TECHNISCHE UNIVERSITÄT ILMENAU
Fakultät für Informatik und Automatisierung
Institut für Automatisierungs- und Systemtechnik
Fachgebiet Prozessoptimierung

MATLAB® für Ingenieure

Dr.-Ing. Siegbert Hopfgarten

Tel., E-Mail: +49 3677 69-1418, siegbert.hopfgarten@tu-ilmenau.de
Fachgebietsinternetseite: `http://www.tu-ilmenau.de/prozessoptimierung`
Vorlesungsskript: Lehre/Vorlesungen, Seminare, Praktika/MATLAB für Ingenieure

MATLAB®/Simulink® sind eingetragene Warenzeichen und weitere Produktnamen Handelsmarken der The Mathworks Inc.

Inhalt

1. MATLAB®/Simulink®
2. Kontinuierliche Regelungssysteme
3. Optimierung

1 MATLAB®/Simulink® - Einführung

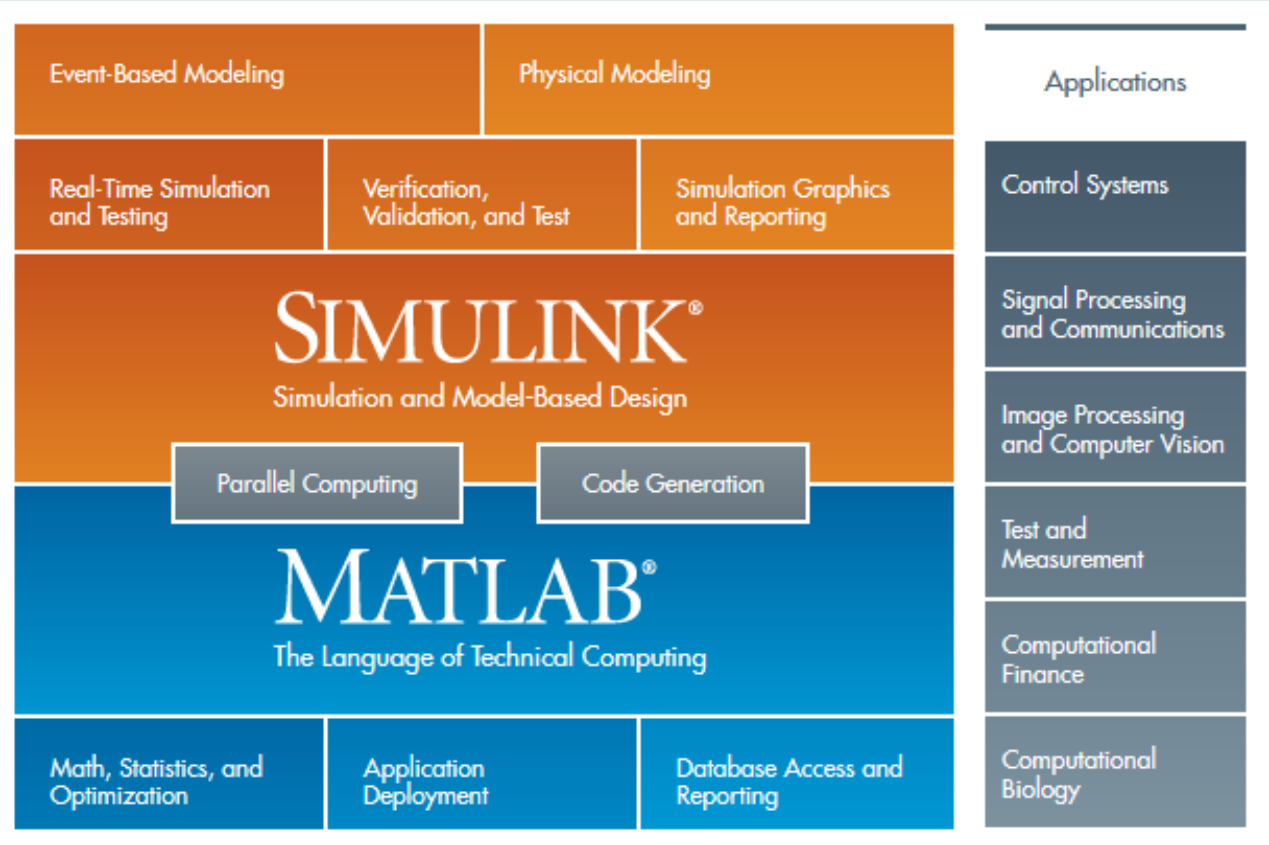
1.1 Einleitung

- MATLAB®: Hochsprache für wissenschaftlich-technische Berechnungen (Berechnungen, Visualisierung, Programmierung) in leicht handhabbarer Umgebung
  - Mathematisch-numerische Berechnungen (auch symbolische)
  - Modellierung, Simulation, Prototyping, Algorithmen, ...
  - Datenanalyse, -auswertung und -visualisierung
  - Wissenschaftlich-technische grafische Darstellungen
  - Anwenderprogrammentwicklung (einschl. grafisches Benutzerinterface)
  - Physikalische objekt-orientierte Modellierung
- Simulink®: Block- (tw. objekt-)orientiertes grafisches Simulationssystem
- Fachgebietsspezifische Toolboxen
- Abkürzung MATLAB®: matrix laboratory
  - Grundlegendes Datenelement: Feld (array, matrix)
  - Interaktiv (Kommando–Interpreter)
  - Ursprüngliche Entwicklung: leichter Zugang zu LINPACK und EISPACK (zwei Bibliotheken, „state-of-the-art“ der Matrizenrechnung) und grafische Veranschaulichung

1 MATLAB®/Simulink® - Einführung

Kleiner Überblick

(Quelle: `https://de.mathworks.com`)
(erhebt keinen Anspruch auf Vollständigkeit)



1 MATLAB®/Simulink® - Einführung

1.1 Einleitung

- Entwicklung von MATLAB® im Verlaufe der Zeit: Toolboxen, Objektorientierung (Klassendefinitionen, ...)
- Verfügbarkeit: PCs und Workstations, verschiedene Betriebssysteme (WINDOWS, Unix, Linux, ...), TU Ilmenau: im Uni-Rechnernetz (Thüringer Landeslizenz der Unis; zeitparallele Zugriffe)
- Kommerzielles Produkt; Studentenversion (69,00 €) (`https://de.mathworks.com/store/link/products/student/SV?s_tid=ac_buy_sv_but1`) mit (kleinen) Einschränkungen (MATLAB®, Simulink®, 10 Toolboxen; weitere Toolboxen à 7 €; auch: Prototyping, Test auf Arduino, LEGO MINDSTORMS NXT, Raspberry Pi)
- Informationen
  - `https://de.mathworks.com`
  - Literaturliste (Bibliothek), Copy-Shop, Internet
- Alternativen zu MATLAB®:
  - Scilab/Xcos unter Linux, Windows, usw. (Freie, quelloffene Software, GPL-kompatibel), `https://www.scilab.org`
  - GNU Octave unter Linux, Windows, usw. (Freie, quelloffene Software), `https://www.gnu.org/software/octave`

1 MATLAB®/Simulink® - Einführung

1.2 Arbeitsumgebung

- Kommando-, Workspace-, Verzeichnis-, Historiefenster
- Reiter Home, Plots, Apps
- Kommandomanipulation; -folgen in ASCII-Skript-Dateien; (`*.m`)

Hilfe, Informationen, Übersichten, usw.

Befehl	Erläuterung
`demo`	Beispiele
`doc` *schlüsselwort*	Hypertext-Dokumentation (auch PDF-Format)
`help`, `help help`, `help` *fcn* | *m-file* | *thema*	Textuelle Online-Hilfe
`exist`	Test, ob Variable oder Funktione definiert
`type`	Auflisten vom M-Files
`edit`	Editieren von M-Files

1 MATLAB®/Simulink® - Einführung

1.2 Arbeitsumgebung

Wichtige „Betriebssystemkommandos“

`cd`	Verzeichniswechsel
`pwd`	Ausgabe des gegenwärtigen Arbeitsverzeichnisses
`ls`, `dir`	Inhaltsverzeichnis
`delete`	Löschen von Dateien
`!`*befehl*	Betriebssystemskommandos ((WIN)DO(W)S, LINUX, ...)

Sonstiges

`format`	Zahlenformat (short, long, e, g, bank, ...)
`;` / `...`	Ausgabeunterdrückung/Kommandofortsetzung
Workspace: `who`, `whos`	Info zu Variablen (Kurz-, Langform); s. auch Workspace
`clear`	Löschen von Variablen oder Funktionen
`save`, `load`	Speichern bzw. Laden von Variablen in bzw.
	von Datei mit angebbarem Format (-mat, -ascii, u.a.)
`diary`	Tagebuchfunktion, d. h. Speichern aller Kommandos und Ergebnisausgaben des Kommandofensters in einer Datei

1 MATLAB®/Simulink® - Einführung

1.3 MATLAB®-Kommandosprache

Variable und Datentypen

- Variable: erste 31 Zeichen signifikant; Buchstaben, Ziffern, _ ; beginnend mit Buchstaben; keine explizite Vereinbarung; Speicherung im Workspace (Arbeitsspeicher)

Spezielle Variable und Variablenwerte

Variable	Erläuterung
ans	Antwortvariable, wenn das Ergebnis keiner Variablen zugewiesen wurde
eps	Relative Genauigkeit
realmax, realmin	Größte bzw. kleinste darstellbare Gleitkommazahl
pi	$\pi = 3.1415926\ldots$
i,j	Imaginäre Einheit
inf	Unendlich (∞)
NaN	„not a number“ , Ergebnis eines unbestimmten Ausdruckes, z. B. $\frac{0}{0}$

1 MATLAB®/Simulink® - Einführung

1.3 MATLAB®-Kommandosprache

Datentypen (Klassen) – eine Auswahl

Datentyp	Erläuterung
`double`	Vektoren, Matrizen, 3D-Felder; Elemente reelle oder komplexe Zahlen; voll besetzt Indizierung bei 3-D-F.: (Zeile, Spalte, Seite)
`sparse`	Schwach besetzte Matrizen (sparse matrices)
`char`	Zeichenketten(-felder)
`cell`	Mehrdimensionale „Zellenfelder“; Zellen: Elemente unterschiedlicher Datentypen, auch Strukturen
`struct`	Elemente unterschiedlicher Datentypen
	einer Variablen zusammengefaßt
`int8, ..., int64,` `uint16, ..., uint64, ...`	Vorzeichenbehaftete und vorzeichenlose ganze Zahlenfelder
`logical`	Boolesche Felder
...	...

1 MATLAB®/Simulink® - Einführung

1.3 MATLAB®-Kommandosprache

Zahlen, Zeichen(ketten)

- Vorzeichenbehaftete reelle oder komplexe Zahlen in Festkomma- oder in wissenschaftlicher Darstellung (Mantisse, Exponent):

  Bsp.: `3 9.6 -.99 1e3 -5.34e-4 -3.14j 1i 3-4i`
- Zeichen(ketten) in Hochkomma eingeschlossen, Bsp. `'xyz'` , `'er hat''s'`

Operatoren (`doc ops`)

- Arithmetische, Vergleichs–, logische, bitweise, Mengen- und spezielle Operatoren

Funktionen (`doc elfun`)

- Mathematische Funktionen (Winkelfunktionen, Exponentialfunktion, Logarithmen, Wurzel, Rundung, Teiler, Rest, Komplexe Zahlen, ...) (`doc elfun`) u.v.a., z.B. Umwandlung von Datentypen (Zeichenketten, numerisch), Zeichenkettenmanipulation

Ausdrücke

- Ausdrücke werden aus Variablen, Zahlen, Zeichenketten, Operatoren und Funktionen gebildet.

1 MATLAB®/Simulink® - Einführung

1.3 MATLAB®-Kommandosprache

Ein-/Ausgabe

`save`	Speicherung aller Workspace-Variablen in einer binären oder ASCII-Datei
Bsp.:	
`save` *fname*	Speicherung der Datei *fname*`.mat`
`save`	`matlab.mat`
`save` *fname* `X,Y,Z`	*fname*`.mat`; Inhalt: X, Y und Z
`save` *fname* `-ascii`	*fname*`.mat` im ASCII-Format
`load`	Laden von Variablen aus einer binären oder ASCII-Datei in den Workspace
Bsp.:	
`load` *fname*	Laden aus dem binären MAT-File *fname*`.mat`
`load`	dto. `matlab.mat`
`load test.dat`	Laden aus der ASCII-Datei `test.dat` bzw.
(oder `load test -ascii`)	`test`; Abspeichern in Variable/Matrix `test`

1 MATLAB®/Simulink® - Einführung

1.3 MATLAB®-Kommandosprache

Ein-/Ausgabe (Fortsetzung)

`input`	Tastatureingabe durch Nutzer Bsp.: `n=input('Wieviele Messwerte ?');` Eingabe: MATLAB®-Ausdruck Bsp.: `Name=input('Wie ist Ihr Name? : ','s');` Eingabe: Zeichenkette; wird nicht ausgewertet
`fopen`, `fclose`, `fread`, `fwrite`, `fscanf`, `fprintf` ...	„low–level"-E/A-Kommandos
`print`	Ausgabe von Grafiken auf Drucker, Plotter oder in Datei; Formate (PS, EPS, JPG, PNG, TIF, HPGL, druckerspezifisch); Zusammenwirken mit GhostScript
`plot(x,y)`	Grafik mit linearer Achseneinteilung, $y = f(x)$

- Weitere Kommandos zur grafischen Darstellung: siehe Visualisierung
- Weitere Sprachelemente: siehe M-File-Programmierung

1 MATLAB®/Simulink® - Einführung

1.4 Matrizen, lineare Algebra

Matrizen, Teilmatrizen (`doc elmat`)

- Matrix $\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 6 \end{bmatrix}$

  ```
  A=[1 2 3;4 5 6;7 8 6]
  ```

- Teilmatrix $\boldsymbol{B} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$

  ```
  B=A(1:2,1:2)
  ```

Generierung von Matrizen

Funktion	Erläuterung
`zeros, ones, eye`	Null-, Eins-, Einheitsmatrizen
`rand, randn`	zufällig belegte Matrizen (Gleich-, Normalverteilung)
`linspace, logspace`	Vektor mit äquidistanter/logarithmischer Teilung
`diag`	Diagonalmatrizen/Diagonalen

1 MATLAB®/Simulink® - Einführung

1.4 Matrizen, lineare Algebra

Matrizenoperationen

Operator	Erläuterung
'	Transposition
+, -	Addition, Subtraktion
*	Matrizenmultiplikation
.*	Elementweise Multiplikation
\, /, ./	Links-, Rechts- und elementweise Division
^, .^	Potenzierung, elementweise Potenzierung

Lineare Gleichungssysteme ($m = n$)

- 1 Bsp.: m, n - Anzahl d. Gleichungen bzw. Anzahl d. Variablen

$$
\begin{aligned}
3\,x_1 + 2\,x_2 - 3\,x_3 &= 2 \\
-5\,x_1 - 4\,x_2 + 6\,x_3 &= -1 \\
2\,x_1 \qquad\quad - x_3 &= 1
\end{aligned}
$$

- Matrix-/Vektor-Schreibweise:

$$
\begin{bmatrix} 3 & 2 & -3 \\ -5 & -4 & 6 \\ 2 & 0 & -1 \end{bmatrix}
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}
= \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}
$$

$$
\begin{aligned}
\boldsymbol{A}\,\boldsymbol{x} &= \boldsymbol{b} \\
\boldsymbol{x} &= \boldsymbol{A}^{-1}\,\boldsymbol{b}
\end{aligned}
$$

- Lösung: `x=inv(A)*b` oder `x=A\b`

Lineare Gleichungssysteme ($m = n$)

- 1 Bsp.: m, n - Anzahl d. Gleichungen bzw. Anzahl d. Variablen

$$\begin{aligned} 3\,x_1 + 2\,x_2 - 3\,x_3 &= 2 \\ -5\,x_1 - 4\,x_2 + 6\,x_3 &= -1 \\ 2\,x_1 \qquad\quad - x_3 &= 1 \end{aligned}$$

- Matrix-/Vektor-Schreibweise:

$$\begin{bmatrix} 3 & 2 & -3 \\ -5 & -4 & 6 \\ 2 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}$$

$$\begin{aligned} \boldsymbol{A}\,\boldsymbol{x} &= \boldsymbol{b} \\ \boldsymbol{x} &= \boldsymbol{A}^{-1}\,\boldsymbol{b} \end{aligned}$$

- Lösung: `x=inv(A)*b` oder `x=A\b`

Lineare Gleichungssysteme ($m = n$)

- 1 Bsp.: m, n - Anzahl d. Gleichungen bzw. Anzahl d. Variablen

$$\begin{aligned} 3\,x_1 + 2\,x_2 - 3\,x_3 &= 2 \\ -5\,x_1 - 4\,x_2 + 6\,x_3 &= -1 \\ 2\,x_1 \qquad\quad - x_3 &= 1 \end{aligned}$$

- Matrix-/Vektor-Schreibweise:

$$\begin{bmatrix} 3 & 2 & -3 \\ -5 & -4 & 6 \\ 2 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}$$

$$\begin{aligned} \boldsymbol{A}\,\boldsymbol{x} &= \boldsymbol{b} \\ \boldsymbol{x} &= \boldsymbol{A}^{-1}\,\boldsymbol{b} \end{aligned}$$

- Lösung: `x=inv(A)*b` oder `x=A\b`

Lineare Gleichungssysteme (überbestimmt; $m > n$)

- 2. Bsp.:

$$
\begin{aligned}
3\,x_1 + 2\,x_2 &= 3 \\
x_1 - 2\,x_2 &= 1 \\
x_1 - x_2 &= 1 \\
4\,x_1 + x_2 &= 4
\end{aligned}
$$

- Matrix-/Vektor-Schreibweise:

$$
\begin{bmatrix} 3 & 2 \\ 1 & -2 \\ 1 & -1 \\ 4 & 1 \end{bmatrix}
\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}
=
\begin{bmatrix} 3 \\ 1 \\ 1 \\ 4 \end{bmatrix}
$$

$$
\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}
$$

- Lösung (i. S. kleinste Fehlerquadrate): hier nur möglich: `x=A\b`

Lineare Gleichungssysteme (überbestimmt; $m > n$)

- 2. Bsp.:

$$
\begin{aligned}
3\,x_1 + 2\,x_2 &= 3 \\
x_1 - 2\,x_2 &= 1 \\
x_1 - \quad x_2 &= 1 \\
4\,x_1 + \quad x_2 &= 4
\end{aligned}
$$

- Matrix-/Vektor-Schreibweise:

$$
\begin{bmatrix} 3 & 2 \\ 1 & -2 \\ 1 & -1 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ 1 \\ 4 \end{bmatrix}
$$

$$
\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}
$$

- Lösung (i. S. kleinste Fehlerquadrate): hier nur möglich: `x=A\b`

Lineare Gleichungssysteme (überbestimmt; $m > n$)

- 2. Bsp.:

$$
\begin{aligned}
3\,x_1 + 2\,x_2 &= 3 \\
x_1 - 2\,x_2 &= 1 \\
x_1 - x_2 &= 1 \\
4\,x_1 + x_2 &= 4
\end{aligned}
$$

- Matrix-/Vektor-Schreibweise:

$$
\begin{bmatrix} 3 & 2 \\ 1 & -2 \\ 1 & -1 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ 1 \\ 4 \end{bmatrix}
$$

$$
\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}
$$

- Lösung (i. S. kleinste Fehlerquadrate): hier nur möglich: `x=A\b`

Lineare Gleichungssysteme (unterbestimmt; $m < n$)

- 3. Bsp.: keine eindeutige Lösung

$$\begin{aligned} 6\,x_1 + 8\,x_2 + 7\,x_3 + 3\,x_4 &= 7 \\ 3\,x_1 + 5\,x_2 + 4\,x_3 + \,x_4 &= 8 \end{aligned}$$

- Lösung: siehe MATLAB®-Online-Hilfe (Linear Equations/Systems of Linear Equations/Underdetermined Systems (Stichworte: Basislösungen; höchstens m Nicht-Null-Komponenten; Linearkombination; Nullraum-Vektor))

Lineare Gleichungssysteme (unterbestimmt; $m < n$)

- 3. Bsp.: keine eindeutige Lösung

$$\begin{aligned} 6\,x_1 + 8\,x_2 + 7\,x_3 + 3\,x_4 &= 7 \\ 3\,x_1 + 5\,x_2 + 4\,x_3 + \,x_4 &= 8 \end{aligned}$$

- Lösung: siehe MATLAB®-Online-Hilfe (Linear Equations/Systems of Linear Equations/Underdetermined Systems (Stichworte: Basislösungen; höchstens m Nicht-Null-Komponenten; Linearkombination; Nullraum-Vektor))

1 MATLAB®/Simulink® - Einführung

1.4 Matrizen, lineare Algebra

Matrizenfunktionen (`doc matfun`)

Befehl	Erläuterung
`norm, cond`	Matrix- oder Vektornorm, Konditionszahl
`rank, inv`	Rang, Inverse einer Matrix
`det`	Determinante einer Matrix
`trace`	Spur einer Matrix (Summe der Diagonalelemente)
`chol, lu`	Cholesky-, LU-Faktorisierung
`eig`	Eigenwerte, -vektoren
`poly`	Koeffizienten des charakteristischen Polynoms
...	

1 MATLAB®/Simulink® - Einführung

1.5 Gewöhnliche Differentialgleichungen - Anfangswertprobleme -

Funktionen für Dgln. (`doc funfun`)

Befehl/Datei	Erläuterung
`odeset, odeget`	Einstellen/Lesen von Parametern zur Dgl.lösung
	Num. Integrationsverf. zur Lösung von Dgl.systemen
`ode45, ode23, ode113`	... nicht-steife
`ode15s, ode23s, ...`	
`ode23t, ode23tb`	... steife
`ode15s, (ode23t)`	... DAEs (mit konst. Masse-Matrix)
`ode15i`	... implizite
`odeplot`	Zeitverlaufgrafik während der Simulation
`odephas2, odephas3`	2D-, 3D-Phasendiagramm während der Simulation
`odeprint`	Tabellarische Ergebnisse im Kommandofenster

1 MATLAB®/Simulink® - Einführung

1.5 Gewöhnliche Differentialgleichungen

Beispiel 1: Logistisches Wachstum (Wachstums-Dgl.)

$$\dot{y}(t) = a\,y(t)(1-y(t))\,, \quad y(0) = y_0 = 0.1$$

mit Wachstumsparameter $a = 3.2$

MATLAB®-Funktion (Datei: `growth.m`)

```
function ypunkt=growth(t,y)
a=3.2; % Wachstumsparameter
ypunkt=a*y*(1-y);
```

Aufruf (im Kommandofenster, im Skript, in einer Funktion)

```
[t,y]=ode45(@growth,[0 5],0.1);plot(t,y) % oder
sol=ode45(@growth,[0 5],0.1);plot(sol.x,sol.y) % oder
```

```
sol=ode45(@growth,[0 5],0.1);x=linspace(0,5,101); % Zeitbasis neu
y=deval(sol,x,1); % Interpolation fuer neue Zeitbasis
plot(x,y)
```

1 MATLAB®/Simulink® - Einführung

1.5 Gewöhnliche Differentialgleichungen

Beispiel 2: Van-der-Pol-Schwinger (Dgl.-system)

$$\dot{\boldsymbol{y}}(t) = \begin{bmatrix} \dot{y}_1(t) \\ \dot{y}_2(t) \end{bmatrix} = \begin{bmatrix} y_2(t) \\ \mu\left(1 - y_1^2(t)\right) y_2(t) - y_1(t) \end{bmatrix}, \quad \boldsymbol{y}(0) = \boldsymbol{y}_0 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

$$\mu = \{1{,}1000\}$$

MATLAB®-Funktion (Datei: `shvdpmue.m`)

```
function dydt=shvdpmue(t,y,mue) % mit weiterer
Parameteruebergabe
dydt=[y(2);mue*(1-y(1)^2)*y(2)-y(1)];
```

Aufruf (im Kommandofenster, im Skript, in einer Funktion)

```
options=odeset('OutputFcn','odeplot'); mue=1;
[t,y]=ode45(@shvdpmue,[0 50],[2;0],options,mue); % oder
[t,y]=ode15s(@shvdpmue,[0 50],[2;0],options,mue);
```

- Eigenschaft für $\mu = \{1{,}1000\}$?
- $\mu = 1$: nicht-steif, $\mu = 1000$: steif

1 MATLAB®/Simulink® - Einführung

1.5 Gewöhnliche Differentialgleichungen

Beispiel 2: Van-der-Pol-Schwinger (Dgl.-system)

$$\dot{\boldsymbol{y}}(t) = \begin{bmatrix} \dot{y}_1(t) \\ \dot{y}_2(t) \end{bmatrix} = \begin{bmatrix} y_2(t) \\ \mu\left(1 - y_1^2(t)\right) y_2(t) - y_1(t) \end{bmatrix}, \quad \boldsymbol{y}(0) = \boldsymbol{y}_0 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

$$\mu = \{1{,}1000\}$$

MATLAB®-Funktion (Datei: `shvdpmue.m`)

```
function dydt=shvdpmue(t,y,mue) % mit weiterer
Parameteruebergabe
dydt=[y(2);mue*(1-y(1)^2)*y(2)-y(1)];
```

Aufruf (im Kommandofenster, im Skript, in einer Funktion)

```
options=odeset('OutputFcn','odeplot'); mue=1;
[t,y]=ode45(@shvdpmue,[0 50],[2;0],options,mue); % oder
[t,y]=ode15s(@shvdpmue,[0 50],[2;0],options,mue);
```

- Eigenschaft für $\mu = \{1{,}1000\}$?
- $\mu = 1$: nicht-steif, $\mu = 1000$: steif

1 MATLAB®/Simulink® - Einführung

1.5 Gewöhnliche Differentialgleichungen

Beispiel 2: Van-der-Pol-Schwinger (Steifes Dgl.-system)

$$\dot{\boldsymbol{y}}(t) = \begin{bmatrix} \dot{y}_1(t) \\ \dot{y}_2(t) \end{bmatrix} = \begin{bmatrix} y_2(t) \\ \mu\left(1 - y_1^2(t)\right) y_2(t) - y_1(t) \end{bmatrix}, \quad \boldsymbol{y}(0) = \boldsymbol{y}_0 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

$$\mu = 1000$$

Jacobi-Matrix, MATLAB®-Funktion (Datei: `shvdpjacob1000.m`)

Jacobi-Matrix: $$\boldsymbol{J} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{y}} = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -2\mu y_1 y_2 - 1 & \mu(1 - y_1^2) \end{bmatrix}$$

```
function J = shvdpjacob1000(t,y,mue)
J = [0 1; -2*mue*y(1)*y(2)-1 mue*(1-y(1)^2)];
```

Aufruf (im Kommandofenster, im Skript, in einer Funktion)

```
options=odeset('Jacobian',@shvdpjacob1000);mue=1000;
[t,y]=ode15s(@shvdpmue,[0 50],[2;0],options,mue);
```

Beispiel 3: DAE-Systeme (Datei: dae1.m)

a) Implizites ODE- oder DAE-System (höchstens Index 1) liegt bereits vor

- 2 Schritte:

1. Konsistente Anfangsbedingungen (AB) zur Integration mit ode15i

```
[y0mod,yp0mod] =
decic(odefun,t0,y0,fixed_y0,yp0,fixed_yp0,options)
y0 = [1; 0; 1e-3]; % Nicht konsistente AB
yp0 = [0; 0; 0];   % Nicht konsistente AB
tspan = [0 4*logspace(-6,6)];
M = [1 0 0;0 1 0;0 0 0];   % Masse-Matrix
options = odeset('RelTol',1e-4, ...
'AbsTol',[1e-6 1e-10 1e-6],'Jacobian',{[],M});
[y0,yp0] = decic(@f,0,y0,[1 1 0],yp0,[],options)
```

Beispiel 3: DAE-Systeme (Fortsetzung)

zu a) Implizites ODE- oder DAE-System (höchstens Index 1) liegt bereits vor

2. Lösen mit `ode15i`

```
[t,y] = ode15i(@f,tspan,y0,yp0,options)
```

MATLAB®-Funktion f (evtl. Datei: `f.m`)

```
function res = f(t,y,yp)
res = [ yp(1) + 0.04*y(1) - 1e4*y(2)*y(3);
yp(2) - 0.04*y(1) + 1e4*y(2)*y(3) + 3e7*y(2)*y(2);
y(1) + y(2) + y(3) - 1];
```

- Bsp.: Chemische Reaktion (s. o., `ihb1dae.m`)
- Weiteres Bsp.: Diskretisierte partielle Dgl. mit beweglichem Gitter (Burgers-Gleichung; Mechanik, Nichtlineare Akustik, Fluid-, Gasdynamik, Verkehrsfluss), `iburgersode.m`)

Beispiel 3: DAE-Systeme (Fortsetzung)

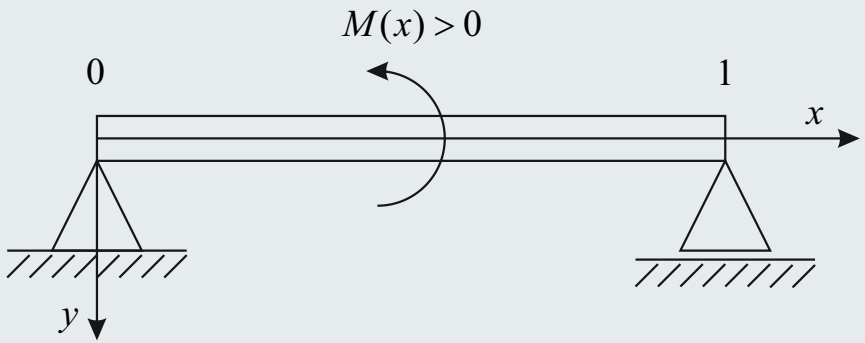
b) DAE-System (Differentiationsindex > 1) liegt vor

- 6 Schritte (symbolisch (Symbolic Math Toolbox™), numerisch):
  1. Definition der symbolischen Variablen und des DAE-Systems
  2. Falls Dgl.-system höherer Ordnung: $\rightarrow$ Dgl.-system 1 Ordnung
  3. Falls Differentiationsindex > 1: Reduzierung
  4. Symbolisches System $\rightarrow$ numerisches „MATLAB® function handle“
  5. Konsistente Anfangsbedingungen (AB)
  6. Lösen mit `ode15i`, `ode15s`, `ode23t`
- Bsp.: Planares Pendel (siehe MATLAB®-Online-Dokumentation))

1 MATLAB®/Simulink® - Einführung

1.5 Gewöhnliche Differentialgleichungen - Mehrpunkt-Randwertproblem -

Beispiel 4: Dgl. mit Anfangs- und Endbedingung (s. [1]): Mehrpunkt-Randwertproblem (Kleine Durchbiegungen am Biegebalken; $1 + \left(\frac{dy}{dx}\right)^2 \approx 1$; Datei: biegebalken.m)



Dgl.: $$-\frac{d^2y}{dx^2} = \frac{M(x)}{EI(x)} = b(x) = \frac{5-4x}{100e^{-x}}$$

mit $l = 1$, $y(0) = 0$, $y(1) = 0$, Biegemomentverlauf $M(x)$, Biegesteifigkeit $EI(x)$

Beispiel 4: (Fortsetzung)

- Umwandlung in Dgl.-system 1 Ordnung: $y_1 = y$, $y_2 = \frac{dy}{dx}$
- MATLAB®-Routinen (Dgl., Randbedingungen, Lösung):

```
function dydx = ystrich_b(x,y)
dydx = [ y(2); (4*x-5)/(100*exp(-x))];
—
function res = bc_b(ya,yb)
res=[ya(1);yb(1)]; % d. h. y(0) - 0 und y(1) - 0
—
xmesh=linspace(0,1,100); % Schaetzung fuer Startloesung
yinit=[0;0]; % Anfangswerte
loesinit=bvpinit(xmesh,yinit); % Anfangsloesung
loesung=bvp4c(@ystrich_b,@bc_b,loesinit); % Loesung
plot(loesung.x,loesung.y(1,:)) % Druck der Biegelinie
```

Beispiel 5: Dgl.-system mit Totzeiten (Datei: Tt_ode.m)

$$\begin{aligned} \dot{x}_1 &= -0.4\, x_1(t-1) \\ \dot{x}_2 &= x_1(t-1) - 2\, x_2(t-0.5) \end{aligned}$$

- MATLAB®-Routinen (Dgl., Historie, Lösung):

```
function xp = dgl_Ttbsp(t,x,z)
xT1=z(:,1);xT2=z(:,2);xp=[-0.4*xT1(1); xT1(1)-2*xT2(2)];
—
function s = history_Ttbsp(t)
s=[1;-2];
—
tspan=[0 6]; Tt=[1 0.5]; options=ddeset('InitialY',[0;0]);
loesung=dde23(@dgl_Ttbsp,Tt,@history_Ttbsp,tspan,options);
t_interp=linspace(0,6,80); x=deval(loesung,t_interp);
plot(t_interp,x(1,:),t_interp,x(2,:)) % Druck
```

Im Gegensatz: numerische Integration zur Ermittlung des bestimmten Integrals

- Flächeninhalt (Bsp.)

$$I = \int_0^5 |x| dx$$

Befehl	Erläuterung
`quad, quad8`	Numerische Integration (bestimmte Integration)

- MATLAB®-Befehl:

```
F=quad('abs',0,5);
```

- Ergebnis:

```
F=12.5
```

1.6 Partielle Differentialgleichungen

Partial Differential Equation Toolbox™

- Funktionen zur Lösung von PDEs (2D-, 3D-, Zeit) mittels Finite-Differenzen-Methode
- Statische, Zeit-, Frequenzbereichs-, Eigenwertprobleme
- Nachbearbeitung, Visualisierung
- Diffusion, Wärmeübertragung, Strukturmechanik, Elektrostatik, Magnetostatik, AC-Leistungsmagnetismus; nutzerspezifische, gekoppelte Systeme
- Siehe Online-Nutzerdokumentation

1 MATLAB®/Simulink® - Einführung

1.7 Polynome, Interpolation

- Darstellung von Polynomen: Koeffizienten des Polynoms in absteigender Reihenfolge der Potenzen in vektorieller Form,
  z. B. steht `p=[1 0 -2 -5]` für $p(x) = x^3 - 2x - 5$

Polynomfunktionen (`doc polyfun`)

Befehl	Erläuterung
`roots(p)`	Wurzeln (Nullstellen) des Polynoms
`poly`	Berechnung der Polynomkoeffizienten aus den Wurzeln (siehe auch `poly` bei Matrizen)
`polyval(p,5)`	Berechnung des Wertes des Polynoms für $x = 5$
`conv, deconv`	Polynommultiplikation, -division

Bsp.: `a=[1 2 3], b=[4 5 6], c=conv(a,b)`
(Ergebnis: c=[4 13 28 27 18])
`[q,r]=deconv(c,a)`
(Ergebnisse: q=[4 5 6], r=[0 0 0 0 0])

1 MATLAB®/Simulink® - Einführung

1.7 Polynome, Interpolation

Polynomfunktionen (Fortsetzung)

Befehl	Erläuterung
`polyder`	Ableitung eines Polynoms
`polyfit`	Anpassung der Polynomkoeffizienten an Daten
`residue`	Residuenberechnung (Partialbruchzerlegung)

Bsp.: $\frac{2\,s+5}{s^4+3\,s^3+2\,s^2} = \frac{A}{s+2} + \frac{B}{s+1} + \frac{C}{s} + \frac{D}{s^2}$

Ergebnis: `A=-0.25, B=3, C=-2.75, D=2.5`

Interpolation

Befehl	Erläuterung
`interp1`	1D-Interpolation (linear, spline, kubisch)
`interp2`	2D–Interpolation
...	
`spline`	Spline-Interpolation (siehe auch Curve Fitting Toolbox)

1.8 Schwach besetzte Matrizen

Funktionen für schwach besetzte Matrizen (`doc sparfun`)

Befehl	Erläuterung
`speye`	Einheitsmatrix
`sprand, sprandn`	Zufällige belegte Matrizen (Gleich-, Normalverteilung)
`spdiags`	Diagonalmatrizen/Diagonalen (jeweils schwach besetzt)
...	
`sparse, full`	Umwandlung in schwach besetzte Matrix bzw. umgekehrt
`spconvert`	Einlesen von Daten und Speichern in schwach besetzter Matrix
...	

1.8 Schwach besetzte Matrizen

Beispiel

$$D = \begin{bmatrix} 8 & 1 & 6.00 \\ 3 & 5 & 7.00 \\ 4 & 9 & 2.00 \\ 9 & 9 & 0 \end{bmatrix} \quad \text{(Variante 1)}$$

bzw. eine ASCII-Datei mydata.dat enthält die o.a. Werte der Matrix D (Variante 2)

MATLAB®-Befehle

`S=spconvert(D)`	(Variante 1)
`load mydata.dat`	(Variante 2)

1 MATLAB®/Simulink® - Einführung

1.8 Schwach besetzte Matrizen

Weitere Funktionen für schwach besetzte Matrizen

Befehl	Erläuterung
`nnz`	Anzahl von Null verschiedener Elemente
`find`	Auffinden von Null verschiedener Elemente
`spfun`	Anwendung von Funktionen auf von Null verschiedene Elemente
`spy`	Visualisierung der Besetzheitsstruktur

Beispiel

```
Z=sprandn(100,100,0.3);
[L,U,P]=lu(Z,1e-3);
spy(Z), spy(L), spy(U), spy(P)
```

- Algorithmen zur Umordnung
- Funktionen zur lineraren Algebra
- Iterative Methoden zur Lösung linearer Gleichungen, usw.

1 MATLAB®/Simulink® - Einführung

1.9 M-File-Programmierung

– Ergänzung zu Abschnitt 1.3; weitere Sprachelemente (`doc lang`) –

Skripte, Funktionen (`doc function`)

Skripte, Funktionen und Unterfunktionen mit variabler Anzahl von Argumenten

```
function c=test(a,b)
if nargin== 1,
c = a.^ 2; % Quadratzahl einer Zahl
elseif nargin==2
c = a + b; % Summe zweier Zahlen
end
```

Lokale und globale Variable

- Variable innerhalb von Funktionen lokal
- Wenn Variable vor ersten Auftreten mit `global` vereinbart, Zugriff in allen Funktionen bzw. im Workspace, in denen Variable auch mit `global` vereinbart wurde, auf die gleiche Variable

Beispiel: Räuber-Beute-Modell

$$\begin{aligned} \dot{y}_1(t) &= y_1 - \alpha\, y_1(t)\, y_2(t) \\ \dot{y}_2(t) &= -y_2(t) + \beta\, y_1(t)\, y_2(t) \end{aligned}$$

```
function [yp]=sh_lotka(t,y)
% LOTKA Lotka-Volterra predator-prey model
global ALPHA BETA
yp = [y(1) - ALPHA*y(1)*y(2); -y(2) + BETA*y(1)*y(2)];
```

Befehle im Workspace:

```
global ALPHA BETA
ALPHA=0.01; BETA=0.02;
[t,y]=ode23(@sh_lotka,[0 50],[1;1]); plot(t,y)
```

Programmablaufsteuerung

- `if`, `else`, `elseif` - Verzweigung

Beispiel: Test, ob nicht negative Zahl gerade oder ungerade

```
if n < 0, % n negativ ? : Fehlermitteilung
disp('n muss nicht-negativ sein')
elseif rem(n,2) == 0,
disp('n gerade')
else
disp('n ungerade')
end
```

Programmablaufsteuerung (Fortsetzung)

- `switch` - Ausführung bestimmter Anweisungen in Abhängigkeit des Wertes einer Variablen oder eines Ausdruckes

Beispiel:

```
switch zahl
case -1
disp('minus Eins')
case 0
disp('Null')
case 1
disp('plus Eins')
case {2,3,4}
disp('plus Zwei, Drei oder Vier')
otherwise
disp('anderer Wert')
end
```

Programmablaufsteuerung (Fortsetzung)

- `while` - Ausführung von Anweisungen solange ein Ausdruck wahr ist

Beispiel:

```
n=1;
while prod(1:n) < 1e100,
n = n + 1;
end
```

Ausstieg aus `while`-Schleife mit break möglich

- `for` - Laufanweisung

Beispiel:

```
A=zeros(6);
for i=1:6; for j=1:2:6;
A(i,j)=j*3+i;
end; end
```

1 MATLAB®/Simulink® - Einführung

1.9 M-File-Programmierung

Programmablaufsteuerung

Befehl	Erläuterung
`break`	Unterbrechung von Schleifen
`pause`	Pause (Benutzereingabe, Zeitverzögerung)
`keyboard`	Eingaben von Tastatur (aus M-File heraus aufgerufen)
`return`	Rücksprung aus Funktionen, Beenden des `keyboard`-Modus
`eval, feval`	Zeichenkettenauswertung und -berechnung

Beispiele:

```
n=3; t='1/(i+j-1)';
for i=1:n; for j=1:n;
a(i,j)=eval(t);
end; end
```

```
for i=1:10,eval(['P',int2str(i),'=i.^2']);end
```

```
fun=['sin';'cos';'log'];
k=input('Waehle Funktions-Nummer: ');
x=input('Gib einen Zahlenwert ein: ');
feval(fun(k,:),x)
```

1 MATLAB®/Simulink® - Einführung

1.9 M-File-Programmierung

Programmablaufsteuerung

Befehl	Erläuterung
`error, warning`	Fehlermitteilung (danach Programmstopp), Warnung (danach Programmfortsetzung)
Bsp.: `if n <= 1, error('n muss groesser als 1 sein!'),end`	
...	
Debugger	Programmtest (Schrittbetrieb, Unterbrechungspunkte, ...)

Hinweise zur Optimierung des MATLAB®-Codes

- Vektorisierung der Schleifen

  Anstelle von

  `i=0; for t=0:0.01:10; i=i+1; y(i)=sin(t); end`

  ist die vektorisierte Version besser:

  `t=0:0.01:10;y=sin(t);`

- Vorherige Speicherplatzzuweisung für Matrizen:

  `y=zeros(1,100); for i=1:100, y(i)=det(x^ i);end`

1 MATLAB®/Simulink® - Einführung

1.10 Visualisierung

2D-Darstellung (`doc graph2d`)

Befehl	Erläuterung
`plot`	2D-Grafik mit linearer Achsenteilung
Beispiele: `plot(x), plot(A), plot(x,y)` `plot(x,y,'m--'), plot(x,y,'o',x,2*y,'g:')`	
`plotyy`	2D-Grafik mit linearer Achsenteilung, 2 Ordinaten mit unterschiedlicher Skalierung
`loglog`	2D-Grafik mit logarithmischer Achsenteilung
`semilogx, semilogy`	2D-Grafik mit linearer Teilung der Abszisse und logarithmischer Teilung der Ordinate u. umgekehrt

1 MATLAB®/Simulink® - Einführung

1.10 Visualisierung

Grafik- und Achsenbeschriftung, allgemeine Grafikbefehle (`doc graph2d`, `doc graphics`)

Befehl	Erläuterung
`title`	Grafiktitel
`xlabel, ylabel`	Abszissen-, Ordinatenbeschriftung
`legend`	Grafiklegende
`text`	Text an Position (xpos,ypos); Bsp. `text(2,3,'Min.')`
`gtext`	Platzierung des Textes mittels Maus
`axis`	Achsenskalierung; Bsp. `axis([xmin xmax ymin ymax])`
`grid`	Gitternetzlinien
`hold on`	Beibehalten der Grafik (weitere Kurven in gleiche Grafik)
`hold off`	Freigabe der Grafik zum Überschreiben
`subplot`	Unterteilung eines Bildes (figure) in mehrere Grafiken

Bsp.: `subplot(3,2,1)` 6 Grafiken, 3x2, Ansprechen der 1. Grafik

1 MATLAB®/Simulink® - Einführung

1.10 Visualisierung

2D-Darstellung (`doc graph2d`)

Befehl	Erläuterung
`figure`	Erzeugung von Bildern (Grafikfenstern)/ Bild im Vordergrund
`clf`	Löschen des Bildinhaltes
`close [all]`	Schließen des/aller Fenster(s)
`gcf`	„Handle“ des aktuellen Bildes (**g**et **c**urrent **f**igure)
`gca`	„Handle“ des aktuellen Achsenkreuzes (**g**et **c**urrent **a**xis)
`get, set`	Lesen bzw. Setzen von Eigenschaften
`print`	Drucken von Grafikfensterinhalten

- Grafikobjekte: figure, axes, line, text, patch, surface, image, light, uicontrol, uimenu

1 MATLAB®/Simulink® - Einführung

1.10 Visualisierung

3D-Darstellung (`doc graph3d`)

Befehl	Erläuterung
`plot3`	Linien und Punkte im 3D-Raum
`meshgrid, mesh`	3D-Werteberechnung und 3D-Netz
`surf`	3D-Oberfläche
`colorbar`	Farbskalierung
`colormap`	Farbeinstellung
`view`	Beleuchtung, Schattierung, Ansicht (Blickwinkel)
`rotate, rotate3d`	interaktive Ansicht

1 MATLAB®/Simulink® - Einführung

1.10 Visualisierung

Spezielle grafische Darstellungen

Befehl	Erläuterung
`bar, barh, bar3, bar3h`	Balkendiagramme
`area`	ausgefüllte Flächen unter Kurven
`comet`	„Kometen-Bahn“
`errorbar`	Funktionsverlauf mit „Fehlerbereich“
`ezplot, fplot`	Grafik von Funktionen; Bsp.: `ezplot('sin(x)')`
`hist`	Histogramm (Häufigkeitsverteilung)
`pie, pie3`	2D- bzw. 3D-Tortendiagramm
`stem, stem3, stairs`	2D-, 3D-Verlauf mit Stützstellen, 2D-Stufenverlauf

Spezielle grafische Darstellungen (Fortsetzung)

`contour, contourf, contour3`	Iso-Liniendarstellung (2D, ausgefüllt) Iso-Liniendarstellung (3D)
`pcolor`	„Farbausdruck“ einer Matrix
`meshc, meshz, waterfall, surfc`	kombinierte Darstellungen (räumlich, Iso-Linien)
`slice`	Scheiben in einer bestimmten Ebene

- Bildverarbeitungsbefehle, Image Processing Toolbox

1 MATLAB®/Simulink® - Einführung

1.11 Simulink® – Simulation dynamischer Systeme

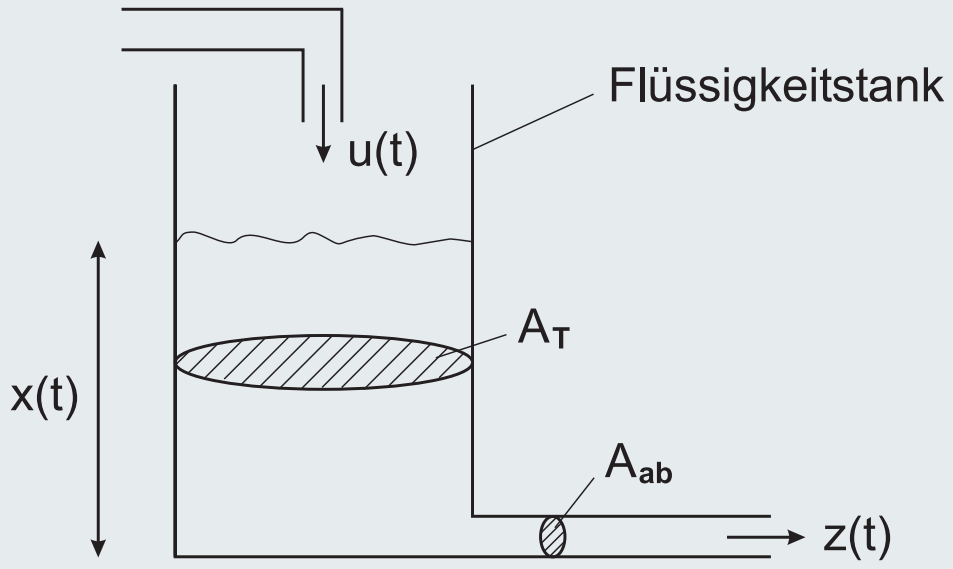
Modellierung, Simulation und Analyse dynamischer Systeme

- Erzeugen von Simulationsmodellen mit Hilfe hinterlegter Elemente in Bibliotheken: Continuous, Discrete, Nonlinear, Sources, Sinks, Math Operations, Signal Routing, Ports & Subsystems, User-Defined Functions, Simulink Extras
- Verbinden und Parametrieren der Elemente (Blöcke); auch selbst programmierte Inhalte
- Einstellen der Experimentierbedingungen
- Simulation
- Auswertung

Merkmale

- Zusammenfassung von Blöcken zu Untermodellen und Bibliotheken
- Kopplung zwischen MATLAB® und Simulink®, MATLAB®/Simulink® und anderen Programmiersprachen (z. B. C, C++, FORTRAN)
- Simulink®-Debugger

Beispiel: Tanksystem



- Variablen, Parameter: $0 \leq x(t) \leq 0.6$ m - Füllhöhe , $0 \leq u(t) \leq 10^{-4} \frac{m^3}{s}$ - Zufluss , $z(t)$ - Abfluss, $A_T = 0.0154$ m^2 - Tankquerschnitt, $A_{ab} = 3 \cdot 10^{-5}$ m^2 - Abflussquerschnitt.

Beispiel: Tanksystem (Fortsetzung)

- Systembeschreibung
  - Toricelli-Gesetz: $v_{ab} = \sqrt{2\,g\,x(t)}$
  - Kontinuitätsgleichung: $z(t) = A_{ab} v_{ab} = A_{ab}\sqrt{2\,g\,x(t)}$
  - Volumenbilanzgleichung

$$
\begin{aligned}
A_T\,\dot{x}(t) &= u(t) - z(t) \\
\dot{x}(t) &= \frac{1}{A_T}\left(u(t) - A_{ab}\sqrt{2\,g\,x(t)}\right)
\end{aligned}
$$

- Aufbau eines Simulink® -Simulationsdiagramms: `eintank.slx`, `eintank1.slx` - Modell mit interner/externer Eingangsgröße

1.11 Simulink® – Simulation dynamischer Systeme

Beispiel: Tanksystem (Fortsetzung)

- Simulation
- Arbeitspunktbestimmung (`et_l.slx`, externe Ein- und Ausgangsgröße) und Linearisierung (`eintank_ap_lin.m`)
- Sprungantwort des linearisierten Modells (`et_l_ss.slx`, Zustandsraumdarstellung)
- Verhalten im Regelkreis mit PI-Regler (`et_vlim.slx`)
- Verhalten im Regelkreis mit Zwei-Punkt-Regler (`et_zpr.slx`)

Kopplung MATLAB®-Simulink®, Simulink®-MATLAB®

- MATLAB®-Workspace (From/To Workspace-), MAT-Files (From/To File-Blöcke)
- MATLAB® unter Simulink® nutzen (Interpreted MATLAB Fcn, MATLAB Function, Fcn, S-function (system function))
- MATLAB® unter MATLAB® nutzen (SLX-File), Bsp:

```
[t,x,y]=sim('eintank1',[0 500],[],[[0:500]' ...
8e-5*ones(501,1)]);
```

Warum S-functions? (Level-2 S-function als M-File)

- Hinzufügen neuer algorithmischer Blöcke
- Einbeziehung z. B. existierenden C-Codes
- Systembeschreibung als mathematisches Gleichungssystem
- Grafische Animationen

Arbeitsweise von S-functions

- MATLAB®-Block mit Eingang, Ausgang und Zustand $x = [x_c^\top x_d^\top]^T$ mit x_c - kontinuerlicher Zustandsvektor und x_d - diskontinuierlicher Zustandsvektor; **Achtung!** : Direkter Durchgriff $\rightarrow$ Gefahr algebraischer Schleifen
- Simulationsschritte in MATLAB® (wiederholter Aufruf von Blöcken)

Aufbau von M-File S-functions

- M-File S-Function Template (Vorlage):
  matlabroot\`toolbox`\`simulink`\`blocks`\`sfuntmpl.m` (doc sfuntmpl, type sfuntmpl)
- Empfehlung: Orientierung an Beispielen

Simulationsschritt	Funktion
Initialisierung	mdlInitializeSizes
Berechnung nächster Abtastzeitpunkt (optional bei variabler Abtastzeit)	mdlGetTimeOfNextVarHit
Berechnung der Ausgänge	mdlOutputs
Aktualisierung der diskreten Zustände	mdlUpdate
Berechnung der Ableitungen (rechte Seite der Dgl.)	mdlDerivatives
Beenden der Simulation	mdlTerminate

Anpassung von M-File S-functions am Bsp. Tanksystem; MATLAB®-Modell: `et_m.slx`, M-FIle S-function: `et_mfsf.m`

Initialisierung: (`mdlInitializeSizes`)

```
Sizes.NumContStates=1; % Anzahl kontinuierl. Zustände
Sizes.Outputs=1; % Anzahl Ausgänge
Sizes.Inputs=1; % Anzahl Eingänge
Sizes.DirFeedthrough=0; % Durchgriff
x0=0.2; % Anfangszustand
```

Ausgangsgleichung: (`mdlOutputs`)

```
sys=x; % Ausgang(1)=Zustand(1)
```

Ableitung (rechte Seite der Dgl.):

```
sys=64.9612*(u(1)-3e-5*sqrt(2*9.81*x(1)));
```

Aufbau von CMEX S-functions

- CMEX S-Function Template (Vorlage):
  matlabroot\`simulink`\`src`\`sfuntmpl_basic.c`
  (type *matlabroot*\`simulink`\`src`\`sfuntmpl`\`basic.c`)
- Empfehlung: Orientierung an Beispielen

Simulationsschritt	Funktion
Initialisierung/Dimensionen	mdlInitializeSizes
Initialisierung/Anfangszustand	mdlInitializeConditions
Berechnung der Ausgänge	mdlOutputs
Aktualisierung der diskreten Zustände	mdlUpdate
Berechnung der Ableitungen (rechte Seite der Dgl.)	mdlDerivatives
Beenden der Simulation	mdlTerminate

Anpassung von CMEX S-functions am Bsp. Tanksystem; MATLAB®-Modell: `et_c.slx`, CMEX S-function: `et_cmsf.c`

Typ und Name der S-function:

```
# define S_FUNCTION_LEVEL 2
# define S_FUNCTION_NAME et_cmsf
```

Initialisierung/Dimensionen: (`mdlInitializeSizes`)

```
ssSetNumContStates(S, 1); /* Anzahl kontinuierl. Zustände */
ssSetOutputPortWidth(S, 0, 1); /* Anzahl Ausgänge */
ssSetInputPortWidth(S, 0, 1); /* Anzahl Eingänge */
ssSetInputPortDirectFeedThrough(S, 0, 1); /* Durchgriff */
```

Initialisierung/Anfangszustand: (`MDL_INITIALIZE_CONDITIONS`)

```
*x0=0.2; % Anfangszustand
```

Ausgangsgleichung: (`mdlOutputs`)

```
y[0]=x[0];
```

Ableitung (rechte Seite der Dgl.):

```
dx[0]=64.9612*(u(0)-3e-5*sqrt(2*9.81*x[0]));
```

1 MATLAB®/Simulink® - Einführung

1.11 Simulink® – Simulation dynamischer Systeme

Einbeziehung der CMEX S-function `et_cmsf.c`

- Übersetzung: `mex et_cmsf.c`; Erzeugung einer dynamischen Bibliothek (WINDOWS: et_cmsf.mexw32, et_cmsf.mexw64, Linux: et_cmsf.mexglx, et_cmsf.mexa64)
- Simulationsoptionen (Menü oder `simset`) und Simulation

Hinweise zur effizienten Simulation

- Vermeidung von „Interpreted MATLAB-Fcn-Blöcken“ (besser: Fcn mit elementaren mathematischen Funktionen)
- Vermeidung von M-File S-functions (etwas besser: CMEX S-function)
- Nachteilig: Memory-Blöcke
- Maximale Schrittweite zu klein, Zeitbasis zu lang?
- Zu große Genauigkeit? (Standard: 0.1 % meist ausreichend)
- Liegt steifes Dgl.-system vor? (`ode15s`)
- Verschiedene Abtastzeiten?
- Algebraische Schleifen?

1 MATLAB®/Simulink®

1.12 Simscape™ und Objektorientierung

Objektorientierte Erweiterungen

- Simscape™: Modellierungs- und Simulationsumgebung für physikalische Systeme (domäneübergreifend); Bibliotheken; Erweiterbarkeit; Modellaustausch; HiL; C-Code-Export
- SimPowerSystems™ (früher: SimPower™): Elektrische Energieversorgungssysteme
- SimMecanics™: Mehrkörper-Systeme (3D-Mechanik)
- SimElectronics™: elektronische und mechatronische Systeme

2 Kontinuierliche Regelungssysteme

2.1 Mathematische Modelle im Zeit,- Bild- und Frequenzbereich - Erzeugung, Konvertierung, Verknüpfu

Lineare Modelle

- Zeitbereich: Lineare (linearisierte) gewöhnliche Differenzialgleichungen n-ter Ordnung (Ein-/Ausgangsbeschreibung), n Dgln. erster Ordnung (Zustandsbeschreibung)
- Frequenzbereich, Bildbereich: Frequenzgang, Übertragungsfunktion (in unterschiedlichen Formen)

2 Kontinuierliche Regelungssysteme

2.1 Mathematische Modelle ...

Erzeugung/Konvertierung linearer Modelle

Übertragungsfunktion in Polynom-, Pol-Nullstellen-, Zeitkonstantenform; (Zähler-, Nennerpolynom, Pole/Nullstellen, Zeitkonstanten)

$$G(s) = \frac{Y(s)}{U(s)} = \frac{1}{10s^3 + 7s^2 + s} = \frac{0{,}1}{s(s+0{,}2)(s+0{,}5)} = \frac{1}{s(sT_1+1)(sT_2+1)}$$

- **tf** - (transfer function), Übertragungsfunktion (Polynomform)
  `sys_Gs_tf=tf(1,[10 7 1 0])` o. `s=tf('s');sys=1/(10*s^3+7*s^2+s)`
- **zpk** - (zero-pole-gain), Übertragungsfunktion (Pol-Nullstellen-Form)
  `sys_Gs_zpk=zpk([],[0 -0.2 -0.5],0.1)`
- **ss** - (state space), Matrizen der Zustandsbeschreibung
  `A=[-0.2 0.2 0;0 -0.5 0.5;0 0 0]; B=[0;0;1];C=[1 0 0];D=0;`
  `sys_Gs_ss=ss(A,B,C,D)` `sys_Gs_ss.A(1,2)` - Zugriff auf a_{12}
  `sys_ss1=ss(sys_Gs_tf)` - Konvertierung
- **c2d**, **d2c** - (continuous to discrete, discr. to cont.), nur Konvertierung!, Umwandlung von zeitkontinuierlichen und zeitdiskreten Modellen

2 Kontinuierliche Regelungssysteme

2.1 Mathematische Modelle ...

Verknüpfung von Teilsystemen

- **parallel, +** - Parallelschaltung
  `sys=parallel(sys1,sys2)` `sys=sys1 + sys2`
- **series, *** - Reihenschaltung
  `sys=series(sys1,sys2)` `sys=sys1*sys2`
- **feedback** - Rückführschaltung
  `sys=feedback(sys1,sys2)` (Negative Rückführung angenommen; angebbar)
- **minreal** - Minimalrealisierung (Kürzen von Polen und Nullstellen)
  `sys14=minreal(sys11*sys12)`

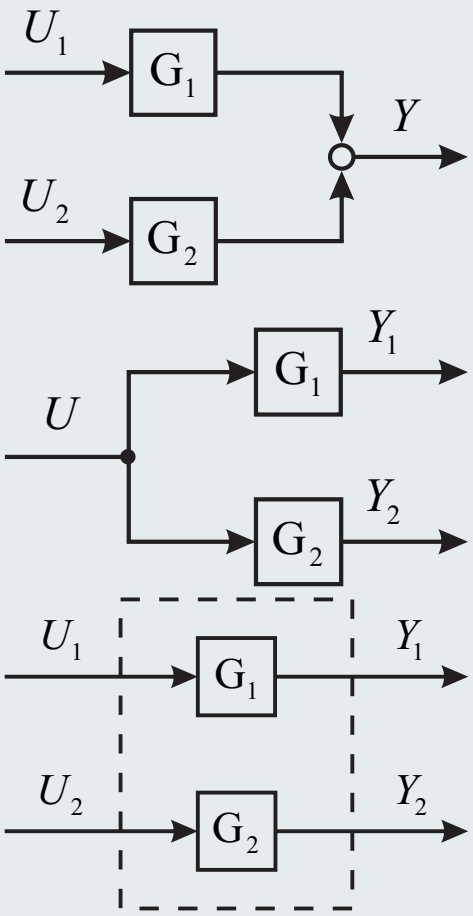
2 Kontinuierliche Regelungssysteme

2.1 Mathematische Modelle ...

Erweiterung von Modellen (1)

- Horizontal (MIMO-Modelle)
  `sys=[sys1, sys2]`
- Vertikal
  `sys=[sys1; sys2]`
- Blockdiagonal
  `sys=append(sys1,sys2)`

Zugehörige Grafiken



2 Kontinuierliche Regelungssysteme

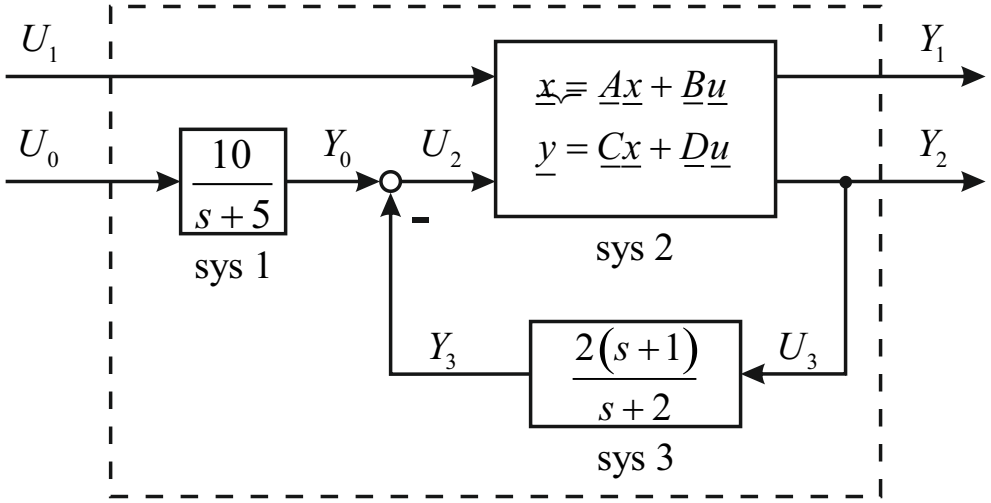
2.1 Mathematische Modelle ...

Erweiterung von Modellen (2); zugehörige Grafik: s. u.

Eingänge (E)		Ausgänge (A)	
Lfd. Nr.	Bez.	Lfd. Nr.	Bez.
1	U_0	1	Y_0
2	U_1	2	Y_1
3	U_2	3	Y_2
4	U_3	4	Y_3

```
                E   A   A        E   A   A
Q   =   [   3   1   -4   ;   4   3   0   ]      ;
```

Externe Ein-/Ausgänge: `inputs=[1 2]; outputs=[2 3];`



2 Kontinuierliche Regelungssysteme

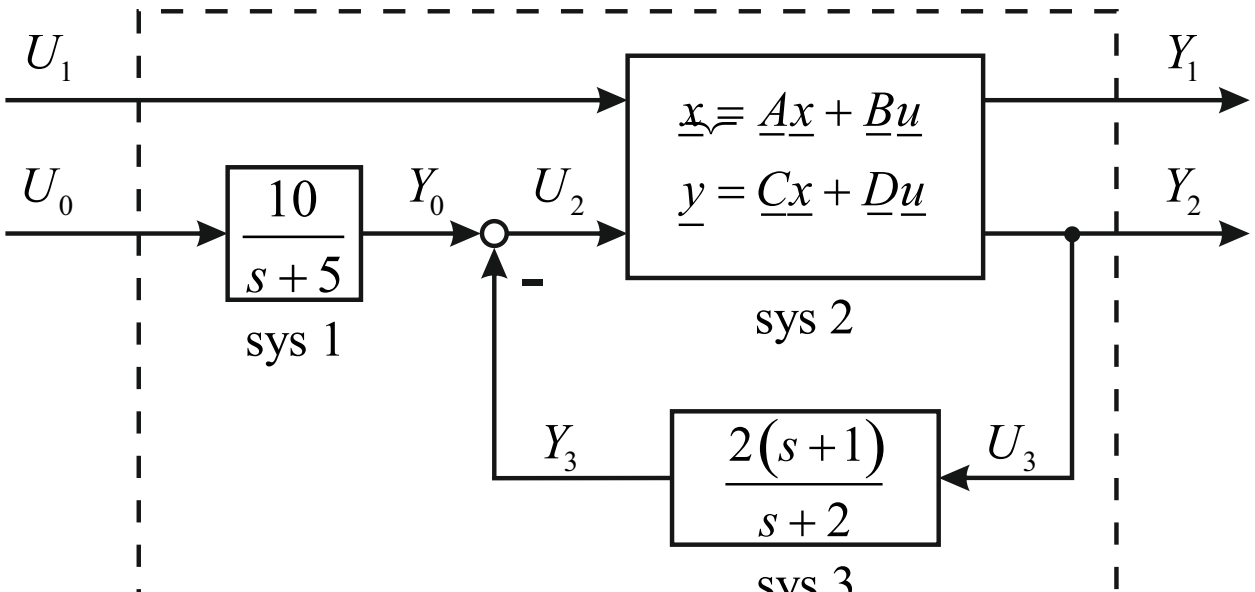
2.1 Mathematische Modelle ...

Erweiterung von Modellen (2); zugehörige Grafik: s. u.

- Zustandsbeschreibung aus Signalflussbild:

```
sys1=tf(10,[1 5])
sys2=ss(A,B,C,D)
sys3=zpk(-1,-2,2)
...
```

```
...
sys=append(sys1,sys2,sys3)
Q = [3 1 -4; 4 3 0]
inputs=[1 2]
outputs=[2 3]
sysconnected=connect(sys,Q,inputs,outputs)
```



Erweiterung von Modellen (3)

- Erweiterung des Ausgangsvektor um den Zustandsvektor
  `augsys=augstate(sys)`
- Felder von LTI-Modellen, z. B. 3 einzelne Modelle `sys1, sys2, sys3`
  `sysltia=[sys1,sys2,sys3]`
  `step(sysltia)` - effizienter als Einzelaufrufe
- Demo-Dateien: `weitere_modelle.m`

2 Kontinuierliche Regelungssysteme

2.2 Analyse im Zeitbereich

- **initial** - freie Bewegung als Funktion des Anfangszustandes (Zustandsraummodell)
  ```
  initial(sys_Gs_ss,[1;0;0])
  [y,t,x]=initial(sys_Gs_ss,[1;0;0])
  ```
- **step, impulse** - Sprung-, Impulsantwort
  ```
  [y,t]=step(sys_Gs_tf)
  [y,t,x]=step(sys_Gs_ss)
  plot(t,x)
  [y,t,x]=impulse(sys_Gs_ss)
  ```
- **gensig** - Testsignalerzeugung
  ```
  [u,t]=gensig('square',tau,tf,ts)
  ```
- **lsim** - Simulation des Zeitverhaltens linearer Systeme mit beliebigen Eingangssignal
  ```
  lsim(sys_Gs_ss,u,t,[0;0;0])
  [y,t,x]=lsim(sys_Gs_ss,u,t,[0;0;0])
  [y,t,x]=lsim(sys_Gs_tf,u,t)
  ```
- Demo-Dateien: `zeitbereich.m`, `ltiarrays.m`

2 Kontinuierliche Regelungssysteme

2.3 Analyse im Frequenzbereich

- **nyquist** - Nyquist-, Frequenzgangortskurve
  `nyquist(sys_Gs_ss)`
- **bode** - Bode-, Frequenzkennliniendiagramm
  ```
  bode(sys_Gs_ss)
  bode(sys1,'r',sys2,'g--')
  omega=logspace(-2,2,100);
  [mag(1:100),phase(1:100)]= ...
  bode(sys_Gs_ss,omega);
  semilogx(omega,20*log10(mag))
  semilogx(omega,phase)
  ```
- **margin** - Amplituden-, Phasenrand (grafisch)
  `margin(sys_Gs_ss)`
- **allmargin** - Amplituden-, Phasenrand (numerisch)
  `allmargin(sys_Gs_ss)`
- Demo-Dateien: `frequenzbereich.m`

2 Kontinuierliche Regelungssysteme

2.4 Analyse im Bildbereich

- **pzmap** - Pol-Nullstellen-Bild (PN-Bild)
  `pzmap(sys_Gs_ss)`
- **rlocus** - Wurzelortskurve
  `rlocus(sys_Gs_ss)`
- Demo-Dateien: `bildbereich.m`

2 Kontinuierliche Regelungssysteme

2.5 Weitere Funktionen zu Modelldynamik/Reglerentwur

Bereichsübergreifend

- Verhalten linearer zeitinvarianter Systeme:
  `linearSystemAnalyzer(sys1)` (vormals: `ltiview(sys1)`)
  `linearSystemAnalyzer({'bode';'step'},sys1)`
- Eigenwerte, Eigenfrequenzen, Dämpfungsfaktoren: `damp(sys)`
- Statische Verstärkung: `dcgain(sys)`
- Pole, Eigenwerte des Systems, Sortierung der Eigenwerte nach der Größe des Realteils: `pole(sys)`, `eig(sys)`, `esort(eig(sys))`
- Nullstellen des Systems: `zero(sys)`
- Demo-Dateien: `modelldynamik.m`

2 Kontinuierliche Regelungssysteme

2.5 Weitere Funktionen ...

Bereichsübergreifend

- Grafische Werkzeuge zum Entwurf von PID-Reglern/-Kompensatoren in einer Kreisstruktur (einschleifig/mehrschleifig; einschleifiger Standardregelkreis, Vorsteuerung, modellbasierte Regelung (engl.: internal model control (IMC)), Kaskadenregelung) für Eingrößensysteme (erfordert MATLAB® Control Design™ software):
  - `controlSystemDesigner (vormals: sisotool)`
    - Entwurfsmethoden: grafisch: Bode-Diagramm, Wurzelortskurve; automatisch: PID-Entwurf, optimierungsbasiert, LQG, IMC
    - Verschiedene grafische Darstellungen (Zeit-, Frequenz-, Bildbereich); gleichzeitige Änderungen sichtbar

2 Kontinuierliche Regelungssysteme

2.5 Weitere Funktionen ...

Bereichsübergreifend

- `pidTuner (vormals: pidtool)` - Auslegung von PID-Reglern
  $\rightarrow$ „Apps“; auch integriert in `controlSystemDesigner`
  - Standardkriterium: Wahl einer (Amplituden-)Durchtrittsfrequenz/ Schnittfrequenz (Bandbreite des geschlossenen Regelkreises)
    - Ziel: Phasenrand $\Phi_R = 60^\circ$
    - $\curvearrowright$ Berechnung der Verstärkung
  - Performance:
    (1) Sollwertfolge (track reference)
    (2) schnelle Störunterdrückung (suppress distubances rapidly)
  - Robustheit: genügend Phasen-/Amplitudenrand (wegen Modellungenauigkeiten und Änderungen der Systemdynamik)

2 Kontinuierliche Regelungssysteme

2.5 Weitere Funktionen ...

Bereichsübergreifend

- `pidTuner` - Fortsetzung
  - Regler mit 1 Freiheitsgrad (1-DOF): P, I, PI, PD, PDF, PID, PIDF (F=Filter)
  - Regler mit 2 Freiheitsgraden (2-DOF): PI2, PD2, PDF2, PID2, PIDF2;
    - schließt Sollwertwichtung bei proportionalem und Ableitungsterm ein;
    - $\Rightarrow$ schnelle Störungsbekämpfung ohne signifikantes Überschwingen bei der Sollwertfolge;
    - nützlich, um den Einfluss von Sollwertänderungen auf die Stellgröße abzumildern
  - Standard-Formel (b, c: Wichtungskoeffizienten für Proportional- und Ableitungsterm):

$$u = K_R \left[(bw - y) + \frac{1}{T_I s} (w - y) \frac{T_D s}{\frac{T_D}{N} s + 1} (cw - y) \right]$$

2.6 Einstellungen

Einstellungen

- ... für Control System Toolbox™ und System Identification Toolbox™: `ctrlpref` ; Einstellungen werden permanent gespeichert
- ... für `linearSystemAnalyzer`, `controlSystemDesigner`: Einstellungen werden temporär für die Dauer der Sitzung beibehalten
- ... für Antwort-Plots: Einstellungen werden temporär für die Dauer der Sitzung beibehalten

2 Kontinuierliche Regelungssysteme

2.7 Funktionen zur Nutzung der Zustandsbeschreibung

- Vollständige Steuerbarkeit: $rg(\mathbf{Q}_S) = n$, n - Systemordnung
  Steuerbarkeitsmatrix $\boldsymbol{Q}_S = \left[\boldsymbol{B}\ \boldsymbol{AB}\ \ldots\ \boldsymbol{A}^{n-1}\boldsymbol{B}\right]$
  `ctrb(A,B), rank(ctrb(A,B))`
- Vollständige Beobachtbarkeit: $rg(\mathbf{Q}_B) = n$
  Beobachtbarkeitsmatrix $\boldsymbol{Q}_B = \left[\boldsymbol{C}\ \boldsymbol{CA}\ \ldots\ \boldsymbol{CA}^{n-1}\right]^\top$
  `obsv(A,C), rank(obsv(A,C))`
- Regelungsnormalform (auch: Steuerungs-, FROBENIUS-Form, Begleitmatrix); aus der Dgl. bzw. Übertragungsfunktion; $\boldsymbol{T}$ - Transformationsmatrix `[csys,T]=canon(sys_Gs_ss,'companion');`
  `csys.A'`

$$G(s) = \frac{Y(s)}{U(s)} = \frac{s^n + b_{n-1}s^{n-1} + \ldots + b_1 s + b_0}{s^n + a_{n-1}s^{n-1} + \ldots + a_1 s + a_0}$$

$$\boldsymbol{A}_R = \begin{bmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & & \vdots \\ \vdots & \vdots & & \ddots & 0 \\ 0 & & \ldots & 0 & 1 \\ -a_0 & -a_1 & \ldots & & -a_{n-1} \end{bmatrix}$$

2 Kontinuierliche Regelungssysteme

2.7 Funktionen zur Nutzung der Zustandsbeschreibung

- Beobachtungsnormalform (auch: FROBENIUS-Form, , Begleitmatrix); aus der Dgl. bzw. Übertragungsfunktion (s. o.);
  `[bsys,T]=canon(sys_Gs_ss,'companion'); bsys.A`

$$\boldsymbol{A}_B = \begin{bmatrix} 0 & 0 & \dots & 0 & -a_0 \\ 1 & 0 & \dots & 0 & -a_1 \\ 0 & 1 & \ddots & \vdots & \vdots \\ \vdots & & \ddots & 0 & \\ 0 & \dots & 0 & 1 & -a_{n-1} \end{bmatrix}$$

Ähnlichkeitstransformation

- Ziel: Abhängigkeit der Zustandsableitung nur vom jeweiligen, transformierten Zustand

$$\begin{aligned} \boldsymbol{x} &= \boldsymbol{T}\hat{\boldsymbol{x}}\,, \quad \hat{\boldsymbol{x}} \text{ - transformierter Zustand} \\ \boldsymbol{T} &= [\boldsymbol{t}_1 \;\; \boldsymbol{t}_2 \;\; \ldots \boldsymbol{t}_n]\,, \quad \boldsymbol{T} \text{ - Transformationsmatrix} \\ & \qquad\qquad\qquad\qquad \boldsymbol{t}_i \text{ - Eigenvektoren} \end{aligned}$$

Einsetzen in Zustandsraumbeschreibung $\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}$:

$$\begin{aligned} \boldsymbol{T}\dot{\hat{\boldsymbol{x}}} &= \boldsymbol{A}\boldsymbol{T}\hat{\boldsymbol{x}} + \boldsymbol{B}\boldsymbol{u} \\ \dot{\hat{\boldsymbol{x}}} &= \underbrace{\boldsymbol{T}^{-1}\boldsymbol{A}\boldsymbol{T}}_{\hat{\boldsymbol{A}}=\boldsymbol{\Lambda}}\hat{\boldsymbol{x}} + \underbrace{\boldsymbol{T}^{-1}\boldsymbol{B}}_{\hat{\boldsymbol{B}}}\boldsymbol{u}\,, \quad \boldsymbol{\Lambda} = \begin{bmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \ldots & 0 & \lambda_n \end{bmatrix} \\ \boldsymbol{y} &= \underbrace{\boldsymbol{C}\boldsymbol{T}}_{\hat{\boldsymbol{C}}}\hat{\boldsymbol{x}} + \boldsymbol{D}\boldsymbol{u} \end{aligned}$$

2 Kontinuierliche Regelungssysteme

2.7 Funktionen zur Nutzung der Zustandsbeschreibung

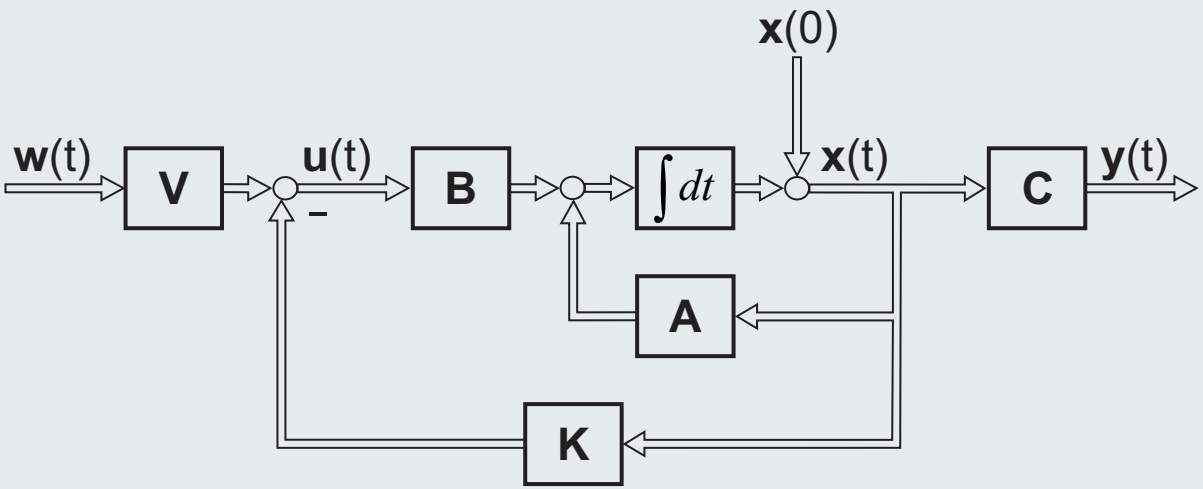
- Modaltransformation (JORDANsche Normalform); Bsp. für einfache Eigenwerte
  `msys=canon(sys,'modal')`

$$\boldsymbol{A}_J = \boldsymbol{\Lambda}\,, \qquad \text{s.o.}$$

- Ein-/Ausgangs-Balancierung im Zustandsraum mittels GRAMscher Matrix; mögliche Ordnungsreduktion:
  `balreal, gram, modred`
- Pole, Eigenwerte des Systems, Sortierung der Eigenwerte nach der Größe des Realteils
  `pole(sys_Gs_ss), eig(sys_Gs_ss), esort(eig(sys_Gs_ss))`
- Erweiterung des Ausgangsvektors um den Zustandsvektor $[\boldsymbol{y}\ \boldsymbol{x}]^\top$ und Anpassung der Zustandsbeschreibung
  `augsys=augstate(sys_Gs_ss)`
- Demo-Dateien: `modelle.m, fkt_abcd.m`

2 Kontinuierliche Regelungssysteme

2.8 Zustandsregelung



- $\sim$ für Eingrößensysteme (SISO) mittels Polplatzierung: `acker`
- $\sim$ für Mehrgrößensysteme (MIMO) mittels Polplatzierung: `place` (auch geeignet für Beobachterentwurf)
- RICCATI-Optimalregler ($t_f \to \infty$): `lqr`
- Demo-Dateien: `modelle.m, zustandsregler.m`

2 Kontinuierliche Regelungssysteme

2.9 Systeme mit Totzeit

- Zeitkontinuierlich:
  `GTt=tf(1,[1 1],'iodelay',0.1)`
  bzw. `GTt=zpk([],-1,1,'iodelay',0.1)`
  (Zustandsbeschreibung mit Totzeit hat nur eine Repräsentation im Frequenzbereich (E/A-Totzeit))
- Zwei Zustandsbeschreibungen:

$$
\begin{aligned}
(1a) \qquad & \dot{x}(t) = -x(t) + u(t-0.1) \\
(1b) \qquad & y(t) = x(t) \\
(2a) \qquad & \dot{z}(t) = -z(t) + u(t) \\
(2a) \qquad & y(t) = z(t-0.1)
\end{aligned}
$$

$\rightarrow$ liefern gleiche Übertragungsfunktion:

$$G(s) = \frac{1}{s+1} e^{-sT_t} \ , \ T_t = 0.1$$

```
M1=ss(-1,1,1,0,'inputdelay',0.1)
M2=ss(-1,1,1,0,'outputdelay',0.1)
```

2 Kontinuierliche Regelungssysteme

2.9 Systeme mit Totzeit

- Zeitdiskrete Systeme (3-fache Verzögerung gegenüber Abtastzeit T_s

$$G(z) \quad = \quad z^{-3}\frac{1}{z^2+0.5z+0.2}\ , \quad T_s = 0.1$$

  `G=tf(1,[1 0.5 0.2]),0.1,'inputdelay',3)` - effizient !
  bzw. `G=tf(1,conv([1 0 0 0],[1 0.5 0.2]),0.1)` - ineffizient !

- Weitere Befehle, Ansprechen von Objekteigenschaften
  `sys.iodelay, get(sys,'iodelay'), hasdelay(sys), totaldelay(sys)`

- Padé-Approximation von Totzeitsystemen
  `pade(0.1,3), pade(0.1,5), pade(0.1,10)`
  `sysx=pade(GTt,3), pzmap(sysx), rlocus(sysx), step(sysx)`

- Demo-Dateien: `modelle.m, totzeit.m`

2 Kontinuierliche Regelungssysteme

2.10 Simulink® Control Design™

- Analyse und Entwurf von Regelungssystemen, die mit Simulink® modelliert wurden
- Arbeitspunktbestimmung
- Linearisierung
- Klassischer PID-Reglerentwurf im Regelkreis an linearen/linearisierten Regelstrecken anhand Kenngrößen
  - des Regelgrößenverlaufs oder
  - des Bodediagramms
- Simulationsbasierte Frequenzkennlinien
- Unterschiedliche Regelkreisstrukturen (einschleifig, mehrschleifig, Kaskaden-Regelung, Regelung mit Vorsteuerung)
- s. ausführliche MATLAB®/Simulink®-Dokumentation

2 Kontinuierliche Regelungssysteme

2.11 Simulink® Design Optimization™

- Modellparameterschätzung, Zustandsschätzung und Modellparameteroptimierung unter Nutzung numerischer Optimierungsverfahren
- Sensitivitäsanalyse
- Entwurfsoptimierung von Systemantworten (z. B. Systemperformance, Energieoptimierung)
    - Modellparameter, Reglerparameter
    - Sowohl Optimierung des Zeit- als auch des Frequenzverhaltens; Einhaltung von Beschränkungen im Zeit- und Frequenzbereich (z.B. Überschwingweite, Phasenreserve)
    - Möglich: grafische Spezifizierung der Entwurfsanforderungen
    - s. ausführliche MATLAB®/Simulink®-Dokumentation

3 Optimierung

3.1 Einleitung, Motivation, Begriffe, Beispiele

Optimierung

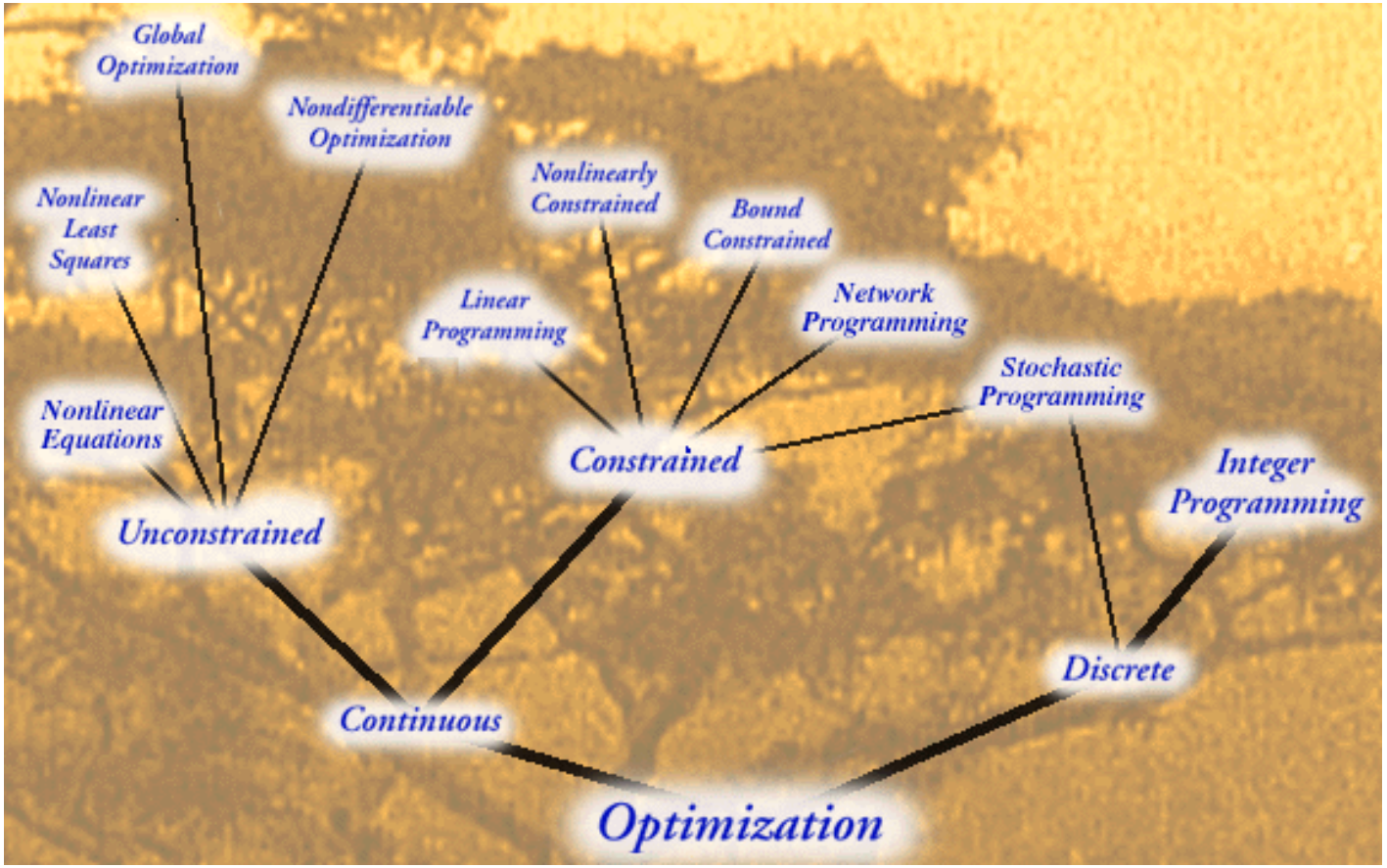
- umgangssprachlich: meist eine Verbesserung eines Vorgangs oder Zustands bzgl. eines Gesichtspunktes, (z. B. Qualität, Kosten, Geschwindigkeit, Effizienz), manchmal auch zu Lasten eines anderen Aspektes
- mathematisch: Bestimmung optimaler zulässiger Punkte eines Optimierungsproblems hinsichtlich einer gegebenen Zielfunktion, d. h. Finden eines Minimums (Maximums) einer Zielfunktion in einer gewissen Umgebung

Optimal lässt sich nicht steigern!

3 Optimierung

3.1 Einleitung, Motivation, Begriffe, Beispiele

Einteilung der Optimierung – eine (unvollständige) Übersicht



„NEOS Tree of optimization“

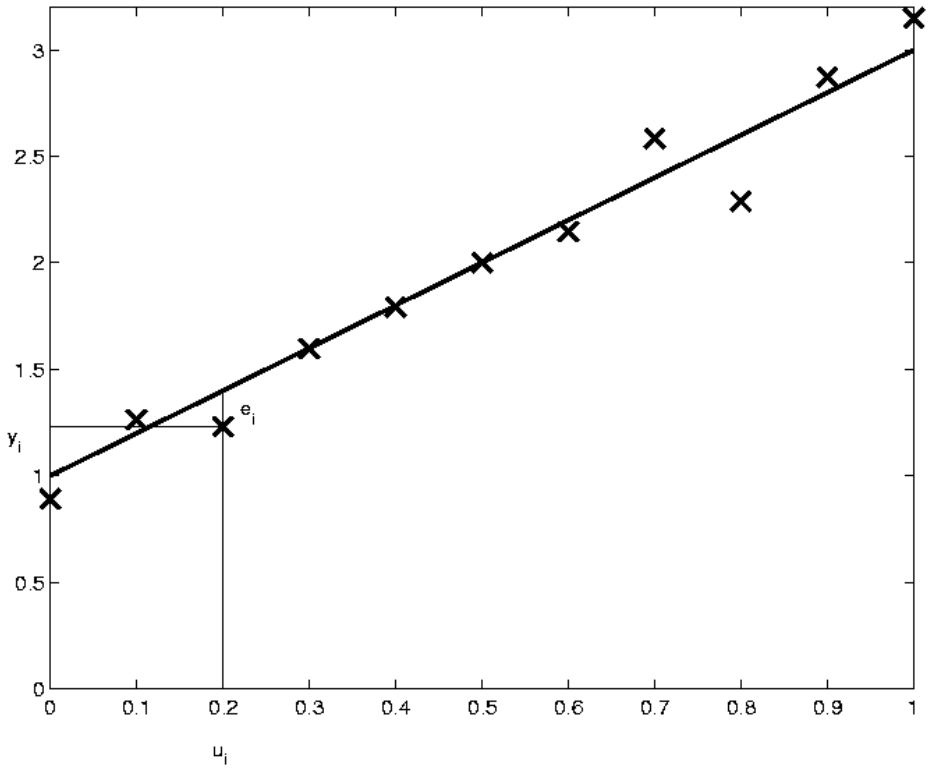
Übersicht: `http://www.neos-guide.org/content/optimization-tree-alphabetical`

3 Optimierung

3.1 Einleitung, Motivation, Begriffe, Beispiele

Beispiele für Optimierungsaufgaben

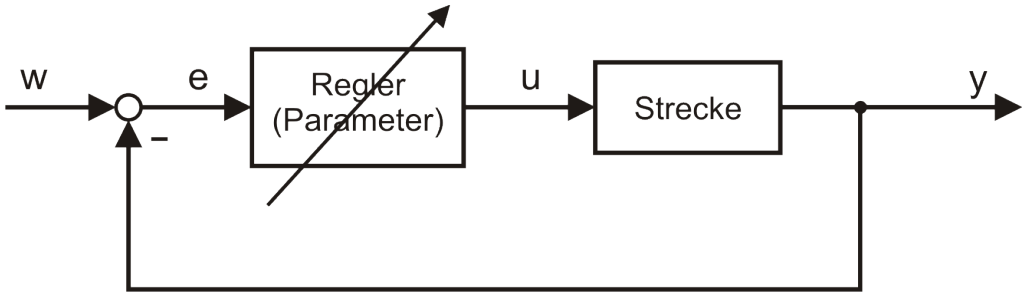
- Parameterschätz-, Modellbildungsproblem (Methode der kleinsten Fehlerquadrate: Minimiere die Summe der Fehlerquadrate!)



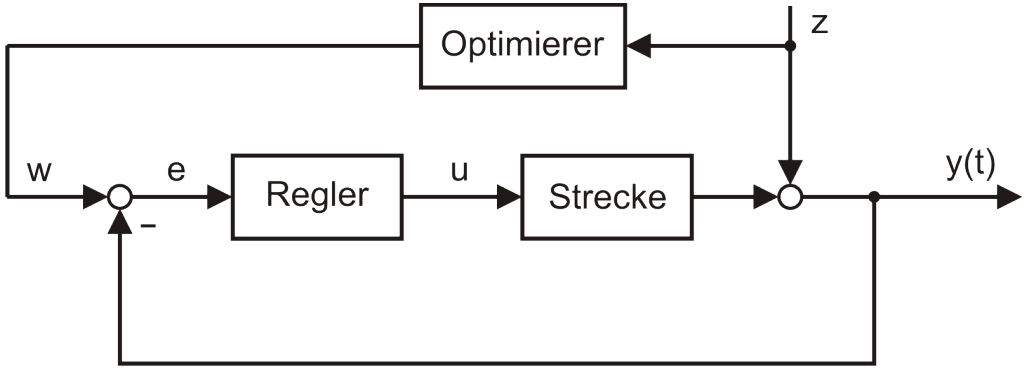
3 Optimierung

3.1 Einleitung, Motivation, Begriffe, Beispiele

- Reglerparameteroptimierung (Minimierung der quadratischen Regelfläche, Optimierung hinsichtlich einzelner Parameter des Führungsübergangs- vorganges, Mehrzielproblemstellung)



- Optimale Sollwert-, Arbeitspunktvorgabe



3 Optimierung

3.1 Einleitung, Motivation, Begriffe, Beispiele

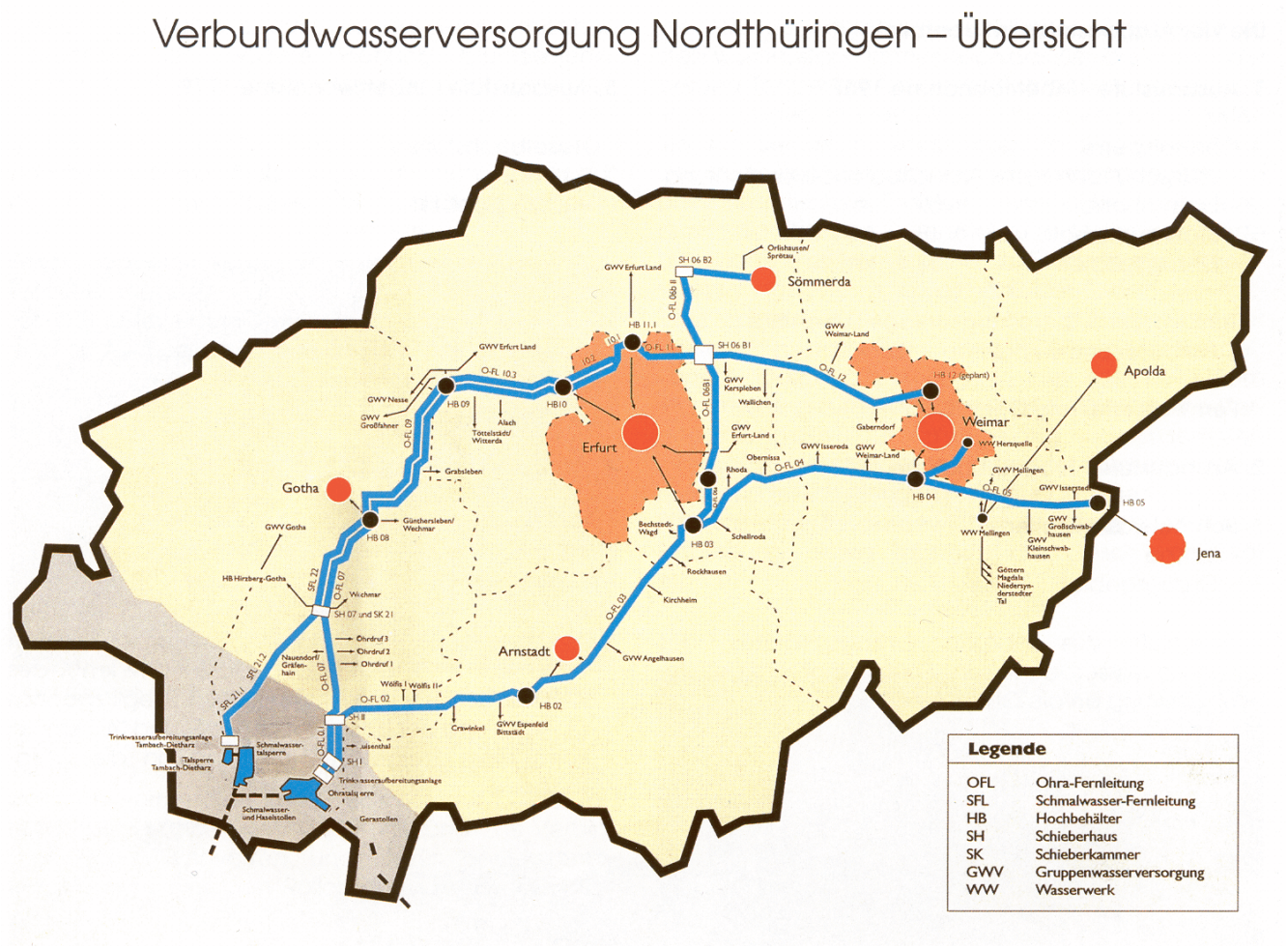
- Kostenoptimierung, Transportoptimierung, ...
- Dynamische Optimierung (Optimale (Um-)Steuerung, modellprädiktive Regelung, optimale Führungsgrößenvorgabe)

Anwendungsbeispiel aus einem Forschungsprojekt

- CHEWS - The impact of **C**limate change and other **H**ydrological events on **E**uropean **W**ater **S**upply planning and management
  - Pilotsystem in Thüringen: Talsperrensystem Ohra/Schmalwasser/Tambach-Dietharz mit seinen natürlichen Zuflüssen, Überleitungsstollen, Rohwasser- und Wildbettabgaben und der Anbindung an die Nordthüringer Fernwasserversorgung; Versorgung von ca. 700.000 Einwohnern in 220 Kommunen mit 88.000 m^3/d

3 Optimierung

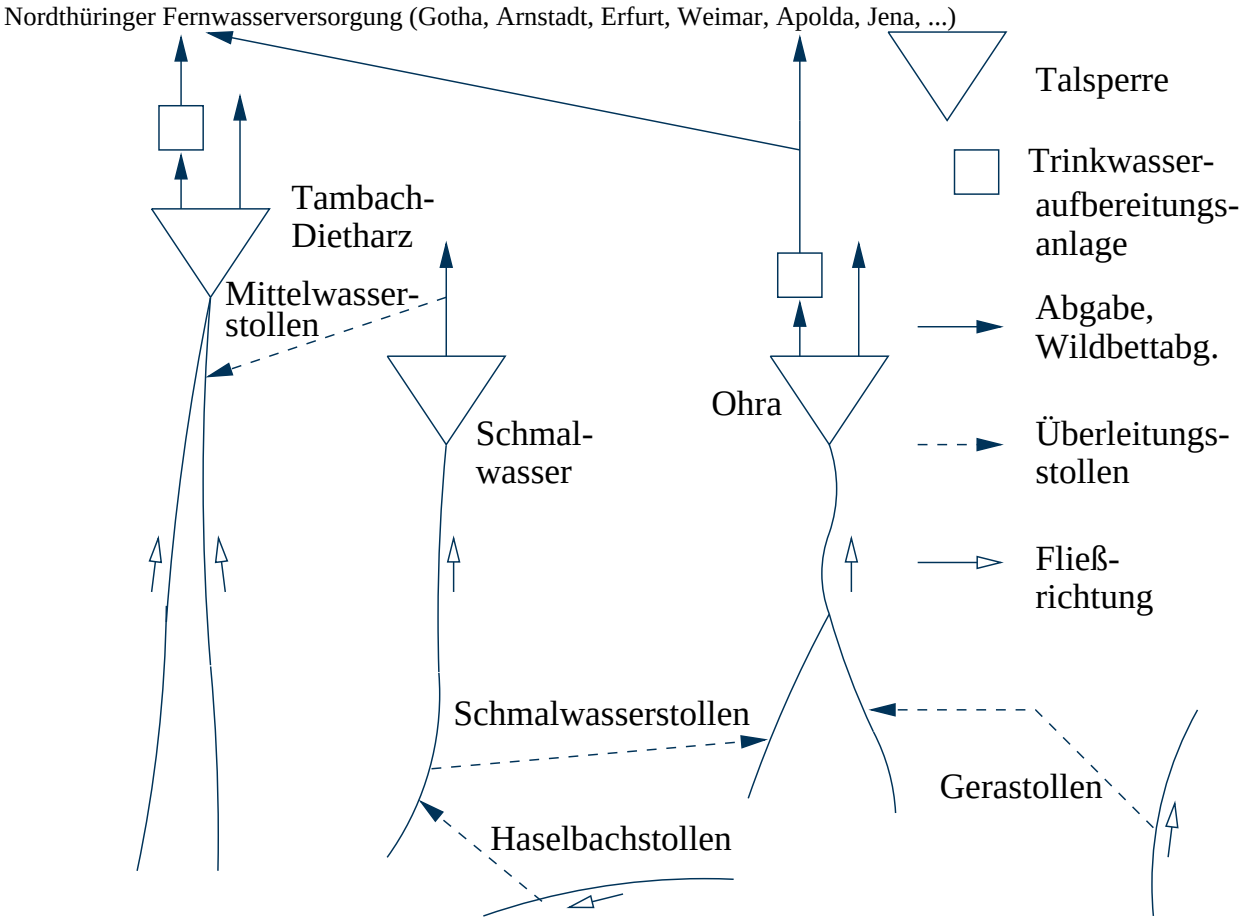
3.1 Einleitung, Motivation, Begriffe, Beispiele



Bildquelle: `www.thueringer-fernwasser.de`

3 Optimierung

3.1 Einleitung, Motivation, Begriffe, Beispiele



Talsperrensystem Ohra/Schmalwasser/Tambach-Dietharz

3 Optimierung

3.1 Einleitung, Motivation, Begriffe, Beispiele

- Ziel:
  Untersuchung des Einflusses von Klimaänderungen und der damit verbundenen Änderung der Bedingungen auf die künftige Wasserressourcenplanung und –bewirtschaftung
- Besonderheiten:
  Daten aus Klimamodellen, veränderte hydrologische Bedingungen, Planung durch Simulation der zukünftigen Bewirtschaftung
- Eingesetzte Methode
  SQP–Verfahren
- Ergebnisse:
  Geringe Veränderung der hydrologischen Bedingungen und damit geringfügig veränderte Bewirtschaftung in künftigen Jahren

3 Optimierung

3.1 Einleitung, Motivation, Begriffe, Beispiele

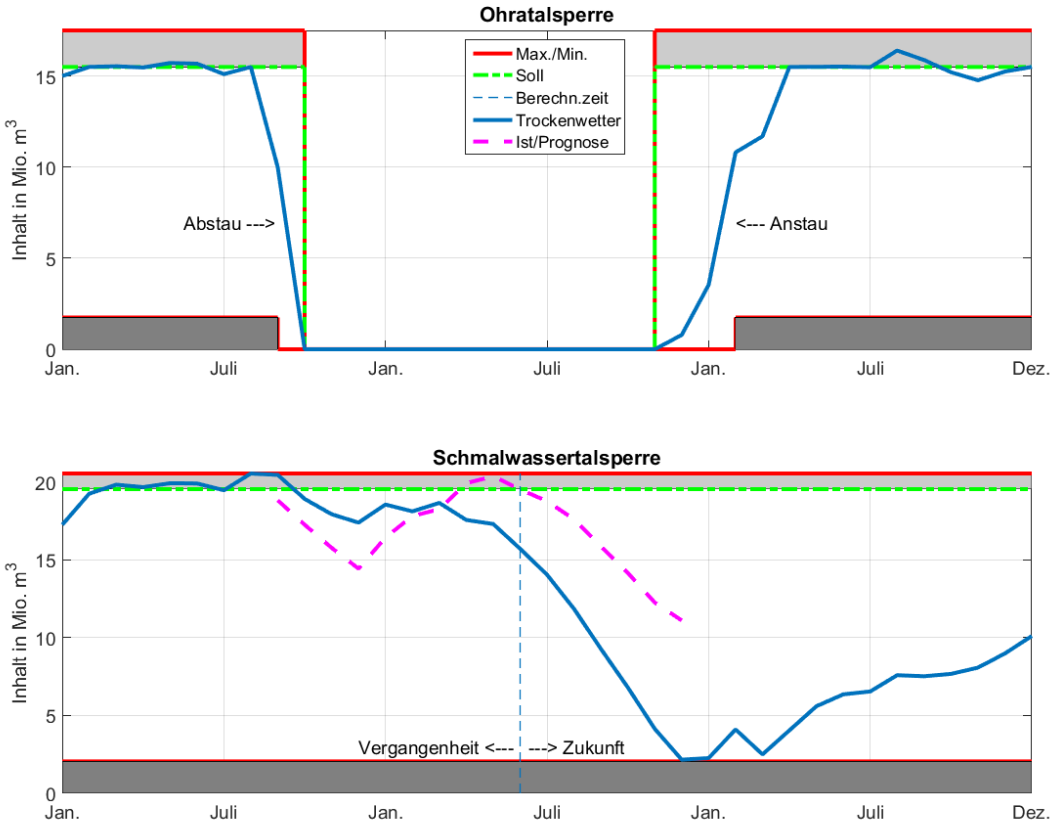
- Weiteres Szenario:
  Rekonstruktion der Talsperre Ohra (16 Monate)

Bildquelle: `www.thueringer-fernwasser.de`

- Aufgabe:
  Untersuchung, ob das Restsystem die Wasserversorgung in Normal- und Trockenwettersituation gewährleisten kann
- Berechnungen für Normal- und 50-jährige Trockenwettersituation
- Ergebnisse:
  Entscheidungshilfevorschläge für das Staatliche Umweltamt Erfurt

3 Optimierung

3.1 Einleitung, Motivation, Begriffe, Beispiele



Inhalte der Talsperren Ohra und Schmalwasser

3 Optimierung

3.1 Einleitung, Motivation, Begriffe, Beispiele

Optimierung mit MATLAB®

- MATLAB® Optimization Toolbox™: Lineare, quadratische, ganzzahlige und nichtlineare Optimierungsprobleme; Algorithmen: Simplex∼, Aktive-Restrik- tionen-∼, Sequentielle-Quadratische-Programmierungs∼, Innere-Punkte-∼
- MATLAB® Global Optimization Toolbox: Multi-modale und nicht-glatte Optimierungsprobleme; Algorithmen: Mustersuche, Genetische ∼, Partikelschwarm∼, Globale Suche
- Parameteroptimierung von Simulink®-Modellen
- Simulink® Control Design™
- Simulink® Design Optimization™

MATLAB® als Programmierumgebung

- Schnittstellen zu anderen Optimierungsprogrammen (MEX)
- Datenverwaltung, Ergebnisdarstellung, GUI, ...
- Zielfunktionsberechnungen (MATLAB®, Simulation von Systemen unter Simulink®, ...; Berechnung der Zielfunktion)

3 Optimierung

3.1 Einleitung, Motivation, Begriffe, Beispiele

MATLAB® Optimization Toolbox™ (1)

Name	Description
Nonlinear minimization of functions	
fminbnd	Scalar bounded nonlinear function minimization
fmincon	Multidimensional constrained nonlinear minimization
fminsearch	Multidimensional unconstrained nonlinear minimization, by Nelder-Mead direct search method
fminunc	Multidimensional unconstrained nonlinear minimization
fseminf	Multidimensional constrained minimization, semi-infinite constraints.hline
Nonlinear minimization of multi-objective functions	
fgoalattain	Multidimensional goal attainment optimization
fminimax	Multidimensional minimax optimization
Linear least squares (of matrix problems)	
lsqlin	Linear least squares with linear constraints
lsqnonneg	Linear least squares with nonnegativity constraints

MATLAB® Optimization Toolbox™ (2)

Name	Description
Nonlinear least squares (of functions)	
lsqcurvefit	Nonlinear curvefitting via least squares (with bounds)
lsqnonlin	Nonlinear least squares with upper and lower bounds
Nonlinear zero finding (equation solving)	
fzero	Scalar nonlinear zero finding
fsolve	Nonlinear system of equations solve (function solve)
Minimization of matrix problems	
intlinprog	Mixed-integer linear programming
linprog	Linear programming
quadprog	Quadratic programming
Controlling defaults and options	
optimoptions, optimset	Create optimization options
Graphical user interface	
optimtool	Optimization toolbox graphical user interface

3 Optimierung

3.1 Einleitung, Motivation, Begriffe, Beispiele

Problemformulierung

- Verbale Aufgabenstellung; Ziel: Finde die bestmögliche Einstellung (der freien Variablen), die die Zielfunktion, evtl. unter Berücksichtigung von Nebenbedingungen (Beschränkungen, Restriktionen), minimiert!
- Mathematische Notation eines Optimierungsproblems:

$$\left.\begin{array}{l} \min\limits_{\boldsymbol{x}\in X} \{J(\boldsymbol{x})|\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0},\, \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}\} \\ \boldsymbol{x} \in X,\ X = \{\boldsymbol{x} : \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0},\, \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}\} \subset R^n, \\ J(\boldsymbol{x}) : R^n \to R^1,\ \boldsymbol{h}(\boldsymbol{x}) : R^n \to R^m,\ \boldsymbol{g}(\boldsymbol{x}) : R^n \to R^r \end{array}\right\} (OP)$$

mit $\boldsymbol{x}$: Vektor der Optimierungs-, Entscheidungsvariablen

$J(\boldsymbol{x})$: Ziel-, Güte-, Kostenfunktion

$\boldsymbol{h}(\boldsymbol{x})$: Gleichungsnebenbedingungen (GNB)

$\boldsymbol{g}(\boldsymbol{x})$: Ungleichungsnebenbedingungen (UNB), auch in Form von Koordinatenbeschränkungen $\boldsymbol{x}_{min} \leq \boldsymbol{x} \leq \boldsymbol{x}_{max}$

3 Optimierung

3.1 Einleitung, Motivation, Begriffe, Beispiele

Überführung einer Maximierungsaufgabe in eine Minimierungsaufgabe

$$\begin{aligned} J^* &= J(\boldsymbol{x}^*) = \max_{\boldsymbol{x}} J(\boldsymbol{x}) \\ -J^* &= -J(\boldsymbol{x}^*) = \min_{\boldsymbol{x}} (-J(\boldsymbol{x})) \end{aligned}$$

Optimale Lösung

$$\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}} J(\boldsymbol{x}) = \arg\max_{\boldsymbol{x}} (-J(\boldsymbol{x}))$$

3 Optimierung

3.1 Einleitung, Motivation, Begriffe, Beispiele

Begriffe

- Zulässiger Punkt: $\boldsymbol{x} \in X$
- Zulässige Menge: $X = \{\boldsymbol{x} : \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}\ ,\ \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}\} \subset R^n$
- Globale und lokale Lösung

$$\begin{aligned} \text{global:} \quad & J(\boldsymbol{x}^*) \leq J(\boldsymbol{x}) \quad \forall\, \boldsymbol{x} \in X \\ \text{lokal:} \quad & J(\boldsymbol{x}^*) \leq J(\boldsymbol{x}) \quad \forall\, \boldsymbol{x} \in U(\boldsymbol{x}^*) \subseteq X\ , U(\boldsymbol{x}^*)\ -\ \text{Umgebung} \end{aligned}$$

Anmerkungen

- Globale Lösung ist anzustreben; meist schwierig zu bestimmen
- Verfahren konvergieren in der Regel gegen lokale Lösung; im Allgemeinen keine Aussage möglich, ob (globales) Optimum gefunden
- Globale Optimierung (Folge von lokalen Optimierungen; Heuristiken (Evolutionsstrategien, Genetische Algorithmen, hybride Verfahren))

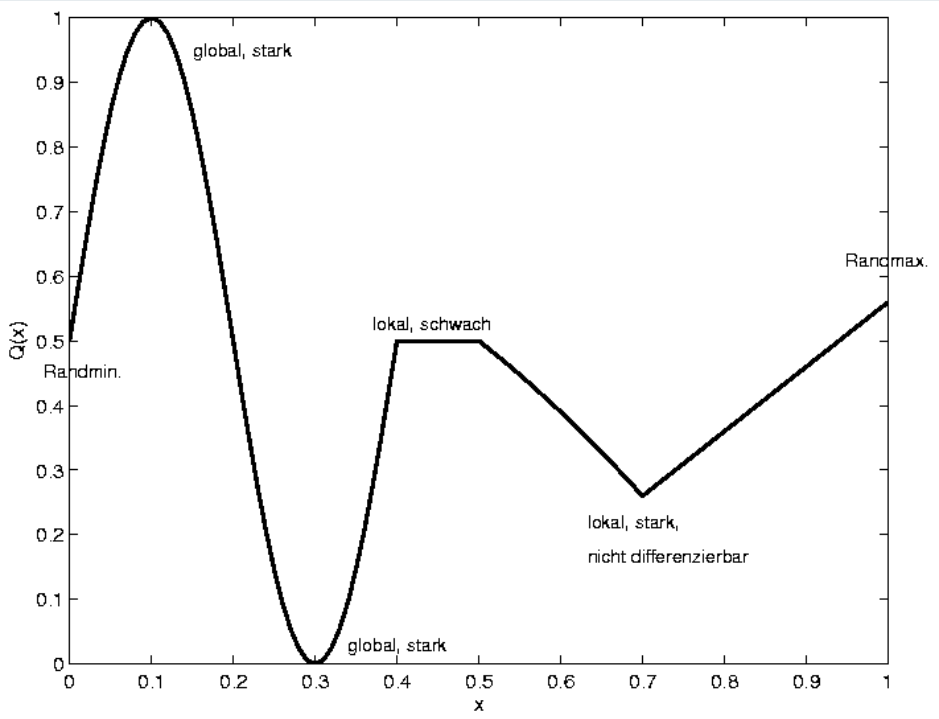
3 Optimierung

3.1 Einleitung, Motivation, Begriffe, Beispiele

Starkes und schwaches Minimum

$$\text{stark:} \quad J(\boldsymbol{x}^*) < J(\boldsymbol{x}) \quad \forall\, \boldsymbol{x} \in U(\boldsymbol{x}^*) \text{ mit } \boldsymbol{x} \neq \boldsymbol{x}^*$$

$$\text{schwach:} \quad J(\boldsymbol{x}^*) \leq J(\boldsymbol{x}) \quad \forall\, \boldsymbol{x} \in U(\boldsymbol{x}^*)$$



3 Optimierung

3.1 Einleitung, Motivation, Begriffe, Beispiele

Kontinuierliche und diskrete Probleme

- Kontinuierlich: $\boldsymbol{x} \in R^n$
- Diskret: $\boldsymbol{x} \in Z^n$ (ganzzahlig)
- Gemischt-ganzzahliges Problem (engl.: mixed-integer problem): Variablen aus beiden Mengen; Problem schwieriger zu lösen („branch and bound"; „Näherungslösung": Lösung eines kontinuierlichen Problems und nachfolgende Rundung

Ein- und mehrkriterielles Problem

- Im Gegensatz zu bisher betrachteten einkriteriellen Problemen: mehrkriterielles Problem, d. h.

$$\min_{\boldsymbol{x}} \boldsymbol{J}(\boldsymbol{x}) \ , \ \boldsymbol{J}(\boldsymbol{x}) : R^n \to R^r$$

- Keine eindeutige Lösung
- PARETO-Menge: kein Kriterium kann verbessert werden, ohne ein anderes zu verschlechtern

3 Optimierung

3.1 Einleitung, Motivation, Begriffe, Beispiele

Ein- und mehrkriterielles Problem (Fortsetzung)

- Transformation in einkriterielles Problem:

$$\min_{\boldsymbol{x}} \sum_{i=1}^{r} \alpha_i J_i(\boldsymbol{x}) \qquad \alpha_i \geq 0, \quad \text{z. B. } \sum_{i=1}^{r} \alpha_i = 1$$

- Resultat: eine spezielle Lösung für gewählte α-Werte; Variation der α_i

Größe (Dimension) des Optimierungsproblems

- Anzahl der Optimierungsvariablen
- Hochdimensionale Probleme (engl.: large-scale): stark anwachsender Rechenaufwand
- Ausnutzung der Struktur (Ableitungen; schwache Besetzheit von Matrizen)
- Grenze fließend zwischen niedrig- und hochdimensional:
  $n = 10 \ldots 10.000$: niedrigdimensional; $n \gtrsim 10.000$: hochdimensional

3 Optimierung

3.1 Einleitung, Motivation, Begriffe, Beispiele

Deterministische und stochastische Optimierungsprobleme

- Im Gegensatz zu bisher betrachteten deterministischen Problemen: stochastisches Problem: Zufallsvariable in Zielfunktion oder Beschränkungen
- Kriterium: z. B. Erwartungswert der Zielfunktion
- Lösungsaufwand in der Regel höher

Statische und dynamische Optimierung

- Besser: Optimierung des statischen bzw. dynamischen Systemverhaltens; (nichtlineare) Optimierung bzw. optimale Steuerung
- Optimale Steuerung: Optimierungsvariable sind Zeitfunktionen $\boldsymbol{x}(t)$
- Unterschiedliche Lösungstechniken: z. B. Überführung in endlich-dimensionales Optimierungsproblem: Steuerungsparametrisierung, Kollokation, ...

3 Optimierung

3.1 Einleitung, Motivation, Begriffe, Beispiele

Ableitungen

- Gradient (Ableitung einer Funktion nach einem Vektor)

$$\nabla J(\boldsymbol{x}) = \frac{\partial J(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial J}{\partial x_1} \\ \vdots \\ \frac{\partial J}{\partial x_n} \end{bmatrix}$$

- Jacobi-Matrix (Ableitung einer Vektorfunktion nach einem Vektor)

$$\nabla \boldsymbol{J}(\boldsymbol{x}) = \frac{\partial \boldsymbol{J}(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial J_1}{\partial x_1} & \cdots & \frac{\partial J_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial J_n}{\partial x_1} & \cdots & \frac{\partial J_n}{\partial x_n} \end{bmatrix}$$

3 Optimierung

3.1 Einleitung, Motivation, Begriffe, Beispiele

Ableitungen (Fortsetzung)

- HESSE-Matrix (Zweite Ableitung einer Funktion nach einem Vektor)

$$\nabla^2 J(\boldsymbol{x}) = \frac{\partial^2 J(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial^2 J}{\partial x_1^2} & \cdots & \frac{\partial^2 J}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 J}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 J}{\partial x_n^2} \end{bmatrix}$$

3 Optimierung

3.2 Unbeschränkte Optimierung

Optimierungsproblem und notwendige Optimalitätsbedingungen

- Optimierungsproblem

$$\min_{\boldsymbol{x}} J(\boldsymbol{x}) \ , \quad \boldsymbol{x} \in R^n, \ J(\boldsymbol{x}) : R^n \to R^1,$$

- Notwendige Optimalitätsbedingung
  Voraussetzung: stetige Differenzierbarkeit von $J(\boldsymbol{x})$ in der Umgebung eines lokalen Minimums $\boldsymbol{x}^*$:

$$\nabla J(\boldsymbol{x}^*) \quad = \quad 0 \qquad \text{(nichtlineares Gleichungssystem)}$$

- Hinreichende Optimalitätsbedingung für ein starkes Minimum:
  Voraussetzung: stetige Differenzierbarkeit von $J(\boldsymbol{x})$ und Stetigkeitkeit von $\nabla^2 J(\boldsymbol{x})$ in der Umgebung eines lokalen Minimums $\boldsymbol{x}^*$:

$$\nabla J(\boldsymbol{x}^*) \quad = \quad 0 \qquad \text{und} \qquad \nabla^2 J(\boldsymbol{x}^*) \text{ positiv definit}$$

3 Optimierung

3.2 Unbeschränkte Optimierung

Beispiele für ableitungsfreies und ableitungsbehaftetes Optimierungsverfahren:

Simplex-Verfahren nach NELDER-MEAD

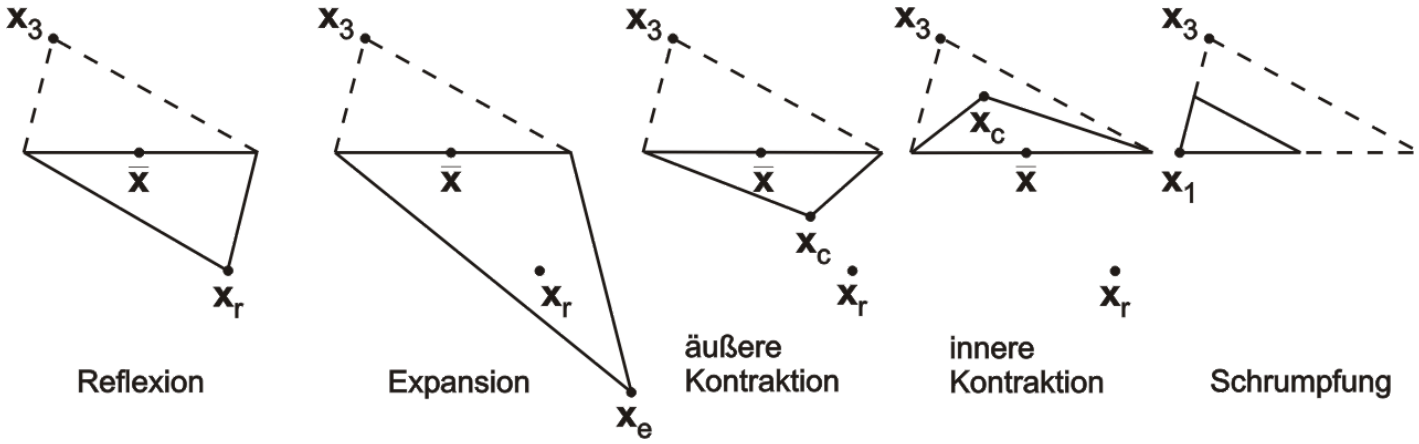
- Direktes Verfahren; ableitungsfrei; besonders geeignet für niedrigdimensionale Probleme
- Kein Konvergenzbeweis; Gegenbeispiele; trotzdem häufig eingesetzt; ein Standardverfahren der MATLAB® Optimization Toolbox
- Simplex: $(n+1)$-Eck im n-dimensionalen Raum
  (! Achtung ! : Namensähnlichkeit mit Simplexalgorithmus zur Lösung linearer Optimierungsprobleme (bzw. linearer Gleichungssysteme))
- Ablauf:
  - Wahl eines Startsimplex $\boldsymbol{x}^{0,i}$, $i \in [1, n+1]$
  - Wahl von Verfahrensparametern ρ, ξ, γ, σ
  - Wahl der Abbruchschranke ε
  - Iterationszähler $k := 0$

3 Optimierung

3.2 Unbeschränkte Optimierung

Simplex-Verfahren nach NELDER-MEAD (Fortsetzung)

- Ablauf (Fortsetzung):
  - Wiederhole folgende Anweisungen solange, bis die Abbruchbedingung $\max\limits_{2 \le i \le n+1} ||\boldsymbol{x}^{k,i} - \boldsymbol{x}^{k,1}|| \le \varepsilon \max\{1, ||\boldsymbol{x}^{k,1}||\}$ erfüllt ist.
    - Sortiere $J(\boldsymbol{x}^{k,1}) \le J(\boldsymbol{x}^{k,2}) \le \ldots \le J(\boldsymbol{x}^{k,n+1})$
    - Tausche $\boldsymbol{x}^{k,n+1}$ aus durch Reflexion, Expansion, Kontraktion bzw. tausche $\boldsymbol{x}^{k,i}$, $i \in [2, n+1]$ durch Schrumpfung
    - $k := k + 1$



3 Optimierung

3.2 Unbeschränkte Optimierung

Quasi-Newton-Verfahren (Variable-Metrik-V.)

- Ausnutzung der 1. und 2. Ableitung (Gradient, Hesse-Matrix)
- Formulierung eines quadratischen Problems (bzw. Approximation)

$$\min_{\boldsymbol{x}}\{\frac{1}{2}\boldsymbol{x}^\top \boldsymbol{H}\boldsymbol{x} + \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{b}\}$$

$$\begin{aligned} \nabla J(\boldsymbol{x}) &= 0 = \boldsymbol{H}\boldsymbol{x}^* + \boldsymbol{c} \\ \boldsymbol{x}^* &= -\boldsymbol{H}^{-1}\boldsymbol{c} \end{aligned}$$

- Annäherung der Hesse-Matrix durch Rekursionsformel (bzw. ihrer Inversen) nach Broyden, Fletcher, Goldfarb, Shanno, BFGS-Formel:

$$\begin{aligned} \boldsymbol{H}_{k+1} &= \boldsymbol{H}_k + \frac{\boldsymbol{q}_k \boldsymbol{q}_k^\top}{\boldsymbol{q}^\top \boldsymbol{s}_k} - \frac{\boldsymbol{H}_k \boldsymbol{s}_k^\top \boldsymbol{s}_k \boldsymbol{H}_k}{\boldsymbol{s}_k^\top \boldsymbol{H}_k \boldsymbol{s}_k} \\ \text{mit} \quad & \boldsymbol{s}_k = \boldsymbol{x}_k - \boldsymbol{x}_{k-1} \\ & \boldsymbol{q}_k = \nabla J(\boldsymbol{x}_k) - \nabla J(\boldsymbol{x}_{k-1}) \end{aligned}$$

Allgemeiner Ablauf numerischer Suchverfahren (hier: Konkretisierung für Quasi-Newton-Verfahren

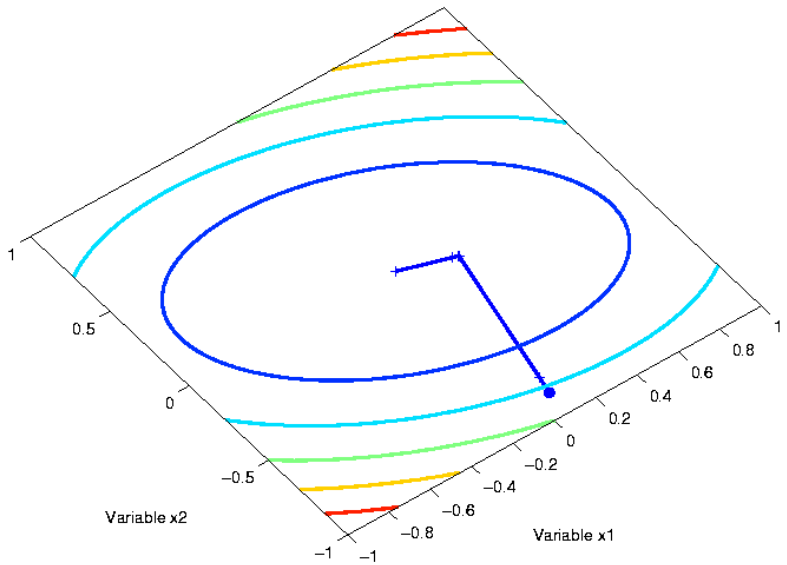
- Wähle Startpunkt $\boldsymbol{x}_0$
- Wähle Verfahrensparameter (z.B. Abschaltschranke ε, Anfangsschrittweite α_0, Startnäherung für (Inverse der) Hesse-Matrix $\boldsymbol{H}_0$)
- Setze Iterationszähler $k := 0$
- Führe folgende Anweisungen solange aus, bis die Abschaltbedingung erfüllt ist
    - Bei ableitungsbehafteten Verfahren: Bilde Gradienten $\nabla J(\boldsymbol{x}_k)$
    - Abschaltbedingung erfüllt ? Ja: Abbruch, Nein: Weiter.
      Bei ableitungsbehafteten Verfahren: $||\nabla J(\boldsymbol{x}_k)|| \leq \varepsilon$
      Bei ableitungsfreien Verfahren ab der 1. Iteration:
      $|J(\boldsymbol{x}_k) - J(\boldsymbol{x}_{k-1})| \leq \varepsilon$

3 Optimierung

3.2 Unbeschränkte Optimierung

Allgemeiner Ablauf ... (Fortsetzung)

- Anweisungsfolge (Fortsetzung):
  - Bestimme Suchrichtung $\boldsymbol{d}_k$ je nach Verfahren (hier: $\boldsymbol{d}_k = -\boldsymbol{H}_k^{-1}\nabla J(\boldsymbol{x}_k)$; dazu: Bestimmung von $\boldsymbol{H}_k$ bzw. $\boldsymbol{H}_k^{-1}$ nach Aufdatierungsvorschrift (s.o.))
  - Bestimme optimale Schrittweite $\alpha_k^* = \arg\min_{\alpha_k} J(\boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k)$
  - Berechne $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k^* \boldsymbol{d}_k$
  - $k := k + 1$



Beispiele

- Demonstrationsbeispiele (quadratische, allgemeine nichtlineare) aus dem Praktikumsversuch „Statische Prozessoptimierung - StatPO-1“
- Verfahren
  - Ableitungsbehaftete Verfahren (Gradienten-, Konjugierte-Gradienten-, Quasi-Newton-Verfahren)
  - Ableitungsfreie Verfahren (Nelder-Mead-Simplex-Suchverfahren, Koordinatensuchverfahren, Rosenbrock-Verfahren, Evolutionsstratregien)

Methode der kleinsten Fehlerquadrate - Varianten

1. Lösung eines linearen Gleichungssystems im Sinne der kleinsten Fehlerquadrate; zusätzliche lineare Gleichungen und/oder lineare Ungleichungen, Koordinatenbeschränkungen

   Optimierungsproblem (OP), s. S. 98:

$$\min_{\boldsymbol{x}\in X} \{J(\boldsymbol{x}) | \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0},\ \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}\}$$

$$\begin{aligned} \text{mit} \quad J(\boldsymbol{x}) &= \frac{1}{2}||\boldsymbol{C}\boldsymbol{x} - \boldsymbol{d}||_2^2 \\ \boldsymbol{h}(\boldsymbol{x}) &= \boldsymbol{A}_{eq}\boldsymbol{x} - \boldsymbol{b}_{eq} = \boldsymbol{0} \\ \boldsymbol{g}(\boldsymbol{x}) &= \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \leq \boldsymbol{0} \end{aligned}$$

   Beschränkungsnotation unter MATLAB®:

$$\begin{aligned} \boldsymbol{A}_{eq}\boldsymbol{x} &= \boldsymbol{b}_{eq} \\ \boldsymbol{A}\boldsymbol{x} &\leq \boldsymbol{b} \\ \boldsymbol{x}_{min} \leq \boldsymbol{x} &\leq \boldsymbol{x}_{max} \end{aligned}$$

Methode der kleinsten Fehlerquadrate - Varianten

- Gegebene Messwerte: Eingangsvektor $\boldsymbol{u} = [u_1\ u_2\ u_3]^\top = [1\ 2\ 3]^\top$, Ausgangsvektor $\boldsymbol{y} = [y_1\ y_2\ y_3]^\top = [0{,}9\ 1{,}95\ 3{,}1]^\top$
- Funktioneller Zusammenhang: lineare Gleichungen; $y_i = f(u_i) = x_1 u_i + x_2$ mit Parametervektor $\boldsymbol{x} = [x_1\ x_2]^\top$, $i = 1{,}2{,}3$.
- Lösung des linearen Gleichungssystems im Sinne der kleinsten Fehlerquadrate

$$\boldsymbol{C}\boldsymbol{x} = \boldsymbol{d} \qquad d.\ h.$$

$$u_i x_1 + x_2 = y_i\,, \qquad i = 1{,}2{,}3$$

$$\boldsymbol{C}\boldsymbol{x} - \boldsymbol{d} = \boldsymbol{0}$$

$$\begin{bmatrix} u_1 & 1 \\ u_2 & 1 \\ u_3 & 1 \end{bmatrix} \boldsymbol{x} - \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \boldsymbol{0}$$

Aufruf: `u=[1 2 3]';y=[0.9 1.95 3.1]';`

`result_lsqlin=lsqlin([u ones(size(u))],y)` bzw.

`result_division=[u ones(size(u))]\y`

Methode der kleinsten Fehlerquadrate - Varianten

2. Lineares Modell, Koordinatenbeschränkungen (Positivitätsforderung $\boldsymbol{x} \geq 0$)

   Optimierungsproblem (OP), s. S. 98:

$$\min_{\boldsymbol{x} \in X} \{J(\boldsymbol{x}) | \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}\}$$

$$\text{mit} \quad J(\boldsymbol{x}) = \frac{1}{2}||\boldsymbol{C}\boldsymbol{x} - \boldsymbol{d}||_2^2$$

$$\boldsymbol{g}(\boldsymbol{x}) = -\boldsymbol{x} \leq \boldsymbol{0}$$

   Aufruf: `u=[1 2 3]';y=[0.9 1.95 3.1]';`

   `result_lsqnonneg=lsqlin([u ones(size(u))],y)`

   Datei: `bsp_lsqnonneg.m`

Methode der kleinsten Fehlerquadrate - Varianten

3. Nichtlineare Modelle und kleinste Fehlerquadrate; Koordinatenbeschränkungen

   Optimierungsproblem (OP), s. S. 98:

$$\min_{\boldsymbol{x}\in X}\left\{||J(\boldsymbol{x})||_2^2 = J_1^2(\boldsymbol{x}) + \ldots + J_p^2(\boldsymbol{x}) \middle| \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}\right\}$$
$$\boldsymbol{g}(\boldsymbol{x}) \;\leq\; \boldsymbol{0}$$

   Im Bsp.: eine Zielfunktion, eine Optimierungsvariable

   Aufruf: ...

```
f_lsqnonlin1=@(a)exp(-a*u)-y_mess;
[x,resnorm]=lsqnonlin(f_lsqnonlin1,-1)
```

   Datei: `bsp_lsqnonlin.m`

   Auch: Anwendung von `fmincon` möglich, s. u.

Methode der kleinsten Fehlerquadrate - Varianten

4. Nichtlineare Modelle und kleinste Fehlerquadrate; Kurvenanpassung (engl.: curve fitting); Koordinatenbeschränkungen

   Optimierungsproblem (OP), s. S. 98:

$$\min_{\boldsymbol{x}\in X}\left\{||J(\boldsymbol{x},\boldsymbol{x}_{mess})-\boldsymbol{y}_{mess}||_2^2=\sum_{i=1}^{p}(J_i(\boldsymbol{x},\boldsymbol{x}_{mess})-\boldsymbol{y}_i)^2\Big|\boldsymbol{g}(\boldsymbol{x})\leq\boldsymbol{0}\right\}$$

$$\boldsymbol{g}(\boldsymbol{x})\ \leq\ \boldsymbol{0}$$

   Aufruf: ...

```
fun_decay = @(x,xdata)x(1)*exp(x(2)*xdata)
x = lsqcurvefit(fun_decay,x0,xdata,ydata)
```

   Datei: `bsp_lsqcurvefit.m`

Lineare Optimierung

(Programmierung=Synonym für Optimierung)
Optimierungsproblem mit linearer Zielfunktion, linearen Gleichungs- und Ungleichungsnebenbedingungen, s. S. 98:

$$\min_{\boldsymbol{x}\in X} \{J(\boldsymbol{x}) | \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0},\, \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}\}$$

$$\text{mit} \quad J(\boldsymbol{x}) \;=\; \boldsymbol{c}^\top \boldsymbol{x} \;=\; \sum_{i=1}^{n} c_i x_i$$

$$\boldsymbol{h}(\boldsymbol{x}) \;=\; \boldsymbol{A}_{eq}\boldsymbol{x} - \boldsymbol{b}_{eq} \;=\; \boldsymbol{0}$$

$$\boldsymbol{g}(\boldsymbol{x}) \;=\; \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \;\leq\; \boldsymbol{0}$$

Beschränkungsnotation unter MATLAB®: s. o.

Lineare Optimierung - Beispiel

$$\max_{\boldsymbol{x}\in X}\left\{J(\boldsymbol{x}) = \boldsymbol{c}^\top \boldsymbol{x} \;=\; [1\ 2]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \;=\; x_1 + 2x_2 \,|\, \boldsymbol{g}(\boldsymbol{x}) \le \boldsymbol{0}\right\} \quad \text{bzw.}$$

$$\min_{\boldsymbol{x}\in X}\left\{J(\boldsymbol{x}) = -\,\boldsymbol{c}^\top \boldsymbol{x} \;=\; -\,[1\ 2]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \;=\; -\,(x_1 + 2x_2) \,|\, \boldsymbol{g}(\boldsymbol{x}) \le \boldsymbol{0}\right\}$$

$$\boldsymbol{x} \in X,\ X = \{\boldsymbol{x} : \boldsymbol{g}(\boldsymbol{x}) \le \boldsymbol{0}\} \subset R^n$$

$$J(x) : R^n \to R^1, \boldsymbol{g}(\boldsymbol{x}) : R^2 \to R^3$$

$$g_1(\boldsymbol{x}) \;=\; x_1 + x_2 - 4 \;\le\; 0$$

$$g_2(\boldsymbol{x}) \;=\; -x_1 \;\le\; 0$$

$$g_3(\boldsymbol{x}) \;=\; -x_2 \;\le\; 0\,, \qquad \text{d. h.}$$

$$\boldsymbol{g}(\boldsymbol{x}) \;=\; \begin{bmatrix} 1 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \cdot \boldsymbol{x} \;\le\; \begin{bmatrix} 4 \\ 0 \\ 0 \end{bmatrix}$$

Aufruf: `[x_stern,J_stern]=linprog(-[1 2],[1 1;-1 0;0 -1],[4;0;0])`
bzw.
Datei: `linprog_unb.m`

Quadratische Optimierung

Linear-quadratisches (LQ) Optimierungsproblem (lineare Beschränkungen, quadratische Zielfunktion), s. S. 98:

$$\min_{\boldsymbol{x}}\{J(\boldsymbol{x})|\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}\,, \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}\}$$

$$\begin{aligned}
\text{mit} \quad J(\boldsymbol{x}) &= \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{H}\boldsymbol{x} + \boldsymbol{c}^\top \boldsymbol{x} \\
\boldsymbol{h}(\boldsymbol{x}) &= \boldsymbol{A}_{eq}\boldsymbol{x} - \boldsymbol{b}_{eq} = \boldsymbol{0} \\
\boldsymbol{g}(\boldsymbol{x}) &\leq \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \leq \boldsymbol{0}
\end{aligned}$$

Beschränkungsnotation unter MATLAB®: s. o.

Quadratische Optimierung - Beispiele

1. Bsp. 1: Gleichungsbeschränktes LQ-Problem

$$\min_{\boldsymbol{x}}\{J(\boldsymbol{x})|\boldsymbol{h}(\boldsymbol{x})=\boldsymbol{0}\}$$

$$\begin{aligned}\text{mit}\quad J(\boldsymbol{x}) &= x_1^2+4x_2^2\\ \boldsymbol{h}(\boldsymbol{x}) &= \boldsymbol{A}_{eq}\boldsymbol{x}-\boldsymbol{b}_{eq} = x_1+3x_2-7 = 0\end{aligned}$$

Beschränkungsnotation unter MATLAB®:

$$\boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{b}_{eq}$$

$$\begin{bmatrix}1 & 3\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} = 7$$

```
H=[2 0;0 8];f=[];x0=[0;0];Aeq=[1 3];beq=7;
Aufruf: [x_stern,J_stern]=quadprog(H,f,[],[],Aeq,beq)
```

Datei: `opt_gnb.m`

Quadratische Optimierung - Beispiele

2. Bsp. 2: Lösung einer transzendenten Gleichung

$$x = e^{-x}, x \in [-1, 2] \quad \text{mit } f_1(x) = x \text{ und } f_2(x) = e^{-x}$$

$$\curvearrowright \; f_1(x) = f_2(x) \quad \text{bzw. } f_1(x) - f_2(x) = 0$$

$\curvearrowright$ Minimierung der (quadrierten) Differenz zwischen beiden Funktionen

$$\min_{x \in X} \{ J(x) = \left(x - e^{-x}\right)^2 \mid \boldsymbol{g}(x) \leq \mathbf{0} \}$$

$$-1 \leq x \leq 2, \qquad \text{d.h.}$$

$$\boldsymbol{g}(x) = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \cdot x \leq \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Aufruf: `[x_stern,f_stern]=fminbnd(@(x)(x-exp(-x))^2,-1,2)` bzw.
`[x_stern,f_stern]=fminbnd(@transzendent_graf,-1,2)`
mit graf. Illustr.
`[x_stern,f_stern]=fminbnd(@transzendent_grad,-1,2)`
mit Grad.-ber.
Datei: `transzendent_graf.m`, `transzendent_grad.m`,

Nichtlineare Optimierung

Optimierungsproblem (Zielfunktion und/oder Beschränkungen nichtlinear), s. S. 98:

$$\min_{\boldsymbol{x}}\{J(\boldsymbol{x})|\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}\,, \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}\}$$

Zielfunktion und Beschränkungen in Form von Funktionen formulierbar

Nichtlineare Optimierung - Beispiel

Optimierungsproblem: Zielfunktion und Gleichungsbeschränkung jeweils quadratisch, s. S. 98

$$\min_{\boldsymbol{x}}\{J(\boldsymbol{x})|\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}\,, \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}\}$$

$$\text{mit} \quad J(\boldsymbol{x}) \;=\; x_1^2 + x_2^2 + 2x_1 + 1$$

$$\boldsymbol{h}(\boldsymbol{x}) \;=\; 2x_1 - x_2^2 \;=\; 0$$

Koordinatenbeschränkungen: $\begin{bmatrix} -5 \\ -5 \end{bmatrix} \leq \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 10 \\ 5 \end{bmatrix}$

Aufruf:
```
[x_stern,J_stern]=fmincon(@f52, ...
        x0,[],[],[],[],[-5 -5]',[10 5]',@nlgl52)
```
Dateien: `op52nlgl.m`, `f52.m`, `nlgl52.m`; Grafik: `aufg_52g.pdf`

Grafisches Werkzeug `optimtool`, Optimization App

- Formulierung von (statischen) Optimierungsproblemen/Parameteroptimie- rungsproblemen (lin., quadr. mit linearen Beschränkungen, gemischt-ganz- zahlig lin., nichtlin., Kleinste Fehlerquadrate, Lösen nichtlin. Gleichungen)
- Setzen von Optimierungsoptionen
- Auswahl von Optimierungsverfahren und -algorithmen (Einschränkung)
- Lösen der Optimierungsprobleme (MATLAB® Optimization Toolbox™)

Auswahl weiterer „Apps“ mit Nutzung von Optimierungstechniken

- `cftool` - „curve fitting tool“ (Modellanpassung)
- `MuPad` - Symbolische Berechnungen (Symbolic Math Toolbox™)
- `System Identification` - Systemidentifikation aus Messdaten (experimentelle Prozessanalyse; System Identification Toolbox™)
- `Signal Analysis` - Filterentwurf, Signal- und Spektralanalyse (Signal Processing Toolbox™)
- Simulink® Control Design™ (s. z. B. Online-Dokumentation und Beispiele)
- Simulink® Design Optimization (s. z. B. Tutorial mit Simulink®-Modell `spe_engine_throttle.slx`)

Literatur I

A. Angermann, M. Beuschel, M. Rau, and U. Wohlfahrt.
MATLAB®-Simulink®-Stateflow®.
Oldenbourg, 2009.

O. Beucher.
MATLAB und Simulink. Eine kursorientierte Einführung.
mitp, 2013.

The Mathworks.
Simulink® Control Design™.
https://de.mathworks.com/products/simcontrol.html, 2014.
Letzter Zugriff: 2020-03-24.

The Mathworks.
Matlab® – Control System Toolbox™.
https://de.mathworks.com/products/control.html, 2020.
Letzter Zugriff: 2020-03-24.

The Mathworks.
Matlab® – Mathworks Support Documentation.
https://de.mathworks.com/help/, 2020.
Letzter Zugriff: 2020-03-24.

The Mathworks.
Matlab® – Support Documentation – Get started.
https://de.mathworks.com/help/matlab/getting-started-with-matlab.html, 2020.
Letzter Zugriff: 2020-03-24.

Literatur II

The Mathworks.
Optimization Toolbox™.
https://de.mathworks.com/products/optimization.html, 2020.
Letzter Zugriff: 2020-03-24.

The Mathworks.
Simulink®.
https://de.mathworks.com/products/simulink.html, 2020.
Letzter Zugriff: 2020-03-24.

The Mathworks.
Simulink® Design Optimization™.
https://de.mathworks.com/products/sl-design-optimization.html, 2020.
Letzter Zugriff: 2020-03-24.

W. D. Pietruszka.
MATLAB® und Simulink® in der Ingenieurpraxis. Modellbildung, Berechnung und Simulation.
Springer, 2014.

W. Schweitzer.
MATLAB® kompakt.
Oldenbourg, 2013.