Inhaltsverzeichnis

1 Simplex-Algorithmus

Es existieren zahlreiche Probleme, welche das Ziel haben, die optimalen Parameter eines Systems zu finden. In diesem Zusammenhang bedeutet »optimal«, dass eine Zielfunktion minimiert oder maximiert wird. Entsprechende »Optimierungsprobleme« treten häufig in verschiedenen naturwissenschaftlichen Disziplinen auf. Auch die Wirtschaftswissenschaften bieten diverse Beispiele für Optimierungsaufgaben. Gewöhnlich lassen sich diese Entscheidungsprobleme mathematisch formulieren. Oftmals können die Optimierungsprobleme analytisch nicht gelöst werden. Somit müssen häufig numerische Verfahren zur Bestimmung der optimalen Parameter eingesetzt werden. [5, 1]
Zu den einfachsten mathematischen Optimierungsproblemen gehören die linearen Optimierungsmodelle. Ein wirkungsvolles numerisches Verfahren zur Lösung dieser linearen Optimierungsprobleme ist der Simplex-Algorithmus. Der Simplex-Algorithmus wurde bereits 1947 von George Dantzig veröffentlicht. Gewiss gab es seither bedeutende Fortschritte im Bereich der Programmierung und Rechenleistung. Somit existieren gegenwärtig neue Methoden, welche auch komplexere Optimierungsaufgaben lösen. Ungeachtet dessen bleibt der Simplex-Algorithmus auch in unserer Zeit einer der am stärksten verbreiteten Algorithmen zur Lösung linearer Optimierungsprobleme. [5, 1]
Um in sich eine geschlossene Arbeit zu präsentieren, werden zu Beginn die wesentlichen Begrifflichkeiten der linearen Programmierung und des Simplex-Verfahrens dargestellt. Anschließend werden verschiedene Variationsmöglichkeiten des Simplex-Algorithmus aufgeführt. Die verschiedenen Varianten des Simplex-Verfahrens werden jeweils für Maximierungsprobleme dargestellt. Hier beginnen wir mit dem primalen Simplexverfahren, welcher von einer bereits bekannten zulässigen Basislösung ausgeht. Daran anknüpfend beschreiben wir den dualen Simplex-Algorithmus. Dieser Algorithmus beschreibt die Vorgehensweise zur Bestimmung einer ersten zulässigen Basislösung. Der Ablauf dieser Algorithmen wird mithilfe von Pseudocode in einer vereinfachten Form dargestellt. Der Pseudocode dient zudem auch als Referenz für den entsprechend implementierten Programmiercode. Der Programmcode für die Anwendung zur Lösung von linearen Optimierungsproblemen mithilfe des Simplex-Algorithmus ist in C++ geschrieben. Standard C++ kennt keine Elemente für grafische Benutzeroberflächen (graphical user interfaces, GUI), folglich wurde die GUI mit Qt [6] entwickelt. [4, 5]

1.1 Grundlagen lineare Optimierung

Im Folgenden werden die wesentlichen Begrifflichkeiten der linearen Programmierung dargestellt. Die linearen Optimierungsprobleme werden in der Sprache der linearen Algebra repräsentiert. Ziel der linearen Optimierungsmodelle ist es, eine lineare Zielfunktion f

$$f : \mathbb{R}^n \to \mathbb{R} \tag{1.1}$$

$$f(\mathbf{x}) = \sum_{j=1}^{n} c_j \, x_j = \mathbf{c}^\mathsf{T} \, \mathbf{x} \tag{1.2}$$

unter Berücksichtigung von linearen Nebenbedingungen der Form

$$a_{i1} \, x_1 + \ldots + a_{in} \, x_n \leq b_i \qquad \text{für i } = 1, \ldots, \mathrm{m}_1 \tag{1.3a}$$

$$a_{i1} \, x_1 + \ldots + a_{in} \, x_n \geq b_i \qquad \text{für i } = \mathrm{m}_1, \ldots, \mathrm{m}_2 \tag{1.3b}$$

$$a_{i1} \, x_1 + \ldots + a_{in} \, x_n = b_i \qquad \text{für i } = \mathrm{m}_2, \ldots, \mathrm{m} \tag{1.3c}$$

und unter Beachtung der Nichtnegativitätsbedingungen

$$x_j \geq 0 \qquad \text{für j } = 1, \ldots, \text{n} \tag{1.4}$$

zu maximieren bzw. zu minimieren. [2] Im weiteren Verlauf betrachten wir das Vorgehen für Maximierungsprobleme.

> Eine zu maximierende Zielfunktion $f(\mathbf{x})$ kann durch eine zu minimierende Zielfunktion $-f(\mathbf{x})$ substituiert werden. Die Maximierung von $\mathbf{c}^\intercal \mathbf{x}$ entspricht somit der Minimierung von $(-\mathbf{c}^\intercal \mathbf{x})$. [1, 2]

Der skalare Wert $f(\mathbf{x})$ der linearen Zielfunktion f für eine Lösung $\mathbf{x}$ wird mittels der Zielfunktionskoeffizienten $\mathbf{c}$ definiert. (siehe Gleichung 1.2) In der numerischen Berechnung der linearen Optimierungsprobleme wird die Effektivität von Lösungsalgorithmen für lineare Gleichungssysteme angewendet. Daher werden die allgemeinen Nebenbedingungen (siehe Gleichungen 1.3) so umgeformt, dass das lineare Optimierungsproblem durch ein lineares Gleichungssystem abgebildet wird. In einem ersten Zwischenschritt transformieren wir das System mit den allgemeinen Nebenbedingungen (siehe Gleichungen 1.3) in die kanonische Form, welche sich in Matrixweise folgendermaßen formulieren lässt: [1]

$$\max\{\mathbf{c}^\intercal \mathbf{x} \mid \mathbf{A}\,\mathbf{x} \leq \mathbf{b},\, \mathbf{x} \geq \mathbf{0}\} \tag{1.5}$$

Zu diesem Zweck werden die $\geq$ Nebenbedingungen (siehe Gleichung 1.3b) mittels Multiplikation beider Operanden der Ungleichungsnebenbedingungen mit -1 in $\leq$ Nebenbedingungen transformiert. Entsprechend gilt die folgende Äquivalenz [2, 5]

$$a_{i1}\,x_1 + \ldots + a_{in}\,x_n \geq b_i \Leftrightarrow -a_{i1}\,x_1 - \ldots - a_{in}\,x_n \leq -b_i \tag{1.6}$$

für i $\in \{1, \ldots, \text{m}\}$.
Eine Gleichungsnebenbedingung (siehe Gleichung 1.3c) wird durch zwei $\leq$ Nebenbedingung mit jeweils unterschiedlichen Vorzeichen ersetzt: [2]

$$a_{i1} + \ldots + a_{in} = b_i \quad \longrightarrow \quad a_{i1} + \ldots + a_{in} \leq b_i \quad \text{und} \quad -a_{i1} - \ldots - a_{in} \leq -b_i \tag{1.7}$$

Bei diesem Fall erweitert sich die Zeilenanzahl m des Systems der Nebenbedingungen, um die Anzahl der zu transformierenden Gleichungsrestriktionen. Somit lassen sich sämtliche lineare Optimierungsprobleme in der folgenden kanonischen Form darstellen: [2]

$$\text{Maximiere } f(\mathbf{x}) = \sum_{j=1}^{n} c_j\,x_j \tag{1.8a}$$

$$\sum_{j=1}^{n} a_{ij}\,x_j \leq b_i \qquad \text{für i } = 1, \ldots, \text{m} \tag{1.8b}$$

$$x_j \geq 0 \qquad \text{für j } = 1, \ldots, \text{n} \tag{1.8c}$$

Im nächsten Schritt beschreiben wir die Lösungen des Ungleichungssystems in kanonischer Form (siehe Gleichung 1.8) als Lösungen eines Gleichungssystems in Normalform. Für die Ungleichungsnebenbedingungen (siehe Gleichung 1.8b) führen wir nicht negative Schlupfvariablen $x_{n+1}, \ldots, x_{n+m}$ ein, die in der Zielfunktion f mit 0 bewertet werden. Die Schlupfvariablen schließen die Lücken in den Ungleichungen, d. h. [1, 2]

$$x_{n+i} = b_i - a_{i1}\,x_1 - \ldots - a_{in}\,x_n \qquad \text{für i } = 1, \ldots, \text{m} \tag{1.9}$$

Um eine Unterscheidung der Variablen des Vektors $\mathbf{x}$ aus $\mathbb{R}^{n+m}$ zu gewährleisten, bezeichnet man die ursprünglichen Variablen $x_1, \ldots, x_n$ des linearen Optimierungsproblems als Strukturvariablen. Somit lässt sich die lineare Optimierungsaufgabe in kanonischer Form im $\mathbb{R}^n$ in der Normalform im $\mathbb{R}^{n+m}$ darstellen: [1, 2]

$$\text{Maximiere } f(\mathbf{x}) = \sum_{j=1}^{n} c_j \, x_j + \sum_{j=n+1}^{n+m} 0 \cdot x_j \tag{1.10a}$$

$$\begin{array}{ccccccccccc} a_{11}\,x_1 & + & \ldots & + & a_{1n}\,x_n & + & x_{n+1} & & & = & b_1 \\ a_{21}\,x_1 & + & \ldots & + & a_{2n}\,x_n & + & x_{n+2} & & & = & b_2 \\ \vdots & & \ddots & & \vdots & & & \ddots & & & \vdots \\ a_{m1}\,x_1 & + & \ldots & + & a_{mn}\,x_n & & & + & x_{n+m} & = & b_m \end{array} \tag{1.10b}$$

Mit $x_j \geq 0$ für j $= 1, \ldots,$ n+m.

Wie in Gleichung 1.10b ersichtlich, ist der Teil der ergänzten Koeffizientenmatrix $\mathbf{A}^{m\times(n+m)}_{Normal}$ eine Einheitsmatrix $\mathbf{E}^{m\times m}$. Demnach setzt sich die erweiterte Koeffizientenmatrix $\mathbf{A}^{m\times(n+m)}_{Normal}$ aus der Koeffizientenmatrix in kanonischer Form $\mathbf{A}^{m\times n}_{Kanonisch}$ und der zu den Schlupfvariablen gehörenden Einheitsmatrix $\mathbf{E}^{m\times m}$ zusammen: [1]

$$\mathbf{A}^{m\times(n+m)}_{Normal} = \left[\mathbf{A}^{m\times n}_{Kanonisch} \mid \mathbf{E}^{m\times m}\right] \tag{1.11}$$

Folglich lässt sich die lineare Optimierungsaufgabe in Kurzform mit Matrixschreibweise folgendermaßen beschreiben: [1]

$$\max\{\mathbf{c}^\intercal \, \mathbf{x} \mid \mathbf{A}\,\mathbf{x} = \mathbf{b},\, \mathbf{x} \geq \mathbf{0}\} \tag{1.12}$$

Im Folgenden gehen wir davon aus, dass die linearen Optimierungsaufgaben bereits in der Normalform gegeben sind. Entsprechend ist vorausgesetzt, dass die Koeffizientenmatrix $\mathbf{A}^{m\times(n+m)}$ den Rang m besitzt. Somit sind alle Gleichungen linear unabhängig. Wir definieren eine Untermatrix $\mathbf{B}^{m\times m}$, die aus den m linear unabhängigen Spalten der Koeffizientenmatrix $\mathbf{A}^{m\times(n+m)}$ gebildet wird. Die Spalten der Matrix $\mathbf{B}^{m\times m}$ bezeichnet man als Basisvektoren und die zugehörigen x_j nennt man Basisvariablen. Die restlichen nicht linear unabhängigen Spalten der Koeffizientenmatrix $\mathbf{A}^{m\times(n+m)}$ heißen Nichtbasisvektoren und die entsprechenden x_j nennt man Nichtbasisvariablen. Im Allgemeinen wird $\mathcal{B}$ als die Menge der Indizes der Basisvariablen definiert. Analog stellt $\mathcal{N}$ die Menge der Indizes der Nichtbasisvariablen dar. Mithilfe der Notationen von $\mathcal{B}$ und $\mathcal{N}$ lässt sich die entsprechende Untermatrix bzw. Basismatrix $\mathbf{B}^{m\times m}$ alternativ auch folgendermaßen mathematisch formulieren: [1, 2, 5]

$$\mathbf{A}_{\mathcal{B}} = \left(\mathbf{A}_{\mathcal{B}(1)}, \ldots, \mathbf{A}_{\mathcal{B}(m)}\right) \tag{1.13}$$

Existiert eine Basis $\mathcal{B}$ mit [1]

$$\mathbf{A}_{\mathcal{B}}\,\mathbf{x}_{\mathcal{B}} = \mathbf{b}, \quad \mathbf{x}_{\mathcal{N}} = \mathbf{0} \tag{1.14}$$

so bezeichnet man die Lösung $\mathbf{x} \in \mathbb{R}^{n+m}$ als Basislösung der lineren Optimierungsaufgabe. Erfüllt eine Basislösung $\mathbf{x}$ die Nichtnegativitätsbedingungen $\mathbf{x}_{\mathcal{B}} \geq \mathbf{0}$, so nennt man sie eine zulässige Basislösung der linearen Optimierungsaufgabe. Und die Basis $\mathcal{B}$ wird entsprechend als zulässige Basis bezeichnet. $\mathbf{X}$ ist die Menge aller zulässigen Basislösungen $\mathbf{x}$. Die Menge $\mathbf{X}$ definiert ein konvexes Polyeder mit endlich vielen Eckpunkten. Die optimale Basislösung $\mathbf{x}^\star$ einer linearen Funktion f, welche auf einem konvexen Polyeder $\mathbf{X}$ bestimmt ist, befindet sich

in mindestens einem Eckpunkt des Polyeders. [1, 2] Im Folgenden werden die zuvor definierten Begrifflichkeiten mithilfe eines einfachen konkreten Beispiels im $\mathbb{R}^2$ geometrisch betrachtet.

$$\text{Maximiere } f(x_1, x_2) = 3\,x_1 + 2\,x_2 \tag{1.15a}$$

$$\begin{aligned} x_1 &+ 4\,x_2 \leq 12 \\ -\,x_1 &+ 4\,x_2 \geq 4 \quad \text{und } x_1, x_2 \geq 0 \end{aligned} \tag{1.15b}$$

Die Optimierungsaufgabe (siehe Gleichung 1.15) im $\mathbb{R}^2$ geht durch Transformation der allgemeinen Nebenbedingungen (siehe Gleichung 1.15b) und durch die Einführung von Schlupfvariablen in eine äquivalente Darstellung in Normalform im $\mathbb{R}^4$ über:

$$\text{Maximiere } f(x_1, x_2, x_3, x_4) = 3\,x_1 + 2\,x_2 + 0\,x_3 + 0\,x_4 \tag{1.16a}$$

$$\begin{aligned} x_1 &+ 4\,x_2 + x_3 = 12 \\ x_1 &- 4\,x_2 + x_4 = -4 \quad \text{und } x_1, x_2, x_3, x_4 \geq 0 \end{aligned} \tag{1.16b}$$

Grafisch betrachtet wird die zulässige Lösungsmenge **X** durch die beiden Nebenbedingungsgeraden und die Nichtnegativitätsbedingungen bestimmt. (siehe Abbildung 1.1)

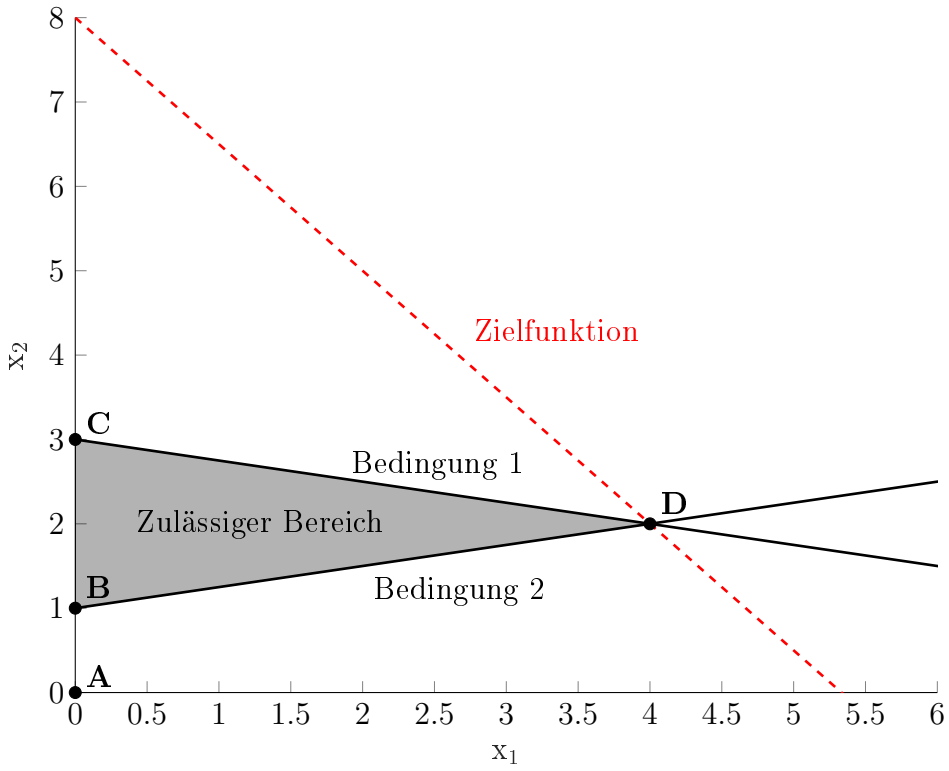


Abbildung 1.1: Graphische Darstellung der linearen Optimierungsaufgabe

In der Abbildung 1.1 sind einige Basislösungen der linearen Optimierungsaufgabe abgebildet, die sich aus den Schnittpunkten von Nebenbedingungsgeraden und/oder den Nichnegativitätsbedingungen (Koordinatenachsen) ergeben. An den drei Eckpunkten **B**, **C** und **D** befinden sich zulässige Basislösungen mit jeweils zwei Basisvariablen und zwei Nichtbasisvariablen. An dem Eckpunkt **A** ist eine unzulässige Basislösung, da sich dieser Punkt nicht auf dem konvexen Polyeder **X** der zulässigen Basislösungen befindet. [2]
Der Simplex-Algorithmus untersucht der Rand des konvexen Polyeders nach einer optimalen Basislösung. In dem Eckpunkt der optimalen Basislösung nimmt der Wert der Zielfunktion seine maximale Größe an (Maximierungsaufgabe). Im Folgenden beschreiben wir den primalen und dualen Simplex-Algorithmus für Maximierungsprobleme. [2]

1.2 Simplexverfahren bei bekannter zulässiger Lösung

Im Folgenden schildern wir das Vorgehen von zwei verschiedenen Modifikationen des Simplex-Algorithmus für Maximierungsprobleme. Zunächst wird die Implementation des primalen Simplex-Algorithmus, welcher von einer bekannten zulässigen Basislösung ausgeht, beschrieben. Zur Veranschaulichung des Verfahrens erklären wir das praktische Vorgehen am folgenden in kanonischer Form vorgegebenen Beispiel: [2]

$$\text{Maximiere } f(x_1, x_2) = 40\,x_1 + 30\,x_2 \tag{1.17a}$$

$$\begin{array}{rcrcll} x_1 & + & x_2 & \leq & 8 & \\ 2\,x_1 & + & x_2 & \leq & 12 & \\ 2\,x_1 & + & 3\,x_2 & \leq & 18 & \text{und } x_1, x_2 \geq 0 \end{array} \tag{1.17b}$$

Die lineare Optimierungsaufgabe stammt aus dem Lehrbuch »Übungen und Fallbeispiele zum Operations Research« (Aufgabe 2.10) [3]
Um das Simplex-Verfahren anwenden zu können, wird die Aufgabenstellung in ihre Normalform transformiert. Es liegen drei Ungleichungsnebenbedingungen vor, demzufolge werden drei Schlupfvariablen eingeführt. [5]

$$\text{Maximiere } f(x_1, x_2, x_3, x_4, x_5) = 40\,x_1 + 30\,x_2 + 0\,x_3 + 0\,x_4 + 0\,x_5 \tag{1.18a}$$

$$\begin{array}{rcrcrcrcll} x_1 & + & x_2 & + & x_3 & & & = & 8 & \\ 2\,x_1 & + & x_2 & & & + & x_4 & = & 12 & \\ 2\,x_1 & + & 3\,x_2 & & & & + \; x_5 & = & 18 & \text{und } x_1, x_2, x_3, x_4, x_5 \geq 0 \end{array} \tag{1.18b}$$

Zur weiteren Veranschaulichung des Simplex-Verfahrens stellt man die Daten in einem Tableau zusammen.

		NBV			BV			
		x_1	$\dots$	x_n	x_{n+1}	$\dots$	x_{n+m}	b_i
BV	x_{n+1}	a_{11}	$\dots$	a_{1n}	1	$\dots$	0	b_1
	$\vdots$	$\vdots$	$\ddots$	$\vdots$		$\ddots$		$\vdots$
	x_{n+m}	a_{m1}	$\dots$	a_{nn}	0	$\dots$	1	b_m
		$-c_1$	$\dots$	$-c_n$	0	$\dots$	0	Zfw.

Tabelle 1.1: Vorlage Tableau

In dem Anfangstableau (siehe Tabelle 1.2) stimmen die Indizes der Nichtbasisvariablen (NBV) und Strukturvariablen und die Indizes der Basisvariablen (BV) und Schlupfvariablen überein. Die Zielfunktionskoeffizienten $(c_1, \dots, c_n)$ für die Nichtbasisvariablen werden in der Ergebniszeile mit negativen Vorzeichen eingetragen. Somit kann die Lösung der linearen Optimierungsaufgabe immer verbessert werden, solange mindestens eine Nichtbasisvariable mit negativer Eintragung in der Ergebniszeile vorliegt. [2] Das hinreichende Optimalitätskriterium in einer Ecke $\mathbf{x}$ wird somit folgendermaßen definiert: [1]

$$\mathbf{c}_{\mathcal{N}} \geq \mathbf{0} \tag{1.19}$$

Im Folgenden wird das Anfangstableau für das Beispiel der linearen Maximierungsaufgabe (siehe Gleichung 1.18) dargestellt. In unserem Ausgangstableau sind alle $b_i \geq 0$, somit sind die Nichtnegativitätsbedingungen (siehe Gleichung 1.4) erfüllt und wir erhalten über die Schlupfvariablen eine zulässige Basislösung ($\mathbf{x}^{(0)} = (0, 0, 8, 12, 18)$ siehe Punkt **A** in Abbildung 1.2).

Wie man im Falle einer unzulässigen Basislösung eine zulässige Basislösung ermittelt, falls z. B. die Nichtnegativitätsbedingungen nicht erfüllt sind, wird in Abschnitt 1.3 beschrieben. Die erste Spalte des Tableaus notiert die Indizes der aktuellen Basisvariablen.

> Im Folgenden wird der Ablauf des primalen Simplex-Verfahren zusätzlich mithilfe von Pseudocode beschrieben. Der Pseudocode dient zudem auch als Referenz für den in C++ implementierten Programmiercode. Um eine gewisse Übersichtlichkeit sicherzustellen, werden die entsprechenden Verweise auf die Pseudocode- und C++-Funktionen differenziert dargestellt. Die Verweise auf den Pseudocode werden in blauer Farbe und die auf den C++ Code mit der Farbe Cyan umrandet. Der Verweis auf den C++ Code befindet sich jeweils in der entsprechenden Pseudocode Implementation.

Der Ablauf des primalen Simplex-Verfahrens wird in dem Algorithmus 1 wiedergegeben.

	x_1	x_2	x_3	x_4	x_5	b	
x_3	1	1	1	0	0	8	$\frac{8}{1}$
x_4	[2]	1	0	1	0	12	$\frac{12}{2}$
x_5	2	3	0	0	1	18	$\frac{18}{2}$
F	-40	-30	0	0	0	0	

Tabelle 1.2: Anfangstableau mit zulässiger Basislösung

Vorab überprüfen wir, ob die zulässige Basislösung $\mathbf{x}^{(0)}$ bereits eine optimale Basislösung repräsentiert (siehe Gleichung 1.19 und Funktion BasisloesungOptimal). In der Ergebniszeile liegen Zielfunktionskoeffizienten $\mathbf{c}_{\mathcal{N}}$ kleiner Null vor. Somit erfüllt die zulässige Basislösung $\mathbf{x}^{(0)}$ nicht das hinreichende Optimalitätskriterium.
Im nächsten Schritt versuchen wir, einen zulässigen Basiswechsel vorzunehmen, wobei der Wert der Zielfunktion nicht fallen darf. Die Zielfunktionskoeffizienten der Nichtbasisvariablen $\mathbf{c}_{\mathcal{N}}$ bestimmen, um welchen Wert sich die Zielfunktion ändert, wenn die entsprechende Nichtbasisvariable erhöht wird. Folglich bestimmen wir die Pivotspalte t so, dass der Zielfunktionswert einen größtmöglichen Zuwachs erfährt (siehe Funktion PivotspaltePrimal). [1] Der kleinste Wert in der Ergebniszeile ist -40, demnach erhalten wir nach unseren Überlegungen die erste Spalte als Pivotspalte (siehe Tabelle 1.2). Sind in der betrachteten Pivotspalte t alle Koeffizienten $a_{it} \leq 0, \forall\, i \in \{1, \ldots, \mathrm{m}\}$, so ist eine unbeschränkte Steigerung von x_t möglich und somit würde sich der Wert der Zielfunktion uneingeschränkt vergrößern lassen (siehe Funktion PivotspaltePrimalZulaessig). Unter diesen Umständen würde das Problem keine optimale Basislösung besitzen und das Verfahren müsste abgebrochen werden (siehe Algorithmus 1). [2] In dem Anfangstableau (siehe Tabelle 1.2) sind alle $a_{i1} > 0$, somit fahren wir mit der Wahl der Pivotzeile s fort (siehe Funktion PivotzeilePrimal).
Zur Ermittlung der Pivotzeile werden die Quotienten $\frac{b_i}{a_{i1}}, \forall\, i \in \{1, 2, 3\}$, berechnet. Um nach dem Basiswechsel eine zulässige Basislösung zu erhalten, wird der kleinste Quotient bestimmt, dieser steht in der Tabelle 1.2 in der zweiten Zeile. [5]
Folglich erhalten wir als Pivotelement den Koeffizienten $a_{21} = 2$ (siehe Tabelle 1.2). Mithilfe des Pivotelements a_{21} bestimmen wir den Index der Nichtbasisvariablen x_1 und Basisvariablen x_4, die beim Basiswechsel bzw. Pivotschritt ausgetauscht werden (siehe Tabelle 1.2). In dem Pivotschritt ersetzen wir die bisherige Basisvariable x_4 mit der bisherigen Nichtbasisvariablen x_1. Durch lineare Transformation des Nebenbedingungssystems mit der neuen Basisvariablen x_1 erhalten wir mittels der Prozedur Pivotschritt einen Einheitsvektor mit $a_{21} = 1$ (siehe Tabelle 1.3). [2] Dementsprechend erhalten wir mit dem neuen Basisvektor $\mathbf{A}_{\mathcal{B}(1)}$ als neue Basismatrix $\mathbf{A}_{\mathcal{B}} = \big(\mathbf{A}_{\mathcal{B}(1)}, \mathbf{A}_{\mathcal{B}(3)}, \mathbf{A}_{\mathcal{B}(5)}\big)$. Analog dazu ergibt sich die mit dem neuen Nicht-Basisvektor $\mathbf{A}_{\mathcal{N}(4)}$ als neue Nicht-Basismatrix $\mathbf{A}_{\mathcal{N}} = \big(\mathbf{A}_{\mathcal{N}(2)}, \mathbf{A}_{\mathcal{N}(4)}\big)$.

	x_1	x_2	x_3	x_4	x_5	b	
x_3	0	0.5	1	-0.5	0	2	$\frac{2}{0.5}$
x_1	1	0.5	0	0.5	0	6	$\frac{6}{0.5}$
x_5	0	[2]	0	-1	1	6	$\frac{6}{2}$
F	0	-10	0	20	0	240	

Tabelle 1.3: Tableau mit zulässiger Basislösung

	x_1	x_2	x_3	x_4	x_5	b
x_3	0	0	1	-0.25	-0.25	0.5
x_1	1	0	0	0.75	-0.25	4.5
x_2	0	1	0	-0.5	0.5	3
F	0	0	0	15	5	270

Tabelle 1.4: Tableau mit optimaler Basislösung

Der neue Basispunkt ist somit $\mathbf{x}^{(1)} = (6, 0, 2, 0, 6)$ (siehe Punkt **B** in Abbildung 1.2) und der neue Zielfunktionswert ist 240. Die neue zulässige Basislösung erfüllt nicht das Kriterium für die Optimalität (siehe Gleichung 1.19). Demnach bestimmen wir erneut eine Pivotspalte, -zeile und das entsprechende Pivotelement. Der kleinste negative Wert in der Ergebniszeile ist -10 und der kleinste Quotient $\frac{b_i}{a_{i2}}, \forall i \in \{1, 2, 3\}$ befindet sich in der dritten Zeile. Infolgedessen erhalten wir das Pivotelement $a_{32} = 2$ (siehe Tabelle 1.3). Wir ersetzen die bisherige Basisvariable x_5 durch die bisherige Nichtbasisvariable x_2 und erhalten durch die lineare Transformation des Nebenbedingungssystems mit der neuen Basisvariable x_2 (siehe Funktion Pivotschritt) das Tableau in Tabelle 1.4. [2]
Die neue zulässige Basislösung ist $\mathbf{x}^{(2)} = (4.5, 3, 0.5, 0, 0)$ (siehe Punkt **C** in Abbildung 1.2) und der Wert der Zielfunktion ist 270. In der Ergebniszeile ist der Vektor der Zielfunktionskoeffizienten $\mathbf{c}_\mathcal{N}$ größer Null. Folglich ist das Optimalitätskriterium erfüllt und das Verfahren terminiert. Demzufolge ist die zulässige Basislösung $\mathbf{x}^{(2)}$ eine optimale Basislösung $\mathbf{x}^\star$. Der zugehörige optimale Wert der Zielfunktion $f(\mathbf{x}^\star)$ ist somit 270.

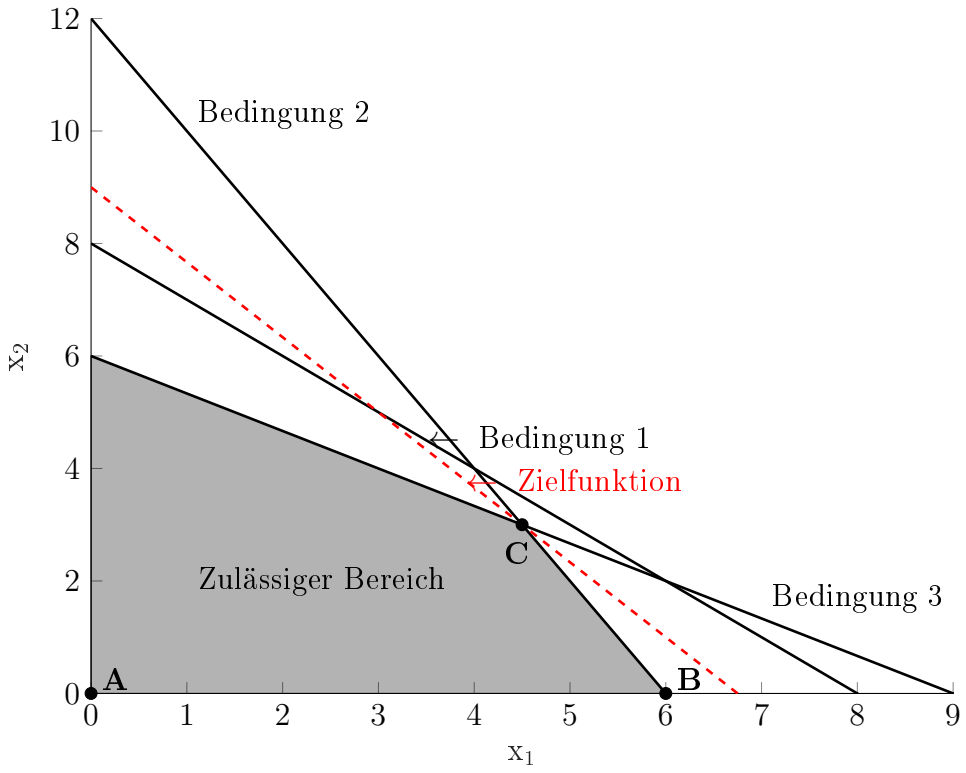


Abbildung 1.2: Graphische Darstellung der linearen Maximierungsaufgabe, primaler Simplex

Algorithmus 1 : Primaler Simplex

Eingabe : Lineare Optimierungsaufgabe $\max\{\mathbf{c}^\intercal\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$ mit $\mathbf{b} \geq \mathbf{0}$.
Ausgabe : Optimale Lösung

```
while !BasisloesungOptimal(c) do
    t = PivotspaltePrimal(c);
    if PivotspaltePrimalZulaessig(t, A) then
        s = PivotzeilePrimal(t, A, b);
    else
        return c^T x ist unbeschränkt;
    end
    Pivotschritt(s, t, A, b);
end
```

C++ Code: Listing 3.1

Funktion BasisloesungOptimal(**c**)

Daten : F-Zeile
Ergebnis : (bool) true, wenn eine optimale Basislösung vorhanden ist

```
for j = 1 to n+m do
    if c_j < 0 then return false;
end
return true;
```

C++ Code: Listing 3.2

Funktion PivotspaltePrimal(**c**)

Daten : F-Zeile
Ergebnis : t »Pivotspaltenindex«

Wähle Spalte t mit $c_t := \min\{c_j \mid j = 1, \ldots, \text{n+m}\}$;

```
return t
```

C++ Code: Listing 3.3

Funktion PivotspaltePrimalZulaessig(t, **A**)

Daten : t »Pivotzeile«, **A** »Koeffizientenmatrix«
Ergebnis : (bool) true, wenn die Pivotspalte zulässig ist

```
for i = 1 to m do
    if a_it > 0 then return true;
end
return false;
```

C++ Code: Listing 3.4

Funktion PivotzeilePrimal(t, **A**, **b**)

Daten : t »Pivotspalte«, **A** »Koeffizientenmatrix«, **b** »Kapazitätenvektor«
Ergebnis : s »Pivotzeilenindex«

Wähle Zeile s mit $\frac{b_s}{a_{st}} := \min\left\{\frac{b_i}{a_{it}} \,\middle|\, i = 1, \ldots, \text{m mit } a_{it} > 0\right\}$;

return s;

C++ Code: Listing 3.5

Prozedur Pivotschritt(s, t, **A**, **b**)

Daten : s »Pivotzeile«, t »Pivotspalte«, **A** »Koeffizientenmatrix«,
b »Kapazitätenvektor«
Ergebnis : neue Anordnung des Tableaus T

`/* Dividiere Pivotzeile durch das Pivotelement` a_{st} `*/`
for $j = 1$ **to** $n{+}m$ **do**
  $a_{sj} = \frac{a_{sj}}{a_{st}}$
end

`/* Multipliziere für alle Zeilen i (außer s) die neue Pivotzeile mit -`a_{it}
`und addiere sie zur jeweiligen Zeile i hinzu */`
for $i = 1$ **to** m **do**
  if $i \neq s$ **then**
    for $j = 1$ **to** $n{+}m$ **do**
      $a_{ij} = a_{ij} - a_{it} \cdot a_{sj}$
    end
    $b_i = b_i - a_{it} \cdot b_s$
  end
end

for $j = 1$ **to** $n{+}m$ **do**
  $c_j = c_j - c_t \cdot a_{sj}$
end

return T;

C++ Code: Listing 3.6

1.3 Bestimmung einer zulässigen Startbasis

In Abschnitt 1.2 haben wir zur Vereinfachung angenommen, dass sich aus dem Starttableau unmittelbar ein zulässiger Basispunkt ablesen lässt. Im Allgemeinen ist dieser Ausgangspunkt nicht gegeben. Um eine erste zulässige Basislösung zu bestimmen, schildern wir im Folgenden das Verfahren des dualen Simplex-Algorithmus. Diese Methode ist notwendig, wenn das lineare Optimierungsproblem nicht in seiner kanonischen Form (siehe Gleichungen 1.8) vorhanden ist und nur schwer in diese umformbar ist. [2] Zur Veranschaulichung des Verfahrens erklären wir das Vorgehen am folgenden Beispiel:

$$\text{Maximiere } f(x_1, x_2) = -x_1 - 2\,x_2 \tag{1.20a}$$

$$\begin{array}{rcrcll} x_1 & + & x_2 & \geq & 3 & \\ & & x_2 & \geq & 2 & \\ -x_1 & + & x_2 & \leq & 3 & \\ x_1 & - & x_2 & \leq & 3 & \text{und } x_1, x_2 \geq 0 \end{array} \tag{1.20b}$$

Die lineare Optimierungsaufgabe stammt aus dem Lehrbuch »Einführung in die Mathematische Optimierung« (Beispiel 3.4.1) [1]
Durch die Einbindung von Schlupfvariablen wird die lineare Optimierungsaufgabe so transformiert, dass die Koeffizientenmatrix **A** und der Zielfunktionskoeffizienten Vektor **c** den in der Gleichung 1.10 beschriebenen notwendigen Kriterien nachkommt, der Kapazitätenvektor **b** allerdings negative Werte enthält (siehe Gleichung 1.21). [2]
Der Ablauf des dualen Simplex-Verfahrens wird in dem Algorithmus 2 wiedergegeben.

$$\text{Maximiere } f(x_1, x_2, x_3, x_4, x_5, x_6) = -x_1 - 2\,x_2 + 0\,x_3 + 0\,x_4 + 0\,x_5 + 0\,x_6 \tag{1.21a}$$

$$\begin{array}{rcrcrcrcrcll} -x_1 & - & x_2 & + & x_3 & & & & & = & -3 & \\ & - & x_2 & & & + & x_4 & & & = & -2 & \\ -x_1 & + & x_2 & & & & & + & x_5 & = & 3 & \\ x_1 & - & x_2 & & & & & & + \; x_6 & = & 3 & \text{und } x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \end{array} \tag{1.21b}$$

Nachdem der Kapazitätenvektor **b** negative Elemente aufweist, sind die Nichnegativitätsbedingungen 1.4 nicht erfüllt und der Ausgangsbasispunkt $\mathbf{x}^{(0)} = (0, 0, -3, -2, 3, 3)$ ist somit nicht zulässig (siehe Funktion BasisloesungZulaessig). Die unzulässige Basislösung $\mathbf{x}^{(0)}$ ist in dem Punkt **A** der Darstellung 1.3 abgebildet. Es ist direkt erkennbar, dass sich die unzulässige Basislösung $\mathbf{x}^{(0)}$ nicht auf dem konvexen Polyeder **X** der zulässigen Basislösungen befindet.

	x_1	x_2	x_3	x_4	x_5	x_6	b
x_3	[-1]	-1	1	0	0	0	(-3)
x_4	0	-1	0	1	0	0	-2
x_5	-1	1	0	0	1	0	3
x_6	1	-1	0	0	0	1	3
F	(1)	2	0	0	0	0	0
	$\frac{1}{-1}$	$\frac{2}{-1}$	–	–	–	–	

Tabelle 1.5: Anfangstableau mit unzulässiger Basislösung

Der duale Simplex-Algorithmus beginnt mit der Wahl der Pivotzeile s (siehe Funktion PivotzeileDual). Wir bestimmen die Zeile s mit dem kleinsten negativen b_s als Pivotzeile. Der kleinste Wert im Kapazitätenvektor **b** ist -3, somit erhalten wir die erste Zeile als Pivotzeile (siehe Tabelle 1.5). Gibt es in der betrachteten Pivotzeile s keinen Koeffizienten $a_{sj} < 0$, $\forall\, i \in \{1, \ldots, \text{n+m}\}$,

so ist die Menge alle zulässigen Basislösungen $\mathbf{X}$ leer $\mathbf{X} = \varnothing$ und das Nebenbedingungssystem nicht widerspruchsfrei (siehe Funktion PivotzeileDualZulaessig). In diesem Fall würde das Problem keine zulässige Basislösung besitzen und das Verfahren müsste terminiert werden (siehe Algorithmus 2). [2] In dem Anfangstableau (siehe Tabelle 1.5) sind Koeffizienten a_{sj} kleiner Null vorhanden, somit fahren wir mit der Wahl der Pivotspalte fort (siehe Funktion PivotspalteDual).
Zur Bestimmung der Pivotspalte t werden die Quotienten $\frac{c_j}{a_{sj}}, \forall\, j \in \{1, \ldots, 6\}$ mit $a_{sj} < 0$, berechnet. Es wird die Spalte t mit dem größten Quotienten bestimmt, dieser befindet sich in der Tabelle 1.5 in der ersten Spalte. [2]
Wir erhalten somit den Koeffizienten $a_{11} = -1$ als Pivotelement (siehe Tabelle 1.5). Mithilfe des Pivotelements bestimmen wir den Index der Nichtbasisvariablen x_1 und der Basisvariablen x_3, die beim Basiswechsel bzw. Pivotschritt ausgetauscht werden. Das Vorgehen für den Pivotschritt (siehe Funktion Pivotschritt) ist äquivalent zu der Vorgehensweise des Pivotschritts beim primalen Simplex-Verfahren (siehe Abschnitt 1.2).

	x_1	x_2	x_3	x_4	x_5	x_6	b
x_1	1	1	-1	0	0	0	3
x_4	0	[-1]	0	1	0	0	(-2)
x_5	0	2	-1	0	1	0	6
x_6	0	-2	1	0	0	1	0
F	0	(1)	1	0	0	0	-3
	–	$\frac{1}{-2}$	–	–	–	–	

Tabelle 1.6: Tableau nach dem ersten Pivotschritt, unzulässige Basislösung

Der neue Basispunkt nach dem Basiswechsel ist $\mathbf{x}^{(1)} = (3, 0, 0, -2, 6, 0)$ (siehe Punkt **B** in Abbildung 1.3) und der neue Zielfunktionswert ist -3. In dem neuen Tableau (siehe Tabelle 1.6) gibt es ein $b_i < 0$, somit ist die Basislösung $\mathbf{x}^{(1)}$ unzulässig (siehe Funktion BasisloesungZulaessig). Nach den bisherigen Überlegungen wählen wir s = 2 als Pivotzeile und t = 2 als Pivotspalte und erhalten nach einer weiteren Iteration des dualen Simplex-Verfahrens das folgende Tableau:

	x_1	x_2	x_3	x_4	x_5	x_6	b
x_1	1	0	-1	0	0	0	1
x_2	0	1	0	-1	0	0	2
x_5	0	0	-1	2	1	0	2
x_6	0	0	1	-2	0	1	4
F	0	0	1	1	0	0	-5

Tabelle 1.7: Tableau mit optimaler (zulässigen) Basislösung

Der neue Basispunkt ist $\mathbf{x}^{(2)} = (1, 2, 0, 0, 2, 4)$ (siehe Punkt **C** in Abbildung 1.3) und der neue Wert der Zielfunktion ist -5. In dem Tableau (siehe Tabelle 1.7) ist kein b_i kleiner Null vorhanden, somit ist die Basislösung $\mathbf{x}^{(2)}$ eine zulässige Basislösung. In der Ergebniszeile sind zudem alle Zielfunktionskoeffizienten $\mathbf{c}_{\mathcal{N}}$ größer Null. Dementsprechend erfüllt die neue zulässige Basislösung $\mathbf{x}^{(2)}$ das Optimalitätskriterium und das Verfahren terminiert. Die zulässige Basislösung $\mathbf{x}^{(2)}$ ist somit eine optimale Basislösung $\mathbf{x}^{\star}$. Der entsprechende Wert der optimalen Zielfunktion $f(\mathbf{x}^{\star})$ ist -5.

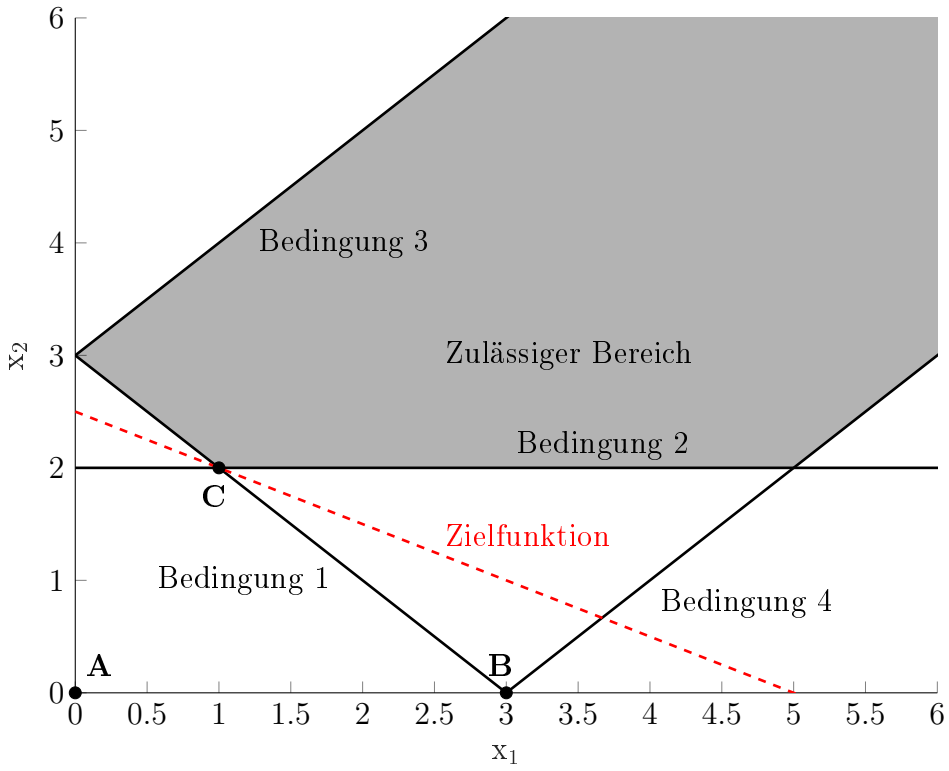


Abbildung 1.3: Graphische Darstellung der linearen Maximierungsaufgabe, dualer Simplex

Algorithmus 2 : Dualer Simplex

Eingabe : Lineare Optimierungsaufgabe $\max\{\mathbf{c}^\intercal\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$ mit $\mathbf{b} \geq \mathbf{0}$.
Ausgabe : Tableau T mit zulässiger Basislösung

```
while !BasisloesungZulaessig(b) do
    s = PivotzeileDual(b);
    if PivotzeileDualZulaessig(s, A) then
        t = PivotspalteDual(s, c, A);
    else
        return c^T x besitzt keine zulässige Basislösung;
    end
    Pivotschritt(s, t, A, b);
end
```

C++ Code: Listing 3.7

Funktion BasisloesungZulaessig(**b**)

Daten : **b** »Kapazitätenvektor«
Ergebnis : (bool) true, wenn eine zulässige Basislösung vorhanden ist

for $i = 1$ **to** m **do**
  if $b_i < 0$ **then**
    return false;
  end
end

return true;

C++ Code: Listing 3.8

Funktion PivotzeileDual(**b**)

Daten : **b** »Kapazitätenvektor«
Ergebnis : s »Pivotzeilenindex«

Wähle Zeile s mit $b_s := \min\{b_i \mid i = 1, \ldots, m\}$;

return s;

C++ Code: Listing 3.9

Funktion PivotzeileDualZulaessig(s, **A**)

Daten : s »Pivotzeile«, **A** »Koeffizientenmatrix«
Ergebnis : (bool) true, wenn die Pivotzeile zulässig ist

for $j = 0$ **to** $n+m$ **do**
  if $a_{sj} < 0$ **then**
    return true;
  end
end

return false;

C++ Code: Listing 3.10

Funktion PivotspalteDual(s, **c**, **A**)

Daten : s »Pivotzeile«, F-Zeile, **A** »Koeffizientenmatrix«
Ergebnis : t »Pivotspalte«

Wähle Spalte t mit $\frac{c_t}{a_{st}} := \max\left\{\frac{c_j}{a_{sj}} \,\middle|\, j = 1, \ldots, \text{n+m mit } a_{sj} < 0\right\}$;

return t;

C++ Code: Listing 3.11

2 Anwendungsleitfaden der Simplex-Algorithmus Programmierung

Im Folgenden schildern wir das praktische Vorgehen zur Nutzung der in C++ implementierten Anwendung zur Lösung von linearen Optimierungsproblemen mithilfe des Simplex-Algorithmus. Das in der Abbildung 2.2 abgebildete Ablaufdiagramm beschreibt in grober Form die Folge von Operationen des Programmcodes zur Lösung einer linearen Optimierungsaufgabe.

> Die .exe Datei der Anwendung lässt sich unter einem Windows Betriebssystem starten. Unter Umständen erfolgt vor dem Start der Anwendung eine Abfrage der Virensoftware ob das Programm als vertrauenswürdig eingestuft werden kann. Um die .exe Datei unter einem anderem Betriebssystem (z. B. Linux oder macOS) auszuführen, müssen zuvor geeignete Schritte vorgenommen werden.

Die »Simplex.exe« Anwendung befindet sich in dem Ordner »release«. In dem Ordner »release« befinden sich noch weitere Unterordner und .dll Dateien. Diese Unterordner und Dateien sind notwendig, um das »Simplex.exe« Programm ausführen zu können. Der gesamte zugehörige Quellcode befindet sich in dem Ordner »Simplex«. Über die Datei »Simplex.pro« in dem Ordner »Simplex«, lässt sich im Qt-Creator das gesamte Projekt öffnen. Es existieren zwei Möglichkeiten die »Simplex.exe« zu starten:

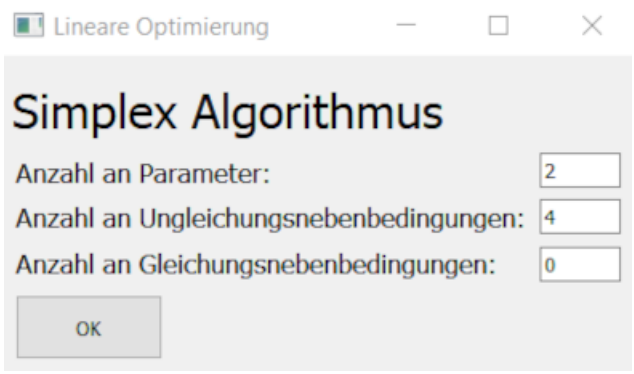
- »Simplex.exe« in dem Order »release« starten
- Das »Simplex.pro« Projekt über den Qt-Creator ausführen

Die praktische Anwendung des Programms wir anhand des in Abschnitt 1.3 beschriebenen Maximierungsproblems erläutert. Nach dem Start der Anwendung wird der Benutzer aufgefordert die Anzahl an Parametern (bzw. Strukturvariablen), Ungleichungsnebenbedingungen und Gleichungsnebenbedingungen der linearen Optimierungsaufgabe anzugeben (siehe Abbildung 2.1a). Nach der Bestätigung der Eingabe öffnet sich das Diaglogfenster »LP-Modell definieren« (siehe Abbildung 2.1b). Hier wird der Benutzer dazu aufgefordert, in dem bereits vordefinierten Layout die Koeffizientenmatrix und den Kapazitätenvektor des Nebenbedingungssystems und die Zielfunktion der linearen Maximierungsaufgabe anzugeben.

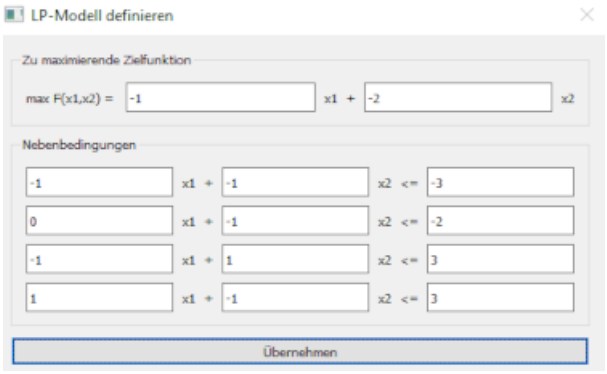
> Die lineare Optimierungsaufgabe muss vom Anwender in Form einer linearen Maximierungsaufgabe angegeben werden. Um die Lösung einer linearen Minimierungsaufgabe mit dem implementierten Simplex-Verfahren zu berechnen, muss das zu dem linearen Minimierungsproblem entsprechende duale Maximierungsproblem aufgestellt werden. Die Vorgehensweise, um das duale Modell zu bestimmen, wird in Kapitel 2.5.1 »Einführung in Operations Research« [2] beschrieben. Die Werte der Koeffizientenmatrix und des Kapazitätenvektors dürfen nicht als Bruch eingelesen werden (z. B. nicht 3/4 sondern 0.75).

Vorhandene $\geq$ Nebenbedingungen muss der Anwender mittels Multiplikation beider Operanden der Ungleichungsnebenbedingungen mit -1 in $\leq$ Nebenbedingungen transformieren (siehe Gleichung 1.6). Erforderliche Gleichungsrestriktionen werden vom Benutzer im Dialogfenster auch als Gleichungsnebenbedingungen eingelesen. In diesem Fall erfolgt die notwendige Umformung nach Gleichung 1.7 durch die C++ Anwendung. Nach der Bestätigung der Eingabe öffnet sich das Ergebnisfenster »Lösung des LP-Modells«. Hier werden die berechneten Tableaus und die dazugehörigen Basislösungen ausgegeben (siehe Abbildung 2.1c). Die Anordnung der Tableaus richtet sich nach der Vorlage in der Tabelle 1.1. Die C++ Anwendung liefert für eine lineare Optimierungsaufgabe, mit genau zwei Strukturvariablen und keinen Gleichungsnebenbedingungen, zusätzlich eine grafische Lösung (siehe 2.1d). In der grafischen Darstellung

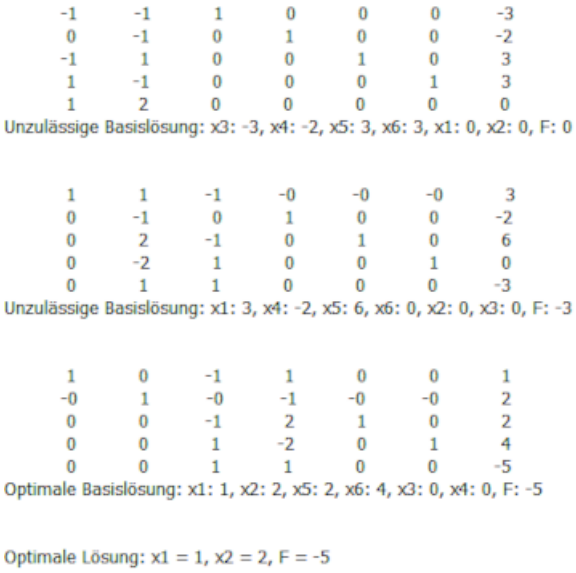
werden die Nebenbedingungsgeraden, die Gerade der maximierten Zielfunktion und die berechneten Basispunkte abgebildet. Der Bereich der zulässigen Basislösungen (siehe Abbildung 1.3) wird in der grafischen Ausgabe nicht wiedergegeben.



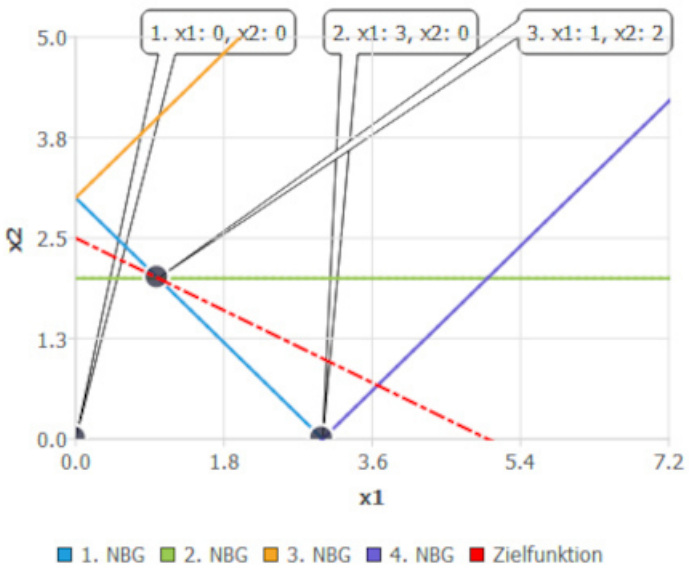
(a) Abfrage der Anzahl von Strukturvariablen und Nebenbedingungen



(b) Eingabe der Koeffizienten der Zielfunktion und des Nebenbedingungssystems



(c) Ausgabe der berechneten Tableaus



(d) Ausgabe der graphischen Darstellung der Optimierungsaufgabe

Abbildung 2.1: Anwendung der Implementierung des Simplex-Algorithmus

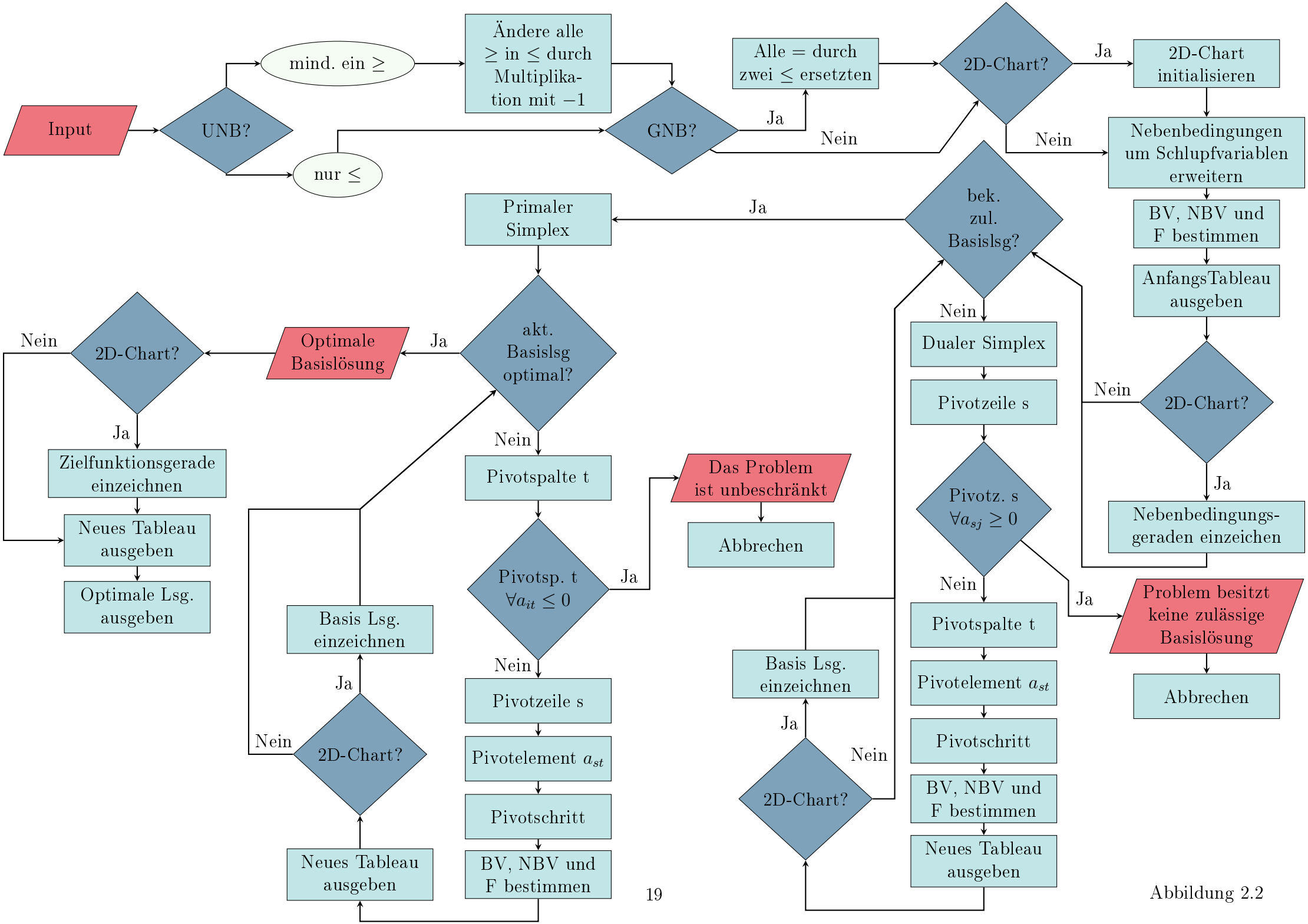


Abbildung 2.2

3 Quellcode

Die folgenden Abschnitte beinhalten den in C++ implementierten Programmiercode. Es wurden lediglich die Funktionen und Prozeduren, auf welche in den Abschnitten 1.2 und 1.3 Bezug genommen wurde, mit in die Ausarbeitung aufgenommen. Der Quellcode für die Datenverarbeitung, die grafische Benutzeroberfläche bzw. Ausgabe und weitere notwendige Hilfsfunktionen befinden sich in dem Ordner »Simplex«. Über die Datei »Simplex.pro« in dem Ordner »Simplex«, lässt sich im Qt-Creator das gesamte Projekt öffnen.

3.1 Primaler Simplex-Algorithmus

Listing 3.1: Primaler Simplex

```
1     //Den primalen Simplex Algorithmus solange wiederholen bis die ↩
          Basisloesung optimal ist
2     while (!simplex.isOptim())
3     {
4         //Schritt 1:
5         //Suche die Spalte t mit dem kleinsten (negativen) Wert in ↩
              Kapazitaetenvektor c (stehen mehrere Spalten mit einem ↩
              kleinstem Wert zur Auswahl, so waehlt der Algorithmus std::↩
              min_element das Element mit dem kleineren index). Die zugehö↩
              rige Nichtbasisvariable xt wird neu in die Basis aufgenommen. ↩
              Die Spalte t ist die Pivospalte
6         simplex.setPivotCol(simplex.indexPivotCol());
7
8         //Schritt 2:
9         //Sind in der Pivotspalte t alle a_it < 0, so kann für das ↩
              betrachtete Problem keine optimale Loesung angegeben werden -->↩
               Abbruch des Verfahrens.
10        if (!simplex.isColPivotValid(simplex.getPivotCol()))
11        {
12            qDebug() << "Das betrachtete Problem ist unbeschraenkt.";
13            simplex.setSolutionType(2);
14            printSolution(simplex);
15            return;
16        }
17        //Pivotzeile bestimmen
18        simplex.setPivotRow(simplex.indexPivotRow());
19
20        //Schritt 3:
21        //Berechnung der neuen Basisloesung, des neuen Simplex-Tableaus
22        simplex.pivotStep();
23
24        //BV, NBV und F setzen
25        simplex.setBasePrimParam();
26        //print Tableau
27        printTableau(simplex);
28        //BV, NBV und F ausgeben
29        printBaseSol(simplex);
30        //Basisloesung in Chart
31        if(numberStruct == 2 && numberGNB == 0)
32            baseSolChart(simplex);
33    }
34
```

```
35      //Loesung ausgeben
36      printSolution(simplex);
37      //Zielfungtionsgerade im Chart abbilden
38      if(numberStruct == 2 && numberGNB == 0)
39          chartZielfunktion(simplex);
40  }
```

Listing 3.2: Optimale Basislösung vorhanden

```
1  //Pruefen ob die Basisloesung optimal ist
2  bool Simplex::isOptim()
3  {
4      for (int j = 0; j < cols; ++j)
5      {
6          if (c[j] < 0)
7              return false;
8      }
9      this->optim = true;
10
11     return true;
12 }
```

Listing 3.3: Pivotspalte

```
1  //Pivotspalte bestimmmen
2  int Simplex::indexPivotCol()
3  {
4      return  std::min_element(c.begin(), c.end()) - c.begin();
5  }
```

Listing 3.4: Zulässige Pivotspalte vorhanden

```
1  //ist eines der Elemente a_it der Pivotspalte t groeßer 0 --> a_it > 0
2  bool Simplex::isColPivotValid(int t)
3  {
4      for (int i = 0; i < rows; ++i)
5      {
6          if (A[i][t] > 0)
7              return true;
8      }
9      return false;
10 }
```

Listing 3.5: Pivotzeile

```
//Pivotzeile bestimmen
int Simplex::indexPivotRow()
{
    double tmp{ 0 }, min{ 0 }, index_min{ 0 }, index_tmp{ 0 };
    bool first{ true };
    for (int i = 0; i < rows; ++i)
    {
        if (A[i][pivotCol] > 0)
        {
            if (first)
            {
                min = (b[i] / A[i][pivotCol]);
                index_min = i;
                first = false;
            }
            tmp = (b[i] / A[i][pivotCol]);
            index_tmp = i;
            index_min = tmp < min ? index_tmp : index_min;
        }
    }
    return index_min;
}
```

Listing 3.6: Pivotschritt

```
//Neue Basisloesung berechnen --> Transformation des Tableaus
void Simplex::pivotStep()
{
    //Dividiere die Pivotzeile s durch das Pivotelement a_st
    double a_st = A[pivotRow][pivotCol];
    for (int j = 0; j < cols; ++j)
    {
        A[pivotRow][j] = (A[pivotRow][j] / a_st);
    }
    b[pivotRow] = (b[pivotRow] / a_st);

    //Multipliziere für alle Zeilen i (außer s) die neue Pivotzeile mit
        (-a_it) und addiere sie zur jeweiligen Zeile i hinzu.
    double a_it{ 0 };
    for (int i = 0; i < rows; ++i)
    {
        if (i != pivotRow)
        {
            a_it = A[i][pivotCol];
            for (int j = 0; j < cols; ++j)
            {
                A[i][j] = A[i][j] - a_it * A[pivotRow][j];
            }
            b[i] = b[i] - a_it * b[pivotRow];
        }
    }
    double c_col = c[pivotCol];
    for (int j = 0; j < cols; ++j)
        c[j] = c[j] - c_col * A[pivotRow][j];
}
```

3.2 Dualer Simplex-Algorithmus

Listing 3.7: Dualer Simplex

```
1     //Abfrage ob der Vektor b nicht negative Elemente auweist --> nicht ↩
          zulaessige Basisloesung -->dualer Siplex-Algorithmus. Wenn kein b ↩
          nicht negative Elemente aufweist dann liegt bereits eine zulaessige↩
           Basisloesung vor
2     while (!simplex.isBaseValid())
3        {
4            //Schritt 1:
5            //Wahl der Pivotzeile s, Zeile mit dem kleinsten b_s < 0 als ↩
                 Pivotzeile
6            simplex.setPivotRow(simplex.indexPivotRowDual());
7
8            //Schritt 2:
9            //Befindet sich in der Pivotzeile s kein a_sj < 0, so besitzt ↩
                 das Problem keine zulaessige Basisloesung --> Abbruch des ↩
                 Verfahrens
10           if (!simplex.isRowPivotDoubleValid(simplex.getPivotRow()))
11           {
12               qDebug() << "Das Problem hat keine zulaessige Basisloesung";
13               simplex.setSolutionType(3);
14               printSolution(simplex);
15               return;
16           }
17
18           //Pivotspalte bestimmen
19           simplex.setPivotCol(simplex.indexPivotColDual());
20
21           //Schritt 3:
22           //Tableautransformation
23           simplex.pivotStep();
24
25           //BV, NBV und F setzen
26           simplex.setBasePrimParam();
27           //print Tableau
28           printTableau(simplex);
29           //BV, NBV und F ausgeben
30           printBaseSol(simplex);
31           //Basisloesung in Chart
32           if(numberStruct == 2 && numberGNB == 0)
33               baseSolChart(simplex);
34       }
```

Listing 3.8: Zulässige Basislösung vorhanden

```
1  //Abfrage ob die Basisloesung zulaessig ist
2  bool Simplex::isBaseValid()
3  {
4      for (int i = 0; i < rows; ++i)
5      {
6          if (b[i] < 0)
7              return false;
8      }
9      return true;
10 }
```

Listing 3.9: Pivotzeile-Dual

```
//(dualer Simplex) Wahl der Pivotzeile s, Zeile mit dem kleinsten b_s < 0 als Pivotzeile
int Simplex::indexPivotRowDual()
{
    return  std::min_element(b.begin(), b.end()) - b.begin();
}
```

Listing 3.10: Zulässige Pivotzeile-Dual vorhanden

```
bool Simplex::isRowPivotDoubleValid(int s)
{
    for (int j = 0; j < cols; ++j)
    {
        if (A[s][j] < 0)
            return true;
    }
    return false;
}
```

Listing 3.11: Pivotspalte-Dual

```
//Bestimmen der Pivotspalte t
int Simplex::indexPivotColDual()
{
    double tmp{ 0 }, max{ 0 }, index_max{ 0 }, index_tmp{ 0 };
    bool first{ true };
    for (int j = 0; j < cols; ++j)
    {
        if (A[pivotRow][j] < 0)
        {
            if (first)
            {
                max = (c[j] / A[pivotRow][j]);
                index_max = j;
                first = false;
            }
            tmp = (c[j] / A[pivotRow][j]);
            index_tmp = j;
            index_max = tmp > max ? index_tmp : index_max;
        }
    }
    return index_max;
}
```

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Burkard, Rainer E. und Uwe Zimmermann: *Einführung in die Mathematische Optimierung*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, ISBN 978-3-642-28673-5. https://doi.org/10.1007/978-3-642-28673-5_3.

[2] Domschke, Wolfgang, Andreas Drexl, Robert Klein und Armin Scholl: *Einführung in Operations Research*, Seiten 17–70. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015, ISBN 978-3-662-48216-2. https://doi.org/10.1007/978-3-662-48216-2_2.

[3] Domschke, Wolfgang, Andreas Drexl, Robert Klein, Armin Scholl und Stefan Voß: *Übungen und Fallbeispiele zum Operations Research*, Seiten 11–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015, ISBN 978-3-662-48230-8. https://doi.org/10.1007/978-3-662-48230-8_2.

[4] Koop, Andreas und Hardy Moock: *Lineare Optimierung – eine anwendungsorientierte Einführung in Operations Research*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2018, ISBN 978-3-662-56141-6. https://doi.org/10.1007/978-3-662-56141-6_3.

[5] Müller, Rebekka: *Eine Einführung in den Simplex-Algorithmus*, 2015. https://kops.uni-konstanz.de/handle/123456789/32197.

[6] Qt-Group: *GUI-Programming*. https://www.qt.io/.