odeint

Solving ordinary differential equations in C++

Karsten Ahnert[1,2] and Mario Mulansky[2]

[1] Ambrosys GmbH, Potsdam
[2] Institut für Physik und Astronomie, Universität Potsdam

April 19, 2012

Outline

What is odeint?

ODE – ordinary differential equation

int – Integration

A C++ library for solving ordinary differential equations

`odeint.com`

What is an ODE?

Vielleicht Marios Folie(n) kopieren

Who uses odeint

netevo
ompl (Open Motion planning library)
…

The interface problem in C/C++

- Many frameworks exist to do numerical computations.
- Data has to be stored in containers or collections.
- GSL: `gsl_vector`, `gsl_matrix`
- NR: pointers with Fortran-style indexing
- Blitz++, MTL4, boost::ublas
- QT: `QVector`, wxWidgets: `wxArray`, MFC: `CArray`

But: All books on C++ recommend the use of the STL containers `std::vector`, `std::list`, ...

The interface problem in C/C++

- Many frameworks exist to do numerical computations.
- Data has to be stored in containers or collections.
- GSL: `gsl_vector`, `gsl_matrix`
- NR: pointers with Fortran-style indexing
- Blitz++, MTL4, boost::ublas
- QT: `QVector`, wxWidgets: `wxArray`, MFC: `CArray`

But: All books on C++ recommend the use of the STL containers `std::vector`, `std::list`, ...

Theoretical solution of the interface mess

GoF Design Pattern: Adaptor, also known as Wrapper

The interface problem in C/C++

- Many frameworks exist to do numerical computations.
- Data has to be stored in containers or collections.
- GSL: `gsl_vector`, `gsl_matrix`
- NR: pointers with Fortran-style indexing
- Blitz++, MTL4, boost::ublas
- QT: `QVector`, wxWidgets: `wxArray`, MFC: `CArray`

But: All books on C++ recommend the use of the STL containers `std::vector`, `std::list`, ...

Theoretical solution of the interface mess

GoF Design Pattern: Adaptor, also known as Wrapper

Alternative

Generic, container independent algorithms

Example

```
void CrankNicolsonEvolution::prepareVector(gsl_vector_complex* phi) {
  gsl_vector_complex* phi_temp = gsl_vector_complex_alloc(dim);
  //we need a copy of phi for this
  gsl_vector_complex_memcpy(phi_temp, phi);
  for( int i=1; i<dim-1; i++ ) {
    //phi_n = phi_n - i*dt/2 * (phi_n-1 + phi_n+1 + pot[n]*phi_n)
    gsl_vector_complex_set(phi, i, gsl_complex_add(
                gsl_vector_complex_get(phi_temp, i),
                gsl_complex_mul_imag(
                   gsl_complex_add(
                      gsl_complex_add( gsl_vector_complex_get(phi_temp, i-1),
                                    gsl_vector_complex_get(phi_temp, i+1)),
                      gsl_complex_mul_real( gsl_vector_complex_get(phi_temp, i),
                                          potential[i] )),
                   -dt/2.0)));
  }
  if( periodic ) {
    //periodic boundaries: i=0
    gsl_vector_complex_set(phi, 0, gsl_complex_add(
                gsl_vector_complex_get(phi_temp, 0),
                gsl_complex_mul_imag(
                   gsl_complex_add(
                      gsl_complex_add( gsl_vector_complex_get(phi_temp, dim-1),
                                       gsl_vector_complex_get(phi_temp, 1)),
                      gsl_complex_mul_real( gsl_vector_complex_get(phi_temp, 0),
                                          potential[0] )),
                   -dt/2.0)));
    //periodic boundaries: i=dim-1
    gsl_vector_complex_set(phi, dim-1, gsl_complex_add(
                gsl_vector_complex_get(phi_temp, dim-1),
                gsl_complex_mul_imag(
                   gsl_complex_add(
                      gsl_complex_add( gsl_vector_complex_get(phi_temp, dim-2),
                                       gsl_vector_complex_get(phi_temp, 0)),
                      gsl_complex_mul_real( gsl_vector_complex_get(phi_temp, dim-1),
                                          potential[dim-1] )),
                   -dt/2.0)));
  } else {
```

Portability of your algorithm

How to run your algorithm?

- Single machine, single CPU
- Single machine, multiple CPU's (OpenMP, threads, ...)
- Multiple machines (MPI)
- GPU (Cuda, Thrust, OpenCL)

Portability of your algorithm

How to run your algorithm?

- Single machine, single CPU
- Single machine, multiple CPU's (OpenMP, threads, ...)
- Multiple machines (MPI)
- GPU (Cuda, Thrust, OpenCL)

Which data types are used by your algorithm?

- Build-in data types – `double`, `complex<double>`
- Arbitrary precision types – GMP, MPFR
- Vectorial data types `float2d`, `float3d`

Portability of your algorithm

How to run your algorithm?

- Single machine, single CPU
- Single machine, multiple CPU's (OpenMP, threads, ...)
- Multiple machines (MPI)
- GPU (Cuda, Thrust, OpenCL)

Which data types are used by your algorithm?

- Build-in data types – `double`, `complex<double>`
- Arbitrary precision types – GMP, MPFR
- Vectorial data types `float2d`, `float3d`

Theoretical solution

GoF Design Pattern: Strategy, also known as Policy

Gamma, Holm, Johnson, Vlissides: *Design Patterns, Elements of Reuseable Object-Oriented Software*, 1998.

Numerical integration of ODEs

Find a numerical solution of an ODE an its initial value problem

$$\dot{x} = f(x,t)\ , \qquad x(t=0) = x_0$$

Example: Explicit Euler

$$x(t+\Delta t) = x(t) + \Delta t\ f(x(t),t) + \mathcal{O}(\Delta t^2)$$

General scheme of order s

$$x(t) \ \mapsto \ x(t+\Delta t) \qquad \text{, or}$$

$$x(t+\Delta t) = \mathcal{F}_t x(t) + \mathcal{O}(\Delta t^{s+1})$$

odeint

Solving ordinary differential equations in C++

Open source

- Boost license – do whatever you want do to with it

odeint

Solving ordinary differential equations in C++

Open source

- Boost license – do whatever you want do to with it

Download

- **`www.odeint.com`**

odeint

Solving ordinary differential equations in C++

Open source

- Boost license – do whatever you want do to with it

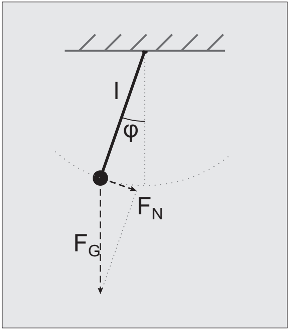
Download

- **`www.odeint.com`**

Modern C++

- Generic programming, functional programming
- Heavy use of the C++ template system
- Fast, easy-to-use and extendable.
- Container independent
- Portable

Example – Pendulum



Pendulum – Newtons law

$ma = F$

Acceleration

$a = l\ddot{\varphi}$

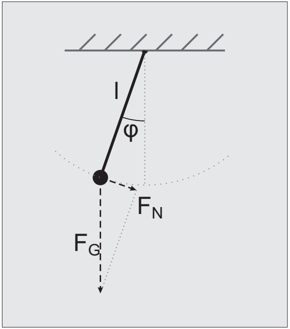
Force

$F = F_N = -mg \sin \varphi$

Result in an ode for the angle

$\ddot{\varphi} = -g/l \sin \varphi$

Example – Pendulum



$\ddot{\varphi} = -g/l \sin \varphi$

Small angle $\sin \varphi \approx \phi$

Harmonic oscillator

$\ddot{\varphi} = -g/l\varphi$

An analytic solution is known

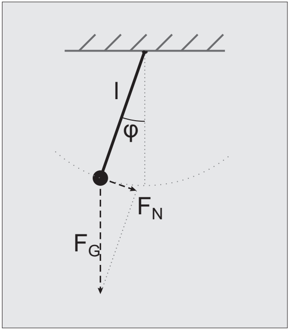
$\varphi = A \cos \omega t + B \sin \omega t$

Amplitude A and B must be determined from initial conditions:

$\varphi(t = 0) = \varphi_0$, $\dot{\varphi}(t = 0) = \dot{\varphi}_0$

$B = \varphi_0$, $A = \dot{\varphi}_0/\omega$

Example – Pendulum

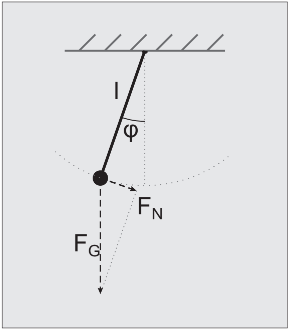


Full equation $\ddot{\varphi} = g/l \sin \varphi$ has also analytic solution Jacobi elliptic function

Lets enhance the ODE, add friction and external driving $\ddot{\varphi} = g/l \sin \varphi - \mu\dot{\varphi} + \varepsilon \sin \omega t$

No analytic solution is known. We need to solve this equation numerically.

Example – Pendulum



$\ddot{\varphi} = g/l \sin \varphi - \mu \dot{\varphi} + \varepsilon \sin \omega t$
Create a first order ODE
$x_1 = \varphi$, $x_2 = \dot{\varphi}$
$\dot{x}_1 = x_2$, $\dot{x}_2 = -g/l \sin x_1 - \mu x_2 + \varepsilon \sin \omega t$
x_1 and x_2 are the state space variables.

Let's solve the pendulum example numerically

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

namespace odeint = boost::numeric::odeint;
```

$\dot{x}_1 = x_2$, $\dot{x}_2 = -g/l \sin x_1 - \mu x_2 + \varepsilon \sin \omega t$

```
typedef std::array<double,2> state_type;
```

Let's solve the pendulum example numerically

$\dot{x}_1 = x_2$, $\dot{x}_2 = -g/l \sin x_1 - \mu x_2 + \varepsilon \sin \omega t$

```
struct pendulum
{
  double m_mu , m_omega , m_epsilon;

  pendulum( double mu , double omega , double
      epsilon )
  : m_mu( mu ) , m_omega( omega ) , m_epsilon(
      epsilon ) { }

  void operator()( const state_type &x ,
      state_type &dxdt , double t ) const
  {
    dxdt[0] = x[1];
    dxdt[1] = - sin( x[0] ) - m_mu * x[1] +
        m_epsilon * sin( m_omega * t );
  }
};
```

Let's solve the pendulum example numerically

$\varphi(0) = 1$, $\dot{\varphi}(0) = 0$

```
odeint::rk4< state_type > rk4;
pendulum p( 0.1 , 1.05 , 1.5 );

state_type x = {{ 1.0 , 0.0 }};
double t = 0.0;

const double dt = 0.01;
rk4.do_step( p , x , t , dt );
t += dt;
```

$x(0) \mapsto x(\Delta t)$

```
std::cout << t << " " << x[0] << " " << x[1] <<
    "\n";
for( size_t i=0 ; i<10 ; ++i )
{
  rk4.do_step( p , x , t , dt );
  t += dt;
  std::cout << t << " " << x[0] << " " << x[1]
      << "\n";
}
```

Simulation
$\mu = 0$, $\omega_E = 0$, $\varepsilon = 0$
$\mu = 0.1$, $\omega_E = 0$, $\varepsilon = 0$
$\mu = 0.1$, $\omega_E = 1.05$, $\varepsilon = 1.5$

Steppers

```
odeint::runge_kutta_fehlberg78< state_type >
    stepper;
```

```
odeint::runge_kutta_dopri5< state_type >
    stepper;
```

but controlled steppers are much better

Controlled steppers
insert graphic

```
auto stepper = make_controlled( 1.0e-6 , 1.0e6
    ,  odeint::runge_kutta_fehlberg78<
    state_type >() );
odeint::controlled_step_result res = stepper.
    try_step( p , x , t , dt );
```

tries to perform the step and updates *x*, *t*, and *dt*
it works because runge kutta fehlberg has error estimation

Controlled steppers

```
auto stepper = make_controlled( 1.0e-6 , 1.0e6
    ,  odeint::runge_kutta_fehlberg78<
    state_type >() );
while( t < t_end )
{
  odeint::controlled_step_result res = stepper.
      try_step( p , x , t , dt );
  while( res != odeint::success )
  {
    res = stepper.try_step( p , x , t , dt );
  }
}
```

Use integrate functions

```
integrate_adaptive( stepper , x , p , t_start ,
    t_end , dt );
integrate_adaptive( stepper , x , p , t_start ,
    t_end , dt , observer );
```

```
integrate_adaptive( stepper , p , x , t_start ,
```

```
integrate_const( stepper , p , x , t , dt ,
    observer );
```

problem with controlled stepper

```
integrate_const( make_dense_output( 1.0e-6 ,
    1.0e-6 , runge_kutta_dopri5< state_type >()
    ) , p , x , t , dt );
```

More steppers
implicit, symplectic, predictor-corrector, multistep-methods
maybe small table

small summary (kann vielleicht auch wieder weg)

- Very easy example – harmonic oscillator
- Basic features of odeint
- Different stepper
- Controlled steppers
- Dense output steppers
- integrate functions

Now, advanced features

extended systems

- Lattice systems

High-Performance-Computing
zu jedem Punkt ein Bildchen

extended systems

- Lattice systems
- Discretizations of PDEs

High-Performance-Computing
zu jedem Punkt ein Bildchen

extended systems

- Lattice systems
- Discretizations of PDEs
- ODEs on Graphs

High-Performance-Computing
zu jedem Punkt ein Bildchen

extended systems

- Lattice systems
- Discretizations of PDEs
- ODEs on Graphs
- granular systems

High-Performance-Computing
zu jedem Punkt ein Bildchen

Phase oscillator lattices

Any oscillator can be described by one variable, its phase.

(Bild aus phd Talk)

Trivial dynamics: $\dot{\varphi} = \omega\varphi$

Vielleicht zusammenfuehren mit der naechsten Folie

Phase oscillator lattices

Coupled phase oscillators

Neurosciences

Heart dynamics

Synchronization

Any weakly perturbed oscillator system

$\dot{\varphi}_k = \omega_k \varphi_k + q(\varphi_{k+1}, \varphi_k) + q(\varphi_k, \varphi_{k-1})$

Phase compacton lattices

$\dot{\varphi}_k = \cos \varphi_{k+1} - \cos \varphi_{k-1}$

state space contains N variables

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

Animation with compactons and chaos

space-time plot for visualization of compactons and chaos

Ensemble of phase oscillators

$$\dot{\varphi}_k = \omega_k + \sum_l \sin(\varphi_l - \varphi_k)$$

Synchronization, all oscillator oscillates with the same frequency

Synchronized state $\varphi_k = \omega_S t + \varphi_{0,k}$

Classical implementation

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

struct phase_ensemble
{
    state_type m_omega;
    double m_epsilon;

    phase_ensemble(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) { ... }

    void operator()(const state_type &x,
       state_type &dxdt,double t) const
    {
        ...
    }
```

Solving ODEs with CUDA using thrust

What is Thrust

Thrust is a parallel algorithms library which resembles the C++ Standard Template Library (STL). Thrust's high-level interface greatly enhances developer productivity while enabling performance portability between GPUs and multicore CPUs. Interoperability with established technologies (such as CUDA, TBB and OpenMP) facilitates integration with existing software. Develop high-performance applications rapidly with Thrust!

Solving ODEs with CUDA using thrust

Where to use it

- Large systems, discretizations of ODE, lattice systems, granular systems, etc.
- Parameter studies, integrate many ODEs in parallel with different parameters
- Initial value studies, integrate the same ODE with many different initial conditions in parallel

Lorenz system – Parameter study

$$\dot{x} = \sigma(y - x) \qquad \dot{y} = Rx - y - xz \qquad \dot{z} = -bz + xy \tag{1}$$

Standard parameters $\sigma = 10$, $R = 28$, $b = 8/3$ deterministic chaos, butterfly effect
picture of Lorenz system

Lorenz system – Parameter study

Vary R from 0 to 50, for which parameters the system is chaotic?

Lyapunov exponents, perturbations of the original system

Algebras and operations

Euler method

$x_i(t + \Delta t) = x_i(t) + \Delta t * f_i(x)$

Algebras perform the iteration over i and operation the elementary addition.

Algebras and operations enter the stepper as template parameters

```
typedef runge_kutta4<state_type,value_type,
    deriv_type,time_type,
     algebra,operations,resize_policy> stepper;
```

- `default_operations`
- `range_algebra` – Boost.Ranges
- `vector_space_algebra` – Passes the state directly to the operations
- `fusion_algebra` – For compile time sequences, like $std :: tuple < double, double >$
- `thrust_algebra` and `thrust_algebra` – for thrust

Thrust example for Lorenz system,
Implementation of the system function

More advanced features, die themen kÃ¶nnen auch auf mehreren folien zusammengefasst werden

Boost::ref

boost::range

complex state types, vielleicht auch nicht arbitrary precision types

matrices as state types

graph as state types

self expanding lattices

Old Stuff

First example – Lorenz system

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

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

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

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

void lorenz( const state_type &x , 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];
}
```

Lorenz system continued

Different steppers:

```
runge_kutta4< state_type > stepper;
```

Lorenz system continued

Different steppers:

```
runge_kutta4< state_type > stepper;
```

```
controlled_runge_kutta< runge_kutta_cash_karp54
    < state_type > > stepper;
```

Lorenz system continued

Different steppers:

```
runge_kutta4< state_type > stepper;
```

```
controlled_runge_kutta< runge_kutta_cash_karp54
    < state_type > > stepper;
```

```
dense_output_runge_kutta<
    controlled_runge_kutta<
     runge_kutta_dopri5< state_type > > >
         stepper;
```

Lorenz system continued

Different steppers:

```
runge_kutta4< state_type > stepper;
```

```
controlled_runge_kutta< runge_kutta_cash_karp54
    < state_type > > stepper;
```

```
dense_output_runge_kutta<
    controlled_runge_kutta<
     runge_kutta_dopri5< state_type > > >
         stepper;
```

```
runge_kutta_dopri5< state_type > stepper;
make_dense_output( 1.0e-6 , 1.0e-6 , stepper );
         // incomplete
```

Lorenz system continued

Different steppers:

```
runge_kutta4< state_type > stepper;
```

```
controlled_runge_kutta< runge_kutta_cash_karp54
    < state_type > > stepper;
```

```
dense_output_runge_kutta<
    controlled_runge_kutta<
     runge_kutta_dopri5< state_type > > >
        stepper;
```

```
runge_kutta_dopri5< state_type > stepper;
make_dense_output( 1.0e-6 , 1.0e-6 , stepper );
       // incomplete
```

All together:

```
int main( int argc , char **argv )
{
    state_type x = {{ 10.0 , 1.0 , 1.0 }};
```

Second example – Fermi-Pasta-Ulam lattice

$$\begin{aligned}\dot{q}_k &= p_k \\ \dot{p}_k &= -q_k^2 + \Delta q_k + \beta\{(q_{k+1} - q_k)^3 - (q_k - q_{k-1})^3\}\end{aligned}$$

$$\Delta q_k = q_{k+1} - 2q_k + q_{k-1}$$

Second example – Fermi-Pasta-Ulam lattice

$$
\begin{aligned}
\dot{q}_k &= p_k \\
\dot{p}_k &= -q_k^2 + \Delta q_k + \beta\{(q_{k+1} - q_k)^3 - (q_k - q_{k-1})^3\}
\end{aligned}
$$

$$\Delta q_k = q_{k+1} - 2q_k + q_{k-1}$$

State type consists of coordinates q and momentas p

```
typedef std::vector<double> vector_type;
vector_type q( 256 ) , p( 256 );
// initialize q,p
std::pair< state_type , state_type > state =
    std::make_pair( q , p );
```

Second example – Fermi-Pasta-Ulam lattice

$$\begin{aligned}\dot{q}_k &= p_k \\ \dot{p}_k &= -q_k^2 + \Delta q_k + \beta\{(q_{k+1} - q_k)^3 - (q_k - q_{k-1})^3\}\end{aligned}$$

$$\Delta q_k = q_{k+1} - 2q_k + q_{k-1}$$

State type consists of coordinates q and momentas p

```
typedef std::vector<double> vector_type;
vector_type q( 256 ) , p( 256 );
// initialize q,p
std::pair< state_type , state_type > state =
    std::make_pair( q , p );
```

Hamiltonian system $\Longrightarrow$ Symplectic solvers needed

```
symplectic_rkn_sb3a_mclachlan< vector_type >
    stepper;
```

Fermi-Pasta-Ulam lattice continued

Trivial first component $\dot{q}_k = p_k$

```
struct fpu {
    double m_beta;
    fpu(double beta) : m_beta(beta) { }

    void operator()(const vector_type &q,
        vector_type &dpdt) const {
        // ...
    }
};
```

Fermi-Pasta-Ulam lattice continued

Trivial first component $\dot{q}_k = p_k$

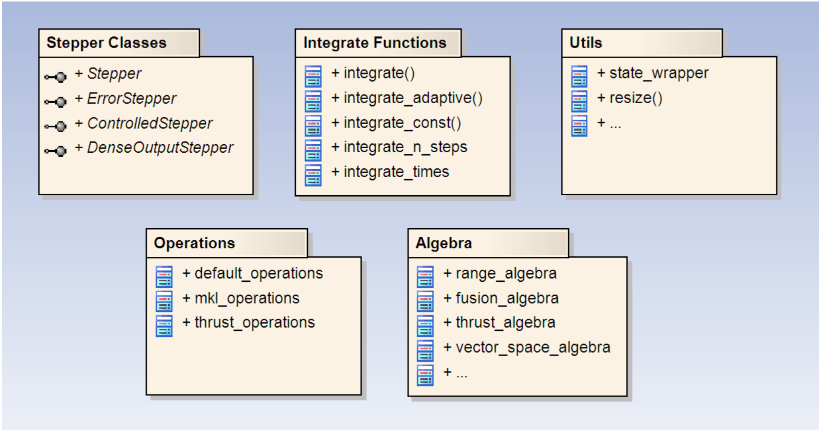
```
struct fpu {
    double m_beta;
    fpu(double beta) : m_beta(beta) { }

    void operator()(const vector_type &q,
        vector_type &dpdt) const {
         // ...
    }
};
```

All together

```
struct statistics_observer {
    void operator()( const state_type &x ,
        double t ) const {
    // write the statistics
    }
};
```

Structure of odeint



Internals – Example Euler's method

User provides

$$y_i = f_i(x(t), t)$$

odeint provides

$$x_i(t + \Delta t) = x_i(t) + \Delta t \cdot y_i$$

(In general vector operations like $z_i = a_1 x_{1,i} + a_2 x_{2,i} + \dots$)

Instantiation

```
euler<state_type, value_type, deriv_type,
    time_type, algebra, operations > stepper;
```

All elements for container independence and portability are already included in this line!

Internals – Example Euler's method

$$y_i = f_i(x(t))$$
$$x_i(t + \Delta t) = x_i(t) + \Delta t \cdot y_i$$

```
euler<state_type, value_type, deriv_type,
    time_type, algebra, operations > stepper;
```

Data types

- `state_type` – the type of x
- `value_type` – the basic numeric type, e.g. `double`
- `deriv_type` – the type of y
- `time_type` – the type of t, Δt

Internals – Example Euler's method

$$y_i = f_i(x(t))$$
$$x_i(t + \Delta t) = x_i(t) + \Delta t \cdot y_i$$

```
euler<state_type, value_type, deriv_type,
    time_type, algebra, operations > stepper;
```

Algebra policies, perform the iteration

Algebra must be a class with public methods

- `for_each1(x,op)` – Performs $op(x_i)$ for all i
- `for_each2(x1,x2,op)` – Performs $op(x1_i, x2_i)$ for all i
- ...

Internals – Example Euler's method

$$y_i = f_i(x(t))$$
$$x_i(t + \Delta t) = x_i(t) + \Delta t \cdot y_i$$

```
euler<state_type, value_type, deriv_type,
    time_type, algebra, operations > stepper;
```

Operations do the basic computation

Operations must be a class with the public classes (functors)

- `scale_sum1` – Calculates $x = a1 \cdot y1$
- `scale_sum2` – Calculates $x = a1 \cdot y1 + a2 \cdot y2$
- ...

Internals – Example Euler's method

$$y_i = f_i(x(t))$$
$$x_i(t + \Delta t) = x_i(t) + \Delta t \cdot y_i$$

```
euler<state_type, value_type, deriv_type,
    time_type, algebra, operations > stepper;
```

All together

```
m_algebra.for_each3(xnew ,xold, y ,
   operations_type::scale_sum2<value_type,time_type>(1.0,dt));
```

Stepper concepts

Concepts

"... In generic programming, a concept is a description of supported operations on a type..."

Stepper concepts

Concepts

"... In generic programming, a concept is a description of supported operations on a type..."

odeint provides

- Stepper concept
  `stepper.do_step(sys,x,t,dt);`
- ErrorStepper concept
  `stepper.do_step(sys,x,t,dt,xerr);`
- ControlledStepper concept
  `stepper.try_step(sys,x,t,dt);`
- DenseOutputStepper concept
  `stepper.do_step(sys);`
  `stepper.calc_state(t,x);`

Supported methods

Method	Class name	Concept
Euler	`euler`	SD
Runge-Kutta 4	`runge_kutta4`	S
Runge-Kutta Cash-Karp	`runge_kutta_cash_karp54`	SE
Runge-Kutta Fehlberg	`runge_kutta_runge_fehlberg78`	SE
Runge-Kutta Dormand-Prince	`runge_kutta_dopri5`	SED
Runge-Kutta controller	`controlled_runge_kutta`	C
Runge-Kutta dense output	`dense_output_runge_kutta`	D
Symplectic Euler	`symplectic_euler`	S
Symplectic RKN	`symplectic_rkn_sb3a_mclachlan`	S
Rosenbrock 4	`rosenbrock4`	ECD
Implicit Euler	`implicit_euler`	S
Adams-Bashforth-Moulton	`adams_bashforth_moulton`	S
Bulirsch-Stoer	`bulirsch_stoer`	CD

S – fulfills stepper concept
E – fulfills error stepper concept
C – fulfills controlled stepper concept
D – fulfills dense output stepper concept

Integrate functions

- `integrate_const`
- `integrate_adaptive`
- `integrate_times`
- `integrate_n_steps`

Perform many steps, use all features of the underlying method

Integrate functions

- `integrate_const`
- `integrate_adaptive`
- `integrate_times`
- `integrate_n_steps`

Perform many steps, use all features of the underlying method

An additional observer can be called

```
integrate_const(stepper, sys, x, t_start,
t_end, dt, obs);
```

More internals

- Header-only, no linking $\rightarrow$ powerful compiler optimization
- Memory allocation is managed internally
- No virtual inheritance, no virtual functions are called
- Different container types are supported, for example
  - STL containers (`vector`, `list`, `map`, `tr1::array`)
  - MTL4 matrix types, blitz++ arrays, Boost.Ublas matrix types
  - `thrust::device_vector`
  - Fancy types, like Boost.Units
  - ANY type you like
- Explicit Runge-Kutta-steppers are implemented with a new template-metaprogramming method
- Different operations and algebras are supported
  - MKL
  - Thrust
  - gsl

ODEs on GPUs

Graphical processing units (GPUs) are able to perform up to 10^6 operations at once in parallel

Frameworks

- CUDA from NVIDIA
- OpenCL
- Thrust a STL-like library for CUDA and OpenMP

Applications:

- Parameter studies
- Large systems, like ensembles or one- or two dimensional lattices
- Discretizations of PDEs

odeint supports CUDA, through Thrust

Example: Parameter study of the Lorenz system

```
typedef thrust::device_vector<double>
    state_type;
typedef runge_kutta4<state_type ,value_type ,
    state_type ,value_type ,thrust_algebra ,
    thrust_operations > stepper_type;

struct lorenz_system {

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

    void operator()(  const state_type &x ,
        state_type &dxdt , double t ){
        // ..
    }

    size_t m_N;
    const state_type &m_beta;
};
```

Conclusion

- odeint provides a fast, flexible and easy-to-use C++ library for numerical integration of ODEs.
- Its container independence is a large advantage over existing libraries.
- Portable
- Generic programming is the main programming technique.

Outlook

- Submission to the boost libraries
- Dynamical system classes for easy implementation of interacting dynamical systems
- More methods: implicit methods and multistep methods.
- Implementation of the Taylor series method

```
taylor_fixed_order< 25 , 3 > taylor_type
    stepper;

stepper.do_step(
    fusion::make_vector
    (
        sigma * ( arg2 - arg1 ) ,
        R * arg1 - arg2 - arg1 * arg3 ,
        arg1 * arg2 - b * arg3
    ) , x , t , dt );
```

Resources

Download and documentation

`odeint.com`

An article about the used techniques exists at

`http://www.codeproject.com/KB/recipes/odeint-v2.aspx`

Development

`https://github.com/headmyshoulder/odeint-v2`

Contributions and feedback

are highly welcome