Aleksandar Karakas, BSc BSc

Zeitreihenanalyse mit dem vektorwertigen Fehlerkorrekturmodell

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

Masterstudium Finanz- und Versicherungsmathematik

eingereicht an der

Technischen Universität Graz

Betreuer

Univ.-Prof.i.R. Dipl.-Ing. Dr.techn. Ernst Stadlober

Institut für Statistik

Graz, Juli 2017

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Datum

Unterschrift

Danksagung

Ich möchte Herrn Prof. Stadlober danken, der sich trotz nahender Rente der Betreuung der vorliegenden Arbeit annahm. Sein Fachwissen und gründliches Korrekturlesen führten zu einer wesentlichen Aufwertung dieser Masterarbeit.

Mein Dank gilt auch Josef Perktold und Kevin Sheppard, die mich als Mentoren im Google Summer of Code 2016 unterstützten. Sie zeichneten sich durch ständige Erreichbarkeit während des Projektes aus und ersparten mir mit hilfreichen Tipps viel Zeit bei der Programmierung.

Vor allem möchte ich meinen Eltern danken, die mir das Mathematikstudium sowie meine weiteren Studien überhaupt erst ermöglichten.

Abstract

This master's thesis covers the theoretical fundamentals of the vector autoregressive model and based on this model the vector error correction model (VECM) is introduced. Many topics of interest for a time series analyst are discussed starting with the model specification which includes the determination of the model order as well as the cointegration rank. Next, the parameter estimation is discussed followed by the presentation of diagnostic tests based on the residuals. The theoretical part is concluded by the discussion of forecasts and structural analysis which includes two causality tests as well as impulse response analysis. After the theoretical discussion a way to analyze a VECM using a Python package called *statsmodels* is presented. The VECM-related module contained in *statsmodels* was contributed by the author of this thesis as a result of a project funded by *Google*. After the demonstration of the module's functionality the VECM-module is compared to the software packages *R*, *Stata*, and *JMulTi*.

Kurzfassung

Die vorliegende Arbeit befasst sich mit den theoretischen Grundlagen des Vektorautoregressiven Modells und darauf aufbauend mit dem vektorwertigen Fehlerkorrekturmodell (VECM). Neben der Modellspezifikation, welche die Bestimmung von Modellordnung und Kointegrationsrang zum Inhalt hat, wird die Schätzung der Parameter besprochen. Es folgt eine Diskussion über diagnostische Tests, welche auf der Analyse der Residuen beruhen. Der theoretische Teil endet mit der Betrachtung von Vorhersagen und strukturellen Analyseverfahren. Letztere beinhalten die Impuls-Response-Analyse sowie zwei Kausalitätstests. Es folgt eine praktische Anwendung der Theorie mit Hilfe des Python-Packages *statsmodels*. Dessen VECM-Modul wurde vom Autor der vorliegenden Arbeit im Rahmen eines von *Google* finanzierten Projekts erstellt. Am Ende der Arbeit folgt ein Vergleich mit den Softwarepaketen *R*, *Stata* sowie *JMulTi*.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einführung

Wie Lütkepohl, (2005) in seinem Vorwort feststellt, verlieren klassische Modelle zur Behandlung multivariater Zeitreihen, wie das Vektorautoregressive Modell, in der angewandten Ökonometrie an Bedeutung. Grund dafür ist das Aufkommen neuer Ansätze. Einen solchen stellt das vektorwertige Fehlerkorrekturmodell dar, dessen Beschreibung ein Ziel der vorliegenden Arbeit ist. Einen Überblick über die theoretischen Grundlagen liefern wir in Kapitel 2.

Das zweite Ziel dieser Arbeit ist die Demonstration der Anwendung dieses Modells. Diese wird computergestützt erfolgen. Angesichts der wachsenden Bedeutung des Modells haben die Entwickler verschiedener statistikbezogener Softwarepakete ihre Programme entsprechend erweitert. In *statsmodels* (vgl. Seabold und Perktold, (2010)), einer in Python geschriebenen Statistikbibliothek, waren jedoch bis vor Kurzem die zur Behandlung vektorwertiger Fehlerkorrektormodelle notwendigen Verfahren nicht implementiert. In einem von Google unterstützen Programm namens Google Summer of Code nahm sich der Autor der vorliegenden Arbeit der Aufgabe an, *statsmodels* mit dieser Funktionalität auszustatten. Anhand des aus diesem Projekt resultierenden Python-Moduls wird in Kapitel 3 die in Kapitel 2 behandelte Theorie um einen praktischen Gesichtspunkt ergänzt.

Es folgt mit Kapitel 4 eine Zusammenfassung. Anhang A beinhaltet eine Auflistung häufig genutzter Rechenregeln. Anhang B dient der Vorstellung der Software *JMulTi*. Es werden darin auch Ergebnisse angeführt, die von dieser Software produziert wurden, da diese die Vergleichsgrundlage für die aus dem Python-Modul gewonnenen Resultate darstellen.

2. Vektorwertige AR- und Fehlerkorrekturmodelle

2.1. Einleitung

Dieses Kapitel dient der Einführung vektorwertiger Fehlerkorrekturmodelle. Die Reihenfolge der vorgestellten Themen richtet sich nach der in Abbildung 2.1 skizzierten Vorgangsweise bei der Analyse von Zeitreihen. Die Abbildung ist an jene auf S. 6 in Lütkepohl, (2005) – der Hauptquelle dieses Kapitels – angelehnt.

Nach der Definition des Modells in Abschnitt 2.2 werden wir in den Abschnitten 2.3 und 2.4 die Frage der Modellspezifikation behandeln. Ausgehend von einem festgelegten Modell können wir uns der Schätzung der Parameter widmen. In Abschnitt 2.5 leiten wir dazu die Maximum-Likelihood-Schätzer her. Nach der Wahl der Parameter ist das Modell diagnostischen Verfahren zu unterziehen, um etwaige Verletzungen grundlegender Modellannahmen zu erkennen. Zwei derartige Verfahren werden in Abschnitt 2.6 vorgestellt. Wird das Modell im Zuge dieser Bewertung abgelehnt, so ist – abhängig vom Verwendungszweck des Modells – eine Anpassung der Spezifikation angezeigt. Andernfalls können wir das Modell etwa zur Vorhersage zukünftiger Werte der Zeitreihe nutzen. Dies wird in Abschnitt 2.7 beschrieben. Neben der Vorhersage sind auch strukturelle Analysen denkbar, in denen Zusammenhänge zwischen einzelnen Komponenten der Zeitreihe untersucht werden. Entsprechende Verfahren sind Gegenstand der Abschnitte 2.8, 2.9 und 2.10.

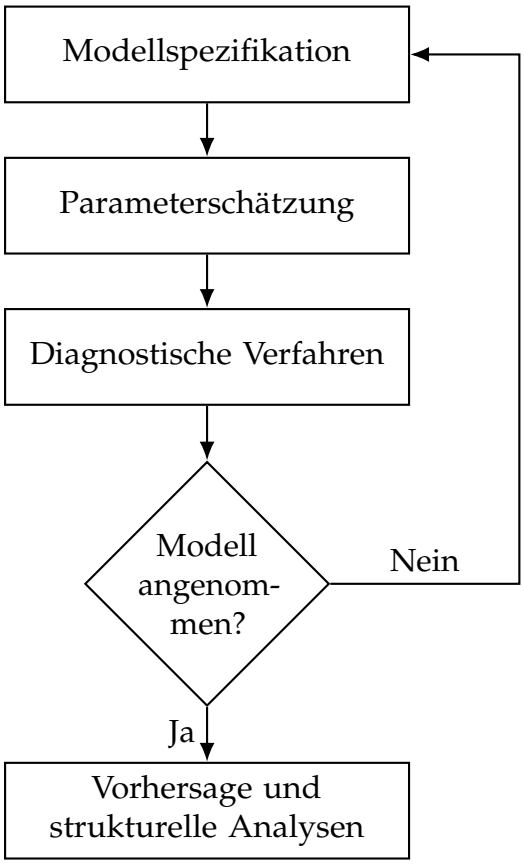

Abbildung 2.1.: Vorgangsweise in der Zeitreihenanalyse

Die meisten der vorgestellten Verfahren erbt das vektorwertige Fehlerkorrekturmodell vom Vektorautoregressiven Modell. In jenen Abschnitten, in denen die Verwandschaft dieser beiden Modelle genutzt wird, werden wir das entsprechende Thema zunächst aus dem Gesichtspunkt des Vektorautoregressiven Modells behandeln, ehe wir zu dem Modell schreiten, auf dem das Hauptaugenmerk der vorliegenden Arbeit liegt.

In der Folge werden wir die Namen beider Modelle abkürzen. Das Vektorautoregressive Modell bezeichnen wir mit VAR. Der Begriff des vektorwertigen Fehlerkorrekturmodells wird in Anlehnung an das englische *Vector Error Correction Model* in der Einzahl (mit Ausnahme des Genitivs) mit VECM abgekürzt. In der Mehrzahl und im Genitiv der Einzahl wird die Abkürzung

VECMs Verwendung finden.

2.2. Die Modelle

2.2.1. Vektorwertige autoregressive (VAR) Prozesse

Dieser Abschnitt dient der Definition und Diskussion vektorwertiger autoregressiver Prozesse. Die folgenden Ausführungen bis zum Beginn von Unterabschnitt 2.2.4 basieren auf Abschnitt 2.1 in Lütkepohl, (2005).

Definition 2.1. Modellieren wir einen Prozess mit einem VAR(p)-Modell, so nehmen wir an, dass er von der Form

$$y_t = \nu + A_1 y_{t-1} + \cdots + A_p y_{t-p} + u_t \tag{2.1}$$

ist. Dabei bezeichnet $y_t = (y_{1,t}, \ldots, y_{K,t})' \in \mathbb{R}^K$ einen Zufallsvektor. Bei $\nu \in \mathbb{R}^K$ und $A_i \in \mathbb{R}^{K \times K}$ für $i \in 1, \ldots, p$ handelt es sich um die Parameter des Modells. Wir nehmen an, dass der Prozess $u_t \in \mathbb{R}^K$ weißes Rauschen darstellt, d.h. $\mathbb{E}(u_t) = 0$, $\mathbb{E}(u_t u_t') = \Sigma_u$ und $\mathbb{E}(u_t u_s') = 0$ für $s \neq t$. Wir setzen voraus, dass die Matrix Σ_u positiv definit ist.

Ein VAR(p)-Modell lässt sich stets in ein VAR(1) umwandeln. Dazu definieren wir die Kp-dimensionalen Vektoren

$$Y_t := \begin{pmatrix} y_t \\ y_{t-1} \\ \vdots \\ y_{t-p+1} \end{pmatrix}, \quad \boldsymbol{\nu} := \begin{pmatrix} \nu \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad U_t := \begin{pmatrix} u_t \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

sowie die Parametermatrix

$$\boldsymbol{A} := \begin{pmatrix} A_1 & A_2 & \cdots & \cdots & A_p \\ I_K & 0 & \cdots & \cdots & 0 \\ 0 & I_K & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & I_K & 0 \end{pmatrix}, \tag{2.2}$$

wobei I_K die K-dimensionale Einheitsmatrix bezeichnet. Das aus dieser Umformung resultierende VAR(1)-Modell

$$Y_t = \boldsymbol{\nu} + \boldsymbol{A} Y_{t-1} + U_t \tag{2.3}$$

ist zum VAR(p)-Modell in Gleichung (2.1) äquivalent. Daher lassen sich Aussagen über VAR(1)-Modelle leicht auf VAR-Modelle mit p Parametermatrizen verallgemeinern.

2.2.2. Stabile VAR-Prozesse

Betrachten wir ein VAR(1)-Modell in der Notation von (2.1) ausgehend vom Zeitpunkt 1, so lässt sich mittels Induktion

$$\begin{aligned}
y_1 &= \nu + A_1 y_0 + u_1 \\
y_2 &= \nu + A_1 y_1 + u_2 = \nu + A_1(\nu + A_1 y_0 + u_1) + u_2 = \\
&= (I_K + A_1)\nu + A_1^2 y_0 + A_1 u_1 + u_2 \\
&\vdots \\
y_t &= (I_K + A_1 + \cdots + A_1^{t-1})\nu + A_1^t y_0 + \sum_{i=0}^{t-1} A_1^i u_{t-i}
\end{aligned}$$

als Darstellung der Zeitreihe y_t herleiten. Lassen wir als Ausgangszeitpunkt $t - j$ mit beliebigem $j \in \mathbb{N}$ zu, so gilt analog

$$y_t = (I_K + A_1 + \cdots + A_1^j)\nu + A_1^{j+1} y_{t-j-1} + \sum_{i=0}^{j} A_1^i u_{t-i}. \tag{2.4}$$

Vor der weiteren Diskussion dieses Resultats führen wir den Begriff der absoluten Konvergenz für Reihen von Matrizen ein.

Definition 2.2. Eine Reihe von $m \times n$ Matrizen $\sum_{k=0}^{\infty} M_k$ heißt absolut konvergent, falls für alle $1 \leq i \leq m$ und alle $1 \leq j \leq n$ die Reihe $\sum_{k=0}^{\infty} (M_k)_{i,j}$ absolut konvergent ist.

Äquivalent dazu ist die Bedingung, dass $\sum_{k=0}^{\infty} \|M_k\|$ absolut konvergent ist, wobei $\|M_k\| := \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} (M_k)_{i,j}{}^2}$ gilt. Die Äquivalenz dieser beiden Definitionen wird auf S. 687 f. in Lütkepohl, (2005) bewiesen.

Ist $\sum_{k=0}^{\infty} M_k$ mit $M_k \in \mathbb{R}^{K \times K}$ absolut konvergent und $(z_t)_{t \in \mathbb{Z}}$ eine Folge K-dimensionaler Zufallsvektoren mit $\exists c \in \mathbb{R}_+ : \forall t \in \mathbb{Z} : \mathbb{E}(z_t' z_t) \leq c$, dann existiert für alle $t \in \mathbb{Z}$ ein Zufallsvektor y_t mit $\operatorname{plim}_{l \to \infty} \sum_{k=-l}^{l} M_k z_{t-k} = y_t$. Den Beweis im Eindimensionalen erbringt Fuller, (1996) in seinem Satz 2.2.1. Die Verallgemeinerung auf K-dimensionale Zufallsvariablen ist möglich.

Damit existiert in (2.4) der (fast sicher festgelegte) Grenzwert in Wahrscheinlichkeit des letzten Summanden $\sum_{i=0}^{j} A_1^i u_{t-i}$ für $j \to \infty$ unter der Voraussetzung, dass die Reihe $\sum_{i=0}^{j} A_1^i$ absolut konvergent ist. Außerdem gilt unter dieser Voraussetzung für den ersten Summanden in (2.4) $(I_K + A_1 + A_1^2 + \cdots + A_1^j)\nu = \left(\sum_{i=0}^{j} A_1^i\right) \nu \xrightarrow{j \to \infty} (I_K - A_1)^{-1} \nu$. Die angenommene absolute Konvergenz ist erfüllt, wenn alle Eigenwerte von A_1 betragsmäßig kleiner als 1 sind. In diesem Fall verschwindet auch der zweite Summand in (2.4) für $j \to \infty$. Somit erhalten wir aus (2.4) durch Grenzwertbildung

$$y_t = \mu + \sum_{i=0}^{\infty} A_1^i u_{t-i} \tag{2.5}$$

für alle $t \in \mathbb{Z}$, wobei $\mu := (I_K - A_1)^{-1} \nu$. Diese Darstellung des Prozesses y_t wird auch Moving-Average-Darstellung genannt. Auf sie wird in Unterabschnitt 2.2.3 noch genauer eingegangen.

Für den Erwartungswert folgt

$$\mathbb{E}(y_t) = \mu \tag{2.6}$$

und als Autokovarianzfunktion erhalten wir

$$\begin{aligned} \Gamma_y(h) &:= \mathbb{E}((y_t - \mu)(y_{t-h} - \mu)') \\ &= \lim_{n \to \infty} \sum_{i=0}^{n} \sum_{j=0}^{n} A_1^i \mathbb{E}(u_{t-i} u_{t-h-j}')(A_1^j)' \\ &\overset{(*)}{=} \lim_{n \to \infty} \sum_{i=0}^{n} A_1^{h+i} \Sigma_u A_1^{i\prime} = \sum_{i=0}^{\infty} A_1^{h+i} \Sigma_u A_1^{i\prime}, \end{aligned} \tag{2.7}$$

wobei $\Sigma_u := \mathbb{E}(u_t u_t')$ unabhängig von t ist. Sowohl bei der Berechnung von (2.6) als auch bei (2.7) war die Vertauschbarkeit des Limes und des Erwartungswertes notwendig. Sie ist laut Satz 2.2.2 in Fuller, (1996) gegeben. Beim mit $(*)$ versehenen Gleichheitszeichen nutzten wir die Annahme, dass u_t weißes Rauschen darstellt und daher $\mathbb{E}(u_t u_s') = 0$ für $t \neq s$ gilt.

Die Herleitung von (2.5), $\mathbb{E}(y_t)$ sowie $\Gamma_y(h)$ ist wie oben erwähnt möglich, wenn alle Eigenwerte von A_1 im Betrag kleiner als 1 sind. Dies ist äquivalent zu $\det(\lambda I_K - A_1) \neq 0$ für $|\lambda| \geq 1$ bzw. $\det(I_K - A_1 z) \neq 0$ für $|z| \leq 1$. Diese Bedingung wird auch Stabilitätsbedingung genannt. Bei VAR(p)-Modellen lautet diese Bedingung in der Schreibweise aus (2.3) $\det\left(I_{Kp} - \boldsymbol{A}z\right) \neq 0$ für $|z| \leq 1$. In der Notation aus (2.1) schreibt sie sich als

$$\det\left(I_K - A_1 z - \cdots - A_p z^p\right) \neq 0 \tag{2.8}$$

für $|z| \leq 1$.

2.2.3. Moving-Average-Darstellung von VAR-Prozessen

Die Moving-Average-Darstellung (MA-Darstellung) für einen Prozess, der einem stabilen VAR(1)-Modell folgt, wurde bereits in Gleichung (2.5) gezeigt. Das Resultat können wir mit Hilfe der Notation aus (2.3) auf VAR(p)-Modelle verallgemeinern. Damit gilt analog zum Fall $p = 1$

$$Y_t = \boldsymbol{\mu} + \sum_{i=0}^{\infty} \boldsymbol{A}^i U_{t-i}, \tag{2.9}$$

wobei $\boldsymbol{\mu} := \left(I_{Kp} - \boldsymbol{A}\right)^{-1} \boldsymbol{\nu}$. Mit

$$J := \begin{pmatrix} I_K & 0 & \dots & 0 \end{pmatrix} \in \mathbb{R}^{K \times Kp} \tag{2.10}$$

leiten wir die MA-Darstellung von y_t her. Es gilt

$$\begin{aligned} y_t = JY_t &= J\boldsymbol{\mu} + \sum_{i=0}^{\infty} J\boldsymbol{A}^i U_{t-i} = J\boldsymbol{\mu} + \sum_{i=0}^{\infty} J\boldsymbol{A}^i J' J U_{t-i} \\ &= \mu + \sum_{i=0}^{\infty} \phi_i u_{t-i}, \end{aligned} \tag{2.11}$$

mit $\mu := J\boldsymbol{\mu}$ und

$$\phi_i := JA^iJ'. \tag{2.12}$$

Die angenommene Stabilität impliziert die absolute Konvergenz von $\sum_{i=0}^{\infty} A^i$ und liefert damit auch

$$\sum_{i=0}^{\infty} |\phi_i| < \infty. \tag{2.13}$$

Da (2.13) erfüllt ist, folgt aus (2.11)

$$E(y_t) = \mu$$

sowie

$$\begin{aligned}
\Gamma_y(h) &= \mathbb{E}\left((y_t - \mu)\,(y_{t-h} - \mu)'\right) \\
&= \mathbb{E}\left(\left(\sum_{i=0}^{h-1} \phi_i u_{t-i} + \sum_{i=0}^{\infty} \phi_{h+i} u_{t-h-i}\right)\left(\sum_{i=0}^{\infty} \phi_i u_{t-h-i}\right)'\right) \\
&= \sum_{i=0}^{\infty} \phi_{h+i} \Sigma_u \phi_i'
\end{aligned}$$

analog zu (2.6) und (2.7).

Auf S. 22 f. in Lütkepohl, (2005) wird darüber hinaus mit

$$\begin{aligned}
\phi_0 &= I_K \\
\phi_i &= \sum_{j=1}^{i} \phi_{i-j} A_j \quad \text{für } i \in \mathbb{N}
\end{aligned} \tag{2.14}$$

eine nützliche rekursive Berechnungsmethode für die MA-Koeffizienten erarbeitet.

2.2.4. Vektorwertige Fehlerkorrekturmodelle (VECMs)

Nachdem das VAR-Modell eingeführt wurde, soll in diesem Abschnitt das VECM vorgestellt werden. Die beiden Modelle sind eng verwandt. Ein Nachteil des VAR-Modells ist, dass die entsprechende Theorie oft die Stabilitätsbedingung voraussetzt. Mit dem vektorwertigen Fehlerkorrekturmodell

gelingt es, die theoretischen Aussagen auf eine Klasse von Modellen zu erweitern, welche die Stabilitätsbedingung nicht erfüllen. Wir werden jedoch stets davon ausgehen, dass Δy_t stabil ist, auch wenn dies nicht auf die betrachtete Zeitreihe y_t zutrifft. Als literarische Grundlage dieses Unterabschnitts diente Abschnitt 6.3 in Lütkepohl, (2005).

Definition 2.3. Ein vektorwertiges Fehlerkorrekturmodell (VECM) hat die Form

$$\Delta y_t = \Pi y_{t-1} + \Gamma_1 \Delta y_{t-1} + \cdots + \Gamma_{p-1} \Delta y_{t-p+1} + u_t, \tag{2.15}$$

wobei $\Pi \in \mathbb{R}^{K \times K}$ sowie $\Gamma_i \in \mathbb{R}^{K \times K}$ für $i \in \{1, 2, \ldots, p-1\}$ die Parameter des Modells sind. Der Rang der Matrix Π beträgt $r := \operatorname{rk}(\Pi) \leq K$ und wird auch Kointegrationsrang genannt. Bei $u_t \in \mathbb{R}^K$ handelt es sich wie in (2.1) um weißes Rauschen mit positiv definiter Kovarianzmatrix Σ_u.

Dieses Modell lässt sich mittels

$$\begin{aligned} A_1 &= \Pi + I_K + \Gamma_1 \\ A_i &= \Gamma_i - \Gamma_{i-1}, \quad i \in \{2, \ldots, p-1\} \\ A_p &= -\Gamma_{p-1} \end{aligned} \tag{2.16}$$

in ein VAR-Modell wie in Definition 2.1 umformen. Außerdem lässt sich jedes VAR-Modell mit

$$\begin{aligned} \Pi &= -(I_K - A_1 - \cdots - A_p) \\ \Gamma_i &= -(A_{i+1} + \cdots + A_p), \quad i \in \{1, \ldots, p-1\} \end{aligned} \tag{2.17}$$

in VECM-Schreibweise angeben.

Satz 2.4. *Eine Matrix $\Pi \in \mathbb{R}^{K \times K}$ mit Rang $0 < r \leq K$ lässt sich als Produkt $\alpha\beta'$ mit $\alpha, \beta \in \mathbb{R}^{Kxr}$ und $rk(\alpha) = rk(\beta) = r$ schreiben.*

Beweis. Π hat r linear unabhängige Zeilen. Sei $\{\beta_1, \beta_2, \ldots, \beta_r\}$ eine Basis des Zeilenraums von Π mit $\beta_j \in \mathbb{R}^K \ \forall j$. Dann lässt sich die i-te Zeile π_i von Π darstellen als $\pi_i = \sum_{j=1}^{r} \alpha_{ij} \beta_j'$, wobei $\alpha_{ij} \in \mathbb{R}$ für alle $i \in \{1, \ldots, K\}$ und $j \in \{1, \ldots, r\}$. Der Vektor β_j auf der rechten Seite wurde transponiert, da wir die Basisvektoren als Spaltenvektoren auffassen, während π_i einen

Zeilenvektor darstellt. Fassen wir nun die Basisvektoren zu einer Matrix $\beta := (\beta_1, \dots, \beta_r) \in \mathbb{R}^{K \times r}$ zusammen, ergibt sich mit $\alpha := (\alpha_{ij})_{\substack{1 \le i \le K \\ 1 \le j \le r}} \in \mathbb{R}^{K \times r}$ die gewünschte Faktorisierung

$$\Pi = \begin{pmatrix} \pi_1 \\ \vdots \\ \pi_K \end{pmatrix} = \begin{pmatrix} \alpha_{11} & \cdots & \alpha_{1r} \\ \vdots & & \vdots \\ \alpha_{K1} & \cdots & \alpha_{Kr} \end{pmatrix} \begin{pmatrix} \beta_1' \\ \vdots \\ \beta_r' \end{pmatrix} = \alpha\beta'.$$

Aus der Basiseigenschaft von β folgt $\operatorname{rk}(\beta) = r$. Aus $\operatorname{rk}(\Pi) = r$ folgt selbiges für α. □

Somit lässt sich (2.15) auch schreiben als

$$\Delta y_t = \alpha\beta' y_{t-1} + \Gamma_1 \Delta y_{t-1} + \cdots + \Gamma_{p-1} \Delta y_{t-p+1} + u_t, \tag{2.18}$$

wobei $\alpha, \beta \in \mathbb{R}^{K \times r}$ und r den Rang von α, β und Π bezeichnet. Diese Zerlegung von Π ist nicht eindeutig, denn mit einer regulären Matrix $Q \in \mathbb{R}^{r \times r}$ gilt $\Pi = \alpha Q' Q'^{-1} \beta' = (\alpha Q')(\beta Q^{-1})'$. Also sind $\alpha Q'$ und βQ^{-1} eine Alternative zu α und β. Die Nicht-Eindeutigkeit von Q erlaubt es, Restriktionen bezüglich der Form der Faktorisierung von Π festzulegen. Eine derartige Restriktion, die auch bei der Festlegung von Schätzern für α und β in Unterabschnitt 2.5.2 Anwendung finden wird, ist $\beta^* = \begin{pmatrix} I_r \\ \beta^*_{(K-r)} \end{pmatrix}$, wobei $\beta^*_{(K-r)}$ für die letzten $K - r$ Zeilen von β^* steht. Dies können wir ausgehend von $\Pi = \alpha\beta'$ mit der Wahl

$$Q = \begin{pmatrix} I_r & 0 & \dots & 0 \end{pmatrix} \beta$$

erreichen, d.h. Q besteht aus den ersten r Zeilen von β. Die so gewählte Matrix Q ist invertierbar, da β Rang r hat und wir o.B.d.A. annehmen, dass die ersten r Zeilen von β linear unabhängig sind. Dies lässt sich durch eine Umordnung der Komponenten der Zeitreihe y_t immer erreichen.

Im Fall von $r = 0$ besitzt Δy_t eine VAR$(p-1)$-Darstellung mit $\Gamma_1, \dots, \Gamma_{p-1}$ als Parametermatrizen. Dieser VAR$(p-1)$-Prozess ist laut unserer Annahme am Beginn dieses Unterabschnitts stabil. Falls $r = K$, so lässt sich mit Hilfe von (2.18) ein VAR-Modell für y_t angeben, welches wegen $|I_K - A_1 - \cdots - A_p| = |-\Pi| \neq 0$ die Stabilitätsbedingung (2.8) für $z = 1$ erfüllt.

2.3. Wahl der Modellordnung

2.3.1. Modellordnung bei VAR-Modellen

Die Gestalt der in der Praxis betrachteten Prozesse ist häufig unbekannt. Versuchen wir die beobachteten Zeitreihen mit einem VAR-Modell wie in (2.1) anzupassen, so sind in der Regel nicht nur die Parameter ν, A_i für $i \in \{1, \ldots, p\}$ und Σ_u des Modells unbekannt. Auch verfügen wir über die Ordnung p des VAR-Modells keine Information. Die Bestimmung dieser Größe ist jedoch notwendig, bevor die Schätzung der Parameter in Angriff genommen werden kann. Es gibt verschiedene Ansätze, um eine Schätzung von p vorzunehmen. Eine mögliche Vorgehensweise basiert auf statistischen Tests und wird in Abschnitt 4.2 in Lütkepohl, (2005) beschrieben. Einen weiteren Weg, p zu bestimmen, stellt die Optimierung von Informationskriterien dar.

Die zweite Vorgangsweise soll im Rahmen dieser Arbeit beschrieben werden. Sie ist auch Gegenstand von Abschnitt 4.3 in Lütkepohl, (2005), auf dem dieser Unterabschnitt basiert. Dabei werden Modelle verschiedener Ordnungen aufgestellt. Sei dazu M die maximale in Betracht zu ziehende Ordnung. Dann wird jedem VAR(m)-Modell für $m \in \{0, \ldots, M\}$ ein Wert zugeordnet, der gemäß einer Berechnungsvorschrift – dem Informationskriterium – berechnet wird. Jenes m mit dem besten der $M + 1$ auf diese Weise berechneten Werte, wird als Schätzer $\hat{p}$ für die unbekannte Modellordnung p gewählt. In dieser Arbeit werden drei verschiedene Informationskriterien angesprochen. Sie umfassen jenes nach Akaike (AIC), eine nach Hannan und Quinn benannte Größe (HQ) sowie das Bayessche Informationskriterium (BIC), das gelegentlich auch als Schwarzkriterium bezeichnet wird.

Das Akaike-Informationskriterium AIC ist definiert als

$$AIC(m) = \ln \left| \tilde{\Sigma}_u(m) \right| + \frac{2mK^2}{T}, \tag{2.19}$$

wobei $\tilde{\Sigma}_u$ den ML-Schätzer für Σ_u bezeichnet. Dieser wird in dieser Arbeit zugunsten der genaueren Diskussion des ML-Schätzers für VECM-Modelle nicht näher beleuchtet. Die Berechnung von $\tilde{\Sigma}_u$ wird auf S. 89 f. in Lütkepohl, (2005) beschrieben. Dort wird auch die Likelihood-Funktion hergeleitet, die

wie der AIC den Ausdruck $\ln|\Sigma_u(m)|$ beinhaltet – jedoch mit negativem Vorzeichen. Mit größerer Modellordnung wird das Modell besser an die Daten angepasst - die Likelihood steigt, $\ln\left|\tilde{\Sigma}_u(m)\right|$ sinkt und damit auch der AIC.

Auch die Variable T in der Berechnungsvorschrift (2.19) bedarf einer Erklärung. Dabei handelt es sich um die Anzahl der Beobachtungen abzüglich der Modellordnung m.

Nach der Berechnung des Informationskriteriums für alle $m \in \{1, \ldots, M\}$ wählen wir jenes m mit dem niedrigsten AIC(m) als Modellordnung. Während der erste Summand in (2.19) also mit zunehmendem m sinkt, steigt der Wert des zweiten Summanden. Durch diesen Strafterm soll die Wahl eines zu großen Modells vermieden werden. Der in diesem Summanden vorkommende Faktor mK^2 ist die Anzahl aller geschätzten Elemente in den Parametermatrizen. Etwaige zu schätzende deterministische Terme im Modell fließen nicht in den AIC ein. Sie würden lediglich eine Verschiebung um eine Konstante bewirken und hätten somit keine Auswirkungen auf $\operatorname{argmin}_m \mathrm{AIC}(m)$.

Das Verfahren ändert sich bei Verwendung eines anderen Informationskriteriums nicht. Auch mit einem anderen Kriterium IC wird $\operatorname{argmin}_m \mathrm{IC}(m)$ als VAR-Ordnung gewählt. Eine Alternative zum AIC stellt das Informationskriterium nach Hannan und Quinn (HQ) dar. Es wird durch

$$HQ(m) = \ln\left|\tilde{\Sigma}_u(m)\right| + \frac{2\ln\ln T}{T} mK^2$$

definiert. Das Bayessche Informationskriterium hat die Form

$$BIC(m) = \ln\left|\tilde{\Sigma}_u(m)\right| + \frac{\ln T}{T} mK^2.$$

Proposition 4.2 in Lütkepohl, (2005) liefert eine notwendige und hinreichende Bedingung für die Konsistenz eines Schätzers für die Modellordnung. Ein Schätzer $\hat{p}$ für p heißt konsistent, wenn für alle $\epsilon > 0$

$$\lim_{T\to\infty} \mathbb{P}\left[|\hat{p} - p| < \epsilon\right] = 1 \tag{2.20}$$

gilt, wobei $\mathbb{P}$ ein Wahrscheinlichkeitsmaß bezeichnet. Da $\hat{p}$ ein ganzzahliger Schätzer ist, ist (2.20) äquivalent zu

$$\lim_{T\to\infty} \mathbb{P}\left[\hat{p} = p\right] = 1.$$

Aus der erwähnten Proposition folgt, dass sowohl die Verwendung von HQ als auch von BIC zu konsistenten Schätzern für die Modellordnung führt. Dagegen liefert das Verfahren mit dem AIC keinen konsistenten Schätzer. Sei in der Folge $\hat{p}(IC)$ der Schätzer für p, der aus der Verwendung des Informationskriteriums IC resultiert. Da $\ln(T) \geq 2\ln(\ln(T)) \geq 2$ für $T \geq 16$ gilt, ist der Strafterm für zusätzliche Parameter für $T \geq 16$ unter den von uns betrachteten Informationskriterien beim BIC am größten und beim AIC am kleinsten. Daraus lässt sich die Beziehung

$$\hat{p}(BIC) \leq \hat{p}(HQ) \leq \hat{p}(AIC)$$

herleiten (vgl. S. 151 f. in Lütkepohl, (2005)).

Aus diesen Ungleichungen folgt in Verbindung mit der Tatsache, dass der AIC zu keinem konsistenten Schätzer führt, dass

$$\lim \mathbb{P}[\hat{p}(AIC) > p] > 0$$

gilt. Auch wenn wir somit mit dem AIC den wahren Wert asymptotisch mit positiver Wahrscheinlichkeit überschätzen, warnt Lütkepohl, (2005) auf S. 151 davor, die anderen beiden Informationskriterien dem AIC immer vorzuziehen. Demnach können für kleines T oder wenn Vorhersagen das Ziel sind, mit dem AIC unter Umständen bessere Resultate erzielt werden als mit HQ oder BIC.

2.3.2. Modellordnung bei VECMs

Jedem VECM($p-1$) entspricht wegen (2.18) ein VAR(p)-Modell. Wie in Abschnitt 8.1 in Lütkepohl, (2005) beschrieben, erbt das VECM das Verfahren zur Ermittlung der Modellordnung vom VAR -Modell.

Um die Modellordnung $p-1$ eines VECMs zu ermitteln, wird es zunächst in das entsprechende VAR(p)-Modell übersetzt. Von diesem ausgehend wird

wie in Unterabschnitt 2.3.1 der Wert des interessierenden Informationskriteriums für verschiedene Modellordnungen m berechnet und jenes m gewählt, für welches das Informationskriterium den geringsten Wert liefert. Die somit gefundene Modellordnung p gilt für das VAR-Modell. Damit ist auch die Modellordnung $p-1$ des VECMS gefunden. Zu beachten ist, dass bei diesem Verfahren nicht wie oben $m \in \{0,\dots,M\}$ mit $M \in \mathbb{N}$ gilt, sondern $m \in \{1,\dots,M+1\}$, um $p-1 \in \{0,\dots,M\}$ sicherzustellen. Aus Proposition 8.1 in Lütkepohl, (2005) folgt, dass auch im nicht stabilen Fall sowohl HQ als auch BIC konsistente Schätzer für die Modellordnung liefern, während dies auf das Informationskriterium AIC nicht zutrifft.

2.4. Bestimmung des Kointegrationsranges

Neben der Modellordnung $p-1$ setzt die Schätzung der Parameter eines VECMS die Bestimmung des Kointegrationsranges $r := \mathrm{rk}(\Pi)$ voraus. Diese ist Gegenstand dieses Abschnittes, dessen Grundlage Abschnitt 8.2 in Lütkepohl, (2005) bildet.

Wir werden davon ausgehen, dass die Modellordnung bereits gewählt ist. Die Festlegung von r basiert in der Praxis häufig auf statistischen Tests der Form

$$H_0 : \mathrm{rk}(\Pi) = r_0 \quad \text{gegen} \quad H_1 : r_0 < \mathrm{rk}(\Pi) \leq r_1,$$

wobei z.B. $r_0 + 1$ oder K mögliche Werte für r_1 darstellen.

Das Verfahren zur Ermittlung von r beginnt mit dem Test der Nullhypothese $\mathrm{rk}(\Pi) = 0$ gegen die Alternativhypothese H_1. Kann H_0 nicht verworfen werden, so ist für die weitere Analyse (Parameterschätzung, Vorhersagen, etc.) ein Kointegrationsrang von 0 anzunehmen. Das bedeutet auch, dass $\Pi = 0$ gilt und somit Δy_t mit einem VAR$(p-1)$-Modell angepasst wird.

Wird die Nullhypothese jedoch verworfen, so formulieren wir eine neue Nullhypothese $H_0 : \mathrm{rk}(\Pi) = 1$. Wenn H_0 dieses Mal nicht abgelehnt werden kann, wird $r = 1$ angenommen. Im Falle des Verwerfens von H_0 setzt sich das Verfahren mit einem stets um 1 wachsenden r_0 fort, bis eine Nullhypothese beibehalten wird (in diesem Fall wird das in der letzten Nullhypothese

enthaltene r_0 als Kointegrationsrang angenommen). Falls alle Nullhypothesen $H_0 : \mathrm{rk}(\Pi) = i$ mit $i = 0, \ldots, K-1$ abgelehnt werden, wird $r = K$ gesetzt.

Das Verfahren setzt keinen bestimmten Wert r_1 in der Alternativhypothese voraus. Je nach Wahl von r_1 ändert sich aber die Teststatistik, die über Annahme oder Ablehnung der Nullhypothese entscheidet. Um eine Teststatistik herzuleiten, betrachten wir die maximale Loglikelihood l_r eines VECMS bei gegebenem Kointegrationsrang r. Sie kann laut S. 295 in Lütkepohl, (2005) mit

$$\max l_r = -\frac{KT}{2}\ln(2\pi) - \frac{T}{2}\left[\ln(|S_{00}|) + \sum_{i=1}^{r}\ln(1-\lambda_i)\right] - \frac{KT}{2} \tag{2.21}$$

angegeben werden, wobei S_{00} und λ_i in Satz 2.5 definiert werden. Damit folgt für die Likelihood L_r bei gegebenem Kointegrationsrang r

$$\begin{aligned}\max L_r &= (2\pi)^{-\frac{KT}{2}}\, \mathrm{e}^{-\frac{T}{2}\left[\ln(|S_{00}|)+\sum_{i=1}^{r}\ln(1-\lambda_i)\right]}\, \mathrm{e}^{-\frac{KT}{2}} \\ &= (2\pi)^{-\frac{KT}{2}} \left(|S_{00}| \prod_{i=1}^{r}(1-\lambda_i)\right)^{-\frac{T}{2}} \mathrm{e}^{-\frac{KT}{2}}.\end{aligned}$$

Daraus erhalten wir die Likelihood-Ratio

$$\begin{aligned}\frac{L_{r_0}}{L_{r_1}} &= \frac{(2\pi)^{-\frac{KT}{2}} \left(|S_{00}| \prod_{i=1}^{r_0}(1-\lambda_i)\right)^{-\frac{T}{2}} \mathrm{e}^{-\frac{KT}{2}}}{(2\pi)^{-\frac{KT}{2}} \left(|S_{00}| \prod_{i=1}^{r_1}(1-\lambda_i)\right)^{-\frac{T}{2}} \mathrm{e}^{-\frac{KT}{2}}} \\ &= \left(\prod_{i=r_0+1}^{r_1}(1-\lambda_i)\right)^{-\frac{T}{2}}.\end{aligned}$$

Lütkepohl, (2005) betrachtet anstelle dieser Größe

$$\begin{aligned}\lambda_{LR}(r_0, r_1) &:= 2\ln\left(\frac{L_{r_0}}{L_{r_1}}\right) = 2\cdot\left(-\frac{T}{2}\right)\ln\left(\prod_{i=r_0+1}^{r_1}(1-\lambda_i)\right) \\ &= -T\sum_{i=r_0+1}^{r_1}\ln(1-\lambda_i)\end{aligned} \tag{2.22}$$

und bezeichnet sie als LR-Statistik. In (2.22) ist die Abhängigkeit der Teststatistik von r_1 und damit von der Alternativhypothese offensichtlich. Für $r_1 = r_0 + 1$ nimmt sie die Form

$$\lambda_{LR}(r_0, r_0 + 1) = -T \ln(1 - \lambda_{r_0+1}) \tag{2.23}$$

an, während für $r_1 = K$

$$\lambda_{LR}(r_0, K) = -T \sum_{i=r_0+1}^{K} \ln(1 - \lambda_i) \tag{2.24}$$

gilt. Lütkepohl, (2005) weist darauf hin, dass für keine Teststatistik $\lambda_{LR}(r_0, r_1)$ die exakte dazugehörige Verteilung bekannt ist. Approximative kritische Bereiche für $\lambda_{LR}(r_0, r_0 + 1)$ und $\lambda_{LR}(r_0, K)$ wurden unter anderem in MacKinnon et al., (1999), Johansen und Schaumburg, (1999) und MacKinnon, (2010) veröffentlicht. Dort werden sie in verschiedenen Tabellen abgebildet – in Abhängigkeit vom Signifikanzniveau, der Dimension K der Zeitreihe und gesondert nach den im Modell enthaltenen deterministischen Termen. Die Ergebnisse in MacKinnon, (2010) gehen dabei sogar über asymptotische Resultate hinaus. Mittels Regression gewann er mit seiner Gleichung (A.1) auf S. 11 eine Formel, die es erlaubt, kritische Bereiche für jede Zeitreihenlänge T zu berechnen.

2.5. Parameterschätzung

2.5.1. Parameterschätzung bei VAR-Modellen

In diesem Abschnitt geben wir wichtige Resultate für die Parameterschätzung eines stabilen VAR(p)-Modells wieder. Eine Herleitung der hier angeführten Ergebnisse findet sich in den Unterabschnitten 3.2.1 sowie 3.2.2 in Lütkepohl, (2005).

Für die Schätzung der Parameter in (2.1) fassen wir diese zu einem Vektor

$$\beta := \operatorname{vec}(\nu, A_1, \ldots, A_p) \tag{2.25}$$

zusammen. Unter Verwendung der Symbole

$$\boldsymbol{y} := \operatorname{vec}(y_1, \ldots, y_T) \quad \text{und}$$

$$Z := (Z_0, \ldots, Z_{T-1}) \quad \text{mit } Z_t := \begin{pmatrix} 1 \\ y_t \\ \vdots \\ y_{t-p+1} \end{pmatrix}$$

lässt sich mit Rechenregel (A.5)

$$\hat{\beta} = \left((ZZ')^{-1} \otimes \Sigma_u \right) \left(Z \otimes \Sigma_u^{-1} \right) \boldsymbol{y} \overset{(A.5)}{=} \left((ZZ')^{-1} Z \otimes I_K \right) \boldsymbol{y} \tag{2.26}$$

als Least-Squares-Schätzer herleiten. Für diesen gilt

$$\sqrt{T} \left(\hat{\beta} - \beta \right) \xrightarrow{d} N \left(0, \Gamma^{-1} \otimes \Sigma_u \right), \tag{2.27}$$

wobei $\Gamma := \operatorname{plim} ZZ'/T$. Ein unverzerrter Schätzer für Σ_u ist

$$\hat{\Sigma}_u = \frac{1}{T - Kp - 1} \sum_{t=1}^{T} \hat{u}_t \hat{u}_t' \tag{2.28}$$

mit den aus der Schätzung resultierenden Residuen $\hat{u}_t$, während

$$\hat{\Gamma} = \frac{ZZ'}{T} \tag{2.29}$$

einen Schätzer für Γ darstellt. Somit können wir die Kovarianzmatrix in (2.27) schätzen, indem wir $\hat{\Gamma}$ und $\hat{\Sigma}_u$ anstelle von Γ und Σ_u verwenden.

2.5.2. Parameterschätzung bei VECMs

In diesem Abschnitt soll die Anpassung eines VECMs an beobachtete Daten beleuchtet werden. Wir nehmen in der Folge an, dass Beobachtungen y_t zu den Zeitpunkten $t = -p+2, -p+1, \ldots, T$ vorliegen. Die ersten $p-1$ Beobachtungen $y_{-p+2}, \ldots, y_0$ werden in der Literatur auch als presample bezeichnet. Es gibt verschiedene Methoden, die Parameter eines VECMs zu

schätzen. Im Rahmen dieser Arbeit soll der Maximum-Likelihood-Ansatz (ML) vorgestellt werden. Dazu nehmen wir an, dass $u_t \sim N(0, \Sigma_u)$ gilt. Damit handelt es sich bei y_t um einen Gauß'schen Prozess. Weitere Schätzmethoden wie etwa Least Squares (LS), Estimated Generalized Least Squares (EGLS) und zweistufige Verfahren werden in den Unterabschnitten 7.2.1, 7.2.2 sowie 7.2.5 in Lütkepohl, (2005) diskutiert. Die hier besprochene ML-Schätzung basiert auf Unterabschnitt 7.2.3 in Lütkepohl, (2005).

Mit der Notation

$$\begin{aligned}
\Delta Y &:= (\Delta y_1, \ldots, \Delta y_T), \\
Y_{-1} &:= (y_0, \ldots, y_{T-1}), \\
\Gamma &:= (\Gamma_1, \ldots, \Gamma_{p-1}), \\
\Delta X &:= (\Delta X_0, \ldots, \Delta X_{T-1}) \text{ mit } \Delta X_{t-1} := \begin{pmatrix} \Delta y_{t-1} \\ \vdots \\ \Delta y_{t-p+1} \end{pmatrix} \text{ und} \\
U &:= (u_1, \ldots, u_T)
\end{aligned}$$

lässt sich (2.15) als

$$\Delta Y = \Pi Y_{-1} + \Gamma \Delta X + U \tag{2.30}$$

darstellen.

Satz 2.5. *Wir definieren die Matrizen* $M := I_T - \Delta X'(\Delta X \Delta X')^{-1}\Delta X$, $R_0 := \Delta Y M$, $R_1 := Y_{-1} M$ *sowie* $S_{ij} := R_i R_j'/T$ *für* $i \in \{0,1\}$. *Seien* $\lambda_1 \geq \cdots \geq \lambda_K$ *die Eigenwerte von* $S_{11}^{-1/2} S_{01} S_{00}^{-1} S_{01} S_{11}^{-1/2}$ *und* $v_1, \ldots, v_K$ *die dazugehörigen orthonormalen Eigenvektoren. Dann lauten die ML-Schätzer für die* VECM*-Parameter*

$$\beta = \tilde{\beta} := (v_1, \ldots, v_r)' S_{11}^{-1/2}, \tag{2.31}$$

$$\alpha = \tilde{\alpha} := \Delta Y M Y_{-p}' \tilde{\beta} (\tilde{\beta}' Y_{-1} M Y_{-1}' \tilde{\beta})^{-1} = S_{01} \tilde{\beta} (\tilde{\beta}' S_{11} \tilde{\beta})^{-1}, \tag{2.32}$$

$$\Gamma = \tilde{\Gamma} := (\Delta Y - \tilde{\alpha} \tilde{\beta}' Y_{-1}) \Delta X' (\Delta X \Delta X')^{-1}, \tag{2.33}$$

$$\Sigma_u = \tilde{\Sigma}_u := \frac{1}{T} (\Delta Y - \tilde{\alpha} \tilde{\beta}' Y_{-1} - \tilde{\Gamma} D X)(\Delta Y - \tilde{\alpha} \tilde{\beta}' Y_{-1} - \tilde{\Gamma} D X)'. \tag{2.34}$$

Beweis. Seien $\boldsymbol{\Delta y} := \text{vec}(\Delta Y)$, $\boldsymbol{y_{-1}} := \text{vec}(Y_{-1})$, $\boldsymbol{u} := \text{vec}(U)$ und $\boldsymbol{\Gamma} := \text{vec}(\Gamma)$. Es gilt $\boldsymbol{u} \sim N(0, I_T \otimes \Sigma_u)$ und damit

$$\begin{aligned} f_{\boldsymbol{u}}(\boldsymbol{u}) &= \left(\frac{1}{\sqrt{2\Pi}}\right)^{KT} \frac{1}{\sqrt{|I_T \otimes \Sigma_u|}} \cdot \exp\left[-\frac{1}{2}\boldsymbol{u}'(I_T \otimes \Sigma_u)^{-1}\boldsymbol{u}\right] \\ &= \left(\frac{1}{\sqrt{2\Pi}}\right)^{KT} \frac{1}{\sqrt{|I_T \otimes \Sigma_u|}} \cdot \exp\left[-\frac{1}{2}\boldsymbol{u}'(I_T \otimes \Sigma_u^{-1})\boldsymbol{u}\right], \end{aligned}$$

wobei wir für die letzte Umformung Regel (A.3) in Anhang A und $I_T^{-1} = I_T$ nutzten. Der Vektor $\boldsymbol{u}$ lässt sich mittels

$$\begin{aligned} \boldsymbol{u} &= \begin{pmatrix} \Delta y_1 \\ -\Gamma_1 \Delta y_1 + \Delta y_2 \\ \vdots \\ -\Gamma_{p-1}\Delta y_1 - \cdots - \Gamma_1 \Delta y_{p-1} + \Delta y_p \\ \vdots \\ -\Gamma_{p-1}\Delta y_{T-p+1} - \cdots - \Gamma_1 \Delta y_{T-1} + \Delta y_T \end{pmatrix} + \begin{pmatrix} -\alpha\beta' y_0 \\ \vdots \\ -\alpha\beta' y_{T-1} \end{pmatrix} + P_2 \\ &= P_1 \boldsymbol{\Delta y} - (I_T \otimes \alpha\beta')\boldsymbol{y_{-1}} + P_2 \end{aligned}$$

zerlegen, wobei

$$P_1 := \begin{pmatrix} I_K & 0 & & & \cdots & & & 0 \\ -\Gamma_1 & \ddots & \ddots & & & & & \\ -\Gamma_2 & -\Gamma_1 & \ddots & \ddots & & & & \\ \vdots & & \ddots & \ddots & \ddots & & & \vdots \\ -\Gamma_{p-1} & & & \ddots & \ddots & \ddots & & \\ 0 & \ddots & & & \ddots & \ddots & \ddots & \\ \vdots & \ddots & \ddots & & & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -\Gamma_{p-1} & \cdots & \cdots & -\Gamma_1 & I_K \end{pmatrix} \text{ und}$$

$$P_2 := \begin{pmatrix} -\Gamma_1\Delta y_0 - \cdots - \Gamma_{p-1}\Delta y_{-p+2} \\ -\Gamma_2\Delta y_0 - \Gamma_{p-1}y_{-p+3} \\ \vdots \\ \Gamma_{p-1}\Delta y_0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Unter Verwendung von $-\alpha\beta' y_t = -\alpha\beta'(\Delta y_t + y_{t-1}) = -\alpha\beta'(\Delta y_t + \Delta y_{t-1} + y_{t-2}) = \ldots = -\alpha\beta'(\sum_{i=1}^{t}\Delta y_i + y_0)$ können wir dies umschreiben zu

$$\boldsymbol{u} = \bar{P}_1\boldsymbol{\Delta y} + \bar{P}_2, \tag{2.35}$$

wobei

$$\bar{P}_1 := \begin{pmatrix} I_K & 0 & & & \cdots & & 0 \\ -\bar{\Gamma}_1 & \ddots & \ddots & & & & \\ -\bar{\Gamma}_2 & -\bar{\Gamma}_1 & \ddots & \ddots & & & \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ -\bar{\Gamma}_{p-1} & & & \ddots & \ddots & \ddots & \\ -\alpha\beta' & \ddots & & & \ddots & \ddots & \ddots & \\ \vdots & \ddots & \ddots & & & \ddots & \ddots & 0 \\ -\alpha\beta' & \ldots & -\alpha\beta' & -\bar{\Gamma}_{p-1} & \ldots & \ldots & -\bar{\Gamma}_1 & I_K \end{pmatrix} \tag{2.36}$$

mit $\bar{\Gamma}_i := \Gamma_i - \alpha\beta'$ für $i = 1, \ldots, p-1$ und

$$\bar{P}_2 := P_2 - \begin{pmatrix} \alpha\beta' y_0 \\ \vdots \\ \alpha\beta' y_0 \end{pmatrix}.$$

In (2.35) hängt der erste Summand von $\boldsymbol{\Delta y}$ ab und der zweite Summand vom presample, dessen Werte $y_{-p+1}, \ldots, y_0$ als gegeben betrachtet werden. Daher gilt $\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{\Delta y}'} = \bar{P}_1$ und damit $f_{\boldsymbol{\Delta y}}(\boldsymbol{\Delta y}) = \left|\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{\Delta y}'}\right| f_{\boldsymbol{u}}(\boldsymbol{u}) = |\bar{P}_1| f_{\boldsymbol{u}}(\boldsymbol{u})$. Da $\bar{P}_1$ eine untere Dreiecksmatrix ist, ist ihre Determinante das Produkt der

Diagonalelemente. Somit gilt $\left|\frac{\partial u}{\partial \Delta y'}\right| = 1$. Daraus folgt

$$f_{\Delta y}(\Delta y) = \left(\frac{1}{\sqrt{2\Pi}}\right)^{KT} \frac{1}{\sqrt{|I_T \otimes \Sigma_u|}} \cdot \exp\Bigg[-\frac{1}{2} \underbrace{\left(\Delta y - (I_T \otimes \alpha\beta')y_{-1} - (\Delta X' \otimes I_K)\Gamma\right)}_{=u}' \left(I_T \otimes \Sigma_u^{-1}\right) \underbrace{\left(\Delta y - (I_T \otimes \alpha\beta')y_{-1} - (\Delta X' \otimes I_K)\Gamma\right)}_{=u} \Bigg].$$

Aus dieser Dichtefunktion folgt für die Loglikelihoodfunktion

$$l := \ln(L(\alpha, \beta, \Gamma, \Sigma_u))$$

die Darstellung

$$\begin{aligned} l = &-\frac{KT}{2}\ln(2\Pi) - \frac{1}{2}\ln(|I_T \otimes \Sigma_u|) \\ &-\frac{1}{2}\left(\Delta y - (I_T \otimes \alpha\beta')y_{-1} - (\Delta X' \otimes I_K)\Gamma\right)' \left(I_T \otimes \Sigma_u^{-1}\right) \\ &\left(\Delta y - (I_T \otimes \alpha\beta')y_{-1} - (\Delta X' \otimes I_K)\Gamma\right). \end{aligned} \tag{2.37}$$

Wir werden zuerst den ML-Schätzer für Γ herleiten. Dazu bilden wir die entsprechende Ableitung

$$\begin{aligned} \frac{\partial l}{\partial \Gamma} = &-(\Delta X' \otimes I_K)'(I_T \otimes \Sigma_u^{-1})(\Delta y - (I_T \otimes \alpha\beta')y_{-1}) \\ &+(\Delta X' \otimes I_K)'(I_T \otimes \Sigma_u^{-1})(\Delta X' \otimes I_K)\Gamma \\ \overset{(A.4)}{=} &(\Delta X \otimes I_K)(I_T \otimes \Sigma_u^{-1})[-\Delta y + (I_T \otimes \alpha\beta')y_{-1} + (\Delta X' \otimes I_K)\Gamma] \\ \overset{(A.5)}{=} &\left((\Delta X \cdot I_T) \otimes (I_K \cdot \Sigma_u^{-1})\right)[-\Delta y + (I_T \otimes \alpha\beta')y_{-1} + (\Delta X' \otimes I_K)\Gamma] \\ = &(\Delta X \otimes \Sigma_u^{-1})[-\Delta y + (I_T \otimes \alpha\beta')y_{-1} + (\Delta X' \otimes I_K)\Gamma] \end{aligned}$$

und setzen diese 0, um die Loglikelihood zu maximieren. Daraus erhalten wir

$$\begin{aligned} (\Delta X \otimes \Sigma_u^{-1})[\Delta y - (I_T \otimes \alpha\beta')y_{-1}] &= (\Delta X \otimes \Sigma_u^{-1})(\Delta X' \otimes I_K)\Gamma \\ &\overset{(A.5)}{=} ((\Delta X \cdot \Delta X') \otimes \Sigma_u^{-1})\Gamma. \end{aligned}$$

Mit den Regeln (A.7) und (A.8) folgt

$$\text{vec}(\Sigma_u^{-1}[\Delta Y - \alpha\beta' Y_{-1}]\Delta X') = \text{vec}(\Sigma_u^{-1}\Gamma\Delta X'\Delta X)$$

und somit gilt $\Sigma_u^{-1}[\Delta Y - \alpha\beta' Y_{-1}]\Delta X' = \Sigma_u^{-1}\Gamma\Delta X'\Delta X$. Da Σ_u^{-1} invertierbar ist, muss

$$[\Delta Y - \alpha\beta' Y_{-1}]\Delta X' = \Gamma\Delta X'\Delta X$$

erfüllt sein. Damit erhalten wir in Abhängigkeit von α und β den ML-Schätzer für Γ

$$\tilde{\Gamma}(\alpha\beta') = (\Delta Y - \alpha\beta' Y_{-1})\Delta X'(\Delta X\Delta X')^{-1}.$$

Damit ist (2.33) gezeigt.

Das VECM in der Notation aus (2.30) lautet mit dem geschätzten Parameter Γ

$$\Delta Y = \alpha\beta' Y_{-1} + (\Delta Y - \alpha\beta' Y_{-1})\Delta X'(\Delta X\Delta X')^{-1}\Delta X + U$$

bzw.

$$(\Delta Y - \alpha\beta' Y_{-1})(I_T - \Delta X'(\Delta X\Delta X')^{-1}\Delta X) = (\Delta Y - \alpha\beta' Y_{-1})M = U,$$

wobei $M := (I_T - \Delta X'(\Delta X\Delta X')^{-1}\Delta X)$. Dieses Resultat können wir nun in die Loglikelihood-Funktion einsetzen. Davor merken wir an, dass (2.37) unter Verwendung von

$$\begin{aligned} |I_T \otimes \Sigma_u| &\overset{(A.6)}{=} |\Sigma_u|^T, \\ (I_T \otimes \alpha\beta')\boldsymbol{y_{-1}} &\overset{(A.8)}{=} \text{vec}(\alpha\beta' Y_{-1}), \\ (\Delta X' \otimes I_K)\boldsymbol{\Gamma} &\overset{(A.8)}{=} \text{vec}(\Gamma\Delta X) \end{aligned}$$

und (A.9) auch als

$$\begin{aligned} l = &-\frac{KT}{2}\ln(2\Pi) - \frac{T}{2}\ln(|\Sigma_u|) \\ &-\frac{1}{2}\text{tr}\left((\Delta Y - \alpha\beta' Y_{-1} - \Gamma\Delta X)'\Sigma_u^{-1}(\Delta Y - \alpha\beta' Y_{-1} - \Gamma\Delta X)\right) \end{aligned} \tag{2.38}$$

geschrieben werden kann. Somit folgt

$$\begin{aligned} l = &- \frac{KT}{2}\ln(2\Pi) - \frac{T}{2}\ln(|\Sigma_u|) \\ &- \frac{1}{2}\mathrm{tr}\left(\left((\Delta Y - \alpha\beta' Y_{-1})M\right)'\Sigma_u^{-1}(\Delta Y - \alpha\beta' Y_{-1})M\right). \end{aligned} \tag{2.39}$$

Diese Funktion möchten wir nun in den verbliebenen drei Parametern maximieren. Zunächst leiten wir dazu die Loglikelihood-Funktion nach Σ_u ab

$$\begin{aligned} \frac{\partial l}{\partial \Sigma_u} \underset{(A.10)}{\overset{(A.11)}{=}} &- \frac{T}{2}\Sigma_u^{-1} \\ &+ \frac{1}{2}\Sigma_u^{-1}((\Delta Y - \alpha\beta' Y_{-1})M)((\Delta Y - \alpha\beta' Y_{-1})M)'\Sigma_u^{-1} \end{aligned}$$

und setzen diese Ableitung gleich 0. Dies wird von

$$\Sigma_u^{-1} = T\left(\left((\Delta Y - \alpha\beta' Y_{-1})M\right)\left((\Delta Y - \alpha\beta' Y_{-1})M\right)'\right)^{-1}$$

bzw.

$$\tilde{\Sigma}_u(\alpha\beta') = \frac{1}{T}\left((\Delta Y - \alpha\beta' Y_{-1})M\right)\left((\Delta Y - \alpha\beta' Y_{-1})M\right)'$$

erfüllt und (2.34) ist gezeigt. Damit müssen wir die Loglikelihoodfunktion in den beiden Parametern α und β maximieren. Durch die Wahl von $\Sigma_u = \tilde{\Sigma}_u$ wird wegen (A.1) der Wert der Spur in (2.39) festgelegt. Nur

$$\begin{aligned} -\frac{T}{2}\ln(|\tilde{\Sigma}_u|) = -\frac{T}{2}\ln\Big(\Big|\frac{1}{T}&\left(\Delta Y M - \alpha\beta' Y_{-1}M\right) \\ &\left(\Delta Y M - \alpha\beta' Y_{-1}M\right)'\Big|\Big) \end{aligned}$$

ist von α und β abhängig. Die noch zu zeigenden Gleichungen (2.31) und (2.32) folgen aus Lemma A.5. □

Die in den Gleichungen (2.31) und (2.32) angegebenen ML-Schätzer für α und β sind nicht eindeutig. Mit einer regulären Matrix $Q \in \mathbb{R}^{r\times r}$ maximieren auch die Schätzer $\bar{\alpha} := \tilde{\alpha}Q^{-1}$, $\bar{\beta} := \tilde{\beta}Q'$ die Likelihood. Dies folgt aus

$\bar{\alpha}\bar{\beta}' = \tilde{\alpha}Q^{-1}(\tilde{\beta}Q')' = \tilde{\alpha}Q^{-1}Q\tilde{\beta}' = \tilde{\alpha}\tilde{\beta}'$. Eindeutigkeit erhalten wir unter der zusätzlichen Bedingung, dass die ersten r Zeilen von β die Einheitsmatrix bilden, also

$$\beta = \begin{pmatrix} I_r \\ \beta_{(K-r)} \end{pmatrix}, \tag{2.40}$$

wobei $\beta_{(K-r)}$ die letzten $K-r$ Zeilen von β darstellen.

Für die ML-Schätzer können die folgenden asymptotischen Eigenschaften angegeben werden. Es gilt

$$\sqrt{T}\text{vec}\left(\begin{pmatrix}\tilde{\alpha}\tilde{\beta}' & \tilde{\Gamma}\end{pmatrix} - \begin{pmatrix}\Pi & \Gamma\end{pmatrix}\right) \xrightarrow{d} N\left(0, \Sigma_{co}\right)$$

sowie

$$\sqrt{T}\text{vech}\left(\tilde{\Sigma}_u - \Sigma_u\right) \xrightarrow{d} N\left(0, 2\boldsymbol{D}_K^{+}(\Sigma_u \otimes \Sigma_u)\boldsymbol{D}_K^{+\prime}\right).$$

Dabei sind

$$\Sigma_{co} := \left(\begin{pmatrix}\beta & 0 \\ 0 & I_{Kp-K}\end{pmatrix}\Omega^{-1}\begin{pmatrix}\beta' & 0 \\ 0 & I_{Kp-K}\end{pmatrix}\right) \otimes \Sigma_u, \tag{2.41}$$

mit

$$\Omega := \text{plim}\,\frac{1}{T}\begin{pmatrix}\beta' Y_{-1}Y_{-1}'\beta & \beta' Y_{-1}\Delta X' \\ \Delta X Y_{-1}'\beta & \Delta X \Delta X'\end{pmatrix} \tag{2.42}$$

und $\boldsymbol{D}_K^{+} := (\boldsymbol{D}_K'\boldsymbol{D}_K)^{-1}\boldsymbol{D}_K'$, wobei $\boldsymbol{D}_K$ die Duplikationsmatrix darstellt, welche für eine symmetrische $K \times K$-Matrix A die Bedingung

$$\text{vec}(A) = \boldsymbol{D}_K\text{vech}(A)$$

erfüllt.

2.5.3. VECMs mit deterministischen Termen

Das in Definition 2.3 eingeführte VECM lässt sich um deterministische Terme erweitern. Dies wird in Abschnitt 6.4 sowie Unterabschnitt 7.2.4 in Lütkepohl, (2005), welchen dieser Unterabschnitt folgt, vorgeführt. In diesem Zusammenhang erweist sich die Schreibweise

$$y_t = \mu_t + x_t$$

als nützlich, wobei x_t ein VECM ohne deterministische Terme darstellt und μ_t ein rein deterministischer Prozess ist. Bei μ_t kann es sich etwa um eine Konstante handeln oder um einen linearen Trend. Außerdem eröffnet μ_t die Möglichkeit, saisonale Effekte in das Modell einzuschleusen.

Betrachten wir ein VECM mit einem konstanten deterministischen Term $\mu_t = c$. In diesem Fall gilt $\Delta\mu_t = 0$ und daher

$$\begin{aligned}\Delta y_t = \Delta x_t &= \alpha\beta' x_{t-1} + \Gamma_1 \Delta x_{t-1} + \cdots + \Gamma_{p-1}\Delta x_{t-p+1} + u_t \\ &= \alpha\beta'(y_{t-1} - c) + \Gamma_1 \Delta y_{t-1} + \cdots + \Gamma_{p-1}\Delta y_{t-p+1} + u_t \\ &= \alpha \begin{pmatrix}\beta' & -\beta c\end{pmatrix} \begin{pmatrix} y_{t-1} \\ 1 \end{pmatrix} + \Gamma_1 \Delta y_{t-1} + \cdots + \Gamma_{p-1}\Delta y_{t-p+1} + u_t.\end{aligned} \tag{2.43}$$

Wir sehen, dass sich der deterministische Term nur auf die Kointegrationsbeziehung $\alpha\beta' y_{t-1}$ auswirkt. Im Fall eines linearen Trends $\mu_t = \mu_0 + t\mu_1$ erhalten wir $\Delta\mu_t = \mu_1$ und damit

$$\begin{aligned}\Delta y_t - \mu_1 = \Delta x &= \alpha\beta'(y_{t-1} - \mu_0 - (t-1)\mu_1) \\ &\quad + \Gamma_1(\Delta y_{t-1} - \mu_1) + \cdots + \Gamma_{p-1}(\Delta y_{t-p+1} - \mu_1) + u_t \\ &= \alpha \begin{pmatrix}\beta' & -\beta'\mu_0 & -\beta'\mu_1\end{pmatrix} \begin{pmatrix} y_{t-1} \\ 1 \\ t-1 \end{pmatrix} \\ &\quad + \Gamma_1 \Delta y_{t-1} + \cdots + \Gamma_{p-1}\Delta y_{t-p+1} + u_t \\ &\quad + \left(I_K - \Gamma_1 - \cdots - \Gamma_{p-1}\right)\mu_1.\end{aligned}$$

In diesem Fall verändert sich das Modell nicht nur im ersten Summanden, sondern erhält mit $\left(I_K - \Gamma_1 - \cdots - \Gamma_{p-1}\right)\mu_1$ noch einen zusätzlichen Effekt. Eine Verallgemeinerung der beiden angesprochenen Fälle erreichen wir, indem wir das Modell als

$$\Delta y_t = \alpha \begin{pmatrix}\beta' & \eta'\end{pmatrix} \begin{pmatrix} y_{t-1} \\ D^{co}_{t-1} \end{pmatrix} + \Gamma_1 \Delta y_{t-1} + \cdots + \Gamma_{p-1}\Delta y_{t-p+1} + CD_t + u_t \tag{2.44}$$

schreiben. Dabei steht der Vektor D^{co}_t für die deterministischen Terme, die im ersten Summanden der rechten Seite von (2.15) Einzug halten. Wir

nennen derartige deterministische Terme auch *auf die Kointegrationsbeziehung beschränkt*. Dagegen umfasst D_t die verbleibenden deterministischen Terme. Mit η und C bezeichnen wir die entsprechenden Parametermatrizen.

Mit einer geeigneten Notation können wir dieses Modell wie jenes ohne deterministische Terme behandeln. Sowohl die oben diskutierte Schätzung der Parameter als auch die noch einzuführenden Verfahren, wie etwa die Vorhersage, funktionieren dann analog zum Modell ohne deterministischen Anteil. Dazu führen wir die Symbole

$$\begin{aligned} y_t^+ &:= \begin{pmatrix} y_t \\ D_t^{co} \end{pmatrix}, \\ Y_{-1}^+ &:= (y_0^+, \ldots, y_{T-1}^+), \\ \Gamma^+ &:= (\Gamma_1, \ldots, \Gamma_{p-1}, C) \text{ sowie} \\ \Delta X^+ := (\Delta X_0^+, \ldots, \Delta X_{T-1}^+) \text{ mit } \Delta X_t^+ &:= \begin{pmatrix} \Delta y_t \\ \vdots \\ \Delta y_{t-p+2} \\ D_t \end{pmatrix} \end{aligned}$$

ein. Für die Parameterschätzung etwa sind damit lediglich die hier definierten Symbole zu verwenden anstelle von jenen, die in Unterabschnitt 2.5.2 ohne dem hochgestellten $+$ definiert wurden.

2.6. Untersuchung der Residuen

2.6.1. Test auf Nichtnormalität

Definition 2.1 erfordert keine Normalverteilung von u_s mit $s \in \mathbb{N}$. Jedoch beruhte die Herleitung der ML-Schätzer in Unterabschnitt 2.5.2 auf der Annahme, dass u_t ein Gauß'scher Prozess ist. Auch im Zusammenhang mit Vorhersagen in Abschnitt 2.7 werden wir auf diese Annahme treffen. Daher wäre die Möglichkeit einer Überprüfung, ob diese Annahme auch legitim ist, nützlich. Entsprechende Tests werden in Abschnitt 4.5 und

Unterabschnitt 8.4.2 in Lütkepohl, (2005) vorgestellt, welche die Grundlage dieses Unterabschnitts bilden.

Wir werden das Problem zunächst im Kontext des VAR-Modells besprechen und danach auf das VECM eingehen. Die gezeigten Tests basieren auf dem Vergleich der Schiefe und/oder Wölbung der Standardnormalverteilung mit entsprechenden empirischen Werten.

Betrachten wir ein VAR(0)-Modell ohne Intercept, also einen Random Walk, und nehmen wir zusätzlich an, dass $u_t \sim N(0, \Sigma_u)$ für alle $t \in \mathbb{N}$ gilt. Aus Anhang A.2 wissen wir, dass die Zerlegung $\Sigma_u = PP'$ mit einer invertierbaren Matrix P existiert. Daraus folgt

$$w_t = (w_{1t}, \dots, w_{Kt})' := P^{-1} u_t \sim N(0, I_K)$$

für $t \in \mathbb{N}$. Damit sind die Komponenten von w_t unabhängig standardnormalverteilt und für das dritte und vierte Moment gilt

$$\mathbb{E}\begin{pmatrix} w_{1t}^3 \\ \vdots \\ w_{Kt}^3 \end{pmatrix} = 0 \quad \text{und} \quad \mathbb{E}\begin{pmatrix} w_{1t}^4 \\ \vdots \\ w_{Kt}^4 \end{pmatrix} = \begin{pmatrix} 3 \\ \vdots \\ 3 \end{pmatrix} =: \mathbf{3}_K.$$

Betrachten wir nun die Residuen $u_1, \dots, u_T$. Seien

$$\bar{u} := \frac{1}{T}\sum_{t=1}^{T} u_t, \quad S_u := \frac{1}{T-1}\sum_{t=1}^{T}(u_t - \bar{u})(u_t - \bar{u})'$$

und P_s derart, dass $S_u = P_s P_s'$ gilt. Weiters definieren wir

$$v_t = (v_{1t}, \dots, v_{Kt})' := P_s^{-1}(u_t - \bar{u}), \quad \text{für} \quad t = 1, \dots, T,$$

$$b_1 = (b_{11}, \dots, b_{K1})' \quad \text{mit} \quad b_{k1} := \frac{1}{T}\sum_t v_{kt}^3, \quad \text{für} \quad k = 1, \dots, K \quad \text{sowie}$$

$$b_2 = (b_{12}, \dots, b_{K2})' \quad \text{mit} \quad b_{k2} := \frac{1}{T}\sum_t v_{kt}^4, \quad \text{für} \quad k = 1, \dots, K.$$

Unter der vorausgesetzten Normalität kann gezeigt werden (siehe Proposition 4.9 in Lütkepohl, (2005)), dass

$$\sqrt{T}\begin{pmatrix} b_1 \\ b_2 - \mathbf{3}_K \end{pmatrix} \xrightarrow{d} N\left(0, \begin{pmatrix} 6I_K & 0 \\ 0 & 24I_K \end{pmatrix}\right),$$

gilt. Daraus folgern wir

$$\lambda_S := \frac{T}{6} b_1' b_1 \xrightarrow{d} \chi^2(K) \quad \text{und}$$
$$\lambda_W := \frac{T}{24}(b_2 - \mathbf{3}_K)(b_2 - \mathbf{3}_K) \xrightarrow{d} \chi^2(K).$$

Dabei stehen die Indizes von λ_S und λ_W für *Schiefe* bzw. *Wölbung*. Mit diesen Teststatistiken können wir

$$H_0 : \mathbb{E} \begin{pmatrix} w_{1t}^3 \\ \vdots \\ w_{Kt}^3 \end{pmatrix} = 0 \quad \text{gegen} \quad H_1 : \mathbb{E} \begin{pmatrix} w_{1t}^3 \\ \vdots \\ w_{Kt}^3 \end{pmatrix} \neq 0$$

bzw.

$$H_0 : \mathbb{E} \begin{pmatrix} w_{1t}^4 \\ \vdots \\ w_{Kt}^4 \end{pmatrix} = \mathbf{3}_K \quad \text{gegen} \quad H_1 : \mathbb{E} \begin{pmatrix} w_{1t}^4 \\ \vdots \\ w_{Kt}^4 \end{pmatrix} \neq \mathbf{3}_K$$

testen. Ein Test, der beide Nullhypothesen gemeinsam testet, kann mit der Teststatistik

$$\lambda_{sw} = \lambda_S + \lambda_W \xrightarrow{d} \chi^2(2K)$$

konstruiert werden. Er testet gleichzeitig über λ_S auf die Abweichung von einer symmetrischen Verteilung und über λ_W auf zu große oder zu kleine Tails.

Das im Zusammenhang mit dem VAR(0)-Modell demonstrierte Verfahren bedarf für die Gültigkeit im allgemeineren VAR(p)-Modell nur geringfügiger Adaptionen, wie aus Proposition 4.10 in Lütkepohl, (2005) folgt. So werden statt $\bar{u}$ und S_u die aus der Schätzung der Parameter gewonnenen Größen

$$\hat{u}_t := (y_t - \bar{y}) - \hat{A}_1(y_{t-1} - \bar{y}) - \cdots - \hat{A}_p(y_{t-p} - \bar{y}), \quad t = 1, \ldots, T \quad \text{bzw.}$$
$$\hat{\Sigma}_u := \frac{1}{T - Kp - 1} \sum_{t=1}^{T} \hat{u}_t \hat{u}_t' \tag{2.45}$$

verwendet. Das weitere Verfahren folgt dem oben beschriebenen. Die resultierenden Teststatistiken λ_S, λ_W und λ_{SW} weisen die selbe asymptotische Verteilung auf wie beim VAR(0)-Modell.

Auch das VECM erbt das Verfahren zur Untersuchung der Normalität von u_s mit $s \in \mathbb{N}$. Dies haben Kilian und Demiroglu, (2000) in ihrem Abschnitt 2.2 gezeigt. Es sind dazu lediglich die Variablen aus (2.45) durch die auf S. 42 in Kilian und Demiroglu, (2000) definierten Größen $\tilde{u}_t$ und $\tilde{\Sigma}_u$ zu ersetzen.

2.6.2. Test auf Autokorrelation

Wenn u_t weißes Rauschen darstellt, dann ist die Autokovarianzmatrix $C_i = 0$ für Lag $i \geq 1$. Selbiges gilt für die Autokorrelation R_i zu einem Lag $i \geq 1$. Um zu überprüfen, ob diese Annahme bei gegebenen Beobachtungen gerechtfertigt ist, bedarf es eines statistischen Tests. Ein solcher soll in der Folge zunächst für VAR-Modelle und sodann für VECMs vorgestellt werden. Dieser Unterabschnitt stützt sich dabei auf Abschnitt 4.4 und Unterabschnitt 8.4.1 in Lütkepohl, (2005).

Ein Schätzer für die Autokovarianzmatrix zum Lag $i < T$ ist

$$\hat{C}_i := \frac{1}{T} \sum_{t=i+1}^{T} \hat{u}_t \hat{u}'_{t-i}.$$

Wir definieren

$$\hat{\boldsymbol{C}}_h := (\hat{C}_1, \ldots, \hat{C}_h), \quad \hat{\boldsymbol{c}}_h := \operatorname{vec}(\hat{\boldsymbol{C}}_h)$$

und analog

$$\hat{R}_i := \hat{D}^{-1} \hat{C}_i \hat{D}^{-1} \quad \text{sowie} \quad \hat{\boldsymbol{R}}_h := (\hat{R}_1, \ldots, \hat{R}_h).$$

Dabei ist $\hat{R}_i$ ein Schätzer für die Autokorrelation zum Lag i und $\hat{D}$ eine Diagonalmatrix deren Diagonale die Wurzeln der Diagonalelemente von $\hat{C}_i$ enthält.

In Proposition 4.5 leitet Lütkepohl, (2005) die asymptotische Verteilung von $\sqrt{T}\hat{\boldsymbol{c}}_h$ her. Daraus folgert er in Proposition 4.7 einen Test auf Autokorrelation der Residuen bis zum Lag h. Damit wird

$$H_0 : \boldsymbol{R}_h := (R_1, \ldots, R_h) = 0 \quad \text{gegen} \quad H_1 : \boldsymbol{R}_h \neq 0$$

getestet. Die Teststatistik lautet

$$\begin{aligned} Q_h &:= T \sum_{i=1}^{h} \operatorname{tr}\left(\hat{R}_i' \hat{R}_0^{-1} \hat{R}_i \hat{R}_0^{-1}\right) \\ &= T \sum_{i=1}^{h} \operatorname{tr}\left(\hat{R}_i' \hat{D}^{-1} \hat{D} \hat{R}_0^{-1} \hat{D}^{-1} \hat{D} \hat{R}_i \hat{D}^{-1} \hat{D} \hat{R}_0^{-1} \hat{D}^{-1} \hat{D}\right) \\ &= T \sum_{i=1}^{h} \operatorname{tr}\left(\hat{D} \hat{R}_i' \hat{D}^{-1} \hat{D} \hat{R}_0^{-1} \hat{D}^{-1} \hat{D} \hat{R}_i \hat{D}^{-1} \hat{D} \hat{R}_0^{-1} \hat{D}^{-1}\right) \\ &= T \sum_{i=1}^{h} \operatorname{tr}\left(\hat{C}_i' \hat{C}_0^{-1} \hat{C}_i \hat{C}_0^{-1}\right), \end{aligned}$$

wobei beim vorletzten Gleichheitszeichen Rechenregel (A.1) einfloss. Diese Teststatistik besitzt für große T und h eine approximative $\chi^2(K^2(h-p))$-Verteilung. Der auf ihr basierende Test wird Portmanteau-Test genannt.

Auf S. 171 beschreibt Lütkepohl, (2005), dass die Teststatistik Q_h bei kurzen Beobachtungsdauern T Schwächen aufweist. Daher wird alternativ auch die adaptierte Größe

$$\bar{Q}_h := T^2 \sum_{i=1}^{h} (T-i)^{-1} \operatorname{tr}\left(\hat{C}_i' \hat{C}_0^{-1} \hat{C}_i \hat{C}_0^{-1}\right)$$

verwendet. Die asymptotischen Verteilungen von Q_h und $\bar{Q}_h$ unterscheiden sich aufgrund von $\frac{T}{T^2(T-i)^{-1}} \overset{T\to\infty}{\longrightarrow} 1$ nicht.

Es lässt sich zeigen, dass im Zusammenhang mit VECMs ein ähnlicher Test anwendbar ist. Dazu behalten alle in diesem Abschnitt definierten Symbole ihre bisherige Bedeutung. Lediglich $\hat{u}_t$ bezieht sich nun auf die Residuen

des VECMs. In ihrem Abschnitt 4.2 zeigen Brüggemann et al., (2006), dass die Teststatistik

$$Q_h = T \sum_{i=1}^{h} \operatorname{tr}\left(\hat{C}_i' \hat{C}_0^{-1} \hat{C}_i \hat{C}_0^{-1}\right)$$

auch bei VECMs für große T und h annähernd χ^2-verteilt ist – nun jedoch mit $K^2(h-p+1)-Kr$ Freiheitsgraden. Wiederum steht eine analog zu $\bar{Q}_h$ definierte alternative Teststatistik zur Verfügung.

2.7. Vorhersagen

2.7.1. Vorhersagen für VAR-Prozesse

Vorhersagen für bekannte VAR-Prozesse

In diesem Unterabschnitt führen wir ein Verfahren zur Vorhersage zukünftiger Werte eines VAR-Prozesses ein. Es verfolgt das Ziel, den erwarteten quadratischen Fehler (in weiterer Folge MSE nach der englischen Entsprechung *mean squared error*) zwischen dem vorhergesagten und realisierten Wert der Zeitreihe zu minimieren. Für die Vorhersage sollen nur Funktionen betrachtet werden, die linear in den beobachteten Werten der Zeitreihe sind. Die Grundlage dieses Unterabschnitts bilden die Abschnitte 2.2 und 3.5 in Lütkepohl, (2005).

Wir betrachten zunächst bekannte VAR-Prozesse und beginnen mit einem VAR(1)-Prozess ohne Intercept

$$y_t = A_1 y_{t-1} + u_t.$$

Für diesen gilt

$$\begin{aligned} y_{t+h} &= A_1 y_{t+h-1} + u_{t+h} = A_1^2 y_{t+h-2} + A_1 u_{t+h-1} + u_{t+h} \\ &= \cdots = A_1^h y_t + \sum_{i=0}^{h-1} A_1^i u_{t+h-i}. \end{aligned}$$

Eine Vorhersage für y_{t+h} zum Zeitpunkt t bezeichnen wir mit $y_t(h)$. Die vorausgesetzte Linearität in den bisherigen Beobachtungen bedeutet, dass es sich bei der Vorhersage um eine Funktion der Form $B_0 y_t + B_1 y_{t-1} + \dots$ handelt mit $B_i \in \mathbb{R}^{K \times K}$ für $i \geq 0$. Damit nimmt der Vorhersagefehler die Form

$$y_{t+h} - y_t(h) = \sum_{i=0}^{h-1} A_1^i u_{t+h-i} + (A_1^h - B_0) y_t - \sum_{i=1}^{\infty} B_i y_{t-i}$$

an. Unter Verwendung der Unkorreliertheit zwischen zukünftigem weißen Rauschen und bereits beobachteten Ausprägungen der Zeitreihe erhalten wir

$$\begin{aligned}
\operatorname{MSE}(y_t(h)) &= \mathbb{E}\left((y_{t+h} - y_t(h)) \, (y_{t+h} - y_t(h))' \right) \\
&\overset{(*)}{=} \mathbb{E}\left(\left(\sum_{i=0}^{h-1} A_1^i u_{t+h-i} \right) \left(\sum_{i=0}^{h-1} A_1^i u_{t+h-i} \right)' \right) \\
&\quad + \mathbb{E}\left(\left((A_1^h - B_0) y_t - \sum_{i=1}^{\infty} B_i y_{t-i} \right) \left((A_1^h - B_0) y_t - \sum_{i=1}^{\infty} B_i y_{t-i} \right)' \right),
\end{aligned}$$

wobei die Unkorreliertheit beim mit $(*)$ markierten Gleichheitszeichen einfließt. Mit der Wahl $B_0 = A_1^h$ und $B_i = 0$ für $i \geq 0$ verschwindet der zweite Summand auf der rechten Seite und der MSE ist minimal. Damit ist

$$y_t(h) = A_1^h y_t = A_1 y_t(h-1) \tag{2.46}$$

die optimale Vorhersage. Der Fehler beträgt $\sum_{i=0}^{h-1} A_1^i u_{t+h-i}$ und der MSE lautet

$$\operatorname{MSE}(y_t(h)) = \mathbb{E}\left(\left(\sum_{i=0}^{h-1} A_1^i u_{t+h-i} \right) \left(\sum_{i=0}^{h-1} A_1^i u_{t+h-i} \right)' \right) \overset{(*)}{=} \sum_{i=0}^{h-1} A_1^i \Sigma_u (A_1^i)'.$$

Wiederum wurde die Stelle, an der Unkorreliertheitseigenschaft des weißen Rauschens einfließt, mit $(*)$ gekennzeichnet.

Der Fall eines VAR(p)-Prozesses ohne Intercept

$$y_t = A_1 y_{t-1} + \dots + A_p y_{t-p} + u_t \tag{2.47}$$

lässt sich auf jenen des soeben besprochenen VAR(1)-Prozesses zurückführen und wir erhalten

$$Y_t(h) = \boldsymbol{A}^h Y_t \quad \text{mit} \quad Y_t(h) = \begin{pmatrix} y_t(h) \\ y_t(h-1) \\ \vdots \\ y_t(h-p+1) \end{pmatrix},$$

wobei wir $y_t(j) := y_{t+j}$ für $j \leq 0$ setzen. Auch für diesen Fall können wir mit

$$\begin{aligned} y_t(h) &= J\boldsymbol{A}Y_t(h-1) = \left(A_1, \ldots, A_p\right) Y_t(h-1) \\ &= A_1 y_t(h-1) + \cdots + A_p y_t(h-p) \end{aligned}$$

eine rekursive Formel herleiten, wobei J wie in (2.10) definiert ist. Für den Fall eines VAR(p)-Prozesses mit konstantem Vektor ν wie in Definition 2.1 lässt sich

$$y_t(h) = \nu + A_1 y_t(h-1) + \cdots + A_p y_t(h-p) \tag{2.48}$$

herleiten. Im Fall $\nu = 0$ gilt $Y_{t+h} = \boldsymbol{A}^h Y_t + \sum_{i=0}^{h-1} \boldsymbol{A}^i U_{t+h-i}$ und wir erhalten für den Vorhersagefehler

$$\begin{aligned} y_{t+h} - y_t(h) &= J\left(Y_{t+h} - Y_t(h)\right) = J\left(\sum_{i=0}^{h-1} \boldsymbol{A}^i U_{t+h-i}\right) \\ &= \sum_{i=0}^{h-1} J\boldsymbol{A}^i J' J U_{t+h-i} = \sum_{i=0}^{h-1} \phi_i u_{t+h-i}, \end{aligned} \tag{2.49}$$

wobei $\phi_i = J\boldsymbol{A}^i J'$ wie in (2.12). Dies gilt auch im Fall $\nu \neq 0$, da sich dieser konstante Term sowohl in y_{t+h} als auch in $y_t(h)$ befindet und somit wegfällt. Aus (2.11) und (2.49) gewinnen wir mit

$$y_t(h) = \mu + \sum_{i=h}^{\infty} \phi_i u_{t+h-i} = \mu + \sum_{i=0}^{\infty} \phi_{h+i} u_{t-i} \tag{2.50}$$

eine weitere Darstellung der Vorhersage. Aus (2.49) folgt

$$\Sigma_y(h) := \mathrm{MSE}(y_t(h)) = \sum_{i=0}^{h-1} \phi_i \Sigma_u \phi_i' = \Sigma_y(h-1) + \phi_{h-1} \Sigma_u \phi_{h-1}'. \tag{2.51}$$

Mit Hilfe dieser Kovarianzmatrix ist es möglich, Vorhersageintervalle zu berechnen. Sie wird auch benötigt, wenn geschätzte VAR(p)-Prozesse betrachtet werden, worauf wir in der Folge eingehen möchten.

Vorhersagen für geschätzte VAR-Prozesse

Werden die Parameter eines VAR(p)-Modells geschätzt, so hat dies Auswirkungen auf die Vorhersage. Schätzer werden in der Folge stets mit einem Dach gekennzeichnet, wie z.B. beim Schätzer $\hat{A}_1$ für die wahre Parametermatrix A_1. Ersetzen wir in (2.48) die Parameter durch ihre Schätzer, so erhalten wir die Vorhersage

$$\hat{y}_t(h) = \hat{v} + \hat{A}_1 \hat{y}_t(h-1) + \cdots + \hat{A}_p \hat{y}_t(h-p),$$

wobei wiederum $\hat{y}_t(j) := y_{t+j}$ für $j \leq 0$ gilt. Für den Vorhersagefehler erhalten wir

$$y_{t+h} - \hat{y}_t(h) = (y_{t+h} - y_t(h)) + (y_t(h) - \hat{y}_t(h)). \tag{2.52}$$

Den ersten Summanden auf der rechten Seite haben wir bereits in (2.49) berechnet. Demnach fließt das weiße Rauschen hier nur nach dem Zeitpunkt t ein. Die Differenz der beiden Vorhersagen, die den zweiten Summanden bildet, enthält das weiße Rauschen hingegen nur bis zum Zeitpunkt t. Damit sind die beiden Summanden unkorreliert und wir folgern

$$\begin{aligned}
\mathrm{MSE}(\hat{y}_t(h)) &= \mathbb{E}\left((y_{t+h} - \hat{y}_t(h)) \, (y_{t+h} - \hat{y}_t(h))' \right) \\
&= \mathbb{E}\left((y_{t+h} - y_t(h)) \, (y_{t+h} - y_t(h))' \right) \\
&\quad + \mathbb{E}\left((y_t(h) - \hat{y}_t(h)) \, (y_t(h) - \hat{y}_t(h))' \right) \\
&= \Sigma_y(h) + \mathbb{E}\left((y_t(h) - \hat{y}_t(h)) \, (y_t(h) - \hat{y}_t(h))' \right)
\end{aligned}$$

Aufgrund von $\mathbb{E}\left(y_{t+h} - \hat{y}_t(h)\right) = 0$ ist dies die Kovarianzmatrix von $\hat{y}_t(h)$. Wir bezeichnen sie mit $\Sigma_{\hat{y}}(h) := \mathrm{MSE}(\hat{y}_t(h))$. Unter der Annahme, dass u_t mehrdimensional normalverteilt ist und die Parameter mit der Least-Squares-Methode geschätzt wurden, lässt sich zeigen, dass

$$\sqrt{T}\left(\hat{y}_t(h) - y_t(h)\right) \xrightarrow{d} N(0, \Omega(h))$$

mit

$$\Omega(h) = \sum_{i=0}^{h-1}\sum_{j=0}^{h-1} \mathrm{tr}\left(\left(\boldsymbol{B}'\right)^{h-1-i}\Gamma^{-1}\boldsymbol{B}^{h-1-j}\Gamma\right)\phi_i\Sigma_u\phi_j'. \tag{2.53}$$

Die in diesem Ausdruck verwendeten Symbole sind definiert als

$$\boldsymbol{B} := \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ \nu & A_1 & A_2 & \dots & A_{p-1} & A_p \\ 0 & I_K & 0 & \dots & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & I_K & 0 \end{pmatrix} \in \mathbb{R}^{(Kp+1)\times(Kp+1)} \tag{2.54}$$

und $\Gamma := E(Z_t Z_t')$ mit $Z := \left(Z_0, \dots, Z_{T-1}\right)$, wobei T die Anzahl der bisherigen Beobachtungen abzüglich Presample bezeichnet und

$$Z_t := \left(1, y_t, \dots, y_{t-p+1}\right)'$$

für $t \in \{0, \dots, T-1\}$. Damit erhalten wir asymptotisch

$$\Sigma_{\hat{y}}(h) = \Sigma_y + \frac{1}{T}\Omega(h). \tag{2.55}$$

Für $h = 1$ gilt beispielsweise

$$\Sigma_y(1) = \phi_0\Sigma_u\phi_0' = \Sigma_u$$

sowie

$$\Omega(1) = \mathrm{tr}\left(\left(\boldsymbol{B}'\right)^0\Gamma^{-1}\boldsymbol{B}^0\Gamma\right)\phi_0\Sigma_u\phi_0' = \mathrm{tr}\left(I_{Kp+1}\right)I_K\Sigma_u I_K' = (Kp+1)\Sigma_u.$$

und damit $\Sigma_{\hat{y}}(1) = \Sigma_u + \frac{Kp+1}{T}\Sigma_u = \frac{T+Kp+1}{T}\Sigma_u$.

Mit Hilfe der Vorhersage sowie $\Sigma_{\hat{y}}(h)$ können wir für alle $\alpha \in (0,1)$ ein Prognoseintervall zum Niveau $1-\alpha$ angeben. Dieses lautet für jede Komponente $y_{k,t}$ für $k = 1, \ldots, K$ des Prozesses y_t

$$\left[\hat{y}_{k,t}(h) - z_{1-\alpha/2}\,\hat{\sigma}_k(h), \hat{y}_{k,t}(h) + z_{1-\alpha/2}\,\hat{\sigma}_k(h)\right]. \tag{2.56}$$

Dabei bezeichnen wir mit $\hat{\sigma}_k(h)$ die Wurzel des k-ten Diagonalelements von $\Sigma_{\hat{y}}(h)$ und mit $z_{1-\alpha/2}$ das $1-\frac{\alpha}{2}$-Quantil der Standardnormalverteilung.

Für $h > 0$ enthält die Formel für $\Omega(h)$ in (2.53) die wahren Parameter. Somit setzt die Berechnung von Vorhersageintervallen die Kenntnis dieser Parameter voraus. Da die wahren Parameter nach unserer Annahme unbekannt sind, werden sie durch die entsprechenden Schätzer ersetzt. Dazu kann auf die mit einem Dach gekennzeichneten Least-Squares-Schätzer für ν, $A_1, \ldots, A_p$ und Σ_u zurückgegriffen werden. Statt $\boldsymbol{B}$ aus (2.54) wird dann mit

$$\hat{\boldsymbol{B}} := \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 & 0 \\ \hat{\nu} & \hat{A}_1 & \hat{A}_2 & \ldots & \hat{A}_{p-1} & \hat{A}_p \\ 0 & I_K & 0 & \ldots & \ldots & 0 \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \ldots & \ldots & 0 & I_K & 0 \end{pmatrix} \in \mathbb{R}^{(Kp+1)\times(Kp+1)}$$

gerechnet, während ϕ_i und Γ durch $\hat{\phi}_i = J_1 \hat{\boldsymbol{B}}^i J_1'$ mit $J_1 := (\overbrace{0}^{K\times 1}, I_K, \overbrace{0, \ldots, 0}^{K\times K(p-1)}) \in \mathbb{R}^{K\times(Kp+1)}$ bzw. $\hat{\Gamma} = \frac{1}{T}ZZ'$ ersetzt werden. Aus diesen Anpassungen resultiert die approximative Kovarianzmatrix $\hat{\Sigma}_{\hat{y}}(h)$ und wir erhalten als Prognoseintervall

$$\left[\hat{y}_{k,t}(h) - z_{1-\alpha/2}\,\hat{\hat{\sigma}}_k(h), \hat{y}_{k,t}(h) + z_{1-\alpha/2}\,\hat{\hat{\sigma}}_k(h)\right], \tag{2.57}$$

wobei $\hat{\hat{\sigma}}_k(h)$ das k-te Diagonalelement von $\hat{\Sigma}_{\hat{y}}(h)$ ist.

2.7.2. Vorhersagen für VECMs

Vorhersagen mit bekannten VECM-Parametern

Vorhersagen für VECMs behandelt Lütkepohl, (2005) in seinen Abschnitten 6.5 und 7.5. Ihnen wird dieser Unterabschnitt folgen.

Um Vorhersagen für ein VECM aufzustellen, bietet es sich an, das Modell in ein VAR-Modell zu übersetzen und für dieses die Prognosen zu berechnen. Dazu wird das VECM zunächst als Summe eines deterministischen Terms μ_t und einer rein stochastischen Zeitreihe x_t betrachtet. Das Modell für x_t wird sodann unter Verwendung von (2.16) in ein VAR-Modell umgewandelt und mit den oben vorgestellten Verfahren werden in weiterer Folge Vorhersagen $x_t(h)$ für $h \in \mathbb{N}$ erstellt. Zu diesen Vorhersagen werden abschließend die bekannten, zukünftigen Werte μ_{t+h} des deterministischen Teils der Zeitreihe addiert.

Bei der Berechnung von Prognoseintervallen gibt es einen wichtigen Unterschied im Vergleich zur VAR-bezogenen Theorie zu beachten. Dieser wird bei genauer Betrachtung von

$$\Sigma_y(h) = \sum_{i=0}^{h-1} \phi_i \Sigma_u \phi_i'$$

aus Gleichung (2.51) deutlich. Die darin vorkommende Folge von Matrizen ϕ_i konvergiert im Falle stabiler VAR-Prozesse aufgrund von (2.13) gegen 0. Die fehlende Stabilitätseigenschaft, die ein Ausweichen auf ein VECM erst erforderlich macht, bedeutet, dass nicht alle Eigenwerte der Matrix $\boldsymbol{A}$ aus (2.2) betragsmäßig < 1 sind. Damit ist die Konvergenz $\phi_i \overset{i\to\infty}{\to} 0$ nicht gegeben und einige Diagonalelemente von $\Sigma_y(h)$ können gegen ∞ streben, wenn der Prognosehorizont h gegen ∞ geht. Dies bedeutet, dass die Prognoseintervalle für einige Komponenten von y_t für weiter in der Zukunft liegende Zeitpunkte $t + h$ sehr groß werden können – ein Signal für große Unsicherheit.

Vorhersagen mit unbekannten VECM-Parametern

Wenn die Parameter eines VECMS $y_t = \mu_t + x_t$ nicht bekannt sind, so werden die geschätzten VECM-Parameter für die Umwandlung in ein VAR-Modell für x_t verwendet. Damit erhalten wir die Vorhersage

$$\hat{x}_t = \hat{A}_1 \hat{x}_t(h-1) + \cdots + \hat{A}_p \hat{x}_t(h-p) \tag{2.58}$$

mit $\hat{x}_t(j) := x_{t+j}$ für $j \leq 0$. Analog zu (2.52) in Verbindung mit (2.49) erhalten wir

$$\begin{aligned} x_{t+h} - \hat{x}_t(h) &= (x_{t+h} - x_t(h)) + (x_t(h) - \hat{x}_t(h)) \\ &= \sum_{i=0}^{h-1} \phi_i u_{t+h-i} + (x_t(h) - \hat{x}_t(h)) \end{aligned} \tag{2.59}$$

als Vorhersagefehler, wobei die zwei Summanden wie in (2.52) unkorreliert sind. Es kann gezeigt werden, dass der zweite Summand $(x_t(h) - \hat{x}_t(h))$ in Wahrscheinlichkeit gegen 0 konvergiert. Ein aus ihm erwachsender additiver Beitrag zur Kovarianzmatrix der Vorhersage in Form von $\frac{\Omega(h)}{T}$ lässt sich für ein VECM nicht herleiten, da die Herleitung von (2.53) die Stabilität des VAR-Modells voraussetzt. Daher ist

$$\hat{\Sigma}_x(h) = \sum_{i=0}^{h-1} \hat{\phi}_i \hat{\Sigma}_u \hat{\phi}_i'$$

eine mögliche Wahl für eine approximative Kovarianzmatrix. Zusammen mit der Vorhersage aus (2.59) liefert sie analog zu (2.57) ein Prognoseintervall.

2.8. Granger-Kausalität

2.8.1. Granger-Kausalität bei VAR-Modellen

Seien x_t und z_t zwei Zeitreihen. Dann ist x_t Granger-kausal für z_t, wenn die Information über $x_t, x_{t-1}, \ldots$ zum Zeitpunkt t eine bessere Vorhersage

für die Zeitreihe z_t ermöglicht. Mit Vorhersagen sind dabei – wie in Unterabschnitt 2.7.1 eingeführt – lineare Vorhersagen gemeint. In der Folge werden wir die Granger-Kausalität zwischen den Komponenten $z_t \in \mathbb{R}^M$ und $x_t \in \mathbb{R}^{K-M}$ des K-dimensionalen VAR-Prozesses $y_t = \begin{pmatrix} z_t \\ x_t \end{pmatrix}$ untersuchen. Die Einführung dieses theoretischen Konzepts für VAR-Modelle folgt jener in den Unterabschnitten 2.3.1 und 3.6.1 in Lütkepohl, (2005).

Für die Diskussion der Granger-Kausalität ist die MA-Darstellung eines VAR-Prozesses aus (2.11) nützlich. Mit dem Polynom $\phi(L) := \sum_{i=0}^{\infty} \phi_i L^i$, wobei L den Lagoperator $Ly_t := y_{t-1}$ bezeichnet, erreichen wir die Schreibweise

$$y_t = \mu + \sum_{i=0}^{\infty} \phi_i u_{t-i}, = \mu + \phi(L)u_t$$

bzw.

$$y_t = \begin{pmatrix} z_t \\ x_t \end{pmatrix} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} + \begin{pmatrix} \phi_{11}(L) & \phi_{12}(L) \\ \phi_{21}(L) & \phi_{22}(L) \end{pmatrix} \begin{pmatrix} u_{1t} \\ u_{2t} \end{pmatrix}, \tag{2.60}$$

wobei die Partitionierungen von μ und u_t jener von y_t entsprechen und die Matrix $\phi(L)$ ebenfalls in naheliegender Weise partitioniert ist. In Proposition 2.2 in Lütkepohl, (2005) wird Nichtkausalität von x_t für z_t im Sinne von Granger für Prozesse mit MA-Darstellung durch die Bedingung

$$z_t(1|\{y_s|s \leq t\}) = z_t(1|\{z_s|s \leq t\}). \tag{2.61}$$

definiert. Diese ist laut der erwähnten Proposition genau dann gegeben, wenn $\phi_{12,i} = 0$ für alle $i = 1, 2, \ldots$ gilt. Dabei bezeichnet $\phi_{12,i}$ den i-ten Koeffizienten des Polynoms $\phi_{12}(L)$. Die ausschließliche Betrachtung eines Prognosehorizonts von 1 in (2.61) ist keine Einschränkung desselben. Vielmehr folgt aufgrund von (2.50) die Gleichheit für beliebige Prognosehorizonte $h \in \mathbb{N}$.

Nach Korollar 2.2.1 in Lütkepohl, (2005), das aus (2.14) folgt, lässt sich für stabile VAR-Prozesse alternativ

$$A_{12,i} = 0 \quad \text{für } i = 1, \ldots, p \tag{2.62}$$

als notwendige und hinreichende Bedingung für

$$z_t(h|\{y_s|s \leq t\}) = z_t(h|\{z_s|s \leq t\}) \quad \text{für } h \in \mathbb{N}$$

angeben. Dabei bezeichnet $A_{12,i}$ einen Block von

$$A_i = \begin{pmatrix} A_{11,i} & A_{12,i} \\ A_{21,i} & A_{22,i} \end{pmatrix}. \tag{2.63}$$

Somit geben bei bekannten, stabilen VAR-Prozessen die Parametermatrizen Auskunft über eine etwaige Granger-Kausalität. Sind die Parameter unbekannt, so müssen sie zunächst geschätzt werden. In weiterer Folge können die Schätzer einem statistischen Test unterzogen werden. Im Falle einer signifikanten Abweichung von 0 für einen Schätzer von $A_{12,i}$ für ein $i \in \{1, \ldots, p\}$, wird von einer vorhandenen Granger-Kausalität ausgegangen. Die Hypothesen des Tests lauten somit

$$H_0 : C\beta = 0 \text{ sowie } H_1 : C\beta \neq 0,$$

wobei $\beta = \text{vec}(\begin{pmatrix} A_1 & A_2 & \ldots & A_p \end{pmatrix}) \in \mathbb{R}^{K+K^2p}$ alle Parameter enthält und die Matrix $C \in \mathbb{R}^{N \times (K+K^2p)}$ derart gewählt ist, dass das Produkt $C\beta$ ein Vektor jener N Parameter ist, deren Wert auf Verschiedenheit von 0 getestet wird. Anhand eines Beispiels soll das Aufstellen der Hypothesen verdeutlicht werden. Dazu nehmen wir an, dass $z_t \in \mathbb{R}$ und $x_t \in \mathbb{R}^2$ eine Partition von $y_t \in \mathbb{R}^3$ bilden. Die Zeitreihe y_t werde als VAR(2)

$$y_t = \nu + A_1 y_{t-1} + A_2 y_{t-2} + u_t$$

bzw.

$$\begin{pmatrix} z_t \\ x_{1,t} \\ x_{2,t} \end{pmatrix} = \begin{pmatrix} \nu_1 \\ \nu_2 \\ \nu_3 \end{pmatrix} + \begin{pmatrix} a_{11,1} & a_{12,1} & a_{13,1} \\ a_{21,1} & a_{22,1} & a_{23,1} \\ a_{31,1} & a_{32,1} & a_{33,1} \end{pmatrix} \begin{pmatrix} z_{t-1} \\ x_{1,t-1} \\ x_{2,t-1} \end{pmatrix} + \begin{pmatrix} a_{11,2} & a_{12,2} & a_{13,2} \\ a_{21,2} & a_{22,2} & a_{23,2} \\ a_{31,2} & a_{32,2} & a_{33,2} \end{pmatrix} \begin{pmatrix} z_{t-2} \\ x_{1,t-2} \\ x_{2,t-2} \end{pmatrix} + \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}$$

modelliert. Damit nimmt der Vektor aller Parameter die Gestalt

$$\beta = (\nu_1, \nu_2, \nu_3, a_{11,1}, a_{21,1}, a_{31,1}, a_{12,1}, \ldots, a_{23,2}, a_{33,2})'$$

an. Die Nullhypothese besagt, dass x_t nicht Granger-kausal für z_t ist. Dies ist gleichbedeutend mit

$$H_o : a_{12,1} = a_{13,1} = a_{12,2} = a_{13,2} = 0.$$

Mit

$$C = \begin{pmatrix} 0\,0\,0 & 0\,0\,0 & 1\,0\,0 & 0\,0\,0 & 0\,0\,0 & 0\,0\,0 & 0\,0\,0 \\ 0\,0\,0 & 0\,0\,0 & 0\,0\,0 & 1\,0\,0 & 0\,0\,0 & 0\,0\,0 & 0\,0\,0 \\ 0\,0\,0 & 0\,0\,0 & 0\,0\,0 & 0\,0\,0 & 0\,0\,0 & 1\,0\,0 & 0\,0\,0 \\ 0\,0\,0 & 0\,0\,0 & 0\,0\,0 & 0\,0\,0 & 0\,0\,0 & 0\,0\,0 & 1\,0\,0 \end{pmatrix}$$

gilt $C\beta = \left(a_{12,1}, a_{13,1}, a_{12,2}, a_{13,2}\right)'$.

Im Falle des Least-Squares-Schätzers $\hat{\beta}$ für β gilt aufgrund von (2.27)

$$\sqrt{T}\left(C\hat{\beta} - C\beta)\right) \xrightarrow{d} N\left(0, C\left(\Gamma^{-1} \otimes \Sigma_u\right) C'\right)$$

und damit

$$T\left(C\hat{\beta} - C\beta\right)' \left(C\left(\Gamma^{-1} \otimes \Sigma_u\right) C'\right)^{-1} \left(C\hat{\beta} - C\beta\right) \xrightarrow{d} \chi^2\left(N\right). \tag{2.64}$$

Ersetzen wir die in (2.64) vorkommenden unbekannten Größen gemäß (2.29) und (2.28), erhalten wir unter der Nullhypothese die asymptotisch $\chi^2(N)$-verteilte Wald-Statistik

$$\lambda_W = \left(C\hat{\beta}\right)' \left(C\left(\left(ZZ'\right)^{-1} \otimes \hat{\Sigma}_u\right) C'\right)^{-1} C\hat{\beta}. \tag{2.65}$$

2.8.2. Granger-Kausalität bei VECMs

In diesem Unterabschnitt setzen wir voraus, dass die betrachtete Zeitreihe y_t einem VECM$(p-1)$-Modell folgt. Weiters nehmen wir an, dass sie analog zu Unterabschnitt 2.8.1 aus den Komponenten z_t und x_t besteht. Im Rahmen von VAR-Modellen galt, dass x_t genau dann nicht Granger-kausal für z_t ist, wenn (2.62) erfüllt ist. Da jedem VECM$(p-1)$-Modell ein VAR(p)-Modell entspricht, können wir diese Bedingung auch für vektorwertige Fehlerkorrekturmodelle verwenden. Dieser Unterabschnitt, der auf den Abschnitten

6.6 und 7.6 in Lütkepohl, (2005) basiert, erklärt diesen Zusammenhang genauer.

Aufgrund von (2.17) stellt

$$\Pi_{12} = 0$$

in Verbindung mit

$$\Gamma_{12,i} = 0 \quad \text{für } i = 1, \ldots, p-1$$

eine äquivalente Bedingung zu (2.62) dar, wobei Π_{12} sowie $\Gamma_{12,i}$ analog zu $A_{12,i}$ in (2.63) als Blöcke von

$$\Pi = \begin{pmatrix} \Pi_{11} & \Pi_{12} \\ \Pi_{21} & \Pi_{22} \end{pmatrix}$$

bzw.

$$\Gamma = \begin{pmatrix} \Gamma_{11,i} & \Gamma_{12,i} \\ \Gamma_{21,i} & \Gamma_{22,i} \end{pmatrix}$$

definiert sind. Unter Verwendung dieser Matrizen nimmt das betrachtete VECM($p-1$)-Modell die Form

$$\begin{pmatrix} \Delta z_t \\ \Delta x_t \end{pmatrix} = \begin{pmatrix} \Pi_{11} & \Pi_{12} \\ \Pi_{21} & \Pi_{22} \end{pmatrix} \begin{pmatrix} z_{t-1} \\ x_{t-1} \end{pmatrix} + \sum_{i=1}^{p-1} \begin{pmatrix} \Gamma_{11,i} & \Gamma_{12,i} \\ \Gamma_{21,i} & \Gamma_{22,i} \end{pmatrix} \begin{pmatrix} \Delta z_{t-i} \\ \Delta x_{t-i} \end{pmatrix}$$

an. Somit genügt im Falle eines VECM-Modells mit bekannten Parametermatrizen die Betrachtung derselben, um eine etwaige Granger-Kausalität zu erkennen – wie im Falle eines bekannten VAR-Prozesses.

Falls die Parameter jedoch unbekannt sind, so müssen ihre Schätzer einem statistischen Test unterzogen werden, in welchem sie auf eine signifikante Verschiedenheit von 0 geprüft werden. Eine analoge Vorgehensweise zu Unterabschnitt 2.8.1 birgt jedoch Probleme. Die Teststatistik aus (2.65) beinhaltet die Inverse von

$$M := C \left((ZZ')^{-1} \otimes \hat{\Sigma}_u \right) C'$$

als Faktor, wobei $(ZZ')^{-1} \otimes \hat{\Sigma}_u$ die Kovarianz aus (2.27) schätzt. Die entsprechende Kovarianzmatrix im Rahmen von VECMs ist jedoch sowohl im Falle der Least-Squares-Schätzung als auch im Falle der ML-Schätzung im Allgemeinen nicht regulär. Dies wird in Bemerkung 1 auf Seite 287 sowie in Bemerkung 2 auf Seite 296 in Lütkepohl, (2005) gezeigt. Damit ist die Matrix M im Allgemeinen auch nicht invertierbar, und somit die Teststatistik nicht berechenbar. Dieses Problem tritt bei VAR-Prozessen nicht auf, da wir bei diesen im Rahmen dieser Arbeit stets die Stabilität voraussetzen. Dies stellt sicher, dass M invertierbar ist.

Ein Lösungsansatz für das beschriebene Problem hat daher zum Ziel, das Modell derart umzuformen, sodass die interessierenden Parametermatrizen $A_1, \ldots, A_p$ mit einem stabilen Regressor multipliziert werden. Laut unserer Annahme in Unterabschnitt 2.2.4 ist y_t nicht stabil, während Δy_t stabil ist. Diese Eigenschaft werden wir uns nach den folgenden Umformungen zu Nutze machen. In einem ersten Schritt erweitern wir das Modell künstlich um eine weitere Parametermatrix. Ausgangspunkt ist das VAR(p)-Modell (dieses Mal ohne deterministische Terme), d.h.

$$y_t = A_1 y_{t-1} + \cdots + A_p y_{t-p} + u_t.$$

Wenn p – wie oben vorausgesetzt – die korrekte Ordnung des VAR-Modells ist, können wir eine weitere Parametermatrix A_{p+1} hinzufügen, von der wir wissen, dass sie 0 ist – damit entfällt auch die Notwendigkeit $A_{12,p+1} = 0$ als Bedingung in H_0 aufzunehmen. Aus diesem Grund werden nicht mehr wie bisher (siehe (2.62)) alle Parametermatrizen des Modells in die Nullhypothese aufgenommen. Mit A_{p+1} schreibt sich das Modell als

$$\begin{aligned} y_t &= A_1 y_{t-1} + \cdots + A_p y_{t-p} + \overbrace{A_{p+1}}^{=0} y_{t-p-1} + u_t \\ &= \sum_{j=1}^{p} A_j y_{t-j} + A_{p+1} y_{t-p-1} + u_t \\ &= \sum_{j=1}^{p} A_j \left(y_{t-j} - y_{t-p-1}\right) + \left(\sum_{j=1}^{p+1} A_j\right) y_{t-p-1} + u_t. \end{aligned}$$

Subtrahieren wir auf der linken und rechten Seite y_{t-p-1} so erhalten wir

mit der Notation

$$\Delta_k y_t := y_t - y_{t-k} \quad \text{für } k = \pm 1, \pm 2, \ldots$$

aufgrund von (2.17) die Gleichung

$$\Delta_{p+1} y_t = \sum_{j=1}^{p} A_j \Delta_{p+1-j} y_{t-j} + \Pi y_{t-p-1} + u_t. \tag{2.66}$$

Dabei ist für $k > 0$ der Prozess $\Delta_k y_t = (y_t - y_{t-1}) + (y_{t-1} - y_{t-2}) + \cdots + (y_{t-k+1} - y_{t-k})$ als Summe stabiler Prozesse stabil. Ein großer Unterschied zwischen dem ursprünglichen VAR(p)-Modell und (2.66) ist daher, dass in letzterer Gleichung alle interessierenden Parametermatrizen A_j einem stabilen Regressor $\Delta_{p+1-j} y_{t-j}$ zugeordnet sind. Unter dieser Voraussetzung verschwindet auch das oben erwähnte Problem. Nun können die Parameter $\alpha := \text{vec}\left(A_1 \ldots A_p\right)$ mittels Least-Squares-Verfahren geschätzt werden und es lässt sich das Resultat

$$\sqrt{T}\left(\hat{\alpha} - \alpha\right) \xrightarrow{d} N(0, \Sigma_\alpha)$$

mit einer regulären Kovarianzmatrix Σ_α gewinnen. Wie oben erwähnt, ist die Regularität von Σ_α für die Wald-Statistik $\lambda_W = T\hat{\alpha}' C' \left(C \Sigma_\alpha C'\right)^{-1} C\alpha$ von entscheidender Bedeutung, stellt sie doch sicher, dass die Inverse von $C\Sigma_\alpha C'$ existiert. Mit dieser Voraussetzung kann gezeigt werden, dass $\lambda_W \xrightarrow{d} \chi^2(N)$ unter $H_0 : C\alpha = 0$ gilt.

Eine alternative Teststatistik stellt $\lambda_F = \frac{\lambda_W}{N}$ dar. Sie ist asymptotisch $F(N, T)$-verteilt.

Noch zu erwähnen ist, dass bei der Least-Squares-Schätzung nicht das reparametrisierte Modell (2.66) behandelt werden muss. Stattdessen kann einfach das VAR($p+1$)-Modell, von dem ausgehend die Reparametrisierung vollzogen wurde, verwendet werden. Damit gleicht der Test auf Nichtkausalität im Falle eines VECMs jenem eines VAR-Modells. Es ist lediglich zu beachten, dass ein VAR($p+1$)-Modell zu formulieren ist und keine Umwandlung des VECMs in ein VAR(p)-Modell wie in (2.18) zu erfolgen hat.

2.9. Sofortige Kausalität

2.9.1. Sofortige Kausalität bei VAR-Modellen

Die sofortige Kausalität stellt ein weiteres Konzept zur Untersuchung von Wechselwirkungen zwischen zwei Komponenten einer Zeitreihe dar. Daher werden wir auch in diesem Abschnitt eine Partitionierung der betrachteten Zeitreihe $y_t = (z_t, x_t)'$ voraussetzen. Lütkepohl, (2005) führt die sofortige Kausalität in seinen Unterabschnitten 2.3.1 und 3.6.3. Diese stellen die Quelle dieses Unterabschnitts dar.

Auch bei dieser Form der Kausalität wird geprüft, ob sich die Information über eine Komponente auf die Vorhersage der anderen auswirkt. Wie unten gezeigt wird, ist die durch die sofortige Kausalität definierte Relation symmetrisch, d.h. x_t ist genau dann sofort kausal für z_t, wenn z_t sofort kausal für x_t ist.

Für das weitere Vorgehen ist die Cholesky-Zerlegung aus Anhang A.2 hilfreich. Demnach existiert eine reguläre, untere Dreiecksmatrix P mit positiven Hauptdiagonalelementen, sodass $\Sigma_u = PP'$ gilt. Aus der MA-Darstellung (2.11) folgern wir

$$y_t = \mu + \sum_{i=0}^{\infty} \phi_i P P^{-1} u_{t-i} = \mu + \sum_{i=0}^{\infty} \theta_i w_{t-i} \tag{2.67}$$

mit $\theta_i := \phi_i P$ und $w_t := P^{-1} u_t$. Damit stellt w_t ebenfalls weißes Rauschen dar. Die Kovarianzmatrix von w_t lautet

$$\Sigma_w = P^{-1} \Sigma_u \left(P^{-1}\right)' = P^{-1} P P' \left(P^{-1}\right)' = I_K.$$

Mit entsprechender Partitionierung der Matrizen in (2.67) analog zu (2.60) erhalten wir

$$\begin{pmatrix} z_t \\ x_t \end{pmatrix} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} + \begin{pmatrix} \theta_{11,0} & 0 \\ \theta_{21,0} & \theta_{22,0} \end{pmatrix} \begin{pmatrix} w_{1,t} \\ w_{2,t} \end{pmatrix} + \begin{pmatrix} \theta_{11,1} & \theta_{12,1} \\ \theta_{21,1} & \theta_{22,1} \end{pmatrix} \begin{pmatrix} w_{1,t-1} \\ w_{2,t-1} \end{pmatrix} + \dots.$$

Dabei resultiert die spezielle Form von θ_0 aus der Tatsache, dass ϕ_0 die Einheitsmatrix (siehe (2.12)) und P eine untere Dreiecksmatrix ist. Die Gleichung ist zu

$$\begin{aligned} z_{t+1} &= \mu_1 + \theta_{11,0} w_{1,t+1} + \theta_{11,1} w_{1,t} + \theta_{12,1} w_{2,t} + \dots \\ x_{t+1} &= \mu_2 + \theta_{21,0} w_{1,t+1} + \theta_{22,0} w_{2,t+1} + \theta_{21,1} w_{1,t} + \theta_{22,1} w_{2,t} + \dots \end{aligned}$$

äquivalent. Daraus erkennen wir, dass die Information $\{z_s, x_s | s \leq t\} \cup \{z_{t+1}\}$ der Information $\{w_s | s \leq t\} \cup \{w_{1,t+1}\}$ entspricht. Wir folgern damit für die Vorhersage $x_t(1)$ von x_{t+1} zum Zeitpunkt t

$$\begin{aligned} x_t\left(1 | \{z_s, x_s | s \leq t\} \cup \{z_{t+1}\}\right) &= x_t\left(1 | \{w_s | s \leq t\} \cup \{w_{1,t+1}\}\right) \\ &= \mathbb{E}\left(x_{t+1} | \{w_s | s \leq t\} \cup \{w_{1,t+1}\}\right) \\ &= x_t(1 | \{z_s, x_s | s \leq t\}) + \theta_{21,0} w_{1,t+1}. \end{aligned}$$

Damit hängt die Vorhersage für x_{t+1} zum Zeitpunkt t genau dann nicht von der Information über z_{t+1} ab, wenn $\theta_{21,0} = 0$ gilt.

Die notwendige und hinreichende Bedingung für Nichtkausalität $\theta_{21,0} = 0$ können wir auch ausdrücken als

$$\begin{pmatrix} \theta_{11,0} & 0 \\ 0 & \theta_{22,0} \end{pmatrix} = \theta_0 = \phi_0 P = I_K P = P.$$

Daher ist

$$\Sigma_u = PP' = \begin{pmatrix} \theta_{11,0} & 0 \\ 0 & \theta_{22,0} \end{pmatrix} \begin{pmatrix} \theta'_{11,0} & 0 \\ 0 & \theta'_{22,0} \end{pmatrix} = \begin{pmatrix} \theta_{11,0}\theta'_{11,0} & 0 \\ 0 & \theta_{22,0}\theta'_{22,0} \end{pmatrix}$$

eine Blockdiagonalmatrix. Das bedeutet, dass das zu z_t gehörende weiße Rauschen u_{1t} und das zu x_t gehörende u_{2t} unkorreliert sind. Deshalb ist die durch die sofortige Kausalität definierte Relation symmetrisch. Die sofortige Kausalität drückt damit auch keine Ursache-Wirkungsbeziehung zwischen z_t und x_t aus. Zusammenfassend können wir festhalten, dass genau dann keine sofortige Kausalität zwischen z_t und x_t vorliegt, wenn $\mathbb{E}(u_{1t} u'_{2t}) = 0$.

Bei bekannten VAR-Prozessen lässt sich anhand von Σ_u eine etwaige sofortige Kausalität ermitteln. Bei VAR-Modellen mit unbekannten Parametern können

wir dazu Σ_u einem statistischen Test unterziehen. Dabei wird untersucht ob sich die Form von Σ_u signifikant von der einer Blockdiagonalmatrix unterscheidet. Da Σ_u symmetrisch ist, wird in diesem Test lediglich der linke untere Block auf Verschiedenheit von 0 getestet. Daher fließt $\sigma := \text{vech}(\Sigma_u)$ (siehe Definition A.2) in die Nullhypothese ein. Um sicherzustellen, dass nur die Elemente des linken unteren Blocks getestet werden, wird σ mit einer Matrix von links multipliziert, die die entsprechende Auswahl vornimmt. Das bedeutet, dass wir

$$H_0 : C\sigma = 0 \quad \text{gegen} \quad H_1 : C\sigma \neq 0$$

testen. Die Vorgehensweise ähnelt also jener im Zusammenhang mit der Granger-Kausalität. Als Beispiel betrachten wir die Kovarianzmatrix

$$\Sigma_u = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0{,}5 \\ 0 & 0{,}5 & 1 \end{pmatrix}.$$

Für $y_t = (z_t, x_{1t}, x_{2t})$ besteht in diesem Beispiel keine sofortige Kausalität zwischen z_t und x_t, da $\sigma_{21} = \sigma_{31} = 0$. Für die in der Nullhypothese verwendete Größe σ gilt

$$\sigma = \text{vech}(\Sigma_u) = \begin{pmatrix} \sigma_{11} & \sigma_{21} & \sigma_{31} & \sigma_{22} & \sigma_{32} & \sigma_{33} \end{pmatrix}'.$$

Damit von diesem Vektor lediglich die relevanten Werte σ_{21} und σ_{31} in der Nullhypothese verbleiben, wählen wir

$$C = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

Als mögliche Teststatistik lässt sich

$$\lambda_W = T\tilde{\sigma}'C' \left(2CD_K^+ \left(\tilde{\Sigma}_u \otimes \tilde{\Sigma}_u\right) D_K^{+\prime} C'\right)^{-1} C\tilde{\sigma} \tag{2.68}$$

mit asymptotischer $\chi^2(\text{rk}(C))$-Verteilung herleiten. Dabei stellen $\tilde{\Sigma}_u$ und $\tilde{\sigma}$ Least-Squares- oder ML-Schätzer für Σ_u bzw. σ dar. Bei D_K^+ handelt es sich um die Moore-Penrose-Inverse (siehe Anhang A.3) der Duplikationsmatrix D_K (siehe Definition A.3).

2.9.2. Sofortige Kausalität bei VECMs

Bei der Beschreibung der sofortigen Kausalität bei VECMs folgen wir Abschnitt 6.6 und Unterabschnitt 7.6.4 in Lütkepohl, (2005). Die Untersuchung der sofortigen Kausalität bei einem VECM($p-1$)-Modell basiert auf dem VAR(p)-Modell. Bei einem VECM($p-1$)-Modell mit bekannten Parametern genügt daher wie bei VAR-Modellen die Untersuchung der Matrix Σ_u auf eine Blockdiagonalform, die der interessierenden Partitionierung von y_t in z_t und x_t entspricht.

Falls das VECM($p-1$)-Modell unbekannt ist, können wir ein VAR(p)-Modell an die beobachteten Daten anpassen und mit diesem den oben beschriebenen Test auf sofortige Kausalität durchführen.

2.10. Impuls-Response-Analyse

In diesem Abschnitt werden wir untersuchen, wie sich eine Veränderung einer der K Komponenten der Zeitreihe y_t (Impuls) auf die übrigen Komponenten auswirkt (englisch: Response). Ist y_t zum Beispiel eine aus Inflations- und Zinsrate in der EU bestehende Zeitreihe, so kann etwa eine Zinsänderung durch die EZB einen derartigen Impuls darstellen. Die Impuls-Response-Analyse würde in diesem Beispiel helfen, die Auswirkungen einer derartigen Entscheidung auf zukünftige Werte der Inflationsrate abzuschätzen. Der Inhalt dieses Abschnitts basiert auf Unterabschnitt 2.3.2 sowie den Abschnitten 6.7 und 7.7 in Lütkepohl, (2005).

Wir betrachten zunächst ein VAR(1)-Modell mit der MA-Darstellung (2.11). Um einen etwaigen Effekt einer Variablenänderung isoliert untersuchen zu können, nehmen wir an, dass $y_t = \mu = 0$ bzw. $u_t = 0$ für $t < 0$ gilt. Das bedeutet, dass y_t dem Modell $y_t = \nu + A_1 y_{t-1} + u_t$ mit $\nu = 0$ folgt. Zum Zeitpunkt 0 verändern wir die k-te Variable, indem wir $u_0 = e_k$ setzen. Dabei ist e_k der k-te Einheitsvektor. Störterme in der Zukunft wollen wir ausschließen und nehmen daher $u_1 = u_2 = \cdots = 0$ an.

Damit ist $y_0 = e_k$ und zum Zeitpunkt 1 nimmt die Zeitreihe wegen $y_1 = A_1 e_k$ den Wert der k-ten Spalte von A_1 an. Zum Zeitpunkt 2 gilt $y_2 = A_1 y_1 =$

$A_1^2 e_k$, d.h. die Zeitreihe erreicht den Wert der k-ten Spalte von A_1^2. Dieses Muster setzt sich mit wachsendem t fort und wir erhalten $y_t = A_1^t e_k$.

Wegen (2.5) gilt $A_1^i = \phi_i$ und damit $y_t = \phi_i e_k$. Für VAR(p)-Modelle kann Ähnliches gezeigt werden. Unter Rückgriff auf (2.3) folgt $Y_t = \boldsymbol{A}^i E_k$ mit $E_k = \begin{pmatrix} e_k' & 0 & \dots & 0 \end{pmatrix}'$ analog zum VAR(1)-Fall. Damit gilt für die interessierende Zeitreihe $y_t = J\boldsymbol{A}^i E_k$ mit J wie in (2.10). Mit $E_k = J'JE_k = J'e_k$ und (2.12) folgt $y_t = J\boldsymbol{A}^i J'e_k = \phi_i e_k$, d.h. auch beim VAR($p$)-Modell gilt unter der Annahme

$$u_t = \begin{cases} e_k & \text{für } t = 0 \\ 0 & \text{sonst,} \end{cases}$$

dass y_t für $t \geq 0$ den Wert der k-ten Spalte von ϕ_t annimmt.

Eine Alternative zu $u_0 = e_k$ ist die Wahl $u_0 = \sigma_k e_k$, wobei σ_k die Wurzel des k-ten Diagonalelements von Σ_u darstellt. Dies dient der Analyse des Effekts auf y_t, wenn die k-te Komponente der Zeitreihe um ihre Standardabweichung erhöht wird. Mit dieser Wahl von u_0 erhalten wir für y_t das σ_k-fache der k-ten Spalte von ϕ_t.

Unabhängig von den beiden angesprochenen Alternativen für die Wahl von u_0 bleibt ein Impuls in der k-ten Variable folgenlos für alle übrigen Komponenten von y_t, wenn die k-te Spalte von ϕ_t ein Vielfaches von e_k für alle $t \geq 0$ ist. Durch Umordnung der Komponenten erkennen wir eine Ähnlichkeit zur Kausalität im Sinne von Granger. Der Impuls einer Variable bleibt ohne Auswirkung auf die übrigen Komponenten, wenn sie nicht Granger-kausal für die Menge der übrigen Variablen ist.

Die Impuls-Response-Analyse lässt sich auch auf eine Responsevariable beschränken. Die durch eine Veränderung in $y_{k,0}$ hervorgerufene Änderung in $y_{j,t}$ wird durch $\phi_{jk,t}$ angegeben. Also bleibt ein Impuls in der k-ten Variable für die j-te Komponente ohne Auswirkung, wenn $\phi_{jk,t} = 0$ für alle $t \geq 0$. Laut Proposition 2.4 in Lütkepohl, (2005) muss diese Bedingung für ϕ_t nicht für alle $t > 0$ erfüllt sein. Es genügt demnach, wenn die Bedingung für alle $t \in \{1, 2, \dots, (K-1)p\}$ erfüllt ist.

Statt des Effekts auf eine Variable $\phi_{jk,t}$ zu einem bestimmten Zeitpunkt $t > 0$ kann auch der kumulierte Effekt $\sum_{t=1}^{n} \phi_{jk,t}$ bis zum Zeitpunkt n betrachtet

werden. Laut (2.13) konvergiert diese Größe bei stabilen VAR-Prozessen für $n \to \infty$. Daraus folgt, dass die Wirkung $\phi_{jk,t}$ eines Impulses mit wachsendem t gegen 0 konvergiert, und somit verebbt.

Auch für VECM$(p-1)$-Modelle lässt sich eine Impuls-Response-Analyse durchführen. Dazu können wir gemäß (2.18) die Umwandlung in ein VAR(p)-Modell vollziehen. Dessen MA-Darstellung kann – wie im VAR-Kontext beschrieben – für die Analyse verwendet werden. So gibt $\phi_{jk,t}$ Aufschluss darüber, wie sehr sich ein Impuls in der k-ten Variable zum Zeitpunkt 0 auf den Wert der j-ten Variable zum Zeitpunkt t auswirkt. Ein wesentlicher Unterschied zum stabilen VAR-Modell ist, dass nicht notwendigerweise $\phi_{jk,t} \to 0$ gilt. Die Auswirkungen eines Impulses können also von Dauer sein.

Auf S. 62 f. in Lütkepohl, (2005) werden potenzielle Fehlerquellen bei der Impuls-Response-Analyse diskutiert. Eine wichtige Voraussetzung, um fehlerhafte Ergebnisse zu vermeiden ist die Berücksichtigung aller relevanten Daten. Umfasst y_t nicht alle relevanten Variablen, liefert die Impuls-Response-Analyse unter Umständen ein falsches Bild.

3. Anwendung mit dem Python-Modul statsmodels

3.1. Statsmodels

In diesem Kapitel beschreiben wir die computergestützte Anwendung der oben beschriebenen Theorie mit dem für *statsmodels* geschriebenen VECM-Modul. Dazu soll dieser Abschnitt zunächst einen groben Überblick über den VECM-bezogenen Teil von *statsmodels* liefern. Ehe wir ein praktisches Beispiel in Angriff nehmen, welches sich über die Abschnitte 3.3 bis 3.10 erstreckt, werden in Abschnitt 3.2 entsprechende Konventionen und Voraussetzungen festgelegt. Abschließend befassen wir uns in Abschnitt 3.11 mit der Herausforderung, die Korrektheit des VECM-Moduls sicherzustellen.

Statsmodels ist ein auf `https://github.com/statsmodels/statsmodels` verwaltetes, in der Sprache Python geschriebenes Open-Source-Projekt. Seine Selbstbeschreibung auf `http://www.statsmodels.org/stable/index.html` lautet

> **statsmodels** is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration. An extensive list of result statistics are available for each estimator. The results are tested against existing statistical packages to ensure that they are correct. The package is released under the open source Modified BSD (3-clause) license.

Im Rahmen dieses Kapitels werden wir speziell auf die Funktionalität für VECMs eingehen. Ein Teil der Ordnerstruktur von *statsmodels* wird im Dateibaum in Abbildung 3.1 veranschaulicht. Darin sind nur Dateien und

Ordner mit VECM-Bezug enthalten. Für diese Arbeit nicht relevante Teile des Projekts sind ausgeblendet und werden jeweils durch drei Punkte (...) angedeutet. Ordner sind durch einen Schrägstrich (/) am Ende des Namens gekennzeichnet.

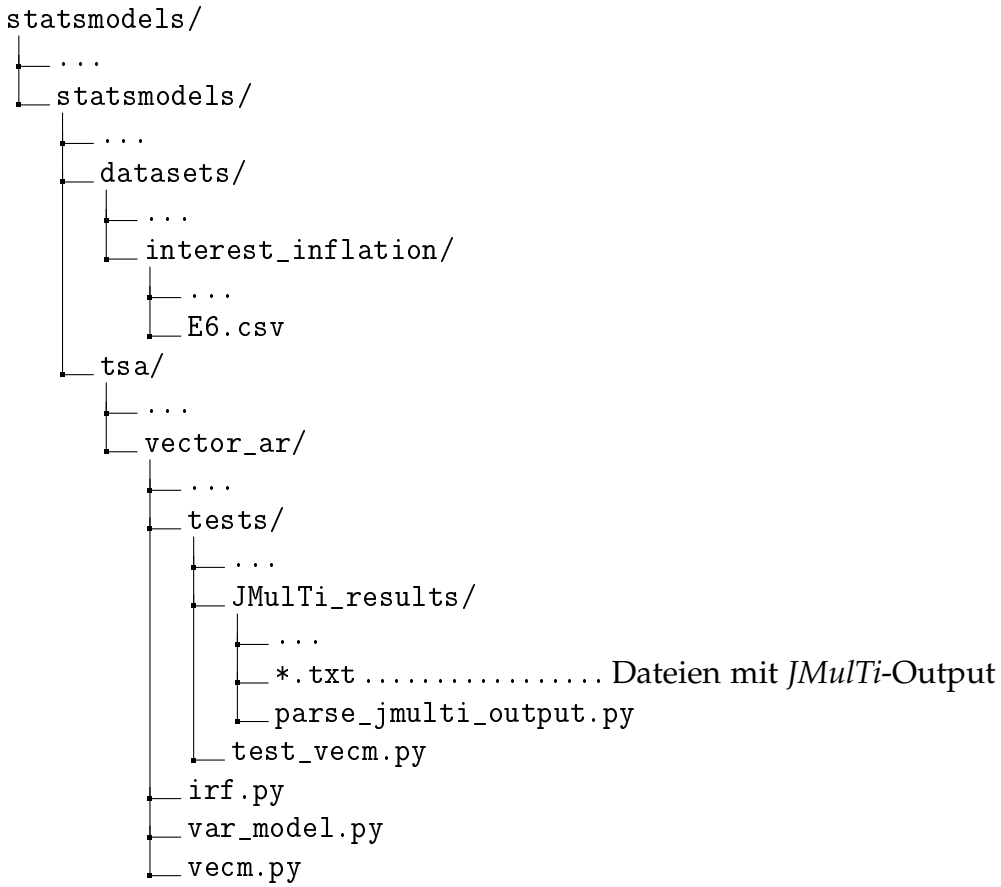

Abbildung 3.1.: Dateibaum von statsmodels

Bevor wir diesen Dateibaum diskutieren, weisen wir darauf hin, dass *statsmodels* aktiv weiterentwickelt wird und sich somit im Laufe der Zeit diese Dateistruktur sowie der Inhalt der Dateien ändern kann. Da sich derartige Änderungen auf die in diesem Kapitel folgenden Codebeispiele auswirken können, legen wir fest, dass die gezeigten Beispiele auf jenem Code basieren, der sich direkt nach der Aufnahme des VECM-Moduls im Hauptentwicklungszweig von *statsmodels* befand. Die Integration des VECM-Moduls

erfolgte mit jenem Commit, der durch den mit 09bd0c5cdd0 beginnenden SHA-1-Hash gekennzeichnet ist (siehe https://github.com/statsmodels/statsmodels/commit/09bd0c5cdd0e0da9281cde983653969c0557ce5f). Dank des Versionsverwaltungssystems *git* ist es möglich, eine lokale Kopie von *statsmodels* jederzeit auf den Stand dieses Commits zu bringen.

Im obigen Dateibaum können wir erkennen, dass das gesamte Programm im obersten Ordner mit dem Namen `statsmodels/` enthalten ist. In diesem befinden sich unter anderem die Dokumentation, die Installationsdatei und die Open-Source-Lizenz. Für uns relevant ist auf dieser Ebene der Ordner der ebenfalls mit `statsmodels/` benannt ist. In diesem sind vor allem die Ordner `datasets/` sowie `tsa/` interessant.

In `datasets/` befindet sich der Ordner `interest_inflation/`, der wiederum die Datei `E6.csv` beinhaltet. Diese CSV-Datei stellt einen Datensatz vierteljährlicher Zins- und Inflationswerte in (West)Deutschland dar. Der Zeitraum der Beobachtungen erstreckt sich vom zweiten Quartal 1972 bis zum vierten Quartal 1998. Die Daten wurden von http://www.jmulti.de/download/datasets/e6.dat übernommen. Wir erwähnen die CSV-Datei hier, da sie als Grundlage für die Beispiele in diesem Kapitel dienen wird, anhand derer die Funktionalität des VECM-Moduls von *statsmodels* veranschaulicht werden soll. Der gleiche Datensatz wird auch in Lütkepohl, (2005) mehrmals als praktisches Beispiel aufgegriffen und diskutiert. Des weiteren wurde vom Autor der vorliegenden Arbeit anhand dieses Beispiels die Funktionalität des VECM-Moduls auf https://gist.github.com/yogabonito/5461b26bed335cad6907aa4e613acb99#file-gsoc-vecm-ipynb demonstriert.

Der Ordner namens `tsa/` bietet verschiedene für die Zeitreihenanalyse notwendige Funktionen. Teil dieses Ordners ist `vector_ar/`. Im darin befindlichen `var_model.py` sind Funktionen implementiert, die das Arbeiten mit VAR-Modellen ermöglichen. In `irf.py` wurde jener Code ausgelagert, der für die Impuls-Response-Analyse gedacht ist. In `vecm.py` befindet sich die VECM-bezogene Funktionalität von *statsmodels*.

Auf der selben Ebene im Dateibaum liegt der Ordner `tests/`. Dessen Unterordner `JMulTi_results/` enthält zahlreiche Textdateien. Für sie scheint stellvertretend `*.txt` im Dateibaum auf. Sie sind der Output von *JMulTi*, einem Open-Source-Programm für die Analyse von Zeitreihen (vgl. Lütkepohl und Krätzig, (2004)). *JMulTi* bietet eine grafische Benutzeroberfläche,

die in Java geschrieben ist. Die Berechnungen wurden in der Programmiersprache GAUSS implementiert. Der auf S. 53 zitierten Selbstbeschreibung von *statsmodels* entsprechend, lag ein Ziel bei der Entwicklung des VECM-Moduls darin, die Ergebnisse mit Resultaten bereits existierender Software abzugleichen, um die Korrektheit des Programms sicherzustellen. Die von *JMulTi* produzierten Textdateien erlauben derartige Tests. Das Python-Skript `parse_jmulti_output.py` stellt die Einleseprozedur für diese Textdateien zur Verfügung und legt damit den Grundstein für die Automatisierung der Tests.

Die eingelesenen Werte werden von `test_vecm.py` genutzt. Diese Datei veranlasst das VECM-Modul von *statsmodels*, ebenfalls Ergebnisse zu berechnen. In weiterer Folge vergleicht `test_vecm.py` die eingelesenen Resultate von *JMulTi* mit den von *statsmodels* berechneten Werten. Auf das Starten dieser Testprozedur und ihre Ergebnisse wird in Abschnitt 3.11 eingegangen.

3.2. Konventionen und Voraussetzungen

3.2.1. Konventionen für abgebildete Codeausschnitte

In den folgenden Unterabschnitten werden vermehrt Anweisungen in der Programmiersprache Python aufgelistet. Dazu möchten wir an dieser Stelle die Konventionen im Zusammenhang mit der Abbildung von Codeausschnitten festlegen. Ein Befehl in Python mit dazugehörigem Output wird in Anlehnung an die Notebook-Umgebung von *Jupyter* (siehe http://jupyter.org/) dem Schema

```
In [1]: def meine_funktion(argument):
            print(argument)
            return argument + 1
        meine_funktion(1)

        1

Out[1]: 2
```

folgen. Der graue Bereich ist eine Eingabe- bzw. Inputzelle. Sie wird durch `In [j]:` eingeleitet, wobei $j \in \mathbb{N}$ die Befehle nummeriert. Diese Nummerierung `[j]` werden wir nutzen, um uns auf bestimmte Codeausschnitte zu beziehen. Es sei darauf hingewiesen, dass dies lediglich eine Markierung der Eingabezelle darstellt und nicht Teil des eingegebenen Codes ist. Der auszuführende Python-Code ist in der grauen Eingabezelle abgebildet. Im Fall von `[1]` ist dies die Definition der einparametrigen Funktion `meine_funktion`, die ihr Argument zunächst mit der `print`-Funktion ausgibt und anschließend um eins erhöht zurückgibt. Diese Funktion wird in der vierten Zeile der Eingabe mit dem Argument `1` aufgerufen.

Wir nutzen die Zelle ohne Beschriftung zu ihrer Linken, die sich direkt unter der Eingabezelle befindet, um Ausgaben der `print`-Funktion anzuzeigen und nennen sie dementsprechend Printzelle. Im obigen Beispiel wird an dieser Stelle mit `1` das Ergebnis des `print`-Aufrufs in der Funktion `meine_funktion` sichtbar.

Die letzte der drei Zellen dient der Anzeige des Rückgabewertes der letzten Inputzeile. Im obigen Beispiel enthält diese Zelle der Returnwert von `meine_funktion(1)`, also `2`. Wir nennen diese Zelle Ausgabe- bzw. Outputzelle. Sie wird immer mit `Out[j]:` eingeleitet, wobei $j \in \mathbb{N}$ der Nummer der Inputzelle entspricht.

Führt der Code in der Eingabezelle zu keinem `print`-Aufruf oder zu keinem Rückgabewert in seiner letzten Zeile, so fehlt die entsprechende Zelle unter der Eingabezelle, wie etwa in `[2]` oder `[3]`.

```
In [2]: meine_variable = 1  # kein print & kein Rückgabewert

In [3]: print("test")  # kein Rückgabewert

        test
```

Genau genommen hat jede Funktion in Python – also auch `print` – einen Returnwert. Gibt eine Funktion nicht explizit einen Wert zurück, so ist der Returnwert `None`. Die Ausgabe dieses Objekts wird jedoch in Pythons Kommandozeilenumgebung sowie in Jupyter unterdrückt und daher wird es auch in der vorliegenden Arbeit nicht abgebildet. Die Verwendung der `print`-Funktion in `[1]` und `[3]` impliziert, dass der gezeigte Code in Python

3 geschrieben wurde – in Python 2 handelt es sich bei `print` um keine Funktion. Auch wenn *statsmodels* die Kompatibilität zu Python 2.7 gewährleistet, setzen wir in der Folge Python 3 voraus, da wir die `print`-Funktion verwenden werden.

3.2.2. Voraussetzungen für die Reproduzierbarkeit der Resultate

Um die Reproduzierbarkeit der in der Folge abgebildeten Codebeispiele zu gewährleisten, wird nun auf die entsprechenden Voraussetzungen hingewiesen. Es wird angenommen, dass die in diesem Kapitel verwendeten Python-Module installiert sind. Um *statsmodels* in der hier betrachteten Version zu installieren, ist ein Download mittels `git clone https://github.com/statsmodels/statsmodels.git` nötig. Im damit erstellten Ordner erreichen mit `git checkout 09bd0c5cdd0` die Rücksetzung auf den Stand direkt nach der Aufnahme des VECM-Moduls in das Open-Source-Projekt. Die Installation erfolgt mit dem Befehl `python setup.py install` im selben Ordner.

Es wird auch angenommen, dass die browserbasierte Notebookumgebung von Jupyter für die Ausführung der Befehle zum Einsatz kommt. Da die in diesem Kapitel abgebildeten Codeausschnitte ein durchgehendes Beispiel bilden, setzen wir weiters voraus, dass sie in der gezeigten Reihenfolge eingegeben werden. Wir beginnen mit den folgenden Importen.

```
In [4]: %matplotlib inline
        import numpy as np
        from statsmodels.tsa.vector_ar.vecm import *
        import statsmodels.datasets.interest_inflation.data as d
        import pandas
```

Die erste Anweisung in [4] ist Jupyter- bzw. IPython-spezifisch und würde ohne spezielle Behandlung durch die Entwicklungsumgebung zu einem `SyntaxError` führen. Sie stellt sicher, dass vom Programm erstellte Plots in Printzellen angezeigt werden. In der vorliegenden Arbeit werden Plots nicht in Printzellen abgebildet sondern gesondert in Form von Abbildungen. In der zweiten Zeile importieren wir mit `numpy` (vgl. Walt et al., (2011)) eine Python-Bibliothek für Matrizen- und allgemeine Arrayoperationen. Die

dritte Zeile importiert alle Funktionen und Klassen aus `vecm.py` im Ordner `vecm/` und macht sie darüber hinaus über ihren Namen (ohne Angabe des Moduls) direkt aufrufbar. Die vierte Zeile sorgt dafür, dass `data.py` im Ordner `interest_inflation/` vom Python-Interpreter importiert und das Modul der Variable `d` zugewiesen wird. Dieses gibt uns mit seiner Funktion `load_pandas` Zugriff auf die Rohdaten in Form eines `DataFrame`-Objekts. Die Daten sind in `E6.csv` im Ordner interest_inflation/ gespeichert, welche die Form der Tabelle 3.1 hat. Abschließend importieren wir in [4] `pandas` (vgl. McKinney, (2010)) für die Verwendung in der nächsten Eingabezelle.

year	quarter	Dp	R
1972	2	−0.00313258	0.083
1972	3	0.0188713	0.083
1972	4	0.0248036	0.087
1973	1	0.0162776	0.087
⋮	⋮	⋮	⋮
1998	4	0.0239234	0.038

Tabelle 3.1.: Ein Ausschnitt aus der csv-Datei mit den Daten.

In Tabelle 3.1 stellen die Werte in den mit *Dp* und *R* betitelten Spalten vierteljährliche Inflations- bzw. für ein Jahr geltende Zinsraten dar. Wir lesen diese Daten in [5] ein und speichern die Werte der letzten beiden Spalten in der Variable `data`.

```
In [5]: df = d.load_pandas().data
        dates = df[["year", "quarter"]].astype(int).astype(str)
        quarterly = dates["year"] + "Q" + dates["quarter"]
        from statsmodels.tsa.base.datetools import \
            dates_from_str
        quarterly = dates_from_str(quarterly)
        data = df[["Dp", "R"]]
        data.index = pandas.DatetimeIndex(quarterly)
```

Damit ist `data` ein `DataFrame`-Objekt, das die Form von Tabelle 3.2 besitzt, wobei die angezeigten Werte zum Teil gerundet sind. Wie wir auf Seite 74 zeigen, können wir diese Daten mit *statsmodels* einfach veranschaulichen. Den entsprechenden Plot enthält Abbildung 3.2.

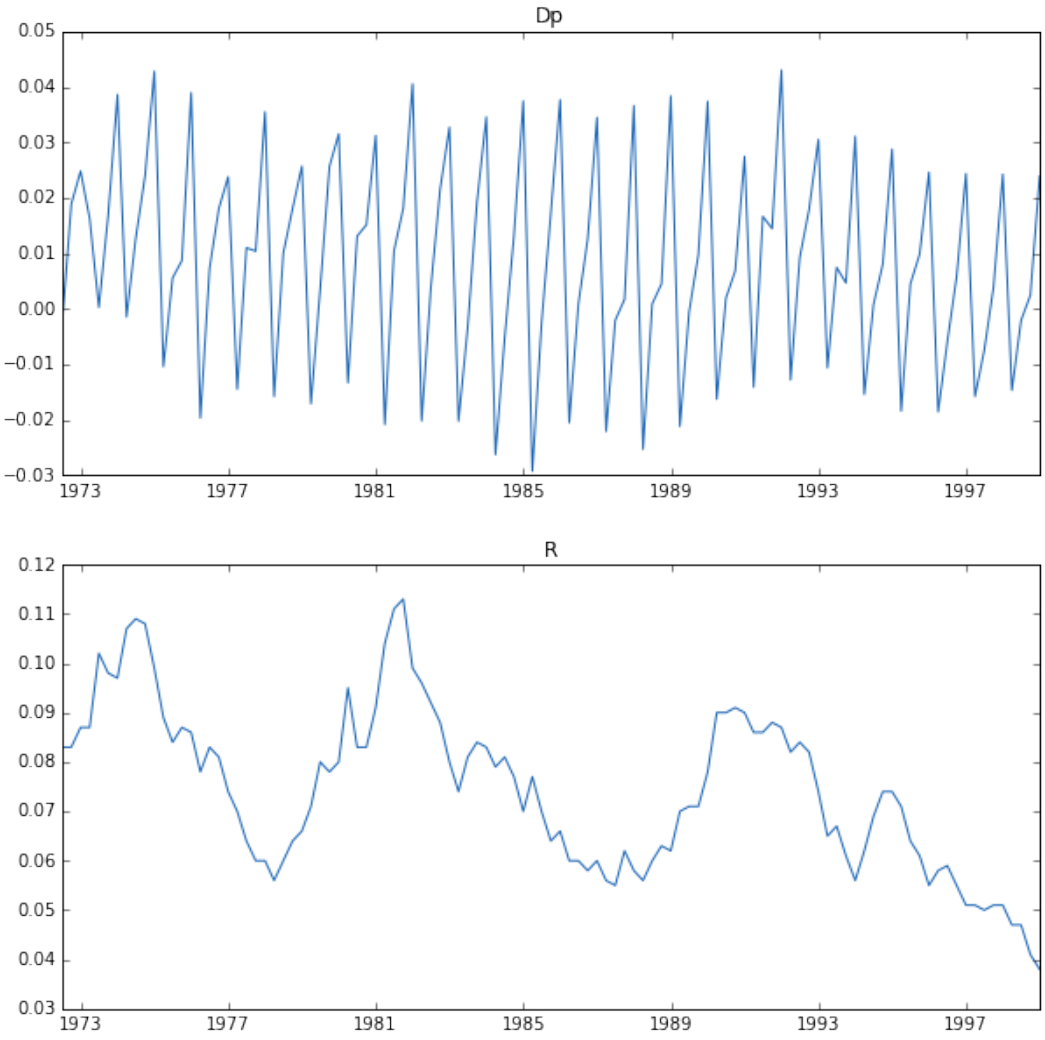

Abbildung 3.2.: Zeitreihen für Inflation (Dp) und Zinsrate (R).

	Dp	R
1972 – 06 – 30	−0.003133	0.083
1972 – 09 – 30	0.018871	0.083
1972 – 12 – 31	0.024804	0.087
1973 – 03 – 31	0.016278	0.087
⋮	⋮	⋮
1998 – 12 – 31	0.023923	0.038

Tabelle 3.2.: Ein Ausschnitt des `DataFrame`-Objekts mit den importierten Daten.

3.3. Modellspezifikation

3.3.1. Deterministische Terme

Vor der Schätzung der Parameter ist das VECM zu spezifizieren. Dies umfasst zum einen die Festlegung der Modellordnung $p-1$. Zum anderen ist der Kointegrationsrang r zu bestimmen. Diese beiden Voraussetzungen sind Inhalt der nachfolgenden zwei Unterabschnitte.

Wie in Unterabschnitt 2.3.2 erläutert, werden für die Wahl der Modellordnung VAR(p)-Modelle mit verschiedenen Werten für p betrachtet. Es ist zu beachten, dass diese VAR(p)-Modelle auch von der Wahl deterministischer Terme abhängen und somit die realisierten Informationskriterien davon ebenfalls beeinflusst werden. Daher werden wir nun die deterministischen Terme für unser VECM wählen. Betrachten wir die Zeitreihe in Abbildung 3.2, so fallen die stark ausgeprägten saisonalen Schwankungen in der Inflationsrate `Dp` auf. Daher werden wir saisonale deterministische Terme in unser VECM einfließen lassen. Bei der Zinsrate ist gut zu erkennen, dass diese im betrachteten Zeitraum stets positiv ist. Aus diesem Grund werden wir auch einen Intercept in unser Modell aufnehmen.

3.3.2. Modellordnung

In Abschnitt 2.3 wurden die theoretischen Grundlagen zur Wahl der Modellordnung $p-1$ eines VECM($p-1$)-Modells erläutert. Dort dargestellte

Rechenwege sind in der Funktion `select_order` im Modul `vecm.py` im Ordner `vecm/` implementiert. Durch den Import aller Funktionen und Klassen von `statsmodels.tsa.vector_ar.vecm` in [4] können wir diese Funktion nun nutzen. Wie diese Funktion zu benutzen ist, erläutert die dazugehörige Hilfe. Diese kann durch `help(select_order)` oder `print(select_order.__doc__)` aufgerufen werden.

```
In [6]: help(select_order)
```

```
Help on function select_order in module statsmodels.tsa.vector_ar.
    vecm:

select_order(data, maxlags, deterministic='nc', seasons=0,
             exog=None, exog_coint=None)
Compute lag order selections based on each of the available
information criteria.

Parameters
----------
data : array-like (nobs_tot x neqs)
    The observed data.
maxlags : int
    All orders until maxlag will be compared according to the
    information criteria listed in the Results-section of this
    docstring.
deterministic : str {``"nc"``, ``"co"``, ``"ci"``, ``"lo"``,
                     ``"li"``}
    * ``"nc"`` - no deterministic terms
    * ``"co"`` - constant outside the cointegration relation
    * ``"ci"`` - constant within the cointegration relation
    * ``"lo"`` - linear trend outside the cointegration relation
    * ``"li"`` - linear trend within the cointegration relation

    Combinations of these are possible (e.g. ``"cili"`` or
    ``"colo"`` for linear trend with intercept). See the docstring
    of the :class:`VECM`-class for more information.
seasons : int, default: 0
    Number of periods in a seasonal cycle.
exog : ndarray (nobs_tot x neqs) or `None`, default: `None`
    Deterministic terms outside the cointegration relation.
exog_coint : ndarray (nobs_tot x neqs) or `None`, default: `None`
    Deterministic terms inside the cointegration relation.

Returns
-------
selected_orders :
    :class:`statsmodels.tsa.vector_ar.var_model.LagOrderResults`
```

In der zweiten gedruckten Zeile dieses Hilfetextes erkennen wir, dass der Funktion die zwei Argumente `data` sowie `maxlags` übergeben werden müs-

sen. Für die anderen Parameter existieren Defaultwerte nach den Gleichheitszeichen.

Ein Blick in den *Parameters* betitelten Abschnitt der Hilfe verrät, dass es sich bei `data` um ein Objekt handeln muss, dass zu einem Numpy-`array` kompatibel ist und die Dimension `nobs_tot`×`neqs` aufweist. Dabei bezeichnet `nobs_tot` als Abkürzung für *number of observations (total)* die Anzahl der Beobachtungen inklusive Presample. Mit `neqs` (*number of equations*) wird die Anzahl der Variablen in der Zeitreihe bezeichnet. In Kapitel 2 nannten wir diese Größe K. Die Erklärung zu `data` in der Hilfe macht deutlich, dass mit diesem Parameter die beobachteten Daten an die Funktion `select_order` übergeben werden. Diese sind in unserer Python-Sitzung nach `[5]` in der Variable `data` gespeichert.

Der zweite Parameter ist laut Hilfetext vom Typ `int`, also eine ganze Zahl. Damit legen wir fest, wie viele Modellordnungen miteinander verglichen werden. In Unterabschnitt 2.3.2 wurde dieser Wert mit M bezeichnet. Wir werden eine maximale Modellordnung von $p - 1 = 10$ annehmen.

Die nächsten beiden Parameter die in der Printzelle `[6]` zu finden sind, erlauben die Angabe von deterministischen Werten. Wir werden den Intercept mit Hilfe des Parameters `deterministic` an `select_order` übergeben. Dieser Parameter kann einen von fünf verschiedenen Werten annehmen. Während `"nc"` keinen deterministischen Wert darstellt, repräsentieren `"ci"` und `"co"` Konstanten und `"li"` und `"lo"` lineare Trends. Um den Unterschied zwischen den mit *i* und *o* endenden Argumenten zu verstehen, lohnt sich ein Blick auf Gleichung (2.44). In deren Schreibweise können deterministische Terme sowohl über D_{t-1}^{co} als auch D_t in das Modell einfließen. Wir nennen deterministische Terme in D_{t-1}^{co} *auf die Kointegrationsbeziehung* $\alpha\beta' y_{t-1}$ *beschränkt*. In (2.43) haben wir gezeigt, dass dies der Ort ist, an dem sich eine Konstante in der betrachteten Zeitreihe y_t bemerkbar macht. Wir werden `"ci"` als Argument wählen, da dieses Terme in D_{t-1}^{co} spezifiziert (*i* in `"ci"` und `"li"` steht für *inside the cointegration relation*; *o* in `"co"` und `"lo"` für *outside*).

Um die quartalsmäßigen Schwankungen zu berücksichtigen, legen wir für das `seasons`-Argument den Wert 4 fest. Die weiteren Parameter der Funktion `select_order` stellen Möglichkeiten dar, beliebige deterministische Terme außerhalb (`exog`) und in der Kointegrationsbeziehung (`exog_coint`)

anzugeben. Wir werden hierfür die Defaultwerte nutzen, wie in der folgenden Eingabezelle zu sehen ist.

```
In [7]: order = select_order(data=data, maxlags=10,
                             deterministic="ci", seasons=4)
```

Der unterste Abschnitt des in [6] ausgegebenen Hilfetextes gibt den Rückgabewert von `select_order` an. Dabei handelt es sich um ein Objekt vom Typ `LagOrderResults`. Dieses bietet mit seiner `summary()`-Methode eine Auflistung von Informationskriterien zu verschiedenen Modellordnungen. Mit dem Aufruf von `order.summary()` erhalten wir daher die Tabelle 3.3.

VECM Order Selection (highlights the minimums)

	AIC	BIC	FPE	HQIC
0	−20.67	−20.30*	$1.055e-09$	−20.52*
1	−20.68	−20.19	$1.050e-09$	−20.48
2	−20.56	−19.97	$1.182e-09$	−20.32
3	−20.76*	−20.06	$9.717e-10$*	−20.47
4	−20.64	−19.84	$1.089e-09$	−20.32
5	−20.60	−19.69	$1.143e-09$	−20.23
6	−20.63	−19.62	$1.109e-09$	−20.22
7	−20.44	−19.32	$1.350e-09$	−19.98
8	−20.33	−19.10	$1.514e-09$	−19.83
9	−20.39	−19.06	$1.428e-09$	−19.85
10	−20.14	−18.70	$1.844e-09$	−19.56

Tabelle 3.3.: Bestimmung der Modellordnung nach verschiedenen Informationskriterien.

Ihre Zeilen sind mit 0 bis 10 nummeriert. Jede dieser Zeilennummern entspricht einem möglichen Wert für die Modellordnung $p-1$. Die vier Spalten neben dieser Nummerierung stehen für jeweils ein Informationskriterium. Die ersten beiden Spalten umfassen AIC und BIC, während das Kriterium HQ in der letzten Spalte abgebildet ist. FPE wird in der vorliegenden Arbeit nicht behandelt. Für dieses Kriterium sei auf Lütkepohl, (2005) verwiesen. Also verbindet die von `select_order` produzierte Tabelle mögliche Modellordnungen mit ihren Bewertungen in Form von Informationskriterien. In jeder Spalte wird der minimale Wert mit einem Stern markiert. Damit können

wir die nach einem bestimmten Informationskriterium zu wählende Modellordnung ablesen, ohne alle Werte in einer Spalte miteinander vergleichen zu müssen.

Neben der Ausgabe dieser Tabelle bietet ein `LagOrderResults`-Objekt auch den direkten Zugriff auf die zu wählende Modellordnung. Dies wird in [8] gezeigt. Dort sehen wir, dass AIC und FPE die Modellordnung drei nahelegen, während die Kriterien BIC und HQIC für eine Modellordnung von null sprechen.

```
In [8]: order.aic, order.bic, order.fpe, order.hqic
```

```
Out[8]: (3, 0, 3, 0)
```

Wir werden dem AIC folgend $p - 1 = 3$ wählen. Der Entscheidung zugunsten des AIC liegt die Überlegung zugrunde, dass anhand des größeren Modells etwas mehr von der Funktionalität des VECM-Moduls gezeigt werden kann.

3.3.3. Kointegrationsrang

An der Implementation von Tests für den Kointegrationsrang r arbeitete Josef Perktold, einer der Hauptentwickler von *statsmodels*, bereits vor dem Beginn des hier beschriebenen Projekts. Sein Code im Pull Request 453 auf https://github.com/statsmodels/statsmodels/pull/453/files basiert wiederum auf einer Implementation in Matlab durch LeSage, welche unter http://www.spatial-econometrics.com/html/jplv7.zip zum Download bereit steht.

Der Code aus dem erwähnten Pull Request wurde im Wesentlichen auch in das neue VECM-Modul übernommen. Er befindet sich in der Funktion `coint_johansen`, welche mit den Beobachtungen, der Ordnung der deterministischen Terme und der Modellordnung $(p - 1)$ aufgerufen wird. Die Ordnung der deterministischen Terme ist 0 für einen Intercept, 1 für einen linearen Term und -1, wenn kein deterministischer Term vorhanden ist. Das von der Funktion zurückgegebene Objekt bietet Zugriff auf die Trace-Teststatistik 2.24 sowie die Maximum-Eigenvalue-Teststatistik 2.23 und die

entsprechenden kritischen Werte. Wir betrachten in den folgenden Codeausschnitten die Wahl des Kointegrationsranges mit Hilfe der Trace-Teststatistik. Die Teststatistiken $\lambda_{LR}(0,2)$ und $\lambda_{LR}(1,2)$ werden in [9] ausgegeben.

```
In [9]: coint_results = coint_johansen(data, 0, 3)
        coint_results.lr1  # trace statistic

Out[9]: array([ 17.17287503,   3.0323193 ])
```

Wir testen – wie in Abschnitt 2.4 beschrieben – zuerst $H_0 : r = 0$ gegen $H_1 : 0 < r \leq K$. Auf den kritischen Wert zur oben berechneten Teststatistik $\lambda_{LR}(0,2) = 17.17$ greifen wir in [10] zu.

```
In [10]: coint_results.cvt[0]

Out[10]: array([ 13.4294,  15.4943,  19.9349])
```

Wir sehen, dass drei verschiedene kritische Werte ausgegeben werden. Dies liegt daran, dass die drei angezeigten Werte zu verschiedenen Signifikanzniveaus gehören (10 %, 5 % und 1 %). Wir führen den Test zu einem Niveau von 5 % durch. Da die Teststatistik über dem kritischen Wert liegt $(17,17 > 15,49)$ verwerfen wir die Nullhypothese. Es folgt daher der nächste Test mit $r_0 = 1$. Der kritische Wert liegt bei diesem Test laut [11] bei 3.84.

```
In [11]: coint_results.cvt[1][1]

Out[11]: 3.8414999999999999
```

Das bedeutet, dass die Teststatistik mit einem Wert von $3,03 < 3,84$ nicht im kritischen Bereich liegt. Wir akzeptieren somit $r = 1$ als Kointegrationsrang.

Die Funktion `coint_johansen` ist eigentlich nicht für den Endnutzer gedacht, da ihr Rückgabewert keinen direkten Zugriff auf den zu wählenden Kointegrationsrang erlaubt. Das heißt, dass der Nutzer für das Vergleichen von Teststatistiken und kritischen Werten und die daraus gezogenen Schlüsse verantwortlich ist. Daher steht alternativ die Funktion `select_coint_rank` zur Verfügung, die diese Aufgaben übernimmt. Sie greift intern auf `coint_johansen` zurück.

```
In [12]: rank = select_coint_rank(data, 0, 3, method="trace",
                                  signif=0.05)
```

Das in [12] von `select_coint_rank` zurückgegebene Objekt bietet etwa eine `summary`-Methode, mit der wir eine tabellarische Ausgabe des Kointegrationstests erhalten. Das bedeutet, dass nach dem Ausführen von [12] der Aufruf von `rank.summary()` die Tabelle 3.4 erzeugt.

Johansen cointegration test using trace test statistic with 5% significance level

r_0	r_1	test statistic	critical value
0	2	17.17	15.49
1	2	3.032	3.841

Tabelle 3.4.: Tests für die Bestimmung des Kointegrationsranges.

Darin werden alle durchgeführten Tests zusammengefasst. Wir sehen, dass die Teststatistik beim ersten Test ($r = r_0 = 0$ gegen $0 < r \leq r_1 = 2$) im kritischen Bereich liegt. Daher folgt in der Zeile darunter der nächste Test ($r = r_0 = 1$ gegen $r = r_1 = 2$). Hier wird liegt die Teststatistik außerhalb des kritischen Bereichs und damit ist der zu wählende Kointegrationsrang 1. Auf dieses Ergebnis können wir über das von `select_coint_rank` zurückgegebene Objekt direkt zugreifen, wie [13] zeigt.

```
In [13]: rank.rank
```

```
Out[13]: 1
```

3.4. Parameterschätzung

In Abschnitt 3.3 haben wir das Modell spezifiziert. Demnach wählen wir für die Modellordnung $p - 1$ in unserem Beispiel den Wert 3 und als Kointegrationsrang den Wert 1. Als Spezialfall von Gleichung (2.44) lautet

das Modell

$$\Delta y_t = \alpha \begin{pmatrix} \beta' & \eta' \end{pmatrix} \begin{pmatrix} y_{t-1} \\ D^{co}_{t-1} \end{pmatrix} + \Gamma_1 \Delta y_{t-1} + \Gamma_2 \Delta y_{t-2} + \Gamma_3 \Delta y_{t-3} + C D_t + u_t.$$

Dabei nehmen wir an, dass die Matrix $\Pi = \alpha\beta'$ den Rang 1 hat. D^{co}_{t-1} bzw. D_t bestehen aus Dummy-Variablen, die den Intercept bzw. die saisonalen Effekte repräsentieren.

Nachdem das VECM vollständig spezifiziert ist, können wir nun die Parameter des Modells schätzen. Dazu initialisieren wir in *statsmodels* ein Objekt vom Typ `VECM` mit den in Abschnitt 3.3 gewählten deterministischen Termen und Werten für Modellordnung sowie Kointegrationsrang. Analog zu `select_order` kann mittels `help(VECM)` eine Erklärung zur Verwendung dieser Klasse aufgerufen werden.

Mit der in [5] definierten Variable `data` übergeben wir die beobachteten Daten. Analog zu `select_order` legen wir die deterministischen Terme mittels `deterministic="ci"` und `seasons=4` fest. Die gewünschte Modellordnung geben wir über das Keyword-Argument `k_ar_diff` bekannt. Der Name des Arguments soll verdeutlichen, dass das Programm in der Notation von Kapitel 2 nicht p sondern $p-1$ erwartet, da das Modell 2.15 $p-1$ Lags in den Differenzen Δy enthält. Den Kointegrationsrang geben wir mit `coint_rank=1` an. Wir verwenden daher den folgenden Befehl, um die `VECM`-Instanz einer Variable `model` zuzuweisen.

```
In [14]: model = VECM(data, deterministic="ci", seasons=4,
                      k_ar_diff=3, coint_rank=1)
```

Dieses Objekt erlaubt mit seiner Methode `fit` die in Unterabschnitt 2.5.2 besprochene Maximum-Likelihood-Schätzung der Modellparameter. Wir weisen im folgenden Listing den Returnwert von `fit` einer Variable `vecm_res` zu.

```
In [15]: vecm_res = model.fit(method="ml")
```

Damit sind alle Ergebnisse der Parameterschätzung über das `VECMResults`-Objekt `vecm_res` verfügbar. Der Zugriff auf die Schätzer $\tilde{\beta}$, $\tilde{\alpha}$, $\tilde{\Gamma}$ und $\tilde{\Sigma}_u$ aus den Gleichungen (2.31)-(2.34) erfolgt mit den Befehlen [16]-[19].

```
In [16]: vecm_res.beta
```

```
Out[16]: array([[ 1.         ],
                [-0.25083368]])
```

```
In [17]: vecm_res.alpha
```

```
Out[17]: array([[-0.63174988],
                [ 0.39724572]])
```

```
In [18]: print(vecm_res.gamma[:, :2])  # Schätzer für Γ1
         vecm_res.gamma.round(2)
```

```
         [[-0.33391039  0.06771514]
          [-0.202384    0.27238102]]
```

```
Out[18]: array([[-0.33,  0.07, -0.39, -0.  , -0.35,  0.02],
                [-0.2 ,  0.27, -0.22, -0.02, -0.11,  0.23]])
```

```
In [19]: vecm_res.sigma_u
```

```
Out[19]: array([[  2.30733384e-05,  -1.39899038e-06],
                [ -1.39899038e-06,   2.60161799e-05]])
```

Schön zu erkennen in [16] ist die in (2.40) gezeigte Normierung der oberen $r \times r$-Teilmatrix von β. Diese Forderung stellt wie in Unterabschnitt 2.5.2 dargelegt, die Eindeutigkeit des ML-Schätzers sicher.

Die in Outputzelle [18] gezeigte Matrix fasst von links nach rechts aneinandergereiht die ML-Schätzer für die 2×2-Matrizen Γ_1, Γ_2 und Γ_3 zusammen. Um dies zu verdeutlichen, wurden die ersten beiden Spalten von `vecm_res.gamma` in [18] an die `print`-Funktion übergeben.

Unser Modell verfügt zusätzlich über deterministische Terme. Deren Effekte sind ebenfalls zu schätzen. Die entsprechenden Ergebnisse sind über die Attribute `det_coef` und `det_coef_coint` des `VECMResults`-Objekts `vecm_res` zugänglich.

```
In [20]: vecm_res.det_coef  # saisonale Effekte
```

```
Out[20]: array([[ 0.01620982,  0.0176711 ,  0.03409751],
                [ 0.00735106, -0.00193872, -0.00154358]])
```

```
In [21]: vecm_res.det_coef_coint
```

```
Out[21]: array([[ 0.01067564]])
```

In [20] sind die deterministischen Terme außerhalb der Kointegrationsbeziehung enthalten. Das in der Outputzelle ausgegebene `numpy.ndarray`-Objekt entspricht der Matrix C in der Schreibweise von Gleichung (2.44). In unserem Fall sind dies die quartalsmäßigen Schwankungen. Bestünde die Matrix D_t in (2.44) aus herkömmlichen saisonalen Dummy-Variablen, so würden die Spalten in [20] die notwendige Korrektur für das jeweilige Quartal, ausgehend von einem bestimmten Quartal darstellen. Das würde bedeuten, dass – von Q1 ausgehend – das Modell für die Quartale 2, 3 und 4 die Addition von 0.0162, 0.0177 bzw. 0.034 für die Variable *Dp* vorsieht. Tatsächlich enthält diese Matrix D_t in der Implementierung von *statsmodels* jedoch eine Korrektor um $-\frac{1}{\text{Anzahl der Saisonen}}$, um die gleichen Ergebnisse wie *JMulTi* und das *R*-Package *tsDyn* zu produzieren. Diese Korrektur sorgt für die Orthogonalität der saisonalen Dummy-Variablen zu einem etwaigen konstanten Term.

[21] zeigt die Schätzung für den auf die Kointegrationsbeziehung beschränkten deterministischen Term. In der Notation von Gleichung (2.44) ist dies die Schätzung für η'. Diese Gleichung legt auch die Multiplikation dieses Wertes von links mit $\tilde{\alpha}$ nahe, um den Intercept des Modells zu erhalten (D^{co}_{t-1} hat laut Gleichung (2.43) den Wert 1). Den Intercept berechnen wir in [22].

```
In [22]: vecm_res.alpha.dot(vecm_res.det_coef_coint)  # Intercept
```

```
Out[22]: array([[-0.00674434],
                [ 0.00424085]])
```

Neben den Schätzern bietet `vecm_res` auch den Zugriff auf die entsprechenden Standardfehler, t- und p-Werte. Dazu werden die obigen Befehle derart abgewandelt, dass `stderr_`, `tvalues_` bzw. `pvalues_` vor den Namen des Parameters geschrieben wird. Die folgenden Befehle demonstrieren dieses

Muster anhand des Parameters α.

```
In [23]: vecm_res.stderr_alpha

Out[23]: array([[ 0.16663211],
                [ 0.17693968]])

In [24]: vecm_res.tvalues_alpha

Out[24]: array([[-3.79128545],
                [ 2.24509126]])

In [25]: vecm_res.pvalues_alpha

Out[25]: array([[ 0.00014987],
                [ 0.02476227]])
```

Um eine Übersicht über geschätzte Parameter zu erhalten, führen wir `vecm_res.summary()` aus. Diese Methode liefert ein `statsmodels.iolib.summary.Summary`-Objekt, welches Ergebnisse der Parameterschätzung – in mehrere Abschnitte untergliedert – tabellenförmig aufbereitet. Das Ergebnis zeigen die Tabellen 3.5 sowie 3.6.

Tabelle 3.5 enthält in seiner ersten numerischen Spalte die Einträge der Matrizen $\Gamma_1, \ldots, \Gamma_{p-1}$ und C in Gleichung (2.44). Die Werte in der oberen Teiltabelle stellen dabei die erste Zeile der genannten Matrizen dar. Die zweite Zeile dieser Matrizen ist in der unteren Teiltabelle abzulesen.

Die Werte zu *season1*, *season2* und *season3* in der oberen (unteren) Teiltabelle stellen die erste (zweite) Zeile von C dar.

Die mit *Li* eingeleiteten Tabellenzeilen mit $i \in \{1, 2, 3\}$ gehören zu Γ_i, das in (2.44) mit $L^i \Delta y_t$ multipliziert wird, wobei L den Lag-Operator bezeichnet. Somit gehört beispielsweise *L3.R* in der oberen Teiltabelle zu jenem Element in der ersten Zeile von Γ_3, das mit dem die Zinsrate (R) repräsentierenden Eintrag von Δy_{t-3} multipliziert wird. Da R in unserem Beispiel die zweite der beiden Variablen ist, bezieht sich *L3.R* auf den zweiten Eintrag in der ersten Zeile von Γ_3.

Neben der ersten numerischen Spalte in Tabelle 3.5 folgen die entsprechen-

Det. terms outside the coint. relation & lagged endog. parameters for equation Dp

	coef	std err	z	P>|z|	[0.025	0.975]
season1	0.0162	0.005	3.554	0.000	0.007	0.025
season2	0.0177	0.005	3.690	0.000	0.008	0.027
season3	0.0341	0.005	7.464	0.000	0.025	0.043
L1.Dp	−0.3339	0.141	−2.364	0.018	−0.611	−0.057
L1.R	0.0677	0.095	0.715	0.474	−0.118	0.253
L2.Dp	−0.3874	0.114	−3.399	0.001	−0.611	−0.164
L2.R	−0.0030	0.095	−0.032	0.975	−0.190	0.184
L3.Dp	−0.3457	0.076	−4.524	0.000	−0.495	−0.196
L3.R	0.0204	0.092	0.222	0.824	−0.160	0.201

Det. terms outside the coint. relation & lagged endog. parameters for equation R

	coef	std err	z	P>|z|	[0.025	0.975]
season1	0.0074	0.005	1.518	0.129	−0.002	0.017
season2	−0.0019	0.005	−0.381	0.703	−0.012	0.008
season3	−0.0015	0.005	−0.318	0.750	−0.011	0.008
L1.Dp	−0.2024	0.150	−1.350	0.177	−0.496	0.092
L1.R	0.2724	0.101	2.709	0.007	0.075	0.469
L2.Dp	−0.2180	0.121	−1.802	0.072	−0.455	0.019
L2.R	−0.0164	0.101	−0.162	0.872	−0.215	0.182
L3.Dp	−0.1056	0.081	−1.301	0.193	−0.265	0.053
L3.R	0.2253	0.098	2.303	0.021	0.034	0.417

Tabelle 3.5.: Ergebnisse der Parameterschätzung

den Standardfehler, t- und p-Werte. Die letzten beiden Spalten geben ein 95%-Konfidenzintervall für den jeweiligen Parameter an.

Loading coefficients (α) for equation Dp

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
ec1	−0.6317	0.167	−3.791	0.000	−0.958	−0.305

Loading coefficients (α) for equation R

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
ec1	0.3972	0.177	2.245	0.025	0.050	0.744

Cointegration relations for loading-coefficients-column 1

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
beta.1	1.0000	0	0	0.000	1.000	1.000
beta.2	−0.2508	0.044	−5.671	0.000	−0.338	−0.164
const	0.0107	0.003	3.142	0.002	0.004	0.017

Tabelle 3.6.: Ergebnisse der Parameterschätzung (Fortsetzung von Tabelle 3.5)

Die erste Teil der Tabelle 3.6 bezieht sich auf den ersten Eintrag von α. Die Ergebnisse für den zweiten Eintrag finden sich in der mittleren Teiltabelle. Im dritten Abschnitt von Tabelle 3.6 sind die Resultate zu β und dem auf die Kointegrationsbeziehung beschränkten deterministischen Term abgebildet. In der mit *beta.1* eingeleiteten Zeile wird wieder die Normierung (2.40) deutlich.

Zum Abschluss der Demonstration der Parameterschätzung sei auf eine Problematik im Zusammenhang mit der Bestimmung der Standardfehler hingewiesen. Diese – und damit in weiterer Folge auch die t- und p-Werte – hängen vom in (2.42) definierten Ω, einem Grenzwert in Wahrscheinlichkeit, ab. Dessen Bestimmung wurde bei der Implementierung vermieden, indem

$$\left(\begin{pmatrix}\beta & 0\\ 0 & I_{K(p-1)}\end{pmatrix}\Omega^{-1}\begin{pmatrix}\beta' & 0\\ 0 & I_{K(p-1)}\end{pmatrix}\right), \tag{3.1}$$

in (2.41) durch den konsistenten Schätzer

$$T \begin{pmatrix} Y_{-1}Y'_{-1} & Y_{-1}\Delta X' \\ \Delta X Y'_{-1} & \Delta X \Delta X' \end{pmatrix}^{-1} \tag{3.2}$$

ersetzt wurde (vgl. S. 287 in Lütkepohl, (2005)).

Nach diesem Abschnitt haben wir mit `vecm_res` eine Instanz der Klasse `VECMResults` zur Hand. Damit können wir nun in [26] zeigen, mit welchem Befehl die beobachteten Daten in Abbildung 3.2 auf Seite 60 geplottet wurden.

```
In [26]: vecm_res.plot_data(with_presample=True)
```

3.5. Die VAR-Darstellung und MA-Darstellung eines VECMs

In Kapitel 2 wurde an mehreren Stellen von der Tatsache Gebrauch gemacht, dass sich ein VECM in ein VAR-Modell umwandeln lässt. Das Ergebnis der Umwandlungsvorschrift (2.18) ist über das `VECMResults`-Attribut `var_rep` verfügbar.

```
In [27]: print(vecm_res.var_rep[0])  # A_1
         vecm_res.var_rep

         [[ 0.03433973  0.22617928]
          [ 0.19486172  1.17273842]]
```

```
Out[27]: array([[[ 0.03433973,  0.22617928],
                 [ 0.19486172,  1.17273842]],

                [[-0.0535351 , -0.07072214],
                 [-0.01565624, -0.28875262]],

                [[ 0.04174712,  0.02343397],
                 [ 0.11248905,  0.24162739]],

                [[ 0.34569836, -0.02042696],
                 [ 0.1055512 , -0.22525579]]])
```

Der erste Befehl in [27] zeigt den Zugriff auf eine Parametermatrix – in diesem Fall A_1. Sie ist in der Printzelle zu sehen. Die zweite Eingabezeile liefert das in der Outputzelle ausgegebene `ndarray`-Objekt, welches alle vier Parametermatrizen $A_1, \ldots, A_4$ des VAR(4)-Modells umfasst.

Um Koeffizientenmatrizen der MA-Darstellung zu erhalten, steht uns die Methode `ma_rep` zur Verfügung. Mit dem Argument `maxn` erhalten wir alle ϕ_i für $i \in \{0, \ldots, \texttt{maxn}\}$. [28] zeigt die Matrizen ϕ_i mit $i \leq 2$.

```
In [28]: vecm_res.ma_rep(maxn=2)
```

```
Out[28]: array([[[ 1.        ,  0.        ],
                 [ 0.        ,  1.        ]],

                [[ 0.03433973,  0.22617928],
                 [ 0.19486172,  1.17273842]],

                [[-0.0082822 ,  0.20229393],
                 [ 0.21955708,  1.13063646]]])
```

3.6. Untersuchung der Residuen

Die in Abschnitt 2.6 angesprochenen diagnostischen Verfahren lassen sich ebenfalls mit Hilfe eines `VECMResults`-Objekts durchführen. Mit dessen `test_normality`-Methode testen wir die Annahme der Normalverteilung für das weiße Rauschen u_t. Über das Argument `signif` können wir optio-

nal das Signifikanzniveau des Hypothesentests wählen. Der Aufruf von `vecm_res.test_normality().summary()` erzeugt Tabelle 3.7.

normality (skew and kurtosis) test.
H_0: data generated by normally-distributed process.
Conclusion: fail to reject H_0 at 5% significance level.

Test statistic	**Critical value**	**p-value**	**df**
2.118	9.488	0.714	4

Tabelle 3.7.: Test der Normalverteilungsannahme.

Darin sehen wir, dass der Test in einem p-Wert von 0.714 resultiert. Demnach sprechen die Daten nicht gegen die Normalverteilungsannahme.

Mit der Methode `test_whiteness` können wir die Residuen – wie in Unterabschnitt 2.6.2 beschrieben – einem Test auf Autokorrelation unterziehen. Mit dem Argument `nlags` legen wir die betrachteten Lags fest. In Unterabschnitt 2.6.2 wurde dieser Wert mit h bezeichnet. Mit Hilfe des Arguments `adjusted` weisen wir den Computer an, die alternative Teststatistik $\bar{Q}_h$ zu verwenden. Der Befehl `vecm_res.test_whiteness(nlags=12, adjusted=True).summary()` führt zur Tabelle 3.8. Sie zeigt, dass die Daten nicht gegen die angenommene Freiheit von Autokorrelation des Prozesses u_t sprechen.

Adjusted Portmanteau-test for residual autocorrelation.
H_0: residual autocorrelation up to lag 12 is zero.
Conclusion: fail to reject H_0 at 5% significance level.

Test statistic	**Critical value**	**p-value**	**df**
33.52	48.60	0.491	34

Tabelle 3.8.: Test auf Autokorrelation.

3.7. Vorhersage

Neben dem Zugriff auf die geschätzten Parameter und den gezeigten diagnostischen Tests bietet ein `VECMResults`-Objekt auch die Möglichkeit, Vorhersagen zu berechnen. Die Prognose für die nächsten fünf Perioden zeigt [29].

```
In [29]: vecm_res.predict(steps=5)
```

```
Out[29]: array([[-0.02236238,  0.03961484],
                [-0.00390943,  0.04075971],
                [ 0.00331985,  0.04018359],
                [ 0.02437677,  0.03881549],
                [-0.02489008,  0.04001414]])
```

Dabei umfasst die erste Spalte die Vorhersagen für die Variable *Dp*. Jene für *R* stellen die zweite Spalte des `ndarray`-Objekts in der Ausgabezelle [29] dar. Um nicht nur Punktvorhersagen sondern auch Vorhersageintervalle zu erhalten, verwenden wir zusätzlich das `alpha`-Argument der `predict` -Methode. In diesem Fall gibt die Methode ein `tuple`-Objekt zurück, das nacheinander die Punktvorhersagen, die untere und schließlich die obere Intervallgrenze enthält. Um Platz zu sparen, werden diese Ergebnisse in [30] mit `.T` transponiert.

```
In [30]: [arr.T.round(3) for arr in vecm_res.predict(steps=5,
             alpha=0.05)]
```

```
Out[30]: [array([[-0.022, -0.004,  0.003,  0.024, -0.025],
                 [ 0.04 ,  0.041,  0.04 ,  0.039,  0.04 ]]),
          array([[-0.032, -0.014, -0.007,  0.014, -0.036],
                 [ 0.03 ,  0.025,  0.021,  0.016,  0.013]]),
          array([[-0.013,  0.006,  0.013,  0.034, -0.014],
                 [ 0.05 ,  0.056,  0.059,  0.062,  0.067]])]
```

Das erste der drei Arrays in dieser Outputzelle enthält die gerundeten Werte von Outputzelle [29]. Aus dem zweiten Array lesen wir die untere Grenze des Konfidenzintervalls ab. Sie beträgt demnach für die nächste Periode -0.032 für die Variable *Dp* und 0.03 für die Variable *R*. Die entsprechenden Werte für einen Prognosehorizont von 5 lauten -0.036 und 0.013. Die oberen

Konfidenzintervall-Grenzen befinden sich im dritten Array. Für die nächste Periode lauten sie -0.013 für die Inflationsrate und 0.05 für die Zinsrate.

Die Resultate der Vorhersage können auch in Form eines Plots veranschaulicht werden. Dazu nützen wir die Methode `plot_forecast`. Mit [31] wird der Plot in Abbildung 3.3 produziert. Es ist gut zu erkennen, wie sich der saisonale Verlauf von *Dp* auch in der Prognose fortsetzt.

```
In [31]: vecm_res.plot_forecast(steps=5, plot_conf_int=False)
```

Eingabezelle [32] führt zu Abbildung 3.4. Dabei haben wir mit dem Argument `n_last_obs` die Anzahl der geplotteten Beobachtungen der Zeitreihe auf die letzten zwölf beschränkt. Dadurch erscheint der Plot weniger gestaucht und lässt sich leichter ablesen. In Abbildung 3.2 auf S. 60 ist etwa schlecht zu erkennen, dass einer der ersten Berge in der Zeitreihe *Dp* um ein Quartal schmäler ist als die restlichen. Derartige Unregelmäßigkeiten in den letzten Beobachtungen würden im gestreckten Plot viel leichter auffallen.

```
In [32]: vecm_res.plot_forecast(steps=5, n_last_obs=12)
```

3.8. Granger-Kausalität

Auch bei der Frage nach einer etwaigen Kausalität im Sinne von Granger hilft unser `VECMResults`-Objekt. Dessen Methode `test_granger_causality` führt den in Unterabschnitt 2.8.2 beschriebenen Test durch und liefert ein Objekt zurück, das uns Zugriff auf die Einzelheiten des Tests bietet. Wir erzeugen es in [33] und weisen es der Variable `granger` zu.

```
In [33]: granger = vecm_res.test_granger_causality(caused="Dp",
                                                   signif=0.05)
```

Dieses Objekt besitzt eine Methode namens `summary` und der Aufruf von `granger.summary()` erzeugt die in Tabelle 3.9 gezeigte Zusammenfassung über den Test. Diese umfasst neben der Teststatistik und dem dazugehörigen p-Wert auch die Entscheidung über das Verwerfen oder Beibehalten der Nullhypothese. Der Test legt demnach ein Verwerfen von H_0 nahe. Es ist

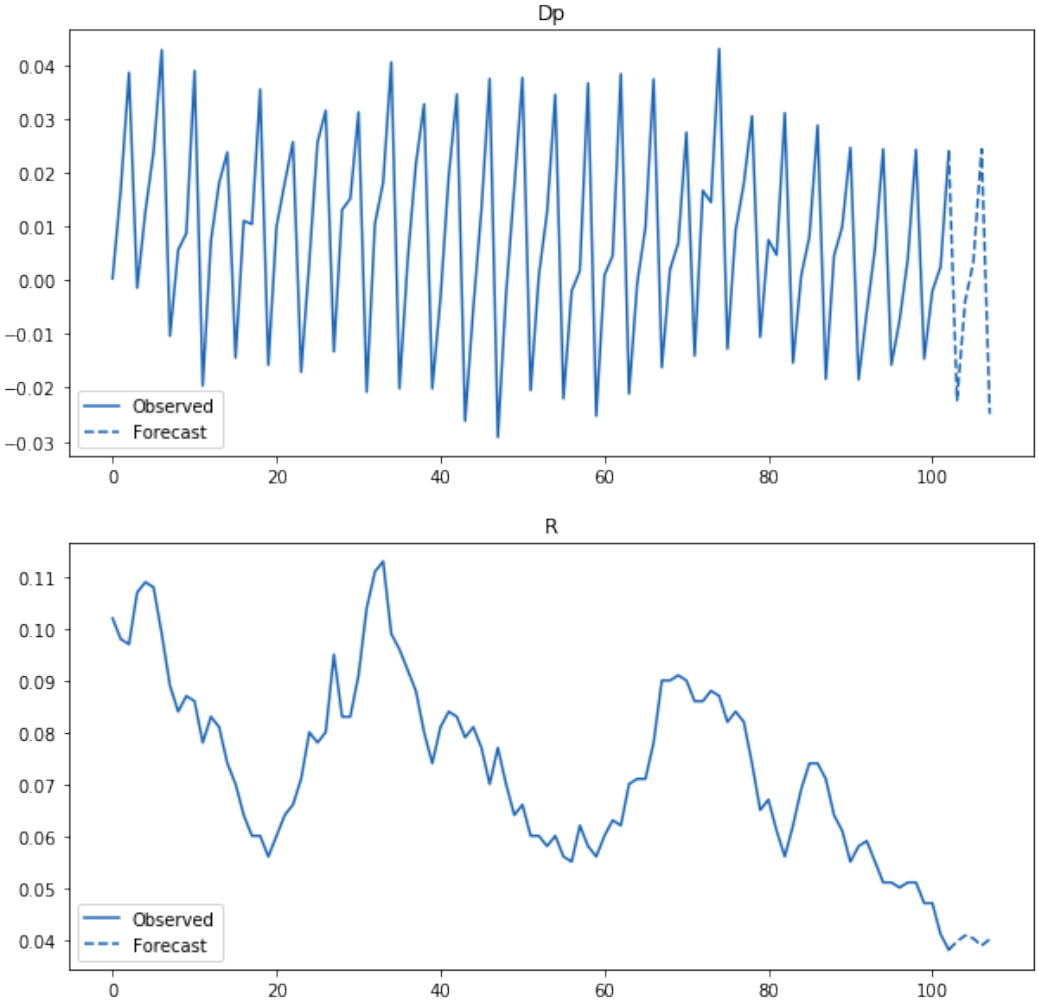

Abbildung 3.3.: Vorhersagen für Zeitreihen Inflation (Dp) und Zinsrate (R).

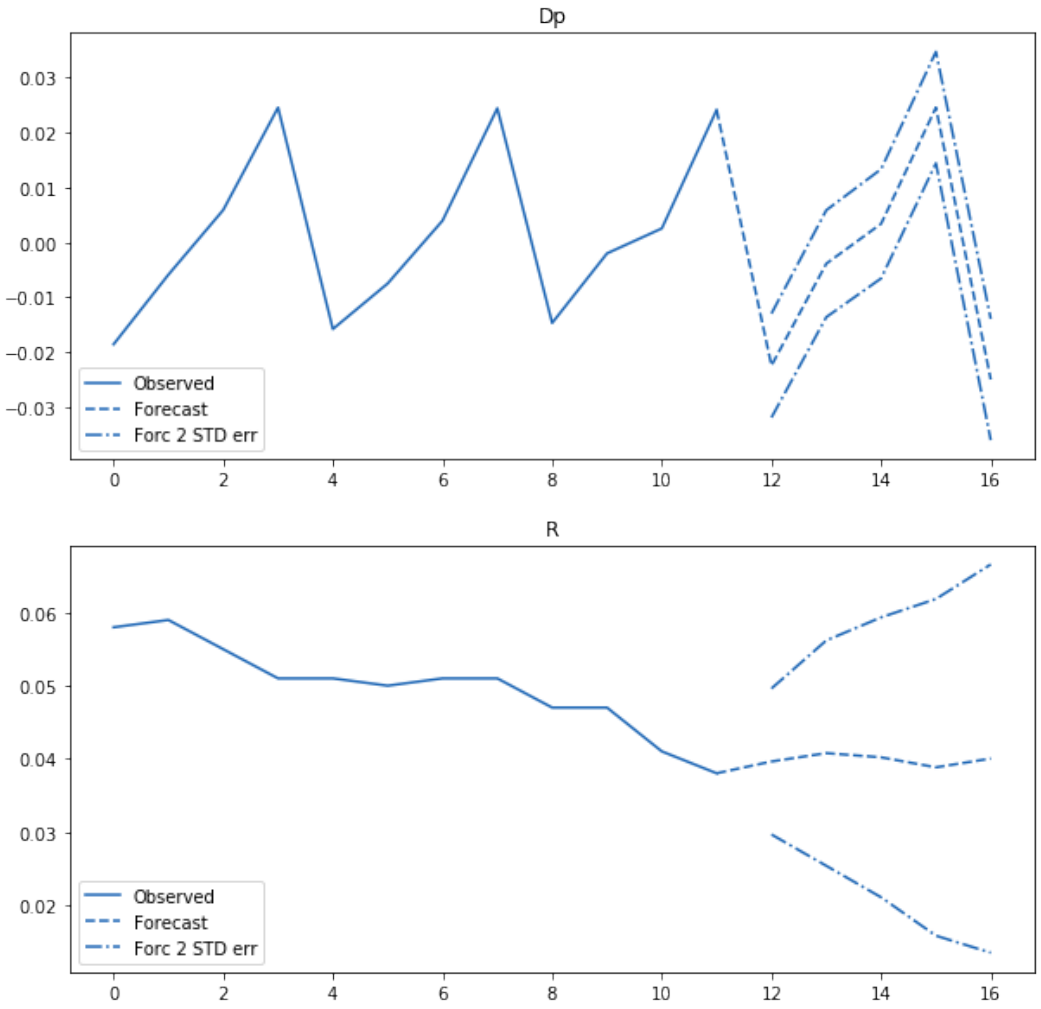

Abbildung 3.4.: Vorhersagen für Inflation (Dp) und Zinsrate (R) mit Konfidenzintervallen.

somit anzunehmen, dass die Zinsrate *R* Granger-kausal für die Inflationsrate *Dp* ist.

Granger causality F-test.
H_0: R does not Granger-cause Dp.
Conclusion: reject H_0 at 5% significance level.

Test statistic	Critical value	p-value	df
3.531	2.423	0.008	$(4, 176)$

Tabelle 3.9.: Test auf Granger-Kausalität von *R* für *Dp*.

Als nächstes testen wir, ob eine Kausalitätsbeziehung im Sinne von Granger auch in die umgekehrte Richtung anzunehmen ist. Dazu rufen wir wieder `test_granger_causality` auf, diesmal jedoch mit `"R"` statt `"Dp"` als `caused`-Argument. Der Befehl in [34] erzeugt die Tabelle 3.10.

```
In [34]: vecm_res.test_granger_causality(caused="R").summary()
```

Granger causality F-test.
H_0: Dp does not Granger-cause R.
Conclusion: reject H_0 at 5% significance level.

Test statistic	Critical value	p-value	df
0.7474	2.423	0.561	$(4, 176)$

Tabelle 3.10.: Test auf Granger-Kausalität von *Dp* für *R*.

Im umgekehrten Test kann die Nullhypothese also nicht verworfen werden. Demnach ist die Annahme, dass die Inflationsrate *Dp* nicht Granger-kausal für die Zinsrate *R* ist, zu akzeptieren.

Im Test [33] wurde das Signifikanzniveau manuell über das Argument `signif` festgelegt, während in [34] der Defaultwert verwendet wurde. Tabelle 3.10 zeigt, dass defaultmäßig ebenfalls `0.05` als Wert für `signif` eingesetzt wird.

Dem Referenzprogramm *JMulTi* folgend wird als Teststatistik λ_F aus Unterabschnitt 2.8.2 ausgegeben. Für die Berechnung des *p*-Wertes und des kriti-

schen Bereiches wird in Statistik-Programmpaketen oft auf die asymptotische Verteilung zurückgegriffen. In diesem Fall wurde die $F(N, K(T - Kp - 1))$-Verteilung gewählt – wiederum um die gleichen Ergebnisse wie *JMulTi* zu erhalten. Angesichts der Tatsache, dass $\lim_{T\to\infty} T = \lim_{T\to\infty} K(T - Kp - 1)$ gilt, erscheint diese Verteilung als Alternative zur $F(N, T)$-Verteilung denkbar. Argumente für diese und eine andere Alternative finden sich in Lütkepohl, (2005) auf S. 103.

3.9. Sofortige Kausalität

Für den Test auf sofortige Kausalität steht die Methode `test_inst_causality` zur Verfügung. Ihre Anwendung wird in [35] demonstriert.

```
In [35]: inst_dp_r = vecm_res.test_inst_causality(causing="Dp")
         inst_r_dp = vecm_res.test_inst_causality(causing="R")
```

Im Test in der ersten Zeile wird dabei *Dp* als potentiell kausale Variable angegeben. In jenem in der zweiten Zeile, nimmt *R* diese Rolle ein. Nach dieser Eingabe können die Details der beiden durchgeführten Tests über die Variablen `inst_dp_r` bzw. `inst_r_dp` abgefragt werden. Wie die Variable `granger` in [33] sind diese vom Typ `CausalityTestResults`, sodass wieder eine `summary`-Methode zur Verfügung steht. Die Aufrufe von `inst_r_dp.summary()` bzw. `inst_dp_r.summary()` führen zu den Tabellen 3.11 bzw. 3.12.

Instantaneous causality Wald-test.
H_0: R does not instantaneously cause Dp.
Conclusion: fail to reject H_0 at 5% significance level.

Test statistic	Critical value	p-value	df
0.6068	3.841	0.436	1

Tabelle 3.11.: Test auf sofortige Kausalität von *R* für *Dp*.

Durch den Vergleich der beiden Tabellen erkennen wir, dass die beiden Tests aus [35] zu den selben Ergebnissen führen. Dies liegt daran, dass die sofortige Kausalität – wie in Abschnitt 2.9 gezeigt – eine symmetrische Relation

Instantaneous causality Wald-test.
H_o: Dp does not instantaneously cause R.
Conclusion: fail to reject H_o at 5% significance level.

Test statistic	Critical value	p-value	df
0.6068	3.841	0.436	1

Tabelle 3.12.: Test auf sofortige Kausalität von *Dp* für *R*.

definiert. `CausalityTestResults`-Objekte unterstützen den Vergleich mit `==` und das Ergebnis dieser Operation, welches in `[36]` `True` ist, zeigt, dass die beiden verglichenen Tests über die selben Informationen verfügen.

```
In [36]: inst_dp_r == inst_r_dp
```

```
Out[36]: True
```

Die in unserem Beispiel gewonnenen Resultate lassen ein Verwerfen der Nullhypothese nicht zu, welche die Abwesenheit einer sofortigen Kausalität zwischen den beiden betrachteten Variablen postuliert.

3.10. Impuls-Response-Analyse

Die `irf`-Methode in der Klasse `VECMResults` dient der Impuls-Response-Analyse. Sie liefert ein Objekt vom Typ `statsmodels.tsa.vector_ar.irf.IRAnalysis`, welches wir in `[37]` der Variable `ir` zuweisen. Die dritte Zeile in dieser Eingabezelle zeigt, dass dieses Objekt Zugriff auf die MA-Darstellung des VECMs hat. Diese Darstellung ist – wie in Abschnitt 2.10 dargelegt – für die Impuls-Response-Analyse von fundamentaler Bedeutung.

```
In [37]: num_periods = 30
         ir = vecm_res.irf(periods=num_periods)
         np.all(vecm_res.ma_rep(maxn=num_periods) == ir.irfs)
```

```
Out[37]: True
```

Die Auswirkungen eines Impulses in einer Variable in Höhe von eins werden in Abbildung 3.5 veranschaulicht, welche vom Befehl in [38] erzeugt wird. Diesen Impuls erfährt die Inflationsrate in den beiden linken Subplots. Auf der rechten Seite ist die Zinsrate von einem derartigen Schock betroffen.

```
In [38]: ir.plot(plot_stderr=False)
```

Der Effekt einer plötzlichen Erhöhung der Inflationsrate auf die Variable selbst weist einen oszillierenden Verlauf mit abnehmender Amplitude um einen Wert von knapp 0,2 auf. Der selbe Impuls wirkt sich auf die Zinsrate in Form einer Erhöhung aus, welche auf einem Niveau von etwas unter 0,7 zum Stillstand zu kommen scheint.

Eine erhöhende Wirkung auf die jeweils andere Variable wird auch vom rechten oberen Subplot angezeigt, wo ein Impuls in R im Modell zu wachsenden Inflationsraten führt. Der rechte untere Teilplot zeigt, dass ein plötzlicher Anstieg der Zinsrate über die nächsten ca. vier Perioden noch stärker wird, ehe er sich auf ein Niveau begibt, das knapp über dem verabreichten Impuls liegt.

Für volkswirtschaftliche Entscheidungsträger können sich derartige Analysen als nutzbringend erweisen, erlauben sie doch den Einfluss der Änderung einer Variable auf andere Variablen abzuschätzen. So bietet die Impuls-Response-Analyse etwa Zentralbanken die Möglichkeit, Auswirkungen geldmengenbezogener Entscheidungen auf den Zinssatz vorherzusehen. Ebenso lassen sich die Folgen anderer Markteingriffe – z.B. in Form von Steuerreformen – auf interessierende makroökonomische Variablen abschätzen. Akteure mit wirtschaftlicher Entscheidungsmacht, die diese Art der Analyse nutzen möchten, haben allerdings die in Abschnitt 2.10 erwähnte Existenz potenzieller Fehlerquellen zu bedenken.

Die in Abbildung 3.5 gezeigten Plots umfassen lediglich einen Zeitraum von 30 Perioden. Damit ermöglichen sie keine Schlüsse auf Auswirkungen jenseits dieses Zeithorizonts. Insbesondere sind damit keine Aussagen über die Konvergenz der Responses möglich. Die weitgehend horizontalen Verläufe aller gezeigten Kurven ab ca. 20 Perioden nach dem Impuls sind jedoch ein Indiz dafür, dass sich die Responses auf diesem Niveau einpendeln. Dies macht den in Abschnitt 2.10 angesprochenen Unterschied zwischen

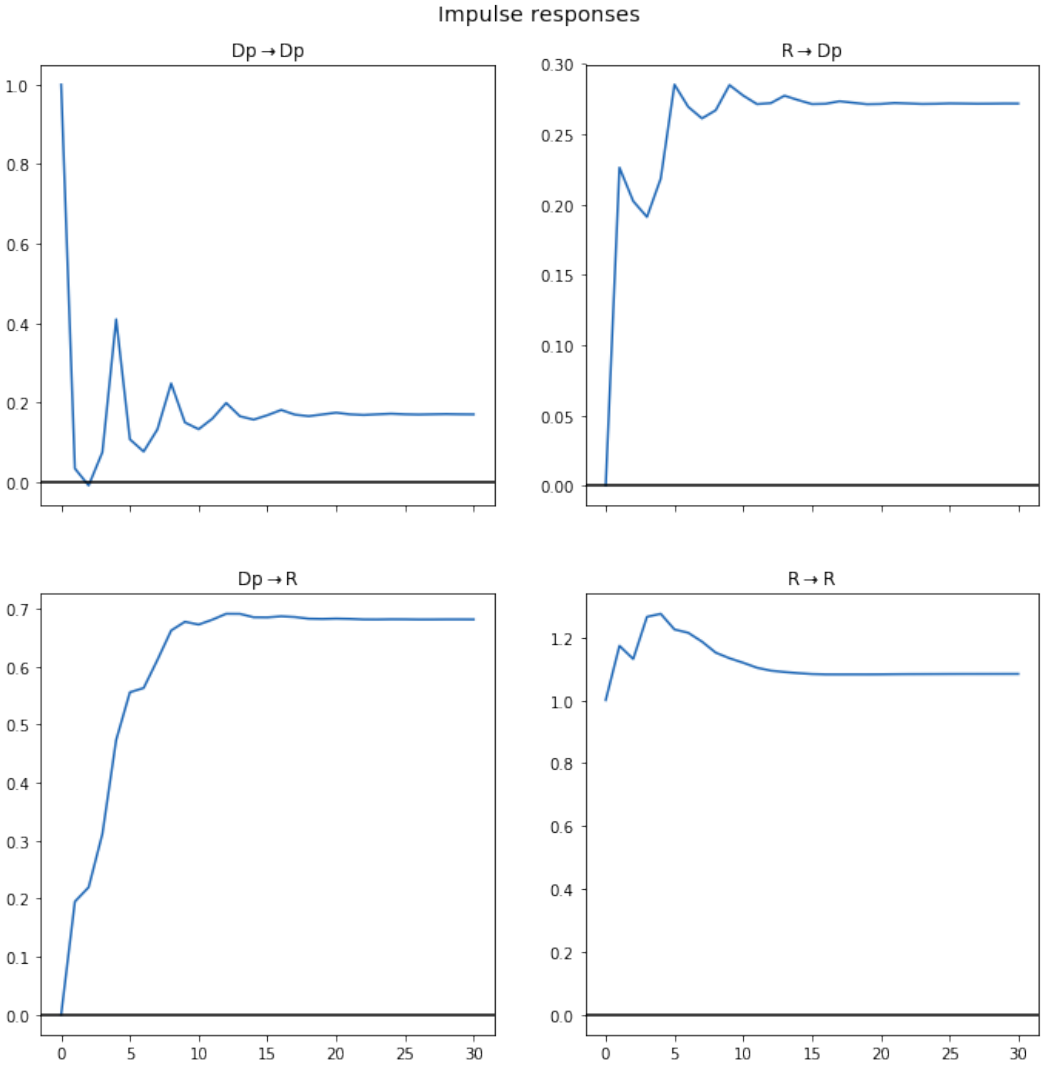

Abbildung 3.5.: Impulse-Response-Analyse.

VAR-Modellen und VECMs deutlich. Während die Responses im Fall des ersteren gegen 0 konvregieren, trifft diese Eigenschaft auf VECMs nicht zu.

3.11. Tests

Um die Korrektheit der vom hier vorgestellten VECM-Modul generierten Resultate sicherzustellen, werden diese mit den Ergebnissen anderer Programme verglichen. Diese Aufgabe erledigt der Computer in Form von Testprozeduren. Wie am Ende von Abschnitt 3.1 beschrieben, wurden auch für das VECM-Modul Tests geschrieben, wobei die Referenzergebnisse von der Software *JMulTi* erzeugt wurden. Die Tests des gesamten `vector_ar`-Moduls können mit den in [39] auskommentierten Befehlen gestartet werden.

```
In [39]: # from statsmodels import api as sm
         # sm.tsa.var.test()
```

Um nur die VECM-bezogenen Testfälle isoliert auszuführen, können wir auf die Kommandozeilenumgebung ausweichen. Dies wird in der Folge demonstriert, wobei das Dollar-Zeichen den Eingabeprompt symbolisiert und wir voraussetzen, dass die Kommandozeilenumgebung in jenem Ordner gestartet wurde, in dem sich die beiden Dateien `test_vecm.py` und `test_coint.py` befinden.

```
$ python /usr/local/bin/nosetests test_vecm.py test_coint.py
... TESTOUTPUT ...
----------------------------------------------------------------------
Ran 1099 tests in 3.145s

OK
```

Es sei darauf hingewiesen, dass die obige Ausgabe verkürzt wurde. Dies wird durch die Zeile mit `TESTOUTPUT` angedeutet. Am Ende der Ausgabe finden wir eine Zusammenfassugn über die durchgeführten Tests. Demnach wurden über 1000 Testfälle betrachtet. Jeder Test attestiert der jeweiligen überprüften Funktionalität des VECM-Moduls Fehlerfreiheit.

Neben dem vorgeführten manuellen Testen werden nach jedem Commit auf GitHub automatisch alle Testfälle ausgeführt und etwaige Fehler gemeldet. Für diese Funktionalität werden die Services von Travis CI und AppVeyor genutzt. Mit Travis CI werden alle Testfälle in einer Linuxumgebung je ein Mal für verschiedene Versionen von Python und der benutzten Bibliotheken, wie etwa Numpy, ausgeführt. AppVeyor erlaubt entsprechende Tests für das Betriebssystem Windows.

Neben der eigentlichen Durchführung von Tests verwendet *statsmodels* mit Coveralls auch ein Service, das die Testabdeckung analysiert. Dabei wird untersucht, wie viele Codezeilen im Rahmen der Tests zur Ausführung kommen. Diese Zahl wird zur Gesamtzahl an Codezeilen ins Verhältnis gesetzt. Dem neuen VECM-Modul wird auf `https://coveralls.io/builds/12166209` eine hohe Testabdeckung von 96.2 % für `vecm.py` bescheinigt, während das Gesamtprojekt eine Testabdeckung von 90.8 % aufweist.

Neben JMulTi wurden im Rahmen dieser Arbeit auch weitere Programme mit VECM-Funktionalität betrachtet. Zu diesen gehören *Stata* (vgl. `http://www.stata.com/manuals14/tsvecintro.pdf`) sowie das R-Package *tsDyn* (vgl. `https://CRAN.R-project.org/package=tsDyn`).

In *Stata* kann ein VECM(3) mit den folgenden Befehlen an die Daten angepasst werden, wobei die hier verwendete CSV-Datei lediglich zwei Spalten – Dp und R – umfasst.

```
insheet using "Z:\vecm\E6.csv", comma
generate dates = tq(1972q2) + _n-1
tsset dates
format dates %tq
vec dp r, lags(4) trend(trend)
save "Z:\vecm\stata\vecm.dta"
```

Dabei wurden durch die Angabe von `trend(trend)` festgelegt, dass sowohl ein Intercept als auch ein linearer Trend im Modell vorhanden ist. Das entspricht der Angabe von `deterministic="colo"` im VECM-Modul von *statsmodels*. Andere Kombinationen deterministischer Terme können durch `trend(rconstant)` (auf die Kointegrationsbeziehung beschränkter Intercept), `trend(constant)` (Intercept außerhalb der Kointegrationsbeziehung) und `trend(rtrend)` (Intercept außerhalb der Kointegrationsbeziehung und

auf die Kointegrationsbeziehung beschränkter linearer Term) anstelle von `trend(trend)` angegeben werden. Die Verwendung von `trend(none)` führt zu einem Modell ohne deterministische Terme. Die durch die letzte Zeile der obigen Befehle erzeugte Datei hat die folgende Form.

```
Vector error-correction model

Sample:  1973q2 - 1998q4                        No. of obs      =        103
                                                AIC             =   -14.7245
Log likelihood =  777.3118                      HQIC            =  -14.52765
Det(Sigma_ml)  =  9.55e-10                      SBIC            =  -14.23848

Equation           Parms      RMSE     R-sq       chi2     P>chi2
------------------------------------------------------------------
D_dp                   9     .006263   0.9606   2289.044   0.0000
D_r                    9     .005417   0.1411   15.43745   0.0796
------------------------------------------------------------------


------------------------------------------------------------------------------
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
D_dp         |
        _ce1 |
         L1. |  -.6354181   .2109086    -3.01   0.003    -1.048791   -.2220448
             |
          dp |
         LD. |  -.5189543   .1608721    -3.23   0.001    -.8342579   -.2036506
        L2D. |  -.6569904   .1099723    -5.97   0.000    -.8725322   -.4414487
        L3D. |  -.8040351   .0581788   -13.82   0.000    -.9180635   -.6900068
             |
           r |
         LD. |   .0440729   .1226858     0.36   0.719    -.1963869    .2845326
        L2D. |    .117636   .1221657     0.96   0.336    -.1218043    .3570763
        L3D. |   -.054507   .1193761    -0.46   0.648    -.2884799    .1794659
             |
      _trend |  -2.59e-06   .0000211    -0.12   0.902    -.0000439    .0000387
       _cons |  -.0001143   .0012538    -0.09   0.927    -.0025717     .002343
-------------+----------------------------------------------------------------
D_r          |
        _ce1 |
         L1. |   .4186457   .1824196     2.29   0.022     .0611099    .7761814
             |
          dp |
         LD. |  -.3183074   .1391419    -2.29   0.022    -.5910205   -.0455943
        L2D. |  -.1980921   .0951175    -2.08   0.037    -.3845189   -.0116653
        L3D. |  -.0690681   .0503201    -1.37   0.170    -.1676937    .0295575
             |
           r |
         LD. |   .2516031   .1061137     2.37   0.018     .0436241    .4595821
        L2D. |   .0116358   .1056638     0.11   0.912    -.1954615     .218733
        L3D. |   .2190574   .1032511     2.12   0.034      .016689    .4214258
             |
      _trend |  -3.94e-06   .0000182    -0.22   0.829    -.0000396    .0000318
```

```
       _cons |   -.0001757    .0010844     -0.16    0.871     -.0023011     .0019497
------------------------------------------------------------------------------

Cointegrating equations

Equation            Parms     chi2      P>chi2
-------------------------------------------------
_ce1                    1   17.67106    0.0000
-------------------------------------------------

Identification:  beta is exactly identified

                  Johansen normalization restriction imposed
------------------------------------------------------------------------------
        beta |      Coef.   Std. Err.       z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
_ce1         |
          dp |           1          .        .       .            .           .
           r |  -.2763998   .0657516    -4.20   0.000    -.4052705    -.147529
      _trend |  -1.57e-06           .        .       .            .           .
       _cons |   .0125696           .        .       .            .           .
------------------------------------------------------------------------------
```

Um diese Ergebnisse mit jenen des Python-Moduls zu vergleichen, bilden wir in [40] ein entsprechendes Modell.

```
In [40]: model2 = VECM(data, deterministic="colo",
                       k_ar_diff=3, coint_rank=1)
         vecm_res2 = model2.fit()
```

Rufen wir nun `vecm_res2.summary()` auf, so erhalten wir eine Zusammenfassung der Resultate, die aus Platzgründen auf die Tabellen 3.13 und 3.14 aufgeteilt wurde.

Wie das Python-Modul zeigt auch *Stata* die geschätzten Einträge der Matrizen Γ_i zusammen mit den deterministischen Termen an. *Stata* gibt in diesen Tabellen auch die geschätzten Einträge von α aus. Die entsprechenden Zeilen sind mit `_ce1` betitelt. Auf den ersten Blick scheint *Stata* andere Werte für die deterministischen Terme auszugeben und es fällt auf, dass in der Kointegrationsbeziehung ebenfalls deterministische Terme aufscheinen, was bei *statsmodels* nicht der Fall ist. Der Grund dafür ist, dass die deterministischen Terme bei *Stata* aufgespaltet werden in einen Anteil im von den Spalten von α aufgespannten Raum und einen Anteil, der zu α orthogonal ist. Wir zeigen in [41] exemplarisch für den konstanten deterministischen Term, dass die Ergebnisse der beiden Softwarepakete in keinem Widerspruch stehen.

Det. terms outside the coint. relation &
lagged endog. parameters for equation Dp

	coef	std err	z	P>|z|	[0.025	0.975]
const	−0.0081	0.003	−3.011	0.003	−0.013	−0.003
lin_trend	−1.597e−6	2.01e−5	−0.080	0.937	−4.09e−5	3.77e−5
L1.Dp	−0.5190	0.154	−3.377	0.001	−0.820	−0.218
L1.R	0.0441	0.117	0.376	0.707	−0.186	0.274
L2.Dp	−0.6570	0.105	−6.254	0.000	−0.863	−0.451
L2.R	0.1176	0.117	1.008	0.313	−0.111	0.346
L3.Dp	−0.8040	0.056	−14.467	0.000	−0.913	−0.695

Det. terms outside the coint. relation &
lagged endog. parameters for equation R

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0051	0.002	2.195	0.028	0.001	0.010
lin_trend	−4.592e−6	1.74e−5	−0.264	0.791	−3.86e−5	2.94e−5
L1.Dp	−0.3183	0.133	−2.395	0.017	−0.579	−0.058
L1.R	0.2516	0.101	2.482	0.013	0.053	0.450
L2.Dp	−0.1981	0.091	−2.180	0.029	−0.376	−0.020
L2.R	0.0116	0.101	0.115	0.908	−0.186	0.209
L3.Dp	−0.0691	0.048	−1.437	0.151	−0.163	0.025
L3.R	0.2191	0.099	2.221	0.026	0.026	0.412

Tabelle 3.13.: Ergebnisse der Parameterschätzung

Loading coefficients (α) for equation Dp

	coef	std err	z	P>|z|	[0.025	0.975]
ec1	−0.6354	0.201	−3.154	0.002	−1.030	−0.241

Loading coefficients (α) for equation R

	coef	std err	z	P>|z|	[0.025	0.975]
ec1	0.4186	0.174	2.402	0.016	0.077	0.760

Cointegration relations for loading-coefficients-column 1

	coef	std err	z	P>|z|	[0.025	0.975]
beta.1	1.0000	0	0	0.000	1.000	1.000
beta.2	−0.2764	0.063	−4.400	0.000	−0.400	−0.153

Tabelle 3.14.: Ergebnisse der Parameterschätzung (Fortsetzung von Tabelle 3.13)

In [41]:
```
stata_alpha = np.array([[-.6354181],
                        [.4186457]])
stata_const = np.array([[-.0001143],
                        [-.0001757]])
stata_const_coint = .0125696

const = stata_const + stata_alpha * stata_const_coint

print("const Stata:\n", const.round(4))
print("const statsmodels:\n", vecm_res2.const.round(4))
```

```
const Stata:
 [[-0.0081]
 [ 0.0051]]
const statsmodels:
 [[-0.0081]
 [ 0.0051]]
```

Demnach stimmen die von den beiden Programmen berechneten Schätzer auf den ersten vier Nachkommastellen überein. Auf einen kleinen Unterschied gehen wir unten im Zusammenhang mit dem R-Package *tsDyn* ein. Es

fallen jedoch neben den Schätzern auch Unterschiede bei den Standardfehlern und damit in weiterer Folge auch bei den t- und p-Werten Unterschiede auf. Keine Unterschiede weisen die Ergebnisse des Python-Moduls zu jenen von JMulTi auf. Der Output von JMulTi befindet sich aufgrund seiner Länge in Anhang B.

Das dritte Programm, dessen Ergebnisse zum Vergleich betrachtet wurden, ist das R-Package *tsDyn*. Es wird mit

```
> install.packages("tsDyn")
```

installiert. Das Laden des Packages, das Einlesen der Daten und das Anpassen des Modells wird mit den folgenden Befehlen erreicht.

```
> library(tsDyn)
> dta = read.table("E6.csv", header = FALSE, sep = " ")
> vecm.jo <- VECM(dta, lag=3, estim="ML", include="both",
   LRinclude="none")
```

Es sei angemerkt, dass die hier verwendete csv-Datei wie im *Stata*-Beispiel nur aus den beiden Spalten Dp und R besteht. Mit `include="both"` geben wir an, dass wir sowohl eine Konstante als auch einen linearen deterministischen Term jeweils außerhalb der Kointegrationsbeziehung in das Modell aufnehmen möchten. Neben `"both"` sind auch die Optionen `"const"` (nur Intercept), `"trend"` (linearer Trend ohne Intercept) und `"none"` (keine deterministischen Terme) möglich. Die letzte dieser vier Alternativen wurde für `LRinclude` include gewählt, womit deterministische Terme angegeben werden, die auf die Kointegrationsbeziehung beschränkt sind. Mit den folgenden Befehlen geben wir die Schätzer für α, die deterministischen Terme sowie Γ inkl. Standardfehler, t- und p-Werten aus.

```
> sum.jo <- summary(vecm.jo)
> sum.jo$coefMat
                 Estimate   Std. Error      t value     Pr(>|t|)
V1:ECT       -6.354180e-01 2.109086e-01  -3.01276416 3.326345e-03
V1:Intercept -8.102291e-03 2.789883e-03  -2.90416919 4.589495e-03
V1:Trend     -1.597127e-06 2.101373e-05  -0.07600398 9.395775e-01
V1:V1-1      -5.189544e-01 1.608722e-01  -3.22588053 1.728442e-03
```

```
V1:V2-1        4.407308e-02 1.226858e-01    0.35923537 7.202247e-01
V1:V1-2       -6.569905e-01 1.099723e-01   -5.97414568 4.107073e-08
V1:V2-2        1.176359e-01 1.221657e-01    0.96292132 3.380576e-01
V1:V1-3       -8.040352e-01 5.817878e-02 -13.82007557 2.283626e-24
V1:V2-3       -5.450704e-02 1.193761e-01   -0.45659915 6.490125e-01
V2:ECT         4.186458e-01 1.824196e-01    2.29496069 2.396006e-02
V2:Intercept   5.087145e-03 2.413032e-03    2.10819638 3.767421e-02
V2:Trend      -4.591507e-06 1.817524e-05   -0.25262426 8.011102e-01
V2:V1-1       -3.183075e-01 1.391419e-01   -2.28764661 2.440042e-02
V2:V2-1        2.516031e-01 1.061137e-01    2.37107079 1.977729e-02
V2:V1-2       -1.980922e-01 9.511749e-02   -2.08260499 4.000323e-02
V2:V2-2        1.163572e-02 1.056638e-01    0.11012022 9.125488e-01
V2:V1-3       -6.906817e-02 5.032013e-02   -1.37257548 1.731504e-01
V2:V2-3        2.190575e-01 1.032511e-01    2.12159980 3.650185e-02
```

Den Schätzer für β erhalten wir wie folgt.

```
> sum.jo$model.specific$coint
           r1
V1  1.0000000
V2 -0.2763998
```

Die Ergebnisse in *R* gleichen im Wesentlichen jenen, die von *Stata* berechnet wurden. Bei genauerer Untersuchung der Schätzer fällt auf, dass der Schätzer für den Intercept in *statsmodels* von jenen in *tsDyn* und *Stata* abweicht, auch wenn dieser Unterschied bei entsprechender Rundung – wie in [41] – nicht sichtbar ist. Da es sich bei *tsDyn* um ein Open-Source-Projekt handelt, war es möglich, die Ursache für diese Abweichung zu finden. Das folgende Listing zeigt einen Ausschnitt des Sourcecodes von *tsDyn*, wobei der Whitespace zugunsten einer besseren Lesbarkeit verändert wurde.

```
  incReg <- switch(include,
                   const = matrix(1, nrow = t, ncol = 1),
                   trend = matrix(seq_len(t), ncol = 1),
                   both = cbind(rep(1, t), seq_len(t)),
                   none = NULL)
```

Darin wird dem Regressor im Fall eines linearen Trends (in der `switch`-Funktion für die Fälle `trend` und `both`) durch die Funktion `seq_len` jeweils eine Spalte mit den Werten $1, \ldots, T$ hinzugefügt, wobei T die Anzahl der Beobachtungen ohne Presample bezeichnet. Im hier vorgestellten VECM-Modul sind diese Werte um die Modellordnung verschoben. Die Bildung

dieses Vektors erfolgt beim Python-Modul in der Funktion `_linear_trend`. Um mit *statsmodels* das gleiche Ergebnis wie mit *tsDyn* zu erhalten, können wir die Funktion `_linear_trend` entsprechend abwandeln. Dies geschieht in [42]. [43] zeigt, dass damit tatsächlich die gleichen Ergebnisse produziert werden.

```
In [42]: import statsmodels

         def new_trend(nobs, k_ar, coint=False):
             return np.arange(nobs) + (1 if not coint else 0)

         statsmodels.tsa.vector_ar.vecm._linear_trend = new_trend
```

```
In [43]: model3 = VECM(data, deterministic="colo",
                       k_ar_diff=3, coint_rank=1)
         vecm_res3 = model3.fit(method="ml")
```

```
In [44]: vecm_res3.const
```

```
Out[44]: array([[-0.00810229],
                [ 0.00508714]])
```

In [45] zeigen wir, dass sich die Ergebnisse mit der unveränderten Berechnungsmethode von *tsDyn* unterscheiden.

```
In [45]: vecm_res2.const
```

```
Out[45]: array([[-0.0080959 ],
                [ 0.00510551]])
```

Wie [46] zeigt, unterscheiden sich die zwei verschiedenen Ergebnisse für den Intercept lediglich um ein Vielfaches des linearen Trends.

```
In [46]: p = vecm_res2.k_ar
         vecm_res2.const + p * vecm_res2.lin_trend
```

```
Out[46]: array([[-0.00810229],
                [ 0.00508714]])
```

Der Unterschied, der auch der Grund war, den linearen Trend im Vergleich der Programme zu verwenden, resultiert also aus verschiedenen internen

Nummerierungen der Beobachtungszeitpunkte. Er hat jedoch keinen Einfluss auf die Analyse von Zeitreihen. So stimmen etwa die Vorhersagen vom Python-Modul mit jenen von *tsDyn* überein, wie [47] und das darauf folgende Listing zeigen.

```
In [47]: vecm_res2.predict()

Out[47]: array([[-0.01794271,  0.03880988],
                [-0.00398786,  0.03903207],
                [ 0.00041692,  0.03922466],
                [ 0.02096604,  0.03840987],
                [-0.01868855,  0.03880252]])
```

```
> predict(vecm.jo)
              V1          V2
108 -0.017942709 0.03880988
109 -0.003987861 0.03903207
110  0.000416921 0.03922466
111  0.020966043 0.03840987
112 -0.018688547 0.03880252
```

Auch geben die verschiedenen Programme die selbe Loglikelihood aus. Wir geben dieses Ergebnis nun in Python und R wieder. Die Loglikelihood in *Stata* war bereits in der Zusammenfassung auf Seite 88 enthalten.

```
In [48]: vecm_res2.llf

Out[48]: 777.31178176742276
```

```
> logLik(vecm.jo)
[1] 777.3118
```

Auf Seite 112 sehen wir, dass *JMulTi* eine Loglikelihood von 769,77 ausweist. Dies liegt daran, dass dieses Programm die Loglikelihoods für VECMs mit einem zusätzlichen Faktor von $T/(T-1)$ berechnet, wobei T die Anzahl der Beobachtungen ohne Presample bezeichnet. Dieser Faktor stört bei der Wahl des ML-Schätzers nicht und strebt für $T \to \infty$ gegen 1.

Auch bei der Untersuchung der verschiedenen Standardfehler stellen wir fest, dass sich diese lediglich um einen konstanten Faktor unterscheiden. Durch die Division der Ergebnisse in Python oder *JMulTil* mit 0.955 gelangen wir zu jenen in *Stata* oder *R*, wie wir in [49] sehen. Die darin der Variablen `r_se` zugewiesenen Werte stammen aus dem Auszug der *R*-Ergebnisse auf Seite 92. Die Ursache für den Faktor von 0.955 wurde im Rahmen dieser Arbeit nicht gefunden. Wie im Fall von *JMulTi*s Loglikelihood könnte es sich auch hier um einen asymptotisch zu vernachlässigenden Faktor handeln.

```
In [49]: py_se = vecm_res2.stderr_gamma[0]
         r_se = np.array([1.608722e-01, 1.226858e-01,
                          1.099723e-01, 1.221657e-01,
                          5.817878e-02, 1.193761e-01])
         py_se / r_se
```

```
Out[49]: array([0.95531191, 0.95531191, 0.95531191,
                0.95531191, 0.95531191, 0.95531191])
```

Stata sowie das R-Package *tsDyn* bieten Funktionen für weite Teile der in dieser Arbeit vorgestellten Analysemöglichkeiten im Zusammenhang mit VECMs. Keines der beiden Programme scheint jedoch die in den Abschnitten 2.8 und 2.9 vorgestellten Kausalitätsanalysen zu unterstützen.

JMulTi deckt dagegen alle in der vorliegenden Arbeit vorgestellten Konzepte ab. Neben der angesprochenen leichten Abweichung bei der Berechnung der Likelihood, fiel bei der Entwicklung des VECM-Moduls von *statsmodels* noch eine weitere kleine Abweichung von der in Kapitel 2 vorgestellten Theorie auf. *JMulTi* scheint VECM($p-1$)-Modelle bei der Berechnung der sofortigen Kausalität – wie bei der Granger-Kausalität – in ein VAR($p+1$)-Modell zu übersetzen. Das hier vorgestellte Python-Modul nutzt für die Granger-Kausalität ein VAR($p+1$)-Modell und für die sofortige Kausalität ein VAR(p)-Modell, wie in den entsprechenden Abschnitten in Kapitel 2 beschrieben.

Wir halten abschließend fest, dass *statsmodels* in der Lage ist, die Resultate anderer Softwarepakete gut nachzuahmen. Der Wahl eines Modells mit Intercept für den Vergleich der verschiedenen Programme lag die Überlegung zugrunde, möglichst alle Unterschiede zwischen dem VECM-Modul von *statsmodels* und anderen Softwarepaketen aufzuzeigen.

4. Zusammenfassung

Im theoretischen Teil der vorliegenden Arbeit wurden die Grundlagen des vektorwertigen Fehlerkorrekturmodells und dem damit verwandten Vektorautoregressiven Modell besprochen. Dabei haben wir zunächst die Frage der Modellspezifikation betrachtet. Ausgehend von einem festgelegten Modell wurden die Maximum-Likelihood-Schätzer der Parameter hergeleitet. Im Anschluss daran wurden zwei diagnostische Verfahren präsentiert. Es folgte eine Diskussion zur Vorhersage zukünftiger Werte der Zeitreihe. Den Abschluss des theoretischen Teils bildete ein Überblick über drei verschiedene Möglichkeiten der strukturellen Analyse, bei der Zusammenhänge zwischen verschiedenen Komponenten der Zeitreihe untersucht werden.

Im praktischen Teil dieser Arbeit wurde das neue VECM-Modul von *statsmodels* präsentiert. Wie wir gesehen haben, stellt dieses für alle im Theorieteil besprochenen Konzepte entsprechende Funktionen bereit. Es wurde gezeigt, dass das VECM-Modul in der Lage ist, die Ergebnisse bereits bestehender Software auf diesem Gebiet zu replizieren. Angesichts der hohen Testabdeckung könnte das Python-Modul damit auf dem Gebiet der VECMs eine ernsthafte Alternative zu anderen Softwarepakten darstellen.

Anhang A.

Rechenregeln

An dieser Stelle werden Rechenregeln angeführt, die in den Herleitungen in dieser Arbeit Verwendung finden. Bei allen Regeln wird vorausgesetzt, dass die darin vorkommenden Matrizen A, B C, D und I derart dimensioniert sind, dass sie in den jeweiligen Operationen kompatibel sind. Auch wird im Falle invertierter Matrizen stets vorausgesetzt, dass die entsprechenden Matrizen regulär sind. Die angeführten Regeln sind auch in Anhang A in Lütkepohl, (2005) zu finden.

A.1. Spur

$$tr(AB) = tr(BA) \tag{A.1}$$

A.2. Cholesky-Zerlegung

Für jede positiv definite Matrix $A \in \mathbb{R}^{m \times m}$ existiert eine untere Dreiecksmatrix P mit positiver Hauptdiagonale, sodass

$$P^{-1}AP'^{-1} = I_m \quad \text{bzw.} \quad A = PP' \tag{A.2}$$

A.3. Moore-Penrose-Inverse einer Matrix

Jede Matrix $A \in \mathbb{R}^{m \times n}$ besitzt eine eindeutig bestimmte Moore-Penrose-Inverse A^+. Sie wird durch die Eigenschaften

$$\begin{aligned} AA^+A &= A \\ A^+AA^+ &= A^+ \\ (AA^+)' &= AA^+ \\ (A^+A)' &= A^+A \end{aligned}$$

definiert.

A.4. Kronecker-Produkt

Das Kronecker-Produkt

$$A \otimes B := \begin{pmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{pmatrix} \text{ mit } A = \left(a_{ij}\right)_{\substack{1 \le i \le m \\ 1 \le j \le n}}$$

findet häufig zusammen mit vektorisierten Matrizen (siehe A.5) Verwendung. Einige wichtige Rechenregeln werden in der Folge aufgelistet.

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}. \tag{A.3}$$

$$(A \otimes B)' = A' \otimes B'. \tag{A.4}$$

$$(A \otimes B)(C \otimes D) = AC \otimes BD. \tag{A.5}$$

$$\forall A \in \mathbb{R}^{m \times m}, B \in \mathbb{R}^{n \times n} : |A \otimes B| = |A|^n |B|^m. \tag{A.6}$$

A.5. Spaltenweise Vektorisierung von Matrizen

Definition A.1. Die spaltenweise Vektorisierung einer Matrix $A \in \mathbb{R}^{m \times n}$ ist definiert durch

$$\operatorname{vec}(A) := (a_{11} \dots a_{m1} \quad a_{12} \dots a_{m2} \quad \dots \quad a_{1n} \dots a_{mn})'.$$

Diese Operation erweist sich etwa dann als nützlich, wenn eine gemeinsame Dichte für alle Elemente einer Matrix angegeben werden soll. Dies ist z.B. im Falle der Likelihood-Maximierung notwendig. Einige wichtige Rechenregeln im Zusammenhang mit dem vec-Operator sind die Folgenden.

$$\operatorname{vec}(ABC) = (C' \otimes A)\operatorname{vec}(B). \tag{A.7}$$

$$\begin{aligned} \operatorname{vec}(AB) &\overset{(A.7)}{=} (I \otimes A)\operatorname{vec}(B), \\ \operatorname{vec}(AB) &\overset{(A.7)}{=} (B' \otimes I)\operatorname{vec}(A). \end{aligned} \tag{A.8}$$

$$\begin{aligned} \operatorname{tr}(ABC) &= \operatorname{vec}(A')'(C' \otimes I)\operatorname{vec}(B) \overset{(A.8)}{=} \operatorname{vec}(A')'(I \otimes B)\operatorname{vec}(C), \\ \operatorname{tr}(ABC) &= \operatorname{vec}(B')'(A' \otimes I)\operatorname{vec}(C) \overset{(A.8)}{=} \operatorname{vec}(B')'(I \otimes C)\operatorname{vec}(A), \\ \operatorname{tr}(ABC) &= \operatorname{vec}(C')'(B' \otimes I)\operatorname{vec}(A) \overset{(A.8)}{=} \operatorname{vec}(C')'(I \otimes A)\operatorname{vec}(B). \end{aligned} \tag{A.9}$$

Eine ähnlicher Operator ist vech. Er unterscheidet sich von vec dadurch, dass nicht alle Elemente im resultierenden Vektor aufscheinen, sondern lediglich jene, die sich auf oder unter der Hauptdiagonale befinden. Formal wird dies in der folgenden Definition ausgedrückt.

Definition A.2. Für eine Matrix $A \in \mathbb{R}^{m \times m}$ definieren wir

$$\operatorname{vech}(A) := \begin{pmatrix} a_{1,1} \dots a_{m,1} & a_{2,2} \dots a_{m,2} & \dots & a_{m-1,m-1} & a_{m,m-1} & a_{m,m} \end{pmatrix}'.$$

Da die Elemente über der Hauptdiagonale unberücksichtigt bleiben, findet diese Operation besonders im Zusammenhang mit symmetrischen Matrizen Verwendung. Eng verwandt mit vech ist die Duplikationsmatrix.

Definition A.3. Die Duplikationsmatrix $D_m \in \mathbb{R}^{m^2 \times \frac{1}{2}m(m+1)}$ ist jene Matrix, die für jede symmetrische $m \times m$-Matrix A die Gleichung

$$\text{vec}\,(A) = D_m \text{vech}\,(A)$$

erfüllt.

A.6. Differentiation im Mehrdimensionalen

Definition A.4. Die Ableitung einer Funktion $f : \mathbb{R}^{m \times n} \to \mathbb{R}$ ist definiert als

$$\frac{\partial f}{\partial A} := \left(\frac{\partial f}{\partial a_{ij}}\right)_{\substack{1 \le i \le m \\ 1 \le j \le n}}.$$

Es gelten die folgenden Regeln.

$$\frac{\partial \text{tr}(BA^{-1}C)}{\partial A} = -(A^{-1}CBA^{-1})'. \tag{A.10}$$

$$\frac{\partial \ln |A|}{\partial A} = (A')^{-1}. \tag{A.11}$$

A.7. Optimierung

Lemma A.5. *Seien $Y, X \in \mathbb{R}^{K \times T}$ mit Rang K und seien $B \in \mathbb{R}^{K \times r}, C \in \mathbb{R}^{r \times K}$ mit Rang r. Seien weiters $\lambda_1 \ge \cdots \ge \lambda_K$ die Eigenwerte von*

$$S = (XX')^{-\frac{1}{2}} XY'(YY')YX'((XX')^{-\frac{1}{2}})',$$

wobei die Matrix $(XX')^{-\frac{1}{2}}$ die Bedingung $(XX')^{-\frac{1}{2}}(XX')(XX')^{-\frac{1}{2}} = I_K$ erfüllt. Der zu λ_i gehörende Eigenvektor wird mit v_i für $i = 1, \ldots, K$ bezeichnet. Dann gilt

$$\min_{B,C} \left|\frac{1}{T}(Y - BCX)(Y - BCX)'\right| = \left|\frac{1}{T}YY'\right| \cdot \prod_{i=1}^{r}(1 - \lambda_i). \tag{A.12}$$

Dieses Minimum wird mit der Wahl $C = (v_1, \ldots, v_r)'(XX')^{-\frac{1}{2}}$ *sowie* $B = YX'C'(CXX'C')^{-1}$ *erreicht.*

Beweis. Für den Beweis wird auf die Abschnitte 4.1 und 4.2 in Tso, (1981) verwiesen. □

Anhang B.

JMulTi

In diesem Abschnitt möchten wir auf die Verwendung von JMulTi eingehen und mit dieser Software produzierte Ergebnisse der Parameterschätzung präsentieren. Die Analyse eines VECMs erfolgt bei JMulTi über eine grafische Oberfläche. Der linke Teil des JMulTi-Fensters in Abbildung B.1 zeigt eine grobe Einteilung der Funktionalität dieses Programms. Wir haben hier den Punkt *VECM Analysis* ausgewählt.

Bevor wir die Analyse durchführen können, müssen wir die zu untersuchenden Daten importieren. Dies bewerkstelligen wir mit einem Klick auf die Schaltfläche *Import Data*, welche den GUI-unterstützten Datenimport startet. In Abbildung B.1 zeigt der Mauszeiger auf diese Schaltfläche.

Nachdem die Daten eingelesen wurden, werden sie zu den Datensätzen im rechten Teil des Fensters hinzugefügt. In Abbildung B.1 sind die beiden interessierenden Zeitreihen Dp und R bereits markiert.

Im mittleren Bereich des Fensters erfolgt die Modellspezifikation. Wir haben hier $p - 1 = 3$ und $r = 1$ gewählt. Als deterministische Terme wurden ein Intercept sowie ein linearer Trend über die entsprechenden Kontrollkästchen ausgewählt. Um gewählte deterministische Terme auf die Kointegrationsbeziehung zu beschränken, sind die Kontrollkästchen im weiß hinterlegten Bereich daneben ebenfalls auszuwählen. Wir haben dies in unserem Beispiel in Abbildung B.1 nicht gemacht.

Um das gewählte Modell an die Daten anzupassen klicken wir – wie in Abbildung B.2 angedeutet – auf *Estimation* und in weiterer Folge auf *Estimation Results*. Dies führt uns zu den Ergebnissen der Parameterschätzung.

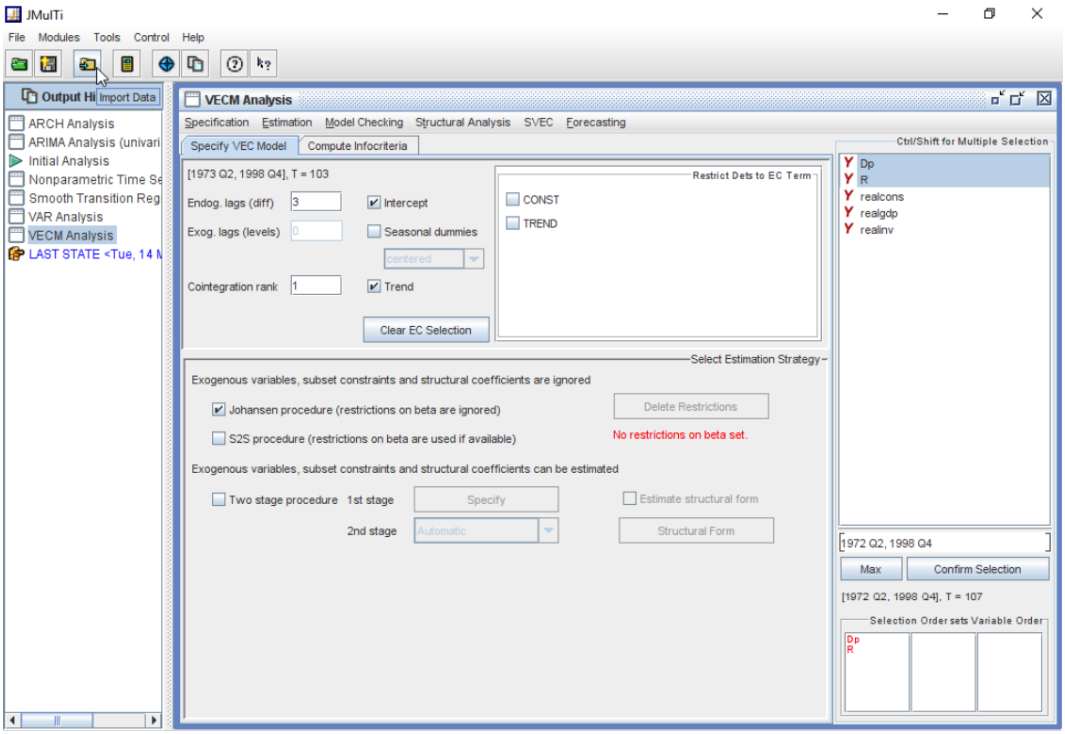

Abbildung B.1.: Import und Modellspezifikation in JMulTi.

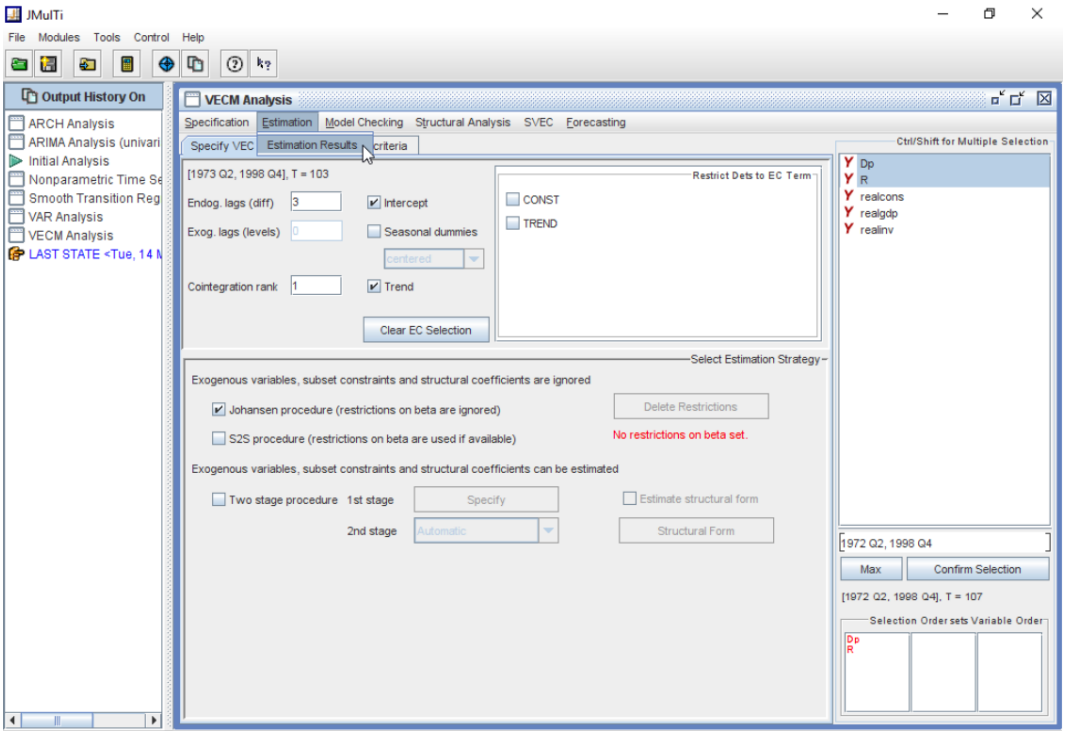

Abbildung B.2.: Übergang zu den Resultaten in JMulTi.

Wir wählen als nächstes den Reiter *Output (save/print)*. Dort befindet sich – wie in Abbildung B.3 ersichtlich – eine Zusammenfassung der Ergebnisse. Mit einem Rechtsklick und einem Klick auf *Save As* können wird diese Resultate speichern.

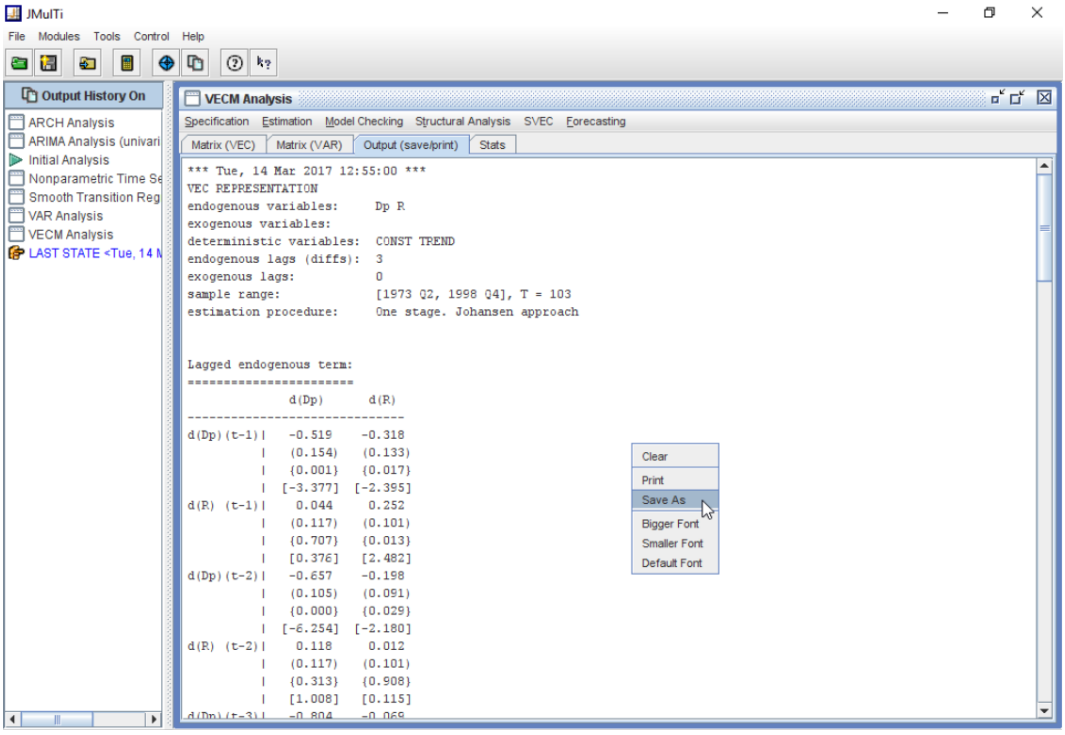

Abbildung B.3.: Anzeige und Speichern der Ergebnisse in JMulTi.

Die Ergebnisse der Parameterschätzung haben die folgende Form.

```
*** Tue, 21 Jun 2016 11:34:28 ***
VEC REPRESENTATION
endogenous variables:      Dp R
exogenous variables:
deterministic variables:   CONST TREND
endogenous lags (diffs):   3
exogenous lags:            0
sample range:              [1973 Q2, 1998 Q4], T = 103
estimation procedure:      One stage. Johansen approach
```

```
Lagged endogenous term:
=======================
               d(Dp)       d(R)
------------------------------
d(Dp)(t-1)|    -0.519     -0.318
          |    (0.154)    (0.133)
          |    {0.001}    {0.017}
          |   [-3.377]   [-2.395]
d(R) (t-1)|     0.044      0.252
          |    (0.117)    (0.101)
          |    {0.707}    {0.013}
          |    [0.376]    [2.482]
d(Dp)(t-2)|    -0.657     -0.198
          |    (0.105)    (0.091)
          |    {0.000}    {0.029}
          |   [-6.254]   [-2.180]
d(R) (t-2)|     0.118      0.012
          |    (0.117)    (0.101)
          |    {0.313}    {0.908}
          |    [1.008]    [0.115]
d(Dp)(t-3)|    -0.804     -0.069
          |    (0.056)    (0.048)
          |    {0.000}    {0.151}
          |  [-14.467]   [-1.437]
d(R) (t-3)|    -0.055      0.219
          |    (0.114)    (0.099)
          |    {0.633}    {0.026}
          |   [-0.478]    [2.221]
------------------------------


Deterministic term:
===================
              d(Dp)       d(R)
------------------------------
CONST     |    -0.008      0.005
          |    (0.003)    (0.002)
          |    {0.003}    {0.028}
          |   [-3.011]    [2.195]
TREND(t)|     0.000      0.000
          |    (0.000)    (0.000)
          |    {0.937}    {0.791}
```

```
        |  [-0.080]  [-0.264]
-----------------------------


Loading coefficients:
=====================
             d(Dp)      d(R)
-----------------------------
ec1(t-1)|    -0.635     0.419
        |    (0.201)   (0.174)
        |    {0.002}   {0.016}
        |   [-3.154]   [2.402]
-----------------------------

Estimated cointegration relation(s):
====================================
          ec1(t-1)
--------------------
 Dp(t-1)|    1.000
        |   (0.000)
        |   {0.000}
        |   [0.000]
 R (t-1)|   -0.276
        |   (0.063)
        |   {0.000}
        |  [-4.400]
--------------------



VAR REPRESENTATION

modulus of the eigenvalues of the reverse characteristic
   polynomial:
|z| = ( 1.0094     1.0116     1.0116     1.0000     1.3356
   1.3356     1.7369     1.7369     )

Legend:
=======
              Equation 1   Equation 2  ...
-----------------------------------------
Variable 1 | Coefficient          ...
           | (Std. Dev.)
           | {p - Value}
```

```
            | [t - Value]
Variable 2  |          ...
...
-----------------------------------------


Lagged endogenous term:
=======================
                  Dp          R
------------------------------
 Dp(t-1)|    -0.154      0.100
        |    (0.253)    (0.219)
        |    {0.542}    {0.647}
        |   [-0.609]    [0.458]
 R (t-1)|     0.220      1.136
        |    (0.130)    (0.112)
        |    {0.090}    {0.000}
        |    [1.693]   [10.121]
 Dp(t-2)|    -0.138      0.120
        |    (0.056)    (0.049)
        |    {0.014}    {0.013}
        |   [-2.456]    [2.473]
 R (t-2)|     0.074     -0.240
        |    (0.170)    (0.147)
        |    {0.665}    {0.102}
        |    [0.433]   [-1.633]
 Dp(t-3)|    -0.147      0.129
        |    (0.057)    (0.049)
        |    {0.009}    {0.008}
        |   [-2.597]    [2.635]
 R (t-3)|    -0.172      0.207
        |    (0.170)    (0.147)
        |    {0.311}    {0.158}
        |   [-1.013]    [1.411]
 Dp(t-4)|     0.804      0.069
        |    (0.056)    (0.048)
        |    {0.000}    {0.151}
        |   [14.467]    [1.437]
 R (t-4)|     0.055     -0.219
        |    (0.114)    (0.099)
        |    {0.633}    {0.026}
        |    [0.478]   [-2.221]
------------------------------
```

```
Deterministic term:
====================
                  Dp          R
------------------------------
CONST    |    -0.008      0.005
         |    (0.000)    (0.000)
         |    {0.000}    {0.000}
         |    [0.000]    [0.000]
TREND(t)|     0.000       0.000
         |    (0.000)    (0.000)
         |    {0.000}    {0.000}
         |    [0.000]    [0.000]
------------------------------
```

Rechts neben dem aktiven Reiter in Abbildung B.3 befindet sich ein weiterer. Er ist mit *Stats* beschriftet. Über diesen Reiter erhalten wir weitere Ergebnisse, wie etwa die Loglikelihood des Modells. Sie sind in der Folge aufgelistet.

```
*** Tue, 21 Jun 2016 11:34:28 ***
VECM MODEL STATISTICS
sample range:   [1973 Q2, 1998 Q4], T = 103

Log Likelihood:        7.697651e+02
Determinant (Cov):     9.551175e-10

Covariance:    3.579404e-05 -1.829322e-06
              -1.829322e-06  2.677719e-05

Correlation:   1.000000e+00 -5.908841e-02
              -5.908841e-02  1.000000e+00
```

Literatur

AppVeyor (2017). *Continuous Integration and Deployment service for Windows developers – AppVeyor*. URL: https://www.appveyor.com/ (besucht am 05. 02. 2017) (siehe S. 87).

Breitung, Jörg et al. (2017). *JMulTi*. URL: http://www.jmulti.de/ (besucht am 06. 01. 2017) (siehe S. 55).

Brüggemann, Ralf, Helmut Lütkepohl und Pentti Saikkonen (2006). »Residual autocorrelation testing for vector error correction models«. In: *Journal of Econometrics* 134.2, S. 579–604. ISSN: 0304-4076. URL: http://www.sciencedirect.com/science/article/pii/S0304407605001569 (siehe S. 32).

Coveralls (2017). *Coveralls - Test Coverage History & Statistics*. URL: https://coveralls.io/ (besucht am 11. 07. 2017) (siehe S. 87).

Fuller, W. A. (1996). *Introduction to Statistical Time Series*. 2. Aufl. John Wiley & Sons New York. ISBN: 978-0-471-55239-0. URL: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0471552399.html (siehe S. 7, 8).

GitHub (2017). *GitHub*. URL: https://github.com/ (besucht am 11. 07. 2017) (siehe S. 53, 55, 65, 87).

Johansen, Søren und Ernst Schaumburg (1999). »Likelihood analysis of seasonal cointegration«. In: *Journal of Econometrics* 88.2, S. 301–339. ISSN: 0304-4076. URL: http://www.sciencedirect.com/science/article/pii/S0304407698000359 (siehe S. 17).

Jupyter (2017). *Project Jupyter | Home*. URL: http://jupyter.org/ (besucht am 03. 02. 2017) (siehe S. 56).

Kilian, Lutz und Ufuk Demiroglu (2000). »Residual-Based Tests for Normality in Autoregressions: Asymptotic Theory and Simulation Evidence«. In: *Journal of Business & Economic Statistics* 18.1, S. 40–50. ISSN: 07350015. URL: http://www.jstor.org/stable/1392135 (siehe S. 30).

LeSage, James P. (2017). *Econometrics Toolbox for MATLAB*. URL: http://www.spatial-econometrics.com/ (besucht am 14. 03. 2017) (siehe S. 65).

Lütkepohl, H. (2005). *New Introduction to Multiple Time Series Analysis*. Springer Berlin Heidelberg. ISBN: 978-3-5402-7752-1. URL: http://www.springer.com/de/book/9783540401728 (siehe S. 1, 3, 5, 7, 9, 10, 12–17, 19, 25, 28–32, 38, 40, 43, 44, 46, 49–51, 55, 64, 74, 82, 99).

Lütkepohl, H. und M. Krätzig, Hrsg. (2004). *Applied Time Series Econometrics*. Cambridge: Cambridge University Press (siehe S. 55).

MacKinnon, James G. (2010). *Critical values for cointegration tests*. Techn. Ber. Queen's Economics Department Working Paper (siehe S. 17).

MacKinnon, James G., Alfred A. Haug und Leo Michelis (1999). »Numerical Distribution Functions of Likelihood Ratio Tests for Cointegration«. In: *Journal of Applied Econometrics* 14.5, S. 563–577. ISSN: 08837252, 10991255. URL: http://www.jstor.org/stable/223206 (siehe S. 17).

McKinney, Wes (2010). »Data Structures for Statistical Computing in Python«. In: *Proceedings of the 9th Python in Science Conference*. Hrsg. von Stéfan van der Walt und Jarrod Millman, S. 51–56 (siehe S. 59).

Narzo, Antonio Fabio Di, Jose Luis Aznarte und Matthieu Stigler (2017). *CRAN - Package tsDyn*. URL: https://CRAN.R-project.org/package=tsDyn (besucht am 12. 03. 2017) (siehe S. 87).

Seabold, Skipper und Josef Perktold (2010). »Statsmodels: Econometric and Statistical Modeling with Python«. In: *Proceedings of the 9th Python in Science Conference*. Hrsg. von Stéfan van der Walt und Jarrod Millman, S. 57–61 (siehe S. 1).

Stata (2017). *Data Analysis and Statistical Software | Stata*. URL: http://www.stata.com/ (besucht am 12. 03. 2017) (siehe S. 87).

statsmodels (2017). *StatsModels: Statistics in Python*. URL: http://www.statsmodels.org/ (besucht am 05. 02. 2017) (siehe S. 53).

Travis CI (2017). *Travis CI - Test and Deploy Your Code with Confidence*. URL: https://travis-ci.org/ (besucht am 05. 02. 2017) (siehe S. 87).

Tso, M. K.-S. (1981). »Reduced-Rank Regression and Canonical Analysis«. In: *Journal of the Royal Statistical Society. Series B (Methodological)* 43.2, S. 183–189. ISSN: 00359246. URL: http://www.jstor.org/stable/2984847 (siehe S. 103).

Walt, Stéfan van der, S. Chris Colbert und Gaël Varoquaux (2011). »The NumPy Array: A Structure for Efficient Numerical Computation«. In: *Computing in Science & Engineering* 13.2, S. 22–30. URL: http://aip.scitation.org/doi/abs/10.1109/MCSE.2011.37 (siehe S. 58).