Co-Simulation-Test Case: Predator-Prey (Lotka-Volterra) System

Andreas Nicolai*

August 2, 2019

Abstract

The test case discussed in this article is a coupled initial value problem, very well suited to investigate the effect of error propagation and the accuracy differences between non-iterating and iterating master algorithms. Further discussed are the error test methods implemented in MASTERSIM. The example case also illustrates, that contrary to expectations the use of the iterating Gauss-Seidel algorithm with adaptive time stepping based on error control generates a solution of slightly less accuracy while using many more model evaluations.

*Institut für Bauklimatik, Technische Universität Dresden, andreas.nicolai@tu-dresden.de

Contents

1 Test Description **3**
1.1 Mathematical Equations . . . 3
1.2 Requested solution . . . 3
1.3 Expected results . . . 3
1.4 Reference Modelica Model . . . 4

2 Co-Simulation **4**
2.1 Decomposition . . . 4
2.2 Evaluation Order . . . 4
2.3 Initial Conditions and Integration Interval . . . 4
2.4 Implementing FMUs . . . 4

3 Evaluation of Master-Algorithms **5**
3.1 Gauss-Seidel without iteration and with constant step sizes . . . 5
3.2 Gauss-Seidel with larger, constant step sizes and with/without iteration . . . 5

4 Error control **6**
4.1 Comparison of time step sizes . . . 7
4.2 Comparison of error norms . . . 7
4.3 Evaluation of the true/global error . . . 8

1 Test Description

The test case is based on the Lotka-Volterra example from Michael Tiller.
Tiller, M; *Modelica by Example*, http://book.xogeny.com

1.1 Mathematical Equations

$$\dot{x} = x\,(a - by) \tag{1}$$
$$\dot{y} = y\,(dx - c) \tag{2}$$

Variable x stands for the amount of prey species, y is the predator species. a is the reproduction rate, b the reduction factor due to predation. c is the rate of starvation and d the rate factor representing grows of predator species due to consumption of species x.

1.2 Requested solution

Solution for variables x and y is to be obtained for the time interval $t \in [0, T]$, $T = 10$.

Constants shall be selected as follows:

- $a = 0.1$
- $b = 0.02$
- $c = 0.4$
- $d = 0.02$

1.3 Expected results

The problem is nonlinear and has a coupled solution, that can be, for example, be generated with a traditional coupled system solver (e.g. Modelica). The solution for the given constants is shown in Figure 1.

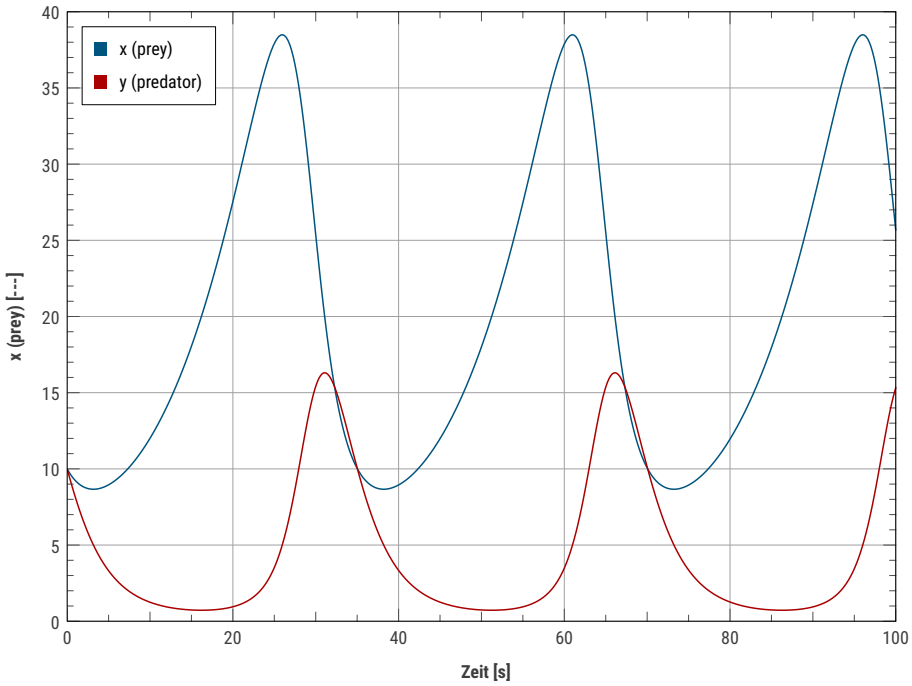

Figure 1: Exact solution

The solution is completely continuous without any state events.

1.4 Reference Modelica Model

The equations can be solved directly in a coupled manner with Modelica. The source code for a fully coupled solution in Modelica is given below in Listing 1.

```
model LotkaVolterra
  parameter Real A=0.1 "Reproduction rate of prey";
  parameter Real B=0.02 "Mortality rate of predator per prey";
  parameter Real C=0.4 "Mortality rate of predator";
  parameter Real D=0.02 "Reproduction rate of predator per prey";
  parameter Real x0=10 "Initial prey population";
  parameter Real y0=10 "Initial predator population";
  Real x(start=x0) "Prey population";
  Real y(start=y0) "Predator population";
initial equation
  x = x0;
  y = y0;
equation
  der(x) = x*(A-B*y);
  der(y) = y*(D*x-C);
end LotkaVolterra;
```

Listing 1: Modelica Code Listing, adapted from M. Tiller *Modelica by Example*

2 Co-Simulation

2.1 Decomposition

For the purpose of testing Co-Simulation masters the test case is split into two parts, one for the prey species and one for the predator species. Both are coupled in a cycle.

FMU	Cycle	Input	Equations	Output
Prey	0	y	$\dot{x} = x\,(a - by)$	x
Predator	0	x	$\dot{y} = y\,(dx - c)$	y

2.2 Evaluation Order

Prey FMU shall be evaluated first.

2.3 Initial Conditions and Integration Interval

The solution shall be obtained for the time interval $T = [0, 100]$ with $x\,(0) = y\,(0) = 10$.

2.4 Implementing FMUs

FMUs are implemented in direct C/C++ code. The time integration has an exact solution, hereby using constant values for input variables throughout the communication interval. For example, the time integration formula for the prey species is derived as follows:

$$\begin{aligned}
\frac{dx}{dt} &= x\,(A - By) = Gx \\
\frac{dx}{x} &= Gdt \\
\int_{x(t_0)}^{x(t_1)} \frac{1}{x} dx &= \int_{t_0}^{t_1} Gdt \\
\ln x\,(t_1) - \ln x\,(t_0) &= G \cdot (t_1 - t_0) \\
\frac{x\,(t_1)}{x\,(t_0)} &= e^{G\cdot(t_1 - t_0)} \\
x\,(t_1) &= x\,(t_0)\, e^{G\cdot(t_1 - t_0)}
\end{aligned} \tag{3}$$

Equation (3) gives the exact solution, if y were a constant. However, since $y(t)$ is a function of time and hence also $G(t) = A - By(t)$ is changing within the integration interval, the solution $x(t_1)$ holds an integration error, they may become, due to the exponential nature of the equation, very large. It is also anticipated, that this may result in instability in the outer solution method, especially when using a non-iterative master algorithm.

The exact integral equation can be implemented quite naturally in C/C++ code. The code example in Listing 2 below shows the relevant portion of the `doStep()` function of the FMU:

```
// state of FMU before integration:
//    m_currentTimePoint = tIntervalStart;
//    m_y[0] = x(tIntervalStart)
//    m_realInput[FMI_INPUT_Y] = y(tIntervalStart...tCommunicationIntervalEnd) = const

// compute time step size
double dt = tCommunicationIntervalEnd - m_currentTimePoint;
double y = m_realInput[FMI_INPUT_Y];
double x = m_y[0];
double x_end = x*std::exp( (A - B*y)*dt);
m_y[0] = x_end;

// update state of FMU to tCommunicationIntervalEnd
m_tInput = tCommunicationIntervalEnd;
m_realOutput[FMI_OUTPUT_X] = x_end;
m_currentTimePoint = tCommunicationIntervalEnd;
```

Listing 2: FMU Listing, doStep-function

3 Evaluation of Master-Algorithms

3.1 Gauss-Seidel without iteration and with constant step sizes

Running the integration with constant step size of $h = 0.1s$ and without iteration already gives pretty good results (see Figure 2).

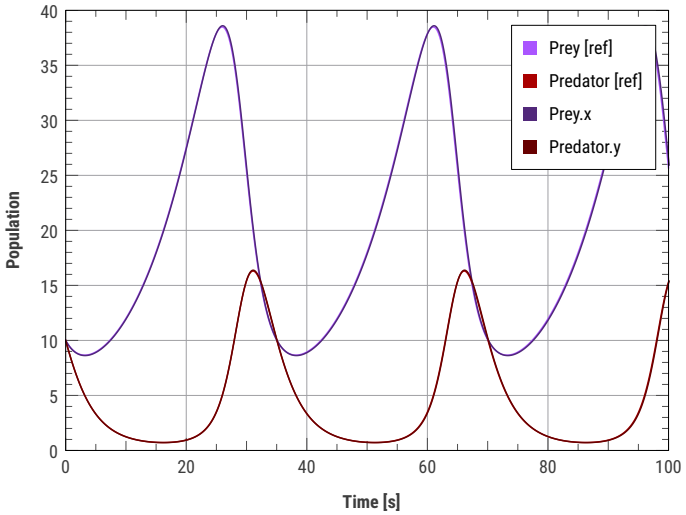

Figure 2: No iteration, constant steps (0.1 s)

3.2 Gauss-Seidel with larger, constant step sizes and with/without iteration

Using larger step sizes (for example $h = 0.5s$) we expect larger errors. In the variant without iteration, the information about the opposing species concentration arrives delayed in the FMU, because the integration in each interval is done using the values at the *begin of the interval*. Hence, the curves have a slight delay behind the correct solution, see Figure 3 (*no iteration*-curves).

Alternatively, an iterative Gauss-Seidel method could be used. When the solution has converged (which corresponds somewhat to an implicit integration scheme), the species concentrations are computed using

the opposite species concentration at the *end* of each integration interval. This leads to a much dampened solution where the curves react faster compared to the reference solution (Figure 3, *GS* curves).

Consider, for example, the interval 25..25,5 s. The concentration of the predator species is strongly increasing in this interval, from $y\,(25) = 3.52$ to $y\,(25.5) = 4.22$. If the integration of the prey species is done over this interval using the input value $y\,(25)$ the concentration $x\,(25.5)$ will be larger than that obtained when integrating with a larger predator concentration $y\,(25.5)$. As a result, the prey concentration will not reach the same concentration as the correct solution, and subsequently, also the predator concentration will be affected leading to an overall dampened solution. With this time step size, effectively a different problem appears to be solved.

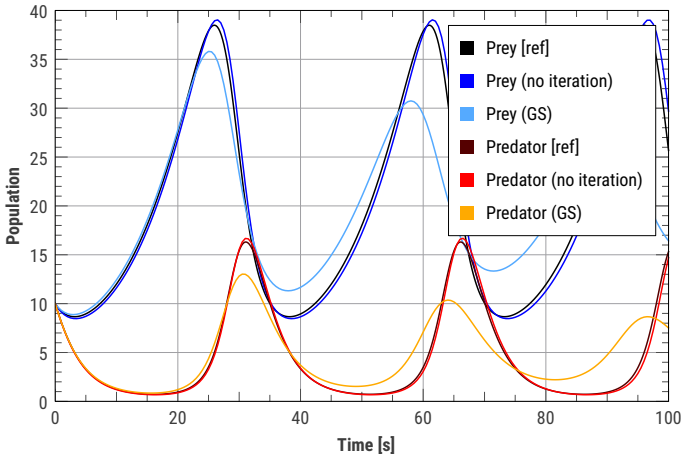

Figure 3: Comparison between non-iterating and iterating Gauss-Seidel, constant steps (0.5 s); iteration limit is fairly strict with reltol = 10^{-7}

Deviation between the iterative solution and the correct solution can be observed already in the first step, for example, when using a fairly large step of 2 seconds (Figure 4).

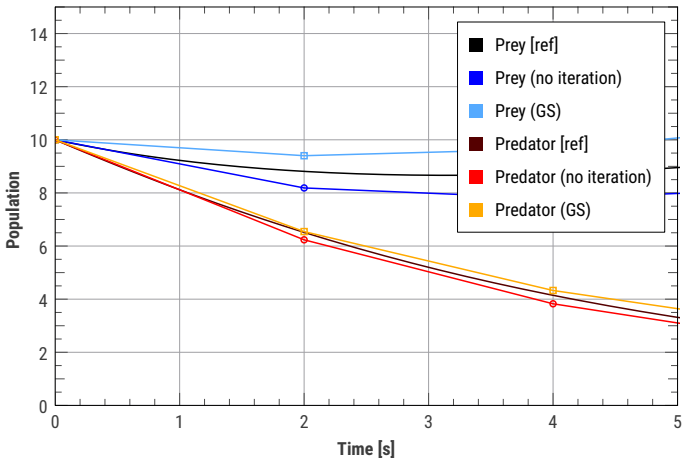

Figure 4: Comparison of iterating and non-iterating GAUSS-SEIDEL, for $h = 2s$

From this example it can be concluded, that using an iterative method does not necessarily improve the accuracy of the result, though, it may increase stability.

4 Error control

When running the test case with error control enabled (Richardson/step-doubling error test), we expect the master algorithm to choose the time steps such, that regardless of the selected master algorithm the results are accurate within the requested tolerance band. This is indeed the case, as can be seen in the result comparison in Figure 5.

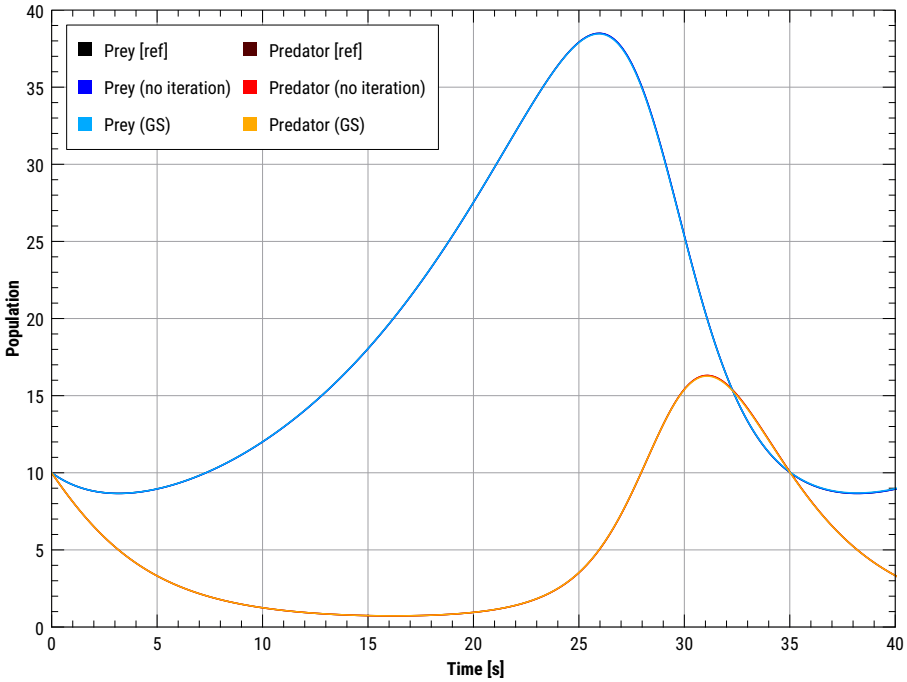

Figure 5: Comparison of iterating and non-iterating GAUSS-SEIDEL when using dynamic step adjustment based on step-doubling/Richardson error control (reltol = abstol = 10^{-6})

4.1 Comparison of time step sizes

The Gauss-Seidel variant takes almost always 3 iterations per step to converge. The comparison of the time step sizes (Fig. 6) shows that the time step sizes are actually very similar, though the variant without iteration permits generally slightly larger time steps.

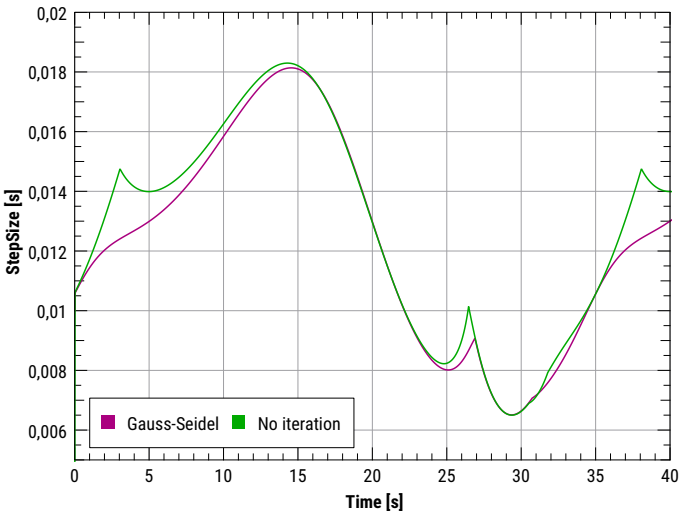

Figure 6: Comparison of accepted time step sizes for iterating and non-iterating variant

4.2 Comparison of error norms

The step-doubling technique is used in MASTERSIM to estimate the integration error. Hereby, two methods are implemented, one using the differences of results (4) and one using the difference of the slopes (5), which

are approximated from the values. Note, constant extrapolation of input signals is used.

$$\epsilon_{Richardson} = \left| y\left(t + 2 \cdot \frac{h}{2}\right) - y\left(t + h\right) \right| \tag{4}$$

$$\begin{aligned} \epsilon_{slope} &= h\left(\dot{y}\left(t+h\right)_h - \dot{y}\left(t+h\right)_{h/2}\right) \\ &= h\left(\frac{y\left(t+h\right) - y\left(t\right)}{h} - \frac{y\left(t+h\right) - y\left(t+\frac{h}{2}\right)}{h/2}\right) \end{aligned} \tag{5}$$

The value $y\left(t + 2 \cdot \frac{h}{2}\right)$ is obtained after executing two half-steps and $y\left(t+h\right)$ is the value obtained with the one full step. The derivatives $\dot{y}\left(t+h\right)_h$ and $\dot{y}\left(t+h\right)_{h/2}$ at $t+h$ are approximated by backward finite differences using the full step and the second half-step, respectively. Both estimates yield approximately the same norms (Fig. 7). For the calculation of the step size always the larger of the two was used, thus resulting in smaller time steps on average.

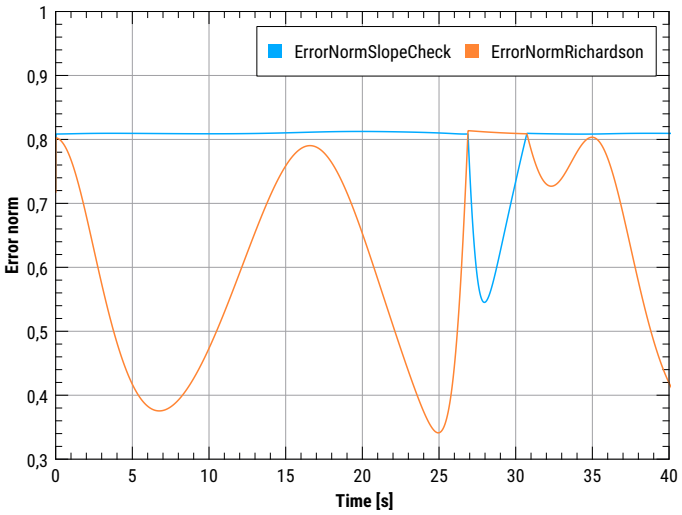

Figure 7: Comparison of error norms that were obtained when accepting a step, for the iterative Gauss-Seidel variant

4.3 Evaluation of the true/global error

By computing the normalized difference between the co-simulation variants and the reference solution the global error can be obtained (Fig. 8). The absolute differences Δ_i and global errors ϵ_i are computed for each data point i with

$$\Delta_i = y_{FMI,i} - y_{ref,i}$$

$$\epsilon_i = \left| \frac{y_{FMI,i} - y_{ref,i}}{y_{ref,i}} \right|$$

where y_{FMI} is one of the FMI solutions (iterative or non-iterative).

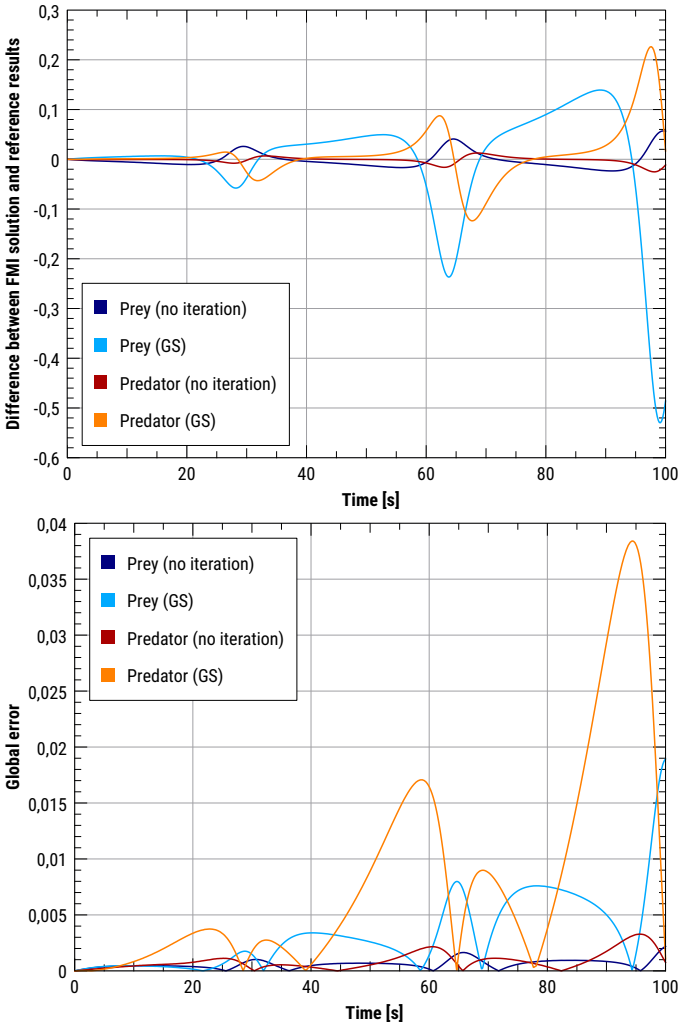

Figure 8: Absolute differences between computed prey and predator species population and reference solution (top) and global error for the iterative and non-iterative variants and time step adjustment based on error control with reltol = abstol = 10^{-6}

Clearly, despite the time step adjustment based on *local error estimates* the dampening of the model equations cannot be fully prevented. And it appears that, at least for this test case, the iterative variant tends to generate larger global errors. Furthermore, it is interesting to note that controling the local error to be below 10^{-6} will still yield a global error of 3 orders of magnitude larger (10^{-3}).

This can be explained by the error propagation in this model. As can be seen from the solver statistics, more than 8000 steps (Master-Algorithm calls) were taken in both variants:

```
Solver statistics
-------------------------------------------------------------------------------
Wall clock time                          =  220.991 ms
-------------------------------------------------------------------------------
Output writing                           =  195.781 ms
Master-Algorithm                         =   16.093 ms        8653
Convergence failures                     =                       0
Convergence iteration limit exceeded     =                       0
Error test time and failure count        =    5.276 ms           7
-------------------------------------------------------------------------------
Prey                               doStep =    5.132 ms       25980
                                getState =    0.656 ms        8653
                                setState =    0.673 ms        8667
Predator                           doStep =    4.351 ms       25980
                                getState =    0.607 ms        8653
                                setState =    0.494 ms        8667
-------------------------------------------------------------------------------
```

Listing 3: Non-iterating GAUSS-SEIDEL algorithm

```
Solver statistics
-------------------------------------------------------------------------------
Wall clock time                          =  257.438 ms
-------------------------------------------------------------------------------
Output writing                           =  204.746 ms
Master-Algorithm                         =   39.133 ms        8872
Convergence failures                     =                       0
Convergence iteration limit exceeded     =                       0
Error test time and failure count        =   11.838 ms           0
-------------------------------------------------------------------------------
Prey                               doStep =   10.061 ms       57775
                                getState =    1.611 ms       26616
                                setState =    2.191 ms       40031
Predator                           doStep =    9.331 ms       57775
                                getState =    1.373 ms       26616
                                setState =    1.947 ms       40031
-------------------------------------------------------------------------------
```

Listing 4: Iteration GAUSS-SEIDEL algorithm; on average 3 iterations per step

Assuming a relative error of about the same magnitude in each step and no error cancellation, this would indeed yield a large global error. This effect is, however, strongly depending on the problem.

Also, it can be noted that even the non-iterating variant (see statistics in Listing 3) requires approximately 3 `doStep()` evaluations per accepted step. For the iterating variant (see statistics in Listing 4) these are more than 6 evaluations per accepted step. Thus, for this example, the application of iterating Gauss-Seidel is not recommended. Further tests may show that this conclusion can be generalized: when employing time step adjustment based on error estimates, non-iterating Gauss-Seidel is to be preferred.