LABORPRAKTIKUM

Praktikumsaufgaben 1, 2 und 3

für

Lehrveranstaltung

Elemente der Modellbildung und Simulationstechnik

Dozent(en):
Dr.-Ing. S. Dyblenko

SS 2020

Hinweise zur Protokollierung

- Es ist ein Praktikumsprotokoll zu jeder Praktikumsaufgabe beizufügen.
- Dabei ist folgendes zu beachten:
  - Die in MATLAB-Skripten und Simulink-Modellen implementierten Lösungen sind i.d.R. gut nachvollziehbar.
  - Die in MATLAB-Skripten und Simulink-Modellen implementierten Lösungen sind i.d.R. gut nachvollziehbar.
  - Die Protokollierung soll daher nur bei Inhalten verwendet werden, die in MATLAB-Skripten und Simulink-Modellen und in ihren Ausführungsergebnissen nicht dargestellt werden. Zum Beispiel:

    die Verifikationsergebnisse mit der Impulsbreite τ_e und -periode in der 2. Praktikumsaufgabe,

    die Bestimmung der maximal zulässigen Simulationsschrittweite und der minimalen Schrittweite,

    die von Ihnen verwendeten Kriterien für die Verifikation sowie die Verifikationsergebnisse,

    die Herleitung von (komplexen) Gleichungen, wie z.B. in der 3. Praktikumsaufgabe (siehe den Hinweis dort).
  - Diese (reduzierten) Protokolle erstellen Sie bitte als pdf-Dateien (auch pdf-Scans von Handnotizen möglich) und geben Sie die Protokolldateien zusammen mit MATLAB-Lösungen ab.

1. PRAKTIKUMSAUFGABE

Blockorientierte Simulation eines nichtlinearen Systems mit Unstetigkeiten

In der 1. Aufgabe erproben Sie einfache, grundlegende Algorithmen der digitalen Simulation dynamischer Systeme. In der 2. Aufgabe ergänzen Sie den implementierten Integrationsalgorithmus "Verbesserte Polygonzugmethode" (VPG) um eine Schrittweitensteuerung. Der Algorithmus soll nach der Verifikation an einem einfachen linearen System zur blockorientierten Simulation eines modularen, nichtlinearen Systems mit Unstetigkeiten in der 3. Aufgabe verwendet werden.

Aufgabe 1: Verbesserte Polygonzugmethode mit Schätzung des LDF

Implementieren Sie ein MATLAB-Rechenmodell für ein einfaches PT_1-Glied mit der Übertragungsfunktion $G_1(s)$, dem das Sprungsignal $u_1(t)$ aufgeprägt wird:

$$G_1(s) = \frac{1}{1 + T_m \cdot s}, \qquad T_m = 10\,\text{sec}$$

$$u_1(t) = \begin{cases} 0 & \text{für } t < 1\ \text{s} \\ 5 & \text{für } t \geq 1\ \text{s} \end{cases}$$

Benutzen Sie die Vorlage in der Anlage Ia (MATLAB-Funktion *system_pt1_vorlage*).

Verwenden Sie zur Integration die "Verbesserte Polygonzugmethode" VPG mit fester Schrittweite und einer Fehlerschätzung für den lokalen Diskretisierungsfehler LDF unter Verwendung der RK3-Koeffizienten (vgl. Vorlesung). Dieser Algorithmus ist nicht in MATLAB vorgefertigt, Sie müssen ihn selbst programmieren. Ihnen steht eine Vorlage zur Verfügung – siehe Anlage Ib.

Verifizieren Sie Ihre Implementation und erklären Sie das Verhalten der Diskretisierungsfehler an der Sprungstelle des Eingangssignals.

Berechnen Sie zur Simulationsverifikation den Sollverlauf der Ausgangsgröße analytisch! Protokollieren Sie grafisch (auf dem Bildschirm) die Eingangs- und die Ausgangsgröße, die Differenz zum Sollverlauf der Ausgangsgröße und den lokalen Diskretisierungsfehler im Simulationsintervall t = 0 ... 300 s.

Aufgabe 2: Automatische Schrittweitensteuerung

Ergänzen Sie den in MATLAB programmierten VPG-Algorithmus aus der Aufgabe 1 um eine Schrittweitensteuerung gemäß Vorlesungs-Script **MODSIM-3, Abs. 3.4 Schrittweitensteuerung**. Verifizieren Sie den neuen Algorithmus (qualitativ und quantitativ) zunächst für das PT_1-Glied.

Berechnen Sie zur Simulationsverifikation zusätzlich den Verlauf der Schrittweite im Simulationsintervall. Erklären Sie das Verhalten der Schrittweite in der Nähe der Sprungstelle des Eingangssignals!

Hinweise:

- Beachten Sie, dass sich die Schrittweite h nur innerhalb eines Intervalls $h_{\min} \le h \le h_{\max}$ ändern darf! Überlegen Sie, durch welche Randbedingungen $h_{\min}$ und $h_{\max}$ bestimmt werden und legen Sie selbst sinnvolle Werte fest!
- Die Norm eines Vektors ist der maximale Betrag der Vektorelemente; in MATLAB: norm_von_x = **max (abs(**x**))**.
- Die Toleranz ε_{LDF} sollte in der Größenordnung 10^{-5} bis 10^{-6} liegen.

Aufgabe 3: Pseudorate-Modulator

Technischer Hintergrund

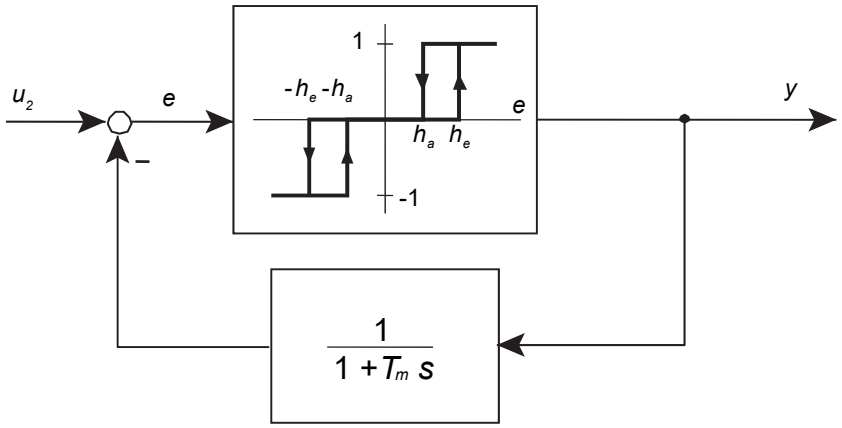
Zur *Steuerung von Stoffströmen* werden häufig *schaltende (unstetige) Ventile* verwendet, die nur 2 diskrete Zustände ermöglichen: Ventil_offen, Ventil_geschlossen. Die Realisierung von variablen Stoffströmen (z. B. proportionales Verhalten als Stellglied in einem Regelkreis) kann dann nur mittels pulsbreiten- und frequenzmodulierter Signale erreicht werden, d. h. der *Mittelwert* des getakteten Stoffstromes ist dann proportional dem gewünschten kontinuierlichen Stellkommando.
Zur *Bahn- und Lageregelung von Satelliten* sind auch noch heute aus technologischen Gründen solche schaltenden Ventile in Verbindung mit Düsenstellsystemen unverzichtbar wenn es darum geht, äußere Kräfte aufzubringen (Bahnmanöver) bzw. „große“ äußere Momente zu erzeugen (Großwinkelmanöver, Kompensation von großen Störmomenten).
Der hier vorgestellte **"Pseudorate-Modulator"** ist als analoge Komponente z. B. in allen Telekommunikationssatelliten der Serien INTELSAT V, EUTELSAT-II, DFS-KOPERNIKUS (Deutsche Telekom) implementiert. Im speziellen realisiert dieser Modulator folgenden Funktionen:

(i) *Tote Zone:* (a) Unempfindlichkeit gegen Messrauschen → Vermeidung von unnötigen Düsenaktivitäten → Minimierung Treibstoffverbrauch
(b) Toleranzband für Lageregelung → für Regelfehler < Tote Zone erfolgt keine Lagekorrektur → Minimierung Treibstoffverbrauch

(ii) *Phasenvorhalt:* PD-Verhalten → Stabilisierung der instabilen Regelstrecke ($1/s^2$) bzw. Kompensation negativer Phasenanteile von Sensor- bzw.Ventildynamik und Rechentotzeiten

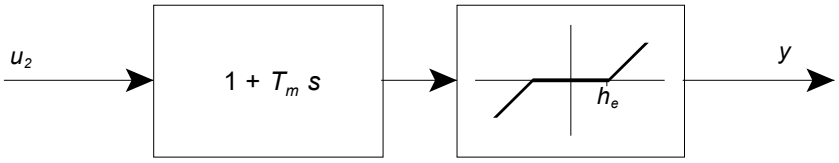
(iii) *minimale Einschaltzeit:* aus Effizienzgründen darf bei einem Schaltvorgang eine minimale Einschaltzeit nicht unterschritten werden

Ein solcher Modulator hat folgendes Blockschaltbild:



$h_a = 0{,}065$
$h_e = 0{,}085$
$T_m = 10$ s

Ersatzschaltbild (für lineare Stabilitätsanalyse etc.; nur zur Information, soll hier **nicht** programmiert werden!):



Für die Simulation dieses modularen Systems programmieren Sie in Analogie zum Vorlesungsbeispiel im Script **MODSIM-4, Abs. 4.8. Programmbeispiel** die entsprechenden Unterprogramme für die Systemtopologie und die drei Module Subtraktionsstelle, Hysterese und PT_1-Glied. Binden Sie das System in Ihren VPG-Algorithmus mit Schrittweitensteuerung ein und simulieren Sie im Intervall $t = 0$ ... 20 s. Protokollieren Sie grafisch (auf dem Bildschirm) den Verlauf aller Blockausgänge, des (geschätzten) Lokalen Diskretisierungsfehlers und der Schrittweite bei Einspeisung einer konstanten Eingangsgröße. Führen Sie dazu drei Experimente mit den folgenden Werten für die Eingangsgröße aus:

$$u_{21} = \ \ 0{,}17$$
$$u_{22} = -0{,}25$$
$$u_{23} = \ \ 0{,}49$$

Verifizieren Sie die sich nach der Einschwingzeit einstellende Impulsbreite τ_e und -periode τ_P mit Hilfe der folgenden Gleichungen:

$$\tau_e = -T_m \cdot \ln\left(1 - \frac{h_e - h_a}{1 + h_e - |u_2|}\right)$$

$$\tau_P = T_m \cdot \left[\ln\frac{1 - h_a/|u_2|}{1 - h_e/|u_2|} - \ln\left(1 - \frac{h_e - h_a}{1 + h_e - |u_2|}\right)\right]$$

Vergleichen Sie die Simulationsergebnisse mit denen bei Verwendung eines VPG-Algorithmus mit **fester** Schrittweite!

Hinweise:

- Testen Sie den Hysteresemodul separat in einer selbst gewählten Testumgebung!
- Beachten Sie, dass der Hysteresemodul ein „Gedächtnis“ hat! Falls bei der Schrittweitensteuerung ein Integrationsschritt mit kleinerer Schrittweite wiederholt werden muss, muss auch dieses "Gedächtnis" auf seinen Wert **vor** dem „fehlerhaften“ Integrationsschritt rückgesetzt werden (z. B. globale Variable verwenden!).
- Der natürliche Logarithmus wird in MATLAB mit der Standardfunktion **log(x)** berechnet.

Anlage Ia

Vorlage für das MATLAB-Unterprogramm (MATLAB-Funktion) mit der Systemfunktion
Kopieren in Textdatei `system_pt1.m`, den Funktionsnamen auf `system_pt1` ändern, den Code ergänzen.

```
% MODSIM Laborpraktikum, 1. Aufgabe
%
% Dr.-Ing. Th. Range, Dr.-Ing. S. Dyblenko
%
%   zu ergänzende Codezeilen sind mit ">>> ergänzen ...." gekennzeichnet

% Berechnung des Systems "PT1-Glied"
%
% (Hinweis: Die Struktur des Programms erlaubt eine Einbindung in
%  SIMULINK als sog. S-Function; dies ist jedoch im Rahmen des
%  Praktikums nicht vorgesehen!)

function [sys, x0] = system_pt1_vorlage ( t, x, u, flag )

% Eingabe-Parameter:
%    t     -  Zeit
%    x     -  x(t) Zustandsvektor zum Zeitpunkt t
%    u     -  u(t) Vektor der Eingangssignale zum Zeitpnukt t
%    flag  -  Steuerparameter, legt fest, welche Ausgabe
%             gefordert wird:

Tm = 10;                    % Zeitkonstante des PT1-Gliedes

if flag == 0                % Ausgabe der Anfangswerte für den
                            % Zustand auf Vektor x0
       x0 = %>>> ergänzen ....

       sys = [1,0,1,1,0,0];% diese Zeile ist nur für Simulink nötig,
                                % sie gilt so NUR in diesem Beispiel!

elseif abs(flag) == 1       % Ausgabe der Ableitungen von x auf
                            % Vektor sys = x' = f(x(t),u(t),t)
       sys = %>>> ergänzen ....

elseif flag == 3            % Ausgabe der Ausgangswerte des
                            % Systems auf Vektor sys:
                            %     sys = y = g(x(t),u(t),t)
       sys = %>>> ergänzen ....

else                        % bei anderen Flagwerten nichts ausgeben
       sys = [];
end
```

Anlage Ib

Vorlage für das MATLAB-Hauptprogramm mit VPG Algorithmus und Fehlerschätzung

Kopieren in die Textdatei `main_a1.m`, den Code ergänzen und anpassen.

```
% MODSIM Laborpraktikum, 1. Aufgabe
% Prof. K. Janschek, Dr.-Ing. Th. Range, Dr.-Ing. S. Dyblenko
%
% main_a1.m  -  Realisierung der VPG-Methode mit Fehlerschätzung
%               für PT1-Glied

%   zu ergänzende Codezeilen sind mit ">>> ergänzen ...." und ..."gekennzeichnet
clear all      % Lösche Arbeitsspeicher

Tm = 10;       % Konstante des PT1, [s]

h  = 0.1;      % Schrittweite, (s)

t0 = 0;        % Integrationsbeginn, [s]
tf = 300;      % Integrationsende, [s]

t  = [];       % Zeitwerte für Plot [s]
d  = [];       % Fehler-Schätzwerte
u  = [];       % Stellwerte u(t)
y  = [];       % Ausgangswerte y(t)
ys = [];       % Soll-Ausgangswerte y_soll(t)

% Initialisierung

[dum,x(1)] = system_pt1([],[],[],0);
d(1) = 0;

% Integration nach VPG-Methode
ti = t0;
i  = 1;

while ti <= tf
   % Berechnung des Soll-Ausgangswertes
   ys(i) = %>>> ergänzen ....

   % Berechnung des Stellwertes
   u(i)  = %>>> ergänzen ....

   % Berechnung des Ausgangswertes
   y(i)  = system_pt1( ... , ... , ... , 3); %die Parameter einsetzen

   % Berechnung der Koeffizienten für VPG-Methode
   k1    = system_pt1( ... , ... , ... , 1); %die Parameter einsetzen
   k2    = system_pt1( ... , ... , ... , 1); %die Parameter einsetzen
   k3    = system_pt1( ... , ... , ... , 1); %die Parameter einsetzen

   % Wichtiger Hinweis: Die Parameter bei den Aufrufen von system_pt1(...)
   % müssen unter Beachtung von jeweiligen Zeitpunkten bestimmt werden!

   % Berechnung des Zustands-Schätzwertes x(ti+h)
   x(i+1) = %>>> ergänzen ....

   % Berechnung der LDF Fehlerabschätzung d(ti+h)
   d(i+1) = %>>> ergänzen ....

   t(i) = ti;      % Zeitwert für Plot speichern
   ti   = ti + h;  % Zeitvariable um einen Schritt erhöhen
   i    = i + 1;   % Index inkrementieren
end

d      = d(1:end-1);
result = [t;d];

% Anzeige der Ergebnisse
figure(1);
subplot(2,1,1); plot(t,u); title('Eingang PT1-Glied');zoom on;grid on;
subplot(2,1,2); plot(t,y); title('Ausgang PT1-Glied');zoom on;grid on;
xlabel('Zeit, s');

figure(2);
subplot(2,1,1); plot(t,y-ys,'.-'); title('GDF berechnet');zoom on;grid on;
tit=sprintf('LDF geschätzt: max. Betrag = %g',max(abs(d)));
subplot(2,1,2); plot(t,d,'.-'); title(tit);zoom on;grid on;
xlabel('Zeit, s');
```

2. PRAKTIKUMSAUFGABE

Blockorientierte Simulation eines zeitkontinuierlich-zeitdiskreten nichtlinearen Systems – Anwendung der Transitionsmatrix

In der 2. Aufgabe soll das blockorientierte Simulationsprogramm MATLAB/Simulink zur Modellierung eines nichtlinearen hybriden (gemischt kontinuierlichen und zeitdiskreten) Systems genutzt werden. Unter Verwendung der Transitionsmatrix wird zum Reglerentwurf und zur Beschleunigung der Simulation ein zeitdiskretes lineares Modell implementiert.

Praktischer Hintergrund

Servohydraulische Systeme werden bevorzugt zur mechanischen Materialprüfung eingesetzt. Dabei werden die Prüflinge vorgegebenen mechanischen Belastungen (Kräfte, Drehmomente) ausgesetzt. Diese Belastungsprofile können statisch oder dynamisch sein. Hauptaufgabe des servohydraulischen Prüfsystems ist also die möglichst exakte Realisierung von dynamischen Kraftprofilen, d. h. eine Sollwertregelung über einen möglichst großen Frequenzbereich.

Die wesentlichen Komponenten eines servohydraulischen Prüfsystems bestehen aus:

- Hydraulikzylinder ... Krafterzeugung über hydraulische Prinzipien (Öl)
- Servoventil ... Steuerung des Ölflusses mittels kleiner Leistung
- Prüfling ... mechanisches System, das mit Prüfkräften beaufschlagt werden soll
- Regel- und Steuereinheit ... Steuerung des geschlossenen Systems
- Messdatenverarbeitung ... Dokumentation des Prüfvorganges

Der Prüfling ist damit Teil der Regelstrecke, die durch Masse-Feder-Systeme charakterisiert werden kann (Ölkompressibiltät, Federsteifigkeit des Prüflings; dies führt bei Materialverformungen zu Parameteränderungen der Regelstrecke).

Eine *konkrete Anwendung*, die dem vorliegenden Übungsbeispiel zugrunde liegt, behandelt die *Materialprüfung von Flugzeugflügeln*. Dabei werden operationelle oder kritische Belastungsprofile mittels einer verteilten Krafteinleitung in die Flugzeugflügel simuliert. Bei Verkehrsflugzeugen benötigt man dazu an die 100 oder mehr Hydraulikzylinder. Wegen der großen Anzahl von Hydraulikzylindern mit den dazu notwendigen Regel- und Steuereinheiten ist aus kommerziellen Gründen eine kostengünstige Realisierung jedes einzelnen Regelkreises unabdingbar. Im vorliegenden Fall soll daher ein bereits vorhandenes digitales Kraftmesssystem (d. h. kein Entwicklungsaufwand nötig!) verwendet werden, das allerdings eine nicht vernachlässigbare Messtotzeit besitzt. Um die Inbetriebnahme der 100 Regelkreise einfach zu halten, soll als Reglerkonfiguration ein einfacher I-Regler (nur ein Parameter!) verwendet werden.

Im Rahmen der Projektierung einer solchen Anlage besteht eine der wichtigen Aufgaben eines Systemingenieurs darin, rechnergestützt den Nachweis für die Machbarkeit der konzipierten Systemlösung zu führen. Dieser Nachweis soll wegen der Nichtlinearitäten im System mittels Simulation untermauert werden.

Anmerkung: Die vorliegenden Modelle sind zwar vereinfacht, reichen jedoch für eine Machbarkeitsaussage aus. Um genauere Aussagen über die Regelgüte (Klirrfaktoren etc.) zu erreichen, müsste man den Prüfling etwas genauer modellieren.

Ein Hydropulszylinder aus der oben beschriebenen Anwendung hat folgenden prinzipiellen Aufbau (Bild 1):

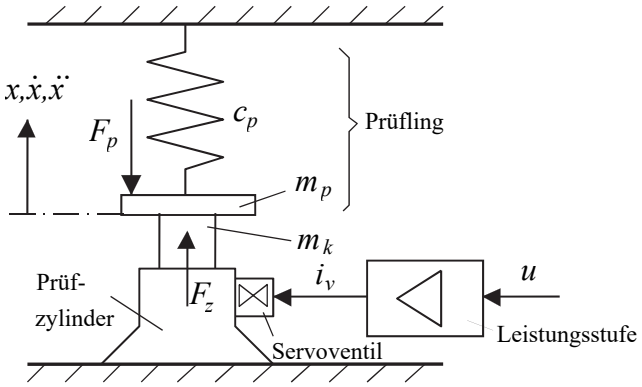


Bild 1: Hydropulszylinder

Unter Vernachlässigung der Dynamik des Servoventils hat der Hydropulszylinder den in Bild 2 gezeigten Signalflussplan.

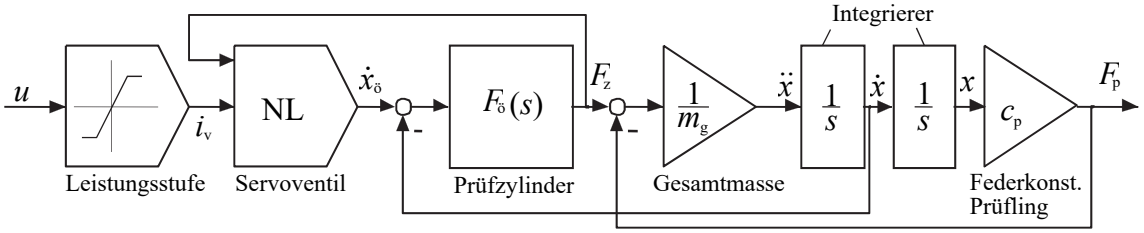


Bild 2: Signalflussplan des Hydropulszylinders (Angaben zu den Blöcken ☞ *Tabelle*)

Durch Linearisierung um $u = 0$ und $\underline{x} = \underline{0}$ erhält man eine Übertragungsfunktion von

$$G_{\mathrm{F}}(s) = \frac{K_{\mathrm{F}}}{1 + a_1 s + a_2 s^2 + a_3 s^3} \text{ mit } K_{\mathrm{F}} = K_{\mathrm{L}} K_{\mathrm{sv}} b_{\mathrm{l}},\ a_1 = \frac{b_{\mathrm{l}}}{c_{\mathrm{p}}} + \frac{b_{\mathrm{l}}}{c_{\mathrm{ö}}},\ a_2 = \frac{m_{\mathrm{g}}}{c_{\mathrm{p}}},\ a_3 = \frac{m_{\mathrm{g}} b_{\mathrm{l}}}{c_{\mathrm{p}} c_{\mathrm{ö}}}$$

Die Übertragungsfunktion hat Pole bei $\lambda_1 = -10{,}27\ \mathrm{s}^{-1}$ und $\lambda_{2/3} = (-2{,}5 \pm 644{,}13\ \mathrm{j})\mathrm{s}^{-1}$.

Leistungsstufe (statisch nichtlinear)	Servoventil (statisch nichtlinear)	Prüfzylinder	Prüfling
$i_{\mathrm{v}} = \begin{cases} i_{\mathrm{vmax}} & u > 1\ \mathrm{V} \\ K_{\mathrm{L}} \cdot u & -1\ \mathrm{V} \le u \le 1\ \mathrm{V} \\ -i_{\mathrm{vmax}} & u < -1\ \mathrm{V} \end{cases}$ $K_{\mathrm{L}} = 1\ \mathrm{A/V}$ $i_{\mathrm{vmax}} = 1\ \mathrm{A}$	$\dot{x}_{\mathrm{ö}} = \begin{cases} 0 & A \le 0 \\ i_{\mathrm{v}} \cdot K_{\mathrm{sv}} \cdot \sqrt{A} & A > 0 \end{cases}$ $A = 1 - \frac{F_{\mathrm{z}}}{F_{\mathrm{N}}} \cdot \mathrm{sign}(i_{\mathrm{v}})$ $K_{\mathrm{sv}} = 0{,}796\ \mathrm{m/As}$ $F_{\mathrm{N}} = 63000\ \mathrm{N}$	$F_{\mathrm{ö}}(s) = \frac{b_{\mathrm{l}}}{1 + \frac{b_{\mathrm{l}}}{c_{\mathrm{ö}}} s}$ $b_{\mathrm{l}} = 2{,}39 \cdot 10^6\ \mathrm{Ns/m}$ $c_{\mathrm{ö}} = 36{,}5 \cdot 10^6\ \mathrm{N/m}$ $m_{\mathrm{k}} = 8{,}7\ \mathrm{kg}$	$m_{\mathrm{g}} = m_{\mathrm{k}} + m_{\mathrm{p}}$ $m_{\mathrm{p}} = 260\ \mathrm{kg}$ $c_{\mathrm{p}} = 75 \cdot 10^6\ \mathrm{N/m}$

Aufgabe 1: Implementierung des nichtlinearen kontinuierlichen Streckenmodells

Implementieren Sie den Signalflussplan (Bild 2) mit Hilfe des blockorientierten Simulationsprogramms MATLAB/Simulink. Verbinden Sie den Eingang (u) und den Ausgang (F_p) mit Simulink-In- bzw. -Out-Ports und speichern Sie das Streckenmodell in einer mdl-Datei. Die Initialisierung bzw. die Berechnung von Variablen für die Modell-Parameter implementieren Sie in einem MATLAB-Script (m-Datei).

Beachte: MATLAB-Skripte sollen bei allen Aufgaben der 2. Praktikumsaufgabe zu modelbezogenen Berechnungen eingesetzt werden!

Hinweis: Testen Sie Ihre Implementation der statischen Nichtlinearität des Servoventils in einer separaten Testumgebung!

Zur Verifikation erzeugen Sie aus Ihrer Modell-Implementierung mit Hilfe der MATLAB-Funktionen **linmod** (Linearisierung, Arbeitspunkt: $\underline{0}$) und **ss2tf** (Transformation) die zugehörige (lineare) Übertragungsfunktion und vergleichen Sie deren Parameter mit K_F, a_1, a_2 und a_3!

Verbinden Sie den Eingang und den Ausgang des nichtlinearen kontinuierlichen Streckenmodells jeweils mit Simulink-Block Step und Simulink-Block Scope und speichern Sie das Streckenmodell in einer neuen mdl-Datei (die Port-Blöcke müssen Sie entfernen). Dieses Simulink-Model werden Sie im letzten Schritt der Aufgabe 2 brauchen.

Aufgabe 2: Implementierung des linearen zeitdiskreten Streckenmodells

Zur Beschleunigung der Simulation und als Basis für den Reglerentwurf soll zusätzlich ein lineares zeitdiskretes Modell geschaffen werden. Berechnen Sie dazu mit Hilfe der MATLAB-Funktionen **linmod** (Linearisierung, Arbeitspunkt: $\underline{0}$) und **expm** (Matrix-Exponentialfunktion) die Transitionsmatrix $\underline{\Phi}$ und die diskrete Eingangsmatrix $\underline{H}$ (hier: Vektor) für die Abtastzeit $T_a = 15$ ms!

Implementieren Sie das lineare zeitdiskrete Streckenmodell mit MATLAB/Simulink analog zu Aufgabe 1 als eine weitere mdl-Datei. Verifizieren Sie es durch den Vergleich einiger Sprungantworten mit denen des nichtlinearen kontinuierlichen Modells!

Aufgabe 3: Reglerauslegung und Simulation mit dem linearen Modell

Die Prüfkraft F_P soll durch einen diskontinuierlich arbeitenden I-Regler mit Halteglied 0. Ordnung (z. B. Rechner) mit der Abtastzeit $T_a = 15$ ms geregelt werden. F_N wird durch ein Messglied mit dem Übertragungsfaktor $K_M = 1/63000$ V/N gemessen, das den Messwert jedoch erst nach Ablauf der Abtastzeit T_a bereitstellt (siehe Bild 3).

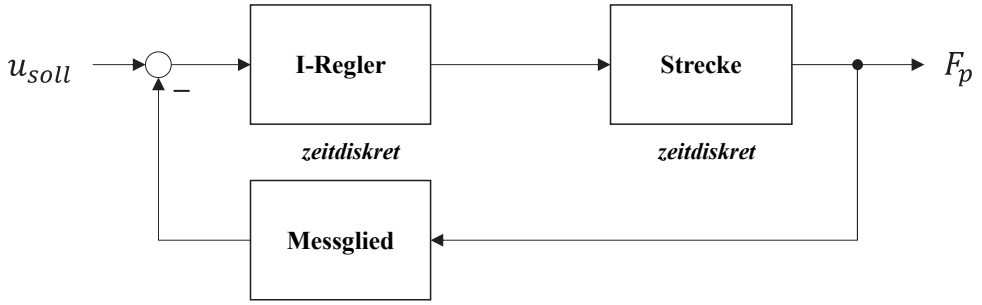


Bild 3: Regelkreis für Prüfkraft-Regelung

Dimensionieren Sie die Reglerverstärkung K_I so, dass ein Phasenrand von ca. 50° eingestellt wird! Benutzen Sie für diesen Entwurfsschritt das lineare zeitdiskrete Streckenmodell aus der Aufgabe 2.

Hinweis: Implementieren Sie dazu in Simulink ein Modell für den aufgeschnittenen Regelkreis; setzen Sie dabei zunächst $K_I = 1\ s^{-1}$. Verbinden Sie die „Enden" der Trennstelle mit einem In- bzw. Out-Port und speichern Sie das Modell in einer mdl-Datei. Berechnen Sie nun mit Hilfe der MATLAB-Funktionen **dlinmod** (Linearisierung, Arbeitspunkt: 0) und **dbode** das BODE-Diagramm. Aus diesem können Sie K_I für den gewünschten Phasenrand ablesen.

Erzeugen Sie nun mit Simulink ein Modell für den geschlossenen Regelkreis mit linearem zeitdiskreten Streckenmodell und protokollieren Sie (grafisch auf dem Bildschirm) die Sprungantwort für einen Führungsgrößensprung von $u_{soll} = 0$ auf $u_{soll} = 0{,}5$ V! Bei richtig dimensioniertem Regler stellt sich ein leichtes Überschwingen ein (entsprechend einem Phasenrand von 50°).

Bestimmen Sie experimentell die kritische Reglerverstärkung $K_{I,krit}$!

Aufgabe 4: Experimentation mit dem nichtlinearen kontinuierlichen Modell

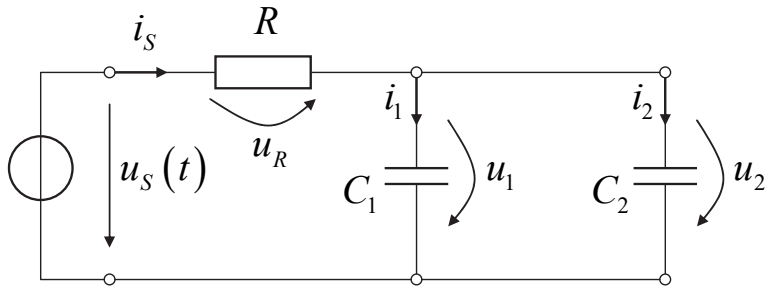
Implementieren Sie ein Simulink-Modell für den geschlossenen Regelkreis mit dem nichtlinearen kontinuierlichen Streckenmodell. Stellen Sie den Regler auf die in Aufgabe 3 ermittelte Verstärkung ein und nehmen Sie Sprungantworten für Führungsgrößensprünge von 0,1 V, 0,5 V und 0,8 V auf. Vergleichen Sie mit der Sprungantwort des linearen Modells aus der Aufgabe 3 und bewerten Sie die Ergebnisse!

Hinweis: Wählen Sie vor Simulationsbeginn die für Ihr Modell zutreffenden Simulationsparameter, insbesondere die maximale Schrittweite!

Ermitteln Sie experimentell die kritischen Reglerverstärkungen für die drei angegebenen Sprunghöhen!

3. PRAKTIKUMSAUFGABE
Numerische Integration von DAE-Systemen

Vorgegeben sei das im Bild dargestellte RC-Netzwerk.



Die Modellkonstanten sind $R = 100$ Ohm, $C_1 = 0{,}01$ F, $C_2 = 0{,}02$ F

Der Verlauf der Ausgangsspannung u_2 soll mit Hilfe der numerischen Integration berechnet werden.

Aufgabe 1: Semi-explizites DAE-System

Geben Sie ein mathematisches Modell des dargestellten RC-Netzwerkes in Standardform eines semi-expliziten DAE-Systems an, in dem alle gezeigten Netzwerkgrößen explizit auftreten.

Stellen Sie das Modell mit folgenden Standardbezeichnungen dar:

$$x_1 = u_1, \quad x_2 = u_2, \quad z_1 = i_1, \quad z_2 = i_2, \quad z_3 = u_R, \quad z_4 = i_S, \quad u = u_S$$

Bestimmen Sie den differenziellen Index des Modells mithilfe der Indexreduktion.

Aufgabe 2: Numerische Integration mit impliziten Verfahren

Stellen Sie ein geeignetes Gleichungssystem für die numerische Integration mit der Trapez-Methode und dem Newton-Raphson-Verfahren dar. Berechnen Sie die Jacobi-Matrix des Gleichungssystems analytisch.

Implementieren Sie das semi-explizite DAE-Modell, das Gleichungssystem für die numerische Integration und die Jacobi-Matrix in MATLAB. Programmieren Sie den Integrationsalgorithmus mit der Trapez-Methode und dem Newton-Raphson-Verfahren in MATLAB. Verwenden Sie dabei eine feste Schrittweite und die analytische Jacobi-Matrix. Implementieren Sie das DAE-Modell, das Gleichungssystem, die Jacobi-Matrix und den Integrationsalgorithmus in unterschiedlichen Funktionen.

Verwenden Sie ausschließlich Variablen vom 32-bit-Gleitkommatyp. Damit kann die numerische Genauigkeit bei kleinen Schrittweiten mit einem erheblich geringeren Rechenaufwand untersucht werden. Zur Erstellung von den Variablen nutzen Sie die Matlab-Funktion **`single(Z)`**, wo **`Z`** die gewünschte Zahl bzw. die Standard-Matlab-Variable ist.

Verifizieren Sie Ihre Implementierung im folgenden Experimentrahmen:

- Simulationsintervall $t = 0 \,..\, 30$ s
- Eingangssprung $u_S(t) = \begin{cases} 10, & t \geq t_0 \\ 0, & t < t_0 \end{cases}$, mit $t_0 = 0$
- die Kondensatoren sind zu $t = 0s$ vollständig entladen, d. h. $u_1(0) = u_2(0) = 0$.

Bestimmen Sie geeignete Anfangswerte der algebraischen Variablen!

Berechnen Sie zur Simulationsverifikation den Sollverlauf der Ausgangsspannung $u_2(t)$ analytisch.

Protokollieren Sie grafisch (auf dem Bildschirm) die folgenden Systemgrößen:

- Eingangsgröße $u_S(t)$ und Sollverlauf der Ausgangsspannung $u_2(t)$
- simulierte Ausgangsgrößen $\hat{u}_1(t)$ und $\hat{u}_2(t)$
- Differenz zum Sollverlauf der Ausgangsspannung $u_2(t)$

Kontrollieren Sie das Konvergenzverhalten des Newton-Raphson-Verfahrens. Bei korrekter Implementierung sollte das Verfahren mit einer kleinen Schranke z.B. $\varepsilon = 1 \cdot 10^{-5}$ höchstens nach zwei Iterationen konvergieren.

Nennen und überprüfen Sie ein geeignetes Kriterium für eine hohe numerische Genauigkeit des Simulationsexperimentes bei kleinen Schrittweiten, wenn ein unterlagertes Newton-Raphson-Verfahren verwendet wird.

Bestimmen Sie die erreichbare numerische Genauigkeit bei sehr kleinen Schrittweiten *experimentell* (z.B. bei $h = 1 \cdot 10^{-5}$ s).

Allgemeine Anforderung

Die Herleitung aller Gleichungen *unbedingt* in einem Protokoll festhalten (siehe Hinweise zur Protokollierung).