1 Systemanalyse mit Matlab/Simulink

Ziel dieser Übung ist es, das Computerprogramm MATLAB/SIMULINK für die Systemanalyse einzusetzen. Alle Aufgabenstellungen dieser Übungseinheit sind mit diesem Softwarepaket zu lösen.

Eine kostenlose MATLAB-Lizenz wird von der TU-Wien zur Verfügung gestellt. Im Computerlabor des Instituts steht MATLAB/SIMULINK in der Version R2025a zur Verfügung. Informationen zum Installieren von MATLAB finden Sie im als QR-Code dargestellten Link. Dieser Link ist im PDF auch anklickbar.

Studieren Sie als Vorbereitung auf die Übung zumindest folgende Kapitel im Skriptum zur VO Automatisierung (WS 2025/26) [1.1]:

- Kapitel 1, vollständig
- Kapitel 2, vollständig
- Kapitel 3, vollständig
- Kapitel 6, vollständig.

Bei Fragen oder Anregungen zu dieser Übung wenden Sie sich bitte an

- Andreas Fischer <fischer2@acin.tuwien.ac.at> oder
- Katharina Schrom <schrom@acin.tuwien.ac.at>.

Für organisatorische Fragen wenden Sie sich bitte an

- Michael Girsch <girsch@acin.tuwien.ac.at>.

Hinweis: Lesen und beachten Sie alle Hinweise.

1.1 Matlab

MATLAB ist ein Computernumerikprogramm. Der Name ist eine Abkürzung für MATrix LABoratory, womit bereits angedeutet wird, dass das Programm gut zum Rechnen mit Vektoren und Matrizen geeignet ist.

Der MATLAB-Desktop (das eigentliche Programmfenster) enthält in der Standardeinstellung folgende Fenster. (Dies kann jedoch individuell angepasst werden.)

- **Command Window**
  Es stellt den Eingabebereich dar, welcher auf jeden Fall angezeigt werden muss. Hier können alle Befehle hinter der Eingabeaufforderung >> eingegeben und direkt ausgeführt werden. Schließt man eine Befehlssequenz mit einem Semikolon ab, so wird die Ausgabe von MATLAB unterdrückt. Das Drücken der Eingabe-Taste bewirkt die sofortige Ausführung der Befehlszeile. Im Falle mehrzeiliger Befehlseingaben kann mit der Umschalt- und der Eingabe-Taste in eine neue Zeile gesprungen werden.

- **Editor**
  Im Editor können Funktionen, Skripte oder Programm-Code (z. B. `m`-Code oder auch C-Code) erstellt und editiert werden. Es werden die in Programmierumgebungen üblichen Möglichkeiten zum schrittweisen Ausführen der Befehlssequenzen, zum Debuggen, etc. zur Verfügung gestellt. Skripte werden auch `m`-files genannt; sie besitzen die Dateiendung `*.m` und werden zur Laufzeit von einem Interpreter abgearbeitet. Skripte enthalten MATLAB-Befehlssequenzen, wie sie prinzipiell auch direkt im Command Window eingegeben werden können. Funktionen besitzen ebenfalls die Dateiendung `*.m`. Sie erhalten meist Eingangsparameter und geben oft auch Rückgabewerte zurück.

- **Workspace Browser**
  Die aktuellen Variablen werden in MATLAB im so genannten Workspace angezeigt. Sie können durch Anklicken aufgerufen und verändert werden.

- **Current Directory Browser**
  Im Current Directory Browser wird das aktuelle Arbeitsverzeichnis dargestellt. Dateien und Ordner können geöffnet, angelegt, bearbeitet, etc. werden. Für kleine Projekte empfiehlt es sich, alle Dateien, auf die während der Rechnung oder Simulation zugegriffen wird, in einem gemeinsamen, lokalen (aktuellen) Verzeichnis abzulegen.

- **Command History Window**
  Die in der Vergangenheit ausgeführten Befehle sind im Command History Window chronologisch sortiert. Die einzelnen Befehle können herauskopiert bzw. durch einen Doppelklick erneut ausgeführt werden.

Alle MATLAB/SIMULINK Dateien, die zum Bearbeiten dieser Übung benötigt werden, finden Sie in `uebung1.zip` auf der Homepage der Lehrveranstaltung.

1.1.1 Grundlagen in Matlab

Auf der Homepage der Lehrveranstaltung finden Sie die MATLAB Scripts `cds_matlab_intro_part1.m`, `cds_matlab_intro_part2.m` und `cds_matlab_intro_part3.m`. Öffnen Sie diese und führen Sie die enthaltenen Befehle schrittweise aus. Beachten Sie, dass die Funktion `mittelwert.m` aufgerufen wird, welche sich daher im aktuellen Arbeitsverzeichnis befinden muss.

Hinweis: Zum Ausführen einzelner Befehlssequenzen markieren Sie diese im Editor und drücken F9. Mithilfe der Symbolfolge `%%␣` kann der Programmcode in einzeln ausführbare Sektionen unterteilt werden, welche durch die Tastenkombination *Strg* + *Enter* abgearbeitet werden. Zum Ausführen eines ganzen `m`-files geben Sie entweder dessen Namen (ohne Dateiendung) im Command Window ein oder öffnen Sie die Datei im Editor und drücken F5. Versuchen Sie alle Befehle zu verstehen und machen Sie gegebenenfalls von der Hilfefunktion Gebrauch.

Hinweis: Machen Sie sich auch mit den in `cds_matlab_intro_part1.m` angeführten function handles vertraut. Diese sind für die Zusatzaufgaben in der Übung von Vorteil.

Aufgabe 1.1. In den folgenden Aufgaben sollen die in `cds_matlab_intro_part1.m` und `cds_matlab_intro_part2.m` beschriebenen Befehle an konkreten Beispielen angewandt werden.

1. Gegeben ist das Gleichungssystem

$$x_1 + 2x_2 + 4x_3 = 5 \tag{1.1a}$$
$$2x_1 + 2x_2 + x_3 = 4 \tag{1.1b}$$
$$3x_1 + 2x_2 = 2\ . \tag{1.1c}$$

   Schreiben Sie dieses Gleichungssystem in Matrixdarstellung an und bestimmen Sie den Lösungsvektor $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^{\mathrm{T}}$. Führen Sie die Rechnung einmal mit dem Befehl `inv()` und einmal mit dem Befehl `mldivide()` (oder in seiner Kurzform \) durch. Untersuchen Sie die Geschwindigkeits- und Genauigkeitsunterschiede der beiden Befehle. Wann würden Sie welchen Befehl verwenden?

   Hinweis: Um die benötigte Rechenzeit zu bestimmen, können die Befehle `tic` und `toc` verwendet werden. Weiters kann mithilfe von `norm`$(\mathbf{Ax} - \mathbf{b})$ der numerische Fehler der Berechnung bestimmt werden.

2. Gegeben ist die Fourier-Reihenentwicklung n-ter Ordnung der Rechteckfunktion

$$\mathrm{rect}(x) \approx A \sum_{k=1}^{n} \frac{4}{\pi(2k-1)} \sin((2k-1)x)\ , \tag{1.2}$$

wobei $A = 10$ die Amplitude bezeichnet. Stellen Sie diese Rechteckfunktion für $n = 1, 2, \ldots, 100$ im Intervall $x \in [0, 10]$ mithilfe einer `for`-Schleife dar. Nutzen Sie zur schrittweisen grafischen Darstellung der Funktion in der `for`-Schleife den `pause`-Befehl.

3. Gegeben ist die Funktion

$$f(x, y, t) = \sin\left(\frac{x\pi}{10}\right) \sin\left(\frac{y\pi}{10}\right) |\sin(t)| \ . \tag{1.3}$$

Stellen Sie die zu dieser Funktion gehörige Fläche $z = f(x, y, t_{\text{konst}})$ mithilfe einer `for`-Schleife für verschiedene Zeitpunkte t_{konst} sowie $x \in [0, 10]$ und $y \in [0, 10]$ grafisch dar.

1.1.2 Control System Toolbox

Toolboxen sind Sammlungen von Funktionen, meist in Form von `m`-files, die den Funktionsumfang des Basisprogramms erweitern. Nach der erstmaligen Installation werden Toolboxen automatisch beim Programmstart von MATLAB geladen. Eine Übersicht der installierten Toolboxen erhält man mit dem Befehl `ver`.

Die Toolbox 'Control System' ist häufig bei regelungstechnischen Aufgabenstellungen nützlich. Sie unterstützt bei der Analyse und dem Reglerentwurf von linearen dynamischen Systemen.

Hinweis: Vor der Bearbeitung der nachstehenden Aufgaben öffnen Sie die Datei `cds_matlab_intro_part3.m` im MATLAB-Editor und arbeiten Sie alle Befehle schrittweise durch. Versuchen Sie alle Befehle zu verstehen und machen Sie gegebenenfalls von der Hilfefunktion Gebrauch.

Zur Bestimmung der numerischen Lösung eines (nichtlinearen) Differentialgleichungssystems der Form

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \tag{1.4}$$

werden in MATLAB zahlreiche Integrationsalgorithmen zur Verfügung gestellt. Die Funktion eines solchen Integrationsalgorithmus soll anhand des expliziten Euler-Verfahrens

$$\mathbf{x}_{k+1} = \mathbf{x}_k + T_a \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) \tag{1.5}$$

gezeigt werden. Hierbei bezeichnet $\mathbf{x}_k \approx \mathbf{x}(kT_a)$ die Approximation des Zustandsvektors zum Zeitpunkt $t = kT_a$ mit der Abtastzeit T_a.

Aufgabe 1.2. Betrachten Sie die Differentialgleichung zweiter Ordnung zur Beschreibung eines harmonischen Oszillators in der Form

$$m\ddot{s} + c\dot{s} + ks = f \ , \qquad s(0) = s_0, \quad \dot{s}(0) = v_0 \tag{1.6}$$

mit der Masse m, der Auslenkung s, der Dämpfungskonstante c, der Federkonstante k sowie der externen Kraft f. Schreiben Sie (1.6) in Zustandsraumdarstellung

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{1.7a}$$

$$y = \mathbf{c}^{\mathrm{T}}\mathbf{x} \tag{1.7b}$$

mit $\mathbf{x}^{\mathrm{T}} = \begin{bmatrix} x_1, x_2 \end{bmatrix}$ und $\mathbf{u} = u$ an. Wählen sie hierfür $u = f$ und $y = s$.

1. Bestimmen Sie die Eigenwerte der Dynamikmatrix und beurteilen Sie das System hinsichtlich Stabilität unter Annahme positiver Konstanten m, c und k, sowie $c^2 \leq 4km$. Berechnen Sie die Transitionsmatrix $\mathbf{\Phi}(t)$ und bestimmen Sie die allgemeine Lösung von (1.7) für den Fall, dass u durch den Einheitssprung $\sigma(t)$ gegeben ist. Diese Berechnungen sind handschriftlich durchzuführen!

   Hinweis: Für die Berechnung der Transitionsmatrix können die Eigenwerte allgemein mit λ_1 und λ_2 bezeichnet werden. Die zuvor berechneten Ausdrücke müssen nicht explizit eingesetzt werden.

2. Schreiben Sie ein `m`-file, in welchem der Integrationsalgorithmus (1.5) auf das Differentialgleichungssystem (1.7) angewendet wird. Verwenden Sie als Integrationszeitraum $[0, 20]$s, als Zeitschrittweite (Abtastzeit) $T_a = 0.1\,\mathrm{s}$ und für die Systemparameter $m = 1\,\mathrm{kg}$, $c = 1\,\mathrm{N\,s/m}$, $k = 2\,\mathrm{N/m}$, $s_0 = 0\,\mathrm{m}$, $v_0 = 0\,\mathrm{m/s}$. Variieren Sie nun die Abtastzeit und untersuchen Sie deren Einfluss auf die Genauigkeit der numerisch berechneten Lösung. Stellen Sie hierzu den Zeitverlauf der Zustände, sowohl der numerischen als auch der exakten Lösung, in einer Grafik dar.

3. Vergleichen Sie Ihre Ergebnisse mit den mittels Control System Toolbox berechneten Sprungantworten des gegebenen kontinuierlichen Systems und des zugehörigen Abtastsystems (Abtastzeit T_a). Zur Diskretisierung können Sie den Befehl `c2d()` verwenden.

1.2 Simulink

Simulink ist eine Erweiterung von Matlab zur Simulation und Analyse dynamischer Systeme. Mithilfe von Simulink gestaltet sich das Lösen von Anfangswertproblemen einfach. Die grafische Bedienoberfläche erlaubt die Beschreibung von dynamischen Systemen mithilfe von Blockschaltbildern. Hierbei stellt Simulink eine Bibliothek mit vorgefertigten Funktionsblöcken zur Verfügung, welche durch benutzerdefinierte Blöcke erweitert werden können.

1.2.1 Grundlagen von Simulink

Mit dem Befehl `simulink` wird der 'Simulink Library Browser' geöffnet. Er enthält die Funktionsblöcke, welche per Drag & Drop in das Simulink Modell gezogen werden können.

Die Blöcke sind mittels Signalflussleitungen zu verbinden. Besonders häufig benötigte Blöcke sind in der Gliederung des Simulink Library Browsers, z. B. in den folgenden Gruppen zu finden.

- `Continuous`: Blöcke zur Simulation zeitkontinuierlicher Systeme, unter anderem der zeitkontinuierliche Integrator `1/s`.
- `Math Operations`: Einige mathematische Operationen, z. B. Addieren, Multiplizieren, Quadrieren.
- `Sinks`: Blöcke, die nur einen Eingang (oder mehrere Eingänge), aber keinen Ausgang besitzen. `Scopes` beispielsweise dienen zur grafischen Darstellung von Signalen. Signale können an beliebiger Stelle direkt von Signalflussleitungen abgegriffen werden. Der Block `To Workspace` erlaubt den Export von Simulationsergebnissen in den MATLAB-Workspace, wo sie dann als Variable zur weiteren Auswertung zur Verfügung stehen.
- `Sources`: Blöcke, die nur einen Ausgang (oder mehrere Ausgänge), aber keinen Eingang besitzen, wie etwa Signalgeneratoren.

Zur effizienten Arbeit mit SIMULINK wird folgendes Vorgehen empfohlen:

1. Parameterwerte werden nicht direkt in SIMULINK eingetragen, sondern zusammengefasst in einem MATLAB-Script definiert, welches vor der Simulation ausgeführt wird. Damit können die Parameter einfach und an zentraler Stelle geändert werden. Zum Update der Parameter muss das Matlab-Script erneut ausgeführt werden.

   > **Hinweis:** Alle Variablen, die sich im Matlab-Workspace befinden, stehen auch in SIMULINK zur Verfügung.

2. Werden die mathematischen Ausdrücke umfangreicher, empfiehlt es sich, die Verschaltung vieler Einzelblöcke durch die Verwendung benutzerdefinierter (programmierter) Blöcke zu umgehen. Die entsprechenden Blöcke befinden sich in der Gruppe `User-Defined Functions`. Im Block `Fcn` können sowohl MATLAB Funktionen als auch benutzerdefinierte Funktionen verwendet werden.
3. Eine weitere Möglichkeit die Übersichtlichkeit von Modellen zu verbessern ist die Verwendung von Subsystemen (`Ports & Subsystems` → `Subsystem`).
4. Um die Anzahl der am Bildschirm dargestellten Signalflussleitungen zu reduzieren, können die Blöcke `From` und `Goto` aus der Gruppe `Signal Routing` verwendet werden.
5. LTI-Systeme, welche mittels der Control System Toolbox als Übertragungsfunktion oder in Zustandsraumdarstellung erzeugt wurden, können direkt in SIMULINK eingebunden werden (`Control System Toolbox` → `LTI System`).

Simulationseinstellungen können im Menü *Modeling → Model Settings (Strg + E)* vorgenommen werden. Besonders wesentlich ist dabei die Wahl der Simulationsdauer und des Integrationsalgorithmus. Ferner können unter *Solver details* für den Integrationsalgorithmus Schranken der Zeitschrittweite und Genauigkeitsanforderungen eingestellt werden. Im Rahmen dieser Einführung soll auf eine detaillierte Diskussion der verwendeten Algorithmen verzichtet werden. Einen Überblick über einige in MATLAB zur Verfügung stehende Lösungsverfahren für Anfangswertprobleme erhalten Sie in der Hilfe zu '*ode23, ode45, ode113, ode15s, ode23s, ode23t, ode23tb*' (aufrufbar z. B. mit `doc ode45`) unter der Überschrift *Algorithms*. Diese Algorithmen werden auch von SIMULINK verwendet. Ferner sei auf die Fachliteratur, z. B. [1.2, 1.3], verwiesen.

Die Simulation kann durch Anklicken des Startknopfes ▶ oder durch Drücken der Tasten *Strg* + *T* gestartet werden. Um eine Simulation alternativ aus dem Eingabefenster oder einem `m`-file zu starten, kann der Befehl `sim()` verwendet werden.

Aufgabe 1.3. Machen Sie sich weiter mit Simulink vertraut.

Auf der Homepage der Lehrveranstaltung finden Sie das Simulink-Modell `cds_simulink_intro_part1.slx`.

Öffnen Sie dieses Modell und starten Sie die Simulation. Achten Sie darauf, zuvor das *Matlab-Script* `init_cds_simulink_intro_part1.m` auszuführen. Dieses *Matlab-Script* erstellt alle für die Simulation notwendigen Variablen im MATLAB-Workspace. Versuchen Sie alle Blöcke zu verstehen und machen Sie gegebenenfalls von der Hilfefunktion Gebrauch.

1.2.2 Implementierung von dynamischen Systemen

Die Implementierung eines dynamischen Systems, für das ein Modell in Form einer (expliziten) Differentialgleichung existiert, kann als Blockschaltbild oder auch als MATLAB Function erfolgen. Die beiden Varianten werden im Folgenden kurz erläutert.

Blockschaltbild

Um eine Differentialgleichung höherer Ordnung mit Simulink lösen zu können, muss diese als System von Differentialgleichungen erster Ordnung dargestellt werden. Dieses Vorgehen wird in Beispiel 1.1 dargestellt.

Beispiel 1.1. Das Anfangswertproblem

$$\dddot{y} + \cos^2(y)\ddot{y} + a\dot{y} + u = 0\,, \qquad \ddot{y}(0) = \dot{y}(0) = y(0) = 0\,, \qquad u = \sin(t) \tag{1.8}$$

mit einem Parameter $a = 0.3$ soll mittels einer Integratorkette dargestellt werden. Zuerst wird die Differentialgleichung in Zustandsraumdarstellung $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, u)$ übergeführt. Da (1.8) eine Differentialgleichung dritter Ordnung ist, wird ein dreidi-

mensionaler Zustand

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y \\ \dot{y} \\ \ddot{y} \end{bmatrix} \tag{1.9}$$

verwendet. Damit ergibt sich ein System von drei Differentialgleichungen erster Ordnung

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} x_2 \\ x_3 \\ -\cos^2(x_1)x_3 - ax_2 - u \end{bmatrix}, \qquad \mathbf{x}(0) = \begin{bmatrix} x_1(0) \\ x_2(0) \\ x_3(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \tag{1.10}$$

Da man an der Lösung $\mathbf{x}(t)$ interessiert ist, wird (1.10) einmal zeitlich integriert. Dadurch ergibt sich

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} = \begin{bmatrix} x_1(0) + \int_0^t x_2 \,\mathrm{d}\tau \\ x_2(0) + \int_0^t x_3 \,\mathrm{d}\tau \\ x_3(0) + \int_0^t -\cos^2(x_1)x_3 - ax_2 - u \,\mathrm{d}\tau \end{bmatrix}, \qquad \mathbf{x}(0) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{1.11}$$

In (1.11) ist ersichtlich, dass drei Integratoren benötigt werden. Jeder dieser Integratoren kann mittels eines Integratorblocks in Simulink dargestellt werden. Damit kann das Differentialgleichungssystem in Simulink wie in Abbildung 1.1 dargestellt werden.

Hinweis: Die Anfangszustände $(x_{1,0}, x_{2,0}, x_{3,0})$ müssen in den Einstellungen der einzelnen Integratorblöcke definiert werden.

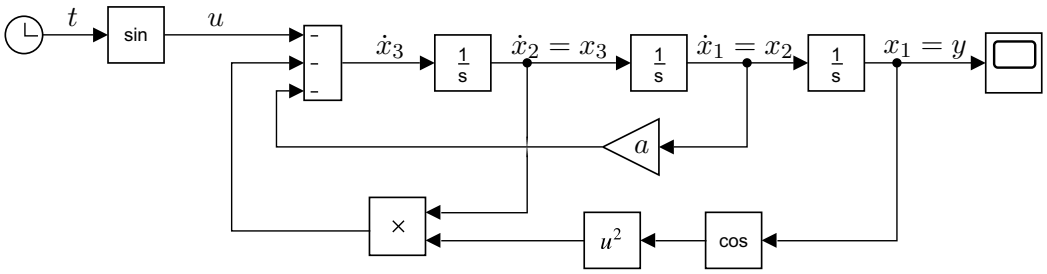

Abbildung 1.1: Implementierung von (1.11) mittels einer Integratorkette.

Aufgabe 1.4. Betrachten Sie für die nachstehenden Teilaufgaben die nichtlineare Differentialgleichung

$$\dddot{x} + \cos(x)^2\ddot{x} + \dot{x} + \mathrm{e}^{-x}u = 0\,. \tag{1.12}$$

1. Ermitteln Sie die numerische Lösung der Differentialgleichung mit dem Zustand x, dem Eingang u und den Anfangsbedingungen $\ddot{x}(0) = 1$, $\dot{x}(0) = -2$ und $x(0) = 3$. Formen Sie dazu die Differentialgleichung in ein Differentialgleichungssystem erster Ordnung um und implementieren Sie das entsprechende Blockschaltbild. Testen Sie die Simulation für $u(t) = \sin(t)$.

2. Linearisieren Sie das Differentialgleichungssystem erster Ordnung um die Ruhelage $x_R = 0$, $u_R = 0$ und überprüfen Sie die Stabilität des linearen Systems. Diese Berechnungen sind händisch durchzuführen.

Control System Toolbox in Simulink

Eine einfache Möglichkeit der Implementierung linearer zeitinvarianter Systeme bietet die Control System Toolbox. Diese kann für zeitkontinuierliche wie auch für zeitdiskrete LTI-Systeme verwendet werden. Dabei können die vordefinierten dynamischen Systeme (siehe auch Abschnitt 1.1.2) sowohl als Übertragungsfunktion, als auch direkt in Zustandsraumdarstellung

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \quad \mathbf{x}(t_0) = \mathbf{x}_0 \tag{1.13a}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \tag{1.13b}$$

angegeben werden. Es bietet sich dabei insbesondere der Block '`LTI System`' aus der Control System Toolbox an, in welchem das System in beliebiger Darstellung der Control System Toolbox eingetragen werden kann.

Matlab Function für zeitkontinuierliche Systeme

Die Implementierung einer Differentialgleichung mittels Integratorkette wird bei hochdimensionalen Problemen und steigender mathematischer Komplexität unübersichtlich. Durch das Benutzen von MATLAB Function Blöcken kann dies verhindert werden. Das Benutzen von MATLAB Function Blöcken wird in Beispiel 1.2 erläutert.

Beispiel 1.2. Das Anfangswertproblem (1.8) soll mittels eines MATLAB Function Blocks gelöst werden. Wird das System in Zustandsraumdarstellung $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, u)$ übergeführt, so kann mittels Integration

$$\mathbf{x}(t) = \int_0^t \mathbf{f}(\mathbf{x}(t), u(t))\,\mathrm{d}\tau + \mathbf{x}_0 \tag{1.14}$$

der Zustand $\mathbf{x}(t)$ bestimmt werden. Die vektorielle Funktion $\mathbf{f}(\mathbf{x}, u)$ wird nun als MATLAB Function in Simulink implementiert. Dazu wird in Simulink ein MATLAB Function Block (`User Defined Functions` → MATLAB Function) eingefügt. Durch Doppelklick öffnet sich ein Fenster im MATLAB-Editor, in welchem programmiert werden kann. Die Differentialgleichung kann mittels dieses Programmcodes

```
function [xxpunkt] = fcn(u,xx,param)
  % Aufteilen der Parameter (zur besseren Lesbarkeit)
  a = param.a;
```

```
    % Aufteilen des Zustandsvektors (zur besseren Lesbarkeit)
    x1 = xx(1);
    x2 = xx(2);
    x3 = xx(3);

    % Implementierung von f(x,u)
    f1 = x2;
    f2 = x3;
    f3 = -cos(x1)^2 * x3 - a*x2 -u;

    % Zuweisen des Ausgangs des Blocks f(x,u)
    ff = [f1;
          f2;
          f3];
end
```

implementiert werden. Hierbei besitzt die Funktion `fcn` drei Übergabeparameter (`xx`, `u`, `param`) und einen Rückgabewert (`xxpunkt`). Ist der MATLAB Function Block im MATLAB Editor geöffnet, so kann man im Reiter `Editor` mit dem Button `Edit Data` den `Ports and Data Manager` öffnen. In diesem kann eingestellt werden, ob ein Übergabeparameter als Eingang oder als Parameter behandelt werden soll. Parameter müssen im MATLAB Workspace definiert werden. Es ist empfehlenswert, alle benötigten Parameter in einem *Struct Array* zu organisieren. Hierbei ist es wichtig, dass im *Ports and Data Manager* eingestellt wird, dass der Parameter nicht *Tunable* ist.

Die Integration von `xxpunkt` wird nun mittels eines Integratorblocks in Simulink durchgeführt. Dieser Block kann mit vektorwärtigen Signalen umgehen. Man beachte, dass die Dimension des Anfangszustandes des Integratorblocks der Dimension des Zustandes des Systems entsprechen muss.

Der Systemausgang $y = h(\mathbf{x}, u)$ wird nun mittels eines weiteren MATLAB Function Blocks implementiert. Dieser benötigt im Allgemeinen die Systemparameter, den Eingang u sowie den Zustand $\mathbf{x}(t)$. Im gegebenen Beispiel ist dies durch den Programmcode

```
function [y] = fcn(u,xx,param)
    % Aufteilen der Parameter (zur besseren Lesbarkeit)
    a = param.a;

    % Aufteilen des Zustandsvektors (zur besseren Lesbarkeit)
    x1 = xx(1);
    x2 = xx(2);
    x3 = xx(3);

    % Implementierung von h(x,u)
    y = x1;
end
```

möglich. Die Abbildung 1.2 zeigt die vollständige Implementierung mittels MATLAB Function Blöcken.

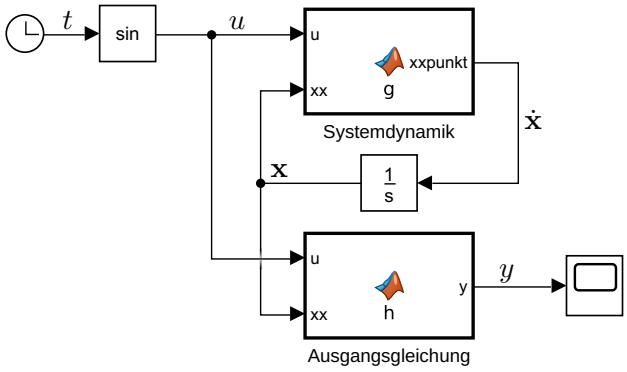

Abbildung 1.2: Implementierung von (1.11) mittels einer MATLAB Function.

Matlab Function für zeitdiskrete Systeme

Auch zeitdiskrete Systeme können mittels MATLAB Function Blöcke implementiert werden. Hierbei gibt es mehrere Möglichkeiten. Die im Folgenden erklärte Methode bietet den Vorteil, dass der resultierende Programmcode direkt in einem Digitalrechner implementiert werden kann. Dadurch können entworfene Regler oder Beobachter mit geringem Zeitaufwand auf industriellen Anlagen oder auf Versuchsaufbauten portiert werden. Im Speziellen ist es auf einfache Weise möglich, die MATLAB Function direkt in der Rapid Prototyping System dSPACE zu verwenden.

In Beispiel 1.3 wird erklärt, wie ein zeitdiskretes System mittels MATLAB Function Blöcken in Simulink implementiert wird.

Beispiel 1.3. Das Anfangswertproblem (1.8) soll mittels des Euler-vorwärts-Verfahrens diskretisiert werden. Das diskretisierte System soll als zeitdiskretes System mit der Abtastzeit $T_a = 100\,\text{ms}$ simuliert werden. Zuerst wird analog zu Beispiel 1.2 das System in Zustandsraumdarstellung übergeführt und zeitlich integriert (vgl. (1.14)). Das Integral (1.14) wird jedoch mittels des Euler-vorwärts-Verfahrens approximiert. Die sich ergebende Differenzengleichung

$$\mathbf{x}_{k+1} = \mathbf{x}_k + T_a \begin{bmatrix} x_{2,k} \\ x_{3,k} \\ -\cos^2(x_{1,k})x_{3,k} - a x_{2,k} - u_k \end{bmatrix}, \qquad \mathbf{x}_0 = \mathbf{x}(0) \tag{1.15}$$

kann nun direkt in einer MATLAB Function mit dem Programmcode

```
function y = fcn(u,param)
  % Aufteilen der Parameter
  a = param.a;
  Ta = param.Ta;

  % Definition des diskreten Zustandes als persistent Variable
  persistent xx;
```

```
    % Initialisierung des diskreten Zustandes
    if isempty(xx); xx=param.xx0; end

    % Aufteilen des Zustandsvektors
    x1 = xx(1);
    x2 = xx(2);
    x3 = xx(3);

    % Implementierung von f(x,u)
    f1 = x2;
    f2 = x3;
    f3 = -cos(x1)^2*x3 -a*x2 -u;

    % Zusammenfuegen zu einem Vektor
    ff = [f1;
          f2;
          f3];

    % Approximation mittels Euler-Verfahren
    xx = xx + Ta*ff;

    % Implementierung von h(x,u)
    y = x1;
end
```

implementiert werden. Hierbei wird der Zustand `xx` als *persistent* Variable definiert, welche über mehrere Funktionsaufrufe erhalten bleiben. Beim Erstellen dieser Variable ist diese leer (das bedeutet das Gleiche wie die Zuweisung `xx=[]`). Man beachte, dass im ersten Aufruf des MATLAB Function Blocks die Variable `xx` mit dem Anfangswert *xx0* initialisiert wird. Wird ein zeitdiskretes System in SIMULINK implementiert, so ist es weiters notwendig die Abtastzeit zu spezifizieren. Dies ist möglich, indem man im MATLAB Function Block mittels *Edit Data* den *Ports and Data Manager* öffnet und die *Update Method* als *Discrete* festlegt, siehe dazu Abbildung 1.3.

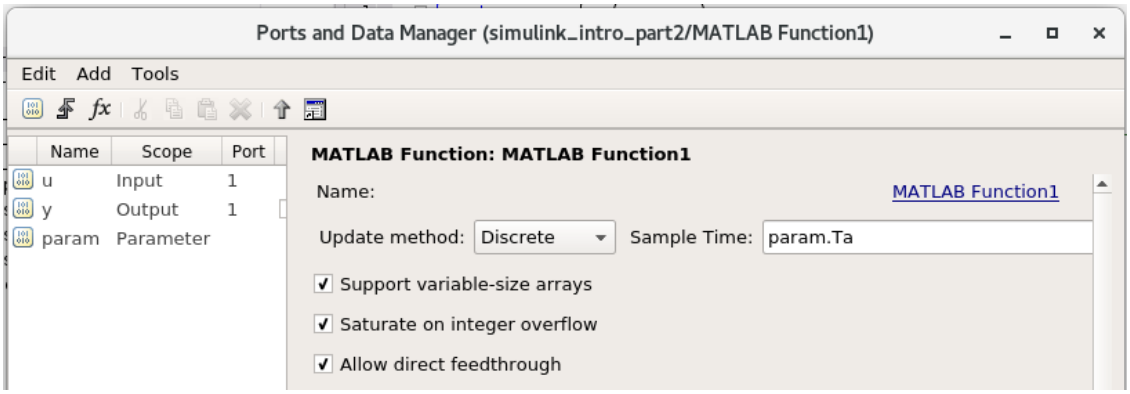

Abbildung 1.3: Einstellungen im *Ports and Data Manager* für zeitdiskrete Systeme.

Aufgabe 1.5. Machen Sie sich mit der Implementierung dynamischer Systeme vertraut.

Auf der Homepage der Lehrveranstaltung finden Sie das Simulink-Modell `simulink_intro_part2.slx`. In diesem Simulinkmodell sind Beispiel 1.1, Beispiel 1.2 und Beispiel 1.3 implementiert.

Öffnen Sie dieses Modell und starten Sie die Simulation. Achten Sie darauf, zuvor das MATLAB-Script `init_cds_simulink_intro_part2.m` auszuführen. Dieses MATLAB-Script erstellt alle für die Simulation notwendigen Variablen im MATLAB-Workspace. Versuchen Sie alle Blöcke zu verstehen und machen Sie gegebenenfalls von der Hilfefunktion Gebrauch. Achten Sie besonders auf die Einstellungen welche im *Ports and Data Manager* getroffen wurden.

1.3 Anwendungsbeispiele

In diesem Abschnitt wird MATLAB zur Simulation und Analyse konkreter Anwendungsbeispiele genutzt. Die folgenden Beispiele sind so konzipiert, dass die Herleitung der mathematischen Modelle händisch durchgeführt werden kann.

1.3.1 Elektrisches Netzwerk

In der folgenden Aufgabe soll das elektrische System aus Abbildung 1.4 untersucht werden. Das betrachtete Netzwerk besteht aus einem idealen Operationsverstärker, einer nichtlinearen Kapazität $C_1(U_{C1})$, einer linearen Kapazität C_2 sowie den linearen Widerständen R_1, R_2 und R_3. Weiterhin bezeichnet U_s eine auftretende Störung.

Hinweis: Unter einem idealen Operationsverstärker versteht man einen Verstärker, bei dem die beiden Eingangsströme $i_{\text{ein},p}$ und $i_{\text{ein},n}$ sowie die Differenzspannung u_d gleich Null sind.

Zur Herleitung der Differentialgleichungen benötigt man 3 Knotengleichungen und 5 Maschengleichungen. Der Strom i_{R4} ist jener Strom, der durch den Widerstand $R_4 = (K-1)R_3$ fließt, und die Spannung u_{R4} entspricht dem Spannungsabfall am Widerstand R_4.

Folgend sind die entsprechenden Gleichungen für eine mögliche Wahl von Knoten und Maschen angeschrieben.

- Knotengleichungen:

$$i_{C1} = i_{R1} - i_{R2} \tag{1.16a}$$
$$i_{C2} = i_{R2} + i_{s,R2} \tag{1.16b}$$
$$i_{R3} = i_{R4} \tag{1.16c}$$

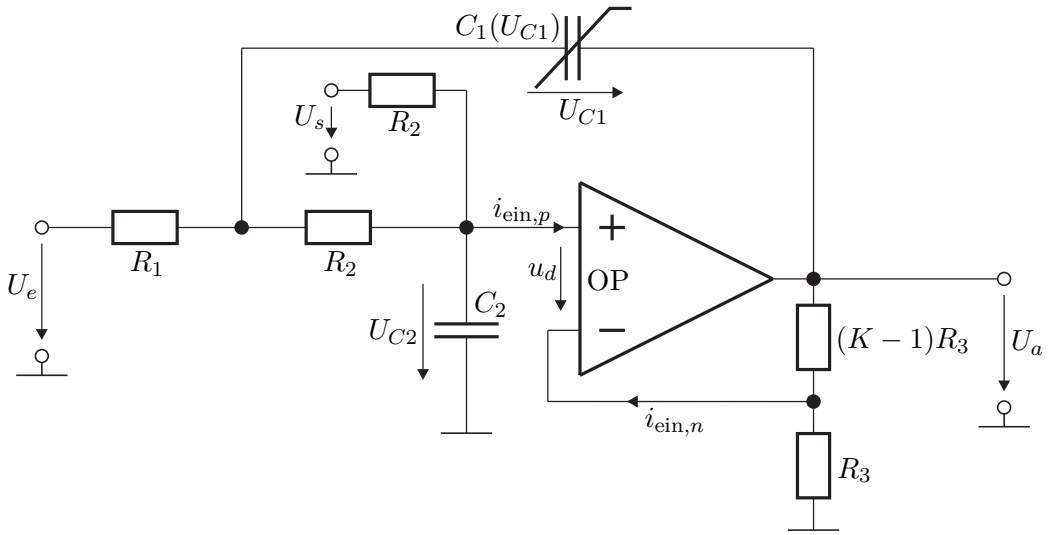

Abbildung 1.4: Elektrisches System.

- Maschengleichungen:

$$U_e = U_{R1} + U_{R2} + U_{C2} \tag{1.17a}$$
$$0 = U_{C1} + U_{R4} + U_{R3} - U_{C2} - U_{R2} \tag{1.17b}$$
$$U_{R3} = U_{C2} \tag{1.17c}$$
$$U_s = U_{s,R2} + U_{C2} \tag{1.17d}$$
$$U_a = U_{R3} + U_{R4} \tag{1.17e}$$

Aufgabe 1.6 (Knoten und Maschen).

1. Welche Knoten und welche Maschen wurden hier zum Anschreiben der Gleichungen gewählt? Zeichnen Sie diese in Abbildung 1.4 ein.

Es werden auch die Beziehungen zwischen Strom und Spannung an den Bauteilen benötigt, welche auch als Bauteilgleichungen bezeichnet werden. Für die Ohmschen Widerstände gilt das Ohmsche Gesetz, woraus die ersten fünf Bauteilgleichungen folgen.

- Bauteilgleichungen für die Widerstände:

$$U_{R1} = R_1 i_{R1} \tag{1.18a}$$
$$U_{R2} = R_2 i_{R2} \tag{1.18b}$$
$$U_{R3} = R_3 i_{R3} \tag{1.18c}$$
$$U_{R4} = (K-1) R_3 i_{R4} \tag{1.18d}$$
$$U_{s,R2} = R_2 i_{s,R2} \tag{1.18e}$$

Für die beiden Kondensatoren gilt, dass der Strom proportional zur zeitlichen Ableitung der Spannung $\mathrm{d}U/\,\mathrm{d}t$ ist. Hierbei ist der Proportionalitätsfaktor die Kapazität C welche

der Ableitung der Ladung nach der Spannung $\mathrm{d}Q/\,\mathrm{d}U$ entspricht. Daraus folgen weitere zwei Bauteilgleichungen, welche im Gegensatz zu allen zuvor aufgestellten Gleichungen auch zeitliche Ableitungen beinhalten.

- Bauteilgleichungen für die Kondensatoren:

$$i_{C1} = \frac{\mathrm{d}Q_1}{\mathrm{d}U_{C1}} \frac{\mathrm{d}}{\mathrm{d}t} U_{C1} = C_1(U_{C1}) \frac{\mathrm{d}}{\mathrm{d}t} U_{C1} \tag{1.19a}$$

$$i_{C2} = C_2 \frac{\mathrm{d}}{\mathrm{d}t} U_{C2} \tag{1.19b}$$

Es kann nun das mathematische Modell des elektrischen Netzwerks aus Abbildung 1.4 in der Form eines nichtlinearen Zustandsraummodells

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, u, d) \tag{1.20a}$$

$$y = \mathbf{h}(\mathbf{x}, u, d) \tag{1.20b}$$

mit der Spannung U_e als Eingang u bzw. der Spannung U_s als Störeingang d sowie der Spannung U_a als Ausgang y angeschrieben werden. Die beiden Kondensatorspannungen werden hier als Systemzustände $\mathbf{x} = \begin{bmatrix} U_{C1} & U_{C2} \end{bmatrix}^{\mathrm{T}}$ gewählt.

Ausgehend von den Bauteilgleichungen für die Kondensatoren (1.19) unter Berücksichtigung der Gleichungen für die Knoten (1.16), Maschen (1.17) und der Ohmschen Widerstände (1.18) ergibt sich folgendes nichtlineares Differentialgleichungssystem in der gewünschten Form:

$$\frac{\mathrm{d}}{\mathrm{d}t} \begin{bmatrix} U_{C1} \\ U_{C2} \end{bmatrix} = \begin{bmatrix} -\frac{R_1+R_2}{R_1 R_2 \frac{\mathrm{d}Q_1}{\mathrm{d}U_{C1}}} U_{C1} - \frac{(K-1)R_1+KR_2}{R_1 R_2 \frac{\mathrm{d}Q_1}{\mathrm{d}U_{C1}}} U_{C2} + \frac{U_e}{R_1 \frac{\mathrm{d}Q_1}{\mathrm{d}U_{C1}}} \\ \frac{U_{C1}}{R_2 C_2} + \frac{(K-2)U_{C2}}{R_2 C_2} + \frac{U_s}{R_2 C_2} \end{bmatrix}, \tag{1.21a}$$

$$y = K U_{C2}. \tag{1.21b}$$

Aufgabe 1.7 (Systemanalyse).

1. Linearisieren Sie das System (1.21) für den stationären Eingang $u_R = U_{e,R}$ und den stationären Störeingang $d_R = U_{s,R}$ um eine allgemeine Ruhelage $\mathbf{x}_R = \begin{bmatrix} U_{C1,R} & U_{C2,R} \end{bmatrix}^{\mathrm{T}}$. Stellen Sie das linearisierte System in der Form

$$\Delta\dot{\mathbf{x}} = \mathbf{A}\Delta\mathbf{x} + \mathbf{b}_u \Delta u + \mathbf{b}_d \Delta d \tag{1.22a}$$

$$\Delta y = \mathbf{c}^{\mathrm{T}} \Delta\mathbf{x} + d_u \Delta u + d_d \Delta d \tag{1.22b}$$

   mit $\Delta\mathbf{x} = \mathbf{x} - \mathbf{x}_R$, $\Delta u = u - u_R$, $\Delta d = d - d_R$ und $\Delta y = y - y_R$ dar.

2. Berechnen Sie die Ruhelagen $\mathbf{x}_R = \begin{bmatrix} U_{C1,R} & U_{C2,R} \end{bmatrix}^{\mathrm{T}}$ des Systems (1.20) für den stationären Eingang $u_R = U_{e,R}$ und den stationären Störeingang $d_R = U_{s,R}$.

Laden Sie das zip-Archiv `uebung1.zip` von der Homepage der Lehrveranstaltung herunter. Das M-file `Esys_Parameter.m` stellt ein Grundgerüst für die Lösung der nachfolgenden Aufgaben dar.

3. Bestimmen Sie für das linearisierte System (1.22) in MATLAB die Übertragungsfunktionen $G(s)$ vom Eingang Δu zum Ausgang Δy und $G_d(s)$ von der Störung Δd zum Ausgang Δy. Spezifizieren Sie dazu das nichtlineare Verhalten von $C_1(U_{C1})$ mittels

$$Q_1(U_{C1}) = U_{C1,ref}(C_{1,ref} - C_{1,\infty}) \arctan\left(\frac{U_{C1}}{U_{C1,ref}}\right) + C_{1,\infty} U_{C1} \tag{1.23}$$

   sowie die Ruhelage durch $U_{e,R} = 5\,\mathrm{V}$ und $U_{s,R} = 5\,\mathrm{V}$. Wählen Sie die Parameter $C_{1,ref} = 9\,\mu\mathrm{F}$, $U_{C1,ref} = 3.5\,\mathrm{V}$, $C_{1,\infty} = 1\,\mathrm{nF}$, $C_2 = 1\,\mu\mathrm{F}$, $R_1 = 600\,\Omega$, $R_2 = 3500\,\Omega$, $K = 3$.

4. Bei der Übertragungsfunktion $G(s)$ aus Punkt 3 handelt es sich um ein Verzögerungsglied 2-ter Ordnung (PT_2) in der Form $G(s) = \dfrac{V}{1 + 2\xi(sT) + (sT)^2}$. Bestimmen Sie den Verstärkungsfaktor V, den Dämpfungsgrad ξ und die Zeitkonstante T. Führen Sie diese Berechnungen händisch aus.

5. Stellen Sie mittels MATLAB die Bode-Diagramme der beiden Übertragungsfunktionen dar und interpretieren Sie diese.

Hinweis: Zur Überprüfung Ihrer Ergebnisse sind an dieser Stelle die numerische Dynamikmatrix und die zugehörigen Eingangsvektoren des Systems (1.22) angegeben

$$\mathbf{A} = \begin{bmatrix} -1985.997 & -5667.357 \\ 285.714 & 285.714 \end{bmatrix}, \quad \mathbf{b}_u = \begin{bmatrix} 1695.363 \\ 0 \end{bmatrix}, \quad \mathbf{b}_d = \begin{bmatrix} 0 \\ 285.714 \end{bmatrix}.$$

Eine positive Beurteilung setzt voraus, dass Ihre Werte mit den oben angeführten Werten übereinstimmen!

Aufgabe 1.8 (Implementierung in MATLAB/SIMULINK). Das nichtlineare Modell (1.20) soll nun als MATLAB Function und das linearisierte Modell (1.22) als `State-Space`-Block implementiert werden.

Verwenden Sie hierzu die Simulationsdatei `Esys.slx` sowie das bereits in Aufgabe 1.7 bearbeitete M-file `Esys_Parameter.m` aus dem zip-Archiv `uebung1.zip`.

Verwenden Sie weiterhin die Parameter aus Punkt 3 von Aufgabe 1.7.

1. Öffnen Sie die Simulationsdatei `Esys.slx` und vervollständigen Sie die MATLAB Function. Unvollständige Einträge sind durch den Kommentar `TODO` gekennzeichnet. Führen Sie anschließend die Simulation `Esys.slx` aus und überzeugen Sie sich von der Lauffähigkeit und Plausibilität der fertiggestellten MATLAB Function.

2. Das Simulationsmodell `Esys.slx` enthält bereits eine Grundstruktur für die Implementierung des linearisierten Modells (1.22) als `State-Space`-Block. Vervollständigen Sie alle durch `TODO` gekennzeichneten Blöcke. Das Ziel ist es, die Zustände $\mathbf{x}$ und den Ausgang y des nichtlinearen Systems mit den aus dem linearisierten Modell abgeleiteten Größen $\mathbf{x}_R + \Delta\mathbf{x}$ und $y_R + \Delta y$ in den vorbereiteten `Scopes` zu vergleichen.

3. Im Folgenden soll das Verhalten des Systems untersucht werden. Im Simulationsmodell `Esys.slx` stehen Ihnen eine sprungförmige und eine sinusförmige Eingangsgröße $\Delta U_e(t)$ ($U_e(t) = \Delta U_e(t) + U_{e,R}$) zur Verfügung. Variieren Sie Winkelfrequenz und Amplitude der sinusförmigen Eingangsgröße z. B. gemäß der Folgen $[10^2\,\mathrm{rad/s},\ 10^3\,\mathrm{rad/s},\ 10^4\,\mathrm{rad/s}]$ und $[0.5\,\mathrm{V},\ 1\,\mathrm{V},\ 3\,\mathrm{V}]$. Welches Systemverhalten können Sie hierbei erkennen? Wie verhält sich das System bei Aufschaltung einer sprungförmigen Störung? Wie wirkt sich die nichtlineare Kapazität auf das dynamische Systemverhalten aus? In welcher Weise beeinflusst sie das stationäre Systemverhalten? Untersuchen Sie die Unterschiede zwischen dem nichtlinearen und dem linearisierten System. Wie erklären sich diese Unterschiede? Dokumentieren und diskutieren Sie ihre Simulationsergebnisse.

1.4 Literatur

[1.1] A. Kugi, *Skriptum zur VO Automatisierung (WS 2025/2026)*, Institut für Automatisierungs- und Regelungstechnik, TU Wien, 2025. Adresse: https://www.acin.tuwien.ac.at/bachelor/automatisierung/.

[1.2] A. Angermann, M. Beuschel, M. Rau und U. Wohlfarth, *Matlab - Simulink - Stateflow, Grundlagen, Toolboxen, Beispiele*. München: Oldenbourg Verlag, 2005.

[1.3] H. Schwarz, *Numerische Mathematik*. Stuttgart: B.G. Teubner, 1997.