POINT KINETICS[*]

CLYDE HUIBREGTSE[†]

Abstract.

Key words. point kinetics, reactor dynamics, high performance ODE solution

AMS subject classifications. 68Q25, 68R10, 68U05

1. Point Kinetics and Reactor Dynamics.

1.1. The Motivation for a Simpler Model.

Reactor dynamics are governed by the transport of energetic neutrons throughout the active core. In particular, we define the diffusion of these neutrons in Eq. (1.1).

$$\frac{\partial N(\vec{r},t)}{\partial t} = Dv\nabla^2 N - \Sigma_a vN + S \tag{1.1}$$

where $N(\vec{r},t)$ and $S(\vec{r},t)$ are the neutron density and produced additional neutron density in some volume dV at location $\vec{r}$ at time t. D is the diffusion constant; v is the neutron speed; and Σ_a is the macroscopic neutron absorption cross-section.[2]

Note that this partial differential equation effectively encodes the conservation of neutrons throughout the core. Neutrons in a given volume of fuel can: (1) move to an adjacent volume, (2) be absorbed in the given volume, or (3) be generated as a result of a fission within the volume.

This branching structure lends itself wonderfully to the use of Monte Carlo codes to simulate neutron transport in very high fidelity [4]. Unfortunately, modeling macroscopic reactor behavior with these high fidelity tools is infeasible. Coupling the thermalhydraulic behavior of the plant's power conversion system to the dynamics of subatomic neutrons is computationally intractable. Consequently, we must create an abstraction from the neutronics to core-wide dynamics that effectively model important plant behavior without knowledge of individual neutrons. To do this, we use the Point Reactor model.

1.2. The Point Reactor.

The point reactor model operates under a few critical assumptions. First, it is assumed that the netrons included in $N(\vec{r},t)$ are all of a single energy (for our purposes, this is "fast", rather than "thermal" energies). Additionally, our treatment of the neutron production term $S(\vec{r},t)$ in Eq. (1.1) must be formally defined for the point reactor. As shown in Eq. (1.2), produced neutrons can be of two types: *prompt* and *delayed*. Prompt neutrons are produced through direct fission of fuel nuclei, while delayed neutrons are the byproduct of fission-product decay significantly later than the prompt generation.

$$S(\vec{r},t) = \underbrace{(1-\beta)k_\infty \Sigma_a vN}_{\text{prompt generation}} + \underbrace{\Sigma_i \lambda_i C_i}_{\text{delay generation}} \quad [2] \tag{1.2}$$

[*]Submitted to the editors DATE.

[†]Massachusetts Institute of Technology Departments of Mathematics and Physics (huibregc@mit.edu, https://github.com/ClydeHuibregtse/point_kinetics).

β_i	λ_i	Λ
0.00009	0.0124	0.00001
0.00087	0.0305	
0.00070	0.111	
0.00140	0.305	
0.00060	1.14	
0.00055	3.01	

Table 1
Parameters for uranium fast reactor used in this analysis

The proportionality constants shown in Eq. (1.2) are outside the scope of this analysis. However, C_i refers to the current concentration of "delayed neutron group" i. It has been shown that the inclusion of six delayed neutron groups constitutes a reasonably accurate approximation of reactor dynamics.[3]

The final, and most critical assumption the Point Reactor model makes can be defined as follows: (1) the densities of prompt and delay neutrons are separable in time and space, and (2) the spatial dependence of prompt neutron density matches that of delayed neutron density.

While the derivation of these final differential equations is outside the scope of this analysis, they are reproduced in Eqs. (1.3) and (1.4).

$$\frac{dn}{dt} = \frac{\rho - \beta}{\Lambda} n + \Sigma_i \lambda_i c_i \tag{1.3}$$

$$\frac{dc_i}{dt} = \frac{\beta_i}{\Lambda} n - \lambda_i c_i \tag{1.4}$$

Here, formal definition of the parameters is critical to understanding of the following analysis. Parameters are defined in Eqs. Equations (1.5)–(1.10).

$$n = \text{normalized number of neutrons in the core (effective reactor power)} \tag{1.5}$$

$$c_i = \text{normalized number of delayed neutrons of group } i \tag{1.6}$$

$$\rho = \text{external reactivity (pcm)} \tag{1.7}$$

$$\beta = \text{delayed neutron fraction } (\Sigma_i \beta_i) \tag{1.8}$$

$$\Lambda = \text{mean neutron generation time (s)} \tag{1.9}$$

$$\lambda_i = \text{precursor time constants (1/s)} \tag{1.10}$$

This ordinary differential equation (ODE) consistutes the primary focus of this analysis. [2] For the purposes of this anaylsis, we focus our data collection on a ^{235}U fast reactor. The parameters of which are shown in Table 1.

1.3. Important Concepts for this Analysis. In order to fully understand the process by which we examine this dynamical system, one must be acquainted with the following terms that are used to describe particular features of a reactor transient.

Definition 1.1. ***External Reactivity:*** *External reactivity is defined as an artificial addition (or subtraction) of some fraction of the total reactor neutron popula-*

tion. We call any external reactivity, an "insertion", whether or not it has positive or negative value. It is measured in pcm, or per cent mille (0.001%)

DEFINITION 1.2. ***Dollar (reactivity):*** *One dollar of reactivity ($) is defined as the reactivity required to cross the threshold from delayed criticality to prompt criticality. It is numerically equal to the β constant defined in Eq. (1.8).*

DEFINITION 1.3. ***Prompt Criticality:*** *Prompt criticality refers to a reactor state in which a positive feedback nuclear chain reaction is sustained* entirely *by the generation of prompt neutrons. This differs from delayed criticality in that it does not require the presense of delayed neutron groups to maintain its fission chain reaction. Prompt criticality is characterised by fast spikes in power due to the short generation times for prompt neutrons.*

1.4. Analytical Jacobian Analysis. The form of the ODE defined in Eqs. (1.3) and (1.4), lead to an analytical definition of this system's Jacobian.[1] Particularly, our system is as defined in Eq. (1.11), so there exists a true form of the Jacobian, shown in Eq. (1.12).

$$\frac{du}{dt} = A(t)u \tag{1.11}$$

$$A(t) = \begin{pmatrix} \frac{\rho(t)-\beta}{\Lambda} & \lambda_1 & \lambda_2 & \cdots & \lambda_6 \\ \frac{\beta_1}{\Lambda} & -\lambda_1 & 0 & \cdots & 0 \\ \frac{\beta_2}{\Lambda} & 0 & -\lambda_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\beta_6}{\Lambda} & 0 & 0 & \cdots & -\lambda_6 \end{pmatrix} \tag{1.12}$$

Here, $A(t)$ is the time dependent Jacobian for our dynamical system. Note the time independence for all but the prompt neutron term. This is a critical feature of the system. We can see that, at operating conditions where $0 \leq \rho(t) < \beta = 1\$$, all of the diagonal terms of our Jacobian are negative. This property ensures that the reactor is in a delayed critical state, and the unbounded growth in power is relatively slow. Transients in which $\rho(t) \geq \beta = 1\$$ have Jacobians with a positive diagonal element. This corresponds to the reactor going prompt critical, and the reactor power grows without bound millions of times faster than when it is delayed critical.

There are a few things to note: (1) the unbounded growth of reactor power seen in transients with positive $\rho(t)$ parameters is a numerical phenomenon. In reality, nuclear material undergoes a "fizzle" (CITE) that limits this rapid overpower (except in the cases of nuclear weapons). This effect is not captured in the Point Reactor model, but does not affect solutions around the time of reactivity insertion. (2) For sections of a reactor transient timeseries over which $\rho(t)$ is constant, there is an analytical solution to the dynamical system involving a series of exponential functions defined by the eigenvalues of A.

For the purposes of building a modeling tool that can aid in iterating reactor design, we focus our analysis on the efficacy of solver methods on the order of 100*pcm* reactivity insertions. These insertions correspond to moderate transients seen during normal operations. To be exhaustive, however, we will also examine the efficacy of our solver algorithms under a 1$ insertion. While it is more an exercise in academia than practical engineering, an understanding of large insertions may give insight into the rubustness of these solution algorithms.

1.5. The Analytical Solution to the Step Insertion. As previously stated, in cases where $\rho(t)$ is constant, there exists an analytical solution to the point kinetics ODE. The This solution is reproduced in Eq. (1.13).

$$\Psi = \Sigma_{k=0}^{K}\left((-1)^k\Sigma_{j=0}^{K}\left(\frac{e^{\omega_j t}B_{K-k,j}}{\Pi_{i=0,i\neq j}^{K}(\omega_i-\omega_j)}\right)A^k\right)\Psi_0 \tag{1.13}$$

$$\text{where } B_{m,n} = \Sigma_{i_1=1,i_1\neq j}^{K}\Sigma_{i_2=i_1+1,i_2\neq j}^{K}...\Sigma_{i_m=i_{m-1}+1,i_m\neq j}^{K}\omega_{i_1}\omega_{i_2}...\omega_{i_m} \tag{1.14}$$

(CITE SOLUTION) Each ω_i is an eigenvalue of the Jacobian A, or is a root of the inhour equation:

$$\rho = \Lambda\omega + \omega\Sigma_{k=1}^{K}\frac{\beta_k}{\omega+\lambda_k} \tag{1.15}$$

Note that the formal definition fo this analytical solution has, in the denominator of each summed term, a large product of eigenvalue differences. This procedure of multiplying a sequence of differences is reminiscent of the Lagrange basis polynomial definition, where our dataset is the set of eigenvalues of A. Consequently, we encounter some numerical instability when evaluating this analytical solution. Figure 1 shows how we avoid this floating point imprecision by making use of the `DoubleFloats.jl` package.

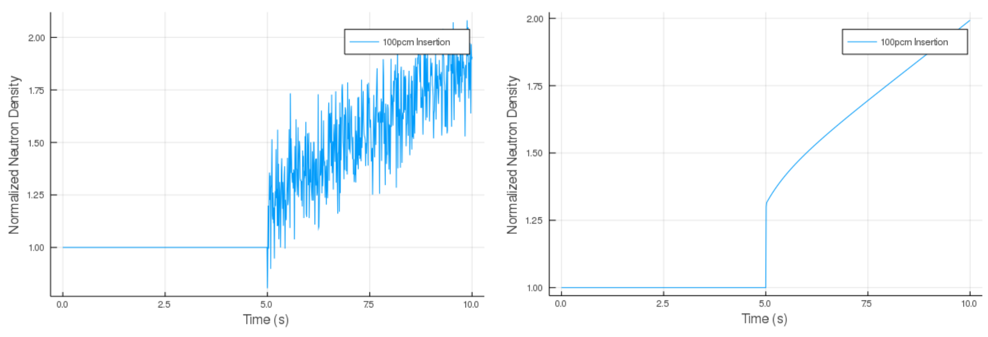

FIG. 1. *Left: The analytical solution to the Point Kinetics equations under a* 100 *pcm insertion as defined in Eq.* (1.13) *with use of the native* `Float64` *precision. Right: The same solution, but instead with* `Double64` *(a* 128 *bit floating point number) as the basis. The error is resolved, and the pertinent features are evident. The insertions occur at* $t = 5s$.

Additionally, we will examine reactivity insertions on the order of 1\$, which diverge very quickly. Figure 2 shows the analytical solution to the point kinetics equations under a step insertion of 1\$.

These two curves are a baseline to which we compare all of our computed solutions, giving a quantifiable metric for accuracy in both tranients.

2. Solving the Point Kinetics Equations.

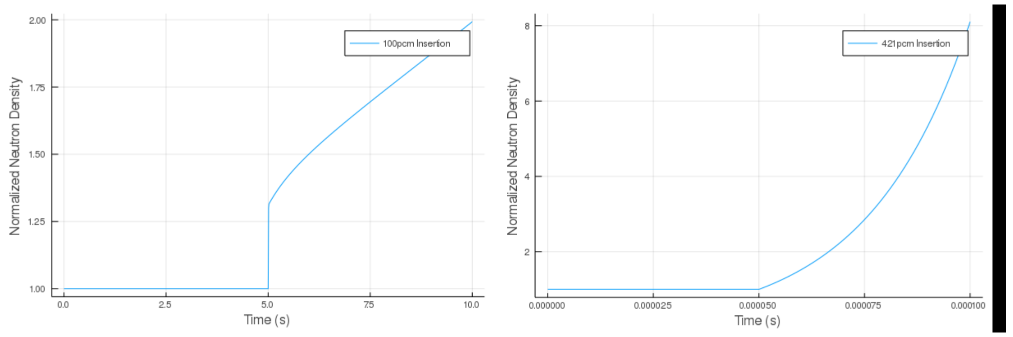

Fig. 2. *Left: A reproduction of the right plot from Figure 1. Right: The analytical solution defined in Eq. (1.13) with a reactivity insertion of* 1$ *or* 421 *pcm. Note that the prompt jump is so large that the solution diverges within a thousandth of a second. These two curves represent our test case solutions.*

2.1. Building an Optimal Derivative Defintion. In order to perform any sort of reasonable benchmarking for solver algorithms, we must ensure that our ODE definition is optimal, such that any discrepancies in solution time are not drowned out by overhead. There are a few things we can check to ensure optimality in our ODE definition: (1) memory allocations, and (2) type stability.

Using the `BenchmarkTools.jl` `@btime` and `@benchmark` macros, we can demonstrate that our inner loop (the `pk!` method) has minimal allocations. Particularly, we can achieve almost no (2) heap allocations per inner loop (a result of our two `@view` macro calls), totalling only 96 bytes.

Now we can use the type inferencing capabilities of `Julia`'s JIT compiler to speed up our solves. To achieve type stability, we define an abstract type family, `AbstractInsert`, that handles arbitraty reactivity insertion functions. This allows the parameters used within the inner loop to be fully type stable. The `@code_warntype` macro shows no potential type ambiguities throughout our inner loop, so we can be assured the JIT compiler is producing optimal type inferences.

Now we can effectively dive into a solver analysis with the guarantee that algorithmic differences in the integrators produce the variance in the total runtimes, and not computational inefficiencies. Note that in the following sections, we define a "branching" insertion as a step function with a jump discontinuity, while a "smoothed" insertion is a hyperbolic tangent function of equal magnitude. Both options are shown in Figure 3

2.2. Metrics for Evaluating Integrator Performance. In the following sections we present a detailed analysis of the efficacy of a litany of solvers at a variety of tolerances. The best metric for rating the performace of a solver algorithm is an open question. The best choice is user-specific, so for the sake of making a conclusion, we specify our user needs here. We set out to create a high-speed, reasonably accurate simulator to be used in core design work and reactor control optimizations. There are a few key threshold criteria that we require of our simulator:

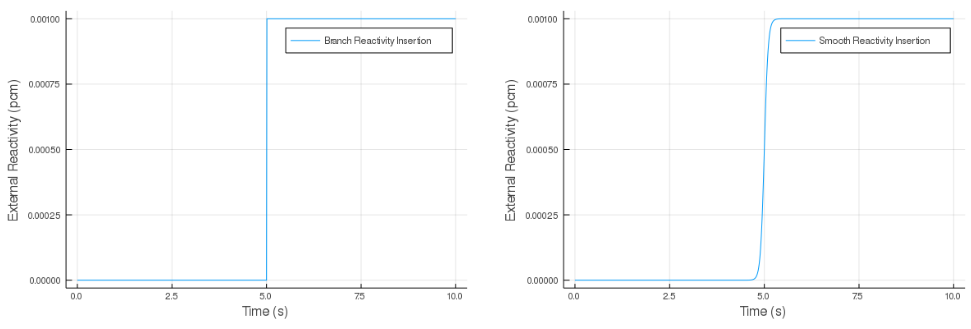

FIG. 3. *Left: a branched insertion of* 100 *pcm. Right: a smoothed insertion of* 100 *pcm.*

1. Resolve the prompt jump to high fidelity
2. Approximate netron densities during delayed effects to a decent accuracy, but less so than than the prompt jump
3. Have the ability to accept a reactivity insertion of arbitrary shape

It is with these demands in mind that we can evaluate each integration algorithm.

Note that for the following sections regarding solver performance, the computer with specifications described in Table 2 is used.

Intel i7-7700	4.2 GHz (4 Core)
DDR4 2400 MHz	32 Gb
GeForce GTX 1080ti	11.34 TFLOPS (Float32)

TABLE 2
Computer specifications for this performance analysis.

2.3. Solving a 100 pcm Step Insertion. We approach the definition of a step insertion in two ways: (1) as a branched function with a jump discontinuty, and (2) as a smooth hyperbolic tangent function of equal size. For each of these two definitions of a step insertion, we evaluate the solvers on three criteria:

1. Total runtime
2. Number of accepted and rejected steps
3. Average and final timepoint error

Let's begin by examining our solver algorithms under a low tolerance for the 100 pcm insertion. Figure 4 shows the work precision plots for both the branching and smoothed insertions.

With low tolerances ($10^{-12} \rightarrow 10^{-7}$), we have a relatively clear delineation of solver performance. The top two performing algorithms are LSODA and CVODE_BDF, both in terms of accuracy and runtime. The native `Julia` RadauIIA5 performs quite similarly to its counterpart RADAU5, which implies that the integration is relatively agnostic to language implementation. The KenCarp methods are consistently the worst performing of the set, and Rosenbrock23, while slightly slower, achieves higher accuracy than its remaining `Julia` counterparts.

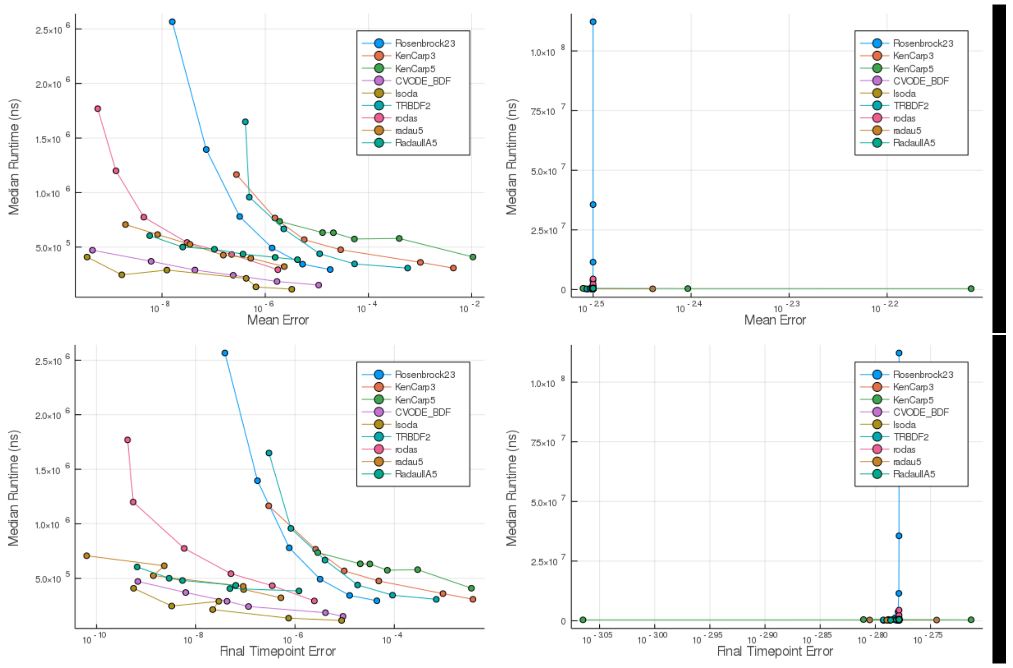

FIG. 4. *Left: mean timeseries error and final timepoint error for a branching step insertion. Right: mean timseries error and final timepoint error for the hyperbolic tangent smoothed insertion.*

We see that the smoothed solutions are effectively the same for all solvers in terms of accuracy. There are some notable exceptions: both KenCarp methods produce higher variance in their solution accuracy. Rosenbrock23 handles the smoothed insertion much more poorly at high tolerances than do the remaining algorithms.

Note that in both of these cases, DOPRI5 was analyzed as a potential "worst-case" Runge-Kutta method, and in fact, it performed so poorly, it has been omitted from these plots. Now we can look into the performance of these solution algorithms in cases where accuracy is of less importance. These results are compiled in Figure 5.

For higher tolerances ($10^{-6} \rightarrow 10^{-1}$), for our branched insertion, we see similar trends to the low tolerance cases, though with some variation in accuracy. Again, from a holistic viewpoint, both LSODA and CVODE_BDF seem to outperform all other solvers, particularly with respect to their runtimes. However, RADAU5 and RadauIIA5 make a strong case for accuracy at higher tolerances at the expence of a few 10ths of a millisecond per run. Once again, the KenCarp methods and TRBDF2 perform quite poorly. The same can be said about the results from the smoothed case, however with an overall reduction in average runtime across all solvers.

As a representative example, let's examine the time dependent error of one of the top performing algorithms, CVODE_BDF. Figure 6 shows the effect that smoothing has on accuracy of a critical part of the solution.

As is shown in Figure 6, there is a considerable difference in the magnitudes of

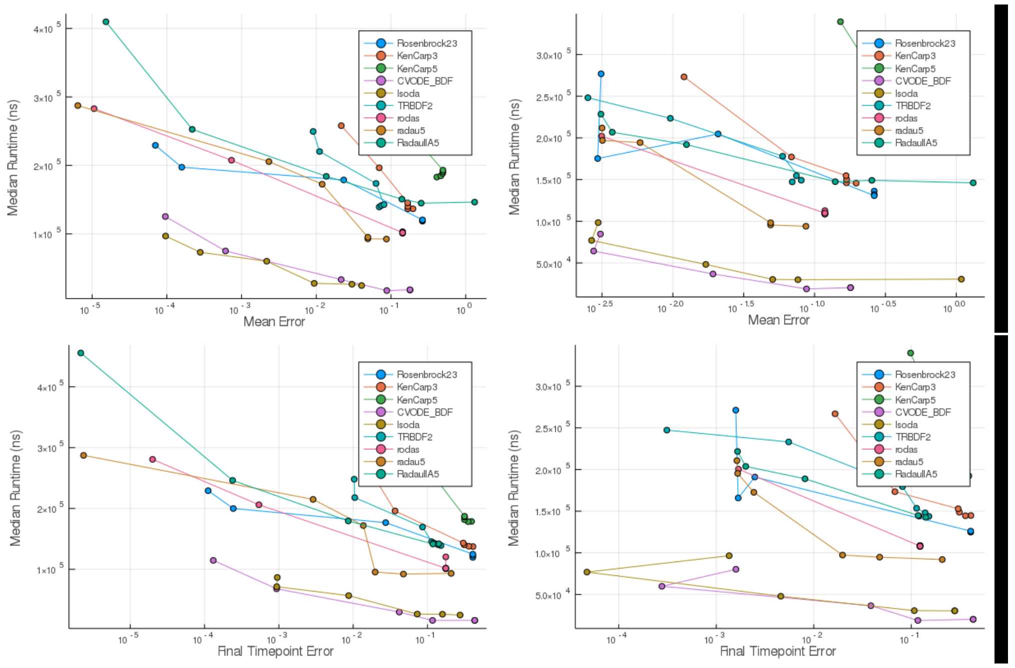

FIG. 5. *Left: mean timeseries error and final timepoint error for a branching step insertion. Right: mean timseries error and final timepoint error for the hyperbolic tangent smoothed insertion.*

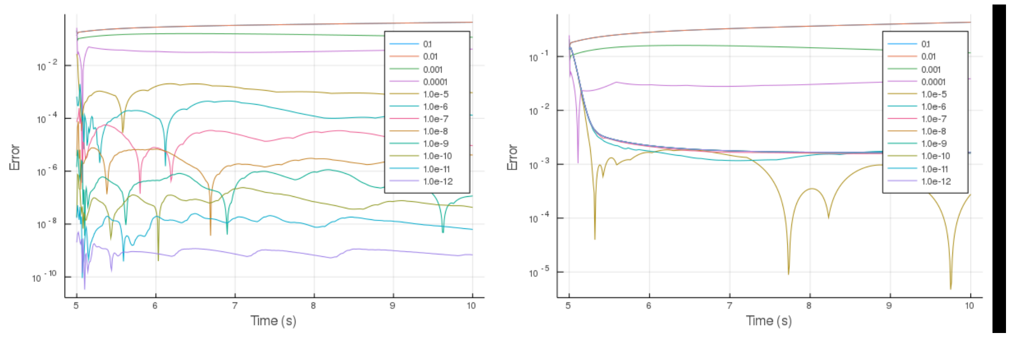

FIG. 6. *Left: error as a function of time for CVODE_BDF with branched insertion. Right: error as a function of time for same algorithm but with smoothed insertion. Both plots begin at the time of insertion.*

the accuracy after the smoothing of the reactivity insertion. What is particularly troublesome is the error during the first ~ 0.1s of the insertion. Accuracy in defining the shape of the prompt jump is important. Afterwards, the error in the delayed neutron propagation is less critical. While this distinction may be enough to sway a user with a need of higher accuracy, for the purposes of approximating reactor dynamics for use in quick iterative engineering tasks, smoothing is feasible. Should we deem the reduction in accuracy due to smoothing acceptable, we can see how much more performant the solvers are in those cases.

In addition to tracking total runtime, we have also recorded the number of solve iterations each algorithm takes. These results are compiled for all tolerances for each solver in Figure 7.

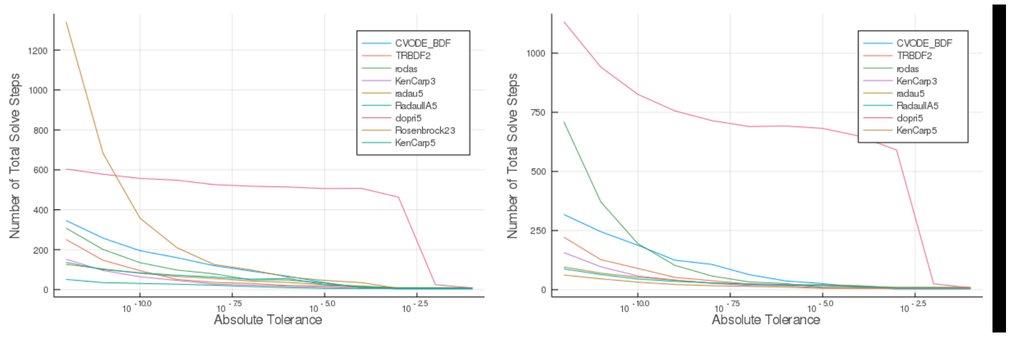

Fig. 7. *Left: number of steps required to solve Point Kinetics equations with branching insertion. Right: number of steps required to solve with a smoothed insertion. Note that Rosenbrock23 took far too many steps in the smoothed case to warrant plotting, and that LSODA does not support the counting of steps during its solve.*

Here we have some interesting results. There are a few algorithms whose number of steps worsens through smoothing (particularly our regular poor performers Rosenbrock23 and DOPRI5). More importantly, though, our faster algorithms such as CVODE_BDF and RADAU5/RadauIIA5 drastically decrease in the number of steps required and therefore the required runtime.

2.4. Solving a 421pcm Insertion. A 421 pcm insertion corresponds to exactly 1$ for the described reactor. While not a transient expected for any sort of normal transient through which the assumptions made by the Point Reactor model would survive for more than a few microseconds, this transient is a industry classic for evaluating solver performance. This transient is colloquially referred to as “prompt criticality”.

Acknowledgments. We would like to acknowledge the assistance of volunteers in putting together this example manuscript and supplement.

REFERENCES

[1] B. Ganopol, *A highly accurate algorithm for the solution of the point kinetics equations*, Annals of Nuclear Energy, 62 (2013), pp. 564–571.
[2] D. Hetrick, *Dynamics of Nuclear Reactors*, The University of Chicago Press, 1971.
[3] G. Keepin, *Physics of Nuclear Kinetics*, Addison-Wesley, New York, 1965.
[4] J. Leppanen, M. Pusa, T. Viitanen, V. Valtavirta, and T. Kaltiaisenaho, *The serpent monte carlo code: Status, development and applications in 2013*, Annals of Nuclear Energy, 82 (2015), pp. 142–150.