1 Discontinuity in the PDE case

1.1 Introduction

In this chapter, we explore the effects of discontinuities on a 1D PDE (partial differential equation) model. We will use an epidemiological model to which we will add a time-dependent discontinuity and then a state-dependent discontinuity and will report on the accuracy and efficiency of the BACOLIKR PDE solvers.

We note that PDE models are often used in epidemiological studies but are very rarely solved with state of the art solvers. In this chapter we will use BACOLIKR, see a description of the solvers used in Section 1.1.1, which is error-controlled and has event detection.

As was the case with the IVODE solvers in the previous chapter, PDE solvers also thrash when they encounter a discontinuity, See Figure 1. As is the case with the IVODE solvers, the PDE solvers use methods whose underlying mathematical theories require that the model being solved as well as some of its partial derivatives are continuous. Discontinuities imply that the PDE solvers are not guaranteed to converge but as we have shown in the IVODE case, error-control still allows error-control to give accurate solutions at the cost of 'thrashing', repeated time step-size resizing until the error-estimates calculated by the solvers satisfy the user-provided tolerance.

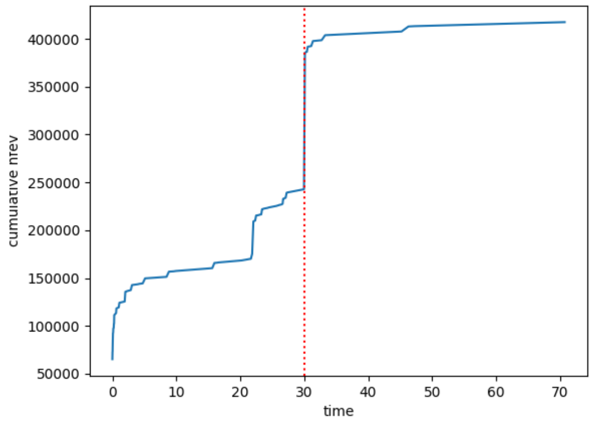

Figure 1: Thrashing in the PDE context

We will show, as was the case in the IVODE problem, that PDE solvers with error-control can integrate through one time-dependent discontinuity but that discontinuity handling lead to more efficient solutions and that event detection is required to integrate state-dependent discontinuity problem with the best accuracy-efficiency trade-offs.

1.1.1 Software used

BACOLIKR BACOLIKR implements a spline collocation method at Gaussian points, with a B-spline basis, for the spatial discretization. The time integration is performed using a modification of the popular DAE solver with root-finding capabilities, DASKR [Need a reference].

It does error control by computing a second, higher order, global solution. A high quality spatial error estimate is obtained after each successful time step. The spatial error is controlled by a sophisticated new mesh selection algorithm based on an equi-distribution principle.

1.1.2 Problem Definition

In this paper, the PDE model we will try to solve is an extension of the SEIR model for epidemiological studies that uses one more spatial variable, x, and time. Similar PDE models have been used before 8888 reference to PDE Cholera papers 8888 and either use the spread in location or the age as the additional spatial variable.

The SEIR model was developed by Andrew Fraser 88888 Reference 88888 but some coefficients were changed.

The model is as follows:

$$S(x,t)_t = D_S(x)S(x,t)_{xx} + \mu N - \mu S(x,t) - \frac{\beta}{N}S(x,t)I(x,t) \tag{1}$$

$$E(x,t)_t = D_E(x)E(x,t)_{xx} + \frac{\beta}{N}S(x,t)I(x,t) - \alpha E(x,t) - \mu E(x,t) \tag{2}$$

$$I(x,t)_t = D_I(x)I(x,t)_{xx} + \alpha E(x,t) - \gamma I(x,t) - \mu I(x,t) \tag{3}$$

$$R(x,t)_t = D_R(x)R(x,t)_{xx} + \gamma I(x,t) - \mu R(x,t) \tag{4}$$

The spatial domain is $-5 \leq x \leq 5$ and the temporal domain is $0 \leq t \leq 70$ for the time-dependent discontinuity problem but we will use $0 \leq t \leq 200$ for the space-dependent discontinuity problem as we attempt a long-term forecast.

The parameters are as such: μ, the birth rate, is set to $\frac{0.01}{365}$. γ, the recovery rate is 0.06, α, the incubation rate is 0.125 and we will vary the transmission rate, β, between 0.035 and 0.9 based on whether measures are implemented or not in the model. The population size, N, is $37 * 10^6$.

The model also uses diffusion functions $D_S(x)$, $D_E(x)$, $D_I(x)$ and $D_R(x)$ as follows:

$$D_S(x) = D_E(x) = D_R(x) = (maxD_s - minD_s)e^{-10(\sqrt{x^2}-1)^2)} + minD_s \tag{5}$$

$$D_I(x) = D_E(x)/10 \tag{6}$$

The parameters $maxD_s$ and $minD_s$ are 0.8 and 0.01 respectively.

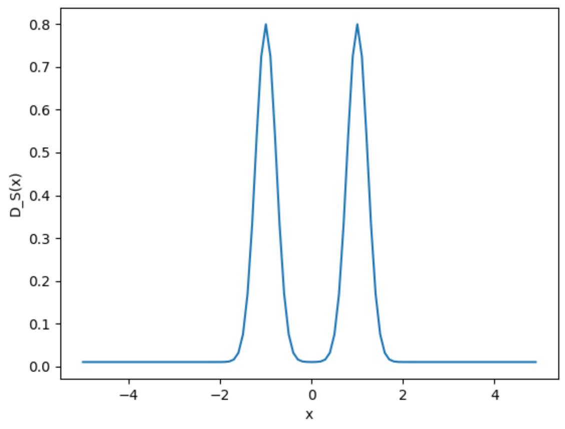

Figure 2: Plot of the diffusivity parameter $D_S(x)$

The initial values are functions of the spatial domain as such:

$$S(x,0) = N - I(x,0) \tag{7}$$

$$I(x,0) = 100e^{-x^2} \tag{8}$$

$$E(x,0) = R(x,0) = 0 \tag{9}$$

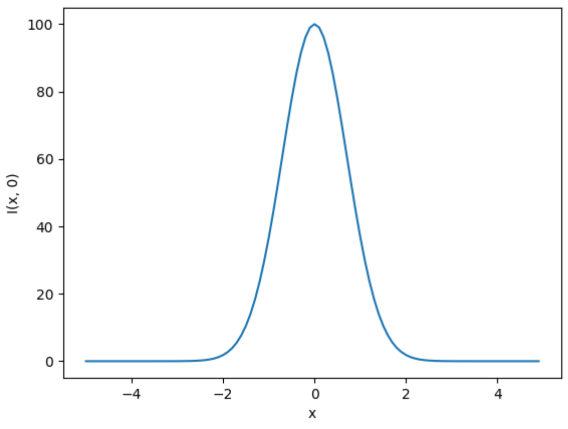

Figure 3: Plot of the initial condition I(x, 0)

This gives us a complete PDE problem definition to which we will add discontinuities as follows.

In the time-dependent discontinuity problem, we will integrate the model with β at a value of 0.9 from t=0 to t=30, we will then change the value of β to 0.035 and integrate until t=70. This change in the parameter introduces a discontinuity.

In the state-dependent discontinuity problem, we start the integration with the value of β at 0.9 until the integral value of the E-component of the solution at a specific time-step is 30000. When it reaches 30000, we change the value of β to 0.035 until the integral reaches 10000. We then integrate with a value β of 0.9. We repeat this process until t=200.

1.2 Time Dependent Discontinuity

In this section, we will add a time dependent discontinuity and report on the thrashing experienced by the solvers. We note that this section is in essence an application of what we demonstrated in Section 8888 reference to ODE time dependent 8888 in that time dependent discontinuities are introduced by simply changing a parameter value, as this is essentially changing the model function being integrated and that time-dependent discontinuities have an easy solution by integrating the solution with two calls, one before the discontinuity and one after. This gives the solver two continuous segments to integrate.

In this PDE case, the value of β will be 0.9 from t=0 to t=30 and the value of β will be changed to 0.035 from t=30 to the end of the integration.

1.2.1 Naive treatment of the time-dependent discontinuity PDE model

In the naive treatment for this kind of discontinuity is to place the the change in the parameter in the right hand side function as an if-statement. The pseudocode for this approach is as follows:

```
function model_with_if(t, x, u, ux, uxx)
    // ...
    beta = 0.9
    if t >= 30:
        beta = 0.035
    // ...
```

This change in the parameter β at t=30 introduces a discontinuity as the model function is different. As shown in Section 8888 Refer to section on discontinuity 8888, as the assumptions of continuity of the function and its derivatives no longer hold, the Taylor Series proof of its convergence is no longer valid. However as we have shown in Section 8888 Refer to naive ODE time problem 8888, error-control solver can reduce the step-size extensively to cross one discontinuity.

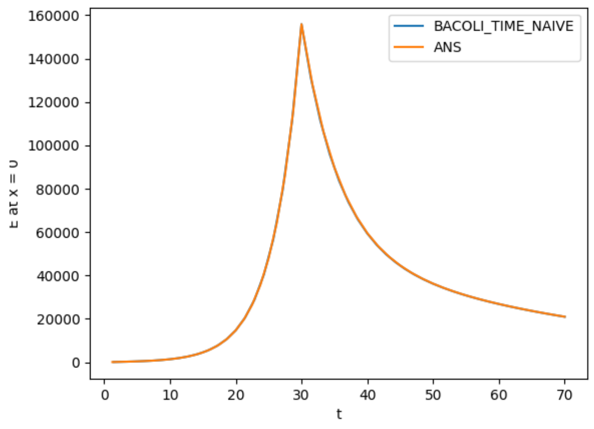

Figure 4: Naive treatment of time discontinuity (at tolerance of 10^{-6})

From Figure 4, we can see that to eye-level accuracy, the results are accurate. We will now show how a cold start can keep the same level of accuracy while using less function evaluations and thus be more efficient.

1.2.2 Discontinuity handling to solve the time-dependent PDE model

Though the error-controlled solvers were able to get accurate solutions, we now solve the same problem using discontinuity handling. Modern solvers like BACOLIKR allows users to set flags to allow it to do a cold start. Thus we integrate the problem with one call from t=0 to t=30 with the model function using 0.9 as the β parameter. We then set up a cold start and integrate from t=30 to the end of the time interval with another call to the solver. The pseudo-code is as follows

```
function model_before(t, x, u, ux, uxx):
    // ...
    beta = 0.9
    // ...

function model_after(t, x, u, ux, uxx):
    // ...
    beta = 0.035
    // ...

tspan_before = [0, 30]
pde_solver(solution, model_before, tspan_before)

solution.cold_start_flag = True

tspan_after = [30, 70]
pde_solver(solution, model_after, tspan_after)
```

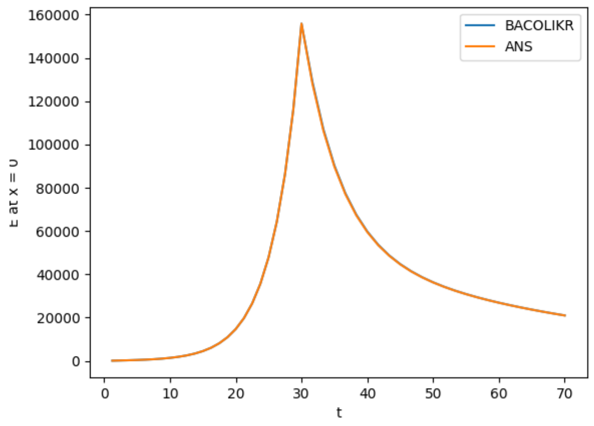

Figure 5: Discontinuity handling for time discontinuity problem (at tolerance of 10^{-6})

As expected, Figure 5, the solvers were able to all accurately solve the problem. However, Table 1 shows how the discontinuity handling allows us to use significantly lower numbers of function evaluations.

We note that without the cold start, the solver used 417505 function evaluations. With a cold start the solver uses 359755 function evaluations. We can see that the cold start improved efficiency.

1.2.3 PDE Time-dependent discontinuity problem tolerance study

In this section, we do a tolerance study on the time dependent problem. We look at how coarse we can reduce the tolerance to still have accurate solution to see if the discontinuity handling allows us to use coarser tolerance as it did in the IVODE case.

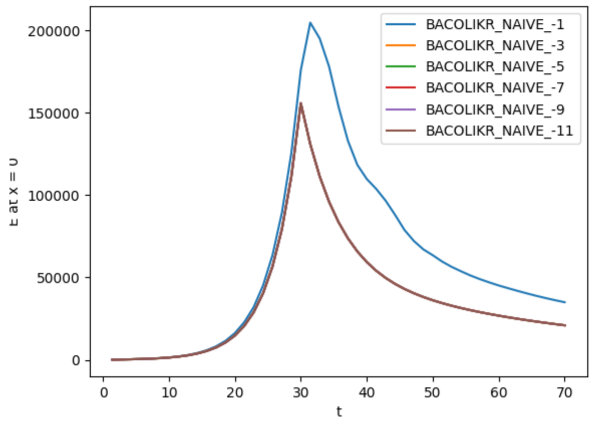

Figure 6: Time dependent discontinuity tolerance study with BACOLIKR without discontinuity handling

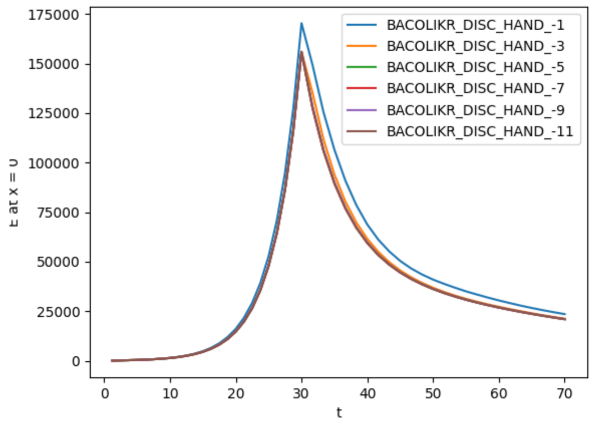

Figure 7: Time dependent discontinuity tolerance study with BACOLIKR using discontinuity handlng

Table 1: PDE time discontinuity tolerance study

tolerance	BACOLIKR naive nfev	BACOLIKR disc hand nfev
1e-1	40, 800	55, 400
1e-3	79, 220	76, 750
1e-5	206, 980	208, 870
1e-7	733, 210	543, 850
1e-9	1, 904, 080	1, 653, 830
1e-11	7, 140, 875	4, 979, 555

1.3 State Dependent Discontinuity

In this section, we discuss a state-dependent discontinuity problem in that we compare the current value of the state or one of its component against a predetermined threshold and if that threshold is crossed, we change the model equation. Unlike, in the IVODE case, we need to account for the spatial domain when finding the state value to compare against a threshold. Some of the ways to do so are listed below:

- Pick a spatial value, say x = 0, and sample the state value at this spatial point at every time interval. If the state value meets a certain threshold, we apply a different model, else we use the same model
- Find some statistic measure (min, max, mean, median) across the spatial domain and use that value for the comparison with the threshold.
- Integrate over the spatial domain and use that integral value for the comparison against the threshold.

In this report we will use the third method in that we will integrate over the spatial domain. If the value of the integral crosses a maximum threshold (integral value of 5) while measures are not implemented, the value of the parameter β is changed from 0.9 to 0.005 and if measures are implemented and we cross a certain minimum threshold (integral value of 1), the value of the parameter β is changed back to 0.9. This is repeated several time for the time period t=0 to t=200.

We note that the discontinuity is introduced by the change in the parameter β and which method to obtain the state value at a certain time does not matter, so the conclusions of this paper equally applies to the other methods of finding the state value at a given time.

1.3.1 Naive treatment of the state-dependent discontinuity model

For the naive treatment of this problem, a user will use a boolean for whether measures are implemented or not, as a global variable. This global variable is toggled based on the integral value over the spatial domain at a given time.

To perform the integration the naive user will have to do a manual time stepping. The user will divide the time into equal intervals and once they reach the end of the time interval, they will make an interpolant over the spatial domain at that time and integrate it.

If measures are not implemented and the integral is greater than 5, the maximum threshold, they will switch the global variable indicating that measures are implemented. When there are no measures implemented, the user will integrate at the end of a step and look for an integral value less than 1, the minimum threshold. When such an integral value is less than 1, the user will switch the global variable indicating that measures are no longer implemented. When measures are implemented, the value of the parameter β is 0.005 while it is 0.9 when measures are not implemented.

The pseudocode for this approach is as shown:

```
measures_implemented = False

function model(t, x, u, ux, uxx):
    // ...
    if (measures_implemented):
                beta = 0.035
        else:
                beta = 0.9
    // ...

tstart = 0
tstop = 200
num_times = 400
time_step_size = (tstop - tstart) / num_times

t_current = tstart
t_next = t_current + time_step_size

while t_current < tstop:
        tspan = [t_current, t_next]
        pde_solver(solution, model, tspan)

        integral_value = integrate(interpolate(solution))

        if (measures_implemented):
                if (integral_value >= 6):
                        measures_implemented = True
        else:
                if (integral_value <= 1):
                        measures_implemented = False

        t_current = t_next
        t_next = t_next + time_step_size
```

The pseudo-code shows a clear problem with the naive solution. The user has an additional variable, the number of time steps, to set. The number of time steps needs to be big enough that we know when the threshold are met but need to be small enough that the integration can occur without issue.

The solution on solving this problem with a naive solution is as shown in Figures 8888 8888 and 8888 8888 where the number of time steps is 200 and 400 respectively.

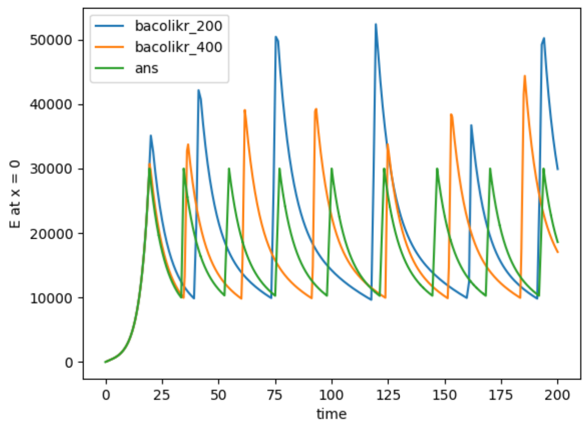

Figure 8: State dependent discontinuity naive treatment with a tolerance of 10^{-6} and 200 vs 400 time steps

1.3.2 Why the naive method cannot give an accurate solution

The naive method cannot solve the problem accurately because of the problem of choosing a correct number of time steps to perform the spatial integration. We note that most of the time, the method will find the correct integral value much after it crosses the threshold and not exactly when it does so. This means that we take up to one additional time step with the previous β value and not the correct one.

One idea to solve this problem would be to use an exceedingly large number of time steps (1000 in our case) such that we take the smallest step possible with the old value. This, however, reduces the efficiency, making us do more time steps than necessary. (See efficiency comparisons in Table 8888 Refer to table in next section 8888). The best solution is to use event detection.

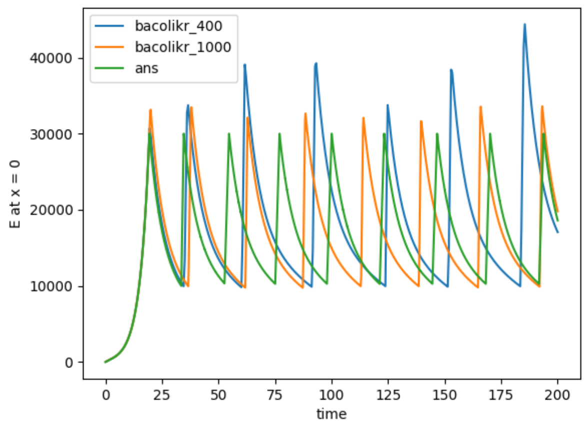

Figure 9: State dependent discontinuity naive treatment with a tolerance of 10^{-6} and 400vs1000 time steps

From the above, we can see that a high enough number of time steps allow us to get better and better solution...

1.3.3 Event Detection solution to the naive state-dependent discontinuity model

As was the case in the IVODE case, event detection is also present in some PDE solvers. Event detection works in the same way: the user provides a root function to the PDE solver, after each time step the solver calls the root function with the solution at the current time step and stores its value. If the value returned by the root function changes sign, the PDE solvers employs a root-finding routine to find where the root function is zero exactly. The solver then returns, setting a flags indicating that it has found a root with the values at the root.

BACOLIKR is an improvement to the BACOLI solver which has root finding capabilities. Instead of using DASSL as its DAE solver, it uses DASKR which can detect roots as its solves a DAE system. We use BACOLIKR to demonstrate that using a PDE SOLVER with root-finding capabilities allows us to not define a time grid and thus allows us to integrate with the best accuracy-efficiency trade-off.

We define two pairs of root and model functions. One pair is to be used when integrating when there are no measures in place. The model function will have the variable β at a value of 0.9 and the root function will do the integration of the spatial domain at the current time step and will return if it is closed to the maximum threshold. 5. The second pair will have the model function with β at a value of 0.005 and the root function looking for a root at 1. The pseudo-code for this approach is as follows:

```
function model_no_measures(t, x, u, ux, uxx):
        // ...
        beta = 0.9
        // ...

function root_max_value(t, solution):
        // ...
        integral_value = integrate(interpolate(solution))
        return integral_value - 5

function model_with_measures(t, x, u, ux, uxx):
        // ...
        beta = 0.035
        // ...

function root_min_value(t, solution):
        // ...
        integral_value = integrate(interpolate(solution))
        return integral_value - 1


tstart = 0
tstop = 200

t_current = tstart
measures_implemented = false

while t_current < tstop:
        tspan = [t_current, t_stop]
        if (measures_implemented):
                pde_solver(solution, model_with_measures, tspan, root_min_valu
        else:
                pde_solver(solution, model_no_measures, tspan, root_max_value)

        if (solution.root_flag == True):
                # root detected, if a max root, add measures else remove meas
                solution.cold_start_flag = True
                if (measures_implemented):
                        measures_implemented = False
                else:
                        measures_implemented = True

        t_current = solution.t
```

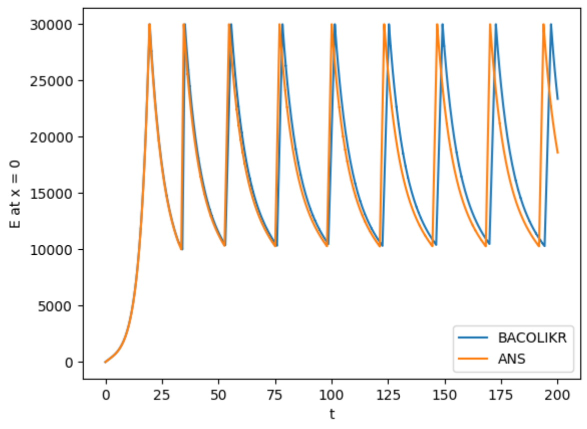

Figure 10: State dependent discontinuity using event detection with a tolerance of 10^{-6}

As we can see, we get a solution that oscillates correctly. However the two solutions are not correctly aligned, especially at lower time period. We will see that the tolerance need to be high enough

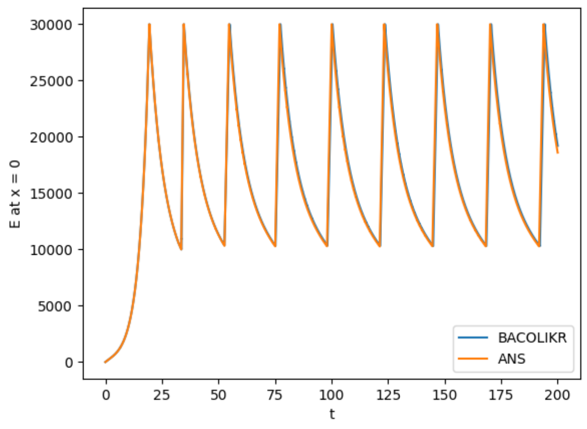

Figure 11: State dependent discontinuity using event detection with a tolerance of 10^{-9}

From Figures 10 and 11, we can see that with event detection, it is only a matter of tolerance to see whether the answer is accurate. We note that this might be because of how the root finding works. A stricter tolerance at the root makes DASKR stop closer to the root and thus begin with a different β value.

Table 2: PDE state discontinuity model

method	nfev
BACOLIKR naive 200 steps	1, 018, 455
BACOLIKR naive 400 steps	1, 184, 880
BACOLIKR naive 1000 steps	1, 280, 080
BACOLIKR tol 10^{-6}	1, 937, 730
BACOLIKR tol 10^{-9}	7, 915, 085

1.3.4 State problem tolerance study

We also perform a tolerance study at 400 and at 1000 to show how the solutions compare for all the solvers. We try to look for why even BACOLIKR does not solve the problem and look to see if there is anything we can do

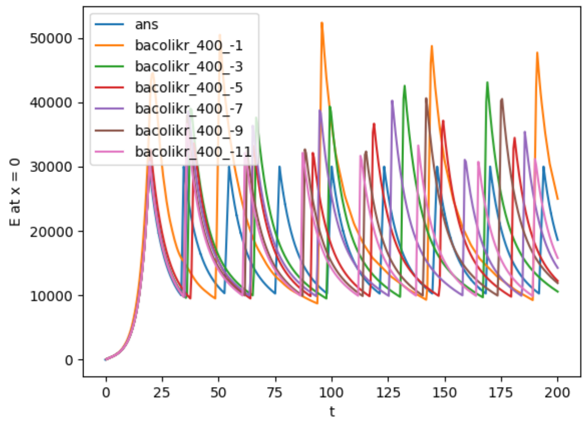

Figure 12: State dependent discontinuity naive treatment tolerance study with BACOLIKR with 400 time steps

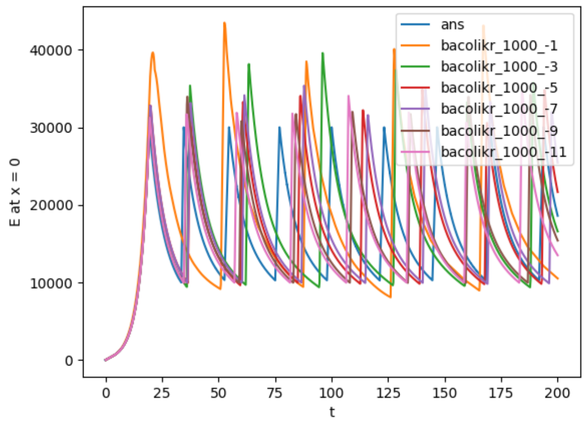

Figure 13: State dependent discontinuity naive treatment tolerance study with BACOLIKR with 1000 time steps

BACOLIKR without event Figures **??** and **??** shows that all solutions given by BACOLRI align with each others despite the tolerance. We see this phenomenon in both the 400 and 1000 time steps plots except that the 1000 time steps plot is closer to the accurate solution. We can conclude that in the BACOLRI case, the solution is entirely dependent on the number of time steps and that the tolerance is not the limiting factor driving us towards inaccurate solutions. A finer time grid will detect the integral values crossing the threshold earlier and will provide a better solution. However, as we will see below such a solution is inefficient and event detection should be prioritized.

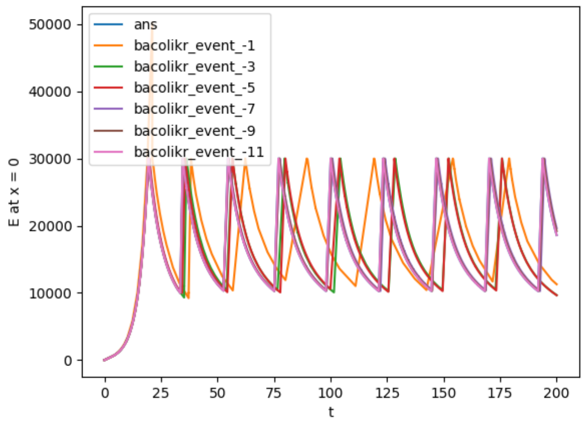

Figure 14: State dependent discontinuity using event detection tolerance study

BACOLIKR with event

Table 3: State dependent discontinuity model tolerance study

tolerance	BACOLIKR with event	BACOLIKR naive 400	BACOLIKR naive 1000
1e-1	137, 300	104, 700	101, 250
1e-3	376, 855	239, 445	290, 600
1e-5	1, 052, 920	736, 360	848, 080
1e-7	3, 217, 450	2, 032, 330	2, 217, 610
1e-9	7, 915, 085	5, 236, 500	6, 040, 485
1e-11	21, 256, 400	16, 845, 765	16, 402, 140

number of function evaluations

1.4 Discussions