Systemidentifikation - Grundlegende Methoden und ihre Implementierung

VO+PR 101.364/101.369 Modellbildung und Simulation

Nicole Geissberger 0826904, Manuel Eder 0827050

6. Januar 2012

1 Zusammenfassung

Diese Arbeit dokumentiert das Projekt zum Thema „Systemidentifikation“ für die Lehrveranstaltung “Modellbildung und Simulation“. Es wurden folgende Verfahren betrachtet:

- Least Squares Identifikation
- Rekursive Least Squares Identifikation
- Kalman-Filter

Die Aufgabenstellung war, diese Verfahren in 2 Simulationsumgebungen zu implementieren - einerseits in Matlab, andererseits in der in Matlab enthaltenen Simulink-Umgebung. Da die Implementierung des Least Squares Verfahrens in Simulink hauptsächlich durch Matlab Einbindung realisierbar wäre, wurde die Arbeit bei diesem Verfahren auf Matlab beschränkt, und die rekursive Least Squares Identifikation in Simulink implementiert, da dies anschaulicher ist. Das Kalman-Filter wurde in beiden Sprachen gleichermaßen implementiert.

2 Die Problemstellung

Im folgenden Abschnitt werden wir näher erläutern, welche Probleme durch die implementierten Verfahren gelöst werden sollen.

2.1 Least Squares Identifikation

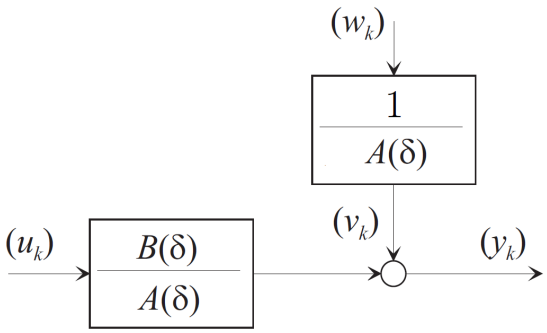
Die Least Squares Methode eignet sich, um beim autoregressiven Modell mit deterministischer (bekannter) Eingangsgröße u_k (ARX, autoregressive with exogenous input) die Modellparameter zu schätzen. Beim ARX-Modell hängt der Ausgang y_k eines Systems vom Eingang u_k ab über

$$y_k = \frac{B(\delta)}{A(\delta)} u_k$$

bzw. im gestörten Fall mit stochastischer Störung w_k

$$y_k = \frac{B(\delta)}{A(\delta)} u_k + \frac{1}{A(\delta)} w_k \ .$$

Dabei ist δ der Shift-Operator mit $\delta^{-1} u_k = u_{k-1}$, und $A(\delta) = \sum_{i=0}^{n} a_i \delta^{-i}$ mit $a_0 = 1$, $B(\delta) = \sum_{i=0}^{m} b_i \delta^{-i}$. w_k ist das stochastische Störsignal.



2.2 Kalman-Filter

Das Kalman-Filter löst ein anderes Problem als die im vorigen Abschnitt besprochene Anwendung der Least Squares Methode. Wir haben es wieder mit einem (schrittweise) linearen System zu tun, diesmal geschrieben in der Form

$$\begin{aligned} x_{k+1} &= \Phi_k x_k + \Gamma_k u_k + G_k w_k \\ y_k &= C_k x_k + D_k u_k + H_k w_k + v_k \ , \end{aligned} \tag{1}$$

mit dem Output y_k, dem Input u_k, dem internen Zustand x_k, sowie den stochastischen Störgrößen w_k und v_k. Die systembeschreibenden Matrizen Φ_k , Γ_k, G_k, C_k, D_k, H_k werden in vielen Anwendungsfällen, und so auch in unseren Testbeispielen konstante Funktionen von k sein. Das Kalman-Filter ist nun ein Algorithmus, der bei gegebenen systembeschreibenden Matrizen, Information über den Startwert x_0 in Form von stochastischen Kenngrößen (d.h. Erwartungswert $\mathbb{E}(x_0)$ sowie die Kovarianzmatrix Σ_{x_0}), sowie bei gegebenem Input u und Output y den internen Zustand x_k schätzt (mit internem Zustand seien hier Erwartungswert $\mathbb{E}(x_k)$ und Kovarianzmatrix Σ_{x_k} gemeint), also nur Statistiken zweiter Ordnung des Zustandes x_k.

3 Theorie

3.1 Least Squares Identifikation

Schreiben wir das ARX-System, das wir identifizieren wollen, explizit an, so errechnet sich im ungestörten Fall zum k-ten Abtastzeitpunkt der Wert y_k aus

$$y_k = -a_1 y_{k-1} - \cdots - a_n y_{k-n} + b_0 u_k + b_1 u_{k-1} + \cdots + b_m u_{k-m}$$

bzw. in Vektorschreibweise

$$
\begin{aligned}
y_k &= (-y_{k-1}, -y_{k-2}, \ldots, -y_{k-n}, u_k, \ldots, u_{k-m})(a_1, \ldots, a_n, b_0, \ldots, b_m)^T \\
y_k &= s_k p \text{ mit} \\
s_k &:= (-y_{k-1}, -y_{k-2}, \ldots, -y_{k-n}, u_k, \ldots, u_{k-m}) \\
&\quad \text{und dem Parametervektor} \\
p^T &:= (a_1, \ldots, a_n, b_0, \ldots, b_m)^T \ .
\end{aligned}
$$

Fassen wir die Zeilen s_k zu einer Matrix S und die Skalare y_k zu einem Vektor y zusammen, so ergibt das

$$y = Sp$$

mit

$$y = \begin{pmatrix} y_{\max\{n,m\}} \\ \vdots \\ y_N \end{pmatrix}$$

und

$$
\begin{aligned}
S &= \begin{pmatrix} s_{max\{n,m\}} \\ \vdots \\ s_N \end{pmatrix} \\
&= \begin{pmatrix} -y_{\max\{n,m\}-1} & \cdots & -y_{\max\{n,m\}-n} & u_{\max\{n,m\}} & \cdots & u_{\max\{n,m\}-m} \\ \vdots & \ddots & & \vdots & \ddots & \\ -y_{N-1} & \cdots & -y_{N-n} & u_N & \cdots & u_{N-m} \end{pmatrix} \ .
\end{aligned}
$$

Dabei ist N die Anzahl der Messungen und $y_1 \ldots y_{\max\{n,m\}-1}$ kommen auf der linken Seite nicht vor, da die rechte Seite für $k < \max\{n, m\}$ nicht definiert ist.
Der Vektor p wird nun geschätzt. Ziel ist es, die euklidische Norm des Outputfehlervektors $e := y - S\hat{p}$ bei Ersetzen von p durch $\hat{p}$ zu minimieren, d.h.:

$$||e||_2 = ||y - S\hat{p}||_2$$

soll minimal werden. Bekannterweise gilt in Hilberträumen: Sind ein Vektor z_0 und ein (abgeschlossener) Unterraum U vorgegeben, so gibt es immer einen Vektor $z \in U$, für den der Ausdruck $||z - z_0||$ minimal wird. Für diesen Vektor z gilt, dass der Vektor $z - z_0$ orthogonal auf U steht. Bezeichne $< \cdot, \cdot >_2$ das Skalarprodukt, das die euklidische Norm induziert, dann wissen wir also, dass

$$
\begin{aligned}
< S, y - S\hat{p} >_2 &= 0 \text{ (spaltenweise für } S = (\vec{S_i}), i = 1, \ldots, N) \\
&\quad \text{bzw.} \\
S^T(y - S\hat{p}) &= 0 \\
S^T y - S^T S\hat{p} &= 0 \\
S^T S\hat{p} &= S^T y \\
\hat{p} &= (S^T S)^{-1} S^T y \ .
\end{aligned}
$$

Dieses Ergebnis lässt sich leicht verallgemeinern auf den Fall, dass wir eine andere von einem Skalarprodukt induzierte Fehlernorm minimieren wollen. Wird unsere Norm vom Skalarprodukt $< x, y >_Q = x^T Q y$ für eine quadratische Matrix Q induziert, so gilt analog zu obiger Rechnung

$$\hat{p} = (S^T Q S)^{-1} S^T Q y \ . \tag{2}$$

3.1.1 Der Fall mit stochastischer Störung

Haben wir nun ein ARX-System mit stochastischer Störung v_k, so verlaufen die Überlegungen anfangs analog.

Der Outputwert zum k-ten Abtastzeitpunkt berechnet sich aus

$$y_k = -a_1 y_{k-1} - \cdots - a_n y_{k-n} + b_0 u_{k-d} + b_1 u_{k-d-1} + \cdots + b_m u_{k-d-m} + v_k$$

bzw. in Vektorschreibweise

$$y_k = [-y_{k-1}, -y_{k-2}, \ldots, -y_{k-n}, u_{k-d}, \ldots, u_{k-d-m}][a_1, \ldots, a_n, b_0, \ldots, b_m]^T + v_k$$
$$y_k = s_k p + v_k \text{ mit } s_k \text{ und } p \text{ wie gehabt.}$$

Zusammenfassend ergibt das

$$y = Sp + v \ . \tag{3}$$

Unter gewissen Voraussetzungen bezüglich Unkorreliertheit der Störgröße mit den Input- und Outputgrößen lässt sich zeigen, dass der im vorigen Abschnitt entwickelte Least Squares Schätzer auch für dieses Problem konsistent im quadratischen Mittel ist. Hat der Störvektor v die Kovarianzmatrix R, so lassen sich gute Ergebnisse erzielen, wenn wir bei der Berechnung unserer Least Squares Schätzung das Skalarprodukt $< \cdot, \cdot >_{R^{-1}}$ verwenden. Für Details siehe Skriptum Prozessidentifikation [PROZID, Seiten 23 ff. und 34 ff.].

3.2 Rekursive Formulierung der Least Squares Schätzung

Der Least Squares Algorithmus aus dem vorigen Abschnitt lässt sich auch rekursiv formulieren. Dies macht zum Beispiel in Simulink Sinn. Bezeichne dazu Y_N den Vektor y aus dem vorigen Abschnitt, so wie er aussieht, wenn das System N Schritte gelaufen ist, S_N die Matrix S aus dem vorigen Abschnitt, so wie sie für N Schritte aussieht, Q_N die Kovarianzmatrix der Störeinträge nach N Schritten (wobei wir verlangen, dass $Q_{N+1}(1 : N - \max\{n, m\}, 1 : N - \max\{n, m\}) = Q_N$, d.h. die Einträge von Q_{N+1} in den Zeilen und Spalten von $1 : N - \max\{n, m\}$ entsprechen den Einträgen von Q_N) und $\hat{p_N}$ die Schätzung des Parametervektors nach N Schritten. Wir haben mit diesen Bezeichnungen

$$\hat{p_N} = (S_N^T Q_N S_N)^{-1} S_N^T Q_N Y_N$$
$$\hat{p_{N+1}} = (S_{N+1}^T Q_{N+1} S_{N+1})^{-1} S_{N+1}^T Q_{N+1} Y_{N+1}$$

mit

$$Y_N = \begin{pmatrix} y_{\max\{n,m\}} \\ \vdots \\ y_N \end{pmatrix}$$

und

$$S_N = \begin{pmatrix} s_{max\{n,m\}} \\ \vdots \\ s_N \end{pmatrix}$$
$$= \begin{pmatrix} -y_{\max\{n,m\}-1} & \cdots & -y_{\max\{n,m\}-n} & u_{\max\{n,m\}} & \cdots & u_{\max\{n,m\}-m} \\ \vdots & \ddots & & \vdots & \ddots & \\ -y_{N-1} & \cdots & -y_{N-n} & u_N & \cdots & u_{N-m} \end{pmatrix} .$$

Indem wir die alten und neuen Einträge von S_{N+1}, Q_{N+1} und y_{N+1} getrennt betrachten und die Woodbury-Matrix-Identität[1] zur Matrizeninversion verwenden, erhalten wir nach einigen Umformungen - mit der Einschränkung, dass Q_{N+1} die Gestalt $\begin{pmatrix} Q_N & 0 \\ 0 & q_{N+1} \end{pmatrix}$ hat - die Rekursionsformeln

$$\begin{aligned} k_{N+1} &= P_N s_{N+1}^T (q_{N+1}^{-1} + s_{N+1} P_N s_{N+1}^T)^{-1} , \\ P_{N+1} &= P_N - k_{N+1} s_{N+1} P_N , \\ \hat{p_{N+1}} &= \hat{p_N} + k_{N+1}(y_{N+1} - s_{N+1}\hat{p_N}) . \end{aligned} \tag{4}$$

3.3 Kalman-Filter

Hier möchten wir das Problem lösen, für ein System der Gestalt (1)

$$\begin{aligned} x_{k+1} &= \Phi_k x_k + \Gamma_k u_k + G_k w_k \\ y_k &= C_k x_k + D_k u_k + H_k w_k + v_k , \end{aligned}$$

unter Berücksichtigung der Messung des Outputs y_k (und der vorangegangen Outputs) einen Schätzwert für den Zustand x_{k+1} zu berechnen. Auch diesmal möchten wir dies rekursiv tun. Dazu werden wir für einen Schritt, mit den Mitteln der Bayes-Statistik, ausgehend von einer a-priori-Verteilung für die stochastische Größe x_k zunächst die a-posteriori-Verteilung von x_k (und w_k) bei bekanntem Messwert $\overline{y}_k$ und daraus dann die Verteilung von x_{k+1}, die als a-priori-Verteilung für den nächsten Schritt im Algorithmus dient, berechnen.

Um das Problem handhabbar zu machen, werden wir annehmen, dass das ursprüngliche x_0, sowie die Störgrößen w_k und v_k Normalverteilt sind. In Folge werden wir sehen, dass aufgrund der Linearität des Systems auch alle anderen auftretenden Verteilungen Normalverteilungen sind, und sich somit durch ihren Mittelwert und ihre Kovarianzmatrix vollständig charakterisieren lassen. Außerdem nehmen wir an, dass der Startwert x_0, w_i und v_j für alle i, j unkorreliert sind. w_i und w_j sollen unkorreliert sein für $i \neq j$ und v_i und v_j unkorreliert für $i \neq j$. Daraus folgt auch, dass x_i nicht mit v_j korreliert, und dass x_i nicht mit w_j korreliert für $j \geq i$.

Bevor wir beginnen brauchen wir ein paar Fakten über Normalverteilte Zufallsvariablen (siehe auch Skriptum Angewandte Statistik [AST]).

Theorem 3.1. *Ist $X \sim N(\mu, \Sigma)$, A eine Matrix, b ein Vektor, so gilt:*

$$AX + b \sim N\left(A\mu + b, A\Sigma A^T\right) .$$

[1] $(A + BCD)^{-1} = A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}A^{-1}$

Theorem 3.2. *Ist* $\begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \sim N\left(\begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}\right)$*, so ist die bedingte Verteilung*

$$X_1|_{X_2=x_2} \sim N\left(\mu_1 + \Sigma_{12}\Sigma_{22}^{-1}\left(x_2 - \mu_2\right), \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}\right) \ .$$

Wir nehmen also an, wir hätten a-priori $x_k \sim N\left(\mu_{x_k}, \Sigma_{x_k}\right)$ und $w_k \sim N\left(0, \Sigma_{w_k}\right)$ sowie $v_k \sim N\left(0, \Sigma_{v_k}\right)$, mit x_k, w_k und v_k untereinander unkorreliert. Da laut (1) gilt

$$\begin{pmatrix} x_k \\ w_k \\ y_k \end{pmatrix} = \begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ C_k & H_k & I \end{pmatrix} \begin{pmatrix} x_k \\ w_k \\ v_k \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ D_k u_k \end{pmatrix}$$

bekommen wir für die gemeinsame Verteilung

$$\begin{pmatrix} x_k \\ w_k \\ y_k \end{pmatrix} \sim N\left(\begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ C_k & H_k & I \end{pmatrix} \begin{pmatrix} \mu_{x_k} \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ D_k u_k \end{pmatrix}, \right.$$
$$\left. \begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ C_k & H_k & I \end{pmatrix} \begin{pmatrix} \Sigma_{x_k} & 0 & 0 \\ 0 & \Sigma_{w_k} & 0 \\ 0 & 0 & \Sigma_{v_k} \end{pmatrix} \begin{pmatrix} I & 0 & {C_k}^T \\ 0 & I & {H_k}^T \\ 0 & 0 & I \end{pmatrix} \right) =$$
$$= N\left(\begin{pmatrix} \mu_{x_k} \\ 0 \\ C_k\mu_{x_k} + D_k u_k \end{pmatrix}, \begin{pmatrix} \Sigma_{x_k} & 0 & \Sigma_{x_k}{C_k}^T \\ 0 & \Sigma_{w_k} & \Sigma_{w_k}{H_k}^T \\ C_k\Sigma_{x_k} & H_k\Sigma_{w_k} & C_k\Sigma_{x_k}{C_k}^T + H_k\Sigma_{w_k}{H_k}^T + \Sigma_{v_k} \end{pmatrix}\right) \ .$$

Jetzt verwenden wir die Information über unseren Messwert $\overline{y}_k$. Laut 3.2 ist die a-posteriori-Verteilung von x_k und w_k im Punkt $y_k = \overline{y}_k$

$$\begin{pmatrix} x_k \\ w_k \end{pmatrix}|_{y_k=\overline{y}_k} \sim N\Bigg(\begin{pmatrix} \mu_{x_k} \\ 0 \end{pmatrix} + \begin{pmatrix} \Sigma_{x_k}{C_k}^T \\ \Sigma_{w_k}{H_k}^T \end{pmatrix} \left(C_k\Sigma_{x_k}{C_k}^T + H_k\Sigma_{w_k}{H_k}^T + \Sigma_{v_k}\right)^{-1}$$
$$\left(\overline{y}_k - C_k\mu_{x_k} - D_k u_k\right),$$
$$\begin{pmatrix} \Sigma_{x_k} & 0 \\ 0 & \Sigma_{w_k} \end{pmatrix} - \begin{pmatrix} \Sigma_{x_k}{C_k}^T \\ \Sigma_{w_k}{H_k}^T \end{pmatrix} \left(C_k\Sigma_{x_k}{C_k}^T + H_k\Sigma_{w_k}{H_k}^T + \Sigma_{v_k}\right)^{-1} \begin{pmatrix} C_k\Sigma_{x_k} & H_k\Sigma_{w_k} \end{pmatrix}\Bigg)$$

Daraus berechnen wir die neue a-priori-Verteilung für x_{k+1} über

$$x_{k+1} = \begin{pmatrix} \Phi_k & G_k \end{pmatrix} \begin{pmatrix} x_k \\ w_k \end{pmatrix} + \Gamma_k u_k \ .$$

Die Verteilung lautet also mit Satz 3.1

$$\begin{aligned} x_{k+1} \sim N\Bigg(& \Phi_k\mu_{x_k} + \Gamma_k u_k + \left(\Phi_k\Sigma_{x_k}{C_k}^T + G_k\Sigma_{w_k}{H_k}^T\right)\left(C_k\Sigma_{x_k}{C_k}^T + H_k\Sigma_{w_k}{H_k}^T + \Sigma_{v_k}\right)^{-1} \\ & \left(\overline{y}_k - C_k\mu_{x_k} - D_k u_k\right), \\ & \Phi_k\Sigma_{x_k}{\Phi_k}^T + G_k\Sigma_{w_k}{G_k}^T - \left(\Phi_k\Sigma_{x_k}{C_k}^T + G_k\Sigma_{w_k}{H_k}^T\right) \\ & \left(C_k\Sigma_{x_k}{C_k}^T + H_k\Sigma_{w_k}{H_k}^T + \Sigma_{v_k}\right)^{-1}\left(C_k\Sigma_{x_k}{\Phi_k}^T + H_k\Sigma_{w_k}{G_k}^T\right)\Bigg) \ . \end{aligned} \tag{5}$$

Da der Bayes-Schätzer (bezüglich einer quadratischen Verlustfunktion) der Erwartungswert ist, können wir hieraus auch gleich eine Punktschätzung für x_{k+1} ablesen.
Die Rekursionsformeln sind also:

$$\begin{aligned}
\hat{K}_k &= \left(\Phi_k \Sigma_{x_k} {C_k}^T + G_k \Sigma_{w_k} {H_k}^T\right) \left(C_k \Sigma_{x_k} {C_k}^T + H_k \Sigma_{w_k} {H_k}^T + \Sigma_{v_k}\right)^{-1} \\
\hat{x}_{k+1} &= \Phi_k \mu_{x_k} + \Gamma_k u_k + \hat{K}_k \left(\overline{y}_k - C_k \mu_{x_k} - D_k u_k\right) \\
P_{k+1} &= \Phi_k \Sigma_{x_k} {\Phi_k}^T + G_k \Sigma_{w_k} {G_k}^T - \hat{K}_k \left(C_k \Sigma_{x_k} {\Phi_k}^T + H_k \Sigma_{w_k} {G_k}^T\right).
\end{aligned}$$

Bemerkung: Es ist auch möglich, den eben hergeleiteten Schätzwert ohne die Annahme, dass die Anfangsgrößen Normalverteilt sind, zu erhalten, indem man stattdessen die Suche auf lineare Schätzer beschränkt. Für eine Durchrechnung dieses Ansatzes siehe Skriptum Prozessidentifikation [PROZID, Seiten 45 ff.]. (Vorsicht: Die Herleitung dort ist nicht ganz korrekt, da die Information, die wir aus der Messung von y_k über w_k erhalten, vernachlässigt wird. Statt dem Term $\left(\Phi_k \Sigma_{x_k} {C_k}^T + G_k \Sigma_{w_k} {H_k}^T\right)$, der in unserem Ausdruck für Schätzer und Kovarianzmatrix vorkommt, findet sich dort nur $\Phi_k \Sigma_{x_k} {C_k}^T$. Wir haben Anfangs auch diese Variante programmiert und in unserem Beispiel beim Ausbessern eine Verkleinerung des Fehlers auf ca. ein Zehntel beobachtet.)

4 Implementierung

Für die Implementierung und das Testen haben wir Zahlen und dazugehörige Kovarianzmatrizen frei erfunden und in das File `variables.m` gespeichert, neben anderen Variablen wie Anfangszeitpunkt, Endzeitpunkt, Schrittweite, Anfangsfrequenz etc. Auf dieses File wird bei allen 3 Verfahren zugegriffen.

4.1 Least Squares Identifikation in Matlab

Die Least Squares Identifikation wie wir sie hergeleitet haben besteht im Wesentlichen einfach daraus, die Matrix S aufzustellen und dann das Gleichungssystem (2) zu lösen. Das haben wir in der Funktion `lsidentify` implementiert. Es sieht im Grunde so aus:

```
firstentry=max(yl+1,ul); % Welches k gibt uns die erste lineare
   Gleichung
N=length(y);

% S so zusammengestueckelt, dass gilt S*[a.';b.']=ys
ys=y(firstentry:N);
S = zeros(N-firstentry+1,yl+ul);

for k=1:yl
    S(:,k)=y(firstentry-k:N-k);
end
for k=0:(ul-1)
    S(:,k+1+yl)=u(firstentry-k:N-k);
end
```

```
phat = linsolve((S.' * Q(firstentry:end,firstentry:end) * S),
   (S.'* Q(firstentry:end,firstentry:end) *ys));
```

Um das Ganze zu testen benötigen wir ein lineares System mit stochastischer Störung. Wir haben eine Funktion `dlsim` geschrieben, die analog zu Matlabs eingebautem `lsim` die Evolution eines LTI (lineares, zeitinvariantes System) berechnet, wenn zusätzlich zu den Parametern von `lsim` noch eine stochastische Störung dazu kommt.

Unser Testaufruf erfolgt dann, indem erst die Variablen sowie eine Test-Input- und -Transferfunktion definiert werden.

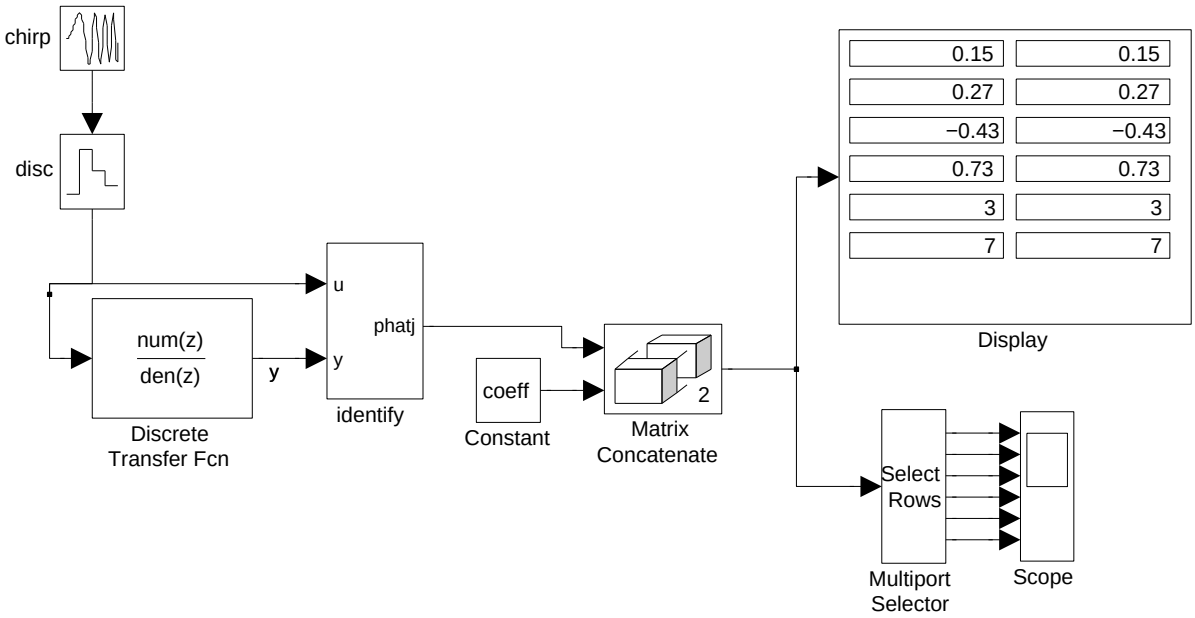
```
variables

ts=(t0:stepsize:tend)'; %sampling times
N=length(ts); % Anzahl Schritte

testinp=@(t) sin(omega*t.*t); %input function
u=testinp(ts); %inputs at times ts

%Störmatrix berechnen:
ddd=[1;2;3;4;3;2;7]*0.1;
disturbance=R*random('norm',0,ones(size(u)));
Q=inv(R);

testt=tf([ucoeff;0].',[1;-ycoeff].',stepsize,'Variable','z');
```

Anschließend simulieren wir mit unserer `dlsim`-Funktion den Verlauf unseres Systems und identifizieren dieses mit `lsidentify`. Um die Qualität des Schätzers zu beurteilen, berechnen wir noch zusätzlich $||p - \hat{p}||$.

```
y=dlsim(testt,u,disturbance,ts);
phat = lsidentify(y,u,yl,ul,Q);
norm(phat-coeff)
```

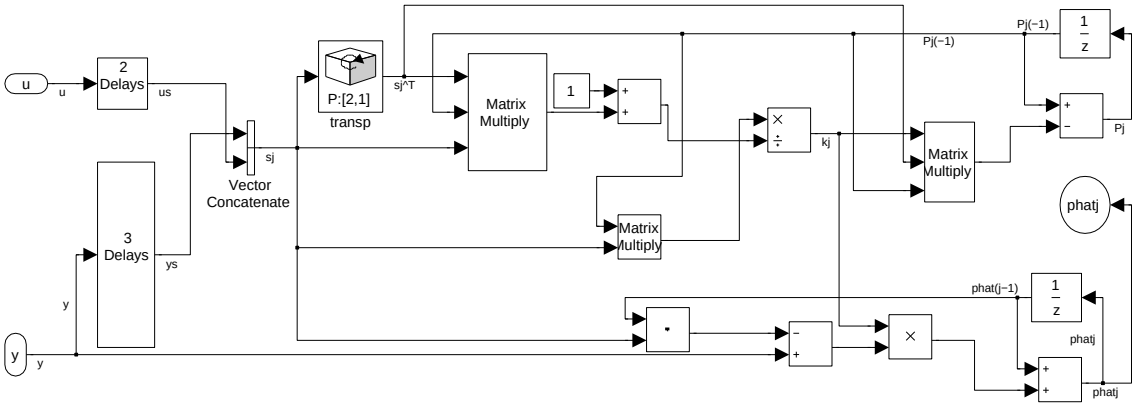
4.2 Least Squares Identifikation in Simulink

In Simulink haben wir der Einfachheit halber nur die Least Squares Identifikation ohne stochastische Störung getestet, da auch in Simulink kein vorgefertigter Block existiert, der ein LTI-System mit Störung simuliert. Das experimentelle Setup sieht wie folgt aus:



Hier diskretisieren wir ein Chirp-Signal für die Eingangsgröße u und bilden uns wieder die diskrete Transferfunktion wie im klassischen Least Squares Verfahren für die Ausgangsgröße. Wir identifizieren das System (siehe Ausführungen im Anschluss) und geben im Display-Fenster links die tatsächlichen Werte von p, in der rechten Spalte die Werte von $\hat{p}$ aus. Der Plot gibt den Verlauf dieser Werte von $\hat{p}$ aus nach der Anzahl der Durchläufe.

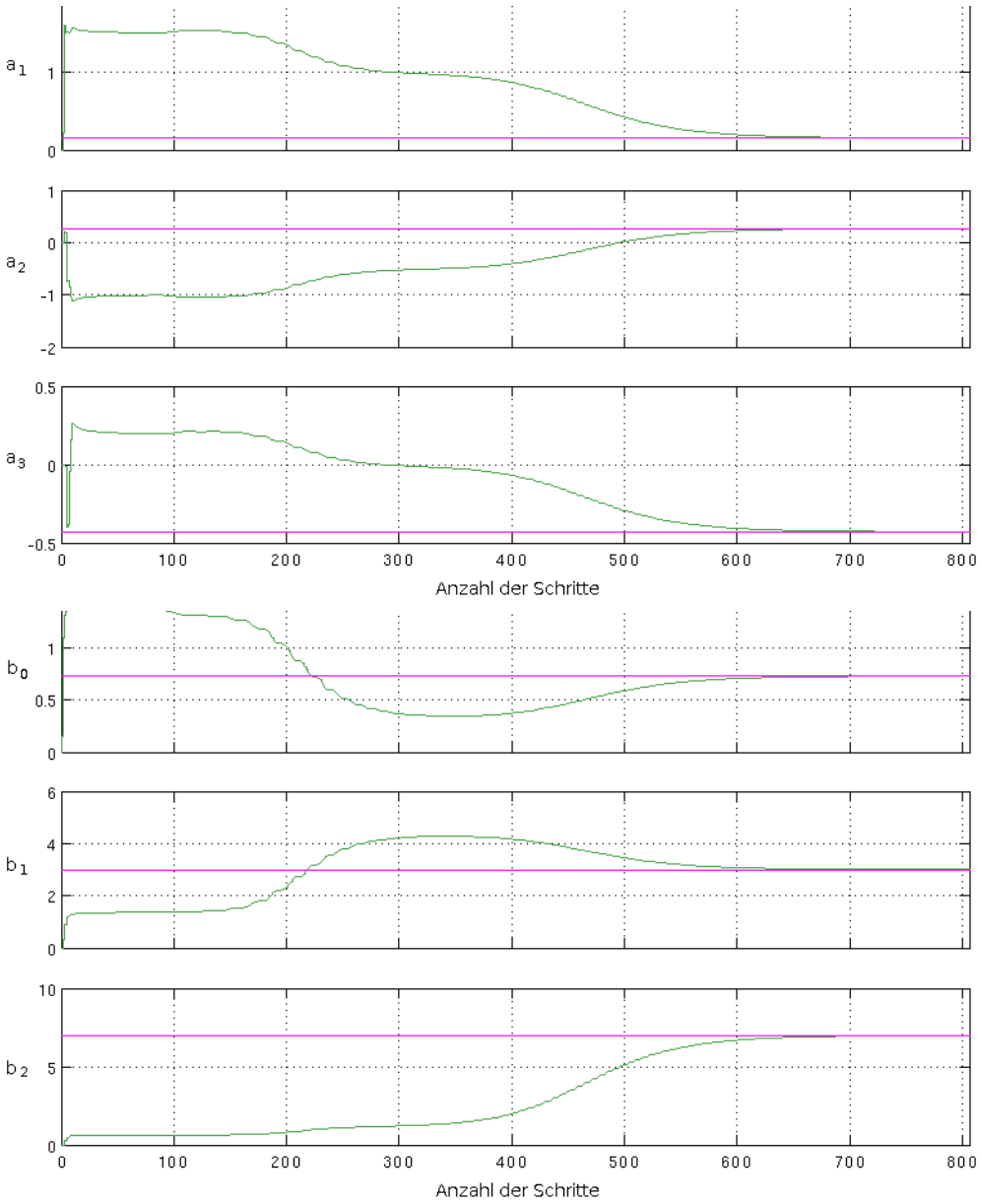
Der `identify`-Block implementiert die Rekursionsformeln (4):



Hier können wir auch die Grenzen der graphischen Programmierumgebung Simulink erkennen. Obwohl wir einige Zeit in das Layout investiert haben, ist das obige Schema nicht unbedingt

die lesbarste Variante, um den RLS-Algorithmus auszudrücken.

In Simulink können wir beobachten, wie sich die Schätzungen der Parameter über die Zeit hinweg verbessern. In der unteren Grafik sehen wir in den ersten drei Zeilen die Koeffizienten a_i und in den unteren drei Zeilen die Koeffizienten b_i. Wir sehen lila jeweils den echten Wert und grün den Verlauf unserer Schätzung.

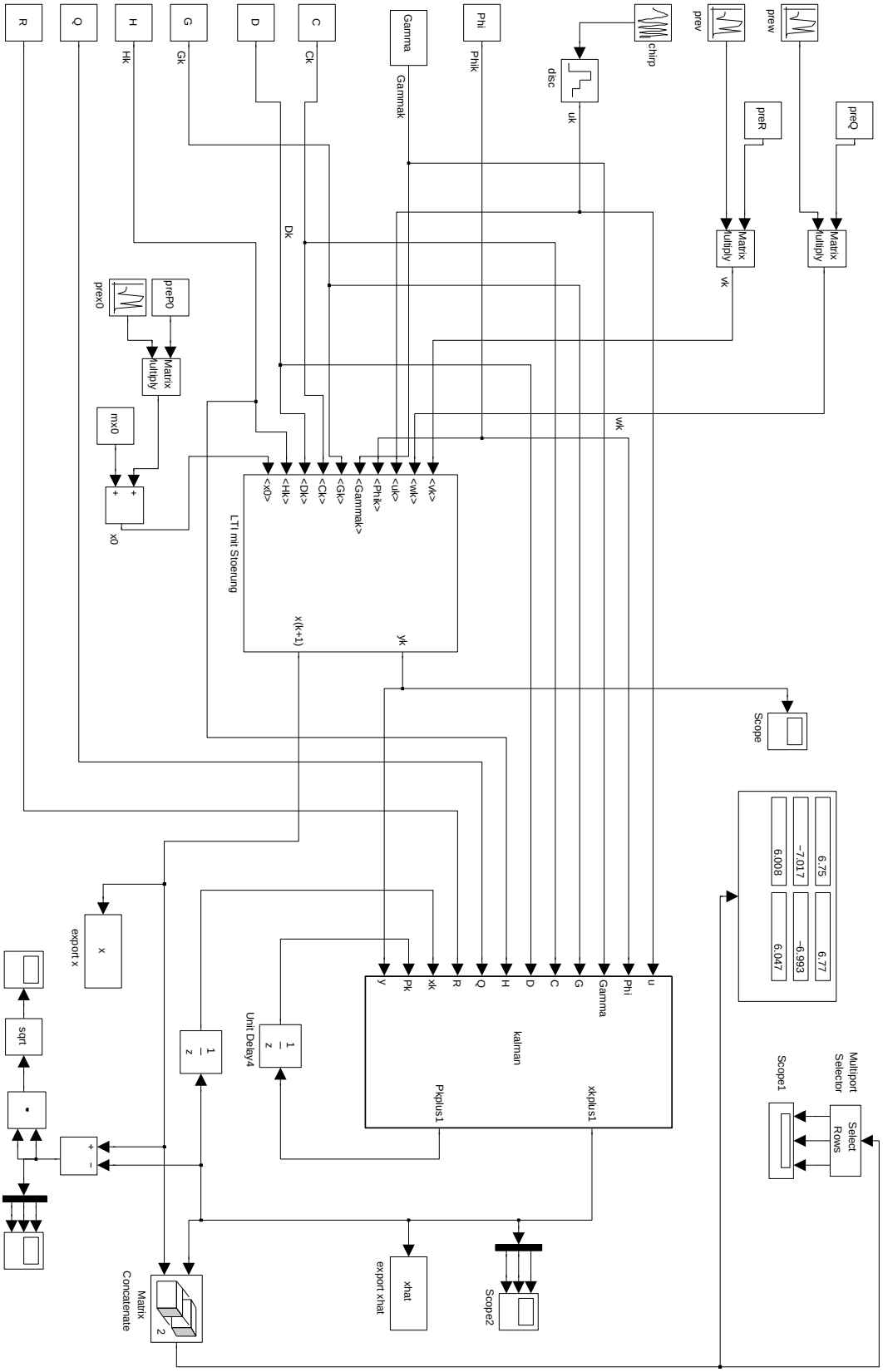


4.3 Kalman-Filter in Simulink

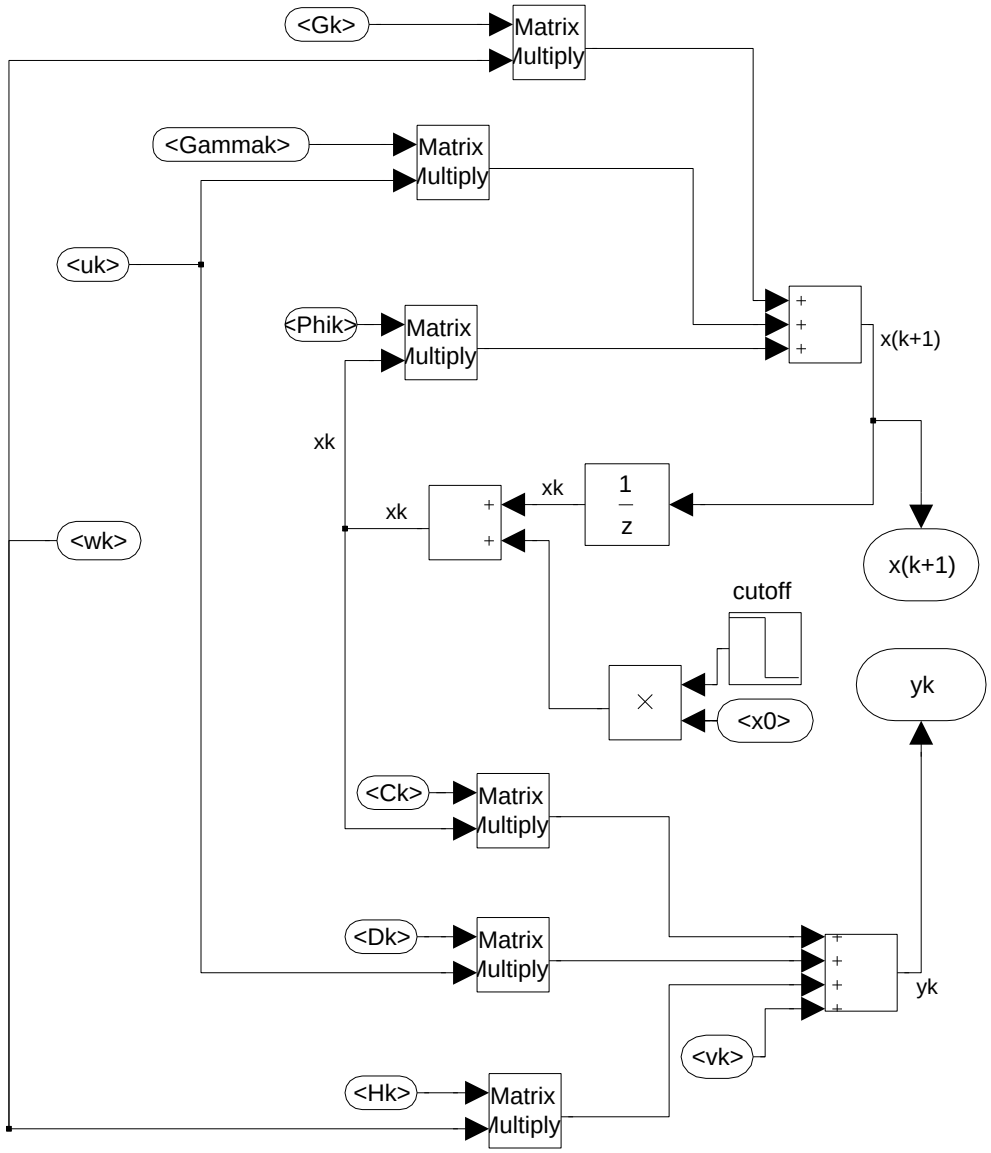
Für das Kalman-Filter haben wir wieder die Variablen in unserem `variables` file definiert. Zunächst haben wir das LTI System mittels der Formeln für y_k und x_{k+1} implementiert. Die sich ergebenden y_k werden an die Funktion übergeben, welche das Kalman-Filter durchführt. Die Ausgabewerte lassen sich in den verschiedenen Plots beobachten, wie z.B. der Verlauf der y_k im Scope, die Differenz zwischen x_{k+1} und $\hat{x}_{k+1}$ im Scope3 und der Differenzbetrag im

Scope4. Im Scope1 werden die Werte von x_{k+1} und $\hat{x}_{k+1}$ direkt übereinander geplottet; das darüberliegende Display zeigt während des Ausführens die aktuellen Werte.

Unser experimentelles Setup:



Das LTI-System haben wir ebenfalls vollständig in Simulink implementiert:



Die Rekursionsformeln für die tatsächliche Durchführung des Kalman-Filter (5) haben wir in eingebettetem Matlab-Code implementiert:

```
function [xkplus1,Pkplus1] =
   kalman(u,Phi,Gamma,G,C,D,H,Q,R,xk,Pk,y)

K = (Phi*Pk*C' + G*Q*H.')*((C*Pk*C'+H*Q*H'+R)^(-1));
xkplus1 = Phi*xk + Gamma*u + K*(y-C*xk-D*u);
Pkplus1 = Phi*Pk*Phi.' + G*Q*G.' - K*(C*Pk*Phi.' + H*Q*G.');

end
```

4.4 Kalman-Filter in Matlab

Für die Implementierung des Kalman-Filter in Matlab wird im Großen und Ganzen dasselbe durchgeführt wie im oben angeführten Simulink-Beispiel. Nach der Initialisierung der Variablen erzeugen wir uns normalverteilte Zufallsvektoren v und w und implementieren damit das LTI mit den Formeln laut Skriptum Prozessidentifikation [PROZID].

```
for k=1:N
    prev=random('Normal',0,1);
    prew=[random('Normal',0,1,2,1)]; % Standardnormalverteilter
       Zufallsvektor mit Dimension 2x1
    v=preR*prev;
    w=preQ*prew;
    x(:,k+1)=Phi*x(:,k)+ Gamma*u(k) + G*w;
    y(k)=C*x(:,k) + D*u(k) + H*w + v;
end
```

Anschließend wenden wir wieder die Formeln für $\hat{K}_k$, $\hat{x}_{k+1}$ sowie P_{k+1} an:

```
for k=1:N
    K = (Phi*P(:,:,k)*C' +
       G*Q*H.')*((C*P(:,:,k)*C'+H*Q*H'+R)^(-1));
    xhat(:,k+1) = Phi*xhat(:,k) + Gamma*u(k) +
       K*(y(k)-C*xhat(:,k)-D*u(k));
    P(:,:,k+1) = Phi*P(:,:,k)*Phi' + G*Q*G' - K*(C*P(:,:,k)*Phi'
       + H*Q*G.');
end
```

Die Resultate der Matlab-Implementierung sind gut mit denen aus Simulink vergleichbar.

5 Anhang

5.1 Code

5.1.1 Variables

```
ycoeff=[0.15;0.27;-0.43]; %nenner der TF = [1,-ycoeff]
ucoeff=[0.73;3;7]; %zaehler der TF

yl=length(ycoeff); %Laenge des y-Koeffizientenvektors
ul=length(ucoeff); %Laenge des u-Koeffizientenvektors

coeff=[ycoeff;ucoeff]; %Kombiniert die Koeffizientenvektoren y
   und u

t0=0;
f0=0;
tend=30;
omega=5;

Phi = [0.15,0.27,-0.43; 1,0,0; 0,1,0];
Gamma = [4;0;0];
C = [0.7774,1.799,-0.07848];
D = [ 0.73 ];
G = [ 1, 2; 0, 5; 4, 1];
H = [ 8, 3];
preQ = [3,7;0,1]*10^(-2);
Q = preQ*preQ'; % Q ist Kovarianzmatrix von w, also symmetrisch
   + pos. definit. preQ = Hilfsmittel dafuer
preR = [10]*10^(-2);
R = preR*preR';
preP0 = [1, 2, 3; 2, 3, 4; 3, 4, 5];
P0 = preP0*preP0.';
mx0 = [0.3;4;-3];

R=spdiags(repmat(ddd,ceil(N/length(ddd)),1),0,N,N);

stepsize=0.01;
```

5.1.2 dlsim

```
function y = dlsim(sys,u,v,t)
% dlsim(sys, u, v, t) returns the time response of the lti model
% sys', where sys is of the form y(z) = B(z)/A(z) * u(z) und sys'
% is y(z) = B(z)/A(z) + 1/A(z) * v(z), that is, a _d_isturbance v
% (hence dlsim) is added in every step. We need this specific
% model structure because it is the model structure identified by
% the basic least squares model identification algorithm, that we
% have programmed and want to test.
```

```
% At the moment t is ignored.

[num,den]=tfdata(sys,'v');

if(abs(den(1)) < 5*eps)
    error('first entry of denominator is singular. this case is
       not handled yet.')
end

if(any(size(num) ~= size(den)) || numel(num) ~= length(num))
    error('I was expecting numerator and denominator vectors
       returned by tfdata to always be the same length and
       one-dimensional. This assumption was just violated.')
end

n=length(num);
%checks whether sizes of u and t and v match
if(not(all(size(u) == size(t)) && all(size(t) == size(v))) ||
   numel(u) ~= length(u) )
    error('Inputs in wrong format.')
end

% make vector dimensions parallel
if(min(size(num) .* size(u)) ~= 1)
    num = num.';
    den = den.';
end

N = length(u);

y = zeros(size(u));

%order is important here
num = num / den(1);
den = den / den(1);

for i=1:(n-1)
    y(i) = sum(y(1:i-1) .* -den(i:-1:2)) + sum(u(1:i) .*
       num(i:-1:1)) + v(i);
end

for i=n:N
    y(i) = sum(y((i-n+1):(i-1)) .* -den(end:-1:2)) +
       sum(u((i-n+1):i) .* num(end:-1:1))  + v(i);
end

end
```

5.1.3 lsidentify

```
function phat = lsidentify(y,u,yl,ul,Q)

% Die Relation y(k) = y(k-1)*a(1)+y(k-2)*a(2)+...+y(k-yl)*a(yl) +
% u(k)*b(0)+k(k-1)*b(1)+...+u(k-ul+1)*b(ul-1) + moeglicherweise
% eine Stoerung v(k) bestimmt fuer jedes k eine Gleichung, die
% linear in [a,b] ist. Wir wollen das entsprechende
% (ueberbestimmte) lineare Gleichungssystem loesen um [a,b] zu
% finden.

firstentry=max(yl+1,ul); % Welches k gibt uns die erste lineare
   Gleichung
N=length(y);

% S so zusammengestueckelt, dass gilt S*[a.';b.']=ys

ys=y(firstentry:N);
S = zeros(N-firstentry+1,yl+ul);

for k=1:yl
    S(:,k)=y(firstentry-k:N-k);
end
for k=0:(ul-1)
    S(:,k+1+yl)=u(firstentry-k:N-k);
end

phat = linsolve((S.' * Q(firstentry:end,firstentry:end) * S),
   (S.'* Q(firstentry:end,firstentry:end) *ys));

end
```

5.1.4 Least Squares Aufruf

```
variables

ts=(t0:stepsize:tend)'; %sampling times
N=length(ts); % Anzahl Schritte

testinp=@(t) sin(omega*t.*t); %input function
u=testinp(ts); %inputs at times ts

ddd=[1;2;3;4;3;2;7]*0.1;
%R=spdiags(repmat(ddd,ceil(N/length(ddd)),1),0,N,N); -> in
   variables
%ausgelagert
disturbance=R*random('norm',0,ones(size(u)));
Q=inv(R);

testt=tf([ucoeff;0].',[1;-ycoeff].',stepsize,'Variable','z');
% SYS = TF(NUM, DEN, TS, 'PropertyName1', 'PropertyValue1')
```

```
%  creates a discrete-time transfer function with
%  sample time TS (set TS=-1 if the sample time is undetermined),
%  with numerator(s) NUM and denominator(s) DEN.
%  The output SYS is a TF object.
%------------------
%testt:
%
%Transfer function:
%    0.73 z^3 + 3 z^2 + 7 z
%------------------------------
%z^3 - 0.15 z^2 - 0.27 z + 0.43
%
%Sampling time: 0.01
%-----------------

y=dlsim(testt,u,disturbance,ts);
% dlsim(sys, u, v, t) returns the time response of the lti model
% sys', where sys is of the form y(z) = B(z)/A(z) * u(z) und sys'
% is y(z) = B(z)/A(z) + 1/A(z) * v(z), that is, a _d_isturbance v
% (hence dlsim) is added in every step. We need this specific
% model structure because it is the model structure identified by
% the basic least squares model identification algorithm, that we
% have programmed and want to test.
% At the moment t is ignored.

phat = lsidentify(y,u,yl,ul,Q);

% Kontrolle: es kommt ca. coeff=[ycoeff,ucoeff] raus
phat
phat-coeff
norm(phat-coeff)

plot(ts,y,'m',ts,u,'g') %Input und Output des Systems, das wir
   schaetzen
```

5.1.5 Kalman-Filter

```
variables
N=(tend-t0)/stepsize;
t=t0:stepsize:tend;

u=chirp(t,f0,tend,tend*omega/pi);
% CHIRP  Swept-frequency cosine generator.
%    Y = CHIRP(T,F0,T1,F1) generates samples of a linear
   swept-frequency
%    signal at the time instances defined in array T.  The
   instantaneous
%    frequency at time 0 is F0 Hertz.  The instantaneous
   frequency F1
```

```
%    is achieved at time T1.  By default, F0=0, T1=1, and F1=100.

x0=[0;0;0];
x=zeros(length(Phi),N);
x(:,1)=x0;
y=zeros(N,1);

for k=1:N
    prev=random('Normal',0,1);
    prew=[random('Normal',0,1,2,1)]; % Standardnormalverteilter
       Zufallsvektor mit Dimension 2x1
    v=preR*prev;
    w=preQ*prew;
    x(:,k+1)=Phi*x(:,k)+ Gamma*u(k) + G*w;
    y(k)=C*x(:,k) + D*u(k) + H*w + v;
end

K=zeros(N,1); % K aus R(ixj), i=Anzahl Eintraege von x, j=anzahl
   eintraege von y.
xhat=zeros(length(Phi),N); %dim(xhat) = dim(x)
P=zeros(length(Phi),length(Phi),N); %P(i) sind quadratisch


for k=1:N
    K = (Phi*P(:,:,k)*C' +
       G*Q*H.')*((C*P(:,:,k)*C'+H*Q*H'+R)^(-1));
    xhat(:,k+1) = Phi*xhat(:,k) + Gamma*u(k) +
       K*(y(k)-C*xhat(:,k)-D*u(k));
    P(:,:,k+1) = Phi*P(:,:,k)*Phi' + G*Q*G' - K*(C*P(:,:,k)*Phi'
       + H*Q*G.');
end

plot(xhat(1,:))
%plot(xhat(2,:))
%plot(xhat(3,:))
```

Literatur

[AST] Gurker, Werner: *Angewandte Statistik*. 2011.

[PROZID] Kugi, Andreas: *Skriptum Prozessidentifikation*. 2009.