Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Wirtschaft

CARF Luzern 2021

Controlling.Accounting.Risiko.Finanzen.

Konferenzband

Konferenz Homepage: www.hslu.ch/carf

Programmieren für Controller – Ein integrierter Ansatz zur Kompetenzvermittlung

Geplantes Lehrprojekt

Prof. Dr. Dominik Kramer
Hochschule Trier, Trier Business School, D 54293 Trier, E-Mail: d.kramer@hochschule-trier

Abstract
Die digitale Transformation wirkt sich auch auf das Controlling aus. Neue Technologien wie Robotic Process Automation, Predictive Analytics und Künstliche Intelligenz können z.B. im Rahmen von Reporting, operativer Planung und Kontrolle, Entscheidungsfindung sowie Risikomanagement zum Einsatz kommen. Neben den Technologien ist für das Gelingen der digitalen Transformation auch das Können der handelnden Personen von Bedeutung. Hierzu kann die Kompetenz im Controlling durch die integrierte Vermittlung von Controlling-Methoden und Programmierung als Basis sowohl für das eigene Handeln als auch für die Kommunikation mit IT-Stellen erweitert werden. Anhand dreier Beispiele aus den Bereichen Investitions-, Kosten- sowie Entscheidungsrechnung wird aufgezeigt, wie eine solche Integration erfolgen kann.

1 Einleitung

Der Begriff der digitalen Transformation beschreibt den durch die digitalen Technologien begründeten Veränderungsprozess in Unternehmen. Dieser Prozess ist i.d.R. umfassend, er wirkt sich auf Geschäftsmodelle, Produkte sowie Organisationsstrukturen aus. Die digitalen Technologien stellen dabei eine notwendige Bedingung für diesen Wandel dar, allein sind sie jedoch nicht hinreichend: Digitale Transformation brauch neben den entsprechenden Technologien auch das Wollen und Können der Mitarbeiter im jeweiligen Unternehmen (Nadkarni & Prügl, 2021, S. 234).

Die digitale Transformation hat auch Auswirkungen auf das Aufgabengebiet, die Verantwortlichkeiten sowie das Rollenverständnis des Controllings (Schoute, 2019, S. 137; Schäffer & Weber 2016, S. 9). Dabei werden grundsätzlich mehrere Entwicklungen diskutiert. Eine Extremposition ist dadurch gekennzeichnet, dass eine weitestgehende Automatisierung auch das Controlling erfasst und damit überflüssig macht (in Anlehnung an Frey & Osborn, 2017). Überwiegend wird die neue Rolle des Controllings jedoch im Spannungsfeld von Begrifflichkeiten wie Business Partner, Chief Digital Performance Officer, Business Analyst, Digital Controller bis hin zum Data Scientist diskutiert (Wolf & Heidlmayer, 2019, S. 31 ff.; in anderen Begrifflichkeiten: Schäffer & Brückner, 2019, S. 21). Konkrete Beispiele dafür, wie die Digitalisierung sich auf Aufgaben des Controllings auswirkt, sind z.B. durch Robotic Process Automation, Predictive Analytics sowie Künstliche Intelligenz gegeben. Diese Technologien können in den Gebieten Reporting, operative Planung, Kosten- und Leistungsrechnung, Forecasting oder Risikomanagement zum Einsatz kommen (Lausberg & Hoffmann, 2019, S. 60; Langmann, 2019, S. 11).

Auch in Bezug auf das Verhältnis digitale Transformation und Controlling gilt, dass neben den jeweiligen Technologien dem Wollen (in Form der Veränderungsbereitschaft) und dem Können (in Form des Know-hows) der betroffenen Personen eine entscheidende Rolle zukommt (Lausberg & Hoffmann, 2019, S. 61). Hieraus ergibt sich in Bezug auf die Kompetenzen der Controller ein erheblicher Weiterbildungsbedarf (Schulte & Blüchmann, 2016, S. 57; Seufert & Oehler, 2016). Ein Bestandteil dieser zukünftigen Kompetenzen ist das grundlegende Verständnis von Programmiersprachen (Mödritscher & Wall, 2019, S. 77; Heupel & Reinhardt, 2019, S. 128, Langmann, 2019, S. 47):

- Ein zentraler Aspekt der digitalen Transformation ist das Aufkommen der Big Data. Diese Datenmengen sind mit dem klassischen Controlling-Tool Excel nicht mehr zu bewältigen. Lausberg & Hoffmann (2019, S. 49) sprechen in diesem Zusammenhang vom beginnenden Ende des Excel-Zeitalters.
- Sollen die angesprochenen Technologien Robotic Process Automation, Predictive Analytics sowie Künstliche Intelligenz auch vom Controlling eingesetzt werden, sind grundlegende Kenntnisse in den entsprechenden Programmiersprachen unumgänglich.
- Das gilt auch für den Fall, dass die eigentliche Umsetzung dieser Technologien nicht durch das Controlling selbst, sondern durch andere Stellen (IT, Data Science, …) erfolgt. Ein Verständnis der grundlegenden Prinzipien der Programmiersprachen stellt dann sicher, dass das Controlling die Fähigkeit hat, aufkommende neue Technologien zu verstehen und mit den umsetzenden Stellen adäquat zu kommunizieren und zu agieren (Richins et all., 2017, S. 74; Heupel & Reinhardt, 2019, S. 128).

Aus Sicht der Ausbildung stellt sich damit die Frage, wie solche programmierspezifischen Kenntnisse vermittelt werden können. Neben der Vermittlung einer Programmiersprache in einem eigenen Kurs bietet sich als zweiter Weg an, Elemente der Programmierung mit den fachspezifischen Problemen und Methoden des Controllings zu verzahnen. Eine solche integrierte Vorgehensweise verdeutlicht den Studierenden die Sinnhaftigkeit des Einsatzes der Programmierung; ggf. gelingt es ergänzend durch die direkte Umsetzung, auch das Verständnis für die Methoden zu vertiefen. An dieser Stelle setzen die vorliegenden Ausführungen an. Nach einer kurzen Charakterisierung der aus Sicht der Themenstellung relevanten Programmiersprachen – auch in Abgrenzung zur Tabellenkalkulation Excel – soll anhand dreier Beispiele aus den Bereichen Investitions-, Kosten- und Entscheidungsrechnung verdeutlicht werden, wie eine solche Verzahnung in der Ausbildung umgesetzt werden kann.

2 Charakterisierung der relevanten Programmiersprachen

2.1 Programmiersprachen in den Wirtschaftswissenschaften

Die in den Wirtschaftswissenschaften verwendeten Programmiersprachen stellen nicht die eigentliche Programmierung, sondern die Nähe zur Mathematik, die Abbildung von Vektoren und Matrizen sowie die Vielfallt der Methoden in den Vordergrund. Dementsprechend bilden sie z.B. breite Bereiche der Statistik, der Optimierung und des Data Science ab. Hier sind insbesondere Matlab (1984), R (2000), Python (1991; zusammen mit NumPy (2006)) sowie Julia (2018) zu nennen (Angaben in Klammern: Datum Release Version 1.0). Die vier hier genannten Sprachen sind durch eine Vielzahl von Paketen gekennzeichnet, die für statistische und ökonomische Fragegestellungen geeignet sind. Ferner können alle Sprache im Dialog genutzt werden: Das Abarbeiten einzelner Programmschritte ist möglich.

Alle vier Sprachen sind für die hier beschriebene Aufgabe grundsätzlich gut geeignet. Bei der Auswahl der einzusetzenden Sprache sollte berücksichtigt werden, welche Programmiersprachen in anderen Kursen der betriebswirtschaftlichen Ausbildung zum Einsatz kommen. Nutzen mehrere Kurse die gleiche Sprache, können Synergien realisiert werden; für die Studierenden bietet sich die Gelegenheit, das in einem Kurs Erlernte in folgenden Kursen zu vertiefen. Wenn hingegen eine Anlehnung an andere Kurse nicht möglich bzw. nicht erwünscht ist, kann die Auswahl der Sprache anhand der Kriterien Spracheigenschaften, Datenhandhabung, Bibliotheken, Graphik, Arbeitsumgebung sowie Ausführungsgeschwindigkeit und Kosten erfolgen. Einen an diesen Kriterien orientierten vollständigen Vergleich bieten z.B. Aguirre & Danielsson (2020).

Operation	Mathematik	Matlab	R	Python	Julia
Erstellen eines Vektors	$A = \begin{pmatrix}1\\2\\3\end{pmatrix}$	A = [1; 2; 3]	A <- c(1, 2, 3)	A = np.array([1, 2, 3])	A = [1; 2; 3]
Erstellen einer Matrix	$A = \begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\end{pmatrix}$	A = [1 2 3; 4 5 6]	A <- rbind(c(1,2,3),c(4,5,6))	A = np.array([[1, 2, 3],[4,5,6]])	A = [1 2 3; 4 5 6]
Matrixmultiplikation	A · B	A * B	A %*% B	np.dot(A, B) A @ B	A * B
Lösen linearer Gleichungssysteme	$x = A^{-1} \cdot b$	x = A^-1 * b x = A\b	x <- solve(A, b)	x = np.linalg.solve(A, b)	x = A^-1 * b x = A\b
Transponieren	A'	A'	t(A)	A.transpose()	A'

Funktionen (Bsp.)	f(x, y) = 3x + 2y - x·y	function out = f(x, y)
		out = 3*x+2*y-x*y
		end
			f <- function(x, y) {
			3*x+2*y-x*y
			}
				def f(x, y):
				return 3*x+2*y-x*y
					f(x, y) = 3x + 2y - x * y

Mehrfachberechnung von f(x, y)	$z_i = f(x_i, y_i) \ \forall i \in I$	z = arrayfun("f", x, y)	z <- f(x, y)	z = f(x, y) z = [f(x[i], y[i]) for i in I]	z = f.(x, y) z = [f(x[i], y[i]) for i in I]

Abbildung 1: Ausgewählte Spracheigenschaften

Da – wie schon im voranstehenden Absatz betont – alle vier Programmiersprachen grundsätzlich gut geeignet sind, soll im Folgenden anhand des Kriteriums Spracheigenschaften eine Auswahl der Sprache für die weiteren Darstellungen in erfolgen: Dabei steht die Nähe zwischen mathematischer Formulierung und Abbildung in der jeweiligen Programmiersprache im Vordergrund. Je größer diese Nähe ist, umso leichter ist es möglich, Controlling-relevante Methoden in den Kursen erst mathematisch einzuführen und dann mit Hilfe der Programmierung umzusetzen. Unter diesem Aspekt gibt Abbildung 1 einen kleinen Einblick in ausgewählte Spracheigenschaften der vier Sprachen. Dabei zeigt sich, dass Julia und – mit Abstrichen – Matlab sehr eng an mathematische Formulierungen angelehnt sind. Bei R und Python (mit NumPy) hingegen sind größere Abweichungen erkennbar. Ergänzend ist Julia dadurch gekennzeichnet, dass eine in der Vermittlung oft problematische Vektorisierung von Berechnungen sowohl aus Sicht der Ausdrücke als auch der Berechnungsgeschwindigkeit problemlos vermieden werden kann. Weiterhin startet die Indizierung von Vektoren und Matrizen mit dem Index 1. Deshalb stützen sich die Ausführungen in Kapitel 3 beispielhaft auf Julia.

2.2 Programmierung versus Excel

Aktuell ist die Tabellenkalkulation Excel ein de Facto-Standard bei der Arbeit im Controlling. Eine Vielzahl von anwendungsorientierten Publikationen zeigt auf, wie Excel im Controlling eingesetzt werden kann (z.B. Nelles, 2019; Oehler & Schwabe, 2020; Schels & Seidel, 2020). Excel bietet den Benutzern einen schnellen, fast intuitiven Zugang und führt – zumindest bei kleineren Problemstellungen – schnell zu Erfolgserlebnissen. Damit stellt sich Excel als so etwas wie das Schweizer Messer des Controllings dar. Im Gegensatz dazu können die Programmiersprachen als vollständig ausgestatte Werkstätten beschrieben werden, die unter anderem durch folgende Vor- und Nachteile, startend mit den Nachteilen, charakterisiert werden können (in Anlehnung an Sauer, 2019, S. 16 ff.):

- Die Einarbeitung in die Programmiersprachen ist aufwendiger. Die formalen Regeln und Notationen der jeweiligen Sprache sind zu beachten, so kann z.B. schon ein zusätzliches Leerzeichen zu einer Fehlermeldung führen. Weiterhin zwingen Programmiersprachen zur Strukturierung, eine intuitive Problembearbeitung wie bei Excel ist kaum möglich.
- Ferner sind Datentypen genau zu unterscheiden. Ein beliebiges Mischen von Datentypen wie in der Tabellenkalkulation ist nicht möglich.
- Die (formalen) Fehlermeldungen sind oft schwer nachvollziehbar.
- Programmiersprachen arbeiten in speziellen Umgebungen, den sogenannten IDEs (Integrated Development Environment). Beispiele hierfür sind das RStudio für R oder die Jupyter Notebooks, in denen mit R, Python oder Julia gearbeitet werden kann. Auch hier ist eine Einarbeitung notwendig.

Diesen Nachteilen, die sich vor allem auf die anfängliche Einarbeitung beziehen, stehen deutliche Vorteile gegenüber:

- Programmiersprachen zwingen zum exakten Arbeiten, zur Strukturierung sowie zur Trennung von Daten und Verarbeitungsschritten. Dadurch wird – insbesondere bei größeren Problemstellungen mit mehreren Schritten – eine bessere Reproduzierbarkeit und Dokumentation erreicht. Dies erleichtert eine spätere (inhaltliche) Fehleranalyse sowie die Weitergabe an andere Personen.
- Die einzelnen Verarbeitungsschritte sind unabhängig von der Größe der zugrundeliegenden Daten. Die Abarbeitung großer Datenbestände ist möglich.
- Eine Automatisierung von Aufgaben ist mit geringem Zusatzaufwand zu realisieren. Hier müsste z.B. bei Excel auf VBA zurückgegriffen und damit dann letztlich doch wieder eine Programmiersprache eingesetzt werden.
- Die Programmiersprachen sind über Zusatzpakete nahezu beliebig erweiterbar. Insbesondere ermöglichen diese Pakete den Zugriff auf moderne und umfängliche Analyseverfahren. Weiterhin erlauben Graphikerweiterungen die Erstellung von sehr hochwertigen Diagrammen.
- Mit Ausnahme von Matlab[1] sind die hier genannten Sprachen kostenlos. Dies gilt nicht für die Tabellenkalkulation Excel.

Zusammenfassend zeigt sich, dass die Vorteilhaftigkeit der Programmierung gegenüber der Tabellenkalkulation vor allem von der Komplexität der jeweiligen Problemstellung abhängt: Erst beim Vorliegen umfangreicherer Problemstellungen mit großen Datensätzen sowie ggf. wiederholter Ausführung lohnt sich der Einsatz der Programmierung. Genau diese Rahmenbedingungen sind aber durch die fortschreitende Digitalisierung vermehrt gegeben.

[1] Octave stellt eine gebührenfreie Alternative zu Matlab dar, allerdings ist der Umfang der Ergänzungspakete eingeschränkt.

3 Anwendungsbeispiele aus dem Controlling

In diesem Kapitel soll beispielhaft verdeutlicht werden, wie ausgewählte Methoden des Controllings mit Hilfe einer Programmiersprache – hier Julia[2,3] – umgesetzt werden können. Die Beispiele stammen dabei aus den Bereichen der Investitions-, der Kosten- sowie der Entscheidungsrechnung. Dabei wird nicht mit vordefinierten Funktionen gearbeitet, sondern die Methoden bzw. Modelle werden eigenständig in Julia realisiert.

3.1 Beispiel Investitionsrechnung

Mit Hilfe der Investitionsrechnung soll die Beurteilung und Auswahl von Investitionen ermöglicht werden. Dabei kann zwischen statischen, dynamisch tabellenorientierten und dynamisch formelorientierten Verfahren unterschieden werden. Beispielhaft werden hier die dynamisch formelorientierten Verfahren betrachtet und der Kapitalwert, der Endwert, die Annuität sowie der interne Zins bestimmt (Götze, 2014, S. 73 ff; Busse von Colbe & Witte, 2018, S. 51 ff.; Müller, 2019, S. 338 ff.). Die zu beurteilende Investition möge durch folgende Daten charakterisiert werden: Die Investition startet im Zeitpunkt $t = 0$ mit einer Auszahlung in Höhe von I. Es folgt über den Zeitraum von $t = 1$ bis $t = T$ der laufende Strom der Ein- und Auszahlungen, die als Z_t zusammengefasst werden. Am Ende der Laufzeit der Investition in $t = T$ kann dann noch ein Liquidationserlös in Höhe von L realisiert werden. Der Diskontierungszinssatz liege bei i. Der Kapitalwert KW der Investition in Abhängigkeit von diesen Daten berechnet sich als:

$$KW = -I + \sum_{t=1}^{T} \frac{Z_t}{(1+i)^t} + \frac{L}{(1+i)^T} \tag{1}$$

Im Gegensatz zum Kapitalwert ergibt sich der Endwert EW durch das Aufdiskontieren der einzelnen Werte. Der Endwert kann auch aufbauend auf dem Kapitalwert bestimmt werden:

$$EW = -I \cdot (1+i)^T + \sum_{t=1}^{T} Z_t \cdot (1+i)^{T-t} + L = KW \cdot (1+i)^T \tag{2}$$

Auch die Annuität AN kann mit Hilfe des Annuitätenfaktors direkt aus dem Kapitalwert ermittelt werden:

$$AN = KW \cdot \frac{i \cdot (1+i)^T}{(1+i)^T - 1} \tag{3}$$

Die nachfolgende Abbildung 2 zeigt die Umsetzung dieser drei Formeln sowie die Anwendung auf ein einfaches Beispiel in einem Jupyter-Notebook.

[2] Zu anderen Programmiersprachen vgl. Ozgur et all. (2018), die den Einsatz von R im Bereich der Finanzierung darstellen, dabei jedoch nicht die Methoden umsetzen, sondern mit vordefinierten Funktionen arbeiten. Schoute (2019) gibt ein Beispiel für den Einsatz von Python im Bereich der Kostenrechnung, die dort gewählte Art der Umsetzung ist allerding sehr umständlich.
[3] Eine Einführung in Julia geben z.B. Bezanson et all. (2018) und Lauwens & Downey (2019).

```
In [1]:
# Funktionen
KW(I, Z, L, i) = -I + sum(Z[t]/(1+i)^t for t=1:length(Z)) + L/(1+i)^length(Z)
EW(I, Z, L, i) = KW(I, Z, L, i)*(1+i)^length(Z)
AN(I, Z, L, i) = KW(I, Z, L, i)*(i*(1+i)^length(Z)/((1+i)^length(Z)-1));

In [2]:
# Daten
I = 100
Z = [30; 50; 40]
L  = 20
i  = 0.1;

In [3]:
KW(I, Z, L, i)

Out[3]: 13.673929376408699

In [4]:
EW(I, Z, L, i)

Out[4]: 18.199999999999985

In [5]:
AN(I, Z, L, i)

Out[5]: 5.498489425981862
```

Abbildung 2: Beispiel zur Investitionsrechnung in Julia

Zuerst werden im Block In [1] die eben besprochenen drei Funktionen definiert:

– In Zeile 2 wird der Kapitalwert KW als eine Funktion in Abhängigkeit von den in den Kapitalwert einfließenden Daten I, Z (als Vektor der Zahlungsreihe Z_t), L und i abgebildet. Die neu definierte Funktion wird blau hinterlegt, bekannte Sprachelemente von Julia werden grün und mathematische Operatoren lila markiert. Die Funktion folgt dem Aufbau der Formel 1: Zuerst wird die Investitionsauszahlung erfasst. Es folgt die Summe (sum) der diskontierten Elemente der Zahlungsreihe, der Laufbereich der Summe wird durch das Wort for abgebildet. Der Endwert des Laufbereichs T wird nicht explizit angegeben, sondern aus der Länge des Vektors der Zahlungsreihe (length(Z)) abgeleitet. Dieser Wert wird auch bei der Diskontierung des Liquidationserlöses verwendet.

– Da die Funktion des Kapitalwerts in Julia nun schon implementiert ist, wird sie gemäß Formel 2 zur Bestimmung des Endwerts EW in Zeile 2 benutzt. Dabei wird zuerst der Kapitalwert berechnet und dann aufdiskontiert. Wiederum wird T aus der Länge des Vektors Z abgeleitet.

– In Zeile 3 wird die Annuität AN gemäß Formel 3 umgesetzt.

Im Block In [2] wird ein einfacher Datensatz für eine beispielhafte Investition mit den schon bekannten Symbolen angelegt. In den nachfolgenden drei Blöcken werden dann Kapitalwert, Endwert und Annuität berechnet, die Ergebnisse werden jeweils im Out-Block darunter angezeigt.

Als letzte Kennziffer der Investitionsrechnung soll nun ergänzend der interne Zins bestimmt werden. Dabei wird – ausgehend von einem gegeben Startwert – mit Hilfe des Newtonverfahrens ein Folgezins berechnet. Dieses Prozedere wird so lange wiederholt, bis der interne Zins mit hinreichender Genauigkeit ermittelt wurde:

$$i_{n+1} = i_n - \frac{KW(i_n)}{KW'(i_n)} \tag{4}$$

Die Umsetzung und Berechnung für den schon definierten Datensatz in Julia werden in Abbildung 3 dokumentiert.

In [6]:
```
1  ΔKW(I, Z, L, i, Δ) = (KW(I, Z, L, i+Δ) - KW(I, Z, L, i-Δ))/(2*Δ)
2
3  function IZ(I, Z, L, i)
4      Δ = 0.0000001
5      i_n = i
6      for j = 1:100
7          i_n = i_n - KW(I, Z, L, i_n) / ΔKW(I, Z, L, i_n, Δ)
8      end
9      return i_n
10 end;
```

In [7]:
```
1  IZ(I, Z, L, i)
```

Out[7]: 0.16794936144622963

Abbildung 3: Beispiel zur Investitionsrechnung in Julia (Fortsetzung)

Das Newtonverfahren in Block In [6] gemäß Formel 4 benötigt zum einen den Kapitalwert – er wurde schon in In [1] implementiert – und zum anderen die erste Ableitung des Kapitalwerts ΔKW. Diese kann über den zentralen Differenzenquotienten in Zeile 1 numerisch approximiert werden.

Der interne Zins wird dann in den Zeilen 3 bis 10 berechnet. In Zeile 3 werden der Funktionsname IZ sowie die Eingabewerte festgelegt. In Zeile 4 wird der zur Bestimmung des Differenzenquotienten benötigte Wert Δ festgelegt. In Zeile 5 wird der Startwert für das Newtonverfahren übergeben. In Zeile 7 erfolgt die eigentliche Berechnung gemäß Formel 4, diese wird (for-end-Schleife in den Zeilen 6 bis 8) 100mal durchgeführt.[4] In Zeile 9 schließlich wird das Ergebnis der Funktion übergeben (return).

In In [7] erfolgt dann die Berechnung mit dem vorher schon definierten Datensatz. Der interne Zins wird in Out [7] angegeben.

Die voranstehenden Ausführungen haben verdeutlicht, dass die klassischen Kennziffern der dynamisch formelorientierten Investitionsrechnung in einfachen Funktionen umgesetzt werden können. Lediglich die Bestimmung des internen Zinses mit Hilfe des Newtonverfahrens ist programmtechnisch etwas aufwendiger; hier kann jedoch die Programmierung gut genutzt werden, um den iterativen Ablauf des Verfahrens zu verdeutlichen.

3.2 Beispiel Kostenrechnung

Das zweite Beispiel kommt aus dem Bereich der Kostenrechnung, es geht um die innerbetriebliche Leistungsverrechnung mit Hilfe des Gleichungsverfahrens im Rahmen der Kostenstellenrechnung (Götze, 2010, S. 89 ff.; Ewert & Wagenhofer, 2014, S. 663; Plinke et all., 2015, S. 95 f.). Dabei wird ein Unternehmen mit mehreren Kostenstellen betrachtet, K kennzeichnet die Indexmenge dieser Stellen. Von diesen Kostenstellen sind einige Hilfskostenstellen, H kennzeichnet die Indexmenge dieser Stellen. Die restlichen Stellen sind Hauptkostenstellen. Die primären Gemeinkosten der einzelnen Kostenstellen liegen schon vor und werden über den Vektor PGK abgebildet. Die Hilfskostenstellen erbringen Leistungen, abgebildet über die Matrix L, sowohl für die Hilfs- als auch für die Hauptkostenstellen. In Relation zu diesen Leistungen sollen die primären Gemeinkosten der Hilfskostenstellen auf alle Kostenstellen mit Hilfe des Gleichungsverfahrens verteilt werden. Dazu sind in einem ersten Schritt die Verrechnungssätze der Hilfskostenstellen, erfasst über den Vektor V, zu bestimmen. Dies geschieht mit Hilfe des folgenden Gleichungssystems:

$$a_i \cdot V_i = PGK_i + \sum_{j \in H} L_{ji} \cdot V_j \quad \forall\, i \in H \tag{5}$$

a_i kennzeichnet dabei die gesamte abgegebene Leistung einer Hilfskostenstelle mit $a_i = \sum_{j \in K} L_{ij} \quad \forall\, i \in H$. Überführt in die Matrixschreibweise ergib sich:

[4] Um die Funktion so einfach wie möglich zu halten, werden sowohl der Wert Δ für die Bestimmung des Differenzenquotienten als auch die Anzahl der Berechnungen vorgegeben, auf die Implementierung eines expliziten Abbruchkriteriums wird bewusst verzichtet.

$$A \cdot V = PGKH + B \cdot V \tag{6}$$

Dabei ist A eine Matrix, die auf der Diagonalen die Leistungsabgaben der Hilfskostenstellen a_i aufweist, die anderen Elemente der Matrix sind Null. $PGKH$ ist der Vektor der primären Gemeinkosten nur der Hilfskostenstellen und die Matrix B umfasst aus der Matrix L die transponierten Leistungsverflechtungen zwischen den Hilfskostenstellen. Die Verrechnungssätze werden dann bestimmt als:

$$V = (A - B)^{-1} \cdot PGKH \tag{7}$$

Im zweiten Schritt können die sekundären Gemeinkosten SGK (als Vektor) der Kostenstellen berechnet werden:

$$SGK_j = \sum_{i \in H} L_{ij} \cdot V_i \quad \forall\, j \in K \tag{8}$$

AGK ist der Vektor der Kosten, die im Rahmen der innerbetrieblichen Leistungsverrechnung abgegeben bzw. weitergereicht werden:

$$AGK_j = \begin{cases} a_j \cdot V_j & j \in H \\ 0 & sonst \end{cases} \quad \forall\, j \in K \tag{9}$$

Die gesamten Gemeinkosten GK (als Vektor) der Kostenstellen ergeben sich dann als:

$$GK_j = PGK_j + SGK_j - AGK_j \quad \forall\, j \in K \tag{10}$$

Abbildung 4 zeigt die Umsetzung der innerbetrieblichen Leistungsverrechnung mit Julia im Jupyter-Notebook. Die Berechnungen lehnen sich an das Beispiel von Togo (2012) an. Betrachtet wird ein Unternehmen mit 7 Kostenstellen, vier davon sind Hilfskostenstellen (HR: Human Resources; PD: Product Development; IT: Information Technology; MA: Maintenance), die restlichen 3 sind Hauptkostenstellen (LT: Laptop Computers; VG: Video Games; ER: Early Readers).

In [2]:
```
# Datenerfassung
# KoSt.    HR      PD      IT      MA       LT       VG       ER
PGK = [80_000; 60_000; 40_000; 20_000; 200_000; 180_000; 120_000]

L   = [     0      15      25      10       40        5        5;  # HR
           25       0      20      15       35        0        5;  # PD
           20      10       0       5        0       45       20;  # IT
           10       5      15       0        5       35       30]  # MA;
```

In [3]:
```
# Indexmengen
K = 1:7    # Index der Kostenstellen
H = 1:4    # Index der Hilfskostenstellen;
```

In [4]:
```
# Bestimmung der Verrechnungspreise mit Hilfe des Gleichungsverfahrens
a    = [sum(L[i, j] for j in K) for i in H]
A    = [i==j ? a[i] : 0 for i in H, j in H]
B    = L[H, H]'
PGKH = PGK[H]
V    = (A - B)^-1 * PGKH
print(V)
```

```
[1275.94, 914.99, 979.05, 513.79]
```

In [5]:
```
# Ermittlung der gesamten Gemeinkosten je Kostenstelle
SGK = [sum(L[i,j]*V[i] for i in H) for j in K]
AGK = [j in H ? a[j]*V[j] : 0 for j in K]
GK  = [PGK[j] + SGK[j] - AGK[j] for j in K]
print(GK)
```

```
[0.00, -0.00, -0.00, 0.00, 285630.87, 248419.71, 165949.42]
```

Abbildung 4: Beispiel zur Kostenrechnung in Julia[5]

[5] Im nicht angezeigten Block In [1] wird die Anzeigengenauigkeit der Zahlen mit zwei Nachkommastellen eingestellt.

In Block In [2] werden die Daten des Beispiels erfasst. Dies sind die primären Gemeinkosten je Kostenstelle PGK (Zeile 3) sowie die Matrix der Leistungserbringung der Hilfskostenstellen an die Kostenstellen L in den Zeilen 5 bis 8. In Block In [3] werden dann die Indexmengen aller Kostenstellen K (Zeile 2) sowie aller Hilfskostenstellen H (Zeile 3) festgelegt.

Die notwendigen Berechnungen zur Bestimmung der Verrechnungssätze finden sich in Block In [4]: In Zeile 2 werden die von den Hilfskostenstellen abgegebenen Mengen a_i ermittelt. Darauf aufbauen wird in Zeile 3 die Matrix A erzeugt, auf der Diagonalen dieser Matrix find sich die vorab bestimmten a_i. In Zeile 4 werden aus der Matrix L die Leistungsverflechtungen zwischen den Hilfskostenstellen (in transponierter Form) in die Matrix B gesetzt und in Zeile 5 der Vektor der primären Gemeinkosten der Hilfskostenstellen $PGKH$ aus dem Vektor PGK ermittelt. Die Bestimmung der Verrechnungssätze gemäß Formel 7 folgt in Zeile 6. Über den print-Befehl in Zeile 7 werden diese Sätze dann unter dem Block angezeigt.

Die Ermittlung der gesamten Gemeinkosten je Kostenstelle schließt sich in Block In [5] an. In Zeile 2 werden die sekundären Gemeinkosten SGK nach Formel 8 und in Zeile 3 die abgegebenen Gemeinkosten AGK nach Formel 9 berechnet. Die gesamten Gemeinkosten GK je Kostenstelle werden in Zeile 4 nach Formel 10 bestimmt und über Zeile 5 unter dem Block angezeigt

Insgesamt folgt damit die innerbetriebliche Leistungsverrechnung genau dem vorher aufgezeigten Formelapparat. Die Blöcke In [2] und In [3] bilden die fallspezifischen Informationen des Beispiels an. Die Berechnungen in den Blöcken In [4] und In [5] sind allgemeingültig und können genau in dieser Form auch auf andere Beispiele angewendet werden. Hier zeigt sich der Vorteil der Trennung von Daten und Berechnungen z.B. im Vergleich zu einer Lösung in einer Tabellenkalkulation: Für andere – z.B. deutlich umfangreichere – Beispiele müssen nur die Blöcke In [2] und In [3] angepasst werden.

3.3 Beispiel Entscheidungsrechnung

Das letzte Beispiel kommt aus dem Feld der Entscheidungsrechnung, welches dem Bereich der Planung zuzuordnen ist. Aufbauend u.a. auf den Informationen des internen Rechnungswesens werden betriebliche Entscheidungsbereiche modellmäßig abgebildet, und mit Hilfe geeigneter Algorithmen wird eine optimale Lösung für das Modell bestimmt. Ein Beispiel hierfür ist die operative Produktionsprogrammplanung (Götze, 2010, S. 186 ff.; Ewert & Wagenhofer, 2014, S. 97 ff.). Dabei gilt es festzulegen, in welchen Mengen die einzelnen Produkte des Unternehmens unter Berücksichtigung der marktseitigen und kapazitativen Beschränkungen hergestellt und abgesetzt werden sollen. Da aus operativer Sicht die fixen Kosten entscheidungsirrelevant sind, liegt diesem Modelltyp die Maximierung des gesamten Deckungsbeitrags zugrunde. Das allgemeine Modell (A-Modell) für diese Problemstellung kann wie folgt formuliert werden:

Indizes.	j	Index der Produkte, $j \in J$
	i	Index der Kapazitäten, $i \in I$
Variablen.	x_j	Produktions- und Absatzmenge von Produkt j [ME], $j \in J$
Parameter.	db_j	(Stück-) Deckungsbeitrag von Produkt j [GE/ME], $j \in J$
	A_i	Verfügbare Kapazität vom Typ i [KE], $i \in I$
	a_{ij}	Beanspruchungskoeffizient von Produkt j bei Kapazität i [KE/ME], $i \in I, j \in J$
	x_j^{max}	Absatzobergrenze von Produkt j [ME], $j \in J$

Zielfunktion. $Z = \sum_{j \in J} db_j \cdot x_j \rightarrow max!$

Restriktionen.

$\sum_{j \in J} a_{ij} \cdot x_j \leq A_i$	$\forall\, i \in I$	(Kapazitäten)
$x_j \leq x_j^{max}$	$\forall\, j \in J$	(Absatzobergrenzen)
$x_j \geq 0$	$\forall\, j \in J$	(Nicht-Negativitätsbedingungen)

Im A-Modell werden zuerst die Indizes festgelegt; hier werden Indizes für die unterschiedlichen Produkte sowie für die die Produktion beschränkenden Kapazitäten benötigt. Anschließend werden die Variablen definiert. In diesem Modell wird nur ein Variablentyp verwendet, es sind dies die zu optimierenden Produktions- und Absatzmengen der einzelnen Produkte. Es folgenden die Parameter, die bei der späteren Ausformulierung zum konkreten Modell (K-Modell) durch konkrete Zahlen ersetzt werden. Parameter sind hier die (Stück-) Deckungsbeiträge der einzelnen Produkte und die für die Produktion zur Verfügung stehenden Kapazitäten z.B. von Maschinen, Rohstoffen oder Personal. Es folgen die produktspezifischen Beanspruchungskoeffizienten, sie zeigen auf, wie viele Kapazitätseinheiten der jeweiligen Kapazitäten für die Fertigung einer Mengeneinheit eines Produktes benötigt werden. Die Absatzobergrenzen als marktseitige Beschränkung schließen den Satz der Parameter ab.

Das eigentliche Modell besteht aus einer Zielfunktion sowie einem Satz von Restriktionen. In der Zielfunktion wird der gesamte Deckungsbeitrag aus den einzelnen Stückdeckungsbeiträgen multipliziert mit den Produktions- und Absatzmengen ermittelt. Der erste Restriktionstyp (Kapazitäten) stellt für jede einzelne Kapazitätsart sicher, dass die Inanspruchnahme durch das Produktionsprogramm (linke Seite) nicht die zur Verfügung stehende Kapazitätsmenge (rechte Seite) überschreitet. Über den zweiten Restriktionstyp (Absatzobergrenzen) wird je Produktart sichergestellt, die maximal mögliche Absatzmenge nicht überschritten wird. Die Nicht-Negativitätsbedingen sorgen dafür, dass negative Produktions- und Absatzmengen ausgeschlossen werden.

In Abbildung 5 wird aufgezeigt, wie dieses A-Modell für ein konkretes Beispiel in Julia umgesetzt werden kann.

In [1]:
```julia
# Produktionsprogrammplanung
using JuMP, Clp   # Modellierungssprache und Solver
```

In [2]:
```julia
# Indizes
J = 1:3     # Index der Produkte
I = 1:2;    # Index der Kapazitäten
```

In [3]:
```julia
# Parameter
#       P1  P2  P3
db   = [30; 50; 60]   # Deckungsbeiträge
a    = [ 3   6   7;   # Maschinenkoeffizienten
         5   4   5]   # Rohstoffkoeffizienten
xmax = [10; 10;  5]   # Absatzobergrenzen

A    = [30; 40];      # Verfügbare Kapazitäten Maschine/Rohstoff
```

In [4]:
```julia
# Modell
PPP = Model(Clp.Optimizer)

@variables PPP begin
    0<= x[j in J]
end

@objective(PPP, Max, sum(db[j]*x[j] for j in J))

@constraints PPP begin
    Kapazitaet[i in I],   sum(a[i,j]*x[j] for j in J) <= A[i]
    Max_Absatz[j in J],   x[j] <= xmax[j]
end
```

Abbildung 5: Beispiel zur Entscheidungsrechnung in Julia

In Block In [1] werden zwei Pakete geladen: JuMP stellt in Julia eine Modellierungssprache zur Verfügung, die eng an die Darstellung des A-Modells angelehnt ist. Clp lädt ein Interface zu einem linearen Solver, der eine Lösung des Modells erlaubt. Über die in Block In [2] festgelegten Indexmengen wird die Dimension des K-Modells festgelegt: In diesem Beispiel werden drei Produkte sowie zwei zur Produktion benötigten Kapazitätsarten (eine Maschine und ein Rohstoff) betrachtet. Es folgen in Block In [3] die konkreten Zahlen für die oben genannten Parameter.

Die eigentliche Modellierung folgt im Block In [4]. In Zeile 2 wird das Modell benannt (PPP) und mit dem Solver verknüpft. In den Zeilen 4 bis 6 folgt die Deklaration der Variablen des Modells PPP: Hier wird für jedes Produkt

der Indexmenge J eine Variable x_j erzeugt; zugleich wird hier schon die Nicht-Negativitätsbedingung abgebildet. Anschließend wird in Zeile 8 die Zielfunktion definiert: Zuerst wird das Modell – hier PPP – benannt, dann die Optimierungsrichtung – hier Maximierung –, es folgt die eigentliche Zielfunktion in der schon bekannten Summenschreibweise von Julia. In den Zeilen 10 bis 13 finden sich die Restriktionen: Zuerst werden die Restriktionen benannt (Kapazitaet bzw. Max_Absatz), dabei wird auch der Allquantor der jeweiligen Restriktion erfasst. Dann folgen die einzelnen Restriktionen mit linker und rechter Seite.

```
In [5]:
1 # Ausdruck
2 print(PPP)
```

```
Max 30 x[1] + 50 x[2] + 60 x[3]
Subject to
 Kapazitaet[1] : 3 x[1] + 6 x[2] + 7 x[3] <= 30.0
 Kapazitaet[2] : 5 x[1] + 4 x[2] + 5 x[3] <= 40.0
 Max_Absatz[1] : x[1] <= 10.0
 Max_Absatz[2] : x[2] <= 10.0
 Max_Absatz[3] : x[3] <= 5.0
 x[1] >= 0.0
 x[2] >= 0.0
 x[3] >= 0.0
```

```
In [6]:
1 # Modelllösung
2 optimize!(PPP)
```

```
Coin0506I Presolve 2 (-3) rows, 3 (0) columns and 6 (-3) elements
Clp0006I 0  Obj -0 Dual inf 148.33333 (3)
Clp0006I 2  Obj 285
Clp0000I Optimal - objective value 285
Coin0511I After Postsolve, objective 285, infeasibilities - dual 0 (0), primal 0 (0)
Clp0032I Optimal objective 285 - 2 iterations time 0.002, Presolve 0.00
```

```
In [7]:
1 # Ausdruck der Lösung
2 println("ZF        ", objective_value(PPP))
3 println("Var       ", value.(x))
```

```
ZF        285.0
Var       1-dimensional DenseAxisArray{Float64,1,...} with index sets:
    Dimension 1, 1:3
And data, a 3-element Array{Float64,1}:
 6.5
 0.0
 1.5
```

Abbildung 6: Beispiel zur Entscheidungsrechnung in Julia (Fortsetzung)

In Abbildung 6 wird aufgezeigt, wie mit dem Modell gearbeitet werden kann. In Block In [5] bewirkt der print-Befehl, dass das hinter dem A-Modell stehende K-Modell angezeigt wird. Im Block In [6] wird das Modell über den Befehl optimize! gelöst. Die dabei vom Solver kommenden Information werden unter dem Block angezeigt. Abschließend wird in Block In [7] auf konkrete Werte der Optimallösung zugegriffen: In diesem Beispiel sind es der optimale Zielfunktionswert (objective_value(PPP)) sowie die Werte der drei Variablen (value.(x)).

Wie schon bei den voranstehenden Beispielen, so zeigt sich auch hier, dass mit Hilfe von Julia eine Abbildung von Entscheidungsmodellen möglich ist, die sich sehr eng an die mathematischen Formulierungen anlegt: Das vorab erstellte A-Modell kann direkt in Julia (mit JuMP) implementiert werden. Weiterhin liegt wieder eine Trennung von Daten und Modell vor: Andere Datensätze für die hier modellierte Produktionsprogrammplanung haben nur Auswirkungen auf die Blöcke In [2] und In [3], nicht aber auf die eigentliche Modellierung in Block In [4]. Somit können problemlos auch sehr viel größere Beispiele berechnet werden.

4 Fazit

Ausgangspunkt der voranstehenden Überlegungen waren die Auswirkungen, die die digitale Transformation auf das Aufgabengebiet des Controllings hat und haben wird. Damit der daraus resultierende Wandel erfolgreich verlaufen kann, kommt neben den Technologien auch dem Wollen und Können der betroffenen Personen eine entscheidende Rolle zu. Aus Sicht des Könnens haben dabei Kenntnisse in der Programmierung sowohl im direkten Umgang mit den Technologien als auch in der Kommunikation mit anderen Stellen (IT, Data Science, …) eine große Bedeutung. Eine Möglichkeit, Kenntnisse in der Programmierung zu vermitteln, besteht in einem integrierten Ansatz. Dieser kombiniert Elemente der Programmierung mit fachspezifischen Methoden des Controllings. Anhand dreier Beispiele aus den Gebieten Investitions-, Kosten- und Entscheidungsrechnung wurde beispielhaft mit der Programmiersprache Julia gezeigt, wie eine solche Integration erfolgen kann. Dabei haben sich folgende Erkenntnisse ergeben:

- Voraussetzung eines integrierten Ansatzes ist eine strikte mathematische Formalisierung der jeweiligen Methoden.
- Aufbauend auf dieser Formalisierung können die einzelnen Methoden einfach in Julia umgesetzt werden. Dieses gelingt u.a. deshalb, weil die Sprachstruktur von Julia eng an mathematische Formulierungen angelehnt ist. Da ferner eine Vektorisierung zwar möglich, aber nicht notwendig ist, können insbesondere auf Summen basierende Berechnungen oder Allquantoren einfach implementiert werden.
- Bei der Umsetzung in Julia werden nur wenige Sprachelemente benötigt. Somit ist ein einführender Kurs zur Programmierung – zumindest auf dem hier demonstrierten Niveau – nicht notwendig.
- Bei der Programmierung kommt es – oft im Gegensatz zur Umsetzung in einer Tabellenkalkulation – zu einer strikten Trennung von Daten und Methoden. Damit können die entwickelten Methoden ohne weiteren Aufwand auf beliebig skalierte Probleme angewendet werden. Lediglich die Daten des jeweiligen Problems müssen neu erfasst bzw. eingelesen werden.
- Neben der eigenständigen Umsetzung von Methoden ermöglicht die Programmierung auch den Zugriff auf eine Vielzahl von schon implementierten Methoden z.B. aus dem Bereich der Optimierung, der Statistik und des Data Science. Dies zeigte sich im Beispiel zur Entscheidungsrechnung, hier wurden die Modellierungssprache JuMP und der Solver Clp eingesetzt. Somit stehen dem Controlling bei Einsatz der Programmierung viele moderne Verfahren und Methoden zur Verfügung, denen im Rahmen der digitalen Transformation große Bedeutung zukommt.
- Weiterhin bietet die Programmierung die Möglichkeit, die entwickelten bzw. vorhandenen Methoden in umfangreiche Berechnungen z.B. im Rahmen einer Risikosimulation oder der robusten Optimierung einzubinden und auszuwerten. Damit erweitern sich die Möglichkeiten der numerischen Analyse, die dem Controlling zur Verfügung stehen, erheblich.

Eine integrierte Vermittlung von fachspezifischen Methoden und Programmierung bietet somit vielfältige Möglichkeiten in der Ausbildung des Controllings. Formale Methoden und deren Umsetzung können eng miteinander verknüpft werden. Schlussendlich kann die Umsetzung in der Programmierung auch dazu beitragen, dass Verständnis des formalen Grundgerüstes der jeweiligen Methode zu vertiefen und damit zu einer Verstärkung des Lernerfolges des integrierten Vorgehens beitragen. Somit kann der integrierte Ansatz eine gute Basis für die Erweiterung des Könnens im Controlling bilden.

Literaturverzeichnis

Aguirre, A, Danielsson, J (2020): Which programming language is best for economic research: Julia, Matlab, Python or R?. https://voxeu.org/print/66134. Abgerufen am 30.04.2021.

Bezanson, J, Edelman, A, Karpinski, S, Shah, V (2017): Julia: A Fresh Approach to Numerical Computing. SIAM Review, 59(1):65-98.

Busse von Colbe, W, Witte, F (2018): Investitionstheorie und Investitionsrechnung, 5. Auflage. Springer, Berlin, Heidelberg.

Ewert, R, Wagenhofer, A (2014): Interne Unternehmensrechnung, 8. Auflage. Springer, Berlin, Heidelberg.

Frey, CB, Osborne, MA (2017): The future of employment: How susceptible are jobs to computerisation? Technological Forecasting and Social Change, 114:254-280.

Götze, U (2010): Kostenrechnung und Kostenmanagement, 5. Auflage. Springer, Berlin, Heidelberg.

Götze, U (2014): Investitionsrechnung, Modelle und Analysen zur Beurteilung von Investitionsvorhaben, 7. Auflage. Springer Gabler, Berlin, Heidelberg.

Heupel, T, Reinhardt, M (2019): Das Controlling-Bild der Zukunft: Welche Chancen und Risiken ergeben sich im Spannungsverhältnis zwischen IT und Controlling für den Controller der Zukunft?. In: Kümpel, T, Schlenkrich, K, Heupel, T (Hrsg), *Controlling & Innovation 2019, Digitalisierung* (S. 111-134). Springer Gabler, Wiesbaden.

Langmann, C (2019): Digitalisierung im Controlling. Springer Gabler, Wiesbaden.

Lausberg, I, Hoffmann, D (2019): Robotic Process Automation, Predictive Analytics und Künstliche Intelligenz – Wo liegen die Anwendungsbereiche im Controlling? Ergebnisse einer empirischen Studie. In: Nadig, L (Hrsg), *CARF Luzern 2019 – Controlling. Accounting. Risiko. Finanzen* (S. 49-63). Verlag IFZ – Hochschule Luzern.

Lauwens, B, Downey AB (2019); Think Julia, How to Think Like a Computer Scientist. O'Reilly, Sebastopol.

Mödritscher, G, Wall, F, (2019): Controlling und Digitalisierung – Änderungen im Kompetenzprofil. In: Feldbauer-Durstmüller, B, Mayr, S (Hrsg), *Controlling – Aktuelle Entwicklungen und Herausforderungen, Digitalisierung, Nachhaltigkeit und Spezialaspekte* (S. 65-81). Springer Gabler, Wiesbaden.

Müller, D (2019): Investitionsrechnung und Investitionscontrolling, 2. Auflage. Springer Gabler, Berlin.

Nadkarni, S, Prügl, R (2021): Digital transformation: a review, synthesis and opportunities for future research. Management Review Quarterly, 71(2):233-341.

Nelles, S (2019): Excel im Controlling, Das umfassende Handbuch. Rheinwerk, Bonn.

Oehler, K; Schwabe, R. (2020): Excel im Controlling für Dummies. Willey, Weinheim.

Ozgur, C, Jha, S, Myer-Tyson, E, Booth, D (2018). The Usage of R Programming in Finance and Banking Research. Journal of Accounting and Finance, 18(3):61-69.

Plinke, W, Rese, M, Utzig, BP (2015): Industrielle Kostenrechnung, Eine Einführung, 8. Auflage. Springer Vieweg, Berlin, Heidelberg.

Richins, G, Stapleton, A, Stratopoulos, TC, Wong, C (2017). Big Data Analytics: Opportunity or Threat for the Accounting Profession? Journal of Information Systems, 31(3):63-79.

Sauer, S (2019): Moderne Datenanalyse mit R, Daten einlesen, aufbereiten, visualisieren, modellieren und kommunizieren. Springer Gabler, Wiesbaden.

Schäffer, U, Brückner, L (2019): Rollenspezifische Kompetenzprofile für das Controlling der Zukunft. Controlling & Management Review 63(7):14-30.

Schäffer, U, Weber, J (2016): Die Digitalisierung wird das Controlling radikal verändern. Controlling & Management Review 60(6):8-17.

Schels, I, Seidel, UM (2020): Controlling mit Excel, Professionelle Lösungen für Controlling, Projekt- und Personalmanagement. Hanser, München.

Schoute, M (2019): Teaching Python to Management Accounting Students: An Illustration Using Support Department Cost-Allocation Methods. The Accounting Educators' Journal, 29:137-161.

Schulte, A, Bülchmann, O (2016): Wie Big Data die Rolle des Controllers verändert. Controlling & Management Review, Sonderheft (1):54-60.

Seufert, A, Oehler, K (2016): Controlling und Big Data: Anforderungen an die Methodenkompetenz. Controlling & Management Review, Sonderheft (1):74-81.

Togo, DF (2012): Support department cost allocations with a matrix-based reciprocal approach. Advances in Accounting Education: Teaching and Curriculum Innovations, 13:277-296.

Wolf, T, Heidlmayer, M (2019): Die Auswirkungen der Digitalisierung auf die Rolle des Controllers. In: Feldbauer-Durstmüller, B, Mayr, S (Hrsg), *Controlling – Aktuelle Entwicklungen und Herausforderungen, Digitalisierung, Nachhaltigkeit und Spezialaspekte* (S. 21-48). Springe Gabler, Wiesbaden.