INSTITUT FÜR INFORMATIK

Die Experimental Design Toolbox zur optimalen Versuchsplanung

Joscha Reimer

Bericht Nr. 1106
Juni 2011
ISSN 2192-6247

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

ZU KIEL

Die Experimental Design Toolbox zur optimalen Versuchsplanung

Joscha Reimer *

30. Juni 2011

Zusammenfassung

Wir stellen nachfolgend die Theorie der optimalen Versuchsplanung vor. Die entstehenden Optimierungsprobleme werden hergeleitet und zugehörige Lösungsverfahren präsentiert. Daraufhin beschreiben wir eine allgemeine Vorgehensweise zur Parameterschätzung mit Hilfe der optimalen Vesuchsplanung. Die Verfahren zur optimalen Versuchsplanung und Parameterschätzung wurden als Matlab Toolbox implementiert. Deren Benutzung und Implementierung beleuchten wir abschließend.

Schlüsselwörter: optimale Versuchsplanung, Matlab Toolbox, Parameterschätzung, Ausgleichsrechnung, Methode der kleinsten Quadrate

*jor@informatik.uni-kiel.de, Institut für Informatik, Algorithmische optimale Steuerung - CO_2 Aufnahme des Meeres, Exzellenzcluster Ozean der Zukunft, Christian-Albrechts-Platz 4, 24118 Kiel, Deutschland.

1 Einleitung

In Industrie, Wissenschaft und Technik werden oft Prozesse durch mathematische Modelle beschrieben, die in der Regel unbekannte Parameter enthalten. Diese werden bestimmt, indem Messungen der modellierten Größe durchgeführt werden. Die Parameter werden daraufhin so gewählt, dass die Modellausgaben möglichst exakt zu den Messergebnissen passen.

Häufig ist das Durchführen von Messungen aufwendig oder kostenintensiv. In diesen Fällen ist es besonders wichtig, dass die Messungen einen bestmöglichen Zugewinn an Informationen liefern und dementsprechend die Parameter mit möglichst wenig Messungen hinreichend exakt bestimmt werden können. Welchen Zugewinn einzelne Messungen liefern und welche Messpunkte zum Durchführen von Messungen optimalerweise gewählt werden sollten, ist Gegenstand der optimalen Versuchsplanung.

Im Folgenden werden wir die Theorie der optimalen Versuchsplanung präsentieren. Die dort vorgestellten Verfahren wurden in einer Matlab Toolbox implementiert. Diese wird im Anschluss vorgestellt.

2 Optimale Versuchsplanung

In diesem Abschnitt stellen wir die bei der optimalen Versuchsplanung zu lösenden Optimierungsprobleme und zugehörigen Lösungsverfahren vor. Weiter präsentieren wir eine allgemeine Vorgehensweise zur Parameterschätzung mit Hilfe der optimalen Versuchsplanung.

Als Grundlage für diesen Abschnitt dienen die Masterarbeit [Re11] von Joscha Reimer und die Doktorarbeit [Kö02] von Dr. Stefan Körkel.

2.1 Optimierungsprobleme der optimalen Versuchsplanung

Wir werden nachfolgend die verschiedenen Optimierungsprobleme der optimalen Versuchsplanung herleiten. Dazu führen wir als erstes einige Bezeichnungen und Anforderungen ein und motivieren danach die verschiedenen Optimierungsprobleme.

Die Modellfunktion

Nachfolgend spezifizieren wir das Aussehen der Modellfunktion, die den modellierten funktionalen Zusammenhang beschreibt und deren Parameter bestimmt werden sollen. Wir gehen davon aus, dass $n \in \mathbb{N}$ Parameter zu bestimmen sind und fassen diese als Vektor im $\mathbb{R}^n$ zusammen. Für die Messpunkte wählen wir eine beliebige Menge $\Omega \subseteq \mathbb{R}^l$ mit $l \in \mathbb{N}$. Diese können zum Beispiel Punkte in der Zeit, im Ort oder in beidem sein. Die Modellfunktion kann dann eine beliebige Funktion

$$f : \Omega \times \mathbb{R}^n \to \mathbb{R}$$

sein, die bezüglich der Parameter stetig differenzierbar ist. Dies ist notwendig, um die im Weiteren vorgestellte Theorie der optimalen Versuchsplanung anwenden zu können.

Die Messpunkte

Weiter gehen wir von einer Menge von potenziellen Messpunkten

$$T_{var} := \{t_1, \ldots, t_{m_{var}}\} \subseteq \Omega$$

aus. Aus dieser soll eine Teilmenge von Messpunkten $S \subseteq T_{var}$ ausgewählt werden, um dort Messungen durchzuführen. Um dieses Entscheidungsproblem mathematisch zu formulieren, identifizieren wir jede Teilmenge $S \subseteq T_{var}$ mit einem binären Vektor $w_S \in \{0,1\}^{m_{var}}$. Dabei setzen wir

$$(w_S)_i := \begin{cases} 1 & \text{falls } t_i \in S \\ 0 & \text{sonst} \end{cases} \quad \text{für alle } i \in \{1, \ldots, m_{var}\}.$$

Eventuell wurden bereits Messungen durchgeführt. Die dazugehörigen Messpunkte fassen wir in folgender Menge zusammen.

$$T_{fix} := \{t_{m_{var}+1}, \ldots, t_{m_{var}+m_{fix}}\} \subseteq \Omega$$

Wurden bisher keine Messungen durchgeführt, so gilt $T_{fix} = \emptyset$ und wir setzen $m_{fix} = 0$. Weiter setzen wir

$$T := T_{var} \cup T_{fix} \text{ und } m := m_{var} + m_{fix}.$$

Beschränkungen an die Auswahl der Messpunkte

Oft tritt der Fall ein, dass die auszuwählenden Messpunkte $S \subseteq T_{var}$ nicht beliebige Teilmenge sein dürfen, sondern Beschränkungen unterliegen. Diese Beschränkungen lassen sich durch lineare Beschränkungen an die Gewichte w_S ausdrücken.

Ein häufiges Szenario ist, dass die Anzahl der auszuwählenden Messpunkte zwischen $a \in \mathbb{R}$ und $b \in \mathbb{R}$ liegen soll. Diese Beschränkung lässt sich folgendermaßen formulieren.

$$a \leq \sum_{i=1}^{m_{var}} (w_S)_i \leq b$$

Eine weitere Möglichkeit ist, die Kosten der Messungen zu beschränken. Dazu kann jedem Messpunkt $t_i \in T$ ein Wert $c_i \in \mathbb{R}$ als resultierende Kosten für dortige Messungen zugewiesen werden. Sollen nun die Gesamtkosten der Messungen kleiner gleich $d \in \mathbb{R}$ sein, so kann folgende Beschränkung gefordert werden.

$$\sum_{i=1}^{m_{var}} (w_S)_i c_i \leq d$$

Allgemein können beliebige Teilmengen $\tilde{S} \subseteq T_{var}$ von der Auswahl ausgeschlossen werden, indem

$$\sum_{i=1}^{m_{var}} (2(w_{\tilde{S}})_i - 1)(w_S)_i \leq \sum_{i=1}^{m_{var}} (w_{\tilde{S}})_i - 1$$

gefordert wird. Nachrechnen ergibt, dass diese Bedingung genau dann erfüllt ist, wenn $S \neq \tilde{S}$ gilt. Durch mehrere dieser linearen Beschränkungen kann also die Auswahl der Messergebnisse beliebig beschränkt werden.

Alle linearen Beschränkungen können wir zusammenfassen, indem wir eine geeignete Matrix $U \in \mathbb{R}^{l \times m_{var}}$ und einen Vektor $v \in \mathbb{R}^l$ für ein $l \in \mathbb{N}$ wählen und

$$Uw_S \leq v$$

fordern. Dabei ist $\leq$ komponentenweise zu verstehen.

Anforderungen an die Messfehler

Nachfolgend stellen wir einige Anforderungen an die zu erwartenden Messfehler. Für jeden Messpunkt $t_i \in T$ betrachten wir die zugehörigen Messergebnisse als Realisierung einer Zufallsvariablen H_i. Wir gehen davon aus,

dass die zum Messpunkt t_i zugehörigen Messfehler unabhängig und normalverteilt mit Erwartungswert 0 und Standardabweichung σ_i sind. Das heißt, es gilt

$$f(t_i, \hat{p}) - H_i \sim N(0, \sigma_i^2) \text{ für alle } i \in \{1, \dots, m\}.$$

Hierbei ist $\hat{p} \in \mathbb{R}^n$ der Vektor der unbekannten wahren Werte der Parameter und damit $f(t_i, \hat{p})$ der wahre Wert der modellierten Größe am Messpunkt t_i.

Um diese Anforderungen kompakt beschreiben zu können, definieren wir

$$F : \mathbb{R}^n \to \mathbb{R}^m \text{ mit } (F(p))_i := f(t_i, p) \text{ für alle } i \in \{1, \dots, m\}$$
$$\Sigma := diag(\sigma_1, \dots, \sigma_m)$$
$$H := (H_1, \dots, H_m)^T.$$

Mit Hilfe der multivariaten Normalverteilung können wir die Annahme über die Messfehler dann folgendermaßen schreiben.

$$F(\hat{p}) - H \sim N(0, \Sigma^2)$$

Gewichtetes Ausgleichsproblem

Würden wir nun an allen Messpunkten Messungen durchführen, so würden wir eine Realisierung $\eta \in \mathbb{R}^m$ von H erhalten. Das zugehörige gewichtete Ausgleichsproblem wäre

$$\min_{p \in \mathbb{R}^n} \|\Sigma^{-1}(F(p) - \eta)\|_2^2.$$

Hierbei werden die einzelnen Messungen entsprechend der Varianz des zugehörigen Messfehlers gewichtet. Dadurch wird erreicht, dass genauere Messungen stärker berücksichtigt werden.

Würden wir nur an den Messpunkten $S \subseteq T_{var}$ Messungen durchführen, so würden diese zusammen mit den vorher durchgeführten Messungen zu folgendem gewichteten Ausgleichsproblem führen

$$\min_{p \in \mathbb{R}^n} \|W_S \Sigma^{-1}(F(p) - \eta)\|_2^2.$$

Hierbei ist

$$W_S := \text{diag}([w_S, 1_{m_{fix}}])$$

die Diagonalmatrix mit dem Vektor $[w_S, 1_{m_{fix}}]$ auf der Diagonalen. Weiter ist $1_{m_{fix}} \in \mathbb{R}^{m_{fix}}$ der Vektor, bei dem alle Komponenten gleich 1 sind und $[w_S, 1_{m_{fix}}]$ der durch Konkatenation der zwei Vektoren w_S und $1_{m_{fix}}$ entstehende Vektor.

Verteilung der Lösung eines linearen Ausgleichsproblems

Wir betrachten als erstes den Fall, dass die Modellfunktion f linear bezüglich der Parameter ist. Ist dies der Fall, so ist auch F linear bezüglich der Parameter und es existieren $A \in \mathbb{R}^{m \times n}$ und $b \in \mathbb{R}^m$ mit

$$F(p) = Ap + b$$

für alle $p \in \mathbb{R}^n$. Das gewichtete Ausgleichsproblem wird im linearen Fall für ein $S \subseteq T_{var}$ zu

$$\min_{p \in \mathbb{R}^n} \|W_S \Sigma^{-1}(Ap + b - \eta)\|_2^2.$$

Dieses lineare Ausgleichsproblem besitzt genau dann eine eindeutige Lösung, wenn $W_S \Sigma^{-1} A$ vollen Rang hat. Siehe dazu Satz 2.1 in [Re11]. Das bedeutet insbesondere, dass insgesamt mindestens so viele Messungen durchgeführt werden müssen, wie Parameter bestimmt werden sollen. Dies ist sinnvoll, da mit weniger Messungen die Parameter nicht eindeutig bestimmt werden könnten. Im Folgenden gehen wir davon aus, dass $W_S \Sigma^{-1} A$ vollen Rang hat.

Für jeden Vektor von Messergebnissen $\eta \in \mathbb{R}^m$ erhalten wir also als Lösung des zugehörigen linearen gewichteten Ausgleichsproblems einen eindeutigen Vektor von Parametern $p_S^* \in \mathbb{R}^n$. Da wir den Vektor der Messergebnisse η als eine Realisierung eines Zufallsvektors H betrachten, können wir auch die zugehörige eindeutige Lösung p_S^* des linearen Ausgleichsproblems als Realisierung eines Zufallsvektors P_S^* betrachten. Für diesen Zufallsvektor gilt

$$P_S^* \sim N(\hat{p}, C_S) \text{ mit}$$
$$C_S := (A^T \Sigma^{-1} W_S \Sigma^{-1} A)^{-1}.$$

Siehe dazu auch Satz 2.3 in [Re11].

Minimierungssproblem der linearen optimalen Versuchsplanung

Unser Ziel ist es, die Menge $S \subseteq T_{var}$ so zu wählen, dass die resultierende Lösung p_S^* des Ausgleichsproblems möglichst nah an den wahren Parametern $\hat{p}$ ist. Da $\hat{p}$ der Erwartungswert von P_S^* ist, müssen wir S also so wählen, dass die Varianzen der einzelnen Komponenten der Lösung P_S^* und damit die Diagonaleinträge der Kovarianzmatrix C_S so klein wie möglich sind.

Wir benutzen dazu ein Gütekriterium $\phi : \mathbb{R}^{n \times n} \to \mathbb{R}$, das die Größe der Diagonaleinträge einer Matrix quantifiziert. Ein häufig genutztes Gütekriterium

ist die Spur der Matrix. Neben diesem Gütekriterium wurden viele weitere Gütekriterien entwickelt. Eine gute Übersicht liefert [AbRaSe09].

Mit Hilfe eines Gütekriteriums ϕ können wir nun die Güte einer Kovarianzmatrix C_S quantifizieren. Da wir S so wählen wollen, dass die Diagonaleinträge von C_S möglichst klein sind und damit auch $\phi(C_S)$ möglichst klein ist, lösen wir folgendes *Minimierungssproblem der linearen optimalen Versuchsplanung.*

$$\min_{w \in \{0,1\}^{m_{var}}} \phi(C(w)) \text{ mit } Uw \leq v \text{ und}$$
$$C : \mathbb{R}^{m_{var}} \to \mathbb{R}^{n \times n}, w \mapsto (A^T \Sigma^{-1} \text{diag}([w, 1_{m_{fix}}]) \Sigma^{-1} A)^{-1}$$

Mit dessen Lösung lässt sich die Menge der optimalen Messpunkte dann direkt bestimmen.

Wird als Gütekriterium die Spur gewählt, so sollte die Kovarianzmatrix skaliert werden, um eine gleichmäßige Minimierung der Varianzen zu erreichen. Siehe dazu Abschnitt 3.5 in [Re11].

Minimierungssproblem der optimalen Versuchsplanung mit Punktschätzung der wahren Parameter

Ist die Modellfunktion f nichtlinear bezüglich der Parameter, so führen wir dieses auf den linearen Fall zurück. Dazu linearisieren wir die Modellfunktion f mit Hilfe des Taylorpolynoms ersten Grades bezüglich der Parameter. Wir benutzen dabei $\hat{p}$ als Entwicklungpunkt. Dadurch wird erreicht, dass die Modellfunktion und deren Linearisierung im Parameter $\hat{p}$ identisch sind und damit unsere Annahme über die Messfehler auch für die linearisierte Modellfunktion gilt.

Wenden wir auf die linearisierte Modellfunktion die vorher vorgestellte Theorie der linearen optimalen Versuchsplanung an, so erhalten wir folgendes Minimierungssproblem.

$$\min_{w \in \{0,1\}^{m_{var}}} \phi(C(w, \hat{p})) \text{ mit } Uw \leq v \text{ und}$$
$$C : \mathbb{R}^{m_{var}} \times \mathbb{R}^n \to \mathbb{R}^{n \times n}, (w, p) \mapsto (J(p)^T \Sigma^{-1} \text{diag}([w, 1_{m_{fix}}]) \Sigma^{-1} J(p))^{-1}$$

Hierbei ist $J(\hat{p})$ die Jacobi-Matrix von F an der Stelle $\hat{p}$. Leider kennen wir den wahren Parameter $\hat{p}$ nicht. Darum müssen wir $\hat{p}$ durch eine Schätzung

p^* ersetzen und erhalten folgendes *Minimierungssproblem der optimalen Versuchsplanung mit Punktschätzung der wahren Parameter.*

$$\min_{w \in \{0,1\}^{m_{var}}} \phi(C(w, p^*)) \text{ mit } Uw \leq v \text{ und}$$
$$C : \mathbb{R}^{m_{var}} \times \mathbb{R}^n \to \mathbb{R}^{n \times n}, (w, p) \mapsto (J(p)^T \Sigma^{-1} \text{diag}([w, 1_{m_{fix}}]) \Sigma^{-1} J(p))^{-1}$$

Sofern bereits mindestens n Messungen durchgeführt wurden, kann als Schätzung die Lösung des zugehörigen Ausgleichsproblems genutzt werden. Anderenfalls kann die Schätzung zum Beispiel aus Erfahrung mit verwandten Prozessen herrühren.

Minimierungs-Maximierungsproblem der optimalen Versuchsplanung mit Gebietsschätzung der wahren Parameter

Eine andere Vorgehensweise ist, die Varianzen der Lösung nicht für eine einzelne Schätzung der Parameter zu minimieren, sondern für ein ganzes Gebiet. Dafür gehen wir davon aus, dass bereits mindestens n Messungen durchgeführt wurden. Dann können wir in Abhängigkeit von den bisherigen Messergebnissen ein minimales Gebiet konstruieren, welches in 100 α % der Fälle $\hat{p}$ enthält. Hierbei wurde $\alpha \in (0, 1)$ gewählt. Ein solches Gebiet wird α-Konfidenzgebiet für $\hat{p}$ genannt.

Mit den bisherigen Messungen lässt sich das zugehörige Ausgleichsproblem aufstellen und dessen Lösung p^* berechnen. Wie schon vorher betrachten wir die Lösung p^* als Realisierung eines Zufallsvektors P^*. Da wir davon ausgehen, dass die Lösung p^* nah an den wirklichen Parametern $\hat{p}$ ist und die linearisierte Modellfunktion eine gute Approximation der Modellfunktion ist, nehmen wir an, dass

$$P^* \sim N(\hat{p}, C_0) \text{ mit } C_0 := C(0, p^*)$$

gilt. Ein zugehöriges α-Konfidenzgebiet von $\hat{p}$ ist dann

$$G(\alpha) := \{p \in \mathbb{R}^n \mid \|p - p^*\|^2_{C_0^{-1}} \leq \gamma(\alpha)\}.$$

Siehe dazu auch Satz 2.5 in [Re11]. Dabei ist $\gamma(\alpha)$ das α-Quantil der χ^2_n-Verteilung. Weiter ist die Energienorm $\|v\|_A$ eines Vektors $v \in \mathbb{R}^n$ bezüglich einer symmetrisch positiv definiten Matrix $A \in \mathbb{R}^{n \times n}$ folgendermaßen definiert.

$$\|v\|_A := \sqrt{v^T A v}.$$

Da wir die Varianzen der Lösung des Ausgleichsproblems im ganzen Gebiet $G(\alpha)$ minimieren wollen, betrachten wir folgendes *Minimierungs-Maximierungsproblem der optimalen Versuchsplanung mit Gebietsschätzung der wahren Parameter.*

$$\min_{w\in\{0,1\}^{m_{var}}} \max_{p\in G(\alpha)} \phi(C(w,p)) \text{ mit } Uw \leq v \text{ und}$$
$$C : \mathbb{R}^{m_{var}} \times \mathbb{R}^n \to \mathbb{R}^{n\times n}, (w,p) \mapsto (J(p)^T\Sigma^{-1}\text{diag}([w, 1_{m_{fix}}])\Sigma^{-1}J(p))^{-1}$$

Minimierungssproblem der optimalen Versuchsplanung mit Gebietsschätzung der wahren Parameter

Das Lösen dieses Problems kann sehr aufwendig sein, wenn das Berechnen der Ableitung der Modellfunktion nach den Parametern aufwendig ist. Für diesen Fall wollen wir das Problem abschwächen.

Wir setzen voraus, dass $\phi \circ C$ bezüglich p stetig differenzierbar ist. Dieses ist zum Beispiel der Fall, wenn die zugrunde liegende Modellfunktion f bezüglich der Parameter zweimal stetig differenzierbar ist und das Gütekriterium stetig differenzierbar ist. Wurde als Gütekriterium die Spur gewählt, so ist dies der Fall.

Um das Problem abzuschwächen, linearisieren wir die Funktion $\phi \circ C$ bezüglich der Parameter. Dazu benutzen wir das Taylorpolynom ersten Grades mit p^* als Entwicklungspunkt. Wir erhalten dann folgendes Problem.

$$\min_{w\in\{0,1\}^{m_{var}}} \max_{p\in G(\alpha)} \phi(C(w,p^*)) + (\nabla_p(\phi \circ C)(w,p^*))^T(p-p^*).$$

Hierbei bezeichnet ∇_p den Gradienten bezüglich p.

Das innere Maximierungsproblem lässt sich direkt lösen. Siehe dazu Lemma 3.1 in [Re11]. Setzen wir dessen Lösung in das Problem ein, so erhalten wir das *Minimierungssproblem der optimalen Versuchsplanung mit Gebietsschätzung der wahren Parameter.*

$$\min_{w\in\{0,1\}^{m_{var}}} \phi(C(w,p^*)) + \gamma(\alpha)^{\frac{1}{2}}\|\nabla_p(\phi \circ C)(w,p^*)\|_{C_0} \text{ mit } Uw \leq v \text{ und}$$
$$C : \mathbb{R}^{m_{var}} \times \mathbb{R}^n \to \mathbb{R}^{n\times n}, (w,p) \mapsto (J(p)^T\Sigma^{-1}diag([1_{m_{fix}}, w])\Sigma^{-1}J(p))^{-1}$$

2.2 Lösungsverfahren für Optimierungsprobleme der optimalen Versuchsplanung

Die vorgestellten Varianten der optimalen Versuchsplanung laufen auf ein Minimierungsproblem der folgenden Gestalt hinaus.

$$\min_{\substack{w\in\{0,1\}^m \\ Uw\leq v}} f(w)$$

Dabei ist $f : \mathbb{R}^m \to \mathbb{R}$ für ein $m \in \mathbb{N}$. Weiter ist $U \in \mathbb{R}^{l\times m}$ und $v \in \mathbb{R}^l$ für ein $l \in \mathbb{N}$ und $\leq$ komponentenweise zu verstehen. Die lineare Beschränkung an w resultiert aus zusätzlichen Beschränkungen an die Auswahl der Messpunkte.

Dies ist ein diskretes nichtlineares Optimierungsproblem mit linearen Nebenbedingungen. Für kleine m lässt sich dieses diskrete Optimierungsproblem direkt lösen, indem die 2^m Möglichkeiten für w durchprobiert werden. Für größere m ist diese Vorgehensweise jedoch zu aufwendig.

Relaxiertes Minimierungsproblem

Aus diesem Grund führen wir das diskrete Optimierungsproblem auf ein kontinuierliches Optimierungsproblem zurück und konstruieren aus dessen Lösung eine Approximation der Lösung des diskreten Optimierungsproblems. Dazu relaxieren wir als erstes die Bedingung $w \in \{0, 1\}^m$, indem wir stattdessen $w \in [0, 1]^m$ fordern. Wir erhalten also folgendes kontinuierliche nichtlineare Optimierungsproblem mit linearen Nebenbedingungen.

$$\min_{\substack{w\in[0,1]^m \\ Uw\leq v}} f(w)$$

Resultiert das kontinuierliche Optimierungsproblem aus der vorgestellten optimalen Versuchsplanung mit Punktschätzung und wurde als Gütekriterium die Spur gewählt, so ist das relaxierte Optimierungsproblem konvex. Siehe dazu zum Beispiel Kapitel 7.5 in [BoVa04]. In diesem Fall können wir mit Algorithmen der konvexen Optimierung effektiv ein globales Minimum berechnen.

Ist hingegen die vorgestellte optimale Versuchsplanung mit Gebietsschätzung die Grundlage des kontinuierlichen Optimierungsproblems, so ist dieses im Allgemeinen nicht konvex. Aus diesem Grund lässt sich hier im Regelfall nur ein lokales Minimum effektiv berechnen.

Das kontinuierliche Optimierungsproblem lässt sich also, mit den oben beschriebenen Einschränkungen, mit Verfahren aus der kontinuierlichen nichtlinearen Optimierung lösen. Ein Beispiel für eines dieser Verfahren ist das SQP-Verfahren (Sequentielle Quadratische Programmierung). Für Details zum SQP-Verfahren und anderen Algorithmen zur kontinuierlichen nichtlinearen Optimierung sei hier auf die gängigen Lehrbücher, zum Beispiel [Alt02], verwiesen.

Erzeugung ganzzahliger Lösung

Die kontinuierliche Lösung $\tilde{w} \in [0,1]^m$ müssen wir danach wieder in eine ganzzahlige Lösung $w^* \in \{0,1\}^m$ überführen. Anwendungsbeispiele zeigen, dass die berechneten Gewichte $\tilde{w}_i$ für $i \in \{1,\dots,m\}$ meistens schon nah an einem ganzzahligen Wert sind. Wir geben im Folgenden mehrere Heuristiken an, wie man aus der Lösung des kontinuierlichen Optimierungsproblems eine Lösung des diskreten Optimierungsproblems approximieren kann und verdeutlichen diese an einem Beispiel.

Eine Vorgehensweise ist, die einzelnen kontinuierlichen Gewichte $\tilde{w}_i \in [0,1]$ für alle $i \in \{1,\dots,m\}$ auf eine ganze Zahl $w_i^* \in \{0,1\}$ zu runden.

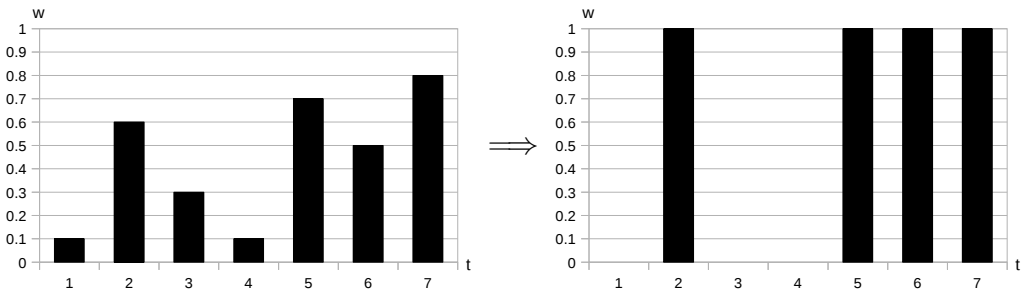

Abbildung 1: Approximieren einer Lösung für das diskrete Problem durch Runden der Lösung des kontinuierlichen Problems.

Eine andere Möglichkeit ist, alle kontinuierlichen Gewichte $\tilde{w}_i \in [0,1]$ für alle $i \in \{1,\dots,m\}$ zu summieren und das Ergebnis zu runden. Dadurch erhält man die Gesamtanzahl m' der durchzuführenden Messungen. Nun wählt man die m' größten kontinuierlichen Gewichte und setzt diese für die diskrete Lösung auf 1. Die anderen Gewichte werden für die diskrete Lösung auf 0 gesetzt.

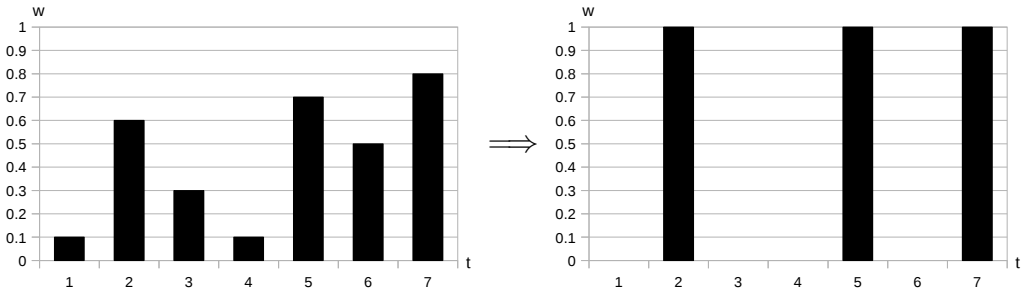

Abbildung 2: Approximieren einer Lösung für das diskrete Problem durch Summieren der Gewichte der Lösung des kontinuierlichen Problems und Auswählen der größten Gewichte.

Beim Konstruieren einer Lösung für das diskrete Problem aus der Lösung des kontinuierlichen Problems ist darauf zu achten, dass die diskrete Lösung weiterhin die Nebenbedingung $Uw \leq v$ erfüllt. Natürlich sind, neben den beiden vorgestellten Vorgehensweisen zum Konstruieren einer Lösung für das diskrete Problem aus der Lösung des kontinuierlichen Problems, weitere Vorgehensweisen möglich.

2.3 Sequentielle optimale Versuchsplanung und Parameterschätzung

Wir haben bereits gesehen, dass wir vorher durchgeführte Messungen in die Versuchsplanung mit einfließen lassen und dadurch geeignete Messpunkte besser bestimmen können. Dies legt ein iteratives Vorgehen zum Bestimmen der Parameter der Modellfunktion nahe.

Dabei starten wir mit einer initialen Schätzung der Parameter und bestimmen mit Hilfe der optimalen Versuchsplanung Messpunkte. Für diese Messpunkte führen wir Messungen durch und schätzen daraus die Parameter. Für diese Parameterschätzung können wir die Güte schätzen. Ist die Güte nicht ausreichend, so bestimmen wir weitere Messpunkte mit Hilfe der neuen Parameterschätzung und verbessern die Parameterschätzung mit weiteren Messungen an diesen Messpunkten. Diese Prozedur wiederholen wir, bis die Güte der Parameterschätzung ausreichend ist.

Dieses Vorgehen wurde unter anderem in [BaBoKöSc99] vorgestellt und wird nachfolgend als Ablaufdiagramm dargestellt.

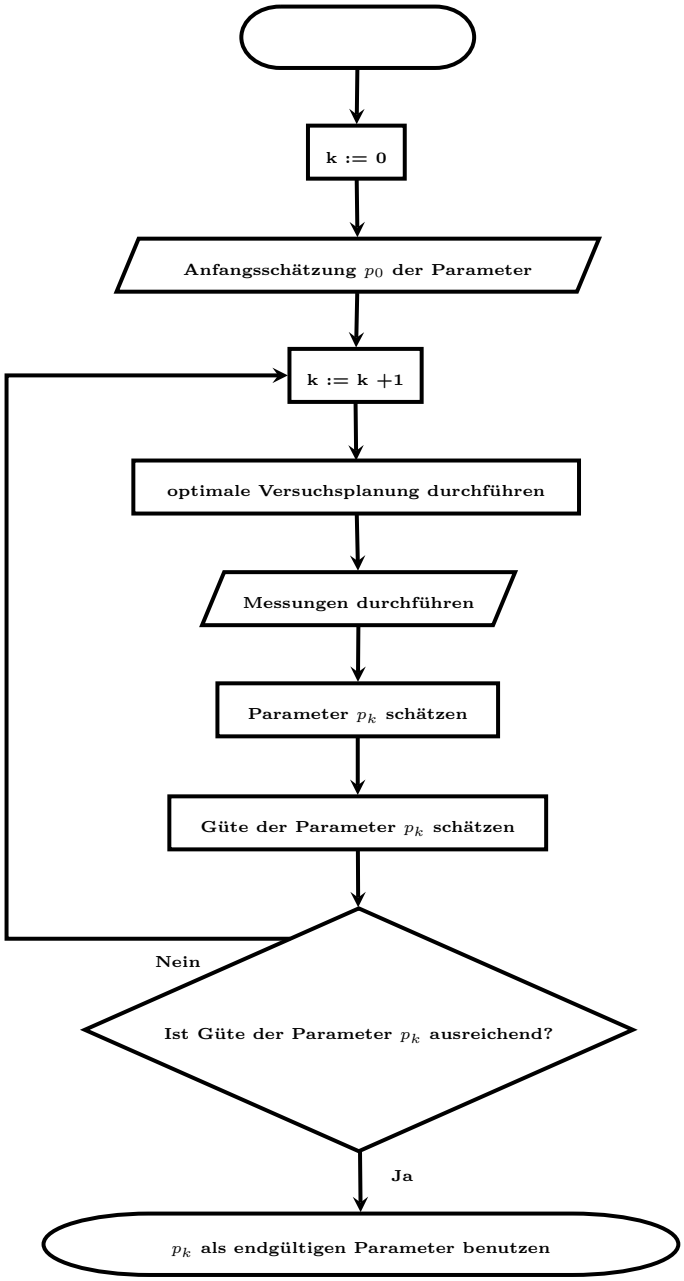

Abbildung 3: Sequentielle optimale Versuchsplanung und Parameterschätzung

3 Die Experimental Design Toolbox

Die vorher vorgestellten Methoden zur Ausgleichsrechnung und optimalen Versuchsplanung haben wir als Matlab Toolbox realisiert. Die Toolbox trägt den Namen *Experimental Design Toolbox*.

Für Matlab wurde sich entschieden, da Matlab Vektor- und Matrix-Operationen unterstützt und viele numerische Algorithmen zur Lösung von mathematischen Problemen, insbesondere zur Optimierung, bereitstellt. Weiter unterstützt Matlab objektorientierte Programmierung und erlaubt dadurch eine einfache Strukturierung, Modifizierung und Erweiterung der Implementierung. Ein weiterer Vorteil von Matlab ist, dass Matlab mit anderen Programmiersprachen, wie zum Beispiel C oder Fortran, interagieren kann. Das heißt, Matlab kann in C oder Fortran geschriebene Programme aufrufen und C- oder Fortran-Programme können in Matlab geschriebene Programme aufrufen. Die Algorithmen werden als Toolbox bereitgestellt, da dieses ein einfaches Einbinden der Implementierung und der zugehörigen Hilfe ermöglicht.

Im Folgenden wollen wir den Gebrauch der *Experimental Design Toolbox* erläutern. Um die *Experimental Design Toolbox* nutzen zu können, muss das Verzeichnis der *Experimental Design Toolbox* als aktuelles Verzeichnis in Matlab ausgewählt oder einmalig in der Path-Variable von Matlab eingetragen werden. Danach steht die Implementierung inklusive ausführlich kommentiertem Quellcode und integrierter Hilfe zur Verfügung.

3.1 Die Modellfunktion

Um eine Parameterschätzung oder eine optimale Versuchsplanung durchführen zu können, wird die zugehörige Modellfunktion und dessen erste und gegebenfalls zweite Ableitung bezüglich der Parameter benötigt.

Allgemeine Modellfunktion

Der Benutzer muss diese zusammen mit den benötigten Ableitungen bereitstellen, indem er das Interface *model* implementiert. Dieses schreibt die Syntax für die Methoden, welche den Funktionswert und die benötigten Ableitungen berechnen, vor. Da Matlab mit einigen anderen Programmiersprachen interagieren kann, können die Modellfunktion und deren Ableitungen auch in

einer dieser Programmiersprachen bereitgestellt werden. Diese müssen dann nur durch eine Implementierung des *model* Interfaces aufgerufen werden.

Quellcode 1: *model* Interface

```
classdef model < handle
% MODEL represents an interface for a model.

    methods (Abstract)

        M = get_M(this, p, t)
        % GET_M returns the result of the model function.
        %
        % M = MODEL_OBJECT.GET_M(P, T)
        %
        % INPUT:
        %
        %     P: the parameters
        %     T: the measuring point
        %
        % OUTPUT:
        %
        %     M: the result of the model function

        dp_M = get_dp_M(this, p, t)
        % GET_DP_M returns the first derivative of the model function
        % with respect to the parameters P.
        %
        % M = MODEL_OBJECT.GET_DP_M(P, T)
        %
        % INPUT:
        %
        %     P: the parameters
        %     T: the measuring point
        %
        % OUTPUT:
        %
        %     M: the first derivative of the model function
        %         with respect to the parameters P

        dpdp_M = get_dpdp_M(this, p, t)
        % GET_DP_M returns the second derivative of the model function
        % with respect to the parameters P.
        %
        % M = MODEL_OBJECT.GET_DPDP_M(P, T)
        %
        % INPUT:
        %
        %     P: the parameters
        %     T: the measuring point
        %
        % OUTPUT:
        %
        %     M: the second derivative of the model function
        %         with respect to the parameters P

    end
end
```

Explizit gegebene Modellfunktion

Für den Fall, dass die Modellfunktion explizit gegeben ist, beinhaltet die *Experimental Design Toolbox* die Klasse *model_explicite*, welche das *model* Interface implementiert. Bei der Instanziierung muss dem Konstruktor nur die Modellfunktion als symbolische Formel und die Variablen der Parameter und der Messpunkte als symbolische Vektoren übergeben werden. Die Ableitungen werden durch symbolisches Differenzieren mit Hilfe der *Symbolic Math Toolbox* berechnet.

Quellcode 2: *model_explicite* Klasse

```
classdef model_explicit < model
% MODEL_EXPLICITE implements the model interface and provides the first
% and second derivatives with respect to the parameters of an explicitly
% given model function.

    methods (Access = public)

        function this = model_explicit(f_sym, p_sym, t_sym)
        % MODEL_EXPLICITE creates a MODEL_EXPLICITE object.
        %
        % OBJ = MODEL_EXPLICITE(F_SYM, P_SYM, T_SYM)
        %
        % INPUT:
        %
        %     F_SYM: the explicite formula of the model function.
        %            F_SYM can depend on P_SYM and T_SYM.
        %     P_SYM: the variables of the parameters P as symbolic vector
        %     T_SYM: the variables of the measuring points T
        %            as symbolic vector
        %
        % OUTPUT:
        %
        %     OBJ: a MODEL_EXPLICITE object with the passed configurations
            (...)
        end

        function M = get_M(this, p, t)
        % GET_M returns the result of the model function.
            (...)
        end

        function dp_M = get_dp_M(this, p, t)
        % GET_DP_M returns the first derivative of the model function
        % with respect to the parameters P.
            (...)
        end

        function dpdp_M = get_dpdp_M(this, p, t)
        % GET_DP_M returns the second derivative of the model function
        % with respect to the parameters P.
            (...)
        end
    end
end
```

Anfangswertproblem als Modellfunktion

In der Praxis tritt jedoch oft der Fall auf, dass die Modellgleichung nicht explizit, sondern nur implizit als Lösung eines Anfangswertproblems, gegeben ist. Aus diesem Grund beinhaltet die *Experimental Design Toolbox* auch die Klasse *model_ivp*, welche ebenfalls das *model* Interface implementiert. Diese löst beliebige parameterabhängige Anfangswertprobleme und berechnet die für die Ausgleichsrechnung und die optimale Versuchsplanung benötigte Lösung und deren Ableitungen. Dafür werden bei der Instanziierung der *model_ivp* Klasse die gewöhnliche Differentialgleichung und der zugehörige Anfangswert spezifiziert. Sowohl die gewöhnliche Differentialgleichung als auch der Anfangswert können von den gesuchten Parametern abhängen. Die *model_ivp* Klasse hat folgende Gestalt.

Quellcode 3: *model_ivp* Klasse

```
classdef model_ivp < model
% MODEL_IVP implements the model interface and provides the solution of an
% initial value problem and its first and second derivatives with respect
% to the parameters.

    methods (Access = public)

        function this = model_ivp(x_sym, y_sym, p_sym, f_sym, x_span, ↩
            y0_sym)
        % MODEL_IVP creates an MODEL_IVP object.
        %
        % OBJ = MODEL_IVP(X_SYM, Y_SYM, P_SYM, F_SYM, X_SPAN, Y0_SYM)
        %
        % INPUT:
        %
        %     X_SYM: the dependent symbolic variable x
        %     Y_SYM: the independent symbolic variable y
        %     P_SYM: the variables of the parameters p as symbolic vector
        %     F_SYM: the symbolic formula f for the derivative of y with
        %            respect to x. f can depend on x, y and p. It holds
        %            (dy/dx)(x,p) = f(x,y,p).
        %     X_SPAN: a vector specifying the interval of integration,
        %             X_SPAN = [x0, xf]. The solver imposes the initial
        %             conditions at x0, and integrates from x0 to xf.
        %     Y0_SYM: the initial value of y at x0 as symbolic formula.
        %             Y0_SYM can depend on the parameters p. It holds
        %             y(x0,p) = y0(p).
        %
        % OUTPUT:
        %
        %     OBJ: a MODEL_IVP object with the passed configurations
            (...)
        end


        function M = get_M(this, p, t)
        % GET_M returns the solution of the initial value problem.
            (...)
        end
```

```
        function dp_M = get_dp_M(this, p, t)
        % GET_DP_M returns the first derivative with respect to the
        % parameters P of the solution of the initial value problem.
            (...)
        end

        function dpdp_M = get_dpdp_M(this, p, t)
        % GET_DPDP_M returns the second derivative with respect to the
        % parameters P of the solution of the initial value problem.
            (...)
        end
    end
end
```

Bei der Berechnung der Ableitung der als Anfangswertproblem gegebenen Modellfunktion wird sich zu nutze gemacht, dass sich die Integration und Differentiation der Modellfunktion vertauschen lassen, sofern die Modellfunktion hinreichend oft stetig differenzierbar ist.

Die Klasse *model_ivp* berechnet die benötigten Ableitungen der Differentialgleichung und des Anfangswertes durch symbolisches Differenzieren mit Hilfe der *Symbolic Math Toolbox*. Die so entstehenden Anfangswertprobleme werden mit einem in Matlab eingebauten numerischen Löser für Anfangswertprobleme gelöst.

Da die, bei der ersten Ableitung der Modellfunktion entstehenden, Anfangswertprobleme unabhängig voneinander sind, können diese parallel gelöst werden. Entsprechend verhält es sich bei der zweiten Ableitung. Diese Unabhängigkeit macht sich die *model_ivp* Klasse zunutze, indem bei einem Mehrkernprozessor die Anfangswertprobleme mit Hilfe der *Parallel Computing Toolbox* auf die vorhandenen Kerne aufgeteilt und parallel gelöst werden.

3.2 Übergeben der benötigen Daten

Eine der wesentlichen Klassen der *Experimental Design Toolbox* ist die Klasse *experimental_design*. Sie verwaltet die für die Ausgleichsrechnung und optimale Versuchsplanung wichtigen Daten und stellt Methoden zur Parameterschätzung und optimalen Versuchsplanung bereit. Um eine Parameterschätzung oder eine optimale Versuchsplanung durchzuführen, muss die *experimental_design* Klasse instanziiert und die benötigen Informationen mit den zugehörigen *set* Methoden übergeben werden.

Übergabe der Modellfunktion

Zum einen wird die Modellfunktion und deren erste Ableitung bezüglich der Parameter benötigt. Möchte man bei der optimalen Versuchsplanung den wahren Parameter nicht durch einen Punkt, sondern durch ein Gebiet schätzen, um eine bessere Approximation der optimalen Messpunkte zu erhalten, so wird ebenfalls die zweite Ableitung der Modellfunktion bezüglich der Parameter benötigt.

Die Modellfunktion muss, wie vorher beschrieben, durch eine Implementierung des *model* Interfaces bereitgestellt werden. Eine Instanziierung dieser Implementierung muss mit Hilfe der *set_model* Methode an das *experimental_design* Objekt übergeben werden.

Quellcode 4: *set_model* Methode der *experimental_design* Klasse

```
function set_model(this, model)
% SET_MODEL sets the model which will be used for the computations.
%
% EXPERIMENTAL_DESIGN_OBJECT.SET_MODEL(MODEL)
%
% INPUT:
%
%     MODEL: an object whose class implements the MODEL interface
    (...)
end
```

Übergabe der initialen Parameterschätzung

Weiter wird eine Schätzung der wahren Parameter der Modellfunktion benötigt. Diese kann zum Beispiel aus vorherigen Parameterschätzungen oder aus Erfahrungen mit verwandten Prozessen resultieren. Diese Schätzung kann über die Methode *set_inital_parameter_estimation* des *experimental_design* Objekts gesetzt werden.

Quellcode 5: *set_initial_parameter_estimation* Methode der *experimental_design* Klasse

```
function set_initial_parameter_estimation(this, p0)
% SET_INITIAL_PARAMETER_ESTIMATION sets the initial estimation of the
% model parameters.
%
% EXPERIMENTAL_DESIGN_OBJECT.SET_INITIAL_PARAMETER_ESTIMATION(P0)
%
% INPUT:
%
%     P0: the initial estimation of the model parameters
    (...)
end
```

Übergabe von durchgeführten Messungen

Um eine Parameterschätzung durchführen zu können, benötigt man zu guter Letzt noch Messergebnisse inklusive der zugehörigen Messpunkte und den zugehörigen Varianzen der Messfehler. Bereits durchgeführte Messungen können auch bei der optimalen Versuchsplanung zur Gebietsschätzung der wahren Parameter der Modellfunktion genutzt werden.

Die Messpunkte können eine beliebige Form annehmen. So kann zum Beispiel ein Messpunkt eine Zeit oder einen Ort oder beides repräsentieren. Die genaue Repräsentation eines Messpunktes ist für die optimale Versuchsplanung beziehungsweise die Parameterschätzung selbst irrelevant, da die Messpunkte nur dazu verwendet werden, die Modellfunktion und deren Ableitungen auszuwerten. Die Messpunkte werden also nur an die Implementierung des *model*-Interfaces übergeben.

Durchgeführte Messungen können mit der *set_accomplished_measurements* Methode an das *experimental_design* Objekt übergeben werden.

Quellcode 6: *set_accomplished_measurements* Methode der *experimental_design* Klasse

```
function set_accomplished_measurements(this, t_fix, v_fix, eta)
% SET_ACCOMPLISHED_MEASUREMENTS sets the accomplished measurements
% including the measuring points and the associated variances of the
% measuring errors.
%
% EXPERIMENTAL_DESIGN_OBJ.SET_ACCOMPLISHED_MEASUREMENTS(T_FIX, V_FIX, ETA)
%
% INPUT:
%
%     T_FIX: the measuring points of accomplished measurements
%     V_FIX: the variances of the measuring errors associated
%            with these measuring points
%     ETA:   the associated measuring results
%
% T_FIX, V_FIX and ETA must be of equal length.
    (...)
end
```

Übergabe von potenziellen Messungen

Für die optimale Versuchsplanung werden weiter die möglichen Messpunkten und die zugehörigen Varianzen der Messfehler benötigt. Aus diesen werden die optimalen Messpunkte ausgewählt. Die möglichen Messpunkte und die zugehörigen Varianzen können mit der *set_possible_measurements* Methode an das *experimental_design* Objekt übergeben werden.

Quellcode 7: *set_possible_measurements* Methode der *experimental_design* Klasse

```
function set_possible_measurements(this, t_var, v_var)
% SET_POSSIBLE_MEASUREMENTS sets the possible measuring points and the
% associated variances of the measuring errors. These are the measuring
% points for which the quality will be calculated or optimized.
%
% EXPERIMENTAL_DESIGN_OBJECT.SET_POSSIBLE_MEASUREMENTS(T_VAR, V_VAR)
%
% INPUT:
%
%     T_VAR: the measuring points for which the quality will be
%            calculated or optimized
%     V_VAR: the variances of the measuring errors associated with
%            these measuring points
%
% T_VAR and V_VAR must be of equal length.
    (...)
end
```

3.3 Optimale Versuchsplanung und Parameterschätzung

Nachdem die nötigen Daten an das *experimental_design* Objekt übergeben wurden, kann eine optimale Versuchsplanung oder eine Parameterschätzung durchgeführt werden.

Durchführung einer optimalen Versuchsplanung

Eine optimale Versuchsplanung kann mit Hilfe der *get_optimized_weightings* Methode des *experimental_design* Objekts vorgenommen werden. Die Auswahl der Messpunkte wird dabei mit Gewichtungen realisiert. Jeder Messpunkt erhält eine Gewichtung 0 oder 1. Dabei bedeutet 0, dass der Messpunkt nicht gewählt wurde und 1, dass der Messpunkt gewählt wurde.

Die vorgestellten Varianten der optimalen Versuchsplanung laufen auf ein diskretes Minimierungsproblem hinaus. Dieses können wir durch Ausprobieren aller Möglichkeiten direkt lösen oder mit Hilfe der Relaxierung des Minimierungsproblems eine Lösung approximieren.

Ist der direkte Löser ausgewählt, so gibt die *get_optimized_weightings* Methode als Lösung eine ganzzahlige Gewichtung zurück. Wird hingegen das Minimierungsproblem mit Hilfe der Relaxierung gelöst, so wird eine ganzzahlige Approximation der reellen Lösung des relaxierten Minimierungsproblem und die reelle Lösung selbst zurückgegeben.

Der *get_optimized_weightings* Methode können lineare Beschränkungen an die Messpunkte, wie zum Beispiel eine obere und untere Grenze für die Anzahl der zu wählenden Messpunkte, übergeben werden.

Quellcode 8: *get_optimized_weightings* Methode der *experimental_design* Klasse

```
function [w_int, w_real] = get_optimized_weightings(this, min, max)
% GET_OPTIMIZED_WEIGHTINGS returns the optimal weightings of the measuring
% points.
%
% [W_INT, W_REAL] = EXP_DESIGN_OBJECT.GET_OPTIMIZED_WEIGHTINGS(MIN, MAX)
%
% INPUT:
%
%     MIN: minimum number of measuring points allowed (optional, default:
%          number of parameters minus number of accomplished measuring
%          points)
%     MAX: maximum number of measuring points allowed (optional, default:
%          number of variable measuring points)
%
% OUTPUT:
%
%     W_INT:  the optimal integer weighting of the variable measuring
%             points which follow the constraint MIN <= sum(W) <= MAX,
%             if the direct solver is chosen or an approximation otherwise
%     W_REAL: the optimal real weighting of the variable measuring points
%             which follow the constraint MIN <= sum(W) <= MAX,
%             if the direct solver is chosen or [] otherwise
%
%
% [W_INT, W_REAL] = EXP_DESIGN_OBJECT.GET_OPTIMIZED_WEIGHTINGS(A, B)
%
% INPUT:
%
%     A: matrix belonging to the constraint A*W <= B (optional)
%     B: vector belonging to the constraint A*W <= B (optional)
%
% OUTPUT:
%
%     W_INT:  the optimal integer weighting of the variable measuring
%             points which follow the constraint A*W <= B, if the direct
%             solver is chosen or an approximation otherwise
%     W_REAL: the optimal real weighting of the variable measuring points
%             which follow the constraint A*W <= B, if the direct solver
%             is chosen or [] otherwise
%
% The model, an initial estimation of the parameter and the
% possible measurements must have been set via the associated
% SET methods.
    (...)
end
```

Weiter kann für eine beliebige Auswahl von Messpunkten die zu erwartende Qualität der resultierenden Parameterschätzung mit Hilfe der *get_quality* Methode des *experimental_design* Objekts berechnet werden. Dadurch kann

zum Beispiel der Zuwachs der Qualität durch weitere Messungen ermittelt werden. Außerdem kann bestimmt werden, wie stark sich die Qualität durch Auslassen einzelner Messungen verändert.

Quellcode 9: *get_quality* Methode der *experimental_design* Klasse

```
function quality = get_quality(this, w_var)
% GET_QUALITY returns the quality resulting from the passed weightings of
% the measuring points. The smaller the value, the better the quality.
%
% QUALITY = EXPERIMENTAL_DESIGN_OBJECT.GET_QUALITY(W_VAR)
%
% INPUT:
%
%     W_VAR: the weightings of the measuring points
%
% OUTPUT:
%
%     QUALITY: the quality resulting of the weightings W_VAR of the
%              measuring points
%
% The model, an initial estimation of the parameter and the possible
% measurements must have been set via the associated SET methods.
    (...)
end
```

Durchführung einer Parameterschätzung

Eine Parameterschätzung, basierend auf den bisher durchgeführten Messungen, vorzunehmen ist mit Hilfe der *get_parameter_estimation* Methode der *experimental_design* Klasse möglich. Vorher müssen die erforderlichen Daten, wie im Vorfeld beschrieben, an das *experimental_design* Objekt übergeben werden.

Quellcode 10: *get_parameter_estimation* Methode der *experimental_design* Klasse

```
function p = get_parameter_estimation(this)
% GET_PARAMETER_ESTIMATION returns a parameter estimation resulting from
% the weighted least square method applied to the accomplished
% measurements or the initial estimation whether no measurements are
% accomplished.
%
% P = EXPERIMENTAL_DESIGN_OBJECT.GET_PARAMETER_ESTIMATION()
%
% OUTPUT:
%
%     P: returns a parameter estimation resulting from the weighted least
%        square method applied to the accomplished measurements or the
%        initial estimation whether no measurements are accomplished
%
% The model and an initial estimation of the parameter must have been set
% via the associated SET methods.
    (...)
end
```

Lösungsverfahren und Laufzeitoptimierung

Das durch die Relaxierung entstehende kontinuierliche Minimierungsproblem wird mit Hilfe des SQP-Verfahrens gelöst. Eine Implementierung des Verfahrens stellt die *Optimization Toolbox* durch die *fmincon* Funktion bereit. Für Information zum SQP-Verfahren siehe zum Beispiel [Alt02].

Die erste Ableitung der Zielfunktion wird durch die *experimental_design* Klasse in analytischer Form bereitgestellt. Dadurch wird im Vergleich dazu, dass die Ableitung durch Finite Differenzen approximiert wird, ein Großteil der Rechenzeit eingespart. Die Hesse-Matrix der Zielfunktion wird mit Hilfe des BFGS-Updates approximiert.

Das aus der Parameterschätzung resultierende Ausgleichsproblem wird mit Hilfe des Levenberg-Marquardt-Verfahrens gelöst. Matlab stellt mit der Funktion *lsqnonlin* der *Optimization Toolbox* eine Implementierung dieses Verfahrens bereit. Für eine Beschreibung des Levenberg-Marquardt-Verfahrens siehe zum Beispiel [DaRe08].

Bei der Berechnung der Zielfunktion und deren Ableitung treten oft Zwischenergebnisse auf, die wiederverwendet werden können. Dieses nutzt die *experimental_design* Klasse aus und speichert intern wiederverwendbare Ergebnisse. So wird die Ausführungszeit erheblich reduziert. Ähnlich verhält es sich beim Aufruf öffentlicher Methoden. Sofern möglich, werden dort ebenfalls Zwischenergebnisse aus dem vorigen Methodenaufruf genutzt.

Optionen

Die *experimental_design* Klasse bietet viele Möglichkeiten, Einstellungen vorzunehmen. So lässt sich zum Beispiel der Löser, für das bei der optimalen Versuchsplanung entstehende Minimierungsproblem, auswählen. Standardmäßig wird die Lösung des diskreten Minimierungsproblem mit Hilfe der Lösung des relaxierten Minimierungsproblems approximiert. Eine andere Möglichkeit ist es, das diskrete Minimierungsproblem durch Ausprobieren aller Möglichkeiten direkt zu lösen.

Sofern schon genügend Messungen durchgeführt wurden, lässt sich weiter einstellen, ob als Schätzung für die wahren Parameter bei der optimalen Versuchsplanung eine Punktschätzung oder eine Gebietsschätzung verwendet werden soll. Standardmäßig wird eine Gebietsschätzung durchgeführt.

Bei einer Gebietsschätzung lässt sich außerdem das Konfidenzniveau des Konfidenzgebiets festlegen. Ein Wert von 0.95 ist standardmäßig vorgegeben.

Ebenfalls lässt sich einstellen, ob bei einer optimalen Versuchsplanung und schon vorhandenen Messungen automatisch vorher eine Parameterschätzung durchgeführt werden soll. Dadurch ließe sich die Schätzung der wahren Parameter verbessern und damit die optimalen Messpunkte zuverlässiger ermitteln. Diese Option ist standardmäßig deaktiviert um Rechenzeit einzusparen.

Eine weitere Option ist das Gütekriterium, mit dem die Güte der Kovarianzmatrix gemessen wird. Im Standardfall wird die Spur als Gütekriterium genutzt. Jedoch können auch andere Gütekriterien ausgewählt werden und selbst implementierte Gütekriterien genutzt werden.

Zu guter Letzt kann auch eingestellt werden, ob Informationen bezüglich des Fortschritts der laufenden Berechnungen über die Matlab-Konsole ausgegeben werden sollen. Standardmäßig werden keine Informationen angezeigt.

3.4 Hilfe zur Experimental Design Toolbox

Zu der *Experimental Design Toolbox* gehören neben der eigentlichen Implementierung auch eine integrierte Hilfe und ausführliche Kommentare im Quellcode. Die Hilfe wird in Form von HTML-Seiten bereitgestellt. Dieses bringt mehrere Vorteile mit sich.

Der größte Vorteil ist, dass die Hilfe dadurch mit dem in Matlab integrierten Help-Browser angezeigt werden kann. Der Benutzter kann also die Hilfe der *Experimental Design Toolbox* aus Matlab aufrufen und durch diese, analog zu den Hilfen von Matlab oder anderer Toolboxen, mit dem Help-Browser navigieren. Weiter wird eine Volltextsuche innerhalb der Hilfe durch den Help-Browser ermöglicht.

Der Aufbau der Hilfe der *Experimental Design Toolbox* wurde dem der Hilfen von Matlab und anderen Toolboxen nachempfunden. Dadurch muss der Benutzer sich nicht an einen neuen Aufbau gewöhnen. Da die Hilfe für die *Experimental Design Toolbox* als HTML-Seiten zu Verfügung steht, kann diese auf einfache Weise online verfügbar gemacht und mit einem beliebigen Web-Browser betrachtet werden.

Neben der eigentlichen Hilfe zur *Experimental Design Toolbox* steht noch ein ausführlich kommentierter Quellcode bereit. Die Kommentare im Quellcode sind nach dem Matlab self-documenting Standard aufgebaut. Dies ermöglicht es, Matlabs *help* und *doc* Kommandos auch bei der *Experimental Design Toolbox* zu nutzen. Das heißt, man kann sich für die Toolbox, für jede deren Klassen und jede deren Methoden Informationen über die Funktionsweise anzeigen lassen. Das *help* Kommando zeigt dabei die Informationen als Klartext auf der Matlab-Konsole an. Mit dem *doc* Kommando lassen sich die Informationen im HTML-Format über den Help-Browser betrachten.

4 Zusammenfassung

In diesem Artikel haben wir die Optimierungsprobleme der optimalen Versuchsplanung hergeleitet und zugehörige Lösungsverfahren präsentiert. Eine allgemeine Vorgehensweise zur Parameterschätzung mit Hilfe der optimalen Vesuchsplanung wurde im Anschluss aufgezeigt. Eine Implementierung der vorgestellten Verfahren zur optimalen Versuchsplanung und Parameterschätzung als Matlab Toolbox und deren Benutzung wurden abschließend vorgestellt.

Literatur

[AbRaSe09] M. M. E. Abd El-Monsef, E. A. Rady, M. M. Seyam. *Relationships among Several Optimality Criteria.* Interstat, 2009. ISSN 1941-689X.

[Alt02] Walter Alt. *Nichtlineare Optimierung.* Vieweg Verlag, 2002. ISBN 3528 031 93X.

[BaBoKöSc99] Irene Bauer, Hans Georg Bock, Stefan Körkel, Johannes P. Schlöder. *A sequential approach for nonlinear optimum experimental design in DAE systems.* In F. Keil, W. Mackens, H. Voss, and J. Werther (eds.), Scientific Computing in Chemical Engineering II, Volume 2, Springer Verlag, Berlin, Heidelberg, 1999.

[BoVa04] Stephen Boyden, Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004. ISBN 0521 833 787.

[DaRe08] Wolfgang Dahmen, Arnold Reusken. *Numerik für Ingenieure und Naturwissenschaftler.* Springer-Verlag, zweite, korrigierte Auflage, 2008. ISBN 9783 540 764 922.

[Kö02] Stefan Körkel. *Numerische Methoden für Optimale Versuchsplanungsprobleme bei nichtlinearen DAE-Modellen.* Doktorarbeit, Universität Heidelberg, Juli 2002.

[Re11] Joscha Reimer. *Entwicklung einer Toolbox mit Algorithmen zur optimalen Versuchsplanung am Beispiel von Überflutungsszenarien.* Masterarbeit, Christian-Albrechts-Universität zu Kiel, Februar 2011.