Using Modelica Models for Complex Virtual Experimentation with the Tornado Kernel

Filip H.A. Claeys  
Department of Applied Mathematics,  
Biometrics and Process Control (BIOMATH)  
Ghent University  
Coupure Links 653  
B-9000 Gent  
Belgium  
E-mail: fc@biomath.ugent.be

Peter Fritzson  
Programming Environments Laboratory (PELAB)  
Linköping University  
Campus Valla  
SE-581 83 Linköping  
Sweden  
E-mail: petfr@ida.liu.se

Peter A. Vanrolleghem  
modelEAU  
Département de génie civil  
Université Laval  
Pavillon Pouliot  
Québec, G1K 7P4  
QC, Canada  
E-mail: peter@modelEAU.org

Abstract

Tornado is a software kernel for virtual experimentation on the basis of ODE/DAE models. Recently, a model compiler has been developed that converts flat Modelica code to executable models suitable for use with the Tornado kernel. As a result, a subset of Modelica models can now be used for tasks such as parameter estimation, scenario analysis, Monte Carlo simulation, sensitivity analysis and steady-state analysis. The inherent computational complexity of the virtual experiment types implemented by Tornado can be efficiently handled by the kernel’s semi-automated distributed execution capabilities.

Keywords: Model compiler; Virtual experimentation; Tornado; Modelica

1 Introduction

Tornado [1] is an advanced kernel for modelling and virtual experimentation (i.e., any evaluation of a model such as simulation, optimization, scenario analysis, . . . ) that was recently jointly developed by BIOMATH (Ghent University) and HEMMIS N.V. (Kortrijk, Belgium). Although the kernel is generic in nature, it is mostly adopted in the water quality domain. In water quality research, the biological and/or chemical quality of water in rivers, sewers and wastewater treatment plants (WWTP) is studied. Research in this domain is facilitated by a number of models that have received a formal or de facto standardization status. Most notable are River Water Quality Model No.1 (RWQM1) [2] and the Activated Sludge Model (ASM) series [3]. Water quality models typically consist of large sets of non-linear Ordinary Differential Equations (ODE) and/or Differential-Algebraic Equations (DAE). These equations are mostly well-behaved, although discontinuities occur regularly. The complexity of water quality models is therefore not in the nature of the equations, but in the sheer number. In WWTP, smaller models such as the well-known Benchmark Simulation Model (BSM) [4] consist of approximately 150 derived variables. Larger systems have up to 1,000 derived variables and over 10,000 (partly coupled) parameters. On a typical workstation, a simulation run usually lasts minutes to hours.

The modelling language that has thus far been used in the scope of Tornado is MSL (Model Specification Language) [5]. This language is similar to Modelica [6] in the sense that it is high-level, declarative and object-oriented. In fact, both MSL and Modelica were designed according to the ideas resulting from the 1993 ESPRIT Basic Research Working Group 8467 on “Simulation for the Future: new concepts, tools and applications” [7]. Although similar in nature, MSL lacks some of the readability and expressiveness of Modelica. Therefore, it was decided to work towards inclusion of support for Modelica-based modelling in the Tornado framework.

The most recent result of our efforts to bridge the gap between Modelica and Tornado is a model compiler that converts flat Modelica (i.e., a Modelica model description that does not rely on inheritance nor decomposition) to executable models suitable for use with the Tornado kernel. At the moment, this compiler is a prototype that supports basic functionalities of the Modelica language. However, it does allow for a subset of Modelica models to be used in the context of Tornado. Since solutions already exist that generate flat Modelica from full Modelica (e.g., omc - the OpenModelica Compiler), only the conversion from flat Modelica to...
executable model code had to be implemented. The sequel of this paper is structured as follows: Section 2 and Section 3 respectively provide a further introduction to Tornado and its complex virtual experimentation capabilities. Subsequently, Section 4 explains how Modelica models can be used in Tornado. Section 5 discusses two simple Modelica models for which virtual experiments were run with Tornado. Finally, Section 6 contains some conclusions and references to future work.

2 Tornado

The Tornado kernel for modelling and virtual experimentation attempts to offer a compromise between the computational efficiency of custom hard-coded (typically FORTRAN or C) model implementations and the flexibility of less computationally efficient generic tools such as MATLAB. In Tornado, hierarchical models are specified in high-level, declarative, object-oriented modelling languages such as MSL [5] and - since recently - also Modelica. From these high-level specifications, efficient executable code is generated by a model compiler. Using the dynamically-loadable executable models generated by the model compiler, Tornado allows for running a variety of so-called virtual experiments. Virtual experiments are the virtual-world counterpart of real-world experiments, similar to the way models relate to real-world systems. A highly simplified conceptual diagram of Tornado is shown in Figure 1.

The Tornado kernel relies on a flexible input provider and output acceptor mechanism to deal with I/O for virtual experiments. Input can be provided by any combination of data files, internal data buffers and data generators. Output will be accepted by any combinations of data files, internal data buffers and plot handles (Note: since Tornado is merely a kernel, it does not have any data visualization interface of its own). In order to allow for the kernel to be deployed in a diverse array of applications, it has been equipped with multiple interfaces. Next to its native C++ interface, Tornado currently also has a C, .NET and MATLAB MEX interface (cf. Figure 2). The kernel is portable across platforms and was designed according to the three-tier principle. Most persistent representations of information types are XML-based. The grammar of these representations is expressed in XSD (XML Schema Definition) format and mimics very closely the internal representation of the respective types of information. An interesting feature of Tornado is the fact that it allows for dynamic loading of numerical solvers for tasks such as integration, optimization and Latin Hypercube Sampling. In order to support this principle, a generalized framework has been set up [8].

Several applications (graphical and other) can be built on top of Tornado. Examples include the next generation of the WEST® [5] commercial modelling and simulation tool for WWTP’s, its research-oriented counterpart named EAST and DHI’s MOUSE-TRAP (cf., http://www.dhigroup.com/Software/Urban.aspx). However, the most direct way of using the kernel is through the Tornado CUI (Command-line User Interface) suite, which is a comprehensive set of tools that is included with the kernel distribution. Full-fledged graphical applications such as WEST® are conceived to be used by all types of users (expert, intermediate, novice). The Tornado CUI suite however focuses on experts only. Table 1 gives an overview of the most commonly used command-line tools. The results dis-

![Figure 1: Tornado Conceptual Diagram](image1.png)

![Figure 2: Tornado-based Interfaces and Applications](image2.png)
3 Complex Virtual Experimentation

Tornado consists of strictly separated modelling and virtual experimentation environments. Virtual experiments can either be atomic or compound. The latter are hierarchically structured whereas the first cannot be further decomposed. Atomic experiment types that are available in Tornado are dynamic simulation and steady-state analysis. The most straightforward types of compound experiments are optimization, scenario analysis, Monte Carlo analysis (e.g. using Latin Hypercube Sampling) and sensitivity analysis. More convoluted types of compound experiments are also available, such as combinations of scenario / Monte Carlo analysis and optimization. Thanks to the object-oriented nature of Tornado, new virtual experiment types can easily be added. Several types of virtual experiments are based on the computation of objective values. As far as possible, the same set of objective types is available for each objective-based experiment type, thereby promoting orthogonality.

Given the hierarchical nature of compound virtual experiments, computational complexity can be substantial. Tornado therefore allows for coarse-grained gridification of certain types of compound virtual experiments. Supported distributed execution environments include BIOMATH’s Typhoon cluster software [9] and CERN’s LCG-2 grid middleware (cf., http://public.egee.org). Tornado generates generic job descriptions for dynamic execution. Typhoon is capable of directly interpreting these generic job descriptions, whereas for LCG-2, an additional conversion step has to be applied.

Using Tornado’s powerful complex virtual experimentation capabilities, large risk/cost/benefit analyses for integrated water systems were carried out, including Latin Hypercube Sampling from multi-dimensional parameter spaces and the automated execution of 1,000’s of simulations [10], each requiring an average of 0.5h of computation time.

4 Using Modelica Models in Tornado

In Tornado, executable models consist of two distinct parts. The first part is represented in C and is made up of the actual model equations, in addition to a number of flat arrays containing data containers for parameters and variables. The second part is a XML representation of meta-information, i.e., information regarding names, descriptions, units, constraints, ... of parameters, variables and models. The relationship between these hierarchically structured meta-information items and the respective elements of the flat C arrays is also expressed in XML. The availability of meta-information in executable models allows for the latter to be self-describing, which is a requirement given the strict separation between modelling and experimentation in Tornado.

In the Tornado framework, model compilers are to generate executable models in the format that was described above. The MSL model compiler that is part of the Tornado suite generates these executable models directly from MSL input. In the case of Modelica however, the approach is two-phased. During the first phase, the OpenModelica Compiler is used to generate flat Modelica (.mof) from full Modelica input (.mo). During the second phase, a new Tornado CUI tool called mof2t is used to convert flat Modelica to the Tornado executable model format. This approach was mainly inspired by practical considerations (lack of resources for the re-implementation of the non-trivial full-to-flat Modelica conversion). At the moment mof2t only supports a subset of flat Modelica.

For the development of the mof2t compiler, the same technologies and libraries were used as for the remainder of the Tornado framework, i.e., C++, flex/bison, and Elcel Technologies OpenTop (cf., http://www.elcel.com). The mof2t compiler nicely

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tbuild</td>
<td>Compiles and links executable model code to a dynamically-loadable binary object (.dll / .so)</td>
</tr>
<tr>
<td>tcreate</td>
<td>Creates an empty XML description of a virtual experiment</td>
</tr>
<tr>
<td>texec</td>
<td>Executes virtual experiments described in XML</td>
</tr>
<tr>
<td>tinitial</td>
<td>Dumps all model quantity values after initialization</td>
</tr>
<tr>
<td>tmso</td>
<td>Compiles a high-level MSL model to executable model code</td>
</tr>
<tr>
<td>tobj</td>
<td>Computes aggregation functions and other criteria from simulation trajectories</td>
</tr>
<tr>
<td>tproject</td>
<td>Manages sets of related experiments and connection graphs</td>
</tr>
<tr>
<td>tsort</td>
<td>Sorts a Tornado-generated data file</td>
</tr>
<tr>
<td>t2msl</td>
<td>Converts a connection graph to MSL code</td>
</tr>
</tbody>
</table>
completes the Tornado CUI suite, which in all consists of approximately 20 tools. The relationship between mof2t and the most important other CUI tools is depicted in Figure 3.

![Diagram](image)

Figure 3: Relationship between the Main Tornado CUI Tools

5 Examples

In this section, two simple cases are presented that illustrate the use of Modelica models in Tornado. The first case is based on the ubiquitous Van der Pol system, which is frequently found as an example in modelling and simulation textbooks when stiff systems are discussed. The second case is based on the ARGESIM-CI simulator comparison. In both cases, results were obtained using the Tornado CUI suite. Evidently, when using Tornado through one of the GUI applications that it supports, most of the technical details shown below are hidden from the user.

5.1 Van der Pol

5.1.1 Model

The Van der Pol system can be described in Modelica as follows:

```modelica
VanDerPol.mof:
---
| fclass VanDerPol |
| Real x(start = 1.0); |
| Real y(start = 1.0); |
| parameter Real mu = 1; |
| equation |
| der(x) = y; |
| der(y) = -x + mu * (1.0 - x * x) * y; |
| end VanDerPol; |
---
```

Given the fact that this model does not rely on inheritance nor decomposition, there is no difference between its full and flattened version. In order to generate executable code for Tornado and convert this code into a dynamically-loadable version, the mof2t and tbuild CUI tools are to be used:

```
> mof2t VanDerPol.mof
```

```
Flat Modelica to Tornado Convertor (Build: Jun 23 2006)
```

```
> tbuild -p win32-msvc7.1 VanDerPol
```

Tornado Model Builder (Build: Jun 23 2006)

The executable C code and XML meta-information that is generated by mof2t is as follows:

```c
VanDerPol.c:
---
#include <math.h>
#include <stdlib.h>
#include "Tornado/EE/Common/DLL.h"
#include "Tornado/EE/MSLE/MSLE.h"
#define _mu_ pModel->Params[0]
#define _time_ pModel->IndepVars[0]
#define _x_ pModel->DerVars[0]
#define _D_x_ pModel->Derivatives[0]
#define _y_ pModel->DerVars[1]
#define _D_y_ pModel->Derivatives[1]
void ComputeInitial(struct TModel* pModel) {} 
void ComputeState(struct TModel* pModel) 
{ 
    _D_x_ = _y_; 
    _D_y_ = -_x_ + _mu_ * (1 - _x_ * _x_) * _y_; 
} 
void ComputeOutput(struct TModel* pModel) {} 
void ComputeFinal(struct TModel* pModel) {} 
void* GetID() { return (void*)L"Tornado.MSLE.Model.VanDerPol"; }
void* Create() 
{ 
    struct TModel* pModel; 
    pModel = (struct TModel*)malloc(sizeof(struct TModel));
    pModel->Type = L"ODE";
    pModel->NoParams = 1;
    pModel->NoIndependentVars = 1;
    pModel->NoInputVars = 0;
    pModel->NoOutputVars = 0;
    pModel->NoAlgVars = 0;
    pModel->NoDesVars = 2;
    pModel->NoDerivatives = 2;
    pModel->NoPrevious = 0;
    pModel->NoResidues = 0;
    pModel->NoSolvedSets = 0;
    pModel->NoEvents = 0;
}
```

The Modelica Association 196  Modelica 2006, September 4th – 5th
The exact semantics of these representations are beyond the scope of this paper and will therefore not be further discussed. Important to note however is that the format of the generated C code has been kept as simple as possible in order to be able to compile the code with as many C compilers as possible. The compilers that have been shown to work so far are Borland C++ 5.5, MS Visual C++ 6.0, 7.1 & 8.0, LCC, INTEL C++ 9.0 and g++.

5.1.2 Simulation

In order to simulate the generated model code, a simulation experiment spec is to be provided to the simulation executor. Specs must conform to the respective XML schemas that have been defined in the scope of Tornado. When using the Tornado CUI suite, empty specs can be generated by the tcreate program and must then be further completed manually. Below is a simulation experiment spec for the Van der Pol model that was generated by invoking tcreate -t Simul VanDerPol and further completed through manual editing:

VanDerPol.Simul.Exp.xml:

```xml
<Exp Version="1.0" Type="Simul">
  <Time>
    <Start Value="-300"/>
    <Stop Value="+INF"/>
  </Time>
  <Solve>
    <Integ Method="CVODE">
      <Stop Value="300"/>
    </Integ>
  </Solve>
  <File Name="VanDerPol.Simul.out.txt" Enabled="true"/>
  <Plot Enabled="false"/>
  <CalcVars Enabled="false"/>
</Exp>
</Exp>
</Model>
</Models>
</Model>
</Symb>
</Tornado>
</Tornado>
</VanderPol.SymModel.xml

The Modelica Association  Modelica 2006, September 4th – 5th
For a simulation experiment, XML specs basically allow for specifying initial values (hereby overruling initializations that were specified through the model’s meta-information), defining input providers / output acceptors, specifying the simulation start / stop time and configuring integrator solver settings. In this case, initial values were given for \( \mu \) and \( .x \), input was disabled and one output file acceptor was defined. The simulation will be run from 0 to 300 and the CVODE stiff system solver (cf. http://www.llnl.gov/CASC/sundials) will be used as an integrator. Important to note is that the settings of the CVODE integrator are given through a flexible attribute-value pair mechanism instead of through tags that are part of the XML grammar. This is required to support dynamic loading of solver plugins. The fragment below shows the output of the experiment executor, when applied to the \texttt{VanDerPol} simulation spec.

One will notice that before execution starts, a number of solver plugins are dynamically loaded (in this case only a subset of the 35 solver plugins that are provided with Tornado are loaded):

```
> texec VanDerPol.Simul.Exp.xml
```

The Tornado CUI suite does not contain any data visualization mechanism, however one can easily use tools such as MS Excel, MATLAB or GNUPlot to display the simulated trajectories. Figure 4 shows the result of invoking the following commands in GNUPlot:

```
set xlabel "t"
set ylabel ".x"
plot 'VanDerPol.Simul.out.txt' using 1:2 with lines
```

Figure 4: Van Der Pol for \( .\mu = 100 \)

5.1.3 Parameter variation

More interesting it becomes if we wish to run the same simulation for different initial values. For instance, suppose we wish to run the simulation for values of \( .\mu \) that are logarithmically spaced between 1 and 100 with a spacing of 2. Suppose also that for each simulation, we want to determine the maximum value and
standard deviation of the trajectory of \( y \), in addition to the value of \( y \) at \( t = 50 \). The scenario analysis experiment shown below provides a solution to this problem:

As one can see, this scenario analysis spec refers to a simulation spec that resides in an external file (VanDerPol.Simul.Exp.xml). Directly embedding the XML content of this file into the scenario analysis experiment is however also possible. One will also notice that other types of objectives and aggregation functions (next to the \( 	ext{Min} \), \( 	ext{StdDev} \), \( 	ext{ValueOnTime} \) functions that are needed for our application) such as \( 	ext{Avg} \) (average) and \( 	ext{Int} \) (integral) are also possible. Table 2 shows the contents of the VanDerPol.Scen.out.txt file that is generated during the execution of the scenario analysis.

<table>
<thead>
<tr>
<th>RunNo</th>
<th>( \mu )</th>
<th>Max(.y)</th>
<th>StdDev(.y)</th>
<th>ValueOnTime(.y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2.6865596</td>
<td>1.4272097</td>
<td>-1.5137494</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3.8300373</td>
<td>1.465402</td>
<td>-0.03490599</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>6.3463996</td>
<td>1.5316644</td>
<td>0.58337344</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>11.553678</td>
<td>1.593544</td>
<td>-0.092195286</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>22.097001</td>
<td>1.6317544</td>
<td>0.058550465</td>
</tr>
<tr>
<td>6</td>
<td>32</td>
<td>43.305241</td>
<td>1.640437</td>
<td>0.052380761</td>
</tr>
<tr>
<td>7</td>
<td>64</td>
<td>85.82672</td>
<td>1.5529595</td>
<td>-0.0439915</td>
</tr>
<tr>
<td>8</td>
<td>100</td>
<td>133.68028</td>
<td>1.4986709</td>
<td>-0.010200556</td>
</tr>
</tbody>
</table>

5.2 ARGESIM - C1

ARGE Simulation News (cf., http://www.argesim.org) is a non-profit working group providing the infrastructure and administration for dissemination of information on modelling and simulation in Europe. ARGESIM is located at Vienna University of Technology, Dept. Simulation and publishes Simulation News Europe (SNE), which features a series on comparisons of simulation software. Based on simple, easily comprehensible models special features of modelling and experimentation within simulation languages, also with respect to an application area, are compared. Features are, for instance: modelling technique, event-handling, numerical integration, steady-state calculation, distribution fitting, parameter sweep, output analysis, animation, complex logic strategies, submodels, macros, statistical features etc. Approximately 20 comparisons have thus far been defined, the first was
published in November 1990, the last in December 2005.

5.2.1 Model

As a second example of the use of Modelica models in Tornado, the CI ARGESIM comparison will be used.

The model that is at the basis of this comparison can be represented in Modelica as follows:

```modelica
fclass C1
parameter Real kr = 1;
parameter Real kf = 0.1;
parameter Real lf = 1000;
parameter Real dr = 0.1;
parameter Real dm = 1;
parameter Real p = 0;
Real f(start = 9.975);
Real m(start = 1.674);
Real r(start = 84.99);
equation
  der(r) = -dr * r + kr * m * f;
  der(m) = dr * r - dm * m + kf * f * f - kr * m * f;
  der(f) = dr * r + 2 * dm * m - kr * m * f -
          2 * kf * f * f - lf * f + p;
end C1;
```

The comparison requires the following tasks to be performed:

- **Simulation of the stiff system over [0,10].**
- **Parameter variation of lf from 1.0e2 to 1.0e4 and a plot of all f(t; lf), logarithmic steps preferred.**
- **Calculation of steady states during constant bombardment (p(t) = pc = 1.0E4) and without bombardment (p(t) = 0).**

5.2.2 Simulation

As in the first example, a dynamically-loadable executable model for Tornado can be generated using mof2t and tbuild. Afterwards, an empty simulation experiment can be generated with tcreate and then be completed through manual editing:

```xml
<Tornado>
  <Exp Version="1.0" Type="Simul">
    <Props>
      <Prop Name="Author" Value="PCFC1.fc"/>
      <Prop Name="Date" Value="Fri Jun 30 12:28:12 2006"/>
      <Prop Name="Desc" Value=""/>
      <Prop Name="FileName" Value="C1.CVODE.Simul.Exp.xml|.Tornado"/>
      <Prop Name="UnitSystem" Value=""/>
    </Props>
    <Simul>
      <Model Name="C1" CheckBounds="false">
        <Quantities/>
      </Model>
      <Inputs Enabled="false"/>
      <Outputs Enabled="true">
        <Output Name="*Calc*">
          <CalcVars Enabled="false"/>
        </Output>
        <Output Name="*Plot*">
          <Plot Enabled="false">
            <Props>
              <Prop Name="CommInt" Value="0"/>
              <Prop Name="CommIntType" Value="Logarithmic"/>
              <Prop Name="ConvergenceControlType" Value=""/>
              <Prop Name="Interpolated" Value="false"/>
              <Prop Name="StopTime" Value="+INF"/>
              <Prop Name="UseDisplayUnits" Value="true"/>
            </Props>
            <Quantities/>
          </Plot>
        </Output>
        <Output Name="File">
          <File Name="C1.Simul.out.txt" Enabled="true">
            <Props>
              <Prop Name="CommInt" Value="1.2"/>
              <Prop Name="CommIntType" Value="Logarithmic"/>
              <Prop Name="Precision" Value="8"/>
              <Prop Name="StartTime" Value="-INF"/>
              <Prop Name="StopTime" Value="+INF"/>
              <Prop Name="UseDisplayUnits" Value="true"/>
            </Props>
            <Quantities>
              <Quantity Name=".f"/>
              <Quantity Name=".m"/>
              <Quantity Name=".r"/>
            </Quantities>
          </File>
        </Output>
      </Outputs>
      <Time>
        <Start Value="0"/>
        <Stop Value="10"/>
      </Time>
      <Solve>
        <Integ Method="CVODE">
          <Props>
            <Prop Name="AbsoluteTolerance" Value="1e-006"/>
            <Prop Name="CVBandLowerBandwidth" Value="0"/>
            <Prop Name="CVBandUpperBandwidth" Value="0"/>
            <Prop Name="CVSPGMRGSType" Value="ModifiedGS"/>
            <Prop Name="IterationMethod" Value="Functional"/>
            <Prop Name="LinearMultistepMethod" Value="Adams"/>
            <Prop Name="LinearSolver" Value="Dense"/>
            <Prop Name="RelativeTolerance" Value="1e-006"/>
          </Props>
        </Integ>
        <Root Method="Broyden">
          <Props>
            <Prop Name="MaxNoSteps" Value="0"/>
            <Prop Name="MaxStepSize" Value="1"/>
          </Props>
        </Root>
      </Solve>
    </Simul>
  </Exp>
</Tornado>
```

Important to notice in this simulation experiment is that for the output file acceptor, the communication interval type (CommIntType) was set to logarithmic. In this case, logarithmic spacing of output timepoints is required in order to be able to accurately represent the dynamics of the simulated trajectories during the initial phase of the simulation (without generating huge amounts of irrelevant data). After running the simulation with texec, one could for instance use GNUPlot to display the results (see Figure 5) onto logarithmic axes using the following commands:

```bash
set logscale xy
set xlabel "t"
set ylabel ".f"
plot 'C1.Simul.out.txt' using 1:2 with lines
```

5.2.3 Parameter variation

The parameter variation that is requested by the comparison can easily be implemented in Tornado us-
The following describes a steady-state experiment for the ARGESIM C1 model where \( p = 1e4 \):

```plaintext
C1.p=1e4.SS.Exp.xml:
```
```
Tornado
<Exp Version="1.0" Type="SS">
<Props>
<Prop Name="Author" Value="PCFC1\fc"/>
<Prop Name="Date" Value="Fri Jun 30 15:39:27 2006"/>
<Prop Name="Desc" Value=""/>
<Prop Name="Filename" Value="C1.p=1e4.SS.Exp.xml\|.Tornado"/>
<Prop Name="UnitSystem" Value=""/>
</Props>
<SS>
<Model Name="C1" CheckBounds="false">
<Quantities>
<Quantity Name=".p" Value="1e4"/>
</Quantities>
</Model>
<Solve>
<Root Method="Hybrid">
<Props>
<Prop Name="Tolerance" Value="1e-008"/>
</Props>
</Root>
</Solve>
</SS>
</Exp>
```

Execution of this experiment with `texec` will instantly yield the correct steady state values for \( f, m \) and \( r \):

```plaintext
> texec "C1.p=1e4.SS.Exp.xml"
```

5.2.4 Calculation of steady states

For the calculation of steady states, the steady-state (SS) experiment type can be used. In Tornado, the steady-state of a system is directly computed through the application of a root finding solver to the system equations, where the derivatives (i.e., the left hand sides) of state equations are used as residues (that are to be brought to zero).

Figure 5: Simulation results for the ARGESIM C1 model

Figure 6: Scenario analysis results for the ARGESIM C1 model

5.2.4 Calculation of steady states

For the calculation of steady states, the steady-state (SS) experiment type can be used. In Tornado, the steady-state of a system is directly computed through the application of a root finding solver to the system equations, where the derivatives (i.e., the left hand sides) of state equations are used as residues (that are to be brought to zero).
For \( p = 0 \), one can proceed in a similar way. However, in this case the process is more sensitive to the initial value of the state variables. The experiment below therefore shows that for \( f \), a differing initial value had to be chosen to ensure convergence of the algorithm.

In the forthcoming months, the \textit{mof2t} will be further stabilized and enhanced.

### Acknowledgement

Peter A. Vanrolleghem is Canadian Research Chair in Water Quality Modelling.

### References


