odeToJava: A problem-solving environment for initial-value problems

Andrew Kroshko

Department of Computer Science
University of Saskatchewan

March 21, 2013
odeToJava

- is a problem-solving environment (PSE) for research into numerical methods for initial-value problems (IVPs) in ordinary differential equations (ODEs)
- implements a broad range of numerical methods
- can provide numerical analysts with fine-grained control over the solution process
- allows composition of the different components of ODE software using modules
ODEs/IVPs

\[
\frac{dy}{dt}(t) = f(t, y(t))
\]

- only has derivatives with respect to independent variable \( t \)
- independent variable is not always time
- \( y(t) \) is a vector of dependent variables
- an initial condition is given at \( t_0 \)
  \[
y(t_0) = y_0
\]
- not the only way to specify problem data (i.e., boundary-value problems)
Scientific problems described by ODEs

\[
\begin{align*}
\frac{dv}{dt} &= u \\
\frac{du}{dt} &= \epsilon(1 - v^2)\frac{dv}{dt} - v, \quad \epsilon > 0
\end{align*}
\]

- Newtonian physics
- chemical reactions
- electronic circuits with capacitors, resistors, and inductors, etc.
- biological reactions and electrical activity
- population dynamics
ODEs to study or simulate other problems

- “method of lines” for simulation of PDEs
- as a component of larger simulation, e.g., chemical reaction in a fluid flow
- creating reduced models to study the dynamics of more complex problems
Requirement for numerical solvers

- only certain ODE systems have analytic solutions
e.g., linear ODEs with constant coefficients
- solutions to most ODEs must be approximated
- even linear constant-coefficient ODEs are often more easily approximated than computed “exactly”!
- in practice, numerical methods must be used
Conventional solvers using function calls

```fortran
SUBROUTINE RODAS(N,FCN,IFCN,X,Y,XEND,H,
&   RTOL,ATOL,ITOL,
&   JAC,IJAC,MLJAC,MUJAC,DFX,IDFX,
&   MAS,IMAS,MLMAS,MUMAS,
&   SOLOUT,IOUT,
&   WORK,LWORK,IWORK,LIWORK,RPAR,IPAR,IDID)
```

- system languages like **Fortran** and **C/C++** widespread
- interface is function call with fixed signature
- take RHS function and limited number of parameters
- return the solution at final time
- return limited set of other data
- do not generally provide other software infrastructure
Problem-solving environments

- provide computational facilities for a target class of problems
- are productivity-oriented software that solves problems in ways familiar to researchers
- use terminology and methodology of the target class of problems
- limit specialized knowledge required of the underlying computer hardware, software, and algorithms
- provide facilities for easily post-processing, visualization, and performing further computations
ODE Solvers in MATLAB and similar PSEs

- similar to conventional solvers based on system languages
  - function call to invoke solvers
  - monolithic ODE solvers
  - limited amount of data returned
- scripting languages make computations simpler
  - more sophisticated data structures
  - straightforward integration of disparate functionality
  - additional mathematics, output, and visualization
  - extensive boilerplate code not required
  - automatic memory management
  - ideal for information handling and interfacing
Targeted or research-oriented PSEs

- Some PSE research has been targeted towards IVPs
  - Make it easier to solve IVPs (existing software is complex)
  - Give researchers additional facilities to conduct their work
- Types
  - Expert systems to help end-users, Kamel, Ma, and Enright (1993)
  - GUI-based systems to assist researchers, Petcu and Dragan (2000)
  - Object-oriented systems that accommodate many methods, Olsson (1997)
  - Web-based or GUI-based systems for educational use, CODEE (Community of ODE educators)
Issues with contemporary scientific software

- future PSEs must solve issues related to scientific software development
- common difficulties with scientific software development
  - increasing reliance on and necessity for computer software
  - researchers lack skills of professional software developers
  - standard tools and methodologies unsuited to scientists
  - individual scientific codes often unique to a research group
  - increasingly complex computations and larger datasets
  - difficulty with documentation and understanding
  - increasingly complex post-processing and analysis
  - difficulties with verification and errors
  - difficulties in structuring code
- Nature article, Merali (2010)
The origin of the software “crisis”

- assumptions shown to be incorrect
  - information systems could be engineered conventionally
  - well-established engineering practices could be used
  - software process could be rigorously planned \textit{a priori}

- Barbara Liskov: Programming the Turing Machine
  
  https://www.youtube.com/watch?v=ibRar7sWulM
  
  - military had difficulties with software for missiles and avionics
  - focus on hardware although sufficient for contemporary needs
  - many of the applications were well-understood

- NATO conference on software engineering, Naur and Randell (1968/1969)

- IBM OS development indicated experienced programmer can only produce 1000 lines/year, Brooks (1978)
Solutions to the software “crisis”

- code must allow reasoning to happen locally
- goto statement considered harmful; Dijkstra (1968)
- structured programming; Dahl, Dijkstra, and Hoare (1972)
- global variables considered harmful; Wulf and Shaw (1973)
- buy, don’t build; Brooks (1987)
- design patterns
- targeted languages and platforms
- improved tools, testing, and overall process
- open source
Similarities between contemporary issues in scientific software and software “crisis”

- development process and counter-intuitive limitations not taken into account during planning of software
- better methodologies for scientific software not identified
- increasing complexity means there exists difficulty creating adequate software
- focus on performance despite adequate computing power being available
- difficulties with maintainability, documentation, and general-purpose use
- optimal tools and training not determined
Additional issues

- *ad hoc* solutions for specific scientific problems common
- many basic results typically demonstrated in a few thousand lines of straightforward *Fortran* or *C*
- modern software and design techniques have not tended to help up until this point; Arge, Bruaset, and Langtangen (1999)
- lack of software to capture more complex computational needs and workflows
- managing both additional resources and resource limitations requires more complex software
Mathematical demonstrations

- term used by Söderlind and Wang (2006)
- computation that indicates analysis is correct
- not empirically rigorous
- identified as inadequate to design optimal adaptive methods
- Söderlind indicates a standard test protocol is required
- better software methodologies required to implement it
Purpose of odeToJava

- limitations to the number and scope of numerical studies
- each numerical study often results in a unique codebase
- there are >5400 journal articles in category 65L05 (numerical analysis of initial value problems) on MathSciNet
- many of these articles describe new methods
- other new IVP methods may exist in other categories or fields
- very few have been tested extensively, compared extensively, or have been used seriously in software
Purpose of `odeToJava`

- give numerical analysts finer-grained control
- common evaluations may not be adequate or useful
  - based on constant stepsize methods
  - analysis can be limited for combinations of numerical methods
  - assumptions used in mathematical analysis may break down
  - practical IVP software has many additional considerations
Purpose of odeToJava

- **Java** a proven software platform
  - widely-used in business and web computing
  - software development tools
  - object-oriented
  - interface building
  - scalable

- share code between all methods as much as possible
- minimal new code required to experiment with new methods
- method- and experiment-specific code easy to reason about
- provide facilities to construct interfaces for an end-user
odeToJava
JScience/Javolution
Java/Java API
Java Virtual Machine

- **JAVOLUTION** contains high-performance data structures
- **JSCIENCE** has a linear algebra library and other scientific code
Modular solver

- promotes decoupling between components of an IVP solver
- global data structure is in PropertySolver
- uses mediator and pipeline patterns
Automatic ordering of modules based on "properties"
Allows building solver by selecting a list of modules
Hides details of implementation
Modular solver

- different types of solvers for flow control
- modular design and decoupling
- allows a small number of solvers to work for most methods
Integration formula modules

- many methods easily implemented
  - standard methods
  - experimental methods
  - problem-specific methods
Error control modules

- common adaptive schemes easily implemented
  - embedded error-estimation
  - step-doubling error-estimation
  - step-control based on error estimates
Modular solver

- common output methods easily implemented
  - different output formats
  - monitoring
  - user interface
  - special formats required for post-processing
Automated running of experiments

- IVPController, Testable, SolutionTester classes
- straightforward selection of
  - integration method
  - solver type
  - initial stepsize
  - tolerances
- designed to be the “model” in a model-view-controller
- can replicate interface to many existing solvers
- easily allows components to be exchanged
- common framework allows rigorous testing
- minimal code changes required to test different methods
- reference solution input and solution output are text files
Test sets and ARK methods

- nonstiff DE set; Hull, Enright, et al. (1972)
- stiff DE test set; Enright et al. (1975)
- total of 60 problems with many different properties
- well-studied, most current is Enright and Pryce (1987)
- odeToJava uses a wide variety of ERK methods and ARK methods with Jacobian-based splitting
- showed widespread adoption of Dormand–Prince 5(4) is well-justified for non-stiff problems
- showed there is no one dominant ARK method
ERK methods

\[
\begin{align*}
  k_i &= f \left( t_n + \Delta t_n c_i, \ y_n + \Delta t_n \sum_{j=1}^{i-1} a_{ij} k_j \right), \quad i = 1, 2, \ldots, s, \\
  y_{n+1} &= y_n + \Delta t_n \sum_{i=1}^{s} b_i k_i,
\end{align*}
\]

- low cost per step
- suitable for non-stiff problems
- not suitable for stiff problems
ARK methods

\[ f(t, y) = \sum_{\nu=1}^{N} f^{[\nu]}(t, y). \]

\[
k^{[1]}_i = f^{[1]} \left( t_n + \Delta t_n c^{[1]}_i, y_n + \Delta t_n \sum_{j=1}^{i-1} \left( a^{[1]}_{ij} k^{[1]}_j + a^{[2]}_{ij} k^{[2]}_j \right) \right), \quad i = 1, 2, \ldots, s,
\]

\[
\left( I - \Delta t a^{[2]}_{ii} J \right) k^{[2]}_i = f^{[2]} \left( t_n + \Delta t_n c^{[2]}_i, y_n + \Delta t_n \sum_{j=1}^{i-1} \left( a^{[1]}_{ij} k^{[1]}_j + a^{[2]}_{ij} k^{[2]}_j \right) \right), \quad i = 1, 2, \ldots, s,
\]

\[
y_{n+1} = y_n + \Delta t_n \sum_{i=1}^{s} \left( b^{[1]}_i k^{[1]}_i + b^{[2]}_i k^{[2]}_i \right),
\]

- one matrix factorization per stage (or per step)
- stiff linear component and nonstiff non-linear component
Arenstorf orbit problem

- 3-body problem of the Earth, moon, and a massless satellite
- Satellite passes close to singularity at position of moon
- Hamiltonian problem that can be solved by symplectic methods that conserve momentum and nearly conserve energy
- Considered ideal problem for variable-stepsize symplectic methods; Leimkuhler and Reich (2004)
Arenstorf orbit problem

\[ H(q, p) = \frac{p_1^2 + p_2^2}{2} - q_1 p_2 + q_2 p_1 - \frac{\mu}{\sqrt{(q_1 - \mu')^2 + q_2^2}} - \frac{\mu'}{\sqrt{(q_1 + \mu)^2 + q_2^2}}, \]

\[ \dot{q} = \nabla H_p(q, p) = \begin{bmatrix} p_1 + q_2 \\ p_2 - q_1 \end{bmatrix}, \]

\[ \dot{p} = -\nabla H_q(q, p) = -\begin{bmatrix} -p_2 + \frac{\mu' (q_1 + \mu)}{((q_1 + \mu)^2 + q_2^2)^{3/2}} + \frac{\mu (q_1 - \mu')}{((q_1 - \mu')^2 + q_2^2)^{3/2}} \\ p_1 + \frac{\mu' q_2}{((q_1 + \mu)^2 + q_2^2)^{3/2}} + \frac{\mu q_2}{((q_1 - \mu')^2 + q_2^2)^{3/2}} \end{bmatrix}, \]
Störmer–Verlet method

\[
\begin{align*}
    p_{n+\frac{1}{2}} &= p_n - \frac{\Delta t}{2} \nabla_q H(q_n, p_{n+\frac{1}{2}}), \\
    q_{n+1} &= q_n + \frac{\Delta t}{2} \nabla_p H(q_n, p_{n+\frac{1}{2}}) + \frac{\Delta t}{2} \nabla_p H(q_{n+1}, p_{n+\frac{1}{2}}), \\
    p_{n+1} &= p_{n+\frac{1}{2}} - \frac{\Delta t}{2} \nabla_q H(q_{n+1}, p_{n+\frac{1}{2}})
\end{align*}
\]

- second-order explicit method
- symplectic for Hamiltonian problems
**Variable-stepsize symplectic methods**

\[
Q(q, p) = \left(\frac{d_1 d_2}{v}\right)^2 - \frac{\alpha}{2},
\]

\[
G(q, p) = \alpha \left[ \frac{v_1}{v^2} \left( \frac{\mu'(q_1 + \mu)}{d_1} + \frac{\mu(q_1 - \mu')}{d_2} \right) - v_1 \left( \frac{q_1 + \mu}{d_1} + \frac{q_1 - \mu'}{d_2} \right) 
+ \frac{v_2 q_2}{v^2} \left( \frac{\mu'}{d_1} + \frac{\mu}{d_2} \right) - q_2 \left( \frac{1}{d_1} + \frac{1}{d_2} \right) - \frac{p_1 v_2 - p_2 v_1}{v^2} \right],
\]

\[
v_1 = p_1 + q_2, \quad v_2 = p_2 - q_1, \quad v^2 = v_1^2 + v_2^2,
\]

\[
d_1 = (q_1 + \mu)^2 + q_2^2, \quad d_2 = (q_1 - \mu')^2 + q_2^2,
\]

- based on Hairer and Söderlind (2005)
- conventional step controllers using heuristic methods cannot be applied to symplectic methods
- stepsize control based on a function of system state
- integrating step control
Variable-stepsize symplectic methods

\[ \rho_{n+\frac{1}{2}} = \rho_n + \epsilon \, G(q_n, p_n)/2, \]

\[ M = \frac{1}{\rho / \rho_{n+\frac{1}{2}} + \frac{\epsilon}{\rho_{n+\frac{1}{2}}} + \frac{\epsilon}{\rho_{n+\frac{1}{2}}}} \left[ \begin{array}{cc} \frac{1}{\rho / \rho_{n+\frac{1}{2}}} & \frac{\epsilon}{\rho_{n+\frac{1}{2}}} \\ \frac{\epsilon}{\rho_{n+\frac{1}{2}}} & 1 \end{array} \right], \]

\[ p_{n+\frac{1}{2}} = M \left( p_n - \frac{\epsilon / \rho_{n+\frac{1}{2}}}{2} \right) \left[ \frac{\mu'(q_{1,n+\mu})}{(q_{1,n+\mu})^2 + (q_{2,n})^2} + \frac{\mu(q_{1,n+\mu})}{(q_{1,n+\mu})^2 + (q_{2,n})^2} \right], \]

\[ q_{n+1} = M \left( q_n + \frac{\epsilon / \rho_{n+\frac{1}{2}}}{2} \right) \left[ \frac{2p_{1,n+\frac{1}{2}} + q_{n,2}}{2p_{2,n+\frac{1}{2}} - q_{n,1}} \right], \]

\[ p_{n+1} = p_{n+\frac{1}{2}} - \frac{\epsilon / \rho_{n+\frac{1}{2}}}{2} \left[ -p_{2,n+\frac{1}{2}} + \frac{\mu'(q_{1,n+1+\mu})}{(q_{1,n+1+\mu})^2 + (q_{2,n+1})^2} + \frac{\mu(q_{1,n+1+\mu})}{(q_{1,n+1+\mu})^2 + (q_{2,n+1})^2} \right], \]

\[ \rho_{n+1} = \rho_{n+\frac{1}{2}} + \frac{\epsilon}{\rho_{n+\frac{1}{2}}} \, G(q_{n+1}, p_{n+1})/2, \]

\[ t_{n+1} = t_n + \epsilon / \rho_{n+\frac{1}{2}}. \]
Variable-stepsize symplectic methods

- there exists an optimum value of epsilon
Variable-stepsize symplectic methods

- variable-stepsize Störmer–Verlet has lowest error for some parameter values
OO programming and its limitations

- OO programming allows creation of a modular ODE solver
- limitations to OO programming
  - Liskov Substitution Principle limits utility of subtyping; Liskov and Wang (1994)
  - difficult to determine best abstractions and interfaces
  - optimal OO design often different from how humans think
  - allows APIs to become large and complex
  - high-quality OO software is labour-intensive
  - often has counter-intuitive restrictions on versatility
Use of Java in scientific computing

- most popular in first 5 years of 21st century
- many C libraries were ported to Java
  - Colt includes bindings to BLAS and LAPACK
- used in GUI code for MATLAB and many other PSEs
- used by CERN and other large deployments
- contemporary interest because of scalability and robustness
Advantages of JAVA

- most popular programming language in industry
- integrates well with internet and web
- highly scaleable
- many well-developed tools for large code bases
  - highly automated development possible
  - automated navigation and understanding of code
  - packaging and deployment
Disadvantages of Java

- tends to be very verbose with a lot of boilerplate code
- type system can be too restrictive
- can require domain-specific languages or complex interfaces for flexible applications
- difficulty interfacing with other platforms
- floating-point arithmetic not tested as extensively as more well-established scientific platforms
Conclusions

- demonstrated a universal and modular IVP solver is possible
- has advantage of only one codebase
- demonstrated method and problem combinations that have not been widely tested
- software techniques learned can be applied to more complex classes of problems
E. Arge, A.M. Bruaset, and H.P. Langtangen.  
Birkhäuser, Boston, 1997

Special issue on time integration (Amsterdam, 1996)

F.P. Brooks, Jr.  
No silver bullet – essence and accidents of software engineering.  
*Computer*, 20(4):10–19, 1987q

F.P. Brooks, Jr.  
*The Mythical Man-Month: Essays on software engineering*.  

http://www.codee.org/
*Structured programming*.  

E. W. Dijkstra.  
Go to statement considered harmful.  

Comparing numerical methods for stiff systems of ODEs.  

W. H. Enright and J. D. Pryce.  
Two FORTRAN packages for assessing initial value methods.  

E. Hairer and G. Söderlind.  
Explicit, time reversible, adaptive step size control.  


Z. Merali.  
Computational science: Error, why scientific programming does not compute.  
*Nature*, 467, 2010

H. Olsson.  
Object-oriented solvers for initial-value problems.  
In *Modern software tools for Scientific computing*. Birkhäuser, 1997

D. Petcu and M. Drăgan.  
Designing an ODE solving environment.  

P. Naur and B. Randell, editors.  
*Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968*.  
Scientific Affairs Division, NATO, Brussels, 1969

G. Söderlind and L. Wang.  
Evaluating numerical ODE/DAE methods, algorithms and software.  

W. Wulf and M. Shaw.  
Global variable considered harmful.  