odeint

solving ODEs in C++

Karsten Ahnert and Mario Mulansky

ambrosys

Universität Potsdam

General

- Boost library (since Boost v1.53)
- Fully Open Source (Boost license)
- Modern and generic C++ code
- Highly flexible
- Fast

Includes

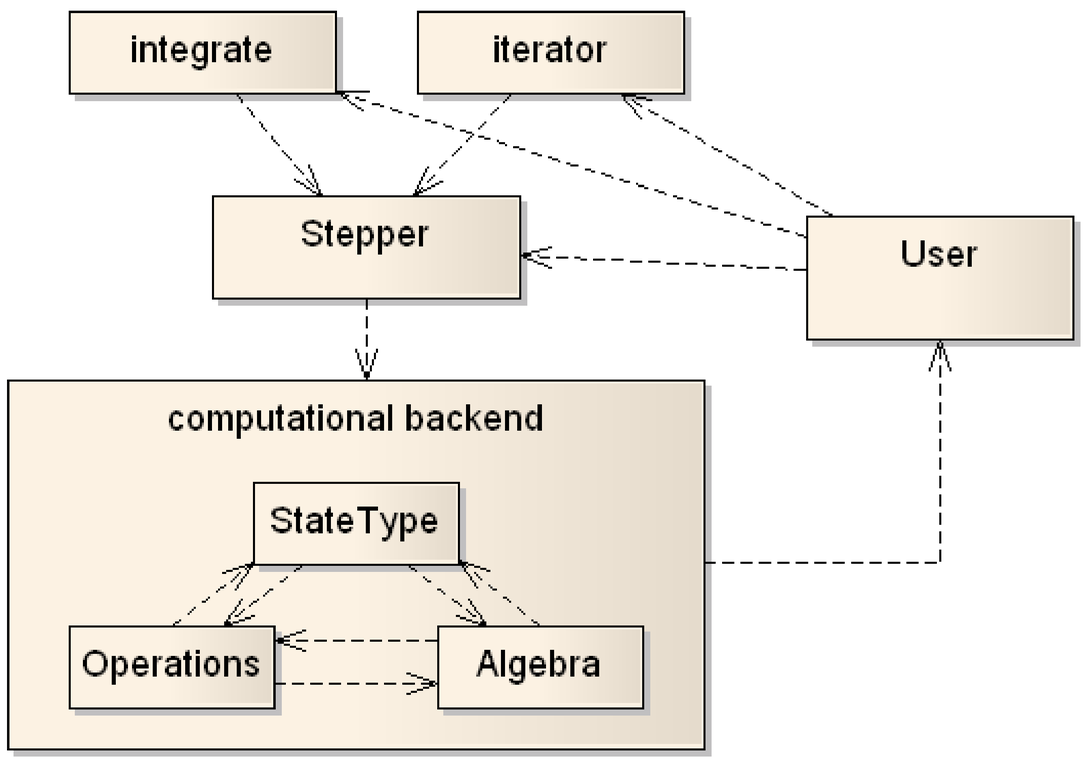
- Runge-Kutta schemes:
  - Runge-Kutta4
  - Runge-Kutta-Dopri5
  - Runge-Kutta-Cash-Karp
  - Runge-Kutta78
- Step-Size Control
- Dense Output
- Multistep Methods (Adams-Bashforth)
- Bulirsch-Stoer
- Implicit Rosenbrock scheme

- High-level integrate routines
- Iterator abstraction

Lorenz Example – 30 lines of code

```
#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];
}

void write_lorenz( const state_type &x , const double t )
{
    cout << t << '\t' << x[0] << '\t' << x[1] << '\t' << x[2] << endl;
}

int main(int argc, char **argv)
{
    state_type x = { 10.0 , 1.0 , 1.0 }; // initial conditions
    integrate( lorenz , x , 0.0 , 25.0 , 0.1 , write_lorenz );
}
```

Structure



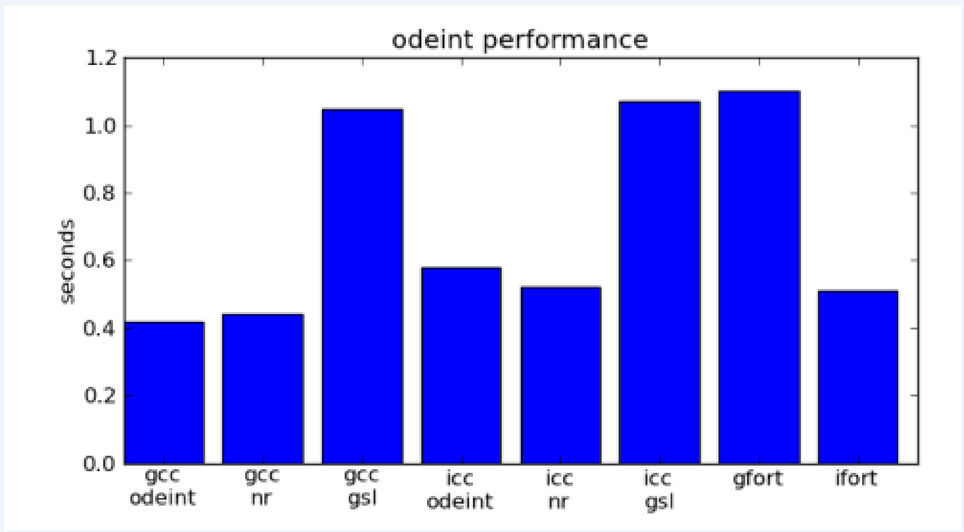
Independent Algorithms

- Separate algorithm from computational details
- Container-independent implementation
- Memory management detached from algorithms
- Each part interchangeable by the user

⟶ Incredible Flexibility:

- Supports any container as state type
- Natively supports complex numbers
- Works with many linear algebra libraries: MTL, uBlas, eigen
- **Runs on GPUs** (via Cuda/Thrust or OpenCL/vexCL)
- Can be used with multi-precision types (e.g. Boost.Multiprecision)
- Easily extendable to run with your own state type, e.g. graphs or complex networks
- Parallelized backends available soon (OpenMP, MPI, HPX)

Performance – Lorenz System, RK4



Details

- Modularization by Generic Programming
- No virtual function calls
- Metaprogramming to generate Runge-Kutta schemes
- Extensive unit tests
- Reviewed by top C++ programmers (Boost community)

Users

- **NetEvo** (Dynamical Networks)
- **OMPL** (Motion Planning)
- **icicle** (Cloud Modeling)
- **Score** (commercial SPH)
- **VLE** (Virtual Laboratory)
- Several research groups