Co-Simulations-Masteralgorithmen - Analyse und Details der Implementierung am Beispiel des Masterprogramms MASTERSIM

Andreas Nicolai*

16. August 2018

Zusammenfassung

In der Version 2.0 des Simulationskopplungsstandards FMI (Functional Mockup Interface) wird die Möglichkeit zur Speicherung und Wiederherstellung einer Simulationseinheit/FMU (Functional Mockup Unit) definiert. Dieses ist eine elementare Voraussetzung für iterierende Co-Simulations-Masteralgorithmen, wie z.B. Gauss-Seidel oder Newton-Iteration. Für die gekoppelte Simulation von solchen Simulationseinheiten ist ein Co-Simulations-Master erforderlich.

Das Simulationsmasterprogramm MASTERSIM ist ein solcher Co-Simulations-Master und enthält zahlreiche Algorithmen und eine effiziente Verwaltung von Simulationseinheiten unter Verwendung dieser neuen Schnittstellenfunktionen. Dieser Artikel dokumentiert grundlegende Co-Simulations-Algorithmen und beteiligte Parameter und illustriert deren Einfluss anhand eines Testbeispiels.

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

Inhaltsverzeichnis

1 Grundlagen

1.1 Der Functional Mockup Interface Standard

Der Functional Mockup Interface (FMI) Standard ist ein etablierter Industriestandard für die Simulationskopplung. Im Standard werden technische Regeln definiert, wie Simulationsprogramme zur Laufzeit gekoppelt ausgeführt werden und dabei Daten untereinander austauschen. So kann ein komplexes System als Zusammenstellung einzelner Simulationskomponenten abgebildet werden.

Bei den Teilsystemen handelt es sich üblicherweise um dynamische Systeme, d.h. Systeme gekoppelter gewöhnlicher Differentialgleichungen, welche mittels numerischer Integration gelöst werden.

1.2 ModelExchange und Co-Simulation

Bei der gekoppelten Lösung gibt es grundsätzlich zwei Möglichkeiten, unterschiedliche Teilmodelle, bzw. Simulationseinheiten numerisch zu integrieren:

ModelExchange Die Erhaltungsgrößen der gekoppelten Modelle werden in einem zentralen Zeitintegrator zusammen gelöst, wobei Fehlerschranken und Zeitschrittanpassung von der Charakteristik der einzelnen Gleichungen abhängen. Jede FMU teilt dabei dem Simulationsmaster die Anzahl der zu integrierenden Differentialgleichungen mit und stellt die Funktionalität zur Berechnung der Zeitableitung der Erhaltungsgrößen bereit. Dieses ermöglicht eine globale Sicht, d.h. eine vollständig gekoppelte Lösung der Gesamtheit der Gleichungen aller FMUs. Dadurch ist Stabilität und Genauigkeit der Lösung gewährleistet.

CoSimulation Die Teilmodelle werden unabhängig voneinander in der Zeit in kleinen Intervallen integriert. Nach jedem Interval können Ergebnisgrößen zwischen den Teilkomponenten ausgetauscht werden. Je nach Kopplungsalgorithmus erhalten die Teilmodelle/FMUs dann Gelegenheit, das gleiche Zeitinterval erneut unter Berücksichtigung der neuen Eingangsgrößen zu lösen. Die Genauigkeit der Lösung, und je nach Problem auch die Stabilität, hängt naturgemäß von der Länge der Kopplungsintervalle ab.

Unabhängig davon, welche Kopplungsmethode gewählt wird, ist ein Simulationsmaster notwendig. Diese Programm verwaltet die einzelnen Simulationskomponenten, integriert im ModelExchange-Modus die Differentialgleichungen und regelt den Datenaustausch zwischen den Simulationseinheiten.

1.3 Simulationseinheiten/Functional Mockup Units (FMU)

Der FMI Standard regelt die Konfiguration und Verteilung von FMUs. Dabei werden alle beteiligten Dateien in eine definierte Verzeichnisstruktur kopiert, welche dann in ein zip-Archiv komprimiert wird. Die Endung der resultierende Datei wird üblicherweise auf `.fmu` geändert.

Innerhalb der Verzeichnisstruktur gibt es im Basisverzeichnis die Datei `ModelDescription.xml`. Diese enthält alle Informationen über die Funktionalität der FMU und der Schnittstellen, d.h. Eingangs- und Ergebnisgrößen.

1.4 Grundlegendes Co-Simulationsprozedere

Der hier vorgestellte Simulationsmaster behandelt das Co-Simulationsverfahren. Dafür initialisiert er die Simulationseinheiten, wobei eine Simulationseinheit mehrfach verwendet (instanziiert) werden kann. Daher wird von Instanzen der Simulationseinheiten gesprochen, bzw. von Simulations-Slaves. Der Master versetzt diese Slaves anfänglich in einen definierten Startzustand, transferiert Startwerte zwischen den FMU-Instanzen bzw. Slaves und beginnt die Integration. Dabei werden alle Slaves nacheinander aufgefordert, die Integration über einen bestimmten Zeitraum, das Kommunikationsintervall, durchzuführen. Die Integration der Slaves erfolgt komplett unabhängig voneinander. Je nach Kopplungsalgorithmus (siehe §5) werden nun Ergebnisgrößen zwischen den Slaves ausgetauscht, die Integration über das Kommunikationsinterval wiederholt oder das nächste Kommunikationsinterval begonnen. Eine Anpassung der Länge des Kommunikationsintervals ist ebenso möglich. Es gibt hierbei viele Kombinationsmöglichkeiten der beteiligten Algorithmen, wobei die Wahl vom mathematischen Problem bzw. der Konfiguration der beteiligten FMUs abhängt.

2 Zeitintegration

Die Zeitintegration erfolgt in einzelnen Intervallen, den sogenannten Kommunikationsintervallen. Je nach Kopplungsalgorithmus werden alle Slaves aufgefordert, unter Umständen mehrfach, das Kommunikationsintervall zu berechnen. Nachdem eine Lösung eines Kommunikationsintervals gefunden wurde, kann eine Genauigkeitsprüfung erfolgen und die Länge des Kommunikationsschritts angepasst werden. Bei Überschreiten einer zulässigen Fehlerschranke kann ein Kommunikationsinterval auch mit kürzerer Länge wiederholt werden.

Aufgrund der verschiedenen Algorithmen zur

- Bestimmung eine gekoppelten Lösung eines Kommunikationsintervals,
- Abschätzung des Integrationsfehlers,
- Anpassung der Intervallänge und
- eventueller Wiederholung eines Intervals

ergeben sich viele mögliche Kombinationsmöglichkeiten, welche Einfluss auf die Berechnungsgenauigkeit haben und maßgeblich die Gesamtrechenzeit beeinflussen. Letztere wird im Wesentlichen durch die Anzahl der aufgerufenen Intervalintegrationsaufrufe der einzelnen FMUs bestimmt und kann im Nachgang der Berechnung mittels der Solverstatistiken ausgewertet werden (siehe Abschnitt 3.1).

3 Simulationsperformance

Die Geschwindigkeit der gekoppelten Simulation hängt zum Großteil von den Berechnungszeiten der einzelnen FMUs ab. Die vom Master selbst benötigte Zeit für den Datenaustausch und das Schreiben der Ergebnis- und Protokolldateien ist im Vergleich dazu vernachlässigbar. Je nach Implementierung der FMU sind verschiedene FMI-Schnittstellenfunktionen für die Simulationszeit verantwortlich. MASTERSIM erfasst die Ausführungszeiten (eingebautes Profiling) und gibt diese nach Simulationsende aus.

3.1 Statistikausgaben

MASTERSIM zeigt Statistikausgaben zur Performanceanalyse nach Simulationsende auf dem Bildschirm dar und schreibt diese in die Logdatei `screenlog.txt`. Beispiel für eine solche Statistik (einer sehr schnellen Simulation):

```
1  Solver statistics
2  ------------------------------------------------------------------------------
3  Wall clock time                          =   16.878 ms
4  ------------------------------------------------------------------------------
5  Output writing                           =    4.825 ms
6  Master-Algorithm                         =    5.876 ms          10
7  Convergence failures                     =                       0
8  Error test time and failure count        =    0.882 ms           0
9  ------------------------------------------------------------------------------
10 Part1                                 doStep =    0.999 ms          66
11                                    getState =    0.111 ms          30
12                                    setState =    0.157 ms          46
13 Part2                                 doStep =    0.961 ms          66
14                                    getState =    0.095 ms          30
15                                    setState =    0.144 ms          46
16 ------------------------------------------------------------------------------
```

Je nach Dauer der Simulation werden Zeiteinheiten ms (Millisekunden), s, min, h, d, a (Jahr) verwendet. Die folgenden Zeiten werden erfasst:

- **Wall clock time** - insgesamt benötigte Simulationszeit

welche sich aus den folgenden Zeiten zusammensetzt:

- **Output writing** - Zeit für das Schreiben der Ergebnisdateien im MASTERSIM
- **Master-Algorithm** - Die gesamte Zeit, welche für die Berechnung der gekoppelten Lösung benötigt wird

- **Error test time and failure count** - Die Zeit, welche für die Fehlerschätzung benötigt wird.

Die Zeit für die Auswertung des Master-Algorithmus setzt sich zudem aus der Auswertungszeit der einzelnen FMUs und dabei der jeweiligen Schnittstellenfunktionen zusammen. So wird für jede FMU-Instanz die Zeit für das Setzen und Holen der Systemzustände (`setState` und `getState`) erfasst und die Zeit für das Integrieren über das Kommunikationsinterval (`doStep`). Die Auswertungszeit aller dieser Aufrufe aller Instanzen summiert sich auf die Master-Algorithmus-Zeit, allerdings wird ein gewisser Arbeitsaufwand im MASTERSIM selbst nicht explizit ausgewiesen (Overhead). Bei sehr schnellen Simulationen wie im Beispiel oben ist die

Die letzte Spalte zeigt die Anzahl der Aufrufe der einzelnen Funktionen. Im Beispiel ist ersichtlich, dass insgesamt 10 Aufrufe des Master-Algorithmus erfolgt sind (d.h. 10 Kommunikationsintervalle bearbeitet wurden). Weiterhin zeigt die Spalte, dass die FMUs häufiger zurückgesetzt wurden (im Durchschnitt 1 mal Holen des Zustands und 2 mal Setzen des Zustands). Dies liegt am verwendeten Fehlerschätzeralgorithmus.

Mit diesen Statistikausgaben lassen sich die maßgeblich für die Simulationszeit verantwortlichen FMUs und Schnittstellenfunktionen bestimmen und damit gezielt optimieren.

4 Testbeispiel

Ein Testbeispiel[1] soll zur Erläuterung der verschiedenen Algorithmen verwendet werden. Die folgenden mathematischen Gleichungen sind zu lösen:

$$x_1 = \begin{cases} 0 & t < 1 \quad \text{or} \quad 2 \leq t < 5 \\ 1 & else \end{cases} \tag{1}$$

$$x_2 = \begin{cases} 0 & t < 3 \quad \text{or} \quad 4 \leq t < 6 \\ 1 & else \end{cases} \tag{2}$$

$$x_3 = \begin{cases} 3 & x_1 = 1 \quad \text{and} \quad x_2 < 0.01 \quad \text{and} \quad x_4 < 2.5 \\ -3 & x_1 < 0.001 \quad \text{and} \quad x_2 > 0 \quad \text{and} \quad x_4 > -2.5 \\ 0 & else \end{cases} \tag{3}$$

$$\dot{x}_4 = 2x_3 \tag{4}$$

Die Lösung der Variablen x_1, x_2, x_3, x_4 soll für das Zeitinterval $t \in [0, T]$, $T = 10$ bestimmt werden.

Dabei ist zu beachten, dass die Bedingungen für die Variable x_3 und Vergleiche mit digitalen Signalen nicht mit 0 oder 1 formuliert sind, sondern kleine Abweichungen in den Variablen x_1 und x_2 erlauben. Das ist für Newton-basierte Kopplungsalgorithmen notwendig, welche Jacobi-Matrizen unter Verwendung von Finiten-Quotionen erstellen. Dabei werden jeweils kleine Änderungen an Variablen durchgeführt, um die Abhängigkeiten zwischen Gleichungen und Variablen zu ermitteln. Mit den etwas toleranteren Tests in diesem Beispielproblem können genauere Jacobi-Matrizen erstellt werden.

4.1 Referenzlösung

Das Problem hat eine exakte Lösung, siehe Abbildung 1.

[1] Der Testfall wurde von C. Clauß vom Fraunhofer IIS EAS in Dresden entworfen.

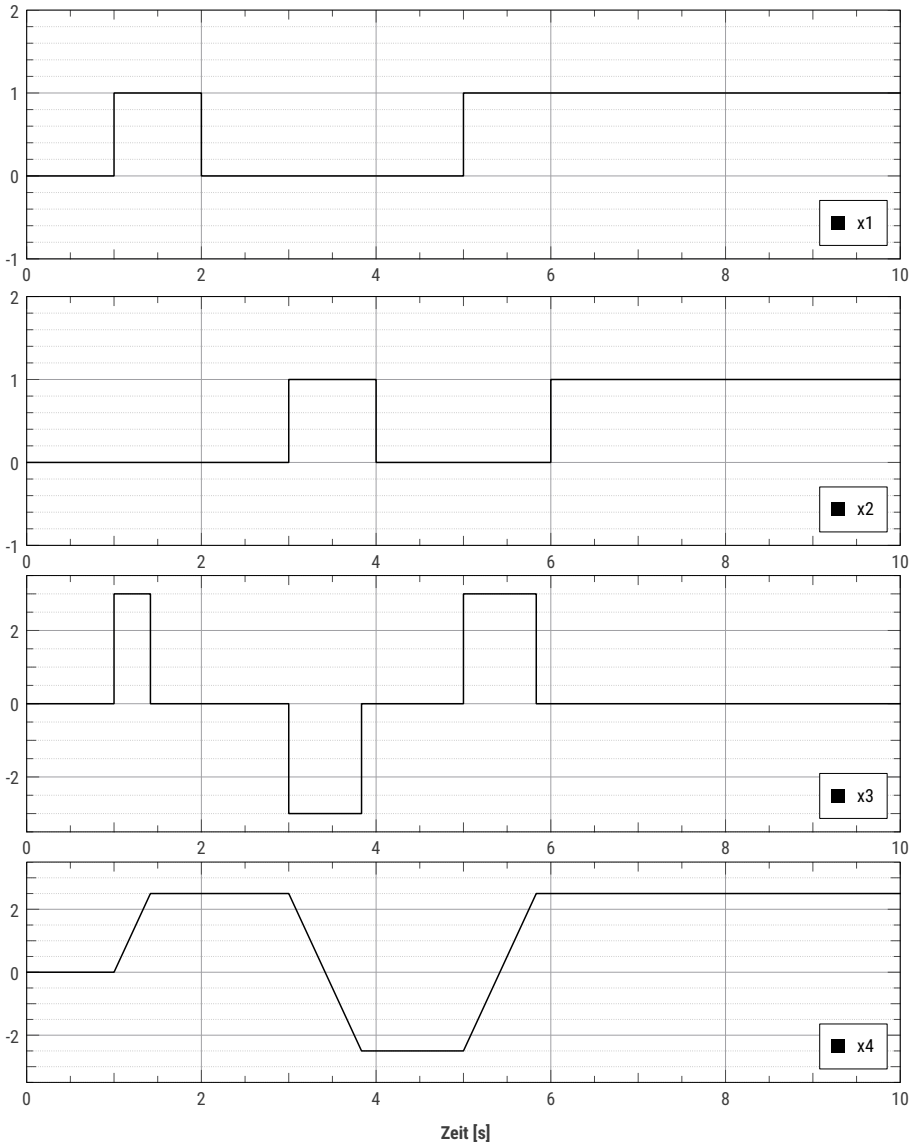

Abbildung 1: Referenzergebnisse

Die Zustandsereignisse treten auf bei: $t = 1 + 2.5/6 = 1.4166667$, $t = 3 + 5/6 = 3.833333$ und $t = 5 + 5/6 = 5.833333$.

t	x1	x2	x3	x4
0	0	0	0	0
0.99999999	0	0	0	0
1	1	0	3	0
1.416666657	1	0	3	2.5
1.41667	1	0	0	2.5
1.99999999	1	0	0	2.5
2	0	0	0	2.5
2.99999999	0	0	0	2.5
3	0	1	-3	2.5
3.83333333	0	1	-3	-2.5
3.83334	0	1	0	-2.5
3.99999999	0	1	0	-2.5
4	0	0	0	-2.5
4.99999999	0	0	0	-2.5
5	1	0	3	-2.5
5.83333333	1	0	3	2.5
5.83334	1	0	0	2.5
5.99999999	1	0	0	2.5
6	1	1	0	2.5
10	1	1	0	2.5

Abbildung 2: Zahlenwerte der exakten Lösung

4.2 Zerlegung des Problems in Teilmodelle

Um die Co-Simulationsalgorithmen zu testen, wird das gekoppelte Problem wie folgt zerlegt:

Teil	Zyklus	Eingabe	Gleichungen	Ausgabe
1	1	–	Gleichungen (1) und (2)	x_1, x_2
2	2	x_1, x_2, x_4	Gleichung (3)	x_3
3	2	x_3	Gleichung(4)	x_4

Nur Teil 2 und 3 sind in einem Zyklus gekoppelt. Als Auswertungsreihenfolge wird festgelegt, dass Zyklus 1 zuerst ausgewertet wird. In allen Algorithmen, in denen die Auswertungsreihenfolge eine Rolle spielt, soll Teil 2 vor Teil 3 ausgeführt werden.

4.3 Implementierung der FMUs

Am einfachsten ist die Erstellung der entsprechenden Modelle in Modelica und der anschließende FMU-Export aus dem Modelica-Simulationstools. Dabei muss allerdings das Simulationstool die Spezifikation des FMI-Standards korrekt implementieren und die numerische Lösung des Modelica-Modells muss korrekt sein. Dies ist nicht immer leicht zu prüfen, weswegen diese trivialen FMUs manuell in C/C++ implementiert wurde. Die relevanten Berechnungsalgorithmen sind nachfolgend gezeigt. Die Funktion `integrateTo()` wird dabei von der FMI-Schnittstellenfunktion `fmi2DoStep()` aufgerufen.

```
1  void Math003Part1::integrateTo(double tCommunicationIntervalEnd) {
2      m_tInput = tCommunicationIntervalEnd;
3
4      if (m_tInput < 1 || (m_tInput >= 2 && m_tInput < 5))
5          m_realOutput[FMI_OUTPUT_X1] = 0;
6      else
7          m_realOutput[FMI_OUTPUT_X1] = 1;
8
9      if (m_tInput < 3 || (m_tInput >= 4 && m_tInput < 6))
10         m_realOutput[FMI_OUTPUT_X2] = 0;
11     else
12         m_realOutput[FMI_OUTPUT_X2] = 1;
13 }
```

Implementierung von *Part 1*

```
1  void Math003Part2::integrateTo(double tCommunicationIntervalEnd) {
2      m_tInput = tCommunicationIntervalEnd;
3
4      // Eingangsgrößen holen
5      double x1 = m_realInput[FMI_INPUT_X1];
6      double x2 = m_realInput[FMI_INPUT_X2];
7      double x4 = m_realInput[FMI_INPUT_X4];
8
9      if (x1 == 1 && x2 < 0.01 && x4 < 2.5)
10         m_realOutput[FMI_OUTPUT_X3] = 3;
11     else if (x1 < 0.001 && x2 > 0 && x4 > -2.5)
12         m_realOutput[FMI_OUTPUT_X3] = -3;
13     else
14         m_realOutput[FMI_OUTPUT_X3] = 0;
15 }
```

Implementierung von *Part 2*

```
1  void Math003Part3::integrateTo(double tCommunicationIntervalEnd) {
2      // Zeitschrittlänge berechnen, Eingangsgröße holen, Anstieg berechnen
3      double dt = tCommunicationIntervalEnd - m_currentTimePoint;
4      double x3 = m_realInput[FMI_INPUT_X3];
5      double deltaX4 = dt*x3*2;
6
7      // Zeitintegration
8      m_yInput[0] += deltaX4;
9      m_realOutput[FMI_OUTPUT_X4] = m_yInput[0];
10     m_currentTimePoint = tCommunicationIntervalEnd;
11 }
```

Implementierung von *Part 3*

Die Zeitintegration ist durch den konstanten Anstieg exakt möglich. Vor dem Aufruf von integrateTo() hat die FMU den Zustand $t_{currentTimePoint}$ und $x_4\,(t_{currentTimePoint})$. Nach dem Aufruf ist $t_{currentTimePoint} = t_{communicationIntervalEnd}$ und $x_4\,(t_{communicationIntervalEnd})$ als Zustand der FMU gespeichert.

5 Kopplungsalgorithmen

5.1 Nicht-iterierender GAUSS-JACOBI-Algorithmus mit fester Schrittweite

Der einfachste Kopplungsalgorithmus ist der GAUSS-JACOBI-Algorithmus. Dieser Algorithmus funktioniert mit FMUs der Version 1 und 2 und stellt keine besonderen Anforderungen an die FMU-Implementierung.

5.1.1 Parameter

- Länge der Kommunikationsschritte

5.1.2 Algorithmus

Im nachfolgenden Algorithmus werden nur die Gleitkommazahlen gezeigt. Die Behandlung von boolischen Werten, Integern und Zeichenketten erfolgt analog.

Algorithmus 1 : Gauss-Jacobi Algorithmus

Input : t, h, $\mathbf{y_t}$ Vektor mit Gleitkommawerten zum Zeitpunkt t
Output : $\mathbf{y_{t+h}}$ Vektor mit der Lösung zum Zeitpunkt $t + h$

begin
  for *cycle* $\in$ *cycles* **do**
    for *slave* $\in$ *cycle.slaves* **do**
      Setze Slave-Eingangswerte unter Verwendung des Eingabevariablenvektors $\mathbf{y_t}$
      Lasse Slave bis Zeitlevel $t + h$ integrieren
      Hole Ausgabewerte und aktualisiere $\mathbf{y_{t+h}}$ (Vektorelemente werden überschrieben)

Obwohl dieser Algorithmus sequentiell beschrieben ist, können die inneren Schleifen parallelisiert werden. Die Ergebnisgrößen einer FMU-Instanz werden jeweils erst im *nächsten* Kommunikationsinterval verwendet.

5.1.3 Ergebnisse

Abbildung 3 zeigt die Ergebnisse bei Verwendung von Kommunikationsschritten der Länge 0,1 s.

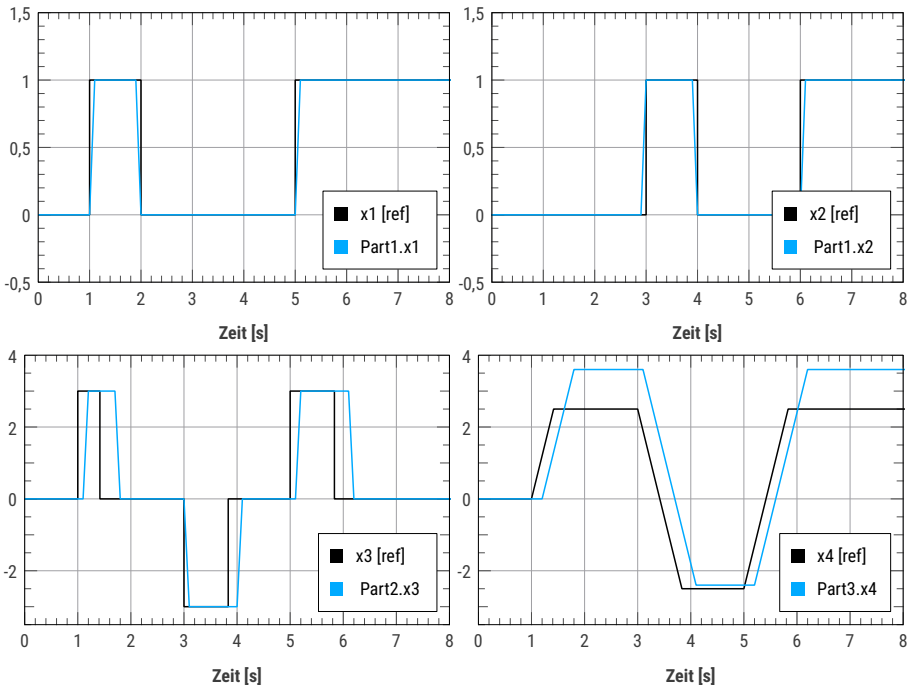

Abbildung 3: GAUSS-JACOBI, nicht-iterierend, Schrittweite: 0,1 s

Im Vergleich zur Referenzlösung (als [ref] gekennzeichnete Kurven) ist bereits eine deutliche Abweichung zu beobachten. Die Variable x_1 ändert den Zustand nicht bei $t = 1$ sondern erst am Ende des nächsten Berechnungsintervals bei $t = 1,1$s ist $x_1 = 1$. Der nächste Sprung bei $t = 2$ wird jedoch korrekt berechnet. Die Variable x_2 ändert sich korrekt zum Zeitpunkt $t = 3$ und $t = 4$, jedoch später wiederum um ein Kommunikationsinterval verzögert (erst bei $t = 6,1$ wird $x_2 = 1$ berechnet, obwohl die korrekte Lösung $x_2(6) = 1$ ist).

Diese Abweichungen von der korrekten analytischen Lösung sind auf *Rundungsfehler* bei der Verwendung von Fließkommazahlen zurückzuführen. So ergibt die 10-fache Addition von 0,1 nicht exakt 1, sondern 0,9999999999999999. Wird 20 mal 0,1 addiert ergibt das wiederum exakt 2,0. Derartige Rundungsfehler erklären die Berechnungsergebnisse der Variablen x_1 und x_2 und sind ein grundlegendes Problem bei der Verwendung von konstanten Kommunikationsschritten und zeitabhängigen Unstetigkeiten. Letztlich können solche Unstetigkeiten in den Variablen nur durch *sehr kleine Kommunikationsschritte* abgebildet werden.

Entsprechend der verzögerten Reaktion der Variablen x_1 wird erst bei Auswertung des Intervals [1,1..1,2] die Variable $x_3 = 3$ gesetzt. Diese Änderung erfährt die Variable x_4 erst im Interval [1,2..1,3], d.h. 2 Intervalle verspätet. Gleichermaßen erhält die Variable x_3 die Information über das Überschreiten der Grenze $x_4 > 2,5$ erst ein Interval zu spät, siehe Abbildung 4.

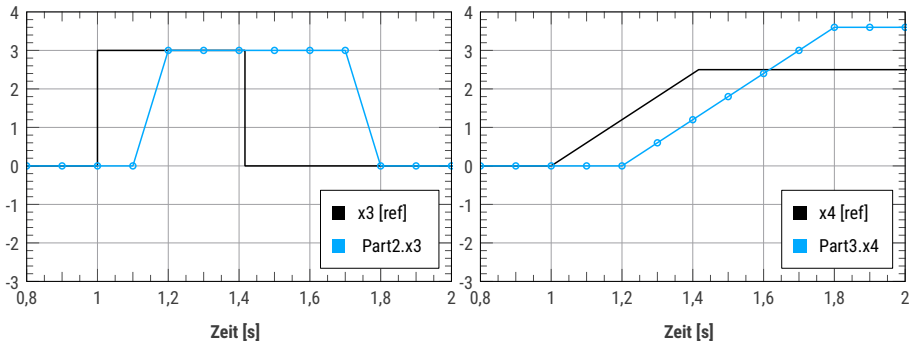

Abbildung 4: GAUSS-JACOBI, nicht-iterierend, Schrittweite: 0,1 s; Ausschnitt

Bei Integration des Intervals [1,6..1,7] in FMU „Part 2“ wurde $x_4 < 2,5$ vor Berechnung des Intervals gesetzt und damit $x_3 = 3$ am Ende des Intervals zurückgeliefert. FMU „Part 3“ wird andererseits noch mit dem alten Wert $x_3 = 3$ integriert und so ist am Ende dieses Intervals $x_4 > 2,5$. Im nächsten Interval [1,7..1,8] wird deshalb $x_3 = 0$ gesetzt (Wert am Ende des Intervals). Diese Information erhält die FMU „Part 3“ jedoch erst im Interval [1,8...1,9]. Diese zeitliche Verschiebung führt bereits zu signifikanten Unterschieden in den Ergebnissen.

5.1.4 Genauigkeitsverbesserung und Geschwindigkeit

Der Algorithmus benötigt für jeden Kommunikationsschritt exakt eine Auswertung jeder FMU-Instanz. Da $100 \times 0,1$ durch Rundungsfehler bei Gleitkommazahlen nicht exakt 10 ergibt, sondern 9,99999, wird zum Ende der Simulation ein zusätzlicher (eigentlich unnötiger) Zeitschritt bis 10,1 s ausgeführt und es werden somit insgesamt 101 Auswertungen benötigt:

```
--------------------------------------------------------------------------------
Wall clock time                             =    4.122 ms
--------------------------------------------------------------------------------
Output writing                              =    3.689 ms
Master-Algorithm                            =    0.106 ms         101
Convergence failures                        =                       0
Error test time and failure count           =    0.000 ms           0
--------------------------------------------------------------------------------
Part1                                 doStep =    0.030 ms         101
                                   getState =    0.000 ms           0
                                   setState =    0.000 ms           0
Part2                                 doStep =    0.021 ms         101
                                   getState =    0.000 ms           0
                                   setState =    0.000 ms           0
Part3                                 doStep =    0.012 ms         101
                                   getState =    0.000 ms           0
                                   setState =    0.000 ms           0
--------------------------------------------------------------------------------
```

Die Genauigkeit des Algorithmus lässt sich durch Reduktion der Kommunikationsschrittweite verbessern. Bei Verwendung von 0,001 Sekunden als Schrittlänge sind die Ergebnisse bereits sehr dicht an der Referenzlösung (siehe Abbildung 5).

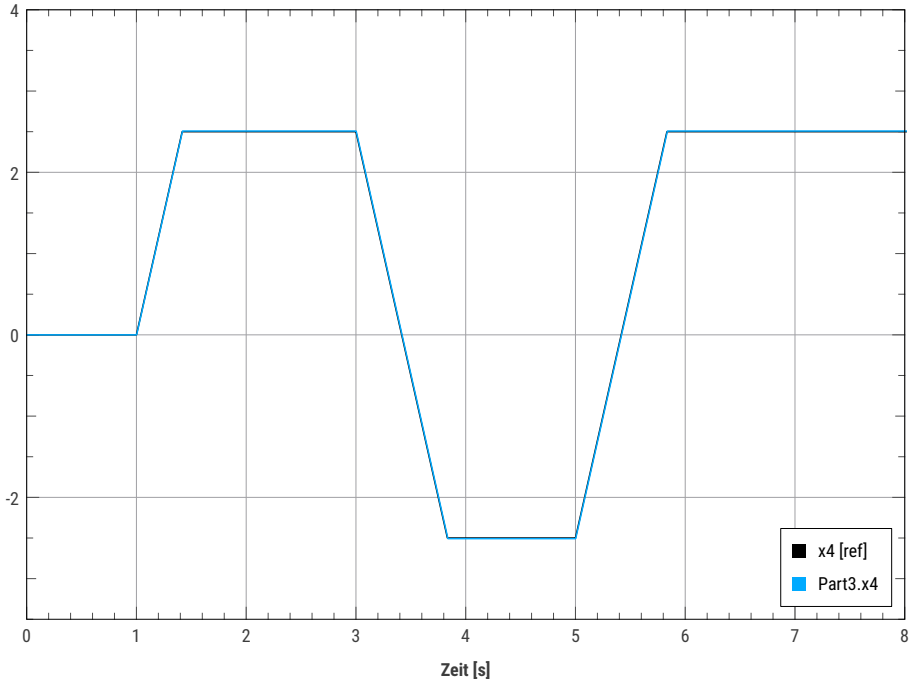

Abbildung 5: GAUSS-JACOBI, nicht-iterierend, Schrittweite: 0,001 s

Die Simulationszeit vervielfacht sich entsprechend:

```
--------------------------------------------------------------------------------
Wall clock time                             =  401.364 ms
--------------------------------------------------------------------------------
Output writing                              =  388.530 ms
Master-Algorithm                            =   10.773 ms       10001
Convergence failures                        =                       0
Error test time and failure count           =    0.000 ms           0
--------------------------------------------------------------------------------
Part1                                 doStep =    2.671 ms       10001
                                   getState =    0.000 ms           0
                                   setState =    0.000 ms           0
Part2                                 doStep =    1.987 ms       10001
                                   getState =    0.000 ms           0
                                   setState =    0.000 ms           0
Part3                                 doStep =    1.479 ms       10001
                                   getState =    0.000 ms           0
                                   setState =    0.000 ms           0
--------------------------------------------------------------------------------
```

Verwendet man die gleichen Ausgabeintervalle wie bei der Variante mit 0,1 Sekunde, ist der Aufwand für

das Ergebnisdateischreiben nun verhältnismäßig klein[2]:

```
-------------------------------------------------------------------------------
Wall clock time                          =    8.593 ms
-------------------------------------------------------------------------------
Output writing                           =    2.722 ms
Master-Algorithm                         =    4.854 ms       10001
Convergence failures                     =                       0
Error test time and failure count        =    0.000 ms           0
-------------------------------------------------------------------------------
Part1                                 doStep =    0.869 ms       10001
                                   getState =    0.000 ms           0
                                   setState =    0.000 ms           0
Part2                                 doStep =    0.820 ms       10001
                                   getState =    0.000 ms           0
                                   setState =    0.000 ms           0
Part3                                 doStep =    0.692 ms       10001
                                   getState =    0.000 ms           0
                                   setState =    0.000 ms           0
-------------------------------------------------------------------------------
```

5.2 Nicht-iterierender GAUSS-SEIDEL-Algorithmus mit fester Schrittweite

Der weitere nicht-iterierende Kopplungsalgorithmus ist der GAUSS-SEIDEL-Algorithmus. Dieser Algorithmus funktioniert ebenso auch mit FMUs der Version 1.

5.2.1 Parameter

- Länge der Kommunikationsschritte

5.2.2 Algorithmus

Algorithmus 2 : Gauss-Seidel Algorithmus

Input : t, h, $\mathbf{y_t}$ Vektor mit Gleitkommawerten zum Zeitpunkt t
Output : $\mathbf{y_{t+h}}$ Vektor mit der Lösung zum Zeitpunkt $t + h$

begin
  Initialisiere Ergebnisvektor mit Werten des letzten Zeitpunkts
  $\mathbf{y_{t+h}} := \mathbf{y_t}$
  for *cycle* $\in$ *cycles* **do**
    for *slave* $\in$ *cycle.slaves* **do**
      Setze Slave-Eingangswerte unter Verwendung des Eingabevariablenvektors $\mathbf{y_{t+h}}$
      Lasse Slave bis Zeitlevel $t + h$ integrieren
      Hole Ausgabewerte und aktualisiere $\mathbf{y_{t+h}}$ (Vektorelemente werden teilweise überschrieben)

Dieser sequentielle Algorithmus unterscheidet sich vom GAUSS-JACOBI-Algorithmus nur dadurch, dass die von einer FMU-Instanz berechneten Ergebnisgrößen bereits beim Aufruf der nächste FMU-Instanz berücksichtigt werden. Es muss also nicht erst bis zum nächsten Kommunikationsinterval gewartet werden. Allerdings setzt dieses eine korrekte bzw. zumindest geeignete Auswertungsreihenfolge voraus. Hängt, z.B. FMU 1 von den Ergebnissen der FMU 2 an, so sollte FMU 2 zuerst ausgewertet werden. Bei zyklisch gekoppelten FMUs, d.h. FMU 1 hängt von FMU 2 ab und umgekehrt, ist ohne Iteration ebenfalls mit einer Verzögerung über mehrere Kommunikationsschritte zu rechnen.

5.2.3 Ergebnisse

Abbildung 6 zeigt die Ergebnisse bei Verwendung von Kommunikationsschritten der Länge 0,1 s.

[2] Bei diesen insgesamt recht kleinen Zeiträumen fallen die parallel ausgeführten betriebssystemabhängigen Operationen zum Schreiben von Dateien besonders ins Gewicht und haben verzögern dadurch auch die eigentlichen Berechnungszeiten der FMUs. Vergleiche z.B. 4,85 ms bei der Variante mit weniger Ausgaben gegenüber 10,77 ms bei der Variante mit größeren Ausgabedateien, wobei die Berechnung der FMU in beiden Fällen identisch ist!

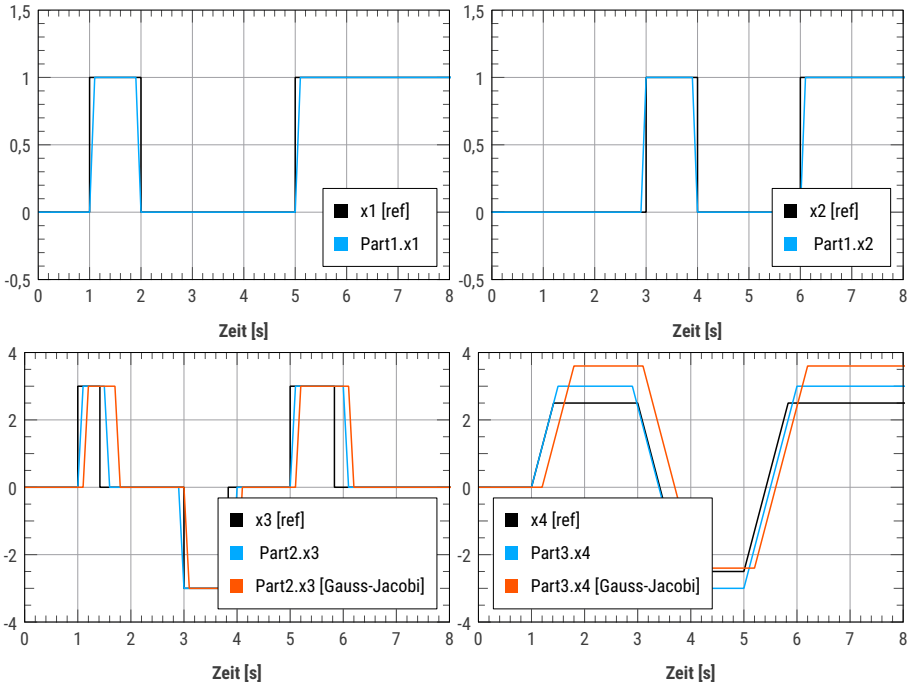

Abbildung 6: GAUSS-SEIDEL, nicht-iterierend, Schrittweite: 0,1 s

Die Ergebnisse der FMUs Part1 und Part2 sind identisch mit denen des GAUSS-JACOBI-Algorithmus, da diese ausschließlich von der Zeit abhängen. Bei den Variablen x_3 und x_4 ist der Unterschied bereits deutlich. Die Änderungen in Variablen x_1 und x_2 wirken sich direkt in der nachfolgenden Auswertung der Variable x_3 aus. Diese Änderung wird wiederum bei der anschließenden Auswertung der Variable x_4 berücksichtigt (siehe Abbildung 7).

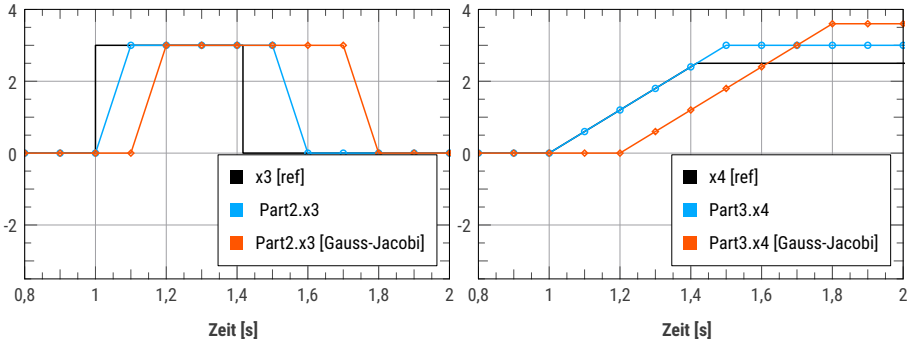

Abbildung 7: GAUSS-SEIDEL, nicht-iterierend, Schrittweite: 0,1 s; Ausschnitt

Variable x_3 ändert bereits im Interval [1...1,1] den Zustand auf $x_3 = 3$. Bei der Auswertung der Variable x_4 im gleichen Interval wird bereits dieser Anstieg für die Integration verwendet, weswegen die Kurven der Variable x_4 vom GAUSS-SEIDEL-Algorithmus und der Referenzlösung bis zum Zeitpunkt $t = 1,4$ s identisch sind. Die Rückwirkung der Variable x_4 auf die Variable x_3 wird erst ein Intervall später berücksichtigt, weswegen es ab Interval [1,4..1,5] zu Abweichungen kommt.

Interessant ist die Veränderung der Auswertungsreihenfolge, d.h. wenn Variable x_4 vor Variable x_3 berechnet wird, siehe Abbildung 8.

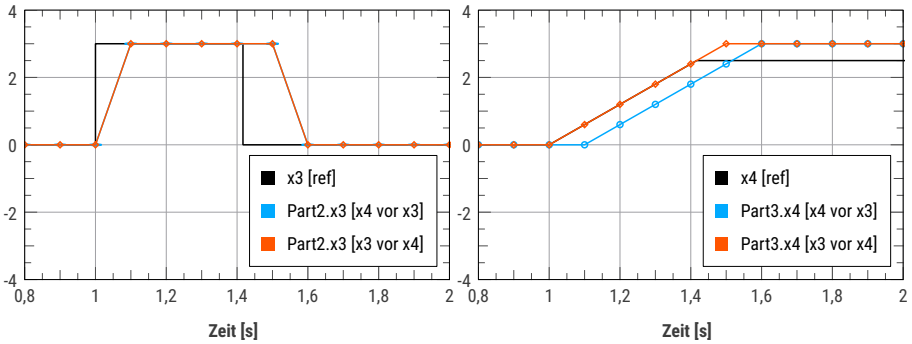

Abbildung 8: GAUSS-SEIDEL, nicht-iterierend, Schrittweite: 0,1 s; Vergleich Auswertungsreihenfolge x_3 vor x_4 bzw. x_4 vor x_3

Bei der Auswertung der Variable x_4 in Interval [1...1,1] hat x_3 noch den Wert 0. Daher ist $x_4\,(1,1) = 0$. Die Kurve x_4 ist daher um eine Kommunikationsintervalllänge verschoben. Bei der Auswertung von x_3 im Intervals [1,5..1,6] wurde x_4 bereits ausgewerted (integriert) und ist dadurch $x_4 > 2,5$. Somit wird zum Ende des Intervals $x_3 = 0$ und die Lösungen liegen wieder übereinander. Insgesamt ist die Variante mit der Auswertung von x_3 vor x_4 dichter an der Referenzlösung dran. Es zeigt sich also, dass auch bei zyklischen Abhängigkeiten die Natur der Gleichungen in Verbindung mit der Auswertungsreihenfolge über die Ergebnisgenauigkeit entscheidet.

5.2.4 Genauigkeitsverbesserung und Geschwindigkeit

Der GAUSS-SEIDEL-Algorithmus benötigt genausoviele FMU-Auswertungen (`doStep`-Aufrufe), wie der GAUSS-JACOBI-Algorithmus und ist deshalb auch gleich schnell. Eine Reduktion der Kommunikationsschrittlänge führt auch hier zur Genauigkeitsverbesserung. Im Vergleich zum GAUSS-JACOBI-Algorithmus ist die zeitliche Verschiebung beim GAUSS-SEIDEL-Algorithmus nur ein statt zwei Intervalllängen, weswegen eine Reduktion der Zeitschrittlänge auf 0,002 s zu annähernd gleich genauen Ergebnissen führt.

5.3 Iterierender GAUSS-SEIDEL-Algorithmus mit konstanter Schrittweite

Die Verzögerung bzw. der Fehler durch die verspätete Rückmeldung der Aktualisierung der Variable x_4 könnte durch ein iteratives Verfahren behoben werden. Dafür wird jedes Interval mehrfach berechnet. Bei iterativen Verfahren gibt es unterschiedliche Möglichkeiten, die Beendigung der Iteration zu regeln. Typischerweise wird Konvergenz anhand einer Norm der Unterschiede aller Variablen definiert, welches natürlich nur für Gleitkommazahlen anwendbar ist. In MASTERSIM wird die Wurzel der gewichteten Quadratsummen als Norm verwendet:

$$|y|_{WRMS} = \frac{1}{n}\sqrt{\sum_{i=1}^{n}\left(\frac{y_{neu,i} - y_{alt,i}}{|y_{neu,i}|\,r_{tol} + a_{tol}}\right)^2}$$

wobei Konvergenz bei $|y|_{WRMS} \leq 1$ erreicht ist.

Für alle boolischen, Zeichenketten- und Integerausgaben kann nur auf Gleichheit geprüft werden. Alternativ kann man diese Ausgabegrößen auch aus der Konvergenzanalyse ausnehmen (in diesem Testfall sind nur Gleitkommazahlenausgaben enthalten, daher hat das keinen Einfluss).

Für die Verwendung eines Iterationsalgorithmus muss jede beteiligte FMU die Rücksetzfunktionalität gemäß FMI-Standard 2.0 implementieren.

5.3.1 Parameter

- Länge der Kommunikationsschritte
- Anzahl der maximal zulässigen Iterationen
- Konvergenzkriterium (relative und absolute Toleranz für Normbildung bei Gleitkommazahlen)

5.3.2 Algorithmus

Der Algorithmus benötigt einen temporären Vektor $\mathbf{y^m_{t+h}}$ zur Speicherung der iterativen Größen.

Algorithmus 3 : Iterativer GAUSS-SEIDEL Algorithmus

Input : t, h, $\mathbf{y_t}$ Vektor mit Gleitkommawerten zum Zeitpunkt t
Output : $\mathbf{y_{t+h}}$ Vektor mit der Lösung zum Zeitpunkt $t + h$

```
begin
    y_{t+h} := y_t
    for cycle ∈ cycles do
        while iteration < maxIterations do
            Speichere iterative Lösung für Konvergenztest
            y^m_{t+h} := y_{t+h}
            if iteration > 1 then
                Slave-Zustand zurücksetzen (nur bei mehr als einem Slave pro Zyklus)
            for slave ∈ cycle.slaves do
                Setze Slave-Eingangswerte unter Verwendung des Eingabevariablenvektors y_{t+h}
                Lasse Slave bis Zeitlevel t + h integrieren
                Hole Ausgabewerte und aktualisiere y_{t+h} (Vektorelemente werden teilweise überschrieben)
            if cycle.slaves.count() == 1 then
                Bei nur einem Slave im Zyklus muss nicht iteriert werden
                break
            Berechne Vektornorm der Unterschiede zwischen neuer und alter (iterativer) Lösung
            res = WRMS (y^m_{t+h}, y_{t+h})
            if res < 1 then
                Konvergiert
                break
        if iteration >= maxIterations then
            Maximale Anzahl der Iterationen überschritten, Algorithmus hat nicht konvergiert
            return IterationLimitExceeded
```

Die Iteration wird jeweils unabhängig für jeden Zyklus angewendet. Enthält ein Zyklus nur eine FMU, ist eine Iteration nicht notwendig. Für den Fall, dass Konvergenz nicht erreicht werden kann, ist es möglich, mit den zuletzt berechneten Werten fortzufahren. Hierbei kann die Anzahl der durchgeführten Iterationen einen maßgeblichen Einfluss auf das Ergebnis haben (siehe Diskussion unten).

5.3.3 Ergebnisse

Abbildung 9 zeigt die Ergebnisse der Variablen x_3 und x_4 bei Verwendung von Kommunikationsschritten der Länge 0,1 s unter Verwendung von Iteration mit maximal 2 Iterationsschritten. Die Ergebnisse für x_1 und x_2 sind unverändert (nur zeitabhängige Größen).

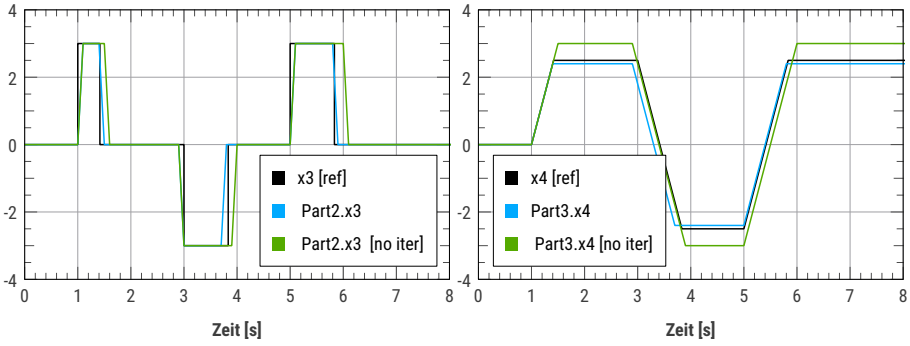

Abbildung 9: GAUSS-SEIDEL, iterierend, max 2. Iterationen, Schrittweite: 0,1 s; Zum Vergleich sind die Ergebnisse des nicht-iterierenden GAUSS-SEIDEL-Algorithmus nocheinmal gezeigt („no-iter"-Kurven)

Die Zustandsereignisse, d.h. die Rückwirkung der Variable x_4 auf die Variable x_3 ist klar zu erkennen. Interessant ist jedoch die Ausgabe das MASTERSIM Algorithmus beim Auftreten des ersten Zustandsereignisses im Interval [1,4..1,5].

```
MASTER: step =    12, t = 1.3, h_next = 0.1, errFails = 0
 1.300 s    01/01/00  0:00:01  0.003 s     7.760 min/s  8.775 min/s  0.017 s
  GAUSS-SEIDEL: t = 1.3, dt = 0.1, Cycle #1, Iteration #1
  GAUSS-SEIDEL: WRMS norm =        10714
  GAUSS-SEIDEL: t = 1.3, dt = 0.1, Cycle #1, Iteration #2
  GAUSS-SEIDEL: WRMS norm =            0
MASTER: step =    13, t = 1.4, h_next = 0.1, errFails = 0
 1.400 s    01/01/00  0:00:01  0.003 s     7.675 min/s  8.588 min/s  0.017 s
  GAUSS-SEIDEL: t = 1.4, dt = 0.1, Cycle #1, Iteration #1
  GAUSS-SEIDEL: WRMS norm =         8824
  GAUSS-SEIDEL: t = 1.4, dt = 0.1, Cycle #1, Iteration #2
  GAUSS-SEIDEL: WRMS norm =        38243
Step failure at t=1.4, taking step size 0.1. Reduction of time step is not
enabled, continuing with potentially inaccurate values.
```

Der entsprechende Zeitbereich ist in Abbildung 10 nocheinmal vergrößert dargestellt:

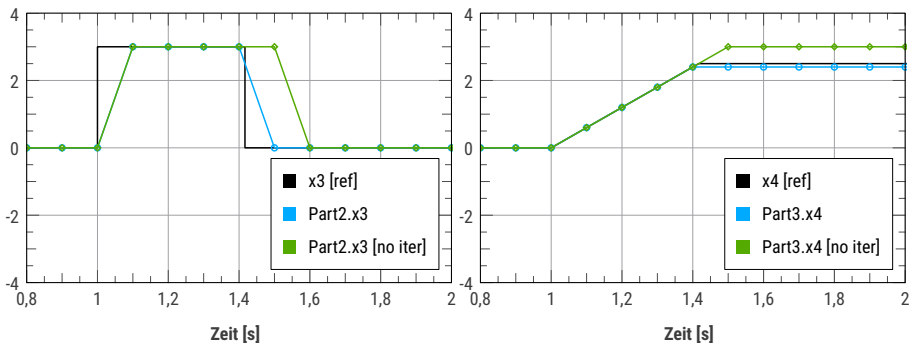

Abbildung 10: GAUSS-SEIDEL, iterierend, max 2. Iterationen, Schrittweite: 0,1 s; Ausschnitt

In Schritt 12, d.h. Interval [1,3..1,4] ist zu Beginn des Intervals $x_3 = 3$ und $x_4 = 1,8$. Nach dem ersten Iterationschritt in diesem Interval ist $x_3 = 3$ (da $x_4 < 2,5$) und $x_4 = 2,4$. Die Abweichung zwischen $1,8$ und $2,4$ übersteigt das Konvergenzlimit und ein weiterer Iterationsschritt wird ausgeführt. Nach diesem ist wiederum $x_3 = 3$ und $x_4 = 2,4$, dadurch ist die Differenz zum letzten Iterationsschritt =0 und Konvergenz ist erreicht.

In Schritt 13, d.h. Interval [1,4..1,5] ändern sich die Zustände im Verlauf der Iterationen wie folgt:

```
                     x3       x4       WRMS-Norm
-------------------------------------------------
Anfangswerte:        3        2.4            -
--
Nach 1. Iteration:   3        3           8824
Nach 2. Iteration:   0        2.4        38243
... (würde man die Iteration fortsetzen) ...
Nach 3. Iteration:   3        3         150259
Nach 4. Iteration:   0        2.4        38243
.
.
.
```

Nach der 1. Iteration ist $x_4 > 2,5$, weswegen x_3 in der 2. Iteration zu $x_3 = 0$ wird. Dadurch bleibt $x_4 = 2,4$ in der darauffolgenden Auswertung der FMU. Für ein solches Problem kann kein Iterationsalgorithmus eine korrekte Lösung finden. Jedoch ist die Lösung nach der 2. bzw. 4. Iteration besser als die nach nur einer oder 3 Iterationen (siehe Vergleich in Abbildung 10).

Diese Aussage lässt sich jedoch nicht verallgemeinern und die optimale Anzahl von Iterationen bzw. des Konvergenzkriteriums hängt maßgeblich von den beteiligten Gleichungen ab.

5.3.4 Genauigkeitsverbesserung und Geschwindigkeit

In der Variante mit 2 Iterationen benötigt der Algorithmus insgesamt ca. 30 % mehr Auswertungen der im Zyklus gekoppelten FMUs:

```
1  ------------------------------------------------------------------------
2  Wall clock time                          =    4.223 ms
3  ------------------------------------------------------------------------
4  Output writing                           =    3.671 ms
5  Master-Algorithm                         =    0.149 ms         101
6  Convergence failures                     =                       0
7  Error test time and failure count        =    0.000 ms           0
8  ------------------------------------------------------------------------
9  Part1                              doStep =    0.020 ms         101
10                                  getState =    0.011 ms         101
11                                  setState =    0.000 ms           0
12 Part2                              doStep =    0.020 ms         130
13                                  getState =    0.005 ms         101
14                                  setState =    0.001 ms          29
15 Part3                              doStep =    0.016 ms         130
16                                  getState =    0.008 ms         101
17                                  setState =    0.002 ms          29
18 ------------------------------------------------------------------------
```

Auch wird etwas Rechenzeit für das Holen und Zurücksetzen der FMU-Zustände benötigt[3]. Insgesamt ist der Mehraufwand jedoch klein im Vergleich zur nicht-iterierenden GAUSS-SEIDEL-Variante. Die Zustandsereignisse werden bei Verwendung von maximal zwei Iterationen wesentlich besser erfasst.

Führt man die Berechnung mit Schrittweiten von 0.005 Sekunden durch, so erhält man nahezu korrekte Ergebnisse (siehe Abbildung) bei fast unveränderter Rechenzeit (Ausgabezeit nimmt zu) [4]:

```
1  ------------------------------------------------------------------------
2  Wall clock time                          =   72.416 ms
3  ------------------------------------------------------------------------
4  Output writing                           =   68.094 ms
5  Master-Algorithm                         =    2.485 ms        2000
6  Convergence failures                     =                       0
7  Error test time and failure count        =    0.000 ms           0
8  ------------------------------------------------------------------------
9  Part1                              doStep =    0.692 ms        2000
10                                  getState =    0.278 ms        2000
11                                  setState =    0.000 ms           0
12 Part2                              doStep =    0.422 ms        2000
13                                  getState =    0.175 ms        2000
14                                  setState =    0.000 ms           0
15 Part3                              doStep =    0.240 ms        2000
16                                  getState =    0.164 ms        2000
17                                  setState =    0.000 ms           0
18 ------------------------------------------------------------------------
```

[3] Bei derart kleinen Messzeiträumen sind die Zeiten höchstens als Indikatoren zu verwenden, da Störgrößen (Resourcenauslastung) durch das Betriebssystem die Messwerte wesentlich verfälschen können.

[4] In diesem trivialen Benchmark ist die Auswertungszeit innerhalb der FMUs marginal und fast der gesamte Arbeitsaufwand liegt beim MASTERSIM Programm. In realen Anwendungsfällen liegt die Auswertungszeit der FMU, konkret der `doStep()` Funktion um ein Vielfaches höher, sodass die Gesamtsimulationszeit sichtbar mit der Anzahl der FMU-Auswertungen skaliert.

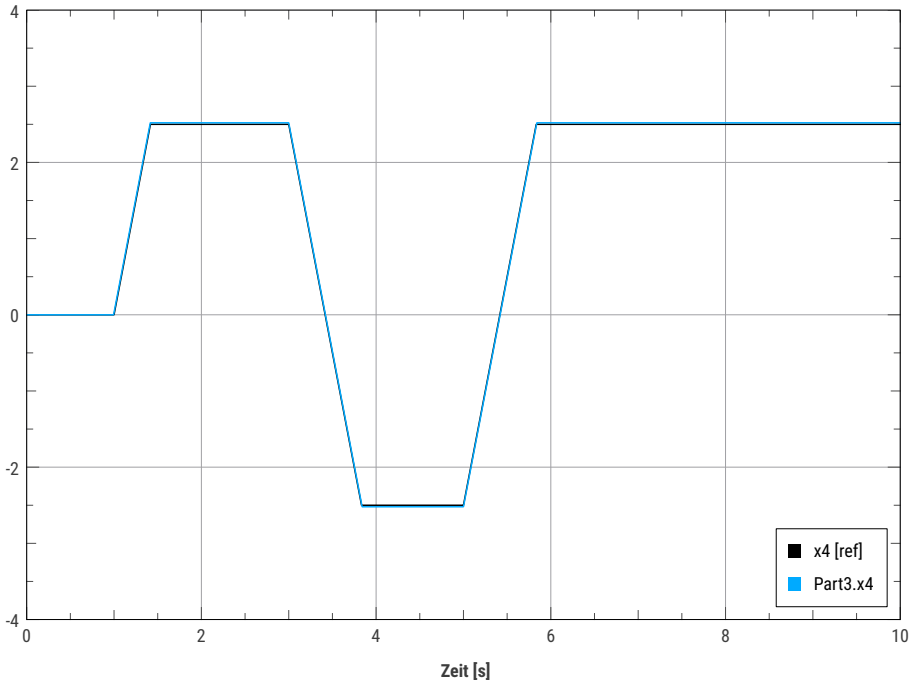

Abbildung 11: GAUSS-SEIDEL, iterierend, max 2. Iterationen, Schrittweite: 0,005 s

5.4 Iterierender GAUSS-SEIDEL-Algorithmus mit variabler Schrittweite, ohne Fehlerschätzer

Eine direkte Erweiterung des iterierenden GAUSS-SEIDEL-Algorithmus ist die Verwendung variabler Zeitschrittlängen. Diese Eigenschaft müssen allerdings alle FMUs unterstützen. Bei diesem Algorithmus wird der Zeitschritt immer dann reduziert, wenn ein Konvergenzfehler auftritt. Damit die Simulation nicht bei oben beschriebenen Situationen einer Unstetigkeit abbricht, sollte eine untere Zeitschrittlänge als Grenze definiert werden, unterhalb derer der Konvergenztest ausgeschaltet ist. Der Zeitschritt wird bei erfolgreicher Konvergenz vergrößert, üblicherweise um einen konstanten Faktor. Bei Konvergenz nach exakt einer Iteration (lineares Problem) kann der Zeitschritt direkt auf den maximal zulässigen Zeitschritt angepasst werden.

5.4.1 Parameter

Zusätzlich zu den für den iterierenden GAUSS-SEIDEL-Algorithmus benötigten Parametern sind folgende Parameter notwendig:

- max. Länge der Kommunikationsschritte
- min. Länge der Kommunikationsschritte (darunter wird Konvergenztest ausgeschaltet)
- Faktor zur Vergrößerung/Verkleinerung der Zeitschritte bei Konvergenzfehler (bei linearen Problemen mit Unstetigkeiten sollte dieser Faktor groß sein, bei nicht-linearen Problemen bzw. nicht-linearer Kopplung sollte der Faktor eher moderat sein)

5.4.2 Algorithmus

Algorithmus 4 : Iterativer Gauss-Seidel Algorithmus mit Zeitschrittweitenanpassung

Input : t, h, $\mathbf{y_t}$ Vektor mit Gleitkommawerten zum Zeitpunkt t
Output : $\mathbf{y_{t+h}}$ Vektor mit der Lösung zum Zeitpunkt $t + h$

begin
- $\mathbf{y_{t+h}} := \mathbf{y_t}$
- **for** *cycle* $\in$ *cycles* **do**
  - **while** *iteration* $<$ *maxIterations* **do**
    - Speichere iterative Lösung für Konvergenztest
    - $\mathbf{y^m_{t+h}} := \mathbf{y_{t+h}}$
    - **if** *iteration* $>$ *1* **then**
      - Slave-Zustand zurücksetzen (nur bei mehr als einem Slave pro Zyklus)
    - **for** *slave* $\in$ *cycle.slaves* **do**
      - Setze Slave-Eingangswerte unter Verwendung des Eingabevariablenvektors $\mathbf{y_{t+h}}$
      - Lasse Slave bis Zeitlevel $t + h$ integrieren
      - Hole Ausgabewerte und aktualisiere $\mathbf{y_{t+h}}$ (Vektorelemente werden teilweise überschrieben)
    - **if** *cycle.slaves.count()* $==$ *1* **then**
      - Bei nur einem Slave im Zyklus muss nicht iteriert werden
      - **break**
    - **if** $h < h_{limit}$ **then**
      - Bei Zeitschrittadaption, breche Iteration ab, wenn Zeitschritt zu klein wird (notwendig, um Unstetigkeiten zu überwinden)
      - **break**
    - Berechne Vektornorm der Unterschiede zwischen neuer und alter (iterativer) Lösung
    - $res = \mathrm{WRMS}\left(\mathbf{y^m_{t+h}}, \mathbf{y_{t+h}}\right)$
    - **if** $res < 1$ **then**
      - Konvergiert
      - **break**
  - **if** *iteration* $>=$ *maxIterations* **then**
    - Maximale Anzahl der Iterationen überschritten, Algorithmus hat **nicht** konvergiert
    - **return IterationLimitExceeded**

Dieser Algorithmus ist im Gegensatz zum vorangehenden Algorithmus nur um dem Teil des schrittweitenabhängigen Auslassen des Konvergenztests verändert. Die eigentliche Anpassung des Zeitschritts erfolgt im allgemeinen Teil des Simulationsmasters (siehe Algorithmus 5).

5.4.3 Ergebnisse

Abbildung 12 zeigt die Ergebnisse bei Verwendung einer maximalen Kommunikationsschrittlänge von 0,14 s, einer minimalen Kommunikationsschrittlänge von 0,005 s, einem Reduktionsfaktor von 5 bei Konvergenzfehler und Vergrößerungsfaktor 2 bei Konvergenz. Maximal 2 Iterationen sind erlaubt.

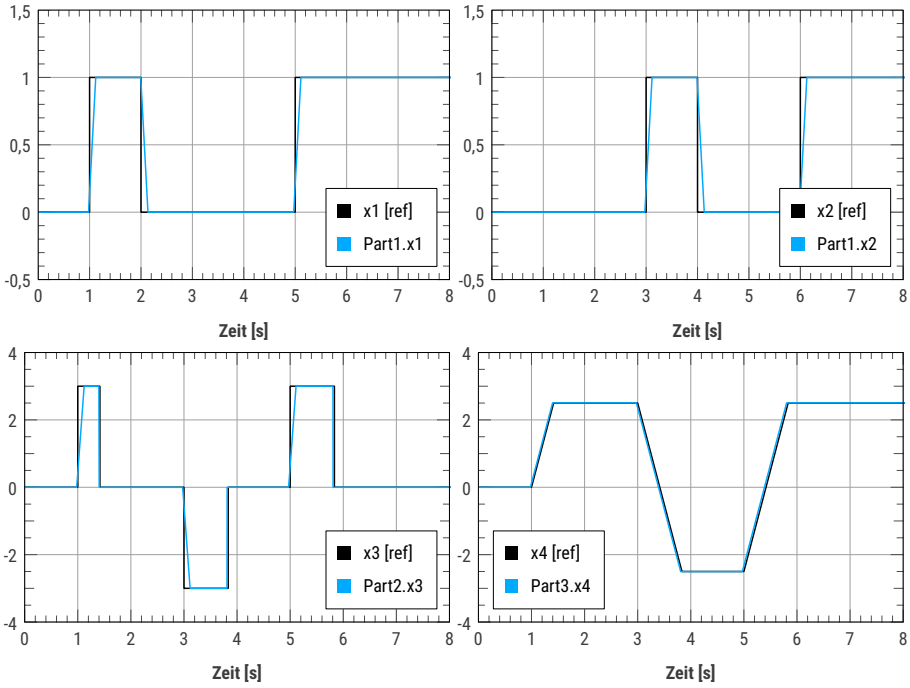

Abbildung 12: GAUSS-SEIDEL, iterierend, max 2. Iterationen, variable Schrittweite (max. 0,14 s, min. 0,005 s)

Die Solverstatistiken zeigen im Vergleich zum GAUSS-SEIDEL Verfahren mit Iteration nur geringfügig höhere Anzahl von FMU Auswertungen.

```
------------------------------------------------------------------------------
Wall clock time                          =    3.574 ms
------------------------------------------------------------------------------
Output writing                           =    3.020 ms
Master-Algorithm                         =    0.198 ms          101
Convergence failures                     =                       15
Error test time and failure count        =    0.000 ms            0
------------------------------------------------------------------------------
Part1                              doStep =    0.030 ms          116
                                getState =    0.012 ms          101
                                setState =    0.001 ms           15
Part2                              doStep =    0.046 ms          159
                                getState =    0.015 ms          101
                                setState =    0.003 ms           58
Part3                              doStep =    0.023 ms          159
                                getState =    0.010 ms          101
                                setState =    0.003 ms           58
------------------------------------------------------------------------------
```

Ohne ins Detail zu schauen, stellt man bereits folgendes fest:

- wie bisher werden Unstetigkeiten in den rein zeitabhängigen Variablen x_1 und x_2 einfach übergangen und wirken sich als Fehler in den Variablen x_3 und x_4 aus,
- Zustandsereignisse werden durch die auftretenden Konvergenzfehler zuverlässig erkannt und der Zeitschritt kurzzeitig bis auf das zulässige Minimum reduziert.

Der interessante Zeitbereich des ersten Auftretens eines Zustandsereignisses ist in Abbildung 13 nocheinmal vergrößert dargestellt:

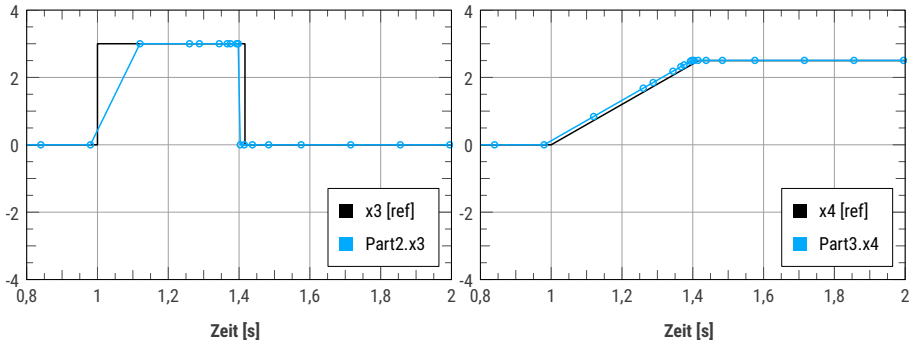

Abbildung 13: GAUSS-SEIDEL, iterierend, max 2. Iterationen, variable Schrittweite (max. 0,14 s, min. 0,005 s); Ausschnitt

Die zeitliche Verschiebung der Kurve x_3 und x_4 liegt am verspäteten Sprung der Variable x_1 (erst zum Zeitpunkt $t = 1,12$). Das weitere Verhalten stellt sich im Algorithmus wie folgt dar:

- $t = 1,344s$ wird mit maximalem Zeitschritt $0,14s$ erreicht, ein Fortsetzen mit gleicher Zeitschrittlänge führt zur Überschreitung von $x_4 > 2,5$ und einem Konvergenzfehler
- Der Zeitschritt wird daraufhin durch 5 geteilt ($\Delta t = 0,0224s$) und es wird $t = 1,366s$ erreicht; es bleibt $x_3 = 3$ und $x_4 < 2,5$ und der Zeitschritt wird wieder verdoppelt auf $\Delta t = 0,0448s$.
- Für das nächste Interval ist $0,0448s$ allerdings wieder zu lang, eine Reduktion auf $\Delta t = 0,00896s$ bringt die Lösung näher an die Sprungstelle. Wieder wird nach Konvergenz der Zeitschritt optimistisch verdoppelt.
- Das Interval von 1,375s bis 1,393s wird mit einer Iteration durchlaufen; es bleibt $x_3 = 3$ und $x_4 < 2,5$.
- Im nächsten Interval ist die Sprungstelle erreicht, d.h. der Zeitschritt wird zunächst auf $\Delta t = 0,007168s$ verringert, und dann erneut verkleinert auf $\Delta t = 0,0014336s$. Diese Zeitschrittlänge ist unterhalb der festgelegten Grenze von $0,005s$, sodass das mit diesem Zeitschritt erhaltene Ergebnis ungeachtet des Konvergenzfehlers beibehalten wird.
- Danach wird der Zeitschritt verdoppelt auf $\Delta t = 0,0028672s$, welcher immer noch unter der Zeitschrittgrenze von $0,005s$ liegt. Dadurch wird in diesem Schritt auf Iteration verzichtet und der Algorithmus verhält sich wie der nicht-iterierende GAUSS-SEIDEL-Algorithmus.
- Da die Unstetigkeit überwunden ist, werden in den nachfolgenden Intervallen jeweils nur eine Iteration benötigt und die Zeitschritte werden in jedem Schritt bis zum Erreichen der maximalen Schrittlänge verdoppelt.

Abbildung 14 zeigt nocheinmal die letztendlich verwendeten Zeitschritte und die dazugehörige Variable x_4.

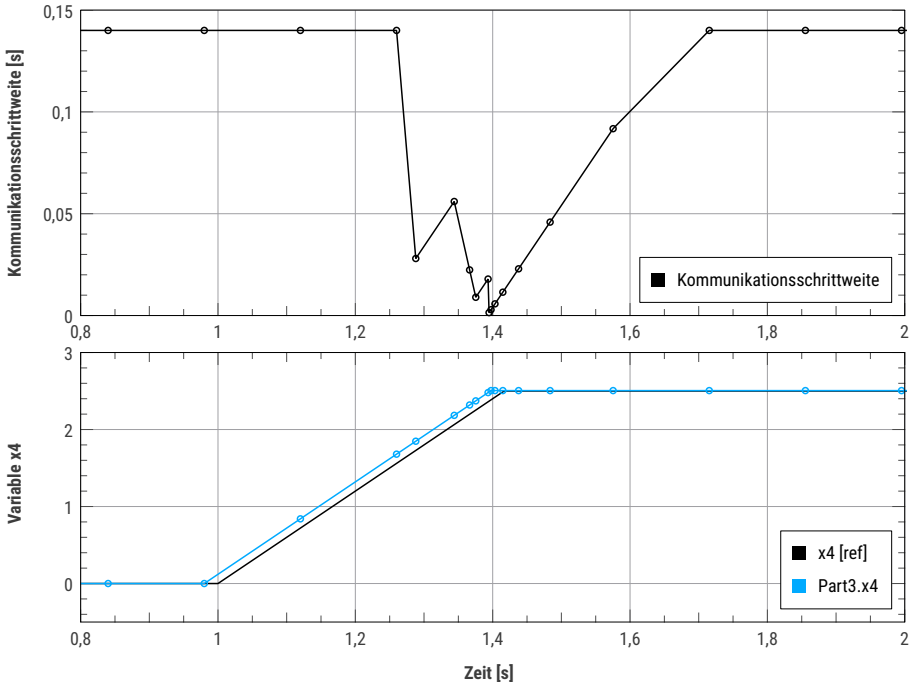

Abbildung 14: GAUSS-SEIDEL, iterierend, max 2. Iterationen, variable Schrittweite (max. 0,14 s, min. 0,005 s); Ausschnitt

5.5 Schrittweitenanpassung durch Fehlerschätzer

Alle bisherigen Algorithmen waren nicht in der Lage, die Unstetigkeiten rein zeitabhängiger Variablen zu erkennen. Auch bei stetig verlaufenden Variablen ist die Genauigkeit der Lösung von den gewählten Zeitschritten abhängig. Daher ist ein Fehlerschätzer sinnvoll, wobei der Doppel-Schritt/Richardson-Fehlerschätzer eine sinnvolle Variante ist. Dabei wird ein Berechnungsschritt zunächst mit dem gewählten Kommunikationsschritt absolviert. Bei Konvergenz wird der gleiche Zeitraum nochmal in zwei Schritten mit jeweils halber Länge berechnet. Aus den Unterschieden in den Ergebnissen lässt sich der Fehler abschätzen (Abb. 15)

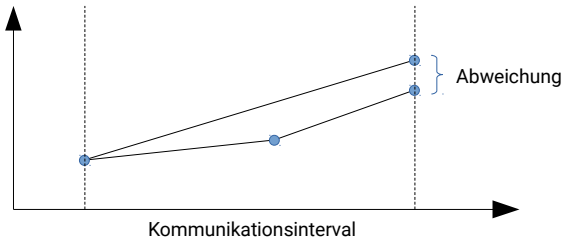

Abbildung 15: Illustration des Doppel-Schritt/Richardson-Fehlerschätzers

Es gibt allerdings ein Problem mit diesem Fehlerschätzer, sobald innerhalb des Intervals eine Unstetigkeit wie im Fall der Variable x_1 auftritt. In diesem Fall sind die Ergebnisse eines vollem Schritts oder von zwei Halbschritten identisch - die Unstetigkeit wird nicht bemerkt (siehe Abbildung 16).

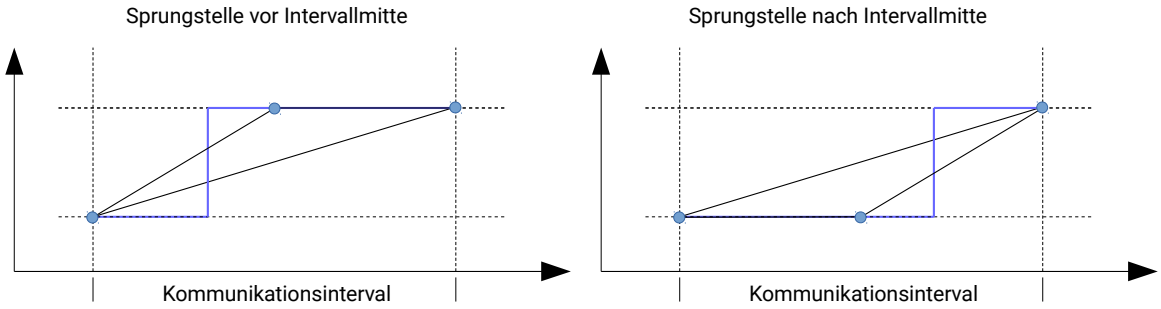

Abbildung 16: Unvermögen des Doppel-Schritt/Richardson-Fehlerschätzers Unstetigkeiten in der Lösung zu erkennen; die blaue Kurve zeigt die korrekte Lösung, die Punkte jeweils die Ergebnisse der FMU Auswertung

Eine Lösung besteht darin, den Fehlerschätzer basierend auf den Funktionsergebnissen *und* Ableitungen am Ende des Doppelschritts zu konstruieren. Ohne spezifische Unterstützung können FMUs keine Ableitungsinformation zurückgeben, daher müssen Ableitungsinformationen aus den Stützstellen approximiert werden (siehe Abbildung 16). Wie in der Abbildung zu sehen ist, stimmen zwar die Ergebnisse am Ende des Intervals überein, jedoch sind die Anstiege im vollen und in den Halbintervallen unterschiedlich.

5.5.1 Parameter

- relative Toleranz für Fehlerschätzer
- absolute Toleranz für Fehlerschätzer
- minimaler Zeitschritt (darunter wird Fehlerschätzer deaktiviert); sollte kleiner oder gleich dem Zeitschrittlimit für das Abschalten des Konvergenztests sein
- maximaler Skalierungsfaktor bei Vergrößerung des Zeitschritts
- minimaler Skalierungsfaktor bei Verkleinerung des Zeitschritts

5.5.2 Algorithmus

Der eigentliche Fehlertestalgorithmus ist in den äußeren Algorithmus zur Anpassung der Kommunikationsschrittweite integriert:

Algorithmus 5 : Algorithmus zur Zeitschrittanpassung

Input : h_0, $\mathbf{y_t}$ Vektor mit Gleitkommawerten zum Zeitpunkt t, Slaves sind positioniert bei t
Output : $\mathbf{y_{t+h}}$ Vektor mit der Lösung zum Zeitpunkt $t + h$, Slaves positioniert bei $t + h$, h_0

begin
- Übertrage Schätzwert h_0 für Kommunikationsintervallänge:
- $h = h_0$
- Falls Iteration oder Fehlerschätzer eingeschaltet ist, sichere aktuelle Zustände der Slaves
- Zeitschrittreduktionsschleife:
- **while** *true* **do**
  - Führe gewählten Masteralgorithmus aus (iterativ oder nicht-iterativ):
  - `doStep()`
  - Slaves sind jetzt positioniert bei $t + h$, Vektor y_{t+h} ist berechnet.
  - **if** *Konvergenzfehler oder Iterationslimit überschritten?* **then**
    - **if** *Zeitschrittanpassung eingeschaltet?* **then**
      - Reduktion des Zeitschritts (konstanter Teiler 5), dabei untere Grenze nicht unterschreiten
      - $h := \mathbf{max}\,(h_{lim}, h/5)$
      - Slavezustände auf t und y_t zurücksetzen.
      - **continue**
    - **else**
      - Warnung ausgeben und mit gleichem Zeitschritt fortfahren
      - **break**
  - **if** *Fehlertest verwenden?* **then**
    - Fehlertest durchführen, dabei wird gegebenenfalls h angepasst.
    - **if** *Fehlertest erfolgreich?* **then**
      - Neuer Schätzwert für den Zeitschritt h_0 für den nächsten Schritt wurde bereits gesetzt.
      - **break**
- Schritt erfolgreich durchgeführt.
- $t := t + h$
- **if** *Zeitschrittadaption eingeschaltet?* **then**
  - **if** *Kein Fehlertest verwendet?* **then**
    - Neuen Schätzwert für nächste Kommunikationsschrittlänge bestimmen (konstanter Faktor 2):
    - $h_0 := \mathbf{min}\,(h_{max}, h * 5)$
  - Sicherstellen, dass das Simulationsende nicht überschritten wird:
  - **if** $t + h_0 > t_{end}$ **then**
    - $h_0 := t_{end} - t$

Der eigentliche Fehlertestalgorithmus wird nach Prüfung auf Konvergenzfehler aufgerufen. Hierbei sind verschiedene Fehlertestalgorithmen möglich. Nachfolgend ist der Doppel-Schritt bzw. Richardson-Extrapolations-

Fehlertestalgorithmus gezeigt (Algorithmus 6):

Algorithmus 6 : Doppel-Schritt/Richardson-Extrapolations-Fehlerschätzer

Input : t, $\mathbf{y_t}$ Vektor mit Gleitkommawerten zum Zeitpunkt t, Slaves sind positioniert bei $t + h$, h verwendeter Kommunikationsschritt, $\mathbf{y_{t+h}}$ Vektor mit Gleitkommawerten zum Zeitpunkt $t + h$
Output : h_0
return *Ob Fehlertest erfolgreich war oder nicht*

begin
- Ergebnissgrößen $y_{t+h}^{(h)}$ zwischenspeichern.
- Slaves auf Zustand zum Zeitpunkt t zurücksetzen
- Systemzustände für Zeitpunkt t für eventuelles Rücksetzen nach Fehlertest speichern
- Masteralgorithmus durchführen
- `doStep`$(t \longrightarrow t + h/2)$
- **if** *Konvergenzfehler?* **then**
  - Fehlertest gescheitet. Fehlerflag setzen.
- **else**
  - Schritt erfolgreich durchgeführt, Zeitpunkt $t + h/2$ erreicht
  - Masteralgorithmus durchführen
  - `doStep`$(t + h/2 \longrightarrow t + h)$
  - **if** *Konvergenzfehler?* **then**
    - Fehlertest gescheitet. Fehlerflag setzen.
- **if** *Fehlerflag gesetzt?* **then**
  - Fehlernorm auf großen Wert setzen
- **else**
  - Richardson-Fehlerschätzer auswerten
  - Anstiegstest auswerten
- **if** *Fehlernorm > 1?* **then**
  - Slaves auf t zurücksetzen
  - h_0 basierend auf Fehlerschätzer reduzieren
  - Rückkehr mit Fehlerindikation.
- **else**
  - h_0 basierend auf Fehlerschätzer anpassen/vergrößern
  - Rückkehr mit Erfolg.

5.5.3 Ergebnisse

Abbildung 17 zeigt die Ergebnisse bei Verwendung der gleichen Parameter wie beim vorangehenden Test und zusätzlich verwendetem Fehlerschätzer. Relative und absolute Toleranz sind je auf 10^{-5} gesetzt und der Kommunikationsschritt ist mindestens 10^{-5} s lang.

Auf den ersten Blick sind die Ergebnisse mit den Referenzergebnissen identisch, oder zumindest sehr ähnlich.

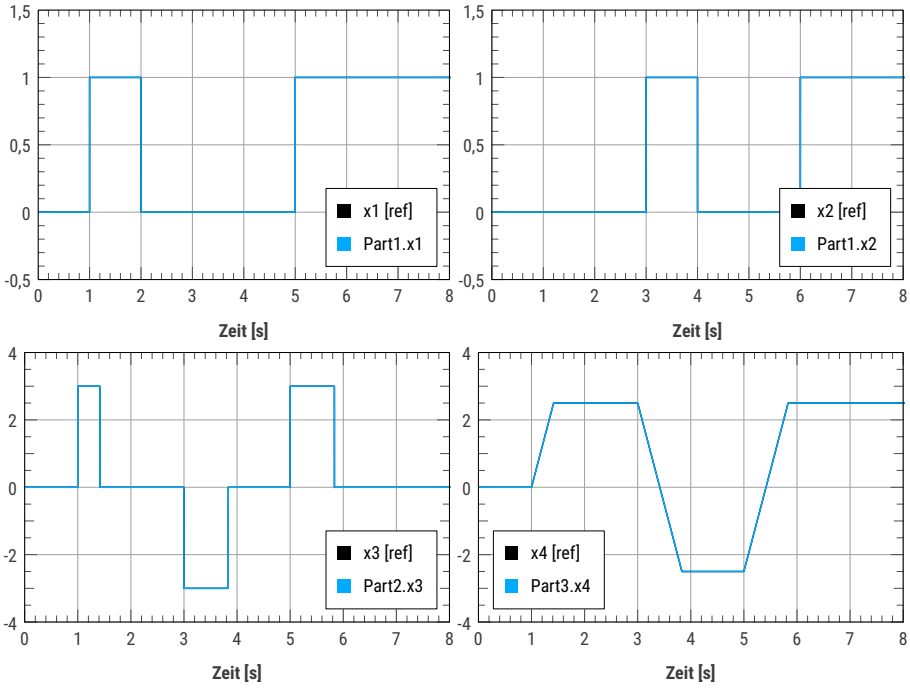

Abbildung 17: GAUSS-SEIDEL, iterierend, max 2. Iterationen, variable Schrittweite, Doppel-Schritt/Richardson-Fehlerschätzer

Betrachtet man wiederum das Detail (Abb. 18) um die ersten Zeit- und Zustandsereignisse, sieht man klar die vielen Auswertungen in der Nähe der Sprungstellen.

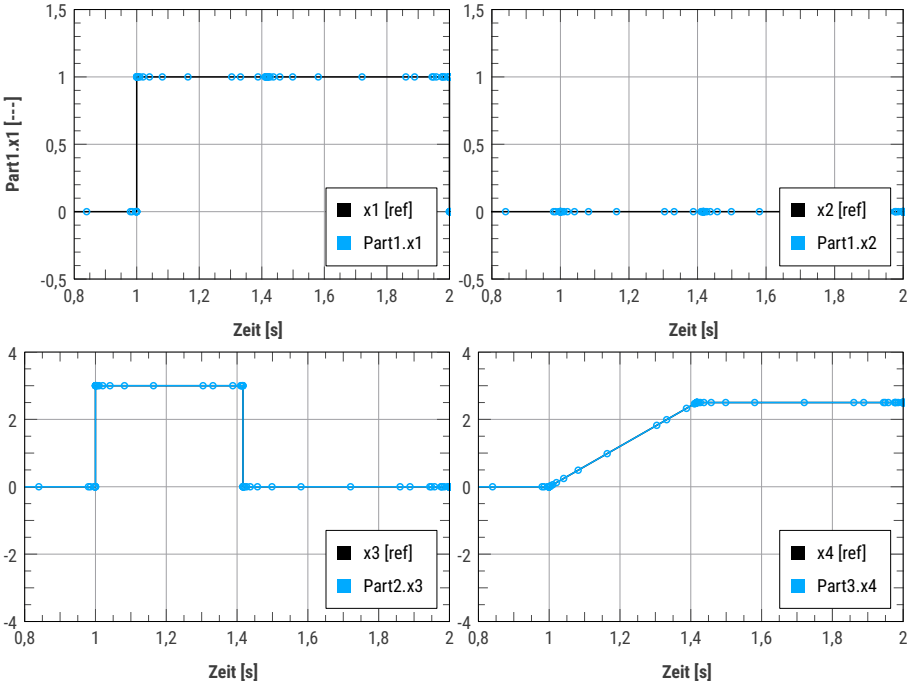

Abbildung 18: GAUSS-SEIDEL, iterierend, max 2. Iterationen, variable Schrittweite, Doppel-Schritt/Richardson-Fehlerschätzer

Nach den Sprung-/Unstetigkeitsstellen wird der Kommunikationsschritt schnell wieder vergrößert (siehe auch Abb. 19). Die dabei auftretenden Zacken sind charakteristisch für dieses lineare Problem mit Unstetigkeitsstellen. Vor einer Unstetigkeit wird der Zeitschritt zunächst drastisch reduziert (Faktor 0,2). Der nächste Schritte kann unter Umständen noch vor Eintreten der Unstetigkeit enden, wobei der Fehlerschätzer einen Fehler von 0 liefert (lineares Problem!). Dadurch wird der nächste Schritt wieder um den maximal möglichen Faktor (hier 2) vergrößert. Das gleiche passiert nach dem Überwinden der Unstetigkeit. Der Fehlerschätzer liefert danach stets den Wert 0 und der Zeitschritt verdoppelt sich mit jedem Schritt. Durch das Verhältnis

von minimalem Abminderungsfaktor (0,2) und maximalem Vergrößerungsfaktor (2) kommen die charakteristischen Zacken zustande.

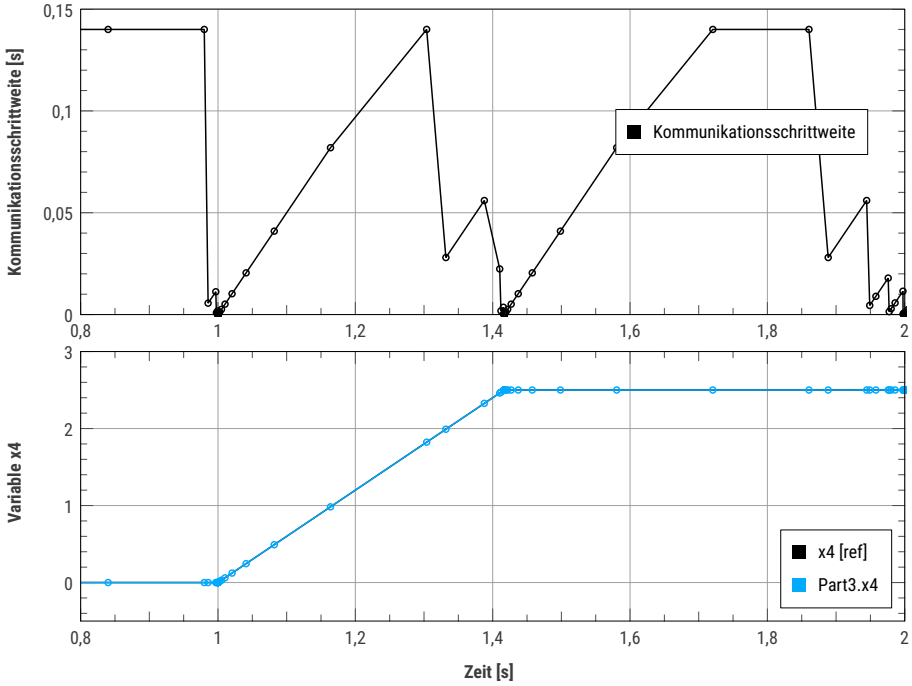

Abbildung 19: GAUSS-SEIDEL, iterierend, max 2. Iterationen, variable Schrittweite, Doppel-Schritt/Richardson-Fehlerschätzer

Der Algorithmus kann die exakte Lösung bis auf die zulässige Abweichung durch die unteren Zeitschrittschranken ($10^{-5}s$) abbilden, wobei die geforderten Toleranzen von max. 10^{-5} eingehalten werden, siehe Detail 20.

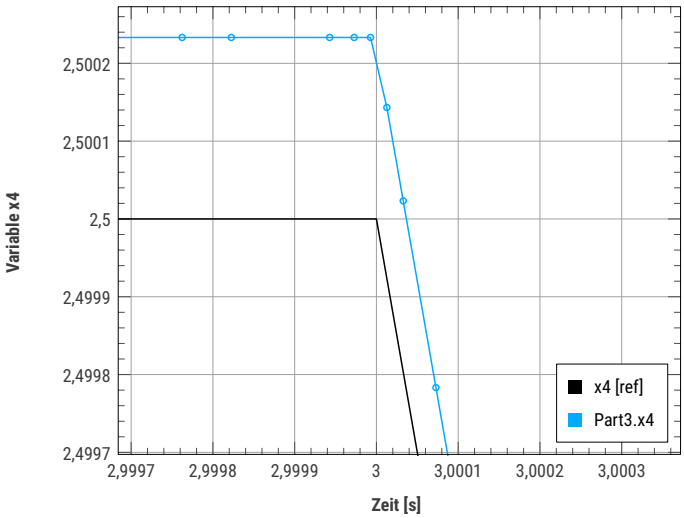

Abbildung 20: GAUSS-SEIDEL, iterierend, max 2. Iterationen, variable Schrittweite, Doppel-Schritt/Richardson-Fehlerschätzer

5.5.4 Genauigkeitsverbesserung und Geschwindigkeit

Der Algorithmus benötigt durch die teilweise stark reduzierten Kommunikationsschrittlängen sehr viel mehr Auswertungen als alle bisherigen Algorithmen. Durch das Doppel-Schritt-Verfahren werden zudem für jeden Kommunikationsschritt 3 statt einer Auswertung des Masteralgorithmus benötigt. Entsprechend viele Auswertungen benötigt der Algorithmus:

```
1  ------------------------------------------------------------------------------
2  Wall clock time                          =   10.390 ms
3  ------------------------------------------------------------------------------
4  Output writing                           =    8.959 ms
5  Master-Algorithm                         =    0.714 ms         326
6  Convergence failures                     =                      41
7  Error test time and failure count        =    0.240 ms          91
8  ------------------------------------------------------------------------------
9  Part1                              doStep =    0.119 ms        1244
10                                  getState =    0.065 ms        1128
11                                  setState =    0.012 ms         517
12 Part2                              doStep =    0.093 ms        1511
13                                  getState =    0.037 ms        1128
14                                  setState =    0.017 ms         784
15 Part3                              doStep =    0.097 ms        1511
16                                  getState =    0.038 ms        1128
17                                  setState =    0.018 ms         784
18 ------------------------------------------------------------------------------
```

Der Einfluss der heuristischen Parameter für die Zeitschrittsteuerung, vor allem die Grenzen für Reduktions- und Vergrößerungsfaktor haben einen großen Einfluss auf die benötigte Zahl der Aufrufe an den Masteralgorithmus. Bei linearen Problemen mit Unstetigkeiten empfehlen sich grundsätzlich größere Grenzen für diese Faktoren. Bei stärker nicht-linearen Problemen können kleinere Vergrößerungsfaktoren helfen, die Anzahl der fehlerbehafteten Kommunikationsschritte zu reduzieren.

6 Zusammenfassung

In diesem Bericht wurden einige grundlegenden Masteralgorithmen für die Co-Simulation beschrieben, welche im Co-Simulations-Master MASTERSIM implementiert sind. Anhand eines Testbeispiels wurden der Einfluss der verschiedenen Parameter auf die Ergebnisse beschrieben und die Konsequenzen für die Berechnungsergebnisse verdeutlicht. Grundsätzlich ist das GAUSS-SEIDEL-Verfahren zu bevorzugen, da es bei gleichen Kommunikationsschrittlängen im Vergleich zu GAUSS-JACOBI genauere Ergebnisse liefert. Dieses kann durch iterative Anwendung verbessert werden, wobei jedoch die Art der gekoppelten Gleichungen und die Anzahl der maximal zulässigen Iterationen einen großen Einfluss auf die Ergebnisse hat.

Allgemein lässt sich die Genauigkeit der gekoppelten Lösung durch Reduzierung der Zeitschritte verbessern, wobei der Simulationsaufwand entsprechend steigt. In dem hier gezeigten Testbeispiel war die Auswertung der FMUs nahezu trivial und dadurch sehr schnell. In realen Anwendungsfällen wird die Auswertungszeit der FMUs maßgeblich für die Gesamtperformance sein, welches die Reduktion der Kommunikationsschritte motiviert. Ohne Genauigkeit einzubüßen, kann dieses nur durch Zeitschrittadaption erfolgen, sodass bei Zustandsereignissen (zeitabhängig oder variablenabhängig) die Schritte verkleinert und andernfalls wieder vergrößert werden.

Die Zeitschrittadaption kann alleine auf dem Konvergenzverhalten beruhen. D.h. bei Überschreiten der maximalen Anzahl der Iterationen oder Divergenz können die Zeitschritte verkleinert werden und sonst wieder vergrößert werden. Dies funktioniert gut bei Modellen mit Zustandsereignissen, welche zu Unstetigkeiten führen. Allerdings sind Zeitereignisse bzw. rein zeitabhängige Unstetigkeiten so nicht zu erkennen.

Um die lokale und damit mittelbar globale Genauigkeit sicherzustellen, kann die Zeitschrittadaption auch auf einem Fehlerschätzer basieren. Im MASTERSIM wurde der Doppel-Schritt/Richardson-Extrapolations-Fehlerschätzer integriert, welcher neben einem normalen Kommunikationsschritt noch 2 Schritte mit halber Länge über dem gleichen Gesamtinterval ausführt. Aus den Unterschieden in den Ergebnissen können die Integrationsfehler im Gesamtinterval abgeschätzt werden. Dadurch lassen sich jedoch Zeitereignisse mit Unstetigkeiten in einzelnen Variablen nicht erfassen. Daher wurde der Fehlerschätzer durch einen gradientenbasierten Schätzer erweitert. Dabei werden die Anstiege im Gesamtintervall und im zweiten Halbinterval verglichen. Durch diese Erweiterung kann eine genaue Lösung im Rahmen gewählten Toleranz bestimmt werden.