Boost.odeint

Solving ordinary differential equations in C++

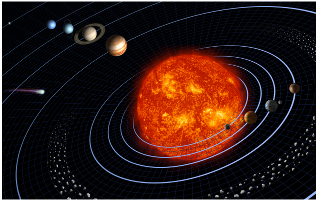
Karsten Ahnert[1,2] and Mario Mulansky[2]

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

February 2, 2013

What is an ODE? – Examples

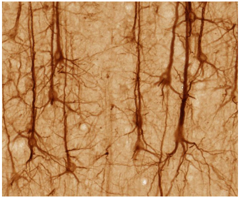
Newtons equations



Reaction and relaxation equations (i.e. blood alcohol content, chemical reaction rates)

Granular systems

Interacting neurons



- Many examples in physics, biology, chemistry, social sciences
- Fundamental in mathematical modelling

What is an ODE?

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = f\big(x(t), t\big) \qquad \text{short form} \qquad \dot{x} = f(x, t)$$

- $x(t)$ – wanted function (trajectorie)
- t – indenpendent variable (time)
- $f(x, t)$ – defines the ODE, r.h.s

Initial Value Problem (IVP):

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

Numerical integration of ODEs

Find a numerical solution of an ODE and its IVP

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

Example: Explicit Euler

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

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
- Boost library – has just been released with v1.53

Download

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

Modern C++

- Paradigms: Generic, Template-Meta and Functional Programming
- Fast, easy-to-use and extendable.
- Container independent
- Portable

Motivation

We want to solve ODEs $\dot{x} = f(x, t)$ with:

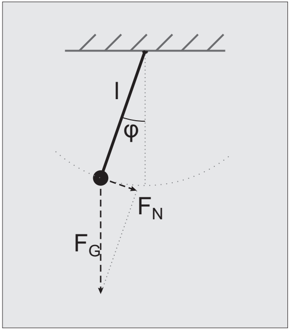
- using `double`, `std::vector`, `std::array`, … as state types.
- with complex numbers,
- on one, two, three-dimensional lattices, and or on graphs.
- on graphic cards.
- with arbitrary precision types.

Existing libraries support only one state type!

Container independent and **portable** algorithms are needed!

Example – Pendulum

Pendulum with friction and driving: no analytic solution



$$\ddot{\varphi} = -\omega_0^2 \sin\varphi - \mu\dot{\varphi} + \varepsilon \sin\omega_E t$$

Create a first order ODE

$$x_1 = \varphi \;, \quad x_2 = \dot{\varphi}$$

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -\omega_0 \sin x_1 - \mu x_2 + \varepsilon \sin\omega_E t \end{aligned}$$

x_1 and x_2 are the state space variables

Let’s solve the pendulum example numerically

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

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

$$\dot{x}_1 = x_2 \;, \quad \dot{x}_2 = -\omega_0 \sin x_1 - \mu x_2 + \varepsilon \sin \omega_E t$$

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

Let’s solve the pendulum example numerically

$\dot{x}_1 = x_2, \dot{x}_2 = -\omega_0^2 \sin x_1 - \mu x_2 + \varepsilon \sin \omega_E t$ $\qquad \omega_0^2 = 1$

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

  pendulum(double mu,double omega,double eps)
  : m_mu(mu),m_omega(omega),m_eps(eps) { }

  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_eps * sin(m_omega*t);
  }
};
```

Let’s solve the pendulum example numerically

$$\varphi(0) = x_1(0) = 1 \; , \quad \dot{\varphi}(0) = x_2(0) = 0$$

```
odeint::runge_kutta4< 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;
```

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

Let’s solve the pendulum example numerically

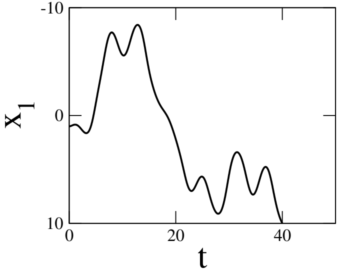
```
std::cout<<t<<" "<< x[0]<<" "<<x[1]<<"\n";
for( size_t i=0 ; i<10 ; ++i )
{
  rk4.do_step( p , x , t , dt );
  t += dt;
  std::cout<<t<<" "<< x[0]<<" "<<x[1]<<"\n";
}
```

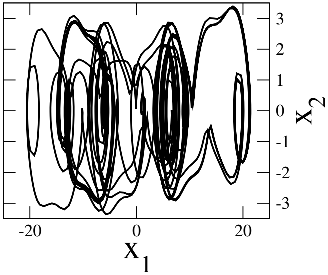
$$x(0) \mapsto x(\Delta t) \mapsto x(2\Delta t) \mapsto x(3\Delta t) \mapsto \dots$$

Let’s solve the pendulum example numerically

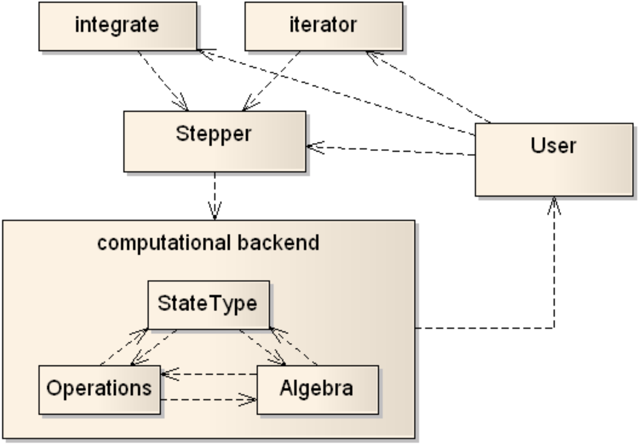
```
std::cout<<t<<" "<< x[0]<<" "<<x[1]<<"\n";
for( size_t i=0 ; i<10 ; ++i )
{
  rk4.do_step( p , x , t , dt );
  t += dt;
  std::cout<<t<<" "<< x[0]<<" "<<x[1]<<"\n";
}
```

$$x(0) \mapsto x(\Delta t) \mapsto x(2\Delta t) \mapsto x(3\Delta t) \mapsto \dots$$





Structure of odeint



Independent Algorithms

What?
Container- and computation-independent implementation of the numerical algorithms.

Why?
High flexibility and applicability, odeint can be used for virtually any formulation of an ODE.

How?
Detach the algorithm from memory management and computation details and make each part interchangeable.

Type Declarations

Tell odeint which types your are working with:

```
/* define your types */
typedef vector<double> state_type;
typedef vector<double> deriv_type;
typedef double value_type;
typedef double time_type;

/* define your stepper algorithm */
typedef runge_kutta4< state_type , value_type ,
    deriv_type , time_type > stepper_type;
```

Reasonable standard values for the template parameters allows for:

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

Vector Computations

$$\vec{x}_1 = \vec{x}_0 + b_1 \cdot \Delta t \cdot \vec{F}_1 + \cdots + b_s \cdot \Delta t \cdot \vec{F}_s$$

Split into two parts:

1. Algebra: responsible for iteration over vector elements
2. Operations: does the mathematical computation on the elements

Similar to `std::for_each`

```
Algebra algebra;

algebra.for_each3( x1 , x0 , F1 ,
    Operations::scale_sum2( 1.0, b1*dt );
```

Vector Computations

$$\vec{x}_1 = \vec{x}_0 + b_1 \cdot \Delta t \cdot \vec{F}_1 + \cdots + b_s \cdot \Delta t \cdot \vec{F}_s$$

Split into two parts:

1. Algebra: responsible for iteration over vector elements
2. Operations: does the mathematical computation on the elements

Similar to `std::for_each`

```
Algebra algebra;

algebra.for_each3( x1 , x0 , F1 ,
    Operations::scale_sum2( 1.0, b1*dt );
```

The types `Algebra` and `Operations` are template parameters of the steppers, hence exchangeable.

For example `vector< double >`:

```
typedef vector< double > state_type;
typedef vector< double > deriv_type;
typedef double value_type;
typedef double time_type;

typedef runge_kutta4< state_type , value_type ,
                      deriv_type , time_type ,
                      range_algebra ,
                      default_operations
                    > stepper_type
```

As these are also the default values, this can be shortened:

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

Other Algebras

Additional computation backends included in odeint:

array_algebra: for `std::array`, faster than range_algebra for some compilers.

vector_space_algebra: for `state_types` that have operators `+, *` defined.

fusion_algebra: works with compile-time sequences like `fusion::vector` of Boost.Units

thrust_algebra & thrust_operations: Use thrust library to perform computation on CUDA graphic cards

mkl_operations: Use Intel’s Math Kernel Library

See tutorial and documentation on `www.odeint.com` for more.

Conclusion

odeint is a modern C++ library for solving ODEs that is

- easy-to-use
- highly-flexible
  - data types (topology of the ODE, complex numbers, precision, . . .)
  - computations (CPU, CUDA, OpenMP, ...)
- fast

Used by:

NetEvo – Simulation dynamical networks
OMPL – Open Motion Planning Library
icicle – cloud/precipitation model
Score – Smooth Particle Hydrodynamics Simulation (com.)
VLE – Virtual Environment Laboratory (planned to use odeint)

Several research groups

Roadmap

Near future:

- Implicit steppers
- Multiprozessor backends (OpenMP, MPI, HPX)

Further plans:

- Dormand-Prince 853 steppers
- More algebras: cublas, TBB, Boost SIMD library

Perspective:

- C++11 version
- sdeint – methods for stochastic differential equations
- ddeint – methods for delay differential equations