TECHNISCHE UNIVERSITÄT GRAZ

Institut für Grundlagen und Theorie der Elektrotechnik

Prof.Dipl.Ing.Dr.techn. Oszkár Bíró

DIPLOMARBEIT

Robuste Optimierung mit Evolutionsstrategien

Vorgelegt von: Markus Murhammer
geboren am: 15. Oktober 1969 in: Gmunden

zum
Erlangen des akademischen Grades

DIPLOMINGENIEUR

(Dipl.-Ing.)

Betreuer: Prof. Dr. Christian Magele

Kurzfassung

In der klassischen Optimierung bestimmt man im Allgemeinen optimale Lösungen in der Annahme, dass die Eingabedaten exakt bekannt sind. In der Praxis ist dies jedoch häufig nicht der Fall, weil Mess-, Rundungs- und Implementationsfehler auftreten können. So können leicht veränderte Eingabedaten die Qualität der nominalen Lösung massiv beeinflussen, und daher existiert die Notwendigkeit eine robuste Lösung generierende Methodik zu entwickeln, die immun gegen Unsicherheiten der Daten ist. Robuste Optimierung (RO), eine computergestützte Methode, versucht diese Unsicherheiten in den Entwurfsprozess miteinzubeziehen. Verschiedene Verfahren Robustheit zu definieren sind möglich. Die Verwendung eines einfachen Modells eines Hyperwürfels in Kombination mit einem »Worst Case« Szenario führt auf nicht total differenzierbare Zielfunktionen. Der Verlust der totalen Differenzierbarkeit limitiert die Anzahl geeigneter Optimierungsalgorithmen. Folglich wird der Fokus auf die einfach implementierbaren Evolutionsstrategien (ES) geleitet. Bedauerlicherweise leiden evolutionäre Algorithmen an einer großen Zahl von Funktionsauswertungen. Werden Robustheitsforderungen in den Optimierungsprozess miteinbezogen erhöht sich die Anzahl der Funktionsaufrufe zusätzlich. Daher müssen effektive Methoden diskutiert werden, die den Rechenaufwand so gering wie möglich halten.

Stichwörter: Messfehler, Datenunsicherheit, Robuste Optimierung (RO), Modell eines Hyperwürfels, Differenzierbarkeit, Evolutionsstrategien (ES), Funktionsauswertungen

Abstract

Classical methods of optimization determine optimal solutions assuming that the input data are exactly known. In real-life applications (RLA) this is usually not the case due to measurement-, rounding- and implementation errors. Thus slightly modified input data can heavily affect the quality of the nominal solution, and so there exists a real need for a methodology to generate a robust solution, which is immunized against the effects of data uncertainty. Robust Optimization (RO), a methodology using computational tools, tries to incorporate these uncertainties into the design process. Different methods of defining robustness are possible. Using a simple hypercube model in combination with a worst case scenario leads to not totally differentiable objective functions. The loss of differentiability limits the number of suitable optimization algorithms. As a consequence the focus is concentrated on simply implementable Evolution Strategies (ES). Unfortunately evolutionary computation suffers from a high number of function evaluations. Embedding robustness measures in the optimization process increases the number of function calls additionally. As a result effective methods keeping the computational cost as small as possible have to be discussed.

Keywords: errors of measurement, data uncertainty, Robust Optimization (RO), hypercube model, differentiability, Evolution Strategies (ES), function evaluations

TU Graz

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Diplomarbeit[1] mit dem Thema

Robuste Optimierung mit Evolutionsstrategien

selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am 22. Februar 2013

Unterschrift (Markus Murhammer)

[1] verfasst unter anderem mit LyX [11, 15, 30, 31], der visuellen WYSIWYM Textverarbeitung für LaTeX.

Danksagung

Die vorliegende Arbeit entstand im Rahmen meiner Diplomarbeit am Institut für Grundlagen und Theorie der Elektrotechnik (IGTE).

Zuallererst möchte ich mich bei meinem Betreuer Prof. Dr. Christian Magele vom *Institut für Grundlagen und Theorie der Elektrotechnik*[2] für die Unterstützung bei der Verfassung meiner Diplomarbeit bedanken.

Mein Dank gilt insbesondere auch Prof. Dr. Wolfgang Herfort vom *Institut für Analysis und Scientific Computing*[3] der TU Wien, der auch bei vielleicht naiven mathematischen Fragestellungen nie aus der Fassung geriet und mir immer persönlich und mit fachlich beratenden Anregungen zur Seite stand.

Ein besonderes Dankeschön gilt auch Frau Astrid Brodtrager vom *Dekanat der Fakultät für Elektrotechnik und Informationstechnik,* die bei diversen Studienplanunstimmigkeiten immer prompt und kompetent Hilfestellung anbot.

Weiters möchte ich mich auch bei Herrn Mag. Franz Spaller für seine persönliche Hilfe und moralische Unterstützung, die er mir zukommen ließ, bedanken.

Aber auch meinem Bruder, Ing. Martin Murhammer, der mit seinen soliden AutoCAD und Rhinoceros Kenntnissen beim Anfertigen von zahlreichen Skizzen und Diagrammen zur Verfügung stand, gebührt Dank.

Besonders bedanken möchte ich mich aber bei meiner Mutter Elisabeth, die auch in den »dunkelsten« Stunden des Studiums nie den Glauben an mich verlor, mich immer wieder aufs Neue motivieren konnte und mir so immer ein unverzichtbarer Rückhalt war.

[2] `www.igte.tugraz.at`
[3] `www.asc.tuwien.ac.at`

Inhaltsverzeichnis

1 Einleitung

1.1 Klassische Optimierung – Ein Beispiel

Der Versuch, eine gestellte Aufgabe in einem bestimmten Sinn »bestmöglich« zu lösen, führt den Ingenieur, unter der Annahme, dass hierzu mathematische Verfahren zur Anwendung gelangen, auf eine Optimierungsaufgabe. Der technische Terminus optimal, der quantitative Messbarkeit voraussetzt, ist dabei strenger als »bestmöglich« und hat die Bedeutung von maximal oder minimal. Die Frage, »bestmöglich« in welchem Sinn, soll anhand eines einfachen Beispieles dargestellt werden.

In einem einführenden Messtechniklabor soll ein konstanter ohmscher Widerstand mittels Spannungs- und Strommessung bestimmt werden. Die drei Messungen erfolgen jeweils mit idealem Volt- und Amperemeter. Dabei ergeben sich nun folgende Ergebnisse:

$$\begin{aligned} &\text{Messung 1:} && U_1 = 15\,\mathrm{V} && I_1 = 3\,\mathrm{A} \\ &\text{Messung 2:} && U_2 = 12\,\mathrm{V} && I_2 = 3\,\mathrm{A} \\ &\text{Messung 3:} && U_3 = 10\,\mathrm{V} && I_3 = 5\,\mathrm{A} \end{aligned}$$

Wie leicht kontrolliert werden kann ergeben alle 3 Messungen unterschiedliche Widerstandswerte ($R_1 = 5\,\Omega$, $R_2 = 4\,\Omega$ und $R_3 = 2\,\Omega$). Gesucht wird der optimale (»bestmögliche«) Schätzwert $\hat{R}^*$ des Widerstandes.

Mathematisch gesehen bedeutet optimal, die Minimierung einer Kosten- oder Zielfunktion $f(\hat{R})$. Es muss also zuerst eine Zielfunktion festgelegt werden. Dazu müssen drei Fehlergrößen beispielsweise in Spannungsrichtung definiert werden:

$$e_1 = 15 - 3\hat{R} \tag{1.1}$$
$$e_2 = 12 - 3\hat{R} \tag{1.2}$$
$$e_3 = 10 - 5\hat{R} \tag{1.3}$$

Aufbauend auf diese Fehlergrößen kann jetzt eine Zielfunktion $f(\hat{R})$ formuliert werden, die unabhängig vom Vorzeichen des Fehlers sein soll. Es bietet sich also

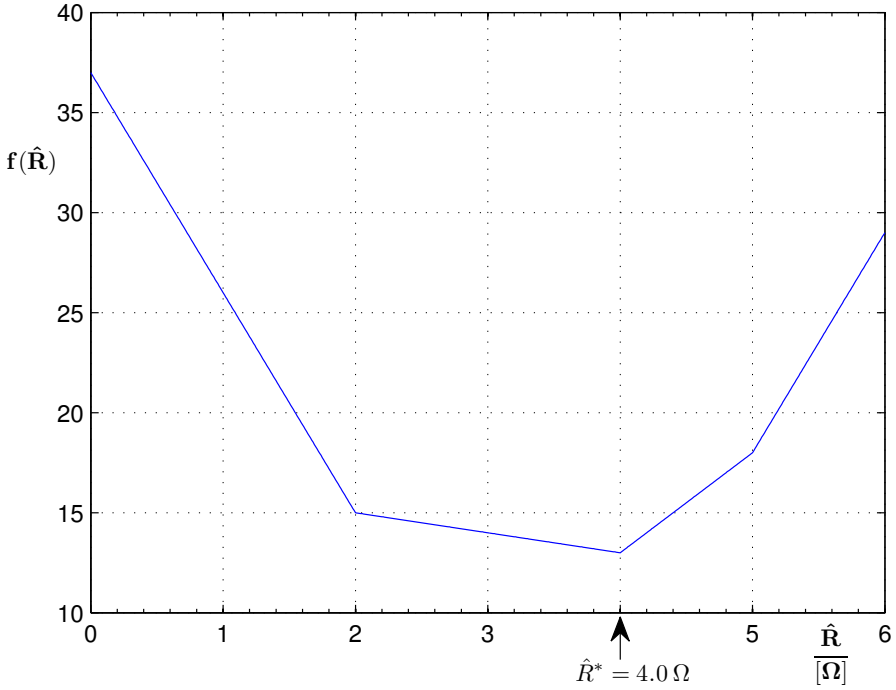


Abb. 1.1: Kostenfunktion $f(\hat{R}) = |e_1| + |e_2| + |e_3|$

$$f(\hat{R}) = |e_1| + |e_2| + |e_3| \tag{1.4}$$

an. Diese Kostenfunktion (Abbildung 1.1) weist ein eindeutiges Minimum auf. Der optimale Schätzwert lautet somit $\hat{R}^* = 4.0\,\Omega$.

Der Vorteil dieser Kostenfunktion besteht darin, dass für die Bestimmung des Minimums nicht differenziert werden muss[1]. Nachteilig ist jedoch die Tatsache, dass die Lösung nicht in geschlossener Form angegeben werden kann.

Eine andere mögliche Zielfunktion lautet

$$f(\hat{R}) = e_1^2 + e_2^2 + e_3^2 \tag{1.5}$$

und besitzt ebenfalls (Abbildung 1.2) ein eindeutiges Minimum, dieses Mal jedoch bei $\hat{R}^* = 3.05\,\Omega$.

Positiv anzumerken ist bei dieser Kostenfunktion, dass die Lösung in geschlossener Form angebbar ist. Diese Methode der Kleinsten Quadrate wurde bereits 1795

[1] Das Minimum liegt genau im Schnittpunkt zweier Geraden.

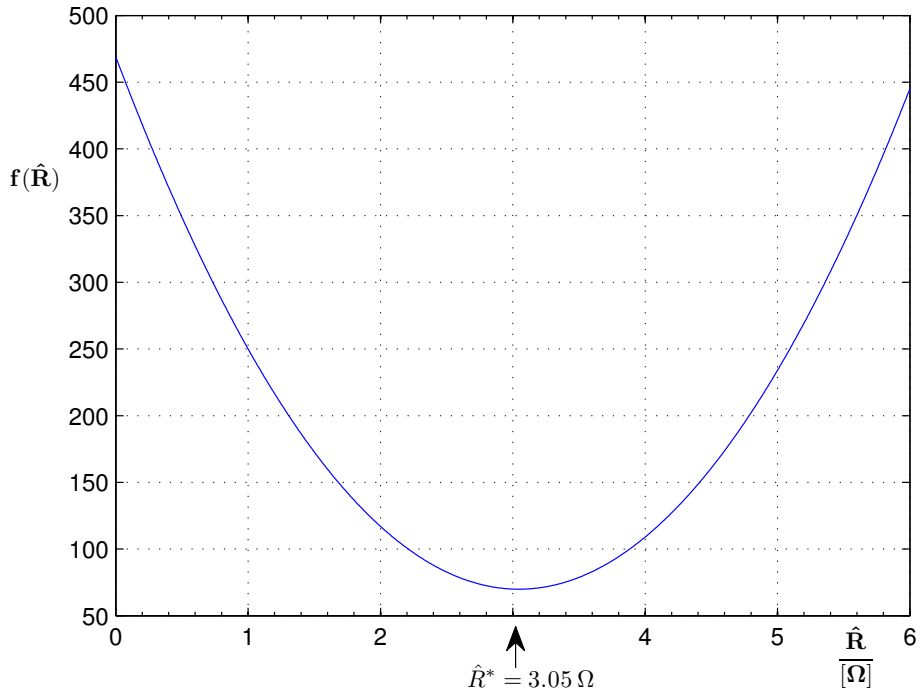


Abb. 1.2: Kostenfunktion $f(\hat{R}) = e_1^2 + e_2^2 + e_3^2$

von C.F. Gauss [10] im Zusammenhang mit der Berechnung von Planetenbahnen eingesetzt.

Gleichung (1.5) kann auch allgemein formuliert werden. Das mathematische Systemmodell lautet

$$\mathbf{y} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{e} \tag{1.6}$$

wobei der Vektor $\mathbf{y}$ die m Messwerte, der Vektor $\mathbf{e}$ die unbekannten Messfehler beinhaltet, und der Vektor $\hat{\mathbf{x}}$ aus den n Systemgrößen besteht, an denen man interessiert ist. Es wird natürlich vorausgesetzt, dass $m > n$ ist. Gleichung (1.6) stellt somit ein überbestimmtes Gleichungssystem dar.

Der Vergleich des Systemmodells (1.6) mit dem Ohmschen Gesetz $U = I \cdot R$ ergibt mit $m = 3$ Messwerten für den Vektor $\mathbf{y^T} = [15\ 12\ 10]$. Die Systemmatrix $\mathbf{A}$ enthält die Stromwerte und schrumpft zu einem Vektor $\mathbf{a^T} = [3\ 3\ 5]$. Die skalare Größe $\hat{x}$ ($n = 1$) entspricht dem gesuchten Schätzwert $\hat{R}$ des Widerstandes.

Die quadratische Zielfunktion für den allgemeinen Fall lautet also

$$f(\hat{\mathbf{x}}) = \mathbf{e^T e} = (\mathbf{y} - \mathbf{A}\hat{\mathbf{x}})^{\mathbf{T}}(\mathbf{y} - \mathbf{A}\hat{\mathbf{x}}). \tag{1.7}$$

Durch Ausmultiplizieren erhält man

$$f(\hat{\mathbf{x}}) = \mathbf{y}^\mathbf{T}\mathbf{y} - 2\,\hat{\mathbf{x}}^\mathbf{T}\mathbf{A}^\mathbf{T}\mathbf{y} + \hat{\mathbf{x}}^\mathbf{T}\mathbf{A}^\mathbf{T}\mathbf{A}\hat{\mathbf{x}}. \tag{1.8}$$

Um das Minimum zu erhalten muss nur mehr der Gradient[2]

$$\nabla f(\hat{\mathbf{x}}) = -2\,\mathbf{A}^\mathbf{T}\mathbf{y} + 2\,\mathbf{A}^\mathbf{T}\mathbf{A}\hat{\mathbf{x}} \tag{1.9}$$

null gesetzt werden. Der optimale Schätzwert lautet

$$\hat{\mathbf{x}}^* = (\mathbf{A}^\mathbf{T}\mathbf{A})^{-1}\mathbf{A}^\mathbf{T}\mathbf{y}. \tag{1.10}$$

Die bei der Lösung (1.10) entstehende $[n \times m]$ Matrix

$$\mathbf{A}^+ = (\mathbf{A}^\mathbf{T}\mathbf{A})^{-1}\mathbf{A}^\mathbf{T} \tag{1.11}$$

wird in der Literatur [18, 19] oft als verallgemeinerte Inverse oder Pseudoinverse der Systemmatrix $\mathbf{A}$ bezeichnet.

Bemerkung: Die Minimierung der Gleichung (1.7) könnte auch numerisch mit Hilfe des Gauß-Newton (GN) Verfahrens [16] elegant gelöst werden.

Für das anfangs formulierte Beispiel gibt es nun zwei verschiedene Ergebnisse:

$$\hat{R}^* = 4.0\,\Omega \quad \text{und} \quad \hat{R}^* = 3.05\,\Omega$$

Welche der beiden Lösungen die optimale Lösung darstellt, kann aber sehr einfach beantwortet werden. Beide Lösungen sind in Bezug auf ihre Kostenfunktionen jeweils optimal. Das heißt, man könnte durch Definition neuer Qualitätsfunktionen und deren anschließender Minimierung weitere andere optimale Lösungen erhalten.

Um es nochmals zu betonen, der Begriff »optimal« muss also stets in Bezug auf eine Qualitätsfunktion betrachtet werden und folglich ist immer zu beachten, ob die Zielfunktion für das zu lösende Optimierungsproblem sinnvoll gewählt wurde. Im Allgemeinen stellt die Methode der kleinsten Fehlerquadrate jedoch eine geeignete und vernünftige Wahl dar.

[2] um exakt zu sein, muss natürlich zusätzlich, als notwendiges Kriterium für ein Minimum, die positive Definitheit der Hesseschen Matrix $\mathbf{G}(\mathbf{x}) = 2\,\mathbf{A}^\mathbf{T}\mathbf{A}$ gefordert werden.

1.2 Robuste Optimierung

In der klassischen Optimierung werden im Allgemeinen optimale Lösungen bestimmt in der Annahme, dass die Eingabedaten und die Parameter der zugrunde liegenden Modelle präzise bekannt sind. Im realen Einsatz ist dies jedoch häufig nicht der Fall. Gründe für solche Störungen, Unsicherheiten und Fehler sind vielfältig.

Der durch ein mathematisches Programm bestimmte optimale Bauteil (z.B. der im vorigen Beispiel ermittelte Widerstand) weist Fertigungstoleranzen auf. Wird diese Komponente anschließend in einer elektronischen Schaltung eingebaut, kann es aufgrund variierender Umgebungsbedingungen (Temperatur, Druck, Fremdfelder etc.) zu geänderten Bauteilwerten kommen. Ein eventuell vorhandener Einbaueinfluss und auftretende »Alterungserscheinungen« sind gegebenenfalls mit einzukalkulieren.

Häufig liegen dem Optimierungsalgorithmus Modelle zugrunde, die nie ein exaktes Abbild des real existierenden Systems sein können. Die beschreibenden Parameter dieser Modelle sind meist mit Unsicherheiten belegt, da sie durch physikalische Messungen bestimmt wurden. Aber auch unter der hypothetischen Annahme eines idealen Modells sind Quantisierungsfehler der Parameter durch das endliche Zahlenformat und Rundungsfehler durch die Arithmetik nicht auszuschließen.

So kann der Fall eintreten, dass eine optimale Lösung mit geänderten Eingabedaten in der Praxis unbrauchbar wird, weil sie sehr weit von der realen Lösung entfernt liegt oder gar notwendige Bedingungen nicht mehr eingehalten werden.

Beispiele für notwendige Bedingungen können bei der Optimierung von Verbrennungskraftmaschinen eine unter keinen Umständen überschreitbare Schadstoffemission sein. Bei elektronischen Schaltungen wäre, in Abhängigkeit von der Fertigungstechnologie, eine Limitierung der maximalen Anzahl von Transistoren, aufgrund begrenzter Chipfläche, denkbar. Andere notwendige Bedingungen können im Bereich der Wirtschaftsmathematik ein nicht überschreitbarer Kosten-, Rohstoff-, Personal- oder Maschineneinsatz sein.

Eine Methode diesem Problem zu begegnen besteht darin, a posteriori eine Empfindlichkeits- bzw. »Sensitivitätsanalyse« (SA) durchzuführen [22]. Dabei erfolgt eine Überprüfung der Empfindlichkeit der optimalen Lösung bezüglich der Variation der Parameter der Zielfunktion und der Nebenbedingungen.

Lokale SA basiert dabei typischerweise auf Differentiationen, zeigt nur die Auswirkung eines einzelnen Parameters, ist einfach zu implementieren, recheneffizient und schnell. Gobale SA hingegen bestimmt mit Hilfe von Monte Carlo Simulationen die Sensitivität für den gesamten Parameterraum und berücksichtigt daher Interaktionen zwischen den Parametern. Nachteilig ist dabei ihr hoher Rechenzeitbedarf und die Tatsache,

dass im Gegensatz zur lokalen Methode die Sensitivität nicht mehr eindeutig einem Parameter zugeordnet werden kann.

Als Kritikpunkt beider Verfahren ist anzuführen, dass im Fall nicht tolerierbarer Schwankungen ein weiterer rechenzeit- und kostenaufwendiger Optimierungslauf notwendig sein kann, der vielleicht wieder zum gleichen Ergebnis führt. Außerdem bleibt unbekannt, ob nicht zusätzlich zur gefundenen optimalen eine weitere aber dafür robuste[3] Lösung existiert.

Den wesentlich vorteilhafteren und flexibleren Ansatz bietet die Robuste Optimierung (RO). Hier werden bereits in den Eingabewerten durch Unsicherheitsmodelle Toleranzen und Störungen berücksichtigt. Gesucht wird eine möglichst gute robuste Lösung, die von der Qualität geringfügig schlechter ist als die optimale Lösung, aber dennoch für alle innerhalb der Toleranzen liegenden Eingabedaten zulässig bleibt.

In der Literatur [1] wird häufig von robusten Algorithmen gesprochen, wenn sie für ein breites Anwendungsspektrum von Optimierungsproblemstellungen zuverlässig arbeiten und robust konvergieren. Dies ist jedoch bei dieser Arbeit über die RO nicht Gegenstand der Betrachtungen.

1.3 Konzeption der Diplomarbeit

Die Arbeit gliedert sich in acht Kapitel und einen Anhang:

- im folgenden 2. Kapitel werden bekannte zweidimensionale Testfunktionen vorgestellt, die hinsichtlich ihrer Komplexität (Anzahl von Minima und Maxima) immer größere Anforderungen an den Optimierungsalgorithmus, bezüglich seiner Fähigkeit auf das globale Optimum zu konvergieren, stellen. Dabei erfolgt eine grobe Unterscheidung in uni- und multimodale Typen.

- Kapitel 3 zeigt eine allgemeine nicht robuste Formulierung des Optimierungsproblems, erläutert kurz die Eigenschaft der Konvexität und gibt Anwort auf die Frage, welche Besonderheit im Zusammenhang mit der zulässigen Lösung zu beachten ist.

- Kapitel 4 zeigt verschiedene Varianten der Definition eines Unsicherheitsbereiches und dessen anschließende mathematische Anwendung auf die Qualitätsfunktion und die Nebenbedingungen. Dadurch wird folglich eine vollständig robuste Formulierung des Optimierungsproblems möglich.

[3] Das Adjektiv robust hat seinen Ursprung im Lateinischen »robustus« und bedeutet fest, stark, widerstandsfähig, unempfindlich [26].

- In Kapitel 5 werden die Auswirkungen der robusten Formulierung auf den Entwurfsraum und die Differenzierbarkeit in der Umgebung der Extremalstellen der Qualitätsfunktion ausgearbeitet. Im Anschluss folgt eine kurze Diskussion über die sich dadurch ergebende Konsequenz für eine zweckmäßige Wahl des Optimierungsalgorithmus.

- Im 6. Kapitel werden verschiedene Ansätze und Methoden vorgestellt das bei der robusten Optimierung auftretende Subproblem effizient und demzufolge mit möglichst geringer Anzahl von zusätzlichen Funktionsauswertungen zu implementieren.

- Das 7. Kapitel präsentiert als wichtigen Vertreter von stochastischen Optimierungsverfahren die Evolutionsstrategien, stellt die für gutes globales Verhalten wichtige Größe des Selektionsdruckes vor und beschreibt verschiedene Selektionsmechanismen und Möglichkeiten der Schrittweitenadaption.

- Kapitel 8 gibt eine Zusammenfassung wesentlicher Aussagen und Erkenntnisse und zeigt mögliche Trends und Perspektiven.

- Im Anhang A werden das Flussdiagramm und die dokumentierten Variablen des Evolutionsalgorithmus, sowie die programmierten MATLAB® -7.9 Skripts angeführt.

2 Analytische Testfunktionen

2.1 Einführung

In diesem Kapitel werden einige bekannte Testfunktionen bzw. Optimierungsprobleme vorgestellt, die wiederholt im Zusammenhang mit genetischen Algorithmen (GA), Evolutionsstrategien (ES), Particle Swarm Optimierung (PSO) und globaler Optimierung erwähnt werden [24, 27].

Ihr Zweck besteht in einer Performanceanalyse des Algorithmus hinsichtlich der Kriterien Rechenzeit, Konvergenzeigenschaften und Qualität des Ergebnisses. Eine wichtige Aufgabe besteht somit darin, Algorithmen untereinander vergleichbar zu machen, andererseits sind sie auch bei der Programmierung hilfreich für die Überprüfung der grundlegenden Funktionalität des Algorithmus.

Von den Eigenschaften ist eine grobe Unterteilung in unimodale und multimodale Typen möglich. Unimodale Funktionen haben nur ein einziges und damit globales Optimum (Minimum), während bei multimodalen Typen mehrere lokale Minima auftreten können und daher höhere Anforderungen an das globale Konvergenzverhalten des Optimierungsalgorithmus gestellt werden.

Zu jeder Funktion wird auch ein Suchraum angegeben, davon abweichende Grenzen bei den Plots dienen nur zur Erhöhung der Übersichtlichkeit. Zu beachten ist weiters, dass bei den Diagrammen immer die zweidimensionalen ($n = 2$) Varianten dargestellt werden, da manche Testfunktionen auch für wesentlich höhere Dimensionen ausgelegt sind.

2.2 Unimodale Typen

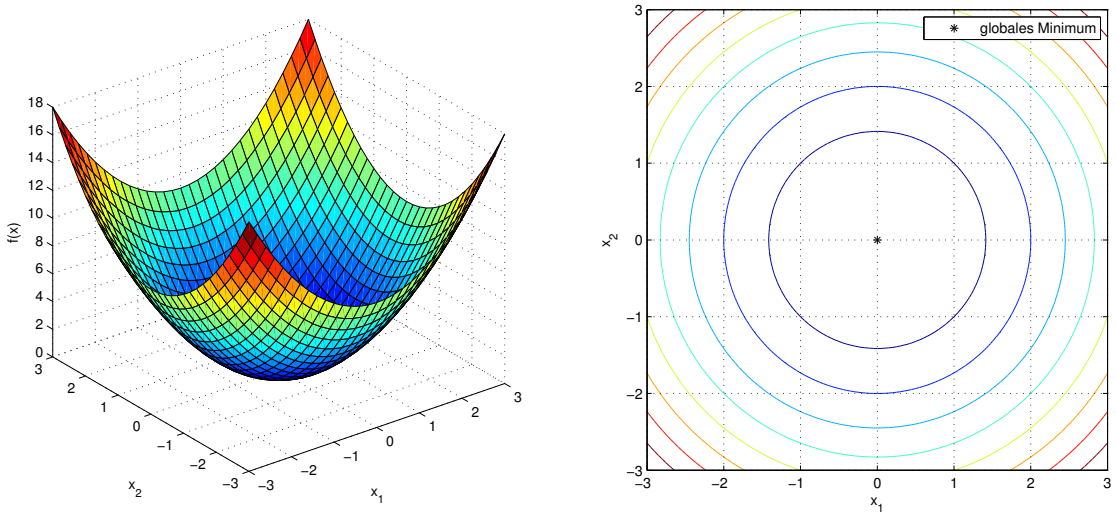
2.2.1 Sphäre Funktion

Die einfachste Funktion ist auch unter »De Jongs Funktion 1« bekannt. Von den Eigenschaften ist sie stetig, konvex[1], symmetrisch und quadratisch.

[1] Die Eigenschaft der Konvexität wird in Abschnitt 3.1 näher erläutert.

$$f_{\text{sphäre}}(\mathbf{x}) = \sum_{i=1}^{n} x_i^2 \qquad \text{mit } x_i \in [\text{-}5.12, 5.12] \tag{2.1}$$

Sie besitzt ein globales Minimum bei $x_i^* = 0$ mit $f(\mathbf{x}^*) = 0$. Das Finden dieses Minimums sollte eigentlich keinen Optimierungsalgorithmus vor Probleme stellen. Infolge ihrer Unkompliziertheit wird sie oft in theoretischen Modellen verwendet.



(a) Funktionsplot (b) Konturlinien

Abb. 2.1: Testfunktion Sphäre

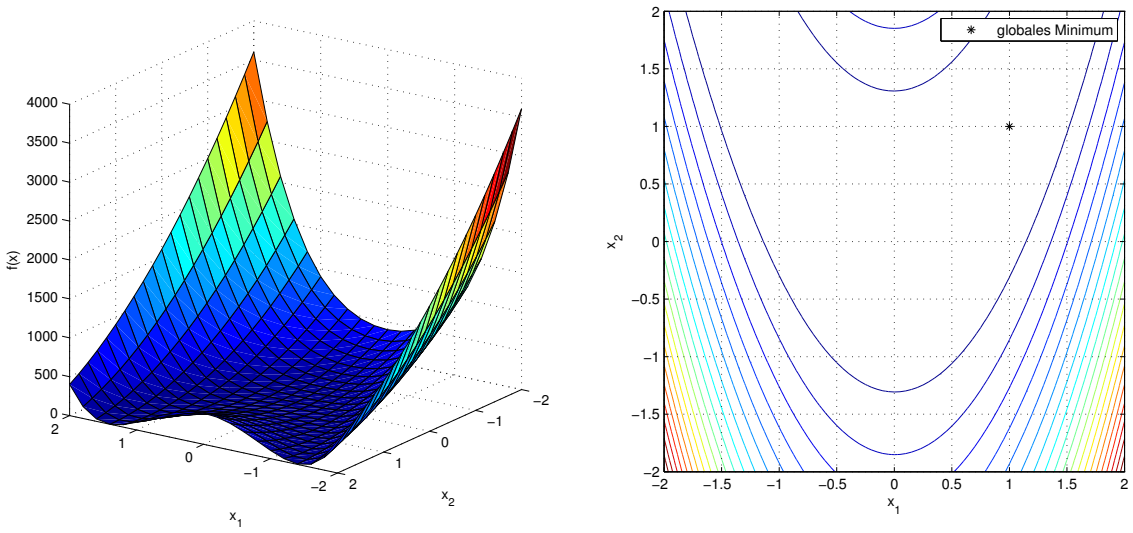
2.2.2 Rosenbrocks Tal

Rosenbrocks Tal, auch bekannt unter »De Jongs Funktion 2« oder »Bananenfunktion«, ist der Klassiker unter den Testfunktionen. Sie ist eine stetige, nicht konvexe und biquadratische Funktion zweier Variablen. Das einzige globale Optimum liegt in einem langen, einer Banane ähnlichem, parabolisch geformten Tal (entlang der Kurve $x_2 = x_1^2$) mit flachem Boden.

Dieses Tal zu finden ist trivial, aber die Konvergenz auf das globale Minimum stellt eine echte Herausforderung dar. Dies ist auch der Grund warum sie häufig zur Beurteilung der Leistungsfähigkeit von Algorithmen Verwendung findet.

$$f_{\text{rosenbrock}}(\mathbf{x}) = \sum_{i=1}^{n-1} 100 \cdot (x_{i+1} - x_i^2)^2 + (1 - x_i)^2 \qquad \text{mit } x_i \in [\text{-}2.048, 2.048] \tag{2.2}$$

Das globale Minimum liegt bei $x_i^* = 1$ mit $f(\mathbf{x}^*) = 0$.



(a) Funktionsplot (b) Höhenlinien

Abb. 2.2: Rosenbrocks Tal

2.3 Multimodale Typen

2.3.1 Testfunktion Markus

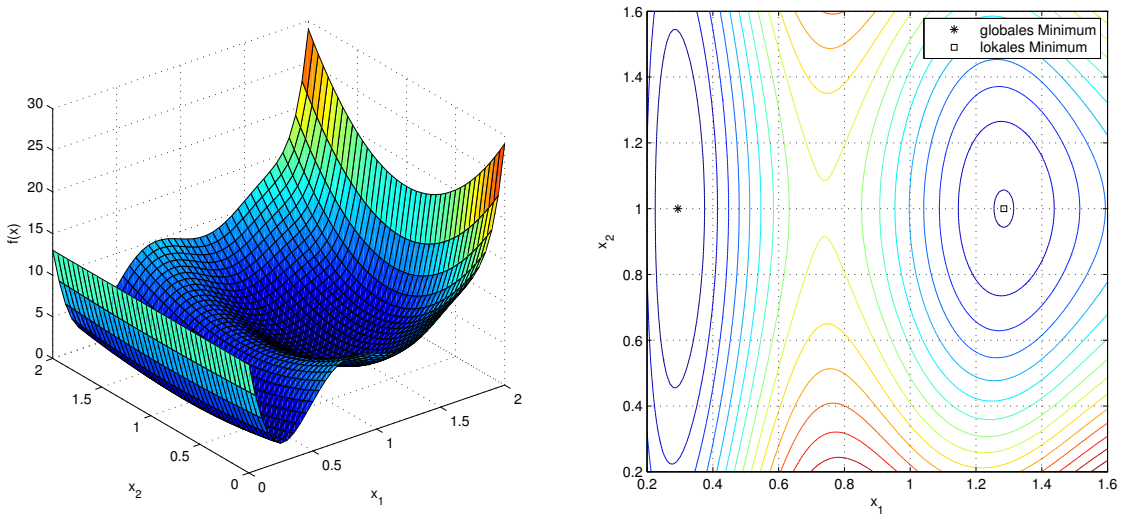
Diese Testfunktion wurde selber entworfen. Sie ist stetig und besteht aus einem nicht monischem Polynom 7. Ordnung, dem ein quadratischer Term überlagert ist.

$$f_{\text{markus}}(\mathbf{x}) = 48.8x_1^7 - 305x_1^6 + 704.3x_1^5 - 681x_1^4 + 139x_1^3 + \cdots \tag{2.3}$$

$$\cdots + 178x_1^2 - 94x_1 + 6x_1 \cdot (x_2 - 1)^2 + 13 \qquad \text{mit } x_i \in [\,0, 2\,] \tag{2.4}$$

Das globale Minimum $\mathbf{x}_1^* = [\,0.294, 1.0\,]$ mit $f(\mathbf{x}_1^*) = 0.553$ liegt in einem steilen Tal, während das lokale Minimum $\mathbf{x}_2^* = [\,1.285, 1.0\,]$ mit $f(\mathbf{x}_2^*) = 1.045$ in einem flacheren Tal mit weniger dicht gedrängten Isolinien liegt (Diagramm 2.3b).

Unter dem Gesichtspunkt von Robustheitsforderungen könnte also letztgenanntes bei der Optimierung bevorzugt werden, dies hängt jedoch von der Größe der tolerierbaren Variationen ab und wird in Abschnitt 5.4 noch intensiver einer Analyse unterzogen.



(a) Funktionsplot (b) Isolinien

Abb. 2.3: Testfunktion MARKUS

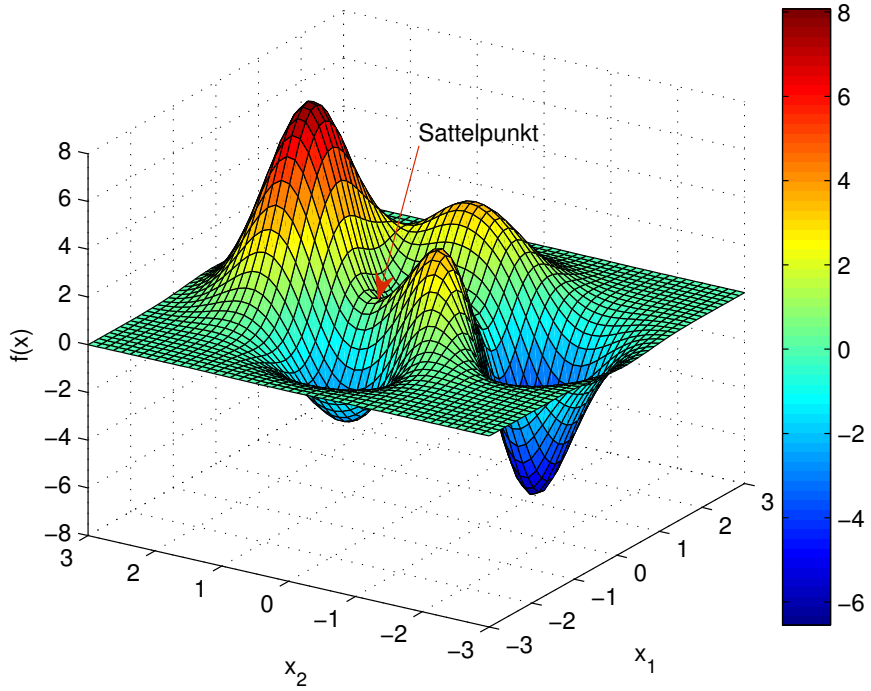
2.3.2 Peaks Funktion

Diese zweidimensionale Funktion ist einer MATLAB® Toolbox entnommen und enthält skalierte Gaußsche Verteilungen.

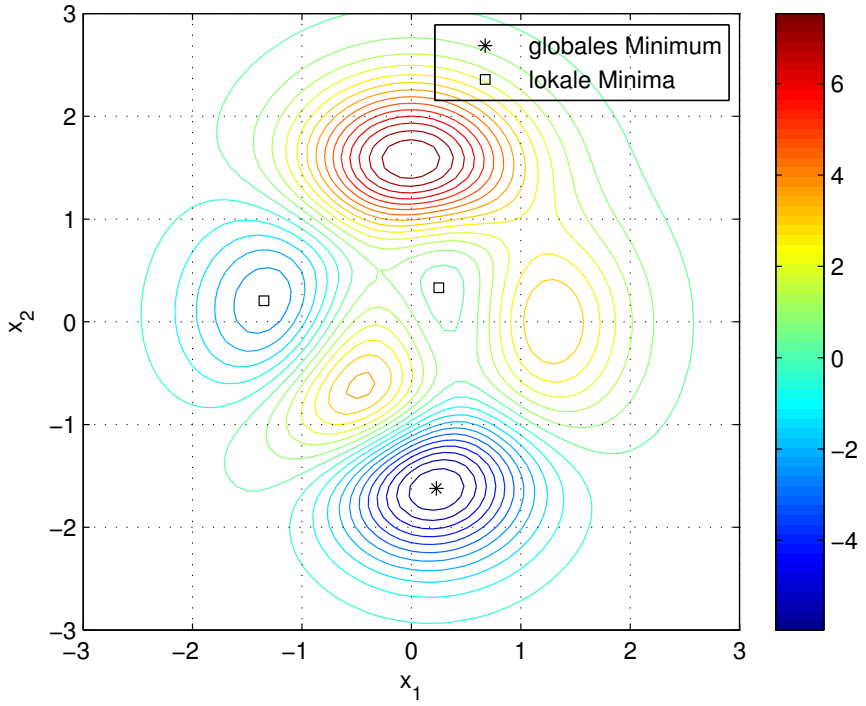
$$f_{\text{peaks}}(\mathbf{x}) = 3 \cdot (1 - x_1)^2 \cdot \mathrm{e}^{(-x_1^2 - (x_2+1)^2)} - 10 \cdot (\frac{x_1}{5} - x_1^3 - x_2^5) \cdot \mathrm{e}^{(-x_1^2 - x_2^2)} - \cdots$$
$$\cdots - \frac{1}{3} \cdot \mathrm{e}^{(-(x_1+1)^2 - x_2^2)} \qquad \text{mit } x_i \in [-3, 3] \tag{2.5}$$

Interessant ist neben ihrer Form aus Bergen und Tälern der mit einem roten Pfeil angedeutete Sattelpunkt im Diagramm 2.4a, also ein Punkt bei dem die Hessesche Matrix $\mathbf{G}(\mathbf{x}) = \nabla^2 f(\mathbf{x})$ indefinit wird.

Ihr globales Minimum liegt bei $\mathbf{x}^* = [0.2283, -1.6255]$ mit $f(\mathbf{x}^*) = -6.551$.



(a) Funktionsplot



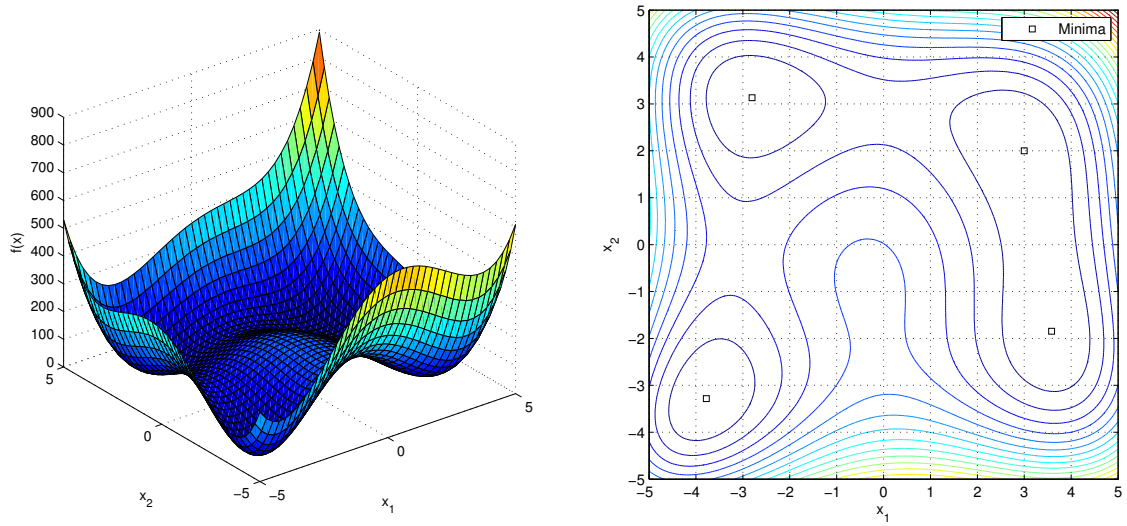
(b) Konturlinien

Abb. 2.4: Testfunktion Peaks

2.3.3 Funktion Himmelblau

$$f_{\text{himmelblau}}(\mathbf{x}) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \qquad \text{mit } x_i \in [\,\text{-}5, 5\,] \tag{2.6}$$

Bemerkenswert ist dabei die Tatsache, dass die vier Minima ($\mathbf{x}_1^* = [\,3.5844, \text{-}1.8481\,]$, $\mathbf{x}_2^* = [\,\text{-}3.7793, \text{-}3.2831\,]$, $\mathbf{x}_3^* = [\,\text{-}2.8051, 3.1313\,]$, $\mathbf{x}_4^* = [\,3.0, 2.0\,]$) alle gleichwertig sind, da alle als Zielfunktionswert $f(\mathbf{x}^*) = 0$ haben. Somit existieren vier unterschiedliche globale Minima, die alle auch analytisch bestimmt werden können.



(a) Funktionsplot (b) Niveaulinien

Abb. 2.5: Testfunktion Himmelblau

2.3.4 Generalisierte Rastrigin Funktion

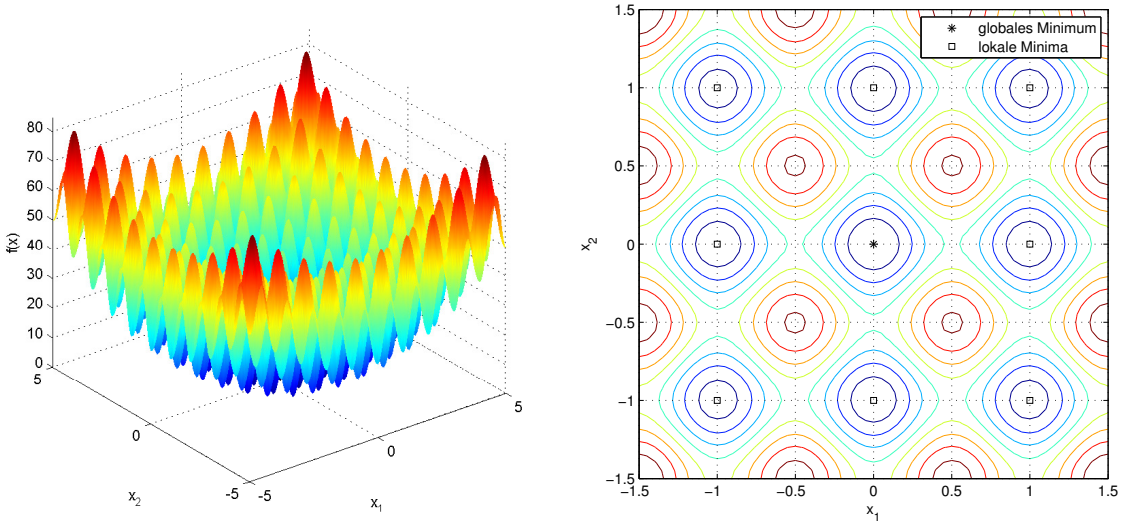
Diese Funktion wurde 1974 von Rastrigin als zweidimensionale Funktion konzipiert und 1991 von Mühlenbein [17] auf beliebige Dimensionen generalisiert. Sie basiert zwar auf der einfachen Sphäre-Funktion (2.1), wird jedoch multimodal durch die zusätzlich aufmodulierte Kosinusschwingung. Wie der Funktionsplot 2.6a zeigt, besitzt die Funktion die Form eines Eierkartons mit vielen gleichverteilten Minima und Maxima.

$$f_{\text{rastrigin}}(\mathbf{x}) = 10 \cdot n + \sum_{i=1}^{n} (x_i^2 - 10 \cdot \cos(2\pi x_i)) \qquad \text{mit } x_i \in [\,\text{-}5.12, 5.12\,] \tag{2.7}$$

Das globale Minimum der symmetrischen, nicht konvexen und höchst multimodalen Funktion liegt bei $x_i^* = 0$ mit $f(\mathbf{x}^*) = 0$.

Je weiter ein lokales Minimum vom Ursprung entfernt ist, desto größer ist sein Funktionswert. Das Diagramm der Höhenlinien 2.6b zeigt eindrucksvoll die alternierende Folge von Maxima und Minima.

Rastrigins Funktion wird oft als Testfunktion für stochastische Algorithmen verwendet, da die vielen lokalen Minima für gradientenbasierte Methoden große Schwierigkeiten beim Auffinden des globalen Minimums darstellen.



(a) Funktionsplot

(b) Höhenlinien

Abb. 2.6: Testfunktion Rastrigin

3 Klassische Optimierung – nicht robuste Methode

Allgemein lautet die Problemstellung für die nicht robuste Optimierung:

Minimiere

$$f(\mathbf{x}) \quad \text{mit} \quad \mathbf{x} \in \mathbb{R}^n \tag{3.1}$$

unter Berücksichtigung von (u.B.v.)

$$\mathbf{h}(\mathbf{x}) \leq 0 \quad \text{mit} \quad \mathbf{h} \in \mathbb{R}^m \tag{3.2}$$

wobei $f(\mathbf{x})$ das zu minimierende Qualitätskriterium (Zielfunktion, Gütefunktion) und Gleichung (3.2) die Ungleichungsnebenbedingungen (UNB), für deren Anzahl m theoretisch keine obere Grenze existiert, in allgemeinster Form definieren. Der Vektor $\mathbf{x} \in \mathbb{R}^n$ bezeichnet die gesuchten n Optimierungs-, Design- oder Entscheidungsvariablen.

Ein Vektor $\mathbf{x}$ (in weiterer Folge auch als Punkt bezeichnet) heißt zulässig, wenn er alle UNB erfüllt. Die Menge aller zulässigen Punkte bilden die zulässige Region S (engl.: *feasible region*).

Die Definition des erlaubten Wertebereiches der Designvariablen erfolgt durch Angabe eines Entwurfsraums $\mathcal{D}$ (engl.: *Design Space*), der durch eine untere $\mathbf{x}_u$ und obere $\mathbf{x}_o$ Grenze spezifiziert wird.

$$\mathcal{D} = \{\mathbf{x} \mid \mathbf{x}_u \leq \mathbf{x} \leq \mathbf{x}_o\} \tag{3.3}$$

Eine Optimierungsaufgabe dieser Art wird als Nichtlineares Programm oder Mathematisches Programm (MP) bezeichnet.

Der Terminus »Programmierung« ist dabei ein Synonym für Optimierung, und wurde ursprünglich (in den 40er und 50er Jahren des vorigen Jahrhunderts) in der Bedeutung

von optimaler Planung benutzt. Er hat hat also nichts mit der Erstellung eines Computerprogrammes (Software) zu tun, wenn auch die Lösung von mathematischen Programmen, für eine üblicherweise große Anzahl n von Entscheidungsvariablen ohne Zuhilfenahme eines Rechners gleichsam unmöglich scheint. Einen interessanten geschichtlichen Überblick stellt [14] dar.

Die in der Literatur häufige Beschränkung der Aufgabenstellung auf Minimierungsprobleme stellt jedoch keine Einschränkung der Allgemeinheit dar, da die Suche nach einem Maximum durch Vorzeichenumkehr der Zielfunktion stets auf ein Minimierungsproblem umformuliert werden kann.

$$\max\ f(\mathbf{x}) = \text{-}\min\left[\text{-}f\left(\mathbf{x}\right)\right] \tag{3.4}$$

Das bedeutet beispielsweise für die multimodale Peaks Testfunktion (siehe Abb. 2.4a), dass bei der Suche des Maximums (also des höchsten Berges) einfach das Minimum der negativen Zielfunktion (2.5) gefunden werden muss.

Ein derartiges Optimierungsproblem, wie es mit Gleichungen (3.1) und (3.2) beschrieben wurde, erfolgreich zu lösen ist besonders dann günstig, wenn die Funktionen $f(\mathbf{x})$ und $\mathbf{h}(\mathbf{x})$ nachfolgend aufgeführte Eigenschaften aufweisen.

3.1 Konvexe Optimierung

Konvexität bezeichnet eine mathematische Eigenschaft, die es ermöglicht eine Klasse von Optimierungsproblemen zu definieren, sodass Optimalitätsbedingungen gleichzeitig auch ausreichende Bedingungen für ein globales Minimum beschreiben.

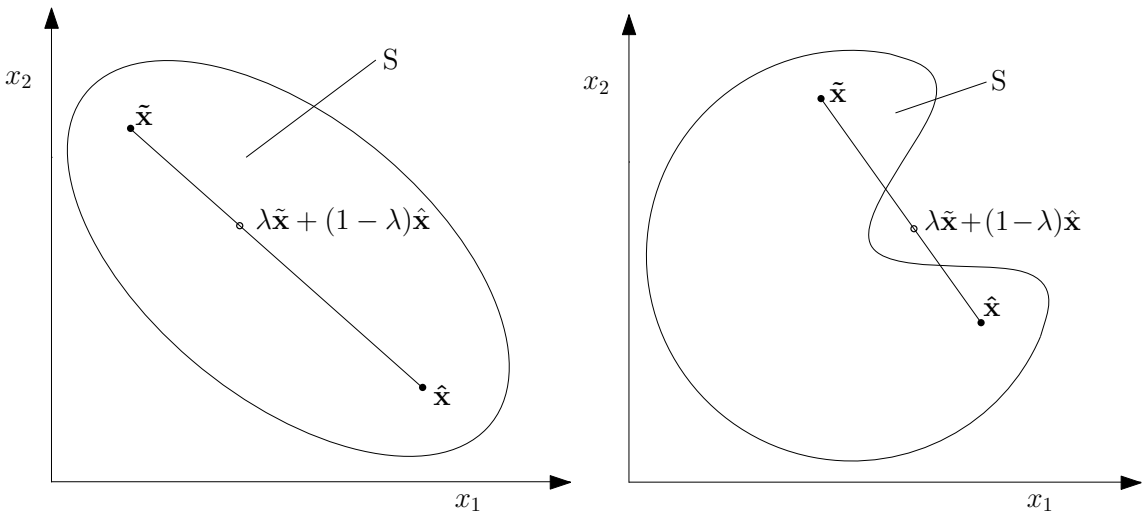
3.1.1 Konvexe Menge

Die Menge S der zulässigen Vektoren, d.h.

$$S = \{\mathbf{x} \mid \mathbf{h}(\mathbf{x}) \leq 0\} \tag{3.5}$$

heißt konvex, wenn die konvexe Kombination zweier beliebiger Punkte $\tilde{\mathbf{x}}$ und $\hat{\mathbf{x}}$ der Menge S wiederum in der Menge S liegt. Mathematisch bedeutet dies

$$\tilde{\mathbf{x}}, \hat{\mathbf{x}} \in S \Rightarrow \lambda\tilde{\mathbf{x}} + (1-\lambda)\hat{\mathbf{x}} \in S \qquad \text{für } \lambda \in [0,1]\,. \tag{3.6}$$



(a) konvexe Menge (b) nicht konvexe Menge

Abb. 3.1: Konvexität

Dies wird in Abbildung 3.1a für den zweidimensionalen Fall anschaulich dargestellt. Liegt also die Verbindungsstrecke (konvexe Kombination der beiden Punkte $\tilde{\mathbf{x}}$ und $\hat{\mathbf{x}}$) wieder in der Menge S, so spricht man von einer konvexen Menge. Vereinfacht ausgedrückt darf die Menge S, wie in Diagramm 3.1b dargestellt, keine »Einbuchtung« aufweisen.

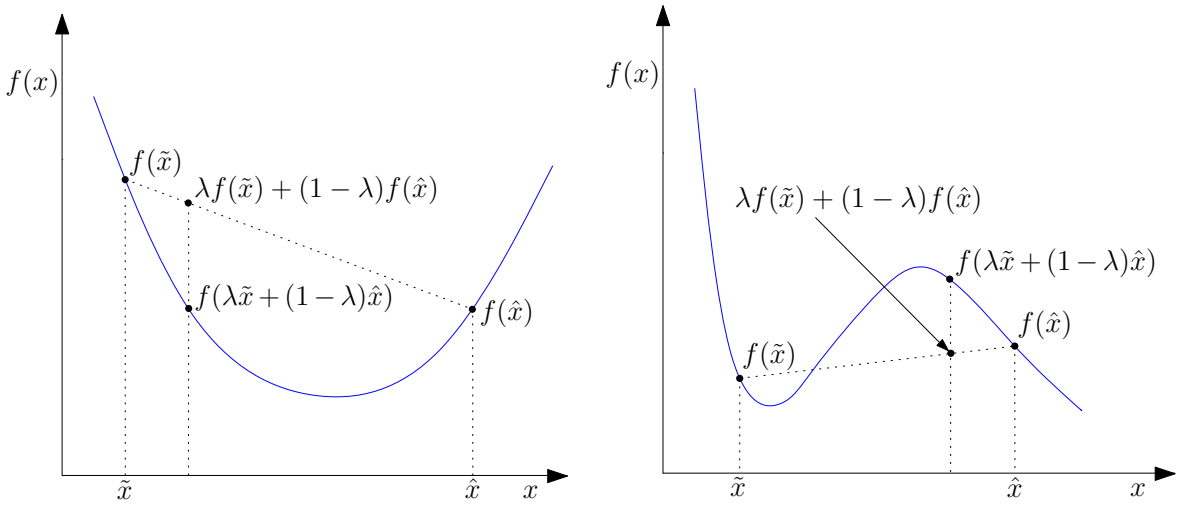
Interessant ist dabei die Tatsache, dass die Schnittmenge zweier konvexer Mengen wieder eine konvexe Menge ergibt. Sind also die durch verschiedene Nebenbedingungen definierten Mengen jeweils für sich konvex, so stellt der Schnitt, also die zulässige Menge S ebenfalls eine konvexe Menge dar.

3.1.2 Konvexe Zielfunktion

Eine stetige Funktion $f(\mathbf{x})$ mit $\mathbf{x} \in S$ und S konvex ist konvex, falls gilt

$$f(\lambda\tilde{\mathbf{x}} + (1-\lambda)\hat{\mathbf{x}}) \leq \lambda f(\tilde{\mathbf{x}}) + (1-\lambda) f(\hat{\mathbf{x}}) \qquad \text{mit } \lambda \in [0,1]\,. \tag{3.7}$$

Das bedeutet der Funktionswert der konvexen Kombination muss immer kleiner oder gleich der konvexen Kombination der Funktionswerte sein oder anders ausgedrückt müssen alle Sekanten, die ein beliebiges Punktepaar verbinden, oberhalb des Graphen liegen. Eine Veranschaulichung einer konvexen Funktion für den univariaten Fall gibt Abbildung 3.2a.



(a) konvexe Funktion (b) nicht konvexe Funktion

Abb. 3.2: Konvexe und nicht konvexe Zielfunktionen

Falls ein allgemeines Optimierungsproblem diesen beiden Einschränkungen (Gleichungen 3.6 und 3.7) genügt, wird es Konvexes Programm genannt. Die Bedeutung konvexer Optimierungsprobleme beruht auf der herausragenden Eigenschaft, dass jedes lokale Minimum auch ein globales Minimum darstellt.

Eine wichtige Klasse von konvexen Optimierungsproblemen stellen die linearen Programme (LP) dar, also Optimierungsaufgaben bei denen sowohl die Zielfunktion als auch die UNB linear bzw. affin von den Designvariablen abhängig sind [6].

Überspitzt könnte man sagen, dass ein Optimierungsproblem heute, aufgrund der Verfügbarkeit exzellenter Software, als gelöst betrachtet werden kann, wenn es gelingt es in Form eines Konvexen Programmes anzugeben.

3.2 Struktur der zulässigen Region

In der vorigen Betrachtung wurde immer angenommen, dass die Menge S der zulässigen Punkte ein einfach zusammenhängendes Gebiet (Abb. 3.3) umfasst. Manchmal besteht die zulässige Region aber auch aus zwei oder mehreren getrennten Subregionen (Abb. 3.4), wodurch sich folgende Schwierigkeit ergeben kann.

Ein typischer deterministischer Optimierungsalgorithmus generiert iterativ eine Serie von kontinuierlich verbesserten Lösungen. Besteht nun die zulässige Region aus zwei Subregionen A und B und startet die Optimierung in B, ist es sehr wahrscheinlich, dass auch eine Lösung in B gefunden wird, während eine vielleicht qualitativ bessere

in Subregion A übersehen wird. Erfreulicherweise kann in den meisten praktischen Optimierungsproblemen diese Schwierigkeit durch sorgfältige Formulierung vermieden werden.

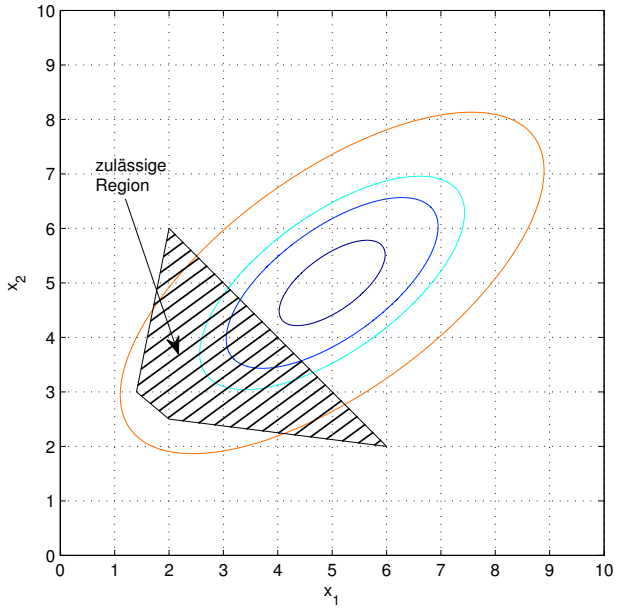


Abb. 3.3: Einfach zusammenhängendes Gebiet

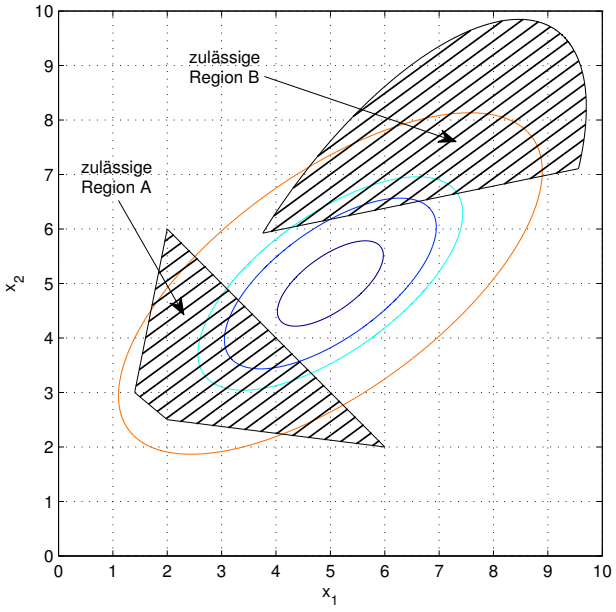


Abb. 3.4: Getrennte zulässige Subregionen

4 Robuste Problemformulierung

— Unsicher zu sein ist unangenehm,
aber sicher zu sein ist törricht. —
Chinesisches Sprichwort

Die traditionelle Formulierung des klassischen nicht robusten Optimierungsproblem, beschrieben durch die Gleichungen (3.1) und (3.2), vernachlässigt Unsicherheiten, Toleranzen und Störungen der Designvariablen.

Diese Darstellung beinhaltet daher keinerlei Information über das Verhalten der Qualitätsfunktion und der Nebenbedingungen in der Umgebung eines nominalen Punktes $\mathbf{x}_0$. In der Folge können Zielfunktionswerte inakzeptable Werte annehmen und auch Nebenbedingungen können beim Auftreten solcher Störungen nicht mehr zuverlässig eingehalten werden.

Um solche Fälle zu vermeiden besteht daher die Notwendigkeit der Definition eines Unsicherheitsbereiches, der sowohl auf die Zielfunktion als auch auf die Nebenbedingungen angewendet wird und somit auf ein vollständig robustes Mathematisches Programm führt.

4.1 Unsicherheitsmodelle

Um Unsicherheiten und Störungen der Optimierungsvariablen zu berücksichtigen wird ein Unsicherheitsbereich

$$U(\mathbf{x}_0) \subset \mathbb{R}^n \tag{4.1}$$

in der Umgebung eines nominalen Punktes $\mathbf{x}_0$ eingeführt, wobei $\mathbf{x}_0$ innerhalb des Entwurfsraumes $\mathcal{D}$ liegt.

Dabei beschreibt $U(\mathbf{x}_0)$ für jeden beliebigen Punkt $\mathbf{x}_0$ alle möglichen Variationen und Abweichungen der Designvariablen. Die Definition des Unsicherheitsbereiches kann dabei auf verschiedene Arten erfolgen.

4.1.1 Hyperwürfel

Dieses Modell stellt immer dann eine sinnvolle Wahl dar, wenn Unsicherheiten voneinander unabhängig und gleich verteilt sind.

$$U(\mathbf{x}_0) = \left\{ \boldsymbol{\xi} \in \mathbb{R}^n \mid \mathbf{x_0} - \boldsymbol{\delta} \leq \boldsymbol{\xi} \leq \mathbf{x_0} + \boldsymbol{\delta} \right\} \tag{4.2}$$

Dabei beinhaltet der Vektor $\boldsymbol{\delta} = [\delta_\mathbf{1} \delta_\mathbf{2} \dots \delta_\mathbf{n}]^\mathrm{T} \in \mathbb{R}^n$ die in alle Richtungen maximal tolerierbaren absoluten Abweichungen der Designvariablen vom nominalen Wert.

Abbildung 4.1 stellt eine grafische Darstellung für den dreidimensionalen ($n = 3$) Fall dar[1].

Manchmal wird die größtmögliche Variation $\boldsymbol{\delta}$ jedoch nicht absolut angegeben, sondern als relativer (prozentualer) Wert von $\mathbf{x}_0$ (z.B. bei Messwerten).

Somit gilt für die i-te Komponente

$$x_{0,i} - \delta_i = x_{0,i} \left(1 - \frac{\delta_i}{x_{0,i}}\right) = x_{0,i} \left(1 - \delta_{i,r}\right) \text{ mit } i = 1, \dots, n \tag{4.3}$$

und für den Unsicherheitsbereich folgt

$$U(\mathbf{x_0}) = \left\{ \boldsymbol{\xi} \in \mathbb{R}^\mathbf{n} \mid x_{0,i} \left(1 - \delta_{i,r}\right) \leq \xi_i \leq x_{0,i} \left(1 + \delta_{i,r}\right) \right\} \text{ mit } i = 1, \dots, n \tag{4.4}$$

wobei $\delta_{i,r}$ die relativen Abweichungen (bezogen auf $x_{0,i}$) vom nominalen Punkt darstellen. Der Hyperwürfel nimmt also für jeden Wert $\mathbf{x}_0$ eine variable Größe an.

[1] Optional wären als Erweiterung des Modells auch unterschiedliche untere $\delta_{i,u}$ und obere $\delta_{i,o}$ Variationen in negativer und positiver Koordinatenrichtung denkbar.

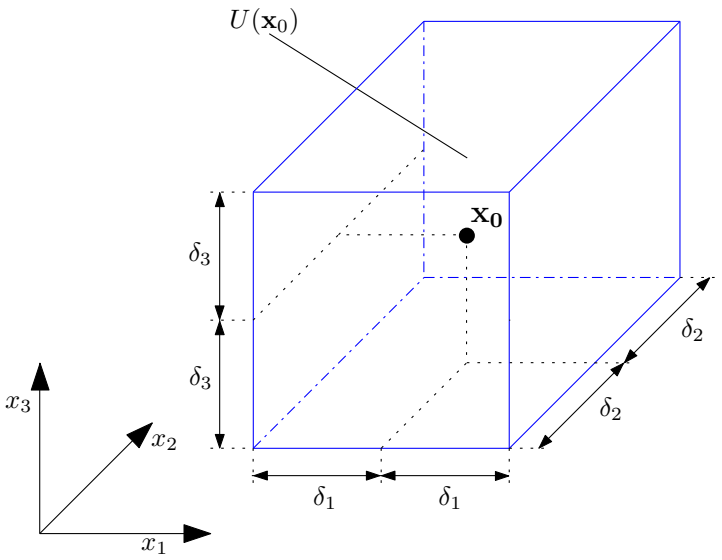


Abb. 4.1: Modell eines Hyperwürfels

4.1.2 Vektornorm

Eine alternative Beschreibung stellt die Vektornorm $\| \mathbf{x} \|_k$ dar, wobei der tiefgestellte Index k die Norm symbolisiert und üblicherweise die Werte 1, 2 oder ∞ annimmt.

$$\| \mathbf{x} \|_1 = |x_1| + |x_2| + \cdots + |x_n| \qquad (l_1 - \text{Norm}) \tag{4.5}$$

$$\| \mathbf{x} \|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} \qquad (\text{Euklidische oder } l_2 - \text{Norm}) \tag{4.6}$$

$$\| \mathbf{x} \|_\infty = \max |x_i| \qquad (l_\infty - \text{Norm}) \tag{4.7}$$

Somit ergibt sich für den Unsicherheitsbereich

$$U(\mathbf{x}_0) = \left\{ \boldsymbol{\xi} \in \mathbb{R}^{\mathbf{n}} \mid \| \boldsymbol{\xi} - \mathbf{x}_0 \|_k \leq \delta \right\}. \tag{4.8}$$

Die spezielle Wahl von $k = 2$ führt im zweidimensionalen Fall ($n = 2$) auf einen Kreis, im dreidimensionalen Fall ($n = 3$) auf eine Kugel und für höhere Dimensionen auf eine Hyperkugel mit Mittelpunkt $\mathbf{x}_0$ und Radius δ. Für $k = \infty$ führt die Methode der Vektornorm wie in Abschnitt 4.1.1 auf einen Hyperwürfel.

Insbesonders die Wahl von $k = 2$ ist sinnvoll, wenn die Verteilungsdichtefunktion der Unsicherheiten bekannt ist.

4.1.3 Elliptisches Modell

Falls die Störungen der Designvariablen von stochastischer Natur sind und unkorrelierte gaussverteilte Unsicherheiten auftreten, stellt ein elliptisches Modell [3] eine vernünftige Wahl dar.

Mit der Diagonalmatrix der Unsicherheiten $\mathbf{Q} = \mathrm{diag}(\delta_i)$ kann das Hyperellipsoid beschrieben werden durch

$$U(\mathbf{x}_0) = \left\{ \boldsymbol{\xi} \in \mathbb{R}^n \mid (\boldsymbol{\xi} - \mathbf{x}_0)^T (\mathbf{Q}^{-1})^2 (\boldsymbol{\xi} - \mathbf{x}_0) \leq 1 \right\}. \tag{4.9}$$

Eine Darstellung für den zweidimensionalen ($n = 2$) Fall gibt Abb. 4.2.

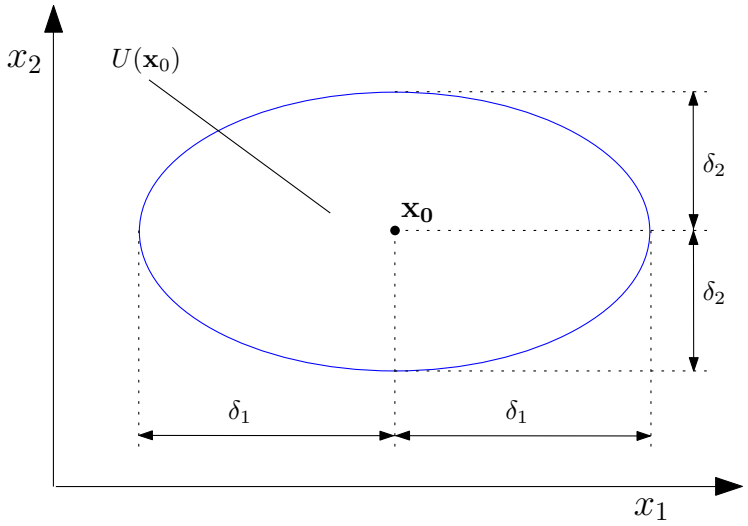


Abb. 4.2: Elliptisches Modell

Natürlich wäre in Unsicherheitsmodellen auch die Berücksichtigung anderer Parameter als der Designvariablen denkbar. Dann muss das Modell um diese durch den Vektor $\boldsymbol{\Theta}$ spezifizierten Parameter zu einem erweiterten Unsicherheitsbereich $U(\mathbf{x}, \boldsymbol{\Theta})$ ergänzt werden [25].

Das Modell des Hyperwürfels (Gleichung 4.2) kann benutzt werden, wenn nur wenig Information über das stochastische Verhalten der Variationen bekannt ist. Daher bildet dieses Modell, dessen Anwendung zu einfachen und effektiven Lösungsstrategien der robusten Optimierung führt, die Grundlage für alle weiteren Betrachtungen in dieser Arbeit.

Die Forderung nach der robusten Lösung eines Optimierungsproblems mit Nebenbedingungen muss nun zwei Ansprüche erfüllen. Einerseits muss die Qualitätsfunktion Robustheitsforderungen genügen (Robuste Zielfunktion) und andererseits muss gewährleistet sein, dass auch die Nebenbedingungen (Robustheit der zulässigen Lösung) eingehalten und nicht verletzt werden.

4.2 Robuste Zielfunktion

Das Ziel bei der robusten Formulierung der Qualitätsfunktion muss die Unempfindlichkeit der Designvariablen gegenüber Störungen und Abweichungen sein.

Es existieren verschiedene Möglichkeiten Robustheit zu quantifizieren, wobei die Entscheidung, welche der nachfolgend dargestellten Methoden zur Anwendung gelangt, stark von der physikalischen Eigenschaft der Optimierungsvariablen und Zielfunktion abhängt.

4.2.1 Worst Case Methode

Hier wird für jeden beliebigen Punkt $\mathbf{x}$ innerhalb des Entwurfsraum $\mathcal{D}$ die Umgebung $U(\mathbf{x})$ analysiert und dabei derjenige mit dem schlechtesten (engl.: *worst*), also dem größten Funktionswert herangezogen.

$$\max_{\boldsymbol{\xi} \in U(\mathbf{x})} \{f(\boldsymbol{\xi})\} \rightarrow \min \tag{4.10}$$

Diese Definition führt auf ein Minimax-Problem, dessen Lösung von den beiden russischen Mathematikern Demjanov und Malozemov in [7] einer Analyse unterzogen wurde.

4.2.2 Mittelwertbildung

Bei dieser Methode wird der Mittelwert aller Funktionswerte innerhalb des Unsicherheitsbereiches minimiert [28].

$$\frac{\int_{U(\mathbf{x})} f(\boldsymbol{\xi}) \, \mathrm{d}\boldsymbol{\xi}}{\int_{U(\mathbf{x})} \mathrm{d}\boldsymbol{\xi}} \rightarrow \min \tag{4.11}$$

Für den einfachen eindimensionalen ($n = 1$) Fall einer Qualitätsfunktion bedeutet das die Bestimmung der Fläche unter der Kurve innerhalb des Unsicherheitsintervalles, und diese Fläche durch ein flächengleiches Rechteck zu ersetzen. Die ermittelte Höhe des Rechtecks (in Ordinatenrichtung) entspricht dann dem gesuchten Funktionswert an der jeweiligen Stelle.

In den meisten praktischen Anwendungsfällen würde die Bestimmung der beiden Integrale, ob jetzt analytisch oder numerisch, einen sehr hohen zeitlichen Rechenaufwand bedeuten, sodass üblicherweise eine Approximation des Mittelwertes durch eine Diskretisierung des Unsicherheitsbereiches herangezogen wird.

4.2.3 Bikriterielle Methode

Dieses Verfahren stellt eine Kombination aus Mittelwertbildung und Berechnung der Standardabweichung der originalen Zielfunktion dar [32]. Dies führt auf ein bikriterielles, also aus zwei verschiedenen Zielfunktionen bestehendes Optimierungsproblem.

$$\begin{pmatrix} \mu_f \{f(\boldsymbol{\xi})\} \\ \sigma_f \{f(\boldsymbol{\xi})\} \end{pmatrix} \rightarrow \min \tag{4.12}$$

Probleme bereitet, wie bei allen multikriteriellen Optimierungsproblemen die Tatsache, den besten Kompromiss für sich gegenseitig widersprechende Anforderungen der Qualitätskriterien zu finden.

Ein Lösungsansatz besteht darin, die beiden Zielfunktionen quasi zu einem skalarem Kriterium zu »verschmelzen«. Eine einfache Methode besteht in der Gewichtungsmethode bei der versucht wird eine konvexe Kombination der beiden Zielfunktionen zu minimieren.

$$\alpha\mu_f + (1-\alpha)\sigma_f \rightarrow \min \qquad \text{mit } 0 \leq \alpha \leq 1 \tag{4.13}$$

Dabei stellt α einen vom Anwender festzulegenden Gewichtungsterm dar, während μ_f und σ_f den Mittelwert und die Standardabweichung der originalen Zielfunktion bezeichnen.

Eine Möglichkeit Mittelwert und Standardabweichung zu berechnen, besteht in der Anwendung der Verteilungsdichtefunktion. Falls diese seltenerweise bekannt wäre, würde ihr Einsatz aber einen sehr hohen rechenintensiven Zeitaufwand bedeuten, sodass üblicherweise eine Taylorreihenentwicklung zum Einsatz gelangt [20].

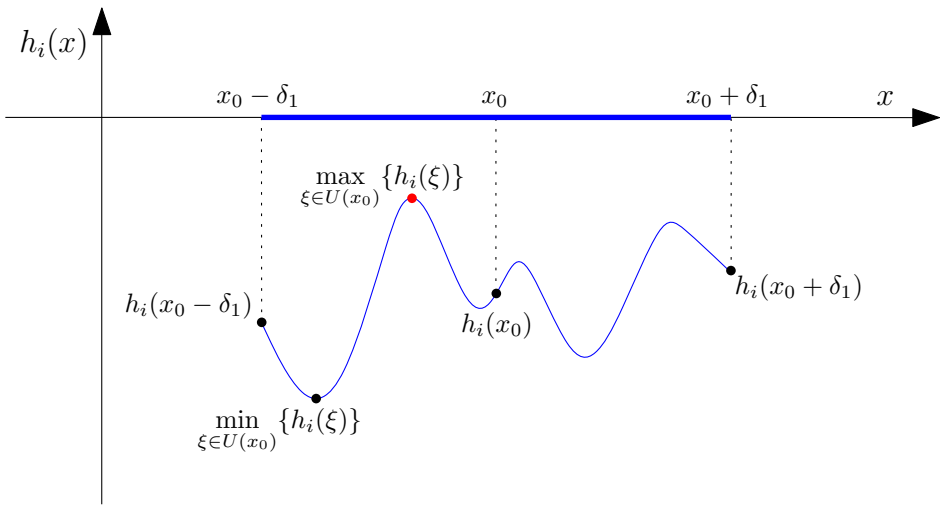


Abb. 4.3: Möglicher Verlauf der UNB $h_i(x)$ in der Umgebung $U(x_0)$

4.3 Robustheit der zulässigen Lösung

Für jeden Punkt $\mathbf{x}$ aus dem zulässigen Bereich muss sichergestellt sein, dass auch seine Umgebung $U(\mathbf{x})$ zulässig bleibt.

Aus der Ungleichungsnebenbedingung

$$\mathbf{h}(\mathbf{x}) \leq 0 \tag{4.14}$$

wird im robusten Fall

$$\mathbf{h}(\boldsymbol{\xi}) \leq 0 \qquad \text{für } \boldsymbol{\xi} \in U(\mathbf{x}) \tag{4.15}$$

Gleichung (4.15) kann auch umformuliert werden, wenn man die Tatsache bedenkt, dass alle Werte von $\mathbf{h}(\mathbf{x})$ in $U(\mathbf{x})$ kleiner oder gleich null sind, wenn der maximale (schlechteste) Wert von $\mathbf{h}(\mathbf{x})$ in $U(\mathbf{x})$ kleiner oder gleich null ist.

Dies wird für die Ungleichungsnebenbedingung $h_i(x)$ anschaulich in Grafik 4.3 für den univariaten Fall dargestellt, wobei in diesem einfachen Fall, der Hyperwürfel zu einer Strecke schrumpft. (Beachte blaues Intervall auf der Abszisse in Abbildung 4.3.)

Somit gilt

$$\max_{\boldsymbol{\xi} \in U(\mathbf{x})} (\mathbf{h}(\boldsymbol{\xi})) \leq 0 \tag{4.16}$$

Gleichung (4.16) zeigt also, dass auch für die Ungleichungsnebenbedingungen quasi ein »Worst Case« Szenario zur Anwendung kommt und daher durch elegante Kombination mit der Worst Case Methode der robusten Zielfunktion (Abschnitt 4.2.1) ein vollständig robustes Mathematisches Programm formuliert werden kann.

4.4 Robustes Mathematisches Programm

Die Kombination der beiden Gleichungen (4.10) und (4.16) führt mit

$$\max_{\boldsymbol{\xi} \in U(\mathbf{x})} f(\boldsymbol{\xi}) \rightarrow \min \tag{4.17}$$

u.B.v.

$$\max_{\boldsymbol{\xi} \in U(\mathbf{x})} (\mathbf{h}(\boldsymbol{\xi})) \leq 0 \tag{4.18}$$

auf ein robustes Mathematisches Programm.

Für die weiteren Betrachtungen wird der Fokus jedoch ausschließlich auf die Robustheit der Zielfunktion (4.17) gelegt, denn durch die mathematisch elegante Anwendung der beiden »Worst Case« Szenarien gelten Eigenschaften der robusten Qualitätsfunktion in analoger Weise auch für die Nebenbedingungen.

Die Berücksichtigung der Robustheit der zulässigen Lösung (4.18) kann mit den üblichen und bekannten Methoden (z.B. Penalty- bzw. Barrier-Verfahren [8, 9]) durchgeführt werden, sodass jetzt nur mehr, ohne Verlust der Allgemeinheit, nicht restringierte Optimierungsprobleme, also Probleme ohne Ungleichungsnebenbedingungen betrachtet werden.

5 Effekte der Robustheitsforderung

Alle nachfolgend angeführten Überlegungen bezüglich Robustheit gelten für das im vorigen Kapitel vorgestellte Unsicherheitsmodell eines Hyperwürfels (4.1.1) in Kombination mit der »Worst Case« Methode (4.2.1) der Zielfunktion.

Für die mathematische Analyse der Auswirkungen der robusten Formulierung wird exemplarisch die in Abschnitt 2.3.1 entwickelte Testfunktion »Markus« herangezogen. Dabei wir zur Vereinfachung die Kurve mit der Ebene $x_2 = 1.0$ (Aufrissebene) geschnitten und somit nur der eindimensionale Fall betrachtet.

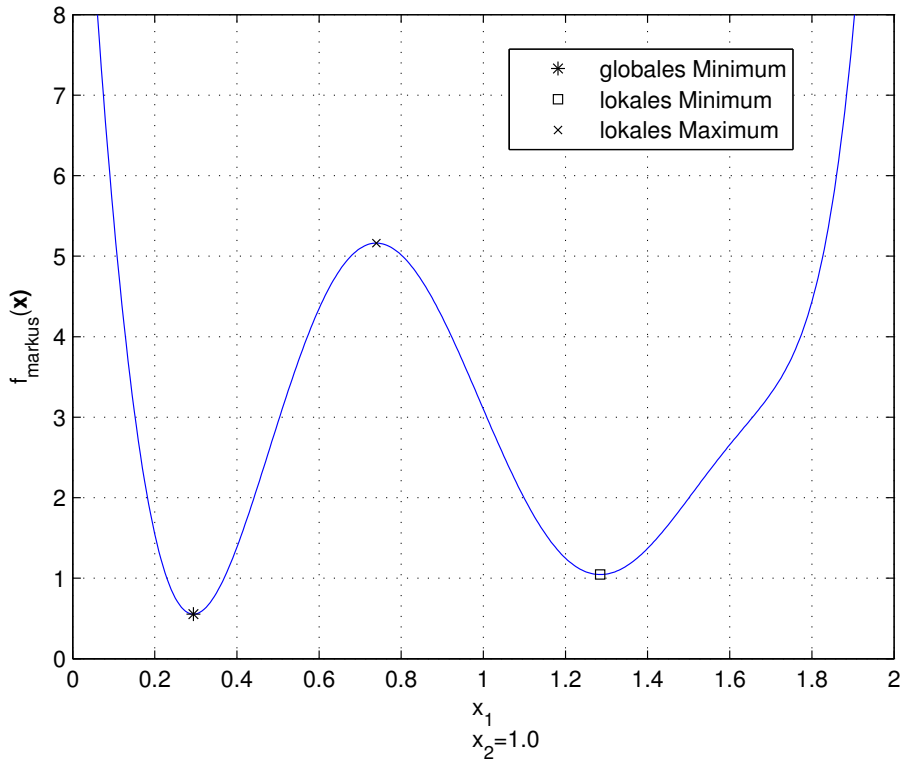


Abb. 5.1: Nicht robuster Verlauf von $f_{\text{markus}}(\mathbf{x})$

Mit der Definition

$$f(x_1) = f_{\text{markus}}(\mathbf{x})\Big|_{x_2=1.0} \tag{5.1}$$

folgen durch Nullsetzen[1] [2] der 1. Ableitung

$$\frac{\partial f(x_1)}{\partial x_1} = 341.6x_1^6 - 1830x_1^5 + 3521.5x_1^4 - 2724x_1^3 + 417x_1^2 + 356x_1 - 94 \qquad (5.2)$$

die in Tabelle 5.1 zusammengefassten Extrema der nicht robusten Funktion $f(\mathbf{x})$.

Tab. 5.1: Extremalstellen von $f(\mathbf{x})$

x_1	x_2	$f(\mathbf{x})$	Typ
0.29402253	1.00	0.55334336	globales Minimum
0.73940878	1.00	5.16184971	lokales Maximum
1.28459531	1.00	1.04505427	lokales Minimum

5.1 Adaption des Entwurfsraumes

Um sicherzustellen, dass Punkte stets innerhalb des Entwurfsraumes $\mathcal{D}$ liegen, muss dieser natürlich angepasst und infolgedessen verkleinert werden.

$$\mathbf{x}_u + \boldsymbol{\delta} \leq \mathbf{x} \leq \mathbf{x}_o - \boldsymbol{\delta} \qquad (5.3)$$

Durch die Einbeziehung der maximal tolerierbaren Variationen $\boldsymbol{\delta}$ in den Entwurfsraum $\mathcal{D}$ bleibt gewährleistet, dass der Unsicherheitsbereich $U(\mathbf{x})$ auch an den Intervallgrenzen von $\mathcal{D}$ stets innerhalb des Entwurfsraumes bleibt und folglich nie über die Grenzen $\mathbf{x}_u$ und $\mathbf{x}_o$ »hinausragt«.

5.2 Konstruktion einer robusten Zielfunktion

Die Herleitung einer Konstruktionsvorschrift für die robuste Qualitätsfunktion

$$\gamma(\mathbf{x}) = \max_{\boldsymbol{\xi} \in U(\mathbf{x})} f(\boldsymbol{\xi}) \qquad (5.4)$$

[1] Auf die Berechnung der 2. Ableitung (Krümmungsverhalten) zur Bestimmung des Typs der Extremalstellen wurde aufgrund der Kenntnis des Verlaufs von $f(\mathbf{x})$ verzichtet.

[2] Die Bestimmung von Polynomnullstellen erfolgte mit der Funktion **solve** aus der Symbolic Toolbox von MATLAB®.

gestaltet sich prinzipiell sehr einfach. Es muss für jeden beliebigen Punkt $\mathbf{x}_0$ die Umgebung $U(\mathbf{x}_0)$ einer Analyse unterzogen werden, um anschließend beim »Worst Case« Verfahren den größten (schlechtesten) Zielfunktionswert innerhalb U zu wählen.

Grafisch kann dies für den einfachen univariaten Fall leicht umgesetzt werden, wenn die beiden um jeweils δ_1 verschobenen Kurven in Abbildung 5.2 betrachtet werden. Dabei bezeichnet $f(x_1 + \delta_1)$ die um δ_1 nach links verschobene Kurve und $f(x_1 - \delta_1)$ die um δ_1 nach rechts verschobene Kurve, die beide für sich weiterhin natürlich vollständig differenzierbar bleiben.

Die resultierende robuste Kurve $\gamma(x_1)$ entsteht nun durch Auswahl und anschließendes Aneinanderfügen entsprechender Kurvenabschnitte gemäß der bildhaften Devise »Man reitet oben entlang«.

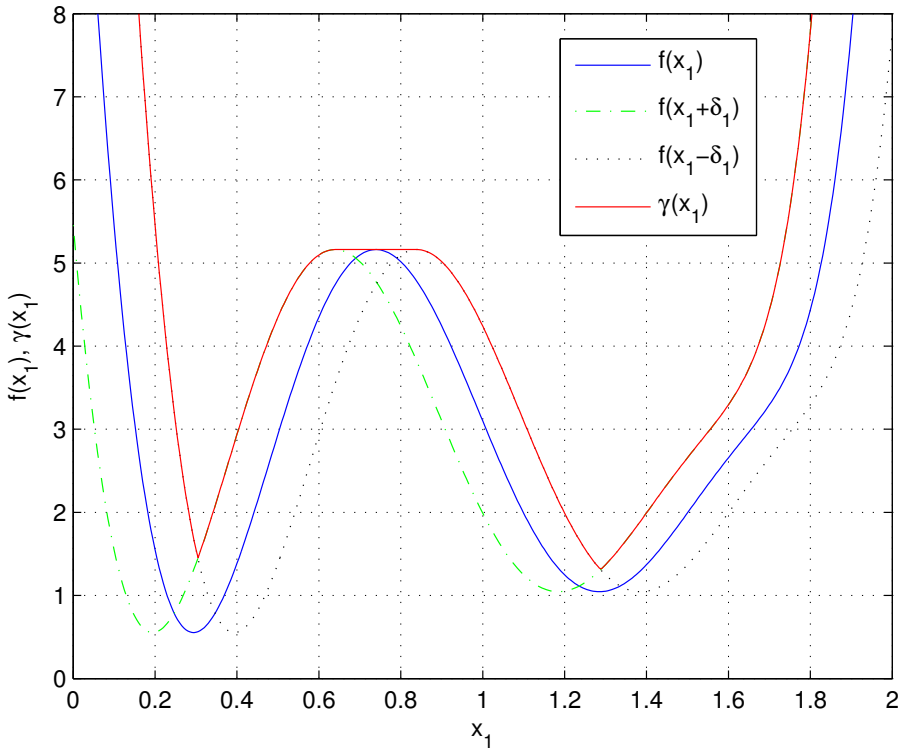


Abb. 5.2: Grafische Konstruktion von $\gamma(x_1)$

5.3 Eigenschaften der robusten Qualitätsfunktion

Bestimmte Eigenschaften der robusten Zielfunktion $\gamma(\mathbf{x})$ können vom ursprünglichen Verhalten der Funktion abweichen. Mit der in Gleichung (5.4) eingeführten Definition

folgt

$$\gamma(\mathbf{x}) \geq f(\mathbf{x}). \tag{5.5}$$

Das bedeutet die robuste Zielfunktion $\gamma(\mathbf{x})$ ist immer größer oder höchstens gleich der nicht robusten Zielfunktion $f(\mathbf{x})$. Sie wird in der Regel also »gehoben«. (Vergleiche Verlauf von $\gamma(x_1)$ und $f(x_1)$ in Diagramm 5.2)

Weiters gilt, wenn $f(\mathbf{x})$ in $\mathbb{R}^n$ kontinuierlich ist, dann bleibt auch $\gamma(\mathbf{x})$ in $\mathbb{R}^n$ kontinuierlich. Leider besteht eine unangenehme Eigenschaft von $\gamma(\mathbf{x})$ darin, dass sie in Regionen wo eine oder mehrere partielle Ableitungen ihr Vorzeichen[3] ändern, nicht mehr vollständig differenzierbar bleibt. Aufgrund des Verlustes gewisser Richtungsableitungen wird die vollständige Berechnung des Gradienten daher unmöglich.

5.3.1 Charakteristika von Minima

Die Koordinaten des robusten Minimums $x^*_{1,r}$ können leicht berechnet werden. Analysiert man die Umgebung von x^*_1, und zwar zunächst geringfügig außerhalb des Unsicherheitsbereichs $U(x^*_1)$, so gilt

$$\gamma(x_1) = f(x_1 - \delta_1) \qquad \text{für } x_1 < x^*_1 - \delta_1 \tag{5.6}$$

$$\gamma(x_1) = f(x_1 + \delta_1) \qquad \text{für } x_1 > x^*_1 + \delta_1 \tag{5.7}$$

Somit liegt das robuste Minimum $x^*_{1,r}$ genau im Schnittpunkt der beiden Kurven irgendwo zwischen

$$x^*_1 - \delta_1 < x^*_{1,r} < x^*_1 + \delta_1 \tag{5.8}$$

weil $\gamma(x_1)$ kontinuierlich ist. Das Minimum befindet sich exakt dort wo

$$f(x_1 - \delta_1) = f(x_1 + \delta_1) \tag{5.9}$$

gilt, also genau an jener Stelle, an der $\gamma(x_1)$ von einem Kurvenast auf den jeweiligen anderen wechselt.

[3] dabei handelt es sich um die Minima von $f(\mathbf{x})$.

Alternativ kann die Lage des robusten Minimums auch durch folgende Überlegung bestimmt werden. Der Schnittpunkt der beiden Kurven liegt eindeutig an jener Stelle, wo die zur Abszisse parallele Strecke mit der Länge von $2\delta_1$ genau in die Kurve passt. Es muss also

$$f(x_{1,u}) = f(x_{1,u} + 2\delta_1) \tag{5.10}$$

gelten. Die Koordinaten von $x^*_{1,r}$ sind dann $[x_{1,u} + \delta_1,\ f(x_{1,u})]^{\mathrm{T}}$, sie liegen somit exakt in der Mitte dieser Strecke[4]. Durch diese alternative Betrachtung können nun zwei Effekte geometrisch wesentlich anschaulicher gedeutet werden.

Je größer δ_1 und je steiler die Kurven sind, desto mehr wird das Minimum gehoben, weil das $2\delta_1$ lange Geradenstück immer weiter nach oben »geschoben« werden muss um vollständig in die Kurve zu passen.

Gleichzeitig tritt aber auch eine seitliche »Versetzung« von $x^*_{1,r}$ auf, wobei diese um so stärker ausfällt, je größer die Unsymmetrie der originalen Funktion $f(x_1)$ ist. Bei der bezüglich des Minimums vollkommen symmetrischen nach oben offenen quadratischen Parabel würde dementsprechend $x^*_{1,r}$ ausschließlich gehoben werden.

Wichtig ist auch eine Analyse des Differentialquotienten. Links von $x^*_{1,r}$ gilt

$$\left.\frac{\partial f(x_1 - \delta_1)}{\partial x_1}\right|_{x_1 < x^*_{1,r}} < 0 \tag{5.11}$$

und rechts gilt

$$\left.\frac{\partial f(x_1 + \delta_1)}{\partial x_1}\right|_{x_1 > x^*_{1,r}} > 0. \tag{5.12}$$

Die Betrachtung des links- und rechtsseitigen Grenzwertes des Differentialquotienten im Punkt $x^*_{1,r}$ führt zur Erkenntnis, dass

$$\lim_{\substack{x_1 \to x^*_{1,r} \\ x_1 < x^*_{1,r}}} \frac{\partial f(x_1 - \delta_1)}{\partial x_1} \neq \lim_{\substack{x_1 \to x^*_{1,r} \\ x_1 > x^*_{1,r}}} \frac{\partial f(x_1 + \delta_1)}{\partial x_1} \tag{5.13}$$

[4]Zur numerischen Lösung von Gleichungen (5.9) und (5.10) wurde wiederum die MATLAB® Funktion **solve** herangezogen.

ist. Dadurch entsteht der durch einen kleinen Kreis in Abbildung 5.3 angedeutete V-förmige Knick im Punkt $x^*_{1,r}$. Die Kurve $\gamma(x_1)$ ist nun leider nicht mehr vollständig differenzierbar. Zu optimierende Zielfunktionen, die nicht vollständig differenzierbar sind werden als »non-smooth Optimization« (NSO) Probleme bezeichnet.

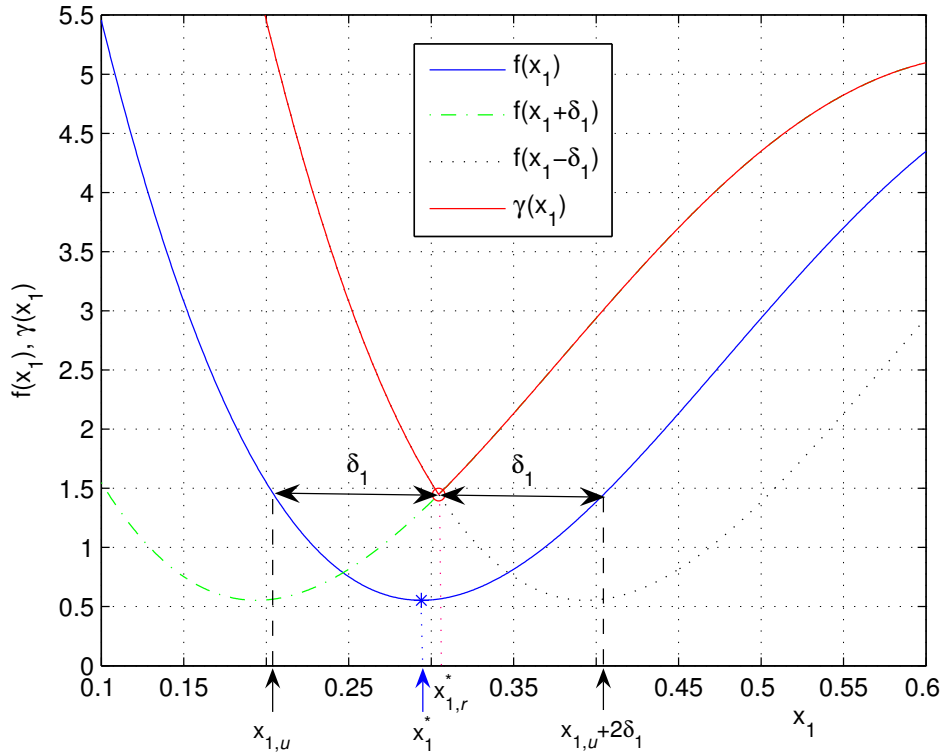


Abb. 5.3: Robustes Minimum

5.3.2 Merkmale von Maxima

Auch hier beginnt die Analyse mit der Umgebung $U(x_1^\times)$. Geringfügig außerhalb gilt

$$\gamma(x_1) = f(x_1 + \delta_1) \qquad \text{für } x_1 < x_1^\times - \delta_1 \tag{5.14}$$

$$\gamma(x_1) = f(x_1 - \delta_1) \qquad \text{für } x_1 > x_1^\times + \delta_1 \tag{5.15}$$

Innerhalb des Unsicherheitsbereichs $U(x_1^\times)$ folgt $\gamma(x_1)$ jedoch nicht wie fälschlich vermutet werden könnte zunächst $f(x_1 + \delta_1)$ um anschließend nach dem Knick auf $f(x_1 - \delta_1)$ zu wechseln und bis zu ihrem Maximum anzusteigen. Ganz im Gegenteil sie

folgt einer Geraden (siehe Grafik 5.4), bleibt also konstant auf Höhe des Maximums $f(x_1^\times)$ mit der Länge von $2\delta_1$.

Der Grund liegt darin, dass im linken steigenden Ast von $f(x_1)$ die maximalen Werte immer am rechten Eckpunkt, und beim rechten fallenden Kurvenast am linken Eckpunkt des Unsicherheitsbereichs auftreten. Sobald jedoch der Unsicherheitsbereich $U(x_1^\times)$ erreicht wird, treten die maximalen Werte nicht mehr an den Ecken auf, sondern dazwischen mit dem konstanten maximalen Wert $f(x_1^\times)$. Erfreulicherweise bleibt die Kurve jedoch im Gegensatz zu Minima dabei immer vollständig differenzierbar.

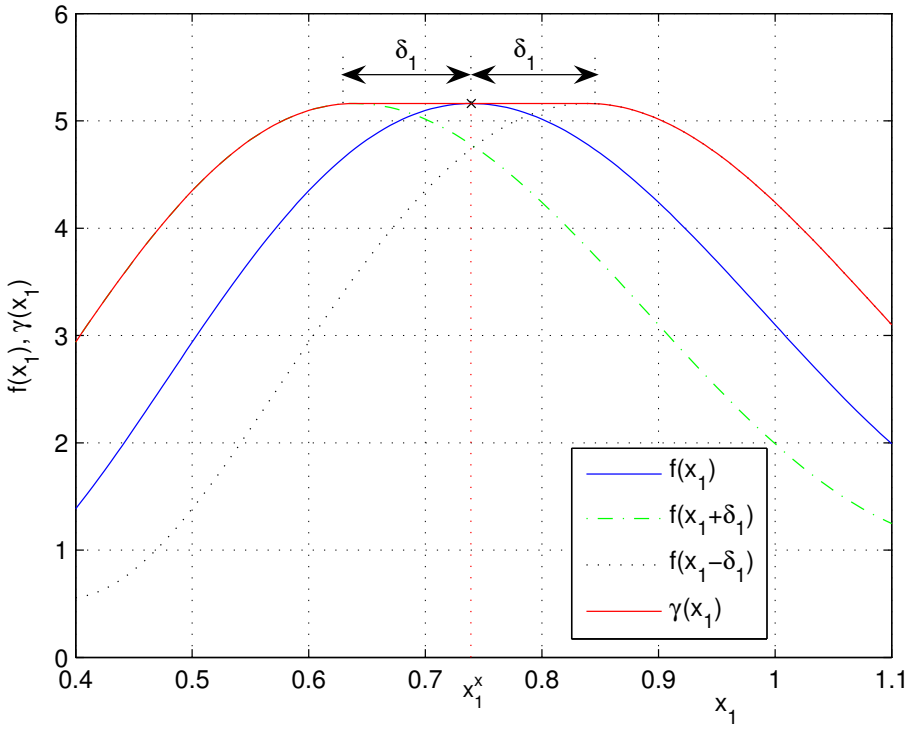


Abb. 5.4: Robustes Maximum

5.4 Einfluss der Variationen auf Lage der Extrema

Robuste Lösungen von Optimierungsproblemen sollten möglichst unempfindlich gegenüber Variationen und Unsicherheiten der Designvariablen sein. Für die Zielfunktion bedeutet dies, dass Minima, unter Akzeptanz von etwas schlechterer Qualität, in flachen Tälern, gegenüber engen Tälern, wo die Höhenlinien dichter gedrängt sind, bevorzugt werden. In steilen Tälern verursachen also gleiche Variationen eine größere Zunahme von Funktionswerten als in flachen Tälern.

Grafik 5.5 stellt einen solchen Fall dar. Für den nicht robusten Fall liegt das globale Minimum x_1^* in einem steilen engen Tal im linken Bereich der Kurve. Auch für $\delta_1 = 0.05$ verharrt das robuste Optimum $x_1^{\square}$ noch in diesem Tal, wenn auch unter Akzeptanz eines schlechteren Zielfunktionswertes. (Vergleiche $\gamma_{\text{global}}(\mathbf{x})$ für $\delta_1 = 0.00$ und $\delta_1 = 0.05$ in Tabelle 5.2.)

Plötzlich bei $\delta_1 = 0.10$ und $\delta_1 = 0.15$ springen die globalen Minima x_1° und x_1^{+}, bei gleichzeitiger Verschlechterung der Zielfunktionswerte in das rechte flachere und weniger störempfindliche Tal.

Hier an diesem Beispiel werden die Vorzüge der robusten Optimierung gegenüber der a posteriori durchgeführten Sensitivitätsanalyse klar deutlich. Beschränkt sich die SA nur auf die Umgebung der nicht robusten Lösung $\mathbf{x}^*$, besteht keine Kenntnis über ein weiteres mit vielleicht etwas schlechterer Qualität vorhandenes Optimum im rechten Teil der Kurve und diese robuste Lösung gerät leicht in Gefahr unbemerkt zu bleiben.

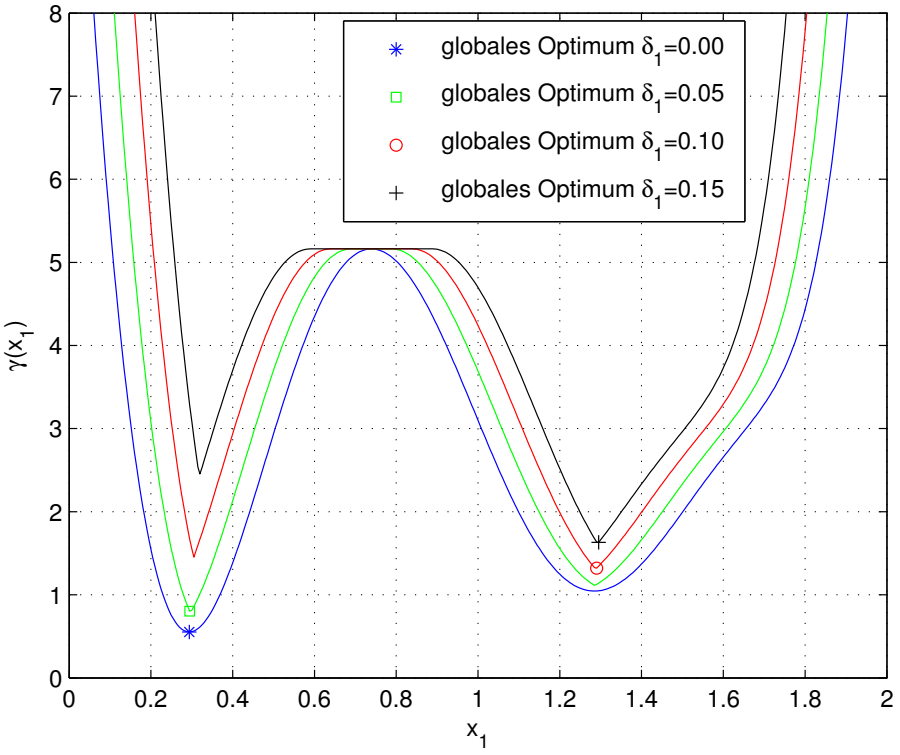


Abb. 5.5: Verlauf $\gamma(x_1)$ in Abhängigkeit von Variation δ_1

Tab. 5.2: Minima von $\gamma(\mathbf{x})$

δ_1	x_1	x_2	$\gamma_{\text{global}}(\mathbf{x})$	x_1	x_2	$\gamma_{\text{lokal}}(\mathbf{x})$
0.00	0.29402253	1.00	0.55334336	1.28459531	1.00	1.04505427
0.05	0.29661966	1.00	0.78634339	1.28541243	1.00	1.11324788
0.10	1.28796530	1.00	1.31037913	0.30455885	1.00	1.44779561
0.15	1.29257253	1.00	1.61471995	0.31830852	1.00	2.42596493

5.5 Konsequenz für Optimierungsalgorithmen

Der Verlust der totalen Differenzierbarkeit bei Minima führt zu weitreichenden Einschränkungen bezüglich der Wahl des Optimierungsalgorithmus. Bekannte deterministische Optimierungsverfahren, wie das Newton- und das Quasi-Newton-Verfahren [1], benötigen in der Theorie zumindest einfach kontinuierlich partiell differenzierbare Funktionen.

Sie können aber eventuell trotzdem Erfolg haben, wenn der benötigte Gradient durch Differenzenquotienten approximiert wird. Andererseits existieren aber auch spezielle NSO Algorithmen [9], die keine differenzierbare Funktionen voraussetzen.

Einen Ausweg aus diesem Dilemma bieten »Direct Search« Methoden, deren typische Vertreter das Simplex Verfahren[5] und die Pattern Search Methode bei [29] beschrieben werden. Sie zählen ebenfalls zur Klasse der deterministischen Optimierungsalgorithmen, jedoch mit dem Vorteil, dass sie nur gewöhnliche Funktionsaufrufe benötigen. Sie leiden jedoch wie alle deterministischen Verfahren an der nicht zufriedenstellenden Konvergenz zum globalen Optimum.

Erschwerend kommt bei derart prinzipiell einfachen Algorithmen das Problem mit dem »Fluch der Dimensionalität« hinzu, einen vom amerikanischen Mathematiker Richard Bellmann [2] geprägten Begriff, der die typischerweise exponentiell mit der Dimension zunehmende Anzahl von benötigten Funktionsauswertungen beschreibt.

Die Lösung liegt daher im Einsatz von stochastischen Optimierungsalgorithmen. Sie benötigen weder Gradienten noch Hessesche Matrizen, sind leicht zu implementieren und basieren lediglich auf gewöhnlichen Funktionsaufrufen.

Eine ihrer wesentlichen Vorzüge besteht in der Tatsache, dass sie auch mit einer gewissen Wahrscheinlichkeit Verschlechterungen von Funktionswerten tolerieren, sie

[5] nicht zu verwechseln mit der vom gleichen geometrischen Konzept abgeleiteten Simplex Methode zur Lösung von Linearen Programmen.

also in der Lage sind lokalen Minima zu »entkommen« und daher bessere Chancen bieten auf das globale Optimum zu konvergieren.

Nachteilig ist jedoch ihre große Zahl von Funktionsaufrufen. Neben dem eigentlichen Hauptproblem der Minimierung der Qualitätsfunktion (4.17) existiert quasi darin »verschachtelt« ein Subproblem, welches für jeden einzelnen Funktionsaufruf zusätzlich gelöst werden muss und daher für eine weitere Steigerung der Funktionsaufrufe sorgt. Es ist die Aufgabe des folgenden 6. Kapitels effiziente Methoden vorzustellen, die die Zahl der zusätzlichen Funktionsauswertungen zur Lösung dieses Subproblems gering halten.

Aufgrund der geschilderten Vorteile von stochastischen Optimierungsalgorithmen, werden in dieser Arbeit zur Lösung von robusten Optimierungsproblemen die in einer rudimentären Weise den biologischen Entwicklungsverlauf imitierenden Evolutionsstrategien eingesetzt.

6 Praktische Implementierung des Subproblems

— So weit geht niemand, der nicht muss. —
Friedrich Schiller aus Wallensteins Tod

Zur Lösung des robusten Problems (4.17) ist es notwendig den maximalen (schlechtesten) Funktionswert innerhalb des Unsicherheitsbereichs $U(\mathbf{x}_0)$ zu bestimmen. Dieses Subproblem muss für jede einzelne Funktionsauswertung des minimierenden Hauptalgorithmus gelöst werden. Der Einsatz eines eigenen Optimierungsalgorithmus scheint daher verlockend.

Das zu lösende Maximierungsproblem[1] lautet

$$\max_{\boldsymbol{\xi} \in U(\mathbf{x}_0)} \{f(\boldsymbol{\xi})\} \tag{6.1}$$

u.B.v.

$$\mathbf{x}_0 - \boldsymbol{\delta} \leq \boldsymbol{\xi} \leq \mathbf{x}_0 + \boldsymbol{\delta} \tag{6.2}$$

[1] durch Vorzeichenumkehr der Zielfunktion $\{\text{-}f(\boldsymbol{\xi})\}$ kann daraus leicht ein Minimierungsproblem formuliert werden.

Die Lösung dieses Optimierungsproblems scheint prinzipiell sehr einfach, weil nur feste obere und untere Grenzen der Designvariablen (6.2) zu berücksichtigen sind. Ein grundsätzlich sehr effektives Verfahren zur Lösung von allgemeinen Problemstellungen der nichtlinearen Programmierung stellt die Methode der Sequentiellen Quadratischen Programmierung (SQP) dar, die als Generalisierung des klassischen Newton-Verfahrens interpretiert werden kann [18].

Schwierigkeiten können jedoch auftreten, wenn innerhalb des Unsicherheitsbereichs mehrere lokale Maxima auftreten, die das Auffinden des gesuchten globalen Maximums erschweren. Weiters benötigen SQP Algorithmen Gradienteninformation und passen somit nur schlecht in das ausschließlich auf Funktionsaufrufe beruhende Konzept der Evolutionsstrategien. Aufgrund dieser geschilderten Nachteile scheint daher die Anwendung eines eigenen separaten Optimierungsalgorithmus zur Lösung des Subproblems wenig praktikabel.

Wesentlich erfolgversprechender hinsichtlich eines minimalen Aufwandes von zusätzlichen Funktionsauswertungen ist die Methode der Diskretisierung des Unsicherheitsbereichs $U(\mathbf{x}_0)$. In den folgenden Abschnitten werden immer effizientere Methoden vorgestellt, die in der Lage sind das Subproblem mit geringst möglicher Anzahl von Funktionsauswertungen zu lösen.

6.1 Vollständige Diskretisierung des Unsicherheitsbereichs (rwcu)[2]

Hier wird der Unsicherheitsbereich $U(\mathbf{x}_0)$, wie in Bild 6.1 für den zweidimensionalen Fall dargestellt ist, vollständig diskretisiert. Anstelle der gesamten Umgebung müssen jetzt »nur« mehr die durch einen roten Punkt gekennzeichneten Funktionswerte evaluiert werden.

Falls das Optimierungsproblem auch nur aus einer vergleichsweise geringen Anzahl von $n = 10$ Designvariablen besteht und in jeder Dimension $d = 10$ Diskretisierungen vorgenommen werden, ergeben sich bereits 10^{10} zusätzliche Funktionsauswertungen pro Subproblem (Tabelle 6.1). Diese Methode ist daher aus Effizienzgründen in der Praxis von Anfang an zum Scheitern verurteilt.

[2] die in Klammern angegebenen Akronyme kennzeichnen die in Algorithmus A.3 implementierten Funktionen.

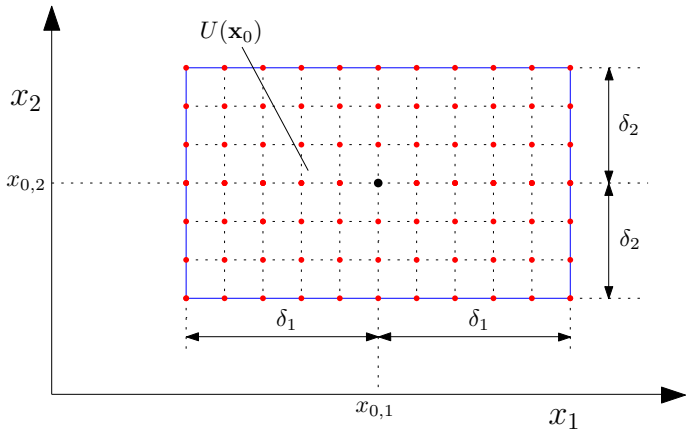


Abb. 6.1: Vollständige 2-D Diskretisierung

6.2 Reine Berücksichtigung der Kanten des Hyperwürfels

Als Zwischenschritt bis zur alleinigen Berücksichtigung der Eckpunkte wäre die reine Diskretisierung der Kanten des Hyperwürfels denkbar.

Die Anzahl der Kanten des Hyperwürfels mit der Dimension n berechnet sich zu

$$n \cdot 2^{n-1} \tag{6.3}$$

und somit ergeben sich bei d Diskretisierungspunkten pro Kante

$$d \cdot n \cdot 2^{n-1} \tag{6.4}$$

zusätzliche Funktionsaufrufe pro Subproblem, die bereits für $d = n = 10$ untragbare 51200 Auswertungen darstellen.

6.3 Auswertung aller Eckpunkte (rwce)

Ein Verfahren, welches für eine spürbare Verminderung der Funktionsaufrufe sorgt, ist die Evaluierung aller Eckpunkte des Hyperwürfels (Abbildung 6.3). Die Idee besteht in der Annahme, dass Ecken, aufgrund größter Distanz zum nominalen Punkt $\mathbf{x}_0$, mit hoher Wahrscheinlichkeit die schlechtesten (größten) Funktionswerte innerhalb des Unsicherheitsbereichs aufweisen.

Diese Methode liefert exakte Resultate, wenn die nicht robuste Funktion $f(\mathbf{x})$ lokal konvex ist (Abschnitt 3.1.2), was immer in der Nähe von strikten lokalen Minima der Fall ist. Weiters liefert sie aber auch exakte Resultate, falls die Funktion $f(\mathbf{x})$ in $U(\mathbf{x})$ monoton bezüglich aller Designvariablen ist [28].

Dies zeigt sich klar in Abbildung 6.2. In diesem einfachen univariaten Fall schrumpft der Hyperwürfel auf eine Gerade, sodass nur mehr der linke und rechte Eckpunkt dieser Strecke auszuwerten sind. Im Bereich monoton wachsender Kurvenabschnitte tritt der schlechteste Funktionswert immer an der rechten Ecke auf, und die Kurvenverläufe von »volle Diskretisierung« und »rechter Eckpunkt« stimmen überein. Bei monoton fallenden Abschnitten erfolgt die Auswertung analog am linken Eckpunkt.

Ein Gebiet, wo diese Methode versagt liegt im Bereich des Maximums, denn hier wechselt die Kurve innerhalb des Unsicherheitsbereichs von monoton wachsend auf monoton fallend. Die schlechtesten Funktionswerte treten nicht mehr an den Eckpunkten auf, sondern dazwischen. Hier liefern nur mehr die volle Diskretisierung (rwcu) und Kantendiskretisierung exakte Resultate.

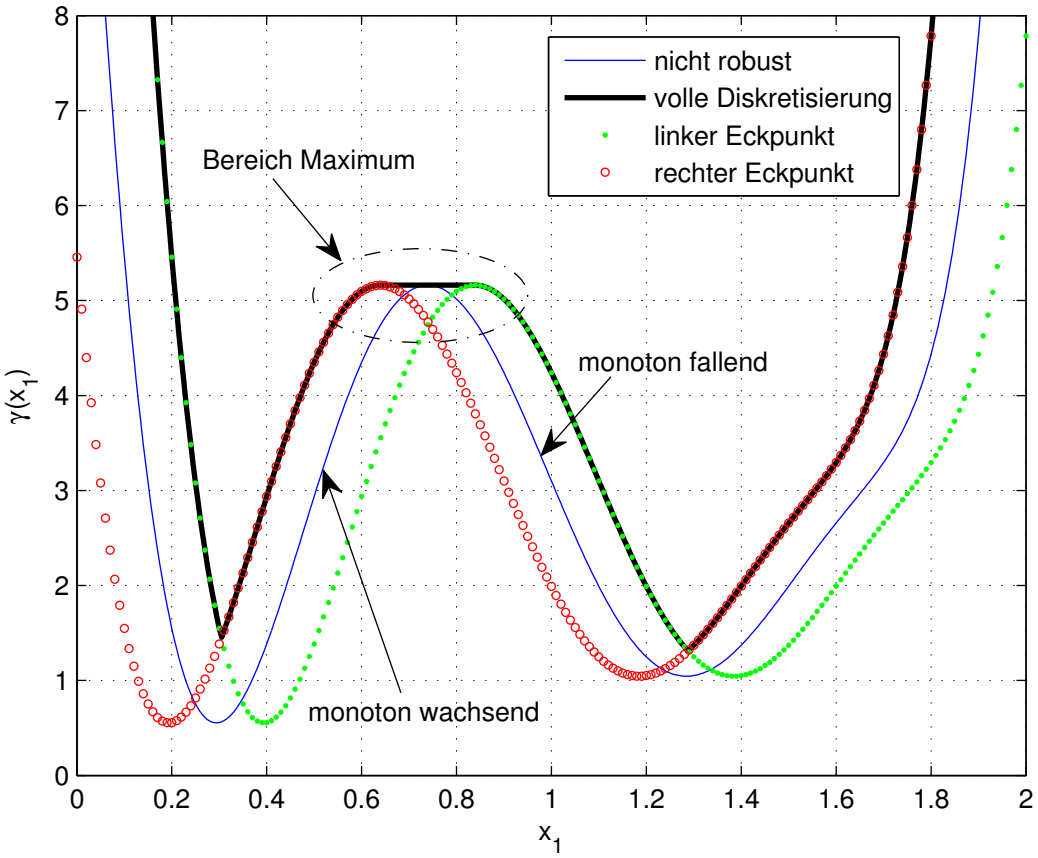


Abb. 6.2: Evaluierung der Eckpunkte

Die allgemeine Bestimmung der Koordinaten der Eckpunkte eines mehrdimensionalen Hyperwürfels soll nun anhand eines dreidimensionalen ($n = 3$) Würfels mit der Kantenlänge eins (Einheitswürfel) hergeleitet werden. Betrachtet man das Koordinatentripel des Einheitswürfels in Abbildung 6.3a fällt sofort auf, dass die Koordinaten als binäre (duale) Repräsentation der dezimalen Eckennummerierung in Bild 6.3b gedeutet werden können.

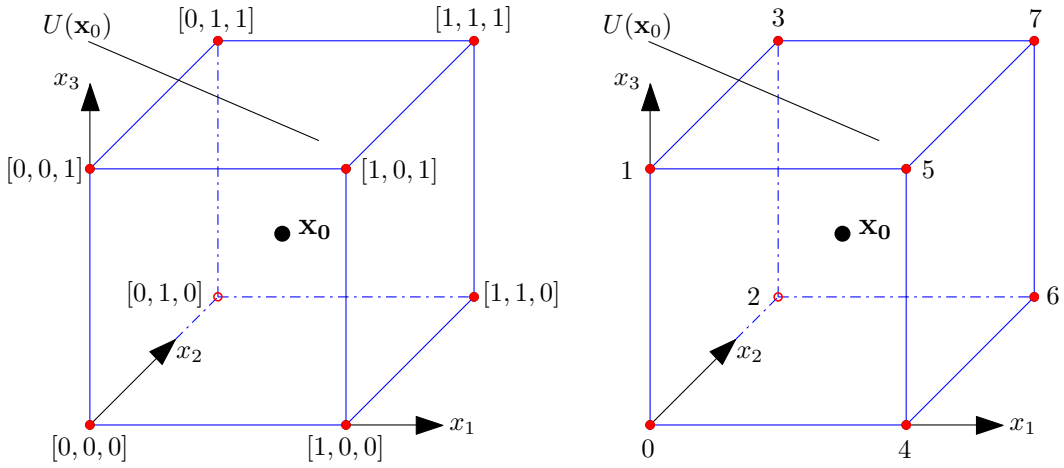
Diese Erkenntnis kann für die Berechnung eines beliebig positionierten und höher dimensionalen Hyperwürfels elegant und vorteilhaft ausgenützt werden. Mit gegebenem nominalen Punkt $\mathbf{x}_0$ und dem Vektor der maximalen Abweichung $\boldsymbol{\delta}$ folgt für die Koordinate des Eckpunktes Null

$$\mathbf{E}_0 = \mathbf{x}_0 - \boldsymbol{\delta}. \tag{6.5}$$

Die k-te Ecke berechnet sich zu[3]

$$\mathbf{E}_k = \mathbf{E}_0 + 2\boldsymbol{\delta} .* \mathbf{E}_k^{(2)} \quad \text{mit } k = 0 \ldots 2^{n-1} \tag{6.6}$$

wobei $\mathbf{E}_k^{(2)}$ der binären Repräsentation der $k-$ten Ecke des Hyperwürfels entspricht, beispielsweise $\mathbf{E}_5^{(2)} = [1, 0, 1]^{\mathrm{T}}$ für Ecke 5.



(a) Koordinatentripel (b) Dezimale Nummerierung

Abb. 6.3: 3-D Hyperwürfel

Trotz der unbestrittenen Vorteile dieses Verfahrens bleibt es mit zusätzlichen 1024 Funktionsaufrufen (Tabelle 6.1) leider ebenfalls nicht praktikabel.

[3]Die Notation $.*$ definiert die in MATLAB® als Feld- oder Array-Operation bezeichnete komponentenweise Multiplikation [4].

6.4 Prädiktion des schlechtesten Eckpunktes

Die in den vorigen Abschnitten beschriebenen Verfahren erweisen sich für praxisrelevante Optimierungsproblemstellungen als völlig inakzeptabel. Daher muss über effizientere Alternativen nachgedacht werden, die zu einer eklatanten Reduktion der zusätzlichen Funktionsauswertungen führen. Die Lösung liegt in der Prädiktion (Vorhersage) des schlechtesten Eckpunktes.

6.4.1 Gradientenmethode

Bei diesem Verfahren wird vereinfachend angenommen, dass der Gradient der Kurve im nominalen Punkt $\mathbf{x}_0$ Auskunft gibt über die wahrscheinliche Lage des schlechtesten Eckpunktes. Wenn die partielle Ableitung $\frac{\partial f(\mathbf{x})}{\partial x_i}$ im Punkt $\mathbf{x}_0$ größer Null ist, so wird vermutet, dass die i-te Koordinate des schlechtesten Eckpunktes bei $x_{o,i} + \delta_i$ liegt.

Somit folgt für den schlechtesten Eckpunkt

$$\mathbf{x}_{\text{schlecht}} = \mathbf{x}_0 + \begin{pmatrix} \text{sign}(\frac{\partial f(\mathbf{x})}{\partial x_1}\Big|_{\mathbf{x}_0}) \cdot \delta_1 \\ \cdots \\ \cdots \\ \text{sign}(\frac{\partial f(\mathbf{x})}{\partial x_n}\Big|_{\mathbf{x}_0}) \cdot \delta_n \end{pmatrix} \tag{6.7}$$

und für den prädizierten schlechtesten Funktionswert

$$\max_{\boldsymbol{\xi} \in U(\mathbf{x}_0)} \{f(\boldsymbol{\xi})\} \approx f(\mathbf{x}_{\text{schlecht}}). \tag{6.8}$$

Diese Methode zeigt jedoch gewisse Schwächen. Wenn der Gradient im Punkt $\mathbf{x}_0$ sehr nahe Null ist, also nicht eindeutig positiv oder negativ, so zeigt er nur wenig Information über das tatsächliche Verhalten der Funktion im Unsicherheitsbereich. Weiters kann natürlich trotz positiver partieller Ableitung im Punkt $\mathbf{x}_0$ der schlechteste Eckpunkt auch bei $x_{o,i} - \delta_i$ positioniert sein.

Vorteilhaft bei diesem Verfahren ist die Tatsache, dass bei analytisch gegebenem Gradient nur n zusätzliche Funktionsauswertungen zur Bestimmung des Gradienten im Punkt $\mathbf{x}_0$ notwendig sind. Wird der Gradient jedoch numerisch berechnet so gilt bei zentralem Differenzenquotient für die partielle Ableitung

$$\frac{\partial f(\mathbf{x})}{\partial x_i}\bigg|_{\mathbf{x_0}} \approx \frac{f(\mathbf{x}_0 + h\mathbf{e}_i) - f(\mathbf{x}_0 - h\mathbf{e}_i)}{2h} \tag{6.9}$$

wobei $\mathbf{e}_i$ den Einheitsvektor der i-ten Koordinatenrichtung repräsentiert. Die Anzahl der zusätzlichen Funktionsauswertungen verdoppelt sich somit auf $2n$, weil pro partieller Ableitung nun zwei Funktionswerte zu bestimmen sind.

6.4.2 Vergleich der Funktionswerte

Eine wesentlich zuverlässigere Methode als die Gradienteninformation ist der direkte Vergleich der Funktionswerte. Es werden immer die an den Kanten, dem nominalen Punkt $\mathbf{x}_0$ gegenüberliegenden Werte, gegenübergestellt. Dabei wird die schlechteste Ecke in jener Richtung vermutet, in der der größere Funktionswert detektiert wird.

Die Koordinate des schlechtesten Eckpunktes berechnet sich zu

$$\mathbf{x}_{\text{schlecht}} = \mathbf{x}_0 + \begin{pmatrix} \text{sign}(f(x_{0,1} + \delta_1) - f(x_{0,1} - \delta_1)) \cdot \delta_1 \\ \cdots \\ \cdots \\ \text{sign}(f(x_{0,n} + \delta_n) - f(x_{0,n} - \delta_n)) \cdot \delta_n \end{pmatrix}. \tag{6.10}$$

Die Anzahl der zusätzlichen Funktionsauswertungen pro Subproblem bleibt dabei mit $2n$ gleich niedrig wie bei der Gradientenmethode.

6.4.3 Modifiziertes Pattern Search Verfahren (rwcp)

Dieses dem Funktionswertevergleich (Abschnitt 6.4.2) ähnliche Verfahren stellt eine spezielle Adaption der Pattern Search Methode [29] dar. Das eigentliche Pattern Search Verfahren kann durch die zwei Bewegungsrichtungen Exploration und die als »Pattern Move« bezeichnete Extrapolation charakterisiert werden, wobei erstere nun zur Prädiktion des schlechtesten Eckpunktes herangezogen wird.

Bei dieser Methode werden nicht wie beim vorigen Verfahren immer die dem nominalen Punkt $\mathbf{x}_0$ gegenüberliegenden Funktionswerte verglichen, sondern jenes Funktionspaar, welches näher dem im vorigen Iterationsschritt ermittelten Gebiet liegt. Dadurch ergibt sich eine höhere Wahrscheinlichkeit den schlechtesten Eckpunkt richtig zu prädizieren.

Startet man vom nominalen Punkt `x0`, so werden diesem Punkt gegenüberliegende Funktionswerte verglichen, und zwar `f_plus` als Funktionswert von `x_plus(1)`[4] und

[4] die in Klammern angegebenen Zahlen kennzeichnen die jeweilige Koordinatenrichtung bzw. Iteration.

`f_wert` als Funktionswert von `x_minus(1)`, siehe Codezeilen 7 und 8 von Algorithmus 6.1. Ist nun beispielsweise `f_plus` kleiner als `f_wert` (Codezeile 12), so wird der neue Vergleichspunkt `x_schlecht(2)` definiert (Codezeile 13), der an der linken rot eingefärbten Kante (Grafik 6.4) liegt. Im nächsten Iterationsschritt werden nun diesem Punkt `x_schlecht(2)` gegenüberliegende Werte `x_plus(2)` und `x_minus(2)` verglichen. Hier liegt der wesentliche Unterschied zum vorigen Verfahren. Beim Funktionswertevergleich (Abschnitt 6.4.2) würde das dem Punkt `x0` gegenüberliegende grüne Punktepaar einer Analyse unterzogen werden.

Weil pro Koordinatenrichtung immer zwei Werte verglichen werden bleibt die Anzahl der zusätzlichen Funktionsauswertungen pro Subproblem mit $2n$ gleich niedrig wie vorher.

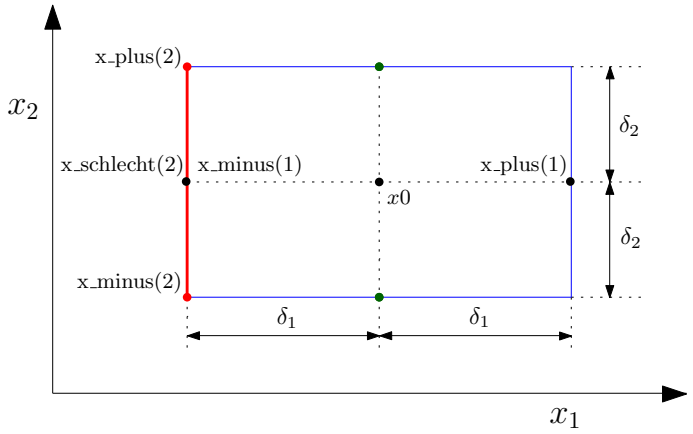


Abb. 6.4: 2-D Pattern Search

Algorithmus 6.1 Pattern Search in MATLAB® Syntax

```
1 x_schlecht=x0;
2    for i=1:n
3         x_plus=x_schlecht;
4         x_plus(i)=x_plus(i)+delta(i);
5         x_minus=x_schlecht;
6         x_minus(i)=x_minus(i)-delta(i);
7         f_plus=evaluate(x_plus);
8         f_wert=evaluate(x_minus);
9              if f_plus > f_wert
10                 x_schlecht=x_plus;
11                 f_wert=f_plus;
12             else
13                 x_schlecht=x_minus;
14             end
15   end
```

Abbildung 6.5 zeigt einen Vergleich aller in der Arbeit implementierter Diskretisierungsvarianten. Dabei zeigt sich, dass ausschließlich die vollständige Auswertung des Unsicherheitsbereichs (rwcu) im gesamten Bereich richtige Ergebnisse liefert. Die Auswertung aller Eckpunkte (rwce) und das Pattern Search Verfahren (rwcp) zeigen vollkommen identische Verläufe und unterscheiden sich beide nur im Bereich des Maximums bei $x_1 = 0.74$ vom exakten Verlauf.

Im Bereich des wichtigen Minimums haben sie gleichen Verlauf wie rwcu. Dem Pattern Search Verfahren wird daher, aufgrund wesentlich geringerer zusätzlicher Funktionsauswertungen von nun an für robuste Optimierungen der Vorzug gegeben.

Der Vollständigkeit halber wurde auch die Mittelwert Auswertung (rmw) in die Grafik mit aufgenommen. Diese zeigt im Vergleich zu rwcu immer unter-konservatives Verhalten und hat aber den Vorteil, dass es weder bei Maxima noch bei Minima zu Verletzungen der Differenzierbarkeit kommt.

Tab. 6.1: Diskretisierungsmethoden

Auswertung	Algorithmus	Aufrufe allgemein	Aufrufe ($d = n = 10$)
Unsicherheitsbereich vollständig	rwcu	d^n	10^{10}
Mittelwert	rmw	d^n	10^{10}
Kanten	–	$d \cdot n \cdot 2^{n-1}$	51200
Eckpunkte	rwce	2^n	1024
Prädiktion Gradient analytisch	–	n	10
Prädiktion Gradient numerisch	–	$2n$	20
Prädiktion Wertevergleich	–	$2n$	20
Prädiktion Pattern Search	rwcp	$2n$	20
Zielfunktion unverändert	nr	–	0

rwcu[e/p]	**r**obust **W**orst **C**ase **U**nsicherheitsbereich[**E**ckpunkte/**P**attern Search]		
rmw	**r**obust **M**ittel**w**ert	–	nicht implementiert bzw. nicht existent
nr	**n**icht **r**obust		

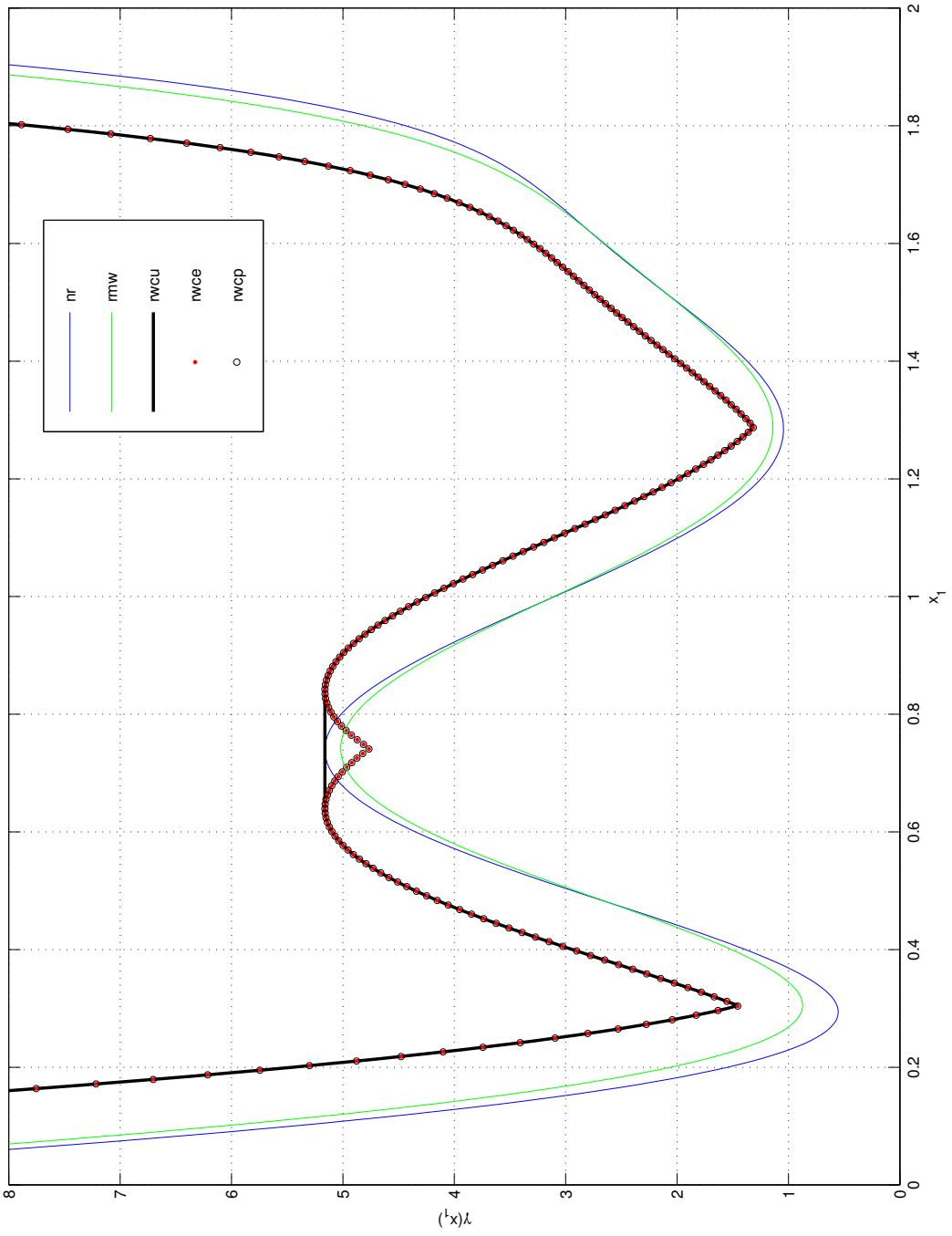


Abb. 6.5: Vergleich implementierter Diskretisierungsarten

7 Evolutionsstrategien

— Die natürliche Auslese sorgt dafür, dass immer die Stärksten oder die am besten Angepassten überleben. —
Charles Darwin

7.1 Einführung

Evolutionsstrategien (ES) bilden eine Subklasse der von der Natur inspirierten evolutionären Algorithmen (EA). Sie wenden die in Charles Darwins Evolutionstheorie beschriebenen interagierenden stochastischen Prozesse der Rekombination, Mutation und Selektion auf einzelne Individuen innerhalb einer Population an, um iterativ im Mittel immer bessere Lösungen eines Optimierungsproblems zu generieren.

Dabei werden in diesem evolutionärem Modell die einzelnen Individuen durch üblicherweise Ganzkomma- bzw. Fließkommazahlen als Phänotyp und nicht wie beim klassischen binären Genetischen Algorithmus (GA) als Gene entlang des Chromosoms (Genotyp) repräsentiert.

Die Entwicklung der ES hat ihren Ursprung in den Arbeiten von Bienert, Rechenberg und Schwefel Mitte der sechziger Jahre an der TU-Berlin, die ihre praktische technische Anwendung in der Entwicklung eines widerstandsoptimierten Strömungskörpers im Windkanal fand.

Nach dem ursprünglichen, das evolutionäre Verhalten der Natur nur sehr primitiv und rudimentär beschreibenden Modell der $(1+1)$-ES, entstand unter SCHWEFEL [23] die allgemeinere und die Natur wesentlich besser imitierende $(\mu/\rho, \lambda)$-ES und später die geschachtelten, aus mehreren Subpopulationen bestehenden ES, die bei der Lösung von komplexen multimodalen Optimierungsproblemen ihren Einsatz finden.

Dabei bezeichnet μ die Gesamtanzahl der Eltern und ρ die Anzahl der an der Rekombination beteiligten Eltern. Häufig in Verwendung sind die üblicherweise bisexuelle Rekombination mit $\rho = 2$ und die multisexuelle globale Rekombination mit $\rho = \mu$ Eltern. Mit λ ist die Anzahl der Kinder bzw. Nachkömmlinge festgelegt. Die Kinder gelangen entweder allein (Komma-Notation), oder zusammen mit den Eltern (Plus-Notation) zur Selektion.

7.2 Definitionen

Für die Evolutionsstrategie wird ein Individuum

$$\mathbf{a} = \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\sigma} \\ F(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_n \\ F(\mathbf{x}) \end{bmatrix} \tag{7.1}$$

definiert.

Dieses enthält neben den n Objektparametern x_i, eine gleiche Anzahl von Strategieparametern σ_i (Schrittweiten) und einen Fitnesswert $F(\mathbf{x})$, der im einfachsten Fall identisch mit dem Zielfunktionswert $f(\mathbf{x})$ ist. Die Unterscheidung zwischen $F(\mathbf{x})$ und $f(\mathbf{x})$ ist notwendig, weil $F(\mathbf{x})$ das Resultat einer weiteren ES (Meta-Optimierung), das Ergebnis einer lokalen auf $f(\mathbf{x})$ angewendeten Optimierung oder auch das Ergebnis eines verrauschten $f(\mathbf{x})$ Prozesses sein kann.

Somit ergibt sich für die Elternpopulation eine Matrix vom Typ $[(2n+1) \times \mu]$, die spaltenweise die μ Eltern nebeneinander anordnet und für die Kinderpopulation folgt vollkommen analog die identische Struktur der Matrix aber mit Typ $[(2n+1) \times \lambda]$.

Durch Quotientenbildung von λ und μ erhält man mit

$$S = \frac{\lambda}{\mu} \qquad 1 < S < \infty \tag{7.2}$$

den Selektionsdruck[1] der ES-Strategie.

Der Selektionsdruck S beeinflusst das globale Verhalten der Strategie. Ist die Anzahl der Kinder λ im Verhältnis zur Anzahl der Eltern μ groß, so findet unter den vielen Nachkommen beim anschließenden Selektionsschritt, aufgrund geringer Elternzahl, eine sehr starke Auslese statt.

Schlechtere Individuen, die sich dem Einflussbereich eines Minimums entziehen könnten werden folglich von einer kleinen Schar von sehr guten Kindern verdrängt und die Strategie endet leicht in einem lokalen Minimum. Daher sollte für gutes globales Verhalten ein eher geringer Selektionsdruck von $S = 2$ bevorzugt werden.

7.3 Detailanalyse des ES-Konzeptes

Zunächst folgt eine elementare Beschreibung der Struktur der $(\mu/2, \lambda)$-ES, die im Anschluss detailliert um verwendete Konzepte und Mechanismen ergänzt wird.

Grundlegender Ablaufplan einer $(\mu/2, \lambda)$-ES

1. Gründung einer zufälligen Startpopulation $P_\mu = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_\mu\}$.
2. Generierung von λ Nachkommen, die die Kinderpopulation bilden $\tilde{P}_\lambda = \{\tilde{\mathbf{a}}_1, \tilde{\mathbf{a}}_2, \ldots, \tilde{\mathbf{a}}_\lambda\}$.
   a) Auswahl von $\rho = 2$ Eltern $\mathbf{e}_1$ und $\mathbf{e}_2$.
   b) Rekombination der beiden Eltern zu zwei Nachkömmlinge $\mathbf{n}_1$ und $\mathbf{n}_2$.
   c) Mutation der Strategieparameter.
   d) Mutation der Objektparameter unter Verwendung des im Punkt (c) erzeugten Satz von Strategieparametern.
3. Selektion der neuen Elternpopulation aus der nachkommenden Generation $\tilde{P}_\lambda$.
4. Sprung zurück zu Punkt 2 bis Abbruchkriterium erfüllt ist.

[1] zu beachten ist die in der Literatur gelegentliche Verwendung des Kehrwertes des Selektionsdrucks.

7.3.1 Schritt 1: Initialisierung

Falls kein Vorwissen über die Position des globalen Minimums vorhanden ist, sollten die Individuen möglichst zufällig über den Entwurfsraum $\mathcal{D}$ verteilt werden.

7.3.2 Schritt 2: Generierung von Nachkommen

Aus μ Eltern werden durch Rekombination λ Kinder generiert.

Folgende Teilschritte sind so oft zu wiederholen bis eine Population von λ Nachkommen zur Verfügung steht.

Teilschritt 2a: Auswahl der Eltern

Der Prozess zur Selektion der Eltern kann entweder rein zufällig (stochastische Partnerwahl) oder durch Einbeziehung eines Qualitätsmerkmales in das Auswahlverfahren erfolgen. Dabei wird jedem Elternteil eine seiner Qualität entsprechende Selektionswahrscheinlichkeit zugeordnet, die bildhaft auf einem Rouletterad eingetragen wird.

Eltern mit hoher Fitness nehmen daher am Roulettrad eine große Fläche ein. Die Selektion der Eltern (»Drehen am Rouletterad«) bleibt noch immer zufällig und Eltern mit höheren Qualitätsmerkmalen haben eine bessere Wahrscheinlichkeit am anschließenden Rekombinationsprozess teilzunehmen. Gleichzeitig haben aber auch »schlechtere« Eltern eine, wenn auch geringe Chance gezogen zu werden.

Verschiedene Möglichkeiten Qualität zu bewerten sind möglich.

Fitnessbasierte Selektion (Listing A.5) Bei diesem Verfahren wird der Kehrwert des Zielfunktionswertes als Qualitätskriterium herangezogen, denn bei einem gesuchten Minimum müssen kleine Zielfunktionswerte einem guten (hohen) Fitnesswert (Qualitätsmerkmal) entsprechen. Diese fitnessproportionale Selektion zeigt jedoch entscheidende Schwächen.

- Versagen des Verfahrens bei negativen Funktionswerten. (Kann durch entsprechende Skalierung verhindert werden.)
- Individuen mit kleinen Zielfunktionswerten (hoher Fitness) übernehmen sehr rasch die Dominanz über den Rest der Bevölkerung und ein frühes Simulationsende durch vorzeitige Konvergenz (engl.: *premature convergence*) ist wahrscheinlich.

- Wenn Fitnesswerte sehr eng beieinander liegen, existiert nahezu kein Selektionsdruck, weil alle Teile des Rouletterades, die den einzelnen Individuen zugeordnet sind, annähernd die gleiche Fläche einnehmen. Dadurch erfolgt die Selektion gleichsam rein zufällig, Individuen mit geringfügig besserer Fitness profitieren nur kaum und die Konvergenzgeschwindigkeit bleibt relativ niedrig.

Rangbasierte Selektion (Listing A.6) Hier werden die Eltern gemäß ihrer Zielfunktionswerte sortiert und jenem Individuum mit dem kleinsten Zielfunktionswert die beste Qualität (höchster Rang) zugeordnet. Dadurch ergeben sich keine Probleme bei negativen Funktionswerten (Rangzahl bleibt auch bei negativen Funktionswerten immer positiv), und der Selektionsdruck innerhalb der Generation bleibt immer konstant.

Die Funktionsweise der rangbasierten Selektion soll anhand eines einfachen Beispiels verdeutlicht werden.

- Gegeben sind fünf Individuen mit den Zielfunktionswerten f_i

  $f_1 = -100,\ f_2 = -5,\ f_3 = 5,\ f_4 = 10,\ f_5 = 100$

- Bestimmung der rangbasierten Fitness F_i

  $F_1 = 5,\ F_2 = 4,\ F_3 = 3,\ F_4 = 2,\ F_5 = 1; \qquad \text{mit } \sum F_i = 15$

- Berechnung der Selektionswahrscheinlichkeiten $p_i = \frac{F_i}{\sum F_i}$

  $p_1 = \frac{5}{15},\ p_2 = \frac{4}{15},\ p_3 = \frac{3}{15},\ p_4 = \frac{2}{15},\ p_5 = \frac{1}{15};\ \text{ mit } \sum p_i = 1$

Vergleicht man nun den Verlauf der Zielfunktionswerte in Abbildung 7.1, so ist selbst an diesem einfachen mit der Testfunktion Rosenbrock erstellten Evolutionsverlauf klar die signifikant verbesserte Konvergenzgeschwindigkeit der rangbasierten Methode gegenüber dem fitnessbasierten Verfahren ersichtlich.

Um einen objektiven Vergleich zu ermöglichen und Fehler durch statistische Einflüsse zu unterbinden wurden alle Simulationen bei identischer Startpopulation zehn mal wiederholt und Durchschnittswerte gebildet.

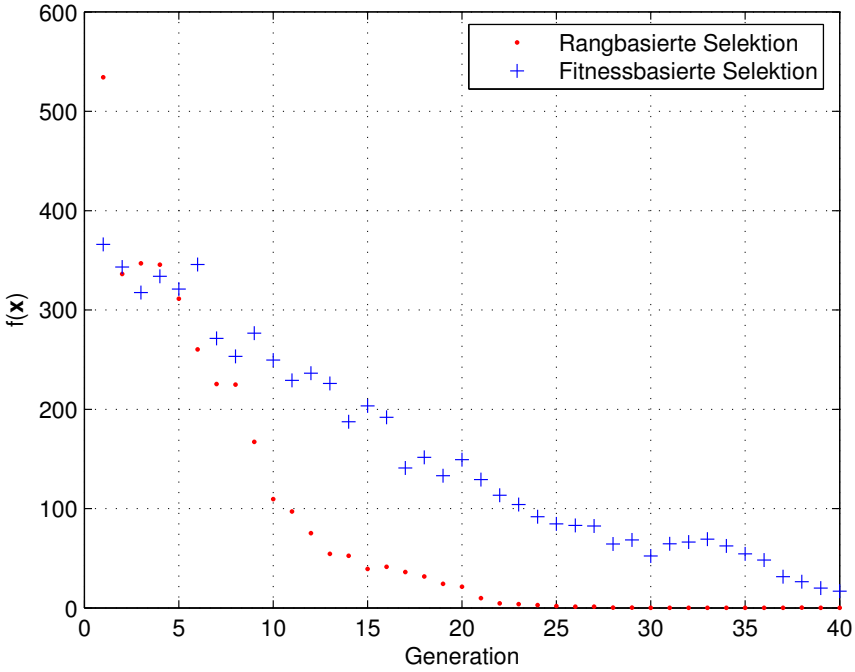


Abb. 7.1: Selektionsvarianten

Teilschritt 2b: Rekombination

Prinzipiell sind zwei unterschiedliche Arten der Rekombination (engl.: *crossover)* zu unterscheiden [13]. Die diskrete Methode, die ihren Ursprung im klassischen GA hat, bei dem die Gene binär codiert sind und die Rekombination durch wechselseitiges Austauschen der Gene durchgeführt wird. Die gleiche Methode bewirkt bei der ES ein gegenseitiges zufälliges Vertauschen der Strategie- und Objektparameter der beiden Elternteile.

Bei der intermediären Rekombination werden die Nachkömmlinge $\mathbf{n}_1$ und $\mathbf{n}_2$ nach folgendem Ansatz erzeugt,

$$\begin{aligned} \mathbf{n}_1 &= \gamma \cdot \mathbf{e}_1 + (1-\gamma) \cdot \mathbf{e}_2 \\ \mathbf{n}_2 &= \gamma \cdot \mathbf{e}_2 + (1-\gamma) \cdot \mathbf{e}_1 \end{aligned} \tag{7.3}$$

wobei $\mathbf{e}_1$ und $\mathbf{e}_2$ die beiden im **Teilschritt 2a** (Abschnitt 7.3.2) selektierten Eltern bezeichnet.

Dabei hat sich für den Gewichtungsfaktor γ eine normalverteilte Zufallsvariable mit Mittelwert $\mu = 0.8$ und Standardabweichung $\sigma = 0.5$ als günstig erwiesen. Entscheidend für den weiteren Verlauf ist dabei, dass Nachkömmlinge einerseits Merkmale von

ihren beiden Eltern erben, aber andererseits auch genügend Variationsmöglichkeiten für eine bessere Fitness zur Verfügung stehen.

Aus empirischen Untersuchungen ist bekannt, dass die Anwendung der diskreten Rekombination auf die Objektparameter $\mathbf{x}$ und die intermediäre Rekombination auf die Strategieparameter $\boldsymbol{\sigma}$ vorteilhaft für den weiteren Simulationsverlauf ist.

Teilschritt 2c: Mutation der Strategieparameter

Aufgrund der Festlegung der Schrittweite als Strategieparameter kann nun die Steuerung dieser adaptiv, also während der Simulation ohne Vorgabe einer deterministischen Regel erfolgen.

Multiplikative Methode Die Mutation der Schrittweite erfolgt durch einen Faktor α, der zufällig durch Multiplikation und Division eine Vergrößerung bzw. Verkleinerung der Strategieparameter bewirkt. Die bei den Objektparametern $\mathbf{x}$ gebräuchliche additive Mutation ist beim Satz der Strategieparameter $\boldsymbol{\sigma}$ nicht empfehlenswert, weil sonst auch negative Schrittweiten möglich wären.

Laut Rechenberg [21] haben sich Schrittweitenfaktoren von $\alpha = 1.3$ bewährt. Genauere Untersuchungen [12] zeigen aber, dass α auch von der Anzahl der Designvariablen und der Nachkommen abhängig ist. Bei kleinem n sollte α eher größere und bei vielen Designvariablen ($n > 100$) eher kleinere Werte innerhalb des Intervalls $[1.1, 1.6]$ annehmen.

Modell nach Schwefel [23] Schwefel erweiterte die einfache multiplikative Schrittweitenänderung durch Aufspaltung in zwei logarithmisch-normalverteilte Anteile. Für die mutierte Schrittweite $\tilde{\sigma}_i$ gilt

$$\tilde{\sigma}_i = \sigma_i \cdot \mathrm{e}^{\tau_0 \cdot N(0,1)} \cdot \mathrm{e}^{\tau_1 \cdot N_i(0,1)} \tag{7.4}$$

$N(0,1)$ bezeichnet eine einmalige Realisierung einer normalverteilten Zufallsvariable (ZV) mit $\mu = 0$ und $\sigma = 1$, die den Satz von Strategieparametern global, das heißt für jeden Nachkömmling einheitlich beeinflusst, während die Normalverteilung $N_i(0,1)$ die Strategieparameter σ_i individuell mutiert[2].

Gründe für die Verwendung von logarithmisch-normalverteilten ZV liegen einerseits darin, dass Schrittweiten immer positiv bleiben und andererseits aus Beobachtungen

[2] durch elegante Anwendung des dyadischen Produktes kann Gleichung (7.4) auf eine einzige Matrizenmultiplikation reduziert werden, siehe Algorithmus A.1, Codezeilen 124–126.

der Natur bekannt ist, dass kleine Variationen der Parameter häufiger auftreten als große.

Für die beiden als Nachfolger- und Dimensionsbeiwert bezeichneten Konstanten

$$\begin{aligned} \tau_0 &\approx \frac{1}{\sqrt{2n}} \\ \tau_1 &\approx \frac{1}{\sqrt{2\sqrt{n}}} \end{aligned} \tag{7.5}$$

ist nach neuesten Erkenntnissen ein Wertebereich von $[0.1, 0.2]$ günstig.

Die Effizienz und somit höhere Konvergenzgeschwindigkeit der mutativen Schrittweitensteuerung nach SCHWEFEL gegenüber der einfachen multiplikativen Methode nach RECHENBERG demonstriert eindrucksvoll der mit der Testfunktion Peaks in mehreren Simulationsversuchen erstellte Schrittweitenverlauf in Grafik 7.2.

Entgegen der Forderung nach ausschließlich positiven Werten der Schrittweiten σ_i treten zwischen Generation 20 und 40 bei der Schrittweitensteuerung nach SCHWEFEL doch leicht negative Werte auf. Die Ursache liegt im Gewichtungsfaktor γ in Gleichung (7.3). Ein Möglichkeit der Lösung besteht in einer nachträglichen Analyse und Zurücksetzen der Schrittweiten an die zulässigen Parametergrenzen $[\sigma_{\min}, \sigma_{\max}]$.

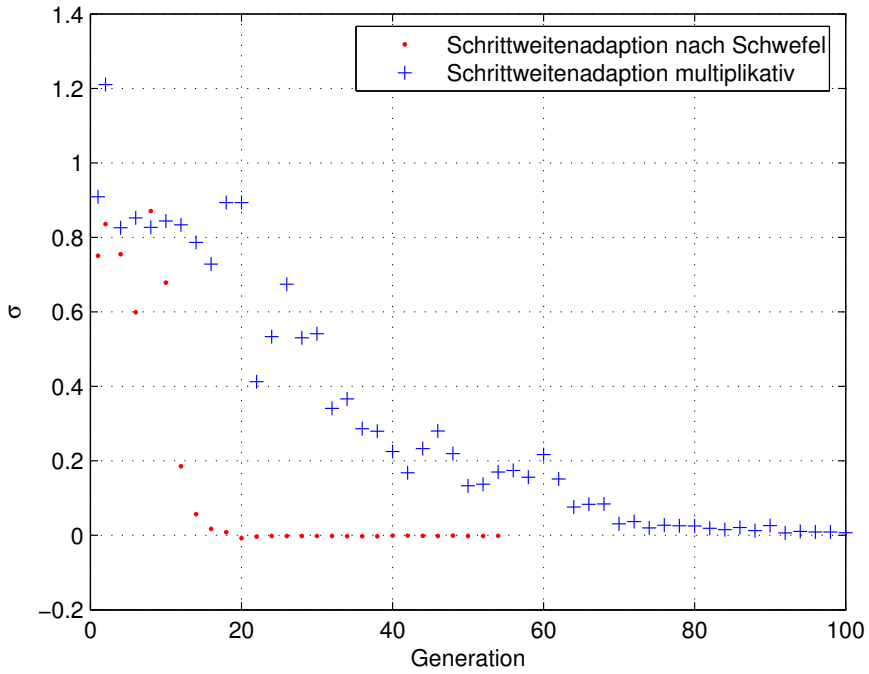


Abb. 7.2: Vergleich Schrittweitenadaptionen

Teilschritt 2d: Mutation der Objektparameter

Durch Addition einer normalverteilten ZV mit $\mu = 0$ und der in Gleichung (7.4) bestimmten Standardabweichung $\tilde{\sigma}_i$ folgt für die i-te Komponente des mutierten Objektparameters

$$\tilde{x}_i = x_i + N_i(0, \tilde{\sigma}_i) \tag{7.6}$$

Bei der additiven Mutation der Objektparameter kann durchaus der Fall eintreten, dass einzelne Individuen der Kinderpopulation $\tilde{P}_\lambda$ außerhalb des Entwurfsraum $\mathcal{D}$ liegen. Daher ist es notwendig die Objektparameter $\mathbf{x}$ wieder an den Rand des Suchraumes (Listing A.7) zurückzusetzen.

7.3.3 Schritt 3: Deterministische Selektion

Aus der Population von λ Kindern werden die besten μ Individuen, durch Bewertung ihrer Zielfunktionswerte ausgewählt und ersetzen somit alle ihre Eltern. Dabei ist bei der beschriebenen Komma-Strategie zu beachten, dass gute Systemzustände der Elternpopulation »vergessen« werden. Daher sollte in der Praxis bei dieser Strategie der jeweils beste ermittelte Systemzustand gespeichert, laufend aktualisiert und am Simulationsende separat ausgegeben werden, siehe Algorithmus A.1, Codezeile 169.

7.3.4 Schritt 4: Abbruchkriterium

Bei der Wahl des Abbruchkriteriums sollte immer die Gefahr eines zu frühen Simulationsendes vermieden werden. Folgende Möglichkeiten bieten sich an:

- Ein einfaches Kriterium stellt die Anzahl der maximalen Generationen dar. Diese Methode ist aber nur selten geeignet die Simulation richtig (d.h. im Optimum) zu beenden, sondern dient in erster Linie, in Kombination mit den beiden nachfolgend beschriebenen, bei Misserfolg eine unendliche Simulationsdauer zu verhindern.

- Ein von Schwefel [23] vorgeschlagenes Kriterium basiert auf einem Funktionswertevergleich des besten und schlechtestes Zielfunktionswertes innerhalb der aktuellen Population. Die Strategie terminiert, sobald die Differenz von

  $$|f_{\text{best}}(\mathbf{x}) - f_{\text{worst}}(\mathbf{x})| \tag{7.7}$$

  ein vom Anwender definiertes ϵ Kriterium unterschreitet.

- Alternativ könnte die Optimierung enden, wenn die quadratische Norm der Population ein festgelegtes ϵ Kriterium unterschreitet (Listing A.4). Für eine Population von λ Individuen mit n Optimierungsvariablen wird zuerst ein mittlerer Aufenthaltsort

$$\bar{\mathbf{x}} = \frac{1}{\lambda} \sum_{i=1}^{\lambda} \mathbf{x}_i \tag{7.8}$$

ermittelt. Anschließend wir die skalare quadratische Norm

$$qn = \frac{1}{\lambda n} \sum_{j=1}^{n} \sum_{i=1}^{\lambda} \left(\frac{x_{j,i} - \bar{x}_j}{\bar{x}_j} \right)^2 \tag{7.9}$$

bestimmt, die angibt, ob noch immer global im Entwurfsraum $\mathcal{D}$ gesucht wird, oder ob die Individuen bereits begonnen haben sich auf eine bestimmte Domäne zu konzentrieren.

7.4 Optimierungsbeispiele

Um die korrekte Funktion der programmierten MATLAB®-Skripts zu überprüfen wurden mit allen implementierten Testfunktionen (Listing A.8) umfangreiche Simulationen durchgeführt, wobei nun zwei Testszenarien näher erläutert werden sollen.

7.4.1 Klassische – nicht robuste Minimierung

Als repräsentatives Beispiel einer klassischen nicht robusten Optimierung wurde die multimodale Testfunktion Peaks (Abschnitt 2.3.2) ausgewählt und die Simulation mit folgender Belegung der Strategieparameter gestartet:

```
testfunktion='peaks';
n=2;          % Anzahl Optimierungsvariablen
xu=[-3;-3];   % Suchraum untere Grenze
xo=[3;3];     % Suchraum obere Grenze
typ='nr';     % Nicht robuste Optimierung
mue=10;       % Zahl der Eltern
lambda=20;    % Anzahl Kinder
sigma=1;      % Anfangsschrittweite
tau_0=0.1;tau_1=0.15; % Nachfolger-, Dimensionsbeiwert
% Abbruchkriterium Quadratische Norm
epsilon_qn=1e-5;
% Vergleich bester und schlechtester Zielfunktionswert
epsilon_bs=1e-4;
```

```
% maximale Anzahl Generationen
genmax=500;
% Grafikausgabe (ja=1, nein=0)
grafik=1;
genstep=5;   % Schrittweite für Grafikausgabe
```

Diskussion der Ergebnisse

Ausgabe der MATLAB®-Routine **EvolutionTool** (Algorithmus A.1) :

```
Ergebnis:

Optimaler Lösungsvektor:
     0.22846263421500
    -1.62526277906245
Zielfunktionswert       :-6.55113212771215
Funktionsaufrufe        :3530
Anzahl Generationen     :175
Quadrat.Norm der Pop.   :1.51749617828826e-005
Differenz best--worst   :9.39419474965674e-005
Rechenzeit in Sekunden  :0.347


kleinster je gefundener Zielfunktionswert: -6.55113310481447 in Generation: 150

zugehörige Optimierungsvariablen:
     0.22816815625191
    -1.62546094402434
>>
```

Nach bereits 3530 Funktionsaufrufen steht das Resultat des minimalen Zielfunktionswertes von -6.5511 fest. Abbildung 7.3 zeigt die noch zufällige Verteilung der Population der ersten Generation, die in Generation 175 am Ende des Evolutionsprozesses durch eine starke Konzentration der Individuen im Minimum (Grafik 7.4) abgeschlossen ist.

Interessant ist bei der Analyse des Schrittweitenverlaufs σ_1, σ_2 und des Zielfunktionswertes $f(\mathbf{x})$, dass bereits in Generation 80 eine starke Tendenz auf das Optimum zu erkennen ist. Die Simulation endet jedoch nicht, weil noch keine der eingestellten Abbruchbedingungen (`epsilon_qn`, `epsilon_bs`) erfüllt ist.

Zu beachten ist auch, dass der absolut kleinste Zielfunktionswert nicht am Evolutionsende, sondern in Generation 150 gefunden wurde. Dies ist kein Fehler, sondern eine wesentliche Eigenschaft der Komma-Strategie, die ihre Ursache im »Vergessen« guter Systemzustände (Abschnitt 7.3.3) hat.

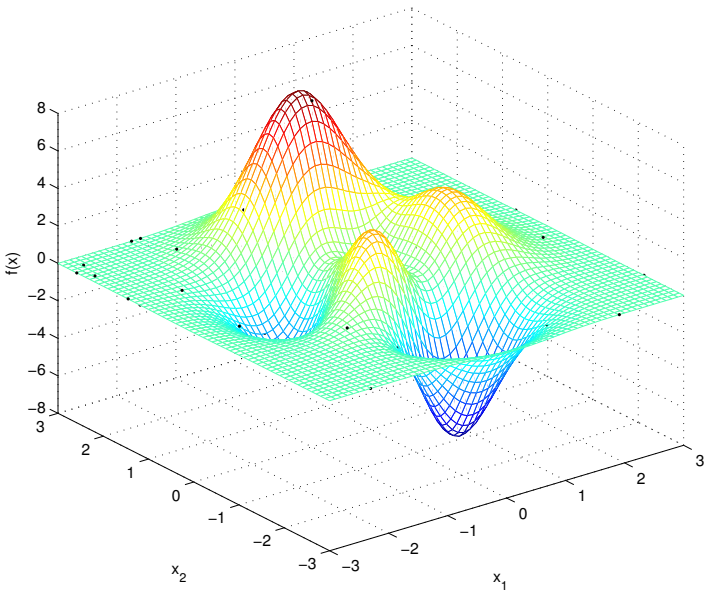


Abb. 7.3: Verteilung der Population erster Generation

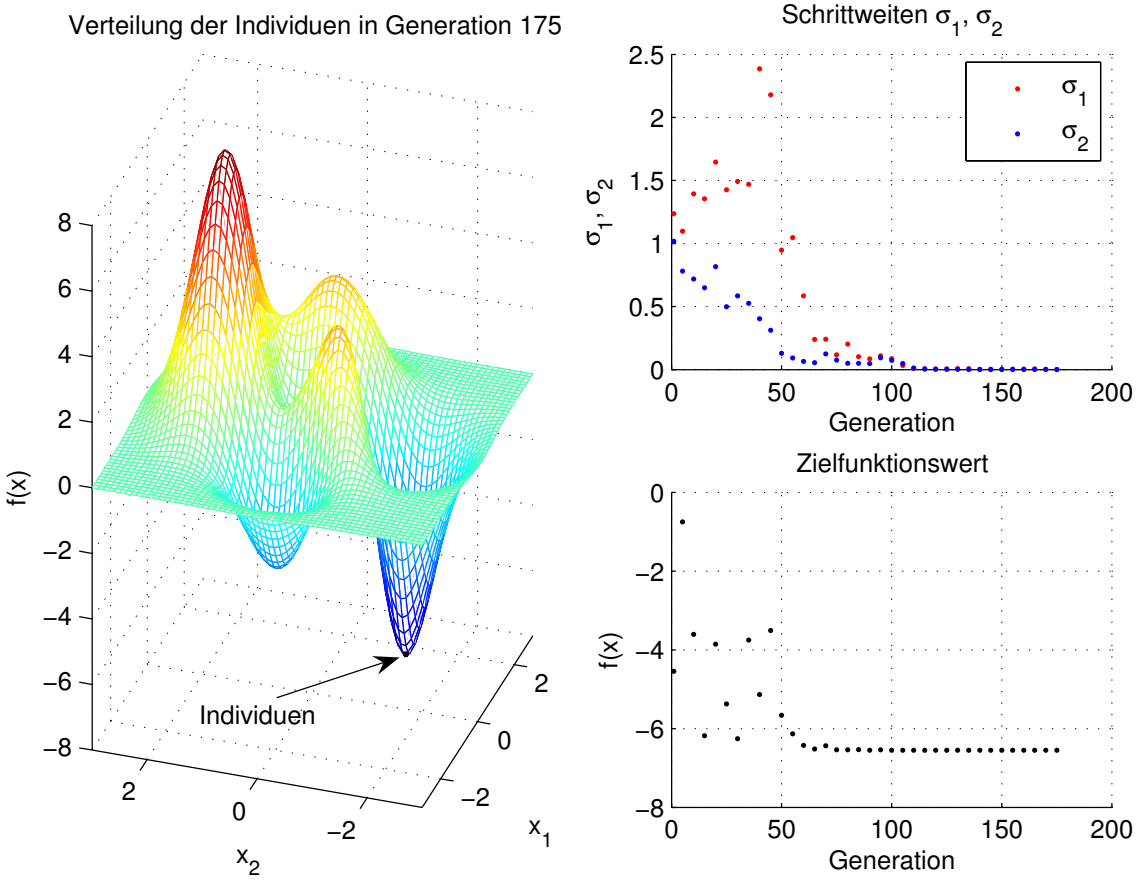


Abb. 7.4: Simulationsende Klassische Optimierung

7.4.2 Berücksichtigung von Robustheitsforderungen

Exemplarisch für die robuste Optimierung wurde die selber entworfene Testfunktion Markus (Abschnitt 2.3.1) herangezogen. Zur Berücksichtigung der Robustheitsforderungen wurde das in Abschnitt 6.4.3 besprochene Pattern Search Verfahren (rwcp) eingesetzt.

Belegung der Strategieparameter und anschließende Ausgabe von **EvolutionTool**:

```
testfunktion='markus';
n=2;          % Anzahl Optimierungsvariablen
xu=[0;0];     % Suchraum untere Grenze
xo=[2;2];     % Suchraum obere Grenze
typ='rwcp';  % Robuste Optimierung worst case pattern search
% Unsicherheitsbereich
delta_u=[0.15;0];
delta_o=[0.15;0];
mue=10;       % Zahl der Eltern
lambda=20;    % Anzahl Kinder
sigma=1;      % Anfangsschrittweite
tau_0=0.1; tau_1=0.15; % Nachfolger-, Dimensionsbeiwert
% Abbruchkriterium Quadratische Norm
epsilon_qn=1e-9;
% Vergleich bester und schlechtester Zielfunktionswert
epsilon_bs=1e-7;
% maximale Anzahl Generationen
genmax=500;
% Grafikausgabe (ja=1, nein=0)
grafik=1;
genstep=5;   % Schrittweite für Grafikausgabe

Ergebnis:

Optimaler Lösungsvektor:
     1.29257196007221
     1.00200780204806
Zielfunktionswert       :1.61475209306248
Funktionsaufrufe        :9240
Anzahl Generationen     :115
Quadrat.Norm der Pop.   :8.80153052859033e-010
Differenz best--worst   :8.44916506519410e-005
Rechenzeit in Sekunden  :0.64


kleinster je gefundener Zielfunktionswert: 1.61475149132161 in Generation: 114

zugehörige Optimierungsvariablen:
     1.29257197213597
     1.00198930341262
>>
```

Analyse der Resultate

Nicht direkt vergleichbar mit dem vorigen Szenario aber doch klar ersichtlich ist die um den Faktor 3 gesteigerte Anzahl von 9240 Funktionsaufrufen, die aus der Berücksichtigung der Robustheitsforderungen resultiert.

Abbildung 7.5 zeigt einen »Schnappschuss« des Evolutionsprozesses in Generation 40. Die Individuen beginnen sich bereits leicht im globalen Minimum zu sammeln, erkennbar an der Angleichung der beiden Schrittweiten σ_1 und σ_2, und auch der Zielfunktionswert nähert sich bereits dem optimalen Wert von ca. 1.6.

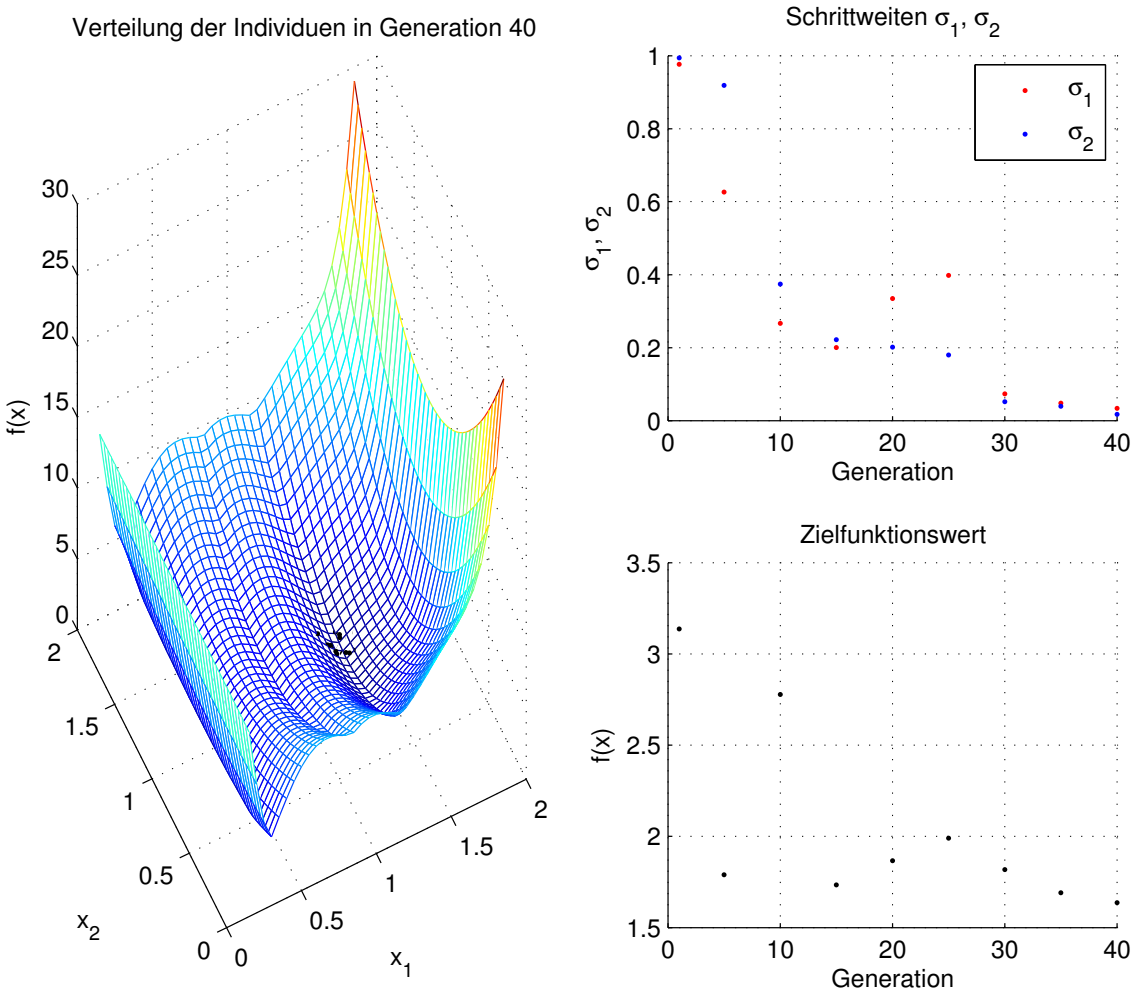


Abb. 7.5: Momentaufnahme Evolutionsprozess in Generation 40

Das Evolutionsende ist in Generation 115 erreicht, das durch eine starke punktförmige Konzentration der Individuen im Optimum charakterisiert ist. Grafik 7.6 zeigt einen dreidimensionalen Blick von unten durch die von x_1 und x_2 aufgespannte Grundrissebene in richtiger positiver Funktionswerte.

Vergleicht man die gefundene optimale Lösung in Generation 115, die sich nur minimal vom kleinsten Wert in Generation 114 unterscheidet, mit dem durch theoretische Überlegungen ermittelten Wert $\gamma_{\text{global}}(\mathbf{x})$ in Tabelle 5.2 für $\delta_1 = 0.15$, so sind die Ergebnisse praktisch identisch.

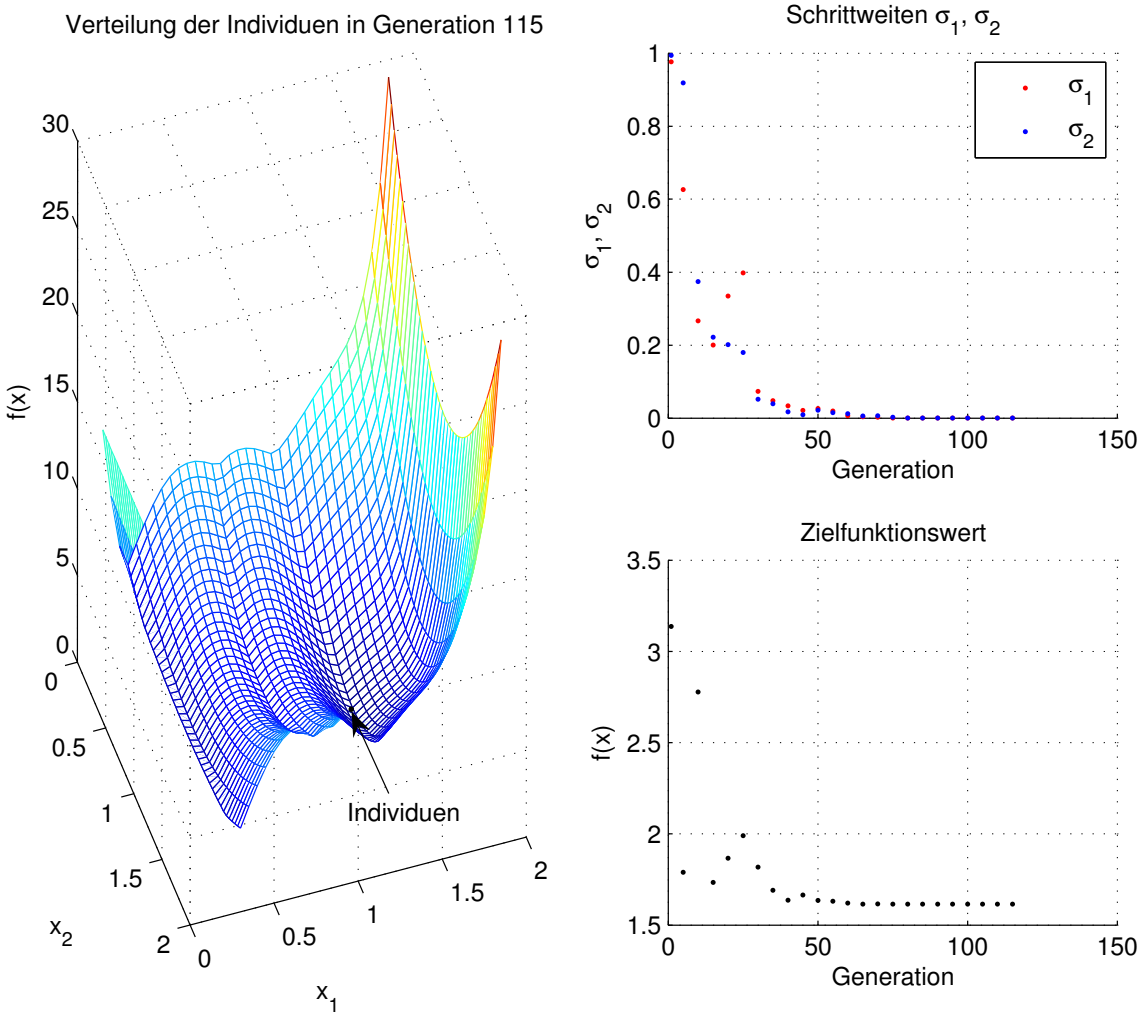


Abb. 7.6: Evolutionsende Robuste Optimierung

8 Schluss, Resümee

— Des Lernens ist kein Ende. —
Robert Schumann

8.1 Zusammenfassung

In dieser Diplomarbeit wurde versucht, die Vorteile der robusten Problemformulierung, bei der Unsicherheiten und Störungen der Designvariablen in den Entwurfsprozess miteinbezogen werden, gegenüber der a posteriori Sensitivitätsanalyse hervorzuheben. Um Robustheitsforderungen quantitativ zu beschreiben wurden verschiedene Unsicherheitsmodelle vorgestellt, die es ermöglichen, angewendet auf die Zielfunktion und die Nebenbedingungen, ein vollständig robustes mathematisches Programm zu formulieren.

Die Einbeziehung von Robustheitsforderungen in den Entwurfsprozess führt zu Auswirkungen auf die Zielfunktion (Differenzierbarkeit, Lage der Maxima und Minima, etc.) und hat somit weitreichende Konsequenzen bezüglich der Wahl geeigneter Optimierungsalgorithmen. Verschiedene Optimierungsstrategien wurden diskutiert, miteinander verglichen und schließlich den einfach implementierbaren Evolutionsstrategien der Vorzug gegeben.

Im Bereich der Evolutionsstrategien wurde die für gutes globales Verhalten verantwortliche Größe des Selektionsdrucks eingeführt und die Vorteile der rangbasierten

Selektion gegenüber fitnessbasierter Selektionsmechanismen betont. Aber auch die für stabile und schnelle Konvergenzgeschwindigkeit verantwortliche dynamische Adaption der Schrittweite wurde besprochen.

Um die hohe Zahl der Funktionsaufrufe, an der alle stochastischen Algorithmen prinzipbedingt leiden, nicht noch weiter zu erhöhen, wurden praktische Implementierungen analysiert, die in der Lage sind, das bei der robusten Formulierung auftretende Subproblem möglichst effizient zu lösen.

8.2 Ausblick – Trends und Perspektiven

Um robuste Optimierungsprobleme zu lösen, werden geeignete »Direct Search« Methoden benötigt. In diese Klasse fallen auch prinzipiell[1] evolutionäre Algorithmen, die somit ebenfalls prädestiniert sind derartige Probleme erfolgreich zu lösen.

Viele Optimierungsprobleme, die in der Literatur beschrieben werden, sind niedrigdimensional, das bedeutet höchstens von der Ordnung zehn. Unter solchen Voraussetzungen funktionieren das Simplex Verfahren, die Pattern Search Methode, Response Surface Modeling Verfahren, etc. natürlich sehr gut. Andererseits gibt es mittlerweile eine wachsende Zahl von Problemstellungen, bei denen die Zahl der Designvariablen rasant wächst.

Aus diesem Grund gibt es einige Diskussionspunkte und Fragestellungen, die in Angriff genommen werden müssen, um Robuste Optimierung zur Regel und nicht zur Ausnahme im ingenieurmäßigen Entwurfsprozess zu machen [5].

- Welche Direct Search Methoden sollen für die jeweilige Klasse von Anwendungsfällen eingesetzt werden. Um dies zufriedenstellend beantworten zu können müssen entsprechende Testszenarien für robustes Design entworfen und im Detail analysiert werden.

- Wie kann die Miteinbeziehung der Nebenbedingungen in den Entwurfsprozess effizient bewerkstelligt werden, und wie interagieren beide Robustheitsforderungen theoretisch und praktisch für einen Satz von charakteristischen Testfunktionen.

- Wie hängt die Leistung (Performance) eines Algorithmus im Allgemeinen von Robustheitsforderungen ab, und gibt es weitere Möglichkeiten Robustheitsforderungen durch Adaption während des Iterationsprozesses zu implementieren.

[1] prinzipiell bedeutet, dass versucht wird das evolutionäre Verhalten der Natur zu imitieren. In diese Gruppe fallen somit Evolutionsstrategien und Genetische Algorithmen.

A Anhang

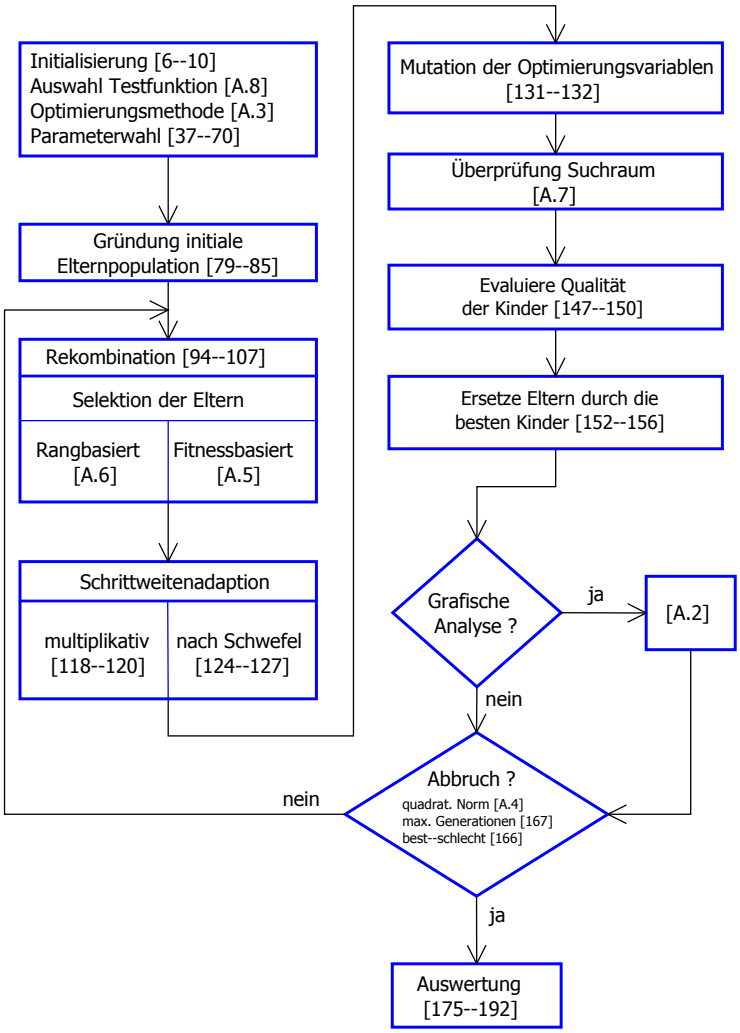


Abb. A.1: Flussdiagramm der $(\mu/\rho, \lambda)$ - Evolutionsstrategie[1]

[1] in eckigen Klammern sind entweder Codezeilen des Evolutionsalgorithmus **EvolutionTool** [A.1] oder programmierte MATLAB®-Funktionen [A.2, A.3, A.4, A.5, A.6, A.7, A.8] angeführt.

Tab. A.1: Variablen der $(\mu/2, \lambda)$ - ES für $n = 2$, $\mu = 10$ und $\lambda = 40$

Variable	Dimension	Bytes	Beschreibung
a	1x1	8	Gewichtungsfaktor für intermediäre Rekombination
alpha	1x1	8	Faktor multiplikative Methode
anzahl	1x1	8	Gesamtanzahl Funktionsaufrufe
best	1x1	8	bester Zielfunktionswert pro Generation
bs	1x1	8	Differenz bester, schlechtester Funktionswert
delta_o	2x1	16	maximaler Abweichungsvektor obere Grenze
delta_u	2x1	16	maximaler Abweichungsvektor untere Grenze
e1	1x1	8	Für Rekombination selektierter Elternteil 1
e2	1x1	8	Für Rekombination selektierter Elternteil 2
elter	2x1	16	Elternteil zufällig innerhalb Suchraum positioniert
elternpop	5x10	400	gesamte Elternpopulation
epsilon_bs	1x1	8	Abbruchkriterium bester, schlechtester Funktionswert
epsilon_qn	1x1	8	Abbruchkriterium Quadratische Norm
f_aufrufe	1x1	8	Zusätzliche Funktionsaufrufe pro Subproblem
f_wert	1x1	8	berechneter Zielfunktionswert
faktor_0	40x1	320	Hilfsvektor für Schrittweitenadaption nach Schwefel
faktor_1	2x1	16	Hilfsvektor für Schrittweitenadaption nach Schwefel
faktoren	2x40	640	Matrix aus zufälliger Folge von +1 und -1
faktoren_opt	2x40	640	Matrix für Mutation der Optimierungsvariablen
faktoren_sigma	2x40	640	Matrix für Mutation der Schrittweiten
fbest	1x1	8	kleinster je gefundener Zielfunktionswert
gen	1x1	8	Anzahl Generationen
genmax	1x1	8	Abbruchkriterium Generationenanzahl
genstep	1x1	8	Schrittweite (Generationen) für Grafikausgabe
grafik	1x1	8	Grafikausgabe (ja=1, nein=0)
i	1x1	8	Schleifenvariable
index	1x1	8	Indexvariable
kind1	4x1	32	Durch Rekombination generiertes Kind 1
kind2	4x1	32	Durch Rekombination generiertes Kind 2
kinderpop	5x40	1600	gesamte Kinderpopulation
lambda	1x1	8	Anzahl Kinder
mue	1x1	8	Zahl der Eltern
n	1x1	8	Anzahl Optimierungsvariablen
qn	1x1	8	Quadratische Norm der Population
qualitaet	1x1	1	Ergebnis (true,false) der Abbruchbedingungen
rad	1x10	80	Roulette-Rad (für Selektion Eltern)
rechenzeit	1x1	8	Rechenzeit in [s]
sigma	1x1	8	Anfangsschrittweite
tau_0	1x1	8	Nachfolgerbeiwert (Schrittweite nach Schwefel)
tau_1	1x1	8	Dimensionsbeiwert (Schrittweite nach Schwefel)
testfunktion	1x5	10	Auswahl Testfunktion
typ	1x2	4	Art der Optimierung (nr,rwcp,rwce,rwcu,rmw)
worst	1x1	8	schlechtester Zielfunktionswert pro Generation
wuerfel	2x1	16	Zufallsvektor im Intervall [0..1]
xo	2x1	16	Suchraum (Entwurfsraum) obere Grenze

```
Variable     Dimension   Bytes          Beschreibung
-------------------------------------------------------------------------------

xu               2x1       16  Suchraum (Entwurfsraum) untere Grenze
ziel             1x40     320  sortierte Funktionswerte pro Kindergeneration
zielbest         5x24     960  beste Individuen über alle [24] Generationen
```

Algorithmus A.1 $(\mu/\rho, \lambda)$ - Evolutionsstrategie **EvolutionTool**

```
 1 %   (mue/rho,lambda)-Evolutionsstrategie
 2 %            EvolutionTool
 3 %
 4 % Initialisierung (keine Veränderung notwendig)
 5 % ================================================================================
 6 clear all;close all;clc;
 7 qualitaet=0;
 8 zielbest=[];
 9 anzahl=0;       % Anzahl Funktionsaufrufe
10 gen=0;          % Anzahl Generationen
11
12 % Auswahl Testfunktion
13 % ================================================================================
14 testfunktion='peaks';
15 % Anzahl Optimierungsvariablen
16 n=2;
17 % Suchraum (Entwurfsraum) untere Grenze
18 xu=[-3;-3];
19 % Suchraum (Entwurfsraum) obere Grenze
20 xo=[3;3];
21
22 % Art der Optimierung
23 % ================================================================================
24 typ='nr';         % Nicht robuste Optimierung
25 %typ='rwcp';      % Prädiktion Pattern Search
26 %typ='rwce';      % Auswertung aller Ecken
27 %typ='rwcu';      % vollständige Diskretisierung von U(x)
28 %typ='rmw';       % robuste Analyse mit Mittelwertmethode
29
30 % Unsicherheitsbereich
31 delta_u=[0;0];
32 delta_o=[0;0];
33
34 % Parameter Optimierungsalgorithmus
35 % ================================================================================
36 % Zahl der Eltern
37 mue=10;
38 % Anzahl Kinder
39 lambda=40;
40 % Anfangsschrittweite
```

```
41 sigma=1;
42 % Schrittweitenadaption
43 %alpha=1.05; % Faktor multiplikative Methode
44 tau_0=0.1;    % Nachfolgerbeiwert
45 tau_1=0.15;  % Dimensionsbeiwert
46
47 % Abbruchkriterium Quadratische Norm
48 % setze '0', wenn Kriterium nicht beachtet werden soll
49 epsilon_qn=1e-9;
50 % Vergleich bester und schlechtester Zielfunktionswert
51 % setze '0', wenn Kriterium nicht beachtet werden soll
52 epsilon_bs=1e-7;
53 % maximale Anzahl Generationen
54 % setze 'Inf', wenn Kriterium nicht beachtet werden soll
55 genmax=500;
56
57 % Grafikausgabe (für 2 und 3 dimensionale Probleme)
58 % ================================================================================
59 % Grafikausgabe (ja=1, nein=0)
60 grafik=0;
61 % Schrittweite für Grafikausgabe
62 genstep=1;
63
64 % Anpassung untere, obere Grenze Suchraum
65 % ================================================================================
66 if strcmp(typ,'nr') == 0
67       xu=xu+delta_u;
68       xo=xo-delta_o;
69 end
70
71 % Ende Initialisierung und Parameterwahl
72 % ================================================================================
73 % ================================================================================
74 % Starte Optimierung
75
76 tic; % Beginn Zeitnehmung
77
78 % Gründung initiale Elternpopulation
79 elternpop=[];
80 for i=1:mue
81    elter = [xu + (xo-xu).*rand(n,1)];
82    [f_wert f_aufrufe]=robust(elter,testfunktion,n,delta_u,delta_o,typ);
83    elternpop=[elternpop,[elter;sigma*ones(n,1);f_wert]];
84    anzahl=anzahl+f_aufrufe;
85 end
86
87 while qualitaet==0
88
89    % Gründung Kinderpopulation
90
91    kinderpop=[];
```

```
92      % Erstelle Roulette-Rad
93      % [rad]=roulettefit(elternpop(2*n+1,:));
94      [rad]=rouletterank(elternpop(2*n+1,:));
95
96      for i=1:2:lambda
97
98          wuerfel=rand(2,1);% Drehe Roulette-Rad 2 mal
99          e1=find(rad>wuerfel(1,1),1,'first');
100         e2=find(rad>wuerfel(2,1),1,'first');
101         a=0.8+0.5*randn(1,1);
102         % Intermediäre Rekombination
103         kind1=a*elternpop(1:2*n,e1)+(1-a)*elternpop(1:2*n,e2);
104         kind2=a*elternpop(1:2*n,e2)+(1-a)*elternpop(1:2*n,e1);
105         kinderpop=[kinderpop,[kind1;eps],[kind2;eps]];
106
107     end
108
109     % Mutation der Kinderpopulation
110
111     % 1...Adaption der Schrittweite
112
113         % (a) Multiplikative Methode
114         % die Schrittweiten werden zufällig mit alpha multipliziert
115         % oder durch alpha dividiert
116         % Matrix 'faktoren' besteht aus zufälliger Folge von +1 und -1
117         % alpha^-1 == 1/alpha und alpha^1 == alpha
118 %       faktoren=sign(round(unifrnd(-1,0,n,lambda))+0.1*ones(n,lambda));
119 %       faktoren_sigma=alpha.^faktoren;
120 %       kinderpop(n+1:2*n,:)=kinderpop(n+1:2*n,:).*faktoren_sigma;
121
122         % (b) nach Schwefel[23], konvergiert schneller
123
124         faktor_0=exp(tau_0*randn(lambda,1));
125         faktor_1=exp(tau_1*randn(n,1));
126         faktoren_sigma=faktor_1*faktor_0';
127         kinderpop(n+1:2*n,:)=kinderpop(n+1:2*n,:).*faktoren_sigma;
128
129     % 2...Mutation der Optimierungsvariablen
130
131         faktoren_opt=randn(n,lambda);
132         kinderpop(1:n,:)=kinderpop(1:n,:)+kinderpop(n+1:2*n,:).*faktoren_opt;
133
134     % Überprüfung ob Kinder innerhalb Suchraum
135     % falls außerhalb, bringe sie zurück an Rand
136     [kinderpop]=testsuchraum(kinderpop,xu,xo,lambda,n);
137
138     % Berechne für mutierte Kinder neue Zielfunktionswerte
139     for i=1:lambda
140         [f_wert f_aufrufe]=robust(kinderpop(1:n,i),testfunktion,n, ...
141                                   delta_u,delta_o,typ);
142         kinderpop(2*n+1,i)=f_wert;
```

```
143         anzahl=anzahl+f_aufrufe;
144     end
145 
146     % Qualität der Kinder
147     [ziel,index]=sort(kinderpop(2*n+1,:));
148     best=kinderpop(2*n+1,index(1)); % bester Zielfunktionswert
149     worst=kinderpop(2*n+1,index(end));% schlechtester Zielfunktionswert
150     index=index(1:mue);
151 
152     elternpop=[];% Eltern sterben
153     % Ersetze Eltern durch die besten mue Kinder
154     for i=1:mue
155         elternpop=[elternpop,kinderpop(:,index(i))];
156     end
157     gen=gen+1;% neue Generation
158 
159     % Ausgabe Grafik, falls erwünscht
160     if (grafik==1) & (gen==1 | floor(gen/genstep)==(gen/genstep))
161         grafikanalyse(kinderpop,testfunktion,xu,xo,delta_u,delta_o,gen,n,typ);
162     end
163 
164     % Abbruchbedingung(en) erfüllt ?
165     qn=quadnorm(kinderpop(1:n,:),n,lambda);
166     bs=abs(abs(best)-abs(worst));
167     qualitaet=(qn < epsilon_qn) | (bs < epsilon_bs) | (gen>=genmax);
168     % bestes Individuum pro Generation
169     zielbest=[zielbest,kinderpop(:,index(1))];
170 
171 end % while Schleife
172 
173 % Auswertung
174 % ================================================================================
175 rechenzeit=toc; % Ende Zeitnehmung
176 disp('Ergebnis:');
177 fprintf('\n');
178 disp('Optimaler Lösungsvektor: ');
179 disp(kinderpop(1:n,index(1)));
180 disp(['Zielfunktionswert     :', num2str(best)]);
181 disp(['Funktionsaufrufe      :', num2str(anzahl)]);
182 disp(['Anzahl Generationen   :', num2str(gen)]);
183 disp(['Quadrat.Norm der Pop. :', num2str(qn)]);
184 disp(['Differenz best--worst :', num2str(bs)]);
185 disp(['Rechenzeit in Sekunden :', num2str(rechenzeit)]);
186 [fbest,index]=min(zielbest(2*n+1,:));
187 fprintf('\n\n');
188 disp(['kleinster je gefundener Zielfunktionswert: ',num2str(fbest), ...
189       ' in Generation: ',num2str(index)]);
190 fprintf('\n');
191 disp('zugehörige Optimierungsvariablen: ');
192 disp(zielbest(1:n,index))
```

Algorithmus A.2 grafikanalyse

```
function grafikanalyse(kinderpop,testfunktion,xu,xo,delta_u,delta_o,gen,n,typ)
%
% Grafische Analyse (für Dimension n=2 und n=3)
%
  switch n

    case 2 % Dimension 2
        subplot(2,2,[1 3]);
        [x1,x2]=meshgrid([xu(1,1):0.05:xo(1,1)],[xu(2,1):0.05:xo(2,1)]);
        [zeile,spalte]=size(x1);
        f=[];
        for i=1:zeile
            for j=1:spalte
               f(i,j)=robust([x1(i,j);x2(i,j)],testfunktion, ...
                             n,delta_u,delta_o,typ);
            end
        end
        mesh(x1,x2,f);hold on;
        plot3(kinderpop(1,:),kinderpop(2,:),kinderpop(2*n+1,:),'k.');hold off;
        title(['Verteilung der Individuen in Generation ',num2str(gen)]);
        xlabel('x_1');ylabel('x_2');zlabel('f(x)');
        subplot(2,2,2);
        [y,index]=min(kinderpop(2*n+1,:));hold on;
        plot(gen,kinderpop(n+1,index),'r.',gen, ...
             kinderpop(n+2,index),'b.');grid on;
        title('Schrittweiten \sigma_1, \sigma_2');
        xlabel('Generation');ylabel('\sigma_1, \sigma_2');
        legend('\sigma_1','\sigma_2');

    case 3 % Dimension 3
        subplot(2,2,[1 3]);
        plot3(kinderpop(1,:),kinderpop(2,:),kinderpop(3,:),'k.');grid;
        axis([xu(1,1) xo(1,1) xu(2,1) xo(2,1) xu(3,1) xo(3,1)]);
        title(['Verteilung der Individuen Generation ',num2str(gen)]);
        xlabel('x_1');ylabel('x_2');zlabel('x_3');
        subplot(2,2,2);
        [y,index]=min(kinderpop(2*n+1,:));hold on;
        plot(gen,kinderpop(n+1,index),'r.',gen, ...
             kinderpop(n+2,index),'b.',gen,kinderpop(n+3,index),'k.');grid on;
        title('Schrittweiten \sigma_1, \sigma_2, \sigma_3');
        xlabel('Generation');
       ylabel('\sigma_1, \sigma_2, \sigma_3');
        legend('\sigma_1','\sigma_2','\sigma_3');
  end
subplot(2,2,4);hold on;
plot(gen,kinderpop(2*n+1,index),'k.');grid on;
title('Zielfunktionswert');xlabel('Generation');ylabel('f(x)');
pause;
end
```

Algorithmus A.3 robust

```
function[f_wert f_aufrufe]=robust(x,testfunktion,n,delta_u,delta_o,typ)
%
% verschiedene Diskretisierungsmethoden
%
 switch typ

% ================================================================================
   case 'nr'   % nicht robuste Analyse

        f_wert=testfunk(x,testfunktion);
        f_aufrufe=1;

% ================================================================================
   case 'rwcp' % Prädiktion mit Pattern Search

        x_schlecht=x;
          for j=1:n
             x_plus=x_schlecht;
             x_plus(j)=x_plus(j)+delta_o(j);
             x_minus=x_schlecht;
             x_minus(j)=x_minus(j)-delta_u(j);
             f_plus=testfunk(x_plus,testfunktion);
             f_wert=testfunk(x_minus,testfunktion);
                 if f_plus > f_wert
                      x_schlecht=x_plus;
                      f_wert=f_plus;
                 else
                      x_schlecht=x_minus;
                 end
          end
        f_aufrufe=2*n;

% ================================================================================
   case 'rwce' % Auswertung aller Eckpunkte

        vertex_lu=x-delta_u; % Ecke ganz links unten
        vertex_ro=x+delta_o; % Ecke ganz rechts oben
        vertex_diff=delta_u+delta_o;
        anzahl_ecken=2^n-2; % zusätzlich berechnete Ecken
        f_ecken=[testfunk(vertex_lu,testfunktion), ...
                 testfunk(vertex_ro,testfunktion)];
          for j=1:anzahl_ecken
               % Berechne eine Ecke des Hyperwürfels
               vertex_kubus=vertex_lu+vertex_diff.*str2num(dec2bin(j,n)');
               f_ecken=[f_ecken,testfunk(vertex_kubus,testfunktion)];
          end
        f_aufrufe=anzahl_ecken+2;
        f_wert=max(f_ecken);

```

```
% ========================================================================
   case 'rwcu' % vollständige Diskretisierung von U(x) nur für Dimension 2

         xu=x-delta_u;xo=x+delta_o;
         [x1,x2]=meshgrid([xu(1,1):0.005:xo(1,1)],[xu(2,1):0.005:xo(2,1)]);
         [zeile,spalte]=size(x1);
         f=[];
            for i=1:zeile
                  for j=1:spalte
                       f(i,j)=testfunk([x1(i,j);x2(i,j)],testfunktion);
                  end
            end
         f_wert=max(max(f));
         f_aufrufe=zeile*spalte;

% ========================================================================
   case 'rmw' % robuste Analyse mit Mittelwertmethode nur für Dimension 2

         xu=x-delta_u;xo=x+delta_o;
         [x1,x2]=meshgrid([xu(1,1):0.005:xo(1,1)],[xu(2,1):0.005:xo(2,1)]);
         [zeile,spalte]=size(x1);
         f=[];
            for i=1:zeile
                  for j=1:spalte
                       f(i,j)=testfunk([x1(i,j);x2(i,j)],testfunktion);
                  end
            end
         f_wert=mean(mean(f));
         f_aufrufe=zeile*spalte;

  end
end
```

Algorithmus A.4 quadnorm

```
function[qn]=quadnorm(pop,n,lambda)
%
% Berechnung der quadratischen Norm der Population
%
xq=mean(pop,2);
qn=0;
for j=1:n
    for i=1:lambda
          qn=qn+((pop(j,i)-xq(j,1))/xq(j,1))^2;
    end
end
qn=qn/(lambda*n);
end
```

Algorithmus A.5 roulettefit

```
function[rad]=roulettefit(elter)
%
% Fitnessproportionale Selektion mit Roulette Rad Methode
% funktioniert nur bei positiven Zielfunktionswerten
%
kehrwert=1./elter;
g=sum(kehrwert);
fitness=kehrwert/g;
rad=cumsum(fitness); % Kumulierte Summe
end
```

Algorithmus A.6 rouletterank

```
function[rad]=rouletterank(elter)
%
% Rangbasierte Selektion mit Roulette Rad Methode
% funktioniert auch bei negativen Zielfunktionswerten
%
[y,index]=sort(elter,'descend'); % sortiere in absteigender Reihenfolge
[y,fitness]=sort(index); % sortiere Index in aufsteigender Reihenfolge
g=0.5*(y(1)+y(end))*length(y); % Summenbildung arithmetische Reihe
wkt=fitness/g;
rad=cumsum(wkt); % Kumulierte Summe
end
```

Algorithmus A.7 testsuchraum

```
function[kinderpop]=testsuchraum(kinderpop,xu,xo,lambda,n)
%
% falls mutierte Kinder außerhalb des Suchraums
% bringe sie wieder zurück an den Rand
%
for kind=1:lambda
    for zeile=1:n
        if (kinderpop(zeile,kind)<xu(zeile,1))
            kinderpop(zeile,kind)=xu(zeile,1);
        end
        if (kinderpop(zeile,kind)>xo(zeile,1))
            kinderpop(zeile,kind)=xo(zeile,1);
        end
    end
end
end
```

Algorithmus A.8 testfunk

```
function[f]=testfunk(x,testfunktion)
%
% Auswahl der Testfunktionen
%
  switch testfunktion

    case 'sphäre'

        f=x(1)^2+x(2)^2;

    case 'rosenbrock'

        f=100*(x(2)-x(1)^2)^2+(1-x(1))^2;

    case 'markus'

        f=48.8*x(1)^7-305*x(1)^6+704.3*x(1)^5-681*x(1)^4 ...
          +139*x(1)^3+178*x(1)^2-94*x(1)+6*x(1)*(x(2)-1)^2+13;

    case 'peaks'

        f=3*(1-x(1))^2*exp(-(x(1)^2)-(x(2)+1)^2) ...
          -10*(x(1)/5-x(1)^3-x(2)^5)*exp(-x(1)^2-x(2)^2) ...
          -1/3*exp(-(x(1)+1)^2-x(2)^2);

    case 'himmelblau'

        f=(x(1)^2+x(2)-11)^2+(x(1)+x(2)^2-7)^2;

    case 'rastrigin'

        f=20+x(1)^2+x(2)^2-10*(cos(2*pi*x(1))+cos(2*pi*x(2)));

  end
end
```

Verzeichnisse

Literaturverzeichnis

[1] Antoniou, A. ; Lu, W.-S. : *Practical Optimization - Algorithms and Engineering Applications*. 1. Auflage. Berlin, Heidelberg : Springer, 2007. – ISBN 978–0–387–71106–5 6, 36

[2] Bellman, R. E.: *Dynamic Programming*. Neuauflage. Mineola, New York : Courier Dover Publications, 2003. – ISBN 978–0–486–42809–3 36

[3] Ben-Tal, A. ; Nemirovski, A. : Robust Optimization — Methodology and Applications. In: *Mathematical Programming* 92 (2002), Nr. 3, Ser. B, S. 453–480. http://dx.doi.org/10.1007/s101070100286. – DOI 10.1007/s101070100286. – ISSN 0025–5610. – ISMP 2000, Part 2 (Atlanta, GA) 23

[4] Beucher, O. : *MATLAB und Simulink - Grundlegende Einführung für Studenten und Ingenieure in der Praxis*. 4. aktualisierte Auflage. München : Pearson Deutschland GmbH, 2008. – ISBN 978–3–827–37340–3 42

[5] Beyer, H.-G. ; Sendhoff, B. : Robust Optimization – A Comprehensive Survey. In: *Computer Methods in Applied Mechanics and Engineering* 196 (2007), März, Nr. 33-34, S. 3190–3218. http://dx.doi.org/10.1016/j.cma.2007.03.003. – DOI 10.1016/j.cma.2007.03.003. – ISSN 0045–7825 64

[6] Dantzig, G. B. ; Thapa, M. N.: *Linear Programming: 1: Introduction*. Springer, 1997 (Springer Series in Operations Research). – 435 S. – ISBN 978–0–387–94833–1 18

[7] Demjanov, V. F. ; Malozemov, V. N.: On the Theory of Nonlinear Minimax Problems. In: *Russian Mathematical Society* 26 (1971), S. 57–115 24

[8] Du, X. ; Chen, W. : Towards a better Understanding of Modeling Feasibility Robustness in Engineering Design. In: *Transactions-American Society of Mechanical Engineers Journal of Mechanical Design* 122 (2000), Nr. 4, S. 385–394 27

[9] Fletcher, R. : *Practical Methods of Optimization*. 2. Auflage. New York : Wiley, 2000. – ISBN 978–0–471–49463–8 27, 36

[10] Gauss, C. F.: *Theory of the Motion of the Heavenly Bodies Moving about the Sun in Conic Sections - Eine Übersetzung von Gauß' "Theoria Motus"*. New York : Little, Brown, 1857 3

[11] Kofler, M. : *Linux - Installation, Konfiguration, Anwendung*. 5. Auflage. Amsterdam : Addison-Wesley, 2000. – ISBN 3–8273–1658–8 IV

[12] Kost, B. : *Optimierung mit Evolutionsstrategien*. 1. Auflage. Thun, Frankfurt am Main : Harri Deutsch Verlag, 2003. – ISBN 978–3–817–11699–7 54

[13] Krefft, M. : *Aufgabenangepasste Optimierung von Parallelstrukturen für Maschinen in der Produktionstechnik*. 1. Auflage. Essen : Vulkan-Verlag, 2006. – ISBN 978–3–802–78689–1 53

[14] Lenstra, J. ; Rinnooy-Kan, A. ; Schrijver, A. : *History of mathematical programming: a collection of personal reminiscences*. 1. Auflage. Amsterdam, New York : CWI, 1991. – 141 S. – ISBN 978–0–444–88818–1 16

[15] Lyx: *LyX – die bessere Textverarbeitung*. `http://www.lyx.org/WebDe.Home` IV

[16] Magele, C. ; Ebner, T. ; Baumgartner, U. : *Numerical Optimization*. Vorlesungsskript, 2008 4

[17] Mühlenbein, H. ; Schomisch, D. ; Born, J. M.: The Parallel Genetic Algorithm as Function Optimizer. In: *Parallel Computing* 17 (1991), September, Nr. 6-7, S. 619–632. `http://dx.doi.org/10.1016/S0167-8191(05)80052-3`. – DOI 10.1016/S0167–8191(05)80052–3. – ISSN 0167–8191 13

[18] Papageorgiou, M. : *Optimierung - Statische, dynamische, stochastische Verfahren für die Anwendung*. Erweiterte und verbesserte Auflage. Deutschland : Oldenbourg, 1996. – ISBN 978–3–486–23775–7 4, 39

[19] Petersen, K. B. ; Pedersen, M. S.: *The Matrix Cookbook*. Oktober 2008. `http://www2.imm.dtu.dk/pubdb/p.php?3274` 4

[20] Ramakrishnan, B. ; Rao, S. : A Robust Optimization Approach using Taguchi's loss Function for solving Nonlinear Optimization Problems. In: *Advances in Design Automation* 32 (1991), Nr. 1, S. 241–248 25

[21] Rechenberg, I. : *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart : Frommann-Holzboog, 1973. – ISBN 978–3–772–80373–4 54

[22] Saltelli, A. ; Ratto, M. ; Andres, T. ; Campolongo, F. ; Cariboni, J. ; Gatelli, D. ; Saisana, M. ; Tarantola, S. : *Global Sensitivity Analysis - The Primer*. 1. Auflage. West Sussex : John Wiley, 2008. – ISBN 978–0–470–05997–5 5

[23] Schwefel, H.-P. : *Evolution and Optimum Seeking*. New York : Wiley, 1995. – 444 S. – ISBN 978–0–471–57148–3 49, 54, 56

[24] Silagadze, Z. : Finding Two-Dimensional Peaks. In: *Physics of Particles and Nuclei Letters* 4 (2007), Nr. 1, S. 73–80. http://dx.doi.org/10.1134/S154747710701013X. – DOI 10.1134/S154747710701013X. – ISSN 1547–4771 8

[25] Steiner, G. ; Weber, A. ; Magele, C. : Managing Uncertainties in Electromagnetic Design Problems with Robust Optimization. In: *Magnetics, IEEE Transactions on Magnetics* 40 (2004), März, Nr. 2, S. 1094 – 1099. http://dx.doi.org/10.1109/TMAG.2004.824556. – DOI 10.1109/TMAG.2004.824556. – ISSN 0018–9464 23

[26] Stowasser, J. ; Petschenig, M. ; Skutsch, F. : *Der kleine Stowasser*. Wien : Hölder-Pichler-Tempsky, 1991. – ISBN 978–3–209–00225–9 6

[27] Sun, W. ; Yuan, Y.-X. : *Optimization Theory And Methods - Nonlinear Programming*. 1. Auflage. Berlin, Heidelberg : Springer, 2006. – ISBN 978–0–387–24975–9 8

[28] Sundaresan, S. ; Ishii, K. ; Houser, D. R.: A Robust Optimization Procedure with Variations on Design Variables and Constraints. In: *Engineering Optimization* 24 (1995), Nr. 2, S. 101–117. http://dx.doi.org/10.1080/03052159508941185. – DOI 10.1080/03052159508941185 24, 41

[29] Swann, W. H.: Direct Search Methods. In: Murray, W. (Hrsg.): *Numerical Methods for Unconstrained Optimization*, Academic Press, London, 1972, S. 13–28 36, 44

[30] Wikipedia: *Ipe – Vektorzeichenprogramm*. http://de.wikipedia.org/w/index.php?title=Ipe_(Zeichenprogramm)&oldid=81975881. – [Online; Stand 8. Dezember 2012] IV

[31] Wikipedia: *JabRef – freies Literaturverwaltungsprogramm für BibTeX Format*. http://de.wikipedia.org/w/index.php?title=JabRef&oldid=112907504. – [Online; Stand 7. Februar 2013] IV

[32] Yoon, S.-B. ; Jung, I.-S. ; Hyun, D.-S. ; Hong, J.-P. ; Kim, Y.-J. : Robust shape optimization of electromechanical devices. In: *Magnetics, IEEE Transactions on Magnetics* 35 (1999), Mai, Nr. 3, S. 1710 –1713. http://dx.doi.org/10.1109/20.767356. – DOI 10.1109/20.767356. – ISSN 0018–9464 25

Abkürzungen, Symbole

Abkürzungen

DOI	**D**igital **O**bject **I**dentifier
EA	**E**volutionärer **A**lgorithmus
ES	**E**volutions**s**trategie
GA	**G**enetischer **A**lgorithmus
GN	**G**auß-**N**ewton
IGTE	**I**nstitut für **G**rundlagen und **T**heorie der **E**lektrotechnik
ISBN	**I**nternational **S**tandard **B**ook **N**umber
ISSN	**I**nternational **S**tandard **S**erial **N**umber
LP	**L**ineare **P**rogrammierung
MP	**M**athematische **P**rogrammierung
nr	**n**icht **r**obust
NSO	**N**on-**S**mooth **O**ptimization
PSO	**P**article **S**warm **O**ptimierung
rmw	**r**obust **M**ittel**w**ert
RO	**R**obuste **O**ptimierung
rwce	**r**obust **W**orst **C**ase **E**ckpunkte
rwcp	**r**obust **W**orst **C**ase **P**attern Search
rwcu	**r**obust **W**orst **C**ase **U**nsicherheitsbereich
SA	**S**ensitivitäts**a**nalyse
SQP	**S**equentielle **Q**uadratische **P**rogrammierung
u.B.v.	**u**nter **B**erücksichtigung **v**on
UNB	**U**ngleichungs**n**eben**b**edingungen
WYSIWYM	**W**hat **y**ou **s**ee **i**s **w**hat **y**ou **m**ean (deutsch: »Was man sieht, ist was man meint«)
ZV	**Z**ufalls**v**ariable

Mathematische Symbole

α, λ	Parameter der konvexen Kombination
$\boldsymbol{\delta}$	Vektor der maximalen Abweichung vom nominalen Punkt $\mathbf{x}_0$
$\boldsymbol{\xi}$	lokale Koordinaten des Unsicherheitsbereiches $U(\mathbf{x}_0)$

$\mathcal{D}$	Entwurfsraum (engl.: *Design Space*), Suchraum
$\delta_{i,r}$	relative Abweichung (bezogen auf $x_{0,i}$) vom nominalen Punkt
$\gamma(\mathbf{x})$	robuste Ziel-, Qualitätsfunktion
$\hat{\mathbf{x}}, \hat{\mathrm{R}}$	Schätzwert von $\mathbf{x}$, R
$\in$	Element
$[a, b]$	geschlossenes Intervall
$\mathbb{R}^n$	n-dimensionaler Raum
$\mathbf{A}^+$	verallgemeinerte Inverse, Pseudoinverse
$\mathbf{A}$	Matrix
$\mathbf{a}$	Vektor (üblicherweise ein Spaltenvektor)
$\mathbf{A}^\mathrm{T}, \mathbf{a}^\mathrm{T}$	transponiert
$\mathbf{g}(\mathbf{x}) = \nabla f(\mathbf{x})$	Gradient
$\mathbf{G}(\mathbf{x}) = \nabla^2 f(\mathbf{x})$	Hessesche Matrix
$\mathbf{h}(\mathbf{x})$	Ungleichungsnebenbedingungen
$\mathbf{x}^*$	lokales oder globales Minimum
$\mathbf{x}$	Vektor der Optimierungsvariablen
$\mathbf{x}_0$	nominaler Punkt
$\mathbf{x}_r^*$	robustes lokales oder globales Minimum
$\mathbf{x}_u$, $\mathbf{x}_o$	untere, obere Grenze des Entwurfsraumes
∇	Nabla Operator (Elemente: $\frac{\partial}{\partial x_i}$)
$\|\cdot\|_k$	l_k-Norm eines Vektors oder Matrix
$\subset$	Teilmenge
$\mathrm{sign}(x)$	Vorzeichenfunktion, Signumfunktion
$\tilde{\mathbf{x}}$, $\hat{\mathbf{x}}$	beliebige Vektoren (Punkte) des zulässigen Bereiches
$\|\cdot\|$	Absolutbetrag
d	Anzahl der Diskretisierungen pro Dimension
$f(\mathbf{x}), f(\hat{R})$	Zielfunktion, Kostenfunktion
m	Anzahl der Ungleichungsnebenbedingungen
n	Anzahl der Optimierungs-, Entscheidungsvariablen
S	zulässige Region (engl.: *feasible Region*)
$U(\mathbf{x}_0)$	Unsicherheitsbereich, Umgebung vom nominalen Punkt $\mathbf{x}_0$

Parameter Evolutionsstrategien

$\boldsymbol{\sigma}$	Satz von Strategieparametern, Schrittweiten

ϵ	Abbruchkriterium
γ	Gewichtungsfakor multiplikative Schrittweitenadaption
λ	Zahl der Nachkommen, Kinder
$\mathbf{a}$	beliebiges Individuum
$\mathbf{e}_1, \mathbf{e}_2$	Elternteil 1, Elternteil 2
$\mathbf{n}$	Nachkömmling
$\mathbf{x}$	Satz von Objektparametern, Designvariablen
μ	Zahl der Eltern
ρ	Zahl der an der Rekombination beteiligten Eltern
σ	Standardabweichung
τ_0, τ_1	Nachfolger-, Dimensionsbeiwert
$\tilde{\mathbf{a}}$	mutiertes Individuum
$\tilde{P}_\lambda$	mutierte Population
$F(\mathbf{x})$	Fitness
$N(0,1)$	normalverteilte ZV, Mittelwert 0 und Standardabweichung 1
p_i	Selektionswahrscheinlichkeit
P_μ	Startpopulation
qn	quadratische Norm der Population
S	Selektionsdruck

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis