The Design and Implementation of a 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

- The Modeling Process
- Why Modeling Packages?
- Difficulties of Designing a Modeling Package
- A Proposed Framework Based on Experiment
The Modeling Process

The Modeling Process - Parameterized Models

\[
\begin{align*}
y(t) & \mapsto y(t; p) \\
y(x) & \mapsto y(x; p) \\
y(x, t) & \mapsto y(x, t; p)
\end{align*}
\]

Parameters are used to:

- Make the model applicable in **similar situations**.
- Analyze the effect of **uncertainties**.
- Represent **unknown quantities**.
Computing a numerical approximation for fixed values of parameters and some initial/boundary conditions.

- Traditionally considered as the core of numerical studies.
- Many researchers are working on developing faster/more reliable methods.
- Generalized Methods - vs - Specialized Methods
Forward Sensitivity Analysis

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

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

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

$$r_{ij}(t; p) = \left\{ \frac{\partial}{\partial p_j} \right\} s_i(t; p) = \left\{ \frac{\partial^2}{\partial p_j \partial p_i} \right\} y(t; p), \quad (i, j = 1, \ldots, \mathcal{L})$$
Sensitivity information can be used to:

- Estimate which parameters are most influential in affecting the behavior of the simulation. Such information is crucial for
  - Experimental Design
  - Data Assimilation
  - Reduction of complex nonlinear models

- Study of Dynamical Systems: Periodic orbits, the Lyapunov exponents, chaos indicators, and bifurcation analysis are fundamental objects for the complete study of a dynamical system, and they require computation of the sensitivities with respect to the initial conditions of the problem.

- Evaluate optimization gradients and Jacobians in the setting of
  - Dynamic Optimization
  - Parameter Estimation
Parameter Estimation Problem

A Parameterized Model

\[ y(t; \mathbf{p}) \]

A Set of Data (Observations/Measurements)

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

Estimate \( \mathbf{p}^* \) by minimizing an objective function.

e.g.

\[ W(\mathbf{p}) = \sum_i [Y(\gamma_i) - y(\gamma_i; \mathbf{p})]^2. \]
Parameter Estimation/Fitting is not easy, like almost any other optimization problem,

- The parameters may not be identifiable \( \Rightarrow \) Numerical Difficulties
- The efficiency may depend strongly on the place/number of data points \( \Rightarrow \) Strategies for Collecting the Best Data
Why Modeling Packages?

- Sensitivity analyzer and parameter estimator have become as essential as simulator for studying a phenomenon using a mathematical model.

- Increase of the processing power ⇒ Use of more complex/detailed models ⇒ sensitivity analysis and parameter estimation become more complicated.

- Sensitivity analysis and parameter estimation are highly dependent on the simulator ⇒ An integrated design could lead to efficient communications and prevent possible redundancies.
Difficulties of Designing a Modeling Package

- We need to develop an efficient simulator, a sensitivity analyzer and a parameter estimator ⇒ Time Consuming/Expensive

- How to deal with the "Generality - vs - Efficiency" problem?

- Want to have a manageable design ⇒ with little effort be able to change some algorithms or add new algorithms.

- Want to have a user-friendly interface ⇒ Easy enough for nonspecialists and controllable enough for specialist.

- Ideally Parallelizable ⇒ At least thread safe ⇒ No Global Variables.

- Easily incorporable into other packages/programms ⇒ Easy to do experiments, for instance, comparing different algorithms ⇒ No Assumption for Global Modules.
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)

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

\[ \sigma_i = \sigma_i(t, y(t)) \geq 0 \text{ delay (constant / time dependent / state dependent)} \]
\[ \phi(t) \text{ history function (constant / time dependent)} \]

Neutral Delay Differential Equations (NDDEs)

\[ y'(t) = f(t, y(t), y(t - \sigma_1), \cdots, y(t - \sigma_\nu), \]
\[ y'(t - \sigma_{\nu+1}), \cdots, y'(t - \sigma_{\nu+\omega})) \quad \text{for} \quad t_0 \leq t \leq t_F \]
\[ y(t) = \phi(t), \quad y'(t) = \phi'(t), \quad \text{for} \quad t \leq t_0, \]
A Parameterized DDE

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

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

\[ y'_1(t) = y_1(t)(1 - y_1(t - \tau) - \rho y'_1(t - \tau)) - \frac{y_2(t)y_1(t)^2}{y_1(t)^2 + 1} \]
\[ y'_2(t) = y_2(t)\left(\frac{y_1(t)^2}{y_1(t)^2 + 1} - \alpha\right) \]

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

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

for \( t \leq 0 \).

Parameters are

\[ \mathbf{p} = [\tau, \rho, \alpha, a = \frac{33}{100}, b = -\frac{1}{10}, c = \frac{111}{50}, d = \frac{1}{10}] \]
DDEs - Numerical Simulation

- 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

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

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.
The predator-prey model
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

\[ \downarrow \]

\[ s'_i = \frac{\partial f}{\partial y} s_i + \frac{\partial f}{\partial p_i}, \quad s_i(t_0) = \frac{\partial y_0(p)}{\partial p_i}, \quad (i = 1, \ldots, \mathcal{L}) \]
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

\[ \downarrow \]

\[ s'_i(t) = \frac{\partial f}{\partial y} s_i(t) + \frac{\partial f}{\partial y(\alpha_k)} \left( y'(\alpha_k) \left( \frac{\partial \alpha}{\partial y} s_i(t) + \frac{\partial \alpha}{\partial p} \right) + s_i(\alpha) \right) \]

\[ + \frac{\partial f}{\partial y'(\alpha)} \left( y''(\alpha) \left( \frac{\partial \alpha}{\partial y} s_i(t) + \frac{\partial \alpha}{\partial p} \right) + s'_i(\alpha) \right) \]

\[ + \frac{\partial f}{\partial p} \]
The predator-prey model
Using the Theory of Hybrid ODE systems [Tolsma & Barton]

Sensitivity Update Equation,

\[
\frac{\partial y}{\partial p_l}(\lambda^+_{r+1}) = \frac{\partial y}{\partial p_l}(\lambda^-_{r+1}) + \left(y'(\lambda^-_{r+1}) - y'(\lambda^+_{r+1})\right) \frac{\partial \lambda^+_{r+1}(p)}{\partial p_l}
\]

where

\[
\frac{\partial \lambda^+_{r+1}(p)}{\partial p_l} = -\frac{\partial \alpha}{\partial y} \frac{\partial y}{\partial p_l} + \frac{\partial \alpha}{\partial p_l} - \frac{\partial \lambda_r(p)}{\partial p_l} \quad \text{for } r \geq 1
\]

\[
\frac{\partial \lambda_1(p)}{\partial p_l} = \frac{\partial t_0(p)}{\partial p_l}
\]
Algorithms for Nonlinear Least-Squares

- **Unconstrained**
  \[
  \min_{\mathbf{p}} W(\mathbf{p}) = \sum_i \left[ Y(\gamma_i) - y(\gamma_i; \mathbf{p}) \right]^2.
  \]
  ↓
  Levenberg-Marquardt
  Variations of Sequential Quadratic Programming (SQP)

- **Constrained**
  \[
  \min_{\mathbf{p}} W(\mathbf{p}) = \sum_i \left[ Y(\gamma_i) - y(\gamma_i; \mathbf{p}) \right]^2,
  \]
  \[
  c_j(\mathbf{p}) = 0, \quad j \in \mathcal{E},
  \]
  \[
  c_j(\mathbf{p}) \geq 0, \quad j \in \mathcal{I}.
  \]
  ↓
  Sequential Quadratic Programming (SQP)

The *smoothness* of functions involved in the problem, the objective function and constraints, is a *necessary* assumption.
DDEs - Parameter Estimation

Computing the Gradient/Hessian

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

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

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

\[\text{a discontinuity point } \lambda_{r+1}(p) \text{ passes a data point } \gamma_i.\]


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
- Sensitivity Analyzer
- Parameter Estimator

Arrows indicate the flow of data or commands:
- A from User to Simulator
- B from User to Sensitivity Analyzer
- C from Parameter Estimator to Sensitivity Analyzer
- C from Parameter Estimator to Simulator
Simulator

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

- Variational Integrator
- Jump Handler
- Simulator:: Master Integrator
Parameter Estimator

- Master Control
  - Optimizer (SQP)
    - $e04unc/nag_opt_nlin_lsq$ from NAG
  - Subfunctions Calculator
  - Constraints Calculator
  - Solution/Gradient Provider
    - Simulator:: Master Integrator
    - Sensitivity Analyzer:: Variational Integrator
User Interface

Simulation

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

myIVP2DDEImprovedCRK1 = new IVP2DDEImprovedCRK(0/*interp_Flag*/);
mySimulator1 = new ddemSimulator(myIVP2DDEImprovedCRK1);

simulator1->simulate(problem1,
        end time, parameters,
        relTolerance, absTolerance,
        simulationSolution1, communication pointer);
```
needSensitivity[0] = TRUE;
needSensitivity[1] = FALSE;
.
.
.
sensitivityAnalyzer1->computeSensitivities(
simulator1,
problem1,
end time, parameters,
relTolerance, absTolerance,
needSensitivity,
sensitivitySolution1,
communication pointer);
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);
```
We need to develop an efficient simulator, a sensitivity analyzer and a parameter estimator ⇒ Time Consuming/Expensive

How to deal with the "Generality - vs - Efficiency" problem?

Want to have a manageable design ⇒ with little effort be able to change some algorithms or add new algorithms.

Want to have a user-friendly interface ⇒ Easy enough for nonspecialists and controllable enough for specialist.

Ideally Parallelizable ⇒ At least thread safe ⇒ No Global Variables.

Easily incorporable into other packages/programms ⇒ Easy to do experiments, for instance, comparing different algorithms ⇒ No Assumption for Global Modules.