# Code Generation for Quantized-State System (QSS) Simulation

Mehrdad Niknami

## Background

- Continuous-time simulation technique
- "Dual" to classical algorithms
  - Classical: discrete time, continuous state
  - QSS: continuous time, quantized state
- Error for LTI systems provably bounded above by constant
  - No such result for discrete-time methods
  - Downside: error also bounded *below* by quantum
- Efficiency for stiff systems
  - Slower dynamics results in less computation
  - Computation goes into where it is needed most

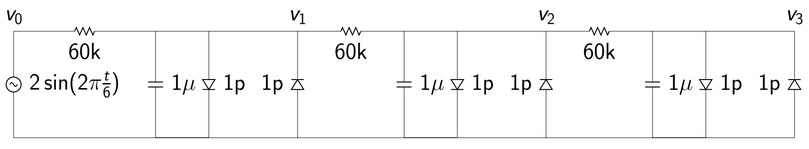
## Contributions [in-progress]

- Code-generator front-end in Java
  - Based on JModelica solver engine
  - Translates to C++ simulator, but easily portable
- QSS simulator backend in C++
  - Discrete-event model
  - Deterministic
- Instantaneous derivative propagation
  - New idea; apparently previously unexplored
  - Seems to improve efficiency in practice
  - No theoretical justifications yet

## Limitations and future work

- Discontinuities not yet working (but close!)
- QSS2+ handling not quite proper
  - Unlike QSS1, QSS2+ cannot be simulated exactly
  - "Most correct" semantics unclear
- Extremely inefficient handling of conditionals
  - Both branches always evaluated, only one needed
  - Some parallelism at very high cost
  - Sparsity analysis of DAG required (NP-hard? unsure)

We compare the behavior of QSS and classical solvers in the following nonlinear example:




(a) Example nonlinear circuit: a three-component *distributed-element model* of a transmission line.

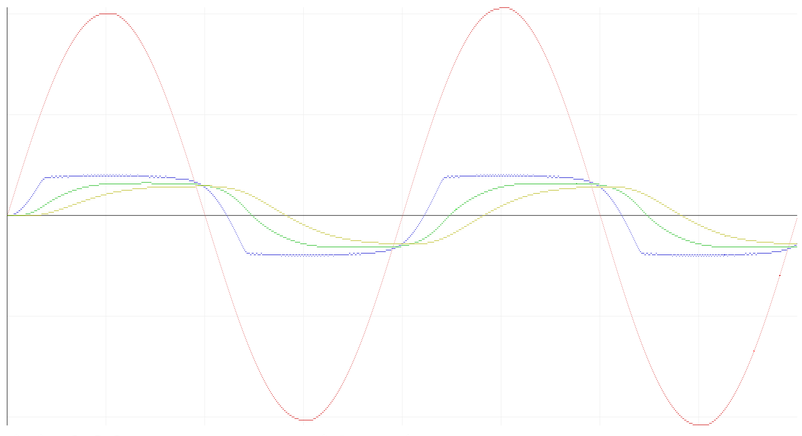
The code generator converts the following Modelica model of the transmission line:

```
model wire_model
    "Distributed-element model of a transmission line."
    constant Real PI = 3.14159265358979323846264338 3,
        A = 2, F = 0.25, vT = 0.026, c = 10^(-6),
        R = 100000/F, i_s = 10^(-12),
        dv0_init = 2 * PI * F * A;
    Real
        v0(start = 0), dv0(start = dv0_init),
        v1(start = 0), v2(start = 0), v3(start = 0);
initial equation
equation
    der( v0) = dv0;
    der(dv0) = -((2*PI*F) * (2*PI*F)) * v0;
    der( v1) = -((v1-v0)/R+i_s*(exp(+v1/vT)-exp(-v1/vT)))/c;
    der( v2) = -((v2-v1)/R+i_s*(exp(+v2/vT)-exp(-v2/vT)))/c;
    der( v3) = -((v3-v2)/R+i_s*(exp(+v3/vT)-exp(-v3/vT)))/c;
end wire_model;
```

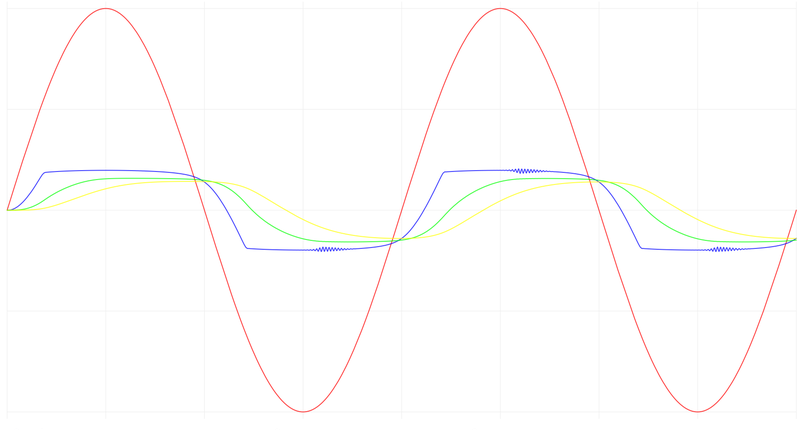
...into the following C++ code:

```
Time const horizon = 100;
Network network;
HystereticQuantizer v0;
Integrator v0_i; v0_i.initial(0);
HystereticQuantizer dv0;
Integrator dv0_i; dv0_i.initial(3.141592653589793);
HystereticQuantizer v1;
Integrator v1_i; v1_i.initial(0);
...
network.connect(real_2.output(), mul_1.left());
network.connect(v0.output(), mul_1.right());
network.connect(neg_4.output(), div_3.left());
...
QSSSimulator sim(network);
sim.order(1);
sim.quantum(1);
for (sim.init(); sim.subsequent() <= horizon; sim.step())
{ ... }
```

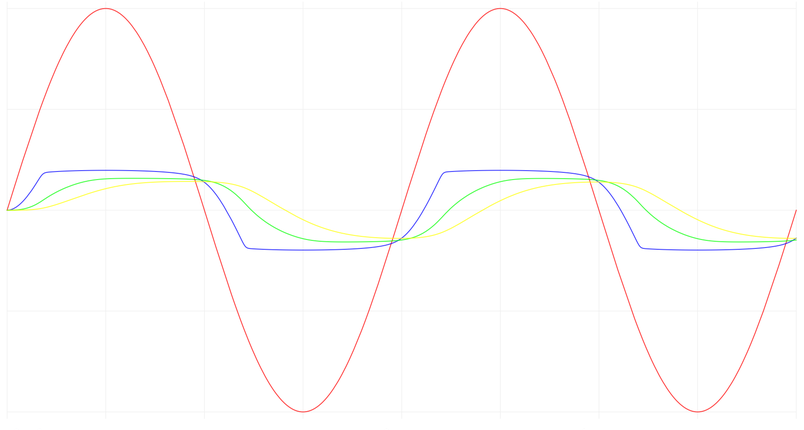
All we then need to do is to manually connect each voltage to an "oscilloscope" (`TimedPlotter`) in order to view the solution trajectory:



(a) QSS1 simulation (C++) with quantum $\Delta Q = 2^{-6}$. Notice that the output voltages clip rather accurately but the input sinusoid signal also diverges gradually, suggesting that the method is accurate but unstable.

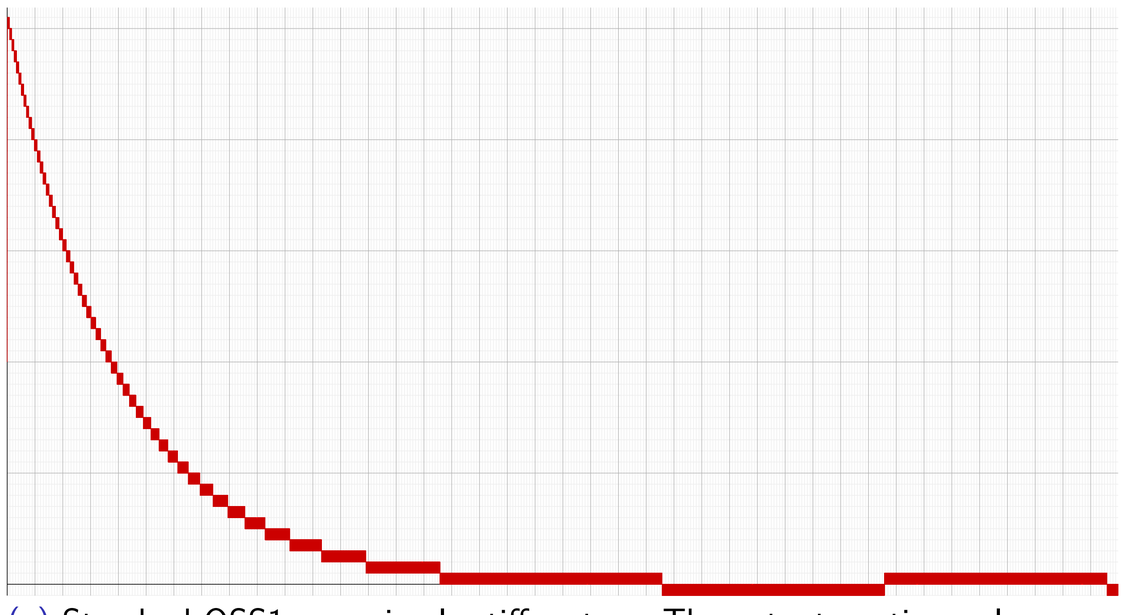


(b) Explicit Euler (Mathematica) simulation with step size $\Delta t = 2^{-6}$. Notice that this basic method is more stable than QSS1, but much less qualitative accurate mid-cycle.

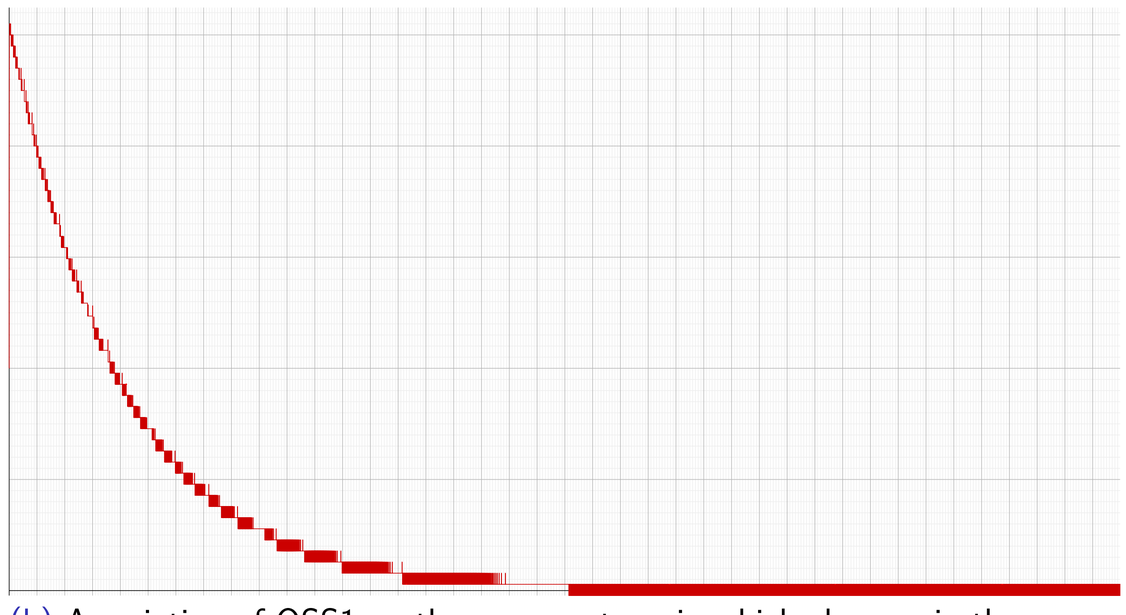


(c) Explicit Runge-Kutta (Mathematica) simulation with step size $\Delta t = 2^{-6}$. While an explicit method and potentially unstable, this algorithm faithfully simulates the dynamics of this system.

**Variation: Derivative Propagation** Integrators in standard QSS always wait for a change of one quantum before propagating propagating their outputs. However, propagating this information instantaneously seems to decrease the number of events in the long term, allowing for a potential gain in simulation speed. Unfortunately, we do not yet have any theoretical justifications for this approach.



(a) Standard QSS1 on a simple stiff system. The output continuously oscillates between two values, generating a large number of events and representing a source of inefficiency in the simulation.



(b) A variation of QSS1 on the same system, in which changes in the derivatives of signals are propagated instantaneously. The output initially oscillates between two values, but eventually settles down, suggesting that asymptotically, this might increase the simulation efficiency.