Developing a User-Friendly Delay Differential Equations Modeling Package

Hossein ZivariPiran
hzp@cs.toronto.edu

Department of Computer Science
University of Toronto

(part of my PhD thesis under the supervision of professor Wayne Enright)
Outline

- Modeling with Differential Equations (IVPs, DDEs)
- Software Design
- User Interface
An Initial Value Problem (IVP) for Ordinary Differential Equations (ODEs)

\[ y'(t) = f(t, y(t)) \]
\[ y(t_0) = y_0 \]

Retarded Delay Differential Equations (RDDEs)

\[ y'(t) = f(t, y(t), y(t - \sigma_1), \ldots, y(t - \sigma_\nu)) \text{ for } t_0 \leq t \leq t_F \]
\[ y(t) = \phi(t), \text{ for } t \leq t_0 \]

\( \sigma_i = \sigma_i(t, y(t)) \geq 0 \) delay (constant / time dependent / state dependent)

\( \phi(t) \) history function (constant / time dependent)

Neutral Delay Differential Equations (NDDEs)

\[ y'(t) = f(t, y(t), y(t - \sigma_1), \ldots, y(t - \sigma_\nu), y'(t - \sigma_{\nu+1}), \ldots, y'(t - \sigma_{\nu+\omega})) \text{ for } t_0 \leq t \leq t_F \]
\[ y(t) = \phi(t), \ y'(t) = \phi'(t), \text{ for } t \leq t_0, \]
Modeling with DE - The Modeling Process

- Physical/Biological Phenomenon
- Physical Laws / Empirical Rules
- Mathematical Model
- Observations
- Sensitivity Analysis
- Refined Mathematical Model
- Parameter Estimation
- Practical Mathematical Model
A Parameterized IVP

\[ y'(t; \mathbf{p}) = f(t, y(t; \mathbf{p}); \mathbf{p}) \]
\[ y(t_0) = y_0(\mathbf{p}) \]

For example, in the Lotka-Volterra predator-prey model

\[ N'(t) = N(a - bP) \]
\[ P'(t) = P(cN - d) \]

Parameters are

\[ \mathbf{p} = [a, b, c, d] \]
A Parameterized DDE

\[ y'(t; p) = f(t, y(t; p), y(t - \sigma(t; p); p)) \text{ for } t_0(p) \leq t \]
\[ y(t; p) = \phi(t; p), \text{ for } t \leq t_0(p) \]

For Example, in the neutral delay logistic Gause-type predator-prey system [Kuang 1991]

\[
\begin{align*}
y_1'(t) &= y_1(t)(1 - y_1(t - \tau) - \rho y_1'(t - \tau)) - \frac{y_2(t)y_1(t)^2}{y_1(t)^2 + 1} \\
y_2'(t) &= y_2(t) \left( \frac{y_1(t)^2}{y_1(t)^2 + 1} - \alpha \right)
\end{align*}
\]

where \( \alpha = 1/10 \), \( \rho = 29/10 \) and \( \tau = 21/50 \), for \( t \in [0, 30] \). The history functions are

\[
\begin{align*}
\phi_1(t) &= \frac{33}{100} - \frac{1}{10} t \\
\phi_2(t) &= \frac{111}{50} + \frac{1}{10} t
\end{align*}
\]

for \( t \leq 0 \).

Parameters are

\[ p = [\tau, \rho, \alpha, a = \frac{33}{100}, b = -\frac{1}{10}, c = \frac{111}{50}, d = \frac{1}{10}] \]
Classical Theory of Step by Step Integration for IVPs.
- Runge-Kutta (RK).
- Linear Multistep (LM).

Continuous Solution using Polynomial Approximation.
- Continuous Runge-Kutta (CRK).
- Linear Multistep methods have natural approximating polynomials.

DDEs: Combining an “interpolation” method (for evaluating delayed solution values) with an ODE integration method (for solving the resulting “ODE”).
Derivative Discontinuities

In general

\[ \phi'(t_0) \neq f(t_0, \phi(t_0), \phi(t_0 - \sigma_1), \ldots, \phi(t_0 - \sigma_\nu)) \]

Due to the existence of delays, discontinuities propagate along the integration interval.

Solution is smoothed for RDDEs but in general not for NDDEs.

The RK and LM methods fail in presence of discontinuities.

Treatment: Tracking Discontinuities and forcing them to be mesh points.
Forward Sensitivity Analysis

- The (first order) solution sensitivity with respect to the model parameter \( p_i \) is defined as the vector

\[
    s_i(t; \mathbf{p}) = \left\{ \frac{\partial}{\partial p_i} \right\} y(t; \mathbf{p}), \quad (i = 1, \ldots, \mathcal{L})
\]

- The second order solution sensitivity with respect to the model parameters \( p_i \) and \( p_j \) is defined as the vector

\[
    r_{ij}(t; \mathbf{p}) = \left\{ \frac{\partial}{\partial p_j} \right\} s_i(t; \mathbf{p}) = \left\{ \frac{\partial^2}{\partial p_j \partial p_i} \right\} y(t; \mathbf{p}), \quad (i, j = 1, \ldots, \mathcal{L})
\]
Internal Differentiation Approach for IVPs

\[ y'(t; p) = f(t, y(t; p); p), \quad y(t_0) = y_0(p) \]

\[ \downarrow \]

Differentiation + Chain Rule + Clairaut’s Theorem

\[ \downarrow \]

\[ s_i' = \frac{\partial f}{\partial y} s_i + \frac{\partial f}{\partial p_i}, \quad s_i(t_0) = \frac{\partial y_0(p)}{\partial p_i}, \quad (i = 1, \ldots, L) \]
Adapted Internal Differentiation Approach for DDEs

\[
y'(t; p) = f(t, y(t; p), y(\alpha(t, y; p); p), y'(\alpha(t, y; p); p); p) \\
y(t; p) = \phi(t; p), \text{ for } t \leq t_0(p)
\]

\[
\downarrow
\]

Differentiation + Chain Rule + Clairaut’s Theorem

\[
\downarrow
\]

\[
s'_i(t) = \frac{\partial f}{\partial y} s_i(t) + \frac{\partial f}{\partial y(\alpha_k)} \left( y'(\alpha_k) \left( \frac{\partial \alpha}{\partial y} s_i(t) + \frac{\partial \alpha}{\partial p} \right) + s_i(\alpha) \right) \\
+ \frac{\partial f}{\partial y'(\alpha)} \left( y''(\alpha) \left( \frac{\partial \alpha}{\partial y} s_i(t) + \frac{\partial \alpha}{\partial p} \right) + s'_i(\alpha) \right) \\
+ \frac{\partial f}{\partial p}
\]
- Hybrid ODE systems [Tolsma & Barton]

- Continuous transition at $t = \lambda$,

$$ y(\lambda^+) = y(\lambda^-) $$

$$ \downarrow $$

$$ \frac{\partial y}{\partial p_l}(\lambda^+) = \frac{\partial y}{\partial p_l}(\lambda^-) + (y'(\lambda^-) - y'(\lambda^+)) \frac{\partial \lambda}{\partial p_l} $$

- Triggered by,

$$ g(t, y, y'; p) = 0 $$

$$ \downarrow $$

$$ \frac{\partial g}{\partial y'} \left( \frac{\partial}{\partial t} \left( \frac{\partial y}{\partial p_l} \right) + y'' \frac{\partial \lambda}{\partial p_l} \right) + \frac{\partial g}{\partial y} \left( \frac{\partial y}{\partial p_l} + y' \frac{\partial \lambda}{\partial p_l} \right) + \frac{\partial g}{\partial p_l} + \frac{\partial g}{\partial t} \frac{\partial \lambda}{\partial p_l} = 0 $$
Modeling with DE - Proper Handling of Sensitivity Jumps

- DDEs
  - Discontinuity points of $y'(t; p)$,
    \[
    \Lambda(p) \equiv \{ \lambda_1(p), \lambda_2(p), \ldots \}.
    \]
  - Identified by,
    \[
    \alpha(\lambda_{r+1}(p), y; p) = \lambda_r(p) \quad r = 1, 2, \ldots
    \]
  - Can be viewed as $\lambda_{r+1}(p)$ being a solution of,
    \[
    \hat{g}(t, y; p) = \alpha(t, y; p) - \lambda_r(p) = 0.
    \]
  - Using the result for hybrid ODEs,
    \[
    \frac{\partial \lambda_{r+1}(p)}{\partial p_l} = -\frac{\partial \alpha}{\partial y} \frac{\partial y'}{\partial p_l} + \frac{\partial \alpha}{\partial p_l} - \frac{\partial \lambda_r(p)}{\partial p_l} \quad \text{for } r \geq 1
    \]
    \[
    \frac{\partial \lambda_1(p)}{\partial p_l} = \frac{\partial t_0(p)}{\partial p_l}
    \]
Integration (first order sensitivities)

1. Initialize \( \lambda_1 = t_0(p) \).

2. \( r \leftarrow 1 \).

3. Integrate the equations up to a \( C^1 \)-discontinuity point \( \lambda_{r+1} \).

4. Update the state variables (sensitivities) using

\[
\frac{\partial y}{\partial p_l}(\lambda_{r+1}^+) = \frac{\partial y}{\partial p_l}(\lambda_{r+1}^-) + (y'(\lambda_{r+1}^-) - y'(\lambda_{r+1}^+)) \frac{\partial \lambda_{r+1}(p)}{\partial p_l}
\]

5. \( r \leftarrow r + 1 \) and restart (3).
Parameter Estimation Problem

♦ A System of Parameterized IVP

\[ y'(t; p) = f(t, y(t; p); p) \]
\[ y(t_0) = y_0(p) \]

or DDE

\[ y'(t; p) = f(t, y(t; p), y(t - \sigma(t; p)); p) \quad \text{for} \quad t_0(p) \leq t \]
\[ y(t; p) = \phi(t; p), \quad \text{for} \quad t \leq t_0(p) \]

♦ A Set of Data (Observations/Measurements)

\[ \{ Y(\gamma_i) \approx y(\gamma_i; p^*) \} \]

♦ Estimate \( p^* \) by minimizing an objective function.

\[ W(p) = \sum_i [Y(\gamma_i) - y(\gamma_i; p)]^2. \]
Algorithms for Nonlinear Least-Squares

♦ Unconstrained

$$\min_p W(p) = \sum_i [Y(\gamma_i) - y(\gamma_i; p)]^2.$$

↓

Levenberg-Marquardt
Variations of Sequential Quadratic Programming (SQP)

♦ Constrained

$$\min_p W(p) = \sum_i [Y(\gamma_i) - y(\gamma_i; p)]^2,$$

$$c_j(p) = 0, \quad j \in \mathcal{E},$$

$$c_j(p) \geq 0, \quad j \in \mathcal{I}.$$  

↓

Sequential Quadratic Programming (SQP)

The *smoothness* of functions involved in the problem, the objective function and constraints, is a *necessary* assumption.
Computing the Gradient/Hessian

\[
\left( \frac{\partial W(p)}{\partial p_l} \right)_\pm = -2 \sum_i [Y(\gamma_i) - y(\gamma_i; p)] \left( \frac{\partial y(\gamma_i; p)}{\partial p_l} \right)_\pm
\]

\[
\left( \frac{\partial^2 W(p)}{\partial p_l \partial p_m} \right)_\pm = 2 \sum_i \left[ \left( \frac{\partial y(\gamma_i; p)}{\partial p_l} \right)_\pm \left( \frac{\partial y(\gamma_i; p)}{\partial p_m} \right)_\pm - [Y(\gamma_i) - y(\gamma_i; p)] \left( \frac{\partial^2 y(\gamma_i; p)}{\partial p_l \partial p_m} \right)_\pm \right]
\]

Recall the jump equation for sensitivities

\[
\frac{\partial y}{\partial p_l}(\lambda_{r+1}^+) = \frac{\partial y}{\partial p_l}(\lambda_{r+1}^-) + (y'(\lambda_{r+1}^-) - y'(\lambda_{r+1}^+)) \frac{\partial \lambda_{r+1}(p)}{\partial p_l}
\]

\[\downarrow\]

The General Rule: A jump occurs in \( W(p) \) when

A discontinuity point \( \lambda_{r+1}(p) \) passes a data point \( \gamma_i \).
Avoiding The Non-smoothness

Force the ordering by adding \( \lambda_r(p) \leq \gamma_i \leq \lambda_{r+1}(p) \) to the set of constraints. The partial derivatives (gradient) of the new constraints, \( \frac{\partial \lambda_{r+1}(p)}{\partial p} \), can be computed recursively.
Software Design

- User Calls

```
User -> Simulator
  |    A
  V    B
  |    C
Sensitivity Analyzer
  |
Parameter Estimator
```

- A
- B
- C
Software Design

- Simulator

Diagram:
- Master Integrator
  - Step Integrator (IVP Solver)
    - Past Values Provider
      - Solution Evaluator
  - Discontinuity Detector/Locator
    - Violation Manager
Sensitivity Analyzer

Variational Integrator

Jump Handler

Simulator::
Master Integrator
Parameter Estimator

Master Control

Optimizer (SQP)
\texttt{e04unc/nag\_opt\_nlin\_lsq} from NAG

Subfunctions Calculator

Constraints Calculator

Solution/Gradient Provider

Simulator:: Master Integrator

Sensitivity Analyzer:: Variational Integrator
Simulation

```c
problem1->create(nVariables, nParameters, nEventFuncs, nHistorySegments);
problem1->setF(f);
problem1->setHistorySegments(history);
problem1->setDelayArguments(nu, omega, delays, stateDependent);
problem1->setY0(initial Value);

simulator1->simulate(problem1,
    end time, parameters,
    relTolerance, absTolerance,
    simulationSolution1, communication pointer);
```
User Interface

- Sensitivity Analysis

```plaintext
needSensitivity[0] = TRUE;
needSensitivity[1] = FALSE;
.
.
.
sensitivityAnalyzer1->computeSensitivities(
  simulator1,
  problem1,
  end time, parameters,
  relTolerance, absTolerance,
  needSensitivity,
  sensitivitySolution1,
  communication pointer);
```
Parameter Estimation

Constraints:
- Linear Constraints (Coefficients Matrix)
- Nonlinear Constraints (Function)
- Simple Bounds

```c
data1->Load("example01.d");
constraints1->setSimpleLowerBound(0, .05);

parameterEstimator1->EstimateParameters(simulator1,
            problem1,
            data1,
            constraints1,
            optimumParameters,
            relTolerance, absTolerance,
            statistics,
            communication pointer);
```