C++ Playground for Numerical Integration Method Developers

S.G. Orlov

Computer technologies in engineering dept.
Peter the Great St. Petersburg Polytechnic University

Russian Supercomputing Days
Sept. 25-26 2017, Moscow, Russia

Outline

- ODEs with additional discrete state
  - Considering initial value problem (IVP)
- Why create yet another framework?
- The `ode_num_int` framework
  - Common infrastructure
  - Linear algebra
  - Nonlinear algebraic Newton-type solver
  - ODE Solvers
  - Code example
  - Problems & future work
- Conclusions

ODEs with discrete state

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}, \phi), \quad \mathbf{x}\big|_{t=t_0} = \mathbf{x}_0, \quad \phi\big|_{t=t_0} = \phi_0$$

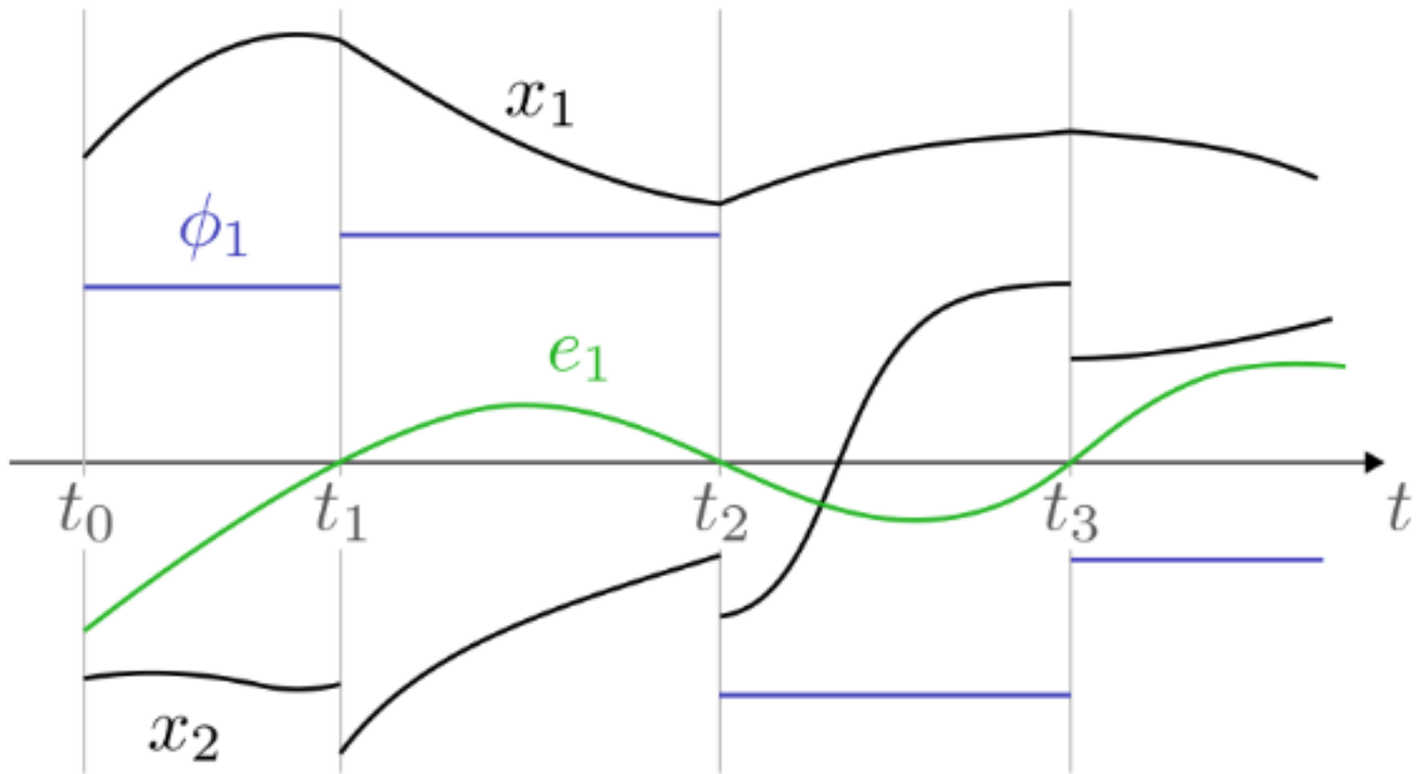
- $\mathbf{x} = [x_1, \ldots, x_n]^T$ — state vector
- t — time
- $\mathbf{f}$ — ODE right hand side
- $\phi = [\phi_1, \ldots, \phi_m]^T$ — discrete state variables
- $\mathbf{x}$ and ϕ may change at t_s, when an event occurs:

$$t_s : \quad e_k(t_s, \mathbf{x}, \phi) = 0, \quad k = 1, \ldots, n_e, \quad s = 1, 2, \ldots$$

- $e_k(t, \mathbf{x}, \phi)$ — event indicator functions

ODEs with discrete state

An example of evolution of system with discrete state



more: Functional Mock-up Interface 2.0

svn.modelica.org/fmi/branches/public/specifications/v2.0/
FMI_for_ModelExchange_and_CoSimulation_v2.0.pdf (p. 69)

Why another framework?

- What we want
  - Construct ODE IVP solvers from smallest building blocks
  - C++
  - Reasonably fast code
  - Convenient linear algebra interface
    - Avoid copying when possible
    - Use a parallel LA library when necessary
  - Easily extensible system (high level of code reuse)
  - Run test cases
    - Configure solver at run time
    - Measure timings & monitor whatever we need
    - Flexible output control
    - ODE system sizes up to ~10000

Why another framework?

- What is available
  - Environments for numerical computing:
    `MATLAB`, `Scilab`, `GNU Octave`, `Euler`, etc. (too slow)
  - ODE IVP codes
    - FORTRAN & C: `PETSc`, `Netlib ode`, `NAG D02`, `HSL`, `CMLIB`, `VODE_F90`, `SUNDIALS`, `Codes by Hairer, Nørsett, Wanner`, `DUMKA3`, etc.
    - C++: `Boost.Numeric.Odeint`
  - Linear algebra codes
    - FORTRAN & C: `LAPACK`, `SuiteSparse`, etc.
    - C++: `Armadillo`, `Eigen`, `Blitz++`, `MTL4`, `IT++`, `TNT`, etc.

Why another framework?

- Can't we just use above ODE IVP codes?
  - `SUNDIALS`
    - Implements a few solvers for ODE, DAE, ...
    - Supports events
  - `Boost.Numeric.Odeint`
    - Many ODE solvers
    - No event support
  - All
    - Too large building blocks
    - Difficult to implement new solver
    - Few means for monitoring
    - Designed for solver users, not solver developers

Why another framework?

- There are basic ideas in software engineering
  - divide & conquer, separation of concerns, single responsibility principle
  - OOP: interfaces, encapsulation, polymorphism
  - design patterns
- Existing numerical codes often
  - are not OO
  - disregard divide & conquer
  - employ awkward design patterns
  - are not in C++
  - that is sad

Why another framework?

- Attempts to provide C++ codes & wrappers often
  - do not use the power of C++ and are inefficient
    - e.g. `IT++`
  - are just outdated & unsupported
    - e.g. `LAPACK++` (2012), `Blitz++` (2011), `TNT` (2004)
  - C++ codes worse than others - that is desperate
- Still there are great things!
  - e.g., `Armadillo`, `Eigen`, `Boost.Numeric.Odeint`
- Homemade vectors & matrices everywhere!

New framework: `ode_num_int`

Common infrastructure

- `observers`
  - to provide basic means for monitoring everything
  - similar to `Boost.Signals2` but simpler & faster
- `property holder`
  - to define uniform rules for component / parameter aggregation
  - class storing a single private field of specified type
  - has public getter & setter & notifier
- `factory`
  - to dynamically create instances
- `optional parameters`
  - to organize trees of values of any type
  - interop. with factories — easy to I/O objects
- `timing utilities`
  - to measure CPU time spent in a block of code

Linear algebra

- Vectors & sparse matrices design
  - storage & element access rules defined by template parameter
  - helps avoid copying by providing
    - proxies for subvector, submatrix, transposed or scaled matrix
      - views in other software are similar
      - but our proxies are vectors & matrices too
        $\Rightarrow$ simpler design
    - move constructors
  - iterators for using `std` algorithms
  - matrices with flexible sparsity pattern (slower)
  - matrices with fixed sparsity pattern (faster)
- Sparse LU

LA example

```
// note: SparseMatrix<T> = SparseMatrixTemplate< SparseMatrixData< T > >;
SparseMatrix<double> a(5, 5);      //        2 0 0 0 0
a.addScaledIdentity( 2 );          //        1 2 0 0 0
a.block(1,0, 4,4).addIdentity();   // a  <-  0 1 2 0 0
cout << a << endl;                 //        0 0 1 2 0
                                   //        0 0 0 1 2

a *= a.transposed();               //        5 2 0 0 0
a.at(0,0) += 1;                    //        2 5 2 0 0
cout << a << endl;                 // a  <-  0 2 5 2 0
                                   //        0 0 2 5 2
                                   //        0 0 0 2 5

// note: Vector<T> = VectorTemplate< VectorData< T > >;
Vector<double> b(5);               //
fill(b.begin(), b.end(), 1);       // b  <- [1 1 1 1 1]'
cout << b << endl;                 //

// Solve a*x = b
auto x = b;                        // x  <-  [0.169231,  0.0769231, 0.138462,
LUFactorizer<double>(a).solve(x); //         0.0769231, 0.169231]'
cout << x << endl;

// Check residual
cout << (a*x - b).euclideanNorm() << endl;    // 1.11022e-016
```

Nonlinear algebraic solver

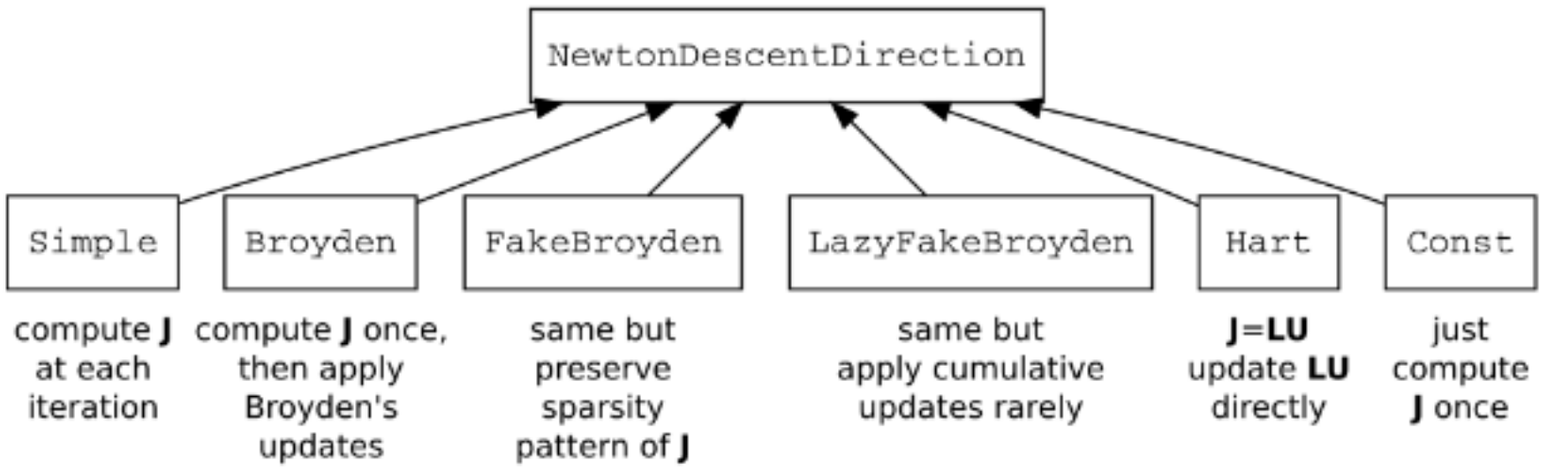
$$\mathbf{f}(\mathbf{x}) = 0, \quad \mathbf{x}_{n+1} = \mathbf{x}_n + \alpha_n \mathbf{d}_n, \quad \mathbf{A}\mathbf{d}_n = -\mathbf{f}(\mathbf{x}_n), \quad \mathbf{A} \approx \frac{D\mathbf{f}}{D\mathbf{x}}\Big|_{\mathbf{x}=\mathbf{x}_n}$$

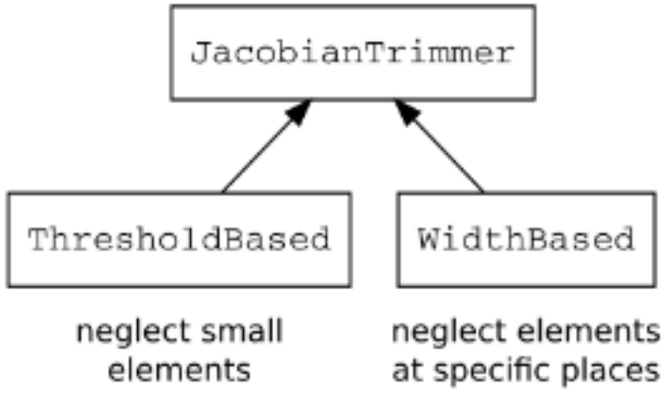
- To build an efficient Newton-type solver one may need to choose
  - appropriate vector norm for residual
  - stopping criterion
  - Jacobian evaluation & update algorithms
  - line search algorithm
  - regularization strategy
  - iteration algorithm
- `ode_num_int` provides abstractions for minimal building blocks
  - developer can concentrate on one separate thing
  - solver is easily built from separate components
  - one or several implementations for each component

Abstractions for Newton solver

- `VectorMapping` — compute $\mathbf{f}(\mathbf{x})$ or $\mathbf{f}(\mathbf{x}, \gamma)$
- `ErrorEstimator` — compute residual norm, judge about convergence
- `NewtonDescentDirection` — compute descent direction $\mathbf{d}_n$
  - holds `JacobianProvider` — compute sparse Jacobian
    - holds `JacobianTrimmer` (optional) — simplify sparsity pattern
- `NewtonLinearSearch` — choose a point along descent dir. — compute α_n
- `NewtonIterationPerformer` — compute $\mathbf{x}_{n+1}$ and return status
  - holds all above components
- `NewtonRegularizationStrategy` — manage regularization parameter γ: $\mathbf{f}(\mathbf{x}) = \mathbf{g}(\mathbf{x}, \gamma)|_{\gamma=0}$
  - holds `VectorMappingRegularizer` — compute $\mathbf{g}(\mathbf{x}, \gamma)$
- `NewtonSolverInterface` — solve nonlinear equation, report status
  - holds `NewtonIterationPerformer` and `NewtonRegularizationStrategy`

NewtonDescentDirection
implementations matter



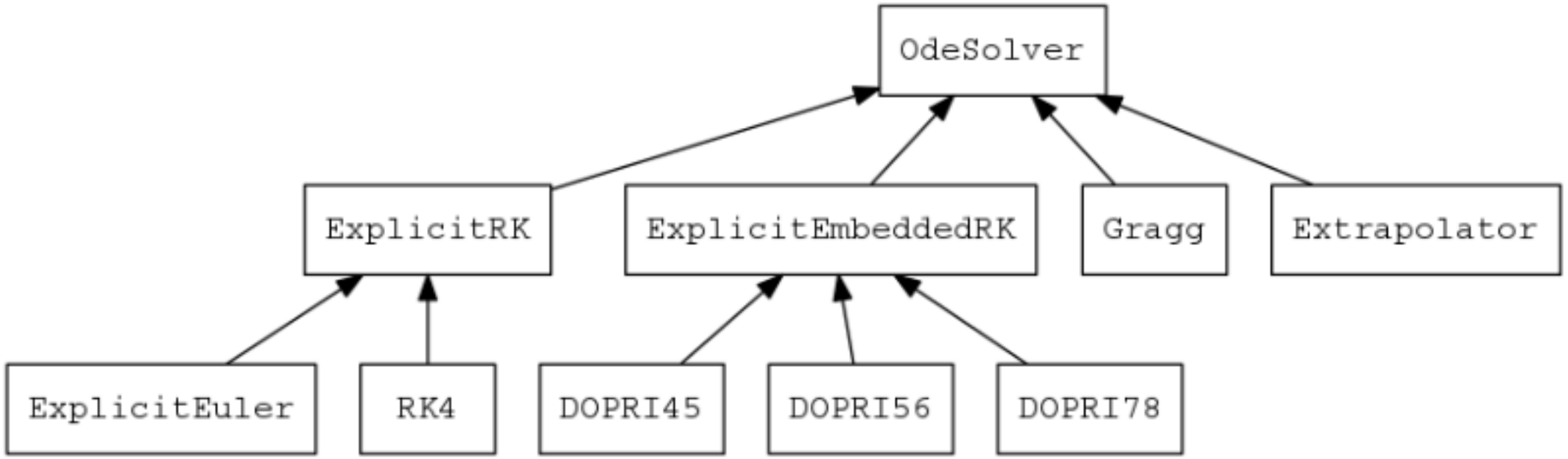


100x difference in performance is well possible (in terms of number of $\mathbf{f}(\mathbf{x})$ evaluations, **LU** decompositions, **LU** solves, and in terms of CPU time)

ODE solvers: abstractions

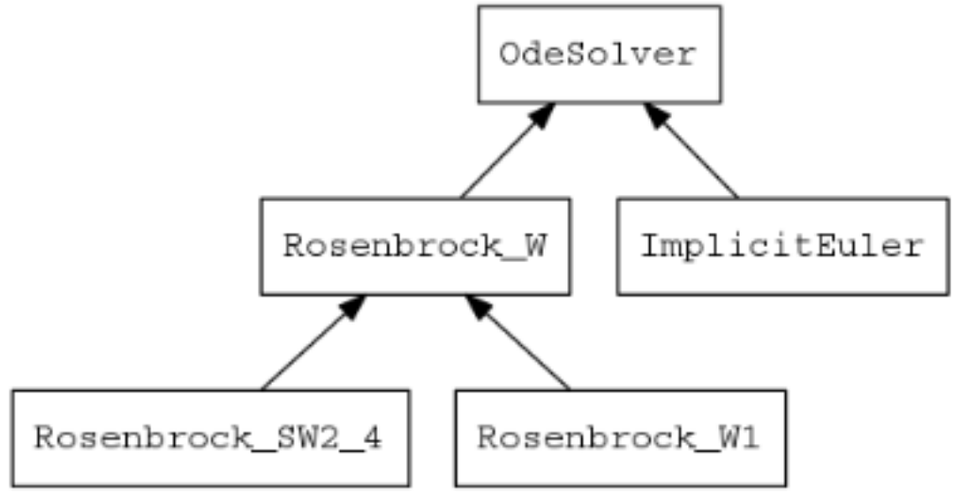
- `OdeRhs` — ODE system definition
  - compute $\mathbf{f}(t, \mathbf{x}, \varphi)$ or e.g. $\mathbf{f}(t, \mathbf{x}, \dot{\mathbf{x}}, \mathbf{z}, \varphi, \gamma)$
    - support for 2nd order ODE, dependency on time & params
  - compute event indicators $e_k(t, \mathbf{x}, \phi)$
  - implement phase state changes
- `OdeSolver` — solve ODE system, maybe with events
  - holds `OdeRhs`
  - optionally holds
    - `OdeSolverErrorNormCalculator` — compute vector norm
    - `OdeStepSizeController` — conrol step size
      for embedded time steppers
    - `OdeEventController` — detect events, truncate steps, change ϕ

Explicit ODE solvers



- Common schemes (explicit Euler, RK4, DOPRI, Gragg)
  - Others easily obtained by providing Butcher tableau
- Richardson extrapolator
  - uses any reference solver & step sequence
  - e.g., GBS method is easily constructed w/o coding

Implicit ODE solvers

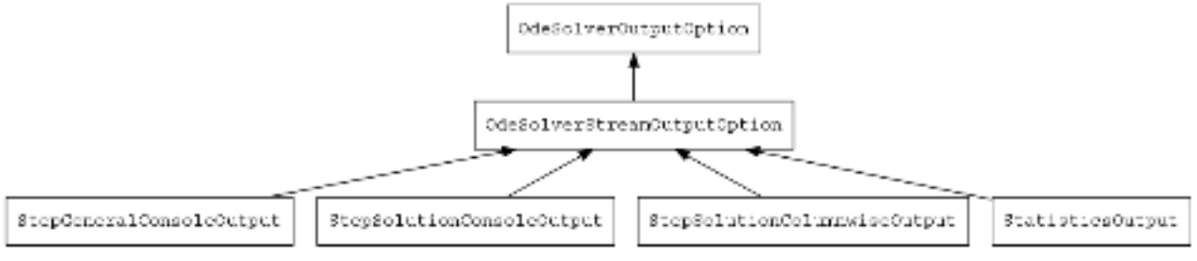


- Linearly implicit
  - Rosenbrock SW2-4 (embedded) by `Steihaug & Wolfbrandt`
  - Rosenbrock W1:

    $$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{k}, \quad \mathbf{W}\mathbf{k} = \mathbf{f}(t_n, \mathbf{x}_n), \quad \mathbf{W} = \mathbf{I} - hd\mathbf{A}, \quad \mathbf{A} \approx \frac{D\mathbf{f}}{D\mathbf{x}}$$
- Completely explicit
  - Modified Euler
    - $\mathbf{x}_{n+1} = \mathbf{x}_n + h[(1-\alpha)\mathbf{f}(t_n, \mathbf{x}_n) + \alpha\mathbf{f}(t_n + h, \mathbf{x}_{n+1})]$
    - trapezoidal rule at $\alpha = 0.5$

ODE solver helpers

- `OdeSolverOutputOption` — interface for objects collecting solver output
  - new output options can easily be created



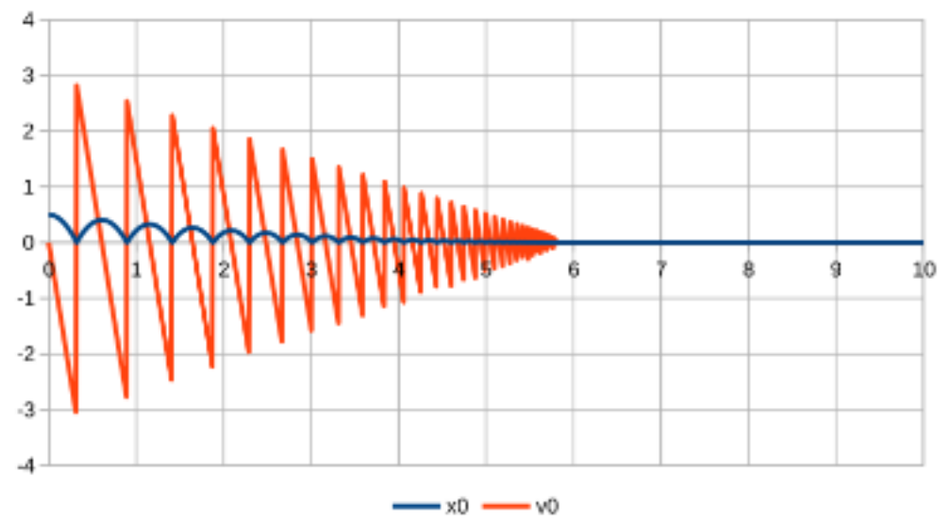
- `OdeSolverConfiguration`
  - holds parameter set describing
    - ODE specification
    - ODE solver with all nested components and parameters
    - output specification
- `solveOde` — solve initial value problem

Code example

```
#include "ode_num_int/OdeSolverConfiguration.h"
#include "reg.h"
#include <iostream>

int main() {
    using namespace std;
    using namespace ctm;
    using namespace math;
    typedef VectorData<double> VD;
    typedef VectorTemplate<VD> V;

    registerTypes();
    try {
        OdeSolverConfiguration<VD> cfg;
        cfg.setValue("rhs", "bouncing_ball");
        cfg.setValue("rhs.stick_speed", 1e-5);
        cfg.setValue("output_con", "con_solution");
        cfg.setValue("time", 10);

        V x0( 2 );
        x0[0] = 0.5;
        auto sc = cfg.apply( 0, x0 );
        solveOde( &cfg, &sc );
        return 0;
    }
    catch( const std::exception& e ) {
        cerr << "ERROR: " << e.what() << endl;
        return 1;
    }
}
```

output:

```
time      x0      v0
0 0.5     0
0.01      0.49951 -0.098
0.02      0.49804 -0.196
0.03      0.49559 -0.294
0.04      0.49216 -0.392
0.05      0.48775 -0.49
0.06      0.48236 -0.588
0.07      0.47599 -0.686
...
```



Code example

```
template< class VD >
class BouncingBall :
    public OdeRhs< VD >,
    public FactoryMixin< BouncingBall<VD>, OdeRhs<VD> >
    {
    public:
        typedef VectorTemplate< VD > V;
        typedef typename V::value_type real_type;
        typedef OptionalParameters::Parameters Parameters;

        enum FrictionType { Atan, Tabular, Linear };

        explicit BouncingBall() :
            m_g( 9.8 ),
            m_recoveryFactor( 0.9 ),
            m_stickSpeed( 1e-5 ),
            m_sticking( false )
            {}

        unsigned int secondOrderVarCount() const {
            return 1;
            }

        unsigned int firstOrderVarCount() const {
            return 0;
            }

        unsigned int zeroFuncCount() const {
            return 1;
            }

        virtual std::vector<unsigned int> zeroFuncFlags() const {
```

Problems & future work

- Dev. docs still missing, sorry for that
- Old school runtime polymprphism
  - needed for configuring solvers at run time
  - doesn't play well with templates
    - extra data copying may happen
    - workarounds are possible
- Build ecosystem needs to be set up to involve more people
  - unit tests
  - continuous integration
- Would be nice to have
  - embedded scripting environment (e.g. JavaScript)
  - common tools (e.g. stability region calculator)
  - faster vectors & matrices
    - expression templates, `BLAS`, `SparseSuite`, better move semantics

Conclusions

- New framework `ode_num_int`
  - open source, GNU GPL 3.0 license
    - `github.com/deadmorous/ode_num_int`
  - written in C++11
  - for ODE numerical integration method developers
  - smallest building blocks
    - easy to extend
    - high level of code reuse
  - good (suboptimal) performance
  - successfully used for real life application
  - There are things to do

Thank you

Questions?