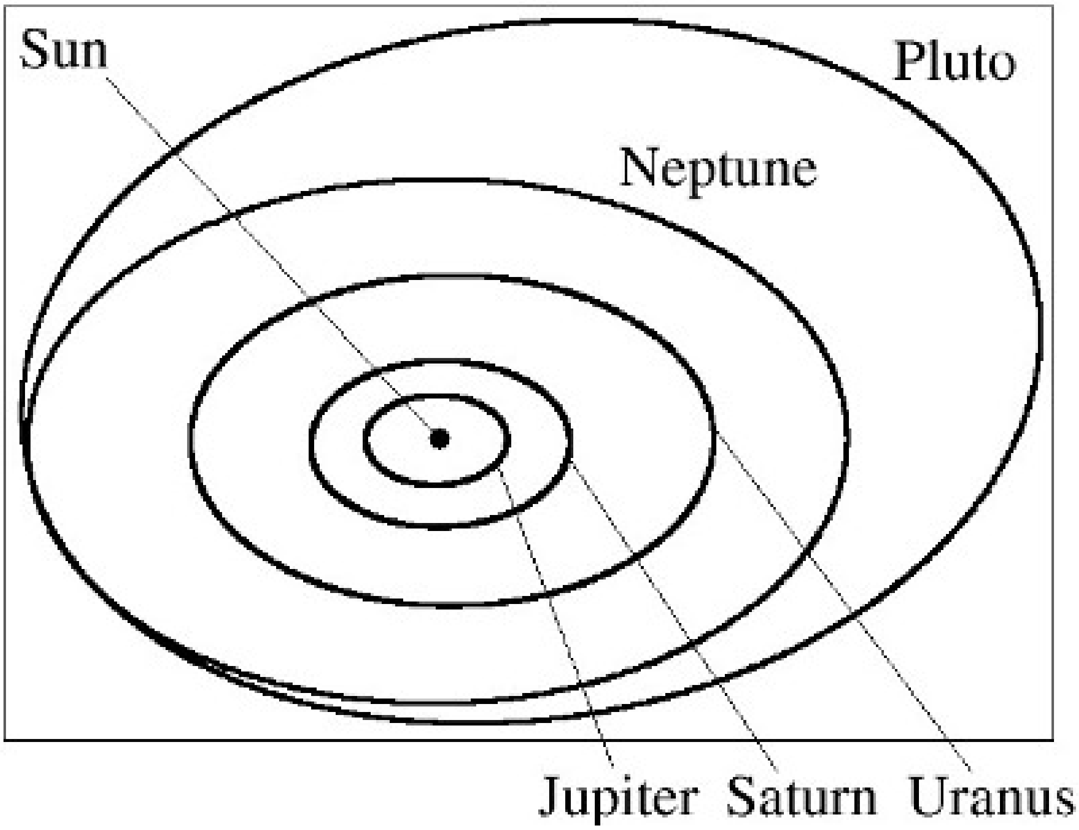
The full source file for this example can be found here: harmonic_oscillator.cpp

Solar system

Gravitation and energy conservation

The next example in this tutorial is a simulation of the outer solar system, consisting of the sun, Jupiter, Saturn, Uranus, Neptune and Pluto.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/



Each planet and of course the sun will be represented by mass points. The interaction force between each object is the gravitational force which can be written as

$F_{ij} = -\gamma m_i m_j (q_i - q_j) / | q_i - q_j |^3$

where γ is the gravitational constant, m_i and m_j are the masses and q_i and q_j are the locations of the two objects. The equations of motion are then

$dq_i / dt = p_i$

$dp_i / dt = 1 / m_i \Sigma_{ji} F_{ij}$

where p_i is the momenta of object i. The equations of motion can also be derived from the Hamiltonian

$H = \Sigma_i p_i^2 / (2 m_i) + \Sigma_j V(q_i , q_j)$

with the interaction potential $V(q_i,q_j)$. The Hamiltonian equations give the equations of motion

$dq_i / dt = dH / dp_i$

$dp_i / dt = -dH / dq_i$

In time independent Hamiltonian system the energy and the phase space volume are conserved and special integration methods have to be applied in order to ensure these conservation laws. The odeint library provides classes for separable Hamiltonian systems, which can be written in the form $H = \Sigma p_i^2 / (2m_i) + H_q(q)$, where $H_q(q)$ only depends on the coordinates. Although this functional form might look a bit arbitrary, it covers nearly all classical mechanical systems with inertia and without dissipation, or where the equations of motion can be written in the form $dq_i / dt = p_i / \mathrm{m_i}$, $dp_i / dt = f(q_i)$.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Note

A short physical note: While the two-body-problem is known to be integrable, that means it can be solved with purely analytic techniques, already the three-body-problem is not solvable. This was found in the end of the 19th century by H. Poincare which led to the whole new subject of Chaos Theory.

Define the system function

To implement this system we define a 3D point type which will represent the space as well as the velocity. Therefore, we use the operators from Boost.Operators:

```
/*the point type */
template< class T , size_t Dim >
class point :
    boost::additive1< point< T , Dim > ,
    boost::additive2< point< T , Dim  > , T ,
    boost::multiplicative2< point< T , Dim > , T
    > > >
    {
    public:

        const static size_t dim = Dim;
        typedef T value_type;
        typedef point< value_type , dim > point_type;

        // ...
        // constructors

        // ...
        // operators

    private:

        T m_val[dim];
    };

    //...
    // more operators
```

The next step is to define a container type storing the values of *q* and *p* and to define system functions. As container type we use `boost::array`

```
// we simulate 5 planets and the sun
const size_t n = 6;

typedef point< double , 3 > point_type;
typedef boost::array< point_type , n > container_type;
typedef boost::array< double , n > mass_type;
```

The `container_type` is different from the state type of the ODE. The state type of the ode is simply a `pair< container_type , container_type >` since it needs the information about the coordinates and the momenta.

Next we define the system's equations. As we will use a stepper that accounts for the Hamiltonian (energy-preserving) character of the system, we have to define the rhs different from the usual case where it is just a single function. The stepper will make use of the separable character, which means the system will be defined by two objects representing *f(p) = -dH/dq* and *g(q) = dH/dp*:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
const double gravitational_constant = 2.95912208286e-4;

struct solar_system_coor
{
    const mass_type &m_masses;

    solar_system_coor( const mass_type &masses ) : m_masses( masses ) { }

    void operator()( const container_type &p , container_type &dqdt ) const
    {
        for( size_t i=0 ; i<n ; ++i )
            dqdt[i] = p[i] / m_masses[i];
    }
};
```

```
struct solar_system_momentum
{
    const mass_type &m_masses;

    solar_system_momentum( const mass_type &masses ) : m_masses( masses ) { }

    void operator()( const container_type &q , container_type &dpdt ) const
    {
        const size_t n = q.size();
        for( size_t i=0 ; i<n ; ++i )
        {
            dpdt[i] = 0.0;
            for( size_t j=0 ; j<i ; ++j )
            {
                point_type diff = q[j] - q[i];
                double d = abs( diff );
                diff *= ( gravitational_constant * m_masses[i] * m_masses[j] / d / d / d );
                dpdt[i] += diff;
                dpdt[j] -= diff;

            }
        }
    }
};
```

In general a three body-system is chaotic, hence we can not expect that arbitrary initial conditions of the system will lead to a solution comparable with the solar system dynamics. That is we have to define proper initial conditions, which are taken from the book of Hairer, Wannier, Lubich [4] .

As mentioned above, we need to use some special integrators in order to conserve phase space volume. There is a well known family of such integrators, the so-called Runge-Kutta-Nystroem solvers, which we apply here in terms of a `symplectic_rkn_sb3a_mclachlan` stepper:

```
typedef symplectic_rkn_sb3a_mclachlan< container_type > stepper_type;
const double dt = 100.0;

integrate_const(
        stepper_type() ,
        make_pair( solar_system_coor( masses ) , solar_system_momentum( masses ) ) ,
        make_pair( boost::ref( q ) , boost::ref( p ) ) ,
        0.0 , 200000.0 , dt , streaming_observer( cout ) );
```

These integration routine was used to produce the above sketch of the solar system. Note, that there are two particularities in this example. First, the state of the symplectic stepper is not `container_type` but a pair of `container_type`. Hence, we must pass such a pair to the integrate function. Since, we want to pass them as references we can simply pack them into Boost.Ref. The second

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

point is the observer, which is called with a state type, hence a pair of `container_type`. The reference wrapper is also passed, but this is not a problem at all:

```
struct streaming_observer
{
    std::ostream& m_out;

    streaming_observer( std::ostream &out ) : m_out( out ) { }

    template< class State >
    void operator()( const State &x , double t ) const
    {
        container_type &q = x.first;
        m_out << t;
        for( size_t i=0 ; i<q.size() ; ++i ) m_out << "\t" << q[i];
        m_out << "\n";
    }
};
```

Tip

You can use C++11 lambda to create the observers

The full example can be found here: solar_system.cpp

Chaotic systems and Lyapunov exponents

In this example we present application of odeint to investigation of the properties of chaotic deterministic systems. In mathematical terms chaotic refers to an exponential growth of perturbations δx. In order to observe this exponential growth one usually solves the equations for the tangential dynamics which is again an ordinary differential equation. These equations are linear but time dependent and can be obtained via

$$d\, \delta x / dt = J(x)\, \delta x$$

where J is the Jacobian of the system under consideration. δx can also be interpreted as a perturbation of the original system. In principle n of these perturbations exist, they form a hypercube and evolve in the time. The Lyapunov exponents are then defined as logarithmic growth rates of the perturbations. If one Lyapunov exponent is larger then zero the nearby trajectories diverge exponentially hence they are chaotic. If the largest Lyapunov exponent is zero one is usually faced with periodic motion. In the case of a largest Lyapunov exponent smaller then zero convergence to a fixed point is expected. More information's about Lyapunov exponents and nonlinear dynamical systems can be found in many textbooks, see for example: E. Ott "Chaos is Dynamical Systems", Cambridge.

To calculate the Lyapunov exponents numerically one usually solves the equations of motion for n perturbations and orthonormalizes them every k steps. The Lyapunov exponent is the average of the logarithm of the stretching factor of each perturbation.

To demonstrate how one can use odeint to determine the Lyapunov exponents we choose the Lorenz system. It is one of the most studied dynamical systems in the nonlinear dynamics community. For the standard parameters it possesses a strange attractor with non-integer dimension. The Lyapunov exponents take values of approximately 0.9, 0 and -12.

The implementation of the Lorenz system is

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
const double sigma = 10.0;
const double R = 28.0;
const double b = 8.0 / 3.0;

typedef boost::array< double , 3 > lorenz_state_type;

void lorenz( const lorenz_state_type &x , lorenz_state_type &dxdt , double t )
{
    dxdt[0] = sigma * ( x[1] - x[0] );
    dxdt[1] = R * x[0] - x[1] - x[0] * x[2];
    dxdt[2] = -b * x[2] + x[0] * x[1];
}
```

We need also to integrate the set of the perturbations. This is done in parallel to the original system, hence within one system function. Of course, we want to use the above definition of the Lorenz system, hence the definition of the system function including the Lorenz system itself and the perturbation could look like:

```
const size_t n = 3;
const size_t num_of_lyap = 3;
const size_t N = n + n*num_of_lyap;

typedef std::tr1::array< double , N > state_type;
typedef std::tr1::array< double , num_of_lyap > lyap_type;

void lorenz_with_lyap( const state_type &x , state_type &dxdt , double t )
{
    lorenz( x , dxdt , t );

    for( size_t l=0 ; l<num_of_lyap ; ++l )
    {
        const double *pert = x.begin() + 3 + l * 3;
        double *dpert = dxdt.begin() + 3 + l * 3;
        dpert[0] = - sigma * pert[0] + 10.0 * pert[1];
        dpert[1] = ( R - x[2] ) * pert[0] - pert[1] - x[0] * pert[2];
        dpert[2] = x[1] * pert[0] + x[0] * pert[1] - b * pert[2];
    }
}
```

The perturbations are stored linearly in the `state_type` behind the state of the Lorenz system. The problem of lorenz() and lorenz_with_lyap() having different state types may be solved putting the Lorenz system inside a functor with templatized arguments:

```
struct lorenz
{
    template< class StateIn , class StateOut , class Value >
    void operator()( const StateIn &x , StateOut &dxdt , Value t )
    {
        dxdt[0] = sigma * ( x[1] - x[0] );
        dxdt[1] = R * x[0] - x[1] - x[0] * x[2];
        dxdt[2] = -b * x[2] + x[0] * x[1];
    }
};

void lorenz_with_lyap( const state_type &x , state_type &dxdt , double t )
{
    lorenz()( x , dxdt , t );
    ...
}
```

This works fine and `lorenz_with_lyap` can be used for example via

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
state_type x;
// initialize x..

explicit_rk4< state_type > rk4;
integrate_n_steps( rk4 , lorenz_with_lyap , x , 0.0 , 0.01 , 1000 );
```

This code snippet performs 1000 steps with constant step size 0.01.

A real world use case for the calculation of the Lyapunov exponents of Lorenz system would always include some transient steps, just to ensure that the current state lies on the attractor, hence it would look like

```
state_type x;
// initialize x
explicit_rk4< state_type > rk4;
integrate_n_steps( rk4 , lorenz , x , 0.0 , 0.01 , 1000 );
```

The problem is now, that x is the full state containing also the perturbations and `integrate_n_steps` does not know that it should only use 3 elements. In detail, odeint and its steppers determine the length of the system under consideration by determining the length of the state. In the classical solvers, e.g. from Numerical Recipes, the problem was solved by pointer to the state and an appropriate length, something similar to

```
void lorenz( double* x , double *dxdt , double t, void* params )
{
    ...
}

int system_length = 3;
rk4( x , system_length , t , dt , lorenz );
```

But odeint supports a similar and much more sophisticated concept: Boost.Range. To make the steppers and the system ready to work with Boost.Range the system has to be changed:

```
struct lorenz
{
    template< class State , class Deriv >
    void operator()( const State &x_ , Deriv &dxdt_ , double t ) const
    {
        typename boost::range_iterator< const State >::type x = boost::begin( x_ );
        typename boost::range_iterator< Deriv >::type dxdt = boost::begin( dxdt_ );

        dxdt[0] = sigma * ( x[1] - x[0] );
        dxdt[1] = R * x[0] - x[1] - x[0] * x[2];
        dxdt[2] = -b * x[2] + x[0] * x[1];
    }
};
```

This is in principle all. Now, we only have to call `integrate_n_steps` with a range including only the first 3 components of *x*:

```
// explicitly choose range_algebra to override default choice of array_algebra
runge_kutta4< state_type , double , state_type , double , range_algebra > rk4;

// perform 10000 transient steps
integrate_n_steps( rk4 , lorenz() , std::make_pair( x.begin() , x.be↵
gin() + n ) , 0.0 , dt , 10000 );
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Note

Note that when using Boost.Range, we have to explicitly configure the stepper to use the `range_algebra` as otherwise odeint would automatically chose the `array_algebra`, which is incompatible with the usage of Boost.Range, because the original state_type is an `array`.

Having integrated a sufficient number of transients steps we are now able to calculate the Lyapunov exponents:

1. Initialize the perturbations. They are stored linearly behind the state of the Lorenz system. The perturbations are initialized such that $p_{ij} = \delta_{ij}$, where p_{ij} is the *j*-component of the *i*.-th perturbation and δ_{ij} is the Kronecker symbol.
2. Integrate 100 steps of the full system with perturbations
3. Orthonormalize the perturbation using Gram-Schmidt orthonormalization algorithm.
4. Repeat step 2 and 3. Every 10000 steps write the current Lyapunov exponent.

```
fill( x.begin()+n , x.end() , 0.0 );
for( size_t i=0 ; i<num_of_lyap ; ++i ) x[n+n*i+i] = 1.0;
fill( lyap.begin() , lyap.end() , 0.0 );

double t = 0.0;
size_t count = 0;
while( true )
{

    t = integrate_n_steps( rk4 , lorenz_with_lyap , x , t , dt , 100 );
    gram_schmidt< num_of_lyap >( x , lyap , n );
    ++count;

    if( !(count % 100000) )
    {
        cout << t;
        for( size_t i=0 ; i<num_of_lyap ; ++i ) cout << "\t" << lyap[i] / t ;
        cout << endl;
    }
}
```

The full code can be found here: chaotic_system.cpp

Stiff systems

An important class of ordinary differential equations are so called stiff system which are characterized by two or more time scales of different order. Examples of such systems are found in chemical systems where reaction rates of individual sub-reaction might differ over large ranges, for example:

$$d S_1 / dt = - 101 S_2 - 100 S_1$$

$$d S_2 / dt = S_1$$

In order to efficiently solve stiff systems numerically the Jacobian

$$J = d f_i / d x_j$$

is needed. Here is the definition of the above example

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef boost::numeric::ublas::vector< double > vector_type;
typedef boost::numeric::ublas::matrix< double > matrix_type;

struct stiff_system
{
    void operator()( const vector_type &x , vector_type &dxdt , double /* t */ )
    {
        dxdt[ 0 ] = -101.0 * x[ 0 ] - 100.0 * x[ 1 ];
        dxdt[ 1 ] = x[ 0 ];
    }
};

struct stiff_system_jacobi
{
    void operator()( const vector_type & /* x */ , matrix_type &J , const double & /* t */ , vec↵
tor_type &dfdt )
    {
        J( 0 , 0 ) = -101.0;
        J( 0 , 1 ) = -100.0;
        J( 1 , 0 ) = 1.0;
        J( 1 , 1 ) = 0.0;
        dfdt[0] = 0.0;
        dfdt[1] = 0.0;
    }
};
```

The state type has to be a `ublas::vector` and the matrix type must by a `ublas::matrix` since the stiff integrator only accepts these types. However, you might want use non-stiff integrators on this system, too - we will do so later for demonstration. Therefore we want to use the same function also with other state_types, realized by templatizing the `operator()`:

```
typedef boost::numeric::ublas::vector< double > vector_type;
typedef boost::numeric::ublas::matrix< double > matrix_type;

struct stiff_system
{
    template< class State >
    void operator()( const State &x , State &dxdt , double t )
    {
        ...
    }
};

struct stiff_system_jacobi
{
    template< class State , class Matrix >
    void operator()( const State &x , Matrix &J , const double &t , State &dfdt )
    {
        ...
    }
};
```

Now you can use `stiff_system` in combination with `std::vector` or `boost::array`. In the example the explicit time derivative of *f(x,t)* is introduced separately in the Jacobian. If *df / dt = 0* simply fill `dfdt` with zeros.

A well know solver for stiff systems is the Rosenbrock method. It has a step size control and dense output facilities and can be used like all the other steppers:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
vector_type x( 2 , 1.0 );

size_t num_of_steps = integrate_const( make_dense_output< rosenbrock4< double > >( 1.0e-6 , 1.0e-
6 ) ,
        make_pair( stiff_system() , stiff_system_jacobi() ) ,
        x , 0.0 , 50.0 , 0.01 ,
        cout << phoenix::arg_names::arg2 << " " << phoenix::arg_names::arg1[0] << "\n" );
```

During the integration 71 steps have been done. Comparing to a classical Runge-Kutta solver this is a very good result. For example the Dormand-Prince 5 method with step size control and dense output yields 1531 steps.

```
vector_type x2( 2 , 1.0 );

size_t num_of_steps2 = integrate_const( make_dense_output< runge_kutta_dopri5< vec↲
tor_type > >( 1.0e-6 , 1.0e-6 ) ,
        stiff_system() , x2 , 0.0 , 50.0 , 0.01 ,
        cout << phoenix::arg_names::arg2 << " " << phoenix::arg_names::arg1[0] << "\n" );
```

Note, that we have used Boost.Phoenix, a great functional programming library, to create and compose the observer.

The full example can be found here: stiff_system.cpp

Complex state types

Thus far we have seen several examples defined for real values. odeint can handle complex state types, hence ODEs which are defined on complex vector spaces, as well. An example is the Stuart-Landau oscillator

$$d \Psi / dt = (1 + i \eta) \Psi + (1 + i \alpha) | \Psi |^2 \Psi$$

where Ψ and i is a complex variable. The definition of this ODE in C++ using complex< double > as a state type may look as follows

```
typedef complex< double > state_type;

struct stuart_landau
{
    double m_eta;
    double m_alpha;

    stuart_landau( double eta = 1.0 , double alpha = 1.0 )
    : m_eta( eta ) , m_alpha( alpha ) { }

    void operator()( const state_type &x , state_type &dxdt , double t ) const
    {
        const complex< double > I( 0.0 , 1.0 );
        dxdt = ( 1.0 + m_eta * I ) * x - ( 1.0 + m_alpha * I ) * norm( x ) * x;
    }
};
```

One can also use a function instead of a functor to implement it

```
double eta = 1.0;
double alpha = 1.0;

void stuart_landau( const state_type &x , state_type &dxdt , double t )
{
    const complex< double > I( 0.0 , 1.0 );
    dxdt = ( 1.0 + m_eta * I ) * x - ( 1.0 + m_alpha * I ) * norm( x ) * x;
}
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

We strongly recommend to use the first ansatz. In this case you have explicit control over the parameters of the system and are not restricted to use global variables to parametrize the oscillator.

When choosing the stepper type one has to account for the "unusual" state type: it is a single `complex<double>` opposed to the vector types used in the previous examples. This means that no iterations over vector elements have to be performed inside the stepper algorithm. Odeint already detects that and automatically uses the `vector_space_algebra` for computation. You can enforce this by supplying additional template arguments to the stepper including the `vector_space_algebra`. Details on the usage of algebras can be found in the section Adapt your own state types.

```
state_type x = complex< double >( 1.0 , 0.0 );

const double dt = 0.1;

typedef runge_kutta4< state_type > stepper_type;

integrate_const( stepper_type() , stuart_landau( 2.0 , 1.0 ) , x , 0.0 , 10.0 , dt , streaming_ob↲
server( cout ) );
```

The full cpp file for the Stuart-Landau example can be found here stuart_landau.cpp

Lattice systems

odeint can also be used to solve ordinary differential equations defined on lattices. A prominent example is the Fermi-Pasta-Ulam system [8] . It is a Hamiltonian system of nonlinear coupled harmonic oscillators. The Hamiltonian is

$$H = \Sigma_i p_i^2/2 + 1/2 (q_{i+1} - q_i)^2 + \beta / 4 (q_{i+1} - q_i)^4$$

Remarkably, the Fermi-Pasta-Ulam system was the first numerical experiment to be implemented on a computer. It was studied at Los Alamos in 1953 on one of the first computers (a MANIAC I) and it triggered a whole new tree of mathematical and physical science.

Like the Solar System, the FPU is solved again by a symplectic solver, but in this case we can speed up the computation because the q components trivially reduce to $dq_i / dt = p_i$. odeint is capable of doing this performance improvement. All you have to do is to call the symplectic solver with an state function for the p components. Here is how this function looks like

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef vector< double > container_type;

struct fpu
{
    const double m_beta;

    fpu( const double beta = 1.0 ) : m_beta( beta ) { }

    // system function defining the ODE
    void operator()( const container_type &q , container_type &dpdt ) const
    {
        size_t n = q.size();
        double tmp = q[0] - 0.0;
        double tmp2 = tmp + m_beta * tmp * tmp * tmp;
        dpdt[0] = -tmp2;
        for( size_t i=0 ; i<n-1 ; ++i )
        {
            tmp = q[i+1] - q[i];
            tmp2 = tmp + m_beta * tmp * tmp * tmp;
            dpdt[i] += tmp2;
            dpdt[i+1] = -tmp2;
        }
        tmp = - q[n-1];
        tmp2 = tmp + m_beta * tmp * tmp * tmp;
        dpdt[n-1] += tmp2;
    }

    // calculates the energy of the system
    double energy( const container_type &q , const container_type &p ) const
    {
        // ...
    }

    // calculates the local energy of the system
    void local_energy( const container_type &q , const container_type &p , contain↵
er_type &e ) const
    {
        // ...
    }
};
```

You can also use `boost::array< double , N >` for the state type.

Now, you have to define your initial values and perform the integration:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
const size_t n = 64;
container_type q( n , 0.0 ) , p( n , 0.0 );

for( size_t i=0 ; i<n ; ++i )
{
    p[i] = 0.0;
    q[i] = 32.0 * sin( double( i + 1 ) / double( n + 1 ) * M_PI );
}


const double dt = 0.1;

typedef symplectic_rkn_sb3a_mclachlan< container_type > stepper_type;
fpu fpu_instance( 8.0 );

integrate_const( stepper_type() , fpu_instance ,
        make_pair( boost::ref( q ) , boost::ref( p ) ) ,
        0.0 , 1000.0 , dt , streaming_observer( cout , fpu_instance , 10 ) );
```

The observer uses a reference to the system object to calculate the local energies:

```
struct streaming_observer
{
    std::ostream& m_out;
    const fpu &m_fpu;
    size_t m_write_every;
    size_t m_count;

    streaming_observer( std::ostream &out , const fpu &f , size_t write_every = 100 )
    : m_out( out ) , m_fpu( f ) , m_write_every( write_every ) , m_count( 0 ) { }

    template< class State >
    void operator()( const State &x , double t )
    {
        if( ( m_count % m_write_every ) == 0 )
        {
            container_type &q = x.first;
            container_type &p = x.second;
            container_type energy( q.size() );
            m_fpu.local_energy( q , p , energy );
            for( size_t i=0 ; i<q.size() ; ++i )
            {
                m_out << t << "\t" << i << "\t" << q[i] << "\t" << p[i] << "\t" << en↵
ergy[i] << "\n";
            }
            m_out << "\n";
            clog << t << "\t" << accumulate( energy.begin() , energy.end() , 0.0 ) << "\n";
        }
        ++m_count;
    }
};
```

The full cpp file for this FPU example can be found here fpu.cpp

Ensembles of oscillators

Another important high dimensional system of coupled ordinary differential equations is an ensemble of N all-to-all coupled phase oscillators [9] . It is defined as

$$d\phi_k / dt = \omega_k + \varepsilon / N \Sigma_j \sin(\phi_j - \phi_k)$$

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

The natural frequencies ω_i of each oscillator follow some distribution and ε is the coupling strength. We choose here a Lorentzian distribution for ω_i. Interestingly a phase transition can be observed if the coupling strength exceeds a critical value. Above this value synchronization sets in and some of the oscillators oscillate with the same frequency despite their different natural frequencies. The transition is also called Kuramoto transition. Its behavior can be analyzed by employing the mean field of the phase

$$Z = K e^{i \Theta} = 1 / N \Sigma_k e^{i \phi_k}$$

The definition of the system function is now a bit more complex since we also need to store the individual frequencies of each oscillator.

```
typedef vector< double > container_type;


pair< double , double > calc_mean_field( const container_type &x )
{
    size_t n = x.size();
    double cos_sum = 0.0 , sin_sum = 0.0;
    for( size_t i=0 ; i<n ; ++i )
    {
        cos_sum += cos( x[i] );
        sin_sum += sin( x[i] );
    }
    cos_sum /= double( n );
    sin_sum /= double( n );

    double K = sqrt( cos_sum * cos_sum + sin_sum * sin_sum );
    double Theta = atan2( sin_sum , cos_sum );

    return make_pair( K , Theta );
}


struct phase_ensemble
{
    container_type m_omega;
    double m_epsilon;

    phase_ensemble( const size_t n , double g = 1.0 , double epsilon = 1.0 )
    : m_omega( n , 0.0 ) , m_epsilon( epsilon )
    {
        create_frequencies( g );
    }

    void create_frequencies( double g )
    {
        boost::mt19937 rng;
        boost::cauchy_distribution<> cauchy( 0.0 , g );
        boost::variate_generator< boost::mt19937&, boost::cauchy_distribu↲
tion<> > gen( rng , cauchy );
        generate( m_omega.begin() , m_omega.end() , gen );
    }

    void set_epsilon( double epsilon ) { m_epsilon = epsilon; }

    double get_epsilon( void ) const { return m_epsilon; }

    void operator()( const container_type &x , container_type &dxdt , double /* t */ ) const
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
    {
        pair< double , double > mean = calc_mean_field( x );
        for( size_t i=0 ; i<x.size() ; ++i )
            dxdt[i] = m_omega[i] + m_epsilon * mean.first * sin( mean.second - x[i] );
    }
};
```

Note, that we have used Z to simplify the equations of motion. Next, we create an observer which computes the value of Z and we record Z for different values of ε.

```
struct statistics_observer
{
    double m_K_mean;
    size_t m_count;

    statistics_observer( void )
    : m_K_mean( 0.0 ) , m_count( 0 ) { }

    template< class State >
    void operator()( const State &x , double t )
    {
        pair< double , double > mean = calc_mean_field( x );
        m_K_mean += mean.first;
        ++m_count;
    }

    double get_K_mean( void ) const { re↲
turn ( m_count != 0 ) ? m_K_mean / double( m_count ) : 0.0 ; }

    void reset( void ) { m_K_mean = 0.0; m_count = 0; }
};
```

Now, we do several integrations for different values of ε and record Z. The result nicely confirms the analytical result of the phase transition, i.e. in our example the standard deviation of the Lorentzian is 1 such that the transition will be observed at $\varepsilon = 2$.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
const size_t n = 16384;
const double dt = 0.1;

container_type x( n );

boost::mt19937 rng;
boost::uniform_real<> unif( 0.0 , 2.0 * M_PI );
boost::variate_generator< boost::mt19937&, boost::uniform_real<> > gen( rng , unif );

// gamma = 1, the phase transition occurs at epsilon = 2
phase_ensemble ensemble( n , 1.0 );
statistics_observer obs;

for( double epsilon = 0.0 ; epsilon < 5.0 ; epsilon += 0.1 )
{
    ensemble.set_epsilon( epsilon );
    obs.reset();

    // start with random initial conditions
    generate( x.begin() , x.end() , gen );

    // calculate some transients steps
    integrate_const( runge_kutta4< container_type >() , boost::ref( en↲
semble ) , x , 0.0 , 10.0 , dt );

    // integrate and compute the statistics
    integrate_const( runge_kutta4< container_type >() , boost::ref( en↲
semble ) , x , 0.0 , 100.0 , dt , boost::ref( obs ) );
    cout << epsilon << "\t" << obs.get_K_mean() << endl;
}
```

The full cpp file for this example can be found here phase_oscillator_ensemble.cpp

Using boost::units

odeint also works well with Boost.Units - a library for compile time unit and dimension analysis. It works by decoding unit information into the types of values. For a one-dimensional unit you can just use the Boost.Unit types as state type, deriv type and time type and hand the `vector_space_algebra` to the stepper definition and everything works just fine:

```
typedef units::quantity< si::time , double > time_type;
typedef units::quantity< si::length , double > length_type;
typedef units::quantity< si::velocity , double > velocity_type;

typedef runge_kutta4< length_type , double , velocity_type , time_type ,
                      vector_space_algebra > stepper_type;
```

If you want to solve more-dimensional problems the individual entries typically have different units. That means that the `state_type` is now possibly heterogeneous, meaning that every entry might have a different type. To solve this problem, compile-time sequences from Boost.Fusion can be used.

To illustrate how odeint works with Boost.Units we use the harmonic oscillator as primary example. We start with defining all quantities

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
#include <boost/numeric/odeint.hpp>
#include <boost/numeric/odeint/algebra/fusion_algebra.hpp>
#include <boost/numeric/odeint/algebra/fusion_algebra_dispatcher.hpp>

#include <boost/units/systems/si/length.hpp>
#include <boost/units/systems/si/time.hpp>
#include <boost/units/systems/si/velocity.hpp>
#include <boost/units/systems/si/acceleration.hpp>
#include <boost/units/systems/si/io.hpp>

#include <boost/fusion/container.hpp>

using namespace std;
using namespace boost::numeric::odeint;
namespace fusion = boost::fusion;
namespace units = boost::units;
namespace si = boost::units::si;

typedef units::quantity< si::time , double > time_type;
typedef units::quantity< si::length , double > length_type;
typedef units::quantity< si::velocity , double > velocity_type;
typedef units::quantity< si::acceleration , double > acceleration_type;
typedef units::quantity< si::frequency , double > frequency_type;

typedef fusion::vector< length_type , velocity_type > state_type;
typedef fusion::vector< velocity_type , acceleration_type > deriv_type;
```

Note, that the `state_type` and the `deriv_type` are now a compile-time fusion sequences. `deriv_type` represents *x'* and is now different from the state type as it has different unit definitions. Next, we define the ordinary differential equation which is completely equivalent to the example in Harmonic Oscillator:

```
struct oscillator
{
    frequency_type m_omega;

    oscillator( const frequency_type &omega = 1.0 * si::hertz ) : m_omega( omega ) { }

    void operator()( const state_type &x , deriv_type &dxdt , time_type t ) const
    {
        fusion::at_c< 0 >( dxdt ) = fusion::at_c< 1 >( x );
        fusion::at_c< 1 >( dxdt ) = - m_omega * m_omega * fusion::at_c< 0 >( x );
    }
};
```

Next, we instantiate an appropriate stepper. We must explicitly parametrize the stepper with the `state_type`, `deriv_type`, `time_type`.

```
typedef runge_kutta_dopri5< state_type , double , deriv_type , time_type > stepper_type;

state_type x( 1.0 * si::meter , 0.0 * si::meter_per_second );

integrate_const( make_dense_output( 1.0e-6 , 1.0e-6 , stepper_type() ) , oscillat↵
or( 2.0 * si::hertz ) ,
                x , 0.0 * si::second , 100.0 * si::second , 0.1 * si::second , streaming_observ↵
er( cout ) );
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Note

When using compile-time sequences, the iteration over vector elements is done by the `fusion_algebra`, which is automatically chosen by odeint. For more on the state types / algebras see chapter Adapt your own state types.

It is quite easy but the compilation time might take very long. Furthermore, the observer is defined a bit different

```
struct streaming_observer
{
    std::ostream& m_out;

    streaming_observer( std::ostream &out ) : m_out( out ) { }

    struct write_element
    {
        std::ostream &m_out;
        write_element( std::ostream &out ) : m_out( out ) { };

        template< class T >
        void operator()( const T &t ) const
        {
            m_out << "\t" << t;
        }
    };

    template< class State , class Time >
    void operator()( const State &x , const Time &t ) const
    {
        m_out << t;
        fusion::for_each( x , write_element( m_out ) );
        m_out << "\n";
    }
};
```

Caution

Using Boost.Units works nicely but compilation can be very time and memory consuming. For example the unit test for the usage of Boost.Units in odeint take up to 4 GB of memory at compilation.

The full cpp file for this example can be found here harmonic_oscillator_units.cpp.

Using matrices as state types

odeint works well with a variety of different state types. It is not restricted to pure vector-wise types, like `vector< double >`, `array< double , N >`, `fusion::vector< double , double >`, etc. but also works with types having a different topology then simple vectors. Here, we show how odeint can be used with matrices as states type, in the next section we will show how can be used to solve ODEs defined on complex networks.

By default, odeint can be used with `ublas::matrix< T >` as state type for matrices. A simple example is a two-dimensional lattice of coupled phase oscillators. Other matrix types like `mtl::dense_matrix` or blitz arrays and matrices can used as well but need some kind of activation in order to work with odeint. This activation is described in following sections,

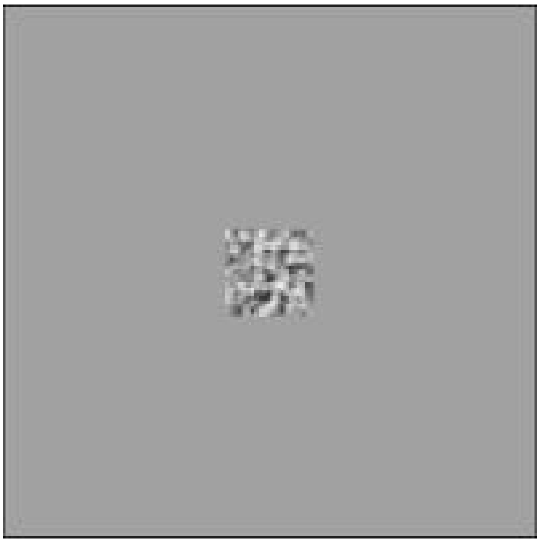
The definition of the system is

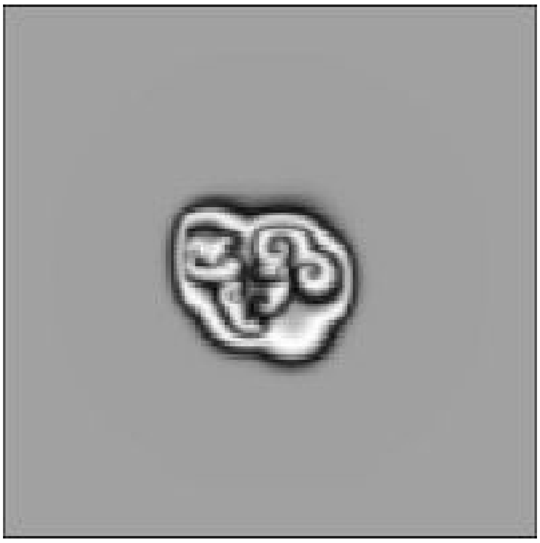
XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

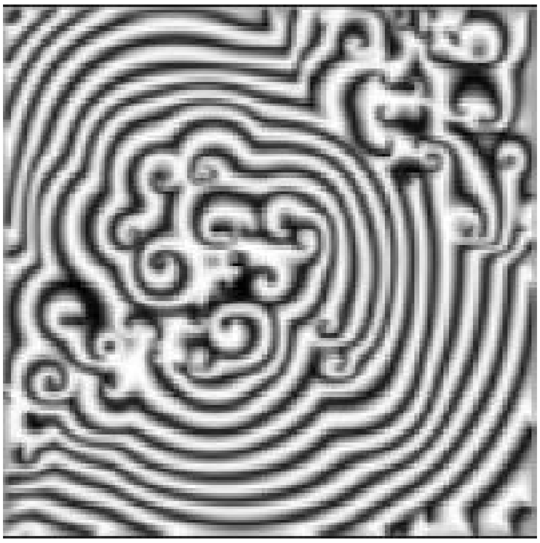
```
typedef boost::numeric::ublas::matrix< double > state_type;

struct two_dimensional_phase_lattice
{
    two_dimensional_phase_lattice( double gamma = 0.5 )
    : m_gamma( gamma ) { }

    void operator()( const state_type &x , state_type &dxdt , double /* t */ ) const
    {
        size_t size1 = x.size1() , size2 = x.size2();

        for( size_t i=1 ; i<size1-1 ; ++i )
        {
            for( size_t j=1 ; j<size2-1 ; ++j )
            {
                dxdt( i , j ) =
                        coupling_func( x( i + 1 , j ) - x( i , j ) ) +
                        coupling_func( x( i - 1 , j ) - x( i , j ) ) +
                        coupling_func( x( i , j + 1 ) - x( i , j ) ) +
                        coupling_func( x( i , j - 1 ) - x( i , j ) );
            }
        }

        for( size_t i=0 ; i<x.size1() ; ++i ) dxdt( i , 0 ) = dxdt( i , x.size2() -1 ) = 0.0;
        for( size_t j=0 ; j<x.size2() ; ++j ) dxdt( 0 , j ) = dxdt( x.size1() -1 , j ) = 0.0;
    }

    double coupling_func( double x ) const
    {
        return sin( x ) - m_gamma * ( 1.0 - cos( x ) );
    }

    double m_gamma;
};
```

In principle this is all. Please note, that the above code is far from being optimal. Better performance can be achieved if every interaction is only calculated once and iterators for columns and rows are used. Below are some visualizations of the evolution of this lattice equation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/







The full cpp for this example can be found here two_dimensional_phase_lattice.cpp.

Using arbitrary precision floating point types

Sometimes one needs results with higher precision than provided by the standard floating point types. As odeint allows to configure the fundamental numerical type, it is well suited to be run with arbitrary precision types. Therefore, one only needs a library that provides a type representing values with arbitrary precision and the fundamental operations for those values. Boost.Multiprecision is a boost library that does exactly this. Making use of Boost.Multiprecision to solve odes with odeint is very simple, as the following example shows.

Here we use `cpp_dec_float_50` as the fundamental value type, which ensures exact computations up to 50 decimal digits.

```
#include <boost/numeric/odeint.hpp>
#include <boost/multiprecision/cpp_dec_float.hpp>

using namespace std;
using namespace boost::numeric::odeint;

typedef boost::multiprecision::cpp_dec_float_50 value_type;

typedef boost::array< value_type , 3 > state_type;
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

As exemplary ODE again the lorenz system is chosen, but here we have to make sure all constants are initialized as high precision values.

```
struct lorenz
{
    void operator()( const state_type &x , state_type &dxdt , value_type t ) const
    {
        const value_type sigma( 10 );
        const value_type R( 28 );
        const value_type b( value_type( 8 ) / value_type( 3 ) );

        dxdt[0] = sigma * ( x[1] - x[0] );
        dxdt[1] = R * x[0] - x[1] - x[0] * x[2];
        dxdt[2] = -b * x[2] + x[0] * x[1];
    }
};
```

The actual integration then is straight forward:

```
state_type x = {{ value_type( 10.0 ) , value_type( 10.0 ) , value_type( 10.0 ) }};

cout.precision( 50 );
integrate_const( runge_kutta4< state_type , value_type >() ,
       ↵
 lorenz() , x , value_type( 0.0 ) , value_type( 10.0 ) , value_type( value_type( 1.0 ) / value_type( 10.0 ) ) ,
        streaming_observer( cout ) );
```

The full example can be found at lorenz_mp.cpp. Another example that compares the accuracy of the high precision type with standard double can be found at cmp_precision.cpp.

Furthermore, odeint can also be run with other multiprecision libraries, e.g. gmp. An example for this is given in lorenz_gmpxx.cpp.

Self expanding lattices

odeint supports changes of the state size during integration if a state_type is used which can be resized, like `std::vector`. The adjustment of the state's size has to be done from outside and the stepper has to be instantiated with `always_resizer` as the template argument for the `resizer_type`. In this configuration, the stepper checks for changes in the state size and adjust it's internal storage accordingly.

We show this for a Hamiltonian system of nonlinear, disordered oscillators with nonlinear nearest neighbor coupling.

The system function is implemented in terms of a class that also provides functions for calculating the energy. Note, that this class stores the random potential internally which is not resized, but rather a start index is kept which should be changed whenever the states' size change.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef vector< double > coord_type;
typedef pair< coord_type , coord_type > state_type;

struct compacton_lattice
{
    const int m_max_N;
    const double m_beta;
    int m_pot_start_index;
    vector< double > m_pot;

    compacton_lattice( int max_N , double beta , int pot_start_index )
       : m_max_N( max_N ) , m_beta( beta ) , m_pot_start_index( pot_start_index ) , m_pot( max_N )
    {
        srand( time( NULL ) );
        // fill random potential with iid values from [0,1]
        boost::mt19937 rng;
        boost::uniform_real<> unif( 0.0 , 1.0 );
        boost::variate_generator< boost::mt19937&, boost::uniform_real<> > gen( rng , unif );
        generate( m_pot.begin() , m_pot.end() , gen );
    }

    void operator()( const coord_type &q , coord_type &dpdt )
    {
        // calculate dpdt = -dH/dq of this hamiltonian system
        // dp_i/dt = - V_i * q_i^3 - beta*(q_i - q_{i-1})^3 + beta*(q_{i+1} - q_i)^3
        const int N = q.size();
        double diff = q[0] - q[N-1];
        for( int i=0 ; i<N ; ++i )
        {
            dpdt[i] = - m_pot[m_pot_start_index+i] * q[i]*q[i]*q[i] -
                    m_beta * diff*diff*diff;
            diff = q[(i+1) % N] - q[i];
            dpdt[i] += m_beta * diff*diff*diff;
        }
    }

   void energy_distribution( const coord_type &q , const coord_type &p , coord_type &energies )
    {
        // computes the energy per lattice site normalized by total energy
        const size_t N = q.size();
        double en = 0.0;
        for( size_t i=0 ; i<N ; i++ )
        {
            const double diff = q[(i+1) % N] - q[i];
            energies[i] = p[i]*p[i]/2.0
                + m_pot[m_pot_start_index+i]*q[i]*q[i]*q[i]*q[i]/4.0
                + m_beta/4.0 * diff*diff*diff*diff;
            en += energies[i];
        }
        en = 1.0/en;
        for( size_t i=0 ; i<N ; i++ )
        {
            energies[i] *= en;
        }
    }

    double energy( const coord_type &q , const coord_type &p )
    {
        // calculates the total energy of the excitation
        const size_t N = q.size();
        double en = 0.0;
        for( size_t i=0 ; i<N ; i++ )
        {
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
            const double diff = q[(i+1) % N] - q[i];
            en += p[i]*p[i]/2.0
                + m_pot[m_pot_start_index+i]*q[i]*q[i]*q[i]*q[i] / 4.0
                + m_beta/4.0 * diff*diff*diff*diff;
        }
        return en;
    }

    void change_pot_start( const int delta )
    {
        m_pot_start_index += delta;
    }
};
```

The total size we allow is 1024 and we start with an initial state size of 60.

```
//start with 60 sites
const int N_start = 60;
coord_type q( N_start , 0.0 );
q.reserve( max_N );
coord_type p( N_start , 0.0 );
p.reserve( max_N );
// start with uniform momentum distribution over 20 sites
fill( p.begin()+20 , p.end()-20 , 1.0/sqrt(20.0) );

coord_type distr( N_start , 0.0 );
distr.reserve( max_N );

// create the system
compacton_lattice lattice( max_N , beta , (max_N-N_start)/2 );

//create the stepper, note that we use an always_resizer because state size might change during ↵
steps
typedef symplectic_rkn_sb3a_mclachlan< coord_type , coord_type , double , coord_type , co↵
ord_type , double ,
        range_algebra , default_operations , always_resizer > hamiltonian_stepper;
hamiltonian_stepper stepper;
hamiltonian_stepper::state_type state = make_pair( q , p );
```

The lattice gets resized whenever the energy distribution comes close to the borders `distr[10] > 1E-150`, `distr[distr.size()-10] > 1E-150`. If we increase to the left, `q` and `p` have to be rotated because their resize function always appends at the end. Additionally, the start index of the potential changes in this case.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
double t = 0.0;
const double dt = 0.1;
const int steps = 10000;
for( int step = 0 ; step < steps ; ++step )
{
    stepper.do_step( boost::ref(lattice) , state , t , dt );
    lattice.energy_distribution( state.first , state.second , distr );
    if( distr[10] > 1E-150 )
    {
        do_resize( state.first , state.second , distr , state.first.size()+20 );
        rotate( state.first.begin() , state.first.end()-20 , state.first.end() );
        rotate( state.second.begin() , state.second.end()-20 , state.second.end() );
        lattice.change_pot_start( -20 );
        cout << t << ": resized left to " << distr.size() << ", energy = " << lattice.en↲
ergy( state.first , state.second ) << endl;
    }
    if( distr[distr.size()-10] > 1E-150 )
    {
        do_resize( state.first , state.second , distr , state.first.size()+20 );
        cout << t << ": resized right to " << distr.size() << ", energy = " << lattice.en↲
ergy( state.first , state.second ) << endl;
    }
    t += dt;
}
```

The `do_resize` function simply calls `vector.resize` of `q` , `p` and `distr`.

```
void do_resize( coord_type &q , coord_type &p , coord_type &distr , const int N )
{
    q.resize( N );
    p.resize( N );
    distr.resize( N );
}
```

The full example can be found in resizing_lattice.cpp

Using CUDA (or OpenMP, TBB, ...) via Thrust

Modern graphic cards (graphic processing units - GPUs) can be used to speed up the performance of time consuming algorithms by means of massive parallelization. They are designed to execute many operations in parallel. odeint can utilize the power of GPUs by means of CUDA and Thrust, which is a STL-like interface for the native CUDA API.

Important

Thrust also supports parallelization using OpenMP and Intel Threading Building Blocks (TBB). You can switch between CUDA, OpenMP and TBB parallelizations by a simple compiler switch. Hence, this also provides an easy way to get basic OpenMP parallelization into odeint. The examples discussed below are focused on GPU parallelization, though.

To use odeint with CUDA a few points have to be taken into account. First of all, the problem has to be well chosen. It makes absolutely no sense to try to parallelize the code for a three dimensional system, it is simply too small and not worth the effort. One single function call (kernel execution) on the GPU is slow but you can do the operation on a huge set of data with only one call. We have experienced that the vector size over which is parallelized should be of the order of 10^6 to make full use of the GPU. Secondly, you have to use Thrust's algorithms and functors when implementing the rhs the ODE. This might be tricky since it involves some kind of functional programming knowledge.

Typical applications for CUDA and odeint are large systems, like lattices or discretizations of PDE, and parameter studies. We introduce now three examples which show how the power of GPUs can be used in combination with odeint.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Important

The full power of CUDA is only available for really large systems where the number of coupled ordinary differential equations is of order $N=10^6$ or larger. For smaller systems the CPU is usually much faster. You can also integrate an ensemble of different uncoupled ODEs in parallel as shown in the last example.

Phase oscillator ensemble

The first example is the phase oscillator ensemble from the previous section:

$$d\phi_k / dt = \omega_k + \varepsilon / N \Sigma_j \sin(\phi_j - \phi_k).$$

It has a phase transition at $\varepsilon = 2$ in the limit of infinite numbers of oscillators N. In the case of finite N this transition is smeared out but still clearly visible.

Thrust and CUDA are perfectly suited for such kinds of problems where one needs a large number of particles (oscillators). We start by defining the state type which is a `thrust::device_vector`. The content of this vector lives on the GPU. If you are not familiar with this we recommend reading the *Getting started* section on the Thrust website.

```
//change this to float if your device does not support double computation
typedef double value_type;

//change this to host_vector< ... > of you want to run on CPU
typedef thrust::device_vector< value_type > state_type;
// typedef thrust::host_vector< value_type > state_type;
```

Thrust follows a functional programming approach. If you want to perform a calculation on the GPU you usually have to call a global function like `thrust::for_each`, `thrust::reduce`, ... with an appropriate local functor which performs the basic operation. An example is

```
struct add_two
{
    template< class T >
    __host__ __device__
    void operator()( T &t ) const
    {
        t += T( 2 );
    }
};

// ...

thrust::for_each( x.begin() , x.end() , add_two() );
```

This code generically adds two to every element in the container `x`.

For the purpose of integrating the phase oscillator ensemble we need

- to calculate the system function, hence the r.h.s. of the ODE.
- this involves computing the mean field of the oscillator example, i.e. the values of R and θ

The mean field is calculated in a class `mean_field_calculator`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
struct mean_field_calculator
{
    struct sin_functor : public thrust::unary_function< value_type , value_type >
    {
        __host__ __device__
        value_type operator()( value_type x) const
        {
            return sin( x );
        }
    };

    struct cos_functor : public thrust::unary_function< value_type , value_type >
    {
        __host__ __device__
        value_type operator()( value_type x) const
        {
            return cos( x );
        }
    };

    static std::pair< value_type , value_type > get_mean( const state_type &x )
    {
        value_type sin_sum = thrust::reduce(
                thrust::make_transform_iterator( x.begin() , sin_functor() ) ,
                thrust::make_transform_iterator( x.end() , sin_functor() ) );
        value_type cos_sum = thrust::reduce(
                thrust::make_transform_iterator( x.begin() , cos_functor() ) ,
                thrust::make_transform_iterator( x.end() , cos_functor() ) );

        cos_sum /= value_type( x.size() );
        sin_sum /= value_type( x.size() );

        value_type K = sqrt( cos_sum * cos_sum + sin_sum * sin_sum );
        value_type Theta = atan2( sin_sum , cos_sum );

        return std::make_pair( K , Theta );
    }
};
```

Inside this class two member structures `sin_functor` and `cos_functor` are defined. They compute the sine and the cosine of a value and they are used within a transform iterator to calculate the sum of $sin(\phi_k)$ and $cos(\phi_k)$. The classifiers `__host__` and `__device__` are CUDA specific and define a function or operator which can be executed on the GPU as well as on the CPU. The line

```
value_type sin_sum = thrust::reduce(
        thrust::make_transform_iterator( x.begin() , sin_functor() ) ,
        thrust::make_transform_iterator( x.end() , sin_functor() ) );
```

performs the calculation of this sine-sum on the GPU (or on the CPU, depending on your thrust configuration).

The system function is defined via

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
class phase_oscillator_ensemble
{

public:

    struct sys_functor
    {
        value_type m_K , m_Theta , m_epsilon;

        sys_functor( value_type K , value_type Theta , value_type epsilon )
        : m_K( K ) , m_Theta( Theta ) , m_epsilon( epsilon ) { }

        template< class Tuple >
        __host__ __device__
        void operator()( Tuple t )
        {
            thrust::get<2>(t) = thrust::get<1>(t) + m_epsi↲
lon * m_K * sin( m_Theta - thrust::get<0>(t) );
        }
    };

    // ...

    void operator() ( const state_type &x , state_type &dxdt , const value_type dt ) const
    {
        std::pair< value_type , value_type > mean_field = mean_field_calculator::get_mean( x );

        thrust::for_each(
                thrust::make_zip_iterator( thrust::make_tuple( x.begin() , m_omega.be↲
gin() , dxdt.begin() ) ),
                thrust::make_zip_iterat↲
or( thrust::make_tuple( x.end() , m_omega.end() , dxdt.end()) ) ,
                sys_functor( mean_field.first , mean_field.second , m_epsilon )
                );
    }

    // ...
};
```

This class is used within the `do_step` and `integrate` method. It defines a member structure `sys_functor` for the r.h.s. of each individual oscillator and the `operator()` for the use in the steppers and integrators of odeint. The functor computes first the mean field of ϕ_k and secondly calculates the whole r.h.s. of the ODE using this mean field. Note, how nicely `thrust::tuple` and `thrust::zip_iterator` play together.

Now we are ready to put everything together. All we have to do for making odeint ready for using the GPU is to parametrize the stepper with the `state_type` and `value_type`:

```
typedef runge_kutta4< state_type , value_type , state_type , value_type > stepper_type;
```

Note

We have specifically define four template parameters because we have to override the default parameter value `double` with `value_type` to ensure our programs runs properly if we use `float` as fundamental data type.

You can also use a controlled or dense output stepper, e.g.

```
typedef runge_kutta_dopri5< state_type , value_type , state_type , value_type > stepper_type;
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Then, it is straightforward to integrate the phase ensemble by creating an instance of the rhs class and using an integration function:

```
phase_oscillator_ensemble ensemble( N , 1.0 );
```

```
size_t steps1 = integrate_const( make_controlled( 1.0e-6 , 1.0e-6 , step↲
per_type() ) , boost::ref( ensemble ) , x , 0.0 , t_transients , dt );
```

We have to use `boost::ref` here in order to pass the rhs class as reference and not by value. This ensures that the natural frequencies of each oscillator are not copied when calling `integrate_const`. In the full example the performance and results of the Runge-Kutta-4 and the Dopri5 solver are compared.

The full example can be found at phase_oscillator_example.cu.

Large oscillator chains

The next example is a large, one-dimensional chain of nearest-neighbor coupled phase oscillators with the following equations of motion:

$$d \phi_k / dt = \omega_k + sin(\phi_{k+1} - \phi_k) + sin(\phi_k - \phi_{k-1})$$

In principle we can use all the techniques from the previous phase oscillator ensemble example, but we have to take special care about the coupling of the oscillators. To efficiently implement the coupling you can use a very elegant way employing Thrust's permutation iterator. A permutation iterator behaves like a normal iterator on a vector but it does not iterate along the usual order of the elements. It rather iterates along some permutation of the elements defined by some index map. To realize the nearest neighbor coupling we create one permutation iterator which travels one step behind a usual iterator and another permutation iterator which travels one step in front. The full system class is:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
//change this to host_vector< ... > if you want to run on CPU
typedef thrust::device_vector< value_type > state_type;
typedef thrust::device_vector< size_t > index_vector_type;
//typedef thrust::host_vector< value_type > state_type;
//typedef thrust::host_vector< size_t > index_vector_type;

class phase_oscillators
{

public:

    struct sys_functor
    {
        template< class Tuple >
        __host__ __device__
        void operator()( Tuple t )  // this functor works on tuples of values
        {
            // first, unpack the tuple into value, neighbors and omega
            const value_type phi = thrust::get<0>(t);
            const value_type phi_left = thrust::get<1>(t);  // left neighbor
            const value_type phi_right = thrust::get<2>(t); // right neighbor
            const value_type omega = thrust::get<3>(t);
            // the dynamical equation
            thrust::get<4>(t) = omega + sin( phi_right - phi ) + sin( phi - phi_left );
        }
    };

    phase_oscillators( const state_type &omega )
       : m_omega( omega ) , m_N( omega.size() ) , m_prev( omega.size() ) , m_next( omega.size() )
    {
        // build indices pointing to left and right neighbours
        thrust::counting_iterator<size_t> c( 0 );
        thrust::copy( c , c+m_N-1 , m_prev.begin()+1 );
        m_prev[0] = 0; // m_prev = { 0 , 0 , 1 , 2 , 3 , ... , N-1 }

        thrust::copy( c+1 , c+m_N , m_next.begin() );
        m_next[m_N-1] = m_N-1; // m_next = { 1 , 2 , 3 , ... , N-1 , N-1 }
    }

    void operator() ( const state_type &x , state_type &dxdt , const value_type dt )
    {
        thrust::for_each(
                thrust::make_zip_iterator(
                        thrust::make_tuple(
                                x.begin() ,
                               thrust::make_permutation_iterator( x.begin() , m_prev.begin() ) ,
                               thrust::make_permutation_iterator( x.begin() , m_next.begin() ) ,
                                m_omega.begin() ,
                                dxdt.begin()
                                ) ),
                thrust::make_zip_iterator(
                        thrust::make_tuple(
                                x.end() ,
                                thrust::make_permutation_iterator( x.begin() , m_prev.end() ) ,
                                thrust::make_permutation_iterator( x.begin() , m_next.end() ) ,
                                m_omega.end() ,
                                dxdt.end()) ) ,
                sys_functor()
                );
    }

private:
    const state_type &m_omega;
    const size_t m_N;
    index_vector_type m_prev;
    index_vector_type m_next;
};
```

Note, how easy you can obtain the value for the left and right neighboring oscillator in the system functor using the permutation iterators. But, the call of the `thrust::for_each` function looks relatively complicated. Every term of the r.h.s. of the ODE is resembled by one iterator packed in exactly the same way as it is unpacked in the functor above.

Now we put everything together. We create random initial conditions and decreasing frequencies such that we should get synchronization. We copy the frequencies and the initial conditions onto the device and finally initialize and perform the integration. As result we simply write out the current state, hence the phase of each oscillator.

```
// create initial conditions and omegas on host:
vector< value_type > x_host( N );
vector< value_type > omega_host( N );
for( size_t i=0 ; i<N ; ++i )
{
    x_host[i] = 2.0 * pi * drand48();
    omega_host[i] = ( N - i ) * epsilon; // decreasing frequencies
}

// copy to device
state_type x = x_host;
state_type omega = omega_host;

// create stepper
runge_kutta4< state_type , value_type , state_type , value_type > stepper;

// create phase oscillator system function
phase_oscillators sys( omega );

// integrate
integrate_const( stepper , sys , x , 0.0 , 10.0 , dt );

thrust::copy( x.begin() , x.end() , std::ostream_iterator< value_type >( std::cout , "\n" ) );
std::cout << std::endl;
```

The full example can be found at phase_oscillator_chain.cu.

Parameter studies

Another important use case for Thrust and CUDA are parameter studies of relatively small systems. Consider for example the three-dimensional Lorenz system from the chaotic systems example in the previous section which has three parameters. If you want to study the behavior of this system for different parameters you usually have to integrate the system for many parameter values. Using thrust and odeint you can do this integration in parallel, hence you integrate a whole ensemble of Lorenz systems where each individual realization has a different parameter value.

In the following we will show how you can use Thrust to integrate the above mentioned ensemble of Lorenz systems. We will vary only the parameter β but it is straightforward to vary other parameters or even two or all three parameters. Furthermore, we will use the largest Lyapunov exponent to quantify the behavior of the system (chaoticity).

We start by defining the range of the parameters we want to study. The state_type is again a `thrust::device_vector< value_type >`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
vector< value_type > beta_host( N );
const value_type beta_min = 0.0 , beta_max = 56.0;
for( size_t i=0 ; i<N ; ++i )
    beta_host[i] = beta_min + value_type( i ) * ( beta_max - beta_min ) / value_type( N - 1 );

state_type beta = beta_host;
```

The next thing we have to implement is the Lorenz system without perturbations. Later, a system with perturbations is also implemented in order to calculate the Lyapunov exponent. We will use an ansatz where each device function calculates one particular realization of the Lorenz ensemble

```
struct lorenz_system
{
    struct lorenz_functor
    {
        template< class T >
        __host__ __device__
        void operator()( T t ) const
        {
            // unpack the parameter we want to vary and the Lorenz variables
            value_type R = thrust::get< 3 >( t );
            value_type x = thrust::get< 0 >( t );
            value_type y = thrust::get< 1 >( t );
            value_type z = thrust::get< 2 >( t );
            thrust::get< 4 >( t ) = sigma * ( y - x );
            thrust::get< 5 >( t ) = R * x - y - x * z;
            thrust::get< 6 >( t ) = -b * z + x * y ;

        }
    };

    lorenz_system( size_t N , const state_type &beta )
    : m_N( N ) , m_beta( beta ) { }

    template< class State , class Deriv >
    void operator()(  const State &x , Deriv &dxdt , value_type t ) const
    {
        thrust::for_each(
                thrust::make_zip_iterator( thrust::make_tuple(
                        boost::begin( x ) ,
                        boost::begin( x ) + m_N ,
                        boost::begin( x ) + 2 * m_N ,
                        m_beta.begin() ,
                        boost::begin( dxdt ) ,
                        boost::begin( dxdt ) + m_N ,
                        boost::begin( dxdt ) + 2 * m_N   ) ) ,
                thrust::make_zip_iterator( thrust::make_tuple(
                        boost::begin( x ) + m_N ,
                        boost::begin( x ) + 2 * m_N ,
                        boost::begin( x ) + 3 * m_N ,
                        m_beta.begin() ,
                        boost::begin( dxdt ) + m_N ,
                        boost::begin( dxdt ) + 2 * m_N ,
                        boost::begin( dxdt ) + 3 * m_N   ) ) ,
                lorenz_functor() );
    }
    size_t m_N;
    const state_type &m_beta;
};
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

As `state_type` a `thrust::device_vector` or a Boost.Range of a `device_vector` is used. The length of the state is *3N* where *N* is the number of systems. The system is encoded into this vector such that all *x* components come first, then every *y* components and finally every *z* components. Implementing the device function is then a simple task, you only have to decompose the tuple originating from the zip iterators.

Besides the system without perturbations we furthermore need to calculate the system including linearized equations governing the time evolution of small perturbations. Using the method from above this is straightforward, with a small difficulty that Thrust's tuples have a maximal arity of 10. But this is only a small problem since we can create a zip iterator packed with zip iterators. So the top level zip iterator contains one zip iterator for the state, one normal iterator for the parameter, and one zip iterator for the derivative. Accessing the elements of this tuple in the system function is then straightforward, you unpack the tuple with `thrust::get<>()`. We will not show the code here, it is to large. It can be found here and is easy to understand.

Furthermore, we need an observer which determines the norm of the perturbations, normalizes them and averages the logarithm of the norm. The device functor which is used within this observer is defined

```
struct lyap_functor
{
    template< class T >
    __host__ __device__
    void operator()( T t ) const
    {
        value_type &dx = thrust::get< 0 >( t );
        value_type &dy = thrust::get< 1 >( t );
        value_type &dz = thrust::get< 2 >( t );
        value_type norm = sqrt( dx * dx + dy * dy + dz * dz );
        dx /= norm;
        dy /= norm;
        dz /= norm;
        thrust::get< 3 >( t ) += log( norm );
    }
};
```

Note, that this functor manipulates the state, i.e. the perturbations.

Now we complete the whole code to calculate the Lyapunov exponents. First, we have to define a state vector. This vector contains *6N* entries, the state *x,y,z* and its perturbations *dx,dy,dz*. We initialize them such that *x=y=z=10*, *dx=1*, and *dy=dz=0*. We define a stepper type, a controlled Runge-Kutta Dormand-Prince 5 stepper. We start with some integration to overcome the transient behavior. For this, we do not involve the perturbation and run the algorithm only on the state *x,y,z* without any observer. Note, how Boost.Range is used for partial integration of the state vector without perturbations (the first half of the whole state). After the transient, the full system with perturbations is integrated and the Lyapunov exponents are calculated and written to `stdout`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
state_type x( 6 * N );

// initialize x,y,z
thrust::fill( x.begin() , x.begin() + 3 * N , 10.0 );

// initial dx
thrust::fill( x.begin() + 3 * N , x.begin() + 4 * N , 1.0 );

// initialize dy,dz
thrust::fill( x.begin() + 4 * N , x.end() , 0.0 );


// create error stepper, can be used with make_controlled or make_dense_output
typedef runge_kutta_dopri5< state_type , value_type , state_type , value_type > stepper_type;


lorenz_system lorenz( N , beta );
lorenz_perturbation_system lorenz_perturbation( N , beta );
lyap_observer obs( N , 1 );

// calculate transients
integrate_adaptive( make_controlled( 1.0e-6 , 1.0e-6 , step↵
per_type() ) , lorenz , std::make_pair( x.begin() , x.begin() + 3 * N ) , 0.0 , 10.0 , dt );

// calculate the Lyapunov exponents -- the main loop
double t = 0.0;
while( t < 10000.0 )
{
    integrate_adaptive( make_controlled( 1.0e-6 , 1.0e-6 , stepper_type() ) , lorenz_perturba↵
tion , x , t , t + 1.0 , 0.1 );
    t += 1.0;
    obs( x , t );
}

vector< value_type > lyap( N );
obs.fill_lyap( lyap );

for( size_t i=0 ; i<N ; ++i )
    cout << beta_host[i] << "\t" << lyap[i] << "\n";
```

The full example can be found at lorenz_parameters.cu.

Using OpenCL via VexCL

In the previous section the usage of odeint in combination with Thrust was shown. In this section we show how one can use OpenCL with odeint. The point of odeint is not to implement its own low-level data structures and algorithms, but to use high level libraries doing this task. Here, we will use the VexCL framework to use OpenCL. VexCL is a nice library for general computations and it uses heavily expression templates. With the help of VexCL it is possible to write very compact and expressive application.

Note

vexcl needs C++11 features! So you have to compile with C++11 support enabled.

To use VexCL one needs to include one additional header which includes the data-types and algorithms from vexcl and the adaption to odeint. Adaption to odeint means here only to adapt the resizing functionality of VexCL to odeint.

```
#include <boost/numeric/odeint/external/vexcl/vexcl.hpp>
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

To demonstrate the use of VexCL we integrate an ensemble of Lorenz system. The example is very similar to the parameter study of the Lorenz system in the previous section except that we do not compute the Lyapunov exponents. Again, we vary the parameter R of the Lorenz system an solve a whole ensemble of Lorenz systems in parallel (each with a different parameter R). First, we define the state type and a vector type

```
typedef vex::vector< double >    vector_type;
typedef vex::multivector< double, 3 > state_type;
```

The `vector_type` is used to represent the parameter R. The `state_type` is a multi-vector of three sub vectors and is used to represent. The first component of this multi-vector represent all `x` components of the Lorenz system, while the second all `y` components and the third all `z` components. The components of this vector can be obtained via

```
auto &x = X(0);
auto &y = X(1);
auto &z = X(2);
```

As already mentioned VexCL supports expression templates and we will use them to implement the system function for the Lorenz ensemble:

```
const double sigma = 10.0;
const double b = 8.0 / 3.0;

struct sys_func
{
    const vector_type &R;

    sys_func( const vector_type &_R ) : R( _R ) { }

    void operator()( const state_type &x , state_type &dxdt , double t ) const
    {
        dxdt(0) = -sigma * ( x(0) - x(1) );
        dxdt(1) = R * x(0) - x(1) - x(0) * x(2);
        dxdt(2) = - b * x(2) + x(0) * x(1);
    }
};
```

It's very easy, isn't it? These three little lines do all the computations for you. There is no need to write your own OpenCL kernels. VexCL does everything for you. Next we have to write the main application. We initialize the vector of parameters (R) and the initial state. Note that VexCL requires the `vector_space_algebra`, but that is automatically deduced and configured by odeint internally, so we only have to specify the `state_type` when instantiating the stepper and we are done:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
// setup the opencl context
vex::Context ctx( vex::Filter::Type(CL_DEVICE_TYPE_GPU) );
std::cout << ctx << std::endl;

// set up number of system, time step and integration time
const size_t n = 1024 * 1024;
const double dt = 0.01;
const double t_max = 1000.0;

// initialize R
double Rmin = 0.1 , Rmax = 50.0 , dR = ( Rmax - Rmin ) / double( n - 1 );
std::vector<double> x( n * 3 ) , r( n );
for( size_t i=0 ; i<n ; ++i ) r[i] = Rmin + dR * double( i );
vector_type R( ctx.queue() , r );

// initialize the state of the lorenz ensemble
state_type X(ctx.queue(), n);
X(0) = 10.0;
X(1) = 10.0;
X(2) = 10.0;

// create a stepper
runge_kutta4< state_type > stepper;

// solve the system
integrate_const( stepper , sys_func( R ) , X , 0.0 , t_max , dt );
```

Parallel computation with OpenMP and MPI

Parallelization is a key feature for modern numerical libraries due to the vast availability of many cores nowadays, even on Laptops. odeint currently supports parallelization with OpenMP and MPI, as described in the following sections. However, it should be made clear from the beginning that the difficulty of efficiently distributing ODE integration on many cores/machines lies in the parallelization of the system function, which is still the user's responsibility. Simply using a parallel odeint backend without parallelizing the system function will bring you almost no performance gains.

OpenMP

odeint's OpenMP support is implemented as an external backend, which needs to be manually included. Depending on the compiler some additional flags may be needed, i.e. `-fopenmp` for GCC.

```
#include <omp.h>
#include <boost/numeric/odeint.hpp>
#include <boost/numeric/odeint/external/openmp/openmp.hpp>
```

In the easiest parallelization approach with OpenMP we use a standard `vector` as the state type:

```
typedef std::vector< double > state_type;
```

We initialize the state with some random data:

```
size_t N = 131101;
state_type x( N );
boost::random::uniform_real_distribution<double> distribution( 0.0 , 2.0*pi );
boost::random::mt19937 engine( 0 );
generate( x.begin() , x.end() , boost::bind( distribution , engine ) );
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Now we have to configure the stepper to use the OpenMP backend. This is done by explicitly providing the `openmp_range_algebra` as a template parameter to the stepper. This algebra requires the state type to be a model of Random Access Range and will be used from multiple threads by the algebra.

```
typedef runge_kutta4<
                  state_type , double ,
                  state_type , double ,
                  openmp_range_algebra
                > stepper_type;
```

Additional to providing the stepper with OpenMP parallelization we also need a parallelized system function to exploit the available cores. Here this is shown for a simple one-dimensional chain of phase oscillators with nearest neighbor coupling:

```
struct phase_chain
{
    phase_chain( double gamma = 0.5 )
    : m_gamma( gamma ) { }

    void operator()( const state_type &x , state_type &dxdt , double /* t */ ) const
    {
        const size_t N = x.size();
        #pragma omp parallel for schedule(runtime)
        for(size_t i = 1 ; i < N - 1 ; ++i)
        {
            dxdt[i] = coupling_func( x[i+1] - x[i] ) +
                      coupling_func( x[i-1] - x[i] );
        }
        dxdt[0  ] = coupling_func( x[1  ] - x[0  ] );
        dxdt[N-1] = coupling_func( x[N-2] - x[N-1] );
    }

    double coupling_func( double x ) const
    {
        return sin( x ) - m_gamma * ( 1.0 - cos( x ) );
    }

    double m_gamma;
};
```

Note

In the OpenMP backends the system function will always be called sequentially from the thread used to start the integration.

Finally, we perform the integration by using one of the integrate functions from odeint. As you can see, the parallelization is completely hidden in the stepper and the system function. OpenMP will take care of distributing the work among the threads and join them automatically.

```
integrate_n_steps( stepper_type() , phase_chain( 1.2 ) ,
                   x , 0.0 , 0.01 , 100 );
```

After integrating, the data can be accessed immediately and be processed further. Note, that you can specify the OpenMP scheduling by calling `omp_set_schedule` in the beginning of your program:

```
int chunk_size = N/omp_get_max_threads();
omp_set_schedule( omp_sched_static , chunk_size );
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

See openmp/phase_chain.cpp for the complete example.

Split state

For advanced cases odeint offers another approach to use OpenMP that allows for a more exact control of the parallelization. For example, for odd-sized data where OpenMP's thread boundaries don't match cache lines and hurt performance it might be advisable to copy the data from the continuous `vector<T>` into separate, individually aligned, vectors. For this, odeint provides the `openmp_state<T>` type, essentially an alias for `vector<vector<T>>`.

Here, the initialization is done with a `vector<double>`, but then we use odeint's `split` function to fill an `openmp_state`. The splitting is done such that the sizes of the individual regions differ at most by 1 to make the computation as uniform as possible.

```
const size_t N = 131101;
vector<double> x( N );
boost::random::uniform_real_distribution<double> distribution( 0.0 , 2.0*pi );
boost::random::mt19937 engine( 0 );
generate( x.begin() , x.end() , boost::bind( distribution , engine ) );
const size_t blocks = omp_get_max_threads();
state_type x_split( blocks );
split( x , x_split );
```

Of course, the system function has to be changed to deal with the `openmp_state`. Note that each sub-region of the state is computed in a single task, but at the borders read access to the neighbouring regions is required.

```
struct phase_chain_omp_state
{
    phase_chain_omp_state( double gamma = 0.5 )
    : m_gamma( gamma ) { }

    void operator()( const state_type &x , state_type &dxdt , double /* t */ ) const
    {
        const size_t N = x.size();
        #pragma omp parallel for schedule(runtime)
        for(size_t n = 0 ; n < N ; ++n)
        {
            const size_t M = x[n].size();
            for(size_t m = 1 ; m < M-1 ; ++m)
            {
                dxdt[n][m] = coupling_func( x[n][m+1] - x[n][m] ) +
                             coupling_func( x[n][m-1] - x[n][m] );
            }
            dxdt[n][0] = coupling_func( x[n][1] - x[n][0] );
            if( n > 0 )
            {
                dxdt[n][0] += coupling_func( x[n-1].back() - x[n].front() );
            }
            dxdt[n][M-1] = coupling_func( x[n][M-2] - x[n][M-1] );
            if( n < N-1 )
            {
                dxdt[n][M-1] += coupling_func( x[n+1].front() - x[n].back() );
            }
        }
    }

    double coupling_func( double x ) const
    {
        return sin( x ) - m_gamma * ( 1.0 - cos( x ) );
    }

    double m_gamma;
};
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Using the `openmp_state<T>` state type automatically selects `openmp_algebra` which executes odeint's internal computations on parallel regions. Hence, no manual configuration of the stepper is necessary. At the end of the integration, we use `unsplit` to concatenate the sub-regions back together into a single vector.

```
integrate_n_steps( runge_kutta4<state_type>() , phase_chain_omp_state( 1.2 ) ,
                   x_split , 0.0 , 0.01 , 100 );
unsplit( x_split , x );
```

> **Note**
>
> You don't actually need to use `openmp_state<T>` for advanced use cases, `openmp_algebra` is simply an alias for `openmp_nested_algebra<range_algebra>` and supports any model of Random Access Range as the outer, parallel state type, and will use the given algebra on its elements.

See openmp/phase_chain_omp_state.cpp for the complete example.

MPI

To expand the parallel computation across multiple machines we can use MPI.

The system function implementation is similar to the OpenMP variant with split data, the main difference being that while OpenMP uses a spawn/join model where everything not explicitly paralleled is only executed in the main thread, in MPI's model each node enters the `main()` method independently, diverging based on its rank and synchronizing through message-passing and explicit barriers.

odeint's MPI support is implemented as an external backend, too. Depending on the MPI implementation the code might need to be compiled with i.e. `mpic++`.

```
#include <boost/numeric/odeint.hpp>
#include <boost/numeric/odeint/external/mpi/mpi.hpp>
```

Instead of reading another thread's data, we asynchronously send and receive the relevant data from neighbouring nodes, performing some computation in the interim to hide the latency.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
struct phase_chain
{
    phase_chain( double gamma = 0.5 )
    : m_gamma( gamma ) { }

    void operator()( const state_type &x , state_type &dxdt , double /* t */ ) const
    {
        const size_t N = x.size();
        #pragma omp parallel for schedule(runtime)
        for(size_t i = 1 ; i < N - 1 ; ++i)
        {
            dxdt[i] = coupling_func( x[i+1] - x[i] ) +
                      coupling_func( x[i-1] - x[i] );
        }
        dxdt[0  ] = coupling_func( x[1  ] - x[0  ] );
        dxdt[N-1] = coupling_func( x[N-2] - x[N-1] );
    }

    double coupling_func( double x ) const
    {
        return sin( x ) - m_gamma * ( 1.0 - cos( x ) );
    }

    double m_gamma;
};
```

Analogous to `openmp_state<T>` we use `mpi_state< InnerState<T> >`, which automatically selects `mpi_nested_algebra` and the appropriate MPI-oblivious inner algebra (since our inner state is a `vector`, the inner algebra will be `range_algebra` as in the OpenMP example).

```
typedef mpi_state< vector<double> > state_type;
```

In the main program we construct a `communicator` which tells us the `size` of the cluster and the current node's `rank` within that. We generate the input data on the master node only, avoiding unnecessary work on the other nodes. Instead of simply copying chunks, `split` acts as a MPI collective function here and sends/receives regions from master to each slave. The input argument is ignored on the slaves, but the master node receives a region in its output and will participate in the computation.

```
boost::mpi::environment env( argc , argv );
boost::mpi::communicator world;

const size_t N = 131101;
vector<double> x;
if( world.rank() == 0 )
{
    x.resize( N );
    boost::random::uniform_real_distribution<double> distribution( 0.0 , 2.0*pi );
    boost::random::mt19937 engine( 0 );
    generate( x.begin() , x.end() , boost::bind( distribution , engine ) );
}

state_type x_split( world );
split( x , x_split );
```

Now that `x_split` contains (only) the local chunk for each node, we start the integration.

To print the result on the master node, we send the processed data back using `unsplit`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
integrate_n_steps( runge_kutta4<state_type>() , phase_chain_mpi_state( 1.2 ) ,
                   x_split , 0.0 , 0.01 , 100 );
unsplit( x_split , x );
```

Note

`mpi_nested_algebra::for_eachN` doesn't use any MPI constructs, it simply calls the inner algebra on the local chunk and the system function is not guarded by any barriers either, so if you don't manually place any (for example in parameter studies cases where the elements are completely independent) you might see the nodes diverging, returning from this call at different times.

See mpi/phase_chain.cpp for the complete example.

Concepts

MPI State

As used by `mpi_nested_algebra`.

Notation

`InnerState`	The inner state type
`State`	The MPI-state type
`state`	Object of type `State`
`world`	Object of type `boost::mpi::communicator`

Valid Expressions

Name	Expression	Type	Semantics
Construct a state with a communicator	`State(world)`	`State`	Constructs the State.
Construct a state with the default communicator	`State()`	`State`	Constructs the State.
Get the current node's inner state	`state()`	`InnerState`	Returns a (const) reference.
Get the communicator	`state.world`	`boost::mpi::communicator`	See Boost.MPI.

Models

- `mpi_state<InnerState>`

OpenMP Split State

As used by `openmp_nested_algebra`, essentially a Random Access Container with `ValueType = InnerState`.

Notation

`InnerState`	The inner state type

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

`State`	The split state type
`state`	Object of type `State`

Valid Expressions

Name	Expression	Type	Semantics
Construct a state for `n` chunks	`State(n)`	`State`	Constructs underlying `vector`.
Get a chunk	`state[i]`	`InnerState`	Accesses underlying `vector`.
Get the number of chunks	`state.size()`	`size_type`	Returns size of underlying `vector`.

Models

- `openmp_state<ValueType>` with `InnerState = vector<ValueType>`

Splitter

Notation

`Container1`	The continuous-data container type
`x`	Object of type `Container1`
`Container2`	The chunked-data container type
`y`	Object of type `Container2`

Valid Expressions

Name	Expression	Type	Semantics
Copy chunks of input to output elements	`split(x, y)`	`void`	Calls `split_impl<Container1, Container2>::split(x, y)`, splits `x` into `y.size()` chunks.
Join chunks of input elements to output	`unsplit(y, x)`	`void`	Calls `unsplit_impl<Container2, Container1>::unsplit(y, x)`, assumes `x` is of the correct size σ *y[i].size()*, does not resize `x`.

Models

- defined for `Container1` = Boost.Range and `Container2 = openmp_state`
- and `Container2 = mpi_state`.

To implement splitters for containers incompatible with Boost.Range, specialize the `split_impl` and `unsplit_impl` types:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template< class Container1, class Container2 , class Enabler = void >
struct split_impl {
    static void split( const Container1 &from , Container2 &to );
};

template< class Container2, class Container1 , class Enabler = void >
struct unsplit_impl {
    static void unsplit( const Container2 &from , Container1 &to );
};
```

All examples

The following table gives an overview over all examples.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 4. Examples Overview

File	Brief Description
bind_member_functions.cpp	This examples shows how member functions can be used as system functions in odeint.
bind_member_functions_cpp11.cpp	This examples shows how member functions can be used as system functions in odeint with `std::bind` in C++11.
bulirsch_stoer.cpp	Shows the usage of the Bulirsch-Stoer method.
chaotic_system.cpp	The chaotic system examples integrates the Lorenz system and calculates the Lyapunov exponents.
elliptic_functions.cpp	Example calculating the elliptic functions using Bulirsch-Stoer and Runge-Kutta-Dopri5 Steppers with dense output.
fpu.cpp	The Fermi-Pasta-Ulam (FPU) example shows how odeint can be used to integrate lattice systems.
generation_functions.cpp	Shows skeletal code on how to implement own factory functions.
harmonic_oscillator.cpp	The harmonic oscillator examples gives a brief introduction to odeint and shows the usage of the classical Runge-Kutta-solvers.
harmonic_oscillator_units.cpp	This examples shows how Boost.Units can be used with odeint.
heun.cpp	The Heun example shows how an custom Runge-Kutta stepper can be created with odeint generic Runge-Kutta method.
list_lattice.cpp	Example of a phase lattice integration using `std::list` as state type.
lorenz_point.cpp	Alternative way of integrating lorenz by using a self defined point3d data type as state type.
my_vector.cpp	Simple example showing how to get odeint to work with a self-defined vector type.
phase_oscillator_ensemble.cpp	The phase oscillator ensemble example shows how globally coupled oscillators can be analyzed and how statistical measures can be computed during integration.
resizing_lattice.cpp	Shows the strength of odeint's memory management by simulating a Hamiltonian system on an expanding lattice.
simple1d.cpp	Integrating a simple, one-dimensional ODE showing the usage of integrate- and generate-functions.
solar_system.cpp	The solar system example shows the usage of the symplectic solvers.
stepper_details.cpp	Trivial example showing the usability of the several stepper classes.
stiff_system.cpp	The stiff system example shows the usage of the stiff solvers using the Jacobian of the system function.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

File	Brief Description
stochastic_euler.cpp	Implementation of a custom stepper - the stochastic euler - for solving stochastic differential equations.
stuart_landau.cpp	The Stuart-Landau example shows how odeint can be used with complex state types.
two_dimensional_phase_lattice.cpp	The 2D phase oscillator example shows how a two-dimensional lattice works with odeint and how matrix types can be used as state types in odeint.
van_der_pol_stiff.cpp	This stiff system example again shows the usage of the stiff solvers by integrating the van der Pol oscillator.
gmpxx/lorenz_gmpxx.cpp	This examples integrates the Lorenz system by means of an arbitrary precision type.
mtl/gauss_packet.cpp	The MTL-Gauss-packet example shows how the MTL can be easily used with odeint.
mtl/implicit_euler_mtl.cpp	This examples shows the usage of the MTL implicit Euler method with a sparse matrix type.
thrust/phase_oscillator_ensemble.cu	The Thrust phase oscillator ensemble example shows how globally coupled oscillators can be analyzed with Thrust and CUDA, employing the power of modern graphic devices.
thrust/phase_oscillator_chain.cu	The Thrust phase oscillator chain example shows how chains of nearest neighbor coupled oscillators can be integrated with Thrust and odeint.
thrust/lorenz_parameters.cu	The Lorenz parameters examples show how ensembles of ordinary differential equations can be solved by means of Thrust to study the dependence of an ODE on some parameters.
thrust/relaxation.cu	Another examples for the usage of Thrust.
ublas/lorenz_ublas.cpp	This example shows how the ublas vector types can be used with odeint.
vexcl/lorenz_ensemble.cpp	This example shows how the VexCL - a framework for OpenCL computation - can be used with odeint.
openmp/lorenz_ensemble_simple.cpp	OpenMP Lorenz attractor parameter study with continuous data.
openmp/lorenz_ensemble.cpp	OpenMP Lorenz attractor parameter study with split data.
openmp/lorenz_ensemble_nested.cpp	OpenMP Lorenz attractor parameter study with nested `vector_space_algebra`.
openmp/phase_chain.cpp	OpenMP nearest neighbour coupled phase chain with continuous state.
openmp/phase_chain_omp_state.cpp	OpenMP nearest neighbour coupled phase chain with split state.
mpi/phase_chain.cpp	MPI nearest neighbour coupled phase chain.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

File	Brief Description
2d_lattice/spreading.cpp	This examples shows how a `vector< vector< T > >` can be used a state type for odeint and how a resizing mechanism of this state can be implemented.
quadmath/black_hole.cpp	This examples shows how gcc libquadmath can be used with odeint. It provides a high precision floating point type which is adapted to odeint in this example.
molecular_dynamics.cpp	A very basic molecular dynamics simulation with the Velocity-Verlet method.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

odeint in detail

Steppers

Solving ordinary differential equation numerically is usually done iteratively, that is a given state of an ordinary differential equation is iterated forward *x(t) -> x(t+dt) -> x(t+2dt)*. The steppers in odeint perform one single step. The most general stepper type is described by the Stepper concept. The stepper concepts of odeint are described in detail in section Concepts, here we briefly present the mathematical and numerical details of the steppers. The Stepper has two versions of the `do_step` method, one with an in-place transform of the current state and one with an out-of-place transform:

`do_step( sys , inout , t , dt )`

`do_step( sys , in , t , out , dt )`

The first parameter is always the system function - a function describing the ODE. In the first version the second parameter is the step which is here updated in-place and the third and the fourth parameters are the time and step size (the time step). After a call to `do_step` the state `inout` is updated and now represents an approximate solution of the ODE at time *t+dt*. In the second version the second argument is the state of the ODE at time *t*, the third argument is t, the fourth argument is the approximate solution at time *t+dt* which is filled by `do_step` and the fifth argument is the time step. Note that these functions do not change the time `t`.

System functions

Up to now, we have nothing said about the system function. This function depends on the stepper. For the explicit Runge-Kutta steppers this function can be a simple callable object hence a simple (global) C-function or a functor. The parameter syntax is `sys( x , dxdt , t )` and it is assumed that it calculates *dx/dt = f(x,t)*. The function structure in most cases looks like:

```
void sys( const state_type & /*x*/ , state_type & /*dxdt*/ , const double /*t*/ )
{
    // ...
}
```

Other types of system functions might represent Hamiltonian systems or systems which also compute the Jacobian needed in implicit steppers. For information which stepper uses which system function see the stepper table below. It might be possible that odeint will introduce new system types in near future. Since the system function is strongly related to the stepper type, such an introduction of a new stepper might result in a new type of system function.

Explicit steppers

A first specialization are the explicit steppers. Explicit means that the new state of the ode can be computed explicitly from the current state without solving implicit equations. Such steppers have in common that they evaluate the system at time *t* such that the result of *f(x,t)* can be passed to the stepper. In odeint, the explicit stepper have two additional methods

`do_step( sys , inout , dxdtin , t , dt )`

`do_step( sys , in , dxdtin , t , out , dt )`

Here, the additional parameter is the value of the function *f* at state *x* and time *t*. An example is the Runge-Kutta stepper of fourth order:

```
runge_kutta4< state_type > rk;
rk.do_step( sys1 , inout , t , dt );               // In-place transformation of inout
rk.do_step( sys2 , inout , t , dt );               // call with different system: Ok
rk.do_step( sys1 , in , t , out , dt );            // Out-of-place transformation
rk.do_step( sys1 , inout , dxdtin , t , dt );      // In-place tranformation of inout
rk.do_step( sys1 , in , dxdtin , t , out , dt );   // Out-of-place transformation
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

In fact, you do not need to call these two methods. You can always use the simpler `do_step( sys , inout , t , dt )`, but sometimes the derivative of the state is needed externally to do some external computations or to perform some statistical analysis.

A special class of the explicit steppers are the FSAL (first-same-as-last) steppers, where the last evaluation of the system function is also the first evaluation of the following step. For such steppers the `do_step` method are slightly different:

`do_step( sys , inout , dxdtinout , t , dt )`

`do_step( sys , in , dxdtin , out , dxdtout , t , dt )`

This method takes the derivative at time `t` and also stores the derivative at time *t+dt*. Calling these functions subsequently iterating along the solution one saves one function call by passing the result for dxdt into the next function call. However, when using FSAL steppers without supplying derivatives:

`do_step( sys , inout , t , dt )`

the stepper internally satisfies the FSAL property which means it remembers the last `dxdt` and uses it for the next step. An example for a FSAL stepper is the Runge-Kutta-Dopri5 stepper. The FSAL trick is sometimes also referred as the Fehlberg trick. An example how the FSAL steppers can be used is

```
runge_kutta_dopri5< state_type > rk;
rk.do_step( sys1 , in , t , out , dt );
rk.do_step( sys2 , in , t , out , dt );         // DONT do this, sys1 is assumed

rk.do_step( sys2 , in2 , t , out , dt );
rk.do_step( sys2 , in3 , t , out , dt );        // DONT do this, in2 is assumed

rk.do_step( sys1 , inout , dxdtinout , t , dt );
rk.do_step( sys2 , inout , dxdtinout , t , dt );           // Ok, internal derivative is not ↵
used, dxdtinout is updated

rk.do_step( sys1 , in , dxdtin , t , out , dxdtout , dt );
rk.do_step( sys2 , in , dxdtin , t , out , dxdtout , dt ); // Ok, internal derivative is not used
```

Caution

The FSAL-steppers save the derivative at time *t+dt* internally if they are called via `do_step( sys , in , out , t , dt )`. The first call of `do_step` will initialize `dxdt` and for all following calls it is assumed that the same system and the same state are used. If you use the FSAL stepper within the integrate functions this is taken care of automatically. See the Using steppers section for more details or look into the table below to see which stepper have an internal state.

Symplectic solvers

As mentioned above symplectic solvers are used for Hamiltonian systems. Symplectic solvers conserve the phase space volume exactly and if the Hamiltonian system is energy conservative they also conserve the energy approximately. A special class of symplectic systems are separable systems which can be written in the form $dqdt/dt = f1(p)$, $dpdt/dt = f2(q)$, where (q,p) are the state of system. The space of (q,p) is sometimes referred as the phase space and q and p are said the be the phase space variables. Symplectic systems in this special form occur widely in nature. For example the complete classical mechanics as written down by Newton, Lagrange and Hamilton can be formulated in this framework. The separability of the system depends on the specific choice of coordinates.

Symplectic systems can be solved by odeint by means of the symplectic_euler stepper and a symplectic Runge-Kutta-Nystrom method of fourth order. These steppers assume that the system is autonomous, hence the time will not explicitly occur. Further they fulfill in principle the default Stepper concept, but they expect the system to be a pair of callable objects. The first entry of this pair calculates $f1(p)$ while the second calculates $f2(q)$. The syntax is `sys.first(p,dqdt)` and `sys.second(q,dpdt)`, where the first and second part can be again simple C-functions of functors. An example is the harmonic oscillator:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef boost::array< double , 1 > vector_type;


struct harm_osc_f1
{
    void operator()( const vector_type &p , vector_type &dqdt )
    {
        dqdt[0] = p[0];
    }
};

struct harm_osc_f2
{
    void operator()( const vector_type &q , vector_type &dpdt )
    {
        dpdt[0] = -q[0];
    }
};
```

The state of such an ODE consist now also of two parts, the part for q (also called the coordinates) and the part for p (the momenta). The full example for the harmonic oscillator is now:

```
pair< vector_type , vector_type > x;
x.first[0] = 1.0; x.second[0] = 0.0;
symplectic_rkn_sb3a_mclachlan< vector_type > rkn;
rkn.do_step( make_pair( harm_osc_f1() , harm_osc_f2() ) , x , t , dt );
```

If you like to represent the system with one class you can easily bind two public method:

```
struct harm_osc
{
    void f1( const vector_type &p , vector_type &dqdt ) const
    {
        dqdt[0] = p[0];
    }

    void f2( const vector_type &q , vector_type &dpdt ) const
    {
        dpdt[0] = -q[0];
    }
};
```

```
harm_osc h;
rkn.do_step( make_pair( boost::bind( &harm_osc::f1 , h , _1 , _2 ) , boost::bind( &harm_osc::f2 , h , _1 , _2 ) ) ,
        x , t , dt );
```

Many Hamiltonian system can be written as *dq/dt=p*, *dp/dt=f(q)* which is computationally much easier than the full separable system. Very often, it is also possible to transform the original equations of motion to bring the system in this simplified form. This kind of system can be used in the symplectic solvers, by simply passing *f(p)* to the `do_step` method, again *f(p)* will be represented by a simple C-function or a functor. Here, the above example of the harmonic oscillator can be written as

```
pair< vector_type , vector_type > x;
x.first[0] = 1.0; x.second[0] = 0.0;
symplectic_rkn_sb3a_mclachlan< vector_type > rkn;
rkn.do_step( harm_osc_f1() , x , t , dt );
```

In this example the function `harm_osc_f1` is exactly the same function as in the above examples.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Note, that the state of the ODE must not be constructed explicitly via `pair< vector_type , vector_type > x`. One can also use a combination of `make_pair` and `ref`. Furthermore, a convenience version of `do_step` exists which takes q and p without combining them into a pair:

```
rkn.do_step( harm_osc_f1() , make_pair( boost::ref( q ) , boost::ref( p ) ) , t , dt );
rkn.do_step( harm_osc_f1() , q , p , t , dt );
rkn.do_step( make_pair( harm_osc_f1() , harm_osc_f2() ) , q , p , t , dt );
```

Implicit solvers

Caution

This section is not up-to-date.

For some kind of systems the stability properties of the classical Runge-Kutta are not sufficient, especially if the system is said to be stiff. A stiff system possesses two or more time scales of very different order. Solvers for stiff systems are usually implicit, meaning that they solve equations like $x(t+dt) = x(t) + dt * f(x(t+1))$. This particular scheme is the implicit Euler method. Implicit methods usually solve the system of equations by a root finding algorithm like the Newton method and therefore need to know the Jacobian of the system $J_{ij} = df_i / dx_j$.

For implicit solvers the system is again a pair, where the first component computes $f(x,t)$ and the second the Jacobian. The syntax is `sys.first( x , dxdt , t )` and `sys.second( x , J , t )`. For the implicit solver the `state_type` is `ublas::vector` and the Jacobian is represented by `ublas::matrix`.

Important

Implicit solvers only work with ublas::vector as state type. At the moment, no other state types are supported.

Multistep methods

Another large class of solvers are multi-step method. They save a small part of the history of the solution and compute the next step with the help of this history. Since multi-step methods know a part of their history they do not need to compute the system function very often, usually it is only computed once. This makes multi-step methods preferable if a call of the system function is expensive. Examples are ODEs defined on networks, where the computation of the interaction is usually where expensive (and might be of order O(N^2)).

Multi-step methods differ from the normal steppers. They save a part of their history and this part has to be explicitly calculated and initialized. In the following example an Adams-Bashforth-stepper with a history of 5 steps is instantiated and initialized;

```
adams_bashforth_moulton< 5 , state_type > abm;
abm.initialize( sys , inout , t , dt );
abm.do_step( sys , inout , t , dt );
```

The initialization uses a fourth-order Runge-Kutta stepper and after the call of `initialize` the state of `inout` has changed to the current state, such that it can be immediately used by passing it to following calls of `do_step`. You can also use you own steppers to initialize the internal state of the Adams-Bashforth-Stepper:

```
abm.initialize( runge_kutta_fehlberg78< state_type >() , sys , inout , t , dt );
```

Many multi-step methods are also explicit steppers, hence the parameter of `do_step` method do not differ from the explicit steppers.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Caution

The multi-step methods have some internal variables which depend on the explicit solution. Hence after any external changes of your state (e.g. size) or system the initialize function has to be called again to adjust the internal state of the stepper. If you use the integrate functions this will be taken into account. See the Using steppers section for more details.

Controlled steppers

Many of the above introduced steppers possess the possibility to use adaptive step-size control. Adaptive step size integration works in principle as follows:

1. The error of one step is calculated. This is usually done by performing two steps with different orders. The difference between these two steps is then used as a measure for the error. Stepper which can calculate the error are Error Stepper and they form an own class with an separate concept.

2. This error is compared against some predefined error tolerances. Are the tolerance violated the step is reject and the step-size is decreases. Otherwise the step is accepted and possibly the step-size is increased.

The class of controlled steppers has their own concept in odeint - the Controlled Stepper concept. They are usually constructed from the underlying error steppers. An example is the controller for the explicit Runge-Kutta steppers. The Runge-Kutta steppers enter the controller as a template argument. Additionally one can pass the Runge-Kutta stepper to the constructor, but this step is not necessary; the stepper is default-constructed if possible.

Different step size controlling mechanism exist. They all have in common that they somehow compare predefined error tolerance against the error and that they might reject or accept a step. If a step is rejected the step size is usually decreased and the step is made again with the reduced step size. This procedure is repeated until the step is accepted. This algorithm is implemented in the integration functions.

A classical way to decide whether a step is rejected or accepted is to calculate

$val = || \, | err_i | \, / \, (\varepsilon_{abs} + \varepsilon_{rel} * (a_x \, | x_i | + a_{dxdt} \, | \, | dxdt_i | \,) ||$

ε_{abs} and ε_{rel} are the absolute and the relative error tolerances, and $|| \, x \, ||$ is a norm, typically $||x||=(\Sigma_i \, x_i^2)^{1/2}$ or the maximum norm. The step is rejected if *val* is greater then 1, otherwise it is accepted. For details of the used norms and error tolerance see the table below.

For the `controlled_runge_kutta` stepper the new step size is then calculated via

$val > 1 : dt_{new} = dt_{current} \, max(0.9 \, pow(val , -1 / (O_E - 1)) , 0.2)$

$val < 0.5 : dt_{new} = dt_{current} \, min(0.9 \, pow(val , -1 / O_S) , 5)$

$else : dt_{new} = dt_{current}$

Here, O_S and O_E are the order of the stepper and the error stepper. These formulas also contain some safety factors, avoiding that the step size is reduced or increased to much. For details of the implementations of the controlled steppers in odeint see the table below.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 5. Adaptive step size algorithms

Stepper	Tolerance formula	Norm	Step size adaption
`controlled_runge_kutta`	$val = \Vert \vert err_i \vert / (\varepsilon_{abs} + \varepsilon_{rel} * (a_x \vert x_i \vert + a_{dxdt} \vert \vert dxdt_i \vert)\Vert$	$\Vert x \Vert = max(x_i)$	$val > 1 : dt_{new} = dt_{current} max(0.9 pow(val , -1 / (O_E - 1)) , 0.2)$ $val < 0.5 : dt_{new} = dt_{current} min(0.9 pow(val , -1 / O_S) , 5)$ $else : dt_{new} = dt_{current}$
`rosenbrock4_controller`	$val = \Vert err_i / (\varepsilon_{abs} + \varepsilon_{rel} max(\vert x_i \vert , \vert xold_i \vert)) \Vert$	$\Vert x \Vert=(\Sigma_i x_i^2)^{1/2}$	$fac = max(1 / 6 , min(5 , pow(val , 1 / 4) / 0.9)$ $fac2 = max(1 / 6 , min(5 , dt_{old} / dt_{current} pow(val^2 / val_{old} , 1 / 4) / 0.9)$ $val > 1 : dt_{new} = dt_{current} / fac$ $val < 1 : dt_{new} = dt_{current} / max(fac , fac2)$
bulirsch_stoer	$tol=1/2$	-	$dt_{new} = dt_{old}^{1/a}$

To ease to generation of the controlled stepper, generation functions exist which take the absolute and relative error tolerances and a predefined error stepper and construct from this knowledge an appropriate controlled stepper. The generation functions are explained in detail in Generation functions.

Dense output steppers

A fourth class of stepper exists which are the so called dense output steppers. Dense-output steppers might take larger steps and interpolate the solution between two consecutive points. This interpolated points have usually the same order as the order of the stepper. Dense-output steppers are often composite stepper which take the underlying method as a template parameter. An example is the `dense_output_runge_kutta` stepper which takes a Runge-Kutta stepper with dense-output facilities as argument. Not all Runge-Kutta steppers provide dense-output calculation; at the moment only the Dormand-Prince 5 stepper provides dense output. An example is

```
dense_output_runge_kutta< controlled_runge_kutta< runge_kutta_dopri5< state_type > > > dense;
dense.initialize( in , t , dt );
pair< double , double > times = dense.do_step( sys );
(void)times;
```

Dense output stepper have their own concept. The main difference to usual steppers is that they manage the state and time internally. If you call `do_step`, only the ODE is passed as argument. Furthermore `do_step` return the last time interval: `t` and `t+dt`, hence you can interpolate the solution between these two times points. Another difference is that they must be initialized with `initialize`, otherwise the internal state of the stepper is default constructed which might produce funny errors or bugs.

The construction of the dense output stepper looks a little bit nasty, since in the case of the `dense_output_runge_kutta` stepper a controlled stepper and an error stepper have to be nested. To simplify the generation of the dense output stepper generation functions exist:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef boost::numeric::odeint::result_of::make_dense_output<
    runge_kutta_dopri5< state_type > >::type dense_stepper_type;
dense_stepper_type dense2 = make_dense_output( 1.0e-6 , 1.0e-
6 , runge_kutta_dopri5< state_type >() );
(void)dense2;
```

This statement is also lengthy; it demonstrates how `make_dense_output` can be used with the `result_of` protocol. The parameters to `make_dense_output` are the absolute error tolerance, the relative error tolerance and the stepper. This explicitly assumes that the underlying stepper is a controlled stepper and that this stepper has an absolute and a relative error tolerance. For details about the generation functions see Generation functions. The generation functions have been designed for easy use with the integrate functions:

```
integrate_const( make_dense_output( 1.0e-6 , 1.0e-
6 , runge_kutta_dopri5< state_type >() ) , sys , inout , t_start , t_end , dt );
```

Using steppers

This section contains some general information about the usage of the steppers in odeint.

Steppers are copied by value

The stepper in odeint are always copied by values. They are copied for the creation of the controlled steppers or the dense output steppers as well as in the integrate functions.

Steppers might have a internal state

Caution

Some of the features described in this section are not yet implemented

Some steppers require to store some information about the state of the ODE between two steps. Examples are the multi-step methods which store a part of the solution during the evolution of the ODE, or the FSAL steppers which store the last derivative at time $t+dt$, to be used in the next step. In both cases the steppers expect that consecutive calls of `do_step` are from the same solution and the same ODE. In this case it is absolutely necessary that you call `do_step` with the same system function and the same state, see also the examples for the FSAL steppers above.

Stepper with an internal state support two additional methods: `reset` which resets the state and `initialize` which initializes the internal state. The parameters of `initialize` depend on the specific stepper. For example the Adams-Bashforth-Moulton stepper provides two initialize methods: `initialize( system , inout , t , dt )` which initializes the internal states with the help of the Runge-Kutta 4 stepper, and `initialize( stepper , system , inout , t , dt )` which initializes with the help of `stepper`. For the case of the FSAL steppers, `initialize` is `initialize( sys , in , t )` which simply calculates the r.h.s. of the ODE and assigns its value to the internal derivative.

All these steppers have in common, that they initially fill their internal state by themselves. Hence you are not required to call initialize. See how this works for the Adams-Bashforth-Moulton stepper: in the example we instantiate a fourth order Adams-Bashforth-Moulton stepper, meaning that it will store 4 internal derivatives of the solution at times `(t-dt,t-2*dt,t-3*dt,t-4*dt)`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
adams_bashforth_moulton< 4 , state_type > stepper;
stepper.do_step( sys , x , t , dt );   // make one step with the classical Runge-Kutta stepper ↵
and initialize the first internal state
                                        // the internal array is now [x(t-dt)]

stepper.do_step( sys , x , t , dt );   // make one step with the classical Runge-Kutta stepper ↵
and initialize the second internal state
                                        // the internal state array is now [x(t-dt), x(t-2*dt)]

stepper.do_step( sys , x , t , dt );   // make one step with the classical Runge-Kutta stepper ↵
and initialize the third internal state
                                        // the internal state array is now [x(t-dt), x(t-
2*dt), x(t-3*dt)]

stepper.do_step( sys , x , t , dt );   // make one step with the classical Runge-Kutta stepper ↵
and initialize the fourth internal state
                                        // the internal state array is now [x(t-dt), x(t-
2*dt), x(t-3*dt), x(t-4*dt)]

stepper.do_step( sys , x , t , dt );   // make one step with Adam-Bashforth-Moulton, the intern↵
al array of states is now rotated
```

In the stepper table at the bottom of this page one can see which stepper have an internal state and hence provide the `reset` and `initialize` methods.

Stepper might be resizable

Nearly all steppers in odeint need to store some intermediate results of the type `state_type` or `deriv_type`. To do so odeint need some memory management for the internal temporaries. As this memory management is typically related to adjusting the size of vector-like types, it is called resizing in odeint. So, most steppers in odeint provide an additional template parameter which controls the size adjustment of the internal variables - the resizer. In detail odeint provides three policy classes (resizers) `always_resizer`, `initially_resizer`, and `never_resizer`. Furthermore, all stepper have a method `adjust_size` which takes a parameter representing a state type and which manually adjusts the size of the internal variables matching the size of the given instance. Before performing the actual resizing odeint always checks if the sizes of the state and the internal variable differ and only resizes if they are different.

Note

You only have to worry about memory allocation when using dynamically sized vector types. If your state type is heap allocated, like `boost::array`, no memory allocation is required whatsoever.

By default the resizing parameter is `initially_resizer`, meaning that the first call to `do_step` performs the resizing, hence memory allocation. If you have changed the size of your system and your state you have to call `adjust_size` by hand in this case. The second resizer is the `always_resizer` which tries to resize the internal variables at every call of `do_step`. Typical use cases for this kind of resizer are self expanding lattices like shown in the tutorial (Self expanding lattices) or partial differential equations with an adaptive grid. Here, no calls of `adjust_size` are required, the steppers manage everything themselves. The third class of resizer is the `never_resizer` which means that the internal variables are never adjusted automatically and always have to be adjusted by hand .

There is a second mechanism which influences the resizing and which controls if a state type is at least resizeable - a meta-function `is_resizeable`. This meta-function returns a static Boolean value if any type is resizable. For example it will return `true` for `std::vector< T >` but `false` for `boost::array< T >`. By default and for unknown types `is_resizeable` returns `false`, so if you have your own type you need to specialize this meta-function. For more details on the resizing mechanism see the section Adapt your own state types.

Which steppers should be used in which situation

odeint provides a quite large number of different steppers such that the user is left with the question of which stepper fits his needs. Our personal recommendations are:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- `runge_kutta_dopri5` is maybe the best default stepper. It has step size control as well as dense-output functionality. Simple create a dense-output stepper by `make_dense_output( 1.0e-6 , 1.0e-5 , runge_kutta_dopri5< state_type >() )`.
- `runge_kutta4` is a good stepper for constant step sizes. It is widely used and very well known. If you need to create artificial time series this stepper should be the first choice.
- 'runge_kutta_fehlberg78' is similar to the 'runge_kutta4' with the advantage that it has higher precision. It can also be used with step size control.
- `adams_bashforth_moulton` is very well suited for ODEs where the r.h.s. is expensive (in terms of computation time). It will calculate the system function only once during each step.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Stepper overview

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 6. Stepper Algorithms

Algorithm	**Class**	**Concept**	**System Concept**	**Order**	**Error Estimation**	**Dense Output**	**Internal state**	**Remarks**
Explicit Euler	`euler`	Dense Output Stepper	System	1	No	Yes	No	Very simple, only for demonstrating purpose
Modified Midpoint	`modified_midpoint`	Stepper	System	configurable (2)	No	No	No	Used in Bulirsch-Stoer implementation
Runge-Kutta 4	`runge_kutta4`	Stepper	System	4	No	No	No	The classical Runge-Kutta scheme, good general scheme without error control
Cash-Karp	`runge_kutta_cash_karp54`	Error Stepper	System	5	Yes (4)	No	No	Good general scheme with error estimation, to be used in controlled_error_stepper
Dormand-Prince 5	`runge_kutta_dopri5`	Error Stepper	System	5	Yes (4)	Yes	Yes	Standard method with error control and dense output, to be used in controlled_error_stepper and in dense_output_controlled_explicit_fsal.
Fehlberg 78	`runge_kutta_fehlberg78`	Error Stepper	System	8	Yes (7)	No	No	Good high order method with error estimation, to be used in controlled_error_stepper.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Algorithm	**Class**	**Concept**	**System Concept**	**Order**	**Error Estimation**	**Dense Output**	**Internal state**	**Remarks**
Adams Bashforth	`adams_bashforth`	Stepper	System	configurable	No	No	Yes	Multistep method
Adams Moulton	`adams_moulton`	Stepper	System	configurable	No	No	Yes	Multistep method
Adams Bashforth Moulton	`adams_bashforth_moulton`	Stepper	System	configurable	No	No	Yes	Combined multistep method
Controlled Runge-Kutta	`controlled_runge_kutta`	Controlled Stepper	System	depends	Yes	No	depends	Error control for Error Stepper. Requires an Error Stepper from above. Order depends on the given ErrorStepper
Dense Output Runge-Kutta	`dense_output_runge_kutta`	Dense Output Stepper	System	depends	No	Yes	Yes	Dense output for Stepper and Error Stepper from above if they provide dense output functionality (like `euler` and `runge_kutta_dopri5`) Order depends on the given stepper.
Bulirsch-Stoer	`bulirsch_stoer`	Controlled Stepper	System	variable	Yes	No	No	Stepper with step size and order control. Very good if high precision is required.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Algorithm	Class	Concept	System Concept	Order	Error Estimation	Dense Output	Internal state	Remarks
Bulirsch-Stoer Dense Output	`bulirsch_stoer_dense_out`	Dense Output Stepper	System	variable	Yes	Yes	No	Stepper with step size and order control as well as dense output. Very good if high precision and dense output is required.
Implicit Euler	`implicit_euler`	Stepper	Implicit System	1	No	No	No	Basic implicit routine. Requires the Jacobian. Works only with Boost.uBLAS vectors as state types.
Rosenbrock 4	`rosenbrock4`	Error Stepper	Implicit System	4	Yes	Yes	No	Good for stiff systems. Works only with Boost.uBLAS vectors as state types.
Controlled Rosenbrock 4	`rosenbrock4_controller`	Controlled Stepper	Implicit System	4	Yes	Yes	No	Rosenbrock 4 with error control. Works only with Boost.uBLAS vectors as state types.
Dense Output Rosenbrock 4	`rosenbrock4_dense_output`	Dense Output Stepper	Implicit System	4	Yes	Yes	No	Controlled Rosenbrock 4 with dense output. Works only with Boost.uBLAS vectors as state types.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Algorithm	Class	Concept	System Concept	Order	Error Estimation	Dense Output	Internal state	Remarks
Symplectic Euler	`symplectic_euler`	Stepper	Symplectic System Simple Symplectic System	1	No	No	No	Basic symplectic solver for separable Hamiltonian system
Symplectic RKN McLachlan	`symplectic_rkn_sb3a_mclachlan`	Stepper	Symplectic System Simple Symplectic System	4	No	No	No	Symplectic solver for separable Hamiltonian system with 6 stages and order 4.
Symplectic RKN McLachlan	`symplectic_rkn_sb3a_m4_mclachlan`	Stepper	Symplectic System Simple Symplectic System	4	No	No	No	Symplectic solver with 5 stages and order 4, can be used with arbitrary precision types.
Velocity Verlet	`velocity_verlet`	Stepper	Second Order System	1	No	No	Yes	Velocity verlet method suitable for molecular dynamics simulation.

Custom steppers

Finally, one can also write new steppers which are fully compatible with odeint. They only have to fulfill one or several of the stepper Concepts of odeint.

We will illustrate how to write your own stepper with the example of the stochastic Euler method. This method is suited to solve stochastic differential equations (SDEs). A SDE has the form

$dx/dt = f(x) + g(x) \xi(t)$

where ξ is Gaussian white noise with zero mean and a standard deviation $\sigma(t)$. $f(x)$ is said to be the deterministic part while $g(x) \xi$ is the noisy part. In case $g(x)$ is independent of x the SDE is said to have additive noise. It is not possible to solve SDE with the classical solvers for ODEs since the noisy part of the SDE has to be scaled differently then the deterministic part with respect to the time step. But there exist many solvers for SDEs. A classical and easy method is the stochastic Euler solver. It works by iterating

$x(t+\Delta t) = x(t) + \Delta t f(x(t)) + \Delta t^{1/2} g(x) \xi(t)$

where $\xi(t)$ is an independent normal distributed random variable.

Now we will implement this method. We will call the stepper `stochastic_euler`. It models the Stepper concept. For simplicity, we fix the state type to be an `array< double , N >` The class definition looks like

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template< size_t N > class stochastic_euler
{
public:

    typedef boost::array< double , N > state_type;
    typedef boost::array< double , N > deriv_type;
    typedef double value_type;
    typedef double time_type;
    typedef unsigned short order_type;
    typedef boost::numeric::odeint::stepper_tag stepper_category;

    static order_type order( void ) { return 1; }

    // ...
};
```

The types are needed in order to fulfill the stepper concept. As internal state and deriv type we use simple arrays in the stochastic Euler, they are needed for the temporaries. The stepper has the order one which is returned from the `order()` function.

The system functions needs to calculate the deterministic and the stochastic part of our stochastic differential equation. So it might be suitable that the system function is a pair of functions. The first element of the pair computes the deterministic part and the second the stochastic one. Then, the second part also needs to calculate the random numbers in order to simulate the stochastic process. We can now implement the `do_step` method

```
template< size_t N > class stochastic_euler
{
public:

    // ...

    template< class System >
    void do_step( System system , state_type &x , time_type t , time_type dt ) const
    {
        deriv_type det , stoch ;
        system.first( x , det );
        system.second( x , stoch );
        for( size_t i=0 ; i<x.size() ; ++i )
            x[i] += dt * det[i] + sqrt( dt ) * stoch[i];
    }
};
```

This is all. It is quite simple and the stochastic Euler stepper implement here is quite general. Of course it can be enhanced, for example

- use of operations and algebras as well as the resizing mechanism for maximal flexibility and portability
- use of `boost::ref` for the system functions
- use of `boost::range` for the state type in the `do_step` method
- ...

Now, lets look how we use the new stepper. A nice example is the Ornstein-Uhlenbeck process. It consists of a simple Brownian motion overlapped with an relaxation process. Its SDE reads

$dx/dt = - x + \xi$

where ξ is Gaussian white noise with standard deviation σ. Implementing the Ornstein-Uhlenbeck process is quite simple. We need two functions or functors - one for the deterministic and one for the stochastic part:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
const static size_t N = 1;
typedef boost::array< double , N > state_type;

struct ornstein_det
{
    void operator()( const state_type &x , state_type &dxdt ) const
    {
        dxdt[0] = -x[0];
    }
};

struct ornstein_stoch
{
    boost::mt19937 m_rng;
    boost::normal_distribution<> m_dist;

    ornstein_stoch( double sigma ) : m_rng() , m_dist( 0.0 , sigma ) { }

    void operator()( const state_type &x , state_type &dxdt )
    {
        dxdt[0] = m_dist( m_rng );
    }
};
```

In the stochastic part we have used the Mersenne twister for the random number generation and a Gaussian white noise generator `normal_distribution` with standard deviation σ. Now, we can use the stochastic Euler stepper with the integrate functions:

```
double dt = 0.1;
state_type x = {{ 1.0 }};
integrate_const( stochastic_euler< N >() , make_pair( ornstein_det() , ornstein_stoch( 1.0 ) ) ,
        x , 0.0 , 10.0 , dt , streaming_observer() );
```

Note, how we have used the `make_pair` function for the generation of the system function.

Custom Runge-Kutta steppers

odeint provides a C++ template meta-algorithm for constructing arbitrary Runge-Kutta schemes [1]. Some schemes are predefined in odeint, for example the classical Runge-Kutta of fourth order, or the Runge-Kutta-Cash-Karp 54 and the Runge-Kutta-Fehlberg 78 method. You can use this meta algorithm to construct you own solvers. This has the advantage that you can make full use of odeint's algebra and operation system.

Consider for example the method of Heun, defined by the following Butcher tableau:

```
c1 = 0

c2 = 1/3, a21 = 1/3

c3 = 2/3, a31 =  0 , a32 = 2/3

          b1  = 1/4, b2  = 0  , b3 = 3/4
```

Implementing this method is very easy. First you have to define the constants:

---

[1] M. Mulansky, K. Ahnert, Template-Metaprogramming applied to numerical problems, arxiv:1110.3233

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template< class Value = double >
struct heun_a1 : boost::array< Value , 1 > {
    heun_a1( void )
    {
        (*this)[0] = static_cast< Value >( 1 ) / static_cast< Value >( 3 );
    }
};

template< class Value = double >
struct heun_a2 : boost::array< Value , 2 >
{
    heun_a2( void )
    {
        (*this)[0] = static_cast< Value >( 0 );
        (*this)[1] = static_cast< Value >( 2 ) / static_cast< Value >( 3 );
    }
};


template< class Value = double >
struct heun_b : boost::array< Value , 3 >
{
    heun_b( void )
    {
        (*this)[0] = static_cast<Value>( 1 ) / static_cast<Value>( 4 );
        (*this)[1] = static_cast<Value>( 0 );
        (*this)[2] = static_cast<Value>( 3 ) / static_cast<Value>( 4 );
    }
};

template< class Value = double >
struct heun_c : boost::array< Value , 3 >
{
    heun_c( void )
    {
        (*this)[0] = static_cast< Value >( 0 );
        (*this)[1] = static_cast< Value >( 1 ) / static_cast< Value >( 3 );
        (*this)[2] = static_cast< Value >( 2 ) / static_cast< Value >( 3 );
    }
};
```

While this might look cumbersome, packing all parameters into a templatized class which is not immediately evaluated has the advantage that you can change the `value_type` of your stepper to any type you like - presumably arbitrary precision types. One could also instantiate the coefficients directly

```
const boost::array< double , 1 > heun_a1 = {{ 1.0 / 3.0 }};
const boost::array< double , 2 > heun_a2 = {{ 0.0 , 2.0 / 3.0 }};
const boost::array< double , 3 > heun_b = {{ 1.0 / 4.0 , 0.0 , 3.0 / 4.0 }};
const boost::array< double , 3 > heun_c = {{ 0.0 , 1.0 / 3.0 , 2.0 / 3.0 }};
```

But then you are nailed down to use doubles.

Next, you need to define your stepper, note that the Heun method has 3 stages and produces approximations of order 3:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    class State ,
    class Value = double ,
    class Deriv = State ,
    class Time = Value ,
    class Algebra = boost::numeric::odeint::range_algebra ,
    class Operations = boost::numeric::odeint::default_operations ,
    class Resizer = boost::numeric::odeint::initially_resizer
>
class heun : public
boost::numeric::odeint::explicit_generic_rk< 3 , 3 , State , Value , Deriv , Time ,
                                             Algebra , Operations , Resizer >
{

public:

    typedef boost::numeric::odeint::explicit_generic_rk< 3 , 3 , State , Value , Deriv , Time ,
                                                         Algebra , Operations , Resizer > step↲
per_base_type;

    typedef typename stepper_base_type::state_type state_type;
    typedef typename stepper_base_type::wrapped_state_type wrapped_state_type;
    typedef typename stepper_base_type::value_type value_type;
    typedef typename stepper_base_type::deriv_type deriv_type;
    typedef typename stepper_base_type::wrapped_deriv_type wrapped_deriv_type;
    typedef typename stepper_base_type::time_type time_type;
    typedef typename stepper_base_type::algebra_type algebra_type;
    typedef typename stepper_base_type::operations_type operations_type;
    typedef typename stepper_base_type::resizer_type resizer_type;
    typedef typename stepper_base_type::stepper_type stepper_type;

    heun( const algebra_type &algebra = algebra_type() )
    : stepper_base_type(
            fusion::make_vector(
                heun_a1<Value>() ,
                heun_a2<Value>() ) ,
            heun_b<Value>() , heun_c<Value>() , algebra )
    { }
};
```

That's it. Now, we have a new stepper method and we can use it, for example with the Lorenz system:

```
typedef boost::array< double , 3 > state_type;
heun< state_type > h;
state_type x = {{ 10.0 , 10.0 , 10.0 }};

integrate_const( h , lorenz() , x , 0.0 , 100.0 , 0.01 ,
                 streaming_observer( std::cout ) );
```

Generation functions

In the Tutorial we have learned how we can use the generation functions `make_controlled` and `make_dense_output` to create controlled and dense output stepper from a simple stepper or an error stepper. The syntax of these two functions is very simple:

```
auto stepper1 = make_controlled( 1.0e-6 , 1.0e-6 , stepper_type() );
auto stepper2 = make_dense_output( 1.0e-6 , 1.0e-6 , stepper_type() );
```

The first two parameters are the absolute and the relative error tolerances and the third parameter is the stepper. In C++03 you can infer the type from the `result_of` mechanism:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::numeric::odeint::result_of::make_controlled< stepper_type >::type stepper3 = make_con↲
trolled( 1.0e-6 , 1.0e-6 , stepper_type() );
(void)stepper3;
boost::numeric::odeint::result_of::make_dense_output< stepper_type >::type step↲
per4 = make_dense_output( 1.0e-6 , 1.0e-6 , stepper_type() );
(void)stepper4;
```

To use your own steppers with the `make_controlled` or `make_dense_output` you need to specialize two class templates. Suppose your steppers are called `custom_stepper`, `custom_controller` and `custom_dense_output`. Then, the first class you need to specialize is `boost::numeric::get_controller`, a meta function returning the type of the controller:

```
namespace boost { namespace numeric { namespace odeint {

template<>
struct get_controller< custom_stepper >
{
    typedef custom_controller type;
};

} } }
```

The second one is a factory class `boost::numeric::odeint::controller_factory` which constructs the controller from the tolerances and the stepper. In our dummy implementation this class is

```
namespace boost { namespace numeric { namespace odeint {

template<>
struct controller_factory< custom_stepper , custom_controller >
{
   custom_controller operator()( double abs_tol , double rel_tol , const custom_stepper & ) const
    {
        return custom_controller();
    }
};

} } }
```

This is all to use the `make_controlled` mechanism. Now you can use your controller via

```
auto stepper5 = make_controlled( 1.0e-6 , 1.0e-6 , custom_stepper() );
```

For the dense_output_stepper everything works similar. Here you have to specialize `boost::numeric::odeint::get_dense_output` and `boost::numeric::odeint::dense_output_factory`. These two classes have the same syntax as their relatives `get_controller` and `controller_factory`.

All controllers and dense-output steppers in odeint can be used with these mechanisms. In the table below you will find, which steppers is constructed from `make_controlled` or `make_dense_output` if applied on a stepper from odeint:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 7. Generation functions make_controlled(abs_error , rel_error , stepper)

Stepper	Result of make_controlled	Remarks
`runge_kutta_cash_karp54`	`controlled_runge_kutta< runge_kutta_cash_karp54 , default_error_checker<...> >`	$a_x=1$, $a_{dxdt}=1$
`runge_kutta_fehlberg78`	`controlled_runge_kutta< runge_kutta_fehlberg78 , default_error_checker<...> >`	$a_x=1$, $a_{dxdt}=1$
`runge_kutta_dopri5`	`controlled_runge_kutta< runge_kutta_dopri5 , default_error_checker<...> >`	$a_x=1$, $a_{dxdt}=1$
`rosenbrock4`	`rosenbrock4_controlled< rosenbrock4 >`	-

Table 8. Generation functions make_dense_output(abs_error , rel_error , stepper)

Stepper	Result of make_dense_output	Remarks
`runge_kutta_dopri5`	`dense_output_runge_kutta< controlled_runge_kutta< runge_kutta_dopri5 , default_error_checker<...> > >`	$a_x=1$, $a_{dxdt}=1$
`rosenbrock4`	`rosenbrock4_dense_output< rosenbrock4_controller< rosenbrock4 > >`	-

Integrate functions

Integrate functions perform the time evolution of a given ODE from some starting time t_0 to a given end time t_1 and starting at state x_0 by subsequent calls of a given stepper's `do_step` function. Additionally, the user can provide an __observer to analyze the state during time evolution. There are five different integrate functions which have different strategies on when to call the observer function during integration. All of the integrate functions except `integrate_n_steps` can be called with any stepper following one of the stepper concepts: Stepper , Error Stepper , Controlled Stepper , Dense Output Stepper. Depending on the abilities of the stepper, the integrate functions make use of step-size control or dense output.

Equidistant observer calls

If observer calls at equidistant time intervals dt are needed, the `integrate_const` or `integrate_n_steps` function should be used. We start with explaining `integrate_const`:

```
integrate_const( stepper , system , x0 , t0 , t1 , dt )

integrate_const( stepper , system , x0 , t0 , t1 , dt , observer )
```

These integrate the ODE given by `system` with subsequent steps from `stepper`. Integration start at `t0` and `x0` and ends at some $t' = t_0 + n dt$ with n such that $t_1 - dt < t' <= t_1$. `x0` is changed to the approximative solution $x(t')$ at the end of integration. If provided, the `observer` is invoked at times t_0, $t_0 + dt$, $t_0 + 2dt$, ... ,t'. `integrate_const` returns the number of steps performed during the integration. Note that if you are using a simple Stepper or Error Stepper and want to make exactly `n` steps you should prefer the `integrate_n_steps` function below.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- If `stepper` is a Stepper or Error Stepper then `dt` is also the step size used for integration and the observer is called just after every step.
- If `stepper` is a Controlled Stepper then `dt` is the initial step size. The actual step size will change due to error control during time evolution. However, if an observer is provided the step size will be adjusted such that the algorithm always calculates $x(t)$ at $t = t_0 + n\ dt$ and calls the observer at that point. Note that the use of Controlled Stepper is reasonable here only if `dt` is considerably larger than typical step sizes used by the stepper.
- If `stepper` is a Dense Output Stepper then `dt` is the initial step size. The actual step size will be adjusted during integration due to error control. If an observer is provided dense output is used to calculate $x(t)$ at $t = t_0 + n\ dt$.

Integrate a given number of steps

This function is very similar to `integrate_const` above. The only difference is that it does not take the end time as parameter, but rather the number of steps. The integration is then performed until the time `t0+n*dt`.

```
integrate_n_steps( stepper , system , x0 , t0 , dt , n )
```

```
integrate_n_steps( stepper , system , x0 , t0 , dt , n , observer )
```

Integrates the ODE given by `system` with subsequent steps from `stepper` starting at x_0 and t_0. If provided, `observer` is called after every step and at the beginning with `t0`, similar as above. The approximate result for $x(t_0 + n\ dt)$ is stored in `x0`. This function returns the end time `t0 + n*dt`.

Observer calls at each step

If the observer should be called at each time step then the `integrate_adaptive` function should be used. Note that in the case of Controlled Stepper or Dense Output Stepper this leads to non-equidistant observer calls as the step size changes.

```
integrate_adaptive( stepper , system , x0 , t0 , t1 , dt )
```

```
integrate_adaptive( stepper , system , x0 , t0 , t1 , dt , observer )
```

Integrates the ODE given by `system` with subsequent steps from `stepper`. Integration start at `t0` and `x0` and ends at t_1. `x0` is changed to the approximative solution $x(t_1)$ at the end of integration. If provided, the `observer` is called after each step (and before the first step at `t0`). `integrate_adaptive` returns the number of steps performed during the integration.

- If `stepper` is a Stepper or Error Stepper then `dt` is the step size used for integration and `integrate_adaptive` behaves like `integrate_const` except that for the last step the step size is reduced to ensure we end exactly at `t1`. If provided, the observer is called at each step.
- If `stepper` is a Controlled Stepper then `dt` is the initial step size. The actual step size is changed according to error control of the stepper. For the last step, the step size will be reduced to ensure we end exactly at `t1`. If provided, the observer is called after each time step (and before the first step at `t0`).
- If stepper is a Dense Output Stepper then `dt` is the initial step size and `integrate_adaptive` behaves just like for Controlled Stepper above. No dense output is used.

Observer calls at given time points

If the observer should be called at some user given time points the `integrate_times` function should be used. The times for observer calls are provided as a sequence of time values. The sequence is either defined via two iterators pointing to begin and end of the sequence or in terms of a Boost.Range object.

```
integrate_times( stepper , system , x0 , times_start , times_end , dt , observer )
```

```
integrate_times( stepper , system , x0 , time_range , dt , observer )
```

Integrates the ODE given by `system` with subsequent steps from `stepper`. Integration starts at `*times_start` and ends exactly at `*(times_end-1)`. `x0` contains the approximate solution at the end point of integration. This function requires an observer which

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

is invoked at the subsequent times `*times_start++` until `times_start == times_end`. If called with a Boost.Range `time_range` the function behaves the same with `times_start = boost::begin( time_range )` and `times_end = boost::end( time_range )`. `integrate_times` returns the number of steps performed during the integration.

- If `stepper` is a Stepper or Error Stepper `dt` is the step size used for integration. However, whenever a time point from the sequence is approached the step size `dt` will be reduced to obtain the state *x(t)* exactly at the time point.
- If `stepper` is a Controlled Stepper then `dt` is the initial step size. The actual step size is adjusted during integration according to error control. However, if a time point from the sequence is approached the step size is reduced to obtain the state *x(t)* exactly at the time point.
- If `stepper` is a Dense Output Stepper then `dt` is the initial step size. The actual step size is adjusted during integration according to error control. Dense output is used to obtain the states *x(t)* at the time points from the sequence.

Convenience integrate function

Additionally to the sophisticated integrate function above odeint also provides a simple `integrate` routine which uses a dense output stepper based on `runge_kutta_dopri5` with standard error bounds 10^{-6} for the steps.

`integrate( system , x0 , t0 , t1 , dt )`

`integrate( system , x0 , t0 , t1 , dt , observer )`

This function behaves exactly like `integrate_adaptive` above but no stepper has to be provided. It also returns the number of steps performed during the integration.

Iterators and Ranges

Examples

odeint supports iterators that iterate along an approximate solution of an ordinary differential equation. Iterators offer you an alternative to the integrate functions. Furthermore, many of the standard algorithms in the C++ standard library and Boost.Range can be used with the odeint's iterators.

Several iterator types are provided, in consistence with the integrate functions. Hence there are `const_step_iterator`, `adaptive_step_iterator`, `n_step_iterator` and `times_iterator` -- each of them in two versions: either with only the `state` or with a `std::pair<state,time>` as value type. They are all single pass iterators. In the following, we show a few examples of how to use those iterators together with std algorithms.

```
runge_kutta4< state_type > stepper;
state_type x = {{ 10.0 , 10.0 , 10.0 }};
double res = std::accumulate( make_const_step_iterator_begin( step↵
per , lorenz() , x , 0.0 , 1.0 , 0.01 ) ,
                              make_const_step_iterator_end( stepper , lorenz() , x ) ,
                              0.0 ,
                              []( double sum , const state_type &x ) {
                                  return sum + x[0]; } );
cout << res << endl;
```

In this example all x-values of the solution are accumulated. Note, how dereferencing the iterator gives the current state `x` of the ODE (the second argument of the accumulate lambda). The iterator itself does not occur directly in this example but it is generated by the factory functions `make_const_step_iterator_begin` and `make_const_step_iterator_end`. odeint also supports Boost.Range, that allows to write the above example in a more compact form with the factory function `make_const_step_range`, but now using `boost::accumulate` from __bost_range:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
runge_kutta4< state_type > stepper;
state_type x = {{ 10.0 , 10.0 , 10.0 }};
double res = boost::accumulate( make_const_step_range( step↲
per , lorenz() , x , 0.0 , 1.0 , 0.01 ) , 0.0 ,
                                [](  double sum , const state_type &x ) {
                                    return sum + x[0]; } );
cout << res << endl;
```

The second iterator type is also a iterator with const step size. But the value type of this iterator consists here of a pair of the time and the state of the solution of the ODE. An example is

```
runge_kutta4< state_type > stepper;
state_type x = {{ 10.0 , 10.0 , 10.0 }};
double res = boost::accumulate( make_const_step_time_range( step↲
per , lorenz() , x , 0.0 , 1.0 , 0.01 ) , 0.0 ,
                               ↲
 []( double sum , const std::pair< const state_type &, double > &x ) {
                                    return sum + x.first[0]; } );
cout << res << endl;
```

The factory functions are now `make_const_step_time_iterator_begin`, `make_const_step_time_iterator_end` and `make_const_step_time_range`. Note, how the lambda now expects a `std::pair` as this is the value type of the `const_step_time_iterator`'s.

Next, we discuss the adaptive iterators which are completely analogous to the const step iterators, but are based on adaptive stepper routines and thus adjust the step size during the iteration. Examples are

```
auto stepper = make_controlled( 1.0e-6 , 1.0e-6 , runge_kutta_cash_karp54< state_type >() );
state_type x = {{ 10.0 , 10.0 , 10.0 }};
double res = boost::accumulate( make_adaptive_range( step↲
per , lorenz() , x , 0.0 , 1.0 , 0.01 ) , 0.0 ,
                                []( double sum , const state_type& x ) {
                                    return sum + x[0]; } );
cout << res << endl;
```

```
auto stepper = make_controlled( 1.0e-6 , 1.0e-6 , runge_kutta_cash_karp54< state_type >() );
state_type x = {{ 10.0 , 10.0 , 10.0 }};
double res = boost::accumulate( make_adaptive_time_range( step↲
per , lorenz() , x , 0.0 , 1.0 , 0.01 ) , 0.0 ,
                               []( double sum , const pair< const state_type& , double > &x ) {
                                    return sum + x.first[0]; } );
cout << res << endl;
```

Note

'adaptive_iterator `and` adaptive_time_iterator' can only be used with Controlled Stepper or Dense Output Stepper.

In general one can say that iterating over a range of a `const_step_iterator` behaves like an `integrate_const` function call, and similarly for `adaptive_iterator` and `integrate_adaptive`, `n_step_iterator` and `integrate_n_steps`, and finally `times_iterator` and `integrate_times`.

Below we list the most important properties of the exisiting iterators:

const_step_iterator

- Definition: `const_step_iterator< Stepper , System , State >`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- `value_type` is `State`
- `reference_type` is `State const&`
- Factory functions
  - `make_const_step_iterator_begin( stepper , system , state , t_start , t_end , dt )`
  - `make_const_step_iterator_end( stepper , system , state )`
  - `make_const_step_range( stepper , system , state , t_start , t_end , dt )`
- This stepper works with all steppers fulfilling the Stepper concept or the DenseOutputStepper concept.
- The value of `state` is the current state of the ODE during the iteration.

const_step_time_iterator

- Definition: `const_step_time_iterator< Stepper , System , State >`
- `value_type` is `std::pair< State , Stepper::time_type >`
- `reference_type` is `std::pair< State const& , Stepper::time_type > const&`
- Factory functions
  - `make_const_step_time_iterator_begin( stepper , system , state , t_start , t_end , dt )`
  - `make_const_step_time_iterator_end( stepper , system , state )`
  - `make_const_step_time_range( stepper , system , state , t_start , t_end , dt )`
- This stepper works with all steppers fulfilling the Stepper concept or the DenseOutputStepper concept.
- This stepper updates the value of `state`. The value of `state` is the current state of the ODE during the iteration.

adaptive_step_iterator

- Definition: `adaptive_iterator< Stepper , System , State >`
- `value_type` is `State`
- `reference_type` is `State const&`
- Factory functions
  - `make_adaptive_iterator_begin( stepper , system , state , t_start , t_end , dt )`
  - `make_adaptive_iterator_end( stepper , system , state )`
  - `make_adaptive_range( stepper , system , state , t_start , t_end , dt )`
- This stepper works with all steppers fulfilling the ControlledStepper concept or the DenseOutputStepper concept.
- For steppers fulfilling the ControlledStepper concept `state` is modified according to the current state of the ODE. For DenseOutputStepper the state is not modified due to performance optimizations, but the steppers itself.

adaptive_step_time_iterator

- Definition: `adaptive_iterator< Stepper , System , State >`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- `value_type` is `std::pair< State , Stepper::time_type >`
- `reference_type` is `std::pair< State const& , Stepper::time_type > const&`
- Factory functions
  - `make_adaptive_time_iterator_begin( stepper , system , state , t_start , t_end , dt )`
  - `make_adaptive_time_iterator_end( stepper , system , state )`
  - `make_adaptive_time_range( stepper , system , state , t_start , t_end , dt )`
- This stepper works with all steppers fulfilling the ControlledStepper concept or the DenseOutputStepper concept.
- For steppers fulfilling the ControlledStepper concept `state` is modified according to the current state of the ODE. For DenseOutputStepper the state is not modified due to performance optimizations, but the stepper itself.

n_step_iterator

- Definition: `n_step_iterator< Stepper , System , State >`
- `value_type` is `State`
- `reference_type` is `State const&`
- Factory functions
  - `make_n_step_iterator_begin( stepper , system , state , t_start , dt , num_of_steps )`
  - `make_n_step_iterator_end( stepper , system , state )`
  - `make_n_step_range( stepper , system , state , t_start , dt , num_of_steps )`
- This stepper works with all steppers fulfilling the Stepper concept or the DenseOutputStepper concept.
- The value of `state` is the current state of the ODE during the iteration.

n_step_time_iterator

- Definition: `n_step_time_iterator< Stepper , System , State >`
- `value_type` is `std::pair< State , Stepper::time_type >`
- `reference_type` is `std::pair< State const& , Stepper::time_type > const&`
- Factory functions
  - `make_n_step_time_iterator_begin( stepper , system , state , t_start , dt , num_of_steps )`
  - `make_n_step_time_iterator_end( stepper , system , state )`
  - `make_n_step_time_range( stepper , system , state , t_start , dt , num_of_steps )`
- This stepper works with all steppers fulfilling the Stepper concept or the DenseOutputStepper concept.
- This stepper updates the value of `state`. The value of `state` is the current state of the ODE during the iteration.

times_iterator

- Definition: `times_iterator< Stepper , System , State , TimeIterator >`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- `value_type` is `State`
- `reference_type` is `State const&`
- Factory functions
  - `make_times_iterator_begin( stepper , system , state , t_start , t_end , dt )`
  - `make_times_iterator_end( stepper , system , state )`
  - `make_times_range( stepper , system , state , t_start , t_end , dt )`
- This stepper works with all steppers fulfilling the Stepper concept, the ControlledStepper concept or the DenseOutputStepper concept.
- The value of `state` is the current state of the ODE during the iteration.

times_time_iterator

- Definition: `times_time_iterator< Stepper , System , State , TimeIterator>`
- `value_type` is `std::pair< State , Stepper::time_type >`
- `reference_type` is `std::pair< State const& , Stepper::time_type > const&`
- Factory functions
  - `make_times_time_iterator_begin( stepper , system , state , t_start , t_end , dt )`
  - `make_times_time_step_iterator_end( stepper , system , state )`
  - `make_times_time_range( stepper , system , state , t_start , t_end , dt )`
- This stepper works with all steppers fulfilling the Stepper concept, the ControlledStepper concept or the DenseOutputStepper concept.
- This stepper updates the value of `state`. The value of `state` is the current state of the ODE during the iteration.

State types, algebras and operations

In odeint the stepper algorithms are implemented independently of the underlying fundamental mathematical operations. This is realized by giving the user full control over the state type and the mathematical operations for this state type. Technically, this is done by introducing three concepts: StateType, Algebra, Operations. Most of the steppers in odeint expect three class types fulfilling these concepts as template parameters. Note that these concepts are not fully independent of each other but rather a valid combination must be provided in order to make the steppers work. In the following we will give some examples on reasonable state_type-algebra-operations combinations. For the most common state types, like `vector<double>` or `array<double,N>` the default values range_algebra and default_operations are perfectly fine and odeint can be used as is without worrying about algebra/operations at all.

Important

state_type, algebra and operations are not independent, a valid combination must be provided to make odeint work properly

Moreover, as odeint handles the memory required for intermediate temporary objects itself, it also needs knowledge about how to create state_type objects and maybe how to allocate memory (resizing). All in all, the following things have to be taken care of when odeint is used with non-standard state types:

- construction/destruction

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- resizing (if possible/required)
- algebraic operations

Again, odeint already provides basic interfaces for most of the usual state types. So if you use a `std::vector`, or a `boost::array` as state type no additional work is required, they just work out of the box.

Construction/Resizing

We distinguish between two basic state types: fixed sized and dynamically sized. For fixed size state types the default constructor `state_type()` already allocates the required memory, prominent example is `boost::array<T,N>`. Dynamically sized types have to be resized to make sure enough memory is allocated, the standard constructor does not take care of the resizing. Examples for this are the STL containers like `vector<double>`.

The most easy way of getting your own state type to work with odeint is to use a fixed size state, base calculations on the range_algebra and provide the following functionality:

Name	**Expression**	**Type**	**Semantics**
Construct State	`State x()`	`void`	Creates an instance of `State` and allocates memory.
Begin of the sequence	boost::begin(x)	Iterator	Returns an iterator pointing to the begin of the sequence
End of the sequence	boost::end(x)	Iterator	Returns an iterator pointing to the end of the sequence

Warning

If your state type does not allocate memory by default construction, you **must define it as resizeable** and provide resize functionality (see below). Otherwise segmentation faults will occur.

So fixed sized arrays supported by Boost.Range immediately work with odeint. For dynamically sized arrays one has to additionally supply the resize functionality. First, the state has to be tagged as resizeable by specializing the struct `is_resizeable` which consists of one typedef and one bool value:

Name	**Expression**	**Type**	**Semantics**
Resizability	`is_resizeable<State>::type`	`boost::true_type` or `boost::false_type`	Determines resizeability of the state type, returns `boost::true_type` if the state is resizeable.
Resizability	`is_resizeable<State>::value`	`bool`	Same as above, but with `bool` value.

Defining `type` to be `true_type` and `value` as `true` tells odeint that your state is resizeable. By default, odeint now expects the support of `boost::size(x)` and a `x.resize( boost::size(y) )` member function for resizing:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Expression	Type	Semantics
Get size	`boost::size( x )`	`size_type`	Returns the current size of x.
Resize	`x.resize( boost::size( y ) )`	`void`	Resizes x to have the same size as y.

Using the container interface

As a first example we take the most simple case and implement our own vector `my_vector` which will provide a container interface. This makes Boost.Range working out-of-box. We add a little functionality to our vector which makes it allocate some default capacity by construction. This is helpful when using resizing as then a resize can be assured to not require a new allocation.

```
template< int MAX_N >
class my_vector
{
    typedef std::vector< double > vector;

public:
    typedef vector::iterator iterator;
    typedef vector::const_iterator const_iterator;

public:
    my_vector( const size_t N )
        : m_v( N )
    {
        m_v.reserve( MAX_N );
    }

    my_vector()
        : m_v()
    {
        m_v.reserve( MAX_N );
    }

// ... [ implement container interface ]
```

The only thing that has to be done other than defining is thus declaring my_vector as resizeable:

```
// define my_vector as resizeable

namespace boost { namespace numeric { namespace odeint {

template<size_t N>
struct is_resizeable< my_vector<N> >
{
    typedef boost::true_type type;
    static const bool value = type::value;
};

} } }
```

If we wouldn't specialize the `is_resizeable` template, the code would still compile but odeint would not adjust the size of temporary internal instances of my_vector and hence try to fill zero-sized vectors resulting in segmentation faults! The full example can be found in my_vector.cpp

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

std::list

If your state type does work with Boost.Range, but handles resizing differently you are required to specialize two implementations used by odeint to check a state's size and to resize:

Name	Expression	Type	Semantics
Check size	`same_size_impl<State,State>::same_size(x , y)`	`bool`	Returns true if the size of x equals the size of y.
Resize	`resize_impl<State,State>::resize(x , y)`	`void`	Resizes x to have the same size as y.

As an example we will use a `std::list` as state type in odeint. Because `std::list` is not supported by `boost::size` we have to replace the same_size and resize implementation to get list to work with odeint. The following code shows the required template specializations:

```
typedef std::list< double > state_type;

namespace boost { namespace numeric { namespace odeint {

template< >
struct is_resizeable< state_type >
{ // declare resizeability
    typedef boost::true_type type;
    const static bool value = type::value;
};

template< >
struct same_size_impl< state_type , state_type >
{ // define how to check size
    static bool same_size( const state_type &v1 ,
                           const state_type &v2 )
    {
        return v1.size() == v2.size();
    }
};

template< >
struct resize_impl< state_type , state_type >
{ // define how to resize
    static void resize( state_type &v1 ,
                        const state_type &v2 )
    {
        v1.resize( v2.size() );
    }
};

} } }
```

With these definitions odeint knows how to resize `std::lists` and so they can be used as state types. A complete example can be found in list_lattice.cpp.

Algebras and Operations

To provide maximum flexibility odeint is implemented in a highly modularized way. This means it is possible to change the underlying mathematical operations without touching the integration algorithms. The fundamental mathematical operations are those of

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

a vector space, that is addition of `state_types` and multiplication of `state_types` with a scalar (`time_type`). In odeint this is realized in two concepts: Algebra and Operations. The standard way how this works is by the range algebra which provides functions that apply a specific operation to each of the individual elements of a container based on the Boost.Range library. If your state type is not supported by Boost.Range there are several possibilities to tell odeint how to do algebraic operations:

- Implement `boost::begin` and `boost::end` for your state type so it works with Boost.Range.
- Implement vector-vector addition operator + and scalar-vector multiplication operator * and use the non-standard `vector_space_algebra`.
- Implement your own algebra that implements the required functions.

GSL Vector

In the following example we will try to use the `gsl_vector` type from GSL (GNU Scientific Library) as state type in odeint. We will realize this by implementing a wrapper around the gsl_vector that takes care of construction/destruction. Also, Boost.Range is extended such that it works with `gsl_vectors` as well which required also the implementation of a new `gsl_iterator`.

Note

odeint already includes all the code presented here, see gsl_wrapper.hpp, so `gsl_vectors` can be used straight out-of-box. The following description is just for educational purpose.

The GSL is a C library, so `gsl_vector` has neither constructor, nor destructor or any `begin` or `end` function, no iterators at all. So to make it work with odeint plenty of things have to be implemented. Note that all of the work shown here is already included in odeint, so using `gsl_vectors` in odeint doesn't require any further adjustments. We present it here just as an educational example. We start with defining appropriate constructors and destructors. This is done by specializing the `state_wrapper` for `gsl_vector`. State wrappers are used by the steppers internally to create and manage temporary instances of state types:

```
template<>
struct state_wrapper< gsl_vector* >
{
    typedef double value_type;
    typedef gsl_vector* state_type;
    typedef state_wrapper< gsl_vector* > state_wrapper_type;

    state_type m_v;

    state_wrapper( )
    {
        m_v = gsl_vector_alloc( 1 );
    }

    state_wrapper( const state_wrapper_type &x )
    {
        resize( m_v , x.m_v );
        gsl_vector_memcpy( m_v , x.m_v );
    }


    ~state_wrapper()
    {
        gsl_vector_free( m_v );
    }

};
```

This `state_wrapper` specialization tells odeint how gsl_vectors are created, copied and destroyed. Next we need resizing, this is required because gsl_vectors are dynamically sized objects:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<>
struct is_resizeable< gsl_vector* >
{
    typedef boost::true_type type;
    const static bool value = type::value;
};

template <>
struct same_size_impl< gsl_vector* , gsl_vector* >
{
    static bool same_size( const gsl_vector* x , const gsl_vector* y )
    {
        return x->size == y->size;
    }
};

template <>
struct resize_impl< gsl_vector* , gsl_vector* >
{
    static void resize( gsl_vector* x , const gsl_vector* y )
    {
        gsl_vector_free( x );
        x = gsl_vector_alloc( y->size );
    }
};
```

Up to now, we defined creation/destruction and resizing, but gsl_vectors also don't support iterators, so we first implement a gsl iterator:

```
/*
 * defines an iterator for gsl_vector
 */
class gsl_vector_iterator
      : public boost::iterator_facade< gsl_vector_iterator , double ,
                                       boost::random_access_traversal_tag >
{
public :

    gsl_vector_iterator( void ): m_p(0) , m_stride( 0 ) { }
    explicit gsl_vector_iterator( gsl_vector *p ) : m_p( p->data ) , m_stride( p->stride ) { }
    friend gsl_vector_iterator end_iterator( gsl_vector * );

private :

    friend class boost::iterator_core_access;
    friend class const_gsl_vector_iterator;

    void increment( void ) { m_p += m_stride; }
    void decrement( void ) { m_p -= m_stride; }
    void advance( ptrdiff_t n ) { m_p += n*m_stride; }
    bool equal( const gsl_vector_iterator &other ) const { return this->m_p == other.m_p; }
    bool equal( const const_gsl_vector_iterator &other ) const;
    double& dereference( void ) const { return *m_p; }

    double *m_p;
    size_t m_stride;
};
```

A similar class exists for the `const` version of the iterator. Then we have a function returning the end iterator (similarly for `const` again):

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
gsl_vector_iterator end_iterator( gsl_vector *x )
{
    gsl_vector_iterator iter( x );
    iter.m_p += iter.m_stride * x->size;
    return iter;
}
```

Finally, the bindings for Boost.Range are added:

```
// template<>
inline gsl_vector_iterator range_begin( gsl_vector *x )
{
    return gsl_vector_iterator( x );
}

// template<>
inline gsl_vector_iterator range_end( gsl_vector *x )
{
    return end_iterator( x );
}
```

Again with similar definitions for the `const` versions. This eventually makes odeint work with gsl vectors as state types. The full code for these bindings is found in gsl_wrapper.hpp. It might look rather complicated but keep in mind that gsl is a pre-compiled C library.

Vector Space Algebra

As seen above, the standard way of performing algebraic operations on container-like state types in odeint is to iterate through the elements of the container and perform the operations element-wise on the underlying value type. This is realized by means of the `range_algebra` that uses Boost.Range for obtaining iterators of the state types. However, there are other ways to implement the algebraic operations on containers, one of which is defining the addition/multiplication operators for the containers directly and then using the `vector_space_algebra`. If you use this algebra, the following operators have to be defined for the state_type:

Name	Expression	Type	Semantics
Addition	`x + y`	`state_type`	Calculates the vector sum 'x+y'.
Assign addition	`x += y`	`state_type`	Performs x+y in place.
Scalar multiplication	`a * x`	`state_type`	Performs multiplication of vector x with scalar a.
Assign scalar multiplication	`x *= a`	`state_type`	Performs in-place multiplication of vector x with scalar a.

Defining these operators makes your state type work with any basic Runge-Kutta stepper. However, if you want to use step-size control, some more functionality is required. Specifically, operations like $max_i(|err_i| / (alpha * |s_i|))$ have to be performed. *err* and *s* are state_types, alpha is a scalar. As you can see, we need element wise absolute value and division as well as an reduce operation to get the maximum value. So for controlled steppers the following things have to be implemented:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	**Expression**	**Type**	**Semantics**
Division	`x / y`	`state_type`	Calculates the element-wise division 'x/y'
Absolute value	`abs( x )`	`state_type`	Element wise absolute value

Reduce	`vector_space_reduce_impl< state_type >::reduce( state , operation , init )`	`value_type`	Performs the `operation` for subsequently each element of `state` and returns the aggregate value. E.g.
			`init = operator( init , state[0] );`
			`init = operator( init , state[1] )`
			`...`

Point type

Here we show how to implement the required operators on a state type. As example we define a new class `point3D` representing a three-dimensional vector with components x,y,z and define addition and scalar multiplication operators for it. We use Boost.Operators to reduce the amount of code to be written. The class for the point type looks as follows:

```
class point3D :
    boost::additive1< point3D ,
    boost::additive2< point3D , double ,
    boost::multiplicative2< point3D , double > > >
{
public:

    double x , y , z;

    point3D()
        : x( 0.0 ) , y( 0.0 ) , z( 0.0 )
    { }

    point3D( const double val )
        : x( val ) , y( val ) , z( val )
    { }

    point3D( const double _x , const double _y , const double _z  )
        : x( _x ) , y( _y ) , z( _z )
    { }

    point3D& operator+=( const point3D &p )
    {
        x += p.x; y += p.y; z += p.z;
        return *this;
    }

    point3D& operator*=( const double a )
    {
        x *= a; y *= a; z *= a;
        return *this;
    }

};
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

By deriving from Boost.Operators classes we don't have to define outer class operators like `operator+( point3D , point3D )` because that is taken care of by the operators library. Note that for simple Runge-Kutta schemes (like `runge_kutta4`) only the `+` and `*` operators are required. If, however, a controlled stepper is used one also needs to specify the division operator `/` because calculation of the error term involves an element wise division of the state types. Additionally, controlled steppers require an `abs` function calculating the element-wise absolute value for the state type:

```
// only required for steppers with error control
point3D operator/( const point3D &p1 , const point3D &p2 )
{
    return point3D( p1.x/p2.x , p1.y/p2.y , p1.z/p1.z );
}

point3D abs( const point3D &p )
{
    return point3D( std::abs(p.x) , std::abs(p.y) , std::abs(p.z) );
}
```

Finally, we have to provide a specialization to calculate the infintity norm of a state:

```
// also only for steppers with error control
namespace boost { namespace numeric { namespace odeint {
template<>
struct vector_space_norm_inf< point3D >
{
    typedef double result_type;
    double operator()( const point3D &p ) const
    {
        using std::max;
        using std::abs;
        return max( max( abs( p.x ) , abs( p.y ) ) , abs( p.z ) );
    }
};
} } }
```

Again, note that the two last steps were only required if you want to use controlled steppers. For simple steppers definition of the simple `+=` and `*=` operators are sufficient. Having defined such a point type, we can easily perform the integration on a Lorenz system by explicitely configuring the `vector_space_algebra` in the stepper's template argument list:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
const double sigma = 10.0;
const double R = 28.0;
const double b = 8.0 / 3.0;

void lorenz( const point3D &x , point3D &dxdt , const double t )
{
    dxdt.x = sigma * ( x.y - x.x );
    dxdt.y = R * x.x - x.y - x.x * x.z;
    dxdt.z = -b * x.z + x.x * x.y;
}

using namespace boost::numeric::odeint;

int main()
{

    point3D x( 10.0 , 5.0 , 5.0 );
    // point type defines it's own operators -> use vector_space_algebra !
    typedef runge_kutta_dopri5< point3D , double , point3D ,
                                double , vector_space_algebra > stepper;
    int steps = integrate_adaptive( make_controlled<stepper>( 1E-10 , 1E-10 ) , lorenz , x ,
                                    0.0 , 10.0 , 0.1 );
    std::cout << x << std::endl;
    std::cout << "steps: " << steps << std::endl;
}
```

The whole example can be found in lorenz_point.cpp

Note

For the most `state_types`, odeint is able to automatically determine the correct algebra and operations. But if you want to use your own `state_type`, as in this example with `point3D`, you have to manually configure the right algebra/operations, unless your `state_type` works with the default choice of `range_algebra` and `default_operations`.

gsl_vector, gsl_matrix, ublas::matrix, blitz::matrix, thrust

Adapt your own operations

to be continued

- thrust
- gsl_complex
- min, max, pow

Using boost::ref

In odeint all system functions and observers are passed by value. For example, if you call a `do_step` method of a particular stepper or the integration functions, your system and your stepper will be passed by value:

```
rk4.do_step( sys , x , t , dt );   // pass sys by value
```

This behavior is suitable for most systems, especially if your system does not contain any data or only a few parameters. However, in some cases you might contain some large amount of data with you system function and passing them by value is not desired since the data would be copied.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

In such cases you can easily use `boost::ref` (and its relative `boost::cref`) which passes its argument by reference (or constant reference). odeint will unpack the arguments and no copying at all of your system object will take place:

```
rk4.do_step( boost::ref( sys ) , x , t , dt );   // pass sys as references
```

The same mechanism can be used for the observers in the integrate functions.

Tip

If you are using C++11 you can also use `std::ref` and `std::cref`

Using boost::range

Most steppers in odeint also accept the state give as a range. A range is sequence of values modeled by a range concept. See Boost.Range for an overview over existing concepts and examples of ranges. This means that the `state_type` of the stepper need not necessarily be used to call the `do_step` method.

One use-case for Boost.Range in odeint has been shown in Chaotic System where the state consists of two parts: one for the original system and one for the perturbations. The ranges are used to initialize (solve) only the system part where the perturbation part is not touched, that is a range consisting only of the system part is used. After that the complete state including the perturbations is solved.

Another use case is a system consisting of coupled units where you want to initialize each unit separately with the ODE of the uncoupled unit. An example is a chain of coupled van-der-Pol-oscillators which are initialized uniformly from the uncoupled van-der-Pol-oscillator. Then you can use Boost.Range to solve only one individual oscillator in the chain.

In short, you can Boost.Range to use one state within two system functions which expect states with different sizes.

An example was given in the Chaotic System tutorial. Using Boost.Range usually means that your system function needs to adapt to the iterators of Boost.Range. That is, your function is called with a range and you need to get the iterators from that range. This can easily be done. You have to implement your system as a class or a struct and you have to templatize the `operator()`. Then you can use the `range_iterator`-meta function and `boost::begin` and `boost::end` to obtain the iterators of your range:

```
class sys
{
    template< class State , class Deriv >
    void operator()( const State &x_ , Deriv &dxdt_ , double t ) const
    {
        typename boost::range_iterator< const State >::type x = boost::begin( x_ );
        typename boost::range_iterator< Deriv >::type dxdt = boost::begin( dxdt_ );

        // fill dxdt
    }
};
```

If your range is a random access-range you can also apply the bracket operator to the iterator to access the elements in the range:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
class sys
{
    template< class State , class Deriv >
    void operator()( const State &x_ , Deriv &dxdt_ , double t ) const
    {
         typename boost::range_iterator< const State >::type x = boost::begin( x_ );
         typename boost::range_iterator< Deriv >::type dxdt = boost::begin( dxdt_ );

         dxdt[0] = f1( x[0] , x[1] );
         dxdt[1] = f2( x[0] , x[1] );
    }
};
```

The following two tables show which steppers and which algebras are compatible with Boost.Range.

Table 9. Steppers supporting Boost.Range

Stepper
adams_bashforth_moulton
bulirsch_stoer_dense_out
bulirsch_stoer
controlled_runge_kutta
dense_output_runge_kutta
euler
explicit_error_generic_rk
explicit_generic_rk
rosenbrock4_controller
rosenbrock4_dense_output
rosenbrock4
runge_kutta4_classic
runge_kutta4
runge_kutta_cash_karp54_classic
runge_kutta_cash_karp54
runge_kutta_dopri5
runge_kutta_fehlberg78
symplectic_euler
symplectic_rkn_sb3a_mclachlan

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 10. Algebras supporting Boost.Range

algebra
range_algebra
thrust_algebra

Binding member functions

Binding member functions to a function objects suitable for odeint system function is not easy, at least in C++03. The usual way of using __boost_bind does not work because of the forwarding problem. odeint provides two `do_step` method which only differ in the const specifiers of the arguments and __boost_bind binders only provide the specializations up to two argument which is not enough for odeint.

But one can easily implement the according binders themself:

```
template< class Obj , class Mem >
class ode_wrapper
{
    Obj m_obj;
    Mem m_mem;

public:

    ode_wrapper( Obj obj , Mem mem ) : m_obj( obj ) , m_mem( mem ) { }

    template< class State , class Deriv , class Time >
    void operator()( const State &x , Deriv &dxdt , Time t )
    {
        (m_obj.*m_mem)( x , dxdt , t );
    }
};

template< class Obj , class Mem >
ode_wrapper< Obj , Mem > make_ode_wrapper( Obj obj , Mem mem )
{
    return ode_wrapper< Obj , Mem >( obj , mem );
}
```

One can use this binder as follows

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
struct lorenz
{
    void ode( const state_type &x , state_type &dxdt , double t ) const
    {
        dxdt[0] = 10.0 * ( x[1] - x[0] );
        dxdt[1] = 28.0 * x[0] - x[1] - x[0] * x[2];
        dxdt[2] = -8.0 / 3.0 * x[2] + x[0] * x[1];
    }
};

int main( int argc , char *argv[] )
{
    using namespace boost::numeric::odeint;
    state_type x = {{ 10.0 , 10.0 , 10.0 }};
   integrate_const( runge_kutta4< state_type >() , make_ode_wrapper( lorenz() , &lorenz::ode ) ,
                    x , 0.0 , 10.0 , 0.01 );
    return 0;
}
```

Binding member functions in C++11

In C++11 one can use `std::bind` and one does not need to implement the bind themself:

```
namespace pl = std::placeholders;

state_type x = {{ 10.0 , 10.0 , 10.0 }};
integrate_const( runge_kutta4< state_type >() ,
                 std::bind( &lorenz::ode , lorenz() , pl::_1 , pl::_2 , pl::_3 ) ,
                 x , 0.0 , 10.0 , 0.01  );
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Concepts

System

Description

The System concept models the algorithmic implementation of the rhs. of the ODE $x' = f(x,t)$. The only requirement for this concept is that it should be callable with a specific parameter syntax (see below). A System is typically implemented as a function or a functor. Systems fulfilling this concept are required by all Runge-Kutta steppers as well as the Bulirsch-Stoer steppers. However, symplectic and implicit steppers work with other system concepts, see Symplectic System and Implicit System.

Notation

`System`	A type that is a model of System
`State`	A type representing the state x of the ODE
`Deriv`	A type representing the derivative x' of the ODE
`Time`	A type representing the time
`sys`	An object of type `System`
`x`	Object of type `State`
`dxdt`	Object of type `Deriv`
`t`	Object of type `Time`

Valid expressions

Name	Expression	Type	Semantics
Calculate $dx/dt := f(x,t)$	`sys( x , dxdt , t )`	`void`	Calculates f(x,t), the result is stored into dxdt

Second Order System

Description

The Second Order System concept models the algorithmic implementation of the rhs for steppers requirering the second order derivative, hence the r.h.s. of the ODE $x'' = f(x,x',t)$. The only requirement for this concept is that it should be callable with a specific parameter syntax (see below). A Second Order System is typically implemented as a function or a functor. Systems fulfilling this concept are required by the Velocity Verlet method.

Notation

`System`	A type that is a model of Second Order System
`Space`	A type representing the state x of the ODE
`Velocity`	A type representing the derivative x' of the ODE
`Acceleration`	A type representing the second order derivative x'' of the ODE
`Time`	A type representing the time

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

`sys`	An object of type `System`
`x`	Object of type `Space`
`v`	Object of type `Velocity`
`a`	Object of type `Acceleration`
`t`	Object of type `Time`

Valid expressions

Name	Expression	Type	Semantics
Calculate $x'' := f(x,x',t)$	`sys( x , v , a , t )`	`void`	Calculates f(x,x',t), the result is stored into a.

Symplectic System

Description

This concept describes how to define a symplectic system written with generalized coordinate `q` and generalized momentum `p`:

$q'(t) = f(p)$

$p'(t) = g(q)$

Such a situation is typically found for Hamiltonian systems with a separable Hamiltonian:

$H(p,q) = H_{kin}(p) + V(q)$

which gives the equations of motion:

$q'(t) = dH_{kin} / dp = f(p)$

$p'(t) = dV / dq = g(q)$

The algorithmic implementation of this situation is described by a pair of callable objects for *f* and *g* with a specific parameter signature. Such a system should be implemented as a std::pair of functions or a functors. Symplectic systems are used in symplectic steppers like `symplectic_rkn_sb3a_mclachlan`.

Notation

`System`	A type that is a model of SymplecticSystem
`Coor`	The type of the coordinate *q*
`Momentum`	The type of the momentum *p*
`CoorDeriv`	The type of the derivative of coordinate *q'*
`MomentumDeriv`	The type of the derivative of momentum *p'*
`sys`	An object of the type `System`
`q`	Object of type Coor
`p`	Object of type Momentum
`dqdt`	Object of type CoorDeriv

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

`dpdt`	Object of type MomentumDeriv

Valid expressions

Name	Expression	Type	Semantics
Check for pair	`boost::is_pair< System >::type`	`boost::mpl::true_`	Check if System is a pair
Calculate $dq/dt = f(p)$	`sys.first( p , dqdt )`	`void`	Calculates $f(p)$, the result is stored into `dqdt`
Calculate $dp/dt = g(q)$	`sys.second( q , dpdt )`	`void`	Calculates $g(q)$, the result is stored into `dpdt`

Simple Symplectic System

Description

In most Hamiltonian systems the kinetic term is a quadratic term in the momentum $H_{kin} = p\^2 / 2m$ and in many cases it is possible to rescale coordinates and set $m=1$ which leads to a trivial equation of motion:

$q'(t) = f(p) = p.$

while for p' we still have the general form

$p'(t) = g(q)$

As this case is very frequent we introduced a concept where only the nontrivial equation for p' has to be provided to the symplectic stepper. We call this concept *SimpleSymplecticSystem*

Notation

System	A type that is a model of SimpleSymplecticSystem
Coor	The type of the coordinate q
MomentumDeriv	The type of the derivative of momentum p'
sys	An object that models System
q	Object of type Coor
dpdt	Object of type MomentumDeriv

Valid Expressions

Name	Expression	Type	Semantics
Check for pair	`boost::is_pair< System >::type`	`boost::mpl::false_`	Check if System is a pair, should be evaluated to false in this case.
Calculate $dp/dt = g(q)$	`sys( q , dpdt )`	`void`	Calculates $g(q)$, the result is stored into `dpdt`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Implicit System

Description

This concept describes how to define a ODE that can be solved by an implicit routine. Implicit routines need not only the function *f(x,t)* but also the Jacobian *df/dx = A(x,t)*. *A* is a matrix and implicit routines need to solve the linear problem *Ax = b*. In odeint this is implemented with use of Boost.uBLAS, therefore, the *state_type* implicit routines is *ublas::vector* and the matrix is defined as *ublas::matrix*.

Notation

`System`	A type that is a model of `Implicit System`
`Time`	A type representing the time of the ODE
`sys`	An object of type `System`
`x`	Object of type ublas::vector
`dxdt`	Object of type ublas::vector
`jacobi`	Object of type ublas::matrix
`t`	Object of type `Time`

Valid Expressions

Name	Expression	Type	Semantics
Calculate *dx/dt := f(x,t)*	`sys.first( x , dxdt , t )`	`void`	Calculates `f(x,t)`, the result is stored into dxdt
Calculate *A := df/dx (x,t)*	`sys.second( x , jacobi , t )`	`void`	Calculates the Jacobian of *f* at *x*,*t*, the result is stored into `jacobi`

Stepper

This concepts specifies the interface a simple stepper has to fulfill to be used within the integrate functions.

Description

The basic stepper concept. A basic stepper following this Stepper concept is able to perform a single step of the solution *x(t)* of an ODE to obtain *x(t+dt)* using a given step size *dt*. Basic steppers can be Runge-Kutta steppers, symplectic steppers as well as implicit steppers. Depending on the actual stepper, the ODE is defined as System, Symplectic System, Simple Symplectic System or Implicit System. Note that all error steppers are also basic steppers.

Refinement of

- DefaultConstructable
- CopyConstructable

Associated types

- **state_type**

  `Stepper::state_type`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

The type characterizing the state of the ODE, hence *x*.

- **deriv_type**

  `Stepper::deriv_type`

  The type characterizing the derivative of the ODE, hence *d x/dt*.

- **time_type**

  `Stepper::time_type`

  The type characterizing the dependent variable of the ODE, hence the time *t*.

- **value_type**

  `Stepper::value_type`

  The numerical data type which is used within the stepper, something like `float`, `double`, `complex&lt; double &gt;`.

- **order_type**

  `Stepper::order_type`

  The type characterizing the order of the ODE, typically `unsigned short`.

- **stepper_category**

  `Stepper::stepper_category`

  A tag type characterizing the category of the stepper. This type must be convertible to `stepper_tag`.

Notation

`Stepper`	A type that is a model of Stepper
`State`	A type representing the state *x* of the ODE
`Time`	A type representing the time *t* of the ODE
`stepper`	An object of type `Stepper`
`x`	Object of type `State`
`t`, `dt`	Objects of type `Time`
`sys`	An object defining the ODE. Depending on the Stepper this might be a model of System, Symplectic System, Simple Symplectic System or Implicit System

Valid Expressions

Name	**Expression**	**Type**	**Semantics**
Get the order	`stepper.order()`	`order_type`	Returns the order of the stepper.
Do step	`stepper.do_step( sys , x , t , dt )`	`void`	Performs one step of step size `dt`. The newly obtained state is written in place in `x`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Models

- `runge_kutta4`
- `euler`
- `runge_kutta_cash_karp54`
- `runge_kutta_dopri5`
- `runge_kutta_fehlberg78`
- `modified_midpoint`
- `rosenbrock4`

Error Stepper

This concepts specifies the interface an error stepper has to fulfill to be used within a ControlledErrorStepper. An error stepper must always fulfill the stepper concept. This can trivially implemented by

```
template< class System >
error_stepper::do_step( System sys , state_type &x , time_type t , time_type dt )
{
    state_type xerr;
    // allocate xerr
    do_step( sys , x , t , dt , xerr );
}
```

Description

An error stepper following this Error Stepper concept is capable of doing one step of the solution *x(t)* of an ODE with step-size *dt* to obtain *x(t+dt)* and also computing an error estimate x_{err} of the result. Error Steppers can be Runge-Kutta steppers, symplectic steppers as well as implicit steppers. Based on the stepper type, the ODE is defined as System, Symplectic System, Simple Symplectic System or Implicit System.

Refinement of

- DefaultConstructable
- CopyConstructable
- Stepper

Associated types

- **state_type**

  `Stepper::state_type`

  The type characterizing the state of the ODE, hence *x*.

- **deriv_type**

  `Stepper::deriv_type`

  The type characterizing the derivative of the ODE, hence *d x/dt*.

- **time_type**

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

`Stepper::time_type`

The type characterizing the dependent variable of the ODE, hence the time *t*.

- **value_type**

  `Stepper::value_type`

  The numerical data type which is used within the stepper, something like `float`, `double`, `complex&lt; double &gt;`.

- **order_type**

  `Stepper::order_type`

  The type characterizing the order of the ODE, typically `unsigned short`.

- **stepper_category**

  `Stepper::stepper_category`

  A tag type characterizing the category of the stepper. This type must be convertible to `error_stepper_tag`.

Notation

`ErrorStepper`	A type that is a model of Error Stepper
`State`	A type representing the state *x* of the ODE
`Error`	A type representing the error calculated by the stepper, usually same as `State`
`Time`	A type representing the time *t* of the ODE
`stepper`	An object of type `ErrorStepper`
`x`	Object of type `State`
`xerr`	Object of type `Error`
`t`, `dt`	Objects of type `Time`
`sys`	An object defining the ODE, should be a model of either System, Symplectic System, Simple Symplectic System or Implicit System.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Valid Expressions

Name	Expression	Type	Semantics
Get the stepper order	`stepper.order()`	`order_type`	Returns the order of the stepper for one step without error estimation.
Get the stepper order	`stepper.stepper_order()`	`order_type`	Returns the order of the stepper for one error estimation step which is used for error calculation.
Get the error order	`stepper.errorr_order()`	`order_type`	Returns the order of the error step which is used for error calculation.
Do step	`stepper.do_step( sys , x , t , dt )`	`void`	Performs one step of step size `dt`. The newly obtained state is written in-place to `x`.
Do step with error estimation	`stepper.do_step( sys , x , t , dt , xerr )`	`void`	Performs one step of step size `dt` with error estimation. The newly obtained state is written in-place to `x` and the estimated error to `xerr`.

Models

- `runge_kutta_cash_karp54`
- `runge_kutta_dopri5`
- `runge_kutta_fehlberg78`
- `rosenbrock4`

Controlled Stepper

This concept specifies the interface a controlled stepper has to fulfill to be used within integrate functions.

Description

A controlled stepper following this Controlled Stepper concept provides the possibility to perform one step of the solution *x(t)* of an ODE with step-size *dt* to obtain *x(t+dt)* with a given step-size *dt*. Depending on an error estimate of the solution the step might be rejected and a smaller step-size is suggested.

Associated types

- **state_type**

  `Stepper::state_type`

  The type characterizing the state of the ODE, hence *x*.

- **deriv_type**

  `Stepper::deriv_type`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

The type characterizing the derivative of the ODE, hence *d x/dt*.

- **time_type**

  `Stepper::time_type`

  The type characterizing the dependent variable of the ODE, hence the time *t*.

- **value_type**

  `Stepper::value_type`

  The numerical data type which is used within the stepper, something like `float`, `double`, `complex&lt; double &gt;`.

- **stepper_category**

  `Stepper::stepper_category`

  A tag type characterizing the category of the stepper. This type must be convertible to `controlled_stepper_tag`.

Notation

`ControlledStepper`	A type that is a model of Controlled Stepper
`State`	A type representing the state *x* of the ODE
`Time`	A type representing the time *t* of the ODE
`stepper`	An object of type `ControlledStepper`
`x`	Object of type `State`
`t`, `dt`	Objects of type `Time`
`sys`	An object defining the ODE, should be a model of System, Symplectic System, Simple Symplectic System or Implicit System.

Valid Expressions

Name	Expression	Type	Semantics
Do step	`stepper.try_step( sys , x , t , dt )`	`controlled_step_result`	Tries one step of step size `dt`. If the step was successful, `success` is returned, the resulting state is written to `x`, the new time is stored in `t` and `dt` now contains a new (possibly larger) step-size for the next step. If the error was too big, `rejected` is returned and the results are neglected - `x` and `t` are unchanged and `dt` now contains a reduced step-size to be used for the next try.

Models

- `controlled_error_stepper< runge_kutta_cash_karp54 >`
- `controlled_error_stepper_fsal< runge_kutta_dopri5 >`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- `controlled_error_stepper< runge_kutta_fehlberg78 >`
- `rosenbrock4_controller`
- `bulirsch_stoer`

Dense Output Stepper

This concept specifies the interface a dense output stepper has to fulfill to be used within integrate functions.

Description

A dense output stepper following this Dense Output Stepper concept provides the possibility to perform a single step of the solution $x(t)$ of an ODE to obtain $x(t+dt)$. The step-size `dt` might be adjusted automatically due to error control. Dense output steppers also can interpolate the solution to calculate the state $x(t')$ at any point $t <= t' <= t+dt$.

Associated types

- **state_type**

  `Stepper::state_type`

  The type characterizing the state of the ODE, hence x.

- **deriv_type**

  `Stepper::deriv_type`

  The type characterizing the derivative of the ODE, hence $d x/dt$.

- **time_type**

  `Stepper::time_type`

  The type characterizing the dependent variable of the ODE, hence the time t.

- **value_type**

  `Stepper::value_type`

  The numerical data type which is used within the stepper, something like `float`, `double`, `complex&lt; double &gt;`.

- **stepper_category**

  `Stepper::stepper_category`

  A tag type characterizing the category of the stepper. This type must be convertible to `dense_output_stepper_tag`.

Notation

`Stepper`	A type that is a model of Dense Output Stepper
`State`	A type representing the state x of the ODE
`stepper`	An object of type `Stepper`
`x0`, `x`	Object of type `State`
`t0`, `dt0`, `t`	Objects of type `Stepper::time_type`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

`sys`	An object defining the ODE, should be a model of System, Symplectic System, Simple Symplectic System or Implicit System.

Valid Expressions

Name	Expression	Type	Semantics
Initialize integration	`stepper.initialize( x0 , t0 , dt0 )`	void	Initializes the stepper with initial values `x0`, `t0` and `dt0`.
Do step	`stepper.do_step( sys )`	`std::pair< Stepper::time_type , Stepper::time_type >`	Performs one step using the ODE defined by `sys`. The step-size might be changed internally due to error control. This function returns a pair containing `t` and `t+dt` representing the interval for which interpolation can be performed.
Do interpolation	`stepper.calc_state( t_inter , x )`	`void`	Performs the interpolation to calculate /x(t_{inter}/) where /t <= t_{inter} <= t+dt/.
Get current time	`stepper.current_time()`	`const Stepper::time_type&`	Returns the current time *t+dt* of the stepper, that is the end time of the last step and the starting time for the next call of `do_step`
Get current state	`stepper.current_state()`	`const Stepper::state_type&`	Returns the current state of the stepper, that is *x(t+dt)*, the state at the time returned by `stepper.current_time()`
Get current time step	`stepper.current_time_step()`	`const Stepper::time_type&`	Returns the current step size of the stepper, that is *dt*

Models

- `dense_output_controlled_explicit_fsal< controlled_error_stepper_fsal< runge_kutta_dopri5 >`
- `bulirsch_stoer_dense_out`
- `rosenbrock4_dense_output`

State Algebra Operations

> **Note**
>
> The following does not apply to implicit steppers like implicit_euler or Rosenbrock 4 as there the `state_type` can not be changed from `ublas::vector` and no algebra/operations are used.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Description

The `State`, `Algebra` and `Operations` together define a concept describing how the mathematical vector operations required for the stepper algorithms are performed. The typical vector operation done within steppers is

$\boldsymbol{y} = \Sigma \alpha_i \boldsymbol{x_i}$.

The `State` represents the state variable of an ODE, usually denoted with *x*. Algorithmically, the state is often realized as a `vector< double >` or `array< double , N >`, however, the genericity of odeint enables you to basically use anything as a state type. The algorithmic counterpart of such mathematical expressions is divided into two parts. First, the `Algebra` is used to account for the vector character of the equation. In the case of a `vector` as state type this means the `Algebra` is responsible for iteration over all vector elements. Second, the `Operations` are used to represent the actual operation applied to each of the vector elements. So the `Algebra` iterates over all elements of the `State`s and calls an operation taken from the `Operations` for each element. This is where `State`, `Algebra` and `Operations` have to work together to make odeint running. Please have a look at the `range_algebra` and `default_operations` to see an example how this is implemented.

In the following we describe how `State`, `Algebra` and `Operations` are used together within the stepper implementations.

Operations

Notation

`Operations`	The operations type
`Value1`, ... , `ValueN`	Types representing the value or time type of stepper
`Scale`	Type of the scale operation
`scale`	Object of type `Scale`
`ScaleSumN`	Type that represents a general scale_sum operation, `N` should be replaced by a number from 1 to 14.
`scale_sumN`	Object of type `ScaleSumN`, `N` should be replaced by a number from 1 to 14.
`ScaleSumSwap2`	Type of the scale sum swap operation
`scale_sum_swap2`	Object of type `ScaleSumSwap2`
`a1, a2, ...`	Objects of type `Value1`, `Value2`, ...
`y, x1, x2, ...`	Objects of `State`'s value type

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Valid Expressions

Name	**Expression**	**Type**	**Semantics**
Get scale operation	`Operations::scale< Value >`	`Scale`	Get `Scale` from `Operations`
`Scale` constructor	`Scale< Value >( a )`	`Scale`	Constructs a `Scale` object
`Scale` operation	`scale( x )`	`void`	Calculates `x *= a`
Get general `scale_sum` operation	`Operations::scale_sum`*`N`*`< Value1 , ... , ValueN >`	`ScaleSum`*`N`*	Get the `ScaleSum`*`N`* type from `Operations`, *`N`* should be replaced by a number from 1 to 14.
`scale_sum` constructor	`ScaleSum`*`N`*`< Value1 , ... , ValueN >( a1 , ... , aN )`	`ScaleSum`*`N`*	Constructs a `scale_sum` object given *`N`* parameter values with *`N`* between 1 and 14.
`scale_sum` operation	`scale_sum`*`N`*`( y , x1 , ... , xN )`	`void`	Calculates `y = a1*x1 + a2*x2 + ... + aN*xN`. Note that this is an *`N`*+1-ary function call.
Get scale sum swap operation	`Operations::scale_sum_swap2< Value1 , Value2 >`	`ScaleSumSwap2`	Get scale sum swap from operations
`ScaleSumSwap2` constructor	`ScaleSumSwap2< Value1 , Value2 >( a1 , a2 )`	`ScaleSumSwap2`	Constructor
`ScaleSumSwap2` operation	`scale_sum_swap2( x1 , x2 , x3 )`	`void`	Calculates `tmp = x1`, `x1 = a1*x2 + a2*x3` and `x2 = tmp`.

Algebra

Notation

`State`	The state type
`Algebra`	The algebra type
`Operation`*`N`*	An *`N`*-ary operation type, *`N`* should be a number from 1 to 14.
`algebra`	Object of type `Algebra`
`operation`*`N`*	Object of type `Operation`*`N`*
`y, x1, x2, ...`	Objects of type `State`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Valid Expressions

Name	Expression	Type	Semantics
Vector Operation with arity 2	`algebra.for_each2( y , x , operation2 )`	void	Calls `operation2( y_i , x_i )` for each element `y_i` of `y` and `x_i` of `x`.
Vector Operation with arity 3	`algebra.for_each3( y , x1 , x2 , operation3 )`	void	Calls `operation3( y_i , x1_i , x2_i )` for each element `y_i` of `y` and `x1_i` of `x1` and `x2_i` of `x2`.
Vector Operation with arity *N*	`algebra.for_eachN( y , x1 , ... , xN , operationN )`	void	Calls `operationN( y_i , x1_i , ... , xN_i )` for each element `y_i` of `y` and `x1_i` of `x1` and so on. *N* should be replaced by a number between 1 and 14.

Pre-Defined implementations

As standard configuration odeint uses the `range_algebra` and `default_operations` which suffices most situations. However, a few more possibilities exist either to gain better performance or to ensure interoperability with other libraries. In the following we list the existing `Algebra`/`Operations` configurations that can be used in the steppers.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

`State`	`Algebra`	`Operations`	**Remarks**
Anything supporting Boost.Range, like `std::vector`, `std::list`, `boost::array`,... based on a `value_type` that supports operators +,* (typically `double`)	`range_algebra`	`default_operations`	Standard implementation, applicable for most typical situations.
`boost::array` based on a `value_type` that supports operators +,*	`array_algebra`	`default_operations`	Special implementation for boost::array with better performance than `range_algebra`
Anything that defines operators + within itself and * with scalar (Mathematically spoken, anything that is a vector space).	`vector_space_algebra`	`default_operations`	For the use of Controlled Stepper, the template `vector_space_reduce` has to be instantiated.
`thrust::device_vector`, `thrust::host_vector`	`thrust_algebra`	`thrust_operations`	For running odeint on CUDA devices by using Thrust
Any RandomAccessRange	`openmp_range_algebra`	`default_operations`	OpenMP-parallelised range algebra
`openmp_state`	`openmp_algebra`	`default_operations`	OpenMP-parallelised algebra for split data
`boost::array` or anything which allocates the elements in a C-like manner	`vector_space_algebra`	`mkl_operations`	Using the Intel Math Kernel Library in odeint for maximum performance. Currently, only the RK4 stepper is supported.

Example expressions

Name	Expression	Type	Semantics
Vector operation	`algebra.for_each3( y , x1 , x2 , Operations::scale_sum2< Value1 , Value2 >( a1 , a2 ) )`	void	Calculates ***y*** = *a1* ***x1*** + *a2* ***x2***

State Wrapper

Description

The `State Wrapper` concept describes the way odeint creates temporary state objects to store intermediate results within the stepper's `do_step` methods.

Notation

`State`	A type that is the `state_type` of the ODE

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

`WrappedState`	A type that is a model of State Wrapper for the state type `State`.
`x`	Object of type `State`
`w`	Object of type `WrappedState`

Valid Expressions

Name	**Expression**	**Type**	**Semantics**
Get resizeability	`is_resizeable< State >`	`boost::false_type` or `boost::true_type`	Returns `boost::true_type` if the `State` is resizeable, `boost::false_type` otherwise.
Create `WrappedState` type	`state_wrapper< State >`	`WrappedState`	Creates the type for a `WrappedState` for the state type `State`
Constructor	`WrappedState()`	`WrappedState`	Constructs a state wrapper with an empty state
Copy Constructor	`WrappedState( w )`	`WrappedState`	Constructs a state wrapper with a state of the same size as the state in `w`
Get state	`w.m_v`	`State`	Returns the `State` object of this state wrapper.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Literature

General information about numerical integration of ordinary differential equations:

[1] Press William H et al., Numerical Recipes 3rd Edition: The Art of Scientific Computing, 3rd ed. (Cambridge University Press, 2007).

[2] Ernst Hairer, Syvert P. Nørsett, and Gerhard Wanner, Solving Ordinary Differential Equations I: Nonstiff Problems, 2nd ed. (Springer, Berlin, 2009).

[3] Ernst Hairer and Gerhard Wanner, Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems, 2nd ed. (Springer, Berlin, 2010).

Symplectic integration of numerical integration:

[4] Ernst Hairer, Gerhard Wanner, and Christian Lubich, Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations, 2nd ed. (Springer-Verlag Gmbh, 2006).

[5] Leimkuhler Benedict and Reich Sebastian, Simulating Hamiltonian Dynamics (Cambridge University Press, 2005).

Special symplectic methods:

[6] Haruo Yoshida, “Construction of higher order symplectic integrators,” Physics Letters A 150, no. 5 (November 12, 1990): 262-268.

[7] Robert I. McLachlan, “On the numerical integration of ordinary differential equations by symmetric composition methods,” SIAM J. Sci. Comput. 16, no. 1 (1995): 151-168.

Special systems:

[8] Fermi-Pasta-Ulam nonlinear lattice oscillations

[9] Arkady Pikovsky, Michael Rosemblum, and Jürgen Kurths, Synchronization: A Universal Concept in Nonlinear Sciences. (Cambridge University Press, 2001).

render X

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Acknowledgments

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

odeint Reference

Header <boost/numeric/odeint/integrate/integrate.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename System, typename State, typename Time, 
               typename Observer> 
        boost::enable_if< typename has_value_type< State >::type, size_t >::type 
        integrate(System, State &, Time, Time, Time, Observer);
      template<typename System, typename State, typename Time> 
        size_t integrate(System, State &, Time, Time, Time);
    }
  }
}
```

Function template integrate

boost::numeric::odeint::integrate — Integrates the ODE.

Synopsis

```
// In header: <boost/numeric/odeint/integrate/integrate.hpp>


template<typename System, typename State, typename Time, typename Observer> 
  boost::enable_if< typename has_value_type< State >::type, size_t >::type 
  integrate(System system, State & start_state, Time start_time, 
            Time end_time, Time dt, Observer observer);
```

Description

Integrates the ODE given by system from start_time to end_time starting with start_state as initial condition and dt as initial time step. This function uses a dense output dopri5 stepper and performs an adaptive integration with step size control, thus dt changes during the integration. This method uses standard error bounds of 1E-6. After each step, the observer is called.

Parameters:	`dt`	Initial step size, will be adjusted during the integration.
	`end_time`	End time of the integration.
	`observer`	Observer that will be called after each time step.
	`start_state`	The initial state.
	`start_time`	Start time of the integration.
	`system`	The system function to solve, hence the r.h.s. of the ordinary differential equation.
Returns:	The number of steps performed.	

Function template integrate

boost::numeric::odeint::integrate — Integrates the ODE without observer calls.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/integrate/integrate.hpp>


template<typename System, typename State, typename Time>
  size_t integrate(System system, State & start_state, Time start_time,
                   Time end_time, Time dt);
```

Description

Integrates the ODE given by system from start_time to end_time starting with start_state as initial condition and dt as initial time step. This function uses a dense output dopri5 stepper and performs an adaptive integration with step size control, thus dt changes during the integration. This method uses standard error bounds of 1E-6. No observer is called.

Parameters:	`dt`	Initial step size, will be adjusted during the integration.
	`end_time`	End time of the integration.
	`start_state`	The initial state.
	`start_time`	Start time of the integration.
	`system`	The system function to solve, hence the r.h.s. of the ordinary differential equation.
Returns:	The number of steps performed.	

Header <boost/numeric/odeint/integrate/integrate_adaptive.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename Stepper, typename System, typename State,
               typename Time, typename Observer>
        size_t integrate_adaptive(Stepper, System, State &, Time, Time, Time,
                                  Observer);

      // Second version to solve the forwarding problem, can be called with Boost.Range as ↵
start_state.
      template<typename Stepper, typename System, typename State,
               typename Time, typename Observer>
        size_t integrate_adaptive(Stepper stepper, System system,
                                  const State & start_state, Time start_time,
                                  Time end_time, Time dt, Observer observer);

      // integrate_adaptive without an observer.
      template<typename Stepper, typename System, typename State,
               typename Time>
        size_t integrate_adaptive(Stepper stepper, System system,
                                  State & start_state, Time start_time,
                                  Time end_time, Time dt);

      // Second version to solve the forwarding problem, can be called with Boost.Range as ↵
start_state.
      template<typename Stepper, typename System, typename State,
               typename Time>
        size_t integrate_adaptive(Stepper stepper, System system,
                                  const State & start_state, Time start_time,
                                  Time end_time, Time dt);
    }
  }
}
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Function template integrate_adaptive

boost::numeric::odeint::integrate_adaptive — Integrates the ODE with adaptive step size.

Synopsis

```
// In header: <boost/numeric/odeint/integrate/integrate_adaptive.hpp>


template<typename Stepper, typename System, typename State, typename Time, 
         typename Observer> 
  size_t integrate_adaptive(Stepper stepper, System system, 
                            State & start_state, Time start_time, 
                            Time end_time, Time dt, Observer observer);
```

Description

This function integrates the ODE given by system with the given stepper. The observer is called after each step. If the stepper has no error control, the step size remains constant and the observer is called at equidistant time points t0+n*dt. If the stepper is a ControlledStepper, the step size is adjusted and the observer is called in non-equidistant intervals.

Parameters:	`dt`	The time step between observer calls, *not* necessarily the time step of the integration.
	`end_time`	The final integration time tend.
	`observer`	Function/Functor called at equidistant time intervals.
	`start_state`	The initial condition x0.
	`start_time`	The initial time t0.
	`stepper`	The stepper to be used for numerical integration.
	`system`	Function/Functor defining the rhs of the ODE.
Returns:	The number of steps performed.	

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Header <boost/numeric/odeint/integrate/integrate_const.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename Stepper, typename System, typename State,
               typename Time, typename Observer>
        size_t integrate_const(Stepper, System, State &, Time, Time, Time,
                               Observer);

      // Second version to solve the forwarding problem, can be called with Boost.Range as ↵
start_state.
      template<typename Stepper, typename System, typename State,
               typename Time, typename Observer>
        size_t integrate_const(Stepper stepper, System system,
                               const State & start_state, Time start_time,
                               Time end_time, Time dt, Observer observer);

      // integrate_const without observer calls
      template<typename Stepper, typename System, typename State,
               typename Time>
        size_t integrate_const(Stepper stepper, System system,
                               State & start_state, Time start_time,
                               Time end_time, Time dt);

      // Second version to solve the forwarding problem, can be called with Boost.Range as ↵
start_state.
      template<typename Stepper, typename System, typename State,
               typename Time>
        size_t integrate_const(Stepper stepper, System system,
                               const State & start_state, Time start_time,
                               Time end_time, Time dt);
    }
  }
}
```

Function template integrate_const

boost::numeric::odeint::integrate_const — Integrates the ODE with constant step size.

Synopsis

```
// In header: <boost/numeric/odeint/integrate/integrate_const.hpp>


template<typename Stepper, typename System, typename State, typename Time,
         typename Observer>
  size_t integrate_const(Stepper stepper, System system, State & start_state,
                         Time start_time, Time end_time, Time dt,
                         Observer observer);
```

Description

Integrates the ODE defined by system using the given stepper. This method ensures that the observer is called at constant intervals dt. If the Stepper is a normal stepper without step size control, dt is also used for the numerical scheme. If a ControlledStepper is provided, the algorithm might reduce the step size to meet the error bounds, but it is ensured that the observer is always called at equidistant time points t0 + n*dt. If a DenseOutputStepper is used, the step size also may vary and the dense output is used to call the observer at equidistant time points.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters:	`dt`	The time step between observer calls, *not* necessarily the time step of the integration.
	`end_time`	The final integration time tend.
	`observer`	Function/Functor called at equidistant time intervals.
	`start_state`	The initial condition x0.
	`start_time`	The initial time t0.
	`stepper`	The stepper to be used for numerical integration.
	`system`	Function/Functor defining the rhs of the ODE.
Returns:	The number of steps performed.	

Header <boost/numeric/odeint/integrate/integrate_n_steps.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename Stepper, typename System, typename State,
               typename Time, typename Observer>
        Time integrate_n_steps(Stepper, System, State &, Time, Time, size_t,
                               Observer);

      // Solves the forwarding problem, can be called with Boost.Range as start_state.
      template<typename Stepper, typename System, typename State,
               typename Time, typename Observer>
        Time integrate_n_steps(Stepper stepper, System system,
                               const State & start_state, Time start_time,
                               Time dt, size_t num_of_steps,
                               Observer observer);

      // The same function as above, but without observer calls.
      template<typename Stepper, typename System, typename State,
               typename Time>
        Time integrate_n_steps(Stepper stepper, System system,
                               State & start_state, Time start_time, Time dt,
                               size_t num_of_steps);

      // Solves the forwarding problem, can be called with Boost.Range as start_state.
      template<typename Stepper, typename System, typename State,
               typename Time>
        Time integrate_n_steps(Stepper stepper, System system,
                               const State & start_state, Time start_time,
                               Time dt, size_t num_of_steps);
    }
  }
}
```

Function template integrate_n_steps

boost::numeric::odeint::integrate_n_steps — Integrates the ODE with constant step size.

Synopsis

```
// In header: <boost/numeric/odeint/integrate/integrate_n_steps.hpp>


template<typename Stepper, typename System, typename State, typename Time,
         typename Observer>
  Time integrate_n_steps(Stepper stepper, System system, State & start_state,
                         Time start_time, Time dt, size_t num_of_steps,
                         Observer observer);
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Description

This function is similar to integrate_const. The observer is called at equidistant time intervals t0 + n*dt. If the Stepper is a normal stepper without step size control, dt is also used for the numerical scheme. If a ControlledStepper is provided, the algorithm might reduce the step size to meet the error bounds, but it is ensured that the observer is always called at equidistant time points t0 + n*dt. If a DenseOutputStepper is used, the step size also may vary and the dense output is used to call the observer at equidistant time points. The final integration time is always t0 + num_of_steps*dt.

Parameters:	`dt`	The time step between observer calls, *not* necessarily the time step of the integration.
	`num_of_steps`	Number of steps to be performed
	`observer`	Function/Functor called at equidistant time intervals.
	`start_state`	The initial condition x0.
	`start_time`	The initial time t0.
	`stepper`	The stepper to be used for numerical integration.
	`system`	Function/Functor defining the rhs of the ODE.
Returns:	The number of steps performed.	

Header <boost/numeric/odeint/integrate/integrate_times.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename Stepper, typename System, typename State, 
               typename TimeIterator, typename Time, typename Observer> 
        size_t integrate_times(Stepper, System, State &, TimeIterator, 
                               TimeIterator, Time, Observer);

      // Solves the forwarding problem, can be called with Boost.Range as start_state.
      template<typename Stepper, typename System, typename State, 
               typename TimeIterator, typename Time, typename Observer> 
        size_t integrate_times(Stepper stepper, System system, 
                               const State & start_state, 
                               TimeIterator times_start, 
                               TimeIterator times_end, Time dt, 
                               Observer observer);

      // The same function as above, but without observer calls.
      template<typename Stepper, typename System, typename State, 
               typename TimeRange, typename Time, typename Observer> 
        size_t integrate_times(Stepper stepper, System system, 
                               State & start_state, const TimeRange & times, 
                               Time dt, Observer observer);

      // Solves the forwarding problem, can be called with Boost.Range as start_state.
      template<typename Stepper, typename System, typename State, 
               typename TimeRange, typename Time, typename Observer> 
        size_t integrate_times(Stepper stepper, System system, 
                               const State & start_state, 
                               const TimeRange & times, Time dt, 
                               Observer observer);
    }
  }
}
```

Function template integrate_times

boost::numeric::odeint::integrate_times — Integrates the ODE with observer calls at given time points.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/integrate/integrate_times.hpp>


template<typename Stepper, typename System, typename State,
         typename TimeIterator, typename Time, typename Observer>
  size_t integrate_times(Stepper stepper, System system, State & start_state,
                         TimeIterator times_start, TimeIterator times_end,
                         Time dt, Observer observer);
```

Description

Integrates the ODE given by system using the given stepper. This function does observer calls at the subsequent time points given by the range times_start, times_end. If the stepper has not step size control, the step size might be reduced occasionally to ensure observer calls exactly at the time points from the given sequence. If the stepper is a ControlledStepper, the step size is adjusted to meet the error bounds, but also might be reduced occasionally to ensure correct observer calls. If a DenseOutputStepper is provided, the dense output functionality is used to call the observer at the given times. The end time of the integration is always *(end_time-1).

Parameters:	`dt`	The time step between observer calls, *not* necessarily the time step of the integration.
	`observer`	Function/Functor called at equidistant time intervals.
	`start_state`	The initial condition x0.
	`stepper`	The stepper to be used for numerical integration.
	`system`	Function/Functor defining the rhs of the ODE.
	`times_end`	Iterator to the end time
	`times_start`	Iterator to the start time
Returns:	The number of steps performed.	

Header <boost/numeric/odeint/iterator/adaptive_iterator.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename Stepper, typename System, typename State>
        class adaptive_iterator;
      template<typename Stepper, typename System, typename State>
        adaptive_iterator< Stepper, System, State >
        make_adaptive_iterator_begin(Stepper, System, State &,
                                     typename traits::time_type< Stepper >::type,
                                     typename traits::time_type< Stepper >::type,
                                     typename traits::time_type< Stepper >::type);
      template<typename Stepper, typename System, typename State>
        adaptive_iterator< Stepper, System, State >
        make_adaptive_iterator_end(Stepper, System, State &);
      template<typename Stepper, typename System, typename State>
         std::pair< adaptive_iterator< Stepper, System, State >, adaptive_iterator< Stepper, Sys↲
tem, State > >
        make_adaptive_range(Stepper, System, State &,
                            typename traits::time_type< Stepper >::type,
                            typename traits::time_type< Stepper >::type,
                            typename traits::time_type< Stepper >::type);
    }
  }
}
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Class template adaptive_iterator

boost::numeric::odeint::adaptive_iterator — ODE Iterator with adaptive step size. The value type of this iterator is the state type of the stepper.

Synopsis

```
// In header: <boost/numeric/odeint/iterator/adaptive_iterator.hpp>

template<typename Stepper, typename System, typename State>
class adaptive_iterator {
public:
  // construct/copy/destruct
  adaptive_iterator(Stepper, System, State &, time_type, time_type, time_type);
  adaptive_iterator(Stepper, System, State &);
};
```

Description

Implements an iterator representing the solution of an ODE from t_start to t_end evaluated at steps with an adaptive step size dt. After each iteration the iterator dereferences to the state x at the next time t+dt where dt is controlled by the stepper. This iterator can be used with ControlledSteppers and DenseOutputSteppers and it always makes use of the all the given steppers capabilities. A for_each over such an iterator range behaves similar to the integrate_adaptive routine.

adaptive_iterator is a model of single-pass iterator.

The value type of this iterator is the state type of the stepper. Hence one can only access the state and not the current time.

Template Parameters

1.
   ```
   typename Stepper
   ```
   The stepper type which should be used during the iteration.

2.
   ```
   typename System
   ```
   The type of the system function (ODE) which should be solved.

3.
   ```
   typename State
   ```
   The state type of the ODE.

`adaptive_iterator` public construct/copy/destruct

1.
   ```
   adaptive_iterator(Stepper stepper, System sys, State & s, time_type t_start,
                     time_type t_end, time_type dt);
   ```

2.
   ```
   adaptive_iterator(Stepper stepper, System sys, State & s);
   ```

Function template make_adaptive_iterator_begin

boost::numeric::odeint::make_adaptive_iterator_begin — Factory function for adaptive_iterator. Constructs a begin iterator.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/iterator/adaptive_iterator.hpp>


template<typename Stepper, typename System, typename State> 
  adaptive_iterator< Stepper, System, State > 
  make_adaptive_iterator_begin(Stepper stepper, System system, State & x, 
                               typename traits::time_type< Stepper >::type t_start, 
                               typename traits::time_type< Stepper >::type t_end, 
                               typename traits::time_type< Stepper >::type dt);
```

Description

Parameters:	`dt`	The initial time step.
	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`t_end`	The end time, at which the iteration should stop.
	`t_start`	The initial time.
	`x`	The initial state.
Returns:	The adaptive iterator.	

Function template make_adaptive_iterator_end

boost::numeric::odeint::make_adaptive_iterator_end — Factory function for adaptive_iterator. Constructs a end iterator.

Synopsis

```
// In header: <boost/numeric/odeint/iterator/adaptive_iterator.hpp>


template<typename Stepper, typename System, typename State> 
  adaptive_iterator< Stepper, System, State > 
  make_adaptive_iterator_end(Stepper stepper, System system, State & x);
```

Description

Parameters:	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`x`	The initial state.
Returns:	The adaptive iterator.	

Function template make_adaptive_range

boost::numeric::odeint::make_adaptive_range — Factory function to construct a single pass range of adaptive iterators. A range is here a pair of adaptive_iterator.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/iterator/adaptive_iterator.hpp>


template<typename Stepper, typename System, typename State> 
  std::pair< adaptive_iterator< Stepper, System, State >, adaptive_iterator< Stepper, Sys↵
tem, State > > 
  make_adaptive_range(Stepper stepper, System system, State & x, 
                      typename traits::time_type< Stepper >::type t_start, 
                      typename traits::time_type< Stepper >::type t_end, 
                      typename traits::time_type< Stepper >::type dt);
```

Description

Parameters:	`dt`	The initial time step.
	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`t_end`	The end time, at which the iteration should stop.
	`t_start`	The initial time.
	`x`	The initial state.
Returns:	The adaptive range.	

Header <boost/numeric/odeint/iterator/adaptive_time_iterator.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename Stepper, typename System, typename State> 
        class adaptive_time_iterator;
      template<typename Stepper, typename System, typename State> 
        adaptive_time_iterator< Stepper, System, State > 
        make_adaptive_time_iterator_begin(Stepper, System, State &, 
                                          typename traits::time_type< Stepper >::type, 
                                          typename traits::time_type< Stepper >::type, 
                                          typename traits::time_type< Stepper >::type);
      template<typename Stepper, typename System, typename State> 
        adaptive_time_iterator< Stepper, System, State > 
        make_adaptive_time_iterator_end(Stepper, System, State &);
      template<typename Stepper, typename System, typename State> 
        std::pair< adaptive_time_iterator< Stepper, System, State >, adaptive_time_iterator< Step↵
per, System, State > > 
        make_adaptive_time_range(Stepper, System, State &, 
                                 typename traits::time_type< Stepper >::type, 
                                 typename traits::time_type< Stepper >::type, 
                                 typename traits::time_type< Stepper >::type);
    }
  }
}
```

Class template adaptive_time_iterator

boost::numeric::odeint::adaptive_time_iterator — ODE Iterator with adaptive step size. The value type of this iterator is a std::pair containing state and time.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/iterator/adaptive_time_iterator.hpp>

template<typename Stepper, typename System, typename State>
class adaptive_time_iterator {
public:
  // construct/copy/destruct
  adaptive_time_iterator(Stepper, System, State &, time_type, time_type,
                         time_type);
  adaptive_time_iterator(Stepper, System, State &);
};
```

Description

Implements an iterator representing the solution of an ODE from t_start to t_end evaluated at steps with an adaptive step size dt. After each iteration the iterator dereferences to a pair containing state and time at the next time point t+dt where dt is controlled by the stepper. This iterator can be used with ControlledSteppers and DenseOutputSteppers and it always makes use of the all the given steppers capabilities. A for_each over such an iterator range behaves similar to the integrate_adaptive routine.

adaptive_iterator is a model of single-pass iterator.

The value type of this iterator is a std::pair of state and time of the stepper.

Template Parameters

1.
   ```
   typename Stepper
   ```

   The stepper type which should be used during the iteration.

2.
   ```
   typename System
   ```

   The type of the system function (ODE) which should be solved.

3.
   ```
   typename State
   ```

   The state type of the ODE.

`adaptive_time_iterator` public construct/copy/destruct

1.
   ```
   adaptive_time_iterator(Stepper stepper, System sys, State & s,
                          time_type t_start, time_type t_end, time_type dt);
   ```

2.
   ```
   adaptive_time_iterator(Stepper stepper, System sys, State & s);
   ```

Function template make_adaptive_time_iterator_begin

boost::numeric::odeint::make_adaptive_time_iterator_begin — Factory function for adaptive_time_iterator. Constructs a begin iterator.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/iterator/adaptive_time_iterator.hpp>


template<typename Stepper, typename System, typename State> 
  adaptive_time_iterator< Stepper, System, State > 
  make_adaptive_time_iterator_begin(Stepper stepper, System system, State & x, 
                                    typename traits::time_type< Stepper >::type t_start, 
                                    typename traits::time_type< Stepper >::type t_end, 
                                    typename traits::time_type< Stepper >::type dt);
```

Description

Parameters:	`dt`	The initial time step.
	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`t_end`	The end time, at which the iteration should stop.
	`t_start`	The initial time.
	`x`	The initial state. `adaptive_time_iterator` stores a reference of s and changes its value during the iteration.
Returns:	The adaptive time iterator.	

Function template make_adaptive_time_iterator_end

boost::numeric::odeint::make_adaptive_time_iterator_end — Factory function for adaptive_time_iterator. Constructs a end iterator.

Synopsis

```
// In header: <boost/numeric/odeint/iterator/adaptive_time_iterator.hpp>


template<typename Stepper, typename System, typename State> 
  adaptive_time_iterator< Stepper, System, State > 
  make_adaptive_time_iterator_end(Stepper stepper, System system, State & x);
```

Description

Parameters:	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`x`	The initial state. `adaptive_time_iterator` stores a reference of s and changes its value during the iteration.
Returns:	The adaptive time iterator.	

Function template make_adaptive_time_range

boost::numeric::odeint::make_adaptive_time_range — Factory function to construct a single pass range of adaptive time iterators. A range is here a pair of adaptive_time_iterators.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/iterator/adaptive_time_iterator.hpp>


template<typename Stepper, typename System, typename State>
  std::pair< adaptive_time_iterator< Stepper, System, State >, adaptive_time_iterator< Step↵
per, System, State > >
  make_adaptive_time_range(Stepper stepper, System system, State & x,
                           typename traits::time_type< Stepper >::type t_start,
                           typename traits::time_type< Stepper >::type t_end,
                           typename traits::time_type< Stepper >::type dt);
```

Description

Parameters:	`dt`	The initial time step.
	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`t_end`	The end time, at which the iteration should stop.
	`t_start`	The initial time.
	`x`	The initial state. `adaptive_time_iterator` stores a reference of s and changes its value during the iteration.
Returns:	The adaptive time range.	

Header <boost/numeric/odeint/iterator/const_step_iterator.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename Stepper, typename System, typename State>
        class const_step_iterator;
      template<typename Stepper, typename System, typename State>
        const_step_iterator< Stepper, System, State >
        make_const_step_iterator_begin(Stepper, System, State &,
                                       typename traits::time_type< Stepper >::type,
                                       typename traits::time_type< Stepper >::type,
                                       typename traits::time_type< Stepper >::type);
      template<typename Stepper, typename System, typename State>
        const_step_iterator< Stepper, System, State >
        make_const_step_iterator_end(Stepper, System, State &);
      template<typename Stepper, typename System, typename State>
        std::pair< const_step_iterator< Stepper, System, State >, const_step_iterator< Step↵
per, System, State > >
        make_const_step_range(Stepper, System, State &,
                              typename traits::time_type< Stepper >::type,
                              typename traits::time_type< Stepper >::type,
                              typename traits::time_type< Stepper >::type);
    }
  }
}
```

Class template const_step_iterator

boost::numeric::odeint::const_step_iterator — ODE Iterator with constant step size. The value type of this iterator is the state type of the stepper.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/iterator/const_step_iterator.hpp>

template<typename Stepper, typename System, typename State>
class const_step_iterator {
public:
  // construct/copy/destruct
  const_step_iterator(Stepper, System, State &, time_type, time_type,
                      time_type);
  const_step_iterator(Stepper, System, State &);
};
```

Description

Implements an iterator representing the solution of an ODE from t_start to t_end evaluated at steps with constant step size dt. After each iteration the iterator dereferences to the state x at the next time t+dt. This iterator can be used with Steppers and DenseOutput-Steppers and it always makes use of the all the given steppers capabilities. A for_each over such an iterator range behaves similar to the integrate_const routine.

const_step_iterator is a model of single-pass iterator.

The value type of this iterator is the state type of the stepper. Hence one can only access the state and not the current time.

Template Parameters

1.
   ```
   typename Stepper
   ```

   The stepper type which should be used during the iteration.

2.
   ```
   typename System
   ```

   The type of the system function (ODE) which should be solved.

3.
   ```
   typename State
   ```

   The state type of the ODE.

`const_step_iterator` public construct/copy/destruct

1.
   ```
   const_step_iterator(Stepper stepper, System sys, State & s, time_type t_start,
                       time_type t_end, time_type dt);
   ```

2.
   ```
   const_step_iterator(Stepper stepper, System sys, State & s);
   ```

Function template make_const_step_iterator_begin

boost::numeric::odeint::make_const_step_iterator_begin — Factory function for const_step_iterator. Constructs a begin iterator.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/iterator/const_step_iterator.hpp>


template<typename Stepper, typename System, typename State> 
  const_step_iterator< Stepper, System, State > 
  make_const_step_iterator_begin(Stepper stepper, System system, State & x, 
                                 typename traits::time_type< Stepper >::type t_start, 
                                 typename traits::time_type< Stepper >::type t_end, 
                                 typename traits::time_type< Stepper >::type dt);
```

Description

Parameters:	`dt`	The initial time step.
	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`t_end`	The end time, at which the iteration should stop.
	`t_start`	The initial time.
	`x`	The initial state. `const_step_iterator` stores a reference of s and changes its value during the iteration.
Returns:	The const step iterator.	

Function template make_const_step_iterator_end

boost::numeric::odeint::make_const_step_iterator_end — Factory function for const_step_iterator. Constructs a end iterator.

Synopsis

```
// In header: <boost/numeric/odeint/iterator/const_step_iterator.hpp>


template<typename Stepper, typename System, typename State> 
  const_step_iterator< Stepper, System, State > 
  make_const_step_iterator_end(Stepper stepper, System system, State & x);
```

Description

Parameters:	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`x`	The initial state. `const_step_iterator` stores a reference of s and changes its value during the iteration.
Returns:	The const_step_iterator.	

Function template make_const_step_range

boost::numeric::odeint::make_const_step_range — Factory function to construct a single pass range of const step iterators. A range is here a pair of const_step_iterator.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/iterator/const_step_iterator.hpp>


template<typename Stepper, typename System, typename State>
  std::pair< const_step_iterator< Stepper, System, State >, const_step_iterator< Stepper, Sys↲
tem, State > >
  make_const_step_range(Stepper stepper, System system, State & x,
                        typename traits::time_type< Stepper >::type t_start,
                        typename traits::time_type< Stepper >::type t_end,
                        typename traits::time_type< Stepper >::type dt);
```

Description

Parameters:		
	`dt`	The initial time step.
	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`t_end`	The end time, at which the iteration should stop.
	`t_start`	The initial time.
	`x`	The initial state. `const_step_iterator` store a reference of s and changes its value during the iteration.
Returns:	The const step range.	

Header <boost/numeric/odeint/iterator/const_step_time_iterator.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename Stepper, typename System, typename State>
        class const_step_time_iterator;
      template<typename Stepper, typename System, typename State>
        const_step_time_iterator< Stepper, System, State >
        make_const_step_time_iterator_begin(Stepper, System, State &,
                                            typename traits::time_type< Stepper >::type,
                                            typename traits::time_type< Stepper >::type,
                                            typename traits::time_type< Stepper >::type);
      template<typename Stepper, typename System, typename State>
        const_step_time_iterator< Stepper, System, State >
        make_const_step_time_iterator_end(Stepper, System, State &);
      template<typename Stepper, typename System, typename State>
        std::pair< const_step_time_iterator< Stepper, System, State >, const_step_time_iterat↲
or< Stepper, System, State > >
        make_const_step_time_range(Stepper, System, State &,
                                   typename traits::time_type< Stepper >::type,
                                   typename traits::time_type< Stepper >::type,
                                   typename traits::time_type< Stepper >::type);
    }
  }
}
```

Class template const_step_time_iterator

boost::numeric::odeint::const_step_time_iterator — ODE Iterator with constant step size. The value type of this iterator is a std::pair containing state and time.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/iterator/const_step_time_iterator.hpp>

template<typename Stepper, typename System, typename State>
class const_step_time_iterator {
public:
  // construct/copy/destruct
  const_step_time_iterator(Stepper, System, State &, time_type, time_type,
                           time_type);
  const_step_time_iterator(Stepper, System, State &);
};
```

Description

Implements an iterator representing the solution of an ODE from t_start to t_end evaluated at steps with constant step size dt. After each iteration the iterator dereferences to a pair containing state and time at the next time point t+dt.. This iterator can be used with Steppers and DenseOutputSteppers and it always makes use of the all the given steppers capabilities. A for_each over such an iterator range behaves similar to the integrate_const routine.

const_step_time_iterator is a model of single-pass iterator.

The value type of this iterator is a pair with the state type and time type of the stepper.

Template Parameters

1.
   ```
   typename Stepper
   ```

   The stepper type which should be used during the iteration.

2.
   ```
   typename System
   ```

   The type of the system function (ODE) which should be solved.

3.
   ```
   typename State
   ```

   The state type of the ODE.

`const_step_time_iterator` public construct/copy/destruct

1.
   ```
   const_step_time_iterator(Stepper stepper, System sys, State & s,
                            time_type t_start, time_type t_end, time_type dt);
   ```

2.
   ```
   const_step_time_iterator(Stepper stepper, System sys, State & s);
   ```

Function template make_const_step_time_iterator_begin

boost::numeric::odeint::make_const_step_time_iterator_begin — Factory function for const_step_time_iterator. Constructs a begin iterator.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/iterator/const_step_time_iterator.hpp>


template<typename Stepper, typename System, typename State>
  const_step_time_iterator< Stepper, System, State >
  make_const_step_time_iterator_begin(Stepper stepper, System system,
                                      State & x,
                                      typename traits::time_type< Stepper >::type t_start,
                                      typename traits::time_type< Stepper >::type t_end,
                                      typename traits::time_type< Stepper >::type dt);
```

Description

Parameters:	`dt`	The initial time step.
	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`t_end`	The end time, at which the iteration should stop.
	`t_start`	The initial time.
	`x`	The initial state. `const_step_time_iterator` stores a reference of s and changes its value during the iteration.
Returns:	The const step time iterator.	

Function template make_const_step_time_iterator_end

boost::numeric::odeint::make_const_step_time_iterator_end — Factory function for const_step_time_iterator. Constructs a end iterator.

Synopsis

```
// In header: <boost/numeric/odeint/iterator/const_step_time_iterator.hpp>


template<typename Stepper, typename System, typename State>
  const_step_time_iterator< Stepper, System, State >
  make_const_step_time_iterator_end(Stepper stepper, System system, State & x);
```

Description

Parameters:	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`x`	The initial state. `const_step_time_iterator` store a reference of s and changes its value during the iteration.
Returns:	The const step time iterator.	

Function template make_const_step_time_range

boost::numeric::odeint::make_const_step_time_range — Factory function to construct a single pass range of const_step_time_iterator. A range is here a pair of const_step_time_iterator.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/iterator/const_step_time_iterator.hpp>


template<typename Stepper, typename System, typename State> 
  std::pair< const_step_time_iterator< Stepper, System, State >, const_step_time_iterator< Step↵
per, System, State > > 
  make_const_step_time_range(Stepper stepper, System system, State & x, 
                             typename traits::time_type< Stepper >::type t_start, 
                             typename traits::time_type< Stepper >::type t_end, 
                             typename traits::time_type< Stepper >::type dt);
```

Description

Parameters:	`dt`	The initial time step.
	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`t_end`	The end time, at which the iteration should stop.
	`x`	The initial state. `const_step_time_iterator` stores a reference of s and changes its value during the iteration.
Returns:	The const step time range.	

Header <boost/numeric/odeint/iterator/n_step_iterator.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename Stepper, typename System, typename State> 
        class n_step_iterator;
      template<typename Stepper, typename System, typename State> 
        n_step_iterator< Stepper, System, State > 
        make_n_step_iterator_begin(Stepper, System, State &, 
                                   typename traits::time_type< Stepper >::type, 
                                   typename traits::time_type< Stepper >::type, 
                                   size_t);
      template<typename Stepper, typename System, typename State> 
        n_step_iterator< Stepper, System, State > 
        make_n_step_iterator_end(Stepper, System, State &);
      template<typename Stepper, typename System, typename State> 
        std::pair< n_step_iterator< Stepper, System, State >, n_step_iterator< Stepper, Sys↵
tem, State > > 
        make_n_step_range(Stepper, System, State &, 
                          typename traits::time_type< Stepper >::type, 
                          typename traits::time_type< Stepper >::type, 
                          size_t);
    }
  }
}
```

Class template n_step_iterator

boost::numeric::odeint::n_step_iterator — ODE Iterator with constant step size. The value type of this iterator is the state type of the stepper.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/iterator/n_step_iterator.hpp>

template<typename Stepper, typename System, typename State>
class n_step_iterator {
public:
  // construct/copy/destruct
  n_step_iterator(Stepper, System, State &, time_type, time_type, size_t);
  n_step_iterator(Stepper, System, State &);
};
```

Description

Implements an iterator representing the solution of an ODE starting from t with n steps and a constant step size dt. After each iteration the iterator dereferences to the state x at the next time t+dt. This iterator can be used with Steppers and DenseOutputSteppers and it always makes use of the all the given steppers capabilities. A for_each over such an iterator range behaves similar to the integrate_n_steps routine.

n_step_iterator is a model of single-pass iterator.

The value type of this iterator is the state type of the stepper. Hence one can only access the state and not the current time.

Template Parameters

1.
```
typename Stepper
```

The stepper type which should be used during the iteration.

2.
```
typename System
```

The type of the system function (ODE) which should be solved.

3.
```
typename State
```

The state type of the ODE.

`n_step_iterator` public construct/copy/destruct

1.
```
n_step_iterator(Stepper stepper, System sys, State & s, time_type t,
                time_type dt, size_t num_of_steps);
```

2.
```
n_step_iterator(Stepper stepper, System sys, State & s);
```

Function template make_n_step_iterator_begin

boost::numeric::odeint::make_n_step_iterator_begin — Factory function for n_step_iterator. Constructs a begin iterator.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/iterator/n_step_iterator.hpp>


template<typename Stepper, typename System, typename State>
  n_step_iterator< Stepper, System, State >
  make_n_step_iterator_begin(Stepper stepper, System system, State & x,
                             typename traits::time_type< Stepper >::type t,
                             typename traits::time_type< Stepper >::type dt,
                             size_t num_of_steps);
```

Description

Parameters:	`dt`	The initial time step.
	`num_of_steps`	The number of steps to be executed.
	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`t`	The initial time.
	`x`	The initial state. `const_step_iterator` stores a reference of s and changes its value during the iteration.
Returns:	The n-step iterator.	

Function template make_n_step_iterator_end

boost::numeric::odeint::make_n_step_iterator_end — Factory function for n_step_iterator. Constructs an end iterator.

Synopsis

```
// In header: <boost/numeric/odeint/iterator/n_step_iterator.hpp>


template<typename Stepper, typename System, typename State>
  n_step_iterator< Stepper, System, State >
  make_n_step_iterator_end(Stepper stepper, System system, State & x);
```

Description

Parameters:	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`x`	The initial state. `const_step_iterator` stores a reference of s and changes its value during the iteration.
Returns:	The const_step_iterator.	

Function template make_n_step_range

boost::numeric::odeint::make_n_step_range — Factory function to construct a single pass range of n-step iterators. A range is here a pair of n_step_iterator.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/iterator/n_step_iterator.hpp>


template<typename Stepper, typename System, typename State>
  std::pair< n_step_iterator< Stepper, System, State >, n_step_iterator< Stepper, Sys↵
tem, State > >
  make_n_step_range(Stepper stepper, System system, State & x,
                    typename traits::time_type< Stepper >::type t,
                    typename traits::time_type< Stepper >::type dt,
                    size_t num_of_steps);
```

Description

Parameters:	`dt`	The initial time step.
	`num_of_steps`	The number of steps to be executed.
	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`t`	The initial time.
	`x`	The initial state. `const_step_iterator` store a reference of s and changes its value during the iteration.
Returns:	The n-step range.	

Header <boost/numeric/odeint/iterator/n_step_time_iterator.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename Stepper, typename System, typename State>
        class n_step_time_iterator;
      template<typename Stepper, typename System, typename State>
        n_step_time_iterator< Stepper, System, State >
        make_n_step_time_iterator_begin(Stepper, System, State &,
                                        typename traits::time_type< Stepper >::type,
                                        typename traits::time_type< Stepper >::type,
                                        size_t);
      template<typename Stepper, typename System, typename State>
        n_step_time_iterator< Stepper, System, State >
        make_n_step_time_iterator_end(Stepper, System, State &);
      template<typename Stepper, typename System, typename State>
        std::pair< n_step_time_iterator< Stepper, System, State >, n_step_time_iterator< Step↵
per, System, State > >
        make_n_step_time_range(Stepper, System, State &,
                               typename traits::time_type< Stepper >::type,
                               typename traits::time_type< Stepper >::type,
                               size_t);
    }
  }
}
```

Class template n_step_time_iterator

boost::numeric::odeint::n_step_time_iterator — ODE Iterator with constant step size. The value type of this iterator is a std::pair containing state and time.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/iterator/n_step_time_iterator.hpp>

template<typename Stepper, typename System, typename State>
class n_step_time_iterator {
public:
  // construct/copy/destruct
  n_step_time_iterator(Stepper, System, State &, time_type, time_type, size_t);
  n_step_time_iterator(Stepper, System, State &);
};
```

Description

Implements an iterator representing the solution of an ODE starting from t with n steps and a constant step size dt. After each iteration the iterator dereferences to a pair of state and time at the next time t+dt. This iterator can be used with Steppers and DenseOutput-Steppers and it always makes use of the all the given steppers capabilities. A for_each over such an iterator range behaves similar to the integrate_n_steps routine.

n_step_time_iterator is a model of single-pass iterator.

The value type of this iterator is pair of state and time.

Template Parameters

1.
   ```
   typename Stepper
   ```

   The stepper type which should be used during the iteration.

2.
   ```
   typename System
   ```

   The type of the system function (ODE) which should be solved.

3.
   ```
   typename State
   ```

   The state type of the ODE.

`n_step_time_iterator` public construct/copy/destruct

1.
   ```
   n_step_time_iterator(Stepper stepper, System sys, State & s, time_type t,
                        time_type dt, size_t num_of_steps);
   ```

2.
   ```
   n_step_time_iterator(Stepper stepper, System sys, State & s);
   ```

Function template make_n_step_time_iterator_begin

boost::numeric::odeint::make_n_step_time_iterator_begin — Factory function for n_step_time_iterator. Constructs a begin iterator.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/iterator/n_step_time_iterator.hpp>


template<typename Stepper, typename System, typename State>
  n_step_time_iterator< Stepper, System, State >
  make_n_step_time_iterator_begin(Stepper stepper, System system, State & x,
                                  typename traits::time_type< Stepper >::type t,
                                  typename traits::time_type< Stepper >::type dt,
                                  size_t num_of_steps);
```

Description

Parameters:	`dt`	The initial time step.
	`num_of_steps`	The number of steps to be executed.
	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`t`	The initial time.
	`x`	The initial state. `const_step_iterator` stores a reference of s and changes its value during the iteration.
Returns:	The n-step iterator.	

Function template make_n_step_time_iterator_end

boost::numeric::odeint::make_n_step_time_iterator_end — Factory function for n_step_time_iterator. Constructs an end iterator.

Synopsis

```
// In header: <boost/numeric/odeint/iterator/n_step_time_iterator.hpp>


template<typename Stepper, typename System, typename State>
  n_step_time_iterator< Stepper, System, State >
  make_n_step_time_iterator_end(Stepper stepper, System system, State & x);
```

Description

Parameters:	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`x`	The initial state. `const_step_iterator` stores a reference of s and changes its value during the iteration.
Returns:	The const_step_iterator.	

Function template make_n_step_time_range

boost::numeric::odeint::make_n_step_time_range — Factory function to construct a single pass range of n-step iterators. A range is here a pair of n_step_time_iterator.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/iterator/n_step_time_iterator.hpp>


template<typename Stepper, typename System, typename State>
  std::pair< n_step_time_iterator< Stepper, System, State >, n_step_time_iterator< Stepper, Sys↵
tem, State > >
  make_n_step_time_range(Stepper stepper, System system, State & x,
                         typename traits::time_type< Stepper >::type t,
                         typename traits::time_type< Stepper >::type dt,
                         size_t num_of_steps);
```

Description

Parameters:	`dt`	The initial time step.
	`num_of_steps`	The number of steps to be executed.
	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`t`	The initial time.
	`x`	The initial state. `const_step_iterator` store a reference of s and changes its value during the iteration.
Returns:	The n-step range.	

Header <boost/numeric/odeint/iterator/times_iterator.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename Stepper, typename System, typename State,
               typename TimeIterator>
        class times_iterator;
      template<typename Stepper, typename System, typename State,
               typename TimeIterator>
        times_iterator< Stepper, System, State, TimeIterator >
        make_times_iterator_begin(Stepper, System, State &, TimeIterator,
                                  TimeIterator,
                                  typename traits::time_type< Stepper >::type);
      template<typename TimeIterator, typename Stepper, typename System,
               typename State>
        times_iterator< Stepper, System, State, TimeIterator >
        make_times_iterator_end(Stepper, System, State &);
      template<typename Stepper, typename System, typename State,
               typename TimeIterator>
       std::pair< times_iterator< Stepper, System, State, TimeIterator >, times_iterator< Step↵
per, System, State, TimeIterator > >
        make_times_range(Stepper, System, State &, TimeIterator, TimeIterator,
                         typename traits::time_type< Stepper >::type);
    }
  }
}
```

Class template times_iterator

boost::numeric::odeint::times_iterator — ODE Iterator with given evaluation points. The value type of this iterator is the state type of the stepper.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/iterator/times_iterator.hpp>

template<typename Stepper, typename System, typename State,
         typename TimeIterator>
class times_iterator {
public:
  // construct/copy/destruct
  times_iterator(Stepper, System, State &, TimeIterator, TimeIterator,
                 time_type);
  times_iterator(Stepper, System, State &);
};
```

Description

Implements an iterator representing the solution of an ODE from *t_start to *t_end evaluated at time points given by the sequence t_start to t_end. t_start and t_end are iterators representing a sequence of time points where the solution of the ODE should be evaluated. After each iteration the iterator dereferences to the state x at the next time *t_start++ until t_end is reached. This iterator can be used with Steppers, ControlledSteppers and DenseOutputSteppers and it always makes use of the all the given steppers capabilities. A for_each over such an iterator range behaves similar to the integrate_times routine.

times_iterator is a model of single-pass iterator.

The value type of this iterator is the state type of the stepper. Hence one can only access the state and not the current time.

Template Parameters

1.
   ```
   typename Stepper
   ```

   The stepper type which should be used during the iteration.

2.
   ```
   typename System
   ```

   The type of the system function (ODE) which should be solved.

3.
   ```
   typename State
   ```

   The state type of the ODE.

4.
   ```
   typename TimeIterator
   ```

   The iterator type for the sequence of time points.

`times_iterator` public construct/copy/destruct

1.
   ```
   times_iterator(Stepper stepper, System sys, State & s, TimeIterator t_start,
                  TimeIterator t_end, time_type dt);
   ```

2.
   ```
   times_iterator(Stepper stepper, System sys, State & s);
   ```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Function template make_times_iterator_begin

boost::numeric::odeint::make_times_iterator_begin — Factory function for times_iterator. Constructs a begin iterator.

Synopsis

```
// In header: <boost/numeric/odeint/iterator/times_iterator.hpp>


template<typename Stepper, typename System, typename State, 
         typename TimeIterator> 
  times_iterator< Stepper, System, State, TimeIterator > 
  make_times_iterator_begin(Stepper stepper, System system, State & x, 
                            TimeIterator t_start, TimeIterator t_end, 
                            typename traits::time_type< Stepper >::type dt);
```

Description

Parameters:	`dt`	The initial time step.
	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`t_end`	End iterator of the sequence of evaluation time points.
	`t_start`	Begin iterator of the sequence of evaluation time points.
	`x`	The initial state. `const_step_iterator` stores a reference of s and changes its value during the iteration.
Returns:	The times iterator.	

Function template make_times_iterator_end

boost::numeric::odeint::make_times_iterator_end — Factory function for times_iterator. Constructs an end iterator.

Synopsis

```
// In header: <boost/numeric/odeint/iterator/times_iterator.hpp>


template<typename TimeIterator, typename Stepper, typename System, 
         typename State> 
  times_iterator< Stepper, System, State, TimeIterator > 
  make_times_iterator_end(Stepper stepper, System system, State & x);
```

Description

This function needs the TimeIterator type specifically defined as a template parameter.

Parameters:	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`x`	The initial state. `const_step_iterator` stores a reference of s and changes its value during the iteration.
Returns:	The times iterator.	

Function template make_times_range

boost::numeric::odeint::make_times_range — Factory function to construct a single pass range of times iterators. A range is here a pair of times_iterator.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/iterator/times_iterator.hpp>


template<typename Stepper, typename System, typename State, 
         typename TimeIterator> 
  std::pair< times_iterator< Stepper, System, State, TimeIterator >, times_iterator< Stepper, Sys↵
tem, State, TimeIterator > > 
  make_times_range(Stepper stepper, System system, State & x, 
                   TimeIterator t_start, TimeIterator t_end, 
                   typename traits::time_type< Stepper >::type dt);
```

Description

Parameters:	`dt`	The initial time step.
	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`t_end`	End iterator of the sequence of evaluation time points.
	`t_start`	Begin iterator of the sequence of evaluation time points.
	`x`	The initial state. `const_step_iterator` store a reference of s and changes its value during the iteration.
Returns:	The times iterator range.	

Header <boost/numeric/odeint/iterator/times_time_iterator.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename Stepper, typename System, typename State, 
               typename TimeIterator> 
        class times_time_iterator;
      template<typename Stepper, typename System, typename State, 
               typename TimeIterator> 
        times_time_iterator< Stepper, System, State, TimeIterator > 
        make_times_time_iterator_begin(Stepper, System, State &, TimeIterator, 
                                       TimeIterator, 
                                       typename traits::time_type< Stepper >::type);
      template<typename TimeIterator, typename Stepper, typename System, 
               typename State> 
        times_time_iterator< Stepper, System, State, TimeIterator > 
        make_times_time_iterator_end(Stepper, System, State &);
      template<typename Stepper, typename System, typename State, 
               typename TimeIterator> 
       std::pair< times_time_iterator< Stepper, System, State, TimeIterator >, times_time_iter↵
ator< Stepper, System, State, TimeIterator > > 
        make_times_time_range(Stepper, System, State &, TimeIterator, 
                              TimeIterator, 
                              typename traits::time_type< Stepper >::type);
    }
  }
}
```

Class template times_time_iterator

boost::numeric::odeint::times_time_iterator — ODE Iterator with given evaluation points. The value type of this iterator is a std::pair containing state and time.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/iterator/times_time_iterator.hpp>

template<typename Stepper, typename System, typename State, 
         typename TimeIterator> 
class times_time_iterator {
public:
  // construct/copy/destruct
  times_time_iterator(Stepper, System, State &, TimeIterator, TimeIterator, 
                      time_type);
  times_time_iterator(Stepper, System, State &);
};
```

Description

Implements an iterator representing the solution of an ODE from *t_start to *t_end evaluated at time points given by the sequence t_start to t_end. t_start and t_end are iterators representing a sequence of time points where the solution of the ODE should be evaluated. After each iteration the iterator dereferences to a pair with the state and the time at the next evaluation point *t_start++ until t_end is reached. This iterator can be used with Steppers, ControlledSteppers and DenseOutputSteppers and it always makes use of the all the given steppers capabilities. A for_each over such an iterator range behaves similar to the integrate_times routine.

times_time_iterator is a model of single-pass iterator.

The value type of this iterator is a pair of state and time type.

Template Parameters

1.
   ```
   typename Stepper
   ```

   The stepper type which should be used during the iteration.

2.
   ```
   typename System
   ```

   The type of the system function (ODE) which should be solved.

3.
   ```
   typename State
   ```

   The state type of the ODE.

4.
   ```
   typename TimeIterator
   ```

   The iterator type for the sequence of time points.

`times_time_iterator` public construct/copy/destruct

1.
   ```
   times_time_iterator(Stepper stepper, System sys, State & s, 
                       TimeIterator t_start, TimeIterator t_end, time_type dt);
   ```

2.
   ```
   times_time_iterator(Stepper stepper, System sys, State & s);
   ```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Function template make_times_time_iterator_begin

boost::numeric::odeint::make_times_time_iterator_begin — Factory function for times_time_iterator. Constructs a begin iterator.

Synopsis

```
// In header: <boost/numeric/odeint/iterator/times_time_iterator.hpp>


template<typename Stepper, typename System, typename State, 
         typename TimeIterator> 
  times_time_iterator< Stepper, System, State, TimeIterator > 
  make_times_time_iterator_begin(Stepper stepper, System system, State & x, 
                                 TimeIterator t_start, TimeIterator t_end, 
                                 typename traits::time_type< Stepper >::type dt);
```

Description

Parameters:	`dt`	The initial time step.
	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`t_end`	End iterator of the sequence of evaluation time points.
	`t_start`	Begin iterator of the sequence of evaluation time points.
	`x`	The initial state. `const_step_iterator` stores a reference of s and changes its value during the iteration.
Returns:	The times_time iterator.	

Function template make_times_time_iterator_end

boost::numeric::odeint::make_times_time_iterator_end — Factory function for times_time_iterator. Constructs an end iterator.

Synopsis

```
// In header: <boost/numeric/odeint/iterator/times_time_iterator.hpp>


template<typename TimeIterator, typename Stepper, typename System, 
         typename State> 
  times_time_iterator< Stepper, System, State, TimeIterator > 
  make_times_time_iterator_end(Stepper stepper, System system, State & x);
```

Description

This function needs the TimeIterator type specifically defined as a template parameter.

Parameters:	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`x`	The initial state. `const_step_iterator` stores a reference of s and changes its value during the iteration.
Returns:	The times_time iterator.	

Function template make_times_time_range

boost::numeric::odeint::make_times_time_range — Factory function to construct a single pass range of times_time iterators. A range is here a pair of times_iterator.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/iterator/times_time_iterator.hpp>


template<typename Stepper, typename System, typename State,
         typename TimeIterator>
  std::pair< times_time_iterator< Stepper, System, State, TimeIterator >, times_time_iterat↵
or< Stepper, System, State, TimeIterator > >
  make_times_time_range(Stepper stepper, System system, State & x,
                        TimeIterator t_start, TimeIterator t_end,
                        typename traits::time_type< Stepper >::type dt);
```

Description

Parameters:	`dt`	The initial time step.
	`stepper`	The stepper to use during the iteration.
	`system`	The system function (ODE) to solve.
	`t_end`	End iterator of the sequence of evaluation time points.
	`t_start`	Begin iterator of the sequence of evaluation time points.
	`x`	The initial state. `const_step_iterator` store a reference of s and changes its value during the iteration.
Returns:	The times_time iterator range.	

Header <boost/numeric/odeint/stepper/adams_bashforth.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<size_t Steps, typename State, typename Value = double,
               typename Deriv = State, typename Time = Value,
               typename Algebra = typename algebra_dispatcher< State >::algebra_type,
              typename Operations = typename operations_dispatcher< State >::operations_type,
               typename Resizer = initially_resizer,
               typename InitializingStepper = runge_kutta4< State , Value , Deriv , Time , Al↵
gebra , Operations, Resizer > >
        class adams_bashforth;
    }
  }
}
```

Class template adams_bashforth

boost::numeric::odeint::adams_bashforth — The Adams-Bashforth multistep algorithm.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/adams_bashforth.hpp>

template<size_t Steps, typename State, typename Value = double, 
         typename Deriv = State, typename Time = Value, 
         typename Algebra = typename algebra_dispatcher< State >::algebra_type, 
         typename Operations = typename operations_dispatcher< State >::operations_type, 
         typename Resizer = initially_resizer, 
        typename InitializingStepper = runge_kutta4< State , Value , Deriv , Time , Algebra , Op↵
erations, Resizer > > 
class adams_bashforth : public algebra_stepper_base< Algebra, Operations > {
public:
  // types
  typedef State                                       state_type;               
  typedef state_wrapper< state_type >                 wrapped_state_type;       
  typedef Value                                       value_type;               
  typedef Deriv                                       deriv_type;               
  typedef state_wrapper< deriv_type >                 wrapped_deriv_type;       
  typedef Time                                        time_type;                
  typedef Resizer                                     resizer_type;             
  typedef stepper_tag                                 stepper_category;         
  typedef InitializingStepper                         initializing_stepper_type;
  typedef algebra_stepper_base< Algebra, Operations > algebra_stepper_base_type;
  typedef algebra_stepper_base_type::algebra_type     algebra_type;             
  typedef algebra_stepper_base_type::operations_type  operations_type;          
  typedef unsigned short                              order_type;               
  typedef unspecified                                 step_storage_type;        

  // construct/copy/destruct
  adams_bashforth(const algebra_type & = algebra_type());

  // public member functions
  order_type order(void) const;
  template<typename System, typename StateInOut> 
    void do_step(System, StateInOut &, time_type, time_type);
  template<typename System, typename StateInOut> 
    void do_step(System, const StateInOut &, time_type, time_type);
  template<typename System, typename StateIn, typename StateOut> 
    void do_step(System, const StateIn &, time_type, StateOut &, time_type);
  template<typename System, typename StateIn, typename StateOut> 
    void do_step(System, const StateIn &, time_type, const StateOut &, 
                 time_type);
  template<typename StateType> void adjust_size(const StateType &);
  const step_storage_type & step_storage(void) const;
  step_storage_type & step_storage(void);
  template<typename ExplicitStepper, typename System, typename StateIn> 
    void initialize(ExplicitStepper, System, StateIn &, time_type &, 
                    time_type);
  template<typename System, typename StateIn> 
    void initialize(System, StateIn &, time_type &, time_type);
  void reset(void);
  bool is_initialized(void) const;
  const initializing_stepper_type & initializing_stepper(void) const;
  initializing_stepper_type & initializing_stepper(void);

  // private member functions
  template<typename System, typename StateIn, typename StateOut> 
    void do_step_impl(System, const StateIn &, time_type, StateOut &, 
                      time_type);
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  template<typename StateIn> bool resize_impl(const StateIn &);

  // public data members
  static const size_t steps;
  static const order_type order_value;
};
```

Description

The Adams-Bashforth method is a multi-step algorithm with configurable step number. The step number is specified as template parameter Steps and it then uses the result from the previous Steps steps. See also en.wikipedia.org/wiki/Linear_multistep_method. Currently, a maximum of Steps=8 is supported. The method is explicit and fulfills the Stepper concept. Step size control or continuous output are not provided.

This class derives from algebra_base and inherits its interface via CRTP (current recurring template pattern). For more details see algebra_stepper_base.

Template Parameters

1.
```
size_t Steps
```

   The number of steps (maximal 8).

2.
```
typename State
```

   The state type.

3.
```
typename Value = double
```

   The value type.

4.
```
typename Deriv = State
```

   The type representing the time derivative of the state.

5.
```
typename Time = Value
```

   The time representing the independent variable - the time.

6.
```
typename Algebra = typename algebra_dispatcher< State >::algebra_type
```

   The algebra type.

7.
```
typename Operations = typename operations_dispatcher< State >::operations_type
```

   The operations type.

8.
```
typename Resizer = initially_resizer
```

   The resizer policy type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

9.
```
typename InitializingStepper = runge_kutta4< State , Value , Deriv , Time , Algebra , Opera↵
tions, Resizer >
```

The stepper for the first two steps.

`adams_bashforth` public construct/copy/destruct

1.
```
adams_bashforth(const algebra_type & algebra = algebra_type());
```

Constructs the `adams_bashforth` class. This constructor can be used as a default constructor if the algebra has a default constructor.

Parameters:	`algebra`	A copy of algebra is made and stored.

`adams_bashforth` public member functions

1.
```
order_type order(void) const;
```

Returns the order of the algorithm, which is equal to the number of steps.

Returns: order of the method.

2.
```
template<typename System, typename StateInOut>
  void do_step(System system, StateInOut & x, time_type t, time_type dt);
```

This method performs one step. It transforms the result in-place.

Parameters:	`dt`	The step size.
	`system`	The system function to solve, hence the r.h.s. of the ordinary differential equation. It must fulfill the Simple System concept.
	`t`	The value of the time, at which the step should be performed.
	`x`	The state of the ODE which should be solved. After calling do_step the result is updated in x.

3.
```
template<typename System, typename StateInOut>
  void do_step(System system, const StateInOut & x, time_type t, time_type dt);
```

Second version to solve the forwarding problem, can be called with Boost.Range as StateInOut.

4.
```
template<typename System, typename StateIn, typename StateOut>
  void do_step(System system, const StateIn & in, time_type t, StateOut & out,
               time_type dt);
```

The method performs one step with the stepper passed by Stepper. The state of the ODE is updated out-of-place.

Parameters:	`dt`	The step size.
	`in`	The state of the ODE which should be solved. in is not modified in this method
	`out`	The result of the step is written in out.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time, at which the step should be performed.

5.
```
template<typename System, typename StateIn, typename StateOut>
  void do_step(System system, const StateIn & in, time_type t,
               const StateOut & out, time_type dt);
```

Second version to solve the forwarding problem, can be called with Boost.Range as StateOut.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

6.

```
template<typename StateType> void adjust_size(const StateType & x);
```

Adjust the size of all temporaries in the stepper manually.

Parameters:		
	`x`	A state from which the size of the temporaries to be resized is deduced.

7.

```
const step_storage_type & step_storage(void) const;
```

Returns the storage of intermediate results.

Returns: The storage of intermediate results.

8.

```
step_storage_type & step_storage(void);
```

Returns the storage of intermediate results.

Returns: The storage of intermediate results.

9.

```
template<typename ExplicitStepper, typename System, typename StateIn> 
  void initialize(ExplicitStepper explicit_stepper, System system, 
                  StateIn & x, time_type & t, time_type dt);
```

Initialized the stepper. Does Steps-1 steps with the explicit_stepper to fill the buffer.

Parameters:		
	`dt`	The step size.
	`explicit_stepper`	the stepper used to fill the buffer of previous step results
	`system`	The system function to solve, hence the r.h.s. of the ordinary differential equation. It must fulfill the Simple System concept.
	`t`	The value of the time, at which the step should be performed.
	`x`	The state of the ODE which should be solved. After calling do_step the result is updated in x.

10.

```
template<typename System, typename StateIn> 
  void initialize(System system, StateIn & x, time_type & t, time_type dt);
```

Initialized the stepper. Does Steps-1 steps with an internal instance of InitializingStepper to fill the buffer.

> **Note**
>
> The state x and time t are updated to the values after Steps-1 initial steps.

Parameters:		
	`dt`	The step size.
	`system`	The system function to solve, hence the r.h.s. of the ordinary differential equation. It must fulfill the Simple System concept.
	`t`	The initial value of the time, updated in this method.
	`x`	The initial state of the ODE which should be solved, updated in this method.

11.

```
void reset(void);
```

Resets the internal buffer of the stepper.

12.

```
bool is_initialized(void) const;
```

Returns true if the stepper has been initialized.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Returns: bool true if stepper is initialized, false otherwise

13.
```
const initializing_stepper_type & initializing_stepper(void) const;
```

Returns the internal initializing stepper instance.

Returns: initializing_stepper

14.
```
initializing_stepper_type & initializing_stepper(void);
```

Returns the internal initializing stepper instance.

Returns: initializing_stepper

`adams_bashforth` private member functions

1.
```
template<typename System, typename StateIn, typename StateOut>
  void do_step_impl(System system, const StateIn & in, time_type t,
                    StateOut & out, time_type dt);
```

2.
```
template<typename StateIn> bool resize_impl(const StateIn & x);
```

Header <boost/numeric/odeint/stepper/adams_bashforth_moulton.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<size_t Steps, typename State, typename Value = double,
               typename Deriv = State, typename Time = Value,
               typename Algebra = typename algebra_dispatcher< State >::algebra_type,
               typename Operations = typename operations_dispatcher< State >::operations_type,
               typename Resizer = initially_resizer>
        class adams_bashforth_moulton;
    }
  }
}
```

Class template adams_bashforth_moulton

boost::numeric::odeint::adams_bashforth_moulton — The Adams-Bashforth-Moulton multistep algorithm.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/adams_bashforth_moulton.hpp>

template<size_t Steps, typename State, typename Value = double,
         typename Deriv = State, typename Time = Value,
         typename Algebra = typename algebra_dispatcher< State >::algebra_type,
         typename Operations = typename operations_dispatcher< State >::operations_type,
         typename Resizer = initially_resizer>
class adams_bashforth_moulton {
public:
  // types
  typedef State                             state_type;
  typedef state_wrapper< state_type > wrapped_state_type;
  typedef Value                             value_type;
  typedef Deriv                             deriv_type;
  typedef state_wrapper< deriv_type > wrapped_deriv_type;
  typedef Time                              time_type;
  typedef Algebra                           algebra_type;
  typedef Operations                        operations_type;
  typedef Resizer                           resizer_type;
  typedef stepper_tag                       stepper_category;
  typedef unsigned short                    order_type;

  // construct/copy/destruct
  adams_bashforth_moulton(void);
  adams_bashforth_moulton(const algebra_type &);

  // public member functions
  order_type order(void) const;
  template<typename System, typename StateInOut>
    void do_step(System, StateInOut &, time_type, time_type);
  template<typename System, typename StateInOut>
    void do_step(System, const StateInOut &, time_type, time_type);
  template<typename System, typename StateIn, typename StateOut>
    void do_step(System, const StateIn &, time_type, const StateOut &,
                 time_type);
  template<typename System, typename StateIn, typename StateOut>
    void do_step(System, const StateIn &, time_type, StateOut &, time_type);
  template<typename StateType> void adjust_size(const StateType &);
  template<typename ExplicitStepper, typename System, typename StateIn>
    void initialize(ExplicitStepper, System, StateIn &, time_type &,
                    time_type);
  template<typename System, typename StateIn>
    void initialize(System, StateIn &, time_type &, time_type);

  // private member functions
  template<typename System, typename StateInOut>
    void do_step_impl1(System, StateInOut &, time_type, time_type);
  template<typename System, typename StateIn, typename StateInOut>
    void do_step_impl2(System, StateIn const &, time_type, StateInOut &,
                       time_type);
  template<typename StateIn> bool resize_impl(const StateIn &);

  // public data members
  static const size_t steps;
  static const order_type order_value;
};
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Description

The Adams-Bashforth method is a multi-step predictor-corrector algorithm with configurable step number. The step number is specified as template parameter Steps and it then uses the result from the previous Steps steps. See also en.wikipedia.org/wiki/Linear_multistep_method. Currently, a maximum of Steps=8 is supported. The method is explicit and fulfills the Stepper concept. Step size control or continuous output are not provided.

This class derives from algebra_base and inherits its interface via CRTP (current recurring template pattern). For more details see algebra_stepper_base.

Template Parameters

1.
```
size_t Steps
```
   The number of steps (maximal 8).

2.
```
typename State
```
   The state type.

3.
```
typename Value = double
```
   The value type.

4.
```
typename Deriv = State
```
   The type representing the time derivative of the state.

5.
```
typename Time = Value
```
   The time representing the independent variable - the time.

6.
```
typename Algebra = typename algebra_dispatcher< State >::algebra_type
```
   The algebra type.

7.
```
typename Operations = typename operations_dispatcher< State >::operations_type
```
   The operations type.

8.
```
typename Resizer = initially_resizer
```
   The resizer policy type.

`adams_bashforth_moulton` public construct/copy/destruct

1.
```
adams_bashforth_moulton(void);
```
   Constructs the `adams_bashforth` class.

2.
```
adams_bashforth_moulton(const algebra_type & algebra);
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Constructs the `adams_bashforth` class. This constructor can be used as a default constructor if the algebra has a default constructor.

Parameters:	`algebra`	A copy of algebra is made and stored.

`adams_bashforth_moulton` public member functions

1.

```
order_type order(void) const;
```

Returns the order of the algorithm, which is equal to the number of steps+1.

Returns:	order of the method.

2.

```
template<typename System, typename StateInOut> 
  void do_step(System system, StateInOut & x, time_type t, time_type dt);
```

This method performs one step. It transforms the result in-place.

Parameters:	`dt`	The step size.
	`system`	The system function to solve, hence the r.h.s. of the ordinary differential equation. It must fulfill the Simple System concept.
	`t`	The value of the time, at which the step should be performed.
	`x`	The state of the ODE which should be solved. After calling do_step the result is updated in x.

3.

```
template<typename System, typename StateInOut> 
  void do_step(System system, const StateInOut & x, time_type t, time_type dt);
```

Second version to solve the forwarding problem, can be called with Boost.Range as StateInOut.

4.

```
template<typename System, typename StateIn, typename StateOut> 
  void do_step(System system, const StateIn & in, time_type t, 
               const StateOut & out, time_type dt);
```

The method performs one step with the stepper passed by Stepper. The state of the ODE is updated out-of-place.

Parameters:	`dt`	The step size.
	`in`	The state of the ODE which should be solved. in is not modified in this method
	`out`	The result of the step is written in out.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time, at which the step should be performed.

5.

```
template<typename System, typename StateIn, typename StateOut> 
  void do_step(System system, const StateIn & in, time_type t, StateOut & out, 
               time_type dt);
```

Second version to solve the forwarding problem, can be called with Boost.Range as StateOut.

6.

```
template<typename StateType> void adjust_size(const StateType & x);
```

Adjust the size of all temporaries in the stepper manually.

Parameters:	`x`	A state from which the size of the temporaries to be resized is deduced.

7.

```
template<typename ExplicitStepper, typename System, typename StateIn> 
  void initialize(ExplicitStepper explicit_stepper, System system, 
                  StateIn & x, time_type & t, time_type dt);
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Initialized the stepper. Does Steps-1 steps with the explicit_stepper to fill the buffer.

Note

The state x and time t are updated to the values after Steps-1 initial steps.

Parameters:	`dt`	The step size.
	`explicit_stepper`	the stepper used to fill the buffer of previous step results
	`system`	The system function to solve, hence the r.h.s. of the ordinary differential equation. It must fulfill the Simple System concept.
	`t`	The initial time, updated in this method.
	`x`	The initial state of the ODE which should be solved, updated after in this method.

8.

```
template<typename System, typename StateIn>
  void initialize(System system, StateIn & x, time_type & t, time_type dt);
```

Initialized the stepper. Does Steps-1 steps using the standard initializing stepper of the underlying `adams_bashforth` stepper.

Parameters:	`dt`	The step size.
	`system`	The system function to solve, hence the r.h.s. of the ordinary differential equation. It must fulfill the Simple System concept.
	`t`	The value of the time, at which the step should be performed.
	`x`	The state of the ODE which should be solved. After calling do_step the result is updated in x.

`adams_bashforth_moulton` private member functions

1.

```
template<typename System, typename StateInOut>
  void do_step_impl1(System system, StateInOut & x, time_type t, time_type dt);
```

2.

```
template<typename System, typename StateIn, typename StateInOut>
  void do_step_impl2(System system, StateIn const & in, time_type t,
                     StateInOut & out, time_type dt);
```

3.

```
template<typename StateIn> bool resize_impl(const StateIn & x);
```

Header <boost/numeric/odeint/stepper/adams_moulton.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<size_t Steps, typename State, typename Value = double,
               typename Deriv = State, typename Time = Value,
               typename Algebra = typename algebra_dispatcher< State >::algebra_type,
               typename Operations = typename operations_dispatcher< State >::operations_type,
               typename Resizer = initially_resizer>
        class adams_moulton;
    }
  }
}
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Class template adams_moulton

boost::numeric::odeint::adams_moulton

Synopsis

```
// In header: <boost/numeric/odeint/stepper/adams_moulton.hpp>

template<size_t Steps, typename State, typename Value = double, 
         typename Deriv = State, typename Time = Value, 
         typename Algebra = typename algebra_dispatcher< State >::algebra_type, 
         typename Operations = typename operations_dispatcher< State >::operations_type, 
         typename Resizer = initially_resizer> 
class adams_moulton {
public:
  // types
  typedef State                                                                          ↵
 state_type;
  typedef state_wrapper< state_type >                                                    ↵
 wrapped_state_type;
  typedef Value                                                                          ↵
 value_type;
  typedef Deriv                                                                           de↵
riv_type;
  typedef state_wrapper< deriv_type >                                                    ↵
 wrapped_deriv_type;
  typedef Time                                                                           ↵
 time_type;
  typedef Algebra                                                                         al↵
gebra_type;
  typedef Operations                                                                     opera↵
tions_type;
  typedef Resizer                                                                         res↵
izer_type;
  typedef stepper_tag                                                                     step↵
per_category;
  typedef adams_moulton< Steps, State, Value, Deriv, Time, Algebra, Operations, Resizer > step↵
per_type;
  typedef unsigned short                                                                  or↵
der_type;
  typedef unspecified                                                                step_stor↵
age_type;

  // construct/copy/destruct
  adams_moulton();
  adams_moulton(algebra_type &);
  adams_moulton & operator=(const adams_moulton &);

  // public member functions
  order_type order(void) const;
  template<typename System, typename StateInOut, typename StateIn, 
           typename ABBuf> 
    void do_step(System, StateInOut &, StateIn const &, time_type, time_type, 
                 const ABBuf &);
  template<typename System, typename StateInOut, typename StateIn, 
           typename ABBuf> 
    void do_step(System, const StateInOut &, StateIn const &, time_type, 
                 time_type, const ABBuf &);
  template<typename System, typename StateIn, typename PredIn, 
           typename StateOut, typename ABBuf> 
    void do_step(System, const StateIn &, const PredIn &, time_type, 
                 StateOut &, time_type, const ABBuf &);
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  template<typename System, typename StateIn, typename PredIn, 
           typename StateOut, typename ABBuf> 
    void do_step(System, const StateIn &, const PredIn &, time_type, 
                 const StateOut &, time_type, const ABBuf &);
  template<typename StateType> void adjust_size(const StateType &);
  algebra_type & algebra();
  const algebra_type & algebra() const;

  // private member functions
  template<typename System, typename StateIn, typename PredIn, 
           typename StateOut, typename ABBuf> 
    void do_step_impl(System, const StateIn &, const PredIn &, time_type, 
                      StateOut &, time_type, const ABBuf &);
  template<typename StateIn> bool resize_impl(const StateIn &);

  // public data members
  static const size_t steps;
  static const order_type order_value;
};
```

Description

`adams_moulton` public construct/copy/destruct

1.
```
adams_moulton();
```

2.
```
adams_moulton(algebra_type & algebra);
```

3.
```
adams_moulton & operator=(const adams_moulton & stepper);
```

`adams_moulton` public member functions

1.
```
order_type order(void) const;
```

2.
```
template<typename System, typename StateInOut, typename StateIn, 
         typename ABBuf> 
  void do_step(System system, StateInOut & x, StateIn const & pred, 
               time_type t, time_type dt, const ABBuf & buf);
```

3.
```
template<typename System, typename StateInOut, typename StateIn, 
         typename ABBuf> 
  void do_step(System system, const StateInOut & x, StateIn const & pred, 
               time_type t, time_type dt, const ABBuf & buf);
```

4.
```
template<typename System, typename StateIn, typename PredIn, 
         typename StateOut, typename ABBuf> 
  void do_step(System system, const StateIn & in, const PredIn & pred, 
               time_type t, StateOut & out, time_type dt, const ABBuf & buf);
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

5.
```
template<typename System, typename StateIn, typename PredIn, 
         typename StateOut, typename ABBuf> 
  void do_step(System system, const StateIn & in, const PredIn & pred, 
               time_type t, const StateOut & out, time_type dt, 
               const ABBuf & buf);
```

6.
```
template<typename StateType> void adjust_size(const StateType & x);
```

7.
```
algebra_type & algebra();
```

8.
```
const algebra_type & algebra() const;
```

`adams_moulton` private member functions

1.
```
template<typename System, typename StateIn, typename PredIn, 
         typename StateOut, typename ABBuf> 
  void do_step_impl(System system, const StateIn & in, const PredIn & pred, 
                    time_type t, StateOut & out, time_type dt, 
                    const ABBuf & buf);
```

2.
```
template<typename StateIn> bool resize_impl(const StateIn & x);
```

Header <boost/numeric/odeint/stepper/bulirsch_stoer.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename State, typename Value = double, 
               typename Deriv = State, typename Time = Value, 
               typename Algebra = typename algebra_dispatcher< State >::algebra_type, 
              typename Operations = typename operations_dispatcher< State >::operations_type, 
               typename Resizer = initially_resizer> 
        class bulirsch_stoer;
    }
  }
}
```

Class template bulirsch_stoer

boost::numeric::odeint::bulirsch_stoer — The Bulirsch-Stoer algorithm.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/bulirsch_stoer.hpp>

template<typename State, typename Value = double, typename Deriv = State, 
         typename Time = Value, 
         typename Algebra = typename algebra_dispatcher< State >::algebra_type, 
         typename Operations = typename operations_dispatcher< State >::operations_type, 
         typename Resizer = initially_resizer> 
class bulirsch_stoer {
public:
  // types
  typedef State      state_type;     
  typedef Value      value_type;     
  typedef Deriv      deriv_type;     
  typedef Time       time_type;      
  typedef Algebra    algebra_type;   
  typedef Operations operations_type;
  typedef Resizer    resizer_type;   

  // construct/copy/destruct
  bulirsch_stoer(value_type = 1E-6, value_type = 1E-6, value_type = 1.0, 
                 value_type = 1.0);

  // public member functions
  template<typename System, typename StateInOut> 
    controlled_step_result 
    try_step(System, StateInOut &, time_type &, time_type &);
  template<typename System, typename StateInOut> 
    controlled_step_result 
    try_step(System, const StateInOut &, time_type &, time_type &);
  template<typename System, typename StateInOut, typename DerivIn> 
    controlled_step_result 
    try_step(System, StateInOut &, const DerivIn &, time_type &, time_type &);
  template<typename System, typename StateIn, typename StateOut> 
    boost::disable_if< boost::is_same< StateIn, time_type >, controlled_step_result >::type 
    try_step(System, const StateIn &, time_type &, StateOut &, time_type &);
  template<typename System, typename StateIn, typename DerivIn, 
           typename StateOut> 
    controlled_step_result 
    try_step(System, const StateIn &, const DerivIn &, time_type &, 
             StateOut &, time_type &);
  void reset();
  template<typename StateIn> void adjust_size(const StateIn &);

  // private member functions
  template<typename StateIn> bool resize_m_dxdt(const StateIn &);
  template<typename StateIn> bool resize_m_xnew(const StateIn &);
  template<typename StateIn> bool resize_impl(const StateIn &);
  template<typename System, typename StateInOut> 
    controlled_step_result 
    try_step_v1(System, StateInOut &, time_type &, time_type &);
  template<typename StateInOut> 
    void extrapolate(size_t, state_table_type &, const value_matrix &, 
                     StateInOut &);
  time_type calc_h_opt(time_type, value_type, size_t) const;
  controlled_step_result 
  set_k_opt(size_t, const inv_time_vector &, const time_vector &, time_type &);
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  bool in_convergence_window(size_t) const;
  bool should_reject(value_type, size_t) const;

  // public data members
  static const size_t m_k_max;
};
```

Description

The Bulirsch-Stoer is a controlled stepper that adjusts both step size and order of the method. The algorithm uses the modified midpoint and a polynomial extrapolation compute the solution.

Template Parameters

1.
   ```
   typename State
   ```

   The state type.

2.
   ```
   typename Value = double
   ```

   The value type.

3.
   ```
   typename Deriv = State
   ```

   The type representing the time derivative of the state.

4.
   ```
   typename Time = Value
   ```

   The time representing the independent variable - the time.

5.
   ```
   typename Algebra = typename algebra_dispatcher< State >::algebra_type
   ```

   The algebra type.

6.
   ```
   typename Operations = typename operations_dispatcher< State >::operations_type
   ```

   The operations type.

7.
   ```
   typename Resizer = initially_resizer
   ```

   The resizer policy type.

`bulirsch_stoer` public construct/copy/destruct

1.
   ```
   bulirsch_stoer(value_type eps_abs = 1E-6, value_type eps_rel = 1E-6,
                  value_type factor_x = 1.0, value_type factor_dxdt = 1.0);
   ```

   Constructs the `bulirsch_stoer` class, including initialization of the error bounds.

Parameters:		
	`eps_abs`	Absolute tolerance level.
	`eps_rel`	Relative tolerance level.
	`factor_dxdt`	Factor for the weight of the derivative.
	`factor_x`	Factor for the weight of the state.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

`bulirsch_stoer` public member functions

1.
```
template<typename System, typename StateInOut> 
  controlled_step_result 
  try_step(System system, StateInOut & x, time_type & t, time_type & dt);
```

   Tries to perform one step.

   This method tries to do one step with step size dt. If the error estimate is to large, the step is rejected and the method returns fail and the step size dt is reduced. If the error estimate is acceptably small, the step is performed, success is returned and dt might be increased to make the steps as large as possible. This method also updates t if a step is performed. Also, the internal order of the stepper is adjusted if required.

Parameters:	`dt`	The step size. Updated.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time. Updated if the step is successful.
	`x`	The state of the ODE which should be solved. Overwritten if the step is successful.
Returns:	success if the step was accepted, fail otherwise.	

2.
```
template<typename System, typename StateInOut> 
  controlled_step_result 
  try_step(System system, const StateInOut & x, time_type & t, time_type & dt);
```

   Second version to solve the forwarding problem, can be used with Boost.Range as StateInOut.

3.
```
template<typename System, typename StateInOut, typename DerivIn> 
  controlled_step_result 
  try_step(System system, StateInOut & x, const DerivIn & dxdt, time_type & t, 
           time_type & dt);
```

   Tries to perform one step.

   This method tries to do one step with step size dt. If the error estimate is to large, the step is rejected and the method returns fail and the step size dt is reduced. If the error estimate is acceptably small, the step is performed, success is returned and dt might be increased to make the steps as large as possible. This method also updates t if a step is performed. Also, the internal order of the stepper is adjusted if required.

Parameters:	`dt`	The step size. Updated.
	`dxdt`	The derivative of state.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time. Updated if the step is successful.
	`x`	The state of the ODE which should be solved. Overwritten if the step is successful.
Returns:	success if the step was accepted, fail otherwise.	

4.
```
template<typename System, typename StateIn, typename StateOut> 
  boost::disable_if< boost::is_same< StateIn, time_type >, controlled_step_result >::type 
  try_step(System system, const StateIn & in, time_type & t, StateOut & out, 
           time_type & dt);
```

   Tries to perform one step.

   > **Note**
   >
   > This method is disabled if state_type=time_type to avoid ambiguity.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

This method tries to do one step with step size dt. If the error estimate is to large, the step is rejected and the method returns fail and the step size dt is reduced. If the error estimate is acceptably small, the step is performed, success is returned and dt might be increased to make the steps as large as possible. This method also updates t if a step is performed. Also, the internal order of the stepper is adjusted if required.

Parameters:	`dt`	The step size. Updated.
	`in`	The state of the ODE which should be solved.
	`out`	Used to store the result of the step.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time. Updated if the step is successful.
Returns:	success if the step was accepted, fail otherwise.	

5.
```
template<typename System, typename StateIn, typename DerivIn,
         typename StateOut>
  controlled_step_result
  try_step(System system, const StateIn & in, const DerivIn & dxdt,
           time_type & t, StateOut & out, time_type & dt);
```

Tries to perform one step.

This method tries to do one step with step size dt. If the error estimate is to large, the step is rejected and the method returns fail and the step size dt is reduced. If the error estimate is acceptably small, the step is performed, success is returned and dt might be increased to make the steps as large as possible. This method also updates t if a step is performed. Also, the internal order of the stepper is adjusted if required.

Parameters:	`dt`	The step size. Updated.
	`dxdt`	The derivative of state.
	`in`	The state of the ODE which should be solved.
	`out`	Used to store the result of the step.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time. Updated if the step is successful.
Returns:	success if the step was accepted, fail otherwise.	

6.
```
void reset();
```

Resets the internal state of the stepper.

7.
```
template<typename StateIn> void adjust_size(const StateIn & x);
```

Adjust the size of all temporaries in the stepper manually.

Parameters:	`x`	A state from which the size of the temporaries to be resized is deduced.

`bulirsch_stoer` private member functions

1.
```
template<typename StateIn> bool resize_m_dxdt(const StateIn & x);
```

2.
```
template<typename StateIn> bool resize_m_xnew(const StateIn & x);
```

3.
```
template<typename StateIn> bool resize_impl(const StateIn & x);
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

4.
```
template<typename System, typename StateInOut> 
  controlled_step_result 
  try_step_v1(System system, StateInOut & x, time_type & t, time_type & dt);
```

5.
```
template<typename StateInOut> 
  void extrapolate(size_t k, state_table_type & table, 
                   const value_matrix & coeff, StateInOut & xest);
```

6.
```
time_type calc_h_opt(time_type h, value_type error, size_t k) const;
```

7.
```
controlled_step_result 
set_k_opt(size_t k, const inv_time_vector & work, const time_vector & h_opt, 
          time_type & dt);
```

8.
```
bool in_convergence_window(size_t k) const;
```

9.
```
bool should_reject(value_type error, size_t k) const;
```

Header <boost/numeric/odeint/stepper/bulirsch_stoer_dense_out.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename State, typename Value = double, 
               typename Deriv = State, typename Time = Value, 
               typename Algebra = typename algebra_dispatcher< State >::algebra_type, 
               typename Operations = typename operations_dispatcher< State >::operations_type, 
               typename Resizer = initially_resizer> 
        class bulirsch_stoer_dense_out;
    }
  }
}
```

Class template bulirsch_stoer_dense_out

boost::numeric::odeint::bulirsch_stoer_dense_out — The Bulirsch-Stoer algorithm.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/bulirsch_stoer_dense_out.hpp>

template<typename State, typename Value = double, typename Deriv = State,
         typename Time = Value,
         typename Algebra = typename algebra_dispatcher< State >::algebra_type,
         typename Operations = typename operations_dispatcher< State >::operations_type,
         typename Resizer = initially_resizer>
class bulirsch_stoer_dense_out {
public:
  // types
  typedef State                    state_type;
  typedef Value                    value_type;
  typedef Deriv                    deriv_type;
  typedef Time                     time_type;
  typedef Algebra                  algebra_type;
  typedef Operations               operations_type;
  typedef Resizer                  resizer_type;
  typedef dense_output_stepper_tag stepper_category;

  // construct/copy/destruct
  bulirsch_stoer_dense_out(value_type = 1E-6, value_type = 1E-6,
                           value_type = 1.0, value_type = 1.0, bool = false);

  // public member functions
  template<typename System, typename StateIn, typename DerivIn,
           typename StateOut, typename DerivOut>
    controlled_step_result
    try_step(System, const StateIn &, const DerivIn &, time_type &,
             StateOut &, DerivOut &, time_type &);
  template<typename StateType>
    void initialize(const StateType &, const time_type &, const time_type &);
  template<typename System> std::pair< time_type, time_type > do_step(System);
  template<typename StateOut> void calc_state(time_type, StateOut &) const;
  const state_type & current_state(void) const;
  time_type current_time(void) const;
  const state_type & previous_state(void) const;
  time_type previous_time(void) const;
  time_type current_time_step(void) const;
  void reset();
  template<typename StateIn> void adjust_size(const StateIn &);

  // private member functions
  template<typename StateInOut, typename StateVector>
    void extrapolate(size_t, StateVector &, const value_matrix &,
                     StateInOut &, size_t = 0);
  template<typename StateVector>
    void extrapolate_dense_out(size_t, StateVector &, const value_matrix &,
                               size_t = 0);
  time_type calc_h_opt(time_type, value_type, size_t) const;
  bool in_convergence_window(size_t) const;
  bool should_reject(value_type, size_t) const;
  template<typename StateIn1, typename DerivIn1, typename StateIn2,
           typename DerivIn2>
    value_type prepare_dense_output(int, const StateIn1 &, const DerivIn1 &,
                                    const StateIn2 &, const DerivIn2 &,
                                    time_type);
  template<typename DerivIn>
    void calculate_finite_difference(size_t, size_t, value_type,
                                     const DerivIn &);
  template<typename StateOut>
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
    void do_interpolation(time_type, StateOut &) const;
  template<typename StateIn> bool resize_impl(const StateIn &);
  state_type & get_current_state(void);
  const state_type & get_current_state(void) const;
  state_type & get_old_state(void);
  const state_type & get_old_state(void) const;
  deriv_type & get_current_deriv(void);
  const deriv_type & get_current_deriv(void) const;
  deriv_type & get_old_deriv(void);
  const deriv_type & get_old_deriv(void) const;
  void toggle_current_state(void);

  // public data members
  static const size_t m_k_max;
};
```

Description

The Bulirsch-Stoer is a controlled stepper that adjusts both step size and order of the method. The algorithm uses the modified midpoint and a polynomial extrapolation compute the solution. This class also provides dense output facility.

Template Parameters

1. `typename State`

   The state type.

2. `typename Value = double`

   The value type.

3. `typename Deriv = State`

   The type representing the time derivative of the state.

4. `typename Time = Value`

   The time representing the independent variable - the time.

5. `typename Algebra = typename algebra_dispatcher< State >::algebra_type`

   The algebra type.

6. `typename Operations = typename operations_dispatcher< State >::operations_type`

   The operations type.

7. `typename Resizer = initially_resizer`

   The resizer policy type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

`bulirsch_stoer_dense_out` public construct/copy/destruct

1.
```
bulirsch_stoer_dense_out(value_type eps_abs = 1E-6, value_type eps_rel = 1E-6,
                         value_type factor_x = 1.0,
                         value_type factor_dxdt = 1.0,
                         bool control_interpolation = false);
```

Constructs the `bulirsch_stoer` class, including initialization of the error bounds.

Parameters:		
	`control_interpolation`	Set true to additionally control the error of the interpolation.
	`eps_abs`	Absolute tolerance level.
	`eps_rel`	Relative tolerance level.
	`factor_dxdt`	Factor for the weight of the derivative.
	`factor_x`	Factor for the weight of the state.

`bulirsch_stoer_dense_out` public member functions

1.
```
template<typename System, typename StateIn, typename DerivIn,
         typename StateOut, typename DerivOut>
  controlled_step_result
  try_step(System system, const StateIn & in, const DerivIn & dxdt,
           time_type & t, StateOut & out, DerivOut & dxdt_new,
           time_type & dt);
```

Tries to perform one step.

This method tries to do one step with step size dt. If the error estimate is to large, the step is rejected and the method returns fail and the step size dt is reduced. If the error estimate is acceptably small, the step is performed, success is returned and dt might be increased to make the steps as large as possible. This method also updates t if a step is performed. Also, the internal order of the stepper is adjusted if required.

Parameters:		
	`dt`	The step size. Updated.
	`dxdt`	The derivative of state.
	`in`	The state of the ODE which should be solved.
	`out`	Used to store the result of the step.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time. Updated if the step is successful.
Returns:	success if the step was accepted, fail otherwise.	

2.
```
template<typename StateType>
  void initialize(const StateType & x0, const time_type & t0,
                  const time_type & dt0);
```

Initializes the dense output stepper.

Parameters:		
	`dt0`	The initial time step.
	`t0`	The initial time.
	`x0`	The initial state.

3.
```
template<typename System>
  std::pair< time_type, time_type > do_step(System system);
```

Does one time step. This is the main method that should be used to integrate an ODE with this stepper.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Note

initialize has to be called before using this method to set the initial conditions x,t and the stepsize.

Parameters:	`system`	The system function to solve, hence the r.h.s. of the ordinary differential equation. It must fulfill the Simple System concept.
Returns:	Pair with start and end time of the integration step.	

4.
```
template<typename StateOut> void calc_state(time_type t, StateOut & x) const;
```

Calculates the solution at an intermediate point within the last step.

Parameters:	`t`	The time at which the solution should be calculated, has to be in the current time interval.
	`x`	The output variable where the result is written into.

5.
```
const state_type & current_state(void) const;
```

Returns the current state of the solution.

Returns: The current state of the solution x(t).

6.
```
time_type current_time(void) const;
```

Returns the current time of the solution.

Returns: The current time of the solution t.

7.
```
const state_type & previous_state(void) const;
```

Returns the last state of the solution.

Returns: The last state of the solution x(t-dt).

8.
```
time_type previous_time(void) const;
```

Returns the last time of the solution.

Returns: The last time of the solution t-dt.

9.
```
time_type current_time_step(void) const;
```

Returns the current step size.

Returns: The current step size.

10.
```
void reset();
```

Resets the internal state of the stepper.

11.
```
template<typename StateIn> void adjust_size(const StateIn & x);
```

Adjust the size of all temporaries in the stepper manually.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters:		
	`x`	A state from which the size of the temporaries to be resized is deduced.

`bulirsch_stoer_dense_out` private member functions

1.
```
template<typename StateInOut, typename StateVector> 
  void extrapolate(size_t k, StateVector & table, const value_matrix & coeff, 
                   StateInOut & xest, size_t order_start_index = 0);
```

2.
```
template<typename StateVector> 
  void extrapolate_dense_out(size_t k, StateVector & table, 
                             const value_matrix & coeff, 
                             size_t order_start_index = 0);
```

3.
```
time_type calc_h_opt(time_type h, value_type error, size_t k) const;
```

4.
```
bool in_convergence_window(size_t k) const;
```

5.
```
bool should_reject(value_type error, size_t k) const;
```

6.
```
template<typename StateIn1, typename DerivIn1, typename StateIn2, 
         typename DerivIn2> 
  value_type prepare_dense_output(int k, const StateIn1 & x_start, 
                                  const DerivIn1 & dxdt_start, 
                                  const StateIn2 &, const DerivIn2 &, 
                                  time_type dt);
```

7.
```
template<typename DerivIn> 
  void calculate_finite_difference(size_t j, size_t kappa, value_type fac, 
                                   const DerivIn & dxdt);
```

8.
```
template<typename StateOut> 
  void do_interpolation(time_type t, StateOut & out) const;
```

9.
```
template<typename StateIn> bool resize_impl(const StateIn & x);
```

10.
```
state_type & get_current_state(void);
```

11.
```
const state_type & get_current_state(void) const;
```

12.
```
state_type & get_old_state(void);
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

13.
    ```
    const state_type & get_old_state(void) const;
    ```

14.
    ```
    deriv_type & get_current_deriv(void);
    ```

15.
    ```
    const deriv_type & get_current_deriv(void) const;
    ```

16.
    ```
    deriv_type & get_old_deriv(void);
    ```

17.
    ```
    const deriv_type & get_old_deriv(void) const;
    ```

18.
    ```
    void toggle_current_state(void);
    ```

Header <boost/numeric/odeint/stepper/controlled_runge_kutta.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename ErrorStepper,
               typename ErrorChecker = default_error_checker< typename ErrorStep↵
per::value_type ,typename ErrorStepper::algebra_type ,typename ErrorStepper::operations_type >,
               typename Resizer = typename ErrorStepper::resizer_type,
               typename ErrorStepperCategory = typename ErrorStepper::stepper_category>
        class controlled_runge_kutta;

      template<typename ErrorStepper, typename ErrorChecker, typename Resizer>
        class controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_step↵
per_fsal_tag>;
      template<typename ErrorStepper, typename ErrorChecker, typename Resizer>
        class controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_step↵
per_tag>;

      template<typename Value, typename Algebra, typename Operations>
        class default_error_checker;
    }
  }
}
```

Class template controlled_runge_kutta

boost::numeric::odeint::controlled_runge_kutta

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/controlled_runge_kutta.hpp>

template<typename ErrorStepper,
        typename ErrorChecker = default_error_checker< typename ErrorStepper::value_type ,type↲
name ErrorStepper::algebra_type ,typename ErrorStepper::operations_type >,
         typename Resizer = typename ErrorStepper::resizer_type,
         typename ErrorStepperCategory = typename ErrorStepper::stepper_category>
class controlled_runge_kutta {
};
```

Description

Specializations

- Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>
- Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>

Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>

boost::numeric::odeint::controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag> — Implements step size control for Runge-Kutta FSAL steppers with error estimation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/controlled_runge_kutta.hpp>

template<typename ErrorStepper, typename ErrorChecker, typename Resizer> 
class controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_step↲
per_fsal_tag> {
public:
  // types
  typedef ErrorStepper                           stepper_type;    
  typedef stepper_type::state_type               state_type;      
  typedef stepper_type::value_type               value_type;      
  typedef stepper_type::deriv_type               deriv_type;      
  typedef stepper_type::time_type                time_type;       
  typedef stepper_type::algebra_type             algebra_type;    
  typedef stepper_type::operations_type          operations_type; 
  typedef Resizer                                resizer_type;    
  typedef ErrorChecker                           error_checker_type;
  typedef explicit_controlled_stepper_fsal_tag stepper_category;

  // construct/copy/destruct
  controlled_runge_kutta(const error_checker_type & = error_checker_type(), 
                         const stepper_type & = stepper_type());

  // public member functions
  template<typename System, typename StateInOut> 
    controlled_step_result 
    try_step(System, StateInOut &, time_type &, time_type &);
  template<typename System, typename StateInOut> 
    controlled_step_result 
    try_step(System, const StateInOut &, time_type &, time_type &);
  template<typename System, typename StateIn, typename StateOut> 
    boost::disable_if< boost::is_same< StateIn, time_type >, controlled_step_result >::type 
    try_step(System, const StateIn &, time_type &, StateOut &, time_type &);
  template<typename System, typename StateInOut, typename DerivInOut> 
    controlled_step_result 
    try_step(System, StateInOut &, DerivInOut &, time_type &, time_type &);
  template<typename System, typename StateIn, typename DerivIn, 
           typename StateOut, typename DerivOut> 
    controlled_step_result 
    try_step(System, const StateIn &, const DerivIn &, time_type &, 
             StateOut &, DerivOut &, time_type &);
  void reset(void);
  template<typename DerivIn> void initialize(const DerivIn &);
  template<typename System, typename StateIn> 
    void initialize(System, const StateIn &, time_type);
  bool is_initialized(void) const;
  template<typename StateType> void adjust_size(const StateType &);
  stepper_type & stepper(void);
  const stepper_type & stepper(void) const;

  // private member functions
  template<typename StateIn> bool resize_m_xerr_impl(const StateIn &);
  template<typename StateIn> bool resize_m_dxdt_impl(const StateIn &);
  template<typename StateIn> bool resize_m_dxdt_new_impl(const StateIn &);
  template<typename StateIn> bool resize_m_xnew_impl(const StateIn &);
  template<typename System, typename StateInOut> 
    controlled_step_result 
    try_step_v1(System, StateInOut &, time_type &, time_type &);
};
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Description

This class implements the step size control for FSAL Runge-Kutta steppers with error estimation.

Template Parameters

1.
```
typename ErrorStepper
```
The stepper type with error estimation, has to fulfill the ErrorStepper concept.

2.
```
typename ErrorChecker
```
The error checker

3.
```
typename Resizer
```
The resizer policy type.

`controlled_runge_kutta` public construct/copy/destruct

1.
```
controlled_runge_kutta(const error_checker_type & error_checker = error_checker_type(),
                       const stepper_type & stepper = stepper_type());
```
Constructs the controlled Runge-Kutta stepper.

Parameters:		
	`error_checker`	An instance of the error checker.
	`stepper`	An instance of the underlying stepper.

`controlled_runge_kutta` public member functions

1.
```
template<typename System, typename StateInOut>
  controlled_step_result
  try_step(System system, StateInOut & x, time_type & t, time_type & dt);
```
Tries to perform one step.

This method tries to do one step with step size dt. If the error estimate is to large, the step is rejected and the method returns fail and the step size dt is reduced. If the error estimate is acceptably small, the step is performed, success is returned and dt might be increased to make the steps as large as possible. This method also updates t if a step is performed.

Parameters:		
	`dt`	The step size. Updated.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time. Updated if the step is successful.
	`x`	The state of the ODE which should be solved. Overwritten if the step is successful.
Returns:	success if the step was accepted, fail otherwise.	

2.
```
template<typename System, typename StateInOut>
  controlled_step_result
  try_step(System system, const StateInOut & x, time_type & t, time_type & dt);
```
Tries to perform one step. Solves the forwarding problem and allows for using boost range as state_type.

This method tries to do one step with step size dt. If the error estimate is to large, the step is rejected and the method returns fail and the step size dt is reduced. If the error estimate is acceptably small, the step is performed, success is returned and dt might be increased to make the steps as large as possible. This method also updates t if a step is performed.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters:	`dt`	The step size. Updated.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time. Updated if the step is successful.
	`x`	The state of the ODE which should be solved. Overwritten if the step is successful. Can be a boost range.
Returns:	success if the step was accepted, fail otherwise.	

3.
```
template<typename System, typename StateIn, typename StateOut>
  boost::disable_if< boost::is_same< StateIn, time_type >, controlled_step_result >::type
  try_step(System system, const StateIn & in, time_type & t, StateOut & out,
           time_type & dt);
```

Tries to perform one step.

Note

This method is disabled if state_type=time_type to avoid ambiguity.

This method tries to do one step with step size dt. If the error estimate is to large, the step is rejected and the method returns fail and the step size dt is reduced. If the error estimate is acceptably small, the step is performed, success is returned and dt might be increased to make the steps as large as possible. This method also updates t if a step is performed.

Parameters:	`dt`	The step size. Updated.
	`in`	The state of the ODE which should be solved.
	`out`	Used to store the result of the step.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time. Updated if the step is successful.
Returns:	success if the step was accepted, fail otherwise.	

4.
```
template<typename System, typename StateInOut, typename DerivInOut>
  controlled_step_result
  try_step(System system, StateInOut & x, DerivInOut & dxdt, time_type & t,
           time_type & dt);
```

Tries to perform one step.

This method tries to do one step with step size dt. If the error estimate is to large, the step is rejected and the method returns fail and the step size dt is reduced. If the error estimate is acceptably small, the step is performed, success is returned and dt might be increased to make the steps as large as possible. This method also updates t if a step is performed.

Parameters:	`dt`	The step size. Updated.
	`dxdt`	The derivative of state.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time. Updated if the step is successful.
	`x`	The state of the ODE which should be solved. Overwritten if the step is successful.
Returns:	success if the step was accepted, fail otherwise.	

5.
```
template<typename System, typename StateIn, typename DerivIn,
         typename StateOut, typename DerivOut>
  controlled_step_result
  try_step(System system, const StateIn & in, const DerivIn & dxdt_in,
           time_type & t, StateOut & out, DerivOut & dxdt_out,
           time_type & dt);
```

Tries to perform one step.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

This method tries to do one step with step size dt. If the error estimate is to large, the step is rejected and the method returns fail and the step size dt is reduced. If the error estimate is acceptably small, the step is performed, success is returned and dt might be increased to make the steps as large as possible. This method also updates t if a step is performed.

Parameters:	`dt`	The step size. Updated.
	`in`	The state of the ODE which should be solved.
	`out`	Used to store the result of the step.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time. Updated if the step is successful.
Returns:	success if the step was accepted, fail otherwise.	

6.
```
void reset(void);
```

Resets the internal state of the underlying FSAL stepper.

7.
```
template<typename DerivIn> void initialize(const DerivIn & deriv);
```

Initializes the internal state storing an internal copy of the derivative.

Parameters:	`deriv`	The initial derivative of the ODE.

8.
```
template<typename System, typename StateIn> 
  void initialize(System system, const StateIn & x, time_type t);
```

Initializes the internal state storing an internal copy of the derivative.

Parameters:	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The initial time.
	`x`	The initial state of the ODE which should be solved.

9.
```
bool is_initialized(void) const;
```

Returns true if the stepper has been initialized, false otherwise.

Returns:	true, if the stepper has been initialized, false otherwise.

10.
```
template<typename StateType> void adjust_size(const StateType & x);
```

Adjust the size of all temporaries in the stepper manually.

Parameters:	`x`	A state from which the size of the temporaries to be resized is deduced.

11.
```
stepper_type & stepper(void);
```

Returns the instance of the underlying stepper.

Returns:	The instance of the underlying stepper.

12.
```
const stepper_type & stepper(void) const;
```

Returns the instance of the underlying stepper.

Returns:	The instance of the underlying stepper.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

`controlled_runge_kutta` private member functions

1.
```
template<typename StateIn> bool resize_m_xerr_impl(const StateIn & x);
```

2.
```
template<typename StateIn> bool resize_m_dxdt_impl(const StateIn & x);
```

3.
```
template<typename StateIn> bool resize_m_dxdt_new_impl(const StateIn & x);
```

4.
```
template<typename StateIn> bool resize_m_xnew_impl(const StateIn & x);
```

5.
```
template<typename System, typename StateInOut> 
  controlled_step_result 
  try_step_v1(System system, StateInOut & x, time_type & t, time_type & dt);
```

Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>

boost::numeric::odeint::controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag> — Implements step size control for Runge-Kutta steppers with error estimation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/controlled_runge_kutta.hpp>

template<typename ErrorStepper, typename ErrorChecker, typename Resizer> 
class controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag> {
public:
  // types
  typedef ErrorStepper                        stepper_type;    
  typedef stepper_type::state_type            state_type;      
  typedef stepper_type::value_type            value_type;      
  typedef stepper_type::deriv_type            deriv_type;      
  typedef stepper_type::time_type             time_type;       
  typedef stepper_type::algebra_type          algebra_type;    
  typedef stepper_type::operations_type       operations_type; 
  typedef Resizer                             resizer_type;    
  typedef ErrorChecker                        error_checker_type;
  typedef explicit_controlled_stepper_tag stepper_category;

  // construct/copy/destruct
  controlled_runge_kutta(const error_checker_type & = error_checker_type(), 
                         const stepper_type & = stepper_type());

  // public member functions
  template<typename System, typename StateInOut> 
    controlled_step_result 
    try_step(System, StateInOut &, time_type &, time_type &);
  template<typename System, typename StateInOut> 
    controlled_step_result 
    try_step(System, const StateInOut &, time_type &, time_type &);
  template<typename System, typename StateInOut, typename DerivIn> 
    controlled_step_result 
    try_step(System, StateInOut &, const DerivIn &, time_type &, time_type &);
  template<typename System, typename StateIn, typename StateOut> 
    boost::disable_if< boost::is_same< StateIn, time_type >, controlled_step_result >::type 
    try_step(System, const StateIn &, time_type &, StateOut &, time_type &);
  template<typename System, typename StateIn, typename DerivIn, 
           typename StateOut> 
    controlled_step_result 
    try_step(System, const StateIn &, const DerivIn &, time_type &, 
             StateOut &, time_type &);
  value_type last_error(void) const;
  template<typename StateType> void adjust_size(const StateType &);
  stepper_type & stepper(void);
  const stepper_type & stepper(void) const;

  // private member functions
  template<typename System, typename StateInOut> 
    controlled_step_result 
    try_step_v1(System, StateInOut &, time_type &, time_type &);
  template<typename StateIn> bool resize_m_xerr_impl(const StateIn &);
  template<typename StateIn> bool resize_m_dxdt_impl(const StateIn &);
  template<typename StateIn> bool resize_m_xnew_impl(const StateIn &);
};
```

Description

This class implements the step size control for standard Runge-Kutta steppers with error estimation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Template Parameters

1.
   ```
   typename ErrorStepper
   ```

   The stepper type with error estimation, has to fulfill the ErrorStepper concept.

2.
   ```
   typename ErrorChecker
   ```

   The error checker

3.
   ```
   typename Resizer
   ```

   The resizer policy type.

`controlled_runge_kutta` public construct/copy/destruct

1.
   ```
   controlled_runge_kutta(const error_checker_type & error_checker = error_checker_type(),
                          const stepper_type & stepper = stepper_type());
   ```

   Constructs the controlled Runge-Kutta stepper.

Parameters:	`error_checker`	An instance of the error checker.
	`stepper`	An instance of the underlying stepper.

`controlled_runge_kutta` public member functions

1.
   ```
   template<typename System, typename StateInOut>
     controlled_step_result
     try_step(System system, StateInOut & x, time_type & t, time_type & dt);
   ```

   Tries to perform one step.

   This method tries to do one step with step size dt. If the error estimate is to large, the step is rejected and the method returns fail and the step size dt is reduced. If the error estimate is acceptably small, the step is performed, success is returned and dt might be increased to make the steps as large as possible. This method also updates t if a step is performed.

Parameters:	`dt`	The step size. Updated.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time. Updated if the step is successful.
	`x`	The state of the ODE which should be solved. Overwritten if the step is successful.
Returns:	success if the step was accepted, fail otherwise.	

2.
   ```
   template<typename System, typename StateInOut>
     controlled_step_result
     try_step(System system, const StateInOut & x, time_type & t, time_type & dt);
   ```

   Tries to perform one step. Solves the forwarding problem and allows for using boost range as state_type.

   This method tries to do one step with step size dt. If the error estimate is to large, the step is rejected and the method returns fail and the step size dt is reduced. If the error estimate is acceptably small, the step is performed, success is returned and dt might be increased to make the steps as large as possible. This method also updates t if a step is performed.

Parameters:	`dt`	The step size. Updated.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time. Updated if the step is successful.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

	`x`	The state of the ODE which should be solved. Overwritten if the step is successful. Can be a boost range.
Returns:	success if the step was accepted, fail otherwise.	

3.

```
template<typename System, typename StateInOut, typename DerivIn>
  controlled_step_result
  try_step(System system, StateInOut & x, const DerivIn & dxdt, time_type & t,
           time_type & dt);
```

Tries to perform one step.

This method tries to do one step with step size dt. If the error estimate is to large, the step is rejected and the method returns fail and the step size dt is reduced. If the error estimate is acceptably small, the step is performed, success is returned and dt might be increased to make the steps as large as possible. This method also updates t if a step is performed.

Parameters:	`dt`	The step size. Updated.
	`dxdt`	The derivative of state.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time. Updated if the step is successful.
	`x`	The state of the ODE which should be solved. Overwritten if the step is successful.
Returns:	success if the step was accepted, fail otherwise.	

4.

```
template<typename System, typename StateIn, typename StateOut>
  boost::disable_if< boost::is_same< StateIn, time_type >, controlled_step_result >::type
  try_step(System system, const StateIn & in, time_type & t, StateOut & out,
           time_type & dt);
```

Tries to perform one step.

Note

This method is disabled if state_type=time_type to avoid ambiguity.

This method tries to do one step with step size dt. If the error estimate is to large, the step is rejected and the method returns fail and the step size dt is reduced. If the error estimate is acceptably small, the step is performed, success is returned and dt might be increased to make the steps as large as possible. This method also updates t if a step is performed.

Parameters:	`dt`	The step size. Updated.
	`in`	The state of the ODE which should be solved.
	`out`	Used to store the result of the step.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time. Updated if the step is successful.
Returns:	success if the step was accepted, fail otherwise.	

5.

```
template<typename System, typename StateIn, typename DerivIn,
         typename StateOut>
  controlled_step_result
  try_step(System system, const StateIn & in, const DerivIn & dxdt,
           time_type & t, StateOut & out, time_type & dt);
```

Tries to perform one step.

This method tries to do one step with step size dt. If the error estimate is to large, the step is rejected and the method returns fail and the step size dt is reduced. If the error estimate is acceptably small, the step is performed, success is returned and dt might be increased to make the steps as large as possible. This method also updates t if a step is performed.

Parameters:	`dt`	The step size. Updated.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

	`dxdt`	The derivative of state.
	`in`	The state of the ODE which should be solved.
	`out`	Used to store the result of the step.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time. Updated if the step is successful.
Returns:	success if the step was accepted, fail otherwise.	

6.
```
value_type last_error(void) const;
```

Returns the error of the last step.

returns The last error of the step.

7.
```
template<typename StateType> void adjust_size(const StateType & x);
```

Adjust the size of all temporaries in the stepper manually.

Parameters:	`x`	A state from which the size of the temporaries to be resized is deduced.

8.
```
stepper_type & stepper(void);
```

Returns the instance of the underlying stepper.

Returns: The instance of the underlying stepper.

9.
```
const stepper_type & stepper(void) const;
```

Returns the instance of the underlying stepper.

Returns: The instance of the underlying stepper.

`controlled_runge_kutta` private member functions

1.
```
template<typename System, typename StateInOut> 
  controlled_step_result 
  try_step_v1(System system, StateInOut & x, time_type & t, time_type & dt);
```

2.
```
template<typename StateIn> bool resize_m_xerr_impl(const StateIn & x);
```

3.
```
template<typename StateIn> bool resize_m_dxdt_impl(const StateIn & x);
```

4.
```
template<typename StateIn> bool resize_m_xnew_impl(const StateIn & x);
```

Class template default_error_checker

boost::numeric::odeint::default_error_checker — The default error checker to be used with Runge-Kutta error steppers.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/controlled_runge_kutta.hpp>

template<typename Value, typename Algebra, typename Operations>
class default_error_checker {
public:
  // types
  typedef Value      value_type;
  typedef Algebra    algebra_type;
  typedef Operations operations_type;

  // construct/copy/destruct
  default_error_checker(value_type = static_cast< value_type >(1.0e-6),
                        value_type = static_cast< value_type >(1.0e-6),
                        value_type = static_cast< value_type >(1),
                        value_type = static_cast< value_type >(1));

  // public member functions
  template<typename State, typename Deriv, typename Err, typename Time>
    value_type error(const State &, const Deriv &, Err &, Time) const;
  template<typename State, typename Deriv, typename Err, typename Time>
    value_type error(algebra_type &, const State &, const Deriv &, Err &,
                     Time) const;
};
```

Description

This class provides the default mechanism to compare the error estimates reported by Runge-Kutta error steppers with user defined error bounds. It is used by the controlled_runge_kutta steppers.

Template Parameters

1.
   ```
   typename Value
   ```

   The value type.

2.
   ```
   typename Algebra
   ```

   The algebra type.

3.
   ```
   typename Operations
   ```

   The operations type.

`default_error_checker` public construct/copy/destruct

1.
   ```
   default_error_checker(value_type eps_abs = static_cast< value_type >(1.0e-6),
                         value_type eps_rel = static_cast< value_type >(1.0e-6),
                         value_type a_x = static_cast< value_type >(1),
                         value_type a_dxdt = static_cast< value_type >(1));
   ```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

`default_error_checker` public member functions

1.
```
template<typename State, typename Deriv, typename Err, typename Time> 
  value_type error(const State & x_old, const Deriv & dxdt_old, Err & x_err, 
                   Time dt) const;
```

2.
```
template<typename State, typename Deriv, typename Err, typename Time> 
  value_type error(algebra_type & algebra, const State & x_old, 
                   const Deriv & dxdt_old, Err & x_err, Time dt) const;
```

Header <boost/numeric/odeint/stepper/controlled_step_result.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {

      // Enum representing the return values of the controlled steppers.
      enum controlled_step_result { success, fail };
    }
  }
}
```

Header <boost/numeric/odeint/stepper/dense_output_runge_kutta.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename Stepper, 
               typename StepperCategory = typename Stepper::stepper_category> 
        class dense_output_runge_kutta;

      template<typename Stepper> 
        class dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>;
      template<typename Stepper> 
        class dense_output_runge_kutta<Stepper, stepper_tag>;
    }
  }
}
```

Class template dense_output_runge_kutta

boost::numeric::odeint::dense_output_runge_kutta

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/dense_output_runge_kutta.hpp>

template<typename Stepper,
         typename StepperCategory = typename Stepper::stepper_category>
class dense_output_runge_kutta {
};
```

Description

Specializations

- Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>
- Class template dense_output_runge_kutta<Stepper, stepper_tag>

Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>

boost::numeric::odeint::dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag> — The class representing dense-output Runge-Kutta steppers with FSAL property.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/dense_output_runge_kutta.hpp>

template<typename Stepper>
class dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag> {
public:
  // types
  typedef Stepper                                 controlled_stepper_type;
  typedef controlled_stepper_type::stepper_type stepper_type;
  typedef stepper_type::state_type                state_type;
  typedef stepper_type::wrapped_state_type        wrapped_state_type;
  typedef stepper_type::value_type                value_type;
  typedef stepper_type::deriv_type                deriv_type;
  typedef stepper_type::wrapped_deriv_type        wrapped_deriv_type;
  typedef stepper_type::time_type                 time_type;
  typedef stepper_type::algebra_type              algebra_type;
  typedef stepper_type::operations_type           operations_type;
  typedef stepper_type::resizer_type              resizer_type;
  typedef dense_output_stepper_tag                stepper_category;
  typedef dense_output_runge_kutta< Stepper >     dense_output_stepper_type;

  // construct/copy/destruct
  dense_output_runge_kutta(const controlled_stepper_type & = controlled_stepper_type());

  // public member functions
  template<typename StateType>
    void initialize(const StateType &, time_type, time_type);
  template<typename System> std::pair< time_type, time_type > do_step(System);
  template<typename StateOut> void calc_state(time_type, StateOut &) const;
  template<typename StateOut>
    void calc_state(time_type, const StateOut &) const;
  template<typename StateIn> bool resize(const StateIn &);
  template<typename StateType> void adjust_size(const StateType &);
  const state_type & current_state(void) const;
  time_type current_time(void) const;
  const state_type & previous_state(void) const;
  time_type previous_time(void) const;
  time_type current_time_step(void) const;

  // private member functions
  state_type & get_current_state(void);
  const state_type & get_current_state(void) const;
  state_type & get_old_state(void);
  const state_type & get_old_state(void) const;
  deriv_type & get_current_deriv(void);
  const deriv_type & get_current_deriv(void) const;
  deriv_type & get_old_deriv(void);
  const deriv_type & get_old_deriv(void) const;
  void toggle_current_state(void);
};
```

Description

The interface is the same as for dense_output_runge_kutta< Stepper , stepper_tag >. This class provides dense output functionality based on methods with step size controlled

Template Parameters

1.

```
typename Stepper
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

The stepper type of the underlying algorithm.

`dense_output_runge_kutta` public construct/copy/destruct

1.
```
dense_output_runge_kutta(const controlled_stepper_type & stepper = controlled_stepper_type());
```

`dense_output_runge_kutta` public member functions

1.
```
template<typename StateType> 
  void initialize(const StateType & x0, time_type t0, time_type dt0);
```

2.
```
template<typename System> 
  std::pair< time_type, time_type > do_step(System system);
```

3.
```
template<typename StateOut> void calc_state(time_type t, StateOut & x) const;
```

4.
```
template<typename StateOut> 
  void calc_state(time_type t, const StateOut & x) const;
```

5.
```
template<typename StateIn> bool resize(const StateIn & x);
```

6.
```
template<typename StateType> void adjust_size(const StateType & x);
```

7.
```
const state_type & current_state(void) const;
```

8.
```
time_type current_time(void) const;
```

9.
```
const state_type & previous_state(void) const;
```

10.
```
time_type previous_time(void) const;
```

11.
```
time_type current_time_step(void) const;
```

`dense_output_runge_kutta` private member functions

1.
```
state_type & get_current_state(void);
```

2.
```
const state_type & get_current_state(void) const;
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

3.
```
state_type & get_old_state(void);
```

4.
```
const state_type & get_old_state(void) const;
```

5.
```
deriv_type & get_current_deriv(void);
```

6.
```
const deriv_type & get_current_deriv(void) const;
```

7.
```
deriv_type & get_old_deriv(void);
```

8.
```
const deriv_type & get_old_deriv(void) const;
```

9.
```
void toggle_current_state(void);
```

Class template dense_output_runge_kutta<Stepper, stepper_tag>

boost::numeric::odeint::dense_output_runge_kutta<Stepper, stepper_tag> — The class representing dense-output Runge-Kutta steppers.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/dense_output_runge_kutta.hpp>

template<typename Stepper>
class dense_output_runge_kutta<Stepper, stepper_tag> {
public:
  // types
  typedef Stepper                            stepper_type;
  typedef stepper_type::state_type           state_type;
  typedef stepper_type::wrapped_state_type   wrapped_state_type;
  typedef stepper_type::value_type           value_type;
  typedef stepper_type::deriv_type           deriv_type;
  typedef stepper_type::wrapped_deriv_type   wrapped_deriv_type;
  typedef stepper_type::time_type            time_type;
  typedef stepper_type::algebra_type         algebra_type;
  typedef stepper_type::operations_type      operations_type;
  typedef stepper_type::resizer_type         resizer_type;
  typedef dense_output_stepper_tag           stepper_category;
  typedef dense_output_runge_kutta< Stepper > dense_output_stepper_type;

  // construct/copy/destruct
  dense_output_runge_kutta(const stepper_type & = stepper_type());

  // public member functions
  template<typename StateType>
    void initialize(const StateType &, time_type, time_type);
  template<typename System> std::pair< time_type, time_type > do_step(System);
  template<typename StateOut> void calc_state(time_type, StateOut &) const;
  template<typename StateOut>
    void calc_state(time_type, const StateOut &) const;
  template<typename StateType> void adjust_size(const StateType &);
  const state_type & current_state(void) const;
  time_type current_time(void) const;
  const state_type & previous_state(void) const;
  time_type previous_time(void) const;

  // private member functions
  state_type & get_current_state(void);
  const state_type & get_current_state(void) const;
  state_type & get_old_state(void);
  const state_type & get_old_state(void) const;
  void toggle_current_state(void);
  template<typename StateIn> bool resize_impl(const StateIn &);
};
```

Description

Note

In this stepper, the initialize method has to be called before using the do_step method.

The dense-output functionality allows to interpolate the solution between subsequent integration points using intermediate results obtained during the computation. This version works based on a normal stepper without step-size control.

Template Parameters

1.
```
typename Stepper
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

The stepper type of the underlying algorithm.

`dense_output_runge_kutta` public construct/copy/destruct

1.
```
dense_output_runge_kutta(const stepper_type & stepper = stepper_type());
```

Constructs the dense_output_runge_kutta class. An instance of the underlying stepper can be provided.

Parameters:	`stepper`	An instance of the underlying stepper.

`dense_output_runge_kutta` public member functions

1.
```
template<typename StateType>
  void initialize(const StateType & x0, time_type t0, time_type dt0);
```

Initializes the stepper. Has to be called before do_step can be used to set the initial conditions and the step size.

Parameters:	`dt0`	The step size.
	`t0`	The initial time, at which the step should be performed.
	`x0`	The initial state of the ODE which should be solved.

2.
```
template<typename System>
  std::pair< time_type, time_type > do_step(System system);
```

Does one time step.

> **Note**
>
> initialize has to be called before using this method to set the initial conditions x,t and the stepsize.

Parameters:	`system`	The system function to solve, hence the r.h.s. of the ordinary differential equation. It must fulfill the Simple System concept.
Returns:	Pair with start and end time of the integration step.	

3.
```
template<typename StateOut> void calc_state(time_type t, StateOut & x) const;
```

Calculates the solution at an intermediate point.

Parameters:	`t`	The time at which the solution should be calculated, has to be in the current time interval.
	`x`	The output variable where the result is written into.

4.
```
template<typename StateOut>
  void calc_state(time_type t, const StateOut & x) const;
```

Calculates the solution at an intermediate point. Solves the forwarding problem.

Parameters:	`t`	The time at which the solution should be calculated, has to be in the current time interval.
	`x`	The output variable where the result is written into, can be a boost range.

5.
```
template<typename StateType> void adjust_size(const StateType & x);
```

Adjust the size of all temporaries in the stepper manually.

Parameters:	`x`	A state from which the size of the temporaries to be resized is deduced.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

6.
```
const state_type & current_state(void) const;
```

Returns the current state of the solution.

Returns: The current state of the solution x(t).

7.
```
time_type current_time(void) const;
```

Returns the current time of the solution.

Returns: The current time of the solution t.

8.
```
const state_type & previous_state(void) const;
```

Returns the last state of the solution.

Returns: The last state of the solution x(t-dt).

9.
```
time_type previous_time(void) const;
```

Returns the last time of the solution.

Returns: The last time of the solution t-dt.

`dense_output_runge_kutta` private member functions

1.
```
state_type & get_current_state(void);
```

2.
```
const state_type & get_current_state(void) const;
```

3.
```
state_type & get_old_state(void);
```

4.
```
const state_type & get_old_state(void) const;
```

5.
```
void toggle_current_state(void);
```

6.
```
template<typename StateIn> bool resize_impl(const StateIn & x);
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Header <boost/numeric/odeint/stepper/euler.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename State, typename Value = double,
               typename Deriv = State, typename Time = Value,
               typename Algebra = typename algebra_dispatcher< State >::algebra_type,
               typename Operations = typename operations_dispatcher< State >::operations_type,
               typename Resizer = initially_resizer>
        class euler;
    }
  }
}
```

Class template euler

boost::numeric::odeint::euler — An implementation of the Euler method.

Synopsis

```
// In header: <boost/numeric/odeint/stepper/euler.hpp>

template<typename State, typename Value = double, typename Deriv = State,
         typename Time = Value,
         typename Algebra = typename algebra_dispatcher< State >::algebra_type,
         typename Operations = typename operations_dispatcher< State >::operations_type,
         typename Resizer = initially_resizer>
class euler : public explicit_stepper_base {
public:
  // types
  typedef explicit_stepper_base< euler< ... >,... > stepper_base_type;
  typedef stepper_base_type::state_type             state_type;
  typedef stepper_base_type::value_type             value_type;
  typedef stepper_base_type::deriv_type             deriv_type;
  typedef stepper_base_type::time_type              time_type;
  typedef stepper_base_type::algebra_type           algebra_type;
  typedef stepper_base_type::operations_type        operations_type;
  typedef stepper_base_type::resizer_type           resizer_type;

  // construct/copy/destruct
  euler(const algebra_type & = algebra_type());

  // public member functions
  template<typename System, typename StateIn, typename DerivIn,
           typename StateOut>
    void do_step_impl(System, const StateIn &, const DerivIn &, time_type,
                      StateOut &, time_type);
  template<typename StateOut, typename StateIn1, typename StateIn2>
    void calc_state(StateOut &, time_type, const StateIn1 &, time_type,
                    const StateIn2 &, time_type) const;
  template<typename StateType> void adjust_size(const StateType &);
};
```

Description

The Euler method is a very simply solver for ordinary differential equations. This method should not be used for real applications. It is only useful for demonstration purposes. Step size control is not provided but trivial continuous output is available.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

This class derives from explicit_stepper_base and inherits its interface via CRTP (current recurring template pattern), see explicit_stepper_base

Template Parameters

1.
   ```
   typename State
   ```

   The state type.

2.
   ```
   typename Value = double
   ```

   The value type.

3.
   ```
   typename Deriv = State
   ```

   The type representing the time derivative of the state.

4.
   ```
   typename Time = Value
   ```

   The time representing the independent variable - the time.

5.
   ```
   typename Algebra = typename algebra_dispatcher< State >::algebra_type
   ```

   The algebra type.

6.
   ```
   typename Operations = typename operations_dispatcher< State >::operations_type
   ```

   The operations type.

7.
   ```
   typename Resizer = initially_resizer
   ```

   The resizer policy type.

`euler` public construct/copy/destruct

1.
   ```
   euler(const algebra_type & algebra = algebra_type());
   ```

   Constructs the euler class. This constructor can be used as a default constructor of the algebra has a default constructor.

Parameters:	`algebra`	A copy of algebra is made and stored inside explicit_stepper_base.

`euler` public member functions

1.
   ```
   template<typename System, typename StateIn, typename DerivIn,
            typename StateOut>
     void do_step_impl(System system, const StateIn & in, const DerivIn & dxdt,
                       time_type t, StateOut & out, time_type dt);
   ```

   This method performs one step. The derivative `dxdt` of `in` at the time `t` is passed to the method. The result is updated out of place, hence the input is in `in` and the output in `out`. Access to this step functionality is provided by explicit_stepper_base and `do_step_impl` should not be called directly.

Parameters:	`dt`	The step size.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

`dxdt`	The derivative of x at t.
`in`	The state of the ODE which should be solved. in is not modified in this method
`out`	The result of the step is written in out.
`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
`t`	The value of the time, at which the step should be performed.

2.
```
template<typename StateOut, typename StateIn1, typename StateIn2> 
  void calc_state(StateOut & x, time_type t, const StateIn1 & old_state, 
                  time_type t_old, const StateIn2 & current_state, 
                  time_type t_new) const;
```

This method is used for continuous output and it calculates the state `x` at a time `t` from the knowledge of two states `old_state` and `current_state` at time points `t_old` and `t_new`.

3.
```
template<typename StateType> void adjust_size(const StateType & x);
```

Adjust the size of all temporaries in the stepper manually.

Parameters:		
	`x`	A state from which the size of the temporaries to be resized is deduced.

Header <boost/numeric/odeint/stepper/explicit_error_generic_rk.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<size_t StageCount, size_t Order, size_t StepperOrder, 
               size_t ErrorOrder, typename State, typename Value = double, 
               typename Deriv = State, typename Time = Value, 
               typename Algebra = typename algebra_dispatcher< State >::algebra_type, 
              typename Operations = typename operations_dispatcher< State >::operations_type, 
               typename Resizer = initially_resizer> 
        class explicit_error_generic_rk;
    }
  }
}
```

Class template explicit_error_generic_rk

boost::numeric::odeint::explicit_error_generic_rk — A generic implementation of explicit Runge-Kutta algorithms with error estimation. This class is as a base class for all explicit Runge-Kutta steppers with error estimation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/explicit_error_generic_rk.hpp>

template<size_t StageCount, size_t Order, size_t StepperOrder,
         size_t ErrorOrder, typename State, typename Value = double,
         typename Deriv = State, typename Time = Value,
         typename Algebra = typename algebra_dispatcher< State >::algebra_type,
         typename Operations = typename operations_dispatcher< State >::operations_type,
         typename Resizer = initially_resizer>
class explicit_error_generic_rk : public explicit_error_stepper_base {
public:
  // types
  typedef explicit_stepper_base< ... >          stepper_base_type;
  typedef stepper_base_type::state_type         state_type;
  typedef stepper_base_type::wrapped_state_type wrapped_state_type;
  typedef stepper_base_type::value_type         value_type;
  typedef stepper_base_type::deriv_type         deriv_type;
  typedef stepper_base_type::wrapped_deriv_type wrapped_deriv_type;
  typedef stepper_base_type::time_type          time_type;
  typedef stepper_base_type::algebra_type       algebra_type;
  typedef stepper_base_type::operations_type    operations_type;
  typedef stepper_base_type::resizer_type       resizer_type;
  typedef unspecified                           rk_algorithm_type;
  typedef rk_algorithm_type::coef_a_type        coef_a_type;
  typedef rk_algorithm_type::coef_b_type        coef_b_type;
  typedef rk_algorithm_type::coef_c_type        coef_c_type;

  // construct/copy/destruct
  explicit_error_generic_rk(const coef_a_type &, const coef_b_type &,
                            const coef_b_type &, const coef_c_type &,
                            const algebra_type & = algebra_type());

  // public member functions
  template<typename System, typename StateIn, typename DerivIn,
           typename StateOut, typename Err>
    void do_step_impl(System, const StateIn &, const DerivIn &, time_type,
                      StateOut &, time_type, Err &);
  template<typename System, typename StateIn, typename DerivIn,
           typename StateOut>
    void do_step_impl(System, const StateIn &, const DerivIn &, time_type,
                      StateOut &, time_type);
  template<typename StateIn> void adjust_size(const StateIn &);

  // private member functions
  template<typename StateIn> bool resize_impl(const StateIn &);

  // public data members
  static const size_t stage_count;
};
```

Description

This class implements the explicit Runge-Kutta algorithms with error estimation in a generic way. The Butcher tableau is passed to the stepper which constructs the stepper scheme with the help of a template-metaprogramming algorithm. ToDo : Add example!

This class derives explicit_error_stepper_base which provides the stepper interface.

Template Parameters

1.
```
size_t StageCount
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

The number of stages of the Runge-Kutta algorithm.

2.
```
size_t Order
```

The order of a stepper if the stepper is used without error estimation.

3.
```
size_t StepperOrder
```

The order of a step if the stepper is used with error estimation. Usually Order and StepperOrder have the same value.

4.
```
size_t ErrorOrder
```

The order of the error step if the stepper is used with error estimation.

5.
```
typename State
```

The type representing the state of the ODE.

6.
```
typename Value = double
```

The floating point type which is used in the computations.

7.
```
typename Deriv = State
```

8.
```
typename Time = Value
```

The type representing the independent variable - the time - of the ODE.

9.
```
typename Algebra = typename algebra_dispatcher< State >::algebra_type
```

The algebra type.

10.
```
typename Operations = typename operations_dispatcher< State >::operations_type
```

The operations type.

11.
```
typename Resizer = initially_resizer
```

The resizer policy type.

`explicit_error_generic_rk` public construct/copy/destruct

1.
```
explicit_error_generic_rk(const coef_a_type & a, const coef_b_type & b,
                          const coef_b_type & b2, const coef_c_type & c,
                          const algebra_type & algebra = algebra_type());
```

Constructs the explicit_error_generik_rk class with the given parameters a, b, b2 and c. See examples section for details on the coefficients.

Parameters:		
	`a`	Triangular matrix of parameters b in the Butcher tableau.
	`algebra`	A copy of algebra is made and stored inside explicit_stepper_base.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

`b`	Last row of the butcher tableau.
`b2`	Parameters for lower-order evaluation to estimate the error.
`c`	Parameters to calculate the time points in the Butcher tableau.

`explicit_error_generic_rk` public member functions

1.
```
template<typename System, typename StateIn, typename DerivIn, 
         typename StateOut, typename Err> 
  void do_step_impl(System system, const StateIn & in, const DerivIn & dxdt, 
                    time_type t, StateOut & out, time_type dt, Err & xerr);
```

   This method performs one step. The derivative `dxdt` of `in` at the time `t` is passed to the method. The result is updated out-of-place, hence the input is in `in` and the output in `out`. Futhermore, an estimation of the error is stored in `xerr`. `do_step_impl` is used by explicit_error_stepper_base.

Parameters:		
	`dt`	The step size.
	`dxdt`	The derivative of x at t.
	`in`	The state of the ODE which should be solved. in is not modified in this method
	`out`	The result of the step is written in out.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time, at which the step should be performed.
	`xerr`	The result of the error estimation is written in xerr.

2.
```
template<typename System, typename StateIn, typename DerivIn, 
         typename StateOut> 
  void do_step_impl(System system, const StateIn & in, const DerivIn & dxdt, 
                    time_type t, StateOut & out, time_type dt);
```

   This method performs one step. The derivative `dxdt` of `in` at the time `t` is passed to the method. The result is updated out-of-place, hence the input is in `in` and the output in `out`. Access to this step functionality is provided by explicit_stepper_base and `do_step_impl` should not be called directly.

Parameters:		
	`dt`	The step size.
	`dxdt`	The derivative of x at t.
	`in`	The state of the ODE which should be solved. in is not modified in this method
	`out`	The result of the step is written in out.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time, at which the step should be performed.

3.
```
template<typename StateIn> void adjust_size(const StateIn & x);
```

   Adjust the size of all temporaries in the stepper manually.

Parameters:		
	`x`	A state from which the size of the temporaries to be resized is deduced.

`explicit_error_generic_rk` private member functions

1.
```
template<typename StateIn> bool resize_impl(const StateIn & x);
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Header <boost/numeric/odeint/stepper/explicit_generic_rk.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<size_t StageCount, size_t Order, typename State,
               typename Value, typename Deriv, typename Time,
               typename Algebra, typename Operations, typename Resizer>
        class explicit_generic_rk;
    }
  }
}
```

Class template explicit_generic_rk

boost::numeric::odeint::explicit_generic_rk — A generic implementation of explicit Runge-Kutta algorithms. This class is as a base class for all explicit Runge-Kutta steppers.

Synopsis

```
// In header: <boost/numeric/odeint/stepper/explicit_generic_rk.hpp>

template<size_t StageCount, size_t Order, typename State, typename Value,
         typename Deriv, typename Time, typename Algebra, typename Operations,
         typename Resizer>
class explicit_generic_rk : public explicit_stepper_base {
public:
  // types
  typedef explicit_stepper_base< ... >          stepper_base_type;
  typedef stepper_base_type::state_type         state_type;
  typedef stepper_base_type::wrapped_state_type wrapped_state_type;
  typedef stepper_base_type::value_type         value_type;
  typedef stepper_base_type::deriv_type         deriv_type;
  typedef stepper_base_type::wrapped_deriv_type wrapped_deriv_type;
  typedef stepper_base_type::time_type          time_type;
  typedef stepper_base_type::algebra_type       algebra_type;
  typedef stepper_base_type::operations_type    operations_type;
  typedef stepper_base_type::resizer_type       resizer_type;
  typedef unspecified                           rk_algorithm_type;
  typedef rk_algorithm_type::coef_a_type        coef_a_type;
  typedef rk_algorithm_type::coef_b_type        coef_b_type;
  typedef rk_algorithm_type::coef_c_type        coef_c_type;

  // construct/copy/destruct
  explicit_generic_rk(const coef_a_type &, const coef_b_type &,
                      const coef_c_type &,
                      const algebra_type & = algebra_type());

  // public member functions
  template<typename System, typename StateIn, typename DerivIn,
           typename StateOut>
    void do_step_impl(System, const StateIn &, const DerivIn &, time_type,
                      StateOut &, time_type);
  template<typename StateIn> void adjust_size(const StateIn &);

  // private member functions
  template<typename StateIn> bool resize_impl(const StateIn &);
};
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Description

This class implements the explicit Runge-Kutta algorithms without error estimation in a generic way. The Butcher tableau is passed to the stepper which constructs the stepper scheme with the help of a template-metaprogramming algorithm. ToDo : Add example!

This class derives explicit_stepper_base which provides the stepper interface.

Template Parameters

1.
```
size_t StageCount
```
The number of stages of the Runge-Kutta algorithm.

2.
```
size_t Order
```
The order of the stepper.

3.
```
typename State
```
The type representing the state of the ODE.

4.
```
typename Value
```
The floating point type which is used in the computations.

5.
```
typename Deriv
```

6.
```
typename Time
```
The type representing the independent variable - the time - of the ODE.

7.
```
typename Algebra
```
The algebra type.

8.
```
typename Operations
```
The operations type.

9.
```
typename Resizer
```
The resizer policy type.

`explicit_generic_rk` public construct/copy/destruct

1.
```
explicit_generic_rk(const coef_a_type & a, const coef_b_type & b,
                    const coef_c_type & c,
                    const algebra_type & algebra = algebra_type());
```
Constructs the `explicit_generic_rk` class. See examples section for details on the coefficients.

Parameters:	`a`	Triangular matrix of parameters b in the Butcher tableau.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

`algebra`	A copy of algebra is made and stored inside explicit_stepper_base.
`b`	Last row of the butcher tableau.
`c`	Parameters to calculate the time points in the Butcher tableau.

`explicit_generic_rk` public member functions

1.

```
template<typename System, typename StateIn, typename DerivIn, 
         typename StateOut> 
  void do_step_impl(System system, const StateIn & in, const DerivIn & dxdt, 
                    time_type t, StateOut & out, time_type dt);
```

This method performs one step. The derivative `dxdt` of `in` at the time `t` is passed to the method. The result is updated out of place, hence the input is in `in` and the output in `out`. Access to this step functionality is provided by explicit_stepper_base and `do_step_impl` should not be called directly.

Parameters:		
	`dt`	The step size.
	`dxdt`	The derivative of x at t.
	`in`	The state of the ODE which should be solved. in is not modified in this method
	`out`	The result of the step is written in out.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time, at which the step should be performed.

2.

```
template<typename StateIn> void adjust_size(const StateIn & x);
```

Adjust the size of all temporaries in the stepper manually.

Parameters:		
	`x`	A state from which the size of the temporaries to be resized is deduced.

`explicit_generic_rk` private member functions

1.

```
template<typename StateIn> bool resize_impl(const StateIn & x);
```

Header <boost/numeric/odeint/stepper/implicit_euler.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename ValueType, typename Resizer = initially_resizer> 
        class implicit_euler;
    }
  }
}
```

Class template implicit_euler

boost::numeric::odeint::implicit_euler

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/implicit_euler.hpp>

template<typename ValueType, typename Resizer = initially_resizer> 
class implicit_euler {
public:
  // types
  typedef ValueType                                          value_type;         
  typedef value_type                                         time_type;          
  typedef boost::numeric::ublas::vector< value_type >        state_type;         
  typedef state_wrapper< state_type >                        wrapped_state_type; 
  typedef state_type                                         deriv_type;         
  typedef state_wrapper< deriv_type >                        wrapped_deriv_type; 
  typedef boost::numeric::ublas::matrix< value_type >        matrix_type;        
  typedef state_wrapper< matrix_type >                       wrapped_matrix_type;
  typedef boost::numeric::ublas::permutation_matrix< size_t > pmatrix_type;       
  typedef state_wrapper< pmatrix_type >                      wrapped_pmatrix_type;
  typedef Resizer                                            resizer_type;       
  typedef stepper_tag                                        stepper_category;   
  typedef implicit_euler< ValueType, Resizer >               stepper_type;       

  // construct/copy/destruct
  implicit_euler(value_type = 1E-6);

  // public member functions
  template<typename System> 
    void do_step(System, state_type &, time_type, time_type);
  template<typename StateType> void adjust_size(const StateType &);

  // private member functions
  template<typename StateIn> bool resize_impl(const StateIn &);
  void solve(state_type &, matrix_type &);
};
```

Description

`implicit_euler` public construct/copy/destruct

1.
```
implicit_euler(value_type epsilon = 1E-6);
```

`implicit_euler` public member functions

1.
```
template<typename System> 
  void do_step(System system, state_type & x, time_type t, time_type dt);
```

2.
```
template<typename StateType> void adjust_size(const StateType & x);
```

`implicit_euler` private member functions

1.
```
template<typename StateIn> bool resize_impl(const StateIn & x);
```

2.
```
void solve(state_type & x, matrix_type & m);
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Header <boost/numeric/odeint/stepper/modified_midpoint.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename State, typename Value = double,
               typename Deriv = State, typename Time = Value,
               typename Algebra = typename algebra_dispatcher< State >::algebra_type,
               typename Operations = typename operations_dispatcher< State >::operations_type,
               typename Resizer = initially_resizer>
        class modified_midpoint;
      template<typename State, typename Value = double,
               typename Deriv = State, typename Time = Value,
               typename Algebra = typename algebra_dispatcher< State >::algebra_type,
               typename Operations = typename operations_dispatcher< State >::operations_type,
               typename Resizer = initially_resizer>
        class modified_midpoint_dense_out;
    }
  }
}
```

Class template modified_midpoint

boost::numeric::odeint::modified_midpoint

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/modified_midpoint.hpp>

template<typename State, typename Value = double, typename Deriv = State, 
         typename Time = Value, 
         typename Algebra = typename algebra_dispatcher< State >::algebra_type, 
         typename Operations = typename operations_dispatcher< State >::operations_type, 
         typename Resizer = initially_resizer> 
class modified_midpoint : public explicit_stepper_base {
public:
  // types
  typedef explicit_stepper_base< modified_midpoint< State, Value, Deriv, Time, Algebra, Opera↵
tions, Resizer >, 2, State, Value, Deriv, Time, Algebra, Operations, Resizer > stepper_base_type;
  typedef stepper_base_type::state_type                                                      ↵
                                                                            state_type;      
  typedef stepper_base_type::wrapped_state_type                                              ↵
                                                                            wrapped_state_type;
  typedef stepper_base_type::value_type                                                      ↵
                                                                            value_type;      
  typedef stepper_base_type::deriv_type                                                      ↵
                                                                            deriv_type;      
  typedef stepper_base_type::wrapped_deriv_type                                              ↵
                                                                            wrapped_deriv_type;
  typedef stepper_base_type::time_type                                                       ↵
                                                                            time_type;       
  typedef stepper_base_type::algebra_type                                                    ↵
                                                                            algebra_type;    
  typedef stepper_base_type::operations_type                                                 ↵
                                                                            operations_type; 
  typedef stepper_base_type::resizer_type                                                    ↵
                                                                            resizer_type;    
  typedef stepper_base_type::stepper_type                                                    ↵
                                                                            stepper_type;    

  // construct/copy/destruct
  modified_midpoint(unsigned short = 2, const algebra_type & = algebra_type());

  // public member functions
  template<typename System, typename StateIn, typename DerivIn, 
           typename StateOut> 
    void do_step_impl(System, const StateIn &, const DerivIn &, time_type, 
                      StateOut &, time_type);
  void set_steps(unsigned short);
  unsigned short steps(void) const;
  template<typename StateIn> void adjust_size(const StateIn &);

  // private member functions
  template<typename StateIn> bool resize_impl(const StateIn &);
};
```

Description

Implementation of the modified midpoint method with a configurable number of intermediate steps. This class is used by the Bulirsch-Stoer algorithm and is not meant for direct usage.

`modified_midpoint` public construct/copy/destruct

1.

```
modified_midpoint(unsigned short steps = 2, 
                  const algebra_type & algebra = algebra_type());
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

`modified_midpoint` public member functions

1.
```
template<typename System, typename StateIn, typename DerivIn, 
         typename StateOut> 
  void do_step_impl(System system, const StateIn & in, const DerivIn & dxdt, 
                    time_type t, StateOut & out, time_type dt);
```

2.
```
void set_steps(unsigned short steps);
```

3.
```
unsigned short steps(void) const;
```

4.
```
template<typename StateIn> void adjust_size(const StateIn & x);
```

`modified_midpoint` private member functions

1.
```
template<typename StateIn> bool resize_impl(const StateIn & x);
```

Class template modified_midpoint_dense_out

boost::numeric::odeint::modified_midpoint_dense_out

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/modified_midpoint.hpp>

template<typename State, typename Value = double, typename Deriv = State, 
         typename Time = Value, 
         typename Algebra = typename algebra_dispatcher< State >::algebra_type, 
         typename Operations = typename operations_dispatcher< State >::operations_type, 
         typename Resizer = initially_resizer> 
class modified_midpoint_dense_out {
public:
  // types
  typedef State ↲
  state_type;
  typedef Value ↲
  value_type;
  typedef Deriv ↲
  deriv_type;
  typedef Time ↲
  time_type;
  typedef Algebra ↲
  algebra_type;
  typedef Operations ↲
  operations_type;
  typedef Resizer ↲
  resizer_type;
  typedef state_wrapper< state_type > ↲
  wrapped_state_type;
  typedef state_wrapper< deriv_type > ↲
  wrapped_deriv_type;
  typedef modified_midpoint_dense_out< State, Value, Deriv, Time, Algebra, Operations, Res↲
izer > stepper_type;
  typedef std::vector< wrapped_deriv_type > ↲
  deriv_table_type;

  // construct/copy/destruct
  modified_midpoint_dense_out(unsigned short = 2, 
                              const algebra_type & = algebra_type());

  // public member functions
  template<typename System, typename StateIn, typename DerivIn, 
           typename StateOut> 
    void do_step(System, const StateIn &, const DerivIn &, time_type, 
                 StateOut &, time_type, state_type &, deriv_table_type &);
  void set_steps(unsigned short);
  unsigned short steps(void) const;
  template<typename StateIn> bool resize(const StateIn &);
  template<typename StateIn> void adjust_size(const StateIn &);
};
```

Description

Implementation of the modified midpoint method with a configurable number of intermediate steps. This class is used by the dense output Bulirsch-Stoer algorithm and is not meant for direct usage.

Note

This stepper is for internal use only and does not meet any stepper concept.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

`modified_midpoint_dense_out` public construct/copy/destruct

1.
```
modified_midpoint_dense_out(unsigned short steps = 2,
                            const algebra_type & algebra = algebra_type());
```

`modified_midpoint_dense_out` public member functions

1.
```
template<typename System, typename StateIn, typename DerivIn,
         typename StateOut>
  void do_step(System system, const StateIn & in, const DerivIn & dxdt,
               time_type t, StateOut & out, time_type dt, state_type & x_mp,
               deriv_table_type & derivs);
```

2.
```
void set_steps(unsigned short steps);
```

3.
```
unsigned short steps(void) const;
```

4.
```
template<typename StateIn> bool resize(const StateIn & x);
```

5.
```
template<typename StateIn> void adjust_size(const StateIn & x);
```

Header <boost/numeric/odeint/stepper/rosenbrock4.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename Value> struct default_rosenbrock_coefficients;

      template<typename Value,
               typename Coefficients = default_rosenbrock_coefficients< Value >,
               typename Resizer = initially_resizer>
        class rosenbrock4;
    }
  }
}
```

Struct template default_rosenbrock_coefficients

boost::numeric::odeint::default_rosenbrock_coefficients

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/rosenbrock4.hpp>

template<typename Value>
struct default_rosenbrock_coefficients {
  // types
  typedef Value          value_type;
  typedef unsigned short order_type;

  // construct/copy/destruct
  default_rosenbrock_coefficients(void);

  // public data members
  const value_type gamma;
  const value_type d1;
  const value_type d2;
  const value_type d3;
  const value_type d4;
  const value_type c2;
  const value_type c3;
  const value_type c4;
  const value_type c21;
  const value_type a21;
  const value_type c31;
  const value_type c32;
  const value_type a31;
  const value_type a32;
  const value_type c41;
  const value_type c42;
  const value_type c43;
  const value_type a41;
  const value_type a42;
  const value_type a43;
  const value_type c51;
  const value_type c52;
  const value_type c53;
  const value_type c54;
  const value_type a51;
  const value_type a52;
  const value_type a53;
  const value_type a54;
  const value_type c61;
  const value_type c62;
  const value_type c63;
  const value_type c64;
  const value_type c65;
  const value_type d21;
  const value_type d22;
  const value_type d23;
  const value_type d24;
  const value_type d25;
  const value_type d31;
  const value_type d32;
  const value_type d33;
  const value_type d34;
  const value_type d35;
  static const order_type stepper_order;
  static const order_type error_order;
};
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Description

`default_rosenbrock_coefficients` public construct/copy/destruct

1.
```
default_rosenbrock_coefficients(void);
```

Class template rosenbrock4

boost::numeric::odeint::rosenbrock4

Synopsis

```
// In header: <boost/numeric/odeint/stepper/rosenbrock4.hpp>

template<typename Value,
         typename Coefficients = default_rosenbrock_coefficients< Value >,
         typename Resizer = initially_resizer>
class rosenbrock4 {
public:
  // types
  typedef Value                                                 value_type;
  typedef boost::numeric::ublas::vector< value_type >           state_type;
  typedef state_type                                            deriv_type;
  typedef value_type                                            time_type;
  typedef boost::numeric::ublas::matrix< value_type >           matrix_type;
  typedef boost::numeric::ublas::permutation_matrix< size_t > pmatrix_type;
  typedef Resizer                                               resizer_type;
  typedef Coefficients                                          rosenbrock_coefficients;
  typedef stepper_tag                                           stepper_category;
  typedef unsigned short                                        order_type;
  typedef state_wrapper< state_type >                           wrapped_state_type;
  typedef state_wrapper< deriv_type >                           wrapped_deriv_type;
  typedef state_wrapper< matrix_type >                          wrapped_matrix_type;
  typedef state_wrapper< pmatrix_type >                         wrapped_pmatrix_type;
  typedef rosenbrock4< Value, Coefficients, Resizer >           stepper_type;

  // construct/copy/destruct
  rosenbrock4(void);

  // public member functions
  order_type order() const;
  template<typename System>
    void do_step(System, const state_type &, time_type, state_type &,
                 time_type, state_type &);
  template<typename System>
    void do_step(System, state_type &, time_type, time_type, state_type &);
  template<typename System>
    void do_step(System, const state_type &, time_type, state_type &,
                 time_type);
  template<typename System>
    void do_step(System, state_type &, time_type, time_type);
  void prepare_dense_output();
  void calc_state(time_type, state_type &, const state_type &, time_type,
                  const state_type &, time_type);
  template<typename StateType> void adjust_size(const StateType &);

  // protected member functions
  template<typename StateIn> bool resize_impl(const StateIn &);
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  template<typename StateIn> bool resize_x_err(const StateIn &);

  // public data members
  static const order_type stepper_order;
  static const order_type error_order;
};
```

Description

`rosenbrock4` public construct/copy/destruct

1.
   ```
   rosenbrock4(void);
   ```

`rosenbrock4` public member functions

1.
   ```
   order_type order() const;
   ```

2.
   ```
   template<typename System> 
     void do_step(System system, const state_type & x, time_type t, 
                  state_type & xout, time_type dt, state_type & xerr);
   ```

3.
   ```
   template<typename System> 
     void do_step(System system, state_type & x, time_type t, time_type dt, 
                  state_type & xerr);
   ```

4.
   ```
   template<typename System> 
     void do_step(System system, const state_type & x, time_type t, 
                  state_type & xout, time_type dt);
   ```

5.
   ```
   template<typename System> 
     void do_step(System system, state_type & x, time_type t, time_type dt);
   ```

6.
   ```
   void prepare_dense_output();
   ```

7.
   ```
   void calc_state(time_type t, state_type & x, const state_type & x_old, 
                   time_type t_old, const state_type & x_new, time_type t_new);
   ```

8.
   ```
   template<typename StateType> void adjust_size(const StateType & x);
   ```

`rosenbrock4` protected member functions

1.
   ```
   template<typename StateIn> bool resize_impl(const StateIn & x);
   ```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

2.
   ```
   template<typename StateIn> bool resize_x_err(const StateIn & x);
   ```

Header <boost/numeric/odeint/stepper/rosenbrock4_controller.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename Stepper> class rosenbrock4_controller;
    }
  }
}
```

Class template rosenbrock4_controller

boost::numeric::odeint::rosenbrock4_controller

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/rosenbrock4_controller.hpp>

template<typename Stepper> 
class rosenbrock4_controller {
public:
  // types
  typedef Stepper                               stepper_type;      
  typedef stepper_type::value_type              value_type;        
  typedef stepper_type::state_type              state_type;        
  typedef stepper_type::wrapped_state_type      wrapped_state_type;
  typedef stepper_type::time_type               time_type;         
  typedef stepper_type::deriv_type              deriv_type;        
  typedef stepper_type::wrapped_deriv_type      wrapped_deriv_type;
  typedef stepper_type::resizer_type            resizer_type;      
  typedef controlled_stepper_tag                stepper_category;  
  typedef rosenbrock4_controller< Stepper >     controller_type;   

  // construct/copy/destruct
  rosenbrock4_controller(value_type = 1.0e-6, value_type = 1.0e-6, 
                         const stepper_type & = stepper_type());

  // public member functions
  value_type error(const state_type &, const state_type &, const state_type &);
  value_type last_error(void) const;
  template<typename System> 
    boost::numeric::odeint::controlled_step_result 
    try_step(System, state_type &, time_type &, time_type &);
  template<typename System> 
    boost::numeric::odeint::controlled_step_result 
    try_step(System, const state_type &, time_type &, state_type &, 
             time_type &);
  template<typename StateType> void adjust_size(const StateType &);
  stepper_type & stepper(void);
  const stepper_type & stepper(void) const;

  // private member functions
  template<typename StateIn> bool resize_m_xerr(const StateIn &);
  template<typename StateIn> bool resize_m_xnew(const StateIn &);
};
```

Description

`rosenbrock4_controller` public construct/copy/destruct

1.
```
rosenbrock4_controller(value_type atol = 1.0e-6, value_type rtol = 1.0e-6, 
                       const stepper_type & stepper = stepper_type());
```

`rosenbrock4_controller` public member functions

1.
```
value_type error(const state_type & x, const state_type & xold, 
                 const state_type & xerr);
```

2.
```
value_type last_error(void) const;
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

3.
```
template<typename System>
  boost::numeric::odeint::controlled_step_result
  try_step(System sys, state_type & x, time_type & t, time_type & dt);
```

4.
```
template<typename System>
  boost::numeric::odeint::controlled_step_result
  try_step(System sys, const state_type & x, time_type & t, state_type & xout,
           time_type & dt);
```

5.
```
template<typename StateType> void adjust_size(const StateType & x);
```

6.
```
stepper_type & stepper(void);
```

7.
```
const stepper_type & stepper(void) const;
```

`rosenbrock4_controller` private member functions

1.
```
template<typename StateIn> bool resize_m_xerr(const StateIn & x);
```

2.
```
template<typename StateIn> bool resize_m_xnew(const StateIn & x);
```

Header <boost/numeric/odeint/stepper/rosenbrock4_dense_output.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename ControlledStepper> class rosenbrock4_dense_output;
    }
  }
}
```

Class template rosenbrock4_dense_output

boost::numeric::odeint::rosenbrock4_dense_output

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/rosenbrock4_dense_output.hpp>

template<typename ControlledStepper> 
class rosenbrock4_dense_output {
public:
  // types
  typedef ControlledStepper                                    controlled_stepper_type;
  typedef controlled_stepper_type::stepper_type                stepper_type;           
  typedef stepper_type::value_type                             value_type;             
  typedef stepper_type::state_type                             state_type;             
  typedef stepper_type::wrapped_state_type                     wrapped_state_type;     
  typedef stepper_type::time_type                              time_type;              
  typedef stepper_type::deriv_type                             deriv_type;             
  typedef stepper_type::wrapped_deriv_type                     wrapped_deriv_type;     
  typedef stepper_type::resizer_type                           resizer_type;           
  typedef dense_output_stepper_tag                             stepper_category;       
  typedef rosenbrock4_dense_output< ControlledStepper > dense_output_stepper_type;

  // construct/copy/destruct
  rosenbrock4_dense_output(const controlled_stepper_type & = controlled_stepper_type());

  // public member functions
  template<typename StateType> 
    void initialize(const StateType &, time_type, time_type);
  template<typename System> std::pair< time_type, time_type > do_step(System);
  template<typename StateOut> void calc_state(time_type, StateOut &);
  template<typename StateOut> void calc_state(time_type, const StateOut &);
  template<typename StateType> void adjust_size(const StateType &);
  const state_type & current_state(void) const;
  time_type current_time(void) const;
  const state_type & previous_state(void) const;
  time_type previous_time(void) const;
  time_type current_time_step(void) const;

  // private member functions
  state_type & get_current_state(void);
  const state_type & get_current_state(void) const;
  state_type & get_old_state(void);
  const state_type & get_old_state(void) const;
  void toggle_current_state(void);
  template<typename StateIn> bool resize_impl(const StateIn &);
};
```

Description

`rosenbrock4_dense_output` public construct/copy/destruct

1.
```
rosenbrock4_dense_output(const controlled_stepper_type & stepper = controlled_stepper_type());
```

`rosenbrock4_dense_output` public member functions

1.
```
template<typename StateType> 
  void initialize(const StateType & x0, time_type t0, time_type dt0);
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

2.
```
template<typename System> 
  std::pair< time_type, time_type > do_step(System system);
```

3.
```
template<typename StateOut> void calc_state(time_type t, StateOut & x);
```

4.
```
template<typename StateOut> void calc_state(time_type t, const StateOut & x);
```

5.
```
template<typename StateType> void adjust_size(const StateType & x);
```

6.
```
const state_type & current_state(void) const;
```

7.
```
time_type current_time(void) const;
```

8.
```
const state_type & previous_state(void) const;
```

9.
```
time_type previous_time(void) const;
```

10.
```
time_type current_time_step(void) const;
```

`rosenbrock4_dense_output` private member functions

1.
```
state_type & get_current_state(void);
```

2.
```
const state_type & get_current_state(void) const;
```

3.
```
state_type & get_old_state(void);
```

4.
```
const state_type & get_old_state(void) const;
```

5.
```
void toggle_current_state(void);
```

6.
```
template<typename StateIn> bool resize_impl(const StateIn & x);
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Header <boost/numeric/odeint/stepper/runge_kutta4.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename State, typename Value = double,
               typename Deriv = State, typename Time = Value,
               typename Algebra = typename algebra_dispatcher< State >::algebra_type,
               typename Operations = typename operations_dispatcher< State >::operations_type,
               typename Resizer = initially_resizer>
        class runge_kutta4;
    }
  }
}
```

Class template runge_kutta4

boost::numeric::odeint::runge_kutta4 — The classical Runge-Kutta stepper of fourth order.

Synopsis

```
// In header: <boost/numeric/odeint/stepper/runge_kutta4.hpp>

template<typename State, typename Value = double, typename Deriv = State,
         typename Time = Value,
         typename Algebra = typename algebra_dispatcher< State >::algebra_type,
         typename Operations = typename operations_dispatcher< State >::operations_type,
         typename Resizer = initially_resizer>
class runge_kutta4 : public boost::numeric::odeint::explicit_generic_rk< StageCount, Order, ↲
State, Value, Deriv, Time, Algebra, Operations, Resizer >
{
public:
  // types
  typedef stepper_base_type::state_type      state_type;
  typedef stepper_base_type::value_type      value_type;
  typedef stepper_base_type::deriv_type      deriv_type;
  typedef stepper_base_type::time_type       time_type;
  typedef stepper_base_type::algebra_type    algebra_type;
  typedef stepper_base_type::operations_type operations_type;
  typedef stepper_base_type::resizer_type    resizer_type;

  // construct/copy/destruct
  runge_kutta4(const algebra_type & = algebra_type());

  // public member functions
  template<typename System, typename StateIn, typename DerivIn,
           typename StateOut>
    void do_step_impl(System, const StateIn &, const DerivIn &, time_type,
                      StateOut &, time_type);
  template<typename StateIn> void adjust_size(const StateIn &);
};
```

Description

The Runge-Kutta method of fourth order is one standard method for solving ordinary differential equations and is widely used, see also en.wikipedia.org/wiki/Runge-Kutta_methods The method is explicit and fulfills the Stepper concept. Step size control or continuous output are not provided.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

This class derives from explicit_stepper_base and inherits its interface via CRTP (current recurring template pattern). Furthermore, it derivs from explicit_generic_rk which is a generic Runge-Kutta algorithm. For more details see explicit_stepper_base and explicit_generic_rk.

Template Parameters

1.
   ```
   typename State
   ```

   The state type.

2.
   ```
   typename Value = double
   ```

   The value type.

3.
   ```
   typename Deriv = State
   ```

   The type representing the time derivative of the state.

4.
   ```
   typename Time = Value
   ```

   The time representing the independent variable - the time.

5.
   ```
   typename Algebra = typename algebra_dispatcher< State >::algebra_type
   ```

   The algebra type.

6.
   ```
   typename Operations = typename operations_dispatcher< State >::operations_type
   ```

   The operations type.

7.
   ```
   typename Resizer = initially_resizer
   ```

   The resizer policy type.

`runge_kutta4` public construct/copy/destruct

1.
   ```
   runge_kutta4(const algebra_type & algebra = algebra_type());
   ```

   Constructs the `runge_kutta4` class. This constructor can be used as a default constructor if the algebra has a default constructor.

Parameters:	`algebra`	A copy of algebra is made and stored inside explicit_stepper_base.

`runge_kutta4` public member functions

1.
   ```
   template<typename System, typename StateIn, typename DerivIn,
            typename StateOut>
     void do_step_impl(System system, const StateIn & in, const DerivIn & dxdt,
                       time_type t, StateOut & out, time_type dt);
   ```

   This method performs one step. The derivative `dxdt` of `in` at the time `t` is passed to the method. The result is updated out of place, hence the input is in `in` and the output in `out`. Access to this step functionality is provided by explicit_stepper_base and `do_step_impl` should not be called directly.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters:		
	`dt`	The step size.
	`dxdt`	The derivative of x at t.
	`in`	The state of the ODE which should be solved. in is not modified in this method
	`out`	The result of the step is written in out.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time, at which the step should be performed.

2.
```
template<typename StateIn> void adjust_size(const StateIn & x);
```

Adjust the size of all temporaries in the stepper manually.

Parameters:		
	`x`	A state from which the size of the temporaries to be resized is deduced.

Header <boost/numeric/odeint/stepper/runge_kutta4_classic.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename State, typename Value = double, 
               typename Deriv = State, typename Time = Value, 
               typename Algebra = typename algebra_dispatcher< State >::algebra_type, 
              typename Operations = typename operations_dispatcher< State >::operations_type, 
               typename Resizer = initially_resizer> 
        class runge_kutta4_classic;
    }
  }
}
```

Class template runge_kutta4_classic

boost::numeric::odeint::runge_kutta4_classic — The classical Runge-Kutta stepper of fourth order.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/runge_kutta4_classic.hpp>

template<typename State, typename Value = double, typename Deriv = State,
         typename Time = Value,
         typename Algebra = typename algebra_dispatcher< State >::algebra_type,
         typename Operations = typename operations_dispatcher< State >::operations_type,
         typename Resizer = initially_resizer>
class runge_kutta4_classic : public explicit_stepper_base {
public:
  // types
  typedef explicit_stepper_base< runge_kutta4_classic< ... >,... > stepper_base_type;
  typedef stepper_base_type::state_type                            state_type;
  typedef stepper_base_type::value_type                            value_type;
  typedef stepper_base_type::deriv_type                            deriv_type;
  typedef stepper_base_type::time_type                             time_type;
  typedef stepper_base_type::algebra_type                          algebra_type;
  typedef stepper_base_type::operations_type                       operations_type;
  typedef stepper_base_type::resizer_type                          resizer_type;

  // construct/copy/destruct
  runge_kutta4_classic(const algebra_type & = algebra_type());

  // public member functions
  template<typename System, typename StateIn, typename DerivIn,
           typename StateOut>
    void do_step_impl(System, const StateIn &, const DerivIn &, time_type,
                      StateOut &, time_type);
  template<typename StateType> void adjust_size(const StateType &);

  // private member functions
  template<typename StateIn> bool resize_impl(const StateIn &);
};
```

Description

The Runge-Kutta method of fourth order is one standard method for solving ordinary differential equations and is widely used, see also en.wikipedia.org/wiki/Runge-Kutta_methods The method is explicit and fulfills the Stepper concept. Step size control or continuous output are not provided. This class implements the method directly, hence the generic Runge-Kutta algorithm is not used.

This class derives from explicit_stepper_base and inherits its interface via CRTP (current recurring template pattern). For more details see explicit_stepper_base.

Template Parameters

1.
   ```
   typename State
   ```

   The state type.

2.
   ```
   typename Value = double
   ```

   The value type.

3.
   ```
   typename Deriv = State
   ```

   The type representing the time derivative of the state.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

4.
```
typename Time = Value
```

The time representing the independent variable - the time.

5.
```
typename Algebra = typename algebra_dispatcher< State >::algebra_type
```

The algebra type.

6.
```
typename Operations = typename operations_dispatcher< State >::operations_type
```

The operations type.

7.
```
typename Resizer = initially_resizer
```

The resizer policy type.

`runge_kutta4_classic` public construct/copy/destruct

1.
```
runge_kutta4_classic(const algebra_type & algebra = algebra_type());
```

Constructs the `runge_kutta4_classic` class. This constructor can be used as a default constructor if the algebra has a default constructor.

Parameters:	`algebra`	A copy of algebra is made and stored inside explicit_stepper_base.

`runge_kutta4_classic` public member functions

1.
```
template<typename System, typename StateIn, typename DerivIn, 
         typename StateOut> 
  void do_step_impl(System system, const StateIn & in, const DerivIn & dxdt, 
                    time_type t, StateOut & out, time_type dt);
```

This method performs one step. The derivative `dxdt` of `in` at the time `t` is passed to the method. The result is updated out of place, hence the input is in `in` and the output in `out`. Access to this step functionality is provided by explicit_stepper_base and `do_step_impl` should not be called directly.

Parameters:		
	`dt`	The step size.
	`dxdt`	The derivative of x at t.
	`in`	The state of the ODE which should be solved. in is not modified in this method
	`out`	The result of the step is written in out.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time, at which the step should be performed.

2.
```
template<typename StateType> void adjust_size(const StateType & x);
```

Adjust the size of all temporaries in the stepper manually.

Parameters:	`x`	A state from which the size of the temporaries to be resized is deduced.

`runge_kutta4_classic` private member functions

1.
```
template<typename StateIn> bool resize_impl(const StateIn & x);
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Header <boost/numeric/odeint/stepper/runge_kutta_cash_karp54.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename State, typename Value = double,
               typename Deriv = State, typename Time = Value,
               typename Algebra = typename algebra_dispatcher< State >::algebra_type,
               typename Operations = typename operations_dispatcher< State >::operations_type,
               typename Resizer = initially_resizer>
        class runge_kutta_cash_karp54;
    }
  }
}
```

Class template runge_kutta_cash_karp54

boost::numeric::odeint::runge_kutta_cash_karp54 — The Runge-Kutta Cash-Karp method.

Synopsis

```
// In header: <boost/numeric/odeint/stepper/runge_kutta_cash_karp54.hpp>

template<typename State, typename Value = double, typename Deriv = State,
         typename Time = Value,
         typename Algebra = typename algebra_dispatcher< State >::algebra_type,
         typename Operations = typename operations_dispatcher< State >::operations_type,
         typename Resizer = initially_resizer>
class runge_kutta_cash_karp54 : public boost::numeric::odeint::explicit_error_generic_rk< Stage↵
Count, Order, StepperOrder, ErrorOrder, State, Value, Deriv, Time, Algebra, Operations, Resizer >
{
public:
  // types
  typedef stepper_base_type::state_type      state_type;
  typedef stepper_base_type::value_type      value_type;
  typedef stepper_base_type::deriv_type      deriv_type;
  typedef stepper_base_type::time_type       time_type;
  typedef stepper_base_type::algebra_type    algebra_type;
  typedef stepper_base_type::operations_type operations_type;
  typedef stepper_base_type::resizer_type    resizer_typ;

  // construct/copy/destruct
  runge_kutta_cash_karp54(const algebra_type & = algebra_type());

  // public member functions
  template<typename System, typename StateIn, typename DerivIn,
           typename StateOut, typename Err>
    void do_step_impl(System, const StateIn &, const DerivIn &, time_type,
                      StateOut &, time_type, Err &);
  template<typename System, typename StateIn, typename DerivIn,
           typename StateOut>
    void do_step_impl(System, const StateIn &, const DerivIn &, time_type,
                      StateOut &, time_type);
  template<typename StateIn> void adjust_size(const StateIn &);
};
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Description

The Runge-Kutta Cash-Karp method is one of the standard methods for solving ordinary differential equations, see en.wikipedia.org/wiki/Cash-Karp_methods. The method is explicit and fulfills the Error Stepper concept. Step size control is provided but continuous output is not available for this method.

This class derives from explicit_error_stepper_base and inherits its interface via CRTP (current recurring template pattern). Furthermore, it derivs from explicit_error_generic_rk which is a generic Runge-Kutta algorithm with error estimation. For more details see explicit_error_stepper_base and explicit_error_generic_rk.

Template Parameters

1.
   ```
   typename State
   ```

   The state type.

2.
   ```
   typename Value = double
   ```

   The value type.

3.
   ```
   typename Deriv = State
   ```

   The type representing the time derivative of the state.

4.
   ```
   typename Time = Value
   ```

   The time representing the independent variable - the time.

5.
   ```
   typename Algebra = typename algebra_dispatcher< State >::algebra_type
   ```

   The algebra type.

6.
   ```
   typename Operations = typename operations_dispatcher< State >::operations_type
   ```

   The operations type.

7.
   ```
   typename Resizer = initially_resizer
   ```

   The resizer policy type.

`runge_kutta_cash_karp54` public construct/copy/destruct

1.
   ```
   runge_kutta_cash_karp54(const algebra_type & algebra = algebra_type());
   ```

   Constructs the `runge_kutta_cash_karp54` class. This constructor can be used as a default constructor if the algebra has a default constructor.

Parameters:		
	`algebra`	A copy of algebra is made and stored inside explicit_stepper_base.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

`runge_kutta_cash_karp54` public member functions

1.
```
template<typename System, typename StateIn, typename DerivIn, 
         typename StateOut, typename Err> 
  void do_step_impl(System system, const StateIn & in, const DerivIn & dxdt, 
                    time_type t, StateOut & out, time_type dt, Err & xerr);
```

This method performs one step. The derivative `dxdt` of `in` at the time `t` is passed to the method. The result is updated out-of-place, hence the input is in `in` and the output in `out`. Futhermore, an estimation of the error is stored in `xerr`. `do_step_impl` is used by explicit_error_stepper_base.

Parameters:	`dt`	The step size.
	`dxdt`	The derivative of x at t.
	`in`	The state of the ODE which should be solved. in is not modified in this method
	`out`	The result of the step is written in out.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time, at which the step should be performed.
	`xerr`	The result of the error estimation is written in xerr.

2.
```
template<typename System, typename StateIn, typename DerivIn, 
         typename StateOut> 
  void do_step_impl(System system, const StateIn & in, const DerivIn & dxdt, 
                    time_type t, StateOut & out, time_type dt);
```

This method performs one step. The derivative `dxdt` of `in` at the time `t` is passed to the method. The result is updated out-of-place, hence the input is in `in` and the output in `out`. Access to this step functionality is provided by explicit_stepper_base and `do_step_impl` should not be called directly.

Parameters:	`dt`	The step size.
	`dxdt`	The derivative of x at t.
	`in`	The state of the ODE which should be solved. in is not modified in this method
	`out`	The result of the step is written in out.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time, at which the step should be performed.

3.
```
template<typename StateIn> void adjust_size(const StateIn & x);
```

Adjust the size of all temporaries in the stepper manually.

Parameters:	`x`	A state from which the size of the temporaries to be resized is deduced.

Header <boost/numeric/odeint/stepper/runge_kutta_cash_karp54_classic.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename State, typename Value = double, 
               typename Deriv = State, typename Time = Value, 
               typename Algebra = typename algebra_dispatcher< State >::algebra_type, 
              typename Operations = typename operations_dispatcher< State >::operations_type, 
               typename Resizer = initially_resizer> 
        class runge_kutta_cash_karp54_classic;
    }
  }
}
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Class template runge_kutta_cash_karp54_classic

boost::numeric::odeint::runge_kutta_cash_karp54_classic — The Runge-Kutta Cash-Karp method implemented without the generic Runge-Kutta algorithm.

Synopsis

```
// In header: <boost/numeric/odeint/stepper/runge_kutta_cash_karp54_classic.hpp>

template<typename State, typename Value = double, typename Deriv = State, 
         typename Time = Value, 
         typename Algebra = typename algebra_dispatcher< State >::algebra_type, 
         typename Operations = typename operations_dispatcher< State >::operations_type, 
         typename Resizer = initially_resizer> 
class runge_kutta_cash_karp54_classic : public explicit_error_stepper_base {
public:
  // types
  typedef explicit_error_stepper_base< runge_kutta_cash_karp54_classic< ... >,... > step↵
per_base_type;
  typedef stepper_base_type::state_type                                           state_type; ↵

  typedef stepper_base_type::value_type                                           value_type; ↵

  typedef stepper_base_type::deriv_type                                           deriv_type; ↵

  typedef stepper_base_type::time_type                                             time_type; ↵

  typedef stepper_base_type::algebra_type                                       algebra_type; ↵

  typedef stepper_base_type::operations_type                                        opera↵
tions_type;
  typedef stepper_base_type::resizer_type                                       resizer_type; ↵


  // construct/copy/destruct
  runge_kutta_cash_karp54_classic(const algebra_type & = algebra_type());

  // public member functions
  template<typename System, typename StateIn, typename DerivIn, 
           typename StateOut, typename Err> 
    void do_step_impl(System, const StateIn &, const DerivIn &, time_type, 
                      StateOut &, time_type, Err &);
  template<typename System, typename StateIn, typename DerivIn, 
           typename StateOut> 
    void do_step_impl(System, const StateIn &, const DerivIn &, time_type, 
                      StateOut &, time_type);
  template<typename StateIn> void adjust_size(const StateIn &);

  // private member functions
  template<typename StateIn> bool resize_impl(const StateIn &);
};
```

Description

The Runge-Kutta Cash-Karp method is one of the standard methods for solving ordinary differential equations, see en.wikipedia.org/wiki/Cash-Karp_method. The method is explicit and fulfills the Error Stepper concept. Step size control is provided but continuous output is not available for this method.

This class derives from explicit_error_stepper_base and inherits its interface via CRTP (current recurring template pattern). This class implements the method directly, hence the generic Runge-Kutta algorithm is not used.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Template Parameters

1.
   ```
   typename State
   ```

   The state type.

2.
   ```
   typename Value = double
   ```

   The value type.

3.
   ```
   typename Deriv = State
   ```

   The type representing the time derivative of the state.

4.
   ```
   typename Time = Value
   ```

   The time representing the independent variable - the time.

5.
   ```
   typename Algebra = typename algebra_dispatcher< State >::algebra_type
   ```

   The algebra type.

6.
   ```
   typename Operations = typename operations_dispatcher< State >::operations_type
   ```

   The operations type.

7.
   ```
   typename Resizer = initially_resizer
   ```

   The resizer policy type.

`runge_kutta_cash_karp54_classic` public construct/copy/destruct

1.
   ```
   runge_kutta_cash_karp54_classic(const algebra_type & algebra = algebra_type());
   ```

   Constructs the `runge_kutta_cash_karp54_classic` class. This constructor can be used as a default constructor if the algebra has a default constructor.

Parameters:	`algebra`	A copy of algebra is made and stored inside explicit_stepper_base.

`runge_kutta_cash_karp54_classic` public member functions

1.
   ```
   template<typename System, typename StateIn, typename DerivIn,
            typename StateOut, typename Err>
     void do_step_impl(System system, const StateIn & in, const DerivIn & dxdt,
                       time_type t, StateOut & out, time_type dt, Err & xerr);
   ```

   This method performs one step. The derivative `dxdt` of `in` at the time `t` is passed to the method.

   The result is updated out-of-place, hence the input is in `in` and the output in `out`. Futhermore, an estimation of the error is stored in `xerr`. Access to this step functionality is provided by explicit_error_stepper_base and `do_step_impl` should not be called directly.

Parameters:	`dt`	The step size.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

	`dxdt`	The derivative of x at t.
	`in`	The state of the ODE which should be solved. in is not modified in this method
	`out`	The result of the step is written in out.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time, at which the step should be performed.
	`xerr`	The result of the error estimation is written in xerr.

2.
```
template<typename System, typename StateIn, typename DerivIn, 
         typename StateOut> 
  void do_step_impl(System system, const StateIn & in, const DerivIn & dxdt, 
                    time_type t, StateOut & out, time_type dt);
```

This method performs one step. The derivative `dxdt` of `in` at the time `t` is passed to the method. The result is updated out-of-place, hence the input is in `in` and the output in `out`. Access to this step functionality is provided by explicit_error_stepper_base and `do_step_impl` should not be called directly.

Parameters:	`dt`	The step size.
	`dxdt`	The derivative of x at t.
	`in`	The state of the ODE which should be solved. in is not modified in this method
	`out`	The result of the step is written in out.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time, at which the step should be performed.

3.
```
template<typename StateIn> void adjust_size(const StateIn & x);
```

Adjust the size of all temporaries in the stepper manually.

Parameters:	`x`	A state from which the size of the temporaries to be resized is deduced.

`runge_kutta_cash_karp54_classic` private member functions

1.
```
template<typename StateIn> bool resize_impl(const StateIn & x);
```

Header <boost/numeric/odeint/stepper/runge_kutta_dopri5.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename State, typename Value = double, 
               typename Deriv = State, typename Time = Value, 
               typename Algebra = typename algebra_dispatcher< State >::algebra_type, 
               typename Operations = typename operations_dispatcher< State >::operations_type, 
               typename Resizer = initially_resizer> 
        class runge_kutta_dopri5;
    }
  }
}
```

Class template runge_kutta_dopri5

boost::numeric::odeint::runge_kutta_dopri5 — The Runge-Kutta Dormand-Prince 5 method.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/runge_kutta_dopri5.hpp>

template<typename State, typename Value = double, typename Deriv = State,
         typename Time = Value,
         typename Algebra = typename algebra_dispatcher< State >::algebra_type,
         typename Operations = typename operations_dispatcher< State >::operations_type,
         typename Resizer = initially_resizer>
class runge_kutta_dopri5 : public explicit_error_stepper_fsal_base {
public:
  // types
  typedef explicit_error_stepper_fsal_base< runge_kutta_dopri5< ... >,... > stepper_base_type;
  typedef stepper_base_type::state_type                                      state_type;
  typedef stepper_base_type::value_type                                      value_type;
  typedef stepper_base_type::deriv_type                                      deriv_type;
  typedef stepper_base_type::time_type                                       time_type;
  typedef stepper_base_type::algebra_type                                    algebra_type;
  typedef stepper_base_type::operations_type                                 operations_type;
  typedef stepper_base_type::resizer_type                                    resizer_type;

  // construct/copy/destruct
  runge_kutta_dopri5(const algebra_type & = algebra_type());

  // public member functions
  template<typename System, typename StateIn, typename DerivIn,
           typename StateOut, typename DerivOut>
    void do_step_impl(System, const StateIn &, const DerivIn &, time_type,
                      StateOut &, DerivOut &, time_type);
  template<typename System, typename StateIn, typename DerivIn,
           typename StateOut, typename DerivOut, typename Err>
    void do_step_impl(System, const StateIn &, const DerivIn &, time_type,
                      StateOut &, DerivOut &, time_type, Err &);
  template<typename StateOut, typename StateIn1, typename DerivIn1,
           typename StateIn2, typename DerivIn2>
    void calc_state(time_type, StateOut &, const StateIn1 &, const DerivIn1 &,
                    time_type, const StateIn2 &, const DerivIn2 &, time_type) const;
  template<typename StateIn> void adjust_size(const StateIn &);

  // private member functions
  template<typename StateIn> bool resize_k_x_tmp_impl(const StateIn &);
  template<typename StateIn> bool resize_dxdt_tmp_impl(const StateIn &);
};
```

Description

The Runge-Kutta Dormand-Prince 5 method is a very popular method for solving ODEs, see . The method is explicit and fulfills the Error Stepper concept. Step size control is provided but continuous output is available which make this method favourable for many applications.

This class derives from explicit_error_stepper_fsal_base and inherits its interface via CRTP (current recurring template pattern). The method possesses the FSAL (first-same-as-last) property. See explicit_error_stepper_fsal_base for more details.

Template Parameters

1.
```
typename State
```

   The state type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

2.
```
typename Value = double
```

The value type.

3.
```
typename Deriv = State
```

The type representing the time derivative of the state.

4.
```
typename Time = Value
```

The time representing the independent variable - the time.

5.
```
typename Algebra = typename algebra_dispatcher< State >::algebra_type
```

The algebra type.

6.
```
typename Operations = typename operations_dispatcher< State >::operations_type
```

The operations type.

7.
```
typename Resizer = initially_resizer
```

The resizer policy type.

`runge_kutta_dopri5` public construct/copy/destruct

1.
```
runge_kutta_dopri5(const algebra_type & algebra = algebra_type());
```

Constructs the `runge_kutta_dopri5` class. This constructor can be used as a default constructor if the algebra has a default constructor.

Parameters:	`algebra`	A copy of algebra is made and stored inside explicit_stepper_base.

`runge_kutta_dopri5` public member functions

1.
```
template<typename System, typename StateIn, typename DerivIn, 
         typename StateOut, typename DerivOut> 
  void do_step_impl(System system, const StateIn & in, 
                    const DerivIn & dxdt_in, time_type t, StateOut & out, 
                    DerivOut & dxdt_out, time_type dt);
```

This method performs one step. The derivative `dxdt_in` of `in` at the time `t` is passed to the method. The result is updated out-of-place, hence the input is in `in` and the output in `out`. Furthermore, the derivative is update out-of-place, hence the input is assumed to be in `dxdt_in` and the output in `dxdt_out`. Access to this step functionality is provided by explicit_error_stepper_fsal_base and `do_step_impl` should not be called directly.

Parameters:	`dt`	The step size.
	`dxdt_in`	The derivative of x at t. dxdt_in is not modified by this method
	`dxdt_out`	The result of the new derivative at time t+dt.
	`in`	The state of the ODE which should be solved. in is not modified in this method
	`out`	The result of the step is written in out.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time, at which the step should be performed.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

2.

```
template<typename System, typename StateIn, typename DerivIn, 
         typename StateOut, typename DerivOut, typename Err> 
  void do_step_impl(System system, const StateIn & in, 
                    const DerivIn & dxdt_in, time_type t, StateOut & out, 
                    DerivOut & dxdt_out, time_type dt, Err & xerr);
```

This method performs one step. The derivative `dxdt_in` of `in` at the time `t` is passed to the method. The result is updated out-of-place, hence the input is in `in` and the output in `out`. Furthermore, the derivative is update out-of-place, hence the input is assumed to be in `dxdt_in` and the output in `dxdt_out`. Access to this step functionality is provided by explicit_error_stepper_fsal_base and `do_step_impl` should not be called directly. An estimation of the error is calculated.

Parameters:		
	`dt`	The step size.
	`dxdt_in`	The derivative of x at t. dxdt_in is not modified by this method
	`dxdt_out`	The result of the new derivative at time t+dt.
	`in`	The state of the ODE which should be solved. in is not modified in this method
	`out`	The result of the step is written in out.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time, at which the step should be performed.
	`xerr`	An estimation of the error.

3.

```
template<typename StateOut, typename StateIn1, typename DerivIn1, 
         typename StateIn2, typename DerivIn2> 
  void calc_state(time_type t, StateOut & x, const StateIn1 & x_old, 
                  const DerivIn1 & deriv_old, time_type t_old, 
                  const StateIn2 &, const DerivIn2 & deriv_new, 
                  time_type t_new) const;
```

This method is used for continuous output and it calculates the state `x` at a time `t` from the knowledge of two states `old_state` and `current_state` at time points `t_old` and `t_new`. It also uses internal variables to calculate the result. Hence this method must be called after two successful `do_step` calls.

4.

```
template<typename StateIn> void adjust_size(const StateIn & x);
```

Adjust the size of all temporaries in the stepper manually.

Parameters:		
	`x`	A state from which the size of the temporaries to be resized is deduced.

`runge_kutta_dopri5` private member functions

1.

```
template<typename StateIn> bool resize_k_x_tmp_impl(const StateIn & x);
```

2.

```
template<typename StateIn> bool resize_dxdt_tmp_impl(const StateIn & x);
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Header <boost/numeric/odeint/stepper/runge_kutta_fehlberg78.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename State, typename Value = double,
               typename Deriv = State, typename Time = Value,
               typename Algebra = typename algebra_dispatcher< State >::algebra_type,
               typename Operations = typename operations_dispatcher< State >::operations_type,
               typename Resizer = initially_resizer>
        class runge_kutta_fehlberg78;
    }
  }
}
```

Class template runge_kutta_fehlberg78

boost::numeric::odeint::runge_kutta_fehlberg78 — The Runge-Kutta Fehlberg 78 method.

Synopsis

```
// In header: <boost/numeric/odeint/stepper/runge_kutta_fehlberg78.hpp>

template<typename State, typename Value = double, typename Deriv = State,
         typename Time = Value,
         typename Algebra = typename algebra_dispatcher< State >::algebra_type,
         typename Operations = typename operations_dispatcher< State >::operations_type,
         typename Resizer = initially_resizer>
class runge_kutta_fehlberg78 : public boost::numeric::odeint::explicit_error_generic_rk< Stage↵
Count, Order, StepperOrder, ErrorOrder, State, Value, Deriv, Time, Algebra, Operations, Resizer >
{
public:
  // types
  typedef stepper_base_type::state_type      state_type;
  typedef stepper_base_type::value_type      value_type;
  typedef stepper_base_type::deriv_type      deriv_type;
  typedef stepper_base_type::time_type       time_type;
  typedef stepper_base_type::algebra_type    algebra_type;
  typedef stepper_base_type::operations_type operations_type;
  typedef stepper_base_type::resizer_type    resizer_type;

  // construct/copy/destruct
  runge_kutta_fehlberg78(const algebra_type & = algebra_type());

  // public member functions
  template<typename System, typename StateIn, typename DerivIn,
           typename StateOut, typename Err>
    void do_step_impl(System, const StateIn &, const DerivIn &, time_type,
                      StateOut &, time_type, Err &);
  template<typename System, typename StateIn, typename DerivIn,
           typename StateOut>
    void do_step_impl(System, const StateIn &, const DerivIn &, time_type,
                      StateOut &, time_type);
  template<typename StateIn> void adjust_size(const StateIn &);
};
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Description

The Runge-Kutta Fehlberg 78 method is a standard method for high-precision applications. The method is explicit and fulfills the Error Stepper concept. Step size control is provided but continuous output is not available for this method.

This class derives from explicit_error_stepper_base and inherits its interface via CRTP (current recurring template pattern). Furthermore, it derivs from explicit_error_generic_rk which is a generic Runge-Kutta algorithm with error estimation. For more details see explicit_error_stepper_base and explicit_error_generic_rk.

Template Parameters

1.
   ```
   typename State
   ```

   The state type.

2.
   ```
   typename Value = double
   ```

   The value type.

3.
   ```
   typename Deriv = State
   ```

   The type representing the time derivative of the state.

4.
   ```
   typename Time = Value
   ```

   The time representing the independent variable - the time.

5.
   ```
   typename Algebra = typename algebra_dispatcher< State >::algebra_type
   ```

   The algebra type.

6.
   ```
   typename Operations = typename operations_dispatcher< State >::operations_type
   ```

   The operations type.

7.
   ```
   typename Resizer = initially_resizer
   ```

   The resizer policy type.

`runge_kutta_fehlberg78` public construct/copy/destruct

1.
   ```
   runge_kutta_fehlberg78(const algebra_type & algebra = algebra_type());
   ```

   Constructs the runge_kutta_cash_fehlberg78 class. This constructor can be used as a default constructor if the algebra has a default constructor.

Parameters:	`algebra`	A copy of algebra is made and stored inside explicit_stepper_base.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

`runge_kutta_fehlberg78` public member functions

1.

```
template<typename System, typename StateIn, typename DerivIn, 
         typename StateOut, typename Err> 
  void do_step_impl(System system, const StateIn & in, const DerivIn & dxdt, 
                    time_type t, StateOut & out, time_type dt, Err & xerr);
```

This method performs one step. The derivative `dxdt` of `in` at the time `t` is passed to the method. The result is updated out-of-place, hence the input is in `in` and the output in `out`. Futhermore, an estimation of the error is stored in `xerr`. `do_step_impl` is used by explicit_error_stepper_base.

Parameters:		
	`dt`	The step size.
	`dxdt`	The derivative of x at t.
	`in`	The state of the ODE which should be solved. in is not modified in this method
	`out`	The result of the step is written in out.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time, at which the step should be performed.
	`xerr`	The result of the error estimation is written in xerr.

2.

```
template<typename System, typename StateIn, typename DerivIn, 
         typename StateOut> 
  void do_step_impl(System system, const StateIn & in, const DerivIn & dxdt, 
                    time_type t, StateOut & out, time_type dt);
```

This method performs one step. The derivative `dxdt` of `in` at the time `t` is passed to the method. The result is updated out-of-place, hence the input is in `in` and the output in `out`. Access to this step functionality is provided by explicit_stepper_base and `do_step_impl` should not be called directly.

Parameters:		
	`dt`	The step size.
	`dxdt`	The derivative of x at t.
	`in`	The state of the ODE which should be solved. in is not modified in this method
	`out`	The result of the step is written in out.
	`system`	The system function to solve, hence the r.h.s. of the ODE. It must fulfill the Simple System concept.
	`t`	The value of the time, at which the step should be performed.

3.

```
template<typename StateIn> void adjust_size(const StateIn & x);
```

Adjust the size of all temporaries in the stepper manually.

Parameters:		
	`x`	A state from which the size of the temporaries to be resized is deduced.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Header <boost/numeric/odeint/stepper/stepper_categories.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename tag> struct base_tag;

      template<> struct base_tag<controlled_stepper_tag>;
      template<> struct base_tag<dense_output_stepper_tag>;
      template<> struct base_tag<error_stepper_tag>;
      template<> struct base_tag<explicit_controlled_stepper_fsal_tag>;
      template<> struct base_tag<explicit_controlled_stepper_tag>;
      template<> struct base_tag<explicit_error_stepper_fsal_tag>;
      template<> struct base_tag<explicit_error_stepper_tag>;
      template<> struct base_tag<stepper_tag>;

      struct controlled_stepper_tag;
      struct dense_output_stepper_tag;
      struct error_stepper_tag;
      struct explicit_controlled_stepper_fsal_tag;
      struct explicit_controlled_stepper_tag;
      struct explicit_error_stepper_fsal_tag;
      struct explicit_error_stepper_tag;
      struct stepper_tag;
    }
  }
}
```

Struct template base_tag

boost::numeric::odeint::base_tag

Synopsis

```
// In header: <boost/numeric/odeint/stepper/stepper_categories.hpp>

template<typename tag>
struct base_tag {
};
```

Struct base_tag<controlled_stepper_tag>

boost::numeric::odeint::base_tag<controlled_stepper_tag>

Synopsis

```
// In header: <boost/numeric/odeint/stepper/stepper_categories.hpp>


struct base_tag<controlled_stepper_tag> {
  // types
  typedef controlled_stepper_tag type;
};
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Struct base_tag<dense_output_stepper_tag>

boost::numeric::odeint::base_tag<dense_output_stepper_tag>

Synopsis

```
// In header: <boost/numeric/odeint/stepper/stepper_categories.hpp>


struct base_tag<dense_output_stepper_tag> {
  // types
  typedef dense_output_stepper_tag type;
};
```

Struct base_tag<error_stepper_tag>

boost::numeric::odeint::base_tag<error_stepper_tag>

Synopsis

```
// In header: <boost/numeric/odeint/stepper/stepper_categories.hpp>


struct base_tag<error_stepper_tag> {
  // types
  typedef stepper_tag type;
};
```

Struct base_tag<explicit_controlled_stepper_fsal_tag>

boost::numeric::odeint::base_tag<explicit_controlled_stepper_fsal_tag>

Synopsis

```
// In header: <boost/numeric/odeint/stepper/stepper_categories.hpp>


struct base_tag<explicit_controlled_stepper_fsal_tag> {
  // types
  typedef controlled_stepper_tag type;
};
```

Struct base_tag<explicit_controlled_stepper_tag>

boost::numeric::odeint::base_tag<explicit_controlled_stepper_tag>

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/stepper_categories.hpp>


struct base_tag<explicit_controlled_stepper_tag> {
  // types
  typedef controlled_stepper_tag type;
};
```

Struct base_tag<explicit_error_stepper_fsal_tag>

boost::numeric::odeint::base_tag<explicit_error_stepper_fsal_tag>

Synopsis

```
// In header: <boost/numeric/odeint/stepper/stepper_categories.hpp>


struct base_tag<explicit_error_stepper_fsal_tag> {
  // types
  typedef stepper_tag type;
};
```

Struct base_tag<explicit_error_stepper_tag>

boost::numeric::odeint::base_tag<explicit_error_stepper_tag>

Synopsis

```
// In header: <boost/numeric/odeint/stepper/stepper_categories.hpp>


struct base_tag<explicit_error_stepper_tag> {
  // types
  typedef stepper_tag type;
};
```

Struct base_tag<stepper_tag>

boost::numeric::odeint::base_tag<stepper_tag>

Synopsis

```
// In header: <boost/numeric/odeint/stepper/stepper_categories.hpp>


struct base_tag<stepper_tag> {
  // types
  typedef stepper_tag type;
};
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Struct controlled_stepper_tag

boost::numeric::odeint::controlled_stepper_tag

Synopsis

```
// In header: <boost/numeric/odeint/stepper/stepper_categories.hpp>


struct controlled_stepper_tag {
};
```

Struct dense_output_stepper_tag

boost::numeric::odeint::dense_output_stepper_tag

Synopsis

```
// In header: <boost/numeric/odeint/stepper/stepper_categories.hpp>


struct dense_output_stepper_tag {
};
```

Struct error_stepper_tag

boost::numeric::odeint::error_stepper_tag

Synopsis

```
// In header: <boost/numeric/odeint/stepper/stepper_categories.hpp>


struct error_stepper_tag : public boost::numeric::odeint::stepper_tag {
};
```

Struct explicit_controlled_stepper_fsal_tag

boost::numeric::odeint::explicit_controlled_stepper_fsal_tag

Synopsis

```
// In header: <boost/numeric/odeint/stepper/stepper_categories.hpp>


struct explicit_controlled_stepper_fsal_tag :
  public boost::numeric::odeint::controlled_stepper_tag
{
};
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Struct explicit_controlled_stepper_tag

boost::numeric::odeint::explicit_controlled_stepper_tag

Synopsis

```
// In header: <boost/numeric/odeint/stepper/stepper_categories.hpp>


struct explicit_controlled_stepper_tag :
  public boost::numeric::odeint::controlled_stepper_tag
{
};
```

Struct explicit_error_stepper_fsal_tag

boost::numeric::odeint::explicit_error_stepper_fsal_tag

Synopsis

```
// In header: <boost/numeric/odeint/stepper/stepper_categories.hpp>


struct explicit_error_stepper_fsal_tag :
  public boost::numeric::odeint::error_stepper_tag
{
};
```

Struct explicit_error_stepper_tag

boost::numeric::odeint::explicit_error_stepper_tag

Synopsis

```
// In header: <boost/numeric/odeint/stepper/stepper_categories.hpp>


struct explicit_error_stepper_tag :
  public boost::numeric::odeint::error_stepper_tag
{
};
```

Struct stepper_tag

boost::numeric::odeint::stepper_tag

Synopsis

```
// In header: <boost/numeric/odeint/stepper/stepper_categories.hpp>


struct stepper_tag {
};
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Header <boost/numeric/odeint/stepper/symplectic_euler.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename Coor, typename Momentum = Coor,
               typename Value = double, typename CoorDeriv = Coor,
               typename MomentumDeriv = Coor, typename Time = Value,
               typename Algebra = typename algebra_dispatcher< Coor >::algebra_type,
               typename Operations = typename operations_dispatcher< Coor >::operations_type,
               typename Resizer = initially_resizer>
        class symplectic_euler;
    }
  }
}
```

Class template symplectic_euler

boost::numeric::odeint::symplectic_euler — Implementation of the symplectic Euler method.

Synopsis

```
// In header: <boost/numeric/odeint/stepper/symplectic_euler.hpp>

template<typename Coor, typename Momentum = Coor, typename Value = double,
         typename CoorDeriv = Coor, typename MomentumDeriv = Coor,
         typename Time = Value,
         typename Algebra = typename algebra_dispatcher< Coor >::algebra_type,
         typename Operations = typename operations_dispatcher< Coor >::operations_type,
         typename Resizer = initially_resizer>
class symplectic_euler : public symplectic_nystroem_stepper_base {
public:
  // types
  typedef stepper_base_type::algebra_type algebra_type;
  typedef stepper_base_type::value_type   value_type;

  // construct/copy/destruct
  symplectic_euler(const algebra_type & = algebra_type());
};
```

Description

The method is of first order and has one stage. It is described HERE.

Template Parameters

1.
   ```
   typename Coor
   ```

   The type representing the coordinates q.

2.
   ```
   typename Momentum = Coor
   ```

   The type representing the coordinates p.

3.
   ```
   typename Value = double
   ```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

The basic value type. Should be something like float, double or a high-precision type.

4.
```
typename CoorDeriv = Coor
```

The type representing the time derivative of the coordinate dq/dt.

5.
```
typename MomentumDeriv = Coor
```

6.
```
typename Time = Value
```

The type representing the time t.

7.
```
typename Algebra = typename algebra_dispatcher< Coor >::algebra_type
```

The algebra.

8.
```
typename Operations = typename operations_dispatcher< Coor >::operations_type
```

The operations.

9.
```
typename Resizer = initially_resizer
```

The resizer policy.

`symplectic_euler` public construct/copy/destruct

1.
```
symplectic_euler(const algebra_type & algebra = algebra_type());
```

Constructs the `symplectic_euler`. This constructor can be used as a default constructor if the algebra has a default constructor.

Parameters:	`algebra`	A copy of algebra is made and stored inside explicit_stepper_base.

Header <boost/numeric/odeint/stepper/symplectic_rkn_sb3a_m4_mclachlan.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename Coor, typename Momentum = Coor,
               typename Value = double, typename CoorDeriv = Coor,
               typename MomentumDeriv = Coor, typename Time = Value,
               typename Algebra = typename algebra_dispatcher< Coor >::algebra_type,
               typename Operations = typename operations_dispatcher< Coor >::operations_type,
               typename Resizer = initially_resizer>
        class symplectic_rkn_sb3a_m4_mclachlan;
    }
  }
}
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Class template symplectic_rkn_sb3a_m4_mclachlan

boost::numeric::odeint::symplectic_rkn_sb3a_m4_mclachlan — Implementation of the symmetric B3A Runge-Kutta Nystroem method of fifth order.

Synopsis

```
// In header: <boost/numeric/odeint/stepper/symplectic_rkn_sb3a_m4_mclachlan.hpp>

template<typename Coor, typename Momentum = Coor, typename Value = double, 
         typename CoorDeriv = Coor, typename MomentumDeriv = Coor, 
         typename Time = Value, 
         typename Algebra = typename algebra_dispatcher< Coor >::algebra_type, 
         typename Operations = typename operations_dispatcher< Coor >::operations_type, 
         typename Resizer = initially_resizer> 
class symplectic_rkn_sb3a_m4_mclachlan : 
  public symplectic_nystroem_stepper_base
{
public:
  // types
  typedef stepper_base_type::algebra_type algebra_type;
  typedef stepper_base_type::value_type   value_type;  

  // construct/copy/destruct
  symplectic_rkn_sb3a_m4_mclachlan(const algebra_type & = algebra_type());
};
```

Description

The method is of fourth order and has five stages. It is described HERE. This method can be used with multiprecision types since the coefficients are defined analytically.

ToDo: add reference to paper.

Template Parameters

1.
   ```
   typename Coor
   ```

   The type representing the coordinates q.

2.
   ```
   typename Momentum = Coor
   ```

   The type representing the coordinates p.

3.
   ```
   typename Value = double
   ```

   The basic value type. Should be something like float, double or a high-precision type.

4.
   ```
   typename CoorDeriv = Coor
   ```

   The type representing the time derivative of the coordinate dq/dt.

5.
   ```
   typename MomentumDeriv = Coor
   ```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

6.
```
typename Time = Value
```

The type representing the time t.

7.
```
typename Algebra = typename algebra_dispatcher< Coor >::algebra_type
```

The algebra.

8.
```
typename Operations = typename operations_dispatcher< Coor >::operations_type
```

The operations.

9.
```
typename Resizer = initially_resizer
```

The resizer policy.

`symplectic_rkn_sb3a_m4_mclachlan` public construct/copy/destruct

1.
```
symplectic_rkn_sb3a_m4_mclachlan(const algebra_type & algebra = algebra_type());
```

Constructs the `symplectic_rkn_sb3a_m4_mclachlan`. This constructor can be used as a default constructor if the algebra has a default constructor.

Parameters:	`algebra`	A copy of algebra is made and stored inside explicit_stepper_base.

Header <boost/numeric/odeint/stepper/symplectic_rkn_sb3a_mclachlan.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename Coor, typename Momentum = Coor,
               typename Value = double, typename CoorDeriv = Coor,
               typename MomentumDeriv = Coor, typename Time = Value,
               typename Algebra = typename algebra_dispatcher< Coor >::algebra_type,
               typename Operations = typename operations_dispatcher< Coor >::operations_type,
               typename Resizer = initially_resizer>
        class symplectic_rkn_sb3a_mclachlan;
    }
  }
}
```

Class template symplectic_rkn_sb3a_mclachlan

boost::numeric::odeint::symplectic_rkn_sb3a_mclachlan — Implement of the symmetric B3A method of Runge-Kutta-Nystroem method of sixth order.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/symplectic_rkn_sb3a_mclachlan.hpp>

template<typename Coor, typename Momentum = Coor, typename Value = double,
         typename CoorDeriv = Coor, typename MomentumDeriv = Coor,
         typename Time = Value,
         typename Algebra = typename algebra_dispatcher< Coor >::algebra_type,
         typename Operations = typename operations_dispatcher< Coor >::operations_type,
         typename Resizer = initially_resizer>
class symplectic_rkn_sb3a_mclachlan : public symplectic_nystroem_stepper_base {
public:
  // types
  typedef stepper_base_type::algebra_type algebra_type;
  typedef stepper_base_type::value_type   value_type;

  // construct/copy/destruct
  symplectic_rkn_sb3a_mclachlan(const algebra_type & = algebra_type());
};
```

Description

The method is of fourth order and has six stages. It is described HERE. This method cannot be used with multiprecision types since the coefficients are not defined analytically.

ToDo Add reference to the paper.

Template Parameters

1.
```
typename Coor
```

   The type representing the coordinates q.

2.
```
typename Momentum = Coor
```

   The type representing the coordinates p.

3.
```
typename Value = double
```

   The basic value type. Should be something like float, double or a high-precision type.

4.
```
typename CoorDeriv = Coor
```

   The type representing the time derivative of the coordinate dq/dt.

5.
```
typename MomentumDeriv = Coor
```

6.
```
typename Time = Value
```

   The type representing the time t.

7.
```
typename Algebra = typename algebra_dispatcher< Coor >::algebra_type
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

The algebra.

8.
```
typename Operations = typename operations_dispatcher< Coor >::operations_type
```

The operations.

9.
```
typename Resizer = initially_resizer
```

The resizer policy.

`symplectic_rkn_sb3a_mclachlan` public construct/copy/destruct

1.
```
symplectic_rkn_sb3a_mclachlan(const algebra_type & algebra = algebra_type());
```

Constructs the `symplectic_rkn_sb3a_mclachlan`. This constructor can be used as a default constructor if the algebra has a default constructor.

Parameters:	`algebra`	A copy of algebra is made and stored inside explicit_stepper_base.

Header <boost/numeric/odeint/stepper/velocity_verlet.hpp>

```
namespace boost {
  namespace numeric {
    namespace odeint {
      template<typename Coor, typename Velocity = Coor, 
               typename Value = double, typename Acceleration = Coor, 
               typename Time = Value, typename TimeSq = Time, 
               typename Algebra = typename algebra_dispatcher< Coor >::algebra_type, 
               typename Operations = typename operations_dispatcher< Coor >::operations_type, 
               typename Resizer = initially_resizer> 
        class velocity_verlet;
    }
  }
}
```

Class template velocity_verlet

boost::numeric::odeint::velocity_verlet — The Velocity-Verlet algorithm.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Synopsis

```
// In header: <boost/numeric/odeint/stepper/velocity_verlet.hpp>

template<typename Coor, typename Velocity = Coor, typename Value = double,
         typename Acceleration = Coor, typename Time = Value,
         typename TimeSq = Time,
         typename Algebra = typename algebra_dispatcher< Coor >::algebra_type,
         typename Operations = typename operations_dispatcher< Coor >::operations_type,
         typename Resizer = initially_resizer>
class velocity_verlet : public algebra_stepper_base< Algebra, Operations > {
public:
  // types
  typedef algebra_stepper_base< Algebra, Operations >   algebra_stepper_base_type;
  typedef algebra_stepper_base_type::algebra_type       algebra_type;
  typedef algebra_stepper_base_type::operations_type    operations_type;
  typedef Coor                                          coor_type;
  typedef Velocity                                      velocity_type;
  typedef Acceleration                                  acceleration_type;
  typedef std::pair< coor_type, velocity_type >         state_type;
  typedef std::pair< velocity_type, acceleration_type > deriv_type;
  typedef state_wrapper< acceleration_type >            wrapped_acceleration_type;
  typedef Value                                         value_type;
  typedef Time                                          time_type;
  typedef TimeSq                                        time_square_type;
  typedef Resizer                                       resizer_type;
  typedef stepper_tag                                   stepper_category;
  typedef unsigned short                                order_type;

  // construct/copy/destruct
  velocity_verlet(const algebra_type & = algebra_type());

  // public member functions
  order_type order(void) const;
  template<typename System, typename StateInOut>
    void do_step(System, StateInOut &, time_type, time_type);
  template<typename System, typename StateInOut>
    void do_step(System, const StateInOut &, time_type, time_type);
  template<typename System, typename CoorIn, typename VelocityIn,
           typename AccelerationIn, typename CoorOut, typename VelocityOut,
           typename AccelerationOut>
    void do_step(System, CoorIn const &, VelocityIn const &,
                 AccelerationIn const &, CoorOut &, VelocityOut &,
                 AccelerationOut &, time_type, time_type);
  template<typename StateIn> void adjust_size(const StateIn &);
  void reset(void);
  template<typename AccelerationIn> void initialize(const AccelerationIn &);
  template<typename System, typename CoorIn, typename VelocityIn>
    void initialize(System, const CoorIn &, const VelocityIn &, time_type);
  bool is_initialized(void) const;

  // private member functions
  template<typename System, typename CoorIn, typename VelocityIn>
    void initialize_acc(System, const CoorIn &, const VelocityIn &, time_type);
  template<typename System, typename StateInOut>
    void do_step_v1(System, StateInOut &, time_type, time_type);
  template<typename StateIn> bool resize_impl(const StateIn &);
  acceleration_type & get_current_acc(void);
  const acceleration_type & get_current_acc(void) const;
  acceleration_type & get_old_acc(void);
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  const acceleration_type & get_old_acc(void) const;
  void toggle_current_acc(void);

  // public data members
  static const order_type order_value;
};
```

Description

The Velocity-Verlet algorithm is a method for simulation of molecular dynamics systems. It solves the ODE a=f(r,v',t) where r are the coordinates, v are the velocities and a are the accelerations, hence v = dr/dt, a=dv/dt.

Template Parameters

1.
   ```
   typename Coor
   ```

   The type representing the coordinates.

2.
   ```
   typename Velocity = Coor
   ```

   The type representing the velocities.

3.
   ```
   typename Value = double
   ```

   The type value type.

4.
   ```
   typename Acceleration = Coor
   ```

   The type representing the acceleration.

5.
   ```
   typename Time = Value
   ```

   The time representing the independent variable - the time.

6.
   ```
   typename TimeSq = Time
   ```

   The time representing the square of the time.

7.
   ```
   typename Algebra = typename algebra_dispatcher< Coor >::algebra_type
   ```

   The algebra.

8.
   ```
   typename Operations = typename operations_dispatcher< Coor >::operations_type
   ```

   The operations type.

9.
   ```
   typename Resizer = initially_resizer
   ```

   The resizer policy type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

`velocity_verlet` public construct/copy/destruct

1.

```
velocity_verlet(const algebra_type & algebra = algebra_type());
```

Constructs the `velocity_verlet` class. This constructor can be used as a default constructor if the algebra has a default constructor.

Parameters:		
	`algebra`	A copy of algebra is made and stored.

`velocity_verlet` public member functions

1.

```
order_type order(void) const;
```

Returns: Returns the order of the stepper.

2.

```
template<typename System, typename StateInOut>
  void do_step(System system, StateInOut & x, time_type t, time_type dt);
```

This method performs one step. It transforms the result in-place.

It can be used like

```
pair< coordinates , velocities > state;
stepper.do_step( sys , x , t , dt );
```

Parameters:		
	`dt`	The step size.
	`system`	The system function to solve, hence the r.h.s. of the ordinary differential equation. It must fulfill the Second Order System concept.
	`t`	The value of the time, at which the step should be performed.
	`x`	The state of the ODE which should be solved. The state is pair of Coor and Velocity.

3.

```
template<typename System, typename StateInOut>
  void do_step(System system, const StateInOut & x, time_type t, time_type dt);
```

This method performs one step. It transforms the result in-place.

It can be used like

```
pair< coordinates , velocities > state;
stepper.do_step( sys , x , t , dt );
```

Parameters:		
	`dt`	The step size.
	`system`	The system function to solve, hence the r.h.s. of the ordinary differential equation. It must fulfill the Second Order System concept.
	`t`	The value of the time, at which the step should be performed.
	`x`	The state of the ODE which should be solved. The state is pair of Coor and Velocity.

4.

```
template<typename System, typename CoorIn, typename VelocityIn,
         typename AccelerationIn, typename CoorOut, typename VelocityOut,
         typename AccelerationOut>
  void do_step(System system, CoorIn const & qin, VelocityIn const & pin,
               AccelerationIn const & ain, CoorOut & qout,
               VelocityOut & pout, AccelerationOut & aout, time_type t,
               time_type dt);
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

This method performs one step. It transforms the result in-place. Additionally to the other methods the coordinates, velocities and accelerations are passed directly to do_step and they are transformed out-of-place.

It can be used like

```
coordinates qin , qout;
velocities pin , pout;
accelerations ain, aout;
stepper.do_step( sys , qin , pin , ain , qout , pout , aout , t , dt );
```

Parameters:		
	`dt`	The step size.
	`system`	The system function to solve, hence the r.h.s. of the ordinary differential equation. It must fulfill the Second Order System concept.
	`t`	The value of the time, at which the step should be performed.

5.
```
template<typename StateIn> void adjust_size(const StateIn & x);
```

Adjust the size of all temporaries in the stepper manually.

Parameters:		
	`x`	A state from which the size of the temporaries to be resized is deduced.

6.
```
void reset(void);
```

Resets the internal state of this stepper. After calling this method it is safe to use all `do_step` method without explicitly initializing the stepper.

7.
```
template<typename AccelerationIn> void initialize(const AccelerationIn & ain);
```

Initializes the internal state of the stepper.

8.
```
template<typename System, typename CoorIn, typename VelocityIn> 
  void initialize(System system, const CoorIn & qin, const VelocityIn & pin, 
                  time_type t);
```

Initializes the internal state of the stepper.

This method is equivalent to

```
Acceleration a;
system( qin , pin , a , t );
stepper.initialize( a );
```

Parameters:		
	`pin`	The current velocities of the ODE.
	`qin`	The current coordinates of the ODE.
	`system`	The system function for the next calls of `do_step`.
	`t`	The current time of the ODE.

9.
```
bool is_initialized(void) const;
```

Returns: Returns if the stepper is initialized.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

`velocity_verlet` private member functions

1.
   ```
   template<typename System, typename CoorIn, typename VelocityIn> 
     void initialize_acc(System system, const CoorIn & qin, 
                         const VelocityIn & pin, time_type t);
   ```

2.
   ```
   template<typename System, typename StateInOut> 
     void do_step_v1(System system, StateInOut & x, time_type t, time_type dt);
   ```

3.
   ```
   template<typename StateIn> bool resize_impl(const StateIn & x);
   ```

4.
   ```
   acceleration_type & get_current_acc(void);
   ```

5.
   ```
   const acceleration_type & get_current_acc(void) const;
   ```

6.
   ```
   acceleration_type & get_old_acc(void);
   ```

7.
   ```
   const acceleration_type & get_old_acc(void) const;
   ```

8.
   ```
   void toggle_current_acc(void);
   ```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Indexes

Class Index

A

adams_bashforth
    Class template adams_bashforth, 146
adams_bashforth_moulton
    Class template adams_bashforth_moulton, 151
adams_moulton
    Class template adams_moulton, 155
adaptive_iterator
    Class template adaptive_iterator, 122
adaptive_time_iterator
    Class template adaptive_time_iterator, 125
algebra_dispatcher
    Class template runge_kutta4_classic, 214
    Class template runge_kutta_cash_karp54_classic, 219
algebra_stepper_base
    Class template adams_bashforth, 146
    Class template velocity_verlet, 239

B

base_tag
    Struct base_tag<controlled_stepper_tag>, 228
    Struct base_tag<dense_output_stepper_tag>, 229
    Struct base_tag<error_stepper_tag>, 229
    Struct base_tag<explicit_controlled_stepper_fsal_tag>, 229
    Struct base_tag<explicit_controlled_stepper_tag>, 230
    Struct base_tag<explicit_error_stepper_fsal_tag>, 230
    Struct base_tag<explicit_error_stepper_tag>, 230
    Struct base_tag<stepper_tag>, 230
    Struct template base_tag, 228
bulirsch_stoer
    Class template bulirsch_stoer, 158
bulirsch_stoer_dense_out
    Class template bulirsch_stoer_dense_out, 163

C

const_step_iterator
    Class template const_step_iterator, 128
const_step_time_iterator
    Class template const_step_time_iterator, 131
controlled_runge_kutta
    Class template controlled_runge_kutta, 169
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175
controlled_stepper_tag
    Struct base_tag<controlled_stepper_tag>, 228
    Struct controlled_stepper_tag, 231
    Struct explicit_controlled_stepper_fsal_tag, 231
    Struct explicit_controlled_stepper_tag, 232
controller_factory
    Generation functions, 75

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

D

default_error_checker
    Class template default_error_checker, 179
default_operations
    Custom Runge-Kutta steppers, 73
default_rosenbrock_coefficients
    Struct template default_rosenbrock_coefficients, 203
dense_output_runge_kutta
    Class template dense_output_runge_kutta, 181
    Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182
    Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185
dense_output_stepper_tag
    Struct base_tag<dense_output_stepper_tag>, 229
    Struct dense_output_stepper_tag, 231

E

error_stepper_tag
    Struct base_tag<error_stepper_tag>, 229
    Struct error_stepper_tag, 231
    Struct explicit_error_stepper_fsal_tag, 232
    Struct explicit_error_stepper_tag, 232
euler
    Class template euler, 188
explicit_controlled_stepper_fsal_tag
    Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182
    Struct base_tag<explicit_controlled_stepper_fsal_tag>, 229
    Struct explicit_controlled_stepper_fsal_tag, 231
explicit_controlled_stepper_tag
    Struct base_tag<explicit_controlled_stepper_tag>, 230
    Struct explicit_controlled_stepper_tag, 232
explicit_error_generic_rk
    Class template explicit_error_generic_rk, 191
    Class template runge_kutta_cash_karp54, 216
    Class template runge_kutta_fehlberg78, 225
explicit_error_stepper_base
    Class template explicit_error_generic_rk, 191
    Class template runge_kutta_cash_karp54_classic, 219
explicit_error_stepper_fsal_base
    Class template runge_kutta_dopri5, 222
explicit_error_stepper_fsal_tag
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170
    Struct base_tag<explicit_error_stepper_fsal_tag>, 230
    Struct explicit_error_stepper_fsal_tag, 232
explicit_error_stepper_tag
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175
    Struct base_tag<explicit_error_stepper_tag>, 230
    Struct explicit_error_stepper_tag, 232
explicit_generic_rk
    Class template explicit_generic_rk, 194
    Class template runge_kutta4, 211
    Custom Runge-Kutta steppers, 73
explicit_stepper_base
    Class template euler, 188
    Class template explicit_generic_rk, 194
    Class template modified_midpoint, 199
    Class template runge_kutta4_classic, 214

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

G

get_controller
    Generation functions, 75
gsl_vector_iterator
    GSL Vector, 87

I

implicit_euler
    Class template implicit_euler, 197
initially_resizer
    Class template runge_kutta4_classic, 214
    Class template runge_kutta_cash_karp54_classic, 219
    Custom Runge-Kutta steppers, 73
is_resizeable
    GSL Vector, 87
    std::list, 86
    Using the container interface, 85

M

modified_midpoint
    Class template modified_midpoint, 199
modified_midpoint_dense_out
    Class template modified_midpoint_dense_out, 201

N

n_step_iterator
    Class template n_step_iterator, 134
n_step_time_iterator
    Class template n_step_time_iterator, 137

O

operations_dispatcher
    Class template runge_kutta4_classic, 214
    Class template runge_kutta_cash_karp54_classic, 219

R

range_algebra
    Custom Runge-Kutta steppers, 73
resize_impl
    GSL Vector, 87
    std::list, 86
rosenbrock4
    Class template rosenbrock4, 204
rosenbrock4_controller
    Class template rosenbrock4_controller, 207
rosenbrock4_dense_output
    Class template rosenbrock4_dense_output, 209
runge_kutta4
    Class template runge_kutta4, 211
runge_kutta4_classic
    Class template runge_kutta4_classic, 214
runge_kutta_cash_karp54
    Class template runge_kutta_cash_karp54, 216
runge_kutta_cash_karp54_classic
    Class template runge_kutta_cash_karp54_classic, 219
runge_kutta_fehlberg78

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- Class template runge_kutta_fehlberg78, 225

S

same_size_impl
- GSL Vector, 87
- std::list, 86

split_impl
- Splitter, 53

state_type
- Custom steppers, 71
- Error Stepper, 102
- std::list, 86

state_wrapper
- GSL Vector, 87

stepper_tag
- Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185
- Struct base_tag<stepper_tag>, 230
- Struct error_stepper_tag, 231
- Struct stepper_tag, 232

symplectic_euler
- Class template symplectic_euler, 233

symplectic_nystroem_stepper_base
- Class template symplectic_euler, 233
- Class template symplectic_rkn_sb3a_m4_mclachlan, 235
- Class template symplectic_rkn_sb3a_mclachlan, 237

symplectic_rkn_sb3a_m4_mclachlan
- Class template symplectic_rkn_sb3a_m4_mclachlan, 235

symplectic_rkn_sb3a_mclachlan
- Class template symplectic_rkn_sb3a_mclachlan, 237

T

times_iterator
- Class template times_iterator, 140

times_time_iterator
- Class template times_time_iterator, 143

time_type
- Custom steppers, 71
- Error Stepper, 102

U

unsplit_impl
- Splitter, 53

V

value_type
- Parameter studies, 42
- Phase oscillator ensemble, 37

vector_space_norm_inf
- Point type, 90

velocity_verlet
- Class template velocity_verlet, 239

Function Index

A

abs

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

  Point type, 90
  Vector Space Algebra, 89
adjust_size
  Class template adams_bashforth, 146, 149
  Class template adams_bashforth_moulton, 151, 153
  Class template adams_moulton, 155, 157
  Class template bulirsch_stoer, 158, 161
  Class template bulirsch_stoer_dense_out, 163, 166
  Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170, 173
  Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175, 178
  Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182-183
  Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185-186
  Class template euler, 188, 190
  Class template explicit_error_generic_rk, 191, 193
  Class template explicit_generic_rk, 194, 196
  Class template implicit_euler, 197
  Class template modified_midpoint, 199-200
  Class template modified_midpoint_dense_out, 201-202
  Class template rosenbrock4, 204-205
  Class template rosenbrock4_controller, 207-208
  Class template rosenbrock4_dense_output, 209-210
  Class template runge_kutta4, 211, 213
  Class template runge_kutta4_classic, 214-215
  Class template runge_kutta_cash_karp54, 216, 218
  Class template runge_kutta_cash_karp54_classic, 219, 221
  Class template runge_kutta_dopri5, 222, 224
  Class template runge_kutta_fehlberg78, 225, 227
  Class template velocity_verlet, 239, 242
advance
  GSL Vector, 87
algebra
  Class template adams_moulton, 155, 157
  MPI, 50

C
calculate_finite_difference
  Class template bulirsch_stoer_dense_out, 163, 167
calc_state
  Class template bulirsch_stoer_dense_out, 163, 166
  Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182-183
  Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185-186
  Class template euler, 188, 190
  Class template rosenbrock4, 204-205
  Class template rosenbrock4_dense_output, 209-210
  Class template runge_kutta_dopri5, 222, 224

D
decrement
  GSL Vector, 87
do_step
  Class template adams_bashforth, 146, 148
  Class template adams_bashforth_moulton, 151, 153
  Class template adams_moulton, 155-157
  Class template bulirsch_stoer_dense_out, 163, 165
  Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182-183
  Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185-186
  Class template implicit_euler, 197

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

    Class template modified_midpoint_dense_out, 201-202
    Class template rosenbrock4, 204-205
    Class template rosenbrock4_dense_output, 209-210
    Class template velocity_verlet, 239, 241
    Custom steppers, 71
    Error Stepper, 102
    Explicit steppers, 58
do_step_impl
    Class template adams_bashforth, 146, 150
    Class template adams_moulton, 155, 157
    Class template euler, 188-189
    Class template explicit_error_generic_rk, 191, 193
    Class template explicit_generic_rk, 194, 196
    Class template modified_midpoint, 199-200
    Class template runge_kutta4, 211-212
    Class template runge_kutta4_classic, 214-215
    Class template runge_kutta_cash_karp54, 216, 218
    Class template runge_kutta_cash_karp54_classic, 219-221
    Class template runge_kutta_dopri5, 222-224
    Class template runge_kutta_fehlberg78, 225, 227
do_step_impl1
    Class template adams_bashforth_moulton, 151, 154
do_step_impl2
    Class template adams_bashforth_moulton, 151, 154
do_step_v1
    Class template velocity_verlet, 239, 243

E
end_iterator
    GSL Vector, 87
error
    Class template default_error_checker, 179-180
    Class template rosenbrock4_controller, 207

F
f
    Custom steppers, 71
    Define the ODE, 10
    Define the system function, 15
    Explicit steppers, 58
    Implicit solvers, 61
    Implicit System, 100
    Overview, 3
    Second Order System, 97
    Short Example, 8
    Stiff systems, 20
    Symplectic solvers, 59
    Symplectic System, 98
    System, 97

G
get_current_acc
    Class template velocity_verlet, 239, 243
get_current_deriv
    Class template bulirsch_stoer_dense_out, 163, 168
    Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182, 184
get_current_state

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

    Class template bulirsch_stoer_dense_out, 163, 167
    Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182-183
    Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185, 187
    Class template rosenbrock4_dense_output, 209-210
get_old_acc
    Class template velocity_verlet, 239, 243
get_old_deriv
    Class template bulirsch_stoer_dense_out, 163, 168
    Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182, 184
get_old_state
    Class template bulirsch_stoer_dense_out, 163, 167-168
    Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182, 184
    Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185, 187
    Class template rosenbrock4_dense_output, 209-210
gsl_vector_iterator
    GSL Vector, 87

I

increment
    GSL Vector, 87
initialize
    Class template adams_bashforth, 146, 149
    Class template adams_bashforth_moulton, 151, 153-154
    Class template bulirsch_stoer_dense_out, 163, 165
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170, 173
    Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182-183
    Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185-186
    Class template rosenbrock4_dense_output, 209
    Class template velocity_verlet, 239, 242
    Using boost::range, 93
    Using steppers, 64
initialize_acc
    Class template velocity_verlet, 239, 243
initializing_stepper
    Class template adams_bashforth, 146, 150
integrate
    Function template integrate, 115-116
    Header < boost/numeric/odeint/integrate/integrate.hpp >, 115
integrate_adaptive
    Function template integrate_adaptive, 117
    Header < boost/numeric/odeint/integrate/integrate_adaptive.hpp >, 116
    Parameter studies, 42
integrate_const
    Ensembles of oscillators, 25
    Function template integrate_const, 118
    Header < boost/numeric/odeint/integrate/integrate_const.hpp >, 118
    Integration with Constant Step Size, 11
    Large oscillator chains, 40
    Using OpenCL via VexCL, 45
integrate_n_steps
    Chaotic systems and Lyapunov exponents, 17
    Function template integrate_n_steps, 119
    Header < boost/numeric/odeint/integrate/integrate_n_steps.hpp >, 119
integrate_times
    Function template integrate_times, 121
    Header < boost/numeric/odeint/integrate/integrate_times.hpp >, 120
iter

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- GSL Vector, 87

M

max
- Adaptive step size algorithms, 62
- Controlled steppers, 62
- Point type, 90

min
- Adaptive step size algorithms, 62
- Controlled steppers, 62

O

ode
- Binding member functions, 95

P

prepare_dense_output
- Class template bulirsch_stoer_dense_out, 163, 167
- Class template rosenbrock4, 204-205

R

range_begin
- GSL Vector, 87

range_end
- GSL Vector, 87

reset
- Class template adams_bashforth, 146, 149
- Class template bulirsch_stoer, 158, 161
- Class template bulirsch_stoer_dense_out, 163, 166
- Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170, 173
- Class template velocity_verlet, 239, 242
- Ensembles of oscillators, 25

resize
- Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182-183
- Class template modified_midpoint_dense_out, 201-202
- GSL Vector, 87
- std::list, 86

resize_dxdt_tmp_impl
- Class template runge_kutta_dopri5, 222, 224

resize_impl
- Class template adams_bashforth, 146, 150
- Class template adams_bashforth_moulton, 151, 154
- Class template adams_moulton, 155, 157
- Class template bulirsch_stoer, 158, 161
- Class template bulirsch_stoer_dense_out, 163, 167
- Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185, 187
- Class template explicit_error_generic_rk, 191, 193
- Class template explicit_generic_rk, 194, 196
- Class template implicit_euler, 197
- Class template modified_midpoint, 199-200
- Class template rosenbrock4, 204-205
- Class template rosenbrock4_dense_output, 209-210
- Class template runge_kutta4_classic, 214-215
- Class template runge_kutta_cash_karp54_classic, 219, 221
- Class template velocity_verlet, 239, 243

resize_k_x_tmp_impl
- Class template runge_kutta_dopri5, 222, 224

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

resize_m_dxdt
    Class template bulirsch_stoer, 158, 161
resize_m_dxdt_impl
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170, 174
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175, 178
resize_m_dxdt_new_impl
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170, 174
resize_m_xerr
    Class template rosenbrock4_controller, 207-208
resize_m_xerr_impl
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170, 174
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175, 178
resize_m_xnew
    Class template bulirsch_stoer, 158, 161
    Class template rosenbrock4_controller, 207-208
resize_m_xnew_impl
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170, 174
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175, 178
resize_x_err
    Class template rosenbrock4, 204, 206

S

same_size
    GSL Vector, 87
    std::list, 86
scale_sum_swap2
    Operations, 108
set_steps
    Class template modified_midpoint, 199-200
    Class template modified_midpoint_dense_out, 201-202
solve
    Class template implicit_euler, 197
split
    Splitter, 53
stepper
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170, 173
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175, 178
    Class template rosenbrock4_controller, 207-208
    Short Example, 8
step_storage
    Class template adams_bashforth, 146, 149
sys
    Large oscillator chains, 40
    Steppers, 58
system
    Parameter studies, 42

T

toggle_current_acc
    Class template velocity_verlet, 239, 243
toggle_current_state
    Class template bulirsch_stoer_dense_out, 163, 168
    Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182, 184
    Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185, 187
    Class template rosenbrock4_dense_output, 209-210
try_step
    Class template bulirsch_stoer, 158, 160-161

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

  Class template bulirsch_stoer_dense_out, 163, 165
  Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170-172
  Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175-177
  Class template rosenbrock4_controller, 207-208
try_step_v1
  Class template bulirsch_stoer, 158, 162
  Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170, 174
  Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175, 178

U

unsplit
  Splitter, 53

Index

A

abs
  Point type, 90
  Vector Space Algebra, 89
acceleration_type
  Class template velocity_verlet, 239
  Using boost::units, 28
adams_bashforth
  Class template adams_bashforth, 146
adams_bashforth_moulton
  Class template adams_bashforth_moulton, 151
adams_moulton
  Class template adams_moulton, 155
Adaptive step size algorithms
  max, 62
  min, 62
adaptive_iterator
  Class template adaptive_iterator, 122
adaptive_time_iterator
  Class template adaptive_time_iterator, 125
adjust_size
  Class template adams_bashforth, 146, 149
  Class template adams_bashforth_moulton, 151, 153
  Class template adams_moulton, 155, 157
  Class template bulirsch_stoer, 158, 161
  Class template bulirsch_stoer_dense_out, 163, 166
  Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170, 173
  Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175, 178
  Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182-183
  Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185-186
  Class template euler, 188, 190
  Class template explicit_error_generic_rk, 191, 193
  Class template explicit_generic_rk, 194, 196
  Class template implicit_euler, 197
  Class template modified_midpoint, 199-200
  Class template modified_midpoint_dense_out, 201-202
  Class template rosenbrock4, 204-205
  Class template rosenbrock4_controller, 207-208
  Class template rosenbrock4_dense_output, 209-210
  Class template runge_kutta4, 211, 213
  Class template runge_kutta4_classic, 214-215
  Class template runge_kutta_cash_karp54, 216, 218

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- Class template runge_kutta_cash_karp54_classic, 219, 221
- Class template runge_kutta_dopri5, 222, 224
- Class template runge_kutta_fehlberg78, 225, 227
- Class template velocity_verlet, 239, 242

advance
- GSL Vector, 87

algebra
- Class template adams_moulton, 155, 157
- MPI, 50

algebra_dispatcher
- Class template runge_kutta4_classic, 214
- Class template runge_kutta_cash_karp54_classic, 219

algebra_stepper_base
- Class template adams_bashforth, 146
- Class template velocity_verlet, 239

algebra_stepper_base_type
- Class template adams_bashforth, 146
- Class template velocity_verlet, 239

algebra_type
- Class template adams_bashforth, 146
- Class template adams_bashforth_moulton, 151
- Class template adams_moulton, 155
- Class template bulirsch_stoer, 158
- Class template bulirsch_stoer_dense_out, 163
- Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170
- Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175
- Class template default_error_checker, 179
- Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182
- Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185
- Class template euler, 188
- Class template explicit_error_generic_rk, 191
- Class template explicit_generic_rk, 194
- Class template modified_midpoint, 199
- Class template modified_midpoint_dense_out, 201
- Class template runge_kutta4, 211
- Class template runge_kutta4_classic, 214
- Class template runge_kutta_cash_karp54, 216
- Class template runge_kutta_cash_karp54_classic, 219
- Class template runge_kutta_dopri5, 222
- Class template runge_kutta_fehlberg78, 225
- Class template symplectic_euler, 233
- Class template symplectic_rkn_sb3a_m4_mclachlan, 235
- Class template symplectic_rkn_sb3a_mclachlan, 237
- Class template velocity_verlet, 239
- Custom Runge-Kutta steppers, 73

All examples
- example, 54

B

base_tag
- Struct base_tag<controlled_stepper_tag>, 228
- Struct base_tag<dense_output_stepper_tag>, 229
- Struct base_tag<error_stepper_tag>, 229
- Struct base_tag<explicit_controlled_stepper_fsal_tag>, 229
- Struct base_tag<explicit_controlled_stepper_tag>, 230
- Struct base_tag<explicit_error_stepper_fsal_tag>, 230
- Struct base_tag<explicit_error_stepper_tag>, 230

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

  - Struct base_tag<stepper_tag>, 230
  - Struct template base_tag, 228
- Binding member functions
  - ode, 95
- book
  - Define the system function, 15
- bulirsch_stoer
  - Class template bulirsch_stoer, 158
- bulirsch_stoer_dense_out
  - Class template bulirsch_stoer_dense_out, 163

C

- calculate_finite_difference
  - Class template bulirsch_stoer_dense_out, 163, 167
- calc_state
  - Class template bulirsch_stoer_dense_out, 163, 166
  - Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182-183
  - Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185-186
  - Class template euler, 188, 190
  - Class template rosenbrock4, 204-205
  - Class template rosenbrock4_dense_output, 209-210
  - Class template runge_kutta_dopri5, 222, 224
- Chaotic systems and Lyapunov exponents
  - equations, 17
  - example, 17
  - integrate_n_steps, 17
  - pre-conditions, 17
  - snippet, 17
  - state_type, 17
- Class template adams_bashforth
  - adams_bashforth, 146
  - adjust_size, 146, 149
  - algebra_stepper_base, 146
  - algebra_stepper_base_type, 146
  - algebra_type, 146
  - deriv_type, 146
  - do_step, 146, 148
  - do_step_impl, 146, 150
  - equations, 148-149
  - initialize, 146, 149
  - initializing_stepper, 146, 150
  - initializing_stepper_type, 146
  - operations_type, 146
  - order_type, 146
  - pre-conditions, 147, 149
  - reset, 146, 149
  - resizer_type, 146
  - resize_impl, 146, 150
  - state_type, 146
  - stepper_category, 146
  - step_storage, 146, 149
  - step_storage_type, 146
  - time_type, 146
  - value_type, 146
  - wrapped_deriv_type, 146
  - wrapped_state_type, 146
- Class template adams_bashforth_moulton

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- adams_bashforth_moulton, 151
- adjust_size, 151, 153
- algebra_type, 151
- deriv_type, 151
- do_step, 151, 153
- do_step_impl1, 151, 154
- do_step_impl2, 151, 154
- equations, 153-154
- initialize, 151, 153-154
- operations_type, 151
- order_type, 151
- pre-conditions, 152, 154
- resizer_type, 151
- resize_impl, 151, 154
- state_type, 151
- stepper_category, 151
- time_type, 151
- value_type, 151
- wrapped_deriv_type, 151
- wrapped_state_type, 151

Class template adams_moulton
- adams_moulton, 155
- adjust_size, 155, 157
- algebra, 155, 157
- algebra_type, 155
- deriv_type, 155
- do_step, 155-157
- do_step_impl, 155, 157
- operations_type, 155
- order_type, 155
- pre-conditions, 155-157
- resizer_type, 155
- resize_impl, 155, 157
- state_type, 155
- stepper_category, 155
- stepper_type, 155
- step_storage_type, 155
- time_type, 155
- value_type, 155
- wrapped_deriv_type, 155
- wrapped_state_type, 155

Class template adaptive_iterator
- adaptive_iterator, 122

Class template adaptive_time_iterator
- adaptive_time_iterator, 125

Class template bulirsch_stoer
- adjust_size, 158, 161
- algebra_type, 158
- bulirsch_stoer, 158
- deriv_type, 158
- operations_type, 158
- reset, 158, 161
- resizer_type, 158
- resize_impl, 158, 161
- resize_m_dxdt, 158, 161
- resize_m_xnew, 158, 161
- state_type, 158
- time_type, 158

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- try_step, 158, 160-161
- try_step_v1, 158, 162
- value_type, 158

Class template bulirsch_stoer_dense_out
- adjust_size, 163, 166
- algebra_type, 163
- bulirsch_stoer_dense_out, 163
- calculate_finite_difference, 163, 167
- calc_state, 163, 166
- deriv_type, 163
- do_step, 163, 165
- equations, 166
- get_current_deriv, 163, 168
- get_current_state, 163, 167
- get_old_deriv, 163, 168
- get_old_state, 163, 167-168
- initialize, 163, 165
- operations_type, 163
- pre-conditions, 163, 166-167
- prepare_dense_output, 163, 167
- reset, 163, 166
- resizer_type, 163
- resize_impl, 163, 167
- state_type, 163
- stepper_category, 163
- time_type, 163
- toggle_current_state, 163, 168
- try_step, 163, 165
- value_type, 163

Class template const_step_iterator
- const_step_iterator, 128

Class template const_step_time_iterator
- const_step_time_iterator, 131

Class template controlled_runge_kutta
- controlled_runge_kutta, 169

Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>
- adjust_size, 170, 173
- algebra_type, 170
- controlled_runge_kutta, 170
- deriv_type, 170
- error_checker_type, 170
- explicit_error_stepper_fsal_tag, 170
- initialize, 170, 173
- operations_type, 170
- reset, 170, 173
- resizer_type, 170
- resize_m_dxdt_impl, 170, 174
- resize_m_dxdt_new_impl, 170, 174
- resize_m_xerr_impl, 170, 174
- resize_m_xnew_impl, 170, 174
- state_type, 170
- stepper, 170, 173
- stepper_category, 170
- stepper_type, 170
- time_type, 170
- try_step, 170-172
- try_step_v1, 170, 174
- value_type, 170

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>
  adjust_size, 175, 178
  algebra_type, 175
  controlled_runge_kutta, 175
  deriv_type, 175
  error_checker_type, 175
  explicit_error_stepper_tag, 175
  operations_type, 175
  resizer_type, 175
  resize_m_dxdt_impl, 175, 178
  resize_m_xerr_impl, 175, 178
  resize_m_xnew_impl, 175, 178
  state_type, 175
  stepper, 175, 178
  stepper_category, 175
  stepper_type, 175
  time_type, 175
  try_step, 175-177
  try_step_v1, 175, 178
  value_type, 175
Class template default_error_checker
  algebra_type, 179
  default_error_checker, 179
  error, 179-180
  operations_type, 179
  value_type, 179
Class template dense_output_runge_kutta
  dense_output_runge_kutta, 181
Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>
  adjust_size, 182-183
  algebra_type, 182
  calc_state, 182-183
  controlled_stepper_type, 182
  dense_output_runge_kutta, 182
  dense_output_stepper_type, 182
  deriv_type, 182
  do_step, 182-183
  explicit_controlled_stepper_fsal_tag, 182
  get_current_deriv, 182, 184
  get_current_state, 182-183
  get_old_deriv, 182, 184
  get_old_state, 182, 184
  initialize, 182-183
  operations_type, 182
  pre-conditions, 182-183
  resize, 182-183
  resizer_type, 182
  state_type, 182
  stepper_category, 182
  stepper_type, 182
  time_type, 182
  toggle_current_state, 182, 184
  value_type, 182
  wrapped_deriv_type, 182
  wrapped_state_type, 182
Class template dense_output_runge_kutta<Stepper, stepper_tag>
  adjust_size, 185-186
  algebra_type, 185

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- calc_state, 185-186
- dense_output_runge_kutta, 185
- dense_output_stepper_type, 185
- deriv_type, 185
- do_step, 185-186
- equations, 186
- get_current_state, 185, 187
- get_old_state, 185, 187
- initialize, 185-186
- operations_type, 185
- pre-conditions, 185, 187
- resizer_type, 185
- resize_impl, 185, 187
- state_type, 185
- stepper_category, 185
- stepper_tag, 185
- stepper_type, 185
- time_type, 185
- toggle_current_state, 185, 187
- value_type, 185
- wrapped_deriv_type, 185
- wrapped_state_type, 185

Class template euler
- adjust_size, 188, 190
- algebra_type, 188
- calc_state, 188, 190
- deriv_type, 188
- do_step_impl, 188-189
- equations, 188
- euler, 188
- explicit_stepper_base, 188
- operations_type, 188
- resizer_type, 188
- state_type, 188
- stepper_base_type, 188
- time_type, 188
- value_type, 188

Class template explicit_error_generic_rk
- adjust_size, 191, 193
- algebra_type, 191
- coef_a_type, 191
- coef_b_type, 191
- coef_c_type, 191
- deriv_type, 191
- do_step_impl, 191, 193
- example, 191-192
- explicit_error_generic_rk, 191
- explicit_error_stepper_base, 191
- operations_type, 191
- resizer_type, 191
- resize_impl, 191, 193
- rk_algorithm_type, 191
- state_type, 191
- stepper_base_type, 191
- time_type, 191
- value_type, 191
- wrapped_deriv_type, 191
- wrapped_state_type, 191

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Class template explicit_generic_rk
    adjust_size, 194, 196
    algebra_type, 194
    coef_a_type, 194
    coef_b_type, 194
    coef_c_type, 194
    deriv_type, 194
    do_step_impl, 194, 196
    example, 195
    explicit_generic_rk, 194
    explicit_stepper_base, 194
    operations_type, 194
    resizer_type, 194
    resize_impl, 194, 196
    rk_algorithm_type, 194
    state_type, 194
    stepper_base_type, 194
    time_type, 194
    value_type, 194
    wrapped_deriv_type, 194
    wrapped_state_type, 194
Class template implicit_euler
    adjust_size, 197
    deriv_type, 197
    do_step, 197
    implicit_euler, 197
    matrix_type, 197
    pmatrix_type, 197
    resizer_type, 197
    resize_impl, 197
    solve, 197
    state_type, 197
    stepper_category, 197
    stepper_type, 197
    time_type, 197
    value_type, 197
    wrapped_deriv_type, 197
    wrapped_matrix_type, 197
    wrapped_pmatrix_type, 197
    wrapped_state_type, 197
Class template modified_midpoint
    adjust_size, 199-200
    algebra_type, 199
    deriv_type, 199
    do_step_impl, 199-200
    explicit_stepper_base, 199
    modified_midpoint, 199
    operations_type, 199
    resizer_type, 199
    resize_impl, 199-200
    set_steps, 199-200
    state_type, 199
    stepper_base_type, 199
    stepper_type, 199
    time_type, 199
    value_type, 199
    wrapped_deriv_type, 199
    wrapped_state_type, 199

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Class template modified_midpoint_dense_out
  adjust_size, 201-202
  algebra_type, 201
  deriv_table_type, 201
  deriv_type, 201
  do_step, 201-202
  modified_midpoint_dense_out, 201
  operations_type, 201
  resize, 201-202
  resizer_type, 201
  set_steps, 201-202
  state_type, 201
  stepper_type, 201
  time_type, 201
  value_type, 201
  wrapped_deriv_type, 201
  wrapped_state_type, 201
Class template n_step_iterator
  n_step_iterator, 134
Class template n_step_time_iterator
  n_step_time_iterator, 137
Class template rosenbrock4
  adjust_size, 204-205
  calc_state, 204-205
  deriv_type, 204
  do_step, 204-205
  matrix_type, 204
  order_type, 204
  pmatrix_type, 204
  pre-conditions, 204-205
  prepare_dense_output, 204-205
  resizer_type, 204
  resize_impl, 204-205
  resize_x_err, 204, 206
  rosenbrock4, 204
  rosenbrock_coefficients, 204
  state_type, 204
  stepper_category, 204
  stepper_type, 204
  time_type, 204
  value_type, 204
  wrapped_deriv_type, 204
  wrapped_matrix_type, 204
  wrapped_pmatrix_type, 204
  wrapped_state_type, 204
Class template rosenbrock4_controller
  adjust_size, 207-208
  controller_type, 207
  deriv_type, 207
  error, 207
  resizer_type, 207
  resize_m_xerr, 207-208
  resize_m_xnew, 207-208
  rosenbrock4_controller, 207
  state_type, 207
  stepper, 207-208
  stepper_category, 207
  stepper_type, 207

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

  time_type, 207
  try_step, 207-208
  value_type, 207
  wrapped_deriv_type, 207
  wrapped_state_type, 207
Class template rosenbrock4_dense_output
  adjust_size, 209-210
  calc_state, 209-210
  controlled_stepper_type, 209
  dense_output_stepper_type, 209
  deriv_type, 209
  do_step, 209-210
  get_current_state, 209-210
  get_old_state, 209-210
  initialize, 209
  pre-conditions, 209-210
  resizer_type, 209
  resize_impl, 209-210
  rosenbrock4_dense_output, 209
  state_type, 209
  stepper_category, 209
  stepper_type, 209
  time_type, 209
  toggle_current_state, 209-210
  value_type, 209
  wrapped_deriv_type, 209
  wrapped_state_type, 209
Class template runge_kutta4
  adjust_size, 211, 213
  algebra_type, 211
  deriv_type, 211
  do_step_impl, 211-212
  equations, 211
  explicit_generic_rk, 211
  operations_type, 211
  resizer_type, 211
  runge_kutta4, 211
  state_type, 211
  time_type, 211
  value_type, 211
Class template runge_kutta4_classic
  adjust_size, 214-215
  algebra_dispatcher, 214
  algebra_type, 214
  deriv_type, 214
  do_step_impl, 214-215
  equations, 214
  explicit_stepper_base, 214
  initially_resizer, 214
  operations_dispatcher, 214
  operations_type, 214
  resizer_type, 214
  resize_impl, 214-215
  runge_kutta4_classic, 214
  state_type, 214
  stepper_base_type, 214
  time_type, 214
  value_type, 214

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Class template runge_kutta_cash_karp54
    adjust_size, 216, 218
    algebra_type, 216
    deriv_type, 216
    do_step_impl, 216, 218
    equations, 217
    explicit_error_generic_rk, 216
    operations_type, 216
    resizer_typ, 216
    runge_kutta_cash_karp54, 216
    state_type, 216
    time_type, 216
    value_type, 216
Class template runge_kutta_cash_karp54_classic
    adjust_size, 219, 221
    algebra_dispatcher, 219
    algebra_type, 219
    deriv_type, 219
    do_step_impl, 219-221
    equations, 219
    explicit_error_stepper_base, 219
    initially_resizer, 219
    operations_dispatcher, 219
    operations_type, 219
    resizer_type, 219
    resize_impl, 219, 221
    runge_kutta_cash_karp54_classic, 219
    state_type, 219
    stepper_base_type, 219
    time_type, 219
    value_type, 219
Class template runge_kutta_dopri5
    adjust_size, 222, 224
    algebra_type, 222
    calc_state, 222, 224
    deriv_type, 222
    do_step_impl, 222-224
    explicit_error_stepper_fsal_base, 222
    operations_type, 222
    resizer_type, 222
    resize_dxdt_tmp_impl, 222, 224
    resize_k_x_tmp_impl, 222, 224
    state_type, 222
    stepper_base_type, 222
    time_type, 222
    value_type, 222
Class template runge_kutta_fehlberg78
    adjust_size, 225, 227
    algebra_type, 225
    deriv_type, 225
    do_step_impl, 225, 227
    explicit_error_generic_rk, 225
    operations_type, 225
    pre-conditions, 226
    resizer_type, 225
    runge_kutta_fehlberg78, 225
    state_type, 225
    time_type, 225

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

    value_type, 225
Class template symplectic_euler
    algebra_type, 233
    pre-conditions, 233
    symplectic_euler, 233
    symplectic_nystroem_stepper_base, 233
    value_type, 233
Class template symplectic_rkn_sb3a_m4_mclachlan
    algebra_type, 235
    pre-conditions, 235
    symplectic_nystroem_stepper_base, 235
    symplectic_rkn_sb3a_m4_mclachlan, 235
    value_type, 235
Class template symplectic_rkn_sb3a_mclachlan
    algebra_type, 237
    pre-conditions, 237
    symplectic_nystroem_stepper_base, 237
    symplectic_rkn_sb3a_mclachlan, 237
    value_type, 237
Class template times_iterator
    times_iterator, 140
Class template times_time_iterator
    times_time_iterator, 143
Class template velocity_verlet
    acceleration_type, 239
    adjust_size, 239, 242
    algebra_stepper_base, 239
    algebra_stepper_base_type, 239
    algebra_type, 239
    coor_type, 239
    deriv_type, 239
    do_step, 239, 241
    do_step_v1, 239, 243
    equations, 241-242
    get_current_acc, 239, 243
    get_old_acc, 239, 243
    initialize, 239, 242
    initialize_acc, 239, 243
    operations_type, 239
    order_type, 239
    reset, 239, 242
    resizer_type, 239
    resize_impl, 239, 243
    state_type, 239
    stepper_category, 239
    time_square_type, 239
    time_type, 239
    toggle_current_acc, 239, 243
    value_type, 239
    velocity_type, 239
    velocity_verlet, 239
    wrapped_acceleration_type, 239
coef_a_type
    Class template explicit_error_generic_rk, 191
    Class template explicit_generic_rk, 194
coef_b_type
    Class template explicit_error_generic_rk, 191
    Class template explicit_generic_rk, 194

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

coef_c_type
    Class template explicit_error_generic_rk, 191
    Class template explicit_generic_rk, 194
Complex state types
    example, 22
    pre-conditions, 22
    state_type, 22
    stepper_type, 22
Construction/Resizing
    example, 84
const_step_iterator
    Class template const_step_iterator, 128
const_step_time_iterator
    Class template const_step_time_iterator, 131
Controlled steppers
    example, 62
    max, 62
    min, 62
    pre-conditions, 62
controlled_runge_kutta
    Class template controlled_runge_kutta, 169
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175
controlled_stepper_tag
    Struct base_tag<controlled_stepper_tag>, 228
    Struct controlled_stepper_tag, 231
    Struct explicit_controlled_stepper_fsal_tag, 231
    Struct explicit_controlled_stepper_tag, 232
controlled_stepper_type
    Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182
    Class template rosenbrock4_dense_output, 209
    Integration with Adaptive Step Size, 11
controller_factory
    Generation functions, 75
controller_type
    Class template rosenbrock4_controller, 207
coor_type
    Class template velocity_verlet, 239
Custom Runge-Kutta steppers
    algebra_type, 73
    default_operations, 73
    deriv_type, 73
    example, 73
    explicit_generic_rk, 73
    initially_resizer, 73
    operations_type, 73
    pre-conditions, 73
    range_algebra, 73
    resizer_type, 73
    state_type, 73
    stepper_base_type, 73
    stepper_type, 73
    time_type, 73
    value_type, 73
    wrapped_deriv_type, 73
    wrapped_state_type, 73
Custom steppers
    deriv_type, 71

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- do_step, 71
- equations, 71
- example, 71
- f, 71
- order_type, 71
- state_type, 71
- stepper_category, 71
- time_type, 71
- value_type, 71

D

- decrement
  - GSL Vector, 87
- default_error_checker
  - Class template default_error_checker, 179
- default_operations
  - Custom Runge-Kutta steppers, 73
- default_rosenbrock_coefficients
  - Struct template default_rosenbrock_coefficients, 203
- Define the ODE
  - equations, 10
  - example, 10
  - f, 10
  - state_type, 10
- Define the system function
  - book, 15
  - equations, 15
  - example, 15
  - f, 15
  - pre-conditions, 15
  - stepper_type, 15
  - value_type, 15
- Dense output steppers
  - example, 63
- dense_output_runge_kutta
  - Class template dense_output_runge_kutta, 181
  - Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182
  - Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185
- dense_output_stepper_tag
  - Struct base_tag<dense_output_stepper_tag>, 229
  - Struct dense_output_stepper_tag, 231
- dense_output_stepper_type
  - Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182
  - Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185
  - Class template rosenbrock4_dense_output, 209
- deriv_table_type
  - Class template modified_midpoint_dense_out, 201
- deriv_type
  - Class template adams_bashforth, 146
  - Class template adams_bashforth_moulton, 151
  - Class template adams_moulton, 155
  - Class template bulirsch_stoer, 158
  - Class template bulirsch_stoer_dense_out, 163
  - Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170
  - Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175
  - Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182
  - Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

    Class template euler, 188
    Class template explicit_error_generic_rk, 191
    Class template explicit_generic_rk, 194
    Class template implicit_euler, 197
    Class template modified_midpoint, 199
    Class template modified_midpoint_dense_out, 201
    Class template rosenbrock4, 204
    Class template rosenbrock4_controller, 207
    Class template rosenbrock4_dense_output, 209
    Class template runge_kutta4, 211
    Class template runge_kutta4_classic, 214
    Class template runge_kutta_cash_karp54, 216
    Class template runge_kutta_cash_karp54_classic, 219
    Class template runge_kutta_dopri5, 222
    Class template runge_kutta_fehlberg78, 225
    Class template velocity_verlet, 239
    Custom Runge-Kutta steppers, 73
    Custom steppers, 71
    Using boost::units, 28
do_step
    Class template adams_bashforth, 146, 148
    Class template adams_bashforth_moulton, 151, 153
    Class template adams_moulton, 155-157
    Class template bulirsch_stoer_dense_out, 163, 165
    Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182-183
    Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185-186
    Class template implicit_euler, 197
    Class template modified_midpoint_dense_out, 201-202
    Class template rosenbrock4, 204-205
    Class template rosenbrock4_dense_output, 209-210
    Class template velocity_verlet, 239, 241
    Custom steppers, 71
    Error Stepper, 102
    Explicit steppers, 58
do_step_impl
    Class template adams_bashforth, 146, 150
    Class template adams_moulton, 155, 157
    Class template euler, 188-189
    Class template explicit_error_generic_rk, 191, 193
    Class template explicit_generic_rk, 194, 196
    Class template modified_midpoint, 199-200
    Class template runge_kutta4, 211-212
    Class template runge_kutta4_classic, 214-215
    Class template runge_kutta_cash_karp54, 216, 218
    Class template runge_kutta_cash_karp54_classic, 219-221
    Class template runge_kutta_dopri5, 222-224
    Class template runge_kutta_fehlberg78, 225, 227
do_step_impl1
    Class template adams_bashforth_moulton, 151, 154
do_step_impl2
    Class template adams_bashforth_moulton, 151, 154
do_step_v1
    Class template velocity_verlet, 239, 243

E

end_iterator
    GSL Vector, 87

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Ensembles of oscillators
    equations, 25
    example, 25
    integrate_const, 25
    reset, 25
equations
    Chaotic systems and Lyapunov exponents, 17
    Class template adams_bashforth, 148-149
    Class template adams_bashforth_moulton, 153-154
    Class template bulirsch_stoer_dense_out, 166
    Class template dense_output_runge_kutta<Stepper, stepper_tag>, 186
    Class template euler, 188
    Class template runge_kutta4, 211
    Class template runge_kutta4_classic, 214
    Class template runge_kutta_cash_karp54, 217
    Class template runge_kutta_cash_karp54_classic, 219
    Class template velocity_verlet, 241-242
    Custom steppers, 71
    Define the ODE, 10
    Define the system function, 15
    Ensembles of oscillators, 25
    Examples, 79
    Examples Overview, 54
    Explicit steppers, 58
    Function template integrate, 115-116
    Gravitation and energy conservation, 13
    Implicit solvers, 61
    Large oscillator chains, 40
    Lattice systems, 23
    Literature, 113
    Overview, 3
    Parameter studies, 42
    Short Example, 8
    Simple Symplectic System, 99
    State Algebra Operations, 107
    Steppers, 58
    Stiff systems, 20
    Symplectic solvers, 59
    Symplectic System, 98
    Using boost::units, 28
    Using CUDA (or OpenMP, TBB, ...) via Thrust, 36
    Using matrices as state types, 30
    Using steppers, 64
error
    Class template default_error_checker, 179-180
    Class template rosenbrock4_controller, 207
Error Stepper
    do_step, 102
    state_type, 102
    time_type, 102
error_checker_type
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175
error_stepper_tag
    Struct base_tag<error_stepper_tag>, 229
    Struct error_stepper_tag, 231
    Struct explicit_error_stepper_fsal_tag, 232
    Struct explicit_error_stepper_tag, 232

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

euler
    Class template euler, 188
example
    All examples, 54
    Chaotic systems and Lyapunov exponents, 17
    Class template explicit_error_generic_rk, 191-192
    Class template explicit_generic_rk, 195
    Complex state types, 22
    Construction/Resizing, 84
    Controlled steppers, 62
    Custom Runge-Kutta steppers, 73
    Custom steppers, 71
    Define the ODE, 10
    Define the system function, 15
    Dense output steppers, 63
    Ensembles of oscillators, 25
    Example expressions, 111
    Examples, 79
    Examples Overview, 54
    Explicit steppers, 58
    Gravitation and energy conservation, 13
    GSL Vector, 87
    Harmonic oscillator, 10
    Integration with Adaptive Step Size, 11
    Integration with Constant Step Size, 11
    Large oscillator chains, 40
    Lattice systems, 23
    MPI, 50
    Multistep methods, 61
    OpenMP, 47
    Overview, 3
    Parameter studies, 42
    Phase oscillator ensemble, 37
    Point type, 90
    Self expanding lattices, 33
    Short Example, 8
    State Algebra Operations, 107
    State types, algebras and operations, 83
    std::list, 86
    Stepper Types, 10
    Stiff systems, 20
    Symplectic solvers, 59
    Using arbitrary precision floating point types, 32
    Using boost::range, 93
    Using boost::ref, 92
    Using boost::units, 28
    Using CUDA (or OpenMP, TBB, ...) via Thrust, 36
    Using iterators, 13
    Using matrices as state types, 30
    Using OpenCL via VexCL, 45
    Using steppers, 64
    Using the container interface, 85
Example expressions
    example, 111
Examples
    equations, 79
    example, 79
Examples Overview

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

  - equations, 54
  - example, 54
  - graphics, 54
  - pre-conditions, 54
- Explicit steppers
  - do_step, 58
  - equations, 58
  - example, 58
  - f, 58
- explicit_controlled_stepper_fsal_tag
  - Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182
  - Struct base_tag<explicit_controlled_stepper_fsal_tag>, 229
  - Struct explicit_controlled_stepper_fsal_tag, 231
- explicit_controlled_stepper_tag
  - Struct base_tag<explicit_controlled_stepper_tag>, 230
  - Struct explicit_controlled_stepper_tag, 232
- explicit_error_generic_rk
  - Class template explicit_error_generic_rk, 191
  - Class template runge_kutta_cash_karp54, 216
  - Class template runge_kutta_fehlberg78, 225
- explicit_error_stepper_base
  - Class template explicit_error_generic_rk, 191
  - Class template runge_kutta_cash_karp54_classic, 219
- explicit_error_stepper_fsal_base
  - Class template runge_kutta_dopri5, 222
- explicit_error_stepper_fsal_tag
  - Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170
  - Struct base_tag<explicit_error_stepper_fsal_tag>, 230
  - Struct explicit_error_stepper_fsal_tag, 232
- explicit_error_stepper_tag
  - Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175
  - Struct base_tag<explicit_error_stepper_tag>, 230
  - Struct explicit_error_stepper_tag, 232
- explicit_generic_rk
  - Class template explicit_generic_rk, 194
  - Class template runge_kutta4, 211
  - Custom Runge-Kutta steppers, 73
- explicit_stepper_base
  - Class template euler, 188
  - Class template explicit_generic_rk, 194
  - Class template modified_midpoint, 199
  - Class template runge_kutta4_classic, 214

F

- f
  - Custom steppers, 71
  - Define the ODE, 10
  - Define the system function, 15
  - Explicit steppers, 58
  - Implicit solvers, 61
  - Implicit System, 100
  - Overview, 3
  - Second Order System, 97
  - Short Example, 8
  - Stiff systems, 20
  - Symplectic solvers, 59
  - Symplectic System, 98

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

  System, 97
Function template integrate
  equations, 115-116
  integrate, 115-116
Function template integrate_adaptive
  integrate_adaptive, 117
Function template integrate_const
  integrate_const, 118
Function template integrate_n_steps
  integrate_n_steps, 119
Function template integrate_times
  integrate_times, 121

G

Generation functions
  controller_factory, 75
  get_controller, 75
  type, 75
Generation functions make_controlled(abs_error , rel_error , stepper)
  remark, 11, 75
Generation functions make_dense_output(abs_error , rel_error , stepper)
  remark, 11, 75
get_controller
  Generation functions, 75
get_current_acc
  Class template velocity_verlet, 239, 243
get_current_deriv
  Class template bulirsch_stoer_dense_out, 163, 168
  Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182, 184
get_current_state
  Class template bulirsch_stoer_dense_out, 163, 167
  Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182-183
  Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185, 187
  Class template rosenbrock4_dense_output, 209-210
get_old_acc
  Class template velocity_verlet, 239, 243
get_old_deriv
  Class template bulirsch_stoer_dense_out, 163, 168
  Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182, 184
get_old_state
  Class template bulirsch_stoer_dense_out, 163, 167-168
  Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182, 184
  Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185, 187
  Class template rosenbrock4_dense_output, 209-210
graphics
  Examples Overview, 54
  Using CUDA (or OpenMP, TBB, ...) via Thrust, 36
Gravitation and energy conservation
  equations, 13
  example, 13
GSL Vector
  advance, 87
  decrement, 87
  end_iterator, 87
  example, 87
  gsl_vector_iterator, 87
  increment, 87

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- is_resizeable, 87
- iter, 87
- pre-conditions, 87
- range_begin, 87
- range_end, 87
- resize, 87
- resize_impl, 87
- same_size, 87
- same_size_impl, 87
- state_type, 87
- state_wrapper, 87
- state_wrapper_type, 87
- type, 87
- value_type, 87

gsl_vector_iterator
- GSL Vector, 87

H

Harmonic oscillator
- example, 10

Header < boost/numeric/odeint/integrate/integrate.hpp >
- integrate, 115

Header < boost/numeric/odeint/integrate/integrate_adaptive.hpp >
- integrate_adaptive, 116

Header < boost/numeric/odeint/integrate/integrate_const.hpp >
- integrate_const, 118

Header < boost/numeric/odeint/integrate/integrate_n_steps.hpp >
- integrate_n_steps, 119

Header < boost/numeric/odeint/integrate/integrate_times.hpp >
- integrate_times, 120

I

Implicit solvers
- equations, 61
- f, 61

Implicit System
- f, 100

implicit_euler
- Class template implicit_euler, 197

increment
- GSL Vector, 87

index
- Indexes, 244
- Large oscillator chains, 40
- Self expanding lattices, 33

Indexes
- index, 244

initialize
- Class template adams_bashforth, 146, 149
- Class template adams_bashforth_moulton, 151, 153-154
- Class template bulirsch_stoer_dense_out, 163, 165
- Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170, 173
- Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182-183
- Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185-186
- Class template rosenbrock4_dense_output, 209
- Class template velocity_verlet, 239, 242
- Using boost::range, 93

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

  Using steppers, 64
initialize_acc
  Class template velocity_verlet, 239, 243
initializing_stepper
  Class template adams_bashforth, 146, 150
initializing_stepper_type
  Class template adams_bashforth, 146
initially_resizer
  Class template runge_kutta4_classic, 214
  Class template runge_kutta_cash_karp54_classic, 219
  Custom Runge-Kutta steppers, 73
integrate
  Function template integrate, 115-116
  Header < boost/numeric/odeint/integrate/integrate.hpp >, 115
Integrate functions
  pre-conditions, 77
integrate_adaptive
  Function template integrate_adaptive, 117
  Header < boost/numeric/odeint/integrate/integrate_adaptive.hpp >, 116
  Parameter studies, 42
integrate_const
  Ensembles of oscillators, 25
  Function template integrate_const, 118
  Header < boost/numeric/odeint/integrate/integrate_const.hpp >, 118
  Integration with Constant Step Size, 11
  Large oscillator chains, 40
  Using OpenCL via VexCL, 45
integrate_n_steps
  Chaotic systems and Lyapunov exponents, 17
  Function template integrate_n_steps, 119
  Header < boost/numeric/odeint/integrate/integrate_n_steps.hpp >, 119
integrate_times
  Function template integrate_times, 121
  Header < boost/numeric/odeint/integrate/integrate_times.hpp >, 120
Integration with Adaptive Step Size
  controlled_stepper_type, 11
  example, 11
Integration with Constant Step Size
  example, 11
  integrate_const, 11
is_resizeable
  GSL Vector, 87
  std::list, 86
  Using the container interface, 85
iter
  GSL Vector, 87
iterator
  Using the container interface, 85

L

Large oscillator chains
  equations, 40
  example, 40
  index, 40
  integrate_const, 40
  pre-conditions, 40
  state_type, 40

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- sys, 40

Lattice systems
- equations, 23
- example, 23
- remark, 23
- stepper_type, 23

links
- Usage, Compilation, Headers, 7

Literature
- equations, 113
- pre-conditions, 113

M

matrix_type
- Class template implicit_euler, 197
- Class template rosenbrock4, 204
- Stiff systems, 20

max
- Adaptive step size algorithms, 62
- Controlled steppers, 62
- Point type, 90

min
- Adaptive step size algorithms, 62
- Controlled steppers, 62

modified_midpoint
- Class template modified_midpoint, 199

modified_midpoint_dense_out
- Class template modified_midpoint_dense_out, 201

MPI
- algebra, 50
- example, 50
- state_type, 50

Multistep methods
- example, 61
- pre-conditions, 61

N

n_step_iterator
- Class template n_step_iterator, 134

n_step_time_iterator
- Class template n_step_time_iterator, 137

O

ode
- Binding member functions, 95

OpenMP
- example, 47
- state_type, 47
- stepper_type, 47

Operations
- scale_sum_swap2, 108

operations_dispatcher
- Class template runge_kutta4_classic, 214
- Class template runge_kutta_cash_karp54_classic, 219

operations_type
- Class template adams_bashforth, 146
- Class template adams_bashforth_moulton, 151

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

    Class template adams_moulton, 155
    Class template bulirsch_stoer, 158
    Class template bulirsch_stoer_dense_out, 163
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175
    Class template default_error_checker, 179
    Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182
    Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185
    Class template euler, 188
    Class template explicit_error_generic_rk, 191
    Class template explicit_generic_rk, 194
    Class template modified_midpoint, 199
    Class template modified_midpoint_dense_out, 201
    Class template runge_kutta4, 211
    Class template runge_kutta4_classic, 214
    Class template runge_kutta_cash_karp54, 216
    Class template runge_kutta_cash_karp54_classic, 219
    Class template runge_kutta_dopri5, 222
    Class template runge_kutta_fehlberg78, 225
    Class template velocity_verlet, 239
    Custom Runge-Kutta steppers, 73
order_type
    Class template adams_bashforth, 146
    Class template adams_bashforth_moulton, 151
    Class template adams_moulton, 155
    Class template rosenbrock4, 204
    Class template velocity_verlet, 239
    Custom steppers, 71
    Struct template default_rosenbrock_coefficients, 203
Overview
    equations, 3
    example, 3
    f, 3

P
Parameter studies
    equations, 42
    example, 42
    integrate_adaptive, 42
    pre-conditions, 42
    stepper_type, 42
    system, 42
    value_type, 42
path
    Usage, Compilation, Headers, 7
Phase oscillator ensemble
    example, 37
    pre-conditions, 37
    state_type, 37
    stepper_type, 37
    value_type, 37
pmatrix_type
    Class template implicit_euler, 197
    Class template rosenbrock4, 204
Point type
    abs, 90
    example, 90

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

    max, 90
    result_type, 90
    vector_space_norm_inf, 90
pre-conditions
    Chaotic systems and Lyapunov exponents, 17
    Class template adams_bashforth, 147, 149
    Class template adams_bashforth_moulton, 152, 154
    Class template adams_moulton, 155-157
    Class template bulirsch_stoer_dense_out, 163, 166-167
    Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182-183
    Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185, 187
    Class template rosenbrock4, 204-205
    Class template rosenbrock4_dense_output, 209-210
    Class template runge_kutta_fehlberg78, 226
    Class template symplectic_euler, 233
    Class template symplectic_rkn_sb3a_m4_mclachlan, 235
    Class template symplectic_rkn_sb3a_mclachlan, 237
    Complex state types, 22
    Controlled steppers, 62
    Custom Runge-Kutta steppers, 73
    Define the system function, 15
    Examples Overview, 54
    GSL Vector, 87
    Integrate functions, 77
    Large oscillator chains, 40
    Literature, 113
    Multistep methods, 61
    Parameter studies, 42
    Phase oscillator ensemble, 37
    Pre-Defined implementations, 110
    Stepper Algorithms, 3, 67
    Steppers, 58
    Usage, Compilation, Headers, 7
    Using arbitrary precision floating point types, 32
    Using OpenCL via VexCL, 45
    Using steppers, 66
Pre-Defined implementations
    pre-conditions, 110
    remark, 110
prepare_dense_output
    Class template bulirsch_stoer_dense_out, 163, 167
    Class template rosenbrock4, 204-205

R

range_algebra
    Custom Runge-Kutta steppers, 73
range_begin
    GSL Vector, 87
range_end
    GSL Vector, 87
remark
    Generation functions make_controlled(abs_error , rel_error , stepper), 11, 75
    Generation functions make_dense_output(abs_error , rel_error , stepper), 11, 75
    Lattice systems, 23
    Pre-Defined implementations, 110
    Stepper Algorithms, 3, 67
reset

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

  - Class template adams_bashforth, 146, 149
  - Class template bulirsch_stoer, 158, 161
  - Class template bulirsch_stoer_dense_out, 163, 166
  - Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170, 173
  - Class template velocity_verlet, 239, 242
  - Ensembles of oscillators, 25
- resize
  - Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182-183
  - Class template modified_midpoint_dense_out, 201-202
  - GSL Vector, 87
  - std::list, 86
- resizer_typ
  - Class template runge_kutta_cash_karp54, 216
- resizer_type
  - Class template adams_bashforth, 146
  - Class template adams_bashforth_moulton, 151
  - Class template adams_moulton, 155
  - Class template bulirsch_stoer, 158
  - Class template bulirsch_stoer_dense_out, 163
  - Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170
  - Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175
  - Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182
  - Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185
  - Class template euler, 188
  - Class template explicit_error_generic_rk, 191
  - Class template explicit_generic_rk, 194
  - Class template implicit_euler, 197
  - Class template modified_midpoint, 199
  - Class template modified_midpoint_dense_out, 201
  - Class template rosenbrock4, 204
  - Class template rosenbrock4_controller, 207
  - Class template rosenbrock4_dense_output, 209
  - Class template runge_kutta4, 211
  - Class template runge_kutta4_classic, 214
  - Class template runge_kutta_cash_karp54_classic, 219
  - Class template runge_kutta_dopri5, 222
  - Class template runge_kutta_fehlberg78, 225
  - Class template velocity_verlet, 239
  - Custom Runge-Kutta steppers, 73
- resize_dxdt_tmp_impl
  - Class template runge_kutta_dopri5, 222, 224
- resize_impl
  - Class template adams_bashforth, 146, 150
  - Class template adams_bashforth_moulton, 151, 154
  - Class template adams_moulton, 155, 157
  - Class template bulirsch_stoer, 158, 161
  - Class template bulirsch_stoer_dense_out, 163, 167
  - Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185, 187
  - Class template explicit_error_generic_rk, 191, 193
  - Class template explicit_generic_rk, 194, 196
  - Class template implicit_euler, 197
  - Class template modified_midpoint, 199-200
  - Class template rosenbrock4, 204-205
  - Class template rosenbrock4_dense_output, 209-210
  - Class template runge_kutta4_classic, 214-215
  - Class template runge_kutta_cash_karp54_classic, 219, 221
  - Class template velocity_verlet, 239, 243
  - GSL Vector, 87

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- std::list, 86

resize_k_x_tmp_impl
- Class template runge_kutta_dopri5, 222, 224

resize_m_dxdt
- Class template bulirsch_stoer, 158, 161

resize_m_dxdt_impl
- Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170, 174
- Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175, 178

resize_m_dxdt_new_impl
- Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170, 174

resize_m_xerr
- Class template rosenbrock4_controller, 207-208

resize_m_xerr_impl
- Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170, 174
- Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175, 178

resize_m_xnew
- Class template bulirsch_stoer, 158, 161
- Class template rosenbrock4_controller, 207-208

resize_m_xnew_impl
- Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170, 174
- Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175, 178

resize_x_err
- Class template rosenbrock4, 204, 206

result_type
- Point type, 90

rk_algorithm_type
- Class template explicit_error_generic_rk, 191
- Class template explicit_generic_rk, 194

rosenbrock4
- Class template rosenbrock4, 204

rosenbrock4_controller
- Class template rosenbrock4_controller, 207

rosenbrock4_dense_output
- Class template rosenbrock4_dense_output, 209

rosenbrock_coefficients
- Class template rosenbrock4, 204

runge_kutta4
- Class template runge_kutta4, 211

runge_kutta4_classic
- Class template runge_kutta4_classic, 214

runge_kutta_cash_karp54
- Class template runge_kutta_cash_karp54, 216

runge_kutta_cash_karp54_classic
- Class template runge_kutta_cash_karp54_classic, 219

runge_kutta_fehlberg78
- Class template runge_kutta_fehlberg78, 225

S

same_size
- GSL Vector, 87
- std::list, 86

same_size_impl
- GSL Vector, 87
- std::list, 86

scale_sum_swap2
- Operations, 108

Second Order System

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

  f, 97
Self expanding lattices
  example, 33
  index, 33
  state_type, 33
set_steps
  Class template modified_midpoint, 199-200
  Class template modified_midpoint_dense_out, 201-202
Short Example
  equations, 8
  example, 8
  f, 8
  state_type, 8
  stepper, 8
Simple Symplectic System
  equations, 99
snippet
  Chaotic systems and Lyapunov exponents, 17
solve
  Class template implicit_euler, 197
split
  Splitter, 53
Splitter
  split, 53
  split_impl, 53
  unsplit, 53
  unsplit_impl, 53
split_impl
  Splitter, 53
State Algebra Operations
  equations, 107
  example, 107
State types, algebras and operations
  example, 83
state_type
  Chaotic systems and Lyapunov exponents, 17
  Class template adams_bashforth, 146
  Class template adams_bashforth_moulton, 151
  Class template adams_moulton, 155
  Class template bulirsch_stoer, 158
  Class template bulirsch_stoer_dense_out, 163
  Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170
  Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175
  Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182
  Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185
  Class template euler, 188
  Class template explicit_error_generic_rk, 191
  Class template explicit_generic_rk, 194
  Class template implicit_euler, 197
  Class template modified_midpoint, 199
  Class template modified_midpoint_dense_out, 201
  Class template rosenbrock4, 204
  Class template rosenbrock4_controller, 207
  Class template rosenbrock4_dense_output, 209
  Class template runge_kutta4, 211
  Class template runge_kutta4_classic, 214
  Class template runge_kutta_cash_karp54, 216
  Class template runge_kutta_cash_karp54_classic, 219

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

    Class template runge_kutta_dopri5, 222
    Class template runge_kutta_fehlberg78, 225
    Class template velocity_verlet, 239
    Complex state types, 22
    Custom Runge-Kutta steppers, 73
    Custom steppers, 71
    Define the ODE, 10
    Error Stepper, 102
    GSL Vector, 87
    Large oscillator chains, 40
    MPI, 50
    OpenMP, 47
    Phase oscillator ensemble, 37
    Self expanding lattices, 33
    Short Example, 8
    std::list, 86
    Using arbitrary precision floating point types, 32
    Using boost::units, 28
    Using matrices as state types, 30
    Using OpenCL via VexCL, 45
state_wrapper
    GSL Vector, 87
state_wrapper_type
    GSL Vector, 87
std::list
    example, 86
    is_resizeable, 86
    resize, 86
    resize_impl, 86
    same_size, 86
    same_size_impl, 86
    state_type, 86
    type, 86
stepper
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170, 173
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175, 178
    Class template rosenbrock4_controller, 207-208
    Short Example, 8
Stepper Algorithms
    pre-conditions, 3, 67
    remark, 3, 67
Stepper Types
    example, 10
Steppers
    equations, 58
    pre-conditions, 58
    sys, 58
stepper_base_type
    Class template euler, 188
    Class template explicit_error_generic_rk, 191
    Class template explicit_generic_rk, 194
    Class template modified_midpoint, 199
    Class template runge_kutta4_classic, 214
    Class template runge_kutta_cash_karp54_classic, 219
    Class template runge_kutta_dopri5, 222
    Custom Runge-Kutta steppers, 73
stepper_category
    Class template adams_bashforth, 146

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- Class template adams_bashforth_moulton, 151
- Class template adams_moulton, 155
- Class template bulirsch_stoer_dense_out, 163
- Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170
- Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175
- Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182
- Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185
- Class template implicit_euler, 197
- Class template rosenbrock4, 204
- Class template rosenbrock4_controller, 207
- Class template rosenbrock4_dense_output, 209
- Class template velocity_verlet, 239
- Custom steppers, 71

stepper_tag
- Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185
- Struct base_tag<stepper_tag>, 230
- Struct error_stepper_tag, 231
- Struct stepper_tag, 232

stepper_type
- Class template adams_moulton, 155
- Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170
- Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175
- Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182
- Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185
- Class template implicit_euler, 197
- Class template modified_midpoint, 199
- Class template modified_midpoint_dense_out, 201
- Class template rosenbrock4, 204
- Class template rosenbrock4_controller, 207
- Class template rosenbrock4_dense_output, 209
- Complex state types, 22
- Custom Runge-Kutta steppers, 73
- Define the system function, 15
- Lattice systems, 23
- OpenMP, 47
- Parameter studies, 42
- Phase oscillator ensemble, 37
- Using boost::units, 28

step_storage
- Class template adams_bashforth, 146, 149

step_storage_type
- Class template adams_bashforth, 146
- Class template adams_moulton, 155

Stiff systems
- equations, 20
- example, 20
- f, 20
- matrix_type, 20

Struct base_tag<controlled_stepper_tag>
- base_tag, 228
- controlled_stepper_tag, 228
- type, 228

Struct base_tag<dense_output_stepper_tag>
- base_tag, 229
- dense_output_stepper_tag, 229
- type, 229

Struct base_tag<error_stepper_tag>
- base_tag, 229

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

  - error_stepper_tag, 229
  - type, 229
- Struct base_tag<explicit_controlled_stepper_fsal_tag>
  - base_tag, 229
  - explicit_controlled_stepper_fsal_tag, 229
  - type, 229
- Struct base_tag<explicit_controlled_stepper_tag>
  - base_tag, 230
  - explicit_controlled_stepper_tag, 230
  - type, 230
- Struct base_tag<explicit_error_stepper_fsal_tag>
  - base_tag, 230
  - explicit_error_stepper_fsal_tag, 230
  - type, 230
- Struct base_tag<explicit_error_stepper_tag>
  - base_tag, 230
  - explicit_error_stepper_tag, 230
  - type, 230
- Struct base_tag<stepper_tag>
  - base_tag, 230
  - stepper_tag, 230
  - type, 230
- Struct controlled_stepper_tag
  - controlled_stepper_tag, 231
- Struct dense_output_stepper_tag
  - dense_output_stepper_tag, 231
- Struct error_stepper_tag
  - error_stepper_tag, 231
  - stepper_tag, 231
- Struct explicit_controlled_stepper_fsal_tag
  - controlled_stepper_tag, 231
  - explicit_controlled_stepper_fsal_tag, 231
- Struct explicit_controlled_stepper_tag
  - controlled_stepper_tag, 232
  - explicit_controlled_stepper_tag, 232
- Struct explicit_error_stepper_fsal_tag
  - error_stepper_tag, 232
  - explicit_error_stepper_fsal_tag, 232
- Struct explicit_error_stepper_tag
  - error_stepper_tag, 232
  - explicit_error_stepper_tag, 232
- Struct stepper_tag
  - stepper_tag, 232
- Struct template base_tag
  - base_tag, 228
- Struct template default_rosenbrock_coefficients
  - default_rosenbrock_coefficients, 203
  - order_type, 203
  - value_type, 203
- Symplectic solvers
  - equations, 59
  - example, 59
  - f, 59
- Symplectic System
  - equations, 98
  - f, 98
- symplectic_euler
  - Class template symplectic_euler, 233

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

symplectic_nystroem_stepper_base
  Class template symplectic_euler, 233
  Class template symplectic_rkn_sb3a_m4_mclachlan, 235
  Class template symplectic_rkn_sb3a_mclachlan, 237
symplectic_rkn_sb3a_m4_mclachlan
  Class template symplectic_rkn_sb3a_m4_mclachlan, 235
symplectic_rkn_sb3a_mclachlan
  Class template symplectic_rkn_sb3a_mclachlan, 237
sys
  Large oscillator chains, 40
  Steppers, 58
system
  Parameter studies, 42
System
  f, 97

T

times_iterator
  Class template times_iterator, 140
times_time_iterator
  Class template times_time_iterator, 143
time_square_type
  Class template velocity_verlet, 239
time_type
  Class template adams_bashforth, 146
  Class template adams_bashforth_moulton, 151
  Class template adams_moulton, 155
  Class template bulirsch_stoer, 158
  Class template bulirsch_stoer_dense_out, 163
  Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170
  Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175
  Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182
  Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185
  Class template euler, 188
  Class template explicit_error_generic_rk, 191
  Class template explicit_generic_rk, 194
  Class template implicit_euler, 197
  Class template modified_midpoint, 199
  Class template modified_midpoint_dense_out, 201
  Class template rosenbrock4, 204
  Class template rosenbrock4_controller, 207
  Class template rosenbrock4_dense_output, 209
  Class template runge_kutta4, 211
  Class template runge_kutta4_classic, 214
  Class template runge_kutta_cash_karp54, 216
  Class template runge_kutta_cash_karp54_classic, 219
  Class template runge_kutta_dopri5, 222
  Class template runge_kutta_fehlberg78, 225
  Class template velocity_verlet, 239
  Custom Runge-Kutta steppers, 73
  Custom steppers, 71
  Error Stepper, 102
  Using boost::units, 28
toggle_current_acc
  Class template velocity_verlet, 239, 243
toggle_current_state
  Class template bulirsch_stoer_dense_out, 163, 168

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

    Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182, 184
    Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185, 187
    Class template rosenbrock4_dense_output, 209-210
try_step
    Class template bulirsch_stoer, 158, 160-161
    Class template bulirsch_stoer_dense_out, 163, 165
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170-172
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175-177
    Class template rosenbrock4_controller, 207-208
try_step_v1
    Class template bulirsch_stoer, 158, 162
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170, 174
    Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175, 178
type
    Generation functions, 75
    GSL Vector, 87
    std::list, 86
    Struct base_tag<controlled_stepper_tag>, 228
    Struct base_tag<dense_output_stepper_tag>, 229
    Struct base_tag<error_stepper_tag>, 229
    Struct base_tag<explicit_controlled_stepper_fsal_tag>, 229
    Struct base_tag<explicit_controlled_stepper_tag>, 230
    Struct base_tag<explicit_error_stepper_fsal_tag>, 230
    Struct base_tag<explicit_error_stepper_tag>, 230
    Struct base_tag<stepper_tag>, 230
    Using the container interface, 85

U

unsplit
    Splitter, 53
unsplit_impl
    Splitter, 53
Usage, Compilation, Headers
    links, 7
    path, 7
    pre-conditions, 7
Using arbitrary precision floating point types
    example, 32
    pre-conditions, 32
    state_type, 32
    value_type, 32
Using boost::range
    example, 93
    initialize, 93
Using boost::ref
    example, 92
Using boost::units
    acceleration_type, 28
    deriv_type, 28
    equations, 28
    example, 28
    state_type, 28
    stepper_type, 28
    time_type, 28
    velocity_type, 28
Using CUDA (or OpenMP, TBB, ...) via Thrust
    equations, 36

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- example, 36
- graphics, 36

Using iterators
- example, 13

Using matrices as state types
- equations, 30
- example, 30
- state_type, 30

Using OpenCL via VexCL
- example, 45
- integrate_const, 45
- pre-conditions, 45
- state_type, 45

Using steppers
- equations, 64
- example, 64
- initialize, 64
- pre-conditions, 66

Using the container interface
- example, 85
- is_resizeable, 85
- iterator, 85
- type, 85

V

value_type
- Class template adams_bashforth, 146
- Class template adams_bashforth_moulton, 151
- Class template adams_moulton, 155
- Class template bulirsch_stoer, 158
- Class template bulirsch_stoer_dense_out, 163
- Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_fsal_tag>, 170
- Class template controlled_runge_kutta<ErrorStepper, ErrorChecker, Resizer, explicit_error_stepper_tag>, 175
- Class template default_error_checker, 179
- Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182
- Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185
- Class template euler, 188
- Class template explicit_error_generic_rk, 191
- Class template explicit_generic_rk, 194
- Class template implicit_euler, 197
- Class template modified_midpoint, 199
- Class template modified_midpoint_dense_out, 201
- Class template rosenbrock4, 204
- Class template rosenbrock4_controller, 207
- Class template rosenbrock4_dense_output, 209
- Class template runge_kutta4, 211
- Class template runge_kutta4_classic, 214
- Class template runge_kutta_cash_karp54, 216
- Class template runge_kutta_cash_karp54_classic, 219
- Class template runge_kutta_dopri5, 222
- Class template runge_kutta_fehlberg78, 225
- Class template symplectic_euler, 233
- Class template symplectic_rkn_sb3a_m4_mclachlan, 235
- Class template symplectic_rkn_sb3a_mclachlan, 237
- Class template velocity_verlet, 239
- Custom Runge-Kutta steppers, 73
- Custom steppers, 71

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

    Define the system function, 15
    GSL Vector, 87
    Parameter studies, 42
    Phase oscillator ensemble, 37
    Struct template default_rosenbrock_coefficients, 203
    Using arbitrary precision floating point types, 32
Vector Space Algebra
    abs, 89
vector_space_norm_inf
    Point type, 90
velocity_type
    Class template velocity_verlet, 239
    Using boost::units, 28
velocity_verlet
    Class template velocity_verlet, 239

W
wrapped_acceleration_type
    Class template velocity_verlet, 239
wrapped_deriv_type
    Class template adams_bashforth, 146
    Class template adams_bashforth_moulton, 151
    Class template adams_moulton, 155
    Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182
    Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185
    Class template explicit_error_generic_rk, 191
    Class template explicit_generic_rk, 194
    Class template implicit_euler, 197
    Class template modified_midpoint, 199
    Class template modified_midpoint_dense_out, 201
    Class template rosenbrock4, 204
    Class template rosenbrock4_controller, 207
    Class template rosenbrock4_dense_output, 209
    Custom Runge-Kutta steppers, 73
wrapped_matrix_type
    Class template implicit_euler, 197
    Class template rosenbrock4, 204
wrapped_pmatrix_type
    Class template implicit_euler, 197
    Class template rosenbrock4, 204
wrapped_state_type
    Class template adams_bashforth, 146
    Class template adams_bashforth_moulton, 151
    Class template adams_moulton, 155
    Class template dense_output_runge_kutta<Stepper, explicit_controlled_stepper_fsal_tag>, 182
    Class template dense_output_runge_kutta<Stepper, stepper_tag>, 185
    Class template explicit_error_generic_rk, 191
    Class template explicit_generic_rk, 194
    Class template implicit_euler, 197
    Class template modified_midpoint, 199
    Class template modified_midpoint_dense_out, 201
    Class template rosenbrock4, 204
    Class template rosenbrock4_controller, 207
    Class template rosenbrock4_dense_output, 209
    Custom Runge-Kutta steppers, 73

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/