Pure Modelica Unit Testing: From Mathematical Algorithms to Physical Modeling

OpenModelica Annual Workshop 2019

Atiyah Elsheikh
Mathemodica.com
4th February 2019
Atiyah.Elsheikh@mathemodica.com
Outline

- First testing experiences
  - Mathematical algorithms in Modelica

- Unit testing in Modelica
  - existing technologies

- A bit more “formal” Unit testing
  - Upgrading to Physical Modeling
Simulation tools not only perform numerical solutions based on the system equations but also assist the modeler in systems analysis. Doubtlessly the most important systems analysis tool is Sensitivity Analysis ...
SA of Modelica Models

Modelica model

\[ F(\dot{x}, x, p, t) = 0 \]
\[ x(0) = x_0(p) \]

simulate

\[ x(t) \]

Dynamic Parameter Sensitivities (DPS)

\[ \frac{\partial x}{\partial p}(t) \]
\[ \frac{\partial x}{\partial x_0}(t) \]
ADModelica (2007)

OMC Compiler

XML Modelica

XML Parser

Analyzer

Unparser

Visualizer

equation
\begin{align}
\frac{d}{dt}v &= R_1 \frac{d}{dt}i \\
\frac{d}{dt}i &= -\frac{U_v}{L_v} - \frac{v}{L_i} \\
\frac{d}{dt}C_i &= C_i \frac{d}{dt}v
\end{align}

\text{for } i \in 1:3
\text{ in } \text{step 1/3}
\text{ step } 1/3
\text{ end for}
Sample of generated code (I/II)
• Declaration part

```plaintext
model ADSimpleReaction
Real S(start=1);  
Real P(start=0);  
Real v;           
Real[3] g_S, g_P, g_v;  
parameter Real vmax=1;  
constant Real[3] g_vmax={1,0,0};  
parameter Real k=1;  
constant Real[3] g_k={0,1,0};  
parameter Real Ik=1;  
constant Real[3] g_Ik={0,0,1};
protected
Real loc01,loc02,loc03,loc04,loc05;  
Real g loc01,g loc02,g loc03,g loc04,g loc05;
```
Sample of generated code (II/II)

- Equation part

```plaintext
equation
  der(S)=-v;
  der(P)=v;
  //v = vmax * (S/(S+k)) * (Ik/(P+Ik));

algorithm
  loc01 := vmax*S;
  loc02 := S+k;
  loc03 := loc01/loc02;
  loc04 := P+Ik;
  loc05 := Ik/loc04;
  v := loc03*loc05;

//Derivatives:

equation
  for i in 1:3 loop
    der(g_S[i])=-g_v[i];
    der(g_P[i])=g_v[i];
  end for;

algorithm
  for i in 1:3 loop
    g_loc01 := g_vmax[i]*S+vmax*g_S[i];
    g_loc02 := g_S[i]+g_k[i];
    g_loc03 := (g_loc01*loc02-loc01*g_loc02)/(loc02^2);
    g_loc04 := g_P[i]+g_Ik[i];
    g_loc05 := (g_Ik[i]*loc04-Ik*g_loc04)/(loc04^2);
    g_v[i] := g_loc03*loc05+loc03*g_loc05;
  end for;
end ADSimpleReaction;
```
Algorithmic Differentiation (AD) of Modelica libraries (2011)

extend
Algorithmic differentiation Modelica Libraries

The ADGenKinetics Library

\[ \frac{\partial}{\partial \alpha} \prod_{i} A_k \prod_{j} S_i \prod_{j} P_j \prod_{i} I_i \]

Algorithmically differentiated Modelica library for biochemical reaction networks

https://github.com/modelica-3rdparty/ADGenKinetics

The ADMSL Library

Serves as an example of algorithmically differentiated Modelica library

https://github.com/AIT-CES-LAB/ADMSL
Chua Circuit

Standard Modelica model for an electrical circuit


This example simulates the current and voltage at all components
Chua Circuit importing ADMSL

The sensitivities of the current w.r.t. $L$

$$\frac{\partial X_i}{\partial L}$$
Chua Circuit importing ADMSL

The sensitivities of the voltage at L w.r.t. all parameters

$$\frac{\partial V_L}{\partial X_p}$$
Applications of DPS

1) Modeling-Oriented
   - Control Coefficients, Local SA, Parameter Sweeping Studies, Model Simplification, ...

2) Statistical
   - Regression Analysis, Global SA, Identifiability Analysis, ...

3) Optimization
   - Cost-functions expressed in terms of DPS

Atiyah Elsheikh and Sergei Kucherenko. (2019)
Dynamic parameter sensitivities: Summary of applications – version 1.0.
Technical Report, to appear
Scaled Parameter Sensitivities

\[ CC_p^x = \frac{p}{x(t)} \frac{\partial x}{\partial p}(t) \]

Several snapshots at different time points
Correlation among parameters

Based on Fischer Information Matrices

Strongly Correlated Parameters
Testing AD

Finite Difference Methods

\[
\frac{\partial x}{\partial p_j}(t, p) = \frac{x(t, p + e_j \delta_j) - x(t, p)}{\delta_j} + O(\delta_j)
\]

\[\delta_j = \epsilon p_j\]

Adhoc Testing (2007-2014)

1) Evaluate DPS with FD externally with Matlab small models

2) Simulate AD models

3) Check if the curves behave similarly Sometimes compute Relative errors

4) Investigate cause of errors, if any
PSTools Library

- Promotes the usage of PS at Modelica level
- Serves as utility package for arbitrary Modelica libraries
- PS Package
  - Extensive set of examples for analytical derivatives
    - Including hybrid systems
  - Generic Models for advanced FD
  - Second-order DPS
- Tools Package
  - Taylor series approximation
  - Parameter Sweeping studies
  - Control Coefficients
  - ...

Under Development (2019) Highly Experimental

Unit Testing is must
FD with PSTools

**model** ParM

**extends** Utilities.Parameterized(
    NP = 2,
    _P = {0.4, 0.5},
    PNAME = {"p1", "p2"},
    NX = 2,
    _X = {_M.x1, _M.x2},
    XNAME = {"x1", "x2"}
);

**protected**

MyModel _M(
    p1 = _P[1],
    p2 = _P[2]);

**end** ParM;

**model** FDPParM

PSTools.PS.FD.CD2
PS(**redeclare replaceable**
    Model ParModel = ParM);

**end** FDPParM;
Parameter Sweeping Studies

Example of Sensitivity Analysis with the Bioprocess Library

Jan Peter Axelsson & Atiyah Elsheikh
Modprod 2019
Unit Testing Technologies for Modelica

Marco Kessler, Testing Tutorial
Dassault Systems, Modelica 2017 Prague (Link)

- CSV Compare – ESI ITI
- BuildingPy – LBNL
- PySimulator – DLR
- test.openmodelica.org
- XogenyTest – Xogeny

- Model Management – Dymola
- Testing tool kits (many Companies)
- Testing Library -- Dymola
XogenyTest

- Pure Modelica
- Minimal
  - external scripting
  - log files
- Easy to use
- No dependencies

Not meant to be comprehensive but definitely an ideal getting start

Michael Tiller
https://github.com/xogeny/XogenyTest

Tiller, Michael M., and Burit Kittirungsi.
"UnitTest: A Library for Modelica Unit Testing"
Modelica Conference Vienna, Austria, 2006
XogenyTest

Low level functions

asserting whether an expected value $x$ lies within $\varepsilon$-neighbourhood of a reference value $r$ (i.e. $N_\varepsilon(r)$)

$|x - r| < \varepsilon$

or

within the same order of magnitude

$\left| \log \frac{x}{r} \right| < \varepsilon$
asserting an expected value $x$ w.r.t. a reference value $r$ at time $T \in \{t_0, t_f\}$

\[ |x(T) - r| < \epsilon \]

or at time $t \in (t_0, t_f)$

\[ |x(t) - r| < \epsilon \]
asserting an expected event \( e(t) \) occurs at time \( T \)

\[
e([T-\epsilon, T+\epsilon]) \rightarrow \text{True}
\]

Asserting a trajectory \( x(t) \) at discrete time points \( \{t_1, t_2, \ldots, t_N\} \) w.r.t. reference values \( r(t_i) \)

\[
|x(t_i) - r(t_i)| < \epsilon
\]
Examples (I/II)

model CheckSuccess

Real x = time^2;

AssertTrajectory check_x(
    actual=x,
    expected=[0,0; 1,1; 2,4; 3,9]);

Annotation(
    TestCase(
        action="simulate",
        result="success"),
    experiment(StopTime=4));

end CheckSuccess;
Examples (II/II)

```plaintext
model CheckSuccess

Real x = time;

AssertBecomesTrueAt
    check_event(event=(x>2), at=2);

Annotation(
    TestCase(
        action="simulate",
        result="success"),
    experiment(StopTime=4));

dend CheckSuccess;
```
Assert that a signal \( u(t) \) is of an average value \( a \) over \( [t_0, t_f] \)

\[
\frac{1}{t_f - t_0} \left| \int_{t_0}^{t_f} u(t) \, dt \right| - a < \epsilon
\]

Assert that difference between trajectories \( x_1(t) \) and \( x_2(t) \) is less than accu. error \( A_{err} \)

\[
\int_{t_0}^{t_f} |x_1(t) - x_2(t)| \, dt < A_{err}
\]
Testing AD vs. FD

```java
model TestT1BioProcess
    import PSTools.Utilities.unitVector;

    Utilities.Validate dmu_dKs(
        AccErr=1E-1,
        name="test T1.Processions dmu / dKs");

        NG=1,
        g_Ks=unitVector(1, NG));

...
```
Visual unit testing

equation

\[
\text{dmu}_d\text{Ks}.T1 = \text{PSAD}.g\_\text{mu}[1]; \\
\text{dmu}_d\text{Ks}.T2 = \text{PSFD}.g\_\text{mu}[1];
\]

end \ TestT1\BioProcess;
Case Study 1 PSTools Library

UT_TestT1_FAIL

// der(VS) = -qS * VX;
g_VS[i] = -g_qS[i] * VX - qS * g_VX[i];

UT_TestT1_WORK

// der(VS) = -qS * VX;
der(g_VS[i]) = -g_qS[i] * VX - qS * g_VX[i];
\[
v = \prod_a \frac{K_{A_a} + [A_a]}{K_{A_a}} \cdot \prod_b \frac{K_{I_b}}{K_{I_b} + [I_b]} \cdot \frac{V^\text{fwd} \max \prod_i \frac{[S_i]}{K_{mS_i}} - V^\text{bwd} \max \prod_j \frac{[P_j]}{K_{mP_j}}}{V^\text{fwd} \max \prod_i 1 + \frac{[S_i]}{K_{mS_i}} + V^\text{bwd} \max \prod_j 1 + \frac{[P_j]}{K_{mP_j}} - 1}
\]
Formal unit testing procedure

1) a unit test model for each component C using
   - Example of a small model employing C
   - Equivalent model implemented only by equations

2) Execute all unit tests by *.mos & OMSHELL
   - after each significant modification

3) Investigate errors, if any

Most useful when the component checks but the UT does not translate or simulate

A library is shipped with its unit tests
All unit tests are within a package called UnitTests
Currently about 40 unit tests distributed GenKinetics
Case Study 2  ADGenKinetics Library

OMC 1.12.0

A = product({KA[i] / (KA[i] + mc_A[i].c) for i in 1:NA});

OMC 1.13.0

A = product({ (KA .+ mc_A.c) ./ KA for i in 1:NA});
Case Study 3  GenKinetics / Biochem Libraries

Numerical Problem

\[ \text{der}(c) = \text{if } (c < \text{tolerance}) \text{ then } 0 \text{ else } r_{\text{net}}; \]

Solution

\[ \text{der}(c) = \text{if } (c < \text{tolerance and } r_{\text{net}} < 0) \text{ then } 0 \text{ else } r_{\text{net}}; \]
Advantages

- Simplicity
- Purely Modelica
- Suitability
  - individual developers of Modelica libraries
    - PSTools
  - small-scale developments
  - Minimum constraints get done
  - Shipping Unit Tests with the product
Outlook

• a user experience report from GenKinetics

• Combine Relative Error w. Absolute Error


• Continuous Integration Solution