Apache Commons-math

Scope of the library

- A general purpose math library
- Fast … but not dedicated to speed
- Self-contained, i.e. no dependencies
- Multiple implementations for many algorithms
  - favor strategy pattern
- 100% Java

History

- Started in 2003 when Commons was part of Jakarta
  - patches submitted for inclusion in Commons Lang went outside Lang scope
- Started with simple numerical analysis and statistics components
- Version 1.0 released in December 2004
- Latest version is 2.1, released in March 2010
- Upcoming 2.2 and 3.0

Features (1/2)

- Raw types
  - Complex numbers, fractions, arbitrary precision
  - polynomials
  - General field interface
- Basic computation
  - Interpolation, univariate/multivariate functions, transforms
  - Quadrature, 3D geometry, special functions
  - random numbers, distributions
- Solvers
  - Root finders, Ordinary Differential Equations

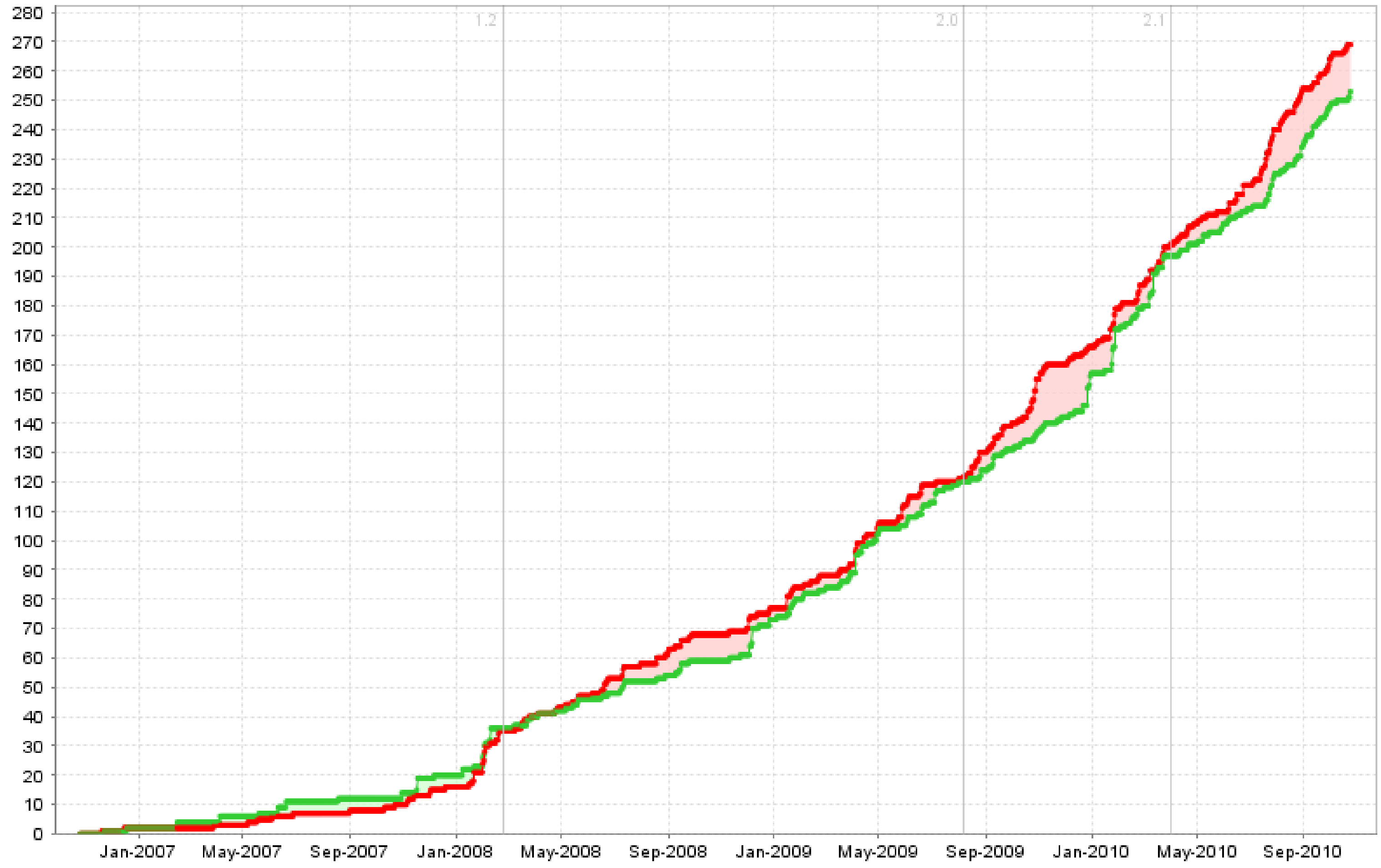
Features (2/2)

- Optimization
  - Optimizers, curve fitting, genetic algorithms
- Linear algebra
  - LU, QR, SVD decompositions
  - algebraic operations, vectors
- Statistics
  - Univariate statistics
  - multiple regression
  - correlation

A live project

- Mailing lists:
  - 2 or 3 [math] threads per month on user list
  - 2 or 3 [math] message per day on dev list
- 5+ Commons committers involved
  - growing list of contributors
- About one release each year
- Diverse user base
  - Astronomy, space flight dynamics, finance
  - quantum physics, NGOs, graphics, libraries ...

Created/Solved Issues chart



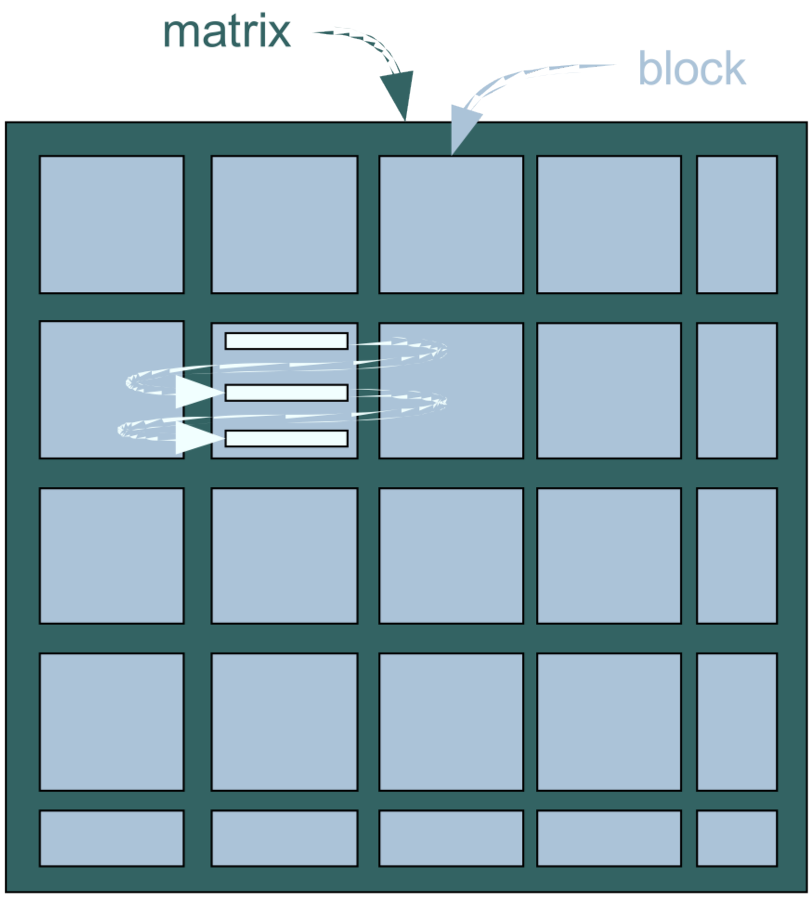
ApacheCon

Leading the Wave
of Open Source

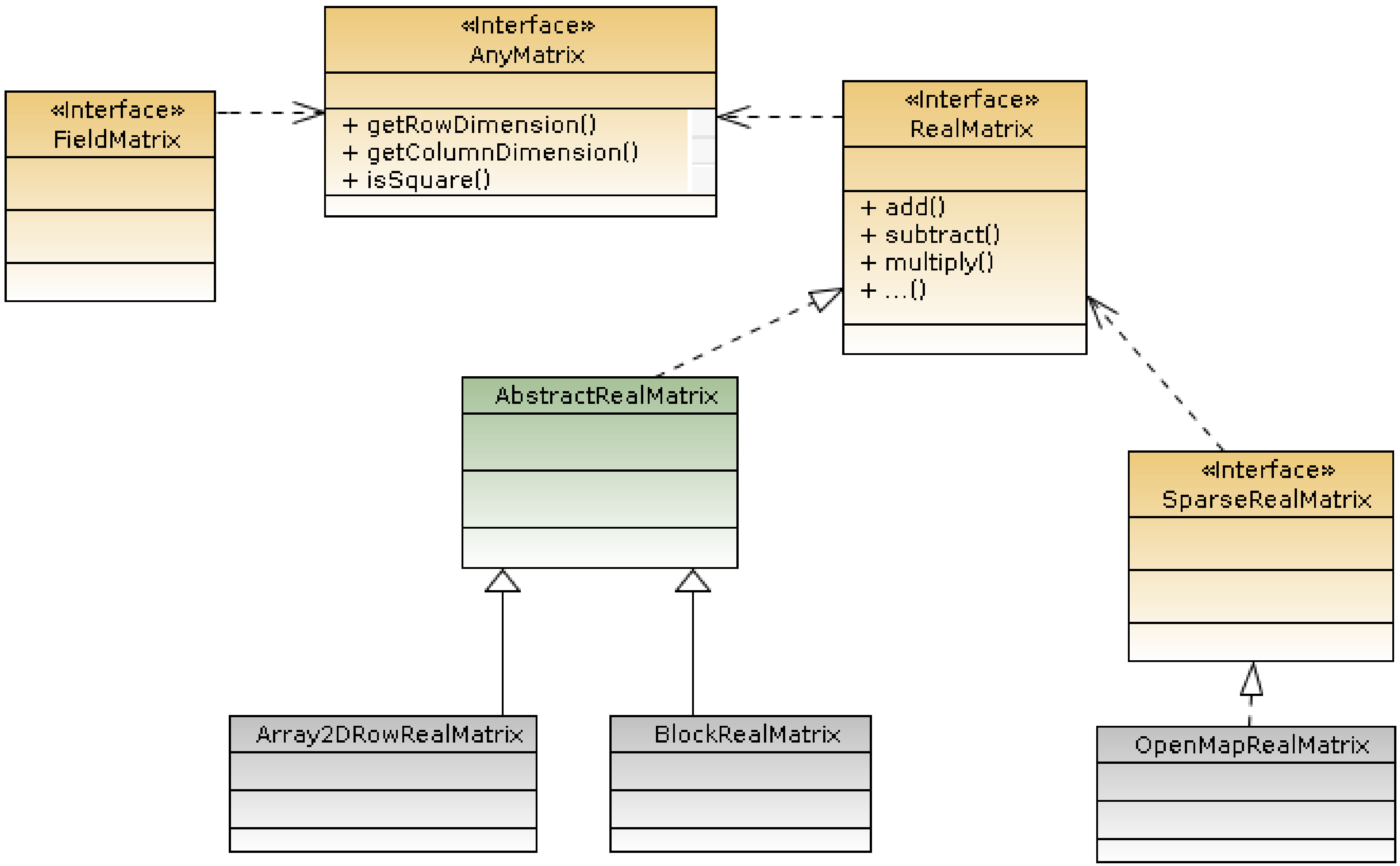
Linear Algebra: several storage schemes

- Full matrices
  - Simple two-dimensions arrays for small matrices
  - Square blocks for large matrices
- Full vectors
  - Simple one-dimension array
- Sparse matrices/vectors
  - Open addressed map with compound index
- Storage scheme is tightly linked to performance
- Users can add their own implementation

Block storage

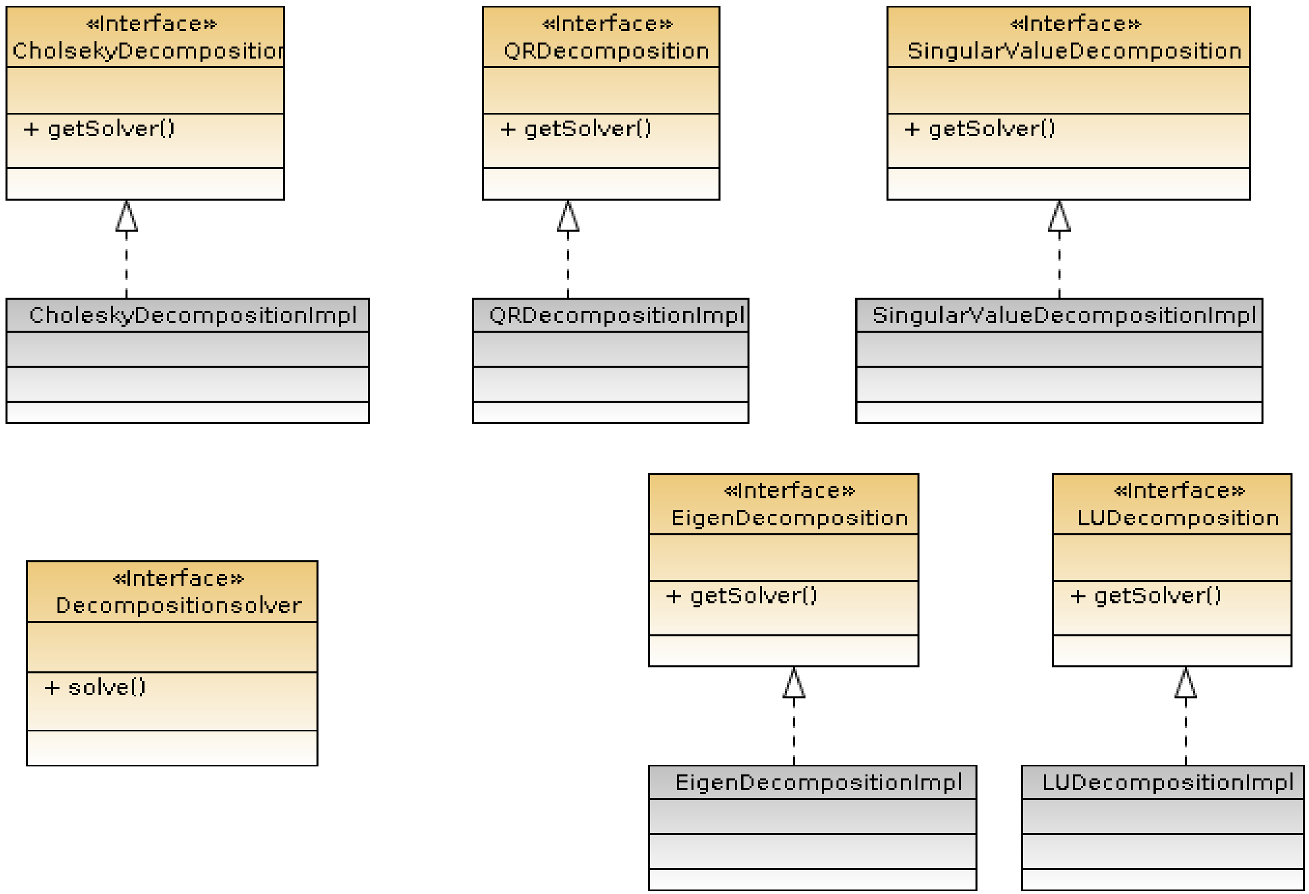


Partial class diagram



Leading the Wave
of Open Source

Available decompositions



Solving a linear system

```
// compute decomposition once for all
double[][] aData = new double[][] {
                        {  2,  3, -2 },
                        { -1,  7,  6 },
                        {  4, -3, -5 }
                    };
boolean    copyArray = false;
RealMatrix a = new Array2DRowRealMatrix(aData, copyArray);
DecompositionSolver solver = new QRDecompositionImpl(a).getSolver();

// solve A X = B
double[]    bData = new double[] { 1, -2, 1 };
RealVector  b     = new ArrayRealVector(bData, copyArray);
RealVector  x     = solver.solve(b);
```

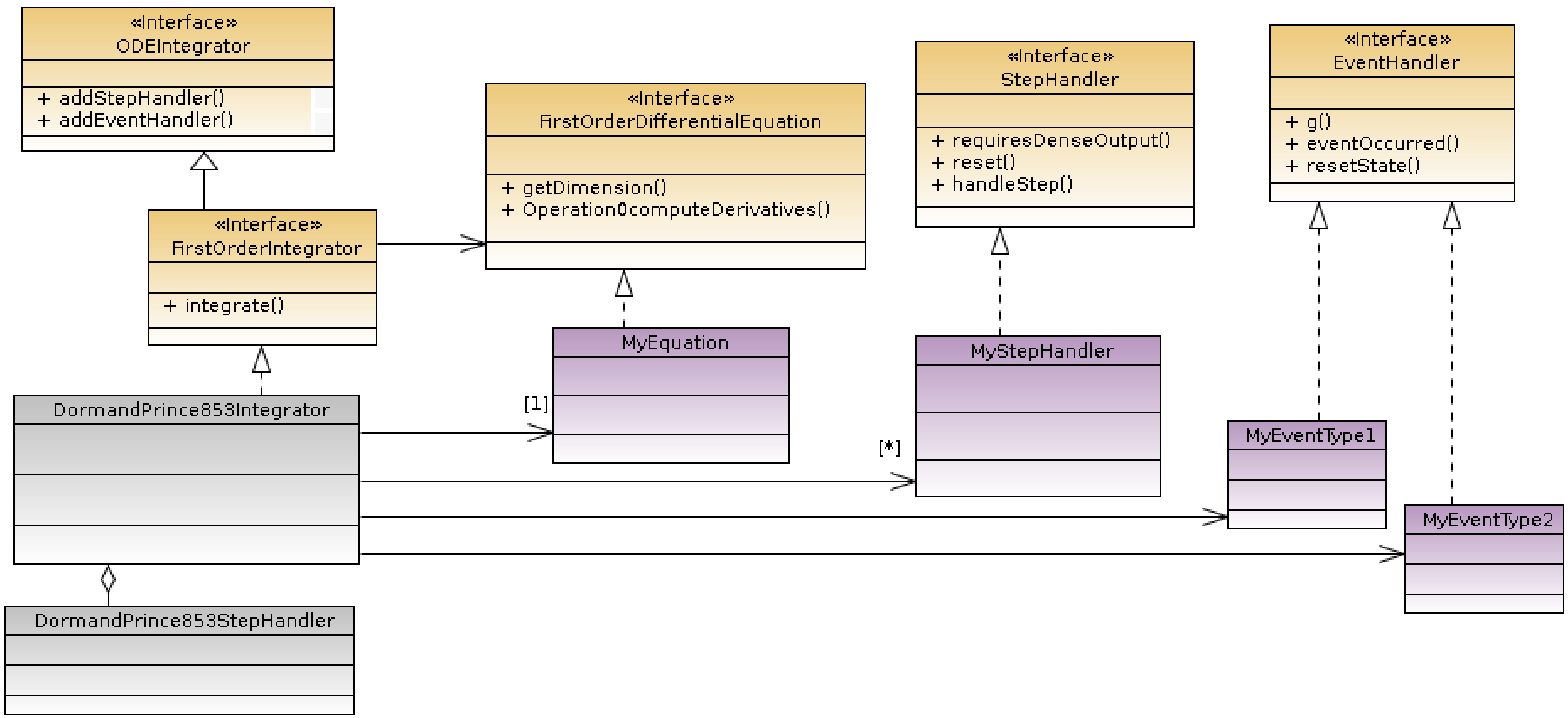
Hints for linear algebra

- Select appropriate decomposition algorithm
  - LU decomposition works only on square matrices
  - QR decomposition is more stable than LU decomposition
  - SVD is more costly, but provides covariance
- Select matrix storage according to size
- Interfaces are similar to JAMA ones
- Matrices inversions are almost never needed
- Commons-math supports fields matrices
  - Available fields: complex, fractions, arbitrary precision floats (in 2.2)
  - Only LU decomposition yet

Ordinary Differential Equations

- Large range of first order ODE solvers
  - Fixed step
    - classical Runge-Kutta, midpoint, 3/8, Euler …
  - Adaptive step
    - Dormand-Prince 8(5,3), Gill, Gragg-Bulirsch-Stoer, Higham-Hall …
  - Multi-steps … better avoid them yet!
    - Adams-Bashforth and Adams-Moulton (adaptive with Nordsieck vector)
- Rich set of features
  - Continuous output
    - Step handlers and complete evolution storage (even on file for later use)
  - Discrete events
    - G-stop, state reset, derivatives reset
  - Automatic step size initialization

ODE organization



Solving an ODE problem

```
// user defined class represention the physical problem to solve
public class MyEquation implements FirstOrderDifferentialEquation {
    public int getDimension() { return 2; }
    public void computeDerivatives(double t, double[] y,
                                   double[] yDot) {
        yDot[0] =  y[1];
        yDot[1] = -y[0];
    }
}

// set up integrator
FirstOrderIntegrator integrator =
    new DormandPrince853(minStep, maxStep, absoluteTol, relativeTol);

// solve Initial Value Problem
double   t0    = 0.0;
double[] y0    = new double[] { 1.0, 0.0 };
double   t     = 1.0;
double[] y     = new double[2];
double realT1 = integrator.integrate(new MyEquation(), t0, y0, t1, y);
```

Continuous Output

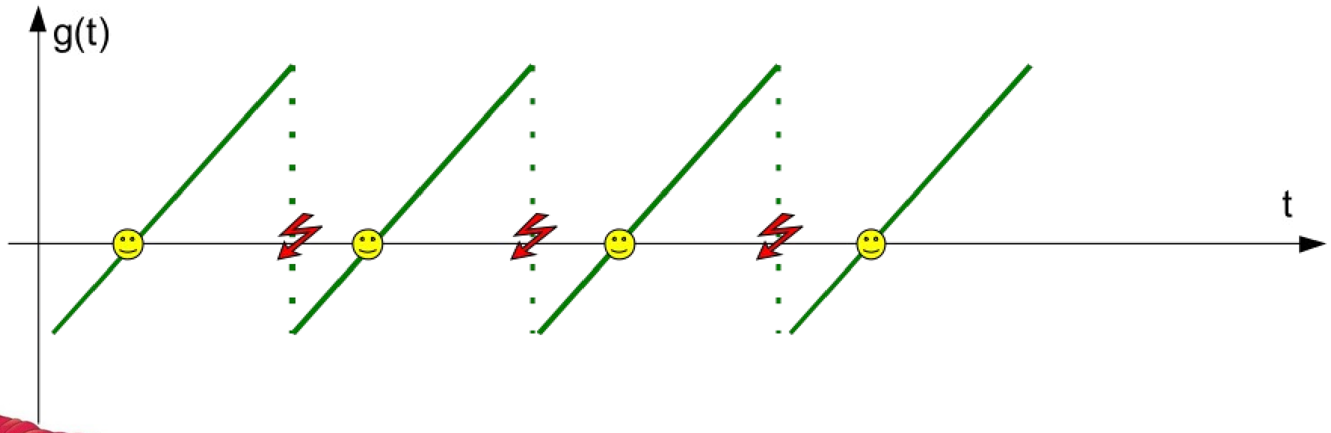
```java
// user defined class called during the integration process
public class MyStepHandler implements StepHandler {

    public void requiresDenseOutput() { return true; }

    public void reset() {}

    public void handleStep(StepInterpolator interp, boolean isLast) {

        double start = interp.getPreviousTime();
        double end   = interp.getCurrentTime();
        for (double t = start; t < end; t += (end - start) / 10) {

            interp.setInterpolatedTime(t);
            double[] interpolatedY = interp.getInterpolatedState();

            for (double yk : interpolatedY) {
               System.out.print(" " + yk);
            }
            System.out.println();

        }
    }
}
```

Discrete Events

```java
// user defined class defining an event
public class MyEventHandler implements EventHandler {

    // method defining the event: it is triggered when the sign changes
    public double g(double t, double[] y) {
        return y[0] - 0.5;
    }

    // method called when the event has been acknowledged by integrator
    public int eventOccurred(double t, double[] y, boolean increasing) {
        if (increasing) {
          System.out.println("starting flight above 0.5 level at " + t);
          return CONTINUE;
        } else {
          System.out.println("back to 0.5 level at " + t + ", stopping");
          return STOP;
        }
    }

    // method called when the event has been acknowledged by integrator
    public void resetState(double t, double[] y) {
    }
}
```

Hints for ODE integration

- Use adaptive step size integrators and fixed step handler
  - Fixed step integrators don't handle changing dynamics well
- Don't use
  - Neither Adams-Bashforth nor Adams-Moulton
    - unreliable yet, step decreases to 0
  - FirstOrderIntegratorWithJacobians (will be completely replaced)
- Best integrators are Dormand-Prince 8(5,3) and GBS
- Don't put discontinuities near events functions roots!



Numerical Analysis

- Types
  - Complex, Fraction, Dfp (Decimal Floating Point), 3D geometry
- Functions
  - Multivariate, univariate, differentiable, polynomials, special functions
- Algorithms
  - Package analysis
    - Root finders, quadrature, interpolation
  - Package optimization
    - Curve fitting
    - Linear optimization with constraints
    - Non linear optimization without constraints (with or without derivatives)
    - Univariate minimization/maximization

Optimization

- Linear case
  - Dantzig's simplex
  - Linear cost function and constraints
  - Full matrices: unsuited for very large problems (size of millions)
- Non-linear case
  - Direct methods
    - Nelder-Mead simplex, Torczon's multi-directional, Powell
  - General methods
    - Conjugate gradient, Gauss-Newton, Levenberg-Marquardt
  - Architecture being revamped for version 3.0
- Curve fitting
  - Based on non-linear optimizers

Hints for optimization

- General methods are the most effective
  - Levenberg-Marquardt can be started far from optimum
- Direct method are robust
  - Handle discontinuities
  - Constraints can be simulated by penalty functions
  - Better than general methods with finite differences derivatives
- Convergence handling is on the move ...

Miscellaneous

- Be aware of floating points properties
  - Avoid cancellation effects
  - Take care of equality tests
  - Use NaN, +∞, -∞ for initialization where appropriate
  - Use FastMath rather than Math (when 2.2 is released ...)
- **Provide feedback**
  - Bugs reports
  - Enhancements requests
  - Participate to discussion and design choices
- **Don't use unreleased versions**
  - They are highly unstable
  - They are more buggy

What will be in 2.2

- Many bugs fixes
- Preparation for transition to 3.0
  - New interfaces
  - Deprecations
- Performances improvements
  - Percentile will be faster by several orders of magnitudes
- New features
  - Computation
    - FastMath, Decimal Floating Points
    - Tricubic interpolation
  - Optimization
    - Powell direct optimizer
    - Gaussian curve fitting
  - Statistics, random numbers
    - WELL family of pseudo-random generators
    - Intercepts in multiple regression

What will be in 3.0

- Revamped interfaces for optimization
  - More consistent handling of convergence
- New set of exceptions
  - More specialized
  - Some will have both a general message and some context data
- Completely rewritten ODE solver with Jacobians
  - Better integration with existing solvers
- Revamped interfaces for statistics
  - Will allow reuse of data sets