

050015 PR Optimierung und Simulation

Version: 1.0.1

Gabriele Uchida, Thomas Klausner,
Tatjana Slavova, Vadym Slavov,
Stephan Pfannerer, Theres Steiner, Martin Fuchs

Wintersemester 2015

INHALTSVERZEICHNIS

i	ALLGEMEINE INFORMATIONEN	5
o	ALLGEMEINE INFORMATIONEN ÜBER DAS PRAKTIKUM	6
o.1	Kursablauf	6
o.2	Aufgaben des Teams und Benotung	7
o.2.1	Achtung!	8
o.3	Kursverwaltung	8
o.4	Tutorium	8
ii	PRAKTIKUMSAUFGABEN	9
1	KOMBINATORISCHE OPTIMIERUNG: ASSIGNMENT PROBLEM UND BRANCH AND BOUND ALGORITHMUS	10
1.1	Einführung	10
1.1.1	Branch and Bound Algorithmus	10
1.1.2	Assignment Problem	10
1.2	Aufgabenstellung	11
1.2.1	Theorie Erarbeitung	11
1.2.2	Implementierung	12
1.2.3	Homepage	13
1.2.4	Warnung	13
2	QUADRATISCHE OPTIMIERUNG OHNE NEBENBEDINGUNG: AUSGLEICHSPOLYNOME	14
2.1	Einführung	14
2.1.1	Quadratische Polynome	14
2.1.2	CG-Verfahren	15
2.1.3	Polynomiale Regression	16
2.2	Aufgabenstellung	16
2.2.1	Theorie Erarbeitung	16
2.2.2	Implementierung	17
2.2.3	Homepage	18
2.2.4	Warnung	18
3	HEURISTISCHE OPTIMIERUNG: AMEISENALGORITHMUS UND GENETISCHE ALGORITHMEN	19
3.1	Einführung	19
3.1.1	Ameisenalgorithmen	19
3.1.2	Genetische Algorithmen	20
3.2	Aufgabenstellung	21
3.2.1	Theorie Erarbeitung	21

3.2.2	Implementierung	21
3.2.3	Homepage	22
3.2.4	Warnung	23
4	NICHTLINEARE OPTIMIERUNG OHNE NEBENBEDINGUNGEN: NEWTON- VERFAHREN	24
4.1	Einführung	24
4.1.1	Newtonverfahren und Optimierung	24
4.2	Aufgabenstellung	25
4.2.1	Theorie-Erarbeitung	25
4.2.2	Implementierung	26
4.2.3	Homepage	26
4.2.4	Warnung	27
5	NICHTLINEARE OPTIMIERUNG OHNE NEBENBEDINGUNGEN: MEHRDI- MENSIONALE VERFAHREN	28
5.1	Einführung	28
5.1.1	Das Nelder-Mead-Verfahren	28
5.1.2	Abstiegsrichtung	29
5.1.3	Das Gradientenverfahren	29
5.1.4	Koordinatenabstiegsmethode	29
5.2	Aufgabenstellung	30
5.2.1	Theorie-Erarbeitung	30
5.2.2	Implementierung	31
5.2.3	Homepage	31
5.2.4	Warnung	32
6	NICHTLINEARE OPTIMIERUNG OHNE NEBENBEDINGUNGEN: EINDIMEN- SIONALE VERFAHREN	33
6.1	Einführung	33
6.1.1	Unimodale Funktionen	33
6.1.2	Goldener Schnitt, Fibonacciverfahren	33
6.1.3	Bisektionsverfahren	34
6.2	Aufgabenstellung	34
6.2.1	Theorie-Erarbeitung	34
6.2.2	Implementierung	35
6.2.3	Homepage	35
6.2.4	Warnung	36
7	LINEARE OPTIMIERUNG: SIMPLEXVERFAHREN	37
7.1	Einführung	37
7.1.1	Lineares Optimierungsprogramm	37
7.1.2	Linearisierung	37
7.1.3	Graphische Darstellung	38
7.2	Aufgabenstellung	38
7.2.1	Theorie-Erarbeitung	38

7.2.2	Implementierung	39
7.2.3	Homepage	40
7.2.4	Warnung	41
iii	ANHANG	42
8	FUNKTIONENLISTE	43
8.1	Eindimensionale Funktionen	43
8.2	Zweidimensionale Funktionen	43
8.3	Weitere interessante Testfunktionen	44
9	HINWEISE FÜR DIE IMPLEMENTIERUNG	45
9.1	Die Klasse Funktion	45
9.1.1	Idee	45
9.1.2	Details zur Verwendung	46
9.2	Matrix aus File einlesen	48

Teil I

ALLGEMEINE INFORMATIONEN

ALLGEMEINE INFORMATIONEN ÜBER DAS PRAKTIKUM

Das Praktikum wird heuer in neuer Form durchgeführt: Teams aus fünf Personen arbeiten gemeinsam über das Semester an einem Thema aus dem Fachgebiet Optimierung und Simulation. Einige Themen werden davon auch in der Vorlesung vorkommen, müssen aber nicht. Daher ist eine eigene Recherche in den meisten Fällen notwendig.

Die Teammitglieder müssen in der selben Gruppe im USPACE/UNIVIS angemeldet sein!

0.1 KURSABLAUF

Bei diesem Praktikum arbeiten Sie die meiste Zeit nur mit Ihrem Team zusammen am Projekt. Zu den Zeiten im Vorlesungsverzeichnis gibt es nur fünf Termine, an denen die ganze Gruppe, zu der Sie sich im USPACE/UNIVIS angemeldet haben, zusammenkommt. Hier herrscht dann aber Anwesenheitspflicht!

- 12. Oktober - 16. Oktober*: Kick-Off: Team- und Themenfindung
- 16. November - 20. November*: Präsentation 1
- 23. November - 27. November*: Präsentation 1
- 11. Jänner - 15. Jänner*: Präsentation 2
- 18. Jänner - 22. Jänner*: Präsentation 2

*genauer Tag und Uhrzeit entspricht Ihrer LV-Gruppe und wird online angeschrieben! Die Termine für diverse Abgaben sind knapp vor diesen Terminen und werden auch online angegeben.

Zwischendurch können Sie sich mit Ihrem Lehrveranstaltungsleiter individuell Feedbackgespräche ausmachen. Genauere Informationen dazu bekommen Sie beim Kick-Off Treffen in Ihrer LV-Gruppe.

Zusätzlich werden wieder Tutorien angeboten. Weitere Informationen gibt es im Punkt Tutorium.

0.2 AUFGABEN DES TEAMS UND BENOTUNG

Jedes Projekt umfasst fünf große Bereiche, die individuell mit Punkten benotet werden. Aus der folgenden Liste können die Aufgaben und maximal erreichbare Punkte entnommen werden.

THEORIE ERARBEITUNG: MAX. 40 PUNKTE Sie erarbeiten zunächst die Theorie und fassen diese in einem Paper/Protokoll zusammen.

IMPLEMENTIERUNG: MAX. 40 PUNKTE Jedes Thema umfasst eine Programmieraufgabe, welche mit C++ zu lösen ist.

1. PRÄSENTATION: MAX. 40 PUNKTE Die Folien der Präsentation sind vorab hochzuladen und bringen max. 20 Punkte, der Vortrag selber bringt auch max. 20 Punkte.

2. PRÄSENTATION: MAX. 40 PUNKTE siehe 1. Präsentation

HOME PAGE: MAX. 40 PUNKTE Jedes Team gestaltet eine Homepage, wo das Thema allgemein verständlich dargestellt werden soll.

ZUSATZPUNKTE: MAX. 10 PUNKTE Für besonders ordentliche Abgaben, Mitarbeit bei den Präsentationsterminen oder anderen herausragenden Leistungen können Bonuspunkte vergeben werden.

Jedes Team hat neben den Aufgaben am Ende auch ein Arbeitsprotokoll abzugeben, aus dem genau hervorgeht, wer an welchem Teil wie viel mitgearbeitet hat.

Jeder Studierende muss an mindestens zwei dieser Bereiche (nicht Zusatzpunkte!) mit mindestens 5% mitgearbeitet haben.

Nachdem jeder Teil mit Punkten benotet wurde, werden diese an die Studierenden entsprechend aufgeteilt und daraus die individuellen Noten bestimmt. Abbildung 1 zeigt das Benotungsschema.

Anschließend wird nach folgendem Notenschlüssel die Note vergeben. Um die Lehrveranstaltung positiv zu absolvieren müssen jedoch genug Punkte ohne die Bonuspunkte erreicht werden!!

ERREICHTE PUNKTE OHNE BONUSPUNKTE < 20*: Nicht Genügend

$20 \leq$ * ERREICHTE PUNKTE < 25: Genügend

$25 \leq$ ERREICHTE PUNKTE < 30: Befriedigend

$30 \leq$ ERREICHTE PUNKTE < 35: Gut

$35 \leq$ ERREICHTE PUNKTE: Sehr gut

* Diese Grenze gilt für erreichte Punkte ohne Bonuspunkte

		Susi	Maxi	Fritzi	Seppi	Lilli	Checksumme	Bewertung	Max. Anzahl Punkte
Implementierung			50%		50%		100,00%	20	40
Erste Präsentation									
	Slides	30%	30%	40%			100,00%	15	20
	Vortrag	50%		50%			100,00%	8	20
Zweite Präsentation									
	Slides					100%	100,00%	20	20
	Vortrag				50%	50%	100,00%	20	20
Paper/Protokoll				55%	25%	20%	100,00%	15	40
Homepage		60%	35%			5%	100,00%	40	40
Checksumme		40	40	40	40	40			
Bewertung		32,5	28,5	18,25	23,75	35			
Bonuspunkte		2	1	1	2	2			
Bewertung plus Bonus		34,5	29,5	19,25	25,75	37			
Note		2	3	5	3	1			

Abbildung 1: Bewertungsschema

0.2.1 Achtung!

Nicht immer sind alle Teile des Projekts gleich aufwändig. Versuchen Sie daher die einzelnen Arbeitsbereiche gemeinsam anzugehen und nach dem Können und den Stärken Ihrer Teamkollegen einzelne Aufgaben zu verteilen.

Sollte es aus organisatorischen Gründen nicht möglich sein, dass Sie in einer Gruppe zu 5 Personen arbeiten, sondern in einer kleineren Gruppe arbeiten müssen, werden die Aufgabenstellungen für Ihre Gruppe angepasst. Kontaktieren Sie dazu Ihren Übungsgruppenleiter!

0.3 KURSVERWALTUNG

Der Kurs wird heuer erstmals über CEWEBS gehalten. Dort können Sie sich in Gruppen einteilen und Ihr Thema wählen. Ebenso sind dort alle Abgaben rechtzeitig hochzuladen.

Nützen Sie auch das Forum für die direkte Kommunikation mit Studienkollegen und Tutoren, die bei Problemen gerne helfen.

0.4 TUTORIUM

Die Lehrveranstaltung wird von drei Tutoren betreut, welche über das Forum oder per E-Mail kontaktiert werden können.

- Martin Fuchs: martin.e104.fuchs@tuwien.ac.at
- Theres Steiner: theres.steiner@univie.ac.at
- Stephan Pfannerer: stephan.pfannerer@univie.ac.at

Neben der elektronischen Betreuung gibt es regelmäßige Tutorien. Inhalte und Termine werden auf der E-Learning-Plattform angekündigt.

Teil II

PRAKTIKUMSAUFGABEN

KOMBINATORISCHE OPTIMIERUNG: ASSIGNMENT PROBLEM UND BRANCH AND BOUND ALGORITHMUS

1.1 EINFÜHRUNG

Ziel dieses Projekts ist es einen Branch and Bound Algorithmus zur Lösung des Assignment Problems zu implementieren (siehe dazu auch die Vorlesungsunterlagen). Andere Algorithmen und Probleme der Kombinatorischen Optimierung sollen eigenständig recherchiert und vorgestellt werden.

1.1.1 *Branch and Bound Algorithmus*









Im Prinzip könnte man ein kombinatorisches Problem so lösen, dass man für alle zulässigen Lösungen (es gibt nur endlich viele) die Zielfunktion bestimmt und das Minimum nimmt. Dies bezeichnet man als vollständige Enumeration.

In der Praxis ist vollständige Enumeration aus Laufzeitgründen meist undurchführbar. Z.B.: Bei 10 Variablen mit je 100 möglichen Werten müsste man $100^{10} = 10^{20}$ Möglichkeiten durchprobieren. Es ist daher in der Regel nötig, die Menge der zulässigen Lösungen, deren Zielfunktionswert man bestimmt, einzuschränken.

Deshalb wird der zulässige Bereich nach und nach in mehrere Teilmengen aufgespalten (Branch). Mittels geeigneter Schranken (Bound) sollen viele suboptimale Belegungen frühzeitig erkannt und daher nicht weiter beachtet werden. So kann der durchsuchte Lösungsraum klein gehalten und rasch ein Optimum gefunden werden.

1.1.2 *Assignment Problem*

Allgemein handelt es sich beim Assignment Problem um eine Menge von Agenten A (engl.: agents), die eine gleich große Menge von Aufgaben T (engl.: tasks) zu erfüllen haben. Zusätzlich gibt es eine Gewichts- oder Kostenfunktion $C : A \times T \rightarrow \mathbb{R}$, welche die Kosten $C(a, t)$ dafür angibt, dass der Agent a die Aufgabe t erfüllt. In Abbildung 2 gibt die Tabelle beispielsweise an, wie viele Stunden ein gewisser Mitarbeiter für einen gewissen Job braucht. Ein Eintrag c_{at} in der Tabelle gibt also die Kosten dafür an, dass ein Agent a die Aufgabe t übernimmt. Natürlich sollen bei

				
	9	2	7	8
	6	4	3	7
	5	8	1	8
	7	6	9	4

Used Icons designed by Freepik

Abbildung 2: Vier Agenten werden vier Aufgaben/Jobs zugeordnet

der Zuordnung die Arbeitsstunden minimiert werden. Gesucht ist also eine Bijektion $f : A \rightarrow T$, sodass die Kostenfunktion

$$\sum_{a \in A} C(a, f(a))$$

minimal wird. Wichtig dabei ist, dass jeder Agent, genau einer Aufgabe und jede Aufgabe genau einem Agenten zugeordnet wird, sonst ergibt sich keine Bijektion.

1.2 AUFGABENSTELLUNG

1.2.1 Theorie Erarbeitung

Recherchieren sie die hier angesprochenen Themen. Gehen Sie in Ihrer Dokumentation zumindest auf folgende Punkte ein:

- Erklären Sie den Branch and Bound Algorithmus allgemein und speziell, wie er für das Assignment Problem umzusetzen ist - siehe auch Vorlesung.
- Verwenden Sie ein konkretes Beispiel um zu zeigen wie der Branch and Bound Algorithmus für das Assignment Problem funktioniert.
- Erklären Sie alle notwendigen Datenstrukturen, die für die Implementierung notwendig sind und erklären grob, wie diese programmiert werden können.
- Recherchieren Sie andere Algorithmen zur Lösung des Assignment Problems.

- Wie kann man das Assignment Problem in ein ganzzahliges lineares Programm umwandeln? Was ist das überhaupt?

Dokumentieren Sie alles in einem gut strukturierten PDF.

1.2.2 Implementierung

Implementieren Sie die folgenden Punkte in C++:

- Entwickeln Sie für das Assignment Problem einen Branch and Bound Algorithmus.
- Implementieren Sie alle notwendigen Datenstrukturen, die für den Algorithmus notwendig sind.
- Schreiben Sie für Ihren Algorithmus eine Funktion mit der Signatur
`int* assignment(int n, int **kosten).`

n gibt an, wie viele Jobs auf wie viele Agenten verteilt werden sollen und entspricht der Größe des zwei dimensional Arrays *kosten*. Der Eintrag *kosten*[i][j] gibt die Kosten an, Agent i den Job j zuzuweisen. Als Rückgabe wird ein Integer Array der Länge n erwartet, dass eine Permutation der Jobs darstellt. Der erste Eintrag wird Person 1 zugewiesen, der zweite Person 2, ...

- Schreiben Sie ein Programm, dass dem User eine einfache Möglichkeit gibt, Ihren Branch and Bound Algorithmus für eigene Kostenmatrizen anzuwenden. Zum Beispiel: Ein einfaches Kommandozeilenprogramm, das neben diversen Parametern einen Pfad zu einem Textfile erwartet. In diesem Textfile ist die Kostentabelle gespeichert.
 - Einzulesendes Format: in der ersten Zeile steht eine einzige Zahl n , sie gibt die Anzahl der nachfolgenden Zeilen an. Jede weitere Zeile enthält n Integers, die durch einen oder mehrere Abstände getrennt sind. Sie geben die Einträge der Kostenmatrix an. Beachten Sie zum Einlesen eines ähnlichen Files auch das Beispielprogramm im Anhang.

Testfiles für Ihr Problem finden Sie auf der Homepage.

- Schreiben Sie sich eigene Testfiles, oder gar ein Programm, das automatisch neue generiert.
- Achten Sie darauf, dass Ihre Programme entsprechende Warnmeldungen ausgeben, wenn etwas unerwartetes passiert.

1.2.3 *Homepage*

Erstellen Sie eine Homepage, die interessierten Leuten erklärt, worum es in Ihrem Projekt geht. Arbeiten Sie zumindest auch folgende Punkte ein:

- Erklären Sie alle Vorgänge anhand händisch durchgeführter Beispiele und veranschaulichen Sie diese durch Graphiken.
- Stellen Sie das entwickelte Programm und den Quellcode zur Verfügung und erklären, wie diese zu Verwenden sind.
- Stellen Sie vor, wie man das Assignment Problem als ganzzahliges lineares Programm formulieren kann. Suchen Sie auch ein Tool (z.B. Microsoft Excel), mit dem dies gelöst werden kann und erklären Sie in Form eines Tutorials, wie dies zu verwenden ist.
- Vergleichen Sie die Ergebnisse Ihres Programms, mit dem ebenen recherchierten Tool.
- Recherchieren Sie gängige Kombinatorische Optimierungsprobleme und allgemeine Lösungsstrategien und stellen Sie diese in einer Infographik entsprechend zusammen.

1.2.4 *Warnung*

Es handelt sich bei den obigen Aufgaben nur um die Mindestanforderungen für Ihr Projekt. Erweitern Sie das Projekt in Eigeninitiative und besprechen Sie Ihre Ideen dazu in den Feedbackgesprächen.

Achten Sie auf korrektes wissenschaftliches Arbeiten und zitieren Sie alle Quellen nach gängigen Regeln.

QUADRATISCHE OPTIMIERUNG OHNE NEBENBEDINGUNG: AUSGLEICHSPOLYNOME

2.1 EINFÜHRUNG

Ziel dieses Projekts ist es ein Verfahren zur Minimierung quadratischer Polynome zu implementieren und dieses konkret für polynomiale Regression einzusetzen.

2.1.1 Quadratische Polynome

Allgemein verstehen wir unter einem quadratischen Polynom in n Variablen eine Funktion der Form:

$$f(x_1, \dots, x_n) = \sum_{1 \leq i, j \leq n} a_{ij} \cdot x_i \cdot x_j + \sum_{1 \leq i \leq n} b_i \cdot x_i + c$$

Man kann voraussetzen, dass $a_{ij} = a_{ji}$ gilt (Überlegen Sie sich, warum dies keine Einschränkung ist). Fasst man die Werte a_{ij} zu einer symmetrischen $n \times n$ Matrix $\underline{\underline{A}}$, b_i zum Vektor \underline{b} und x_i zum Vektor \underline{x} zusammen, so kann man die Funktion in der Form

$$f(\underline{x}) = \underline{x}^T \cdot \underline{\underline{A}} \cdot \underline{x} + \underline{b}^T \cdot \underline{x} + c$$

schreiben.

Wollen wir nun die quadratische Funktion $f(\underline{x}) = \underline{x}^T \cdot \underline{\underline{A}} \cdot \underline{x} - 2\underline{b}^T \cdot \underline{x} + c$ minimieren/maximieren, bekommen wir durch das Nullsetzen des Gradienten die Gleichung

$$\nabla f(\underline{x}) = 2(\underline{\underline{A}} \cdot \underline{x} - \underline{b}) = 0$$

oder äquivalent dazu

$$\underline{\underline{A}} \cdot \underline{x} = \underline{b}$$

Lösen dieses Gleichungssystem liefert die optimale Lösung. Das Invertieren einer großen Matrix ist oftmals mühsam, weshalb wir hier das Verfahren der konjugierten Gradienten, oder kurz CG-Verfahren (conjugated gradients) betrachten wollen, welches man sowohl zum exakten, als auch approximativen Lösen linearer Gleichungen verwenden kann.

2.1.2 CG-Verfahren

Das CG-Verfahren ist eine der beliebtesten iterativen Methoden zum Lösen von linearen Gleichungssystemen $\underline{A}\underline{x} = \underline{b}$ mit symmetrisch positiv definiten Matrix \underline{A} .

Wie bereits oben erwähnt: Ist die Matrix \underline{A} des Gleichungssystems $\underline{A}\underline{x} = \underline{b}$ symmetrisch und positiv definit, dann entspricht das Lösen des Gleichungssystems dem Minimieren der quadratischen Form

$$f(\underline{x}) = \underline{x}^T \underline{A} \underline{x} - 2 \underline{b}^T \underline{x}$$

Anstatt wie bei der Methode des steilsten Abstiegs (steepest descent) in Richtungen der Residuen \underline{r}_k (siehe Algorithmus unten) abzustiegen, wird beim CG-Verfahren in Richtung \underline{d}_k (siehe Algorithmus unten) gelaufen. Diese Richtung \underline{d}_k sind A -konjugiert¹ D.h.

$$\underline{d}_i^T \underline{A} \underline{d}_j = 0 \quad \forall i \neq j$$

Es ergibt sich der Algorithmus

(i) Wähle Startwert \underline{x}_0 und setze $\underline{r}_0 = \underline{b}$

(ii) While $k < N$ und $\|\underline{r}_k\| > \epsilon$

(a) Berechne Schrittweite

$$\alpha_k = \frac{\underline{d}_k^T \underline{r}_k}{\underline{d}_k^T \underline{A} \underline{d}_k}$$

(b) Berechne neuen Ort \underline{x}_{k+1}

$$\underline{x}_{k+1} = \underline{x}_k + \alpha_k \underline{d}_k$$

(c) Update Residuum

$$\underline{r}_{k+1} = \underline{r}_k - \alpha_k \underline{A} \underline{d}_k$$

(d) Bestimme neue Suchrichtung

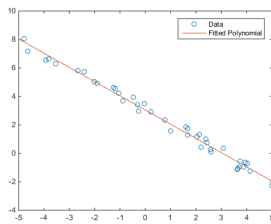
$$\beta_k = \frac{\underline{r}_{k+1}^T \underline{r}_{k+1}}{\underline{r}_k^T \underline{r}_k}$$

$$\underline{d}_{k+1} = \underline{r}_{k+1} + \beta_k \underline{d}_k$$

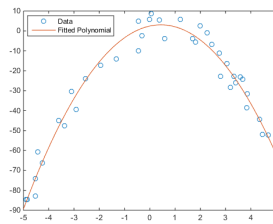
Weitere Quellen: <http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf>

<http://www.mathematik.uni-ulm.de/numerik/teaching/ws07/NUM1b/prog/anleitung01/>

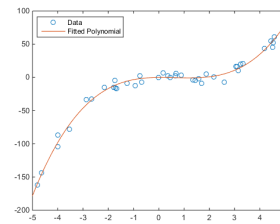
¹ Sie sind bzgl. des durch \underline{A} induzierten Skalarprodukts orthogonal.



(a) Polynom vom Grad 1



(b) Polynom vom Grad 2



(c) Polynom vom Grad 3

Abbildung 3: Ausgleich der Messwerte durch Polynome

2.1.3 Polynomiale Regression

Gegeben seien Messwerte (x_i, y_i) für $1 \leq i \leq k$ und es wird vermutet, dass der Zusammenhang zwischen x_i und y_i mit einem Polynom vom Grad $n < k$ beschrieben werden kann, also dass näherungsweise $p(x_i) = y_i$ gilt. Ein derartiges Polynom heißt Regressions- oder Ausgleichspolynom.

Da Messwerte mit Ungenauigkeiten verbunden sind, wird der Zusammenhang nicht exakt stimmen und wir müssen uns damit begnügen, eine optimale Näherung zu finden, wir suchen also ein Polynom $p(x) = a_0 \cdot x^0 + a_1 \cdot x^1 + a_2 \cdot x^2 + \dots + a_n \cdot x^n$, sodass der Fehler

$$f(a_0, a_1, \dots, a_n) = \sum_{1 \leq i \leq k} (p(x_i) - y_i)^2$$

minimal wird. Sei $\underline{a} = (a_0, a_1, \dots, a_n)^T$, so kann man die obige Funktion in der Form

$$f(\underline{a}) = \underline{a}^T \cdot \underline{\underline{A}} \cdot \underline{a} - 2 \cdot \underline{b}^T \cdot \underline{a} + c \quad (1)$$

anschreiben, wobei die Werte $\underline{\underline{A}}, \underline{b}, c$ in Abhängigkeit der gegebenen Messwerte zu bestimmen sind.

Mit den obigen Überlegungen über quadratische Polynome, kann diese Funktion minimiert werden und damit ein optimales Regressionspolynom vom Grad n bestimmt werden. Abbildung 3 zeigt die Messwerte und das errechnete Ausgleichspolynom. Diese Daten

2.2 AUFGABENSTELLUNG

2.2.1 Theorie Erarbeitung

Recherchieren sie die hier angesprochenen Themen. Gehen Sie in Ihrer Dokumentation zumindest auf folgende Punkte ein:

- Ergänzen Sie Details zum Abschnitt über Quadratische Polynome und erklären Sie mit Beispielen die Formeln.

- Vergleichen Sie die Schritte des CG-Verfahrens mit dem Gram-Schmidtschen Orthogonalisierungsverfahren, das Sie in MBT/GMA kennengelernt haben.
- Erklären Sie den Zusammenhang und Unterschied zum Verfahren Deepest Decent aus der Vorlesung.
- Bestimmen Sie eine Formel zur Berechnung der Werte $\underline{A}, \underline{b}, c$ die für Gleichung 1 notwendig sind.
- Berechnen Sie händisch ein Ausgleichspolynom zu einem selbst gewählten Beispiel. Lösen Sie das auftretende Gleichungssystem einmal durch Invertieren der Matrix oder Gausselimination und einmal durch das CG-Verfahren.
- Wie verwendet man CG-Verfahren zum bestimmen der exakten Lösungen? Wie bestimmt man damit Näherungslösungen?

Dokumentieren Sie alles in einem gut strukturierten PDF.

2.2.2 Implementierung

Implementieren Sie die folgenden Punkte in C++:

- Für das CG-Verfahren benötigen Sie Funktionen zum Rechnen mit Matrizen und Vektoren. Sie können entweder selber derartige Klassen entwickeln, oder frei verfügbare Bibliotheken verwenden.
- Schreiben Sie eine Funktion `double* cg(int n, double** A, double* b)`, die das CG-Verfahren für die $n \times n$ matrix A (2-dimensionales Array) und den Vektor b (Array der Länge n) durchführt. Das Ergebnis soll ein Array der Länge n sein.
- Schreiben Sie eine Funktion `double* polyfit(int n, int k, double* x, double* y)`, welche das Ausgleichspolynom vom Grad n für die gegebenen Messpunkte $(x[i], y[i])$ bestimmt. Dabei x und y sind Arrays der Länge k .
- Schreiben Sie ein Programm, welches mit zwei Parametern n und $file$ gestartet wird. $file$ enthält den Pfad zu einem Textdokument, das unten stehenden Format folgt. Das Programm soll zu den Datensätzen aus dem Textdokument das Ausgleichspolynom vom Grad n bestimmen und ausgeben.
 - Einzulesendes Format: in der erster Zeile steht die Anzahl der Datensätze k und in den weiteren k Zeilen jeweils ein Paar x y durch einen Abstand getrennt. Beachten Sie zum Einlesen eines ähnlichen Files auch das Beispielprogramm im Anhang.

Testfiles für Ihr Problem finden Sie auf der Homepage.

- Schreiben Sie ein Programm, mit dem Sie Testfiles erzeugen können: Das Programm soll also Daten generieren, die ungefähr dem Graphen eines Polynoms folgen.
- Visualisieren Sie die Ergebnisse.
- Achten Sie darauf, dass Ihre Programme entsprechende Warnmeldungen ausgeben, wenn etwas unerwartetes passiert.

2.2.3 *Homepage*

Erstellen Sie eine Homepage, die interessierten Leuten erklärt, worum es in Ihrem Projekt geht. Arbeiten Sie zumindest auch folgende Punkte ein:

- Erklären Sie alle Vorgänge anhand händisch gerechneter Beispiele.
- Veranschaulichen Sie die Ergebnisse durch Graphiken
- Stellen Sie das entwickelte Programm und den Quellcode zur Verfügung und erklären, wie diese zu Verwenden sind.
- Recherchieren Sie, wie man Trendlinien und Ausgleichspolynome mit anderer Software, z.B. Microsoft Excel oder OpenOffice, erstellen kann und machen Sie dazu entsprechende Tutorials.

2.2.4 *Warnung*

Es handelt sich bei den obigen Aufgaben nur um die Mindestanforderungen für Ihr Projekt. Erweitern Sie das Projekt in Eigeninitiative und besprechen Sie Ihre Ideen dazu in den Feedbackgesprächen.

Achten Sie auf korrektes wissenschaftliches Arbeiten und zitieren Sie alle Quellen nach gängigen Regeln.

HEURISTISCHE OPTIMIERUNG: AMEISENALGORITHMUS UND GENETISCHE ALGORITHMEN

3.1 EINFÜHRUNG

Ziel dieses Projekts ist es ein, zwei heuristische Verfahren aufzubereiten und für konkrete Anwendungen zu implementieren.

3.1.1 Ameisenalgorithmen

Eine Biene, Ameise oder jedes andere „social insect“ alleine ist verloren in dieser Welt. Diese Tiere benötigen ihre Kolonie um Nahrung zu finden, ein Nest zu bauen oder sogar zu überleben. Eine einzelne Ameise ist nicht wirklich Klug, aber ihre ganzer Schwarm gemeinsam schafft es beinahe optimale (kürzeste) Wege zu Nahrungsquellen zu finden. Man spricht hier von der Schwarmintelligenz.

Ameisen versprühen Duftstoffe, die Pheromone, welche jene Wege markieren, die sie verwenden. Wege, die öfters passiert werden, und möglicherweise zu einer Futterquelle führen, bekommen dadurch eine höhere Pheromonkonzentration. Über diese Pheromone kommunizieren die Ameisen indirekt, indem sie einen Weg mit höherer Konzentration auch mit höherer Wahrscheinlichkeit wählen. Dadurch bewegen sie sich auf einem Random-Walk durch ihre Landschaft. Auf natürliche Weise verdampfen Pheromone, daher werden Wege, die länger nicht mehr genutzt wurden, auch immer schwerer zu finden und müssen erst zufällig wieder neu entdeckt werden. Abbildung 4 veranschaulicht dies schematisch.

Ameisenalgorithmen oder die sogenannte Ant colony optimization (ACO) ahmen die Kommunikations- und Koordinationsprinzipien von Ameisenschwärmen nach, um z.B. kürzeste Wege in einem Graphen zu finden.

Weitere Informationen dazu finden Sie unter anderem hier: http://www.academia.edu/4675500/Shortest_Path_Problem_Solving_Based_on_Ant_Colony_Optimization_Metaheuristic und im Vorlesungsskriptum.

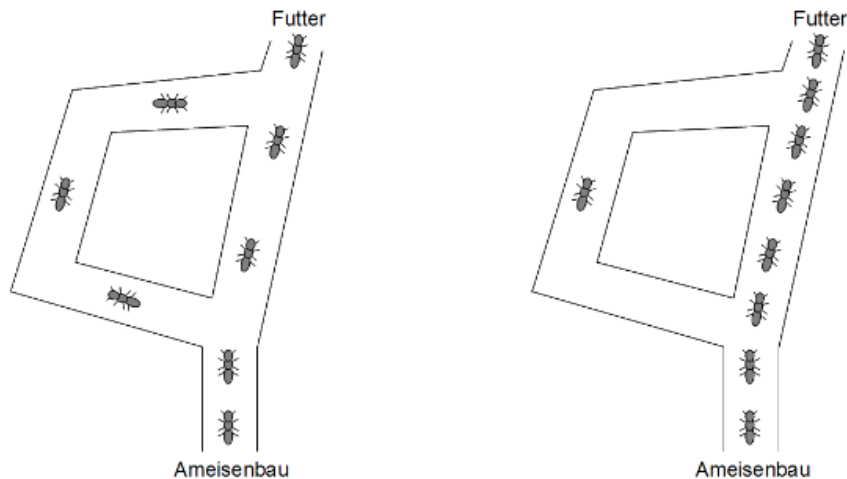


Abbildung 4: Anfangs wählen die Ameisen an der Verzweigung zufällige Wege, durch die höhere Pheromonkonzentration wird der kürzere aber immer stärker verwendet.

3.1.2 Genetische Algorithmen

Die sogenannten genetischen Algorithmen sind stochastische Verfahren um Optimierungsprobleme näherungsweise zu lösen. Dazu verwenden sie virtuelle Organismen, die sich versuchen an ihre künstliche Umwelt anzupassen. Nach biologischen Vorbild der Evolutionstheorie werden die Individuen mutiert, rekombiniert und selektiert, wobei jene Wesen bei der Vermehrung bevorzugt werden, die schon besser an die Umwelt angepasst sind. Die Anpassung an die Umwelt entspricht dabei allgemein einer zu optimierende Zielfunktion und die Lebewesen sind Näherungen an das Optimum. Um solche Algorithmen verwenden zu können, ist es notwendig, die Lösungen geschickt mit einer Art DNA zu codieren.

Hat man die Lösungen erst-einmal codiert, so funktionieren genetische Algorithmen immer sehr ähnlich. Eine Basis, an die man sich halten kann (man kann natürlich immer leichte Änderungen vornehmen) ist die Unterteilung des Programms in 5 Module/Abschnitte.

INITIALISIERUNG Im ersten Schritt ist es notwendig die Population zu Initialisieren, also zufällig gültige Lebewesen zu erzeugen. Ist die erste Population erstellt, wird sie durch die nächsten Module eine Evolution durchlaufen.

EVALUIERUNG Um die besten Lebewesen zu finden, müssen diese entsprechend evaluiert und so ihre Fitness nach der gegebenen Zielfunktion bestimmt werden.

SELEKTION In der Selektionsphase wird das beste Lebewesen als Elternteil bestimmt. Auch das schlechteste Lebewesen wird berechnet und durch die Rekombination des besten mit einem zufälligen anderen ersetzt.

REKOMBINATION Dieses Modul dient der Verpaarung von Individuen. Diese werden nach der Idee des Crossing-Overs rekombiniert.

MUTATION Mutationen treffen ein zufällig gewähltes Lebewesen und ändern dabei ein zufälliges Zeichen des „Gen“-Codes ab.

Ein genetische Algorithmus beginnt immer mit dem Modul „Initialisierung“. Anschließend werden die anderen Module in einer Schleife so lange wiederholt, bis ein Lebewesen mit Fitness über einem gegebenen Schwellwert liegt.

3.2 AUFGABENSTELLUNG

3.2.1 Theorie Erarbeitung

Recherchieren sie die hier angesprochenen Themen. Gehen Sie in Ihrer Dokumentati-on zumindest auf folgende Punkte ein:

- Für welche Probleme eignen sich die angesprochenen Algorithmen?
- In welchem Sinn kann man von einer Optimalen Lösung sprechen?
- Überlegen Sie sich mit Ihren Rechercheergebnissen, wie sie einen Ameisenalgorithmus für das shortest-path-Problem implementieren können: Gegeben ist ein Graph, wobei jede Kante eine Länge/ein Gewicht zugeteilt bekommt. Gesucht ist eine Kantenfolge zwischen zwei gegebenen Knoten, sodass die Summe der Kantenlängen minimal ist.
- Erklären Sie anhand des obigen Problems, wie so ein Programm aufzubauen ist. Erklären Sie alle wichtigen später verwendeten Formeln.
- Wie baut man einen genetischen Algorithmus auf?
- Recherchieren Sie über das n -Damen-Problem und überlegen Sie sich, wie man dieses Problem für genetische Algorithmen kodieren kann. Erklären Sie daran, wie man allgemein vorzugehen hat. Für welche n gibt es überhaupt Lösungen.

Dokumentieren Sie alles in einem gut strukturierten PDF.

3.2.2 Implementierung

Implementieren Sie die folgenden Punkte in C++:

- Entwickeln Sie für das shortest-path-Problem einen Ameisenalgorithmus. Schreiben Sie dazu eine Funktion, welche neben von Ihnen frei wählbare Parameter folgende Argumente hat: (int n, double **adjazenz, int start, int ende)

n gibt an, aus wie vielen Knoten der Graph besteht. Die Matrix/das zwei dimensionale Array *adjazenz* gibt an, welches Länge die Kante zwischen Knoten i und j hat. Die letzten beiden Werte geben die Nummer des Start- und Endknotens an. Als Rückgabe soll die Abfolge der Knoten mit minimaler Länge ausgegeben werden.

- Schreiben Sie ein Programm, das dem User eine einfache Möglichkeit gibt, Ihren Ameisenalgorithmus für eigene Graphen anzuwenden. Also zum Beispiel ein einfaches Kommandozeilenprogramm, das neben diversen Parametern einen Pfad zu einem Textfile erwartet. In diesem Textfile ist die Matrix gespeichert. Halten Sie sich jedenfalls an das folgende Format.
 - Einzulesendes Format: in der ersten Zeile steht eine einzige Zahl n , sie gibt die Anzahl der nachfolgenden Zeilen an. Jede weitere Zeile enthält n Fließkommazahlen, die durch einen oder mehrere Abstände getrennt sind. Sie geben die Einträge der Adjazenzmatrix an. Beachten Sie zum Einlesen eines ähnlichen Files auch das Beispielprogramm im Anhang.

Testfiles für Ihr Problem finden Sie auf der Homepage.

- Schreiben Sie ein Programm, welches als Argument die Größe n eines Schachbretts erwartet (und vielleicht andere Parameter) und mit genetischen Algorithmen das n -Damen-Problem löst.
- Visualisieren Sie, soweit möglich, die Ergebnisse.
- Achten Sie darauf, dass Ihre Programme entsprechende Warnmeldungen ausgeben, wenn etwas unerwartetes passiert.

3.2.3 Homepage

Erstellen Sie eine Homepage, die interessierten Leuten erklärt, worum es in Ihrem Projekt geht. Arbeiten Sie zumindest auch folgende Punkte ein:

- Erklären Sie die beiden Erwähnten Algorithmen.
- Veranschaulichen Sie die Ergebnisse durch Graphiken.
- Stellen Sie das entwickelte Programm und den Quellcode zur Verfügung und erklären, wie diese zu Verwenden sind.
- Recherchieren Sie über weitere Anwendungen der vorgestellten Algorithmen und stellen Sie die Ergebnisse in einer Graphik dar. Wofür eignet sich was? Wann sollte man besser andere Methoden verwenden?
- Erklären Sie andere Algorithmen zum Thema Schwarmintelligenz und erstellen Sie Infographiken, Simulationen oder verlinken anschauliche Videos.

3.2.4 *Warnung*

Es handelt sich bei den obigen Aufgaben nur um die Mindestanforderungen für Ihr Projekt. Erweitern Sie das Projekt in Eigeninitiative und besprechen Sie Ihre Ideen dazu in den Feedbackgesprächen.

Achten Sie auf korrektes wissenschaftliches Arbeiten und zitieren Sie alle Quellen nach gängigen Regeln.

NICHTLINEARE OPTIMIERUNG OHNE NEBENBEDINGUNGEN: NEWTONVERFAHREN

4.1 EINFÜHRUNG

Ziel des Projekts ist es, das Newtonverfahren herzuleiten, zu Implementieren und auf verschiedene Funktionen anzuwenden.

4.1.1 *Newtonverfahren und Optimierung*

Das Newtonverfahren ist eines der effektivsten Verfahren in der nichtlinearen Optimierung. Außerdem ist es sowohl im eindimensionalen, als auch im mehrdimensionalen Raum anwendbar.

Die historische Anwendung des Newtonverfahrens ist die Nullstellensuche. An einen beliebigen Startpunkt wird die Tangente gelegt. Durch Lösen einer einfachen linearen Gleichung wird die Nullstelle der Tangente bestimmt, die zum neuen Ausgangspunkt des Iterationsverfahrens wird. Durch diese Methode wird versucht, sich der Nullstelle immer weiter anzunähern. Da das Minimum einer Funktion zugleich eine Nullstelle der Ableitung ist, kann diese historische Variante auf die Ableitung der Funktion angewendet werden um das Minimum der Funktion zu finden.

Eine andere Herangehensweise ist die, dass man die Funktion zunächst am Startpunkt durch ein quadratisches Polynom approximiert. Durch Ableiten des Approximationspolynoms und Nullsetzen der entstandenen linearen Funktion errechnet man dessen Minimum. Das Minimum des Approximationspolynoms wird zum neuen Ausgangspunkt. Beachten Sie dazu auch die Abbildung 5.

Beide Herangehensweisen für die Optimierung führen jedoch zu der Iterationsformel:

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

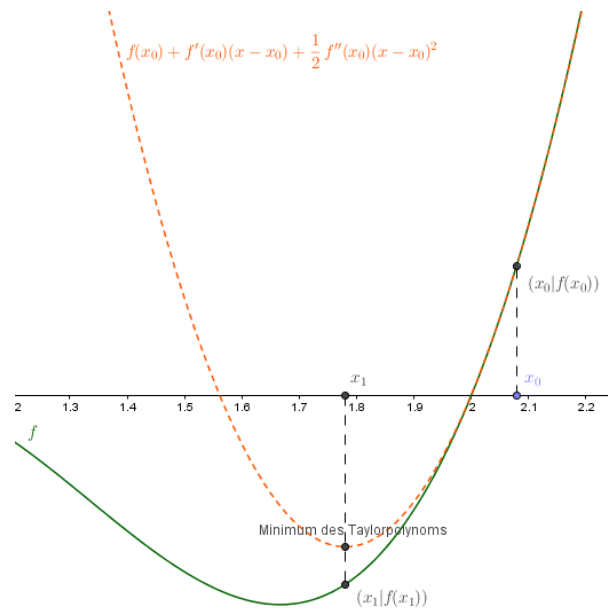


Abbildung 5: Approximation einer Funktion durch ein quadratisches Polynom und ein Iterationsschritt

4.2 AUFGABENSTELLUNG

4.2.1 Theorie-Erarbeitung

Recherchieren Sie die hier angesprochenen Themen. Gehen Sie in Ihrer Dokumentation zumindest auf folgende Punkte ein:

- Leiten Sie die Iterationsformel auf 2 Arten herbei (ein mal durch Nullsetzen der Ableitung, ein mal durch Approximation durch ein quadratisches Polynom). Recherchieren Sie dazu falls notwendig den Begriff "Taylorpolynom".
- Verwenden Sie ein konkretes Beispiel um zu zeigen, wie das Newtonverfahren funktioniert.
- Verwenden Sie das Newtonverfahren für die Funktion $f(x) = \frac{x^4}{4} - x^2 + 2x$ im Startpunkt 0. Was fällt Ihnen hier auf?
- Inwiefern ändert sich die Iterationsformel, wenn man das Newtonverfahren auf eine mehrdimensionale Funktion anwenden will?
- Recherchieren Sie Varianten des Newtonverfahrens, wie zum Beispiel das gedämpfte Newtonverfahren und erklären Sie diese.
- Verwenden Sie das gedämpfte Newtonverfahren für die Funktion $f(x) = \frac{x^4}{4} - x^2 + 2x$ im Startpunkt 0. Kann durch die „Dämpfung“ eine Verbesserung erzielt werden?

Dokumentieren Sie alles in einem gut strukturierten PDF.

4.2.2 Implementierung

Implementieren Sie die folgenden Punkte in C++:

- Schreiben Sie für das ein- und mehrdimensionale Newtonverfahren Funktionen, die zu einer mathematischen Funktion das Minimum findet.
- Verwenden Sie zum Übergeben einer mathematischen Funktion die vorimplementierte Klasse `Funktion`. Eine Dokumentation zu dieser, ist im Anhang dieses Dokuments zu finden. Ein Download dazu, wird auf der E-Learning-Plattform bereit gestellt.
- Achten Sie darauf, der Funktion weitere sinnvolle Parameter wie die gewünschte Rechengenauigkeit und diverse Startwerte zu übergeben.
- Schreiben Sie ein Programm, das dem User eine einfache Möglichkeit gibt, Ihren Implementierungen für (von Ihnen) vorimplementierte Funktionen zu testen.
- Visualisieren Sie die Ergebnisse.
- Achten Sie darauf, dass Ihre Programme entsprechende Warnmeldungen ausgeben, wenn etwas unerwartetes passiert.

Im Anhang dieses Dokuments finden Sie eine Liste von Funktionen, mit denen Sie Ihre Implementierung testen können.

4.2.3 Homepage

Erstellen Sie eine Homepage, die interessierten Leuten erklärt, worum es in Ihrem Projekt geht. Arbeiten Sie zumindest folgende Punkte ein:

- Erklären Sie alle Vorgänge anhand händisch durchgeführter Beispiele und Veranschaulichen Sie diese durch Grafiken und Animationen.
- Erstellen Sie ein oder mehrere Applets in GeoGebra und binden Sie diese auf der Homepage ein. Folgende Punkte bieten sich dafür an:
 - Veranschaulichen Sie eine Iteration des Newton Verfahrens über die Geometrische Konstruktion, durch Schmiegeparabeln. (Ansatz: Minimieren des Taylorpolynoms.)
 - Veranschaulichen Sie das eindimensionale Newton Verfahren, wobei durch einen Schieberegler oder Checkboxes, mehrere Iterationsschritte angezeigt oder ausgeblendet werden können.

- Veranschaulichen Sie das eindimensionale Newton Verfahren, wobei durch Drücken eines Buttons, ein Iterationsschritt durchgeführt wird.

Eine Hilfe zu GeoGebra und zur Erstellung von derartigen Applets wird auf der E-Learning-Plattform hochgeladen.

- Stellen Sie das entwickelte Programm und den Quellcode zur Verfügung und erklären, wie diese zu Verwenden sind.

4.2.4 *Warnung*

Es handelt sich bei den obigen Aufgaben nur um die Mindestanforderungen für Ihr Projekt. Erweitern Sie das Projekt in Eigeninitiative und besprechen Sie Ihre Ideen dazu in den Feedbackgesprächen.

Achten Sie auf korrektes wissenschaftliches Arbeiten und zitieren Sie alle Quellen nach gängigen Regeln.

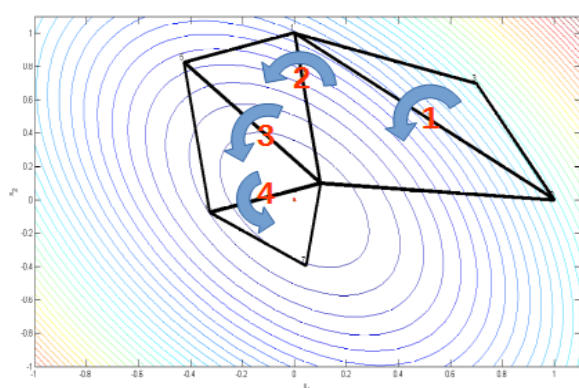
NICHTLINEARE OPTIMIERUNG OHNE NEBENBEDINGUNGEN: MEHRDIMENSIONALE VERFAHREN

5.1 EINFÜHRUNG

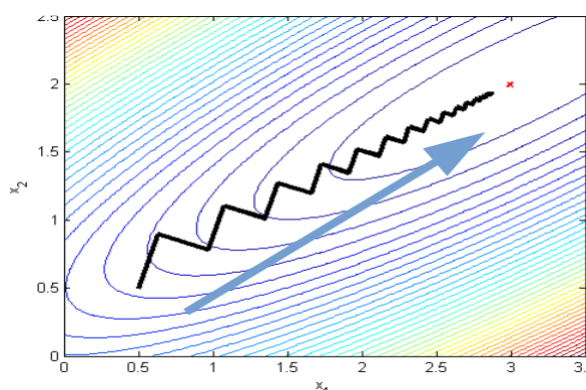
Ziel des Projekts ist es, verschiedene Mehrdimensionale Optimierungsverfahren zu implementieren und auf verschiedene Funktionen anzuwenden.

5.1.1 Das Nelder-Mead-Verfahren

Das Nelder-Mead-Verfahren startet mit einem beliebigen Polytop mit $n + 1$ Ecken, wobei n die Dimension angibt (im zweidimensionalen Fall also mit einem Dreieck). Nun betrachtet man die Funktionswerte der Eckpunkte und spiegelt den Punkt mit dem schlechtesten Funktionswert auf die andere Seite der übrigen Punkte. Nun betrachtet man den Funktionswert des neuen Punktes und vergrößert, beziehungsweise verkleinert das Polytop noch. Auf diese Weise „wandert“ das Polytop herum, und zieht sich schließlich um das Minimum zusammen. Das Verfahren ist auch genauer in den Vorlesungsunterlagen zu finden.



(a) Nelder-Mead-Verfahren



(b) Gradientenverfahren

Abbildung 6: Mehrdimensionale Optimierungsverfahren

5.1.2 Abstiegsrichtung

Wir wollen sogenannte Abstiegsverfahren betrachten. Dafür benötigen wir eine Definition:

Ein Vektor $\underline{d} \in \mathbb{R}^n$ heißt Abstiegsrichtung von f in einem Punkt $\underline{x} \in \mathbb{R}^n$, falls es ein \bar{t} gibt, sodass $f(\underline{x} + t\underline{d}) < f(\underline{x}) \quad \forall t \in (0, \bar{t})$

Ein nützlicher Satz ist auch dieser:

Gilt $\nabla f(\underline{x})^T \underline{d} < 0$, dann ist \underline{d} eine Abstiegsrichtung von f im Punkt \underline{x} .

Die Idee des Verfahrens ist die, dass man einen beliebigen Startpunkt annimmt und von diesem aus in eine Abstiegsrichtung weiter „wandert“. An der Stelle des resultierenden Punktes werden erneut Abstiegsrichtungen berechnet und erneut weitergewandert.

5.1.3 Das Gradientenverfahren

Das Gradientenverfahren verwendet als Abstiegsrichtung $-\nabla f(\underline{x})$. Aufgrund des obigen Satzes und wegen $-\nabla f(\underline{x})^T \nabla f(\underline{x}) < 0$ ist dies immer eine Abstiegsrichtung. Genauer gesagt ist das sogar die Richtung des steilsten Abstieges. In jedem Iterationsschritt wird (meistens mittels „Line Search“) das zu $-\nabla f(\underline{x})$ gehörende \bar{t} gesucht (siehe Definition von Abstiegsrichtung) und $\underline{x} - \bar{t} \nabla f(\underline{x})$ zum neuen Punkt gemacht. Eine genauere Beschreibung dieses Verfahrens findet sich auch in den Vorlesungsunterlagen.

Es gibt auch noch weitere Variationen des Gradientenverfahrens: Um die Minimierung der entstehenden eindimensionalen Funktion in Richtung des steilsten Abstieges zu umgehen, kann man alternativ dazu auch nur ein „Stück“ in Richtung des steilsten Abstieges wandern. Dabei kann man entweder um einen fixen Wert wandern, oder man macht das Stück abhängig von der Nummer der Iteration. Die Idee ist, dass man zuerst große Schritte macht und wenn man sich dem Minimum nähert, die Schrittweite verkleinert, damit man nicht über das Ziel hinausschießt. Als Schrittweite wird also meist eine gegen 0 konvergente Folge genommen, deren Reihe aber divergiert, z.B. $\frac{1}{\sqrt{n}}$, wobei n die Nummer der Iteration ist.

5.1.4 Koordinatenabstiegsmethode

Die Idee, der Koordinatenabstiegsmethode ist, dass man sich in jedem Schritt eine Koordinate i wählt, nach der man optimiert. Man beginnt bei einem beliebigen Startpunkt. In jedem Iterationsschritt setzt man alle Koordinaten des aktuellen Punktes \underline{x}^k in die Funktion ein, bis auf die, nach der man minimieren will. Die gewählte Koordinate i behält man als Variable bei. So entsteht eine eindimensionale Funktion:

$$f_i(y) = f(x_1^k, x_2^k, \dots, x_{i-1}^k, y, x_{i+1}^k, \dots, x_{n-1}^k, x_n^k)$$

Mittels „Line Search“ oder einer anderen Methode minimiert man nun die entstandene eindimensionale Funktion und erhält das Minimum $y^* := \arg \min f_i(y)$ („arg-min“ bedeutet hier, dass die Stelle, an der sich das Minimum befindet, und nicht der minimale Funktionswert genommen wird). Die entsprechende Koordinate i wird durch den gefundenen Wert ersetzt, und der neue Punkt der Iteration ergibt sich als $x^{k+1} = (x_1^k, x_2^k, \dots, x_{i-1}^k, y^*, x_{i+1}^k, \dots, x_{n-1}^k, x_n^k)$. Im Normalfall wählt man die zu optimierenden Koordinaten nach der Reihe. Ist man mit allen Koordinaten durch, so beginnt man wieder vorne bei der ersten. Verändert sich der Punkt innerhalb eines Durchganges der Koordinaten nicht, oder nur kaum, kann abgebrochen werden.

5.2 AUFGABENSTELLUNG

5.2.1 Theorie-Erarbeitung

Recherchieren Sie über alle Verfahren. Gehen Sie in Ihrer Dokumentation zumindest auf folgende Punkte ein:

- Erklären Sie alle Verfahren allgemein.
- Verwenden Sie konkrete Beispiele um zu zeigen, wie die Verfahren funktionieren.
- Zeigen Sie, dass die Koordinatenabstiegsmethode für die stetige Funktion

$$f(x, y) = \begin{cases} (x + y - 5)^2 + (x - y - 2)^2 & \text{falls } x \leq y \\ (x + y - 5)^2 + (x - y + 2)^2 & \text{falls } x > y \end{cases}$$

beginnend beim Startpunkt $(0, 0)$ nicht funktioniert.

- Erklären Sie anhand eines Contourplots, was Abstiegsrichtungen sind.
- Beweisen Sie mit der Ungleichung von Cauchy-Schwarz, warum der Gradient $\nabla f(x_0)$ die Richtung des steilsten Anstiegs angibt, und daher $-\nabla f(x_0)$ die Richtung des steilsten Abstiegs angibt.
- Erklären Sie mit Hilfe des letzten Resultats und einer Skizze, warum der Gradient, im Rechten Winkel auf die Höhengichtlinien im Contourplots steht.
- Erklären Sie mit dieser Überlegung und einer entsprechenden Skizze, warum der Satz über die Abstiegsrichtungen stimmt.
- Erklären Sie außerdem mit einer Skizze, warum beim normalen Gradientenverfahren, die Abstiegsrichtungen im rechten Winkel zueinander stehen.
- Recherchieren Sie über die oben angesprochenen Variationen des Gradientenverfahrens und erklären Sie diese.

- Erklären Sie Vor- und Nachteile der Verschiedenen Variationen des Gradientenverfahrens. Wenn man als Schrittweite eine Folge nimmt, wieso muss dann die dazugehörige Reihe unbedingt divergieren?

Dokumentieren Sie alles in einem gut strukturierten PDF.

5.2.2 Implementierung

Implementieren Sie die folgenden Punkte in C++:

- Schreiben Sie für alle drei Verfahren Funktionen, die zu einer zweidimensionalen mathematischen Funktion das Minimum findet.
- Verwenden Sie zum Übergeben einer mathematischen Funktion die vorimplementierte Klasse `Funktion`. Eine Dokumentation zu dieser, ist im Anhang dieses Dokuments zu finden. Ein Download dazu, wird auf der E-Learning-Plattform bereit gestellt.
- Achten Sie darauf, der Funktion weitere sinnvolle Parameter wie die gewünschte Rechengenauigkeit und diverse Startwerte zu übergeben.
- Schreiben Sie ein Programm, dass dem User eine einfache Möglichkeit gibt, Ihren Implementierungen für (von Ihnen) vorimplementierte Funktionen zu testen.
- Visualisieren Sie die Ergebnisse.
- Achten Sie darauf, dass Ihre Programme entsprechende Warnmeldungen ausgeben, wenn etwas unerwartetes passiert.

Im Anhang dieses Dokuments finden Sie eine Liste von Funktionen, mit denen Sie Ihre Implementierung testen können.

5.2.3 Homepage

Erstellen Sie eine Homepage, die interessierten Leuten erklärt, worum es in Ihrem Projekt geht. Arbeiten Sie zumindest folgende Punkte ein:

- Erklären Sie alle Vorgänge anhand händisch durchgeführter Beispiele und Veranschaulichen Sie diese durch Grafiken.
- Zeigen Sie anhand von Grafiken, wie sich der aktuelle Punkt der Iteration im Koordinatensystem bewegt (Zick-Zack-Kurs bei Gradientenverfahren und Koordinatenabstiegsmethode).

- Vergleichen Sie die einzelnen Verfahren, indem Sie den Weg, den die Iteration bei den verschiedenen Verfahren zurücklegt, in einer Grafik einzeichnen und die Anzahl der Iterationen für eine bestimmte Genauigkeit betrachten.
- Suchen Sie ein Tool, mit dem man gut zweidimensionale Funktionen darstellen kann und erklären, wie man damit umgeht. Erklären Sie insbesondere wie Contourplots entstehen. Was sagt es aus, wenn Höhengichtlinien näher beisammen liegen?
- Erklären Sie die graphischen Resultate aus dem Theorie Teil allgemein Verständlich.
- Wie kann man in derartigen Graphiken eine Nebenbedingung $g(x, y) = 0$ einbauen? Erklären Sie in einer Infographik den Unterschied zwischen Optimierung mit und ohne Nebenbedingung.
 - Versuchen Sie zu erklären, wie man in beiden Fällen (mit oder ohne Nebenbedingung) an der Graphik erkennen kann?
 - Recherchieren Sie für Optimierung mit einer einzigen Nebenbedingung das „Lagrange’sche Multiplikatoren“ und erklären, wie man das resultat Graphisch Interpretieren kann. Erklären Sie alles allgemein verständlich auf der Homepage.
- Stellen Sie das entwickelte Programm und den Quellcode zur Verfügung und erklären, wie diese zu Verwenden sind.

5.2.4 *Warnung*

Es handelt sich bei den obigen Aufgaben nur um die Mindestanforderungen für Ihr Projekt. Erweitern Sie das Projekt in Eigeninitiative und besprechen Sie Ihre Ideen dazu in den Feedbackgesprächen.

Achten Sie auf korrektes wissenschaftliches Arbeiten und zitieren Sie alle Quellen nach gängigen Regeln.

NICHTLINEARE OPTIMIERUNG OHNE NEBENBEDINGUNGEN: EINDIMENSIONALE VERFAHREN

6.1 EINFÜHRUNG

Ziel des Projekts ist es, verschiedene Eindimensionale Optimierungsverfahren zu implementieren und auf verschiedene Funktionen anzuwenden.

6.1.1 Unimodale Funktionen

Sei $[a, b]$ ein Intervall und $f : [a, b] \rightarrow \mathbb{R}$ eine Funktion. Wir nennen f strikt unimodal auf $[a, b]$ falls f genau ein lokales Minimum in $[a, b]$ hat.

Die verwendeten Verfahren dieses Projekts basieren auf dem folgendem Hauptsatz: Für unimodale Funktionen und $a < x < y < b$ gilt:

$$(i) \quad f(x) \geq f(y) \Rightarrow \min_{[a,b]} f = \min_{[x,b]} f.$$

$$(ii) \quad f(x) \leq f(y) \Rightarrow \min_{[a,b]} f = \min_{[a,y]} f.$$

Dieser Satz wird auch in Abbildung 7 dargestellt.

6.1.2 Goldener Schnitt, Fibonacciverfahren

Ausgangspunkt beider Verfahren sind eine Funktion und ein Intervall $[a, b]$, auf dem die Funktion unimodal ist. Nach bestimmten Kriterien werden zwei Punkte x und y im Inneren des Intervalls gewählt, sodass gilt: $a < x < y < b$. Dann werden die Funktionswerte $f(x)$ und $f(y)$ miteinander verglichen. Je nach Funktionswerten kann nun obiger Hauptsatz angewendet werden und das Suchintervall kann auf $[x, b]$ beziehungsweise $[a, y]$ verkleinert werden. Im nächsten Iterationsschritt beginnt man von Vorne. Details zu den beiden Verfahren finden sich auch in den Vorlesungsunterlagen.

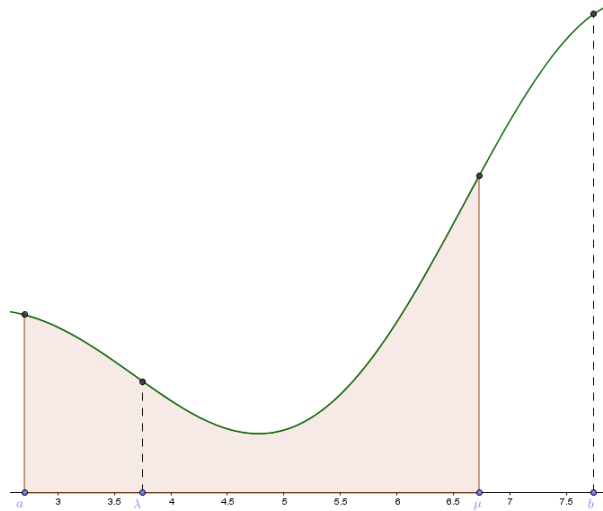


Abbildung 7: Hauptsatz zur Optimierung unimodaler Funktionen

6.1.3 Bisektionsverfahren

Das klassische Bisektionsverfahren wird verwendet, um Nullstellen zu suchen. Sei f eine stetige Funktion mit einer einzigen Nullstelle in $[a, b]$ und $f(a) < 0 < f(b)$. Untersucht man nun, ob das $f(\frac{a+b}{2})$, größer, kleiner oder gleich 0 ist, kann man ein neues kleineres Suchintervall berechnen. Ist zum Beispiel $f(\frac{a+b}{2}) > 0$, so wird man die Suche auf $[a, \frac{a+b}{2}]$ einschränken.

Ein ähnliches Verfahren lässt sich zum Minimieren entwickeln.

6.2 AUFGABENSTELLUNG

6.2.1 Theorie-Erarbeitung

Recherchieren Sie die hier angesprochenen Themen. Gehen Sie in Ihrer Dokumentation zumindest auf folgende Punkte ein:

- Veranschaulichen Sie den obigen Hauptsatz über unimodale Funktionen mit einer Skizze und erklären anschaulich, warum dieser gilt.
- Erklären Sie Goldener Schnitt und Fibonacci-Verfahren allgemein.
- Entwickeln Sie ein Bisektionsverfahren, um das Minimum einer differenzierbaren unimodalen Funktion zu finden. Erklären Sie dieses allgemein und veranschaulichen Sie es mit einer Skizze.
- Verwenden Sie konkrete Beispiele um zu zeigen, wie die 3 Verfahren funktionieren.

- Recherchieren oder erforschen Sie, wieso es sinnvoll ist, den Goldenen Schnitt für die Suche nach neuen Intervallgrenzen zu verwenden. Wie kann dadurch die Anzahl der Rechenschritte verkleinert werden?
- Recherchieren Sie, in welchem Sinn das Fibonacci-Verfahren optimale Stützstellen zur Intervallreduktion liefert.
- Kann man auch ohne Programm und ohne Berechnung sagen, wie viele Iterationsschritte für eine bestimmte Genauigkeit erforderlich sind? Wie viele Schritte sind bei den jeweiligen Verfahren notwendig, um den Fehler zu halbieren?

Dokumentieren Sie alles in einem gut strukturierten PDF.

6.2.2 Implementierung

Implementieren Sie die folgenden Punkte in C++:

- Schreiben Sie für alle drei Verfahren Funktionen, die zu einer mathematischen Funktion das Minimum findet.
- Verwenden Sie zum Übergeben einer mathematischen Funktion die vorimplementierte Klasse `Funktion`. Eine Dokumentation zu dieser, ist im Anhang dieses Dokuments zu finden. Ein Download dazu, wird auf der E-Learning-Plattform bereit gestellt.
- Achten Sie darauf, der Funktion weitere sinnvolle Parameter wie die gewünschte Rechengenauigkeit und diverse Startwerte zu übergeben.
- Schreiben Sie ein Programm, das dem User eine einfache Möglichkeit gibt, Ihren Implementierungen für (von Ihnen) vorimplementierte Funktionen zu testen.
- Visualisieren Sie die Ergebnisse.
- Achten Sie darauf, dass Ihre Programme entsprechende Warnmeldungen ausgeben, wenn etwas unerwartetes passiert.

Im Anhang dieses Dokuments finden Sie eine Liste von Funktionen, mit denen Sie Ihre Implementierung testen können.

6.2.3 Homepage

Erstellen Sie eine Homepage, die interessierten Leuten erklärt, worum es in Ihrem Projekt geht. Arbeiten Sie zumindest folgende Punkte ein:

- Erklären Sie alle Vorgänge anhand händisch durchgeführter Beispiele und Veranschaulichen Sie diese durch Grafiken und Animationen.

- Erstellen Sie ein oder mehrere Applets in GeoGebra und binden Sie dieser auf der Homepage ein. Folgende Punkte bieten sich dafür an:
 - Veranschaulichen Sie mit einem Applet den Hauptsatz.
 - Veranschaulichen Sie die Verfahren, wobei durch einen Schieberegler oder Checkboxen, mehrere Iterationsschritte angezeigt oder ausgeblendet werden können.
 - Veranschaulichen Sie die Verfahren, wobei durch Drücken eines Buttons, ein Iterationsschritt durchgeführt wird.

Eine Hilfe zu GeoGebra und zur Erstellung von derartigen Applets wird auf der E-Learning-Plattform hochgeladen.

- Stellen Sie das entwickelte Programm und den Quellcode zur Verfügung und erklären, wie diese zu Verwenden sind.

6.2.4 *Warnung*

Es handelt sich bei den obigen Aufgaben nur um die Mindestanforderungen für Ihr Projekt. Erweitern Sie das Projekt in Eigeninitiative und besprechen Sie Ihre Ideen dazu in den Feedbackgesprächen.

Achten Sie auf korrektes wissenschaftliches Arbeiten und zitieren Sie alle Quellen nach gängigen Regeln.

LINEARE OPTIMIERUNG: SIMPLEXVERFAHREN

7.1 EINFÜHRUNG

Ziel des Projekts ist es, lineare Optimierungsprogramme zu formulieren, zu verstehen, grafisch darzustellen und mit dem Simplexverfahren zu lösen.

7.1.1 Lineares Optimierungsprogramm

Ein lineares Optimierungsprogramm ist eine Optimierungsaufgabe, bei der sowohl die Zielfunktion, als auch die Nebenbedingungen lineare Funktionen sind. Variablen dürfen also nur mit Skalaren multipliziert und die so entstandenen Terme miteinander addiert werden.

$$\begin{aligned} x_1 + x_2 &\rightarrow \max! \\ x_2 &\leq 4 - 2x_1 \\ -x_1 - 2x_2 &\geq -3 \end{aligned}$$

ist ein lineares Optimierungsprogramm, während

$$\begin{aligned} x_1 x_2 &\rightarrow \max! \\ |x_2| &\leq 4 - 2x_1 \\ -x_1 - 2x_2 &\geq -3 \end{aligned}$$

aufgrund der Multiplikation der Variablen und dem Betrag keines ist.

7.1.2 Linearisierung

In manchen Fällen kann man das Programm durch Einführen neuer Variablen linearisieren. Bei Beträgen, wie $|x_2|$ kann man durch Einführen einer Variable y und geschickt gewählte Ungleichungen an y und x_2 sicher stellen, dass $y = |x_2|$ gilt.

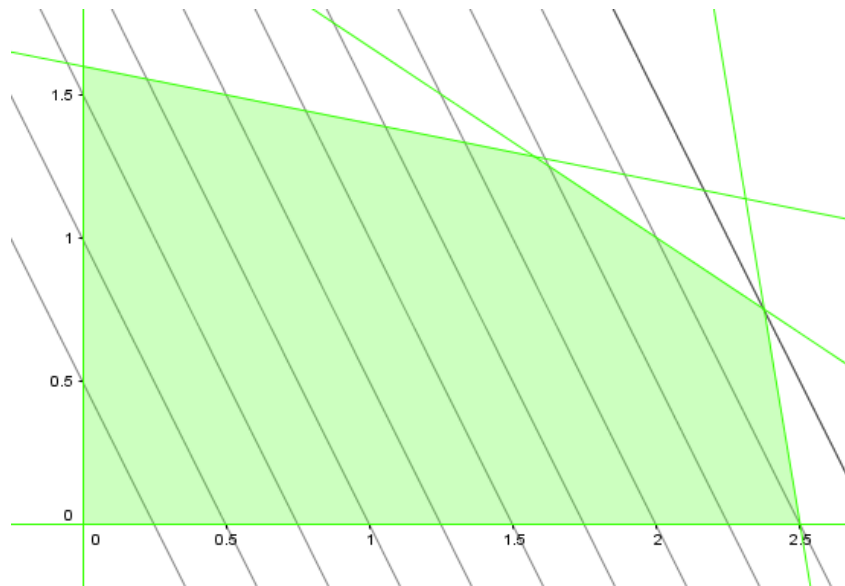


Abbildung 8: Grafische Darstellung eines linearen Programms. Die grünen Linien repräsentieren die Ungleichungen, die grüne Fläche den zulässigen Bereich. Die schwarzen Linien repräsentieren Höenschichtlinien der Zielfunktion.

7.1.3 Graphische Darstellung

Im zweidimensionalen Fall lässt sich ein Lineares Optimierungsprogramm graphisch sehr gut darstellen und lösen. Ungleichungen werden zu Halbebenen und die Zielfunktion kann anhand von parallelen Höenschichtlinien dargestellt werden. Siehe dazu Abbildung 8.

7.2 AUFGABENSTELLUNG

7.2.1 Theorie-Erarbeitung

- Recherchieren und erklären Sie folgende Begriffe (allgemein und anhand von Beispielen): linear, lineares Optimierungsprogramm, Standardform, Transformation zur Standardform, erweiterte Standardform, zulässiger Bereich, Zielfunktion, Polytop, beschränktes Optimierungsprogramm, lösbares Optimierungsprogramm, Schattenpreise, zulässige Erhöhung. Alle diese Begriffe sollten sich im Skriptum finden (natürlich ist es erlaubt, auch andere Quellen zu verwenden).
- Stellen Sie ein zweidimensionales lineares Programm grafisch dar und erklären Sie die Vorgehensweise. Erklären Sie auch, wie man es grafisch löst und tun Sie

das auch. Erklären Sie, wieso die Höhenschichtlinien der Zielfunktion Parallel sind.

- Erklären Sie anhand von Beispielen, wie man Operationen wie die Minimumsbildung oder den Betrag linearisieren kann.
- In der folgenden Tabelle ist der Inhalt ausgewählter Vitamine in Milligramm pro Kilogramm in einigen Lebensmitteln (Quelle: www.vitamine.com) und die empfohlene Tagesdosis für diese Vitamine dargestellt (Quelle: de.wikipedia.org/wiki/Recommended_Daily_Allowance).

	Champignons	Erbsen	Äpfel	Sojabohnen	RDA
Vitamin B1	0,9	8	0,4	9,7	1,1
Vitamin B2	4,2	2,7	0,3	4,9	1,4
Vitamin B6	0,7	1,2	1	10	1,4
Vitamin C	50	20	100	10	80
Vitamin E	1	15	5	15	12

Wenn die empfohlene Tagesdosis möglichst gut nur durch diese vier Lebensmittel abgedeckt werden soll, wie viel von welchem Lebensmittel sollte man dann zu sich nehmen? Modellieren Sie dazu ein Optimierungsproblem bei dem die summierte absolute Abweichung der konsumierten Milligramm an Vitaminen zu der empfohlenen Dosis minimiert werden soll! Linearisieren Sie dieses Optimierungsproblem falls notwendig und lösen Sie es (z.B. in Excel Solver)!

Da sich die Größenverhältnisse bei den verschiedenen Vitaminen stark variieren (Vitamin C wird viel mehr gebraucht und auch konsumiert!) ist es sinnvoll, die relative Abweichung zur empfohlenen Tagesdosis zu betrachten. Adaptieren Sie Ihr Optimierungsproblem, sodass die summierte relative Abweichung (in Prozent) der konsumierten Milligramm an Vitaminen zu der empfohlenen Dosis minimiert werden soll! Linearisieren Sie dieses Optimierungsproblem falls notwendig und lösen Sie es!

- Erklären Sie den Simplex Algorithmus anhand eines händisch gerechneten Beispiels.

Dokumentieren Sie alles in einem gut strukturierten PDF.

7.2.2 Implementierung

Implementieren Sie die folgenden Punkte in C++:

- Implementieren Sie den Simplex Algorithmus. Schreiben Sie dafür eine Funktion mit der Signatur
`double* lpsolve(int n, double *c, int k, double **A, double *b).`

n gibt die Anzahl der Variablen an, k die Anzahl der Nebenbedingung. Die Vektoren c, b und die Matrix A sind entsprechend der Vorlesung zu übergeben, sodass das Lineare Programm in Standardform

$$c^T x \rightarrow \max!$$

$$Ax \leq b$$

$$x \geq 0$$

ist. Rückgabe wird ein Array mit den entsprechenden Lösungswerten sein.

- Schreiben Sie ein Programm, dass dem User eine einfache Möglichkeit gibt, den Simplex Algorithmus für eigene Probleme anzuwenden. Zum Beispiel: Ein einfaches Kommandozeilenprogramm, das einen Pfad zu einem Textfile in folgendem Format erwartet.
 - in der ersten Zeile stehen zwei mit Abstand getrennte Zahlen n, k , die wie oben zu verstehen sind.
 - in der zweiten Zeile stehen n Fließkommazahlen, die mit Absänden getrennt sind, und den Vektor c repräsentieren.
 - nun folgen k Zeilen mit jeweils $n + 1$ Zahlen. Die ersten n Zahlen bilden eine Zeile der Matrix A , die letzte steht für den zugehörigen Eintrag im Vektor b (Jede Zeile entspricht daher einer Nebenbedingung.)
 - Beachten Sie zum Einlesen eines ähnlichen Files auch das Beispielprogramm im Anhang.

Testfiles für Ihr Problem finden Sie auf der Homepage.

- Erweitern Sie Ihr Programm, sodass auch eine Sensitivitätsanalyse ausgegeben werden kann.
- Achten Sie darauf, dass Ihre Programme entsprechende Warnmeldungen ausgeben, wenn etwas unerwartetes passiert.

7.2.3 Homepage

Erstellen Sie eine Homepage, die interessierten Leuten erklärt, worum es in Ihrem Projekt geht. Arbeiten Sie zumindest folgende Punkte ein:

- Erklären Sie den Simplex Algorithmus anhand eines händisch durchgerechneten Beispiels.
- Erstellen Sie ein GeoGebra Applet oder eine Animation, in der gezeigt wird, wie sich die Zielfunktion verschiebt und schließlich das Maximum erreicht und binden Sie diese auf der Homepage ein. Eine Hilfe zu GeoGebra und zur Erstellung von derartigen Applets wird auf der E-Learning-Plattform hochgeladen.

- Erstellen Sie ein Tutorial, das erklärt, wie man den Simplex Algorithmus in Excel, Calc, oder einer anderen Software umsetzt.
- Erklären Sie, wie man das Beispiel mit den Vitaminen aus dem Theorieteil formuliert und linearisiert.
- Stellen Sie das entwickelte Programm und den Quellcode zur Verfügung und erklären, wie diese zu Verwenden sind.

7.2.4 *Warnung*

Es handelt sich bei den obigen Aufgaben nur um die Mindestanforderungen für Ihr Projekt. Erweitern Sie das Projekt in Eigeninitiative und besprechen Sie Ihre Ideen dazu in den Feedbackgesprächen.

Achten Sie auf korrektes wissenschaftliches Arbeiten und zitieren Sie alle Quellen nach gängigen Regeln.

Teil III

ANHANG

FUNKTIONENLISTE

Hier sind einige Testfunktionen aufgelistet, mit denen Sie Ihre Programme testen können.

Bevor Sie die Funktionen „blind“ in Ihr System einbauen, sollten Sie unbedingt diese plotten und überlegen, ob alle Voraussetzungen erfüllt sind.

Dokumentieren Sie die Ergebnisse auf der Homepage oder im Theorie-Teil.

8.1 EINDIMENSIONALE FUNKTIONEN

- $f_1(x) = 2x^2 + e^{-2x}$
- $f_2(x) = \frac{x^4}{4} - x^2 + 2x$
- $f_3(x) = x^5 + 5x^4 + 5x^3 - 5x^2 - 6x$
- $f_4(x) = x^4 - 16x^2 - 1$
- $f_5(x) = \ln(|x^3 + 5x - 5|)$
- $f_6(x) = \ln(|x^4 - 16x^2 - 1|)$

8.2 ZWEIDIMENSIONALE FUNKTIONEN

- $g_1(x, y) = 3x^2 + y^2 - 3xy - 3x$
- $g_2(x, y) = 100(y - x^2)^2 + (x - 1)^2$: man nennt diese Funktion auch Rosenbrock's Bananenfunktion.
- $g_3(x, y) = y^4 + 2x^2 - 3xy + 1$
- $g_4(x, y) = |xy| + x^2 + y^2 - 2x - 4y$
- $g_5(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$: man nennt diese Funktion auch Himmelblau Funktion

8.3 WEITERE INTERESSANTE TESTFUNKTIONEN

Noch mehr interessante Testfunktionen finden sich unter diesem Link:

https://en.wikipedia.org/wiki/Test_functions_for_optimization

HINWEISE FÜR DIE IMPLEMENTIERUNG

9.1 DIE KLASSE FUNKTION

Für einige Praktikumsthemen ist bei der Implementierung vorgegeben, dass Ihre Optimierungsverfahren ein Objekt der Klasse `Funktion` mittels Call by Reference übergeben wird.

9.1.1 *Idee*

Die Idee hinter der Klasse/Struct¹ ist, dass Ihre Verfahren als Parameter eine mathematische Funktion, sowie möglicherweise deren Ableitungen benötigen. Für unsere Anwendungen ist es am einfachsten für so eine mathematische Funktion eine Klasse zu erzeugen und als Memberfunktionen `double value(double x)` für die Auswertung an der Stelle x , sowie weitere Funktionen für die Ableitungen bereitzustellen. Durch eine Übergabe einer Instanz von dieser Klasse, können auf diese Art mathematische Funktionen übergeben werden.

Hier ein Beispiel dazu:

```
class MathematischeFunktion {
public:
    double value( double x ) { return x*x-1; } //f(x)
    double x( double x ) { return 2*x; } //f'(x)
};

void auswerten(MathematischeFunktion &g, double x) {
    cout << "Funktionswert an der Stelle " << x << ": " << g.value(x) << endl;
    cout << "1. Ableitung an der Stelle " << x << ": " << g.x(x) << endl;
}

int main() {
```

¹ Klasse und Struct unterscheiden sich in C++ nur dadurch, dass Funktionen aus Klassen per default private sind, die in Structs sind public

```

    MathematischeFunktion f;
    auswerten(f, 3);
}

```

Will man aber nun zwei mathematische Funktionen f, g implementieren, die Funktion `auswerten` aber nicht weiter verändern, so bietet es sich an, von einer allgemeinen Klasse abzuleiten. Genau so eine allgemeine Klasse ist `Funktion`, welche Sie von der E-Learning Plattform herunterladen können und Ihre Projekte einbauen sollen. Verändern Sie diese Klasse aber unter keinen Umständen!

9.1.2 Details zur Verwendung

Das folgende Beispiel zeigt, wie Sie diese Funktion für 1-dimensionale Funktionen, die also von nur einem Parameter abhängen, verwenden können.

```

#include <iostream>
#include <cmath>
#include "Funktion.h"

using namespace std;

void auswerten(Funktion &g, double x, int ableitung = 0) {
    if( ableitung == 0 ) {
        cout << "Funktionswert an der Stelle " << x << ": " << g(x) << endl;
    } else if ( ableitung == 1 ) {
        cout << "1. Ableitung an der Stelle " << x << ": " << g.x(x) << endl;
    } else if ( ableitung == 2 ) {
        cout << "2. Ableitung an der Stelle " << x << ": " << g.xx(x) << endl;
    }
}

int main() {
    struct : Funktion {
        double value( double x ) { return x*x-1; } //f(x)
        double x( double x ) { return 2*x; } //f'(x)
        double xx( double x ) { return 2; } //f''(x)
    } f;

    auswerten(f, 3);
    auswerten(f, 3, 1);
    auswerten(f, 3, 2);
}

```

```

struct : Funktion {
    double value( double x ) { return sin(x); } //h(x)
    double x( double x ) { return cos(x); } //h'(x)
} h;

auswerten(h, M_PI);
auswerten(h, M_PI, 1);
//auswerten(h, M_PI, 2); // Programm endet mit Exception: runtime_error,
                        // weil zweite Ableitung von h nicht definiert.
}

```

Wichtige Anmerkungen:

- Die Klasse ist so geschrieben, dass der `()`-operator überladen wird. Sie können also so wie in der Funktion `auswerten` den Ausdruck $g(x)$ als Abkürzung für $g.value(x)$ verwenden.
- Sie können, müssen aber nicht!!, auch die Mathematischen ersten $f'(x) = f_x(x)$ und zweiten Ableitungen verwenden. Dazu definieren Sie die Funktionen so wie im Beispiel gezeigt. Höhere Ableitungen werden nicht gebraucht und sind daher nicht implementiert.
- Wie sie sehen können, bekommen die Abgeleiteten Klassen/Structs keinen Namen. Da wir nur ein einziges Objekt dieser Klasse wollen, ist dieses Vorgehen Sinnvoll.
- Um `sin(x)`, `cos(x)`, `M_PI`,... zu verwenden, ist das Inkludieren von `cmath` notwendig.
- Wenn Sie ähnliche Funktionen, die derartige Objekte als Argumente haben (so wie `auswerten`), schreiben, ist unbedingt ein call by reference Funktion `&f` zu verwenden.²

Im zwei dimensionalen Fall, wenn die Funktion also von x und y abhängen soll, gehen Sie sehr ähnlich vor. Hier können Sie folgende Funktionen implementieren

- `double value(double x, double y)` für die Funktionswerte selber. Sie können wie oben statt $f.value(x,y)$ auch $f(x,y)$ verwenden, da auch hier der `()`-Operator überladen ist.
- `double x value(double x, double y)` für die partielle Ableitung $f_x = \frac{\partial f}{\partial x}$.
- `double y value(double x, double y)` für die partielle Ableitung $f_y = \frac{\partial f}{\partial y}$.

² Man kann auch Pointer verwenden, aber dadurch wird nicht unbedingt die Lesbarkeit des Codes erhöht.

- `double xx value(double x, double y)` für die partielle Ableitung $f_{xx} = \frac{\partial^2 f}{\partial x^2}$.
- `double yy value(double x, double y)` für die partielle Ableitung $f_{yy} = \frac{\partial^2 f}{\partial y^2}$.
- `double xy value(double x, double y)` für die partielle Ableitung $f_{xy} = \frac{\partial^2 f}{\partial x \partial y}$.
- `double yx value(double x, double y)` für die partielle Ableitung $f_{yx} = \frac{\partial^2 f}{\partial y \partial x}$.

Sehen Sie sich auch die Beispiele auf der E-Learning-Plattform an.

9.2 MATRIX AUS FILE EINLESEN

Mit dem nachfolgenden Code wird eine $n \times k$ -Matrix (n Zeilen, k Spalten) eingelesen und wieder ausgegeben. n und k stehen in der ersten Zeile des Textfiles.

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
    // Matrix: Einlesen
    /*
    Beispielfile: matrix.txt
        3 4
        -1 1.2 3 6
        2.2 0 1 -1.2
        0 -12 0 1
    */
    ifstream file("matrix.txt");
    int n,k; // n Zeilen, k Spalten
    file >> n >> k;
    double** matrix = new double*[n];

    for (int i = 0; i < n; i++) {
        matrix[i] = new double[k];
        for (int j = 0; j < k; j++) {
            file >> matrix[i][j];
        }
    }

    //Matrix verwenden: Als Beispiel geben wir sie einfach aus.
```



```
    cout << "Ausgabe" << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < k; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```