HTBLuVA St.Pölten
Höhere Abteilung Elektrotechnik
3100 St. Pölten, Waldstrasse 3 Tel: 02742-75051-300
Homepage: http://et.htlstp.ac.at E-Mail: et@htlstp.ac.at

Projekt-Titel:

Simulation

Mitglieder:

Labenbacher Michael
Neulinger David
August Loibl
Eder Daniel

Projektort: HTBL u. VA in St. Pölten

Projektdatum: 25.11.2015

Projektnummer: 04

Projektgruppe: 1

Fach: Laboratorium

Jahrgang/Klasse: 2015/16 5AHET

Lehrer: Dipl.-Ing. Dr. Wilhelm Haager

Protokollführer:	***Unterschriften:***	***Note:***
Labenbacher Michael		

Inhaltsverzeichnis

1 Einleitung & Aufgabenstellung

Die Hauptaufgabe dieses Projektes besteht in der Untersuchung von verschiedenen Lösungsmethoden komplexer Systeme. Dabei sind diverse Programme zu entwickeln, welche ein gewisses Gleichungssystem möglichst exakt berechnen. Die Auswertung der Ergebnisse sind im Anschluss darauf gegenüber zu stellen und die Abweichungen sind zu diskutieren und begründen.

Es werden im Laufe des Projektes verschiedene, komplexe Systeme betrachtet, welche, über lange Zeit gesehen, eine hohe Empfindlichkeit gegenüber kleinen Änderungen aufweisen.
Ein praktisches Beispiel dafür ist die Erdatmosphäre (Wetter).

Eine Art und Weise ein System zu beschreiben, kann mit Hilfe von Funktionsblöcken erfolgen. Mit den zur Verfügung stehenden Programmen, wie z. B. Scilab (Simulationsaufsatz Xcos) oder Matlab (Zusatzpaket Simulink), kann ein Blockschaltbild angefertigt werden, welches das zu berechende Modell beschreibt. Dabei stellen die Integratoren, welche sich aus den Zustandsgleichungen ergeben, den Kern des Modells dar.

Eine zweite Möglichkeit ein System zu beschreiben kann mit dem expliziten *Euler-Verfahren* erfolgen. Dabei kann das Gleichungssystem des Systems aufgestellt und mit Hilfe diverser Berechnungsverfahren gelöst bzw. näherungsweise berechnet werden, indem die Näherung der Zustandsgleichungen mit Differenzengleichungen erolgt. Dafür kann mittels der Entwicklungsumgebung Eclipse mit Java bzw. C oder auch mit LaTeX ein Programm geschrieben werden, welches die Differenzengleichungen schrittweise, ausgehend von Startwerten, auswertet und im Anschluss darauf das System graphisch darstellt. Dabei werden in jedem Zeitschritt die Zuwächse der Zustandsgrößen innerhalb der Schrittweite Δt berechnet und zu den vorherigen Zustandsgrößen dazugezählt. Dadurch entsteht ein näherungsweiser Verlauf durch die Tangenten am Beginn jedes Zeitschrittes. (Bei der Programmierung ist auch darauf zu Achten, dass die Schrittweite nicht zu klein gewählt wird, da sich bei kleiner werdenden Schrittweiten die Rundungsfehler wieder stärker bemerkbar machen.)

Heutzutage findet zur näherungsweisen Lösung von Anfangswertproblemen in der numerischen Mathematik beispielsweise das bekannte Runge-Kutta-Verfahren Anwendung, dessen Ansatz ebenfalls darin besteht Differentialquotienten durch Differenzenquotienten zu ersetzen. Mittels dem Programm Maxima bzw. Matlab ist es somit relativ einfach möglich ein solches System näherungsweise, mit einer höheren Genauigkeit bei größeren Schrittweiten als das Euler-Verfahren, zu berechnen. (Man könnte natürlich auch das Runge-Kutta-Verfahren in Java bzw. C programmieren, jedoch soll durch dieses Projekt gezeigt werden, welche Unterschiede bei verschiedenen Simulationsmethoden, Berechnungsverfahren etc. entstehen.)
Dieses Verfahren berechnet den Zuwachs während eines Zeitschrittes nicht nur aus der Anfangstangente jedes Zeitschrittes, sondern durch eine bestimmte Mittelung aus mehreren Tangenten innerhalb des Zeitschrittes.

In diesem Projekt wird nun versucht, verschiedene Lösungsmethoden für einige bekannte dynamische Systeme aufzuzeigen und gegenüberzustellen. Des Weiteren werden einige Systeme, wie der Lorenz-Attraktor, genauer unter die Lupe genommen und es wird versucht einige Eigenschaften derer zu analysieren und zu begründen.

2 Verwendete Programme

Programm	Kurzbeschreibung/Version
Maxima	Platformunabhängiges Computeralgebrasystem, welches als Open-Source-Projekt entwickelt wurde. Version: 5.37.2-Windows
Java	Objektorientierte Programmiersprache, welche grundsätzlich aus dem Java-Entwicklungswerkzeug (JDK) zum Erstellen von Java--Programmen und der Java-Laufzeitumgebung (JRE) zu deren Ausführung besteht. Als quelloffenes Programmierwerkzeug wurde die platformunabhängige Entwicklungsumgebung Eclipse verwendet. Version: Eclipse Luna (für Java™ 8)
C	Imperative, prozedurale Programmiersprache, wobei wieder als Programmierwerkzeug Eclipse verwendet wird.
Scilab (Xcos)	Leistungsfähiges, freies Software-Paket für Anwendungen aus der numerischen Mathematik. Xcos stellt dabei einen Simulationsaufsatz, zur graphischen Modellierung und Simulation dynamischer Systeme, dar. Version: 5.5.2
Matlab (Simulink)	Kommerzielle Software von MathWorks zur Lösung von mathematischen Problemen und zur graphischen Darstellung des Ergebnisses. Simulink stellt dabei ein Zusatzpaket, das zur Modelierung von Systemen dient, dar. Version: 8.5 (R2015a)
LaTeX	Softwarepaket zur Vereinfachung der Nutzung des Textsatzsystemes TeX. Als TeX-Distribution wird MikTeX 2.9 verwendet. Version: 2_ϵ

Tabelle 2.1: Verwendete Programme

3 Verzögerungsglied 2. Ordnung $\mathbf{PT_2}$

3.1 Grundlagen

Als Einstieg in die Thematik *Simulation* wurde ein PT_2-Element in Scilab (Xcos) mit Funktionsblöcken und in Maxima mit dem Runge-Kutta-Verfahren simuliert. Ausgangspunkt sind die nachfolgenden Differentialgleichungen, welche das PT_2-Element in der Abb. 3.1, dessen Parameterwahl von der Laborübung 03 übernommen worden ist, beschreiben.

Herleitung:

$$z = \sigma - x \qquad \& \qquad y = \frac{1}{T_{\text{I}}} \int_0^t z \, dt \qquad \Rightarrow \quad \dot{y} = \frac{1}{T_{\text{I}}} (\sigma - x)$$

$$x + T_{\text{PT1}} \, \dot{x} = k_{\text{PT1}} \, y \qquad \Rightarrow \quad \dot{x} = \frac{k_{\text{PT1}}}{T_{\text{PT1}}} (y - x)$$

Wird nun eine Stationärverstärkung k_{PT1} von 1 gewählt und als Eingangsgröße ein Sprung, so ergeben sich folgende nichtlineare, gewöhnliche Differentialgleichungen. (Eine gewöhnliche Differentialgleichung (engl. ordinary differential equation (ODE)) ist eine Differentialgleichung, bei der zu einer gesuchten Funktion nur Ableitungen nach genau einer Variablen auftreten.):

$$\begin{aligned} \dot{x} &= \frac{1}{T_{\text{PT1}}} (y - x) \\ \dot{y} &= \frac{1}{T_{\text{I}}} (1 - x) \end{aligned} \tag{3.1}$$

Die in der Laborübung 03 gewählte Parameterwahl beträgt:

$$T_{\text{I}} = 0,1 \qquad T_{\text{PT1}} = 0,1 \tag{3.2}$$

Das Blockschaltbild eines PT_2-Elementes kann sich aus einem Integrator und einem PT_1-Element, wie die nachfolgende Abbildung zeigt, zusammensetzen:

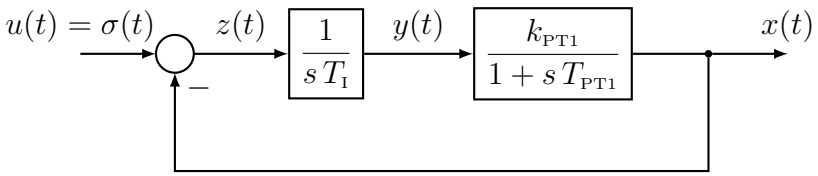


Abbildung 3.1: Blockschaltbild eines PT_2-Elementes

Die gewählte Parameterkonstellation 3.2 führt, in der Laborübung 03 berechnet, auf folgende Kenngrößen eines PT_2-Elementes:

k_PT2	D	ω_n	τ_PT2	ω_0	T_0	$T_\mathrm{ü}$	$\ddot{u}$
[—]	[—]	[sec^{-1}]	[sec]	[sec^{-1}]	[sec]	[sec]	[%]
1	$0,5$	10	0.2	$8,66$	0.73	$0,36$	$16,30$

Tabelle 3.1: Parametergrößen und Eigenschaften des PT_2-Elementes

Dabei bedeutet eine Dämpfung von $0,5$, dass ein oszillatorischer Fall vorliegt und es zu einem Überschwingen von $16,30\,\%$ kommt, sprich der Maximalwert, bei einer Stationärverstärkung von 1, x_max hier $1,163$ beträgt.

3.2 Simulation mit dem Runge-Kutta-Verfahren

3.2.1 Maxima

Mit dem Computeralgebrasystem Maxima lässt sich ein System mit m gewöhnlichen Differentialgleichungen 1. Ordnung mit folgendem Befehl nach dem Runge-Kutta-Verfahren 4. Ordnung lösen:

$$rk(\underbrace{[\mathrm{ODE}_1 \ldots \mathrm{ODE}_\mathrm{m}]}_{\text{functions}}, \underbrace{[v_1 \ldots v_\mathrm{m}]}_{\text{states}}, \underbrace{[\mathrm{init}_1 \ldots \mathrm{init}_\mathrm{m}]}_{\text{initialisation}}, \underbrace{[t,\ t_\mathrm{a},\ t_\mathrm{e},\ \Delta t]}_{\text{domain}});$$

Rückgabewert dieses Befehls ist eine geschachtelte Liste mit einer Unterliste für jeden Rechenschritt, welche aus der unabhängigen Variable und den Zustandsgrößen besteht. Folgendes Programm wurde für das PT_2-Element erstellt:

```
(%i1)   kill(all)$
```

Laden des *dynamics*- und *coma*(*draw*)-Paketes und setzen von Defaultwerten:

```
(%i3)   load(coma)$
        load(dynamics)$set_draw_defaults(
            grid=true,point_type=0,points_joined=true)$
```

coma v.1.73, (Wilhelm Haager, 2015-01-09)

Funktionen:

```
(%i5)   f1 : 1/T_PT1*(y-x)$
        f2 : 1/T_I*(1-x)$
```

Zustandsgrößen:

```
(%i6)   states:[x,y]$
```

Konstanten:

```
(%i7)   constants:[T_I=0.1,T_PT1=0.1]$
```

Anfangsbedingungen:

```
(%i8)   initialisation:[0,0]$
```

Domäne (abhängige Variable t (Zeit), Simulationsbeginn, Simulationsende, Schrittweite):

```
(%i9)   domain:[t,0,1,10e-7]$
```

Berechnung und Zerlegung:
rk (functions, states, initialisation, domain)

```
(%i10)  res:rk([ev(f1,constants),ev(f2,constants)],
               states,initialisation,domain)$
(%i13)  t_werte_maxima:map(first,res)$
        x_werte_maxima:map(second,res)$
        y_werte_maxima:map(third,res)$
```

Zeitverlauf:

```
(%i14)  wxplot_size:[800,400]$
(%i15)  wxdraw2d(color=red,xlabel="t",ylabel="x(rot) / y(blau)",
                 points(t_werte_maxima,x_werte_maxima),
                 color=navy,yrange=[0,1.5],xrange=[0,domain[3]],
                 points(t_werte_maxima,y_werte_maxima),ytics=0.1,
                 xaxis=true,dimensions=[1500,600],xtics=0.1)$
```

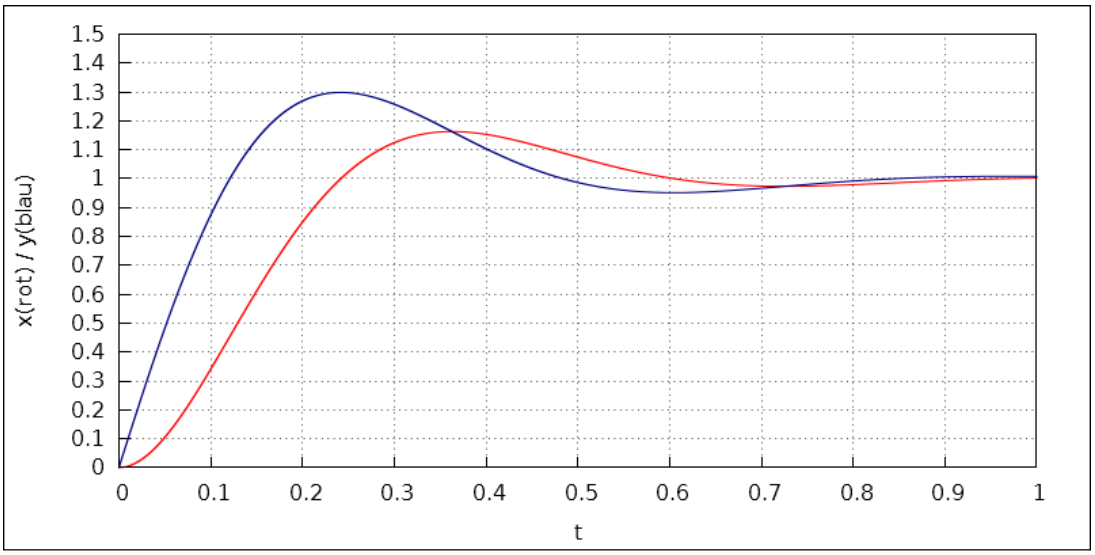


Abbildung 3.2: Zeitverläufe der Zustandsgrößen x & y beim PT_2-Element (Maxima)

(%t15)

3.3 Simulation mit Funktionsblöcken

3.3.1 Scilab (Xcos)

Aus den Zustandsgleichungen 3.1 lässt sich das nachfolgende Blockschaltbild mit Hilfe von Scilab und dem Zusatzpaket Xcos entwickeln.

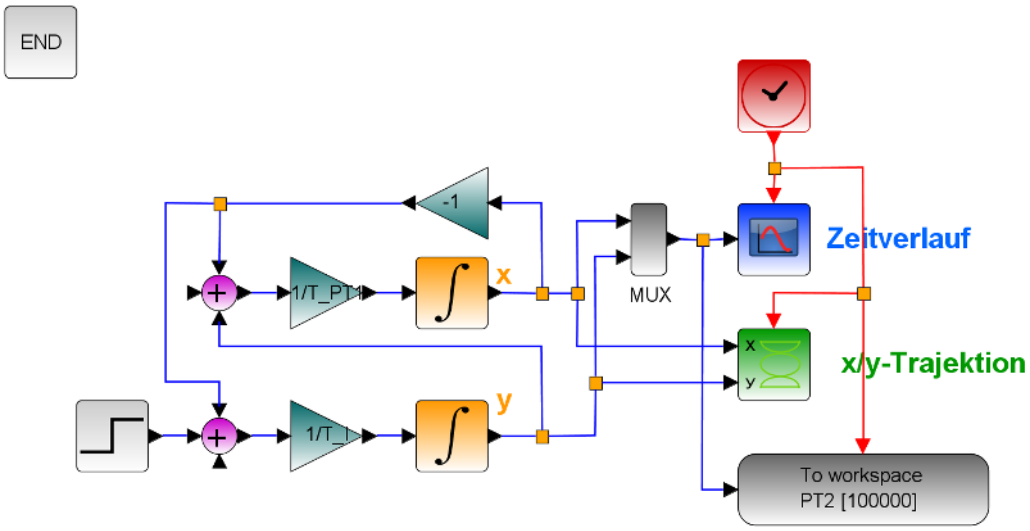


Abbildung 3.3: Blockschaltbild des PT_2-Elementes (Scilab (Xcos))

Dieses Programm wurde so entwickelt, dass der Zeitverlauf angefertigt wird. Für die Simulation müssen noch in Scilab die Parameter des PT_2-Gliedes eingegeben werden,

```
T_PT1=0.1;
T_I=0.1;
```

und die Parameter der einzelnen Blöcke, wie die Startwerte, Grenzwerte, Simulationszeit, etc. sind festzulegen. Mit folgender Eingabe in Scilab können die fertig simulierten Werte in eine csv-Datei geschrieben werden:

```
write_csv(PT2.time,'PT2_xcos_time.csv')
write_csv(PT2.values,'PT2_xcos_values.csv')
```

Als Gleichungslöser wurde *suite of nonlinear and differential/algebraic equation solvers (Sundials)/CVODE - backward differentiation formulas (BDF) - NEWTON* verwendet, wobei 100000 Punkte im Zeitbereich von $0 - 1$ berechnet wurden.

Parameter	Einstellung
Finale Integrationszeit	$1.0E03$
Echt-Zeit-Skalierung	$0,0E00$
Absolute Toleranz des Integrators	$1,0E-10$
Relative Toleranz des Integrators	$1,0E-10$
Zeit-Toleranz	$1,0E-12$
Maximales Zeitintervall der Integration	$1,00001E05$
Gleichungslöser	Sundials/CVODE - BDF - NEWTON
Maximale Schrittweite (0 = kein Limit)	$0,0E00$

Tabelle 3.2: Parametereinstellungen in Scilab für das PT_2-Element (Xcos)

Das Simulationsergebnis ist:

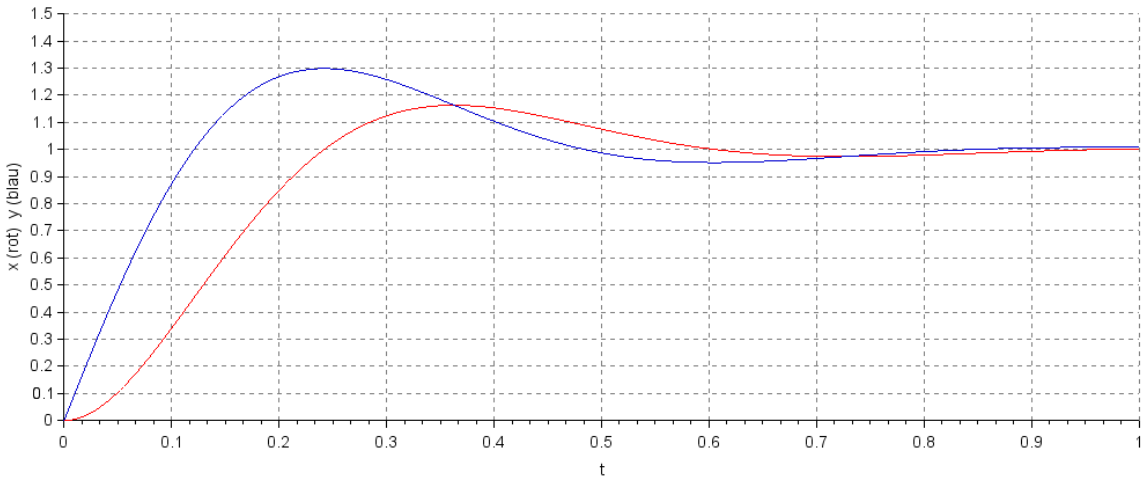


Abbildung 3.4: Zeitverläufe der Zustandsgrößen x & y beim PT_2-Element (Scilab (Xcos))

Der rote Verlauf zeigt den typischen PT_2-Verlauf der Sprungantwort mit einem maximalen Überschwingen von $16,30\,\%$ auf, dies kann mit

```
k_max=max(PT2.values,"row");
ue=(k_max(1)-1)/1*100;
```

in Scilab nach der Simulation ermittelt werden.

3.4 Gegenüberstellung

```
(%i19)  t_werte_xcos:read_list("C:\\Users\\User\\Desktop
        \\Schule\\Laboratorium-5AHET\\04_Simulation
        \\00_PT2\\PT2_xcos_time.csv")$
        werte_xcos:read_nested_list("C:\\Users\\User\\Desktop
        \\Schule\\Laboratorium-5AHET\\04_Simulation
        \\00_PT2\\PT2_xcos_values.csv",comma)$
        x_werte_xcos:map(first,werte_xcos)$
        y_werte_xcos:map(second,werte_xcos)$
(%i20)  wxdraw2d(key="Maxima",color=red,xlabel="t",ytics=0.1,
                 points(t_werte_maxima,x_werte_maxima),
                 key="Xcos",color=black,ylabel="x",xtics=0.1,
                 points(t_werte_xcos,x_werte_xcos),
                 yrange=[0,1.5],xrange=[0,domain[3]],
                 user_preamble="set key bottom left",
                 xaxis=true,dimensions=[1500,600])$
```

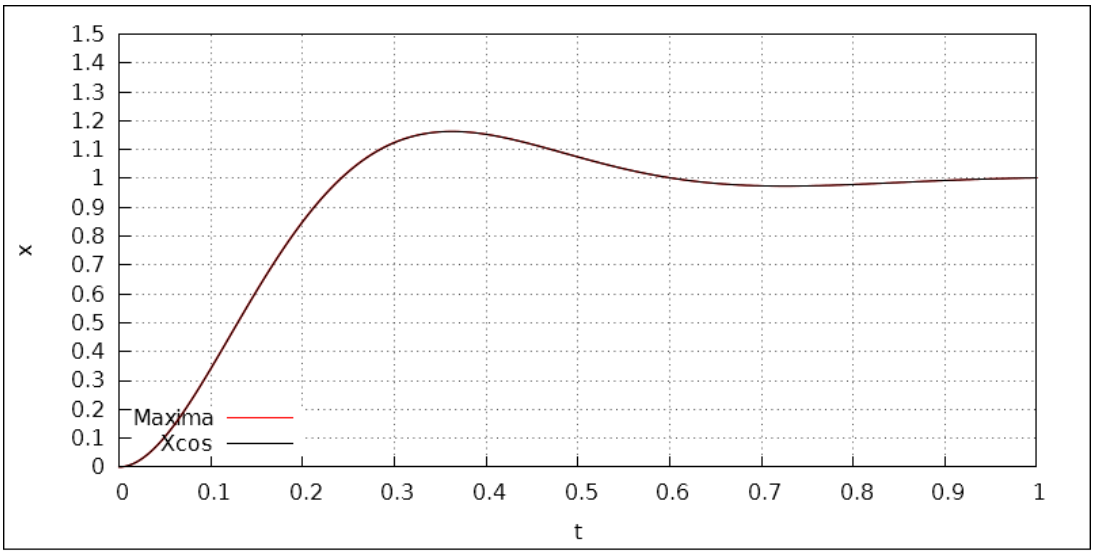


Abbildung 3.5: Zeitverlauf der Zustandsgröße x im Vergleich beim PT_2-Element (-Scilab (Xcos), Maxima)

(%t20)

Maxima erreicht, sowie auch Xcos ein maximales Überschwingen von $16,30\,\%$, was mit dem nachfolgenden Befehl überprüft werden kann:

```
(%i23)  k_max:0$
        for i:1 while i<(length(x_werte_maxima)) do
            if k_max<x_werte_maxima[i] then k_max:x_werte_maxima[i]$
        ue:(k_max-1)/1*100;
(%o23)  16.30

(%i24)  wxdraw2d(key="Maxima",color=red,ytics=0.1,xtics=0.1,
                 points(t_werte_maxima,y_werte_maxima),
                 key="Xcos",color=black,dimensions=[1500,600],
                 points(t_werte_xcos,y_werte_xcos),xaxis=true,
                 yrange=[0,1.5],xrange=[0,domain[3]],ylabel="y",
                 user_preamble="set key bottom left",xlabel="t")$
```

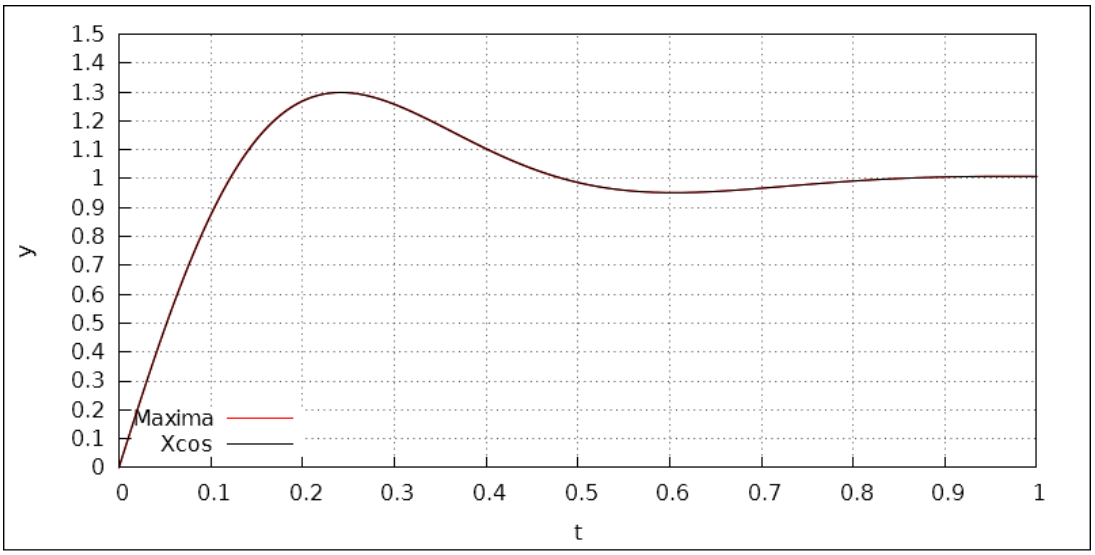


Abbildung 3.6: Zeitverlauf der Zustandsgröße y im Vergleich beim PT_2-Element (- Scilab (Xcos), Maxima)

(%t24)

Die Gegenüberstellung der Simulationsverfahren zeigt im Prinzip keine optischen Differenzen und es lässt sich sagen, dass die Genauigkeit beider Verfahren in diesem Falle ausreichend ist.

4 Lorenz-Attraktor

4.1 Grundlagen

Der Lorenz-Attraktor ist die chaotische Systembewegung des sogenannten Lorenz-Systems, ein System aus drei nichtlinearen, gewöhnlichen Differentialgleichungen, die ein stark vereinfachtes Modell des Wettergeschehens darstellen.
Des Weiteren zählt der Lorenz-Attraktor zu den sogenannten seltsamen Attraktoren.

Seltsamer Attraktor:

Dies ist ein Attraktor, sprich ein Ort im Zustandsraum, der den Endzustand eines dynamischen Prozesses darstellt und folgende Bedingungen erfüllt:

- chaotisches Verhalten: Der Attraktor zeigt eine sensitive Abhängigkeit von den Anfangsbedingungen, das bedeutet, dass beliebig kleine Änderungen des Anfangszustandes zu völlig unterschiedlichen Verläufen führen können.

- fraktale Struktur: Der Attraktor besitzt eine nicht-ganzzahlige Dimension.

- keine Aufteilungsmöglichkeit: Der Attraktor kann nicht in zwei oder mehr Teilen aufgespalten werden.

, und alle Trajektorien müssen innerhalb einer Region, dem Zustandsraum, bleiben.

Chaotisches System:

Dies ist ein System, welches eine sensitive Abhängigkeit von den Anfangsbedingungen besitzt, sich aber trotzdem nur in einem beschränkten Bereich bewegt.

Zustandsraum:

Der Zustandsraum setzt sich aus der Menge aller möglichen Zustände, die ein dynamisches System einnehmen kann, zusammen und beliebige dynamische Systeme, welche durch Differentialgleichungen beschrieben werden können, lassen sich im Zustandsraum beschreiben.

Die drei Gleichungen,

$$\dot{x} = \alpha\,(y - x)$$

$$\dot{y} = x\;(\beta - z) - y \tag{4.1}$$

$$\dot{z} = x\,y - cz$$

$x,\ y,\ z\ \ldots$	Zustandsgrößen
$\alpha\ \ldots\ldots\ldots$	Systemparameter (Prandtl-Zahl (z. B. Maß für die Trägheit))
$\beta\ \ldots\ldots\ldots$	Systemparameter (Rayleigh-Zahl (z. B. Maß für Zellengeometrie))
$c\ \ldots\ldots\ldots$	Systemparameter
$t\ \ldots\ldots\ldots$	Zeit

welche den Lorenz-Attraktor beschreiben, stellen eines der einfachsten Systeme mit, bei bestimmter Wahl der Systemparameter, chaotischem Verhalten dar. Es ist ein vereinfachtest Modell des Wettergeschehens, welches von Edward N. Lorenz formuliert wurde und oszillierende, nichtperiodische Bewegungen vollführt.

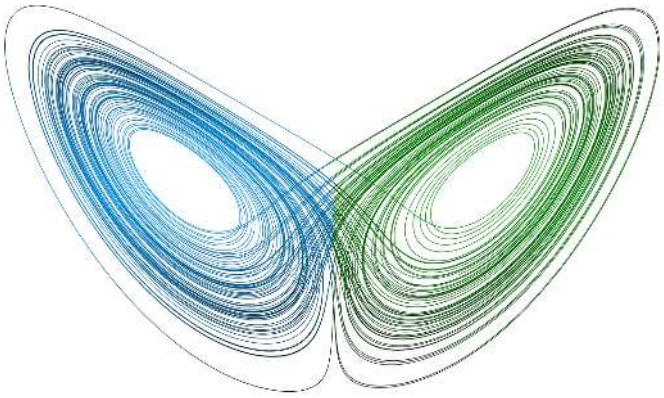


Abbildung 4.1: x/z-Trajektion eines Lorenz-Attraktors simuliert mit dem Dormand-Prince-Verfahren in Matlab

Die typische Parameterwahl, welche zu einer chaotischen Lösung führt ist:

$$\alpha = 10 \qquad \beta = 28 \qquad c = \frac{8}{3} \tag{4.2}$$

Als Anfangswerte werden fast ausschließlich im gesamten Projekt, zum Vergleich der verschiedenen Simulationsverfahren, folgende Werte verwendet:

$$x(0) = 7 \qquad y(0) = 12 \qquad z(0) = 17 \tag{4.3}$$

Symmetrie:

Aus den Formeln 4.1 lässt sich erkennen, dass die Trajektorie durch den Punkt $P\left(x_\mathrm{i}, y_\mathrm{i}, z_\mathrm{i}\right)$ den gleichen Verlauf, wie die durch den Punkt $P\left(-x_\mathrm{i}, -y_\mathrm{i}, z_\mathrm{i}\right)$ nimmt, jedoch an der z-Achse gespiegelt ist.

$$\begin{aligned} P\left(x_\mathrm{i}, y_\mathrm{i}, z_\mathrm{i}\right): \quad & \dot{x}_\mathrm{i} = \alpha\left(y_\mathrm{i} - x_\mathrm{i}\right) \\ & \dot{y}_\mathrm{i} = x_\mathrm{i}\left(\beta - z_\mathrm{i}\right) - y_\mathrm{i} \\ & \dot{z}_\mathrm{i} = x_\mathrm{i}\, y_\mathrm{i} - c z_\mathrm{i} \end{aligned} \qquad\qquad \begin{aligned} P\left(-x_\mathrm{i}, -y_\mathrm{i}, z_\mathrm{i}\right): \quad & \dot{x}_\mathrm{i} = \alpha\left(-y_\mathrm{i} + x_\mathrm{i}\right) \\ & \dot{y}_\mathrm{i} = -x_\mathrm{i}\left(\beta - z_\mathrm{i}\right) + y_\mathrm{i} \\ & \dot{z}_\mathrm{i} = x_\mathrm{i}\, y_\mathrm{i} - c z_\mathrm{i} \end{aligned}$$

Es ändert sich somit nur das Vorzeichen in den ersten beiden Gleichungen und in der dritten bleibt dies unverändert, was somit einer Symmetrie bezüglich der z-Achse, unabhängig von der Parameterwahl, entspricht.

Gleichgewichtspunkte:

Ein Gleichgewichtspunkt ist jener Punkt, bei dem $\dot{x} = \dot{y} = \dot{z} = 0$ gilt, sprich die rechten Seiten der Differentialgleichungen müssen 0 sein. Dies tritt bei den folgenden Punkten beim Lorenz-Attraktor auf:

$$\begin{aligned} &1.)\ 0 = \alpha\left(y - x\right) && 2.)\ \Rightarrow && 0 = x\left(x^2 - \left[c\left(\beta - 1\right)\right]\right) \\ &\quad 0 = x\left(\beta - z\right) - y && 3.)\ \Rightarrow && x_0 = 0 \quad \textbf{v.} \quad x_{1,2} = \pm\sqrt{c\left(\beta - 1\right)} \\ &\quad 0 = x\, y - c z && 4.)\ \Rightarrow && y_0 = 0 \quad \textbf{v.} \quad y_{1,2} = \pm\sqrt{c\left(\beta - 1\right)} \\ & && 5.)\ \Rightarrow && z_0 = 0 \quad \textbf{v.} \quad z_{1,2} = \left(\beta - 1\right) \end{aligned}$$

$$\begin{aligned} & P_0(0,0,0) \\ & P_1\Big(\sqrt{c\left(\beta - 1\right)}, \sqrt{c\left(\beta - 1\right)}, \left(\beta - 1\right)\Big) \\ & P_2\Big(-\sqrt{c\left(\beta - 1\right)}, -\sqrt{c\left(\beta - 1\right)}, \left(\beta - 1\right)\Big) \end{aligned} \tag{4.4}$$

Gleichgewichtspunkte laut der Parameterwahl 4.2: $P_0(0,0,0),\ P_{1,2}\Big(\pm 6\sqrt{2}, \pm 6\sqrt{2}, 27\Big)$

4.2 Simulation mit Differenzengleichungen

4.2.1 Java

Mit Hilfe der Entwicklungsumgebung Eclipse Luna wurde mit der Hochsprache Java ein Programm geschrieben, welches die Differenzengleichungen schrittweise auswertet und die Berechnung in eine txt-Datei abspeichert. Des Weiteren werden die einzelnen Diagramme graphisch dargestellt, wobei die Bibliothek JFreeChart benutzt wurde.

```
1  package simulationen;
2  /*  Import der Bibliotheken */
3  import java.awt.Color;
4  import java.awt.BasicStroke;
5  import java.io.IOException;
6  import org.jfree.chart.ChartPanel;
7  import org.jfree.chart.JFreeChart;
8  import org.jfree.data.xy.XYDataset;
9  import org.jfree.data.xy.XYSeries;
10 import org.jfree.ui.ApplicationFrame;
11 import org.jfree.ui.RefineryUtilities;
12 import org.jfree.chart.ChartFactory;
13 import org.jfree.chart.axis.NumberAxis;
14 import org.jfree.chart.axis.NumberTickUnit;
15 import org.jfree.chart.plot.PlotOrientation;
16 import org.jfree.chart.plot.XYPlot;
17 import org.jfree.data.xy.XYSeriesCollection;
18 import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
```

Abbildung 4.2: Java-Code (Imports) zur Simulation des Lorenz-Attraktors (Java)

```
19 /** Simulation des Lorenz-Attraktors */
20 @SuppressWarnings("serial")
21 public class Lorenz_Attraktor extends ApplicationFrame {
```

```
22    /*  Konstruktor */
23    public Lorenz_Attraktor (String applicationTitle,
24        String chartTitle,String xname,String yname,String zname,
25        double [] x, double [] y, double [] z,
26        double [] t,int anz,int choose,char fx,char fy,char fz,
27        int xmin, int xmax, int ymin, int ymax, int zmin, int zmax)
28        throws  IOException
29    { super(applicationTitle);
30      switch(choose){
31          case 1:
32            JFreeChart xylineChart =
33              ChartFactory.createXYLineChart(
34                chartTitle, xname, yname,
35                createDataset_time(x, y, z, t, anz),
36                PlotOrientation.VERTICAL, true, true, false);
37              ChartPanel chartPanel = new ChartPanel(xylineChart);
38              chartPanel.setPreferredSize
39                  (new java.awt.Dimension(800,400));
40              //  The Plot:
41              final XYPlot xyplot = xylineChart.getXYPlot( );
42              //  x-Achse:
43              NumberAxis domain = (NumberAxis)
44                  xyplot.getDomainAxis();
45              domain.setRange(xmin, xmax);
46              domain.setTickUnit(new NumberTickUnit(10));
47              domain.setVerticalTickLabels(true);
48              //  y-Achse
49              NumberAxis range =
50                  (NumberAxis) xyplot.getRangeAxis();
51              range.setRange(ymin, ymax);
52              range.setTickUnit(new NumberTickUnit(10));
53              //  Verlaeufe
54              XYLineAndShapeRenderer renderer =
55                  new XYLineAndShapeRenderer();
56              xyplot.getRenderer().setSeriesPaint(0,Color.RED);
57              renderer.setSeriesStroke(0,new BasicStroke(1.0f));
58              xyplot.getRenderer().setSeriesPaint
59                  (1,new Color(0,0,128));
60              renderer.setSeriesStroke(1,new BasicStroke(1.0f));
61              xyplot.getRenderer().setSeriesPaint
62                  (2,new Color(34,139,34));
63              renderer.setSeriesStroke(2,new BasicStroke(1.0f));
```

```
64              //  Hintergrundfarbe
65              xylineChart.setBackgroundPaint(Color.WHITE);
66              xyplot.setBackgroundPaint(Color.WHITE);
67              setContentPane(chartPanel);
68              break;
69          case 2:
70           JFreeChart xylineChart2 =
71              ChartFactory.createXYLineChart(
72              chartTitle, xname, yname,
73              createDataset_traj(x, y, z, anz, fx, fy),
74              PlotOrientation.VERTICAL, false, true, false);
75            ChartPanel chartPanel2 = new ChartPanel(xylineChart2);
76            chartPanel2.setPreferredSize
77                (new java.awt.Dimension(600,600));
78            //  The Plot:
79            final XYPlot xyplot2 = xylineChart2.getXYPlot();
80            //  x-Achse:
81            NumberAxis domain2 = (NumberAxis)
82                xyplot2.getDomainAxis();
83            domain2.setRange(xmin, xmax);
84            domain2.setTickUnit(new NumberTickUnit(10));
85            domain2.setVerticalTickLabels(true);
86            //  y-Achse
87            NumberAxis range2 =
88               (NumberAxis) xyplot2.getRangeAxis();
89            range2.setRange(ymin, ymax);
90            range2.setTickUnit(new NumberTickUnit(10));
91            //  Verlaeufe
92            XYLineAndShapeRenderer renderer2 =
93                new XYLineAndShapeRenderer();
94            xyplot2.getRenderer().setSeriesPaint
95                (0,new Color(255,165,0));
96            renderer2.setSeriesStroke(0,new BasicStroke(1.0f));
97            //  Hintergrundfarbe
98            xylineChart2.setBackgroundPaint(Color.WHITE);
99            xyplot2.setBackgroundPaint(Color.WHITE);
100           setContentPane(chartPanel2);
101           break;
102         }
103   }
```

```
104   /*  Methoden  */
105    private XYDataset createDataset_time(double [] x, double [] y,
106           double [] z, double [] t, int anz) throws  IOException
107    {
108         final XYSeries txline = new XYSeries("x", true);
109         final XYSeries tyline = new XYSeries("y", true);
110         final XYSeries tzline = new XYSeries("z", true);
111         for(int i=0;i<Simulation.qet_quantity();i+=anz)
112         {
113           txline.add(t[i],x[i]);
114           tyline.add(t[i],y[i]);
115           tzline.add(t[i],z[i]);
116         }
117         final XYSeriesCollection dataset =
118             new XYSeriesCollection();
119         dataset.addSeries(txline);
120         dataset.addSeries(tyline);
121         dataset.addSeries(tzline);
122         return dataset;
123    }
124    private XYDataset createDataset_traj(double [] x, double [] y,
125     double [] z, int anz, char fx, char fy)
126     throws IOException{
127         final XYSeries line = new XYSeries("", false);
128         double [] f1 = new double[Simulation.qet_quantity()+2];
129         double [] f2 = new double[Simulation.qet_quantity()+2];
130         switch(fx){
131             case 'y': f1=y;break; case 'z': f1=z;break;
132             case 'x': f1=x;break;
133         }
134         switch(fy){
135           case 'y': f2=y;break; case 'z': f2=z;break;
136           case 'x': f2=x;break;
137         }
138         for(int i=0;i<Simulation.qet_quantity();i+=anz)
139         {line.add(f1[i],f2[i]);}
140         final XYSeriesCollection dataset =
141             new XYSeriesCollection();
142         dataset.addSeries(line);
143         return dataset;
144   }
```

```
145   /*  Main-Methode  */
146   public static void main(String[] args) throws IOException {
147     /*  Berechnung: */
148     Simulation Sim_1 =
149         new Simulation(10,28,8/3,7,12,17,0,50,0.00001);
150     Sim_1.calculation();
151     Sim_1.write_to_txt("Lorenz_Attraktor_java_werte.txt", 100);
152       double [] x=Simulation.get_x_werte();
153       double [] y=Simulation.get_y_werte();
154       double [] z=Simulation.get_z_werte();
155       double [] t=Simulation.get_t_werte();
156     /*  Graphische Darstellung  */
157     //  Zeitverlaeufe
158     Lorenz_Attraktor chart1 = new Lorenz_Attraktor
159         ("Zeitverläufe", "Zeitverläufe",
160          "t", "x/y/z", "", x, y, z, t, 100, 1, 'X', 'X', 'X',
161          0, 50, -25, 50, 0, 0);
162     chart1.pack();RefineryUtilities.centerFrameOnScreen(chart1);
163     chart1.setVisible(true);
164     //  Trajektorien
165     Lorenz_Attraktor chart2 = new Lorenz_Attraktor
166         ("Trajektorie", "Trajektorie in der x/z-Phasenebene",
167          "x", "z", "", x, y, z, t, 100, 2, 'x', 'z', 'X',
168          -25, 25, 0, 50, 0, 0);
169     chart2.pack();RefineryUtilities.centerFrameOnScreen(chart2);
170     chart2.setVisible(true);
171     Lorenz_Attraktor chart3 = new Lorenz_Attraktor
172         ("Trajektorie", "Trajektorie in der x/y-Phasenebene",
173          "x", "y", "", x, y, z, t, 100, 2, 'x', 'y', 'X',
174          -25, 25, -25, 25, 0, 0);
175     chart3.pack();RefineryUtilities.centerFrameOnScreen(chart3);
176     chart3.setVisible(true);
177     Lorenz_Attraktor chart4 = new Lorenz_Attraktor
178         ("Trajektorie", "Trajektorie in der y/z-Phasenebene",
179          "y", "z", "", x, y, z, t, 100, 2, 'y', 'z', 'X',
180          -25, 25, 0, 50, 0, 0);
181     chart4.pack();RefineryUtilities.centerFrameOnScreen(chart4);
182     chart4.setVisible(true);
183   }
184 }
```

Abbildung 4.3: Java-Code zur Simulation des Lorenz-Attraktors (Java)

Für die Berechnung und die Ausgabe in eine txt-Datei, wurde die Bibliothek BufferedWriter & FileWriter verwendet. Alle berechneten Werte x, y, z & t werden in einem Array an die Klasse Lorenz_Attraktor zur Darstellung zurückgegeben und in die txt-Datei geschrieben.

```
1  package simulationen;
2  /*  Import der Bibliotheken */
3  import java.io.BufferedWriter;
4  import java.io.FileWriter;
5  import java.io.IOException;
```

Abbildung 4.4: Java-Code (Imports) zur Berechnung des Lorenz-Attraktors (Java)

```
6  /**Die  Klasse "Simulation" im Projekt 01_Lorenz_Attraktor
7   * _in_Java berechnet den Lorenz-Attraktor und gibt
8   * die x- ,y- ,z- und t-Werte in einem Array oder
9   * in einer txt-Datei zurück bzw. aus.*/
10 public class Simulation {
11
12   /*  Variablen der Klasse Simulation */
13   private double  a,b,c,x0,y0,z0,t_begin,t_end,dt;
14   public static int quantity;
15   private static double [] t;
16   private static double [] x;
17   private static double [] y;
18   private static double [] z;
19
20   /**Konstruktor (verkürzt) */
21   public Simulation(
22       double a_constant, double b_constant, double c_constant,
23       double x_init, double y_init, double z_init)
24   {
25     this(a_constant,b_constant,c_constant,
26         x_init,y_init,z_init,0.0,50.0,0.00001);
27   }
```

```
28   /**Konstruktor (allgemein)
29    * Legt die Startwerte und Parameter der Simulation fest
30    * @param a          = Konstante a
31    * @param b          = Konstante b
32    * @param c          = Konstante c
33    * @param x_init     = Startwert x
34    * @param y_init     = Startwert y
35    * @param z_init     = Startwert z
36    * @param t_begin_sim     = Simulationsbegin
37    * @param t_end_sim       = Simulationsende
38    * @param increment_sim  = Schrittweite */
39   public Simulation(
40       double a_constant, double b_constant, double c_constant,
41       double x_init, double y_init, double z_init,
42       double t_begin_sim, double t_end_sim, double increment_sim)
43   {
44     a=a_constant;b=b_constant;c=c_constant;
45     x0=x_init;y0=y_init;z0=z_init;
46     t_begin=t_begin_sim;
47     t_end=t_end_sim;
48     dt=increment_sim;
49     quantity=(int)((t_end-t_begin)/dt);
50   }
51   /**Berechnet die x,y und z-Werte und übergibt die x,y,z-
52    * und t-Werte einem Array */
53   public void calculation(){int i=0;
54     t = new double [quantity+2];
55     x = new double [quantity+2];
56     y = new double [quantity+2];
57     z = new double [quantity+2];
58     x[i]=x0;y[i]=y0;z[i]=z0;t[i]=t_begin;
59     for(double t_zaehler=t_begin;
60         t_zaehler<=t_end; t_zaehler+=dt){i++;
61       double dx=(a*(y[i-1]-x[i-1]))*dt;
62       double dy=(b*x[i-1]-x[i-1]*z[i-1]-y[i-1])*dt;
63       double dz=(x[i-1]*y[i-1]-c*z[i-1])*dt;
64       x[i]=x[i-1]+dx;y[i]=y[i-1]+dy;z[i]=z[i-1]+dz;
65       t[i]=t_zaehler+dt;
66     }
67   }
```

```
68    /**Gibt die t-Werte zurück
69     * @return [] t-Werte */
70    public static double[] get_t_werte(){return t;}
71    /**Gibt die x-Werte zurück
72     * @return [] x-Werte */
73    public static double[] get_x_werte(){return x;}
74    /**Gibt die y-Werte zurück
75     * @return [] y-Werte */
76    public static double[] get_y_werte(){return y;}
77    /**Gibt die z-Werte zurück
78     * @return [] z-Werte */
79    public static double[] get_z_werte(){return z;}
80    /**Gibt die Anzahl zurück
81     * @return quantity */
82    public static int qet_quantity(){return (quantity+2);}
83    /**Schreibt die Array-Werte x,y,z und t in ein txt-File
84     * @param path_write_to_txt = Pfad
85     * @param anz = Anzahl der Schrittweite der Ausgabe */
86    public void write_to_txt(String path_write_to_txt, int anz){
87      String path = path_write_to_txt;
88          String arrayString = "";
89          int anzahl=anz;
90          for (int i = 0; i < t.length ; i+=anzahl){
91              arrayString = arrayString + t[i]+" , "
92                  + x[i]+" , " + y[i]+" , " + z[i]+"\n";
93              //  Fortschrittsanzeige (hilfreich)
94            System.out.println((double)i*100/(t.length)+" %");
95          } System.out.println(100+" %");
96          try {
97              BufferedWriter out =
98                  new BufferedWriter(new FileWriter(path));
99              out.write(arrayString);
100             out.close();
101         } catch (IOException e) {
102           System.out.println("Fehler beim Schreiben des Stringes
103           in die Datei. Überprüfen Sie bitte
104           den angegebenen Pfad.");
105         }
106   }
107 }
```

Abbildung 4.5: Java-Code zur Berechnung des Lorenz-Attraktors (Java)

Das erstellte Java-Programm liefert nach dem Durchlauf folgendes Simulationsergebnis für die angegebene Parameterkonstellation und einer Simulationszeit von 50 sec:

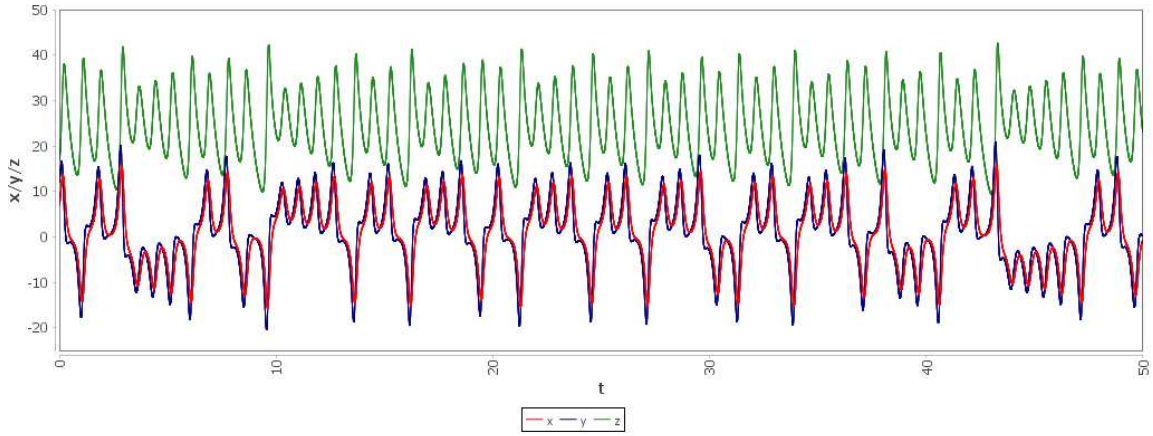


Abbildung 4.6: Zeitverläufe der Zustandsgrößen x, y & z beim Lorenz-Attraktor (Java)

An den Zeitverläufen lässt sich erkennen, dass, einfach gesagt, wenn ein gewisser x- bzw. y-Werte erreicht wurde, der Verlauf in die andere Richtung ausschlägt.

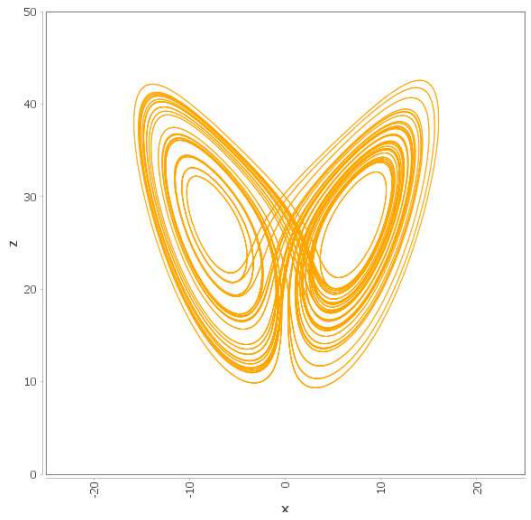


Abbildung 4.7: x/z-Trajektion des Lorenz-Attraktors (Java)

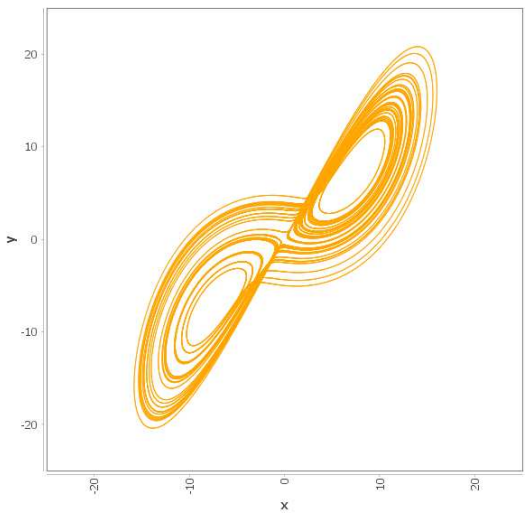


Abbildung 4.8: x/y-Trajektion des Lorenz-Attraktors (Java)

In der Abb. 4.7 ist der typische Verlauf, welcher einem *Schmetterling* ähnelt, ersichtlich.

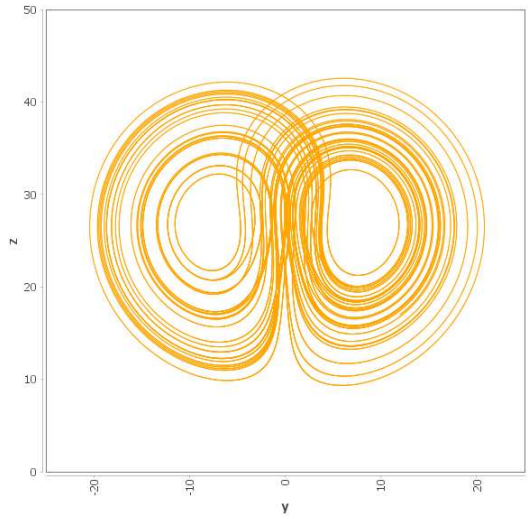


Abbildung 4.9: y/z-Trajektion des Lorenz-Attraktors (Java)

Die Abbildung links zeigt die y/z-Trajektion und es lässt sich hier schon auf eine gewisse Symmetrie schließen.

Die Aktivität dieses Attraktors, welcher in Abb. 4.9 zu erkennen ist, gibt das Chaos wieder, das den Prozess antreibt und ersichtlich ist, dass der Verlauf immer wieder von einem Flügel in den anderen, ab einem gewissen Wert, übergeht.

4.2.2 C

```
1 /*  Lorenz-Attraktor-Berechnung.c
2  *  Erstellungsdatum: 25.11.2015
3  *  Autor: Labenbacher Michael */
4 #include <stdlib.h>
5 #include <math.h>
6 #include <stdio.h>
```

Abbildung 4.10: C-Code (Include-Dateien) zur Berechnung des Lorenz-Attraktors (C)

```
7  /*  ========================= Variables =========================
8   *  States: x, y, z
9   *  Constants: a, b, c
10  *  Domain: dependent variable,
11  *      begin-, end of simulation and increment
12  *  Arrays: x, y, z, t
13  *  File: txt-Datei */
14 volatile long double a,b,c;
15 double t_begin=0,t_end=50,dt=0.00001;
16 volatile long double x[5000000];
17 volatile long double y[5000000];
18 volatile long double z[5000000];
19 volatile long double t[5000000];
20 FILE *datei;
21 /*  ========================= Definitions =======================
22  *  of functions:
23  *    init --> Initialisation
24  *    simulation --> Simulation (Calculation) */
25 void init(long double x_init, long double y_init,
26           long double z_init, long double a_init,
27           long double b_init, long double c_init){
28   x[0]=x_init;
29   y[0]=y_init;
30   z[0]=z_init;
31   a=a_init;
32   b=b_init;
33   c=c_init;
34 }
```

```c
void simulation(){
  t[0]=t_begin;
  int i = 0;
  long double t_zaehler;
  //  for(<initialisation>;<conditions>;<update>)
  for(t_zaehler=t_begin; t_zaehler<=t_end; t_zaehler+=dt){i++;
    t[i]=t[i-1]+dt;
    long double  dx=(a*(y[i-1]-x[i-1]))*dt;
    long double dy=(b*x[i-1]-x[i-1]*z[i-1]-y[i-1])*dt;
    long double dz=(x[i-1]*y[i-1]-c*z[i-1])*dt;
    x[i]=x[i-1]+dx;
    y[i]=y[i-1]+dy;
    z[i]=z[i-1]+dz;
  }
}

int write_to_txt(){
  datei = fopen("Lorenz_Attraktor_c_werte.txt", "w");
  if(NULL == datei) {
      printf("Konnte Datei \"Lorenz_Attraktor_c_werte.txt\"
              nicht öffnen!\n");
      return 1;
  }

  int i;
  for(i=0;i<5000000;i+=100){
    fprintf(datei, "%e,%e,%e,%e\n", (double)x[i],
          (double)y[i], (double)z[i], (double)t[i]);
  }

  fclose(datei);
  return 0;
}

int main(void){
  init(7,12,17,10,28,8/3);
  simulation();
  write_to_txt();
  return 0;
}
```

Abbildung 4.11: C-Code zur Berechnung des Lorenz-Attraktors (C)

Die durch C erstellte exe-Datei wurde ausgeführt, was ein txt-File ergab, welches nun mittels dem Computeralgebrasystem Maxima eingelesen werden kann, um so den Attraktor graphisch darzustellen.

```
(%i1) kill(all);
```

(%o0) *done*

Laden des *dynamics*- und *coma*(*draw*)-Paketes und setzen von Defaultwerten:

```
(%i3)   load(coma)$
        load(dynamics)$
        set_draw_defaults(grid=true,point_type=0,
                          points_joined=true)$
```

coma v.1.73, (Wilhelm Haager, 2015-01-09)

```
(%i8)  werte_c:read_nested_list("C:\\Users\\User\\Desktop\\Schule
              \\Laboratorium-5AHET\\04_Simulation
              \\01_Lorenz_Attraktor
              \\Lorenz_Attraktor_c_werte.csv",comma)$
       x_werte_c:map(first,werte_c)$
       y_werte_c:map(second,werte_c)$
       z_werte_c:map(third,werte_c)$
       t_werte_c:map(fourth,werte_c)$
```

Zeitverläufe:

```
(%i9)  wxplot_size:[800,400];
```

(%o9) [800,400]

```
(%i10)
wxdraw2d(   color=red,          points(t_werte_c,x_werte_c),
            color=navy,         points(t_werte_c,y_werte_c),
            color=forest-green, points(t_werte_c,z_werte_c),
            yrange=[-25,50],xrange=[0,50],
            xaxis=true,dimensions=[1500,600],
      xlabel="t",ylabel="x(rot) / y(blau) / z(gruen)"
);
```

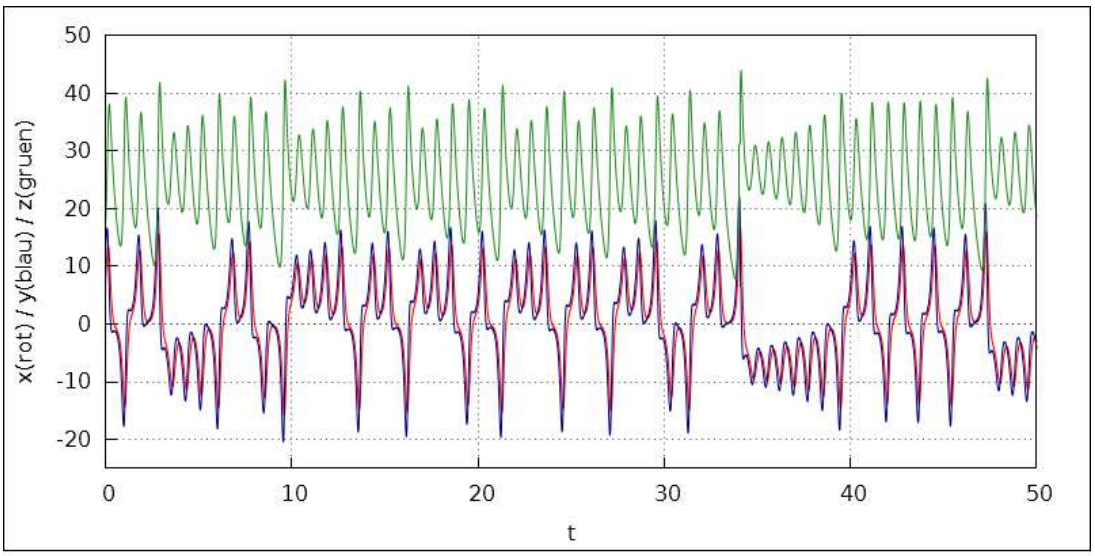


Abbildung 4.12: Zeitverläufe der Zustandsgrößen x, y & z beim Lorenz-Attraktor (C)

(%t10)

(%o10)

Trajektorie in der x/z-Phasenebene:

```
(%i11) wxplot_size:[600,600];
```

(%o11) [600,600]

```
(%i12) wxdraw2d(   color=orange,points(x_werte_c,z_werte_c),
                yrange=[0,50],xrange=[-25,25],
                   xaxis=true,yaxis=true,
            xlabel="x",ylabel="z",
            user_preamble = "set size ratio 1"
       );
```

Trajektorie in der x/y-Phasenebene:

```
(%i13) wxdraw2d(   color=orange,points(x_werte_c,y_werte_c),
                   yrange=[-25,25],xrange=[-25,25],
                   xaxis=true,yaxis=true,
                   xlabel="x",ylabel="y",
                   user_preamble = "set size ratio 1"
       );
```

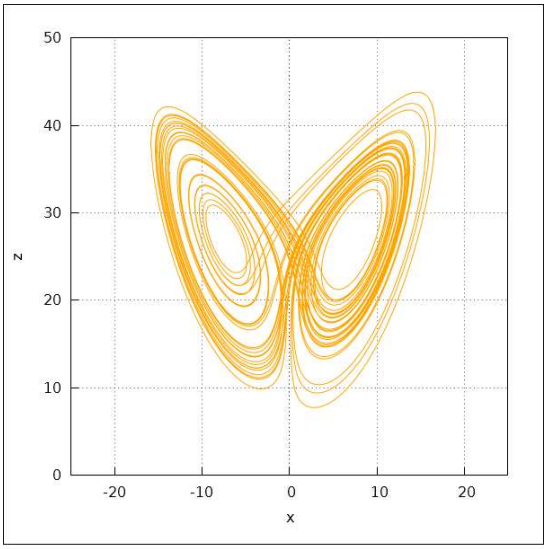


Abbildung 4.13: x/z-Trajektion des Lorenz-Attraktors (C)

(%t12)

(%o12)

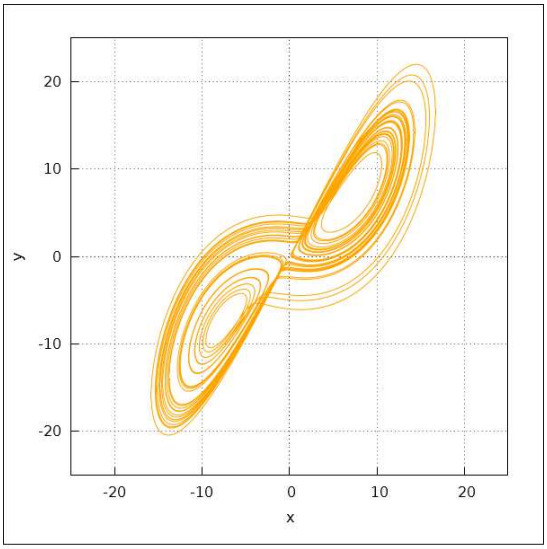


Abbildung 4.14: x/y-Trajektion des Lorenz-Attraktors (C)

(%t13)

(%o13)

Trajektorie in der y/z-Phasenebene:

```
(%i14) wxdraw2d(   color=orange,points(y_werte_c,z_werte_c),
                   yrange=[0,50],xrange=[-25,25],
                   xaxis=true,yaxis=true,
                   xlabel="y",ylabel="z",
                   user_preamble = "set size ratio 1"
       );
```

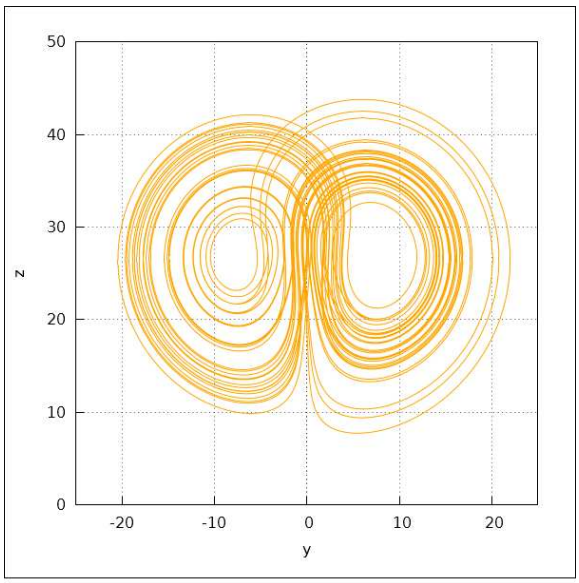


Abbildung 4.15: y/z-Trajektion des Lorenz-Attraktors (C)

(%t14)

(%o14)

Vergleicht man die Abbildungen 4.15 (C) & 4.9 (Java), so kann man im ersten Augenblick keine wirkliche Gleichheit erkennen, jedoch wird die Gegenüberstellung der Verläufe im Kap. 4.5 zeigen, dass C und Java sehr ähnliche Simulationsergebnisse liefern und erst nach relativ langer Zeit abweichen.

4.2.3 LaTeX

Eine einfache Berechnung des Lorenz-Attraktors ist auch mittels LaTeX möglich, indem schrittweise die Differenzengleichungen ausgewertet und in einer Datei abgespeichert werden. (Die Berechnung sollte aber nur ein einziges Mal erfolgen, da man sonst immer sehr lange warten muss.)

Der folgende LaTeX-Code berechnet den Lorenz-Attraktor auf eine sehr einfache Art und Weise:

```
1 \def\distance{ } %% Abstand (für write-outfile)
2 \newboolean{SetCalculationLorenzOn} %% Deklaration
3 \setboolean{SetCalculationLorenzOn}{false} %% Zuweisung
4 \newcounter{cti} %% Counter 1 (i)
5 \newcounter{ctk} %% Counter 2 (k)
6 \newcounter{ctm} %% Counter 3 (m)
```

Abbildung 4.16: LaTeX-Code (Preamble) zur Berechnung des Lorenz-Attraktors (-LaTeX)

```
1  %% Abfrage ob der Lorenzattraktor berechnet werden soll
2  \ifthenelse{\boolean{SetCalculationLorenzOn}}
3  %% IF
4  {
5      \setcounter{cti}{0}
6      \setcounter{ctk}{0}
7      \setcounter{ctm}{0}
8      \FPset\lort{0}
9      \FPset\lora{10}
10     \FPset\lorb{28}
11     \FPset\lorc{0}
12     \FPdiv\lorc 8 3
13     \FPset\lorx{7}
14     \FPset\lory{12}
15     \FPset\lorz{17}
16     \FPset\lorxv{\lorx}
17     \FPset\loryv{\lory}
18     \FPset\lorzv{\lorz}
19     \FPset\lordx{0}
20     \FPset\lordy{0}
21     \FPset\lordz{0}
22     \FPset\lordt{0.001}
```

```
23      %% Variable outfile anlegen:
24      \newwrite\outfile
25      %% Datei öffnen zum Schreiben
26      %% (einmalig! (löscht die Datei und erzeugt eine neue))
27      \immediate\openout\outfile=lorenz_latex.dat
28      \forloop{cti}{0}{\value{cti}<10}
29      {
30          \forloop{ctk}{0}{\value{ctk}<105}
31          {
32              \immediate\write\outfile{
33                \lorx\distance\lory\distance\lorz\distance\lort}
34              \forloop{ctm}{0}{\value{ctm}<30}
35              {
36                  \FPset\lorxv\lorx
37                  \FPset\loryv\lory
38                  \FPset\lorzv\lorz
39                  %% dx = ( a * ( y - x ) ) * dt
40                  \FPeval {\lorx}
41                     {((\lora*(\lory-\lorx))*\lordt)+\lorxv}
42                  %% dy = ( x * (b - z ) - y ) * dt
43                  \FPeval {\lory}
44                     {((\lorx*(\lorb-\lorz)-\lory)*\lordt)+\loryv}
45                  %% dz = ( x * y - c * z ) * dt
46                  \FPeval {\lorz}
47                     {((\lorx*\lory-\lorc*\lorz)*\lordt)+\lorzv}
48                  %% t  = t + dt
49                  \FPeval {\lort} {\lort + \lordt}
50              }
51          }
52      }
53      %% Datei schließen
54      \immediate\closeout\outfile
55  }
56  %% ELSE
57  {}
```

Abbildung 4.17: LaTeX-Code zur Berechnung des Lorenz-Attraktors (LaTeX)

Nun folgt die Darstellung des zuvor berechneten Lorenz-Attraktors mit Hilfe des animate-Packages, wobei diese Methode einige Zeit in Anspruch nimmt und somit eine IF-Abfrage durchgeführt wird:

```
1  \newboolean{SetDisplayLorenzOn} %% Deklaration
2  \setboolean{SetDisplayLorenzOn}{true} %% Zuweisung
3  %% ? Filter Warnungen allgemein ausschalten, da man Filter
4  %% nur einsetzt, wenn man sie braucht. --> Meines Erachtens
5  %% sind diese somit unnötig und können ausgeschalten werden.
6  \pgfplotsset{filter discard warning=false}
7  %% Filter für die Darstellung der Animation.
8  \pgfplotsset{select coords between index/.style 2 args={
9    x filter/.code={
10     \ifnum\coordindex<#1\def\pgfmathresult{}\fi
11     \ifnum\coordindex>#2\def\pgfmathresult{}\fi
12     }
13 }}
```

Abbildung 4.18: LaTeX-Code (Preamble) zur Simulation des Lorenz-Attraktors (-LaTeX)

```
%% Abfrage ob der Lorenz-Attraktor dargestellt werden soll
\ifthenelse{\boolean{SetDisplayLorenzOn}}
{
    \begin{figure}[H]
    \begin{center}
    \begin{animateinline}
        [poster = last, controls]{25}
    \setcounter{cti}{0}
    \forloop{cti}{0}{\value{cti}<1000}
    {
      \begin{tikzpicture}[x=1mm,y=1mm]
        \begin{axis}[xmin=-30,xmax=30,
                ymin=-30,ymax=30,zmin=0,zmax=50,
                view={40}{30},grid,
                scale=1.8,xlabel=x,ylabel=y,zlabel=z]
          \addplot3[mark=none,color=ForestGreen,smooth]
            table[select coords between index={0}{\thecti}]
              {lorenz_latex1.dat};
          \addplot3[only marks,mark=*,color=ForestGreen]
            table[select coords between index={\thecti}{\thecti}]
              {lorenz_latex1.dat};
          \end{axis}
      \end{tikzpicture}
      \ifthenelse{\value{cti}<999}
      {
        \newframe
      }
      {
        \end{animateinline}\relax
      }
    }
    \end{center}
    \caption{Trajektorie des Lorenz-Attraktors im
              dreidimensionalen Zustandsraum (\LaTeX)}
    \label{fig:Trajektorie des Lorenz-Attraktors im
            dreidimensionalen Zustandsraum (LaTeX)}
  \end{figure}
}
%% ELSE
{}
```

Abbildung 4.19: LaTeX-Code zur Simulation des Lorenz-Attraktors (LaTeX)

Das Ziel dieses Kapitels war die Darstellung und Simulation des Lorenz-Attraktors im dreidimensionalen Zustandsraum direkt in LaTeX. Dabei ergab sich folgendes Diagramm:

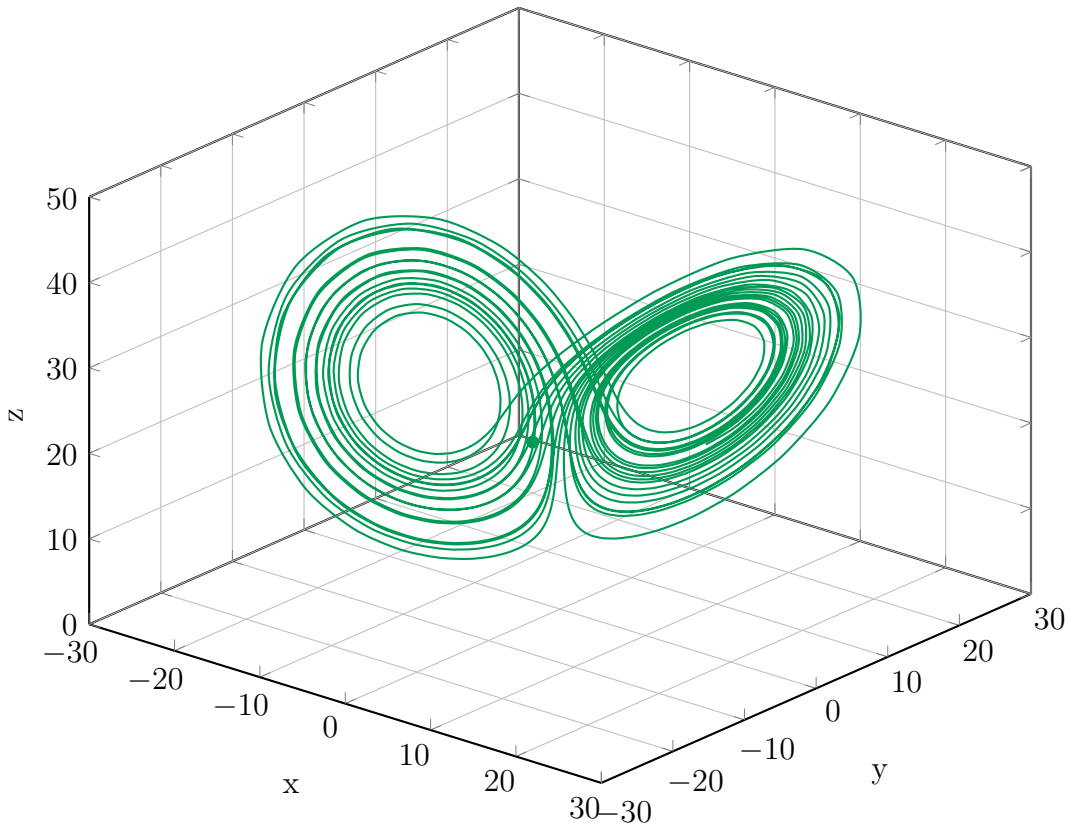


Abbildung 4.20: Trajektorie des Lorenz-Attraktors im dreidimensionalen Zustandsraum (LaTeX)

Man kann somit auch ohne andere Programme einen Attraktor simulieren, um so den Verlauf dessen graphisch darzustellen.

4.3 Simulation mit Funktionsblöcken

4.3.1 Scilab (Xcos)

Mit Hilfe von Scilab und dem Simulationsaufsatz Xcos lässt sich der Lorenz-Attraktor mit einem Blockschaltbild, bestehend aus einzelnen Funktionsblöcken, aufbauen:

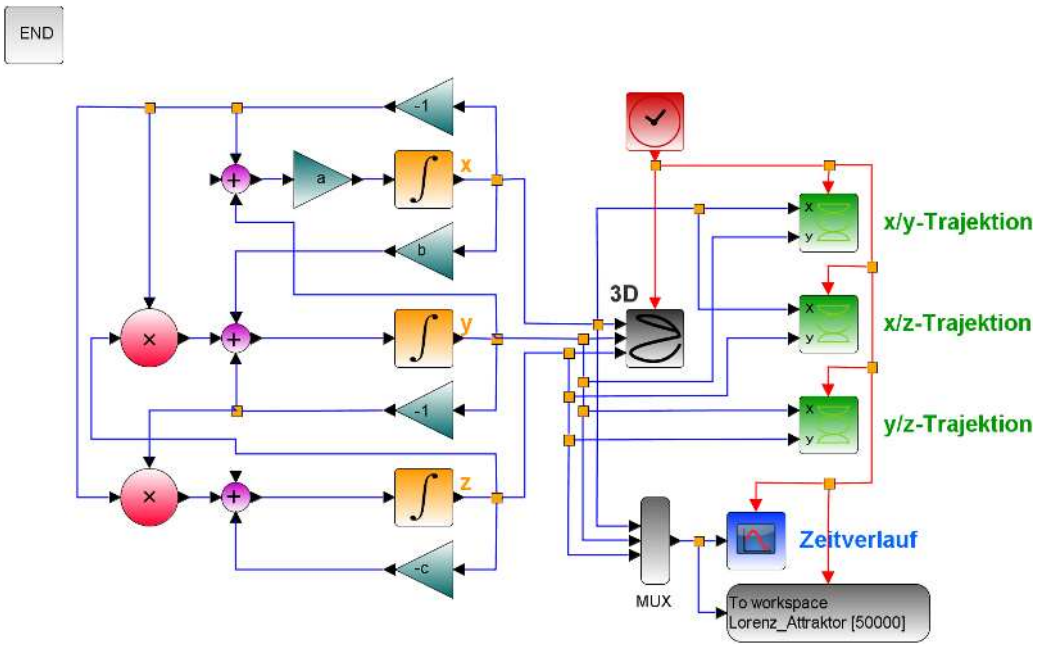


Abbildung 4.21: Blockschaltbild des Lorenz-Attraktors (Scilab(Xcos))

Dieses Programm wurde so entwickelt, dass eine x/y-, x/z- & y/z-Trajektion in einem eigenen Frame dargestellt wird. Des Weiteren wird der Zeitverlauf und eine $3D$-Darstellung angefertigt. Für die Simulation müssen noch in Scilab die Parameter des Lorenz-Attraktors eingegeben werden:

```
a=10;
b=28;
c=8/3;
```

Mit folgender Eingabe können die simulierten Werte in eine csv-Datei geschrieben werden:

```
write_csv(Lorenz_Attraktor.time,'Lorenz_xcos_time.csv')
write_csv(Lorenz_Attraktor.values,'Lorenz_xcos_values.csv')
```

Als Gleichungslöser wurde bei Scilab (Xcos) *Sundials/CVODE - BDF - NEWTON* verwendet, wobei insgesamt 50 000 Punkten im Zeitbereich von $0 - 50$ berechnet wurden.

Parameter	Einstellung
Finale Integrationszeit	$1.0E03$
Echt-Zeit-Skalierung	$0,0E00$
Absolute Toleranz des Integrators	$1,0E-10$
Relative Toleranz des Integrators	$1,0E-10$
Zeit-Toleranz	$1,0E-12$
Maximales Zeitintervall der Integration	$1,00001E05$
Gleichungslöser	Sundials/CVODE - BDF - NEWTON
Maximale Schrittweite (0 = kein Limit)	$0,0E00$

Tabelle 4.1: Parametereinstellungen in Scilab für den Lorenz-Attraktor (Xcos)

Das Ergebnis der Simulation des Lorenz-Attraktors lieferte folgende Zeitverläufe der Zustandsgrößen:

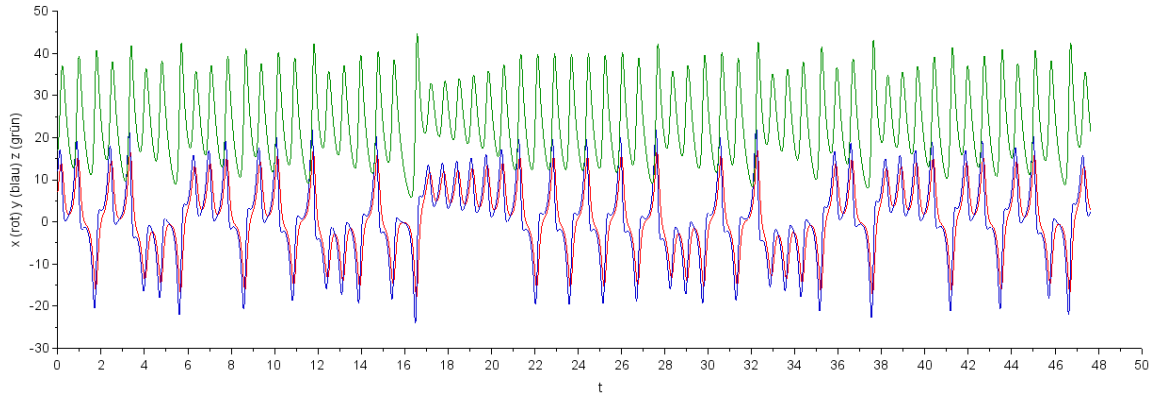


Abbildung 4.22: Zeitverläufe der Zustandsgrößen x, y & z beim Lorenz-Attraktor (Scilab (Xcos))

In der Abb. 4.22 lässt sich der typische Verlauf eines Lorenz-Attraktors erkennen und jedesmal wenn x vom negativen ins positive bzw. umgekehrt wechselt, so ist dies im dreidimensionalen Raum, siehe Abb. 4.23, als Wechsel vom einen „Flügel“ zum anderen bzw. umgekehrt zu erkennen.

Die nachfolgende Abbildung zeigt den Lorenz-Attraktor mit Rotationswinkeln von 60 ° und 120 °:

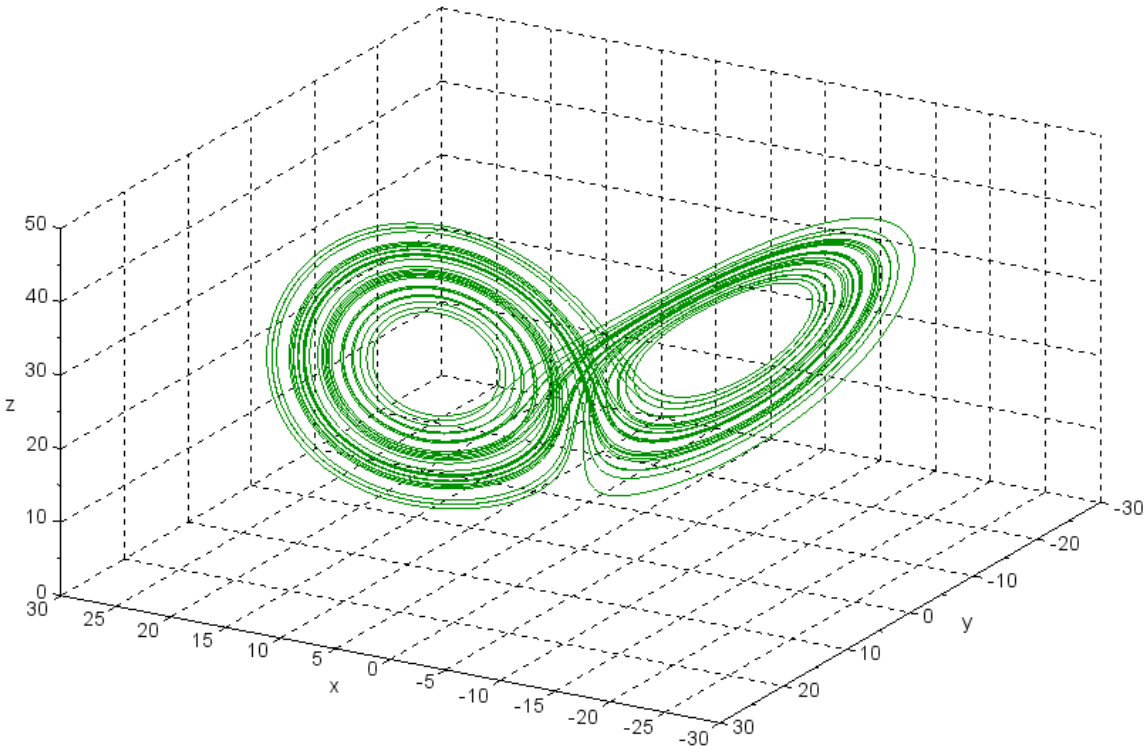


Abbildung 4.23: Trajektorie des Lorenz-Attraktors im dreidimensionalen Zustandsraum (Scilab (Xcos))

Die einzelnen Trajektorien werden hier nicht erneut dargestellt und in Abb. 4.23 ist erkennbar, dass der Lorenz-Attraktor aus zwei scheibenartigen Teilen besteht, welche leicht gegen die z-Achse gekippt sind.

4.3.2 Matlab (Simulink)

Die Software Matlab besitzt als Zusatzpaket Simulink zur Modellierung von System, wie z. B. den Lorenz-Attraktor. Dabei kann wieder das Blockschaltbild erstellt werden und nach einigen Einstellungen kann die Simulation gestartet werden.

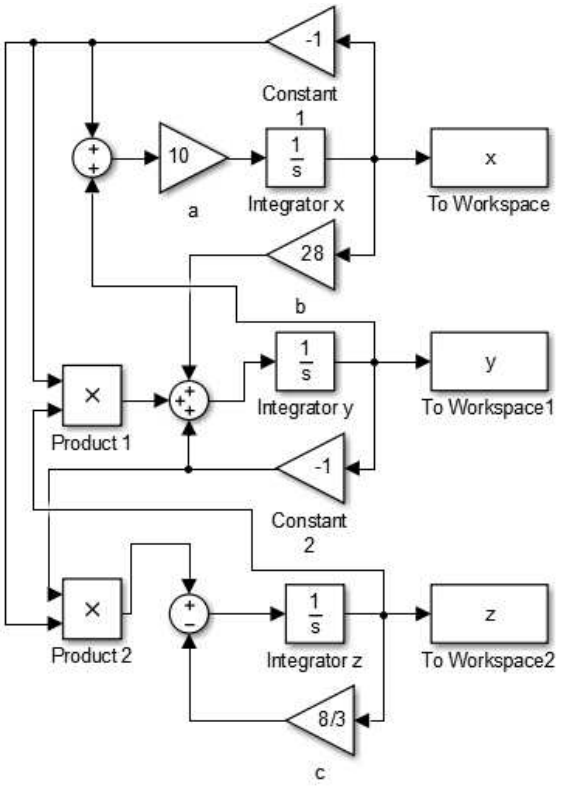


Abbildung 4.24: Blockschaltbild des Lorenz-Attraktors (Matlab (Simulink))

Da das Programm das selbe Ergebnis, wie das im Kap. 4.3.2, liefert, wird das Ergebnis hier nicht angeführt, sondern ist dem nachfolgenden Kapitel zu entnehmen.

4.4 Simulation mit dem Runge-Kutta-Verfahren

4.4.1 Maxima

Mit dem Computeralgebrasystem Maxima lässt sich ein solches System, wie schon im Kapitel 3.2.1 bekannt geworden ist, nach dem Runge-Kutta-Verfahren 4. Ordnung lösen. Folgendes Programm wurde für den Lorenz-Attraktor erstellt, wobei das chaotische Verhalten und die Symmetrie untersucht wurden.

```
(%i1)   kill(all)$
```

Laden des *dynamics*- und *coma*(*draw*)-Paketes und setzen von Defaultwerten:

```
(%i3)   load(coma)$
        load(dynamics)$set_draw_defaults(
           grid=true,point_type=0,points_joined=true)$
```

coma v.1.73, (Wilhelm Haager, 2015-01-09)

Funktionen:

```
(%i6)   f1 : a*(y - x)$
        f2 : b*x - x*z - y$
        f3 : x*y - c*z$
```

Zustandsgrößen:

```
(%i7)   states:[x,y,z]$
```

Konstanten:

```
(%i8)   constants:[a=10,b=28,c=8/3]$
```

Anfangsbedingungen:

```
(%i9)   initialisation:[7,12,17]$
```

Domäne (abhängige Variable t (Zeit), Simulationsbeginn, Simulationsende, Schrittweite):

```
(%i10)  domain:[t,0,50,0.0001]$
```

Berechnung und Zerlegung:
rk (functions, states, initialisation, domain)

```
(%i11)  res:rk([ev(f1,constants),ev(f2,constants),ev(f3,constants)],
               states,initialisation,domain)$
```

Nun wird der Rückgabewert von der *rk*-Funktion in einzelne Listen zerlegt:

```
(%i15)  t_werte_maxima:map(first,res)$x_werte_maxima:
        map(second,res)$y_werte_maxima:map(third,res)$
        z_werte_maxima:map(fourth,res)$
(%i16)  wxplot_size:[800,400]$
(%i17)  wxdraw2d(color=red,points(t_werte_maxima,x_werte_maxima),
                 color=navy,points(t_werte_maxima,y_werte_maxima),
                 color=forest-green,
                 points(t_werte_maxima,z_werte_maxima),
                 yrange=[-25,50],xrange=[0,domain[3]],
                 xaxis=true,dimensions=[1500,600],xtics=5,ytics=5,
                 xlabel="t",ylabel="x(rot) / y(blau) / z(gruen)")$
```

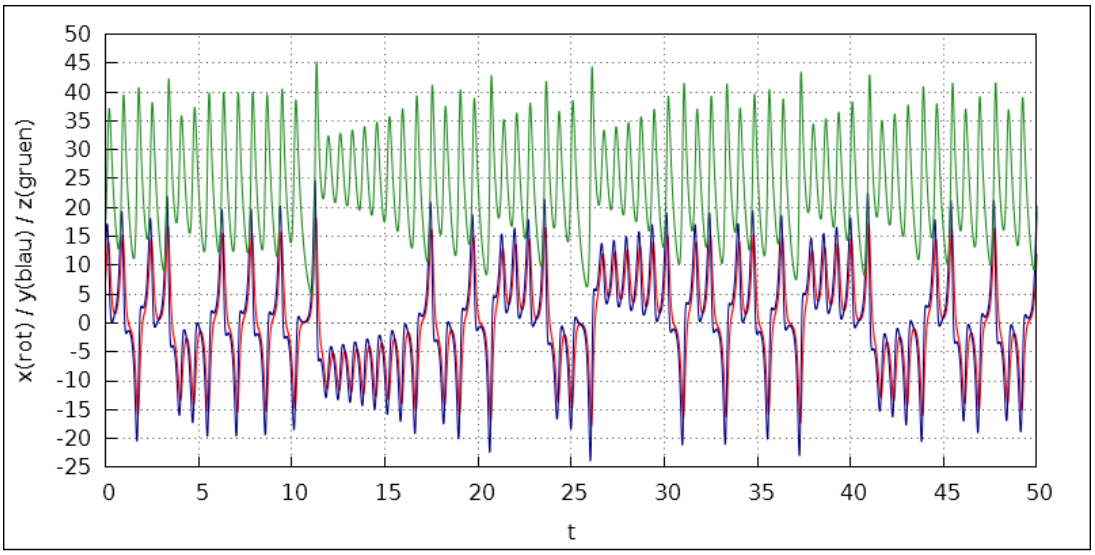


Abbildung 4.25: Zeitverläufe der Zustandsgrößen x, y & z beim Lorenz-Attraktor (Maxima)

(%t17)

Trajektorie in der x/z-Phasenebene:

```
(%i18)  wxplot_size:[600,600]$
(%i19)  wxdraw2d(color=orange,points(x_werte_maxima,z_werte_maxima),
                 yrange=[0,50],xrange=[-25,25],
                 xaxis=true,yaxis=true,xtics=5,ytics=5,
                 xlabel="x",ylabel="z",
                 user_preamble = "set size ratio 1"
        );
```

Trajektorie in der x/y-Phasenebene:

```
(%i20)  wxdraw2d(color=orange,points(x_werte_maxima,y_werte_maxima),
                 yrange=[-25,25],xrange=[-25,25],
                 xaxis=true,yaxis=true,xtics=5,ytics=5,
                 xlabel="x",ylabel="y",
                 user_preamble = "set size ratio 1"
        );
```

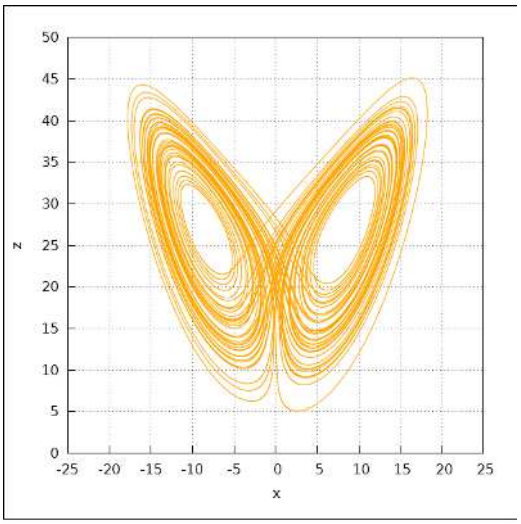


Abbildung 4.26: x/z-Trajektion des Lorenz-Attraktors (Maxima)

(%t19)

(%o19)

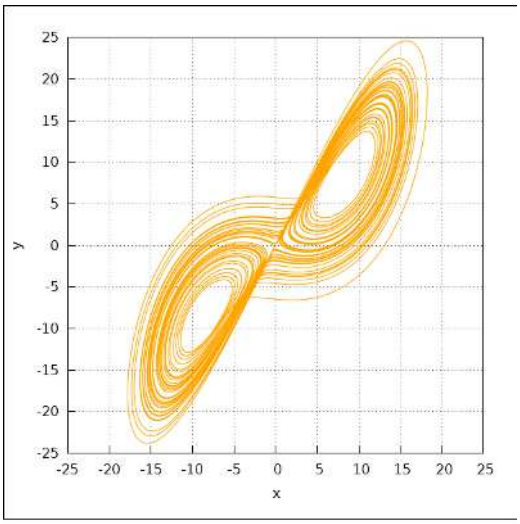


Abbildung 4.27: x/y-Trajektion des Lorenz-Attraktors (Maxima)

(%t20)

(%o20)

Trajektorie in der y/z-Phasenebene:

```
(%i21)  wxdraw2d(color=orange,points(y_werte_maxima,z_werte_maxima),
                 yrange=[0,50],xrange=[-25,25],
                 xaxis=true,yaxis=true,xtics=5,ytics=5,
                 xlabel="y",ylabel="z",
                 user_preamble = "set size ratio 1"
        );
```

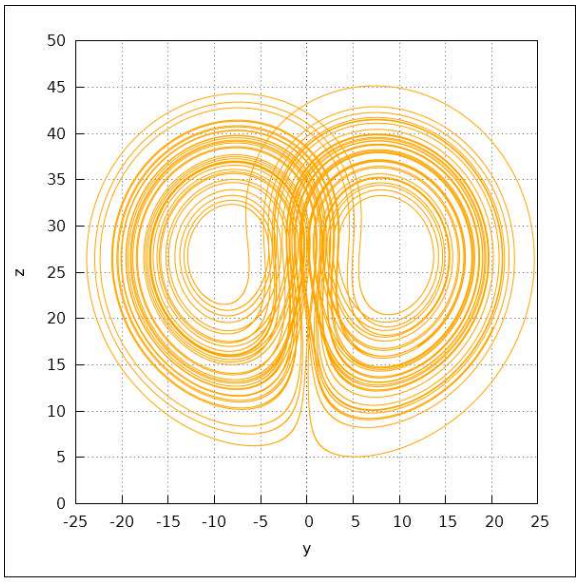


Abbildung 4.28: y/z-Trajektion des Lorenz-Attraktors (Maxima)

(%t21)

(%o21)

```
(%i23)  wxplot_size:[1000,800]$
        wxanimate_framerate:10$
(%i24)  domain_g:[t,0,50,0.001]$
(%i25)  res_g:rk([ev(f1,constants),ev(f2,constants),
           ev(f3,constants)],states,initialisation,domain_g)$
(%i29)  t_werte_maxima_g:map(first,   res_g)$
        x_werte_maxima_g:map(second,  res_g)$
        y_werte_maxima_g:map(third,   res_g)$
        z_werte_maxima_g:map(fourth,  res_g)$
```

```
(%i32)  ob1:makelist(x,i,1,1,1)$
        for n:(length(x_werte_maxima_g)-1) step -10 while n>0 do
        push([x_werte_maxima_g[n],y_werte_maxima_g[n],
        z_werte_maxima_g[n]],ob1)$ ob1:delete(x,ob1)$
(%i33)  with_slider_draw3d(k,makelist(i,i,1,point_size=1,zaxis=true,
            (length(x_werte_maxima_g)-1)/10,10),color=dark-green,
            yrange=[-25,25],xrange=[-25,25],zrange=[0,50],
            ob1p:points(makelist(ob1[n],n,1,k)),xtics=5,ytics=5,
            xlabel="x",ylabel="y",zlabel="z",view=[60,40],
            grid=false,xu_grid=5,yv_grid=5,ztics=5,xaxis=true,
            yaxis=true,user_preamble= "set xyplane at 0")$
```

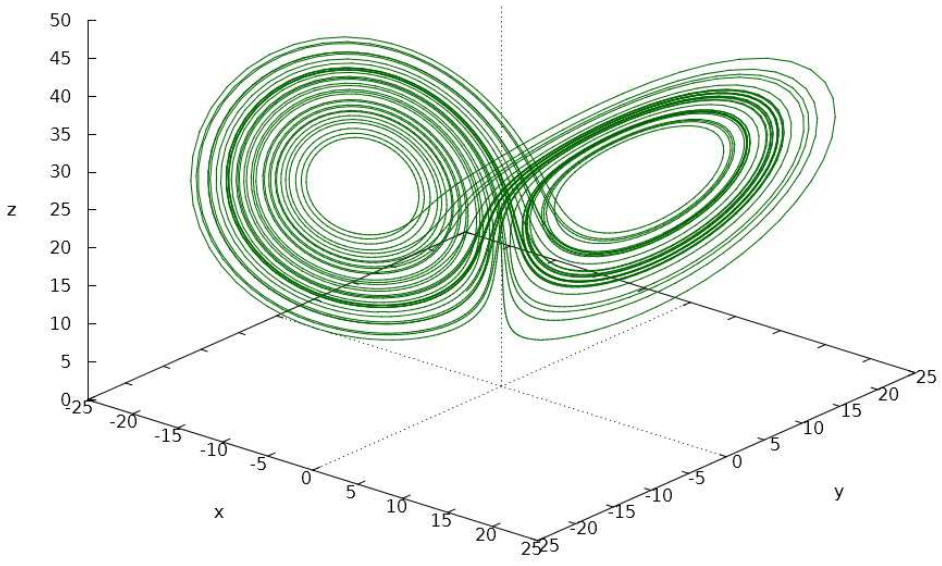


Abbildung 4.29: Trajektorie des Lorenz-Attraktors im dreidimensionalen Zustandsraum (Maxima)

Untersuchung des chaotischen Verhaltens:

Eine beliebig kleine Änderung der Anfangsbedingungen (oder Systemparameter) verursacht eine große Änderung im Zeitverhalten. ⇒ Das charakteristische Merkmal eines chaotischen Systems.

```
(%i36)  initialisations:[7,12,17*(1-0.001/100)]$
        ress:rk([ev(f1,constants),ev(f2,constants),
                ev(f3,constants)],states,initialisations,domain)$
(%i40)  ts_werte_maxima:map(first,ress)$xs_werte_maxima:
        map(second,ress)$ys_werte_maxima:
        map(third,ress)$zs_werte_maxima:map(fourthress)$
(%i41)  wxplot_size:[800,400]$
(%i42)  wxdraw2d(color=navy,points(t_werte_maxima,x_werte_maxima),
        color=dark-gray,xlabel="t",xtics=5,ytics=5,
        points(ts_werte_maxima,xs_werte_maxima),yrange=[-25,25],
        xaxis=true,dimensions=[1500,600],xrange=[0,domain[3]/2],
        ylabel="y_v_o_r_h_e_r(blau) / y_n_a_c_h_h_e_r(grau)");
```

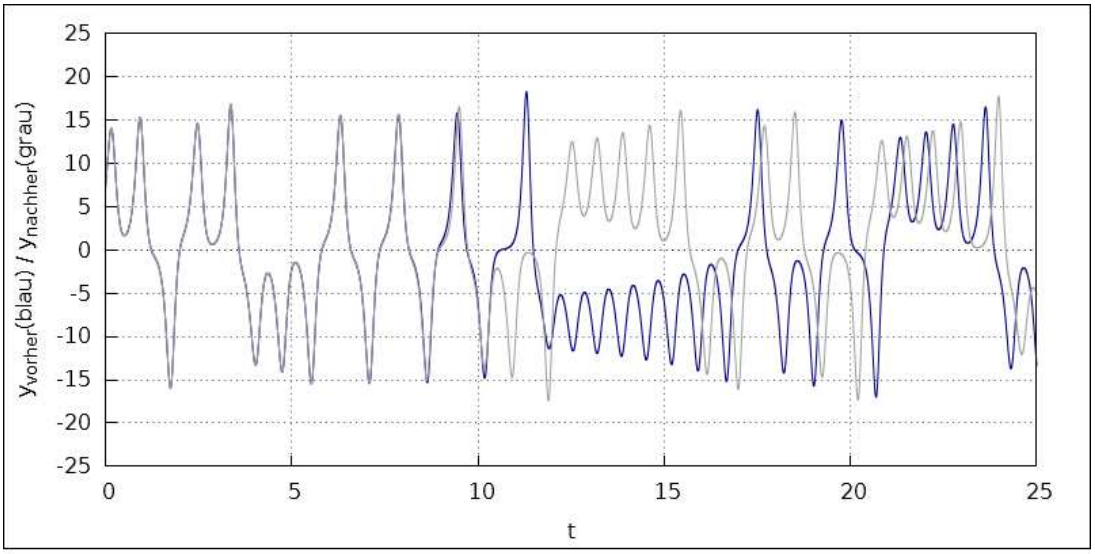


Abbildung 4.30: Zeitverlauf der Zustandsgröße y bei um $0,001\,\%$ unterschiedlichen Anfangsbedingungen beim Lorenz-Attraktor (Maxima)

(%t42)
(%o42)

Untersuchung der Symmetrie:

Hier wird die Symmetrie bezüglich der z-Achse untersucht, indem zwei Anfangsbedingungen gewählt werden, welche sich bezüglich x & y nur ums Vorzeichen unterscheiden.

```
(%i44)  initialisation1:[0.001,0.001,1]$
        initialisation2:[-0.001,-0.001,1]$
(%i45)  domain12:[t,0,25,0.0001]$
(%i47)  res1:rk([ev(f1,constants),ev(f2,constants),
                ev(f3,constants)],states,initialisation1,domain12)$
        res2:rk([ev(f1,constants),ev(f2,constants),
                ev(f3,constants)],states,initialisation2,domain12)$
(%i55)  t1_werte_maxima:map(first,   res1)$
        x1_werte_maxima:map(second,  res1)$
        y1_werte_maxima:map(third,   res1)$
        z1_werte_maxima:map(fourth,  res1)$
        t2_werte_maxima:map(first,   res2)$
        x2_werte_maxima:map(second,  res2)$
        y2_werte_maxima:map(third,   res2)$
        z2_werte_maxima:map(fourth,  res2)$
```

Darstellung anhand der y-Werte:

```
(%i56)  wxplot_size:[800,400]$
(%i57)  wxdraw2d(color=navy,points(t1_werte_maxima,y1_werte_maxima),
                 color=dark-gray,
                 points(t2_werte_maxima,y2_werte_maxima),
                 yrange=[-30,30],xrange=[0,25],xtics=5,ytics=5,
                 xaxis=true,dimensions=[1500,600],
                 xlabel="t",ylabel="y"
        );
```

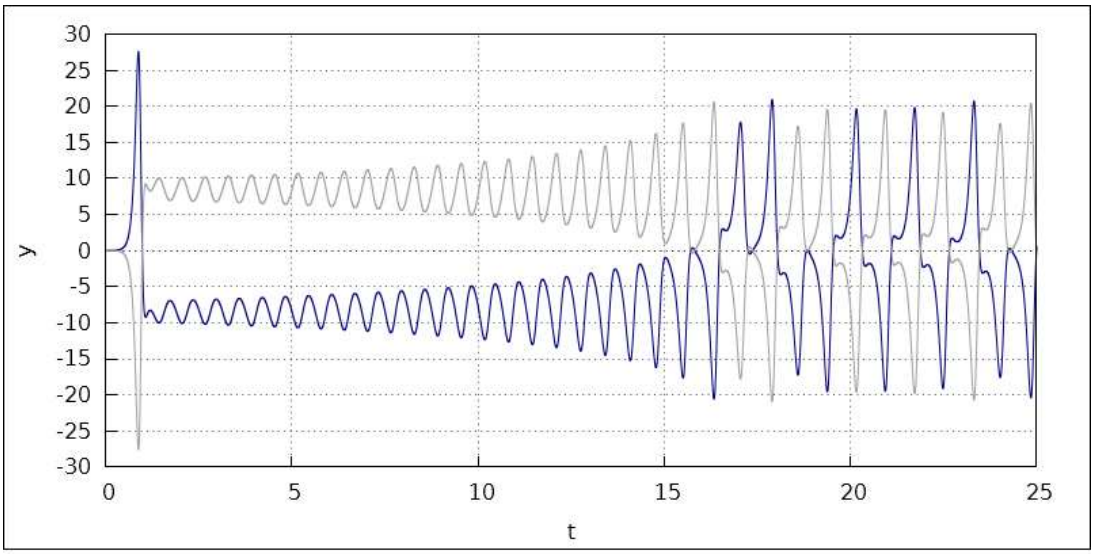


Abbildung 4.31: Zeitverlauf der Zustandsgröße y bei um die z-Achse gespiegelten Anfangsbedingungen beim Lorenz-Attraktor (Maxima)

(%t57)

(%o57)

Dies Abbildung zeigt sehr deutlich die vorhandene Symmetrie, da z. B. die Summe der beiden Verläufe immer 0 ergeben würde.

```
(%i58)  wxplot_size:[600,600]$
(%i59)  wxdraw2d(color=navy,points(x1_werte_maxima,z1_werte_maxima),
                 color=dark-gray,
                 points(x2_werte_maxima,z2_werte_maxima),
                 yrange=[0,50],xrange=[-25,25],yaxis=true,xtics=5,
                 ytics=5,xlabel="x",ylabel="z",xaxis=true,
                 user_preamble = "set size ratio 1");
```

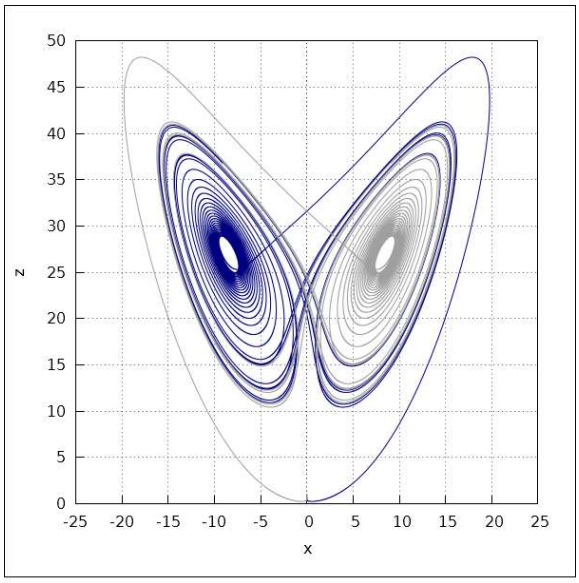


Abbildung 4.32: x/z-Trajektion des Lorenz-Attraktors bei um die z-Achse gespiegelten Anfangsbedingungen (Maxima)

(%t59)

(%o59)

4.4.2 Matlab

Dieses Programm verwendet z. B. das Runge-Kutta-Dormand-Prince (RKDP)-Verfahren zur Lösung von gewöhnlichen Differentialgleichungen. Dieses Verfahren ist ein Mitglied der Runge-Kutta-Methoden, genau genommen ein einschrittiges, adaptives Runge-Kutta-Verfahren (adaptives Verfahren ... Es versucht die Schrittweite zu optimieren). Das RKDP-Verfahren verwendet dabei Approximationen 4. & 5. Ordnung.

Das nachfolgende m-File wurde mit Hilfe von Matlab erstellt und berechnet den Lorenz-Attraktor nach folgender Eingabe:

```
cd C:\Users\User\Desktop\Schule\Laboratorium-5AHET
     \04_Simulation\_Matlab_Workspace\Lorenz_Attraktor
lorenz_function(10,28,8/3)
```

Das m-File:

```
1  function [x,y,z] = lorenz_function(a, b, c, initVar, T, dt)
2  % lorenz_function berechnet den Lorenz-Attraktor mit Hilfe des
3  % ode45-Befehles (Runge-Kutta-Verfahren (4,5))
4
5  % Wenn zu wenige Argumente angegeben wurden:
6  if nargin<3
7    error('MATLAB:lorenz:NotEnoughInputs',
8          'Not enough input arguments.');
9  end
10 % Wenn zu wenige Argumente angegeben wurden,
11 % jedoch die Systemparameter
12 % bekannt sind:
13 if nargin<4
14   initVar   = [7 12 17];% Initialisierung (Startwerte)
15   T         = [0 50];   % Zeitbereich
16   dt        = 10e-12;   % "Schrittweite"
17 end
```

```
18 options = odeset('RelTol',dt,'AbsTol',[dt dt dt/10]);
19 % RelTol = Toleranz, welche ein Maß für die Abweichung in Bezug
20 %           auf die Größe der einzelnen Lösungskomponenten
21 %           darstellt.
22 % AbsTol = Die absolute Fehlertoleranzen bestimmen die
23 %           Genauigkeit, wenn die Lösung gegen Null geht.
24 [T,X]    = ode45(@(T,X) F(T, X, a, b, c), T, initVar, options);
25 % [Zeit, Ergebnisarray]
26 % = solver("Zustandsgleichungen", Zeitbereich,
27 %            Initialisierung, Optionen)
28 t = T;
29 x = X(:,1);
30 y = X(:,2);
31 z = X(:,3);
32
33 % Zusätzliche Farben
34 colour_orange   = [ 246  143 15  ] ./ 255;
35 colour_rot      = [ 181  12  0   ] ./ 255;
36 colour_blau     = [ 0    32  162 ] ./ 255;
37 colour_gruen    = [ 34   139 34  ] ./ 255;
38
39 % 3D-Darstellung
40 figure(1);
41 plot3(x,y,z,'Color',colour_gruen);
42 axis equal;
43 grid;
44 set(figure(1),'Color',[1 1 1]);
45 ax=gca;ax.XTick=-25:5:25;ax.YTick=-25:5:25;ax.ZTick=0:5:50;
46 view(40,30);
47 axis([-25 25 -25 25 0 50]);
48 daspect([1 1 1]);
49 set(figure(1),'Units','points','Position',[0 0 600 600]);
50 xlabel('x'); ylabel('y'); zlabel('z');
```

```
51 % x/z-Trajektion
52 figure(2);
53 hold on;
54     plot(x,z,'Color',colour_orange);
55 hold off;
56 grid;
57 set(figure(2),'Color',[1 1 1]);
58 ax=gca;ax.XTick=-25:5:25;ax.YTick=0:5:50;
59 daspect([1 1 1]);
60 set(figure(2),'Units','points','Position',[0 0 600 600]);
61 xlabel('x'); ylabel('z');
62 axis([-25 25 0 50]);
63
64 % x/y-Trajektion
65 figure(3);
66 hold on;
67     plot(x,y,'Color',colour_orange);
68 hold off;
69 grid;
70 set(figure(3),'Color',[1 1 1]);
71 ax=gca;ax.XTick=-25:5:25;ax.YTick=-25:5:25;
72 daspect([1 1 1]);
73 set(figure(3),'Units','points','Position',[0 0 600 600]);
74 xlabel('x'); ylabel('y');
75 axis([-25 25 -25 25]);
76
77 % y/z-Trajektion
78 figure(4);
79 hold on;
80     plot(y,z,'Color',colour_orange);
81 hold off;
82 grid;
83 set(figure(4),'Color',[1 1 1]);
84 ax=gca;ax.XTick=-25:5:25;ax.YTick=0:5:50;
85 daspect([1 1 1]);
86 set(figure(4),'Units','points','Position',[0 0 600 600]);
87 xlabel('y'); ylabel('z');
88 axis([-25 25 0 50]);
```

```
89  % Zeitverlauf
90  figure(5);
91  hold on;
92      plot(t,x,'Color',colour_rot);
93      plot(t,y,'Color',colour_blau);
94      plot(t,z,'Color',colour_gruen);
95  hold off;
96  grid;
97  set(figure(5),'Color',[1 1 1]);
98  ax=gca;ax.XTick=0:5:50;ax.YTick=-25:5:50;
99  daspect([1 3 3]);
100 set(figure(5),'Units','points','Position',[0 0 800 400]);
101 xlabel('t'); ylabel('x_r_o_t y_b_l_a_u z_g_r_ü_n');
102 axis([0 50 -25 50]);
103
104 % Ausgabe der berechneten Werte in eine Datei
105 % (nur alle 150 Werte)
106 fid = fopen('Lorenz_Attraktor_matlab_werte.txt','wt');
107 for i=1:150:length(x)
108     fprintf(fid,'%e,%e,%e,%e\n',x(i),y(i),z(i),length(x));
109 end
110 fclose(fid);
111
112 return
113  end
114
115 % Zustandsgleichhungen
116 function dx = F(~, X, a, b, c)
117     dx = zeros(3,1);
118     dx(1) = a*(X(2) - X(1));
119     dx(2) = X(1)*(b - X(3)) - X(2);
120     dx(3) = X(1)*X(2) - c*X(3);
121     return
122 end
```

Abbildung 4.33: Matlab-Code zur Berechnung & Simulation des Lorenz-Attraktors (Matlab)

Das Ergebnis von Matlab liefert folgende Zeitverläufe und Trajektionen:

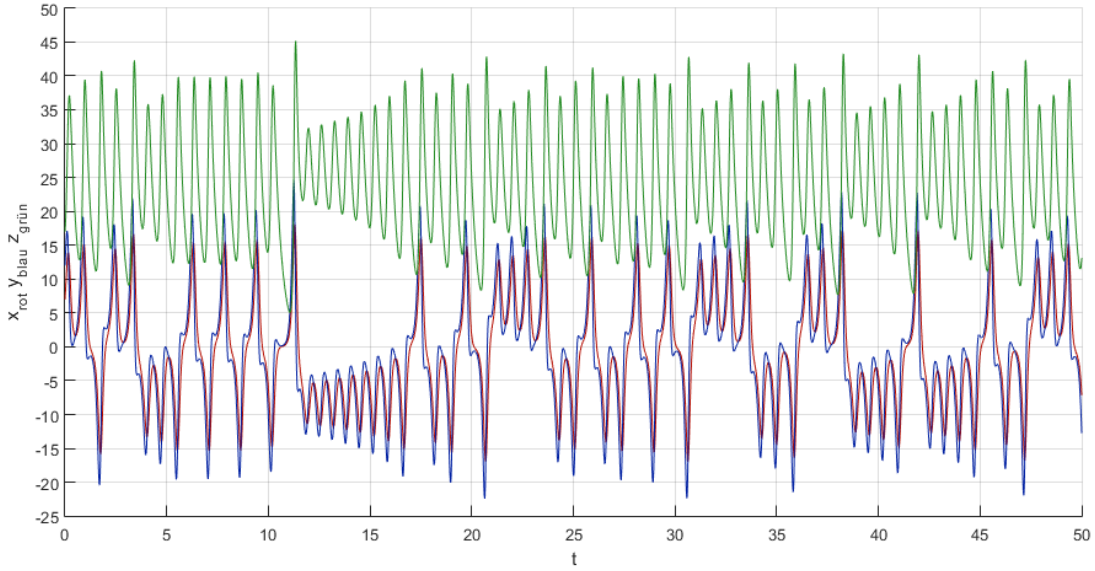


Abbildung 4.34: Zeitverläufe der Zustandsgrößen x, y & z beim Lorenz-Attraktor (Matlab)

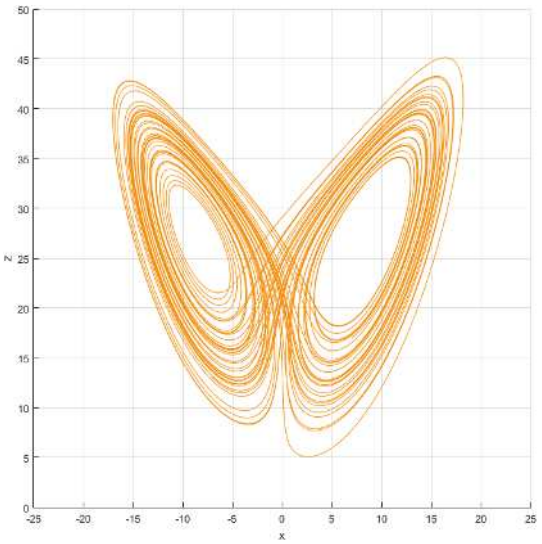


Abbildung 4.35: x/z-Trajektion des Lorenz-Attraktors (Matlab)

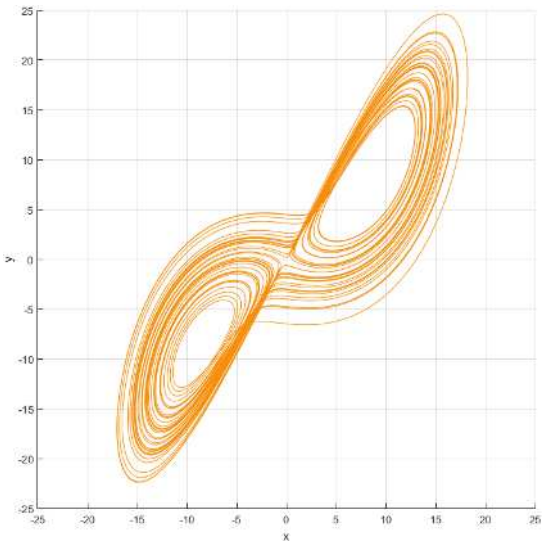


Abbildung 4.36: x/y-Trajektion des Lorenz-Attraktors (Matlab)

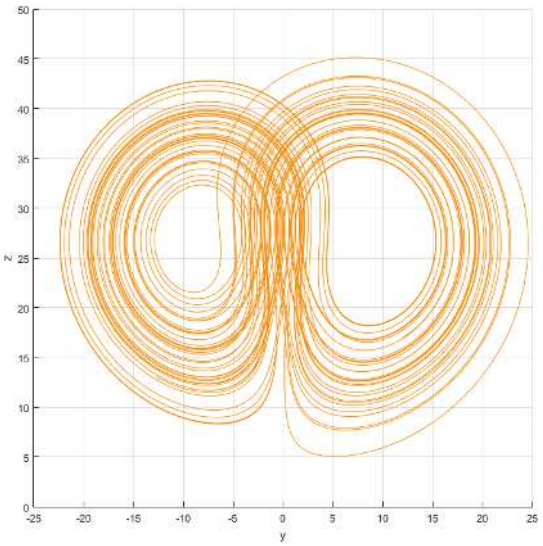


Abbildung 4.37: y/z-Trajektion des Lorenz-Attraktors (Matlab)

Das Ergebnis der Simulation mit den Parametern 4.2 und einer Simulationszeit von 50 sec lässt sich mit dem Simulationsergebnis im Kap. 4.4.1 (Maxima) sehr gut vergleichen.

In der dreidimensionalen Darstellung ist erneut das chaotische System, welches um die beiden Gleichgewichtspunkte P_1 & P_2 schwingt zu erkennen, und je näher der Verlauf zu einem der Punkte kommt, desto länger bleibt er in diesen Flügel.

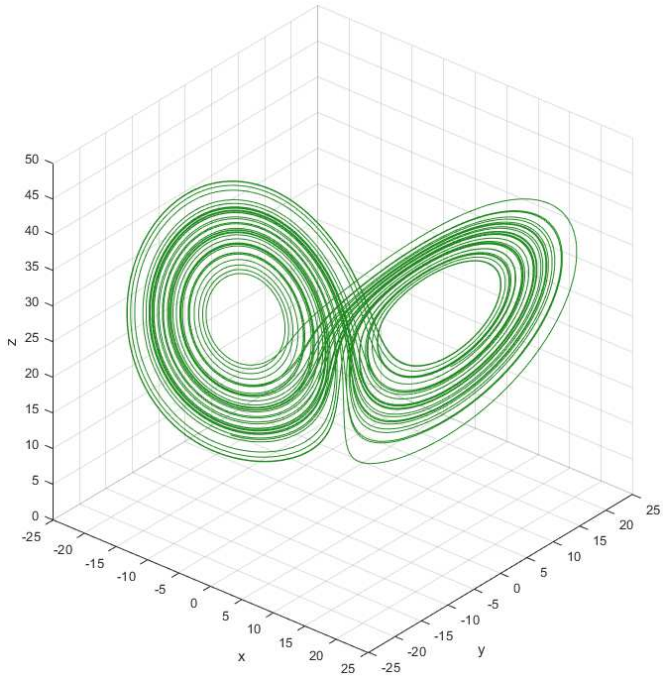


Abbildung 4.38: Trajektorie des Lorenz-Attraktors im dreidimensionalen Zustandsraum (Matlab)

4.5 Gegenüberstellung

Hier werden alle verschiedenen Simulationsergebnisse der einzelnen Programme gegenübergestellt. Das nachfolgende Maxima-Programm ist eine Weiterführung vom Programm im Abschnitt 4.4.1.

Einlesen der Werte von Java:

```
(%i64)  werte_java:read_nested_list("C:\\Users\\User\\Desktop
        \\Schule\\Laboratorium-5AHET\\04_Simulation
        \\01_Lorenz_Attraktor
        \\Lorenz_Attraktor_java_werte.csv",comma)$
        t_werte_java:map(first,werte_java)$
        x_werte_java:map(second,werte_java)$
        y_werte_java:map(third,werte_java)$
        z_werte_java:map(fourth,werte_java)$
```

Einlesen der Werte von Scilab (Xcos):

```
(%i69)  t_werte_xcos:read_list("C:\\Users\\User\\Desktop
        \\Schule\\Laboratorium-5AHET\\04_Simulation
        \\01_Lorenz_Attraktor
        \\Lorenz_Attraktor_xcos_time.csv")$
        werte_xcos:read_nested_list("C:\\Users\\User\\Desktop
        \\Schule\\Laboratorium-5AHET\\04_Simulation
        \\01_Lorenz_Attraktor
        \\Lorenz_Attraktor_xcos_values.csv",comma)$
        x_werte_xcos:map(first,werte_xcos)$
        y_werte_xcos:map(second,werte_xcos)$
        z_werte_xcos:map(third,werte_xcos)$
```

Einlesen der Werte von C:

```
(%i74)  werte_c:read_nested_list("C:\\Users\\User\\Desktop\\Schule
        \\Laboratorium-5AHET\\04_Simulation\\01_Lorenz_Attraktor
        \\Lorenz_Attraktor_c_werte.csv",comma)$
        x_werte_c:map(first,werte_c)$
        y_werte_c:map(second,werte_c)$
        z_werte_c:map(third,werte_c)$
        t_werte_c:map(fourth,werte_c)$
```

Einlesen der Werte von Matlab (Simulink):

```
(%i79)  werte_matlab:read_nested_list("C:\\Users\\User\\Desktop
        \\Schule\\Laboratorium-5AHET\\04_Simulation
        \\01_Lorenz_Attraktor
        \\Lorenz_Attraktor_matlab_werte.csv",comma)$
        x_werte_matlab:map(first,werte_matlab)$
        y_werte_matlab:map(second,werte_matlab)$
        z_werte_matlab:map(third,werte_matlab)$
        t_werte_matlab:map(fourth,werte_matlab)$
```

Einlesen der Werte von LaTeX:

```
(%i84)  werte_latex:read_nested_list("C:\\Users\\User\\Desktop
        \\LaTeX-Laborprotokol-5AHET\\lorenz_latex.dat",comma)$
        x_werte_latex:map(first,werte_latex)$
        y_werte_latex:map(second,werte_latex)$
        z_werte_latex:map(third,werte_latex)$
        t_werte_latex:map(fourth,werte_latex)$
```

```
(%i85)  wxplot_size:[1000,500]$
(%i86)  wxdraw2d(key="Maxima",color=red,xrange=[0,10],
                 points(t_werte_maxima,y_werte_maxima),
                 key="Java",color=orange,yrange=[-35,35],
                 points(t_werte_java,y_werte_java),
                 key="Xcos",color=black,xaxis=true,
                 points(t_werte_xcos,y_werte_xcos),
                 key="C",color=blue,ytics=5,ylabel="y",
                 points(t_werte_c,y_werte_c),
                 key="Matlab",color=forest-green,
                 points(t_werte_matlab,y_werte_matlab),
                 key="LaTeX",color=gray,xlabel="t",
                 points(t_werte_latex,y_werte_latex),xtics=1,
                 user_preamble="set key bottom left");
```

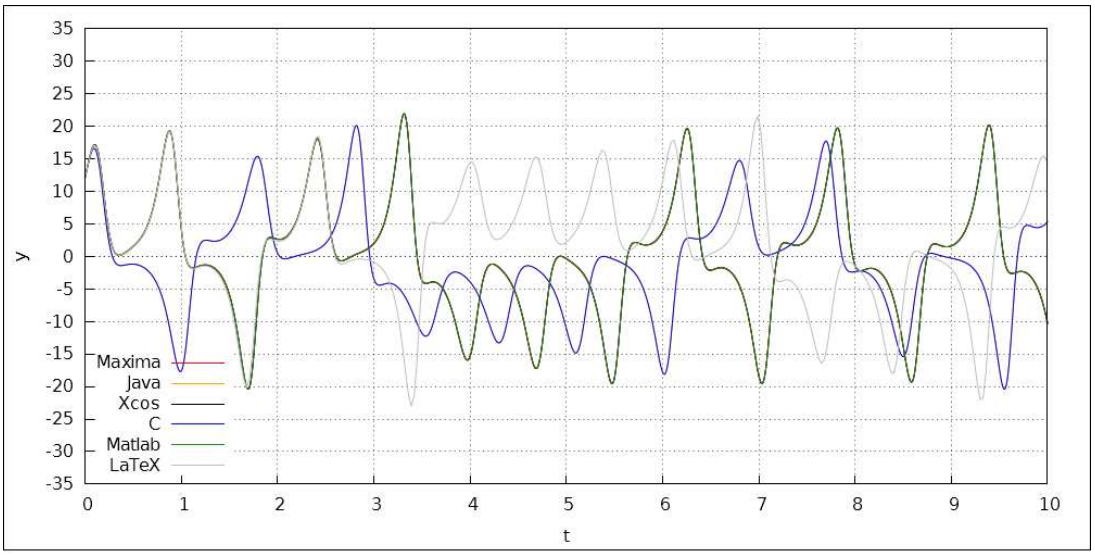


Abbildung 4.39: Zeitverläufe der Zustandsgrößen im Vergleich 1 beim Lorenz-Attraktor (Java, C, LaTeX, Scilab (Xcos), Maxima, Matlab (Simulink))

(%t86) (%o86)

Es lässt sich erkennen, dass prinzipiell sofort am Beginn Abweichungen zwischen Maxima-Matlab-LaTeX-Scilab & Java-C entstehen. Erst nach ungefähr 3 sec weicht LaTeX von Maxima-Matlab ab.

```
(%i87)  wxdraw2d(key="Maxima",    color=red,
         points(t_werte_maxima,y_werte_maxima),
                key="Java",      color=orange,
                points(t_werte_java,y_werte_java),
                key="Xcos",      color=black,
                points(t_werte_xcos,y_werte_xcos),
                key="C",         color=blue,
                points(t_werte_c,y_werte_c),
                key="Matlab",    color=forest-green,
                points(t_werte_matlab,y_werte_matlab),
                key="LaTeX",     color=gray,
                points(t_werte_latex,y_werte_latex),ytics=5,
                yrange=[-35,35],xrange=[10,20], xlabel="t",
                user_preamble="set key bottom left",ylabel="y",
                xaxis=true,dimensions=[1500,600],xtics=1);
```

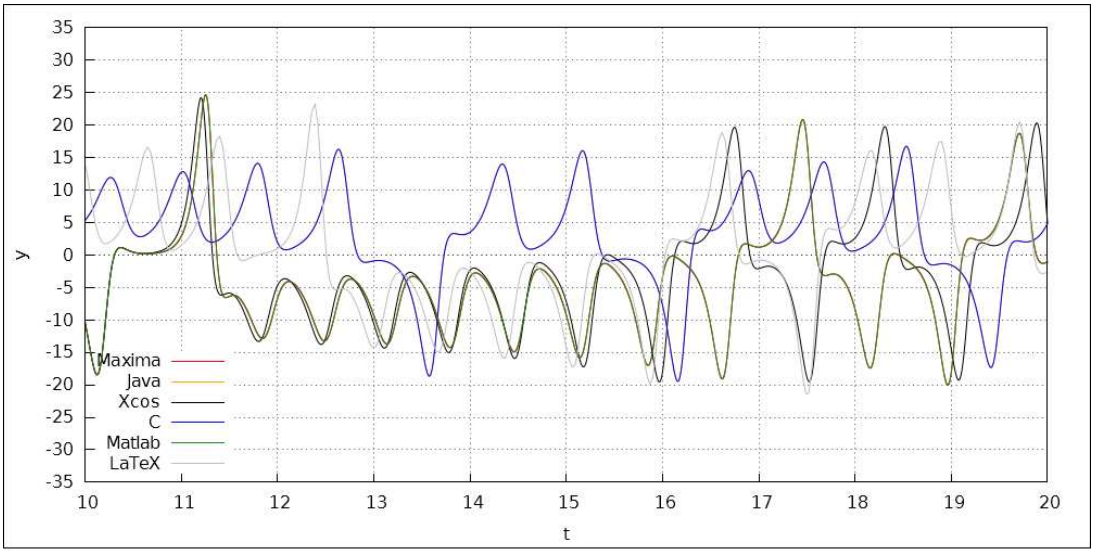


Abbildung 4.40: Zeitverläufe der Zustandsgrößen im Vergleich 2 beim Lorenz-Attraktorr (Java, C, LaTeX, Scilab (Xcos), Maxima, Matlab (Simulink))

(%t88) (%o88)

In der Abb. 4.40 lässt sich erkennen, dass ab 10 sec Scilab von Maxima-Matlab beginnt abzuweichen und nach 16 sec befinden sich die Verläufe erstmals in unterschiedlichen Flügeln.

```
(%i88)  wxdraw2d(key="Maxima",    color=red,
         points(t_werte_maxima,y_werte_maxima),
                key="Java",      color=orange,
                points(t_werte_java,y_werte_java),
                key="Xcos",      color=black,
                points(t_werte_xcos,y_werte_xcos),
                key="C",         color=blue,
                points(t_werte_c,y_werte_c),
                key="Matlab",    color=forest-green,
                points(t_werte_matlab,y_werte_matlab),
                key="LaTeX",     color=gray,
                points(t_werte_latex,y_werte_latex),ytics=5,
                yrange=[-35,35],xrange=[25,35], xlabel="t",
                user_preamble="set key top left",ylabel="y",
                xaxis=true,dimensions=[1500,600],xtics=1);
```

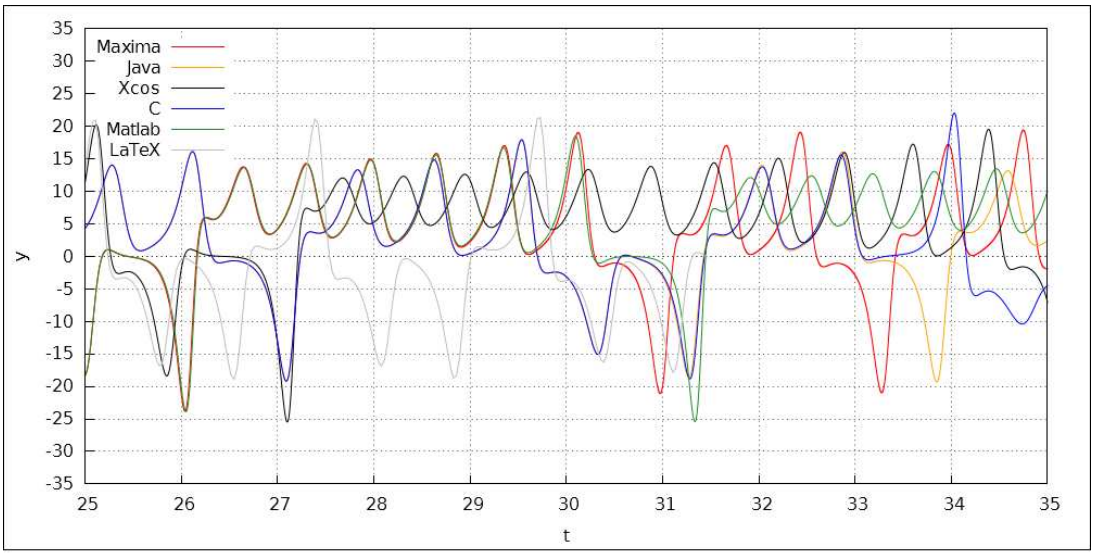


Abbildung 4.41: Zeitverläufe der Zustandsgrößen im Vergleich 3 beim Lorenz-Attraktor (Java, C, LaTeX, Scilab (Xcos), Maxima, Matlab (Simulink))

(%t88) (%o88)

Im weiteren Verlauf zeigt sich, dass Maxima und Matlab voneinander nach ca. 30 sec starkt, aber auch schon früher, voneinander abweichen und nach 33 sec zeigen auch Java und C unterschiedliche Verläufe.

5 Van-der-Pol-Oszillator

5.1 Grundlagen

Der Van-der-Pol-Oszillator beschreibt Schwingungen von diversen selbsterregten Systemen mit einer nichtlinearen Dämpfung. Dieser Attraktor lässt sich mit Hilfe folgender Differntialgleichung beschreiben,

$$\ddot{x} - \epsilon \left(1 - x^2\right) \dot{x} + x = 0 \qquad \text{mit } \epsilon > 0 \tag{5.1}$$

und stellt eine mathematische Beschreibung eines Röhrenoszillators dar, da er das Verhalten der elektrischen Ladung, welche durch eine Triodenröhre fließt, beschreibt.

Man kann erkennen, dass bei Amplituden, wo $x^2 < 1$ ist, eine negative Dämpfung vorliegt und sich so die Amplitude vergrößert und danach stabilisiert sich das System, auf Grund der positiven, auf einen gewissen Wert anwachsenden, Dämpfung, auf einen gewissen Verlauf. Somit ist auch der homogene Van-der-Pol-Oszillator kein chaotisches System.

Umgeformt auf nichtlineare, gewöhnliche Differentialgleichungen 1. Ordnung ergeben sich zwei Gleichungen, wobei als Zwischengröße $y = \dot{x}$ eingeführt wurde:

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= -x + \epsilon \left(1 - x^2\right) y \end{aligned} \tag{5.2}$$

$x,\ y$	Zustandsgrößen
ϵ	Systemparameter (Maß für die Dämpfung)
t	Zeit

Die Parameterwahl, welche in unserem Projekt zum Vergleich der Simulationen benutzt wurde, beträgt:

$$\epsilon = 0.5 \tag{5.3}$$

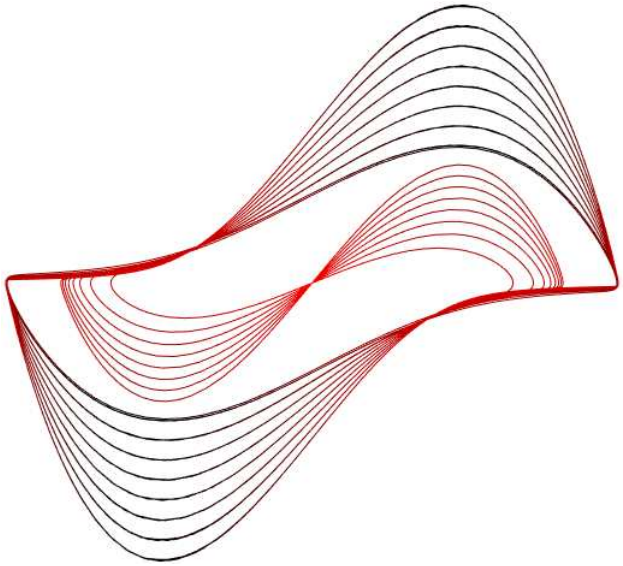


Abbildung 5.1: Van-der-Pol-Oszillatoren mit unterschiedlichen Anfangsbedingungen & Systemparametern ϵ simuliert mit dem Dormand-Prince-Verfahren in Matlab

Als Anfangswerte werden fast ausschließlich im gesamten Projekt folgende Werte verwendet:

$$x(0) = 0.001 \qquad y(0) = 0.001 \tag{5.4}$$

5.2 Simulation mit Differenzengleichungen

5.2.1 Java

Das nachfolgende Java-Programm, welches für diesen Versuch entwickelt wurde, dient lediglich zur Berechnung des Van-der-Pol-Oszillators und nicht zur Darstellung, den diese erfolgt im späteren Verlauf bei der Gegenüberstellung in Maxima.

```
1  package simulationen;
2
3  /** Simulation des Van_der_Pol_Oszillator */
4  public class Van_der_Pol_Oszillator {
5    /*  Main-Methode  */
6    public static void main(String[] args) {
7      /*  Calculation */
8      Simulation Sim_1 =
9          new Simulation(0.5,0.001,0.001,0,50,0.0001);
10     Sim_1.calculation();
11     /*  Output  */
12     Sim_1.write_to_txt(
13         "Van_der_Pol_Oszillator_java_werte.txt", 10);
14   }
15 }
```

Abbildung 5.2: Java-Code zur Simulation des Van-der-Pol-Oszillators (Java)

```
1  package simulationen;
2  /*  Import der Bibliotheken */
3  import java.io.BufferedWriter;
4  import java.io.FileWriter;
5  import java.io.IOException;
```

Abbildung 5.3: Java-Code (Imports) zur Berechnung des Van-der-Pol-Oszillators (Java)

```
6  public class Simulation {
7    /*  Variablen der Klasse Simulation */
8    private double  e,x0,y0,t_begin,t_end,dt;
9    private int quantity;
10   private double [] t; private double [] x;
11   private double [] y;
12   /**Legt die Startwerte und Parameter der Simulation fest
13    * @param e        = Konstante e
14    * @param x_init    = Startwert x
15    * @param y_init    = Startwert y
16    * @param t_begin_sim   = Simulationsbegin
17    * @param t_end_sim   = Simulationsende
18    * @param increment_sim = Schrittweite */
19   public Simulation(
20       double e_constant, double x_init, double y_init,
21       double t_begin_sim, double t_end_sim, double increment_sim)
22     {
23     e=e_constant;
24     x0=x_init;
25     y0=y_init;
26     t_begin=t_begin_sim;
27     t_end=t_end_sim;
28     dt=increment_sim;
29     quantity=(int)((t_end-t_begin)/dt);
30   }
31   /**Berechnet die x- und y-Werte und übergibt die x,y-
32    * und t-Werte einem Array */
33   public void calculation(){
34     int i=0;
35     t = new double [quantity+2];
36     x = new double [quantity+2];
37     y = new double [quantity+2];
38     x[i]=x0;y[i]=y0;t[i]=t_begin;
39     for(double t_zaehler=t_begin;t_zaehler<=t_end;
40         t_zaehler+=dt){i++;
41       double dx=y[i-1]*dt;
42       double dy=(-x[i-1]-e*(x[i-1]*x[i-1]-1)*y[i-1])*dt;
43       x[i]=x[i-1]+dx;
44       y[i]=y[i-1]+dy;
45       t[i]=t_zaehler+dt;
46     }
47   }
```

```
48    /**Gibt die t-Werte zurück
49     * @return [] t-Werte */
50    public double[] get_t_werte(){return t;}
51    /**Gibt die x-Werte zurück
52     * @return [] x-Werte */
53    public double[] get_x_werte(){return x;}
54    /**Gibt die y-Werte zurück
55     * @return [] y-Werte */
56    public double[] get_y_werte(){return y;}
57    /**Schreibt die Array-Werte x,y,z und t in ein txt-File
58     * @param path_write_to_txt = Pfad
59     * @param anz = Anzahl der Schrittweite der Ausgabe */
60    public void write_to_txt(String path_write_to_txt, int anz){
61      String path = path_write_to_txt;
62          String arrayString = "";
63          int anzahl=anz;
64          for (int i=0; i<t.length; i+=anzahl)
65          {
66              arrayString = arrayString + t[i]+" , " + x[i]+" , "
67              + y[i]+" , " +"\n";
68              // Fortschrittsanzeige (nur zur Übersichtlichkeit
69              // beim Buildn)
70            System.out.println((double)i*100/(t.length)+" %");
71          } System.out.println(100+" %");
72          try {
73              BufferedWriter out =
74                 new BufferedWriter(new FileWriter(path));
75              out.write(arrayString);
76              out.close();
77          } catch (IOException e)
78          {
79            System.out.println("Fehler beim Schreiben des Stringes
80                                in die Datei. Überprüfen Sie bitte
81                                den angegebenen Pfad.");
82          }
83    }
84  }
```

Abbildung 5.4: Java-Code zur Berechnung des Van-der-Pol-Oszillators (Java)

5.2.2 C

Auch beim C-Programm erfolgt die Darstellung erst in Maxima im Kap. 5.5 bei der Gegenüberstellung. Der nachfolgende Code dient somit nur zur Berechnung des Oszillators.

```
1 /*  Van_der_Pol_berechnung.c
2  *  Erstellungsdatum: 25.11.2015
3  *  Autor: Labenbacher Michael */
4 #include <stdlib.h>
5 #include <math.h>
6 #include <stdio.h>
```

Abbildung 5.5: C-Code (Include-Dateien) zur Berechnung des Van-der-PolOszillators (C)

```
7  /*  ========================= Variables =========================
8   *  States: x, y,
9   *  Constants: e
10  *  Domain: dependent variable,
11  *      begin-, end of simulation and increment
12  *  Arrays: x, y, t
13  *  File: txt-Datei */
14 volatile long double e;
15 double t_begin=0,t_end=50,dt=0.0001;
16 volatile long double x [500000];
17 volatile long double y [500000];
18 volatile long double t [500000];
19 FILE *datei;
20
21 /*  ========================= Definitions =======================
22  *  of functions:
23  *    init --> Initialisation
24  *    simulation --> Simulation (Calculation) */
25 void init(long double x_init, long double y_init,
26           long double e_init){
27   x[0]=x_init;
28   y[0]=y_init;
29   e=e_init;
30 }
```

```c
void simulation(){
  t[0]=t_begin;
  int i = 0;
  long double t_zaehler;
  //  for(<initialisation>;<conditions>;<update>)
  for(t_zaehler=t_begin; t_zaehler<=t_end; t_zaehler+=dt){i++;
    t[i]=t[i-1]+dt;
    long double dx=y[i-1]*dt;
    long double dy=(-x[i-1]-e*(x[i-1]*x[i-1]-1)*y[i-1])*dt;
    x[i]=x[i-1]+dx;
    y[i]=y[i-1]+dy;
  }
}

int write_to_txt(){
  datei = fopen("Van_der_Pol_Oszillator_c_werte.txt", "w");
  if(NULL == datei) {
      printf("Konnte Datei \"Van_der_Pol_Oszillator_c_werte.txt\"
              nicht öffnen!\n");
      return 1;
  }
  int i;
  for(i=0;i<500000;i+=500000/5000){
    fprintf(datei, "%e,%e,%e\n",
       (double)x[i], (double)y[i], (double)t[i]);
  }
  fclose(datei);
  return 0;
}

int main(void){
  init(0.001,0.001,0.5);
  simulation();
  write_to_txt();
  return 0;
}
```

Abbildung 5.6: C-Code zur Berechnung des des Van-der-Pol-Oszillators (C)

5.3 Simulation mit Funktionsblöcken

5.3.1 Scilab (Xcos)

In Scilab, bzw. im Simulationsaufsatz Xcos, kann nun das dazugehörige Blockschaltbild angefertigt werden und die Einstellungen in den einzelnen Bauelementen, wie z. B. den Anfangswerten, können gesetzt werden.

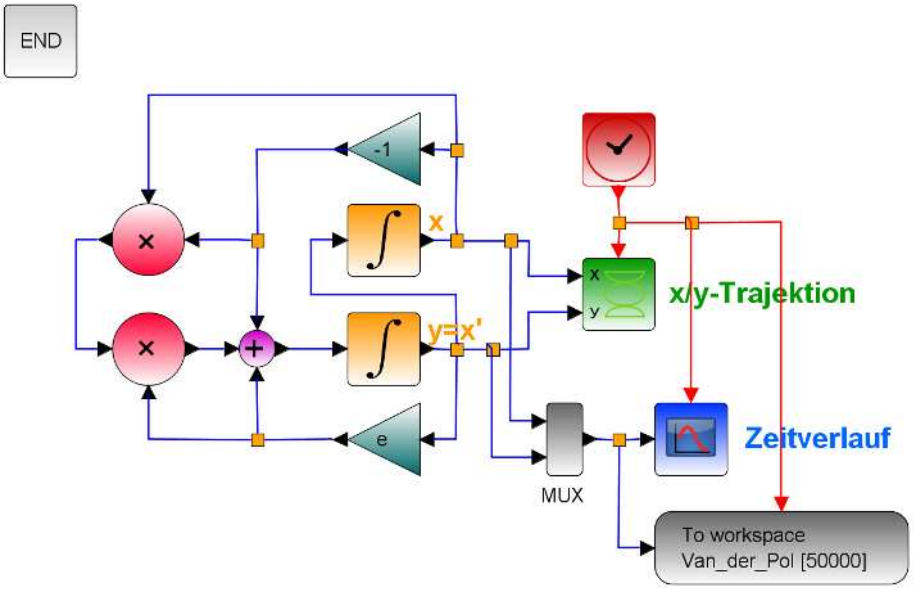


Abbildung 5.7: Blockschaltbild des Van-der-Pol-Oszillators (Scilab (Xcos))

Dieses Programm wurde so entwickelt, dass eine x/y-Trajektion in einem eigenen Frame dargestellt wird. Des Weiteren wird der Zeitverlauf angefertigt und nun muss noch für die Simulation in Scilab der Parameter des Van-der-Pol-Oszillators eingegeben werden:

```
e=0.5;
```

Mit folgender Eingabe in Scilab können die fertig simulierten Werte in eine csv-Datei geschrieben werden:

```
write_csv(Van_der_Pol.time,
        'Van_der_Pol_Oszillator_xcos_time.csv')
write_csv(Van_der_Pol.values,
        'Van_der_Pol_Oszillator_xcos_values.csv')
```

Als Gleichungslöser wurde bei Scilab (Xcos) *Sundials/CVODE - BDF - NEWTON* verwendet, wobei insgesamt 50 000 Punkten im Zeitbereich von $0 - 50$ berechnet wurden.

Parameter	Einstellung
Finale Integrationszeit	$1.0E03$
Echt-Zeit-Skalierung	$0,0E00$
Absolute Toleranz des Integrators	$1,0E-10$
Relative Toleranz des Integrators	$1,0E-10$
Zeit-Toleranz	$1,0E-12$
Maximales Zeitintervall der Integration	$1,00001E05$
Gleichungslöser	Sundials/CVODE - BDF - NEWTON
Maximale Schrittweite (0 = kein Limit)	$0,0E00$

Tabelle 5.1: Parametereinstellungen in Scilab für den Van-der-Pol-Oszillator (Xcos)

Das Simulationsergebnis zeigt folgenden Verlauf der Zustandsgrößen x & y ($\dot{x}$):

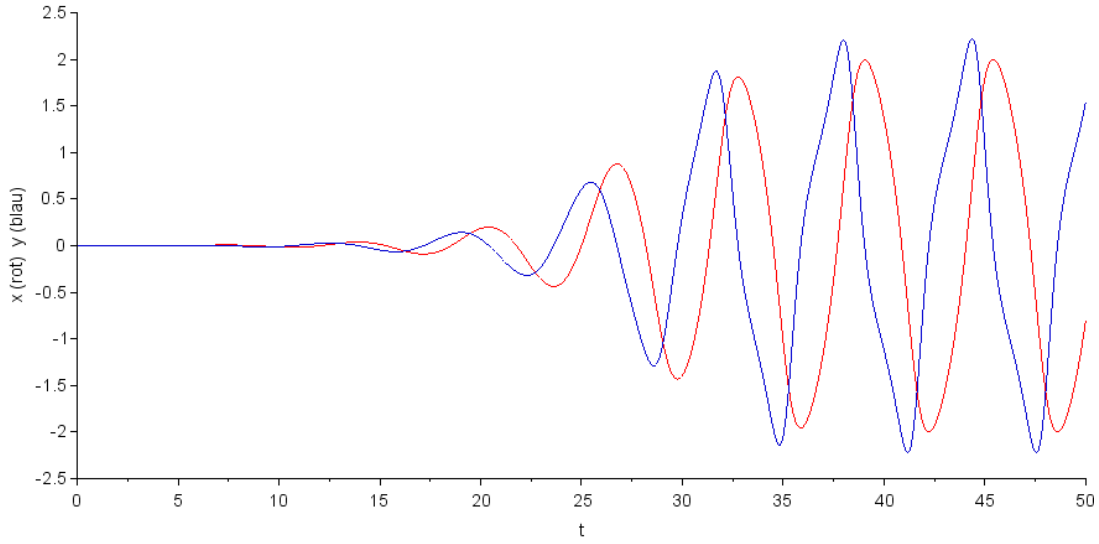


Abbildung 5.8: Zeitverläufe der Zustandsgrößen x, y beim Van-der-Pol-Oszillator (Scilab (Xcos))

Es lässt sich sehr deutlich erkennen, dass sich das System von selbst, wegen der negativen Dämpfung, aufschwingt und in eine „stationäre“ Bahn übergeht.

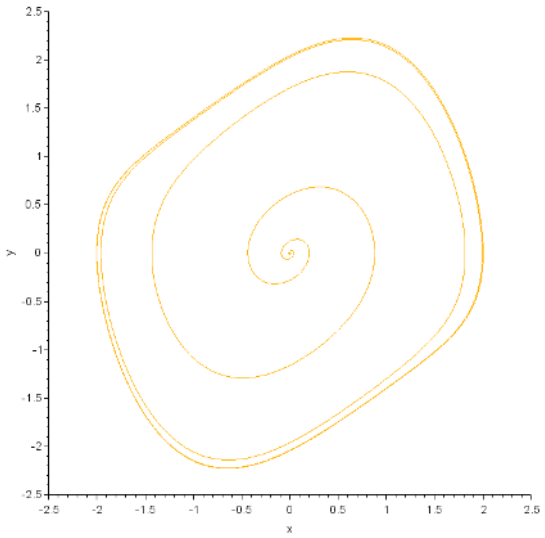


Abbildung 5.9: x/y-Trajektion des Van-der-Pol-Oszillators (Scilab (Xcos))

In der Abb. 5.8 ist auch erkennbar, dass, wenn y, sprich die Änderung der Änderung von x ein Maximum hat, $x = 0$ ist und wenn x ein Maximum hat, muss y somit gleich 0 sein.

Im linken Bild ist der Übergang, von den Anfangswerten ≈ 0, auf die stationäre Bahn, ersichtlich.

5.3.2 Matlab (Simulink)

Mittels Simulink kann das Ganze ebenfalls als Blockschaltbild aufgebaut und simuliert werden, was folgendes Bild in der slx-Datei ergibt:

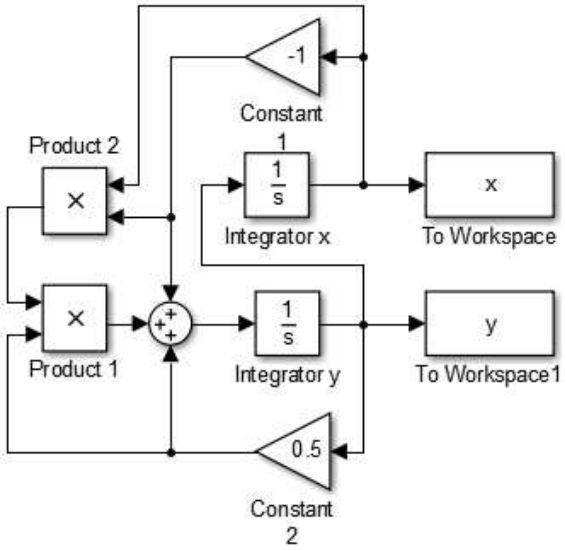


Abbildung 5.10: Blockschaltbild des Van-der-Pol-Oszillators (Matlab (Simulink))

Das Ergebnis der Simulation liefert die x-, y- & t-Werte zurück, welche dann zur Anzeige gebracht werden können und das selbe Ergebnis, wie im Kapitel 5.4.2 liefern.

5.4 Simulation mit dem Runge-Kutta-Verfahren

5.4.1 Maxima

Es wurde dazu folgendes Maxima-Programm erstellt, wobei die Abhängigkeit von ϵ und von den Anfangsbedingungen untersucht wurde.

```
(%i1)   kill(all)$
```

Laden des *dynamics*- und *coma*(*draw*)-Paketes und setzen von Defaultwerten:

```
(%i3)   load(coma)$
        load(dynamics)$set_draw_defaults(grid=true,
           point_type=0,points_joined=true)$
```

coma v.1.73, (Wilhelm Haager, 2015-01-09)

Untersuchung der Abhängigkeit von ϵ

Funktionen

```
(%i5)   f1 : y$
        f2 : -x - e*(x^2-1)*y$
```

Zustandsgrößen:

```
(%i6)   states:[x,y]$
```

Konstanten:

```
(%i7)   constants:[e=0.2,e=1,e=2]$
```

Anfangsbedingungen:

```
(%i8)   initialisation:[0.001,0.001]$
```

Domäne:

```
(%i9)   domain:[t,0,100,0.001]$
```

Berechnung und Zerlegung: *rk* (functions, states, initialisation, domain)

```
(%i13)  res:makelist(0,k,1,3,1)$ t_werte_maxima:makelist(0,k,1,3,1)$
        x_werte_maxima:copylist(t_werte_maxima)$
        y_werte_maxima:copylist(t_werte_maxima)$
(%i14)  for i:1 while i<=3 do
            res[i]:rk([f1,ev(f2,constants[i])],
                      states,initialisation,domain)$
(%i15)  for i:1 while i<=3 do block(
        t_werte_maxima[i]:map(first,res[i]),x_werte_maxima[i]:
        map(second,res[i]),y_werte_maxima[i]:map(third,res[i]))$
```

Zeitverläufe:

```
(%i16)  wxplot_size:[800,400]$
(%i17)  wxdraw2d(color=red,yrange=[-5,5],ylabel="x(rot) / y(blau)",
                 points(t_werte_maxima[1],x_werte_maxima[1]),
                 color=navy,xrange=[0,domain[3]],xlabel="t"
                 points(t_werte_maxima[1],y_werte_maxima[1]),
                 xaxis=true,dimensions=[1500,600],ytics=1)$
```

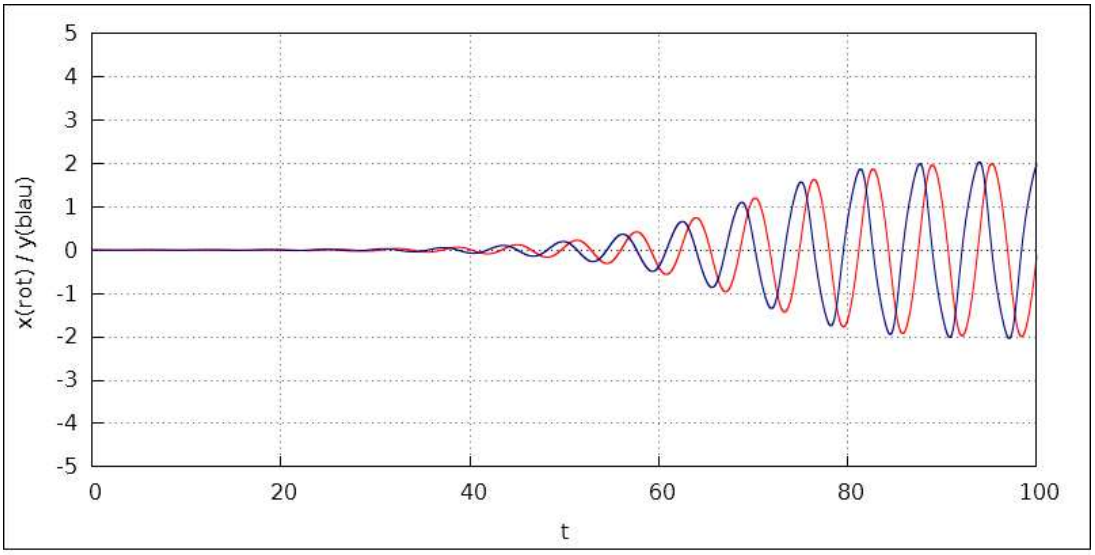


Abbildung 5.11: Zeitverläufe der Zustandsgrößen x & y beim Van-der-Pol-Oszillator bei $\epsilon = 0.2$ (Maxima)

(%t17)

```
(%i18)  wxdraw2d(color=red,yrange=[-5,5],ylabel="x(rot) / y(blau)"
                 points(t_werte_maxima[2],x_werte_maxima[2]),
                 color=navy,xrange=[0,domain[3]/2],xlabel="t"
                 points(t_werte_maxima[2],y_werte_maxima[2]),
                 xaxis=true,dimensions=[1500,600],ytics=1,)$
```

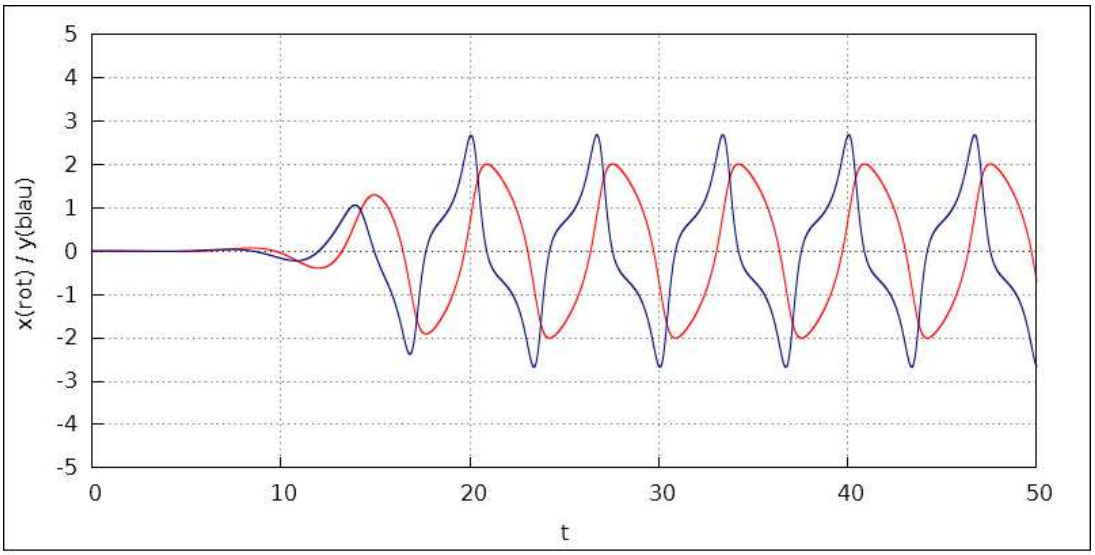


Abbildung 5.12: Zeitverläufe der Zustandsgrößen x & y beim Van-der-Pol-Oszillator bei $\epsilon = 1$ (Maxima)

(%t18)

In diesem Bild ist wieder zu erkennen, dass, wenn bei einer Größe ein Maximum bzw. Minimum vorliegt, die andere Größe gleich 0 ist.
Des Weiteren ist erkennbar, dass die Periodendauer der Schwingung mit steigendem ϵ zunimmt und die Schwingung immer anharmonischer wird.

```
(%i19)  wxdraw2d(color=red,yrange=[-5,5],ylabel="x(rot) / y(blau)"
                 points(t_werte_maxima[3],x_werte_maxima[3]),
                 color=navy,xrange=[0,domain[3]/2],xlabel="t"
                 points(t_werte_maxima[3],y_werte_maxima[3]),
                 xaxis=true,dimensions=[1500,600],ytics=1,)$
```

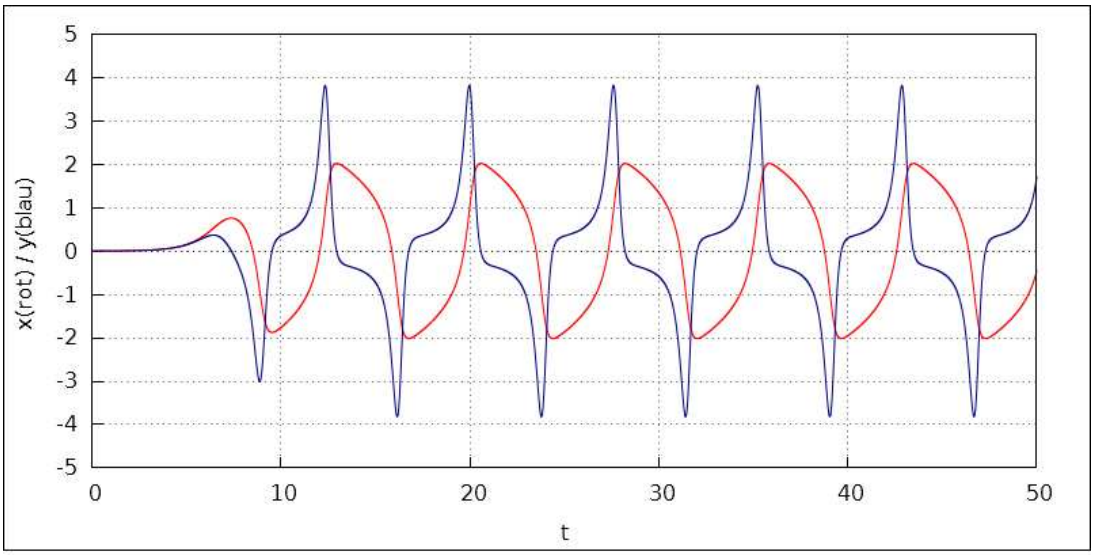


Abbildung 5.13: Zeitverläufe der Zustandsgrößen x & y beim Van-der-Pol-Oszillator bei $\epsilon = 2$ (Maxima)

(%t19)

Trajektorie in der x/y-Phasenebene:

```
(%i20)  wxplot_size:[600,600]$
(%i21)  wxdraw2d(color=orange,yrange=[-4,4],xrange=[-4,4],
                 points(x_werte_maxima[1],y_werte_maxima[1]),
                 xaxis=true,yaxis=true,xlabel="x",ylabel="y",
                 user_preamble="set size ratio -1")$
(%i22)  wxdraw2d(color=orange,yrange=[-4,4],xrange=[-4,4],
                 points(x_werte_maxima[2],y_werte_maxima[2]),
                 xaxis=true,yaxis=true,xlabel="x",ylabel="y",
                 user_preamble="set size ratio -1")$
```

(%t21) (%t22) (%t23)

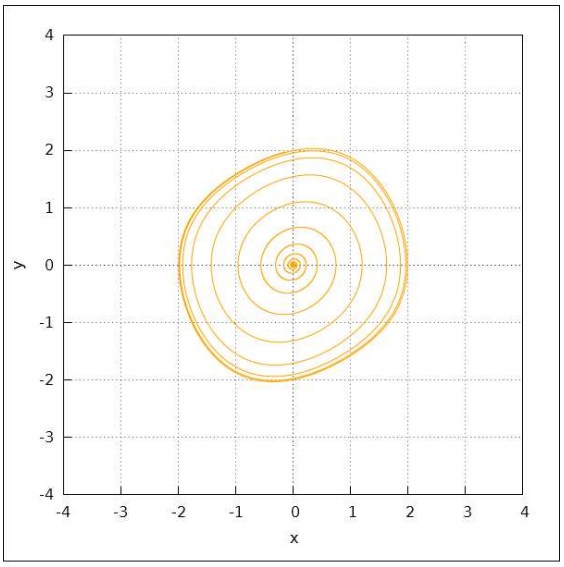


Abbildung 5.14: x/y-Trajektion des Van-der-Pol-Oszillators bei $\epsilon = 0.2$ (Maxima)

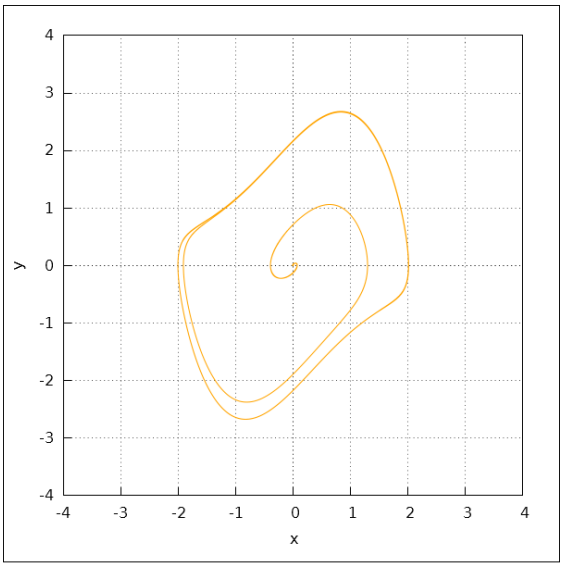


Abbildung 5.15: x/y-Trajektion des Van-der-Pol-Oszillators bei $\epsilon = 1$ (Maxima)

```
(%i23) wxdraw2d(color=orange,yrange=[-4,4],xrange=[-4,4],yaxis=true
,points(x_werte_maxima[3],y_werte_maxima[3]),xaxis=true,
xlabel="x",ylabel="y",user_preamble="set size ratio -1")$
```

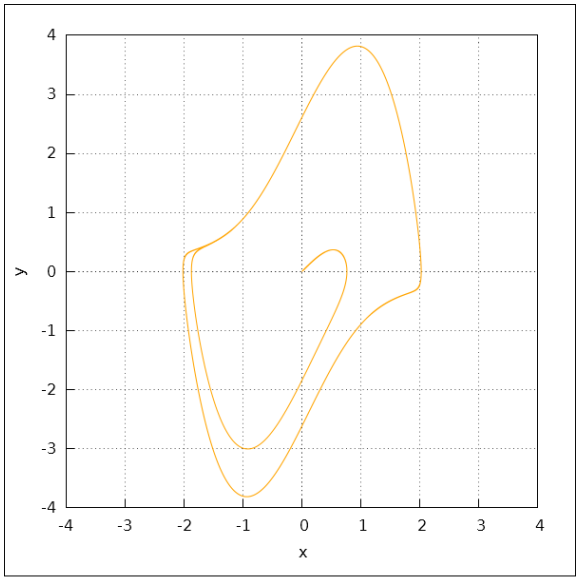


Abbildung 5.16: x/y-Trajektion des Van-der-Pol-Oszillators bei $\epsilon = 2$ (Maxima)

Untersuchung der Abhängigkeit von den Anfangsbedingungen

Laden des *dynamics*- und *coma*(*draw*)-Paketes und setzen von Defaultwerten:

```
(%i3)   kill(all)$
        load(coma)$
        load(dynamics)$set_draw_defaults(grid=true,
           point_type=0,points_joined=true)$
```

coma v.1.73, (Wilhelm Haager, 2015-01-09)

```
(%i5)   f1 : y$
        f2 : -x - e*(x^2-1)*y$
```

Zustandsgrößen:

```
(%i6)   states:[x,y]$
```

Konstanten:

```
(%i7)   constants:[e=0.5]$
```

Anfangsbedingungen:

```
(%i8)   initialisation:makelist(0,k,1,3,1)$
(%i11)  initialisation[1]:[0.010,0.010]$
        initialisation[2]:[0.015,0.015]$
        initialisation[3]:[0.020,0.020]$
```

Domäne:

```
(%i12)  domain:[t,0,50,0.0001]$
```

Berechnung und Zerlegung:
rk (functions, states, initialisation, domain)

```
(%i16)  res:makelist(0,k,1,3,1)$
        t_werte_maxima:makelist(0,k,1,3,1)$
        x_werte_maxima:copylist(t_werte_maxima)$
        y_werte_maxima:copylist(t_werte_maxima)$
(%i17)  for i:1 while i<=3 do
             res[i]:rk([f1,ev(f2,constants)],
                       states,initialisation[i],domain)$
```

```
(%i18)  for i:1 while i<=3 do block(
        t_werte_maxima[i]:map(first,   res[i]),
        x_werte_maxima[i]:map(second,  res[i]),
        y_werte_maxima[i]:map(third,   res[i]))$
(%i19)  wxplot_size:[800,400]$
(%i20)  wxdraw2d(color=magenta,yrange=[-2.5,2.5],xrange=[0,30],
                 points(t_werte_maxima[1],x_werte_maxima[1]),
                 color=red,xaxis=true,dimensions=[1500,600],
                 points(t_werte_maxima[2],x_werte_maxima[2]),
                 color=black,ytics=0.5,xlabel="t",
                 points(t_werte_maxima[3],x_werte_maxima[3]),
                 ylabel="x_1(orange) / x_2(rot) /x_3(schwarz)")$
```

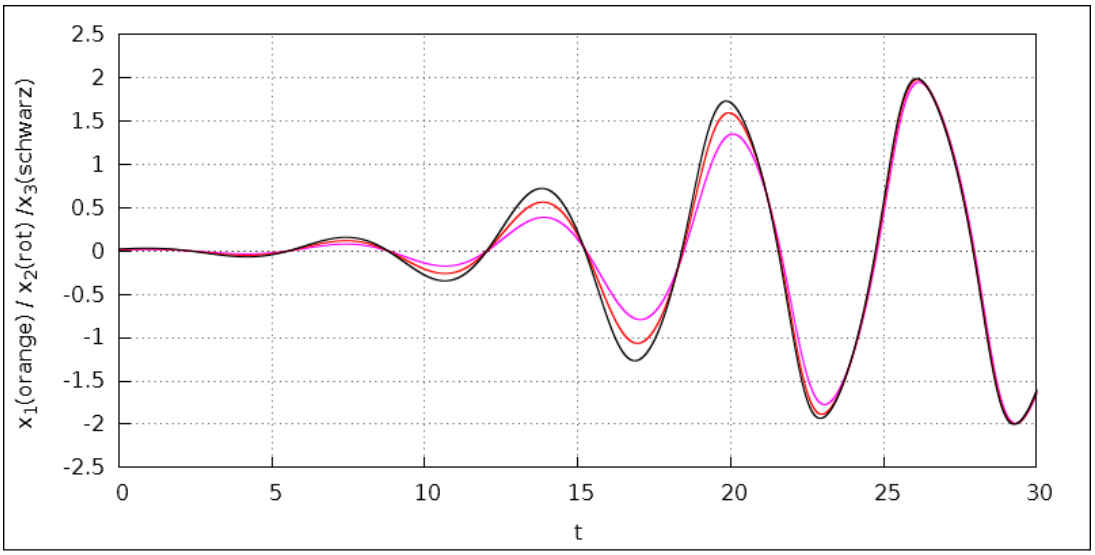


Abbildung 5.17: Zeitverläufe der Zustandsgröße x beim Van-der-Pol-Oszillator bei unterschiedlichen Anfangsbedingungen (Maxima)

(%t20)

```
(%i21)  wxdraw2d(color=forest-green,yrange=[-2.5,2.5],xrange=[0,30],
                 points(t_werte_maxima[1],y_werte_maxima[1]),
                 color=navy,xaxis=true,dimensions=[1500,600],
                 points(t_werte_maxima[2],y_werte_maxima[2]),
                 color=black,ytics=0.5,xlabel="t",
                 points(t_werte_maxima[3],y_werte_maxima[3]),
                 ylabel="y_1(grün) / y_2(blau) /y_3(schwarz) " )$
```

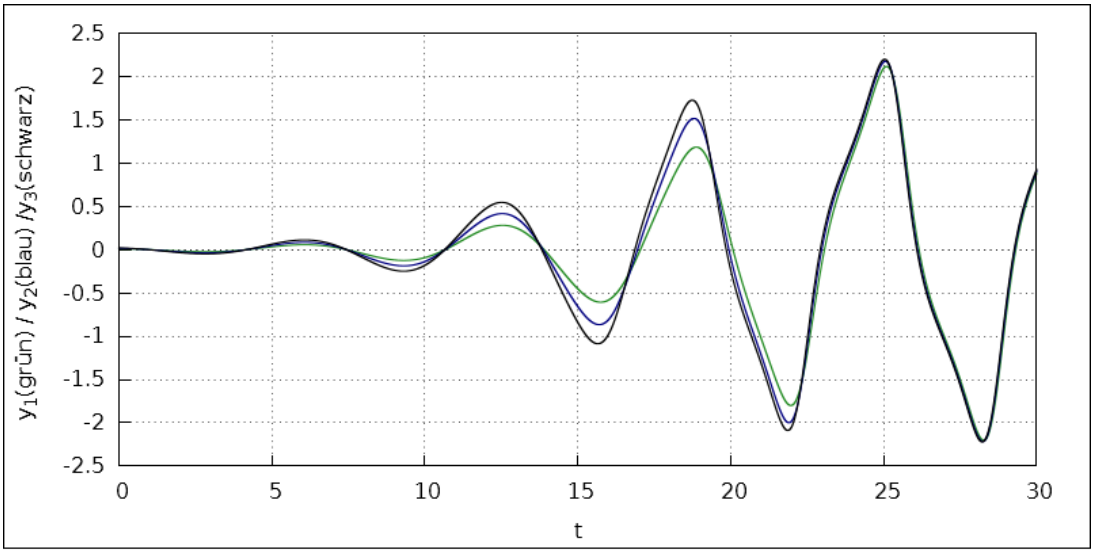


Abbildung 5.18: Zeitverläufe der Zustandsgröße y beim Van-der-Pol-Oszillator bei unterschiedlichen Anfangsbedingungen (Maxima)

(%t21)

```
(%i22)  wxplot_size:[600,600]$
(%i23)  wxdraw2d(color=orange,yrange=[-2.5,2.5],xrange=[-2.5,2.5],
                 points(x_werte_maxima[1],y_werte_maxima[1]),
                 color=red,
                 points(x_werte_maxima[2],y_werte_maxima[2]),
                 color=black,xaxis=true,yaxis=true,
                 points(x_werte_maxima[3],y_werte_maxima[3]),
                 xlabel="x_1(orange) / x_2(rot) / x_3(schwarz)",
                 ylabel="y_1(orange) / y_2(rot) / y_3(schwarz)",
                 user_preamble="set size ratio -1")$
```

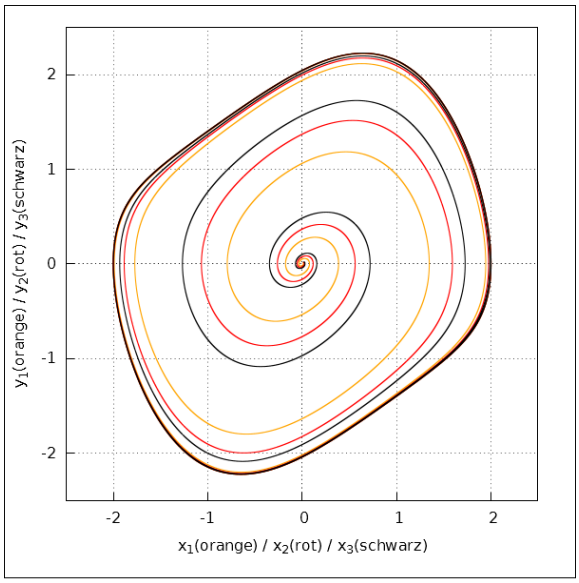


Abbildung 5.19: x/y-Trajektion des Van-der-Pol-Oszillators bei unterschiedlichen Anfangsbedingungen (Maxima)

(%t23)

5.4.2 Matlab

Nach folgender Eingabe in Matlab kann das System, nachdem das zugehörige m-File erstellt wurde, simuliert werden:

```
cd C:\Users\User\Desktop\Schule\Laboratorium-5AHET
     \04_Simulation\_Matlab_Workspace\Van_der_Pol_Oszillator
van_der_pol_function(0.5)
```

Das m-File:

```
1  function [x,y] = van_der_pol_function(e, initVar, T, dt)
2  % van_der_pol_function berechnet den Van-der-Pol-Oszillator
3  % mit Hilfe des
4  % ode45-Befehles (Runge-Kutta-Verfahren (4,5))
5
6  % Wenn zu wenige Argumente angegeben wurden:
7  if nargin<1
8    error('MATLAB:lorenz:NotEnoughInputs',
9          'Not enough input arguments.');
10 end
11 % Wenn zu wenige Argumente angegeben wurden,
12 % jedoch die Systemparameter bekannt sind:
13 if nargin<2
14   initVar   = [0.001 0.001];% Initialisierung (Startwerte)
15   T         = [0 50];       % Zeitbereich
16   dt        = 10e-12;       % "Schrittweite"
17 end
```

```
18 options = odeset('RelTol',dt,'AbsTol',[dt dt/10]);
19 % RelTol = Toleranz, welche ein Maß für die Abweichung in Bezug
20 %           auf die Größe der einzelnen Lösungskomponenten
21 %           darstellt.
22 % AbsTol = Die absolute Fehlertoleranzen bestimmen die
23 %           Genauigkeit, wenn die Lösung gegen Null geht.
24 [T,X]    = ode45(@(T,X) F(T, X, e), T, initVar, options);
25 % [Zeit, Ergebnisarray]
26 % = solver("Zustandsgleichungen", Zeitbereich,
27 %           Initialisierung, Optionen)
28 t = T;
29 x = X(:,1);
30 y = X(:,2);
31
32 % Zusätzliche Farben
33 colour_orange   = [ 246 143 15] ./ 255;
34 colour_rot      = [ 181 12  0]  ./ 255;
35 colour_blau     = [ 0   32  162]./ 255;
36
37 % x/y-Trajektion
38 figure(1);
39 hold on;
40     plot(x,y,'Color',colour_orange);
41 hold off;
42 grid;
43 set(figure(1),'Color',[1 1 1]);
44 ax=gca;ax.XTick=-2.5:0.5:2.5;ax.YTick=-2.5:0.5:2.5;
45 daspect([1 1 1]);
46 set(figure(1),'Units','points','Position',[0 0 600 600]);
47 xlabel('x'); ylabel('y');
48 axis([-2.5 2.5 -2.5 2.5]);
```

```
49 % Zeitverlauf
50 figure(2);
51 hold on;
52     plot(t,x,'Color',colour_rot);
53     plot(t,y,'Color',colour_blau);
54 hold off;
55 grid;
56 set(figure(2),'Color',[1 1 1]);
57 ax=gca;ax.XTick=0:5:50;ax.YTick=-2.5:0.5:2.5;
58 set(figure(2),'Units','points','Position',[0 0 800 400]);
59 xlabel('t'); ylabel('x_r_o_t y_b_l_a_u');
60 axis([0 50 -2.5 2.5]);
61
62 % Ausgabe der berechneten Werte in eine Datei (nur alle 10 Werte)
63 fid = fopen('Van_der_Pol_Oszillator_matlab_werte.txt','wt');
64 for i=1:10:length(x)
65     fprintf(fid,'%e,%e,%e\n',x(i),y(i),t(i));
66 end
67 fclose(fid);
68 return
69  end
70
71 % Zustandsgleichhungen
72 function dx = F(~, X, e)
73     dx = zeros(2,1);
74     dx(1) = X(2);
75     dx(2) = -X(1)+e*(1-X(1)*X(1))*X(2);
76     return
77 end
```

Abbildung 5.20: Matlab-Code zur Berechnung & Simulation des Van-der-Pol-Oszillators (Matlab)

Das Simulationsergebnis von Matlab liefert folgende Diagramme:

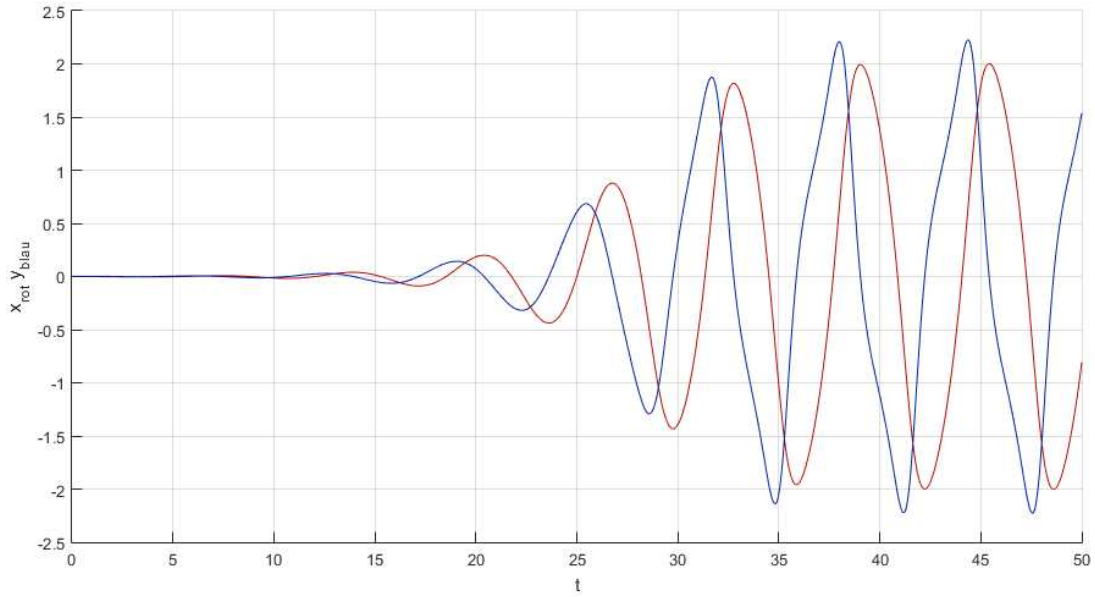


Abbildung 5.21: Zeitverläufe der Zustandsgrößen x & y beim Van-der-Pol-Oszillator (Matlab)

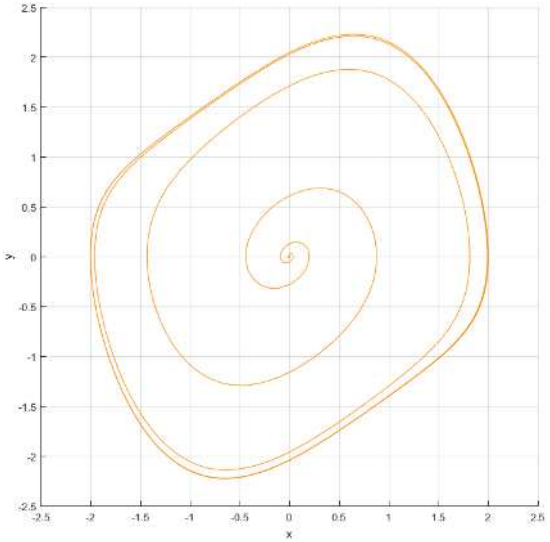


Abbildung 5.22: x/y-Trajektion des Van-der-Pol-Oszillators (Matlab)

Im Zeitverlauf ist gut erkennbar, dass immer wenn x bzw. y ein Maximum hat, dann ist y bzw. x gleich 0 und in der Trajektion ist das Aufschwingen des Attraktors deutlich erkennbar.

5.5 Gegenüberstellung

Die einzelnen Simulationsergebnisse werden nun im Maxima-Programm gegenübergestellt.

Laden des *dynamics*- und *coma*(*draw*)-Paketes und setzen von Defaultwerten:

```
(%i3)   kill(all)$
        load(coma)$
        load(dynamics)$set_draw_defaults(grid=true,
           point_type=0,points_joined=true)$
```

coma v.1.73, (Wilhelm Haager, 2015-01-09)

Berechnen der Werte in Maxima:

```
(%i5)   f1 : y$
        f2 : -x - e*(x^2-1)*y$
```

Zustandsgrößen:

```
(%i6)   states:[x,y]$
```

Konstanten:

```
(%i7)   constants:[e=0.5]$
```

Anfangsbedingungen:

```
(%i8)   initialisation:[0.001,0.001]$
```

Domäne (abhängige Variable t (Zeit), Simulationsbeginn, Simulationsende, Schrittweite):

```
(%i9)   domain:[t,0,50,0.0001]$
```

Berechnung und Zerlegung:
rk (functions, states, initialisation, domain)

```
(%i10)  res:rk([f1,ev(f2,constants)],states,initialisation,domain)$
(%i13)  t_werte_maxima:map(first,   res)$
        x_werte_maxima:map(second,  res)$
        y_werte_maxima:map(third,   res)$
```

Einlesen der Werte von Scilab (Xcos):

```
(%i17)  t_werte_xcos:read_list("C:\\Users\\User\\Desktop\\Schule
        \\Laboratorium-5AHET\\04_Simulation\\
        02_Van_der_Pol_Oszillator\\
        Van_der_Pol_Oszillator_xcos_time.csv")$
        werte_xcos:read_nested_list("C:\\Users\\User\\Desktop
        \\Schule\\Laboratorium-5AHET\\04_Simulation\\
        02_Van_der_Pol_Oszillator\\
        Van_der_Pol_Oszillator_xcos_values.csv",comma)$
        x_werte_xcos:map(first,werte_xcos)$
        y_werte_xcos:map(second,werte_xcos)$
```

Einlesen der Werte von Java:

```
(%i21)  werte_java:read_nested_list("C:\\Users\\User\\Desktop
        \\Schule\\Laboratorium-5AHET\\04_Simulation\\
        02_Van_der_Pol_Oszillator\\
        Van_der_Pol_Oszillator_java_werte.csv",comma)$
        t_werte_java:map(first,werte_java)$
        x_werte_java:map(second,werte_java)$
        y_werte_java:map(third,werte_java)$
```

Einlesen der Werte von C:

```
(%i25)  werte_c:read_nested_list("C:\\Users\\User\\Desktop\\
        Schule\\Laboratorium-5AHET\\04_Simulation\\
        02_Van_der_Pol_Oszillator\\
        Van_der_Pol_Oszillator_c_werte.csv",comma)$
        x_werte_c:map(first,werte_c)$
        y_werte_c:map(second,werte_c)$
        t_werte_c:map(third,werte_c)$
```

Einlesen der Werte von Matlab (Simulink):

```
(%i29)  werte_matlab:read_nested_list("C:\\Users\\User\\Desktop\\
        Schule\\Laboratorium-5AHET\\04_Simulation\\
        02_Van_der_Pol_Oszillator\\
        Van_der_Pol_Oszillator_matlab_werte.csv",comma)$
        x_werte_matlab:map(first,werte_matlab)$
        y_werte_matlab:map(second,werte_matlab)$
        t_werte_matlab:map(third,werte_matlab)$
```

```
(%i30)  wxplot_size:[800,400]$
(%i31)  wxdraw2d(   key="Maxima",   color=red,
                    points(t_werte_maxima,x_werte_maxima),
                    key="Java",     color=orange,
                    points(t_werte_java,x_werte_java),
                    key="Xcos",     color=black,
                    points(t_werte_xcos,x_werte_xcos),
                    key="C",        color=blue,
                    points(t_werte_c,x_werte_c),
                    key="Matlab",   color=forest-green,
                    points(t_werte_matlab,x_werte_matlab),
                    yrange=[-2.5,2.5],xrange=[0,50],
                    user_preamble="set key top left",
                    xaxis=true,dimensions=[1500,600],
                    xlabel="t",ylabel="x");
```

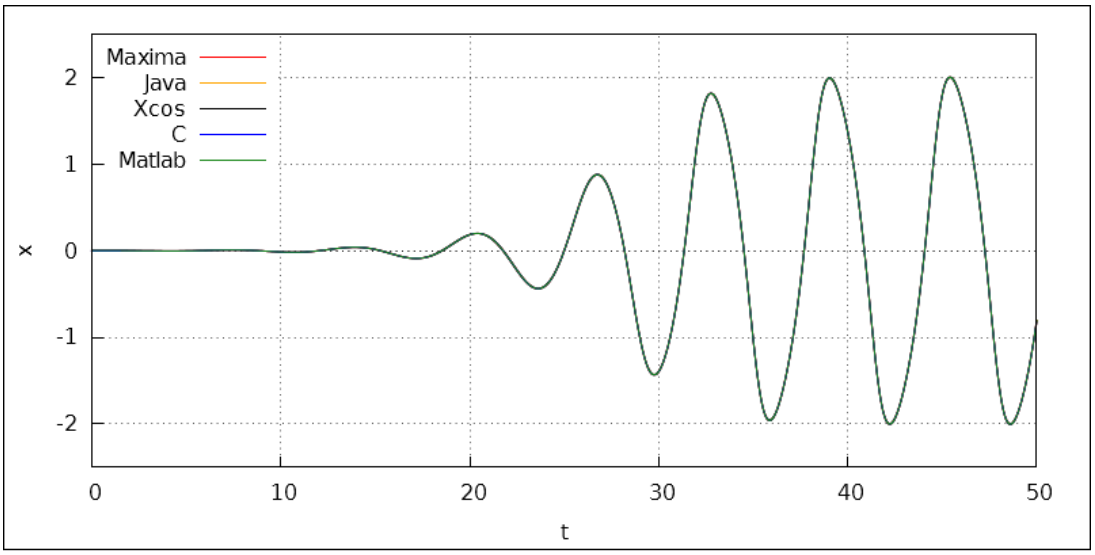


Abbildung 5.23: Zeitverläufe der Zustandsgröße x im Vergleich beim Van-der-Pol-Oszillator (Java, C, Scilab (Xcos), Maxima, Matlab (Simulink))

(%t31) (%o31)

Aus dieser Abbildung zeigt sich, dass alle angewandten Simulationsverfahren in diesem Falle exakt das selbe, optisch gesehene, Ergebnis liefern.

```
(%i32)  wxdraw2d(   key="Maxima",   color=red,
                    points(t_werte_maxima,y_werte_maxima),
                    key="Java",     color=orange,
                    points(t_werte_java,y_werte_java),
                    key="Xcos",     color=black,
                    points(t_werte_xcos,y_werte_xcos),
                    key="C",        color=blue,
                    points(t_werte_c,y_werte_c),
                    key="Matlab",   color=forest-green,
                    points(t_werte_matlab,y_werte_matlab),
                    yrange=[-2.5,2.5],xrange=[0,50],
                    user_preamble="set key top left",
                    xaxis=true,dimensions=[1500,600],
                    xlabel="t",ylabel="y"
        );
```

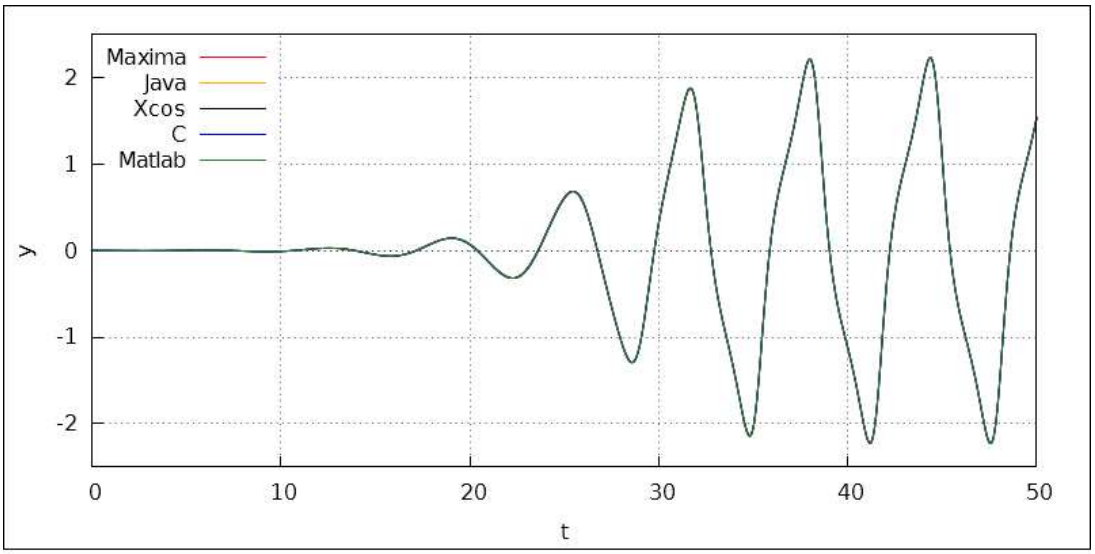


Abbildung 5.24: Zeitverläufe der Zustandsgröße y im Vergleich beim Van-der-Pol-Oszillator (Java, C, Scilab (Xcos), Maxima, Matlab (Simulink))

(%t32) (%o32)

```
(%i33)  wxplot_size:[600,600]$
(%i34)  wxdraw2d(   key="Maxima",   color=red,
                    points(x_werte_maxima,y_werte_maxima),
                    key="Java",     color=orange,
                    points(x_werte_java,y_werte_java),
                    key="Xcos",     color=black,
                    points(x_werte_xcos,y_werte_xcos),
                    key="C",        color=blue,
                    points(x_werte_c,y_werte_c),
                    key="Matlab",   color=forest-green,
                    points(x_werte_matlab,y_werte_matlab),
                    yrange=[-2.5,2.5],xrange=[-2.5,2.5],
                    xaxis=true,dimensions=[1500,600],
                    user_preamble="set size ratio -1",
                    xlabel="x",
                    ylabel="y"
        );
```

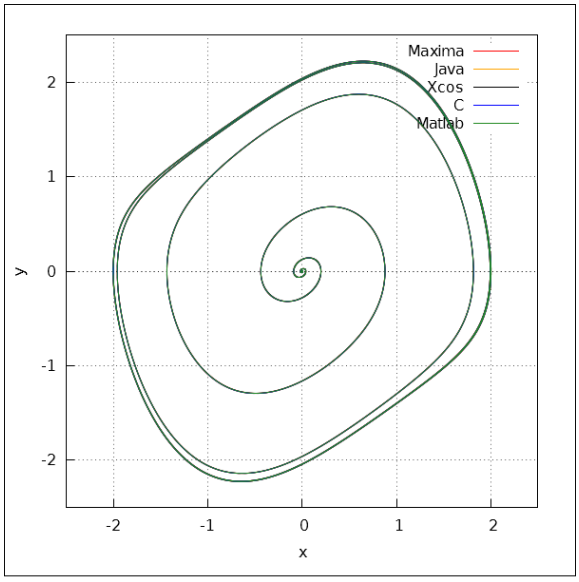


Abbildung 5.25: x/y-Trajektionen im Vergleich beim Van-der-Pol-Oszillator (Java, C, Scilab (Xcos), Maxima, Matlab (Simulink))

(%t34) (%o34)

6 Reibungsschwinger

6.1 Grundlagen

Der Reibungsschwinger kann als Modell für viele beabsichtigte oder unbeabsichtigte selbsterregte Schwingungen in der Natur und Technik angesehen werden.
Als mechanisches Modell besteht dieser aus einem Körper mit der Masse m, der mit einer Feder mit der Federkonstante k verbunden ist. Der Körper befindet sich auf einem umlaufenden Band, welches mit einer Geschwindigkeit v_0 angetrieben wird. Zwischen Band und Körper liegt eine Reibung vor und die Masse kann vom Förderband nicht beliebig mitbewegt werden. Aufgrund der unterschiedlichen Koeffizienten für Haftreibung und Gleitreibung stellt sich bei Bewegung des Förderbandes eine Dauerschwingung der Masse ein. Das nachfolgende Bild veranschaulicht das Modell:

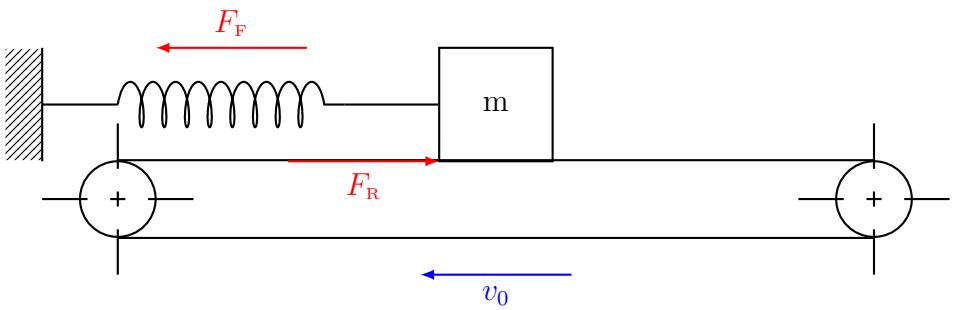


Abbildung 6.1: Mechanisches Modell des Reibungsschwingers

Dabei muss gelten:

$$F = F_R - F_F \tag{6.1}$$

$$F = m \cdot \ddot{x} \qquad \text{mit} \quad \ddot{x} = \dot{v} \tag{6.2}$$

$$F_R = f\left(v_r\right) \qquad \text{mit } v_r = v_0 - v \tag{6.3}$$

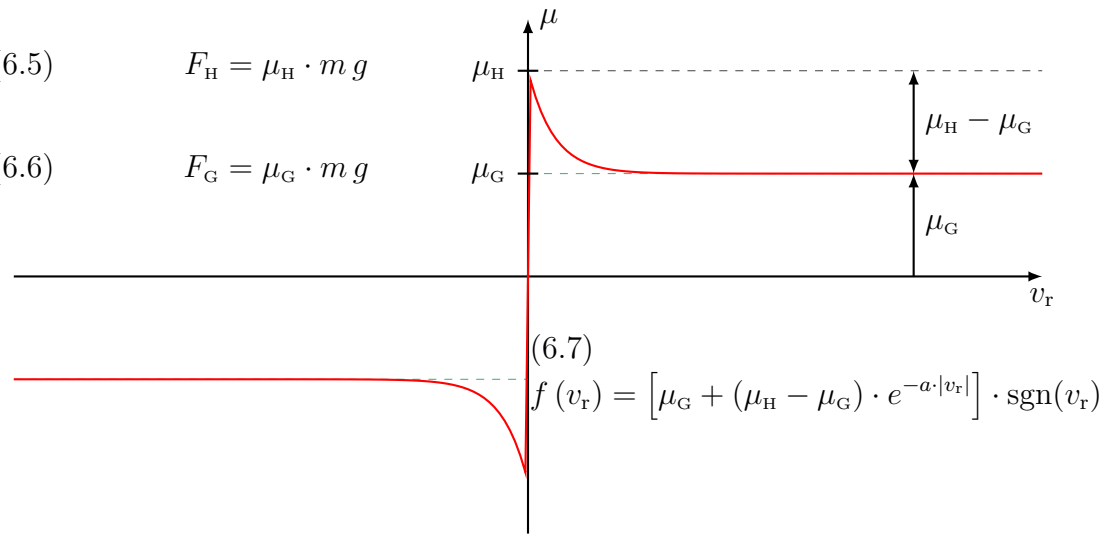
$$F_F = k \cdot x \tag{6.4}$$

F	resultierende Kraft	μ_G	Gleitreibungszahl
F_R	Reibungskraft	μ_H	Haftreibungszahl
F_F	Federkraft	g	Erdbeschleunigung
k	Federkonstante	a	Übergangskoeffizient zwischen Haften und Gleiten
m	Masse		
v	Geschwindigkeit		
v_r	Relativ-		
v_0	Förderband-		
x	Position		

Der Zusammenhang zwischen Haft- und Gleitreibung kann mit Hilfe der Stribeck-Kurve $f(v_r)$ dargestellt werden:

$$F_H = \mu_H \cdot m\,g \tag{6.5}$$

$$F_G = \mu_G \cdot m\,g \tag{6.6}$$



$$f(v_r) = \left[\mu_G + (\mu_H - \mu_G) \cdot e^{-a\cdot|v_r|}\right] \cdot \operatorname{sgn}(v_r) \tag{6.7}$$

Abbildung 6.2: Stribeck-Kurve

Mit dieser Kurve lässt sich nun der von der Geschwindigkeit abhängende Reibungskoeffizient beschreiben.

Formt man die Differentialgleichung 2. Ordnung auf nichtlineare, gewöhnliche Differentialgleichungen 1. Ordnung um, so ergeben sich die folgenden zwei Gleichungen:

$$\dot{x} = v \tag{6.8}$$

$$\dot{v} = \frac{1}{m} \cdot [m\, g \cdot f\,(v_\mathrm{r}) - k \cdot x] \tag{6.9}$$

Um ein wenig Realität in dieses Model hineinzubringen wurde folgende Parameterkonstellation verwendet:

$$\begin{array}{llll} m = 1\,\mathrm{kg} & g = 9.80665\,\mathrm{m/s^2} & v_0 = 0.2\,\mathrm{m/s} & \\ \mu_\mathrm{H} = 0.4 & \mu_\mathrm{G} = 0.2 & a = 20\,\mathrm{s/m} & k = 30\,\mathrm{kg/s^2} = 30\,\mathrm{N/m} \end{array} \tag{6.10}$$

Als Anfangsbedingung wird die Position und Geschwindigkeit gleich 0 gesetzt:

$$x(0) = 0 \qquad v(0) = 0 \tag{6.11}$$

6.2 Simulation mit Differenzengleichungen

6.2.1 Java

Folgendes Java-Programm simuliert den Reibungsschwinger durch Ersetzen der Differentialquotienten durch Differenzenquotienten mit einer Schrittweite von 0.000 1 im Zeitbereich von 0 sec bis 5 sec:

```
1  package simulationen;
2
3  /** Simulation des Reibungsschwingers */
4  public class Reibungsschwinger {
5    public static void main(String[] args) {
6      Simulation Sim_1 =
7          new Simulation(20,9.80665,30,
8              1,0.4,0.2,0.2,0,0,0,5,0.0001);
9      Sim_1.calculation();
10     Sim_1.write_to_txt(
11         "Reibungsschwinger_java_werte.txt");
12   }
13 }
```

Abbildung 6.3: Java-Code zur Simulation des Reibungsschwingers (Java)

```
1  package simulationen;
2
3  import java.io.BufferedWriter;
4  import java.io.FileWriter;
5  import java.io.IOException;
```

Abbildung 6.4: Java-Code (Imports) zur Berechnung des Reibungsschwingers (Java)

```
6  public class Simulation {
7    private double  a,g,k,v_0,mu_G,mu_H,m,x0,y0,t_begin,t_end,dt;
8    private int quantity;
9    private double [] t;
10   private double [] x;
11   private double [] y;
12
13   /**Legt die Startwerte und Parameter der Simulation fest
14    * @param x_init    = Startwert x
15    * @param y_init    = Startwert y
16    * @param a         = Konstant
17    * @param g         = Konstant
18    * @param k         = Konstant
19    * @param m         = Konstant
20    * @param mu_H      = Konstant
21    * @param mu_G      = Konstant
22    * @param v_0       = Konstant
23    * @param t_begin_sim    = Simulationsbegin
24    * @param t_end_sim    = Simulationsende
25    * @param increment_sim = Schrittweite */
26   public Simulation(
27       double a_constant, double g_constant, double k_constant,
28       double m_constant, double mu_H_constant,
29       double mu_G_constant, double v_0_constant, double x_init,
30       double y_init,
31       double t_begin_sim, double t_end_sim, double increment_sim)
32     {
33     a=a_constant;
34     g=g_constant;
35     k=k_constant;
36     m=m_constant;
37     mu_H=mu_H_constant;
38     mu_G=mu_G_constant;
39     v_0=v_0_constant;
40     x0=x_init;
41     y0=y_init;
42     t_begin=t_begin_sim;
43     t_end=t_end_sim;
44     dt=increment_sim;
45     quantity=(int)((t_end-t_begin)/dt);
46   }
```

```
47    /**Berechnet die x- und y-Werte und übergibt die x,y-
48     * und t-Werte einem Array */
49    public void calculation(){
50      int i=0;
51      t = new double [quantity+2];
52      x = new double [quantity+2];
53      y = new double [quantity+2];
54      x[i]=x0;y[i]=y0;t[i]=t_begin;
55      for(double t_zaehler=t_begin;t_zaehler<=t_end;
56          t_zaehler+=dt){i++;
57        double dx=y[i-1]*dt;
58        double dy=1/m*((m*g*(mu_G+(mu_H-mu_G)*
59            Math.exp(-a*Math.abs(v_0-y[i-1])))*
60            Math.signum(v_0-y[i-1]))-k*x[i-1])*dt;
61        x[i]=x[i-1]+dx;
62        y[i]=y[i-1]+dy;
63        t[i]=t_zaehler+dt;
64      }
65    }
66
67    /**Gibt die t-Werte zurück
68     * @return [] t-Werte */
69    public double[] get_t_werte(){
70      return t;
71    }
72    /**Gibt die x-Werte zurück
73     * @return [] x-Werte */
74    public double[] get_x_werte(){
75      return x;
76    }
77    /**Gibt die y-Werte zurück
78     * @return [] y-Werte */
79    public double[] get_y_werte(){
80      return y;
81    }
```

```
82    /**Schreibt die Array-Werte x,y,z und t in ein txt-File
83     * @param path_write_to_txt = Pfad*/
84    public void write_to_txt(String path_write_to_txt){
85      String path = path_write_to_txt;
86          String arrayString = "";
87
88          for (int i = 0; i < t.length ; i+=2){
89              arrayString = arrayString + t[i]+" , " + x[i]+" , "
90                  + y[i]+" , " +"\n";
91              // Fortschrittsanzeige (nur zur Übersichtlichkeit
92              // beim Buildn)
93            System.out.println((double)i*100/(t.length)+" %");
94          } System.out.println(100+" %");
95          try {
96              BufferedWriter out =
97                 new BufferedWriter(new FileWriter(path));
98              out.write(arrayString);
99              out.close();
100         } catch (IOException e) {}
101   }
102 }
```

Abbildung 6.5: Java-Code zur Berechnung des Reibungsschwinger (Java)

Die Darstellung der Berechnung ist im Anschluss im Vergleich mit Maxima ersichtlich.

6.3 Simulation mit Funktionsblöcken

6.3.1 Scilab (Xcos)

Das Blockschaltbild des Modells vom Reibungsschwinger hat folgende Form:

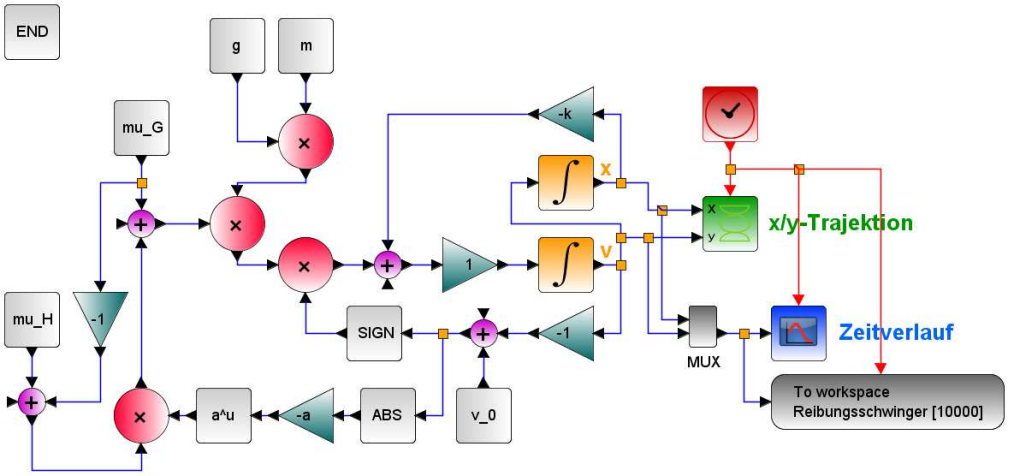


Abbildung 6.6: Blockschaltbild des Reibungsschwingers (Scilab (Xcos))

Das Programm liefert nach der Simulation sowohl den zeitlichen Verlauf der Zustandsgrößen, als auch die x/v-Trajektion. Für die Simulation müssen noch in Scilab die einzelnen Parameter des Modells eingegeben werden:

```
g=9.80665;
k=30;
a=20;
v_0=0.2;
mu_H=0.4;
mu_G=0.2;
```

Mit folgender Eingabe in Scilab können die fertig simulierten Werte in eine csv-Datei geschrieben werden:

```
write_csv(Reibungsschwinger.values,
   'Reibungsschwinger_xcos_values.csv')
write_csv(Reibungsschwinger.time,
   'Reibungsschwinger_xcos_time.csv')
```

Als Gleichungslöser wurde bei Scilab (Xcos) *Dormand-Prince 4(5) (DOPRI5)* verwendet, wobei insgesamt 5000 Punkten im Zeitbereich von $0 - 5$ berechnet wurden.

Parameter	Einstellung
Finale Integrationszeit	$1.0E03$
Echt-Zeit-Skalierung	$0,0E00$
Absolute Toleranz des Integrators	—
Relative Toleranz des Integrators	—
Zeit-Toleranz	—
Maximales Zeitintervall der Integration	—
Gleichungslöser	DOPRI5 - Dormand-Prince 4(5)
Maximale Schrittweite (0 = kein Limit)	$0,0E00$

Tabelle 6.1: Parametereinstellungen in Scilab für den Reibungsschwinger (Xcos)

Der Zeitverlauf der Zustandsgrößen x & v (y) zeigt folgenden Verlauf:

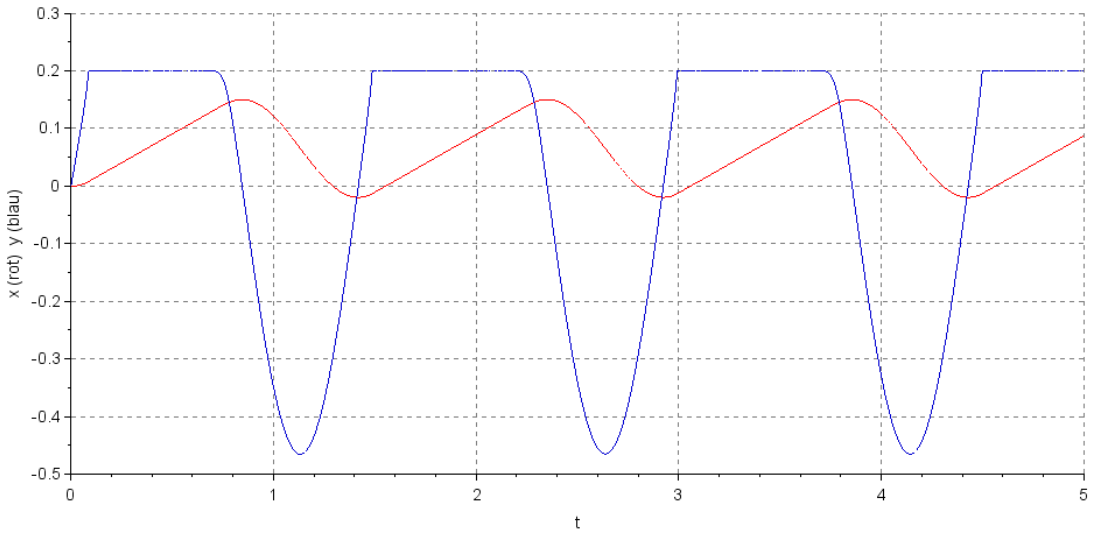


Abbildung 6.7: Zeitverläufe der Zustandsgrößen x & v beim Reibungsschwinger (Scilab (Xcos))

Die Abbildung lässt erkennen, dass, wenn die Geschwindigkeit konstant bei $0,2\,\mathrm{m/s}$ (Förderbandgeschwindigkeit) ist, befindet sich der Körper am Förderband und die

Feder wird zusammengedrückt, bis die Feder wieder ausschlägt, was an der typischen Verlauf der Stribeck-Kurve (Mischreibung) erinnert.

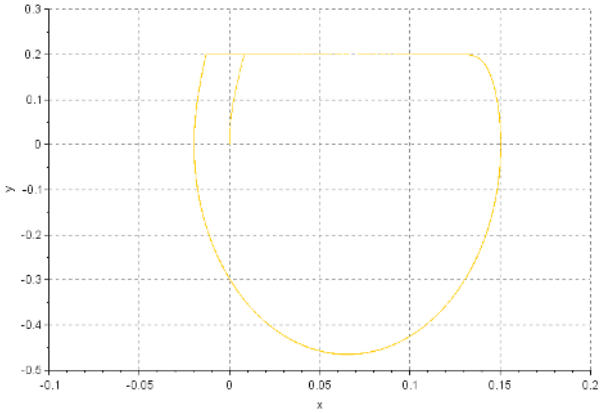


Abbildung 6.8: x/v-Trajektion des Reibungsschwingers (Scilab (Xcos))

Auch in der Trajektion ist die Geschwindigkeit v_0 des Förderbandes von $0,2\,\mathrm{m/s}$ und das Ausschwingen der Feder, wenn diese zu stark zusammengedrückt wurde, erkennbar.

6.4 Simulation mit dem Runge-Kutta-Verfahren

6.4.1 Maxima

```
(%i1)   kill(all)$
```

Laden des *dynamics*- und *coma*(*draw*)-Paketes und setzen von Defaultwerten:

```
(%i3)   load(coma)$
        load(dynamics)$set_draw_defaults(grid=true,
            point_type=0,points_joined=true)$
```

coma v.1.73, (Wilhelm Haager, 2015-01-09)

```
(%i4)   states:[x,v]$
(%i5)   constants:
           [mu_H=0.4,mu_G=0.2,a=20,m=1,g=9.80665,k=30,v_0=0.2]$
(%i6)   initialisation:[0,0]$
(%i7)   domain:[t,0,5,0.0001]$
(%i8)   f_stribeck(v_r):=(mu_G+(mu_H-mu_G)*
           %e^(-a*abs(v_r)))*signum(v_r)$
(%i10)  f1 : v$
        f2 : ev(1/m*((m*g*f_stribeck(v_r))-k*x),v_r=v_0-v)$
(%i11)  res:rk([f1,ev(f2,constants)],states,initialisation,domain)$
(%i14)  t_werte_maxima:map(first,   res)$
        x_werte_maxima:map(second,  res)$
        v_werte_maxima:map(third,   res)$
```

Zeitverlauf:

```
(%i15)  wxplot_size:[800,400]$
(%i16)  wxdraw2d(color=red,
                 points(t_werte_maxima,x_werte_maxima),
                  color=navy,
                  points(t_werte_maxima,v_werte_maxima),
                  yrange=[-0.5,0.3],xrange=[0,domain[3]],
                  xaxis=true,dimensions=[1500,600],
            xlabel="t",ylabel="x(rot) / v(blau) "
         )$
```

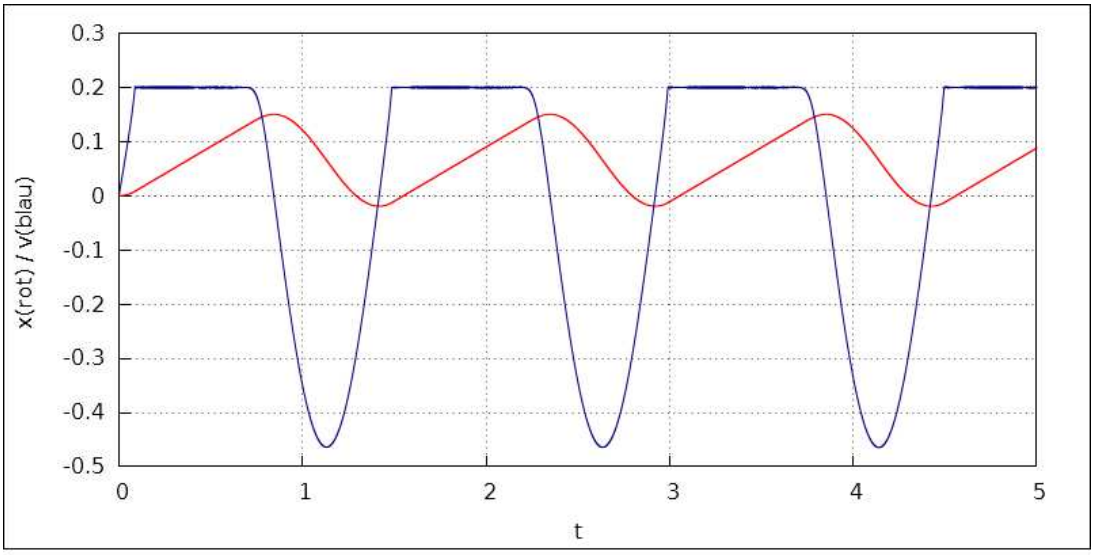


Abbildung 6.9: Zeitverläufe der Zustandsgrößen x & v beim Reibungsschwinger

(%t16)

Trajektorie in der x/v-Phasenebene:

```
(%i17)  wxplot_size:[600,400]$
(%i18)  wxdraw2d(color=orange,
                 points(x_werte_maxima,v_werte_maxima),
                 yrange=[-0.5,0.3],xrange=[-0.1,0.2],
                 xaxis=true,yaxis=true,ytics=0.1,
                 xlabel="x",ylabel="v"
        )$
```

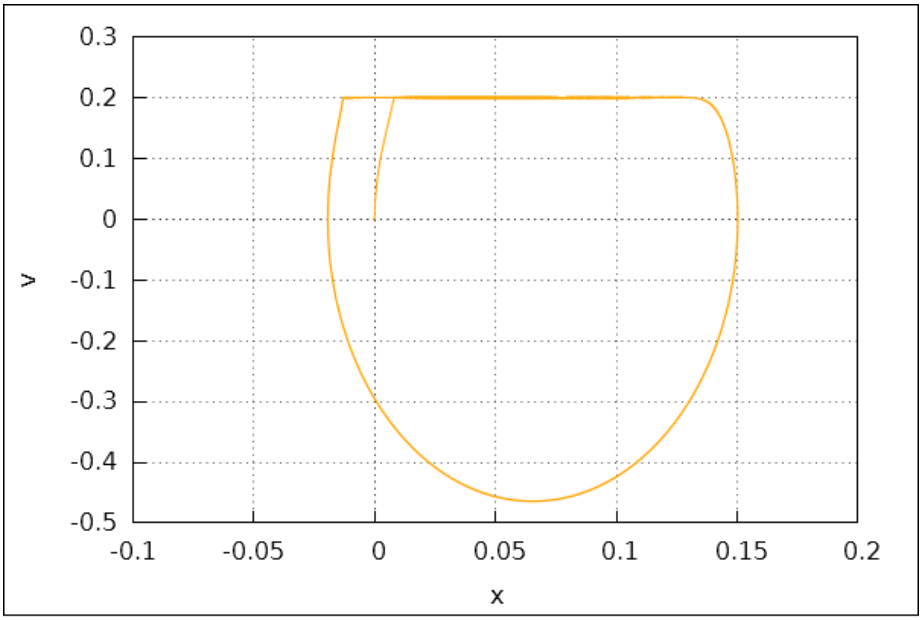


Abbildung 6.10: x/v-Trajektion des Reibungsschwingers (Maxima)

(%t18)

6.4.2 Matlab

Mit Hilfe folgender Eingabe kann der Reibungsschwinger mit Matlab, nach dem Erstellen des m-Files simuliert werden:

```
cd C:\Users\User\Desktop\Schule\Laboratorium-5AHET
     \04_Simulation\_Matlab_Workspace\Reibungsschwinger
reibungsschwinger_function(1,0.2,0.4,0.2,20,30)
```

Das m-File:

```
1  function [x,y] = reibungsschwinger_function(
2      m,mu_G,mu_H,v_0,a,k,g, initVar, T, dt)
3  % reibungsschwinger_function berechnet den Reibungsschwinger
4  % mit Hilfe des
5  % ode45-Befehles (Runge-Kutta-Verfahren (4,5))
6
7  % Wenn zu wenige Argumente angegeben wurden:
8  if nargin<6
9    error('MATLAB:lorenz:NotEnoughInputs',
10         'Not enough input arguments.');
11 end
12 % Wenn zu wenige Argumente angegeben wurden,
13 % jedoch die Systemparameter bekannt sind:
14 if nargin<7
15   g = 9.80665;       % Erdbeschleunigung
16   initVar = [0  0];  % Startwerte
17   T = [0  5];        % Zeitbereich
18   dt = 10e-7;        % Schrittweite
19 end
20
21 options = odeset('RelTol',dt,'AbsTol',[dt dt/10]);
22 % RelTol = Toleranz, welche ein Maß für die Abweichung in Bezug
23 %          auf die Größe der einzelnen Lösungskomponenten
24 %          darstellt.
```

```
25 % AbsTol = Die absolute Fehlertoleranzen bestimmen die
26 %           Genauigkeit, wenn die Lösung gegen Null geht.
27 [T,X]    = ode45(@(T,X) F(T, X, m, mu_G ,mu_H, v_0, a, k, g),
28                  T, initVar, options);
29 % [Zeit, Ergebnisarray]
30 % = solver("Zustandsgleichungen", Zeitbereich,
31 %           Initialisierung, Optionen)
32 t = T;
33 x = X(:,1);
34 y = X(:,2);
35
36 % Zusätzliche Farben
37 colour_orange   = [246  143 15] ./ 255;
38 colour_rot      = [181  12  0]  ./ 255;
39 colour_blau     = [0    32  162]./ 255;
40
41 % x/y-Trajektion
42 figure(1);
43 hold on;
44     plot(x,y,'Color',colour_orange);
45 hold off;
46 grid;
47 set(figure(1),'Color',[1 1 1]);
48 ax=gca;ax.XTick=-0.05:0.025:0.2;ax.YTick=-0.5:0.05:0.3;
49 daspect([1 4 4]);
50 set(figure(1),'Units','points','Position',[0 0 600 600]);
51 xlabel('x'); ylabel('y');
52 axis([-0.05 0.2 -0.5 0.3]);
53
54 % Zeitverlauf
55 figure(2);
56 hold on;
57     plot(t,x,'Color',colour_rot);
58     plot(t,y,'Color',colour_blau);
59 hold off;
60 grid;
61 set(figure(2),'Color',[1 1 1]);
62 ax=gca;ax.XTick=0:0.5:5;ax.YTick=-0.5:0.05:0.3;
63 set(figure(2),'Units','points','Position',[0 0 800 400]);
64 xlabel('t'); ylabel('x_r_o_t y_b_l_a_u');
65 axis([0 5 -0.5 0.3]);
```

```
66 % Ausgabe der berechneten Werte in eine Datei. Es werden aber
67 % relativ viele Werte in der Nähe von der Förderband-
68 % geschwindigkeit berechnet --> Nur alle 20 Werte ausgeben,
69 % wenn der Körper sich auf dem Förderband mit v_0 bewegt.
70 fid = fopen('Reibungsschwinger_matlab_werte.txt','wt');
71 k=0;
72 for i=1:1:length(x)
73     k=k+1;
74     if y(i)>(v_0*(1-0.01))
75         if k>20
76             fprintf(fid,'%e,%e,%e\n',x(i),y(i),t(i));
77             k=0;
78         else
79         end
80
81     else
82         fprintf(fid,'%e,%e,%e\n',x(i),y(i),t(i));
83     end
84 end
85 fclose(fid);
86 return
87 end
88
89 % Zustandsgleichhungen
90 function dx = F(~, X,  m, mu_G ,mu_H, v_0, a, k, g)
91     dx = zeros(2,1);
92     dx(1) = X(2);
93     dx(2) = 1/m*(m*g*(mu_G+(mu_H-mu_G)*exp(-a*abs(v_0-X(2))))
94             *sign(v_0-X(2))-k*X(1));
95     return
96 end
```

Abbildung 6.11: Matlab-Code zur Berechnung & Simulation des Reibungsschwingers (Matlab)

Die Darstellung des Simulationsergebnisses erfolgt im nachfolgendem Kapitel 6.5 (Gegenüberstellung).

6.5 Gegenüberstellung

Nun erfolgt die Gegenüberstellung der einzelnen Simulationswerte anhand von Maxima, indem alle berechneten Werte eingelesen und danach graphisch zur Anzeige gebracht werden.

```
(%i22)  werte_java:read_nested_list("C:\\Users\\User\\Desktop
        \\Schule\\Laboratorium-5AHET\\04_Simulation
        \\03_Reibungsschwinger
        \\Reibungsschwinger_java_werte.csv",comma)$
        t_werte_java:map(first,werte_java)$
        x_werte_java:map(second,werte_java)$
        v_werte_java:map(third,werte_java)$
(%i26)  werte_matlab:read_nested_list("C:\\Users\\User\\Desktop
        \\Schule\\Laboratorium-5AHET\\04_Simulation
        \\03_Reibungsschwinger
        \\Reibungsschwinger_matlab_werte.csv",comma)$
        x_werte_matlab:map(first,werte_matlab)$
        v_werte_matlab:map(second,werte_matlab)$
        t_werte_matlab:map(third,werte_matlab)$
(%i30)  t_werte_xcos:read_list("C:\\Users\\User\\Desktop
        \\Schule\\Laboratorium-5AHET\\04_Simulation
        \\03_Reibungsschwinger
        \\Reibungsschwinger_xcos_time.csv")$
        werte_xcos:read_nested_list("C:\\Users\\User\\Desktop
        \\Schule\\Laboratorium-5AHET\\04_Simulation
        \\03_Reibungsschwinger
        \\Reibungsschwinger_xcos_values.csv",comma)$
        x_werte_xcos:map(first,werte_xcos)$
        v_werte_xcos:map(second,werte_xcos)$
```

```
(%i31)  wxplot_size:[800,400]$
(%i32)  wxdraw2d(key="Maxima",   color=red,
                 points(t_werte_maxima,x_werte_maxima),
                 key="Matlab",   color=forest-green,
                 points(t_werte_matlab,x_werte_matlab),
                 key="Java",     color=orange,
                 points(t_werte_java,x_werte_java),
                 key="Xcos",     color=black,
                 points(t_werte_xcos,x_werte_xcos),
                 yrange=[-0.5,0.3],xrange=[0,5],
                 user_preamble="set key bottom left",
                 xaxis=true,dimensions=[1500,600],
                 xlabel="t",ylabel="x"
        )$
```

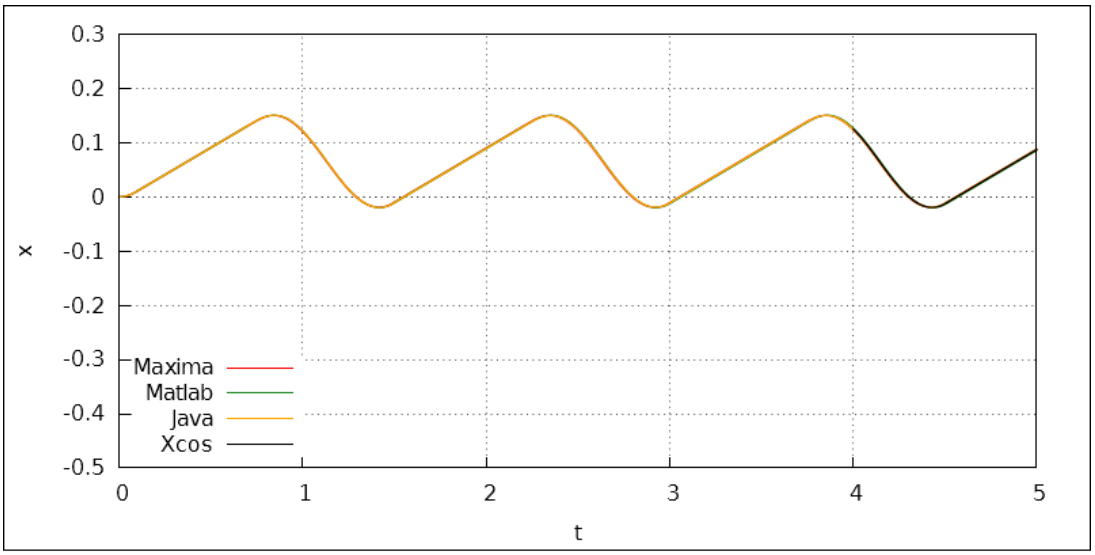


Abbildung 6.12: Zeitverläufe der Zustandsgrößen x im Vergleich beim Reibungsschwinger (Java, Scilab (Xcos), Maxima, Matlab)

(%t32)

```
(%i33)  wxdraw2d(key="Maxima",    color=red,
                 points(t_werte_maxima,v_werte_maxima),
                 key="Matlab",    color=forest-green,
                 points(t_werte_matlab,v_werte_matlab),
                 key="Java",      color=orange,
                 points(t_werte_java,v_werte_java),
                 key="Xcos",      color=black,
                 points(t_werte_xcos,v_werte_xcos),
                 yrange=[-0.5,0.3],xrange=[0,5],
                 user_preamble="set key bottom left",
                 xaxis=true,dimensions=[1500,600],
                 xlabel="t",ylabel="v"
        )$
```

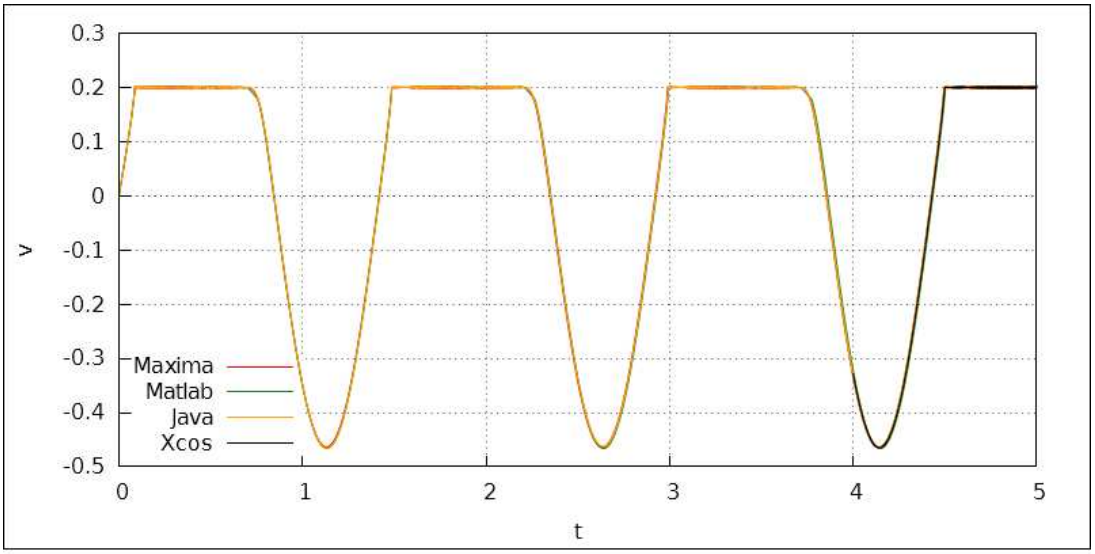


Abbildung 6.13: Zeitverläufe der Zustandsgrößen v im Vergleich beim Reibungsschwinger (Java, Scilab (Xcos), Maxima, Matlab)

(%t33)

Es lässt sich auch hier sagen, dass die verwendeten Schrittweiten (Genauigkeit) ausreichend ist und im Prinzip alle Simulationsverfahren das selbe optische Ergebnis liefern, jedoch sich nach längerer Zeit trotzdem Abweichungen ergeben.

```
(%i34)  wxplot_size:[800,400]$
(%i35)  wxdraw2d(key="Maxima",    color=red,
                 points(x_werte_maxima,v_werte_maxima),
                 key="Matlab",    color=forest-green,
                 points(x_werte_matlab,v_werte_matlab),
                 key="Java",      color=orange,
                 points(x_werte_java,v_werte_java),
                 key="Xcos",      color=black,
                 points(x_werte_xcos,v_werte_xcos),
                 yrange=[-0.5,0.3],xrange=[-0.1,0.2],
                 user_preamble="set key bottom left",
                 xaxis=true,dimensions=[1500,600],
                 xlabel="x",
                 ylabel="v"
        )$
```

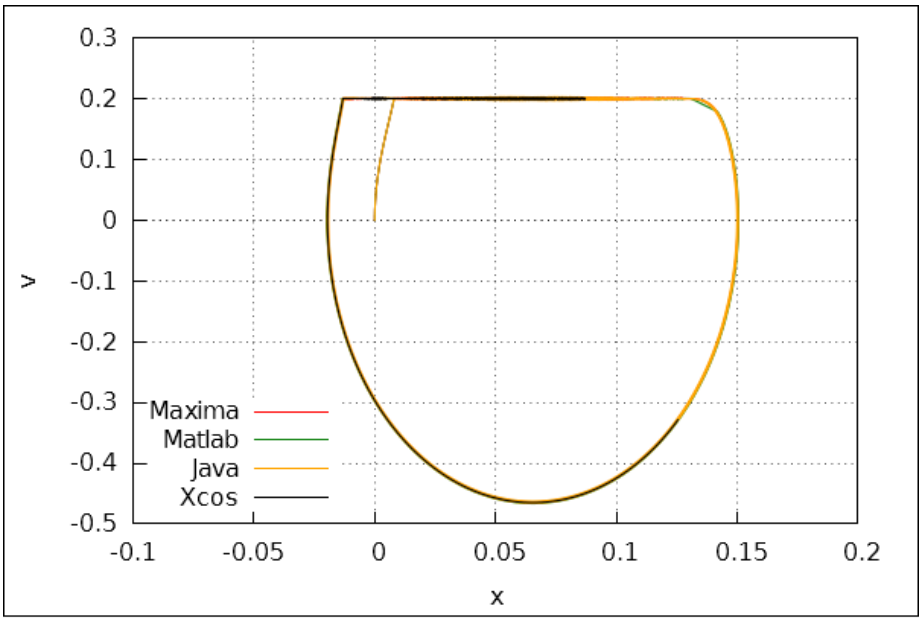


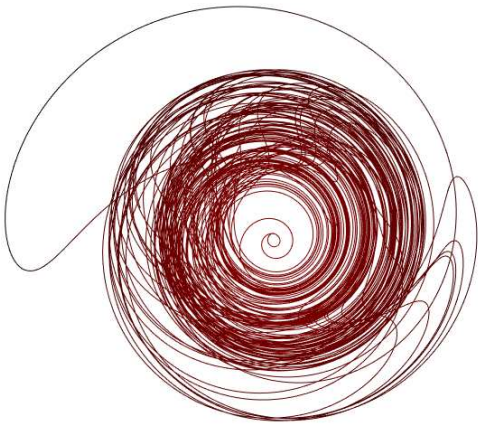
Abbildung 6.14: x/v-Trajektion im Vergleich beim Reibungsschwinger (Java, Scilab (Xcos), Maxima, Matlab)

(%t35)

7 Weitere dynamische Systeme

7.1 Lorenz-84-Attraktor

Das Lorenz-84-System ist aus drei nicht-linearen, gekoppelten, gewöhnlichen Differentialgleichungen gegeben und stellt ein Modell für die langfristige atmosphärische Zirkulation, vorgeschlagen von Edward N. Lorenz, dar:



$$\begin{aligned} \dot{x} &= -\left(x^2 + y^2\right) + \alpha\,(F - x) \\ \dot{y} &= x\,(y - \beta z) - y + G \\ \dot{z} &= x\,(z + \beta y) - z \end{aligned} \tag{7.1}$$

Abbildung 7.1: y/z-Trajektorie des Lorenz-84-Attraktors simuliert mit dem Dormand-Prince-Verfahren in Matlab

Als Anfangswert wurde mit Matlab ein reeller Gleichgewichtspunkt berechnet und gerundet verwendet:

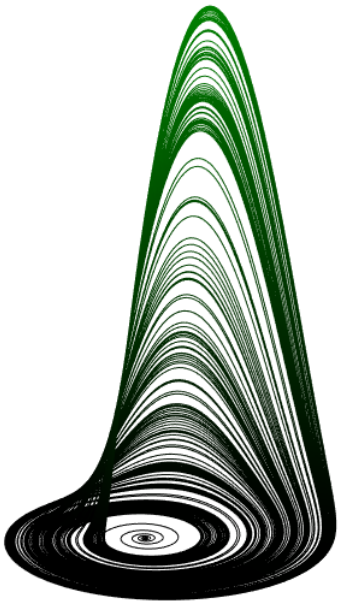
$$x(0) = \frac{13}{10} \qquad y(0) = -\frac{1}{100} \qquad z(0) = \frac{1}{5} \tag{7.2}$$

Dabei betrug die Parameterwahl, welche zu einem chaotischen System führt:

$$\alpha = \frac{1}{4} \qquad \beta = 4 \qquad F = 8 \qquad G = 1 \tag{7.3}$$

7.2 Rössler-Attraktor

Die chaotische Systembewegung des sogenannten Rössler-Systems ergibt den Rössler-Attraktor, ein System aus drei nichtlinearen, gewöhnlichen Differentialgleichungen. Er gehört zu den seltsamen Attraktoren und zeigt somit chaotisches Verhalten und dient heutzutage z. B. zur Modellbildung von Gleichgewichtszuständen in chemischen Reaktionen.



$$\begin{aligned} \dot{x} &= -y - z \\ \dot{y} &= x + \alpha\, y \\ \dot{z} &= \beta + z\,(x - c) \end{aligned} \tag{7.4}$$

Abbildung 7.2: Trajektorie des Rössler-Attraktors im dreidimensionalen Zustandsraum simuliert mit dem Dormand-Prince-Verfahren in Matlab

Eine typische Parameterwahl und Anfangswerte sind:

$$\alpha = \frac{1}{5} \qquad \beta = \frac{1}{5} \qquad c = 5,7 \tag{7.5}$$

$$x(0) = 0 \qquad y(0) = 0 \qquad z(0) = 0 \tag{7.6}$$

Die zwei Gleichgewichtspunkte des Rössler-Attraktors lassen sich folgendermaßen berechnen, indem die rechten Seiten der Differentialgleichungen 7.4 gleich 0 gesetzt werden:

$$P_0\left(\frac{c-\sqrt{c^2-4\alpha^2\beta}}{2}, -\frac{c-\sqrt{c^2-4\alpha^2\beta}}{2\alpha}, \frac{c-\sqrt{c^2-4\alpha^2\beta}}{2\alpha}\right)$$
$$P_1\left(\frac{c+\sqrt{c^2-4\alpha^2\beta}}{2}, -\frac{c+\sqrt{c^2-4\alpha^2\beta}}{2\alpha}, \frac{c+\sqrt{c^2-4\alpha^2\beta}}{2\alpha}\right) \tag{7.7}$$

Dies würde bei der Parameterkonstellation 7.5 zu folgenden Gleichgewichtspunkten des Systemes führen:

$$P_{0,1}\left(\frac{57\pm\sqrt{53}\sqrt{61}}{20}, -\frac{57\pm\sqrt{53}\sqrt{61}}{4}, \frac{57\pm\sqrt{53}\sqrt{61}}{4}\right)$$

Des Weiteren lässt sich aus den Zeitverläufen das typische Verhalten des Rössler-Attraktors erkennen:

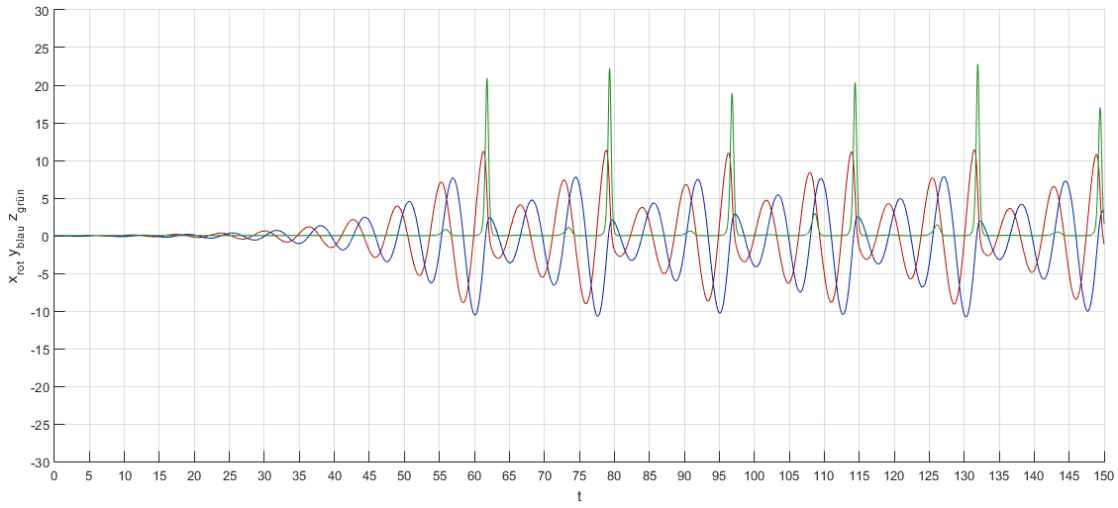


Abbildung 7.3: Zeitverläufe der Zustandsgrößen x, y & z beim Rössler-Attrakor simuliert mit dem Dormand-Prince-Verfahren in Matlab

7.3 Chua-Attraktor

Eine Systembewegung im Chua-System ergibt den Chua-Attraktor, ein System aus drei nichtlinearen, gewöhnlichen Differentialgleichungen, wobei für die Beschreibung von der ϕ-Funktion mehrere Näherungen existieren, wie die hier verwendete kubische Funktion.
Ein solches System lässt sich auch mit dem Oszilloskope zur Anzeigen bringen, wenn die nachfolgende Schaltung aufgebaut wird und man die Spannung an den Kondensatoren, sowie den Spulenstrom zur Anzeige bringt:

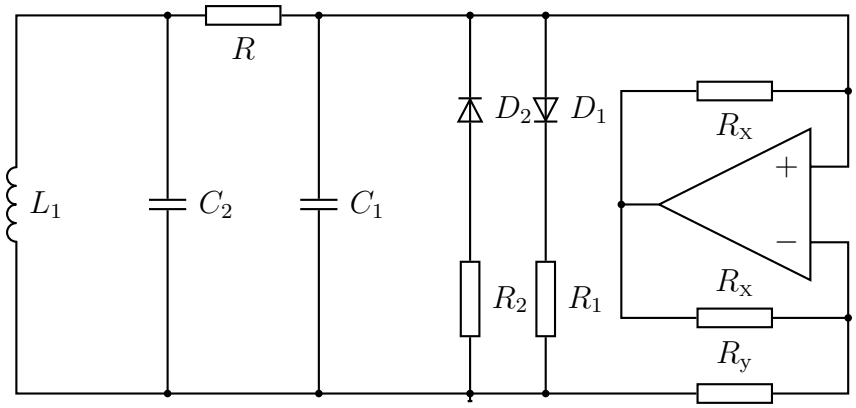


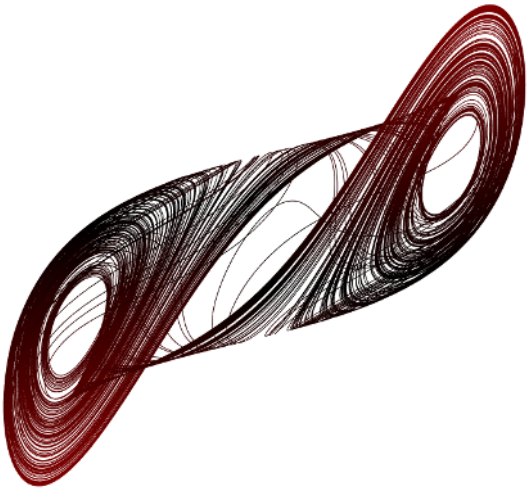
Abbildung 7.4: Schaltungstechnische Realisierung des Chua-Attraktors

Werden nun Knoten- und Maschenregeln auf die Schaltung angewandt, so ergeben sich folgende drei Differentialgleichungen:

$$\begin{aligned}
\dot{u}_{\mathrm{C1}} &= \frac{1}{C_1}\left[\frac{u_{\mathrm{C2}}}{R} - \left(\frac{u_{\mathrm{C1}}}{R} + g\left(u_{\mathrm{C1}}\right)\right)\right] \\
\dot{u}_{\mathrm{C2}} &= \frac{1}{C_2}\left(\frac{u_{\mathrm{C1}}}{R} - \frac{u_{\mathrm{C2}}}{R} + i_{\mathrm{L1}}\right) \\
i'_{\mathrm{L1}} &= -\frac{1}{L_1} u_{\mathrm{C2}}
\end{aligned}$$

Dabei modelliert die Funktion $g\left(u_{\mathrm{C1}}\right)$ das Verhalten des nicht-linearen Widerstandes, wobei natürlich an diesem nicht-linearen Widerstand die gleiche Spannung wie an C_1 anliegt.

Die mathematische, allgemeine Beschreibung des Chua-Attraktors kann lauten:



$$\dot{x} = \alpha\,(y - \phi)$$

$$\dot{y} = x - y + z \tag{7.8}$$

$$\dot{z} = -\beta y$$

$$\phi = m_0 x + \frac{1}{3} m_1 x^3 \tag{7.9}$$

Abbildung 7.5: Trajektorie des Chua-Attraktors im dreidimensionalen Zustandsraum simuliert mit dem Dormand-Prince-Verfahren in Matlab

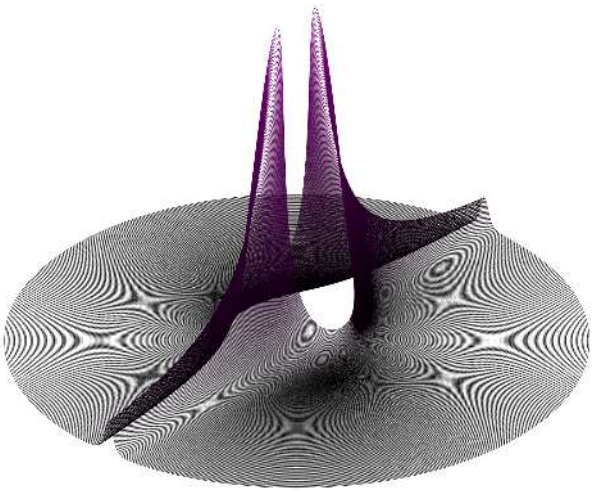
Typische Anfangswerte sind und eine Parameterwahl kann lauten:

$$\alpha = 18 \qquad \beta = 32 \qquad m_0 = -\frac{1}{5} \qquad m_1 = \frac{1}{100} \tag{7.10}$$

$$x(0) = 3 \qquad y(0) = 0 \qquad z(0) = 3 \tag{7.11}$$

7.4 Rabinovich-Attraktor

Dieses System wurde von Mikhail Rabinovich und Anatoly Fabrikant beschrieben und besitzt insgesamt fünf Gleichgewichtspunkte. Der Attraktor weißt, bei bestimmten Parameterkonstellationen, ein chaotisches Verhalten auf und lässt sich mit folgenden drei gekoppelten gewöhnlichen Differentialgleichungen beschreiben:



$$\dot{x} = y\left(z - 1 + x^2\right) + \beta x$$

$$\dot{y} = x\left(3\,z + 1 - x^2\right) + \beta y \quad (7.12)$$

$$\dot{z} = -2\,z\,(\alpha + x\,y)$$

Abbildung 7.6: Trajektorie des Rabinovich-Attraktors im dreidimensionalen Zustandsraum simuliert mit dem Dormand-Prince-Verfahren in Matlab

Eine mögliche Parameterkonstellation, welche zu chaotischem Verhalten führt, ist folgende:

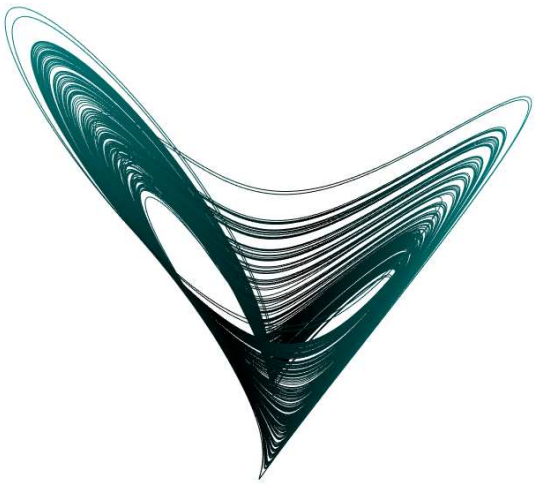
$$\alpha = \frac{14}{100} \qquad \beta = \frac{1}{10} \quad (7.13)$$

Für die Startwerte wurden, für die Simulation des Attraktors, folgende Werte verwendet:

$$x(0) = 10 \qquad y(0) = 10 \qquad z(0) = 10 \quad (7.14)$$

7.5 Chen-Lee-Attraktor

Der Chen-Lee-Attraktor besteht aus drei nichtlinearen Differentialgleichungen und beschreibt eine Kreiselbewegung mit Rückkopplungssteuerung.



$$\dot{x} = -y\,z + \alpha x$$

$$\dot{y} = x\,z - \beta y \tag{7.15}$$

$$\dot{z} = \frac{1}{3} x\,y - \gamma z$$

Abbildung 7.7: Trajektorie des Chen-Lee-Attraktors im dreidimensionalen Zustandsraum simuliert mit dem Dormand-Prince-Verfahren in Matlab

Mit der folgenden Wahl der drei Systemparametern erhält man einen chaotischen Attraktor, welcher eine sensitive Abhängigkeit der Anfangsbedingungen aufweist:

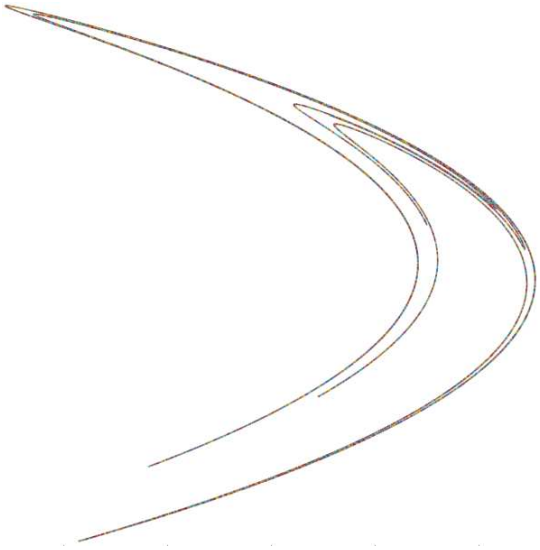
$$\alpha = 5 \qquad \beta = 10 \qquad \gamma = \frac{19}{5} \tag{7.16}$$

Für die Anfangsbedingungen wurden folgende Werte für die Simulation verwendet:

$$x(0) = 5 \qquad y(0) = 0 \qquad z(0) = 5 \tag{7.17}$$

7.6 Hénon-Attraktor

Der Hénon-Attraktor kann im zweidimensionalen Raum als diskretes System beschrieben werden und ist durch folgende Rekursion gegeben:



$$\begin{aligned} x_{n+1} &= y_n + \left(1 - \alpha x_n^2\right) \\ y_{n+1} &= \beta x_n \end{aligned} \tag{7.18}$$

Abbildung 7.8: x/y-Trajektorie des Hénon-Attraktors simuliert durch 100 000 Iterationen in Matlab

Die Hénon Abbildung, wurde von Michel Hénon konstruiert, um selbstvertretend für chaoserzeugende zweidimensionale Systeme die Eigenschaften derartiger Systeme zu untersuchen. Es ist ein System mit einer diskreten Zeitskala $n = 1,\ 2,\ 3,\ \ldots$ und dieser Attraktor weißt mit der nachfolgenden Parameterwahl eine sensitive Abhängigkeit von den Anfangsbedingungen auf.

Für die Parameterwahl:

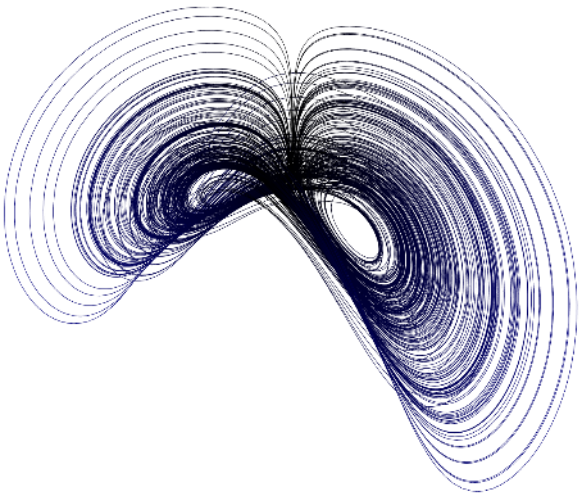
$$\alpha = \frac{7}{5} \qquad\qquad \beta = \frac{3}{10} \tag{7.19}$$

konnte Hénon zeigen, dass das System einen seltsamen Attraktor hat. Typische Anfangswerte sind:

$$x(0) = 0 \qquad\qquad y(0) = 0 \tag{7.20}$$

7.7 Newton-Leipnik-Attraktor

Drei nicht-lineare Differentialgleichungen beschreiben das Newton-Leipnik-System, welches, bei der nachfolgenden Parameterwahl, zu den chaotischen Systemen, mit zwei seltsamen Attraktoren, zählt.



$$\dot{x} = -\alpha x + y + 10\,y\,z$$
$$\dot{y} = -x - 0.4\,y + 5\,x\,z \qquad (7.21)$$
$$\dot{z} = \beta z - 5\,x\,y$$

Abbildung 7.9: Trajektorie des Newton-Leipnik-Attraktors im dreidimensionalen Zustandsraum simuliert mit dem Dormand-Prince-Verfahren in Matlab

Verwendete Parameterkonstellation:

$$\alpha = \frac{2}{5} \qquad \beta = \frac{7}{40} \qquad (7.22)$$

Für die Startwerte wurden, für die Simulation des Attraktors, folgende Werte verwendet:

$$x(0) = 0,349 \qquad y(0) = 0 \qquad z(0) = -0,18 \qquad (7.23)$$

8 Resümee

Dieses Projekt stellt einen Vergleich von den verschiedenen Simulationsverfahren dar und es lässt sich erkennen, dass sowohl das explizite Euler-Verfahren, als auch das Runge-Kutta-Verfahren zur Berechnung von dynamischen System Anwendung finden kann, jedoch ergeben sich, vorallem nach längerer Simulationszeit, immer größere Abweichungen, welche sich dadurch begründen lassen, dass das Runge-Kutta-Verfahren mehrere Steigungen in einem Zeitschritt und nicht nur die Anfangssteigung, wie das Euler-Verfahren, berechnet.

Des Weiteren wurde beim Lorenz-Attraktor ersichtlich, dass sich sehr rasch Abweichungen zwischen Simulationsverfahren ergeben, aber beim Reibungsschwinger war auch zu erkennen, dass nicht nur das Verfahren, sondern auch andere Einstellungen im Programm bzw. in den einzelnen Funktionsblöcken selbst für die Berechnung von Bedeutung sind.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] **Riccardo Caponetto:** Fractional Order Systems. Modeling and Control Applications. Singapure 2010, 1. Auflage,
World Scientific Publishing Co. Pte. Ltd., ISBN-13: 978-981-4304-19-1

[2] **Wilhelm Haager:** Regelungstechnik. Wien 2007, 2. Auflage,
Hölder-Pichler-Tempsky GmbH Verlag, ISBN: 978-3-203-02565-4

Quellenverzeichnis

[1] http://ito.mathematik.uni-halle.de/∼julitz/

Abkürzungsverzeichnis

Abb.	Abbildung
BDF	backward differentiation formulas
bzw.	beziehungsweise
ca.	circa
Co.	company
Dipl.-Ing.	Diplom-Ingenieur
DOPRI5	Dormand-Prince 4(5)
Dr.	Doktor
engl.	englisch
etc.	et cetera
GmbH	Gesellschaft mit beschränkter Haftung
HTBL u. VA	höhere technische Bundeslehr- und Versuchsanstalt
JDK	Java Development Kit
JRE	Java Runtime Environment
Kap.	Kapitel
Ltd.	limited (company)
ODE	ordinary differential equation
Pte.	private (company)
RKDP	Runge-Kutta-Dormand-Prince
Sundials	suite of nonlinear and differential/algebraic equation solvers
z. B.	zum Beispiel