OpenModelica Solvers

Anton de Villiers*

June 8, 2016

1 Introduciton

An ordinary differential equation (ODE) is an equation that involves some ordinary derivatives (as opposed to partial derivatives) of a function. Often, our goal is to solve an ODE, *i.e.* determine what function or functions satisfy the equation. Implementations of solving ODEs generally involves numerical methods of approximating future state variables, since symbolic approaches may quickly become infeasible.

The quantized state systems (QSS) [5] methods are a family of numerical integration solvers based on the idea of state quantization, dual to the traditional idea of time discretization. Unlike traditional numerical solution methods, which approach the problem by discretizing time and solving for the next (real-valued) state at each successive time step, QSS methods keep time as a continuous entity and instead quantize the system's state, instead solving for the time at which the state deviates from its quantized value by a quantum.

They can also have many advantages compared to classical algorithms. They inherently allow for modeling discontinuities in the system due to their discrete-event nature and asynchronous nature. They also allow for explicit root-finding and detection of zero-crossing using explicit algorithms, avoiding the need for iteration, a fact which is especially important in the case of stiff systems, where traditional time-stepping methods require a heavy computational penalty due to the requirement to implicitly solve for the next system state.

In view of HealthQ's requirements, a Linearly Implicit QSS (LIQSS) [5] numerical solver is to be developed and incorporated within the OpenModelica [3] framework.

2 Current state of work

At the moment we have a fully functioning version of a basic LIQSS solver in OpenModelica. The implementation is by no means in a finalized state of operation. There are some known issues that still needs to be addressed and some implementation issues, in OpenModelica, that are being discovered throughout this project.

To incorporate the LIQSS solver, the entire OMCompiler [3] submodule was cloned into a local repository (repo) for development. OMCompiler is coded in C, accompanied by some C++ as well as a scripting language unique to OpenModelica's puposes called Susan Templating. The entire repo is necessary as

*HealthQ Technologies, Office 9, First Floor, The Woodmill Lifestyle, Vredenburg Road, Devon Valley, Stellenbosch, 7600, South Africa

investigation was performed into many different areas of the code within this repo. A new LIQSS solver was added, based on the initial design of the already implemented QSS solver. The solver was tested on a variety of smaller models to ensure that the LIQSS solver would be able to deal with the individual components in isolation. The core test are around the simulation of Valve's (IdealValve's and RegularizedIdealValve's) in the OpenModelica context. Furthermore, the Pulse was also tested. The short-term goal is to successfully simulate the Danielsen Ottesen (Handbook) model. Since the Handbook model contains most of the components that the Virtual Human Model (VHM) contains, the current aim is that the LIQSS solver should work on the VHM if the Handbook model can be simulated successfully.

To compare our numerical results of ODE systems obtained by the LIQSS solver, we compare to our "gold-standard" algorithm DASSL [6], which is known to deliver expected numerical results. The issues generally arise when solving for certain events as well as very stiff mathematical systems. The problems are often centered around C code implementation details or the manner in which events are coded in OpenModelica.

Valve

Table 2.1 contains the graphical results of the various Valve models under consideration. The DASSL solver can simulate all of these models accurately. The LIQSS solver can solve the model with no valve, as well as the Regularized Valve model given that the step size specified in the solver is small enough. The ideal Valve model cannot be solved by LIQSS in its current form. Instead a different approach is required whereby certain small positive values are tweaked to ensure that the valve closes at the appropriate times. This issue is an implementation consideration that exists with numerical specification in both the LIQSS solver and OpenModelica.

3 Future implementation tasks

Current outstanding tasks related to improving the current version of the LIQSS solver is hosted on an Asana account [1]. These improvement mainly align with optimizing the code. To summarize some of the most relevant and important outstanding tasks:

- To only evaluate the "affected" ODEs during each iteration.
- Understanding the event handling process in OpenModelica.
- Remove unnecessary ODE function calculations.
- Incorporating personalized nominal values for all state variables.
- Ensuring increases in step sizes are reasonable, not too small and not too large.
- Investigate various manners of incorporating nominal values in calculating ΔQ_j for each state variable j.

The most recent issues are centered around scripting a manner of flattening the handbook model and adding nominal values.

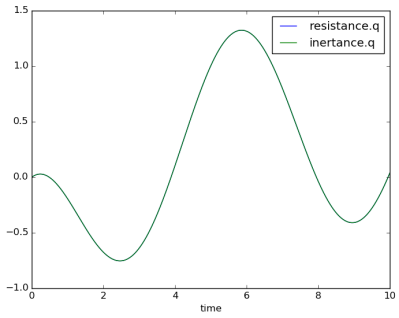
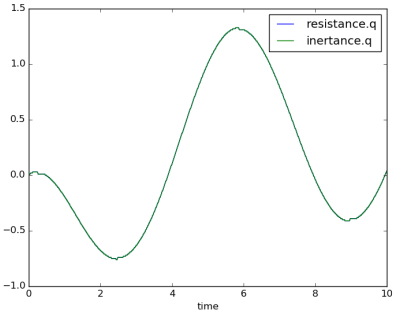
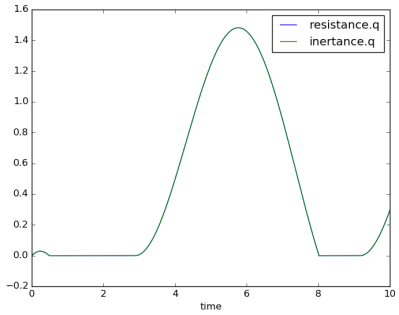
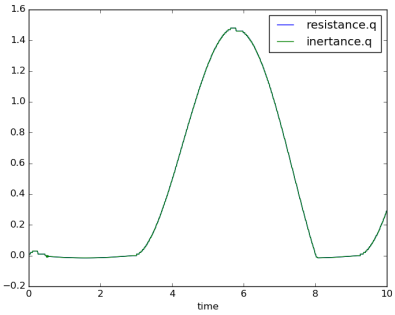
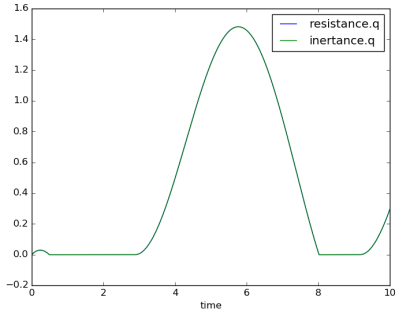
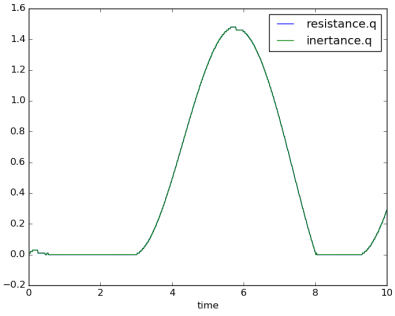
Valve	DASSL	LIQSS
1		
2		
3		

Table 2.1: The graphical results of the Valve simulations. 1) The Base model with no Valve. 2) The Ideal Valve. 3) The Regularized Valve.

4 Impediments

The main impediment centers around implementations on higher order LIQSS solvers. For an n-th order LIQSS solver, we require the symbolic equations of the n-th order derivatives of the ODE system. The initial belief was that one could approximate higher order derivatives numerically, but these results may easily become substantially inaccurate as the first order derivatives are based on approximate state variable assignment, which is then in turn used to approximate a higher order derivative value. The effect of the approximation values is thus compounded.

Instead, QSS Solver [7] used the C++ library to symbolically derive functions and thus obtaining higher order ODEs. This library is called GiNaC [2]. Although this library has its limitations, it can correctly derive a large subset of function, and by initial guesses it will be able to process the ODE system of the VHM.

The impediment is then to incorporate GiNaC into the OpenModelica framework to successfulle derive functions and to the incorporate the derived function in the OpenModelica numerical solvers. This may become a major undertaking as this will essentially entail making notable structural alterations to OpenModelica (and not only the OMCompiler submodule).

5 The envisioned finished product

The finished product should be able to simulate the VHM in its entirety. The aim is to have an ODE solver (LIQSS or a higher order version) that can solve the system of ODEs faster than DASSL, while still maintaining accurate numerical results of the ODE system consideration.

References

[1] Asana, 2016, *Asana task management tool*, [Online], Cited 15th March 2016, Available from `https://app.asana.com`

[2] GiNaC, 2016, *GiNaC is not a CAS*, [Online], Cited 15th March 2016, Available from `http://www.ginac.de/`

[3] OpenModelica, 2016, *Open Source Modelica Consortium*, [Online], Cited 15th March 2016, Available from `https://openmodelica.org/`

[4] OMCompiler, 2016, *HealthQ OMCompiler submodule repository*, [Online], Cited 15th March 2016, Available from `https://bitbucket.org/antonpdv/omcompiler`

[5] Migoni G & Fofman E, 2009, *Linearly implicit discrete event methods for stiff ODE's*, Latin American Applied Research, **39(3)**, pp. 245–254.

[6] Petzold LR, 1982, *A description of DASSL: A differential/algebraic system solver*, Technical Report, Applied Mathematics Division, Sandia National Laboratories, Livermore (CA).

[7] QSS Solver, 2016, *Modeling and simulation tool for continuous and hybrid systems*, [Online], Cited 15th March 2016, Available from `https://sourceforge.net/projects/qssengine/`