Three Numerical Reproducibility Issues That Can Be Explained As Round-Off Error

Prof. Michael Mascagni

Department of Computer Science
Department of Mathematics
Department of Scientific Computing
Graduate Program in Molecular Biophysics
Florida State University, Tallahassee, FL 32306 **USA**
AND
Applied and Computational Mathematics Division, Information Technology Laboratory
National Institute of Standards and Technology, Gaithersburg, MD 20899-8910 **USA**
E-mail: mascagni@fsu.edu or mascagni@nist.gov
URL: http://www.cs.fsu.edu/∼mascagni

Outline of the Talk

Reproducibility

- We assume that computations are completely deterministic and hence inherit the following characteristics
  1. A computation is or can readily be made reproducible
     - Reproducible over time: redoing a computation
     - A computation used for publication can be documented sufficiently to be reproduced
     - Some list of items suffices for reproducibility: code, compiler options/version, code input, execution environment, ...
  2. A code correctly constructed is portable between architectures by virtue of the abstraction provided by the programming language and compilation process
  3. Computations that are not reproducible are deficient in some way(s)
- Reproducibility has become an issue that is being addressed at many levels and in many different ways: CRE2019 at SC19 in Denver

A Reproducibility Story

- We investigated a small network of excitable neurons in: Tabak, Mascagni, Bertram, *Journal of Neurophysiology*, **103**: 2208–2221, 2010.
  1. A network of 100 simple neurons with all-to-all connectivity
     - Each neuron is modeled as single compartment via an ordinary differential equation (ODE) system for the voltage
     - At each time step, the input for each neuron was accumulated by summing up the output of each neuron times the synaptic strength
     - The output was an average of the output of the 100 neurons and perhaps the average of an internal variable
  2. The neurons were all excitatory to simulate early neuronal development
  3. These types of models are used often in neuroscience, and are the bread and butter of computational neuroscience

A Colleague Uses Our Model

- A colleague wanted to use our model, and so he decided to play around with code to see if he understood the model

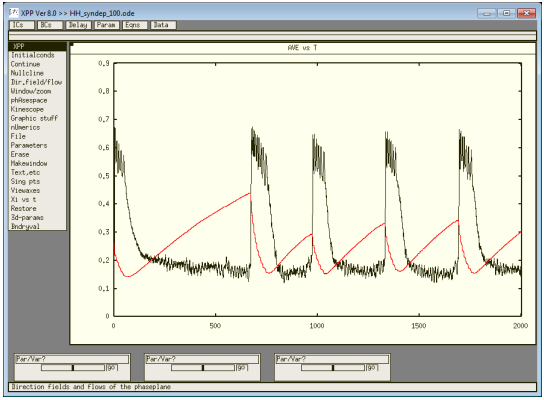

Figure: A Computation with the Original Code Up To 2000 Milliseconds

A Colleague Uses Our Model

- He used the code to define the same problem a second way, that was mathematically the same ODE system

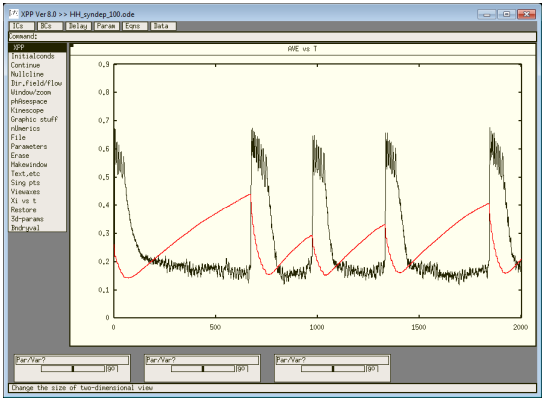

Figure: A Computation with the Code Up To 2000 Milliseconds

Same Code, Different Invocation

- The first figure is created with the following:

```
a.out -case 21 -nBurst 50 -pExcN 1.0 -vInh 70
```

- The second with:

```
a.out -case 21 -nBurst 50 -pExcN 0.8 -vInh 70
```

- The difference is that we identify the last 20 neurons as inhibitory, not excitatory, but we keep the neuronal characteristics the same
- The result is that the synaptic summations done at each time step are done in a slightly different order

The Code Differences Are Minor

- Here are the summation loops:

```
for (int i = 0; i < 10; i++)
        // Total synaptic drive exc/inh for all cells
        atotExc = atotInh = 0.0;

// Synaptic drive from excitatory cells
// atotexc=sum(0,100)of(shift(s0,i')*shift(a0,i'))/100
// remembering the order: v, n, a, s -> 0, 1, 2, 3
for ( n = 0; n < nExcNeurons; n++) {
        atotExc += network[n][3]*network[n][2];

// Synaptic drive from inhibitory cells
for ( n = nExcNeurons; n < nNeurons; n++) {
        atotInh += network[n][3]*network[n][2];
```

Non-reproducibility: Floating-Point Error Accumulation

- The two loops listed above compute the synaptic input to all the neurons by summing up the product of synaptic strength and neuronal activity
- The difference in the two runs is that the last 20 neurons are identified as inhibitory
- The input file describing the neurons and synapses was the same, and so the only difference was that the summation was broken up into two parts in the second case
  1. Thus the sum is the same, but it is done in a slightly different order
  2. Two sums have their orders swapped from the first to the second
- This small difference in summation results in different floating-point errors accumulating, and that manifests itself in long-time differences

Non-reproducibility: Floating-Point Error Accumulation

- Summation of real numbers is associative (order can be permuted), floating-point is not
- Summation is a classic example used to study mitigation of floating-point errors
  1. Divide the sum into positive and negative terms
  2. Accumulate the two sums with summands in decreasing absolute value
- There are other mitigation strategies that have been proposed for summation
  1. Do all accumulations in an extremely long register (Acrith, ARM)
  2. Simulate a long accumulator (Boost)
  3. Incorporate stable summation
- We have used Boost to test if this mitigates
- There are other also more general proposals
  1. Modernization of floating-point arithmetic (Demmel)
  2. Replace floating-point with commuting alternative

Recomputing The Code Using Boost Multiprecision

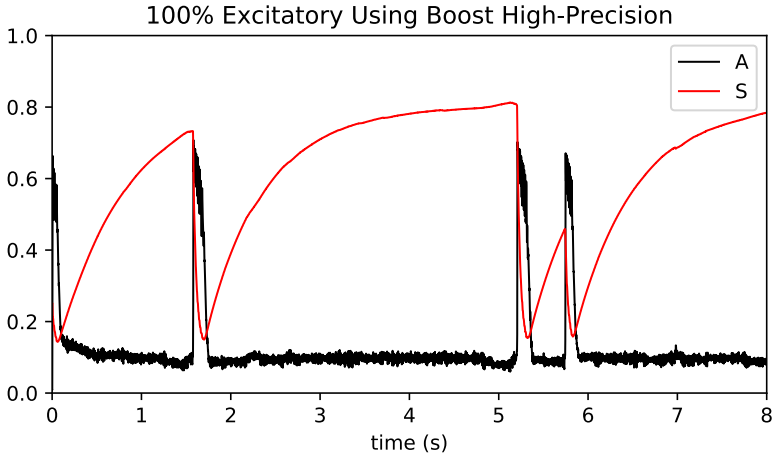

Figure: Computation of All Excitatory Neurons Using Boost

Recomputing The Code Using Boost Multiprecision

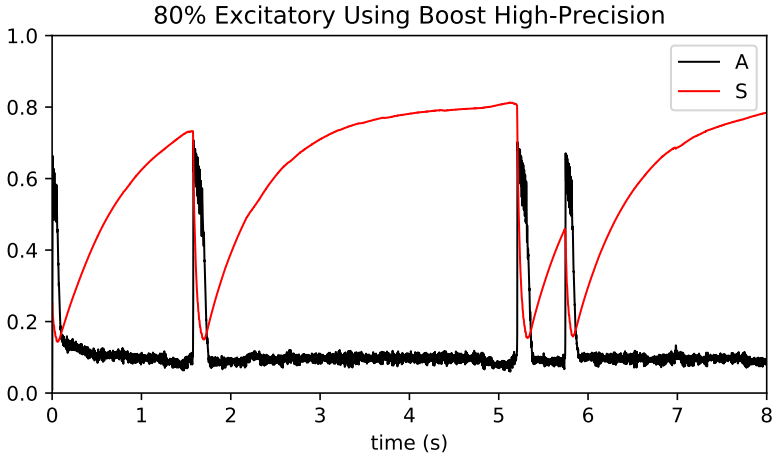

Figure: Computation of Excitatory/Inhibitory Neurons Using Boost

What Effect Does Temporal Refinement Have?

- A colleague who works on uncertainty quantification asked me if I had done a temporal refinement study: I had not
- This is a time evolution problem, and we use a standard time differencing scheme
  1. Theory says this is a stable numerical method under certain circumstances
  2. So as $\Delta t \to 0$ the numerical solution should converge
- We have already seen that this system is extremely sensitive to accumulated round-off errors
- What is the trade off between numerical convergence and the the effect of increasing the number of time steps ($N^{-1} \approx \Delta t \to 0$) and thus increasing the round-off
- We use a different plot to show the system activity, since it acts more like a fingerprint

Results From Temporal Refinement

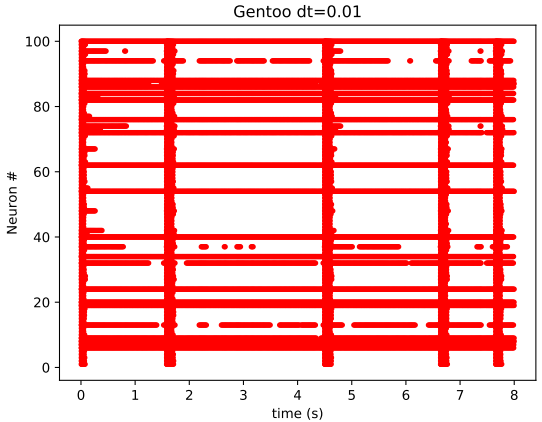

Figure: A Computation with Code Showing Neuron Activity vs. Time

Results From Temporal Refinement

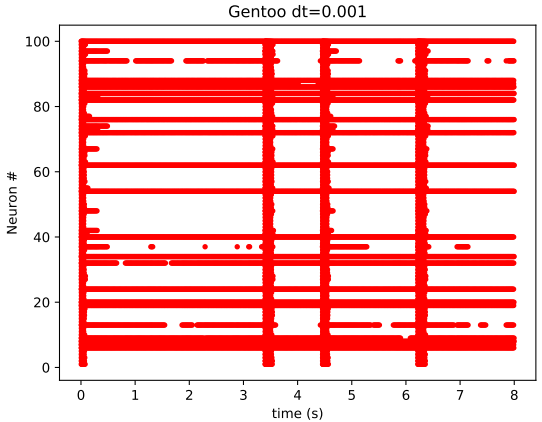

Figure: A Computation with Code Showing Neuron Activity vs. Time

Temporal Refinement Also Causes Issues

- We see that as $\Delta t \to 0$ we do not get convergence
- We only refined to $\Delta t = 0.0001$
  1. We know that these systems are stiff, and this is something that occurs even in convergent systems
  2. We were interested in exploring the sensitivity of this mathematical model
- There is a round-off based explanation
  1. When we take $N^{-1} \approx \Delta t \to 0$ the number of time steps increases
  2. Integration up to a fixed time with Δt reduced by a factor of 10 requires 10 times as many time steps
  3. Each time step generates round-off error of a certain size
  4. The introduction of extra round-off may perturb the solution from the converged result

Do Any Other Execution Changes Cause the Issues?

- The code is written in very generic `C++`, and so we wondered if we could compile and execute the code on different environments and see the same behavior
- We decided to compile and run it on the following systems
  1. Windows/Cygwin; `g++ (GCC) 5.4.0`
  2. Mac OS X: `Apple LLVM version 6.0 (clang-600.0.56) (based on LLVM 3.5svn)`
  3. Mint Linux: `g++ (Ubuntu 5.4.0-6ubuntu1 16.04.4) 5.4.0 20160609`
  4. Gentoo Linux `g++ (Gentoo 5.4.0-r3 p1.3, pie-0.6.5) 5.4.0`
- Cygwin and the two versions of Linux used "the same" version of `g++`

Results From Execution Under Windows/Cygwin

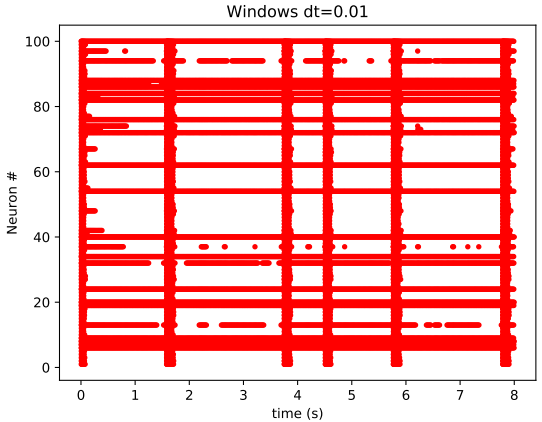

Figure: A Computation with Code Showing Neuron Activity vs. Time in Windows

Results From Execution Under Mac OS

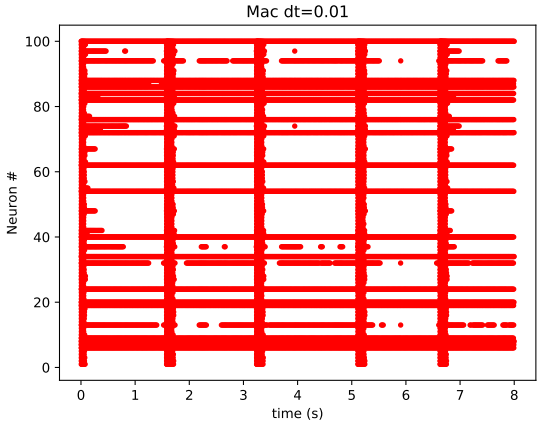

Figure: A Computation with Code Showing Neuron Activity vs. Time in Mac OS

Results From Execution Under Linux Mint

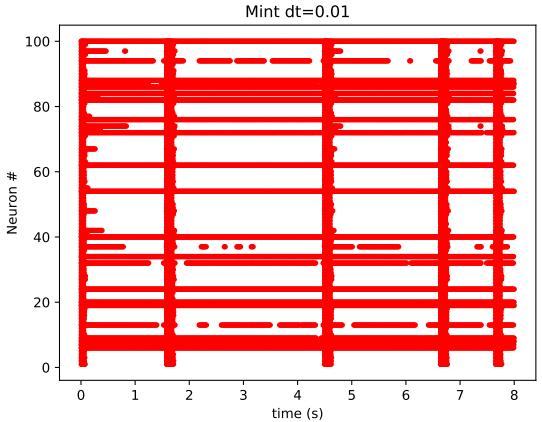

Figure: A Computation with Code Showing Neuron Activity vs. Time in Linux Mint

Results From Execution Under Linux Gentoo

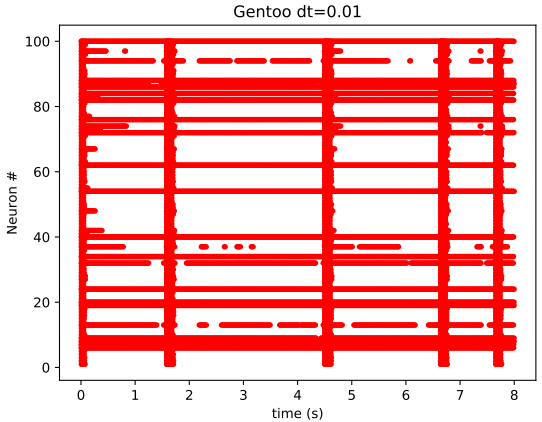

Figure: A Computation with Code Showing Neuron Activity vs. Time in Linux Gentoo

What Was the Actual Cause of the OS Differences?

- The two Linux results were exactly the same while execution under Linux, Cygwin, and Mac OS all gave different results
- We controlled for the compiler as much as possible (Linuxes-Cygwin vs. Mac)
- We noticed that the source code only used two mathematical functions in the `C++` library: `pow` and `exp`
  1. Decided to isolate the code used by the system to compute `pow` and `exp`
  2. Removed the references to the `C++` library versions
  3. Replaced the library calls to the included code for `pow` and `exp`
- We took the Cygwin version of `pow` and `exp` and used it to compile a new version of the code on Gentoo Linux

Results From Execution Under Linux Gentoo/Windows

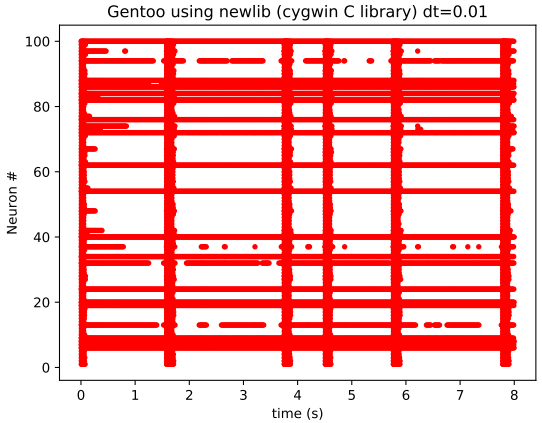

Figure: A Computation with Code Showing Neuron Activity vs. Time in Linux Gentoo

Builtin Mathematical Functions Can Cause Reproducibility Issues!!

- The replacement of `pow` and `exp` caused the Gentoo Linux version of the computation to revert to that previously obtained using Cygwin
  1. The actual algorithms or their implementations of `pow` and `exp` must differ to some extent between Cygwin and Gentoo
  2. Switching between the implementations gives results consistent with different OSes
- The effect of these differences in `pow` and `exp` can be thought of as producing the correct answers with slightly different amounts of round-off error
- As with the temporal refinement, the accumulation of those errors are enough to perturb this system
- We were very lucky to have correctly surmised that the mathematical functions were at fault, and that it was relatively easy to check it

Conclusions

- We have a found a system of ODEs used in neuronal modeling that is extremely sensitive to differences that create challenges for reproducibility
  1. The system is typical of models used in neuroscience
  2. This observation should help others in the field be more careful with results from similar mathematical models
- We observed differences in computed results due to
  1. Round-off caused by summation order change
  2. Change in the number of time steps taken to a given time
  3. The use of different language-based mathematical functions
- Given that reproducibility is not that easy to achieve in some cases
  1. Is there a way to determine if two computations are consistent given the various causes of non-reproducibility?
  2. Can we broaden the notion of reproducibility in a way that allows for scientific computational validation?
  3. What techniques or concepts do we need to explore to create the framework to answer these questions?

Copyright Notice