1 Discontinuity in the PDE case

1.1 Introduction

In this chapter, we present a similar investigation of solutions to problems with discontinuities in the partial differential equation (PDE) case. To that effect, we will use a 1D PDE epidemiological model similar to those often used in research in that area.

Again, it is vital that numerical errors associated with the solvers are neglible compared to the modeling errors so that epidemiologist con perform conclusive errors. However, in the PDE case, we often find that rudimentary solvers (often a variation that reduce to using the Euler method) are used. As in the ODE case, it would be very suprising that these solvers are able to reach an accurate solution. To that effect, we spend Section 1.1.2 on introducing BACOLIKR and explaining its importance for the accurate solving of PDE problems. To our knowledge, it is also the only PDE solver capable of event detection which as, shown in the previous chapter, becomes vital for accurate solutions of state-dependent discontinuity problems.

In Section 1.1.1, we reintroduce thrashing for the PDE case, in Section 1.1.2, we provide the BACOLIKR description and in Section 1.1.3, we give a description of the epidemiological model used.

We provide a treatment of the time-dependent discontinuity problem with and without discontinuity handling in Section 1.2 and a treatment of the state-dependent discontinuity problem with and without event detection in Section 1.3.

1.1.1 Thrashing in PDE models

As was the case with ODE solvers, PDE solvers are also based on mathematical theories that guarantee convergence (a good solution) only when the solution and some of its higher derivatives are continuous. No rigorous mathematical theories guarantee that a PDE solver can solve a discontinuous problems but through experiments, it is known that *error-control* allows solvers to integrate through discontinuities. They do so by repeatedly reducing the step-size until the error of the new step satisfies the user-provided tolerance. Only then is the step accepted.

Thus, when a PDE solver is asked to integrate through a discontinuity, it thrashes. It repeatedly reduces the step-size at that discontinuity until the step-size is small enough to integrate through it. This is observed as a spike in the number of function evaluations at a discontinuity. Figure 1 shows such a phenomenon. A problem with a discontinuity placed at t=30 is solved and we plot the cumulative number of function evaluation at each time interval that the solver, BACOLIKR takes. (We can clearly see the spike at t=30.)

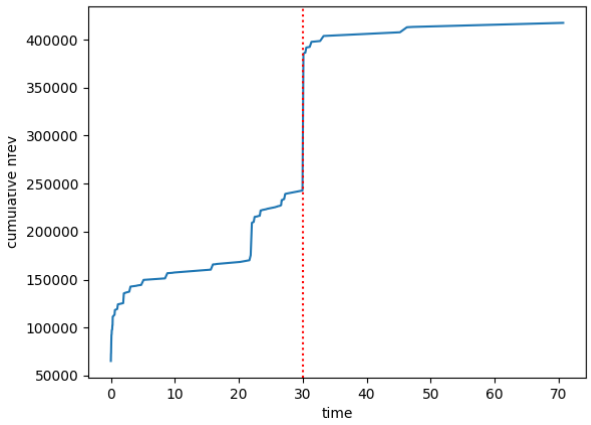

Figure 1: Thrashing in the PDE context

In this chapter, we will show that PDE solvers with error-control, like BACOLIKR, can integrate through one time-dependent discontinuity but that discontinuity handling lead to more efficient solutions as was the case with time-dependent dicontinuous ODE problems (See Section 1.2). We will also show that state-dependent discontinuity problems cannot be solved without event detection and that even with event detection, the user will have to be satisfied with an accuracy-efficiency trade-off (See Section 1.3).

1.1.2 BACOLIKR, error-control event detection PDE solver

BACOLIKR is a member of the BACOL family of PDE solvers 8888 need reference 8888. Its underlying principle is the same a BACOL in that it solves PDE problems with a spline collocation method using the B-spline bases.

The collocation process is applied on the spatial dimension to approximate the PDE system with a time-dependent larger ODE system. Using this ODE system and the provided boundary conditions, BACOLIKR produces a time-dependent system of Differential-Algebraic Equations (DAE) which it solves with the DAE solver, DASKR (a modification of DASSL with root-finding) 888 need a reference 888. DASSL provides adaptive time-stepping and adaptive method order selection (chooses an appropriate BDF method). BACOLIKR also provides control over the spatial error throug adaptive refinement of the spatial mesh.

Being an error-control solver, BACOLIKR tries to satisfy a user provided tolerance in the most efficient way possible and by using a root-finding DAE solver like DASKR, BACOLIKR can also perform event-detection and thus can be used for cold starts around discontinuities. We emphasize these two qualities of BACOLIKR as they guarantee accuracy and efficiency that other implementations of PDE solvers, especially rudimentary ones, very rarely grant.

1.1.3 Problem Definition

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

A PDE problem is fully described using a system of partial differential equations of the form:

$$u_t(x,t) = f(x,t,u(x,t),u_x(x,t),u_{xx}(x,t)) \tag{1}$$

over a spatial domain $a \leq x \leq b$ and an initial time t_0.

It requires a set of initial conditions of the form:

$$u(x,t) = u_0(x) \tag{2}$$

for x in the spatial domain, $a \leq x \leq b$.

It also requires boundary conditions of the form:

$$b_L(t,u(a,t),u_x(a,t)) = 0, b_R(t,u(b,t),u_x(b,t)) = 0 \tag{3}$$

for every time, $t \geq t_0$.

To that effect, we define an SEIR model based on the one developed by Andrew Fraser 88888 Reference 88888 as follows:

The system of PDEs is:

$$S(x,t)_t = D_S(x)S(x,t)_{xx} + \mu N - \mu S(x,t) - \frac{\beta}{N}S(x,t)I(x,t) \tag{4}$$

$$E(x,t)_t = D_E(x)E(x,t)_{xx} + \frac{\beta}{N}S(x,t)I(x,t) - \alpha E(x,t) - \mu E(x,t) \tag{5}$$

$$I(x,t)_t = D_I(x)I(x,t)_{xx} + \alpha E(x,t) - \gamma I(x,t) - \mu I(x,t) \tag{6}$$

$$R(x,t)_t = D_R(x)R(x,t)_{xx} + \gamma I(x,t) - \mu R(x,t) \tag{7}$$

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 for the SEIR 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, such as social distancing and others, 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 $D_R(x)$ to start the infection at a specific point in the spatial domain and spread it over time. These are as follows:

$$D_S(x) = D_E(x) = D_R(x) = (maxD_s - minD_s)e^{-10(\sqrt{x^2}-1)^2)} + minD_s \tag{8}$$

$$D_I(x) = D_E(x)/10 \tag{9}$$

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

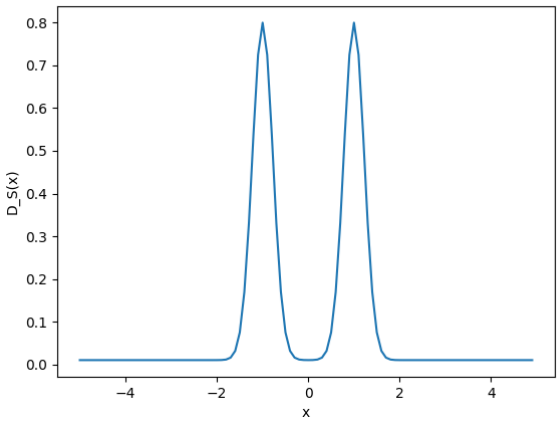

Figure 2: Plot of the diffusivity parameter $D_S(x)$

The set of initial conditions are defined over the spatial domain as such:

$$S(x,0) = N - I(x,0) \tag{10}$$

$$I(x,0) = 100e^{-x^2} \tag{11}$$

$$E(x,0) = R(x,0) = 0 \tag{12}$$

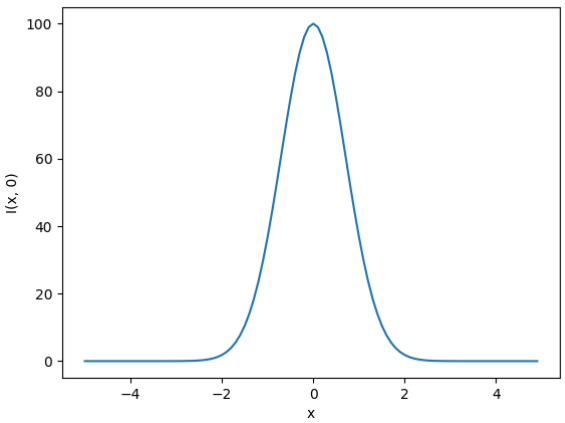

Figure 3: Plot of the initial condition I(x, 0)

We will use the default boundary conditions 888888888888888888 I did not understand what BACOLIKR is doing for the initial conditions here 888888888888888888

The above gives us a complete PDE problem definition. To this problem we will add discontinuities as follows: In the time-dependent discontinuity problem (Section 1.2), 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. This simulates a scenario where 30 days into a disease, measures are introduced to slow its spread.

In the state-dependent discontinuity problem (Section 1.3),, we start the integration with the value of β at 0.9 until the integral value of the E-component of the solution over the spatial domain at a specific time-step is 30000. When that integral value reaches 30000, we change the value of β to 0.035 until the integral value of the E-component over the spatial domain reaches 10000. We then integrate with a value β of 0.9. We repeat this process until t=200. This process simulates a series of introducing and relaxing measures, such as social distancing, based on the total number of exposed individuals across the whole region.

1.2 Time Dependent Discontinuity

In this section, we investigate the thrashing experienced by BACOLIKR on the SEIR model with a time-dependent discontinuity.

The time dependent dicontinuity is introduced by changing the value of SEIR modeling parameter, β from 0.9 to 0.035 at t=30.

We note that this section demonstrates that the PDE sing time-dependent discontnuity is akin to the ODE time-dependent discontinuity problem. We will show that accurate solutions can be found without the use of discontinuity handling (to eye-level accuracy) but that the use of a cold start drastically improves the efficiency.

1.2.1 Naive treatment of the time-dependent discontinuity PDE model

The naive treatment for time-dependent discontinuities is to use if-statements in the right-hand side function itself using the time argument.

For our model, we use the time argument to say that the integration starts with a value of β at 0.9 and that from t=30 onwards, the value for β is 0.035. The pseudo-code for this approach is as follows:

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

This change in the β parameter introduces a discontinuity in the model, which leads to thrashing as discussed in Section 1.1.1 as the assumptions of continuity of the function and its derivatives no longer hold. However as we have shown in Section 8888 Refer to naive ODE time problem 8888 for the ODE case, error-control PDE solvers can also reduce the step-size extensively to integrate through one discontinuity. (See Figure 4.)

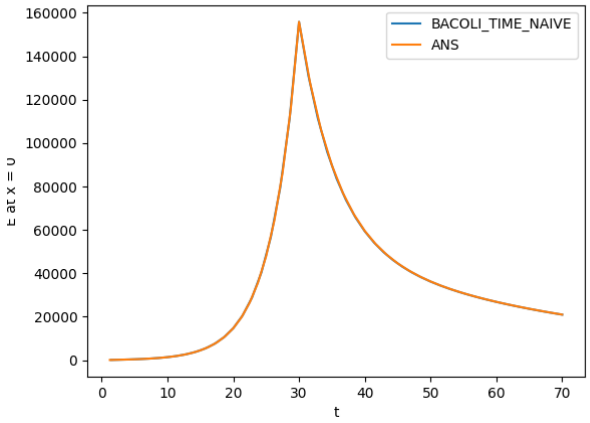

Figure 4: Naive treatment of time discontinuity (at tolerance of 10^{-6})

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

1.2.2 Discontinuity handling to solve the time-dependent PDE model

In this section, we discuss the use of discontinuity handling through cold starts as it pertains to PDE problems. Though the error-controlled solver, BACOLIKR, were able to get accurate solutions, we will show that cold starts allow it to be more efficient.

We re-iterate that a solver is said to perform a cold start when it restarts the integration by using the last previously obtained solution as the initial value for the next step. The solver must clear all its data structures, start with a new spatial mesh, start with a new initial step-size and start with the default order. This way the solver does not allow any results from previous steps to interfere with the next step. Modern PDE solvers like BACOLIKR have several flags that a user can set along the integration to perform cold starts. We will use this flag to improve the efficiency of our solutions.

The idea is to integrate up to the discontinuity, cold start at the discontinuity and integrate past the discontinuity with the new model. In our case,

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 for this approach 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
    // ...

solution = pde_solver.init(...)

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)
```

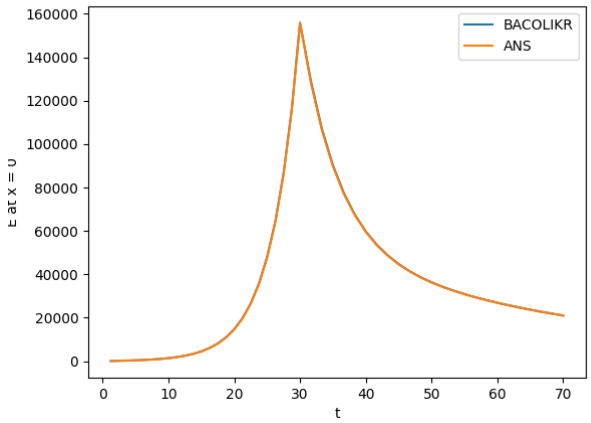

Figure 5: Discontinuity handling for time discontinuity problem (at tolerance of 10^{-6})

As expected, Figure 5 shows that BACOLIKR with the cold start was able to provide a solution with the same eye-level accuracy as it did without the

cold start. We now note that the use of a cold start required the solver to make 359, 755 function evaluations whereas the solver without this cold start required 417505 function evaluations. We were thus able to make 50000 less function evaluations which gives around a 14% gain in 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.

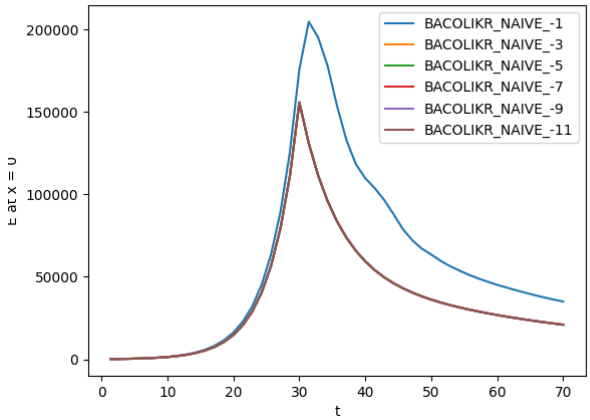

Figure 6: Time dependent discontinuity tolerance study with BACOLIKR without discontinuity handling

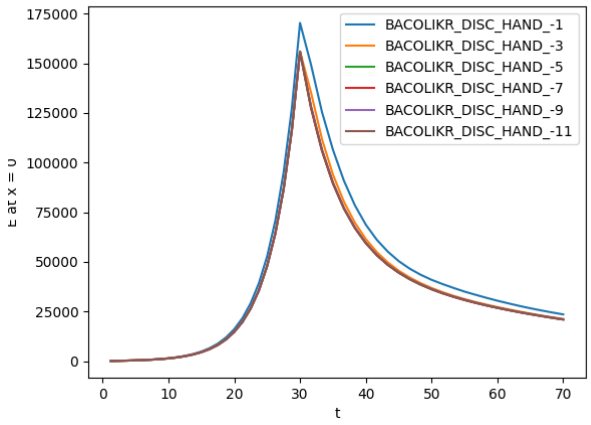

Figure 7: Time dependent discontinuity tolerance study with BACOLIKR using discontinuity handlng

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.

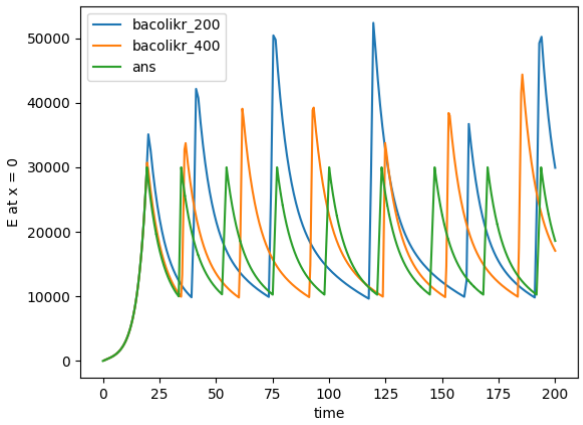

Figure 8: State dependent discontinuity naive treatment with a tolerance of 10^{-6} and 200 vs 400 time steps

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.

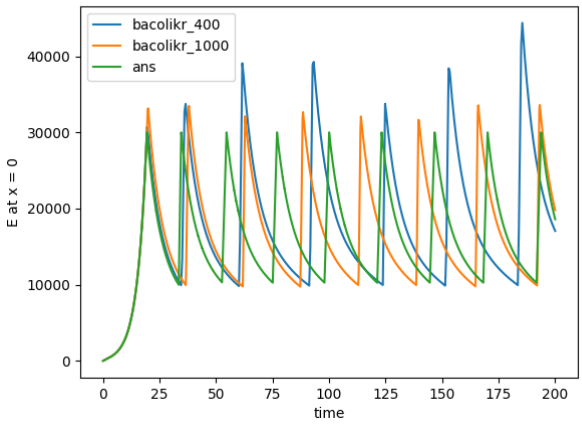

Figure 9: State dependent discontinuity naive treatment with a tolerance of 10^{-6} and 400vs1000 time steps

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 0.035 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
```

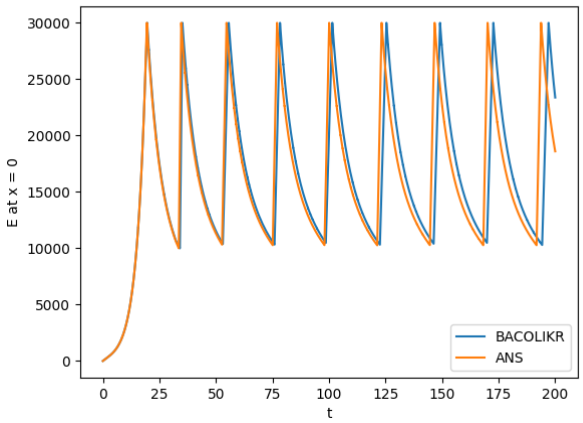

Figure 10: State dependent discontinuity using event detection with a tolerance of 10^{-6}

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

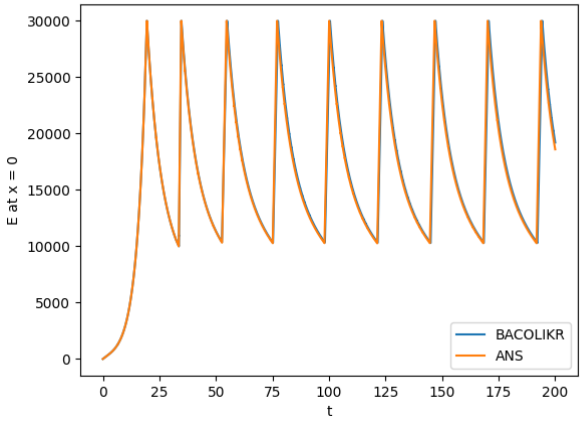

Figure 11: State dependent discontinuity using event detection with a tolerance of 10^{-9}

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 10^{-6} tol.	1, 937, 730
BACOLIKR 10^{-9} tol.	7, 915, 085

Table 2 starts showing the pattern for the PDE state discontinuity. The problem is a trade-off between efficiency and accuracy even with root-finding. Improving the accuracy comes at the price of efficiency. We note that without event detection, the solutions even with 1000 time steps are still inaccurate. The solutions never oscillate correctly between 10, 000 and 30, 000. However, even with event detection, the tolerance needs to be high enough to get accurate solutions.

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

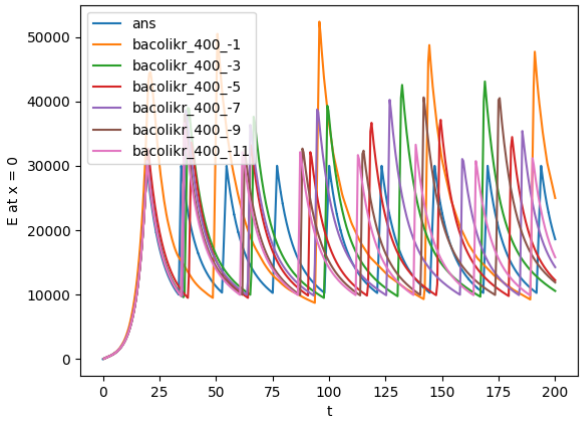

Figure 12: State dependent discontinuity naive treatment tolerance study with BACOLIKR with 400 time steps

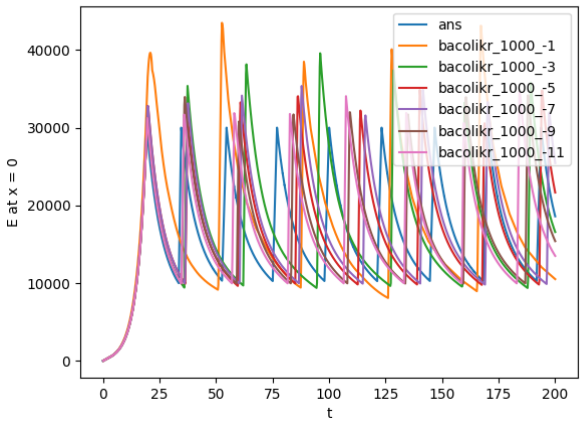

Figure 13: State dependent discontinuity naive treatment tolerance study with BACOLIKR with 1000 time steps

BACOLIKR without event Figures 12 and 13 show that the solutions are not aligned and the roots are not detected exactly at 30000 in the case without event detection. We note that the problem is mostly in the exponentially increasing direction due to the instability. Having a higher time step slightly improves the situation but we will see that with event detection improves the accuracy to a greater degree.

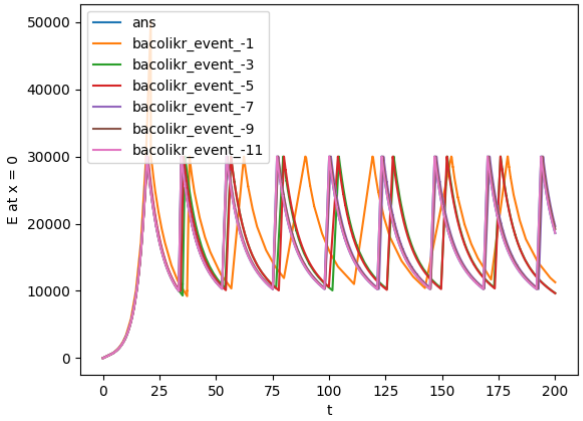

Figure 14: State dependent discontinuity using event detection tolerance study

BACOLIKR with event detection With event detection, the solution oscillate correctly for all tolerances except at 10^{-1}. The other solutions are only slightly misaligned but this can be explainined with the tolerance at the roots. Depending on the tolerance, the root is detected at a different time step and thus different tolerances will lead to initiating the next steps of the integral with a different β value and thus the solutions are not aligned.

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

Efficiency of the solvers Table 3 shows that we are forced to sacrifice efficiency for accuracy in this problem. We note that all naive solutions highly depend on the time stepping and thus should not be trusted for accurate solutions. For the solutions using event detection, the accuracy problem is at the roots where the models are to be changed. Depending on the tolerance, the root is detected at a different location as the root-finding algorithm returns the location of the root based on the tolerance. Thus different tolerances will lead to changing the models at slightly different positions. During the exponential increase phase, we can miss the location of the root by far enough that the solutions are not aligned.

1.4 Discussions