

Discontinuity-Aware Solvers for Vapor-Compression Cycles

Ekanathan, Shreyas; Subramanian, Avinash; Smith, Oscar; Ma, Yingbo; Rackauckas, Chris;
Laughman, Christopher R.

TR2026-153 October 14, 2026

Abstract

Physics-based models of vapor-compression cycles are often formulated as large, stiff DAE systems with derivative discontinuities introduced at the points of phase change. We use an implementation of a discontinuity-aware ODE and DAE solver scheme that uses domain-specific, user-declared information to compute appropriate time steps for more efficient integration. This scheme is made opensource within `DifferentialEquations.jl`. We demonstrate how discontinuity information can be specified symbolically in a declarative modeling language such as `Dyad` and passed to the solver to address the numerical challenges posed. We apply this discontinuity-aware technique to two applications: First, a scalar ODE with two discontinuities, and then a model of a full vapor-compression cycle. The discontinuity-aware solver yields significantly fewer step rejections and improved solver times, particularly with the `Rodas5P` solver at loose tolerances, as compared to standard error control, suggesting that exposing physical discontinuity information from equation-oriented models to adaptive solvers can improve practical simulation performance without requiring reassembly of the model.

American Modelica & FMI Conference 2026

© 2026 MERL. This work may not be copied or reproduced in whole or in part for any commercial purpose. Permission to copy in whole or in part without payment of fee is granted for nonprofit educational and research purposes provided that all such whole or partial copies include the following: a notice that such copying is by permission of Mitsubishi Electric Research Laboratories, Inc.; an acknowledgment of the authors and individual contributions to the work; and all applicable portions of the copyright notice. Copying, reproduction, or republishing for any other purpose shall require a license with payment of fee to Mitsubishi Electric Research Laboratories, Inc. All rights reserved.

Mitsubishi Electric Research Laboratories, Inc.
201 Broadway, Cambridge, Massachusetts 02139

Discontinuity-Aware Solvers for Vapor-Compression Cycles

Shreyas Ekanathan^{1,3} Avinash Subramanian¹ Oscar Smith¹ Yingbo Ma⁴ Christopher Rackauckas¹ Christopher R. Laughman^{2*}

¹JuliaHub, Inc. {shreyas.ekanathan, avinash.subramanian, oscar.smith, chris.rackauckas}@juliahub.com

²Mitsubishi Electric Research Laboratories, Cambridge, MA, USA laughman@merl.com

³University of Texas at Austin

⁴Independent Researcher

Abstract

Physics-based models of vapor-compression cycles are often formulated as large, stiff DAE systems with derivative discontinuities introduced at the points of phase change. We use an implementation of a discontinuity-aware ODE and DAE solver scheme that uses domain-specific, user-declared information to compute appropriate time steps for more efficient integration. This scheme is made open-source within `DifferentialEquations.jl`. We demonstrate how discontinuity information can be specified symbolically in a declarative modeling language such as `Dyad` and passed to the solver to address the numerical challenges posed. We apply this discontinuity-aware technique to two applications: First, a scalar ODE with two discontinuities, and then a model of a full vapor-compression cycle. The discontinuity-aware solver yields significantly fewer step rejections and improved solver times, particularly with the `Rodas5P` solver at loose tolerances, as compared to standard error control, suggesting that exposing physical discontinuity information from equation-oriented models to adaptive solvers can improve practical simulation performance without requiring reassembly of the model. *Keywords: Vapor-compression cycles, HVAC, Differential Equations, Dyad*

1 Introduction

Vapor-compression cycles (VCCs) represent a key technology that serves as a foundation for many heating and cooling systems used in residential, commercial, and industrial buildings today, such as in heat pumps and air-conditioning systems. These systems play an essential role in maintaining comfort in buildings and regulating industrial thermal processes, and also provide an important capability in the continuing trend of electrification as a means of integrating renewable energy sources into the electrical grid. Given ongoing climate trends, these systems are becoming more common throughout the world in both existing and emerging markets.

Considering the wide geographic diversity of equipment installations and the range of applications, model-based design processes for equipment that incorporates vapor-compression cycles is becoming increasingly im-

portant to manage their dynamic behavior in the face of changing operating conditions. Whereas common design processes for these cycles tend to focus on experimental trial-and-error methodologies, system-level modeling and simulation is a valuable tool when used to pursue high system-of-systems efficiencies and reduce the dependence of the design process on time-consuming and expensive experimental laboratory work. These tools can improve the design process across many different system aspects, including refrigerant selection, component sizing, control system design, and initial design evaluations in the larger system-of-systems context.

A variety of approaches have been used to describe the spatially and temporally varying behavior of these systems as a set of computational models (Rasmussen 2012; Rasmussen and Shenoy 2012). While the underlying physics of these systems might be most accurately characterized as a set of partial differential algebraic equations (PDAEs), these are often simplified to form either a set of ordinary DAEs or a set of ordinary differential equations (ODEs). DAEs often arise in these systems due to the algebraic relations between the refrigerant properties, as well as from algebraic equations from flow and energy constraints formed when interconnecting component-based models. This modeling approach has been popularized by the `Modelica` language (Modelica Association 2023) and recently by the `Dyad` language (Rackauckas, Abdelrehim, et al. 2025), in which component models are created using an equation-oriented declarative language, and then interconnected to form larger system-level models. We employ this approach to construct these VCC models, which are based on well-known previous work (Qiao et al. 2015).

While a substantial body of work exists for the numerical solution of general DAEs (Brenan et al. 1995; Petzold 1982), the numerical solution of DAE models of VCCs pose a number of distinct challenges. These models are often quite large, due to the discretization of the heat exchangers into a number of control volumes to approximate the behavior of the partial differential equations describing the conservation of mass, momentum, and energy within the refrigerant flow and across the pipe wall. In addition, these models contain a variety of nonlinear relationships,

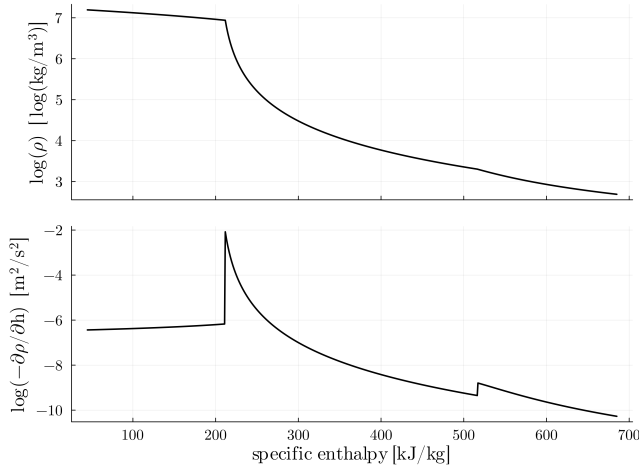


Figure 1. Plot of $\log(\rho)$ and $\log(-\partial\rho/\partial h|_p)$ for the refrigerant R32 at 1 MPa over a range of specific enthalpies, illustrating the derivative discontinuities at the bubble and dew points.

such as between the refrigerant properties (e.g., temperature, density) or in the closure relations relating the pressure drop to the refrigerant flow, which must be accommodated when solving these models.

The C^0 behavior of the refrigerant in the two-phase (condensing or evaporating) flows also introduces numerical challenges to the solver. This behavior is observed when integrating the differential equations related to the conservation of mass, which are often formulated in terms of the density (as the cross-sectional area of the tubes in the heat exchangers are constant). As seen in the example provided in Figure 1, this density has discontinuous derivatives at the boundaries of the two-phase region, known as the bubble point (between the liquid and two-phase regions) and the dew point (between the two-phase and vapor regions). Numerical solvers that integrate the differential equations of the model are often designed under the assumption that the underlying model is smooth, and may either become extremely slow when encountering such derivative discontinuities, or even crash or produce erroneous output in these situations. This is because the solver steps past the discontinuity, incurs a large local error, rejects the step, reduces the step size, and repeats this process many times before successfully crossing the trouble spot (Gear and Østerby 1984).

Several techniques have been developed for solving dynamical systems with discontinuities (Dieci and Lopez 2012), which arise in two distinct varieties: time-triggered and state-triggered. Time-triggered discontinuities occur at prescribed times that are known in advance and so are easier to handle, as the integrator can step to the discontinuity location and restart integration (Mørk et al. 2025). State-triggered discontinuities, on the other hand, require the determination of the time location of the discontinuity during integration. While most numerical integration methods are constructed on the assumption that the solution is smooth, these discontinuities violate this assumption.

Gustafsson illustrates the effect of these discontinuities by showing that a naive integrator can spend a lot of computation on repeated step rejections to handle the large error introduced, whereas an informed integrator may directly step to it (Gustafsson 1994).

A number of works have addressed the numerical integration of dynamical systems with state-triggered discontinuities. Carver (Carver 1978) and Cellier (Cellier 1979) published seminal work on marking the discontinuity using zero-crossing functions, which are checked by the integrator for sign changes that indicate when the discontinuity is crossed. After crossing the discontinuity, the integrator is restarted with the chosen set of equations for the branch. One distinct mode of failure for this approach is that if a discontinuity is crossed twice within an integrator time step, or if two separate discontinuities lie close together, they may be missed. Park and Barton address this shortcoming by presenting an approach to search the entire timestep by leveraging interval arithmetic techniques to guarantee detection of all discontinuities (Park and Barton 1996).

Otter et al. discuss the implementation of this zero-crossing function technique in Modelica compilers, where a conservative approach is taken: every relation (with operators such as less than, greater than etc.) between continuous variables is assumed to introduce a discontinuity, and therefore automatically triggers a state event unless the modeler mentions otherwise (Otter et al. 1999). Fredriksson et al. present an implementation in Assimulo that can be attached to any integrator (Fredriksson et al. 2014) and is compatible with the FMI standard. In addition, rather than relying on a sign-change, they adopt a domain-based formulation whereby a change between the domains $g < 0$ and $g \geq 0$ is detected. This approach handles a failure mode of the sign-change test for the case where a discontinuity is triggered by the function reaching zero and staying there (as can happen in practical examples such as cutting power to a clutch). In other related work, Tummescite and Eborn implement crossing functions for multiphase thermal-fluid flow with chemical reactions (with external property models) that allow numerically robust detection of new phases by using a "deviation index" which is the normalized distance to a phase-separation surface (Tummescite and Eborn 2002).

In the present work, several fundamental ingredients of the solver algorithm are taken from this existing literature. We also use zero-crossing functions are used to mark discontinuities which are then detected by relying on a sign-change, and the location of the discontinuity is determined by performing a root finding search on the integrator's dense interpolant. However, the solver algorithm implements several unique features. First, the solver uses the zero-crossing functions to discover a discontinuity even on step rejections, not merely on step acceptances as is done in alternative approaches. This scheme, designed to handle the anticipated high error in rejected steps, is a key contribution of this paper. For example, the retried

step size after rejection is set at half the rejected step size, which empirically yields the best results for the cases studied. Finally, the proposed algorithm is fully composable with any adaptive methods (such as `Rodas5P`, `Tsit5`, and `RadauIIA5` in the Julia SciML ecosystem).

We make the following contributions:

1. We implement a discontinuity-aware ODE and DAE solver that uses domain-specific (user-declared) information to compute appropriate time steps for more efficient integration. This solver is provided open-source in the Julia SciML ecosystem within `DifferentialEquations.jl` (Rackauckas and Nie 2017). Existing schemes only take advantage of such information on accepted steps, while we extend the scheme to function on rejected schemes as well.
2. We apply this solver algorithm to both a scalar ODE with two discontinuities as well as a vapor-compression cycle. We show that discontinuity-awareness results in significantly fewer step rejections and improved solver times, particularly with the `Rodas5P` solver at loose tolerances, compared to solvers using standard error control methods.
3. Lastly, we demonstrate how the domain-specific discontinuity information can be specified symbolically in a declarative modeling language (such as `Dyad`) and passed on to a numerical solver ecosystem.

The paper is organized as follows: Section 2 presents the two exemplar systems, consisting of a scalar ODE system with two discontinuities and a vapor-compression cycle model with varying compressor controls after a zero-flow startup. Section 3 describes the discontinuity-aware solver algorithm and contrasts with the standard solver, Section 4 presents the results on the two case studies, and Section 5 presents conclusions and directions for future work.

2 Exemplar Systems

2.1 Scalar ODE system

First, we study the following discontinuous ODE system with two state discontinuities:

$$\dot{x} = f(x, t) = \begin{cases} 2x, & x < 1 \\ x + 0.2, & 1 \leq x < 2 \\ -4x + 12, & x \geq 2 \end{cases} \quad (1)$$

with initial condition $x(0) = 0.1$ and $t \in [0, 2.5]$.

The solution is plotted in Figure 2, illustrating the fact that the derivative discontinuities produce kinks in the solution at $x = 1$ and $x = 2$. This simple system allows us to explain the stepping dynamics and differences versus a standard ODE solver, as discussed in Section 4.1.

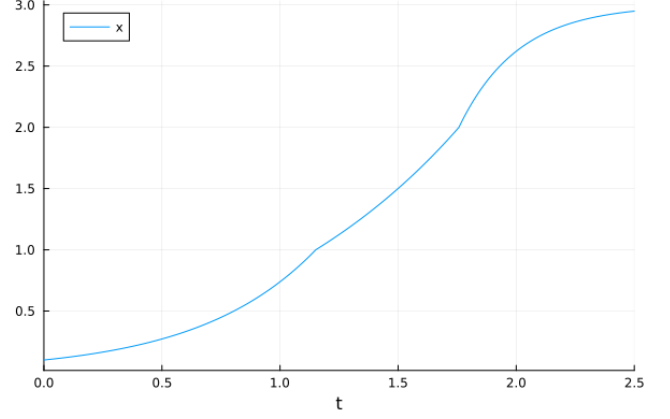


Figure 2. Solution to the toy problem. Note the effects of the derivative discontinuity as the solution crosses $x = 1$ and $x = 2$.

2.2 Vapor-compression cycle model

Figure 3 presents a vapor-compression cycle modeled in `Dyad`. It consists of a positive-displacement compressor, an evaporator and a condenser modeled as a tube-fin heat exchangers with a cross-flow configuration, and a linear expansion valve. The compressor and valve are described via algebraic relations since their dynamics are much faster than the rest of the system. The tube-fin heat exchanger comprises a 1D refrigerant thermal-fluid flow model with a finite-volume discretization along the flow axis consisting of mass, momentum and energy balance equations. Pressure and enthalpy states are selected as differential states to be integrated, while other refrigerant properties are computed from these states using fitted spline-based property models. The heat transfer through the tube walls to the air side is modeled using appropriate efficiency correlations, and the moist air flow has a cross-flow configuration over the finned tubes. A complete description of model is available in (Laughman and Qiao 2017) with a summary in the Appendix.

The `Dyad` model is developed using the `Dyad HVAC Components` library (JuliaHub 2026), in which reusable subcomponents are developed and interconnected. The `Dyad` compiler then reduces the full model to the form of a semi-explicit DAE (3) through alias elimination, index reduction, and tearing. This DAE is then passed to a numerical solver from an ecosystem such as `DifferentialEquations.jl` (Rackauckas and Nie 2017) to solve the equations of the form

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, \mathbf{y}; u(t)) \quad (2)$$

$$\mathbf{0} = \mathbf{g}(\mathbf{x}, \mathbf{y}; u(t)), \quad (3)$$

with $u(t)$ representing the controller inputs (such as compressor speed and valve position), $\mathbf{x}$ representing the differential states (such as refrigerant enthalpies and pressures in each control volume), and $\mathbf{y}$ denoting algebraic states (e.g., heat duties, temperatures and moist-air states). For this case study, compressor speed inputs are chosen so

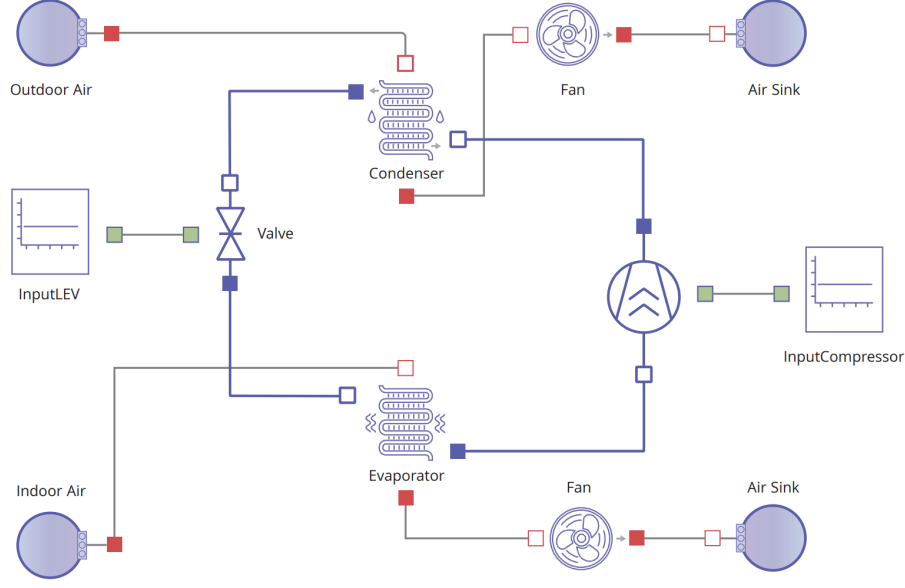


Figure 3. Vapor-compression cycle model in Dyad

as to drive the system through a cold-start, low-load, peak-load, and partial-shutdown regimes in succession without ever reaching a steady-state, so that several system states repeatedly traverse the refrigerant’s two-phase region where discontinuities arise.

Dyad provides a native mechanism for attaching continuous events to an assembled system model. These events are specified declaratively using Dyad’s symbolic language, and thus provide an ergonomic interface for the modeler to communicate domain-specific information to the ODE/DAE solver (1). These continuous events are then passed as callback functions to the integrator and are triggered when the integrated state satisfies the corresponding predicate. Mathematically, these can be written in terms of guard functions $g_j(\mathbf{x})$ which indicate points of discontinuity. We attach two such guards in each refrigerant control volume, one at each phase boundary:

$$\begin{aligned} g_{\text{bub}}(\mathbf{x}) &= h - h_{\text{bub}}(P) = 0 \\ g_{\text{dew}}(\mathbf{x}) &= h - h_{\text{dew}}(P) = 0, \end{aligned} \quad (4)$$

where $h_{\text{bub}}(P)$ and $h_{\text{dew}}(P)$ are the bubble and dew enthalpies evaluated at the particular pressure P , and h is the enthalpy state.

```
component RefrigerantMedium
  # parameters go here
  # variables go here (including P and h)
relations
  # property relations go here
  discontinuityAt(h = h_bub(P))
  discontinuityAt(h = h_dew(P))
end
```

Listing 1. Dyad syntax for declaring the phase-boundary events of Eq. (4) in a refrigerant medium model. Parameters, variables and property relations are omitted.

In this case study, we use eight control volumes per heat exchanger giving rise to thirty two events. The discontinuity-aware solver evaluates this guard information if a step is rejected, which is useful in determining subsequent candidate time steps.

3 Discontinuity-Aware Solver Algorithm

The discontinuity-aware solver is partially inspired by the work on delay differential equations (DDEs) by Guglielmi and Hairer (Guglielmi and Hairer 2008), which proposes a scheme for handling the breaking points in DDEs. These breaking points can be treated similarly to discontinuities in the right hand side of ODEs and DAEs, motivating the scheme presented here.

The new scheme exposes discontinuity information to the step-size controller thereby enabling cleaner navigation around them. The right flowchart in Figure 4 presents the improved adaptive time-step control logic, which is the core feature of the discontinuity-aware solver. In comparison, the left flowchart presents the time-step control without discontinuity-awareness. Of particular note is the fact that the current state of the software only runs on accepted steps, and this is our baseline for comparison. The novelties we introduce are manifest in the behavior on rejected steps, as described below.

Both solvers first proceed by taking a trial step as appropriate for the method, giving a test value $\hat{x}$. After taking the step, the solver uses a lower-order Runge-Kutta solution formed from the same stage evaluations to derive a lower-order estimate $\tilde{x}$. A normalized error estimate for the step is thus derived as

$$E = \sqrt{\frac{1}{n} \sum_{i=1}^n \left(\frac{\hat{x}_i - \tilde{x}_i}{\text{atol} + \text{rtol} \cdot \max(|\hat{x}_i|, |\tilde{x}_i|)} \right)^2}. \quad (5)$$

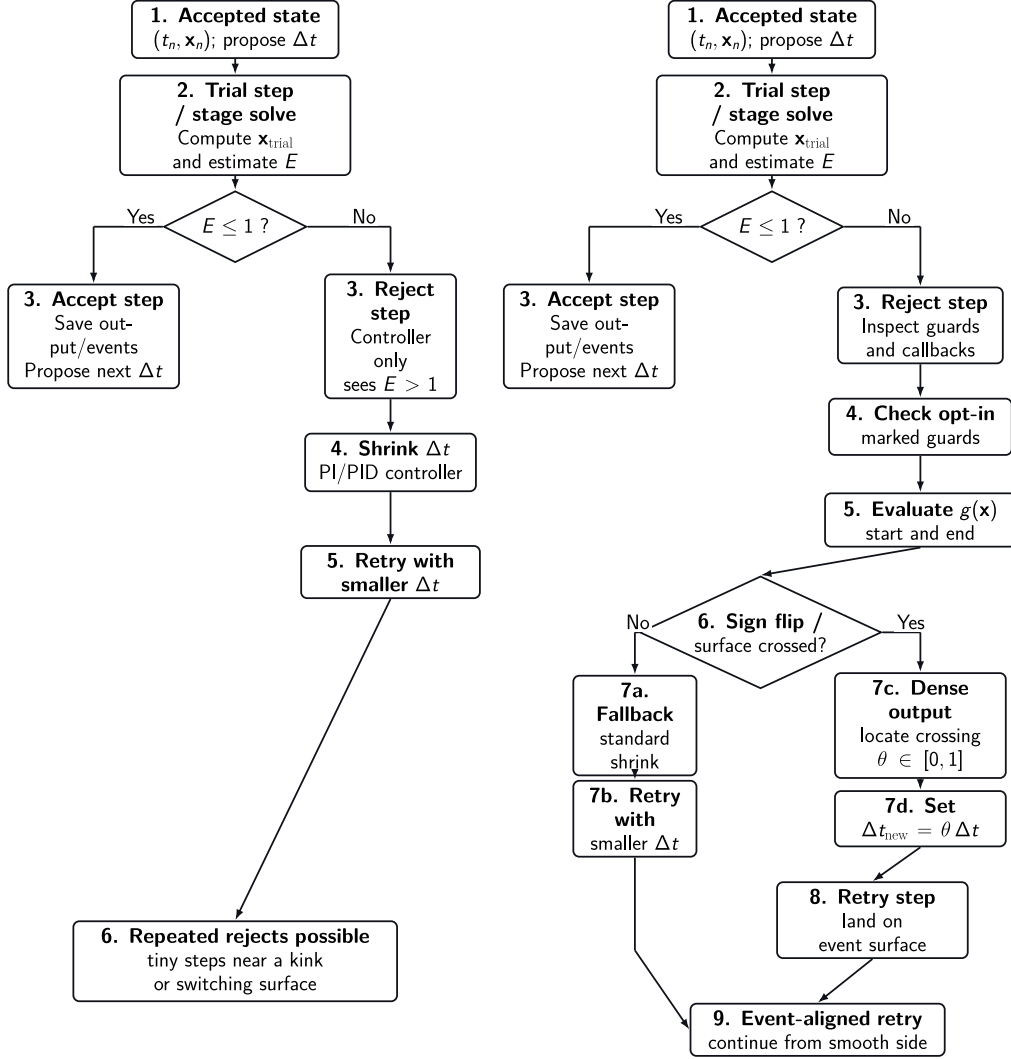


Figure 4. Adaptive time-step control with and without discontinuity-aware logic. The left branch applies the standard rejected-step controller. The right branch uses the marked guard functions together with a nonlinear solve on the dense interpolant to recompute a candidate timestep on the earliest detected discontinuity crossing.

In the event that the error is less than 1, both schemes will accept the step and proceed to the next step. However, the key difference occurs on step rejections, which we will now further explore.

3.1 Traditional step-size controllers

Many step-size controller schemes exist, but the general idea of using heuristic measures to predict the appropriate step size for the next step is often consistent from scheme to scheme. For instance, the Gustafsson controller scheme (Gustafsson 1991) computes the new step size via the following heuristic:

$$h_{\text{new}} = \frac{f_{\text{safe}} \cdot h}{E^{1/(p+1)}}, \quad (6)$$

where f_{safe} is a safety factor typically taken around 0.9, h was the current step size attempted, E was the error of the current step, and p is the order of the method.

This scheme works well when the system is smooth, but derivative discontinuities lead these heuristic estimates to perform very poorly, with clusters of step rejections around the discontinuities. For this reason, we introduce our discontinuity-aware step-size controller to handle these situations more robustly.

3.2 Discontinuity-aware step-size controller

We expose discontinuities via callbacks (Rackauckas, Nie, and SciML Contributors 2024), which describe discontinuity functions in two main forms:

1. Continuous callbacks: A single discontinuity function $g(x)$, which triggers an event at points where $g(x) = 0$.
2. Vector Continuous callbacks: A vector of conditions, allowing multiple discontinuity surfaces to be simultaneously monitored.

These are exposed to the step size controller via the integrator interface, allowing the identification of discontinuities in our scheme. Upon opting to reject a step, we perform the following analysis on the callbacks:

Identifying event occurrences: Consider a step from t to $t+h$, with $f(t) = x_t$, $f(t+h) = x_{t+h}$. For a given callback $g(x)$, a root of g occurs if $g(x_t)$ and $g(x_{t+h})$ have different signs by the intermediate value theorem. Then, $g(x_t) \cdot g(x_{t+h}) < 0$ provides a sufficient condition for detecting the occurrence of an event. We evaluate this guard on all callbacks and only proceed to the subsequent steps on those that had event occurrences; if no events are detected, we fall back to the standard step-size rejection scheme previously described.

Identifying the event locations: For state-dependent callbacks previously described, we seek the time t_{event} at which $g(f(t_{\text{event}})) = 0$. This time is determined by using the dense interpolant for the solver, either derived specially for the integrator or using a Hermite interpolation, and an interval nonlinear solve is done to determine the location of the event using a bisection-like search. This gives a point in time at which the event occurred, and we choose the first such event to step to directly. We note that the dense interpolant is built from data already computed during the rejected trial step, so does not require either full function evaluations, Jacobian evaluations or matrix factorizations. However, given that the error was already high on the step, the interpolant will not be perfect, and hence we expect some numerical error with the computation of the event, but even with this error, this scheme causes the timestep to be chosen much closer to the discontinuity than a standard PI timestepper which is unaware of the discontinuity's existence.

Given this error, we choose to mildly bias the result of the solve, θ , via the following formula:

$$\theta = (0.5 + 0.4 \cdot \sin(\pi \cdot (\theta - 0.5)))$$

Or equivalently in polynomial form (to reasonable error, for faster computational purposes),

$$\theta = 0.5 + 1.2372 \cdot (\theta - 0.5) - 1.7487 \cdot (\theta - 0.5)^3$$

The motivation for this biasing is that we want each step to maximize the information yielded, hence an extremely large step (close to the prior rejected step) or a very small step is suboptimal. Therefore, we choose to try and bias our steps towards roughly $dt/2$, while also maintaining use of the nonlinear solvers results.

Resumption of Smooth Integration: Upon crossing the discontinuity, we reset error history of the controller. This is done to indicate the resumption of integration in a new domain, in which prior assumptions may not apply. This allows us to more efficiently determine the appropriate step-sizes for this regime.

In addition to this error reset, we also restrict the first step after a discontinuity forced step. This is done out of

caution, given that we expect that we have just navigated the discontinuity, but in the event that we have not, we want a limited step to again maximize the information gain from this rejected step.

More technically, suppose our first rejected step ends at time t . Then, after our discontinuity step, we restrict the subsequent step to $\min(dt, (t - t_{\text{curr}})/4)$, where dt is the proposed step-size. Thus, if our intended step takes us beyond the area where we think the discontinuity is, to prevent unnecessary failures, we limit this first step before continuing integration as normal.

We further note that this discontinuity-aware time-step controller is composable with a variety of solvers in the `DifferentialEquations.jl` (Rackauckas and Nie 2017) ecosystem.

4 Results and Discussion

In this work, we study the application of this scheme to three algorithms: `Rodas5P` (a Rosenbrock method applicable to both DAEs and ODEs), `RadauIIA5` (a fully-implicit Runge-Kutta method applicable to both DAEs and ODEs), and `Tsit5` (an explicit Runge-Kutta method applicable to ODEs) for both of the exemplar models.

We note that in both the standard and discontinuity-aware solver configurations, the solver has access to zero-crossing functions at all times. The distinction is that the standard controller only uses this information on accepted steps, while our scheme additionally exploits it on rejected steps.

4.1 Scalar ODE system

Figure 5 presents the time step plots of the `Tsit5` and `Rodas5P` solvers with a discontinuity-aware controller versus a standard controller. On each plot, the left pane shows the time steps taken near the $u = 1$ discontinuity while the right pane shows this near the $u = 2$ discontinuity. The y-axis shows the number of trial time steps. Each horizontal bar spans a proposed time step with the bars classified by whether the time step was accepted (blue) or rejected (red). Studying the right pane of Figure 5 (a), we observe that the first step is rejected in both cases. Both the discontinuity-aware and standard solvers shrink the next proposed time step. However, only the discontinuity-aware solver computes an appropriately-sized time step; while it is still rejected, gives this controller much more information to proceed. The standard solver shrinks the time step proportional to the error of the step, resulting in the rejection of the next time step being rejected before the succeeding one is accepted. This pattern repeats several times as we note that the standard solver has significantly more (red) rejected time steps. This trend also continues with Figure 5 (b) as we observe that the discontinuity-aware solver requires significantly fewer trials to pinpoint the discontinuity and integrate past it so as to solve the ODE efficiently.

Figure 6 presents several solver metrics: wall time (ms), number of function evaluations, and number of

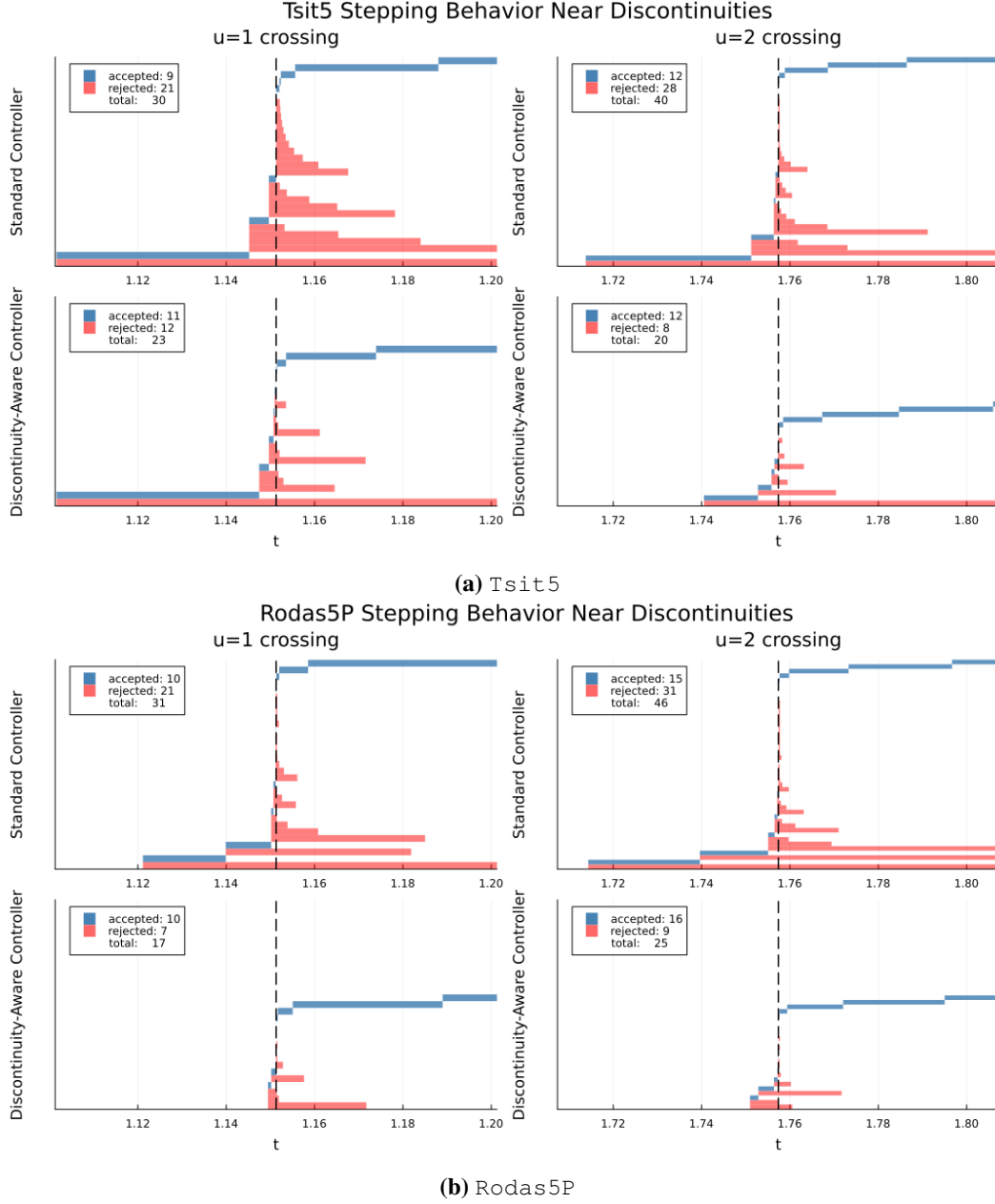


Figure 5. Time steps of the Tsit5 solver and the Rodas5P solver. The y-axis shows the number of trial time steps, each horizontal bar spans a proposed time step with the bars classified by whether the time step was accepted (blue) or rejected (red). The locations of the discontinuities are marked by the vertical dashed line (note that these are only known in the state space; the corresponding location of the discontinuity in time is not known before solution)

rejected steps for both Tsit5 and Rodas5P solvers with discontinuity-aware and standard time-step controllers. For both Tsit5 and Rodas5P, we note that discontinuity-awareness results in significantly fewer rejected steps. Similarly, discontinuity-awareness results in fewer function evaluations which are a major contributor to solver times. Figure 7 presents a work-precision diagram: The x-axis denotes the L2-norm error of the solution versus a ground-truth solution while the y-axis denotes the solver time. The results on the work-precision diagrams build on this as the performance of the discontinuity-aware solver is consistently faster than the

standard solver.

An interesting point to note is that the benefit for the implicit method, Rodas5P, appears larger than the benefit for the explicit method, Tsit5. This gives useful insight into when the discontinuity detection scheme adds value: on an implicit method, with a higher cost per rejected step, the discontinuity-aware step-size controller has a positive effect. On the other hand, while the scheme still helps for an explicit solver, we see a more modest improvement, as the rootfinding overhead introduced by the discontinuity detection scheme is non-trivial relative to the actual step cost.

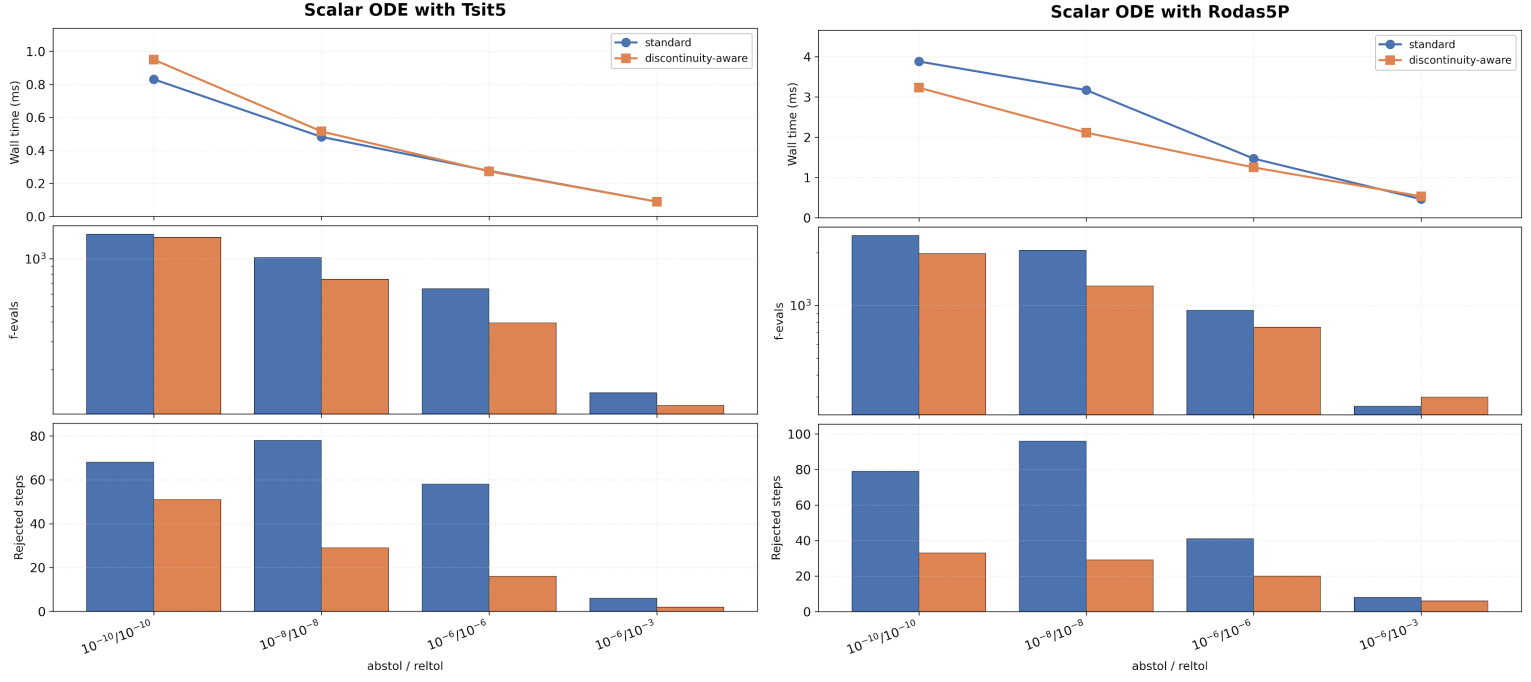


Figure 6. Wall time (ms), number of function evaluations and number of rejected steps at different absolute and relative tolerances with a discontinuity-aware and standard time-step controller. Metrics for `Tsit5` are shown on the left while `Rodas5P` are on the right.

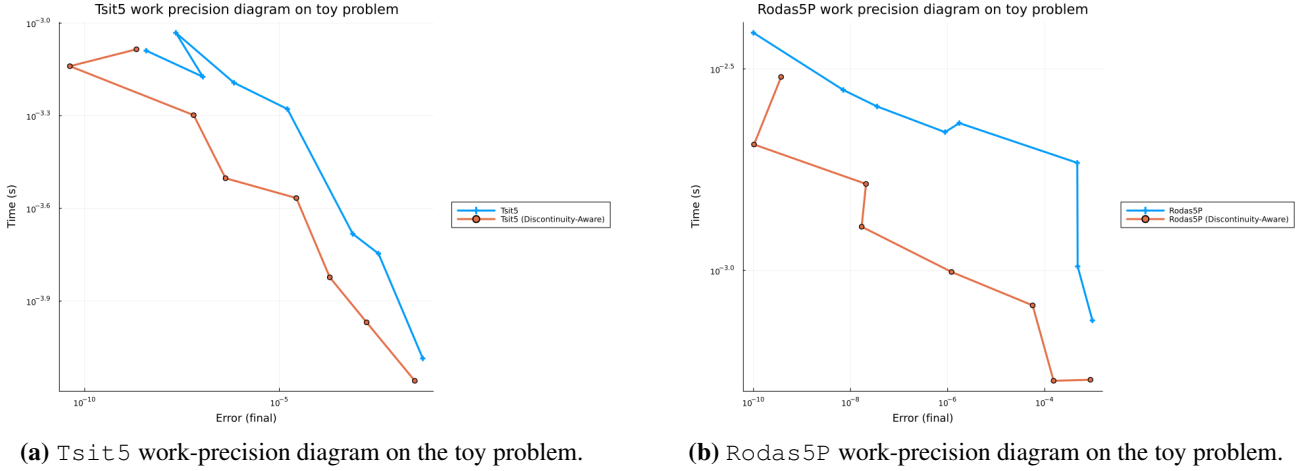


Figure 7. Work-precision diagrams for `Tsit5` and `Rodas5P` on the toy problem. Both plots are taken with `reltol` ranging from 10^{-2} to 10^{-9} , and `abstol` ranging from 10^{-4} to 10^{-11} . The vertical axis shows solve time and the horizontal axis shows the final error in the solution.

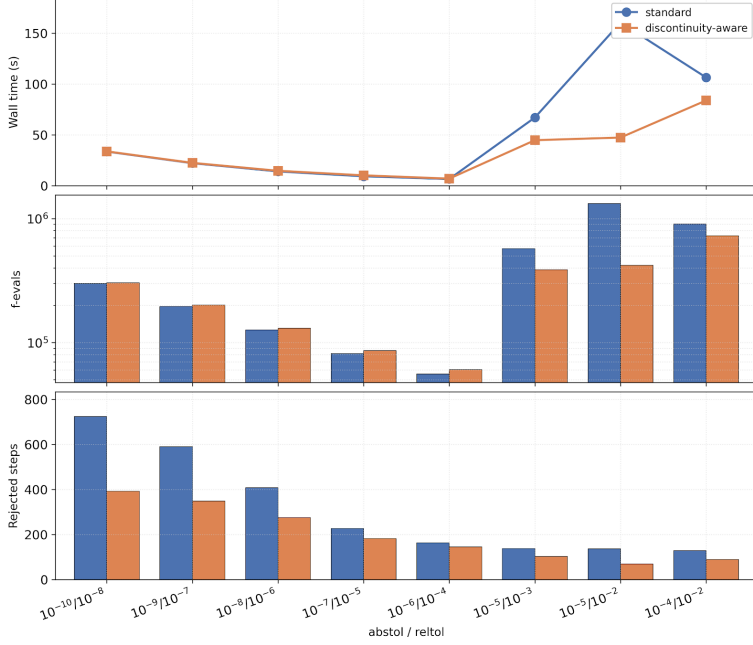
4.2 Vapor-compression cycle

Figure 8 presents the solver metrics for the vapor-compression cycle with the `Rodas5P` and the `RadauIIA5` solver. Note that our results are shown as averages over 15 runs at each tolerance level, as the solver performance is strongly dependent on the solver initialization; some solves perform poorly with discontinuity-awareness and cleanly otherwise, while the opposite also occurs. We therefore analyze general trends for the clarity of results, rather than the individual run variance.

We first analyze the results for `Rodas5P`. Consider-

ing the loose tolerance regimes, we observe performance improvements with the discontinuity-aware step-size controller, with 2x speedups on average across the low tolerances. The poor performance of the standard solver in these regimes likely stems from the fact that the standard controller takes large steps at loose tolerances and incurs high error around discontinuities. While the loose tolerances allow the solver to accept steps despite high error near discontinuities, the large error estimates cause the controller to dramatically shrink subsequent steps (even though the steps are accepted), leading to inefficient navigation of the discontinuity region.

Vapor-compression cycle model with Rodas5P



Vapor-compression cycle model with RadauIIA5

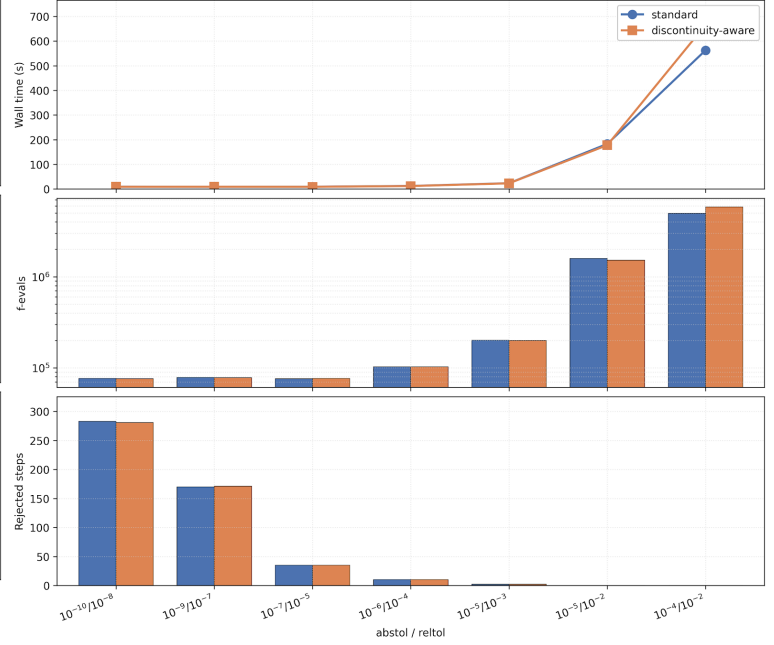


Figure 8. Wall time (ms), number of function evaluations and number of rejected steps at different absolute and relative tolerances with a discontinuity-aware and standard time-step controller. Metrics for Rodas5P are shown on the left while RadauIIA5 are on the right. Metrics are calculated as the average over 15 runs with varying initialization conditions.

Further consideration of these results suggests that the full value of the discontinuity-aware scheme lies in its ability to prevent "stagnations", or larger clusters of tiny accepted steps around discontinuities, which cause solve times to blow up. This behavior was seen in the toy problem above, and the table below shows the tail behavior of these test runs on the full model.

abstol / reltol	Median		P75		P90		Max	
	ND	D	ND	D	ND	D	ND	D
Default	13.2	9.8	38.8	13.3	445.8	81.5	916.9	804.5
$10^{-4} / 10^{-2}$	42.0	69.1	132.2	117.1	168.5	170.4	651.3	217.3
$10^{-5} / 10^{-2}$	34.7	32.5	122.9	50.0	322.7	98.8	1245.9	150.5
$10^{-5} / 10^{-3}$	22.6	24.9	59.8	27.8	223.0	30.5	304.3	380.4
$10^{-6} / 10^{-4}$	6.3	6.9	6.4	7.0	6.9	7.2	9.4	7.6

Table 1. Rodas5P solve time (s) distributions over 15 runs on the complete HVAC model. ND = standard controller, D = discontinuity-aware controller. P75 represents the 75th percentile, while P90 is the 90th percentile.

Strong evidence of the discontinuity-aware scheme's benefits at high tolerances is manifest in this tail behavior. For example, the default tolerance is 5x faster at the 90th percentile of runs, and $10^{-5}/10^{-3}$ is 8x faster at the same percentile. Furthermore, the maximum solve time at $10^{-5}/10^{-2}$ is likewise 8x faster with discontinuity-awareness. The 75th percentile is about 2x faster across all of the tolerances.

While the differences are negligible in the median performance, but the discontinuity-aware scheme performs significantly better when failures and difficulties occur. This serves as key evidence for the benefits of this scheme, as the bounds for the worst case solve time are tighter than

they are for the existing schemes. The successful recovery of a solve that took 20 minutes at $10^{-5}/10^{-2}$ in only 2.5 minutes with our scheme is the strongest indicator of this claim.

In comparison, we see that the discontinuity-aware scheme still rejects far fewer steps with Rodas5P at tight tolerances, but to negligible benefit, and even minor slowdowns. At these tight tolerances, both schemes are forced to take smaller steps; this setting favors the standard scheme, as the cost of rejections decreases. The benefits of our algorithm are minor here, but the overhead is also limited.

We note an interesting contrast between the results with Rodas5P and RadauIIA5. The differences between the standard and discontinuity-aware solvers are negligible across the board with RadauIIA5, with solve times being close to identical and step rejection percentages likewise very similar. Interestingly, the discontinuity-aware scheme often increases both step acceptances and step rejections, but the cost of this appears to be mitigated with each individual step being cheaper. This is apparent when considering the function evaluations with RadauIIA5, as there are minimal increases and even decreases in function evaluations despite the increase in steps. This is likely because of the more efficient navigation of discontinuities by the discontinuity-aware scheme, allowing the Newton iterations to converge more easily and hence reducing function evaluations.

At tight tolerances, RadauIIA5 appears to be more robust than Rodas5P. This difference can be understood as a product of the fact that RadauIIA5 is a

fully implicit Runge-Kutta method, while Rodas5P is linearly implicit. Rodas5P benefits dramatically from discontinuity-awareness at loose tolerances because its single linearization per step is particularly vulnerable to discontinuities, as it takes large steps and hence incurs severe error. On the other hand, RadauIIA5 shows little benefit because the full Newton iteration more effectively handles the discontinuity internally, converging to a valid solution even across discontinuities, at the cost of more iterations but without catastrophic step rejection. Again, this is likely because both schemes are already exposed to discontinuities through the callback mechanism, and differ in how they utilize this information on rejected steps. For fully implicit methods like RadauIIA5, this appears to yield minimal benefit, while there are large gains for linearly implicit methods like Rodas5P. At tight tolerances, both take smaller steps, yielding similar results.

5 Conclusions

We have presented a discontinuity-aware ODE and DAE solver in which user-declared information is used to compute appropriate time steps at discontinuities, and is now available in the open-source `DifferentialEquations.jl` ecosystem. We also demonstrated how this information can be specified symbolically in the Dyad declarative modeling language and passed to the solver.

Benchmarking on a scalar ODE and a vapor-compression cycle DAE model shows that the scheme yields significantly fewer step rejections and improved solver times, with the largest gains observed for Rodas5P at loose tolerances, where we see speedups of up to 8x in the tail behavior of HVAC models. The contrast between Rodas5P and RadauIIA5 reveals an important practical guideline: discontinuity-awareness provides the greatest benefit for linearly implicit methods at loose tolerances, where large step sizes make the solver particularly vulnerable to derivative discontinuities, while fully implicit methods are more naturally robust. Furthermore, the analysis of Rodas5P shows the critical value of the discontinuity-aware scheme in preventing large solve failures, which somewhat mitigates the challenges of difficult integration problems.

References

- Brenan, Kathryn Eleda, Stephen L Campbell, and Linda Ruth Petzold (1995). *Numerical solution of initial-value problems in differential-algebraic equations*. SIAM.
- Carver, MB (1978). “Efficient integration over discontinuities in ordinary differential equation simulations”. In: *Mathematics and Computers in Simulation* 20.3, pp. 190–196.
- Cellier, François E (1979). “Combined continuous/discrete system simulation by use of digital computers”. In.
- Dieci, Luca and Luciano Lopez (2012). “A survey of numerical methods for IVPs of ODEs with discontinuous right-hand side”. In: *Journal of Computational and Applied Mathematics* 236.16. 40 years of numerical analysis: “Is the discrete world an approximation of the continuous one or is it the other way around?”, pp. 3967–3991. ISSN: 0377-0427. DOI: <https://doi.org/10.1016/j.cam.2012.02.011>. URL: <https://www.sciencedirect.com/science/article/pii/S0377042712000684>.
- Fredriksson, Emil, Christian Andersson, and Johan Åkesson (2014). “Discontinuities handled with events in Assimulo”. In: *10th International Modelica Conference*. Linköping University Electronic Press, pp. 827–836.
- Gear, Charles William and Ole Østerby (1984). “Solving Ordinary Differential Equations with Discontinuities”. In: *ACM Transactions on Mathematical Software* 10.1, pp. 23–44. DOI: 10.1145/356068.356071.
- Guglielmi, Nicola and Ernst Hairer (2008). “Computing breaking points in implicit delay differential equations”. In: *Advances in Computational Mathematics* 29.3, pp. 229–247.
- Gustafsson, Kjell (1991). “Control theoretic techniques for step-size selection in explicit Runge-Kutta methods”. In: *ACM Transactions on Mathematical Software* 17, pp. 533–554. DOI: 10.1145/210232.210242.
- Gustafsson, Kjell (1994). “Traps and pitfalls in simulation”. In: *SIMS (Scandinavian Simulation Society) Simulation Conference*.
- JuliaHub (2026). *Dyad HVACComponents Library*. <https://help.juliahub.com/dyad/dev/stdlib/HVACComponents/>. URL: <https://help.juliahub.com/dyad/dev/stdlib/HVACComponents/>.
- Laughman, Christopher R and Hongtao Qiao (2017). “On the influence of state selection on mass conservation in dynamic vapour compression cycle models”. In: *Mathematical and Computer Modelling of Dynamical Systems* 23.3, pp. 262–283.
- Modelica Association (2023-04). *Modelica – A Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.6*. Tech. rep. Linköping: Modelica Association. URL: <https://specification.modelica.org/maint/3.6/MLS.pdf>.
- Mørk, Jenny M., Tor Nordam, and Siren Rühs (2025). “Handling discontinuities in numerical ODE methods for Lagrangian oceanography”. In: *EGUsphere*. preprint. DOI: 10.5194/egusphere-2025-2109.
- Otter, Martin, Hilding Elmqvist, and Sven Erik Mattsson (1999). “Hybrid modeling in Modelica based on the synchronous data flow principle”. In: *Proceedings of the 1999 IEEE International Symposium on Computer Aided Control System Design (Cat. No. 99TH8404)*. IEEE, pp. 151–157.
- Park, Taeshin and Paul I Barton (1996). “State event location in differential-algebraic models”. In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 6.2, pp. 137–165.
- Petzold, Linda (1982). “Differential/algebraic equations are not ODE’s”. In: *SIAM Journal on Scientific and Statistical Computing* 3.3, pp. 367–384.
- Qiao, Hongtao, Vikrant Aute, and Reinhard Radermacher (2015). “Transient modeling of a flash tank vapor injection heat pump system – Part I: Model development”. In: *International Journal of Refrigeration* 49, pp. 169–182. ISSN: 0140-7007. DOI: <https://doi.org/10.1016/j.ijrefrig.2014.06.019>.
- Rackauckas, Christopher, Anas Abdelrehim, et al. (2025). *Software-Defined Machines with Dyad*. Preprint. ResearchGate. DOI: 10.13140/RG.2.2.15807.57760. URL: https://www.researchgate.net/publication/398814645_Software-Defined_Machines_with_Dyad.

- Rackauckas, Christopher and Qing Nie (2017). “Differential-equations.jl—a performant and feature-rich ecosystem for solving differential equations in julia”. In.
- Rackauckas, Christopher, Qing Nie, and SciML Contributors (2024). *DifferentialEquations.jl Callback Functions*. https://docs.sciml.ai/DiffEqDocs/stable/features/callback_functions/. Accessed: 2026-05-31.
- Rasmussen, Bryan P (2012). “Dynamic modeling for vapor compression systems—Part I: Literature review”. In: *HVAC&R Research* 18.5, pp. 934–955.
- Rasmussen, Bryan P and Bhaskar Shenoy (2012). “Dynamic modeling for vapor compression systems—Part II: Simulation tutorial”. In: *HVAC&R Research* 18.5, pp. 956–973.
- Tummescheit, Hubertus and Jonas Eborn (2002). “Chemical reaction modeling with ThermoFluid/MF and MultiFlash”. In.

A Vapor-compression cycle model

A.1 Tube-fin heat exchangers

Both the condenser and the evaporator are tube-fin heat exchangers comprised of the refrigerant-side flow, the tube wall, and the air-side flow.

Refrigerant flow. The refrigerant flow is 1D thermal-fluid flow. Mass, momentum, and energy conservation are then governed by

$$\frac{\partial(\rho A)}{\partial t} + \frac{\partial(\rho A v)}{\partial x} = 0, \quad (7)$$

$$\frac{\partial(\rho v A)}{\partial t} + \frac{\partial(\rho v^2 A)}{\partial x} = -A \frac{\partial P}{\partial x} - F_f, \quad (8)$$

$$\frac{\partial(\rho u A)}{\partial t} + \frac{\partial(\rho v h A)}{\partial x} = v A \frac{\partial P}{\partial x} + v F_f + \frac{\partial Q}{\partial x}. \quad (9)$$

Discretising each refrigerant circuit yields an ordinary differential equation of the form for the mass of volume j , for instance,

$$\frac{d(\rho_j V_j)}{dt} = \dot{m}_k - \dot{m}_{k+1}, \quad (10)$$

with the finite volume enthalpy flow evaluated by up-wind discretization. The frictional pressure drop in both the single- and two-phase regions is taken as a scaled quadratic relation,

$$\Delta P = K \frac{(\Delta P)_0}{\dot{m}_0^2} \dot{m}^2, \quad (11)$$

and the refrigerant heat-transfer coefficient is assigned a constant value within each phase. The derivative discontinuities at the bubble and dew curves discussed in Section 2.2 enter the assembled DAE (3) through refrigerant property computations.

Tube wall. Conductive heat transfer through the property wall takes the form

$$(m_w c_w + m_f c_f) \frac{dT_w}{dt} = \alpha_{\text{ref}} A (T_r - T_w) + \dot{m}_a [c_{p,a} (T_{a,\text{in}} - T_{a,\text{out}}) + (\omega_{a,\text{in}} - \omega_{a,\text{out}}) \Delta h_{fg}]. \quad (12)$$

Air-side flow. The air stream crosses the fin side and exchanges both sensible and latent heat with the wall using

replaceable heat and mass transfer correlations

A.2 Compressor

The compressor is a positive-displacement machine modelled algebraically, with no mass or energy storage. Its behaviour is set by two performance maps fitted to manufacturer data — a volumetric efficiency, which fixes the refrigerant mass flow, and an isentropic efficiency, which fixes the discharge enthalpy:

$$\eta_v(f, p_r, \dots) = \frac{\dot{m}_{\text{comp}}}{\rho_{\text{suc}} V_f}, \quad (13)$$

$$\eta_{\text{is}}(f, \dots) = \frac{h_{\text{dis,isen}} - h_{\text{suc}}}{h_{\text{dis}} - h_{\text{suc}}}, \quad (14)$$

where V_f is the swept volume rate at shaft frequency f , $h_{\text{dis,isen}}$ is the discharge enthalpy of the equivalent isentropic compression, ρ_{suc} is the suction-side density, and both efficiencies are maps in f and the pressure ratio p_r .

A.3 Expansion valve

The LEV is modeled as an isenthalpic, algebraic orifice without mass or energy storage. The mass flow follows the standard orifice relation

$$\dot{m} = C_v(\theta) A_v \sqrt{\rho_{\text{in}} \Delta P}, \quad (15)$$

in which $C_v(\theta) A_v$ is the effective flow coefficient — a smooth, differentiable function of the valve position θ — ρ_{in} is the upstream density, and ΔP is the pressure drop across the valve.