DIPLOMARBEIT

Approximation elliptischer Optimalsteuerprobleme

angefertigt von

René Simon
geb. 13. April 1976

an der

Fakultät II – Mathematik und
Naturwissenschaften
Technische Universität Berlin

bei

Dr. Arnd Rösch

Berlin, den 8. Juli 2004

Inhaltsverzeichnis

Kapitel 1

Linear-quadratische elliptische Optimalsteuerungsprobleme

1.1 Einleitung

Sehr viele praktisch relevante Prozesse werden durch partielle Differentialgleichungen beschrieben. Beispielsweise werden Wärmeleitung, Schwingungen, Diffusion, elektromagnetische Wellen, Strömungen, Erstarrungsvorgänge und andere physikalische Phänomene durch partielle Differentialgleichungen erfasst. Dabei gibt es häufig die Möglichkeit über Steuerfunktionen oder Parameter in diese Prozesse einzugreifen. Optimierungsprobleme bei partiellen Differentialgleichungen sind daher von großem praktischem Interesse.

Die Vielfalt von Typen partieller Differentialgleichungen ist groß. In dieser Arbeit beschäftigen wir uns mit linearen elliptischen Differentialgleichungen. Das zu minimierende Zielfunktional ist quadratisch. Die Steuerfunktion ist auf dem ganzen Gebiet definiert. Diesen Typ nennt man linear-quadratisches elliptisches Optimalsteuerungsproblem mit verteilter Steuerung.

Für die numerische Lösung solcher Aufgaben ist es notwendig, die Gleichungen und die Steuerung zu diskretisieren. Anliegen dieser Arbeit ist es, bei einer speziellen Diskretisierung, den Diskretisierungsfehler abzuschätzen. Damit schafft man sich die Möglichkeit, a-priori Diskretisierungen festzulegen, mit denen man ein Optimalsteuerproblem mit vorgegebener Genauigkeit lösen kann.

Wir werden in dieser Arbeit das folgende Optimalsteuerproblem untersuchen:

$$\min J(y,u) = \frac{1}{2}||y - y_d||^2_{L^2(\Omega)} + \frac{\lambda}{2}||u||^2_{L^2(\Omega)} \tag{1.1}$$

(*Zielfunktional*) bei den Nebenbedingungen

$$\begin{aligned} -\Delta y + cy &= u \qquad \text{in } \Omega \\ y &= 0 \qquad \text{auf } \Gamma \end{aligned} \tag{1.2}$$

(*Zustandsgleichung*) sowie

$$a \leq u(x) \leq b \qquad \text{auf } \Omega \tag{1.3}$$

(*punktweise Beschränkungen an die Steuerfunktion*)

Zu minimieren ist das Zielfunktional bei Erfüllung der Zustandsgleichung und der Steuerbeschränkung.

Gegeben sind dabei $\lambda > 0$, $y_d \in L^2(\Omega)$, nichtnegative Funktion $c \in L^\infty(\Omega)$, reelle Zahlen $a \leq b$. Das Gebiet Ω sei beschränkt und habe einen $C^{0,1}$-Rand Γ. In späteren Kapiteln werden wir allerdings eine höhere Regularität des Gebietes fordern.

Als Raum für die Steuerfunktionen wählen wir den Hilbertraum $L^2(\Omega)$. Die Menge der *zulässigen Steuerungen* nennen wir U_{ad} und definieren also:

$$U_{\text{ad}} := \{u \in L^2(\Omega) : a \leq u(x) \leq b \quad \text{f.ü. auf } \Omega\}$$

In diesem Kapitel werden die theoretischen Grundlagen bereitgestellt. Dazu behandeln wir zunächst die schwache Lösungstheorie partieller Differentialgleichungen und gehen danach auf die Existenz eines Optimums ein. Das Ziel dabei ist, Optimalitätsbedingungen zu formulieren, die die Grundlage für die numerische Berechnung bilden. Dabei folgen wir weitgehend dem Vorlesungsskript von Tröltzsch [10].

1.2 Schwache Lösungstheorie partieller Differentialgleichungen

Nun folgt ein kurzer Abstecher in die schwache Lösungstheorie partieller Differentialgleichungen, der uns zu dem Hauptresultat führt, dass zu jedem $u \in L^2(\Omega)$ genau eine Lösung $y \in H_0^1(\Omega)$ existiert. Grundlage bildet die Variationsformulierung der Zustandsgleichung (1.2): Wir multiplizieren die Gleichung mit einer beliebigen aber festen Testfunktion $v \in C_0^\infty(\Omega)$ und integrieren über Ω:

$$-\int_\Omega v\Delta y \mathrm{d}x + \int_\Omega cyv\mathrm{d}x = \int_\Omega uv\mathrm{d}x$$

und nach partieller Integration

$$-\int_\Gamma v\partial_\nu y \mathrm{d}\Gamma + \int_\Omega \nabla y \cdot \nabla v \mathrm{d}x + \int_\Omega cyv\mathrm{d}x = \int_\Omega uv\mathrm{d}x$$

Da v auf Γ verschwindet, folgt schließlich

$$\int_\Omega \nabla y \cdot \nabla v \mathrm{d}x + \int_\Omega cyv \mathrm{d}x = \int_\Omega uv \mathrm{d}x.$$

Da $C_0^\infty(\Omega)$ dicht in $H_0^1(\Omega)$ liegt, können wir die Variationsformulierung auf $H_0^1(\Omega)$ ausdehnen.

Durch Einführung der Bezeichnung $V = H_0^1(\Omega)$ und Definition der Bilinearform $B : V \times V \to \mathbb{R}$ mit:

$$B[y, v] = \int_\Omega \nabla y \cdot \nabla v \mathrm{d}x + \int_\Omega cyv \mathrm{d}x \tag{1.4}$$

sowie des linearen und stetigen Funktionals $F : V \to \mathbb{R}$ durch

$$F(v) := (u, v)_{L^2(\Omega)}$$

nimmt die Variationsgleichung die allgemeine Form

$$B[y, v] = F(v) \quad \forall v \in V \tag{1.5}$$

an.

Grundlage für die Behandlung der Existenzfrage einer schwachen Lösung ist folgende Aussage:

Lemma 1.1 (Lax-Milgram). *Es sei V ein reeller Hilbertraum und $B : V \times V \to \mathbb{R}$ eine Bilinearform mit folgenden Eigenschaften:*

1. $|B[u, v]| \leq \alpha ||u||_V ||v||_V$ *(Stetigkeit bzw. Beschränktheit)*
2. $B[u, u] \geq \beta ||u||_V^2$ *(Elliptizität)*

Dabei sind α und β positive Konstanten.

Ist $F : V \to \mathbb{R}$ ein lineares und stetiges Funktional, dann existiert genau ein $u \in V$, welches die Variationsgleichung

$$B[u, v] = F(v) \qquad \forall v \in V$$

löst.

Um die Elliptizität unserer Bilinearform zeigen zu können, benötigen wir noch die folgende Abschätzung:

Lemma 1.2 (Friedrich'sche Ungleichung). Γ *sei ein* $C^{0,1}$*-Rand. Dann existiert eine nur von* Ω *abhängige Konstante* $\gamma(\Omega)$*, so dass die Ungleichung*

$$\int_\Omega y^2\,dx \leq \gamma(\Omega)\int_\Omega |\nabla y|^2\,dx$$

für alle $y \in H_0^1(\Omega)$ *erfüllt ist.*

Damit können wir nun die Existenz einer eindeutigen Lösung zeigen.

Satz 1.1. *Es sei* Γ *ein* $C^{0,1}$*-Rand. Dann besitzt die partielle Differentialgleichung (1.2) für jedes* $u \in L^2(\Omega)$ *genau eine schwache Lösung* $y \in H_0^1(\Omega)$*.*

Beweis. Da wir das Lemma von Lax-Milgram anwenden wollen mit $V = H_0^1(\Omega)$ müssen wir die Beschränktheit und Elliptizität der Bilinearform (1.4) nachweisen. Die Beschränktheit von B sieht man mit Hilfe der Hölder-Ungleichung:

$$\begin{aligned}\left|\int_\Omega \nabla y\cdot\nabla \mathrm{d}x\right| &\leq \int_\Omega |\nabla y||\nabla v|\mathrm{d}x\\ &\leq \left(\int_\Omega |\nabla y|^2\mathrm{d}x\right)^{\frac{1}{2}}\left(\int_\Omega |\nabla v|^2\mathrm{d}x\right)^{\frac{1}{2}}\\ &\leq ||y||_{H_0^1(\Omega)}||v||_{H_0^1(\Omega)}.\end{aligned}$$

und

$$\begin{aligned}\left|\int_\Omega cyv\mathrm{d}x\right| &\leq ||c||_{L^\infty(\Omega)}\left(\int_\Omega |y|^2\mathrm{d}x\right)^{\frac{1}{2}}\left(\int_\Omega |v|^2\mathrm{d}x\right)^{\frac{1}{2}}\\ &\leq ||c||_{L^\infty(\Omega)}||y||_{H_0^1(\Omega)}||v||_{H_0^1(\Omega)}\end{aligned}$$

Zum Beweis der Elliptizität benutzen wir die Friedrich'sche Ungleichung und die Tatsache, dass $c(x)$ nichtnegativ ist:

$$\begin{aligned}B[y,y] = \int_\Omega |\nabla y|^2\mathrm{d}x + \int_\Omega cy^2\mathrm{d}x &\geq \frac{1}{2}\int_\Omega |\nabla y|^2\mathrm{d}x + \frac{1}{2}\int_\Omega |\nabla y|^2\mathrm{d}x\\ &\geq \frac{1}{2}\int_\Omega |\nabla y|^2\mathrm{d}x + \gamma(\Omega)^{-1}\frac{1}{2}\int_\Omega y^2\mathrm{d}x\\ &\geq \frac{1}{2}\min(1,\gamma(\Omega)^{-1})||y||^2_{H_0^1(\Omega)}.\end{aligned}$$

□

1.3 Existenz einer optimalen Steuerung

Im letzten Abschnitt haben wir gezeigt, dass zu jedem $u \in U_{\mathrm{ad}}$ genau eine schwache Lösung $y \in H_0^1(\Omega)$ der Zustandsgleichung existiert. Sie heißt zu u gehöriger Zustand und

liegt im Zustandsraum $Y := H_0^1(\Omega)$. Um die Zugehörigkeit zu verdeutlichen, schreiben wir $y = y(u)$.

Definition 1.1. *Eine Steuerung $\bar{u} \in U_{ad}$ heißt optimale Steuerung und $\bar{y} = y(\bar{u})$ optimaler Zustand, wenn*

$$J(\bar{y}, \bar{u}) \leq J(y, u)$$

für alle $u \in U_{ad}$ mit zugehörigem Zustand $y = y(u)$ gilt.

Um die Frage der Existenz einer optimalen Steuerung zu klären, formulieren wir die Optimalsteuerungsaufgabe um. Wir führen dazu einen Operator $G : L^2(\Omega) \to H_0^1(\Omega)$ ein, welcher jeder Steuerung $u \in L^2(\Omega)$ den nach Satz 1.1 eindeutig existierenden zugehörigen Zustand $y \in H_0^1(\Omega)$ zuordnet. Dieser ist linear und stetig. Außerdem bezeichne E den Einbettungsoperator $E : H^1(\Omega) \to L^2(\Omega)$.Wegen

$$||y||_{L^2(\Omega)} \leq ||y||_{H^1(\Omega)}$$

ist dieser linear und stetig. Damit können wir den Lösungsoperator $S = EG$ definieren, d.h.: $S : L^2(\Omega) \to L^2(\Omega)$ mit $S : u \mapsto y$. Dieser ist natürlich ebenso linear und stetig. Die Verwendung von S hat den Vorteil, dass der Hilbertraum-adjungierte Operator S^* ebenfalls im Raum $L^2(\Omega)$ wirkt. Mithilfe von S nimmt die Optimalsteuerungsaufgabe folgende einfache Form einer quadratischen Optimierungsaufgabe an:

$$\min_{u \in U_{\text{ad}}} f(u) = \min_{u \in U_{\text{ad}}} \frac{1}{2}||Su - y_d||^2_{L^2(\Omega)} + \frac{\lambda}{2}||u||^2_{L^2(\Omega)} \tag{1.6}$$

Folgender Satz sichert die Existenz einer optimalen Steuerung:

Satz 1.2. *Gegeben seien reelle Hilberträume U und H, eine nichtleere, abgeschlossene und konvexe Menge $U_{ad} \subset U$, ein $y_d \in H$ und eine Konstante $\lambda \geq 0$. $S : U \to H$ sei ein linearer und stetiger Operator. Dann besitzt die quadratische Optimierungsaufgabe*

$$\min_{u \in U_{ad}} f(u) = \min_{u \in U_{ad}} \frac{1}{2}||Su - y_d||^2_H + \frac{\lambda}{2}||u||^2_U \tag{1.7}$$

eine optimale Lösung $\bar{u}$. Im Fall $\lambda > 0$ ist diese eindeutig bestimmt.

Beweis. Das Funktional $f(u)$ ist nach unten durch 0 beschränkt. Daher existiert das Infimum j aller möglichen Funktionswerte zu zulässigen Steuerungen

$$j := \inf_{u \in U_{\text{ad}}} f(u)$$

Sei $\{u_n\}$ eine Infimalfolge aus U_{ad}, d.h. $f(u_n) \to j$ für $n \to \infty$. Die Menge U_{ad} ist als beschränkte, konvexe und abgeschlossene Teilmenge eines Hilbertraums schwach folgenkompakt. Es existiert also eine Teilfolge $\{u_{n_k}\}$, die schwach gegen ein $\bar{u} \in U$ konvergiert.

Die Abgeschlossenheit und Konvexität von U_{ad} sichert die schwache Abgeschlossenheit, d.h. $\bar{u} \in U_{\text{ad}}$.

Jeder lineare stetige Operator ist schwach stetig. Daher folgt in H:

$$Su_{n_k} \rightharpoonup S\bar{u}, \qquad k \to \infty$$

f ist als stetige und konvexe Funktion auch schwach halbstetig nach unten, d.h.

$$f(\bar{u}) \leq \liminf_{k\to\infty} f(u_{n_k}) = j$$

Da j das Infimum ist, muss Gleichheit gelten. Damit ist $\bar{u}$ optimale Steuerung.

Die Eindeutigkeit für $\lambda > 0$ folgt aus der strengen Konvexität von f: Angenommen es existieren $u_1 \neq u_2$ mit $f(u_1) = f(u_2) = \inf_{u\in U_{\text{ad}}} f(u)$. Da U_{ad} konvex ist, ist mit $u_1, u_2 \in U_{\text{ad}}$ auch $\frac{1}{2}(u_1 + u_2) \in U_{\text{ad}}$.

Für eine streng konvexe Funktion f gilt für alle $\mu \in (0, 1)$

$$f((1-\mu)u_1 + \mu u_2) < (1-\mu)f(u_1) + \mu f(u_2) \quad \text{falls } u_1 \neq u_2$$

Mit $\mu = \frac{1}{2}$ folgt

$$f(\frac{1}{2}(u_1 + u_2)) < \frac{1}{2}f(u_1) + \frac{1}{2}f(u_2) = \inf_{u\in U_{\text{ad}}} f(u)$$

Daher muss $u_1 = u_2$ gelten, was die Eindeutigkeit beweist. □

Die Existenz einer optimalen Steuerung unseres Problems folgt sofort aus diesem Existenzsatz mit $U = L^2(\Omega), H = H_0^1(\Omega), S = EG$ und der Menge der zulässigen Steuerungen $U_{\text{ad}} = \{u \in L^2(\Omega) : a \leq u \leq b\}$, welche offensichtlich nichtleer, beschränkt, abgeschlossen und konvex ist.

1.4 Notwendige Optimalitätsbedingungen

Mit Hilfe der ersten Ableitung des Zielfunktionals lassen sich notwendige Optimalitätsbedingungen formulieren. Dazu betrachten wir wiederum die quadratische Optimierungsaufgabe der Form:

$$\min_{u\in U_{\text{ad}}} f(u) = \frac{1}{2}||Su - y_d||_H^2 + \frac{\lambda}{2}||u||_U^2 \tag{1.8}$$

Grundlage für die Herleitung notwendiger Optimalitätsbedingungen ist folgende Aussage:

Lemma 1.3. *Es sei U ein reeller Banachraum, $C \subset U$ eine konvexe Menge und $f : C \to \mathbb{R}$ ein auf C Gâteaux-differenzierbares reellwertiges Funktional. Mit $\bar{u} \in C$ sei eine Lösung der Aufgabe*

$$\min_{u\in C} f(u)$$

gegeben. Dann ist die Variationsungleichung

$$f'(\bar{u})(u - \bar{u}) \geq 0 \quad \forall u \in C$$

erfüllt.

Beweis. Sei $u \in C$ beliebig. Mit $u \in C$ und $\bar{u} \in C$ ist wegen der Konvexität von C auch die konvexe Linearkombination

$$u(t) = \bar{u} + t(u - \bar{u}) \quad t \in [0, 1]$$

ein Element aus C. Aus der Optimalität von $\bar{u}$ folgt $f(u(t)) \geq f(\bar{u})$ und damit

$$\frac{1}{t}\left(f(\bar{u} + t(u - \bar{u})) - f(\bar{u})\right) \geq 0.$$

Grenzübergang $t \searrow 0$ liefert die Variationsungleichung. □

Diese Aussage stellt eine notwendige Optimalitätsbedingung erster Ordnung dar. Sie bleibt auch gültig, wenn nur die Existenz aller Richtungsableitungen vorausgesetzt wird.

Falls zusätzlich f konvex ist, dann ist diese Optimalitätsbedingung auch hinreichend:

Für beliebiges $u \in C$ gilt aufgrund der Konvexität von f:

$$f(u) - f(\bar{u}) \geq f'(\bar{u})(u - \bar{u}).$$

Wegen der Variationsungleichung ist die rechte Seite nichtnegativ, woraus $f(u) \geq f(\bar{u})$ folgt, also die Optimalität von $\bar{u}$.

Diese Aussagen wenden wir jetzt auf die linear-quadratische Optimierungsaufgabe (1.8) an.

Satz 1.3. *Es seien reelle Hilberträume U und H, eine nichtleere, abgeschlossene und konvexe Menge $U_{ad} \subset U$, $y_d \in H$ sowie eine Konstante $\lambda \geq 0$ gegeben. Ferner sei $S : U \to H$ ein linearer und stetiger Operator. Das Element $\bar{u} \in U_{ad}$ löst genau dann die Aufgabe*

$$\min_{u \in U_{ad}} f(u) = \frac{1}{2}||Su - y_d||_H^2 + \frac{\lambda}{2}||u||_U^2,$$

wenn die Variationsungleichung

$$(S^*(S\bar{u} - y_d) + \lambda\bar{u}, u - \bar{u})_U \geq 0 \quad \forall u \in U_{ad} \tag{1.9}$$

erfüllt ist.

Beweis. Die Behauptung folgt sofort aus Lemma 1.3, da die Ableitung des Funktionals f gerade

$$f'(\bar{u}) = S^*(S\bar{u} - y_d) + \lambda\bar{u}$$

ist. □

Man kann die Variationsungleichung natürlich auch ohne Verwendung des adjungierten Operators S^* formulieren:

$$(S\bar{u} - y_d, Su - S\bar{u})_H + \lambda(\bar{u}, u - \bar{u})_U \geq 0 \quad \forall u \in U_{\text{ad}}$$

1.5 Adjungierter Operator und adjungierte Gleichung

Im vorigen Abschnitt haben wir gezeigt, dass ein Element $\bar{u} \in U_{\text{ad}}$ genau dann die quadratische Optimierungsaufgabe (1.8) löst, wenn die Variationsungleichung (1.9) erfüllt ist. Dabei muss der adjungierte Operator S^* noch bestimmt werden. Wir werden hier die dazu benötigten Techniken nicht näher erläutern, sondern nur die für unsere Optimierungsaufgabe nötigen Resultate angeben.

Lemma 1.4. *Die adjungierte Zustandsgleichung zu unserem Optimalsteuerungsproblem (1.1)-(1.3) ist gegeben durch*

$$\begin{aligned} -\Delta p + cp &= y - y_d \qquad \text{in } \Omega \\ p &= 0 \qquad \text{auf } \Gamma \end{aligned} \tag{1.10}$$

Der adjungierte Operator $S^* : L^2(\Omega) \to L^2(\Omega)$ unseres Optimalsteuerungsproblems ist der Lösungsoperator der adjungierten Gleichung (1.10), wobei wir hier wieder den Einbettungsoperator $E : H_0^1(\Omega) \to L^2(\Omega)$ analog zur Diskussion im Abschnitt 1.3 benutzt haben. Die adjungierte Gleichung lässt sich sehr schön mit der Methode der Transposition gewinnen, die z.B. in [10] diskutiert wird.

Damit erhalten wir

$$S^*(S\bar{u} - y_d) = S^*(\bar{y} - y_d) =: \bar{p}$$

wobei wir den zu $\bar{y}$ gehörigen adjungierten Zustand $\bar{p} = \bar{p}(x) \in H_0^1(\Omega)$ als schwache Lösung der adjungierten Gleichung definieren. Die rechte Seite der Differentialgleichung ist aus $L^2(\Omega)$, daher existiert genau eine Lösung $p = \bar{p} \in H_0^1(\Omega)$. Damit lässt sich die Variationsungleichung auch folgendermaßen formulieren

$$(\bar{p} + \lambda\bar{u}, u - \bar{u})_{L^2(\Omega)} \geq 0 \qquad \forall u \in U_{\text{ad}} \tag{1.11}$$

Eine Steuerung u ist also genau dann optimal, wenn sie mit y und p das Optimalitätssystem

$$\begin{aligned} -\Delta y + cy &= u & \qquad -\Delta p + cp &= y - y_d \\ y|_\Gamma &= 0 & \qquad p|_\Gamma &= 0 \\ & u \in U_{\text{ad}} & & \\ (p + \lambda u, v - u)_{L^2(\Omega)} &\geq 0 \quad \forall v \in U_{\text{ad}} & & \end{aligned}$$

erfüllt.

Es ist intuitiv einleuchtend, dass die Variationsungleichung auch punktweise formuliert werden kann.

Lemma 1.5. *Die Variationsungleichung (1.11) gilt genau dann, wenn für fast alle $x \in \Omega$ die punktweise Variationsungleichung in $\mathbb{R}$*

$$(\bar{p}(x) + \lambda \bar{u}(x))\,(v - \bar{u}(x)) \quad \forall v \in [a, b] \tag{1.12}$$

erfüllt ist.

Beweis. Wir setzen $z(x) = \bar{p}(x) + \lambda \bar{u}(x)$ und haben zu zeigen, dass fast überall

$$z(x)\,(v - \bar{u}(x)) \geq 0$$

gilt. Die Funktion z ist messbar auf Ω. Folglich sind fast alle Punkte von Ω Lebesgue-Punkte von z, also solche Punkte $x_0 \in \Omega$, in denen

$$\lim_{\epsilon \searrow 0} \frac{1}{|B(x_0, \epsilon)|} \int_{B(x_0,\epsilon)} z(x) \mathrm{d}x = z(x_0)$$

gilt. Ist ϵ hinreichend klein, dann ist die Kugel $B(x_0, \epsilon)$ vollständig in Ω enthalten, also ist das Integral erklärt. Das Gleiche gilt auch für $\bar{u}$. Damit sind fast alle Punkte von Ω gemeinsame Lebesgue-Punkte von z und $\bar{u}$.

Wir wählen einen beliebigen gemeinsamen Lebesgue-Punkt $x_0 \in \Omega$ sowie $v \in [a, b]$. Außerdem definieren wir

$$u(x) := \begin{cases} v & \text{in } B(x_0, \epsilon) \\ \bar{u}(x) & \text{sonst} \end{cases}$$

Offenbar ist $u \in U_{\text{ad}}$. Einsetzen in die Variationsungleichung (1.11) und Division durch $|B(x_0, \epsilon)|$ ergibt

$$0 \leq \frac{1}{|B(x_0, \epsilon)|} \int_\Omega z(x)(u(x) - \bar{u}(x)) \mathrm{d}x = \frac{1}{|B(x_0, \epsilon)|} \int_{B(x_0,\epsilon)} z(x)(v - \bar{u}(x)) \mathrm{d}x$$

Grenzübergang $\epsilon \searrow 0$ ergibt

$$0 \leq z(x_0)(v - \bar{u}(x_0))$$

□

Für die optimale Steuerung $\bar{u}$ muss dann also gelten (schwaches Minimumprinzip):

$$\min_{v\in[a,b]}(\bar{p}(x)+\lambda\bar{u}(x))v=(\bar{p}(x)+\lambda\bar{u}(x))\bar{u}(x)\quad\text{für fast alle } x\in\Omega \tag{1.13}$$

1. Fall: $\bar{p}(x)+\lambda\bar{u}(x)>0$

   Dann muss v den größtmöglichen Wert annehmen, d.h. $v=a$

2. Fall: $\bar{p}(x)+\lambda\bar{u}(x)<0$

   Dann muss v den kleinstmöglichen Wert annehmen, d.h. $v=b$

3. Fall: $\bar{p}(x)+\lambda\bar{u}(x)=0$

   Hier liefert (1.13) keine Aussage, aber die Gleichung $\bar{p}(x)+\lambda\bar{u}(x)=0$ selbst führt im Fall $\lambda>0$ zu

$$\bar{u}(x)=-\frac{1}{\lambda}\bar{p}(x)$$

Eine vollständige Diskussion dazu findet man in [7]. Dies führt auf die folgende Projektionsformel:

Satz 1.4. *Im Fall $\lambda>0$ ist $\bar{u}$ genau dann optimale Steuerung der Aufgabe (1.1)-(1.3), wenn mit dem zugehörigen adjungierten Zustand $\bar{p}$ die Beziehung*

$$\bar{u}(x)=\Pi_{[a,b]}\left\{-\frac{1}{\lambda}\bar{p}(x)\right\} \tag{1.14}$$

für fast alle $x\in\Omega$ erfüllt ist. Dabei bezeichnet $\Pi_{[a,b]}$ die Projektion von $\mathbb{R}$ auf $[a,b]$.

1.6 Regularität der Lösung

Im Abschnitt 1.2 haben wir gezeigt, dass zu jedem $u\in L^2(\Omega)$ genau eine Lösung $y\in H_0^1(\Omega)$ der partiellen Differentialgleichung (1.2) existiert. Für die weiteren Untersuchungen benötigen wir aber eine höhere Regularität der Lösung. Diese hängt dabei wesentlich von der Regularität der rechten Seite ab, wie folgendes Resultat von Bonnans und Casas [9] zeigt.

Lemma 1.6. *Für jede Funktion $u\in L^p(\Omega)$, $\Omega\subset\mathbb{R}^n$ gehört die Lösung y von (1.2) zu $H_0^1(\Omega)\cap W^{2,p}(\Omega)$ für beliebiges $p>n$.*

In unserem Fall wird Ω Teilmenge des $\mathbb{R}^2$ sein, sodass die eindeutige Lösung in $H_0^1(\Omega)\cap W^{2,p}(\Omega)$ liegt, falls $u\in L^p(\Omega)$ für $p>2$. Dieser Raum ist eingebettet in $C^{0,1}(\bar{\Omega})$. Das gleiche gilt für den adjungierten Zustand p mit $y_d\in L^p(\Omega)$.

Nach Definition liegen alle Steuerungen im $L^2(\Omega)$. Aufgrund der Steuerbeschränkungen sind alle Steuerungen betragsmäßig punktweise beschränkt. Somit gehören die Steuerungen zu $L^\infty(\Omega)$ und damit zu jedem $L^p(\Omega)$ für alle p.

Kapitel 2

Diskretisierung und Konvergenzuntersuchungen

Um das Optimalsteuerungsproblem (1.1)-(1.3) numerisch zu lösen, benutzen wir eine auf finite Elemente basierende Näherung. Dazu definieren wir eine Familie von Triangulationen $(T_h)_{h>0}$ von $\bar{\Omega}$. Zu jedem Element $T \in T_h$ betrachten wir zwei Parameter $\rho(T)$ und $\sigma(T)$. $\rho(T)$ bezeichnet den Durchmesser der Menge T, d.h. $\rho(T) = \sup\{|x-y| : x, y \in T\}$ und $\sigma(T)$ ist der Durchmesser der größten in T enthaltenen Kugel. Die Maschenweite des Gitters ist definiert durch $h = \max_{T\in T_h} \rho(T)$. Wir nehmen an, dass folgende Regularitätsannahmen erfüllt sind:

(A1) Es existieren zwei positive Konstanten ρ und σ, so dass für alle $T \in T_h$ und alle $h > 0$ gilt

$$\frac{\rho(T)}{\sigma(T)} \leq \sigma, \quad \frac{h}{\rho(T)} \leq \rho$$

(A2) Wir definieren $\bar{\Omega}_h = \bigcup_{T\in T_h} T$ und entsprechend Ω_h und Γ_h. Wir nehmen an, dass $\bar{\Omega}_h$ konvex ist und dass die Knoten von T_h, die auf dem Rand Γ_h liegen, Punkte aus Γ sind. Aus [12] ist bekannt:

$$|\Omega \setminus \Omega_h| \leq Ch^2$$

Zu jedem Dreieck T aus T_h, welches am Rand liegt, definieren wir ein weiteres Dreieck $\hat{T}$ wie folgt: Die Kante zwischen zwei Randknoten von T wird ersetzt durch das zugehörige gekrümmte Randstück von Γ. Die Vereinigung dieser gekrümmten Dreicke zusammen mit den inneren Dreiecken bezeichnen wir mit $\hat{T}_h$, so dass $\bar{\Omega} = \bigcup_{\hat{T}\in\hat{T}_h} \hat{T}$.

Die optimale Steuerung ist eigentlich nur auf Ω definiert. Sie kann aber stetig bis an den Rand fortgesetzt werden. Aufgrund der Optimalitätsbedingung wäre der Wert am

Rand gerade $\Pi_{[a,b]}(0)$. Für die spätere theoretische Konvergenzuntersuchung ist es daher sinnvoll, die Ansatzfunktionen für die Steuerung gerade gleich $\Pi_{[a,b]}(0)$ auf $\Omega \setminus \Omega_h$ zu setzen.

Wir wählen als Ansatzräume:

$$U_h = \{u_h \in L^\infty(\Omega) : u_h \in \mathcal{P}_1(T) \quad \forall T \in T_h \text{ und } u_h = \Pi_{[a,b]}(0) \text{ auf } \bar{\Omega} \setminus \Omega_h\},$$

$$U_h^{\text{ad}} = U_h \cap U_{\text{ad}},$$

$$V_h = \{y_h \in C(\bar{\Omega}) : y_h \in \mathcal{P}_1(T) \quad \forall T \in T_h \text{ und } y_h = 0 \text{ auf } \bar{\Omega} \setminus \Omega_h\}$$

wobei $\mathcal{P}_1$ den Raum der Polynome mit Grad kleiner oder gleich 1 bezeichnet.

2.1 Konvergenzeigenschaften

In diesem Abschnitt beschäftigen wir uns mit der Fehleranalyse der numerischen Näherungen unseres Optimierungsproblems. Unter Gültigkeit von (A1) und (A2) erhalten wir eine lineare Konvergenzordnung. Durch Forderung einer weiteren Voraussetzung, die in vielen praktischen Fällen erfüllt ist, erreichen wir eine Konvergenzordnung von $h^{\frac{3}{2}}$. Im nächsten Kapitel sehen wir dann an einem Testbeispiel, dass diese Konvergenzordnung tatsächlich erreicht wird.

Zur Herleitung der Fehlerabschätzungen betrachten wir nicht gleich die vollständige Diskretisierung. Zuerst wird nur die Steuerung u diskretisiert und der Zustand bleibt kontinuierlich. Im nächsten Schritt diskretisieren wir dann auch den Zustand y sowie den adjungierten Zustand p. Damit erreichen wir eine bessere Übersichtlichkeit.

2.1.1 Diskretisierung der Steuerung

Wir arbeiten hier auf der Menge der stückweise linearen Funktionen:

$$U_{\text{ad}}^h = \{u \in U_{\text{ad}} : u \in \mathcal{P}_1(T_i) \quad \forall T_i \in T_h \ \wedge \ u = \Pi_{[a,b]}(0) \text{ auf } \bar{\Omega} \setminus \Omega_h\}$$

Wir definieren jetzt eine Hilfsfunktion $v \in U_{\text{ad}}^h$ auf dem Dreieck T_i durch:

$$v|_{T_i} = \begin{cases} u_a & \text{falls} \quad \min_{x \in T_i} \bar{u}(x) = a \\ u_b & \text{falls} \quad \max_{x \in T_i} \bar{u}(x) = b \\ \bar{u}|_{T_i} & \text{sonst} \end{cases} \tag{2.1}$$

Dabei wählen wir die Gitterweite h genügend klein, so dass auf einem Dreieck nicht gleichzeitig $\bar{u} = a$ und $\bar{u} = b$ gelten kann.

Lemma 2.1. *Sei $\bar{p}$ der optimale adjungierte Zustand. Dann gilt für alle genügend kleine h und alle $u \in U_{ad}$*

$$(\bar{p} + \lambda\bar{u}, u - v)_{L^2(\Omega)} \geq 0 \tag{2.2}$$

Beweis. Wir betrachten ein beliebiges Dreieck T_i und zeigen, dass die Ungleichung punktweise fast überall gilt. Wir wählen also ein $x \in T_i$ und zeigen: $(\bar{p}(x)+\lambda\bar{u}(x))(u(x)-v(x)) \geq 0$ auf T_i.

1. Fall: Es existiert ein $\hat{x} \in T_i$ mit $\bar{u}(\hat{x}) = a$, dann gilt auf dem ganzen Dreieck: $\bar{u}(x) < b$. Dann kann die Variationsungleichung nur dann für alle $u \in U_{\text{ad}}$ (also auch für $u \equiv b$) gelten, wenn $\bar{p}(x)+\lambda\bar{u}(x) \geq 0$. Da aber in diesem Fall $v = a$ auf dem ganzen Dreieck T_i gilt, ist auch $(\bar{p}(x) + \lambda\bar{u}(x))(u(x) - v(x)) \geq 0$ für alle $u \in U_{\text{ad}}$.

2. Fall: Es existiert ein $\hat{x} \in T_i$ mit $\bar{u}(\hat{x}) = b$. Dieser Fall kann analog dem ersten Fall behandelt werden. Hier muss aufgrund der Variationsungleichung $\bar{p}(x) + \lambda\bar{u}(x) \leq 0$ gelten und wegen $u(x) - v(x) \leq 0$ ist die Ungleichung erfüllt.

3. Fall: Auf dem Dreieck T_i gilt: $a < \bar{u} < b$, dann ist $\bar{p}(x) + \lambda\bar{u}(x) = 0$ und die Ungleichung trivial.

□

Damit können wir die grundlegende Aussage für die weitere Konvergenzuntersuchung formulieren.

Lemma 2.2. *Bezeichne $\bar{u} \in U_{ad}$ die optimale Steuerung, $\bar{u}_h \in U_{ad}^h$ die optimale diskretisierte Steuerung und $v \in U_{ad}^h$ die durch (2.1) definierte Hilfsfunktion. Dann gilt:*

$$||\bar{u} - \bar{u}_h||_{L^2(\Omega)} \leq c||\bar{u} - v||_{L^2(\Omega)} \tag{2.3}$$

Als Beweis verweisen wir auf [4] Lemma 2.1 und die nachfolgende Diskussion.

Im weiteren wollen wir also eine Fehlerabschätzung für $||\bar{u} - v||_{L^2(\Omega)}$ entwickeln. Dazu benötigen wir noch eine Fehlerabschätzung für die Interpolation, die bei quasiuniformen Triangulierungen Gültigkeit hat.

Definition 2.1. *Eine Familie von Zerlegungen $\{T_h\}$ heißt quasiuniform, wenn es eine Zahl $\sigma > 0$ gibt, so dass jedes T einen Kreis vom Durchmesser $\sigma(T)$ mit*

$$\sigma(T) \geq \frac{\rho(T)}{\sigma}$$

enthält, wobei $\rho(T)$ der Durchmesser von T ist.

Aufgrund von (A1) haben wir es hier mit einer quasiuniformen Triangulierung zu tun und erhalten folgende Abschätzung.

Lemma 2.3. *Sei $t \geq 2$ und T_h eine quasiuniforme Triangulierung von Ω. Dann gilt für die Interpolation durch stückweise Polynome vom Grad $t-1$:*

$$\sum_{T_i \in T_h} ||u - I_h u||^2_{L^2(T_i)} \leq c \cdot h^4 |u|^2_{H^2(\Omega)} \quad \forall u \in H^2(\Omega) \tag{2.4}$$

Dabei bezeichnet $|\cdot|_{H^2(\Omega)}$ die H^2-Seminorm.

Den Beweis findet man z.B. in [8].

Wir zerlegen nun die Menge der Dreiecke in folgende drei Teile:

1. I_1 bezeichne die Menge aller Dreiecke, auf denen die optimale Steuerung identisch gleich einer der beiden Schranken ist:

$$I_1 := \{T_i \in T_h : \bar{u}(x) = a \, \forall x \in T_i \ \vee \ \bar{u}(x) = b \, \forall x \in T_i\}$$

2. I_2 beinhaltet alle Dreiecke, auf denen $\bar{u}$ inaktiv ist:

$$I_2 := \{T_i \in T_h : a < \bar{u}(x) < b \, \forall x \in T_i\}$$

3. in der Menge I_3 liegen alle restlichen Dreiecke. Auf diesen Dreiecken wird also eine Schranke erreicht, aber die Steuerung ist nicht konstant:

$$I_3 := T_h \setminus (I_1 \cup I_2)$$

Offensichtlich sind diese drei Mengen disjunkt. Wir betrachten nun die gesuchte Fehlerabschätzung auf diesen Mengen.

$$\begin{aligned}
||\bar{u} - v||^2_{L^2(\Omega)} &= \int_\Omega (\bar{u}(x) - v(x))^2 \mathrm{d}x = \sum_{T_i \in T_h} \int_{T_i} (\bar{u}(x) - v(x))^2 \mathrm{d}x \\
&= \sum_{T_i \in I_1} \int_{T_i} (\bar{u}(x) - v(x))^2 \mathrm{d}x + \sum_{T_j \in I_2} \int_{T_j} (\bar{u}(x) - v(x))^2 \mathrm{d}x \\
&+ \sum_{T_k \in I_3} \int_{T_k} (\bar{u}(x) - v(x))^2 \mathrm{d}x + \int_{\Omega \setminus \Omega_h} (\bar{u}(x) - \Pi_{[a,b]}(0))^2 \mathrm{d}x \\
&= ||\bar{u} - v||^2_{L^2(I_1)} + ||\bar{u} - v||^2_{L^2(I_2)} + ||\bar{u} - v||^2_{L^2(I_3)} + \int_{\Omega \setminus \Omega_h} (\bar{u}(x) - \Pi_{[a,b]}(0))^2 \mathrm{d}x
\end{aligned}$$

Nach Definition der Zerlegung ist

$$||\bar{u} - v||_{L^2(I_1)} = 0.$$

Auf I_2 ist v gerade die lineare Interpolante von $\bar{u}$ und mithilfe von Lemma 2.3 erhält man:

$$||\bar{u} - v||^2_{L^2(I_2)} \leq c \cdot h^4 |\bar{u}|^2_{H^2(\Omega)}$$

Dabei bleibt also nur noch die Fehlerabschätzung auf I_3. Wegen Lemma 1.6 ist $u \in C^{0,1}(\bar{\Omega})$ und damit gilt folgende Lipschitz-Bedingung:

$$|\bar{u}(x_1) - \bar{u}(x_2)| \leq L|x_1 - x_2| \leq Lh \quad \forall x_1, x_2 \in T_i$$

Nach Definition von v gilt auf allen Dreiecken T_i aus I_3: $v \equiv a$ bzw. $v \equiv b$ und es existiert mindestens ein $\hat{x}$ mit $\bar{u}(\hat{x}) = a$ bzw. $\bar{u}(\hat{x}) = b$. Wählen wir nun in obiger Lipschitz-Bedingung $x_1 = x$ beliebig und $x_2 = \hat{x}$, so erhalten wir

$$|\bar{u}(x) - v(x)| \leq Lh \quad \forall x \in T_i$$

Damit ergibt sich:

$$\begin{aligned} ||\bar{u} - v||^2_{L^2(I_3)} &= \sum_{T_i \in I_3} \int_{T_i} (\bar{u}(x) - v(x))^2 \mathrm{d}x \\ &\leq \sum_{T_i \in I_3} \int_{T_i} L^2 h^2 \mathrm{d}x = L^2 h^2 |I_3| \end{aligned}$$

Mit dem gleichen Argument schätzt man den letzten Anteil ab. Hier zahlt sich unsere Definition der Werte auf $\Omega \setminus \Omega_h$ aus. Aufgrund der Definition und der Lipschitzstetigkeit von $\bar{u}$ finden wir:

$$\begin{aligned} \int_{\Omega \setminus \Omega_h} (\bar{u}(x) - \Pi_{[a,b]}(0))^2 \mathrm{d}x &\leq \int_{\Omega \setminus \Omega_h} L^2 h^2 \mathrm{d}x \\ &= L^2 h^2 |\Omega \setminus \Omega_h| \leq \kappa h^4 \quad \text{wegen (A2)} \end{aligned}$$

Fasst man diese Ergebnisse alle zusammen, ergibt sich:

$$||\bar{u} - v||^2_{L^2(\Omega)} \leq C(h^4 + h^2 |I_3|) \tag{2.5}$$

Ohne weitere Voraussetzungen ergibt sich daraus eine Konvergenzordnung von h. Praktische Rechnungen ergeben aber, dass die Approximation in den meisten Fällen besser ist. Das ist darin begründet, dass für die Testbeispiele eine weitere Voraussetzung erfüllt ist, die man allgemein nicht so einfach verifizieren kann.

(A3)

$$|I_3| \leq ch \tag{2.6}$$

Die zusätzliche Forderung ist erfüllt, wenn die Grenze zwischen aktiven Anteilen (optimale Steuerung erfüllt nicht die Beschränkung) und inaktiven Anteilen (optimale Steuerung liegt in U_{ad}) der optimalen Steuerung durch endlich viele Kurven beschrieben werden kann.

Aus (2.5) und (2.6) folgt sofort

$$||\bar{u} - v||^2_{L^2(\Omega)} \leq Ch^{\frac{3}{2}}$$

Damit gilt (wegen Lemma 2.3) die Fehlerabschätzung bei diskretisierter Steuerung

$$||\bar{u} - \bar{u}_h||_{L^2(\Omega)} \leq ch^{\frac{3}{2}} \tag{2.7}$$

Das Ganze fassen wir nochmal zusammen.

Lemma 2.4. *Bezeichne $\bar{u} \in U_{ad}$ die eindeutige Lösung des Optimalsteuerproblems*

$$\min_{u \in U_{ad}} \frac{1}{2}||Su - y_d||^2_{L^2(\Omega)} + \frac{\lambda}{2}||u||^2_{L^2(\Omega)}$$

und $\bar{u}_h \in U^h_{ad}$ die eindeutige Lösung des Steuerung-diskretisierten Optimalsteuerproblems

$$\min_{u \in U^h_{ad}} \frac{1}{2}||Su - y_d||^2_{L^2(\Omega)} + \frac{\lambda}{2}||u||^2_{L^2(\Omega)}$$

mit zugehöriger Zustandsgleichung. Sind die Voraussetzungen (A1) und (A2) erfüllt, so gilt:

$$||\bar{u} - \bar{u}_h||_{L^2(\Omega)} \leq c_1 h$$

Gilt zusätzlich (A3), dann erreicht man eine höhere Konvergenzordnung:

$$||\bar{u} - \bar{u}_h||_{L^2(\Omega)} \leq c_2 h^{\frac{3}{2}}$$

2.1.2 Vollständige Diskretisierung

Nachdem wir im vorangegangenen Abschnitt nur die Steuerung diskretisiert haben, betrachten wir nun das vollständig diskretisierte Optimalsteuerungsproblem. Dabei bezeichne $\bar{u}_h$ die Lösung des Optimalsteuerungsproblem mit diskretisierter Steuerung und $\bar{u}^h_h$ die Lösung des vollständig diskretisierten Optimalsteuerungsproblems. Analog bezeichnen $y, \bar{p}$ die kontinuierlichen Zustände und $y_h, \bar{p}_h$ die diskretisierten Zustände.

Dabei benötigen wir noch ein weiteres Resultat der FEM-Approximation: das Aubin-Nitsche Lemma.

Lemma 2.5 (Aubin-Nitsche). *Es seien (A1) und (A2) erfüllt und $u \in L^2(\Omega)$. Dann gilt*

$$||y(u) - y_h(u)||_{L^2(\Omega)} \leq ch^2||u||_{L^2(\Omega)} \tag{2.8}$$

$$||p(u) - p_h(u)||_{L^2(\Omega)} \leq ch^2 \left(||u||_{L^2(\Omega)} + ||y_d||_{L^2(\Omega)}\right). \tag{2.9}$$

Formuliert man für das semidiskretisierte und das vollständig diskretisierte Optimalsteuerungsproblem die notwendige und hinreichende Optimalitätsbedingung, so erhält man:

$$(\bar{p}(\bar{u}_h) + \lambda\bar{u}_h, u - \bar{u}_h)_{L^2(\Omega)} \geq 0 \quad \forall u \in U_{\text{ad}}^h \tag{2.10}$$

und

$$\left(\bar{p}_h(\bar{u}_h^h) + \lambda\bar{u}_h^h, u - \bar{u}_h^h\right)_{L^2(\Omega)} \geq 0 \quad \forall u \in U_{\text{ad}}^h \tag{2.11}$$

In Ungleichung (2.10) setzen wir $u = \bar{u}_h^h$ und Ungleichung (2.11) testen wir mit $u = \bar{u}_h$. Wir erhalten

$$\left(\bar{p}(\bar{u}_h) + \lambda\bar{u}_h, \bar{u}_h^h - \bar{u}_h\right)_{L^2(\Omega)} \geq 0 \tag{2.12}$$

und

$$\left(\bar{p}_h(\bar{u}_h^h) + \lambda\bar{u}_h^h, \bar{u}_h - \bar{u}_h^h\right)_{L^2(\Omega)} \geq 0 \tag{2.13}$$

Addiert man beide Ungleichungen, so erhält man nach einigen kleinen Umformungen

$$\begin{aligned}
\lambda||\bar{u}_h^h - \bar{u}_h||_{L^2(\Omega)}^2 &\leq \left(\bar{p}(\bar{u}_h) - \bar{p}_h(\bar{u}_h^h), \bar{u}_h^h - \bar{u}_h\right)_{L^2(\Omega)} \\
&= \left(\bar{p}(\bar{u}_h) - \bar{p}_h(\bar{u}_h), \bar{u}_h^h - \bar{u}_h\right)_{L^2(\Omega)} + \left(\bar{p}_h(\bar{u}_h) - \bar{p}_h(\bar{u}_h^h), \bar{u}_h^h - \bar{u}_h\right)_{L^2(\Omega)}
\end{aligned}$$

Den ersten Summanden schätzen wir mit Hilfe der Cauchy-Schwarz-Ungleichung und dem Aubin-Nitsche Lemma ab.

$$\begin{aligned}
\left(\bar{p}(\bar{u}_h) - \bar{p}_h(\bar{u}_h), \bar{u}_h^h - \bar{u}_h\right)_{L^2(\Omega)} &\leq ||\bar{p}(\bar{u}_h) - \bar{p}_h(\bar{u}_h)||_{L^2(\Omega)}||\bar{u}_h^h - \bar{u}_h||_{L^2(\Omega)} \\
&\leq ch^2 \left(||\bar{u}_h||_{L^2(\Omega)} + ||y_d||_{L^2(\Omega)}\right)||\bar{u}_h^h - \bar{u}_h||_{L^2(\Omega)}
\end{aligned}$$

Für den zweiten Summanden ergibt sich

$$\begin{aligned}
\left(\bar{p}_h(\bar{u}_h) - \bar{p}_h(\bar{u}_h^h), \bar{u}_h^h - \bar{u}_h\right)_{L^2(\Omega)} &= \left(y_h(\bar{u}_h) - y_h(\bar{u}_h^h), y_h(\bar{u}_h^h) - y_h(\bar{u}_h)\right)_{L^2(\Omega)} \\
&= -||y_h(\bar{u}_h) - y_h(\bar{u}_h^h)||^2 \leq 0
\end{aligned}$$

Insgesamt erhält man also

$$||\bar{u}_h^h - \bar{u}_h||_{L^2(\Omega)} \leq ch^2 \tag{2.14}$$

Fasst man nun beide Abschätzungen (2.7),(2.14) zusammen, so erhält man mit Hilfe der Dreiecksungleichung

$$\begin{aligned}
||\bar{u} - \bar{u}_h^h||_{L^2(\Omega)} &\leq ||\bar{u} - \bar{u}_h||_{L^2(\Omega)} + ||\bar{u}_h - \bar{u}_h^h||_{L^2(\Omega)} \\
&\leq c_1 h^{\frac{3}{2}} + c_2 h^2 \\
&\leq ch^{\frac{3}{2}}
\end{aligned}$$

Folgerung 2.1. *Die Konvergenzordnung der Steuerung überträgt sich auch auf den Zustand und den adjungierten Zustand, d.h.*

$$\begin{aligned} ||\bar{y}(\bar{u}) - \bar{y}_h(\bar{u}_h^h)||_{L^2(\Omega)} &\leq ch^{\frac{3}{2}} \\ ||\bar{p}(\bar{u}) - \bar{p}_h(\bar{u}_h^h)||_{L^2(\Omega)} &\leq ch^{\frac{3}{2}} \end{aligned}$$

Beweis. Mithilfe des Aubin-Nitsche Lemmas erhält man

$$||\bar{y}(\bar{u}_h^h) - \bar{y}_h(\bar{u}_h^h)||_{L^2(\Omega)} \leq ch^2$$

Die Stetigkeit der Lösung der Zustandsgleichung bezüglich der rechten Seite liefert zusammen mit der Abschätzung des Fehlers in der Steuerung:

$$||\bar{y}(\bar{u}) - \bar{y}(\bar{u}_h^h)||_{L^2(\Omega)} = ||S(\bar{u} - \bar{u}_h^h)||_{L^2(\Omega)} \leq c||\bar{u} - \bar{u}_h^h||_{L^2(\Omega)} \leq ch^{\frac{3}{2}}$$

Daraus folgt mit der Dreiecksungleichung:

$$\begin{aligned} ||\bar{y}(\bar{u}) - \bar{y}_h(\bar{u}_h^h)||_{L^2(\Omega)} &\leq ||\bar{y}(\bar{u}) - \bar{y}(\bar{u}_h^h)||_{L^2(\Omega)} + ||\bar{y}(\bar{u}_h^h) - \bar{y}_h(\bar{u}_h^h)||_{L^2(\Omega)} \\ &\leq ch^{\frac{3}{2}} \end{aligned}$$

Ganz analog verläuft der Nachweis für den adjungierten Zustand. □

Kapitel 3

Numerische Berechnung

Inhalt dieses Kapitels ist die praktische Umsetzung der Diskretisierung und das Aufstellen eines Algorithmus zur Lösung unseres Optimalsteuerungsproblems. Dabei stützen wir uns auf die Dokumentation von Meyer [2], die das gleiche Optimalsteuerungsproblem behandelt, allerdings mit einer anderen Diskretisierung. Die daraus gewonnenen Ergebnisse dienen uns später auch als Vergleich.

Das Gebiet Ω ist hier das Einheitsquadrat. Für polygonale Gebiete sind unsere Regularitätsvoraussetzungen für die Lösungstheorie der partiellen Differentialgleichungen nicht erfüllt. Es existieren aber Resultate, ähnlich dem Lemma 1.6, die zeigen, dass auch für polygonale Gebiete die eindeutige Lösung im $W^{2,p}(\Omega)$ liegt, allerdings mit einem maximalen p, welches abhängig von den Innenwinkeln des Polygons ist. Dazu verweisen wir auf [5] Kapitel 4. Wir benötigen hier jedoch nur, dass die Lösung im $W^{2,p}(\Omega)$ liegt für ein $p > 2$.

Polygonale Gebiete haben den Vorteil, dass $\Omega = \Omega_h$ gilt. Damit ändert sich Ω_h nicht, wenn man die Gitterweite h ändert.

Wir wählen eine regelmäßige Triangulierung des Einheitsquadrats, indem wir jede Seite in N Intervalle unterteilen und die dabei entstandenen Quadrate nochmals halbieren (Abbildung 3.1).

Wie weiter oben schon beschrieben, wird der Zustand durch stückweise lineare Funktionen diskretisiert, die global stetig sind. Das heißt wir definieren uns für jeden inneren Punkt des Gitters die bekannten Hütchenfunktionen, mit folgenden Eigenschaften:

- auf jedem Teildreick ist die Ansatzfunktion linear,
- die Ansatzfunktion nimmt den Wert 1 in "ihrem" Punkt an und verschwindet in allen anderen Gitterpunkten

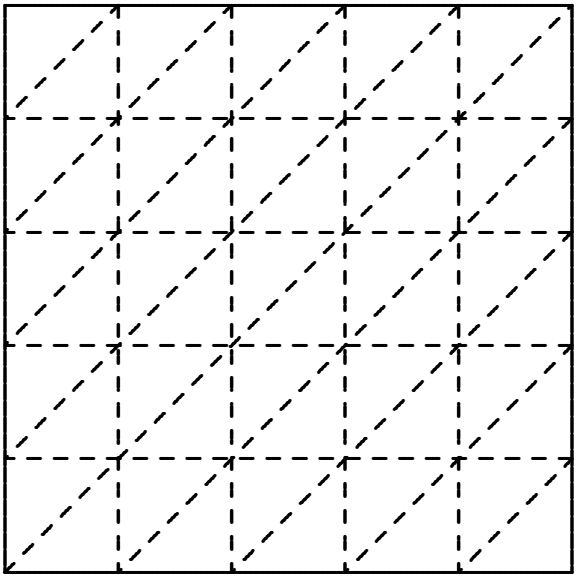

Abbildung 3.1: Triangulierung

Diesen Ansatz weiten wir auf die Randpunkte aus, schneiden die Hütchenfunktionen dann aber am Rand des Einheitsquadrats einfach ab. Da wir hier Dirichlet-Randbedingungen haben, wäre dies eigentlich nicht nötig, ermöglicht aber eine spätere Nutzung des Programms auch für andere Randbedingungen.

Die Steuerung wird ebenfalls durch stückweise lineare Funktionen diskretisiert. Hierbei sind allerdings Unstetigkeiten an den Dreieckskanten erlaubt. Dazu definieren wir auf jedem Teildreieck drei lineare Basisfunktionen, die jeweils in "ihrem" Eckpunkt den Wert 1 annehmen und in den beiden anderen verschwinden. Das ermöglicht die erlaubten Unstetigkeiten an den Dreieckskanten.

Damit erhalten wir also folgende Diskretisierungen:

$$y = \sum_{i=1}^{n_P} y_i \phi_i, \qquad u = \sum_{j=1}^{3n_T} u_j \varphi_j$$

Hierbei bezeichnen n_P die Anzahl der Punkte des Gitters und n_T die Anzahl der Dreiecke.

Die Variationsformulierung der partiellen Differentialgleichung lautet:

$$(\nabla y, \nabla v)_{L^2(\Omega)} = (u, v)_{L^2(\Omega)} \quad \forall v \in H_0^1(\Omega) \tag{3.1}$$

Nun setzen wir die Diskretisierungen für y und u ein und wählen als Testfunktionen gerade die Ansatzfunktionen ϕ_k:

$$\sum_{i=1}^{n_P} (\nabla \phi_i, \nabla \phi_k)_{L^2(\Omega)} y_i = \sum_{j=1}^{3n_T} (\varphi_j, \phi_k)_{L^2(\Omega)} u_j \tag{3.2}$$

Mithilfe folgender Matrizen: $A \in \mathbb{R}^{n_p \times n_p}$ mit Einträgen $A_{ik} = (\nabla\phi_i, \nabla\phi_k)_{L^2(\Omega)}$ (Steifigkeitsmatrix) und $R \in \mathbb{R}^{n_P \times 3n_T}$ mit Einträgen $R_{jk} = (\varphi_j, \phi_k)_{L^2(\Omega)}$ (Transformationsmatrix $R : U \to Y$) kann die diskretisierte Variationsformulierung als Matrixgleichung schreiben:

$$Ay = Ru \quad \Longrightarrow y = A^{-1}Ru = Su \tag{3.3}$$

3.1 Notwendige Optimalitätsbedingung

Jetzt gehen wir für einen Moment davon aus, dass wir keine Beschränkung an die zulässige Menge der Steuerungen haben. Die hier hergeleiteten Formeln benötigen wir später für die Behandlung unserer Optimalsteuerung mit Beschränkung.

Die notwendige Optimalitätsbedingung im Fall ohne Beschränkung ist folgende Variationsgleichung:

$$f'(u)h = (Sh, Su - y_d)_{L^2(\Omega)} + \lambda(h, u)_{L^2(\Omega)} = 0 \tag{3.4}$$

Diese diskretisieren wir mithilfe:

$$Su = y = \sum_{i=1}^{n_P} y_i\phi_i, \quad y_d = \sum_{i=1}^{n_P} y_{d_i}\phi_i, \quad Sh = \eta = \sum_{i=1}^{n_P} \eta_i\phi_i, \quad h = \sum_{i=1}^{3n_T} h_i\varphi_i.$$

In (3.4) eingesetzt:

$$\sum_{i,j=1}^{n_P} (\phi_i, \phi_j)_{L^2(\Omega)}\eta_i(y_j - y_{d_j}) + \lambda \sum_{i,j=1}^{3n_T} (\varphi_i, \varphi_j)_{L^2(\Omega)} h_i u_j = 0 \tag{3.5}$$

Um diese Gleichung in Matrizenform aufzuschreiben, definieren wir die Massenmatrizen $M \in \mathbb{R}^{n_P \times n_P}$ mit den Einträgen $M_{ij} = (\phi_i, \phi_j)_{L^2(\Omega)}$ und $\tilde{M} \in \mathbb{R}^{3n_T \times 3n_T}$ mit den Einträgen $\tilde{M}_{ij} = (\varphi_i, \varphi_j)_{L^2(\Omega)}$.

Folglich erhalten wir:

$$\eta^T M(y - y_d) + \lambda h^T \tilde{M} u = 0 \tag{3.6}$$

Einsetzen von $y = Su = A^{-1}Ru$ und analog $\eta = Sh = A^{-1}Rh$ liefert:

$$(A^{-1}Rh)^T M(A^{-1}Ru - y_d) + \lambda h^T \tilde{M} u = 0$$

und wegen $A = A^T$:

$$h^T(R^T A^{-1} M A^{-1} R u - R^T A^{-1} M y_d + \lambda \tilde{M} u) = 0 \quad \forall h \in \mathbb{R}^{3n_T} \tag{3.7}$$

Damit ergibt sich folgende Optimalitätsbedingung für die diskrete Steuerung:

$$(\lambda\tilde{M} + R^T A^{-1} M A^{-1} R)u = R^T A^{-1} M y_d \tag{3.8}$$

in Kurzform:

$$Bu = b \tag{3.9}$$

mit:

$$B := \lambda\tilde{M} + R^T A^{-1} M A^{-1} R, \qquad b := R^T A^{-1} M y_d \tag{3.10}$$

B ist symmetrisch und positiv definit. Da A, M und $\tilde{M}$ symmetrisch sind, gilt

$$B^T = (\lambda\tilde{M} + R^T A^{-1} M A^{-1} R)^T = \lambda\tilde{M} + R^T A^{-1} M A^{-1} R = B$$

Offensichtlich sind M und $\tilde{M}$ positiv definit, damit aber auch B, denn

$$x^T (R^T A^{-1} M A^{-1} R) x = (A^{-1} R x)^T M (A^{-1} R x) > 0$$

Daher benutzen wir das cg-Verfahren zum Lösen des linearen Gleichungssystems (3.9). Dieses hat zudem den Vorteil, dass die Matrix B nicht explizit berechnet werden muss, sondern nur die Anwendung auf ein Element.

3.2 cg-Verfahren für $Bu = b$

Wir geben hier nur den direkt implementierten Code an:

$r = b - Bu$
$g = r$
while(!Abbruchkriterium)
    $d = Bg$
    $\gamma = g^T d$
    $\alpha = r^T r$
    $u = u + \frac{\alpha}{\gamma} g$
    $r = r - \frac{\alpha}{\gamma}$
    $\beta = \frac{r^T r}{\alpha}$
    $g = r - \beta g$
end

3.3 Adjungierter Zustand und Abbruchbedingung

Ausgehend von der Zustandsgleichung und adjungierten Zustandsgleichung schreiben wir die zugehörigen Variationsformulierungen auf:

$$\int_\Omega \nabla y \cdot \nabla v \mathrm{d}x = \int_\Omega uv \mathrm{d}x \quad \forall v \in H_0^1(\Omega) \tag{3.11}$$

$$\int_\Omega \nabla p \cdot \nabla v \mathrm{d}x = \int_\Omega (y - y_d) v \mathrm{d}x, \quad \forall v \in H_0^1(\Omega) \tag{3.12}$$

Nun setzen wir in die Variationsformulierung der Zustandsgleichung (3.11) als Testfunktion $v = p$ und in die Variationsformulierung der adjungierten Gleichung (3.12) als Testfunktion $v = y$ ein. Dann stimmen die linken Seiten überein und damit auch die rechten.

$$\int_\Omega up \mathrm{d}x = \int_\Omega (y - y_d) y \mathrm{d}x \tag{3.13}$$

bzw. mit Hilfe des Skalarproduktes:

$$(u, p)_{L^2(\Omega)} = (y - y_d, y)_{L^2(\Omega)}. \tag{3.14}$$

Diesen Vorgang bezeichnet man auch als Methode der Transposition. Diskretisierung der Gleichung (mit $p = \sum_{i=1}^{3n_T} p_i \varphi_i$) liefert:

$$\sum_{i,j=1}^{3n_T} (\varphi_i, \varphi_j)_{L^2(\Omega)} u_i p_j - \sum_{i,j=1}^{n_P} (\phi_i, \phi_j)_{L^2(\Omega)} y_i (y_j - y_{d_j}) = 0 \tag{3.15}$$

und mit $y = A^{-1}Ru$ und $A = A^T$:

$$u^T \tilde{M} p - (A^{-1}Ru)^T M (y - y_d) = 0 \quad \forall u \in \mathbb{R}^{3n_T} \tag{3.16}$$

woraus sofort folgt:

$$\tilde{M} p = R^T A^{-1} M (y - y_d) = S^T M (y - y_d) \tag{3.17}$$

Für das Residuum r_k im k-ten Schritt des cg-Verfahrens gilt:

$$Bu_k - b = (\lambda \tilde{M} + R^T A^{-1} M A^{-1} R) u_k - R^T A^{-1} M y_d = r_k. \tag{3.18}$$

Daraus folgt (unter Vernachlässigung des Index k):

$$\lambda \tilde{M} u + S^T M (Su - y_d) = r \tag{3.19}$$

und wegen Gleichung (3.17) ergibt sich:

$$\lambda u + p = \tilde{M}^{-1} r. \tag{3.20}$$

Aus der notwendigen Optimalitätsbedingung (Minimumprinzip) folgt im Fall der Steuerung ohne Beschränkung:

$$u = -\frac{1}{\lambda} p. \tag{3.21}$$

Das heißt, dass die L^2-Norm von $\lambda u + p$ minimiert werden muss. Einsetzen der Diskretisierungen für u und p liefert zusammen mit Gleichung (3.20) und $\tilde{M}^T = \tilde{M}$:

$$\begin{aligned} ||\lambda u + p||_{L^2(\Omega)} &= (\lambda u + p)^T \tilde{M} (\lambda u + p) \\ &= (\tilde{M}^{-1} r)^T \tilde{M} \tilde{M}^{-1} r \\ &= r^T \tilde{M}^{-1} r =: e \end{aligned}$$

Somit erhalten wir als relative Abbruchbedingung:

$$\frac{e}{||u||_{\mathbb{R}^{3n_T}}} \leq \epsilon \tag{3.22}$$

3.4 Aktive Mengen Strategie

In vielen Fällen gibt es Beschränkungen für die Steuerungen. Typisch dabei sind punktweise Beschränkungen, so dass die Menge der zulässigen Steuerungen definiert wird durch:

$$U_{\text{ad}} = \{u(x) \in L^2(\Omega) : a \leq u(x) \leq b \quad \text{f.ü. in } \Omega\}$$

Bei unserem Problem sind a und b konstant.

Grundidee der Aktiven-Mengen-Strategie ist es, die Optimierung mit Steuerbeschränkung durch eine Folge von Optimierungen ohne Beschränkung zu ersetzen. Dazu zerlegt man die Steuerung u in einen aktiven und inaktiven Anteil:

$$u = u_A + u_I. \tag{3.23}$$

Dabei sind u_I und u_A folgendermaßen definiert:

$$u_A(x) = \begin{cases} a & \text{falls} \quad x \in A_- := \{x \in \Omega : u(x) < a\} \\ b & \text{falls} \quad x \in A_+ := \{x \in \Omega : u(x) > b\} \\ 0 & \text{sonst} \end{cases} \tag{3.24}$$

$$u_I(x) = \begin{cases} u(x) & \text{falls} \quad x \in I := \{x \in \Omega : a \leq u(x) \leq b\} \\ 0 & \text{sonst} \end{cases} \tag{3.25}$$

Der inaktive Teil der Steuerung erfüllt die punktweise Beschränkung, so dass sich für u_I ein freies Optimierungsproblem ergibt.

3.4.1 Die Variationsungleichung

Bei der Optimierung mit Beschränkung an die Steuerung erhalten wir keine Gleichung, sondern die entsprechende Variationsungleichung:

$$f'(u)h = (Sh, Su - y_d)_{L^2(\Omega)} + \lambda(h, u)_{L^2(\Omega)} \geq 0 \quad \forall h = u - \bar{u}, u \in U_{\text{ad}}. \tag{3.26}$$

Setzt man in diese Ungleichung unseren Ansatz für die Steuerung (3.23) ein, ergibt sich:

$$(Sh, Su_I - y_d)_{L^2(\Omega)} + \lambda(h, u_I)_{L^2(\Omega)} \geq -(Sh, Su_A)_{L^2(\Omega)} - \lambda(h, u_A)_{L^2(\Omega)} \tag{3.27}$$

Dies kann man als Variationsungleichung für die inaktive Steuerung auffassen. Da diese aber keiner Beschränkung unterliegt, kann diese Ungleichung bei gegebenen aktiven Anteil u_A in eine Variationsgleichung umgewandelt werden.

$$(Sh, Su_I - y_d)_{L^2(\Omega)} + \lambda(h, u_I)_{L^2(\Omega)} + (Sh, Su_A)_{L^2(\Omega)} + \lambda(h, u_A)_{L^2(\Omega)} = 0 \tag{3.28}$$

Hierbei ist der aktive Teil u_A fest. Nun setzen wir wiederum unsere Diskretisierungen ein, wobei sich für u_I und u_A ergeben:

$$u_I = \sum_{j=1}^{3n_T} u_{I_j}\varphi_j \qquad u_A = \sum_{j=1}^{3n_T} u_{A_j}\varphi_j,$$

wobei u_{I_j} und u_{A_j} analog zu (3.25) und (3.24) wie folgt definiert sind:

$$u_{I_j} = \begin{cases} u_j & \text{falls} \quad x_j \in I \\ 0 & \text{sonst} \end{cases} \tag{3.29}$$

$$u_{A_j} = \begin{cases} a & \text{falls} \quad x_j \in A_- \\ b & \text{falls} \quad x_j \in A_+ \\ 0 & \text{falls} \end{cases} \tag{3.30}$$

mit den diskreten Mengen:

$$\begin{aligned} A_- &= \{i \in \{1, 2, \ldots, 3n_T\} : u(x_i) < a\} \\ A_+ &= \{i \in \{1, 2, \ldots, 3n_T\} : u(x_i) > b\} \\ I &= \{i \in \{1, 2, \ldots, 3n_T\} : a \leq u(x_i) \leq b\} \end{aligned} \tag{3.31}$$

Die Rechnungen verlaufen ähnlich den zuvor beschriebenen und es ergibt sich:

$$h^T S^T M(Su_i - y_d) + \lambda h^T \tilde{M} u_I + h^T S^T MSu_A + \lambda h^T \tilde{M} u_A = 0 \quad \forall h \in \mathbb{R}^{3n_T} \tag{3.32}$$

mit einem gegebenen Vektor $u_A \in \mathbb{R}^{3n_T}$. Daraus erhält man ein lineares Gleichungssystem für den unbekannten Vektor u_I:

$$(\lambda\tilde{M} + S^T MS)u_I = S^T My_d - (\lambda M + S^T MS)u_A \tag{3.33}$$

Da die Koeffizientenmatrix B mit der Matrix aus dem Gleichungssystem (3.10) übereinstimmt, kann das cg-Verfahren völlig analog verwendet werden. Dabei ergibt sich als rechte Seite:

$$b = R^T A^{-1} My_d - (\lambda\tilde{M} + R^T A^{-1} MA^{-1} R)u_A. \tag{3.34}$$

3.4.2 Der Primal-Duale Algorithmus

Die Variationsungleichung unter Verwendung des adjungierten Operator S^* lautet:

$$f'(u)h = (h, S^*(y - y_d) + \lambda u)_{L^2(\Omega)} \geq 0 \quad \forall h = u - \bar{u}, u \in U_{\text{ad}}$$

Aus der adjungierten Zustandsgleichung ergibt sich $p = S^*(y - y_d)$ und damit

$$(h, p + \lambda u)_{L^2(\Omega)} \geq 0 \quad \forall h = u - \bar{u}, u \in U_{\text{ad}}.$$

Punktweise Auswertung dieser Ungleichung führt dann bezüglich der optimalen Steuerung $\bar{u}$ zu:

$$\bar{u} = \begin{cases} a & \text{falls} \quad \bar{p} + \lambda\bar{u} > 0 \\ b & \text{falls} \quad \bar{p} + \lambda\bar{u} < 0 \\ -\frac{\bar{p}}{\lambda} & \text{falls} \quad \bar{p} + \lambda\bar{u} = 0 \end{cases}$$

Die optimale Steuerung nimmt also die Werte a und b nur an, wenn $\bar{p} + \lambda\bar{u} \neq 0$ gilt. Diese Bedingung muss also auch von Komponenten des aktiven Vektors u_A erfüllt werden.

Das ganze führt zu folgendem Primal-Dual-Algorithmus:

1. Startwerte: $u_{I_i}^{(0)} = \frac{a+b}{2}$, $u_{A_i}^{(0)} = 0, i = 1, \ldots, 3n_T$ Entsprechend sind die diskreten Mengen: $I^{(0)} = \{1, 2, \ldots, 3n_T\}, A_-^{(0)} = A_+^{(0)} = \{\}$.

2. (a) bestimme über (3.34) die neue Inhomogenität $b^{(n)}$ in Abhängigkeit von $u_A^{(n)}$

   (b) berechne mittels des cg-Verfahrens über $Bu_I^{(n)} = b^{(n)}$ den neuen Vektor der inaktiven Steuerungen $u_I^{(n)}$ und damit $u^{(n)} = u_I^{(n)} + u_A^{(n)}$

   (c) berechne $y^{(n)} = Su^{(n)}$ und $p^{(n)} = S^*(y^{(n)} - y_d)$

   (d)
   - werte $u_{I_j}^{(n)} < a$ mit $j \in I^{(n-1)}$ aus und bilde $A_{\text{neu},-}^{(n)} := \{j \in I^{(n-1)} : u_{I_j} < a\}$
   - werte $u_{I_j}^{(n)} > b$ mit $j \in I^{(n-1)}$ aus und bilde $A_{\text{neu},+}^{(n)} := \{j \in I^{(n-1)} : u_{I_j} > b\}$

   bilde $I^{(n)}$ dann über

   $$I^{(n)} = I^{(n-1)} \setminus \left(A_{\text{neu},-}^{(n)} \cup A_{\text{neu},+}^{(n)}\right)$$

   (e)
   - werte $p_j^{(n)} + \lambda u_{A_j}^{(n)} < 0$ aus und bilde $I_{\text{neu},-}^{(n)} = \{j \in A_-^{(n-1)} : p_j^{(n)} + \lambda u_{A_j}^{(n)} < 0\}$
   - werte $p_j^{(n)} + \lambda u_{A_j}^{(n)} > 0$ aus und bilde $I_{\text{neu},+}^{(n)} = \{j \in A_+^{(n-1)} : p_j^{(n)} + \lambda u_{A_j}^{(n)} > 0\}$

   damit bestimme dann die neuen Mengen

   $$\begin{aligned} A_-^{(n)} &= \left(A_-^{(n-1)} \cup A_{\text{neu},-}\right) \setminus I_{\text{neu},-} \\ A_+^{(n)} &= \left(A_+^{(n-1)} \cup A_{\text{neu},+}\right) \setminus I_{\text{neu},+} \\ I^{(n)} &= I^{(n)} \cup I_{\text{neu},-}^{(n)} \cup I_{\text{neu},+}^{(n)} \end{aligned}$$

3. falls sich mindestens eine der Mengen I, A_-, A_+ in der letzten Iteration verändert haben, wiederhole Schritt 2

Kapitel 4

Beispiele

Um die Konvergenzaussagen testen zu können, behandeln wir zwei Beispiele, von denen die exakte Lösung bekannt ist. Wir nehmen hier die gleichen Testbeispiele wie in [1].

4.1 Beispiel 1 – Dirichlet-Randbedingung

Wir wählen in (1.2) $c \equiv 0$ und erhalten als Zustandsgleichung die Poisson-Gleichung mit homogenen Dirichlet-Randbedingungen:

$$\begin{aligned} -\Delta y &= u \quad \text{in } \Omega \\ y &= 0 \qquad \text{auf } \Gamma \end{aligned}$$

Wir definieren den optimalen Zustand (Abbildung 4.1):

$$\bar{y} = y_a - y_g$$

mit einem analytischen Teil $y_a = \sin(\pi x_1)\sin(\pi x_2)$ und einer weniger glatten Funktion y_g. Diese Funktion y_g ist definiert als Lösung von

$$\begin{aligned} -\Delta y_g &= g \qquad \text{in } \Omega \\ y_g &= 0 \qquad \text{auf } \Gamma. \end{aligned}$$

Dabei ist die Funktion g folgendermaßen definiert:

$$g(x_1, x_2) = \begin{cases} u_f(x_1, x_2) - a & \text{falls} \quad u_f(x_1, x_2) < a \\ 0 & \text{falls} \quad u_f(x_1, x_2) \in [a, b] \\ u_f(x_1, x_2) - b & \text{falls} \quad u_f(x_1, x_2) > b \end{cases}$$

mit $u_f(x_1, x_2) = 2\pi^2 \sin(\pi x_1)\sin(\pi x_2)$.

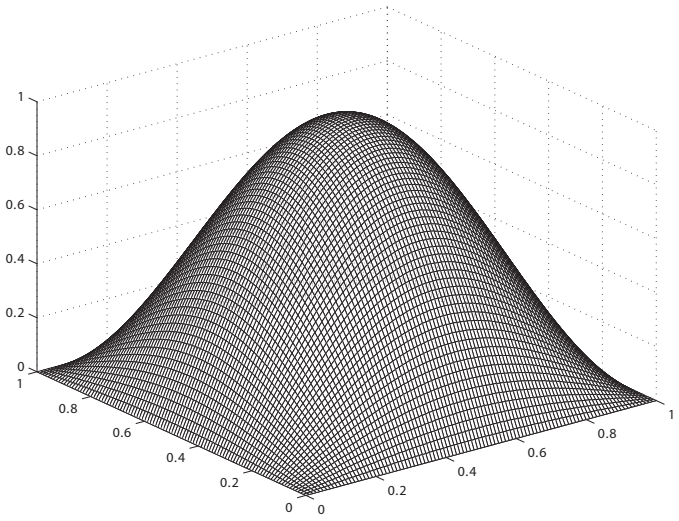

Abbildung 4.1: optimaler Zustand $\bar{y}$

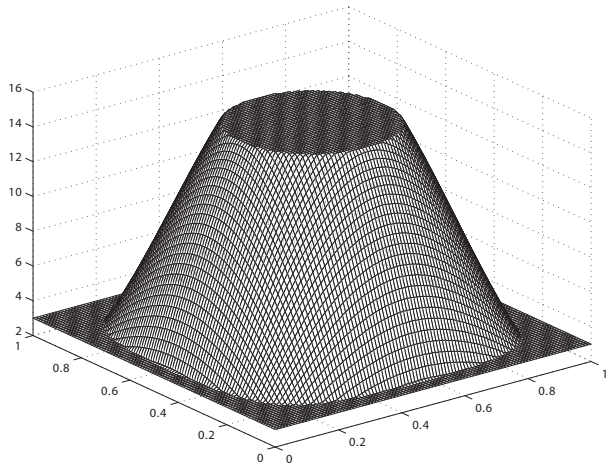

Abbildung 4.2: optimale Steuerung u

Aus der Zustandsgleichung ergibt sich die optimale Steuerung $\bar{u}$ (Abbildung 4.2):

$$\bar{u}(x_1, x_2) = \begin{cases} a & \text{falls} \quad u_f(x_1, x_2) < a \\ u_f(x_1, x_2) & \text{falls} \quad u_f(x_1, x_2) \in [a, b] \\ b & \text{falls} \quad u_f(x_1, x_2) > b \end{cases}$$

Für den optimalen, adjungierten Zustand gilt:

$$\bar{p}(x_1, x_2) = -2\pi^2 \lambda \sin(\pi x_1) \sin(\pi x_2).$$

Aus der adjungierten Zustandsgleichung ergibt sich dann der gewünschte Zustand y_d:

$$y_d(x_1, x_2) = \bar{y} + \Delta\bar{p} = y_a - y_g + 4\pi^4 \lambda \sin(\pi x_1) \sin(\pi x_2).$$

Offensichtlich ist $\bar{u} \in U_{\text{ad}}$. Mit

$$\Delta y_a = 2\pi^2 \sin(\pi x_1) \sin(\pi x_2)$$

ist $\bar{y} = y_a - y_g$ Lösung der Zustandsgleichung. Der optimale adjungierte Zustand ergibt ist durch die Projektionsformel (1.14) gegeben. Damit erfüllen die so definierten Funktionen die notwendigen und hinreichenden Optimalitätsbedingungen.

Zusätzlich kann man die Grenzen zwischen aktiven und inaktiven Anteilen der optimalen Steuerung, also die Mengen mit

$$-\frac{1}{\lambda}\bar{p} = a \quad \text{oder} \quad -\frac{1}{\lambda}\bar{p} = b$$

durch endlich viele Kurven γ_i (hier zwei) beschreiben. Damit ist das Maß der Menge I_3 begrenzt durch die Länge dieser Kurven:

$$|I_3| \leq 2h \sum |\gamma_i|$$

und (A3) ist erfüllt.

4.1.1 Ergebnisse der numerischen Testläufe

In diesem Abschnitt werden einige numerische Ergebnisse präsentiert. Dabei wurde das oben beschriebene Beispiel verwendet, mit $a = 3$, $b = 15$ und $\lambda = 1$. Von Interesse sind hierbei die Fehler der optimalen Steuerung sowohl in der $L^2(\Omega)$-Norm als auch in der $L^\infty(\Omega)$-Norm.

Zum Vergleich werden die Fehlernormen des gleichen Problems, allerdings mit einer Diskretisierung der Steuerung mit Hilfe stückweise konstanter Funktionen, gegenübergestellt.

Abbildung 4.3 zeigt das Konvergenzverhalten von $||\bar{u} - \bar{u}_h^h||_{L^2(\Omega)}$ für Diskretisierungsparameter $h = 0.04, 0.02, 0.01$ und 0.005.

Man erkennt gut das lineare Konvergenzverhalten bei der Diskretisierung mit stückweise konstanten Funktion. Bei einer Diskretisierung mit stückweise linearen Funktionen

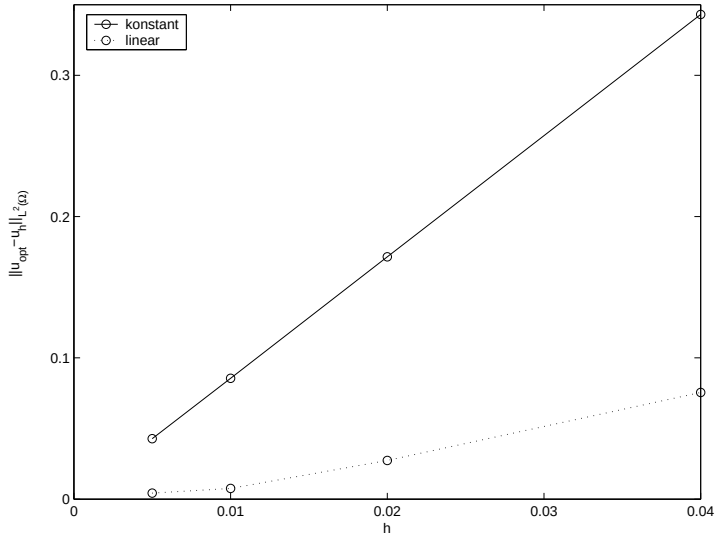

Abbildung 4.3: $||\bar{u} - \bar{u}_h^h||_{L^2(\Omega)}$

erreicht man eine höhere Konvergenzordnung, dass diese aber wirklich $h^{3/2}$ ist, kann man in der Abbildung nicht unbedingt erkennen.

Die Konvergenzordnung lässt sich aber gut an den zur Abbildung gehörenden Daten ablesen (Tabelle 4.1). Bei linearer Konvergenz ist das Produkt aus Norm und Anzahl der Diskretisierungen in einer Raumrichtung ($N = 1/h$) konstant. Analog muss bei einer Konvergenzordnung von $h^{3/2}$ das Produkt aus $N^{3/2}$ und der Norm konstant sein.

	stückweise konstant		stückweise linear	
N	$\|\|\bar{u} - \bar{u}_h^h\|\|_{L^2(\Omega)}$	$N \cdot \|\|\bar{u} - \bar{u}^h u_h\|\|_{L^2(\Omega)}$	$\|\|\bar{u} - \bar{u}_h^h\|\|_{L^2(\Omega)}$	$\cdot N^{3/2} \|\|\bar{u} - \bar{u}_h^h\|\|_{L^2(\Omega)}$
25	0.3431	8.58	0.0755	9.44
50	0.1712	8.56	0.0274	9.69
100	0.0856	8.56	0.0076	11.10
200	0.0428	8.56	0.0043	7.60

Tabelle 4.1: $||\bar{u} - \bar{u}_h^h||_{L^2(\Omega)}$

Man sieht deutlich, dass die Verbesserung bei $N = 100$ deutlich höher als erwartet und bei $N = 200$ deutlich geringer als erwartet ist. Im Durchschnitt ist aber die Konvergenzordnung $h^{(3/2)}$. Die Schwankungen sind dadurch zu erklären, dass manche Gitter die aktiven Mengen exakter treffen als andere.

Betrachtet man die Fehler in der L^∞-Norm (Abbildung 4.4), so stellt man fest, dass

die Konvergenzordnung sowohl bei der Diskretisierung mit stückweise konstanten Funktionen als auch bei der Diskretisierung mit stückweise linearen Funktionen von linearer Ordnung ist. Allerdings bestätigt sich die Erwartung, dass die absoluten Fehler bei der Diskretisierung mit stückweise linearen Funktionen geringer sind.

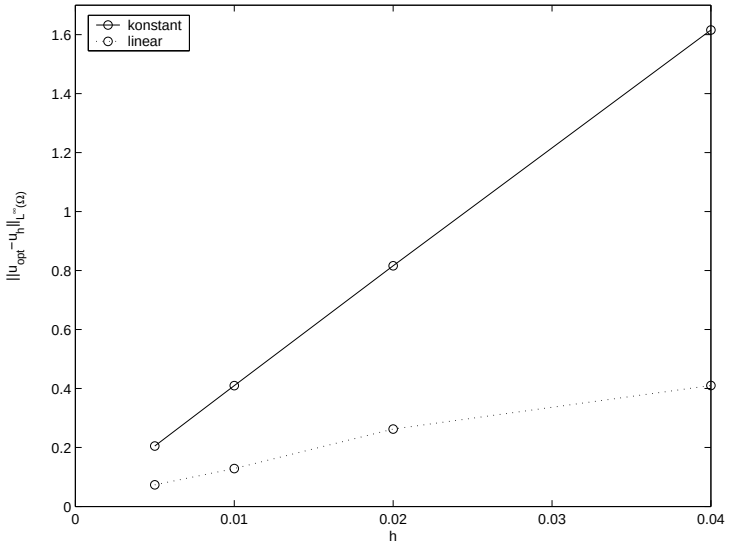

Abbildung 4.4: $||\bar{u} - \bar{u}_h^h||_{L^\infty(\Omega)}$

Die dazugehörigen Daten sind in Tabelle 4.2 aufgelistet.

	stückweise konstant		stückweise linear	
N	$\|\|\bar{u} - \bar{u}_h^h\|\|_{L^\infty(\Omega)}$	$N \cdot \|\|\bar{u} - \bar{u}_h^h\|\|_{L^\infty(\Omega)}$	$\|\|\bar{u} - \bar{u}_h^h\|\|_{L^\infty(\Omega)}$	$N \cdot \|\|\bar{u} - \bar{u}_h^h\|\|_{L^\infty(\Omega)}$
25	1.6155	40.38	0.4101	10.25
50	0.8163	40.82	0.2624	13.12
100	0.4103	41.03	0.1284	12.84
200	0.2049	40.98	0.0735	14.70

Tabelle 4.2: $||\bar{u} - \bar{u}_h^h||_{L^\infty(\Omega)}$

Beim Vergleich der Fehlernormen nach der Projektion (siehe dazu [1]) stellt man fest, dass sowohl bei stückweise linearen Ansatzfunktionen als auch stückweise konstanten Ansatzfunktionen fast identische Resultate erzielt werden. Die Unterschiede sind so gering, dass sie in der Grafik (Abbildungen 4.5 und 4.6) nicht erkennbar sind.

Die dazugehörigen Daten sind in Tabelle 4.3 aufgelistet.

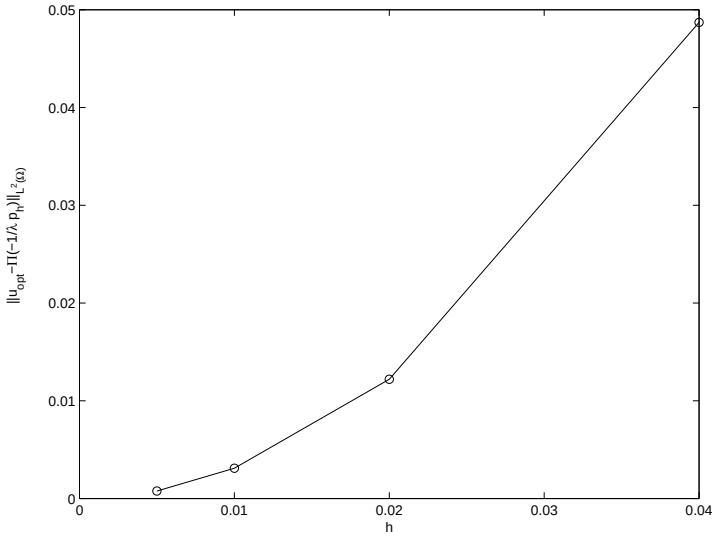

Abbildung 4.5: $||\bar{u} - \tilde{u}||_{L^2(\Omega)}$

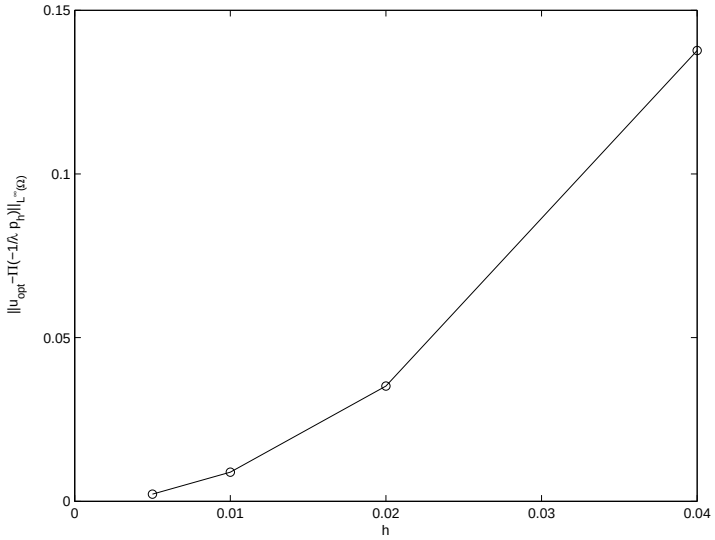

Abbildung 4.6: $||\bar{u} - \tilde{u}||_{L^\infty(\Omega)}$

Bei der Diskretisierung mit stückweise linearen Funktionen waren unstetige Übergänge an den Dreieckskanten erlaubt. Dabei ist von Interesse, wie groß die maximalen Unstetigkeiten sind und wie sich mit feinerer Diskretisierung verändern. Intuitiv erwartet man, dass sich bei einer Verdopplung der Anzahl der Diskretisierungen in einer Raumrichtung die maximale Unstetigkeit halbiert. Anhand der Tabelle 4.4 sieht man, dass dies in unse-

	stückweise konstant		stückweise linear	
N	$\|\|\bar{u}-\tilde{u}\|\|_{L^2(\Omega)}$	$\|\|\bar{u}-\tilde{u}\|\|_{L^\infty(\Omega)}$	$\|\|\bar{u}-\tilde{u}\|\|_{L^2(\Omega)}$	$\|\|\bar{u}-\tilde{u}\|\|_{L^\infty(\Omega)}$
25	0.0486	0.0487	0.1377	0.1377
50	0.0122	0.0122	0.0352	0.0352
100	0.0031	0.0031	0.0088	0.0089
200	0.00077	0.00076	0.00221	0.00221

Tabelle 4.3: $||\bar{u}-\tilde{u}||_{L^2(\Omega)}$ und $||\bar{u}-\tilde{u}||_{L^\infty(\Omega)}$

rem Beipiel auch der Fall ist. Die maximale Unstetigkeit bei der feinsten Diskretisierung entspricht aber nicht mehr der intuitiven Erwartung, sondern liegt deutlich darunter. Allerdings ist dieser Wert eine Ausnahme. Bei Tests mit noch feineren Diskretisierungen ergeben sich wieder die erwarteten Unstetigkeiten.

N	max. Unstetigkeit
25	1.007
50	0.5889
100	0.2719
200	$2.9 \cdot 10^{-7}$

Tabelle 4.4: maximale Unstetigkeit

Um den Rechenaufwand vergleichen zu können, sind in der Tabelle 4.5 die Rechenzeiten beider Programme angegeben, wobei zu beachten ist, dass dies die Rechenzeiten des kompletten Programms, also einschließlich der Berechnung der Normen sind.

N	stückweise konstant	stückweise linear
25	2.6 s	3.3 s
50	10.4 s	29 s
100	106 s	426 s
200	26 min	101 min

Tabelle 4.5: Rechenzeiten

4.2 Beispiel 2 – Neumann-Randbedingungen

In diesem Beispiel behandeln wir ein Problem mit Neumann-Randbedingungen

$$\begin{aligned} -\Delta y + cy &= u \qquad \text{in } \Omega \\ \partial_n y &= 0 \qquad \text{auf } \Gamma \end{aligned} \tag{4.1}$$

Die behandelte Theorie bezog sich aber immer auf Dirichlet-Randprobleme. Allerdings erhalten wir für die spezielle Struktur unseres Beispiels (Laplace-Operator, homogene Neumann-Daten, $\Omega = \Omega_h$) auch die benötigte $W^{2,p}$-Regularität (siehe Theorem 4.4.1.2 in [5]).

Wir konstruieren wieder den optimalen Zustand $\bar{y}$ (Abbildung 4.7) durch $\bar{y} = y_a - y_g$, mit einem analytischen Anteil $y_a(x_1, x_2) = \cos(\pi x_1)\cos(\pi x_2)$. Die Funktion y_g ist bestimmt durch die folgende Gleichung

$$\begin{aligned} -\Delta y_g + cy_g &= g \quad \text{in } \Omega \\ \partial_n y_g &= 0 \quad \text{auf } \Gamma \end{aligned}$$

mit der Inhomogenität

$$g(x_1, x_2) = \begin{cases} u_f(x_1, x_2) - a & \text{falls } u_f(x_1, x_2) < a \\ 0 & \text{falls } u_f(x_1, x_2) \in [a, b] \\ u_f(x_1, x_2) - b & \text{falls } u_f(x_1, x_2) > b \end{cases}$$

und $u_f(x_1, x_2) = (2\pi^2 + c)\cos(\pi x_1)\cos(\pi x_2)$.

Die optimale Steuerung $\bar{u}$ (Abbildung 4.8) ist gegeben durch (4.1)

$$\bar{u}(x_1, x_2) = \begin{cases} a & \text{falls } u_f(x_1, x_2) < a \\ u_f(x_1, x_2) & \text{falls } u_f(x_1, x_2) \in [a, b] \\ b & \text{falls } u_f(x_1, x_2) > b \end{cases}$$

Der optimale adjungierte Zustand ist

$$\bar{p}(x_1, x_2) = -(2\pi^2 + c)\lambda \sin(\pi x_1)\sin(\pi x_2)$$

und für y_d ergibt sich

$$\begin{aligned} y_d(x_1, x_2) &= \bar{y} + \Delta\bar{p} - c\bar{p} \\ &= y_a - y_g + (4\pi^4\lambda + 4\pi^2\lambda c + \lambda c^2)\sin(\pi x_1)\sin(\pi x_2) \end{aligned}$$

Auch hier ist leicht zu sehen, dass die definierten Funktionen die notwendigen und hinreichenden Optimalitätsbedingungen erfüllen. Mit der gleichen Argumentation wie im ersten Beispiel ist auch hier Voraussetzung (A3) erfüllt.

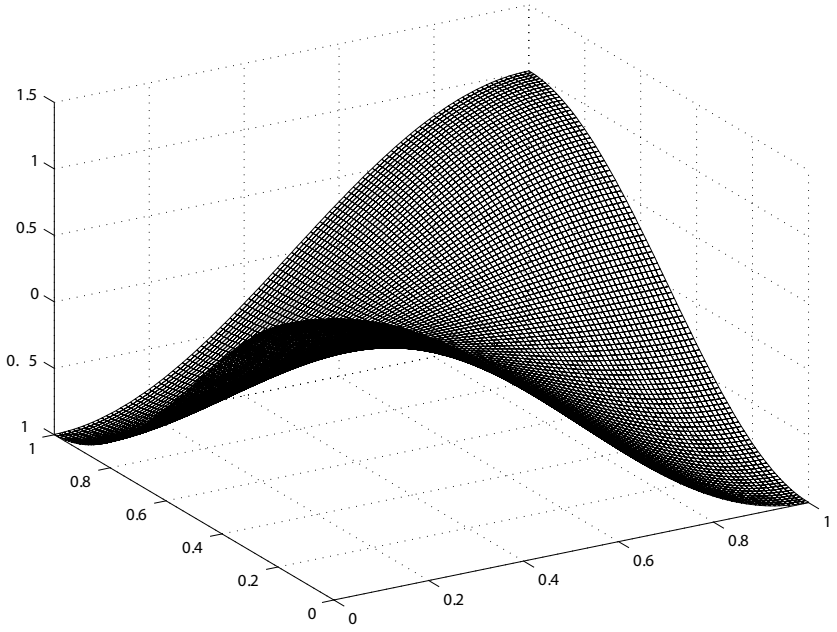

Abbildung 4.7: optimaler Zustand $\bar{y}$

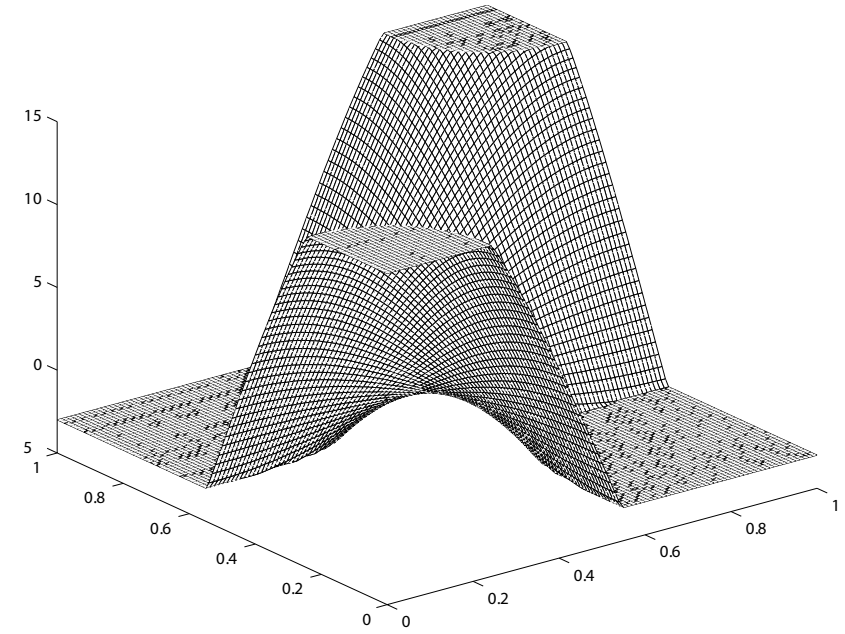

Abbildung 4.8: optimale Steuerung u

4.2.1 Ergebnisse der numerischen Testläufe

In den numerischen Tests wählten wir $a = -3$, $b = 15$, $\lambda = 1$ und $c = 1$.

Auch in diesem Beispiel erhalten wir die gleichen Konvergenzresultate wie im Dirichlet-Fall.

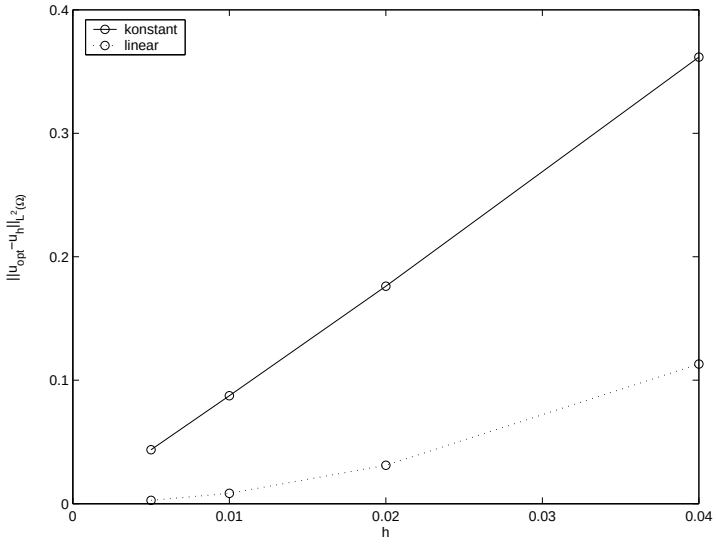

Abbildung 4.9: $||\bar{u} - \bar{u}_h^h||_{L^2(\Omega)}$

	stückweise konstant		stückweise linear	
N	$\|\|\bar{u} - \bar{u}_h^h\|\|_{L^2(\Omega)}$	$N \cdot \|\|\bar{u} - \bar{u}^h u_h\|\|_{L^2(\Omega)}$	$\|\|\bar{u} - \bar{u}_h^h\|\|_{L^2(\Omega)}$	$\cdot N^{3/2}\|\|\bar{u} - \bar{u}_h^h\|\|_{L^2(\Omega)}$
25	0.3617	5.79	0.1131	13.10
50	0.1761	5.83	0.0311	13.97
100	0.0874	5.85	0.0084	13.00
200	0.0437	5.90	0.0027	13.50

Tabelle 4.6: $||\bar{u} - \bar{u}_h^h||_{L^2(\Omega)}$

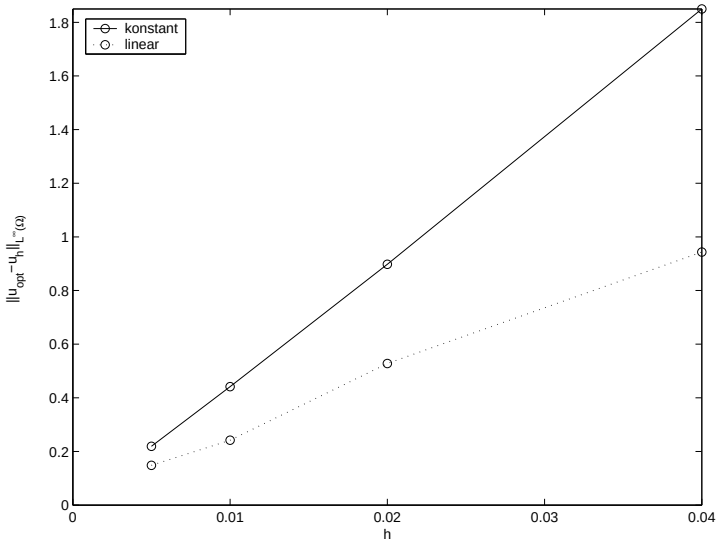

Abbildung 4.10: $||\bar{u} - \bar{u}_h^h||_{L^\infty(\Omega)}$

	stückweise konstant		stückweise linear	
N	$\|\|\bar{u} - \bar{u}_h^h\|\|_{L^\infty(\Omega)}$	$N \cdot \|\|\bar{u} - \bar{u}_h^h\|\|_{L^\infty(\Omega)}$	$\|\|\bar{u} - \bar{u}_h^h\|\|_{L^\infty(\Omega)}$	$N \cdot \|\|\bar{u} - \bar{u}_h^h\|\|_{L^\infty(\Omega)}$
25	1.8496	20.72	0.9434	13.97
50	0.8977	20.41	0.5279	15.52
100	0.4417	20.29	0.2420	15.35
200	0.2190	20.89	0.1482	15.13

Tabelle 4.7: $||\bar{u} - \bar{u}_h^h||_{L^\infty(\Omega)}$

	stückweise konstant		stückweise linear	
N	$\|\|\bar{u} - \tilde{u}\|\|_{L^2(\Omega)}$	$\|\|\bar{u} - \tilde{u}\|\|_{L^\infty(\Omega)}$	$\|\|\bar{u} - \tilde{u}\|\|_{L^2(\Omega)}$	$\|\|\bar{u} - \tilde{u}\|\|_{L^\infty(\Omega)}$
25	0.1052	0.1061	0.02544	0.2558
50	0.0263	0.0261	0.0638	0.0634
100	0.0066	0.0065	0.0159	0.0159
200	0.0016	0.0016	0.0040	0.0040

Tabelle 4.8: $||\bar{u} - \tilde{u}||_{L^2(\Omega)}$ und $||\bar{u} - \tilde{u}||_{L^\infty(\Omega)}$

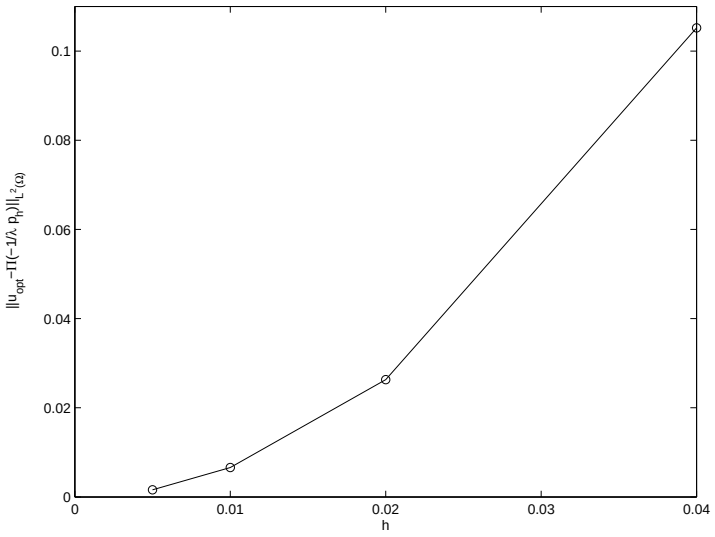

Abbildung 4.11: $||\bar{u} - \tilde{u}||_{L^2(\Omega)}$

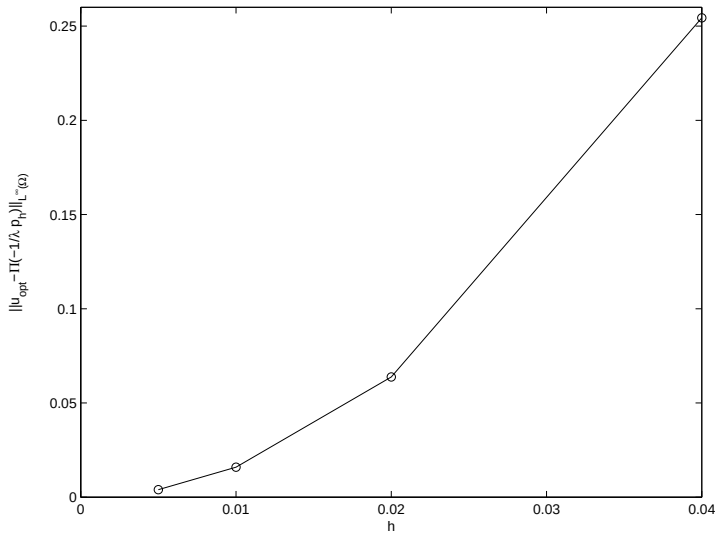

Abbildung 4.12: $||\bar{u} - \tilde{u}||_{L^\infty(\Omega)}$

N	max. Unstetigkeit
25	0.8507
50	0.5642
100	0.2377
200	0.1504

Tabelle 4.9: maximale Unstetigkeit

N	stückweise konstant	stückweise linear
25	32 s	36 s
50	113 s	139 s
100	577 s	848 s
200	72 min	144 min

Tabelle 4.10: Rechenzeiten

Anhang A

Numerische Realisierung

Im Kapitel 3 wurde die Diskretisierung und der Algorithmus zur numerischen Lösung eines Optimalsteuerungsproblems vorgestellt. Dabei sind wir allerdings nicht darauf eingegangen, wie man konkret die Ansatzfunktionen berechnet und die zur Lösung notwendigen Matrizen aufstellt. Dies soll Inhalt dieses Abschnitts sein.

Das Gebiet Ω ist das Einheitsquadrat, so dass wir eine regelmäßige Triangulierung (alle Dreiecke sind kongruent) wählen können. Dies vereinfacht die Berechnung der Matrizen.

A.1 Koordinatenwechsel auf Referenzdreieck

Ein wesentliches Grundprinzip ist die Abbildung eines beliebigen Dreiecks auf das Referenzdreieck. Wir betrachten ein beliebiges Dreieck T mit den Eckpunkten $P_0(x_0, y_0)$, $P_1(x_1, y_1)$ und $P_2(x_2, y_2)$. Dieses Dreieck kann bijektiv auf das Einheitsdreieck $\hat{T}$ mit den Ecken $(0, 0)$, $(1, 0)$ und $(0, 1)$ abgebildet werden (Abbildung A.1).

Dabei ist $F_T : \hat{T} \to T$ gegeben durch

$$F_T(\xi, \eta) := \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + \xi \begin{pmatrix} x_1 - x_0 \\ y_1 - y_0 \end{pmatrix} + \eta \begin{pmatrix} x_2 - x_0 \\ y_2 - y_0 \end{pmatrix}$$

mit

$$|DF_T(\xi, \eta)| = \begin{vmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{vmatrix} = 2|T| \neq 0 \tag{A.1}$$

Die Differentialoperatoren rechnen sich beim Koordinatenwechsel wie folgt um

$$\begin{pmatrix} \partial x \\ \partial y \end{pmatrix} = \begin{pmatrix} \xi_x & \eta_x \\ \xi_y & \eta_y \end{pmatrix} \begin{pmatrix} \partial \xi \\ \partial \eta \end{pmatrix}$$

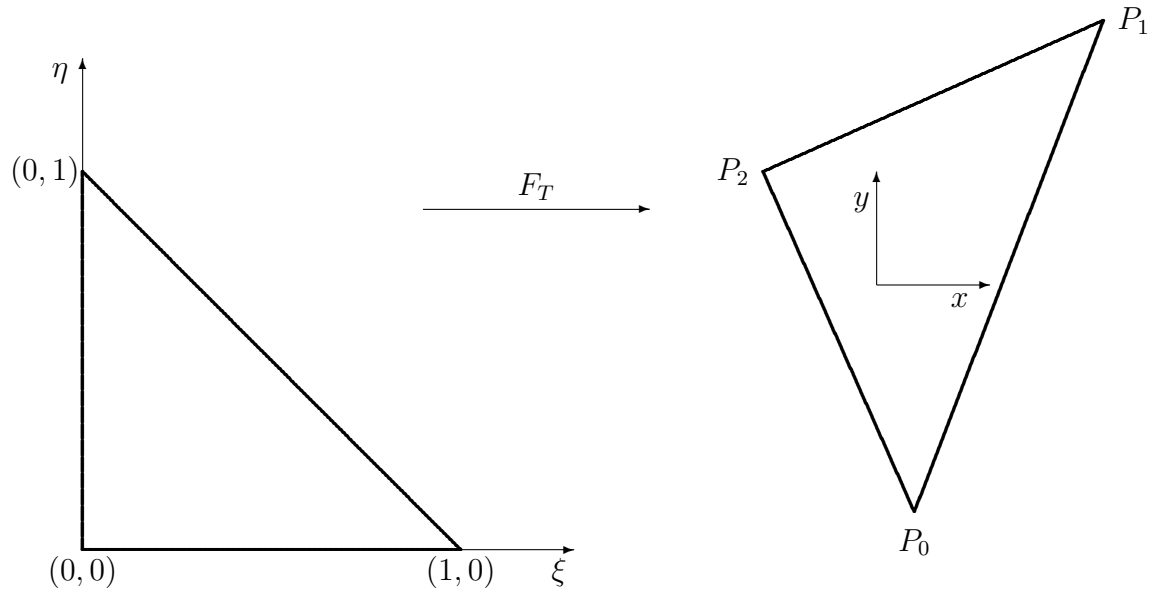

Abbildung A.1: Transformation auf das Einheitsdreieck

wobei

$$\xi_x = \frac{1}{2|T|}\left(y_2 - y_0\right), \qquad \eta_x = -\frac{1}{2|T|}\left(y_1 - y_0\right)$$
$$\xi_y = -\frac{1}{2|T|}\left(x_2 - x_0\right), \qquad \eta_y = \frac{1}{2|T|}\left(x_1 - x_0\right)$$

A.2 Ansatzfunktionen

In unserem Fall haben wir zwei verschiedene Diskretisierungen. Zum einen werden der Zustand y und der adjungierte Zustand p durch stückweise lineare Funktionen diskretisiert, die global stetig sind. Die Steuerung u wird auch durch stückweise lineare Funktionen diskretisiert, allerdings erlauben wir hier Unstetigkeiten an den Dreieckskanten.

A.2.1 global stetig

Wir wählen einen beliebigen inneren Punkt (x_ν, y_μ). Mit $\Omega_{\nu\mu}$ bezeichnen wir nun ein sechseckiges Gebiet mit dem Mittelpunkt (x_ν, y_μ). Die Eckpunkte dieses Sechsecks sind

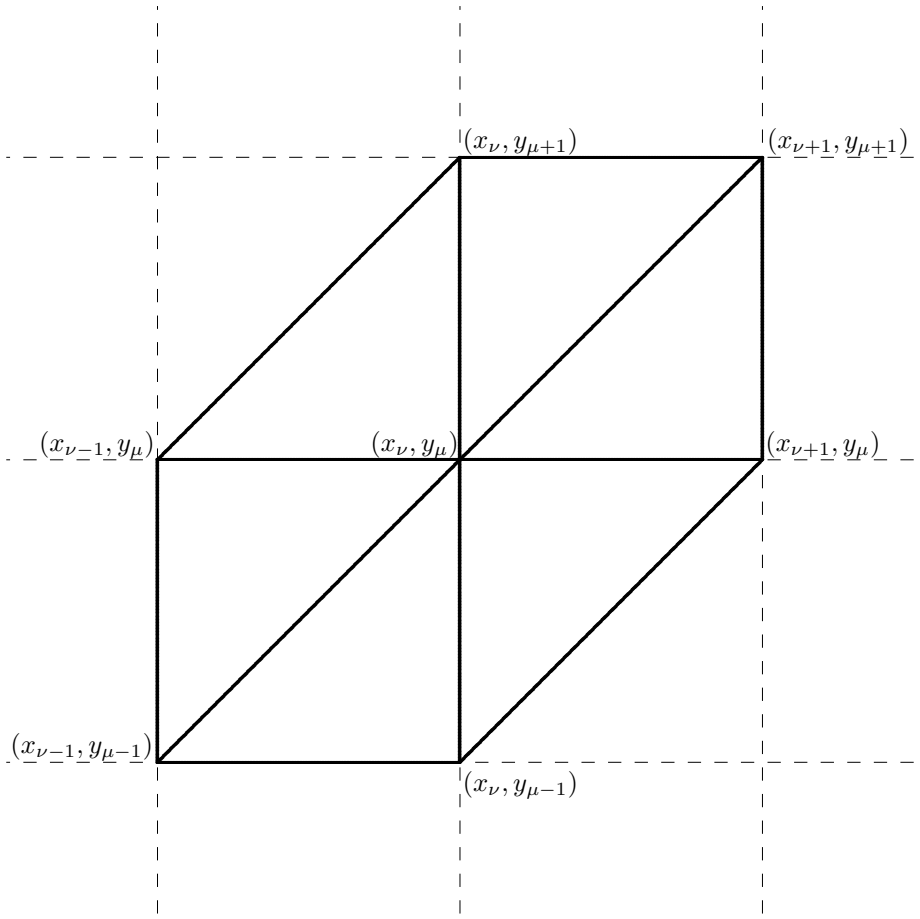

Abbildung A.2: Sechseck $\Omega_{\nu\mu}$

$(x_{\nu-1}, y_{\mu-1}), (x_\nu, y_{\mu-1}), (x_{\nu+1}, y_\mu), (x_{\nu+1}, y_{\mu+1}), (x_\nu, y_{\mu+1})$ und $(x_{\nu-1}, y_\mu)$ (Abbildung A.2).

Wir fordern nun für die Ansatzfunktionen $\phi_{\nu\mu}(x, y)$

1. Die Funktion sei linear in jedem Teildreieck.
2. Die Werte in den Gitterpunkten seien gegeben durch

$$\phi_{\nu\mu}(x_k, y_l) = \begin{cases} 1 & k = \nu, l = \mu \\ 0 & \text{sonst} \end{cases}$$

Insbesondere folgt daraus, dass $\phi_{\nu\mu}(x, y) = 0$ gilt außerhalb von $\Omega_{\nu\mu}$, d.h. diese Funktionen haben lokalen Träger.

Für jeden inneren Punkt erhält man also 6 Teilfunktionen (eine für jedes Teildreieck). Auf jedem Teildreieck ist die lineare Funktion durch Vorgabe der Werte in den Eckpunkten

eindeutig bestimmt. Prinzipiell gilt $\phi_{\nu\mu} = a + bx + cy$, mit $b = D_x\phi$ und $c = D_y\phi$. Betrachten wir die möglichen Konstellationen auf dem Referenzdreieck, so erhält man drei Ansatzfunktionen (eine pro Freiheitsgrad), die sogenannten baryzentrischen Koordinaten:

$$\begin{aligned} \hat{t}_1(\xi,\eta) &= 1-\xi-\eta \\ \hat{t}_2(\xi,\eta) &= \xi \\ \hat{t}_3(\xi,\eta) &= \eta \end{aligned}$$

Die Nummerierung bezieht sich dabei auf die Nummerierung der Eckpunkte, d.h. $\hat{t}_i(P_j) = \delta_{ij}$.

Die Teilfunktion $\phi_{\nu\mu,i}$ des Dreiecks T_i zum Knotenpunkt (x_ν, y_μ) ist dann also gegeben durch:

$$\phi_{\nu\mu,i}(x,y) = \hat{t}_k\left(F_{T_i}^{-1}(x,y)\right)$$

wobei der Index k die lokale Knotennummer ist, in der die Ansatzfunktion den Wert 1 annimmt.

Das ganze Prinzip kann man auch auf die Randknoten ausdehnen. Man schneidet die Hütchenfunktionen dann am Rand einfach ab, definiert sie also nur auf den Teildreiecken, die zum Einheitsquadrat gehören. Dies erlaubt eine allgemeinere Nutzung des Programms auch für andere Randbedingungen.

Der Zustand y und der adjungierte Zustand p wird dann dargestellt durch

$$y = \sum_{i=1}^{n_P} y_i\phi_i, \quad p = \sum_{i=1}^{n_P} p_i\phi_i$$

Dabei bezeichnet n_P die Anzahl der Gitterpunkte.

A.2.2 unstetige Übergänge

Im Gegensatz zum vorigen Abschitt fällt bei der Diskretisierung der Steuerung die Beschränkung der stetigen Übergänge weg. Das führt dazu, dass die Ansatzfunktionen auf jedem Dreieck völlig unabhängig von den anderen Dreiecken definiert werden können. Auf jedem Dreieck T_i definieren wir drei lineare Ansatzfunktionen $\varphi_{i,j}$, $j = 1,2,3$ mit der Eigenschaft $\varphi_{i,j}(P_k) = \delta_{jk}$. Diese sind im Gegensatz zum obigen Fall nur lokal auf dem jeweiligen Dreieck definiert. Damit ergibt sich folgende Diskretisierung für die Steuerung (nummerieren alle der Reihe nach durch, damit wir nur noch einen Index haben, wobei man sich auf eine sinnvolle und vor allem gleichbleibende Nummerierung der lokalen

Knoten eines Dreiecks einigen sollte):

$$u = \sum_{i=1}^{3n_T} u_i \varphi_i$$

Dabei bezeichnet n_T die Anzahl der Dreiecke.

A.3 Aufstellen der Matrizen

Wie im Kapitel 3 beschrieben, benötigen wir folgende Matrizen:

- Massenmatrix $M \in \mathbb{R}^{n_P \times n_P}$ mit Elementen

$$M_{ij} = (\phi_i, \phi_j)_{L^2(\Omega)}$$

- Steifigkeitsmatrix $A \in \mathbb{R}^{n_P \times n_P}$ mit Elementen

$$A_{ij} = (\nabla\phi_i, \nabla\phi_j)_{L^2(\Omega)}$$

- Massenmatrix $\tilde{M} \in \mathbb{R}^{3n_T \times 3n_T}$ mit Elementen

$$\tilde{M}_{ij} = (\varphi_i, \varphi_j)_{L^2(\Omega)}$$

- Transformationsmatrix $R \in \mathbb{R}^{n_P \times 3n_T}$ mit Elementen

$$R_{ij} = (\varphi_i, \phi_j)_{L^2(\Omega)}$$

Etwas ausführlicher besprechen wir die Aufstellung der Massenmatrix M, die anderen Matrizen kann man analog berechnen. Aus der Definition der Ansatzfunktionen folgt sofort, dass $M_{ij} = 0$, falls die Punkte P_i und P_j nicht zum gleichen Dreieck gehören.

Bezeichne $s_i := \text{supp}\phi_i$ den Träger der Ansatzfunktionen, dann gilt:

$$M_{ij} = \int_{s_i \cap s_j} \phi_i(x)\phi_j(x)\mathrm{d}x = \sum_{T \subset s_i \cap s_j} \int_T \phi_i(x)\phi_j(x)\mathrm{d}x$$

und mit (A.1)

$$\int_T \phi_i(x)\phi_j(x)\mathrm{d}x = h^2 \int_{\hat{T}} \hat{t}_{K_i}(\xi,\eta)\hat{t}_{K_j}(\xi,\eta)\mathrm{d}(\xi,\eta)$$

da in unserem Fall alle Dreiecke kongruent sind mit $2|T| = h^2$. Damit kann man sich nun relativ leicht überlegen, wie die Massenmatrix M aufgebaut ist. Betrachten wir zunächst einen inneren Knoten. Zu jedem inneren Knoten gibt es genau 7 Knoten, die einen

nichtleeren Schnitt des Trägers der zugehörigen Ansatzfunktionen haben. Das sind: der betrachtete Knoten selbst (bildet Hauptdiagonale), die Nachbarknoten rechts und links (bilden obere und untere Nebendiagonale), die Nachbarknoten oben und unten (bilden die obere und untere $(N+1)$-te Nebendiagonale) und schließlich der Knoten rechts vom oberen Nachbarknoten (bildet obere $(N+2)$-te Nebendiagonale) sowie der Knoten links vom unteren Nachbarknoten (bildet untere $(N+2)$-te Nebendiagonale). Alle anderen Einträge der Matrix sind Null.

In unserem Fall haben wir homogene Dirichlet-Randbedingungen, so dass alle Spalten und Zeilen zu Randknoten Nullzeilen bzw. Nullspalten sind. Bleibt also nur noch die Werte der Matrixelemente auszurechnen. Es ergeben sich:

$$(\phi_i, \phi_i)_{L^2(\Omega)} = \frac{h^2}{2}$$

$$(\phi_i, \phi_{i\pm1})_{L^2(\Omega)} = (\phi_i, \phi_{i\pm(N+1)})_{L^2(\Omega)} = (\phi_i, \phi_{i\pm(N+2)})_{L^2(\Omega)} = \frac{h^2}{12}$$

wobei in diesen Formeln nur Indizes vorkommen dürfen, die zu inneren Punkten gehören (für alle anderen ist das Matrixelement Null).

Ganz analog werden die anderen Matrizen aufgestellt. Wir geben hier nur den prinzipiellen Aufbau an. Für die Steifigkeitsmatrix gilt:

$$A_{ij} = \int_{s_i \cap s_j} \nabla\phi_i(x)\nabla\phi_j(x)\mathrm{d}x = \sum_{T \subset s_i \cap s_j} \int_T \nabla\phi_i(x)\nabla\phi_j(x)\mathrm{d}x$$

und mit (A.1)

$$\begin{aligned}\int_T \nabla\phi_i(x)\nabla\phi_j(x)\mathrm{d}x = \\ h^2 \int_{\hat{T}} \left(\frac{\partial \hat{t}_{K_i}}{\partial\xi}\xi_x + \frac{\partial \hat{t}_{K_i}}{\partial\eta}\eta_x\right)\left(\frac{\partial \hat{t}_{K_j}}{\partial\xi}\xi_x + \frac{\partial \hat{t}_{K_j}}{\partial\eta}\eta_x\right)\mathrm{d}(\xi,\eta) \\ + \int_{\hat{T}} \left(\frac{\partial \hat{t}_{K_i}}{\partial\xi}\xi_y + \frac{\partial \hat{t}_{K_i}}{\partial\eta}\eta_y\right)\left(\frac{\partial \hat{t}_{K_j}}{\partial\xi}\xi_y + \frac{\partial \hat{t}_{K_j}}{\partial\eta}\eta_y\right)\mathrm{d}(\xi,\eta)\end{aligned}$$

Bei der Massenmatrix $\tilde{M}$ vereinfacht sich der Aufbau ein wenig, da die Ansatzfunktionen immer nur auf genau einem Dreieck definiert sind. Damit entstehen 3×3-Untermatrizen für jedes Dreieck, die dann zur Massenmatrix zusammengefasst werden.

Zur Aufstellung der Transformationsmatrix betrachten wir einen inneren Punkt. Aufgrund unserer regelmäßigen Triangulierung kann man sehr einfach die Nummern der 6

Dreiecke bestimmen, die zu dem Sechseck der Ansatzfunktion des inneren Punktes gehören. So kann man zeilenweise die Matrixelemente bestimmen, deren Werte nicht verschwinden. Im allgemeinen Fall benötigt man Extrastrukturen, in der man zum Beipiel zu jeder Dreiecksnummer die zugehörigen Knotennummern abspeichert.

Um das Programm auch für andere Randwerte benutzen zu können, wurden die Randpunkte mit einbezogen. In unserem speziellen Fall der homogenen Dirichlet-Daten mussten die Matrizen dann angepasst werden. Dazu muss man die Gleichungssysteme betrachten, in denen die jeweiligen Matrizen vorkommen und die Einträge in den Zeilen bzw. Spalten, die zu Randpunkten gehören, so ändern, dass die y_i zu den Randpunkten gerade verschwinden. Beispielsweise lautet das Gleichungssystem, dass sich aus der Diskretisierung der Variationsformulierung der Zustandsgleichung ergibt

$$Ay = Ru$$

Sei i ein beliebiger Index, der zu einem Randpunkt gehört. Setzt man nun $A_{ij} = \delta_{ij}$ für alle $j \in \{1, \ldots, n_P\}$ und $R_{ij} = 0$ für alle $j \in \{1, \ldots, n_P\}$, so erzwingt man $y_i = 0$. Ganz analog modifiziert man die anderen Matrizen.

A.4 Normberechnungen

Wir berechnen die Normen hier nicht analytisch, da die exakte Integration aufgrund der nichtdifferenzierbaren Stellen von u relativ aufwendig wäre. Die L^2-Norm wird mit Hilfe der Gauss-Quadratur 2. Ordnung ausgewertet:

$$\begin{aligned} ||\bar{u} - \bar{u}_h^h||^2_{L^2(\Omega)} &= \int_\Omega (\bar{u} - \bar{u}_h^h)^2 \mathrm{d}x \\ &= \sum_{T_i \in T_h} \int_{T_i} (\bar{u} - \bar{u}_h^h)^2 \mathrm{d}x \end{aligned}$$

mit

$$\begin{aligned} \int_{T_i} (\bar{u} - \bar{u}_h^h)^2 \mathrm{d}x &\approx \frac{|T_i|}{3} \sum_{j=1}^{3} \left\{ \bar{u}\left(x_{e,j}^{(i)}\right) - u_j \right\}^2 \\ &= \frac{|T_i|}{3} \sum_{j=1}^{3} \left\{ \left[\bar{u}\left(x_{e,j}^{(i)}\right) \right]^2 - 2\bar{u}\left(x_{e,j}^{(i)}\right) u_j + u_j^2 \right\} \end{aligned}$$

Dabei bezeichnet $x_{e,j}^{(i)}$ den j-ten Kantenmittelpunkt von T_i. u_j ist gerade der Wert der diskretisierten Steuerung im j-ten Kantenmittelpunkt. Da die Steuerung linear auf jedem Dreieck ist, gilt

$$u_j = \frac{u_k + u_l}{2}$$

wobei u_k, u_l die Werte von $\bar{u}_h^h$ an den zur Kante j gehörenden Knotenpunkten sind. Diese sind gerade die Koeffizienten der passenden Ansatzfunktionen.

Die L^∞-Norm wird folgendermaßen approximiert

$$||\bar{u} - \bar{u}_h^h||_{L^\infty(\Omega)} \approx \max_{T_i \in T_h} \left(\max_{j \in \{1,2,3\}} \left| \bar{u}\left(x_{P,j}^{(i)}\right) - u_j \right| \right)$$

wobei $x_{P,j}^{(i)}$ den j-ten Eckpunkt und u_j der Koeffizient der j-ten Ansatzfunktion (ist gleich dem Wert von $\bar{u}_h^h$ im j-ten Eckpunkt) des Dreiecks T_i bezeichnet.

Dies ist eine recht grobe Abschätzung, deshalb nimmt man noch weitere Testpunkte dazu, so zum Beispiel die Kantenmittelpunkte und den Schwerpunkt der Dreiecke.

Anhang B

Das Programm

In diesem Kapitel wird kurz das verwendete Programm beschrieben und der Quellcode angegeben. Grundlage war das Programm von Christian Meyer, das zur Berechnung der Testdaten für die gleichen Optimalsteuerungsprobleme diente, allerdings mit einer Diskretisierung der Steuerung durch konstante Ansatzfunktionen ([1]).

B.1 Grundstruktur

Startpunkt des Programms ist constrained.m. Alle weiteren Funktionen werden von dort aus aufgerufen. Dies sind:

- gridinit.m: Initialisierung der Gitterdaten
- createYd.m: Berechnung der Zielfunktion y_d und der analytischen Lösung für y, p und u
- stiffnessdiri.m: Aufstellen der Steifigkeitsmatrix
- massdiri.m: Aufstellen der Massenmatrix M
- calcR.m: Aufstellen der Transformationsmatrix R
- calcMtilde.m: Aufstellen der Massenmatrix $\tilde{M}$
- calcaffin.m, calcEo.m: Berechnung des Korrektivterms
- calcb.m: Berechnung der rechten Seite b (siehe (3.10)
- Bmulti.m: Berechnung der Matrix B (siehe (3.10)

- operatorS.m,operatorStrans.m: Anwendung des Lösungsoperators S
- calcAdjoint.m: Berechnung des adjungierten Zustands
- findTris.m: findet die Dreiecke am Rand der aktiven Mengen (dient zur Visualisierung und zur genaueren Berechnung der Fehlernormen durch Gitterverfeinerung)
- L2norm.m: Berechnung der L2-Fehlernorm: $||\bar{u} - \bar{u}_h^h||_{L^2(\Omega)}$
- projL2norm.m: Berechnung der L2-Norm nach der Projektion: $||\bar{u} - \tilde{u}||_{L^2(\Omega)}$
- projLinfnorm.m: Berechnung der zugehörigen L^∞-Normen

Da wir hier nur die Diskretisierung der Steuerung verändert haben, konnten große Teile des Programms unverändert übernommen werden. Modifiziert wurden im Wesentlichen nur calcR, calcMtilde und die Normberechnungen. Zusätzlich wurde noch die Funktion findMaxStep.m geschrieben, die die maximale Unstetigkeit an den Dreieckskanten ermittelt.

B.2 Quellcode

Wir geben hier den Quellcode an, der für die Berechnung des 1. Beispiels mit Dirichlet-Randbedingungen benutzt wurde. Für das 2. Beispiel mussten die Matrixen angepasst werden. Diese geben wir hier auch an (stiffnessneumann.m, massneumann.m, calcRneumann.m). Da sich ansonsten nur die Mathematik ändert, verzichten wir hier auf eine vollständige Auflistung des Quellcodes für das 2. Beispiel.

constrained.m

```
% ---------------------------------------------
% ------- DUAL-PRIMAL ALGORITHM FOR LINEAR-QUADRATIC -------
% ----- OPTIMAL CONTROLPROBLEMS WITH CONTROL-CONSTRAINS -----
% ---------------------------------------------
% ------- OBJECTIVE FUNCTIONAL: 0.5*|y-yd| + 0.5*lambda*|u| -------
% ---------------------------------------------
% --- PDE: -laplace(y) = u WITH DIRICHLET CONDITIONS ---
% ---------------------------------------------
%
% ---------------------------------------------
```

```
%  INTERFACE: [H, L2proj, L2konst] = constrained1(N, lambda, tiny, maxit, ua, ub,
NN)
%
%     INPUTPARAMETERS:
%           N - mesh size = 1/N
%           lambda - regulisation parameter in the objective functional
%           tiny - tolerance for cg-method
%           maxit - maximum number of iterations for cg-method
%           ua - lower bound for the control function u
%           ub - upper bound for the control function u
%           NN - mesh refinement parameter for plotting
%
%     RETURNVALUES:
%            H - mesh size (1/N)
%            L2proj - L2-norm of |u_opt - Proj p|
%            L2konst - L2-norm of |u_opt - u_h|
% ---------------------------------------------
%  Author: Christian Meyer
%  Date: 06.02.2003
%  modified by Rene Simon, 02.04.2004
% ---------------------------------------------
%  FOLLOWING FUNCTIONS ARE USED:
%  - gridinit.m
%  - creatYd.m
%  - stiffnessdiri.m
%  - massdiri.m
%  - calcR.m
%  - calcMtilde.m
%  - calcaffin.m
%  - calcEo.m
%  - calcb.m
%  - Bmulti.m
%  - calcL2norm.m
%  - operatorS.m
%  - operatorStrans.m
%  - calcAdjoint.m
%  - findTris.m
```

```
%  - projL2norm.m
%  - projLinfnorm.m
%  - L2norm.m
%  - plotproj.m
%
% ------------------------------------------

function [H, L2proj, L2konst] = constrained(N, lambda, tiny, maxit, ua, ub, NN)

fprintf('\n==== PROGRAM TO SOLVE A LINEAR-QUADRATIC CONTROL PROBLEM ====\n\n');
fprintf('objective functional: 0.5*|y-yd| + 0.5*lambda*|u|\n');
fprintf('PDE: -laplace(y) = u\n');

%   Checking input data
if (ua >= ub)
    fprintf('Inputerror: ua >= ub -> Exit!\n');
end
if N <= 0
   fprintf('   Inputerror: N negativ => Exit\n');
   break;
end

%   Initializing grid data
gridinit;

%   Calculation of th objective state function yd
createYd;

%   Calculation of the stiffness matrix
stiffnessdiri;

%   Calculation of the mass matrix
%   for linear ansatz functions
massdiri;

%   Calculation of the (nP x 3nT)-matrix R
calcR;
```

```
%   Calculation of the mass matrix
%   for linear ansatz functions
calcMtilde;

%   Calculation of the corrective term in the PDE
%   for the analytical solution
%   ATTENTION: only used for this specific analytical solution
calcaffin;

%  The new objective state:
zd = yd- affin;
%   ATTENTION: only neccessary for this specific analytical solution
%   In case of unknown analytical solution: zd = yd !!!!

%   Setting initial values and initial sets
%   initial value: u = 0.5*(ua+ub)
fprintf('\n==== Setting initial values ====\n');
uInaktiv = 0.5*(ua+ub) * ones(3*nT, 1);
uAktiv = zeros(3*nT, 1);
inaktiv = [1:3*nT]';
aktiv_a = [];
aktiv_b = [];
aktiv = [aktiv_a; aktiv_b];
fprintf('==== done ====\n');

fprintf('\n\n==== STARTING PRIMAL-DUAL ALGORITHM ====\n');
set_change = 1;
pass = 1;
while (set_change ~= 0 & pass < maxit)

    fprintf('\n\n==== PASS %d THROUGH PRIMAL-DUAL ITERATION ====\n', pass);

    %   Calculation of the inhomogenity b
    %   (see "Dokumentation zum free- und constrained-Code", eq.(57))
    calcb;
```

```
 fprintf('\n    == Starting cg-method to solve B uI = b ==\n');
 fprintf('\n    == cg: calculating initial residual ==\n');
 r = b - Bmulti(uInaktiv, A, R, M, Mtilde, lambda, tiny, maxit, nP, aktiv);
 g = r;
 fprintf('    == done ==\n');

 L2u_i = calcL2norm(Mtilde, uInaktiv);
 err = calcL2norm(Mtilde, r);
 if (L2u_i == 0.0)
     L2u_i = 1.0;
 end

check = 1;
 loop = 1;
 while ( err/L2u_i > tiny & check == 1 & loop <= maxit)
     fprintf('\n    == cg: pass %i through cg-iteration: ==\n', loop);

     %   Attention: the aktive parts of b have to be zero
     %   (see "Dokumentation zum free- und constrained-Code", p.15/16)
     g(aktiv) = 0.0;

     L2u_i_old = L2u_i;

     %   Evaluating d = B g
     d = Bmulti(g, A, R, M, Mtilde, lambda, tiny, maxit, nP, aktiv);

     gamma = g'*d;
     alpha = 1/gamma * r'*g;

     uInaktiv = uInaktiv + alpha*g;

     r = r - alpha*d;

     beta = 1/gamma * r'*d;

     g = r - beta*g;
```

```
        %   error calculation for the cg break condition
        %   (err corresponds to the L2-norm of lambda u + p
        %   see "Dokumentation zum free- und constrained-Code", eq.(27))
        err = r'*(MtildeInv * r);
        L2u_i = calcL2norm(Mtilde, uInaktiv);
        Eukl = norm(r, 2) / nT;

        if (abs(L2u_i_old - L2u_i)/L2u_i < tiny )
            check =0;
            fprintf('\n    cg: WARNING: Difference between ');
            fprintf('     two iteratives smaller than %e\n', tiny);
            fprintf('                        => Termination of the cg-iteration\n');
        end

      fprintf('    == cg: pass %i finished:', loop);
        fprintf(' L2-norm of lambda u + p: e = %e\n', err);
        fprintf('                                                  Relative error: e/L2u_i
= %e\n', err/L2u_i);
        fprintf('                                                  Euklidian norm of
the residual= %e ==\n', Eukl);
        loop = loop + 1;
    end

   if (loop == maxit)
        fprintf('\n    cg: WARNING: Maximum number of iteration steps % i reached\n',
maxit);
        fprintf('                       => Termination of the cg-iteration\n');
    else
      fprintf('\n    == cg: cg converged successfully! ==\n');
    end

   fprintf('\n    - cg: L2-norm of lambda u + p: e = %e -\n', err);
    fprintf('\n    == cg-method finished ==\n');


    %   Aktive-Set method:
```

```
%   See pseudo-code on p. 14/15,
%   "Dokumentation zum free- und constrained-Code"

%   Calculation of z = S u
%   (step 2.c in the pseudo-code)
fprintf('\n    == Calculating the state z = S u ==\n');
z = operatorS(uInaktiv, A, R, tiny, maxit, nP) + zAktiv;

fprintf('    == done ==\n');
%   Calculation of p = S^*(z-zd)
calcAdjoint;

fprintf('\n    == Defining new sets I, A-, A+ ==\n');

%   Evaluating uI < ua
%   (step 2.d in the pseudo-code)
bool_a = (uInaktiv(inaktiv) < ua);
set_change = sum(bool_a);
new_aktiv_a = nonzeros(bool_a .* inaktiv);
uInaktiv(new_aktiv_a) = 0.0;
uAktiv(new_aktiv_a) = ua;
inaktiv = nonzeros(~bool_a .* inaktiv);

%   Evaluating uI > ub
%   (step 2.d in the pseudo-code)

bool_b = (uInaktiv(inaktiv) > ub);
set_change = set_change + sum(bool_b);
new_aktiv_b = nonzeros(bool_b .* inaktiv);
uInaktiv(new_aktiv_b) = 0.0;
uAktiv(new_aktiv_b) = ub;
inaktiv = nonzeros(~bool_b .* inaktiv);

%   Evaluating p + uA- < 0
%   (step 2.e in the pseudo-code)

bool_a = (p(aktiv_a)+lambda*uAktiv(aktiv_a) < 0.0);
```

```
    set_change = set_change + sum(bool_a);
    new_inaktiv = nonzeros(bool_a .* aktiv_a);
    uAktiv(new_inaktiv) = 0.0;
    uInaktiv(new_inaktiv) = ua;
    inaktiv= [inaktiv; new_inaktiv];
    aktiv_a = nonzeros(~bool_a .* aktiv_a);
    aktiv_a = [aktiv_a; new_aktiv_a];

    %   Evaluating p + uA+ > 0
    %   (step 2.e in the pseudo-code)

    bool_b = (p(aktiv_b)+lambda*uAktiv(aktiv_b) > 0.0);
    set_change = set_change + sum(bool_b);
    new_inaktiv = nonzeros(bool_b .* aktiv_b);
    uAktiv(new_inaktiv) = 0.0;
    uInaktiv(new_inaktiv) = ub;
    inaktiv= [inaktiv; new_inaktiv];
    aktiv_b = nonzeros(~bool_b .* aktiv_b);
    aktiv_b= [aktiv_b; new_aktiv_b];

    aktiv = [aktiv_a; aktiv_b];
    fprintf('    == done ==\n');

    fprintf('\n==== Pass %d through primal-dual iteration finished:\n', pass);
    fprintf('            Number of elements in I, A-, A+, that were changed: %d ====\n',
set_change);

    pass = pass+1;
end

if (pass == maxit)
     fprintf('\n==== WARNING: Maximum number of iteration steps % i reached\n', ma-
xit);
     fprintf('                    => Termination of the Primal-Dual-iteration ====\n');
else
    fprintf('\n==== PRIMAL-DUAL ALGORITHM CONVERGED SUCCESSFULLY! ====\n');
end
```

```
u = uInaktiv + uAktiv;
findMaxStep;
fprintf('\n maximaler Sprung: %e am Punkt: %i\n',maximal,max_index);

proj = -1/lambda * pP;
%   y results form z plus corrective term in PDE
y = z + affin;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   POSTPROCESSING

fprintf('\n\n==== Postprocessing: Projection of p ====\n');

%   Lokale Gitterverfeinerung um den Rand der aktiven Menge
%   Damit Projektion des adjungierten Zustands visualisiert werden kann
%   (siehe Superkonvergenz-Paper) !!!
fprintf('    == Searching active-set-boundary ==\n');
findTris;

%  Projektion des adjungierten Zustands
%  und Berechnung der L2-Norm |u_opt - Proj p_h|
fprintf('    == Calculating L2-Norm of |u_opt - Proj p| ==\n');
projL2norm;
projLinfNorm;
fprintf('    == Calculating L2-Norm of |u_opt - u_h| ==\n');
L2norm;

fprintf('\nResults:\n');
fprintf('|uopt-uh|_proj = %f\n', L2proj);
fprintf('|uopt-uh|_proj_inf= %f\n',L_inf);

fprintf('|uopt-uh|_konst = %f\n\n', L2konst);
fprintf('\n LInftyNorm: %f\n',L_inf_u);

fprintf('==== done ====\n');
```

gridinit.m

```
% --------------------------------
% ------- INITIALIZATION OF GRID DATA -------
% --- FOR A UNIFORM GRID ON THE UNIT SQUARE ---
% --------------------------------
%
% --------------------------------
%  CALLED BY: constrained.m
%
%  NEEDED INPUT: N (from constrained.m)
% --------------------------------
%  Author: Christian Meyer
%  Date: 06.02.2003
%  modified by Rene Simon, 04.04.2004
% --------------------------------

fprintf('\n==== Initializing grid data ====\n');

%   grid size
fprintf('    grid size:\n');
H = 1.0/N
nP = (N+1)*(N+1);
nT = 2*N*N;

%   calculation of grid coordinates
x1koor = H*[0:N];
x2koor = H*[0:N];

%   tri, pkte: arrays for plot-routine "pdesurf"
%   (taken from Matlab Pde-Toolbox)
x1pkte = [];
x2pkte = [];
```

```
for mu = 1:N+1
    x1pkte = [x1pkte, x1koor];
    x2pkte = [x2pkte, x2koor(mu)*ones(1, N+1)];
end

pkte(1, 1:nP) = x1pkte;
pkte(2, 1:nP) = x2pkte;

corner1 = [];
corner2 = [];
corner3 = [];
for mu = 1:N
    corner1 = [corner1, (mu-1)*(N+1) + (1:N)];
    corner2 = [corner2, (mu-1)*(N+1) + (2:N+1)];
    corner3 = [corner3, mu*(N+1) + (2:N+1)];

    corner1 = [corner1, (mu-1)*(N+1) + (1:N)];
    corner2 = [corner2, mu*(N+1) + (2:N+1)];
    corner3 = [corner3, mu*(N+1) + (1:N)];
end

tri(1, 1:nT) = corner1;
tri(2, 1:nT) = corner2;
tri(3, 1:nT) = corner3;
tri(4, 1:nT) = 1:nT;


%%%%%%%%%%%%%%%%%%%%%%%%

t_dis(1, 1:nT) = 3*(1:nT)-2;
t_dis(2, 1:nT) = 3*(1:nT)-1;
t_dis(3, 1:nT) = 3*(1:nT);
t_dis(4, 1:nT) = (1:nT);


x_dis_low = [];
x_dis_up = [];
```

```
y_dis_low = [];
y_dis_up = [];
for i = 1:N
    x_dis_low = [x_dis_low, i*H, i*H, (i-1)*H];
    x_dis_up = [x_dis_up, (i-1)*H, (i-1)*H, i*H];

    y_dis_low = [y_dis_low, 0, H, 0];
    y_dis_up = [y_dis_up, H, 0, H];
end

p_disx = [];
p_disy = [];

for mu = 1:N
   p_disx = [p_disx, x_dis_low];
   p_disy = [p_disy, (mu-1)*H*ones(1, 3*N)+y_dis_low];

   p_disx = [p_disx, x_dis_up];
   p_disy = [p_disy, (mu-1)*H*ones(1, 3*N)+y_dis_up];
end

p_dis(1, 1:3*nT) = p_disx;
p_dis(2, 1:3*nT) = p_disy;


%%%%%%%%%%%%%%%%%%%%%%%%


trilow = [];
triup = [];
for j = 1:N
    trilow = [trilow, (1:N) + 2*N*(j-1)];
    triup  = [triup, (1:N) + 2*N*(j-1)+N];
end

%   calculation of edge centers of the triangulation
kantex1 = H*[-0.5, 0.0, 0.5, 0.5, 0.0, -0.5];
```

```
kantex2 = H*[-0.5, -0.5, 0.0, 0.5, 0.5, 0.0];
for i = 1:nP
   xe1(i, 1:6) = x1pkte(i) + kantex1;
   xe2(i, 1:6) = x2pkte(i) + kantex2;
end

for j = 1:N
    elem = (j-1)*2*N + 1;
    xte1(elem:elem+N-1, 1) = x1pkte(corner1(elem:elem+N-1))' + 0.5*H;
    xte2(elem:elem+N-1, 1) = x2pkte(corner1(elem:elem+N-1))';

    xte1(elem:elem+N-1, 2) = x1pkte(corner1(elem:elem+N-1))' + H;
    xte2(elem:elem+N-1, 2) = x2pkte(corner1(elem:elem+N-1))' + 0.5*H;

    xte1(elem:elem+N-1, 3) = x1pkte(corner1(elem:elem+N-1))' + 0.5*H;
    xte2(elem:elem+N-1, 3) = x2pkte(corner1(elem:elem+N-1))' + 0.5*H;

    elem = (j-1)*2*N + N+1;

    xte1(elem:elem+N-1, 1) = x1pkte(corner1(elem:elem+N-1))' + 0.5*H;
    xte2(elem:elem+N-1, 1) = x2pkte(corner1(elem:elem+N-1))' + H;

    xte1(elem:elem+N-1, 2) = x1pkte(corner1(elem:elem+N-1))';
    xte2(elem:elem+N-1, 2) = x2pkte(corner1(elem:elem+N-1))' + 0.5*H;

    xte1(elem:elem+N-1, 3) = x1pkte(corner1(elem:elem+N-1))' + 0.5*H;
    xte2(elem:elem+N-1, 3) = x2pkte(corner1(elem:elem+N-1))' + 0.5*H;
end

for j = 1:N
    elem = (j-1)*2*N + 1;
    xke1(elem:elem+N-1, 1) = x1pkte(corner1(elem:elem+N-1))' + 0.5*H;
    xke2(elem:elem+N-1, 1) = x2pkte(corner1(elem:elem+N-1))';

    xke1(elem:elem+N-1, 2) = x1pkte(corner1(elem:elem+N-1))' + H;
    xke2(elem:elem+N-1, 2) = x2pkte(corner1(elem:elem+N-1))' + 0.5*H;
```

```
    xke1(elem:elem+N-1, 3) = x1pkte(corner1(elem:elem+N-1))' + 0.5*H;
    xke2(elem:elem+N-1, 3) = x2pkte(corner1(elem:elem+N-1))' + 0.5*H;

    elem = (j-1)*2*N + N+1;
    xke1(elem:elem+N-1, 1) = x1pkte(corner1(elem:elem+N-1))';
    xke2(elem:elem+N-1, 1) = x2pkte(corner1(elem:elem+N-1))' + 0.5*H;

    xke1(elem:elem+N-1, 2) = x1pkte(corner1(elem:elem+N-1))' + 0.5*H;
    xke2(elem:elem+N-1, 2) = x2pkte(corner1(elem:elem+N-1))' + H;

    xke1(elem:elem+N-1, 3) = x1pkte(corner1(elem:elem+N-1))' + 0.5*H;
    xke2(elem:elem+N-1, 3) = x2pkte(corner1(elem:elem+N-1))' + 0.5*H;
end

%   calculation of the barycenters of the elements
xs1a = x1koor(2:N+1) - H/3.0;
xs2a = x2koor(1:N) + H/3.0;

xs1b = x1koor(1:N) + H/3.0;
xs2b = x2koor(2:N+1) - H/3.0;



%   Achtung: Der Rest wird nur beim POSTPROCESSING benoetigt!!!
%   Feineres Gitter fuer die L2-Norm-Bestimmung auf den Dreiecken mit
%   AM-Rand
%   Kantenmittelpunkte des feineren Gitters
hh = H/NN;

xhh1foo = [1:NN]'*hh - 0.5*hh;
xhh2foo = [1:NN]'*hh;

von = 1;
for i = 1:NN
    bis = von + NN-i;

    xhh1low(von:bis, 1) = xhh1foo(i:NN);
    xhh1low(von:bis, 2) = xhh2foo(i:NN);
```

```
    xhh1low(von:bis, 3) = xhh1foo(i:NN);

    xhh2low(von:bis, 1) = (i-1)*hh*ones(NN-i+1, 1);
    xhh2low(von:bis, 2) = (i-1)*hh*ones(NN-i+1, 1) + 0.5*hh;
    xhh2low(von:bis, 3) = (i-1)*hh*ones(NN-i+1, 1) + 0.5*hh;

    von = bis+1;
    bis = von + NN-i-1;

    xhh1low(von:bis, 1) = xhh2foo(i:NN-1);
    xhh1low(von:bis, 2) = xhh1foo(i+1:NN);
    xhh1low(von:bis, 3) = xhh1foo(i+1:NN);

   xhh2low(von:bis, 1) = i*hh*ones(NN-i, 1) - 0.5*hh;
   xhh2low(von:bis, 2) = i*hh*ones(NN-i, 1) - 0.5*hh;
   xhh2low(von:bis, 3) = i*hh*ones(NN-i, 1);

    von = bis +1;
end

xhh1up = xhh2low;
xhh2up = xhh1low;

fprintf('==== done ====\n');
```

createYd.m

```
fprintf(' \n==== Calculating the objective state function ====\n');

%   Zielfunktion y_d
yd = [];
faktor = 4.0*pi*pi*pi*pi*lambda +1.0;
for mu = 1:N+1
    yd = [yd; faktor*sin(pi*x2koor(mu)) * sin(pi*x1koor')];
end
```

```
%   Berechnung der analytischen Loesung fuer u, y, p
ufrei = [];
for j = 1:N
    ufrei = [ufrei; 2.0*pi*pi * sin(pi*xs2a(j)) * sin(pi*xs1a') ;
2.0*pi*pi * sin(pi*xs2b(j)) * sin(pi*xs1b')];
end

%   Projection on Uad
bool_opt = (ufrei > ub);
uopt4plot = ub*(bool_opt) + (~bool_opt .* ufrei);
bool_opt = (ufrei < ua);
uopt4plot = ua*(bool_opt) + (~bool_opt .* uopt4plot);

pexakt = [];
for mu = 1:N+1
    pexakt = [pexakt; -2.0*pi*pi*lambda*sin(pi*x2koor(mu)) * sin(pi*x1koor')];
end

yexakt = [];
for mu = 1:N+1
    yexakt = [yexakt; sin(pi*x2koor(mu)) * sin(pi*x1koor')];
end

fprintf('==== done ====\n');
```

stiffnessdiri.m

```
% ------------------------------------------
% --- BRECHNUNG DER STEIFIGKEITSMATRIX FUER DAS POISSON-PORBLEM ---
% ----------- (bei linearen Ansatzfunktionen) ------------
% ------------------------------------------
%
% ------------------------------------------
% AUFURF: von constrained.m aufgerufen
```

```
% - nP wird durch in gridinit generiert!
% ------------------------------------------
% Autor:  Christian Meyer
% Datum:  18.12.2002
% ------------------------------------------
%
% - Zeilen/Spalten der Dirichlet-Randpunkte werden zu Null gesetzt
%
% ---------------------------------------



% ------------------------------------------------------
fprintf('\n- Berechnung der Steifigkeitsmatrix fuer die linearen Ansatzfunktionen
-\n');
% ------------------------------------------------------

% --- Berechnung der Hauptdiagonalen ---
vals = ones(1,N+1);
for mu = 2:N
    vals = [vals, 1.0, 4*ones(1, N-1), 1.0];
end
vals = [vals, ones(1, N+1)];

%   Da D duenn besetzt ist, wird die Matrix im sparse-Format abgespeichert
D = sparse(1:nP, 1:nP, vals, nP, nP);

% --- Berechnung der ersten Nebendiagonalen ---
vals = zeros(1, N);
for mu = 2:N
    vals = [vals, 0.0, -ones(1, N-2), 0.0];
end
vals = [vals, zeros(1, N)];

ia1 = [2:N+1];
ia2 = ia1;
for mu = 2:N+1
    ia2 = [ia2, ia1+(mu-1)*(N+1)];
```

```
end
ia3 = ia2-1;

ND1 = sparse(ia2, ia3, vals, nP, nP);

% --- Berechnung der N-ten Nebendiagonalen ---
vals = zeros(1, N+1);
valfoo = [0.0, -ones(1, N-1), 0.0];
for mu = 2:N-1
    vals = [vals, valfoo];
end
vals = [vals, zeros(1, N+1)];

ia2 = [1:nP-(N+1)];
ia3 = ia2+N+1;

NDN= sparse(ia2, ia3, vals, nP, nP);

%   Diagonalmatrizen werden zur Steifigkeitsmatrix D zusammengesetzt
%   (Symmetrie!!!)
A = D+ND1+ND1'+NDN+NDN';

% ------------
fprintf('- done -\n');
% ------------
```

stiffnessneumann.m

```
% -------------------------------
% --- CALCULATION OF THE STIFFNESS MATRIX ---
% ------ FOR LINERAR ANSATZ FUNCTIONS-----
% --- WITH NEUMANN BOUNDARY CONDITIONS ---
% -------------------------------
%
% ---------------------------------
```

```
%  CALLED BY: constrained.m
%
%  NEEDED INPUT:
%       - N (from constrained.m)
%       - nP (from gridinit.m)
% --------------------------------
%  Author: Christian Meyer
%  Date: 06.02.2003
% --------------------------------


fprintf('\n==== Calculating the stiffness matrix ====\n');


% --- main diagonal ---
%   (entries in domain interior: 4, on the border: 2)
vals = [1, 2*ones(1, N-1), 1];
for kappa = 2:N
    vals = [vals, 2, 4*ones(1, N-1), 2];
end
vals = [vals, 1, 2*ones(1, N-1), 1];

%   Matrix stored in sparse-format
D = sparse(1:nP, 1:nP, vals, nP, nP);


% --- first subdiagonal ---
%   (entries in domein interior: -1)
vals = [-0.5*ones(1, N)];
vals = [vals, -ones(1, N*(N-1))];
vals = [vals, -0.5*ones(1, N)];

ia1 = [2:N+1];
ia2 = ia1;
for mu = 2:N+1
    ia2 = [ia2, ia1+(mu-1)*(N+1)];
end
```

```
ia3 = ia2-1;

ND1 = sparse(ia2, ia3, vals, nP, nP);

% --- N-th subdiagonal ---
%   (entries in domaininterior: -1)
valfoo = [-0.5, -ones(1, N-1), -0.5];
vals = [];
for mu = 1:N
    vals = [vals, valfoo];
end

ia2 = [1:nP-(N+1)];
ia3 = ia2+N+1;

NDN= sparse(ia2, ia3, vals, nP, nP);

%   Stiffness matrix is the sum of the diagonal matrices
%   and their respective transposed matrices
%   (because of the symmetry of the stiffness matrix)
S = D+ND1+ND1'+NDN+NDN';

fprintf('==== done ====\n');
```

massdiri.m

```
% ----------------------------------------
% --- BRECHNUNG DER MASSENMATRIX FUER DAS POISSON-PORBLEM ---
% -------- (bei linearen Ansatzfunktionen) -----------
% ----------------------------------------
%
% ----------------------------------------
% AUFURF: von laplace.m aufgerufen
% - nP und H werden durch in gridinit generiert!
```

```
% ----------------------------------------
% Autor:  Christian Meyer
% Datum:  18.12.2002
% ----------------------------------------
%
% - Multiplikation mit c in laplace.m, da die Massenmatrix
%   (ohne Faktor c) noch zur bestimmung der L2-Norm benoetigt wird
% - Zeilen/Spalten der Dirichlet-Randpunkte werden zu Null gesetzt
%
% ----------------------------------------


% ------------------------------------------------
fprintf('\n- Berechnung der Massenmatrix fuer die linearen Ansatzfunktionen -\n');
% ------------------------------------------------

%   Hauptdiagonale
m = H*H/12.0;
vals = zeros(1, N+1);
for mu = 2:N
    vals = [vals, 0.0, 6.0*m*ones(1, N-1), 0.0];
end
vals = [vals, zeros(1, N+1)];

%   Da D bzw. M duenn besetzt ist, wird die Matrix im sparse-Format abgespeichert
D = sparse(1:nP, 1:nP, vals, nP, nP);


% --- Berechnung der ersten Nebendiagonalen ---

vals = zeros(1, N);
for mu = 2:N
    vals = [vals, 0.0, m*ones(1, N-2), 0.0];
end
vals = [vals, zeros(1, N)];

ia1 = [2:N+1];
```

```
ia2 = ia1;
for mu = 2:N+1
    ia2 = [ia2, ia1+(mu-1)*(N+1)];
end
ia3 = ia2-1;
M = sparse(ia2, ia3, vals, nP, nP);

% --- Berechnung der N-ten Nebendiagonalen ---
vals = zeros(1, N+1);
valfoo = [0.0, m*ones(1, N-1), 0.0];
for mu = 2:N-1
    vals = [vals, valfoo];
end
vals = [vals, zeros(1, N+1)];

ia2 = [1:nP-(N+1)];
ia3 = ia2+N+1;

M1 = sparse(ia2, ia3, vals, nP, nP);
M = M+M1;


% --- Berechnung der N+1-ten Nebendiagonalen ---
vals = zeros(1, N);
valfoo = [0.0, m*ones(1, N-2), 0.0];
for mu = 2:N-1
    vals = [vals, valfoo];
end
vals = [vals, zeros(1, N)];

ia1 = [1:N];
ia2 = ia1;
for mu = 2:N
    ia2 = [ia2, ia1+(mu-1)*(N+1)];
end
ia3 = ia2+N+2;
```

```
M1 = sparse(ia2, ia3, vals, nP, nP);
M = M+M1;

%   Diagonalmatrizen werden Massenmatrix M zusammengesetzt
%   (Symmetrie!!!)
M = M+M'+D;
% ------------
fprintf('- done -\n');
% ------------
```

massneumann.m

```
% --------------------------------
% ---- CALCULATION OF THE MASS MATRIX ----
% ------ FOR LINERAR ANSATZ FUNCTIONS-----
% --- WITH NEUMANN BOUNDARY CONDITIONS ---
% --------------------------------
%
% ---------------------------------
%  CALLED BY: constrained.m
%
%  NEEDED INPUT:
%       - N (from constrained.m)
%       - H, nP (from gridinit.m)
% ---------------------------------
%  Author: Christian Meyer
%  Date: 06.02.2003
% ---------------------------------

fprintf('\n==== Calculating the mass matrix for linear ansatz functions====\n');

% --- main diagonal ---
%   (entries in domain interior: (H^2)/2, on the border: (H^2)/4)
```

```
m = H*H/12.0;
vals = [2.0*m, 3.0*m*ones(1, N-1), m];
for mu = 2:N
    vals = [vals, 3.0*m, 6.0*m*ones(1, N-1), 3.0*m];
end
vals = [vals, m, 3.0*m*ones(1, N-1), 2.0*m];

%   Matrix stored in sparse-format
D = sparse(1:nP, 1:nP, vals, nP, nP);

% --- first subdiagonal ---
%   (entries in domein interior: (H^2)/12)
vals = [0.5*m*ones(1, N)];
vals = [vals, m*ones(1, N*(N-1))];
vals = [vals, 0.5*m*ones(1, N)];

ia1 = [2:N+1];
ia2 = ia1;
for mu = 2:N+1
    ia2 = [ia2, ia1+(mu-1)*(N+1)];
end
ia3 = ia2-1;
M = sparse(ia2, ia3, vals, nP, nP);


% --- N-th subdiagonal ---
%   (entries in domaininterior: (H^2)/12)
valfoo = [0.5*m, m*ones(1, N-1), 0.5*m];
vals = [];
for mu = 1:N
    vals = [vals, valfoo];
end

ia2 = [1:nP-(N+1)];
ia3 = ia2+N+1;

M1 = sparse(ia2, ia3, vals, nP, nP);
```

```
M = M+M1;

% --- (N+1)-th subdiagonal ---
%   (entries in domaininterior: (H^2)/12)
valfoo = [m*ones(1, N)];
vals = [];
for mu = 1:N
    vals = [vals, valfoo];
end

ia1 = [1:N];
ia2 = ia1;
for mu = 2:N
    ia2 = [ia2, ia1+(mu-1)*(N+1)];
end
ia3 = ia2+N+2;

M1 = sparse(ia2, ia3, vals, nP, nP);
M = M+M1;


%   Stiffness matrix is the sum of the diagonal matrices
%   and their respective transposed matrices
%   (because of the symmetry of the stiffness matrix)
M = M+M'+D;

fprintf('==== done ====\n');
```

calcR.m

```
% -------------------------------------
% ----- CALCULATION OF THE TRANSFORMATION MATRIX ----
% ----- FOR DIRICHLET BOUNDARY CONDITIONS ---------
% -------------------------------------
%
```

```
% --------------------------------
%  CALLED BY: constrained.m
%
%  NEEDED INPUT:
%       - H, nT (from gridinit.m)
% --------------------------------
%  Author: Rene Simon
%  Date: 06.04.2004
% --------------------------------
fprintf('\n- Berechnung der (nP x 3nT)-Matrix R -\n');

index = [0,1,2, 3*N,3*N+1,3*N+2,3*(N+1),3*(N+1)+1,3*(N+1)+2, 6*N];
index = [index,6*N+1,6*N+2,6*N+3,6*N+4,6*N+5, 9*N+3,9*N+4,9*N+5];

spalte = [];
zeile = [];

for mu = 2:N
    for nu = 2:N
        i = (mu-1)*(N+1) + nu;
        j = (mu-2)*6*N + 3*nu-5;

        spalte = [spalte, index + j];
        zeile = [zeile, i*ones(1, 18)];
    end
end
vals=[];
for a = 1:(N-1)*(N-1)
    vals=[vals,1,2,1,1,1,2,2,1,1,2,1,1,1,1,2,1,2,1];
end;
vals=H^2/24*vals;

R = sparse(zeile, spalte,vals, nP, 3*nT);

fprintf('- done -\n');
```

calcRneumann.m

```
% -------------------------------
% --- CALCULATION OF THE (nP x 3nT)-MATRIX R ---
% -------------------------------
%
% ---------------------------------
%  CALLED BY: constrained.m
%
%  NEEDED INPUT:
%       - N (from constrained.m)
%       - H, nP, nT (from gridinit.m)
% ---------------------------------
%  Author: Rene Simon
%  Date: 04.04.2004
% ---------------------------------



fprintf('\n==== Calculating (nP x nT)-matrix R ====\n');

index = [0,1,2, 3*N,3*N+1,3*N+2,3*(N+1),3*(N+1)+1,3*(N+1)+2, 6*N];
index = [index,6*N+1,6*N+2,6*N+3,6*N+4,6*N+5, 9*N+3,9*N+4,9*N+5];

spalte=[1,2,3,3*(N+1)-2,3*(N+1)-1,3*(N+1)];
zeile = [1, 1, 1, 1, 1, 1];

for nu = 2:N
    zeile = [zeile, nu*ones(1,9)];
    spalte  = [spalte, 3*(nu-1)-2,3*(nu-1)-1, 3*(nu-1), 3*nu-2, 3*nu-1, 3*nu,
3*(nu+N)-2, 3*(nu+N)-1, 3*(nu+N)];
end

spalte = [spalte, 3*N-2,3*N-1,3*N];
zeile = [zeile, N+1,N+1,N+1];

for mu = 2:N
```

```
    i = (mu-1)*(N+1) + 1;
    j = (mu-2)*2*N +N+1;

    zeile = [zeile, i, i, i,i,i,i,i,i,i];
    spalte = [spalte, 3*j-2,3*j-1,3*j, 3*(j+N)-2, 3*(j+N)-1,3*(j+N),
3*(j+2*N)-2,3*(j+2*N)-1,3*(j+2*N)];

    for nu = 2:N

        i = (mu-1)*(N+1) + nu;
        j = (mu-2)*6*N + 3*nu-5;

        spalte = [spalte, index + j];
        zeile = [zeile, i*ones(1, 18)];
    end

   i = mu*(N+1);
    j = (mu-2)*2*N +N;

    zeile = [zeile, i, i, i,i,i,i,i,i,i];
    spalte = [spalte, 3*j-2,3*j-1,3*j, 3*(j+N)-2,3*(j+N)-1,3*(j+N),
3*(j+2*N)-2,3*(j+2*N)-1,3*(j+2*N)];
end

zeile = [zeile, N*(N+1)+1,N*(N+1)+1,N*(N+1)+1];
spalte = [spalte, 3*((N-1)*2*N+N+1)-2,3*((N-1)*2*N+N+1)-1,3*((N-1)*2*N+N+1)];

for nu = 2:N
    i = N*(N+1) + nu;
    j = (N-1)*2*N + nu-1;

    zeile = [zeile, i, i, i,i,i,i,i,i,i];
    spalte = [spalte, 3*j-2,3*j-1,3*j, 3*(j+N)-2,3*(j+N)-1,3*(j+N),
3*(j+N+1)-2,3*(j+N+1)-1, 3*(j+N+1)];
end

zeile = [zeile, nP, nP,nP,nP,nP,nP];
```

```
spalte = [spalte, 3*(nT-N)-2,3*(nT-N)-1, 3*(nT-N),3*nT-2,3*nT-1,3*nT];

%   entries are the integrals of the
%   linear ansatzfunctions over each triangle
vals=[];
vals=[vals,1,1,2,1,2,1];
for mu=2:N
    vals=[vals,2,1,1,1,1,2,1,2,1];
end;
vals=[vals,2,1,1];
for mu=2:N
    vals=[vals,2,1,1,1,1,2,1,2,1];
    for nu=2:N
        vals=[vals,1,2,1,1,1,2,2,1,1,2,1,1,1,1,2,1,2,1];
    end;
    vals=[vals,1,2,1,1,1,2,2,1,1];
end;
vals=[vals,2,1,1];
for nu=2:N
    vals=[vals,1,2,1,1,1,2,2,1,1];
end;
vals=[vals,1,2,1,1,1,2];
vals=H^2/24*vals;

%   R stored in sparse format
R = sparse(zeile, spalte, vals, nP, 3*nT);

fprintf('==== done ====\n');
```

calcMtilde.m

```
% ------------------------------
% ----- CALCULATION OF THE MASS MATRIX ----
% ----- FOR LINEAR ANSATZ FUNCTIONS-----
% --- WITH DIRICHLET BOUNDARY CONDITIONS ---
```

```
% --------------------------------
%
% --------------------------------
%  CALLED BY: constrained.m
%
%  NEEDED INPUT:
%       - H, nT (from gridinit.m)
% --------------------------------
%  Author: Rene Simon
%  Date: 06.04.2004
% --------------------------------

fprintf('\n==== Calculating the mass matrix for linear ansatz functions====\n');

vals1=H^2/12*ones(3*nT,1);
vals2=[];
vals3=[];
for a=1:nT
    vals2=[vals2,0,H^2/24,H^2/24];
    vals3=[vals3,0,0,H^2/24];
end;
D1=spdiags(vals1,0,3*nT,3*nT);      %Hauptdiagonale
D2=spdiags(vals2',1,3*nT,3*nT);     %1. Nebendiagonale
D3=spdiags(vals3',2,3*nT,3*nT);     %2. Nebendiagonale
Mtilde=D1+D2+D2'+D3+D3';

%Inverse
vals1=18/H^2*ones(3*nT,1);
vals2=[];
vals3=[];
for a=1:nT
    vals2=[vals2,0,-6/H^2,-6/H^2];
    vals3=[vals3,0,0,-6/H^2];
end;
D1=spdiags(vals1,0,3*nT,3*nT);      %Hauptdiagonale
D2=spdiags(vals2',1,3*nT,3*nT);     %1. Nebendiagonale
D3=spdiags(vals3',2,3*nT,3*nT);     %2. Nebendiagonale
```

```
MtildeInv=D1+D2+D2'+D3+D3';

fprintf('==== done ====\n');
```

calcaffin.m

```
% ---------------------------
% --- CALCULATION OF THE AFFINE TERM ---
% ---- IN THE OBJECTIVE FUNCTIONAL ----
% ---------------------------
%
% -------------------------------------
%  CALLED BY: constrained.m
%
%  NEEDED INPUT:
%    - N, c, ua, ub, A (from constrained.m)
%    - xe1, xe2, nP, H (from gridinit.m)
% -------------------------------------
%  Author: Christian Meyer
%  Date: 06.02.2003
% -------------------------------------
%  FOLLOWING FUNCTION IS USED:
%  - calcEo.m
% -------------------------------------
%
%  The affine term is needed to posses an analytical solution,
%  see "Dokumentation zum free- und constrained-Code", section 2.2
%
%  In calcEo.m the function eo is calculated, that appears
%  as a corrective term in the inhomogenity of the PDE.
%  In this function the integral of eo*ansatzfunction is
%  calculated as it appears in the variational formulation of
%  the PDE.
%  For this integration 2. order Gauss-Quadrature is used.
%
```

```
% ------------------------------------


fprintf('\n==== Calculating the affine term for the objective functional ====\n');

areaT = H*H/6.0;

%   Calculation of the corrective term in the inhomogenity of the PDE
calcEo;

%   Lower border (x2=0)
ko(1:N+1) = zeros(1, N+1);

%   2. Order Gauss-Quadrature

for mu = 2:N
    i = (mu-1)*(N+1) + 1;
    %   Left border (x1=0)
    ko(i) = 0.0;

    for nu = 2:N
        i = (mu-1)*(N+1) + nu;
        ko(i) = sum(eo(i, 1:6));
    end

   i = mu*(N+1);
    %   Right border (x1=1)
    ko(i) = 0.0;
end

%   Upper border
i  = N*(N+1) + 1;
ko(i:nP) = zeros(1, N+1);

ko = areaT* ko';

%   cg-Method to calculate A^-1 ko
```

```
%   (compare "Dokumentation zum free- und constrained-Code" eq.(71))
fprintf('    Solving A^-1 ko using the Matlab-pcg-function\n');
%   Preconditioner:
%   Diagonalentries of A
MV = sparse(1:nP, 1:nP, diag(A), nP, nP);

%   Calling the Matlab-Routine
fprintf('Output of the Matlab-pcg-function:\n');
affin = pcg(A, ko, tiny, maxit, MV);

fprintf('\n==== done ====\n');
```

calcEo.m

```
% ------------------------------------
% --- CALCULATION OF THE CORRECTIVE TERM THE PDE ---
% ------------------------------------
%
% ------------------------------------
%  CALLED BY: calcaffin.m (<- called by constrained1.m)
%
%  NEEDED INPUT:
%    - c, ua, ub (from constrained.m)
%    - xe1, xe2, nP (from gridinit.m)
% ------------------------------------
%  Author: Christian Meyer
%  Date: 06.02.2003
% ------------------------------------
%
%  The corrective term is needed to posses an analytical solution,
%  see "Dokumentation zum free- und constrained-Code", eq.(66)
%
% ------------------------------------

fprintf('\n    == Calculating the corrective term in the PDE ==\n');
```

```
uf = 2*pi*pi * sin(pi*xe1) .* sin(pi*xe2);
eo = zeros(nP, 6);

%   Projection on Uad
bool_uf = (uf > ub);
eo = eo + (bool_uf .* uf) - ub * bool_uf;
bool_uf = (uf < ua);
eo = eo + (bool_uf .* uf) - ua * bool_uf;

fprintf('    == done ==\n');
```

calcb.m

```
% ----------------------------
% --- CALCULATION OF THE INHOMGENITY b ---
% ----------------------------
%
% ------------------------------------
%  CALLED BY: constrained.m
%
%  NEEDED INPUT:
%    - M,  A, R, tiny, maxit, zd, Mtilde,
%       uAktiv, lambda, aktiv (from constrained.m)
%    - nP (from gridinit.m)
% ------------------------------------
%  Author: Christian Meyer
%  Date: 06.02.2003
% ------------------------------------
%  FOLLOWING FUNCTION IS USED:
%  - operatorS.m
%  - operatorStrans.m
% ------------------------------------
%
%  Calculation of the inhomogenity
%  b = S* yd - (lambda Mtilde + S*S)uAktiv
```

```
%  (see "Dokumentation zum free- und constrained-Code", eq.(57))
%
% ------------------------------------
```

```
fprintf('\n    == Calculating the inhomogenity b ==\n');
term1 = operatorStrans(M*zd, A, R, tiny, maxit, nP);
zAktiv = operatorS(uAktiv, A, R, tiny, maxit, nP);
term2 = M * zAktiv;
term2 = operatorStrans(term2, A, R, tiny, maxit, nP);
term2 = term2 + lambda * (Mtilde*uAktiv);
b = term1 - term2;
%   Attention: the aktive parts of b have to be zero
%   (see "Dokumentation zum free- und constrained-Code", p.15/16)
b(aktiv) = 0.0;
fprintf('    == done ==\n');
```

Bmulti.m

```
% ----------------------------------------
% --- EVALUATION OF THE OPERATOR B = lambda Mtilde + S^T M S ---
% ----------------------------------------
%
% ----------------------------------------
%  CALLED BY:
%       - constrained.m
%
%  INTERFACE: prod = Bmulti(u, A, R, M, Mtilde, lambda, tiny, maxit, nP,
%  aktiv)
%     INPUTPARAMETERS:
%           u - vector, which is multiplied by B
%           A, R - constituents of S = A^-1 R and S^T
%           M, Mtilde - mass matrices
%           tiny - tolerance for cg-method
```

```
%           maxit - maximum number of iterations for cg-method
%           nP - number of degrees of freedom
%           lambda - regulisation parameter in the objective functional
%           aktiv - set of aktiv controls (A+ and A-)
% ----------------------------------------
%  Author: Christian Meyer
%  Date: 06.02.2003
% ----------------------------------------
%  FOLLOWING FUNCTIONS ARE USED:
%  - operatorS.m
%  - operatorStrans.m
% ----------------------------------------
%  See "Dokumentation zum free- und constrained-Code", eq.(16)
% ----------------------------------------
function prod = Bmulti(u, A, R, M, Mtilde, lambda, tiny, maxit, nP, aktiv)

fprintf('    - Evaluating of operator B = lambda Mtilde + S^T M S -\n');

%   Evaluating S u
prod = operatorS(u, A, R, tiny, maxit, nP);
prod = M*prod;

%   Evaluating S^T prod
prod = operatorStrans(prod, A, R, tiny, maxit, nP);
prod = prod + lambda * (Mtilde*u);

%   Attention: the aktive parts of b have to be zero
%   (see "Dokumentation zum free- und constrained-Code", p.15/16)
prod(aktiv) = 0.0;
fprintf('    - done -\n');
```

operatorS.m

```
% ----------------------------------
```

```
% --- EVALUATION OF THE PDE-OPERATOR S = A^-1 R ---
% -----------------------------------
%
% ---------------------------------
%  CALLED BY:
%       - constrained.m
%       - Bmulti.m
%       - calcb.m
%
%  INTERFACE: sol = operatorS(u, A, R, tiny, maxit, nP)
%
%     INPUTPARAMETERS:
%           u - vector, which is multiplied by S
%           A, R - constituents of S = A^-1 R
%           tiny - tolerance for cg-method
%           maxit - maximum number of iterations for cg-method
%           nP - number of degrees of freedom
%
% ---------------------------------
%  Author: Christian Meyer
%  Date: 06.02.2003
% ---------------------------------
%
%  Compare "Dokumentation zum free- und constrained-Code", section1.2
%
% ---------------------------------

function sol = operatorS(u, A, R, tiny, maxit, nP)

% ---------------------------------
fprintf('     - Evaluation of operator S = A^-1 R -\n');
% ---------------------------------

inh = R*u;

fprintf('      Solving A*sol = R*u with Matlab-pcg-function\n');
```

```
%   Preconditioner:
%   Diagonalentries of A
MV = sparse(1:nP, 1:nP, diag(A), nP, nP);

%   Calling Matlab-pcg-function
fprintf('output of the Matlab-pcg-function:\n');
sol = pcg(A, inh, tiny, maxit, MV);

% ------------
fprintf('     - done -\n');
% ------------
```

operatorStrans.m

```
% --------------------------------------
% --- EVALUATION OF THE PDE-OPERATOR S^T = R^T A^-1 ---
% --------------------------------------
%
% --------------------------------------
%  CALLED BY:
%       - constrained1.m
%       - Bmulti.m
%       - calcb.m
%
%  INTERFACE: sol = operatorS(u, A, R, tiny, maxit, nP)
%
%     INPUTPARAMETERS:
%           u - vector, which is multiplied by S
%           A, R - constituents of S = A^-1 R
%           tiny - tolerance for cg-method
%           maxit - maximum number of iterations for cg-method
%           nP - number of degrees of freedom
%
% ------------------------------------
%  Author: Christian Meyer
```

```
%  Date: 06.02.2003
% ------------------------------------
%
%  Compare "Dokumentation zum free- und constrained-Code", section1.2
%
% ------------------------------------

function sol = operatorStrans(y1, A, R, tiny, maxit, nP)

% ---------------------------------
fprintf('     - Evaluation of operator S^T = R^T A^-1 -\n');
% ---------------------------------

fprintf('      Solving R^T A*sol = inh with Matlab-pcg-function\n');

%   Preconditioner:
%   Diagonalentries of A
MV = sparse(1:nP, 1:nP, diag(A), nP, nP);

%   Calling Matlab-pcg-function
fprintf('output of the Matlab-pcg-function:\n');
y2 = pcg(A, y1, tiny, maxit, MV);


sol = R'*y2;

% ------------
fprintf('     - done -\n');
% ------------
```

calcAdjoint.m

```
fprintf('\n    == Calculating the adjoint state p = S*(z-zd) ==\n');

MV = sparse(1:nP, 1:nP, diag(A), nP, nP);
```

```
%   Calling Matlab-pcg-function
fprintf('output of the Matlab-pcg-function:\n');
pP = pcg(A, M*(z-zd), tiny, maxit, MV);
p = MtildeInv * operatorStrans(M*(z-zd), A, R, tiny, maxit, nP);
fprintf('    == done ==\n');
```

findTris.m

```
elems = [1:nT]';

values1 = 2.0*pi*pi * sin(pi*x1pkte(corner1)') .* sin(pi*x2pkte(corner1)');
values2 = 2.0*pi*pi * sin(pi*x1pkte(corner2)') .* sin(pi*x2pkte(corner2)');
values3 = 2.0*pi*pi * sin(pi*x1pkte(corner3)') .* sin(pi*x2pkte(corner3)');


bool_e1 = (values1 < ua & values2 < ua & values3 < ua);
A_a_analyt = elems(bool_e1);

bool_e2 = (proj(corner1) < ua & proj(corner2) < ua & proj(corner3) < ua);
A_a_num = elems(bool_e2);

bool_ea = (bool_e1 & bool_e2);
A_a = elems(bool_ea);


bool_e1 = (values1 > ub & values2 > ub & values3 > ub);
A_b_analyt = elems(bool_e1);

bool_e2 = (proj(corner1) > ub & proj(corner2) > ub & proj(corner3) > ub);
A_b_num = elems(bool_e2);

bool_eb = (bool_e1 & bool_e2);
A_b = elems(bool_eb);
```

```
bool_e1 = (values1 > ua & values2 > ua & values3 > ua & values1 < ub & values2
< ub & values3 < ub);
I_analyt = elems(bool_e1);

bool_e2 = (proj(corner1) > ua & proj(corner2) > ua & proj(corner3) > ua & proj(corner1)
< ub & proj(corner2) < ub & proj(corner3) < ub);
I_num = elems(bool_e2);

bool_ei = (bool_e1 & bool_e2);
I = elems(bool_ei);

bool_Ilow = (x2pkte(corner1(I)) == x2pkte(corner2(I)));
Ilow = I(bool_Ilow);
bool_Iup = (x1pkte(corner1(I)) == x1pkte(corner3(I)));
Iup = I(bool_Iup);



bool_rand = ∼(bool_ea | bool_eb | bool_ei);
AMRand = elems(bool_rand);

bool_AMlow = (x2pkte(corner1(AMRand)) == x2pkte(corner2(AMRand)));
AMRandlow = AMRand(bool_AMlow);
bool_AMup = (x1pkte(corner1(AMRand)) == x1pkte(corner3(AMRand)));
AMRandup = AMRand(bool_AMup);
```

L2norm.m

```
% -------------------------------------
% ----- CALCULATION OF THE L2-NORM OF U_h-U -------
% -------------------------------------
%
% -------------------------------------
%  CALLED BY: constrained.m
```

```
%
%  NEEDED INPUT:
%       - ua, ub (from constrained.m)
%       - nT (from gridinit.m)
% ---------------------------------
%  Author: Rene Simon
%  Date: 06.04.2004
% ---------------------------------


uf4norm = 2*pi*pi * sin(pi*xte1) .* sin(pi*xte2);

uopt4norm = uf4norm;

%   Projection on Uad
bool_uf = (uf4norm > ub);
uopt4norm = ub*(bool_uf) + (~bool_uf .* uf4norm);

bool_uf = (uf4norm < ua);
uopt4norm = ua*(bool_uf) + (~bool_uf .* uopt4norm);

uoptquadr = uopt4norm(1:nT, 1) .* uopt4norm(1:nT, 1);
uoptquadr = uoptquadr + uopt4norm(1:nT, 2) .* uopt4norm(1:nT, 2);
uoptquadr = H*H/6.0 * (uoptquadr + uopt4norm(1:nT, 3) .* uopt4norm(1:nT, 3));

uoptu=0;
uquadr=0;
for i=1:nT
    uoptu=uoptu+uopt4norm(i,1)*1/2*(u(3*i-2)+
u(3*i))+uopt4norm(i,2)*1/2*(u(3*i-2)+
u(3*i-1))+uopt4norm(i,3)*1/2*(u(3*i-1)+ u(3*i));
    uquadr=uquadr+1/4*((u(3*i-2)+
u(3*i))^2+(u(3*i-2)+
u(3*i-1))^2+(u(3*i-1)+u(3*i))^2);
end;

L2konst = sqrt(sum(uoptquadr) - H^2/3*uoptu + H^2/6*uquadr);
```

projL2norm.m

```
sizelow = size(AMRandlow);
sizelow = sizelow(1);
sizeup = size(AMRandup);
sizeup = sizeup(1);

x1AMR = [];
x2AMR = [];
uhAMR = [];

for j = 1:sizelow
    ecke1 = corner1(AMRandlow(j));
    ecke2 = corner2(AMRandlow(j));
    ecke3 = corner3(AMRandlow(j));

    koeff1 = ( proj(ecke1) - proj(ecke2) ) / H;
    koeff2 = ( proj(ecke2) - proj(ecke3) ) / H;
    koeff3 = proj(ecke1);

    x1AMRfoo = x1pkte(ecke1) + xhh1low;
    x2AMRfoo = x2pkte(ecke1) + xhh2low;

    x1AMR = [x1AMR; x1AMRfoo];
    x2AMR = [x2AMR; x2AMRfoo];

    uhAMR = [uhAMR; koeff1*(x1pkte(ecke1) - x1AMRfoo) + koeff2*(x2pkte(ecke1) - x2AMRfoo)
+ koeff3];
end

for j = 1:sizeup
    ecke1 = corner1(AMRandup(j));
    ecke2 = corner2(AMRandup(j));
    ecke3 = corner3(AMRandup(j));
```

```
    koeff1 = ( proj(ecke3) - proj(ecke2) ) / H;
    koeff2 = ( proj(ecke1) - proj(ecke3) ) / H;
    koeff3 = proj(ecke1);

    x1AMRfoo = x1pkte(ecke1) + xhh1up;
    x2AMRfoo = x2pkte(ecke1) + xhh2up;

    x1AMR = [x1AMR; x1AMRfoo];
    x2AMR = [x2AMR; x2AMRfoo];

    uhAMR = [uhAMR; koeff1*(x1pkte(ecke1) - x1AMRfoo) + koeff2*(x2pkte(ecke1) - x2AMRfoo)
+ koeff3];
end


%Projection
bool_hAMRa = (uhAMR < ua);
uhAMR(bool_hAMRa) = ua;
bool_hAMRb = (uhAMR > ub);
uhAMR(bool_hAMRb) = ub;



uoptAMR = 2.0*pi*pi * sin(pi*x1AMR) .* sin(pi*x2AMR);
%   Projection
bool_optAMRa = (uoptAMR < ua);
uoptAMR(bool_optAMRa) = ua;
bool_optAMRb = (uoptAMR > ub);
uoptAMR(bool_optAMRb) = ub;


%   Gauss-Quadratur
deltaAMR = uoptAMR - uhAMR;
deltaAMR = deltaAMR .* deltaAMR;

intAMR = deltaAMR(:, 1) + deltaAMR(:, 2) + deltaAMR(:, 3);
```

```
intAMR = (hh*hh)/6.0 * intAMR;


L2proj = sum(intAMR);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Auf A_a und A_b ist |uopt - uh| = 0, da  uopt = uh (= ua oder ub)
%   => L2-Norm muss nur noch auf I berechnet werden!!!

sizeIlow = size(Ilow);
sizeIlow = sizeIlow(1);
sizeIup = size(Iup);
sizeIup = sizeIup(1);

uhIlow = ones(sizeIlow, 3);
uhIup = ones(sizeIup, 3);

uhIlow(:, 1) = 0.5 * ( proj(corner1(Ilow)) + proj(corner2(Ilow)) );
uhIlow(:, 2) = 0.5 * ( proj(corner2(Ilow)) + proj(corner3(Ilow)) );
uhIlow(:, 3) = 0.5 * ( proj(corner1(Ilow)) + proj(corner3(Ilow)) );

uhIup(:, 1) = 0.5 * ( proj(corner1(Iup)) + proj(corner3(Iup)) );
uhIup(:, 2) = 0.5 * ( proj(corner2(Iup)) + proj(corner3(Iup)) );
uhIup(:, 3) = 0.5 * ( proj(corner1(Iup)) + proj(corner2(Iup)) );

uoptIlow = 2.0*pi*pi * sin(pi*xke1(Ilow, :)).* sin(pi*xke2(Ilow, :));
uoptIup  = 2.0*pi*pi * sin(pi*xke1(Iup, :)).* sin(pi*xke2(Iup, :));


deltaI = uoptIlow - uhIlow;
deltaI = [deltaI; uoptIup - uhIup];
deltaI = deltaI .* deltaI;

intI = deltaI(:, 1) + deltaI(:, 2) + deltaI(:, 3);
intI = (H*H)/6.0 * intI;
```

```
L2proj = L2proj + sum(intI);
L2proj = sqrt(L2proj);
```

projLinfnorm.m

```
sizelow = size(AMRandlow);
sizelow = sizelow(1);
sizeup = size(AMRandup);
sizeup = sizeup(1);

x1AMR = [];
x2AMR = [];
uhAMR = [];
uconstAMR = [];


for j = 1:sizelow
    ecke1 = corner1(AMRandlow(j));
    ecke2 = corner2(AMRandlow(j));
    ecke3 = corner3(AMRandlow(j));

    koeff1 = ( proj(ecke1) - proj(ecke2) ) / H;
    koeff2 = ( proj(ecke2) - proj(ecke3) ) / H;
    koeff3 = proj(ecke1);

    x1AMRfoo = x1pkte(ecke1) + xhh1low;
    x2AMRfoo = x2pkte(ecke1) + xhh2low;

    x1AMR = [x1AMR; x1AMRfoo];
    x2AMR = [x2AMR; x2AMRfoo];

    uhAMR = [uhAMR; koeff1*(x1pkte(ecke1) - x1AMRfoo) + koeff2*(x2pkte(ecke1) - x2AMRfoo)
+ koeff3];
    uconstAMR = [uconstAMR; u(AMRandlow(j)) * ones(size(xhh1low))];
end
```

```
for j = 1:sizeup
    ecke1 = corner1(AMRandup(j));
    ecke2 = corner2(AMRandup(j));
    ecke3 = corner3(AMRandup(j));

    koeff1 = ( proj(ecke3) - proj(ecke2) ) / H;
    koeff2 = ( proj(ecke1) - proj(ecke3) ) / H;
    koeff3 = proj(ecke1);

    x1AMRfoo = x1pkte(ecke1) + xhh1up;
    x2AMRfoo = x2pkte(ecke1) + xhh2up;

    x1AMR = [x1AMR; x1AMRfoo];
    x2AMR = [x2AMR; x2AMRfoo];

    uhAMR = [uhAMR; koeff1*(x1pkte(ecke1) - x1AMRfoo) + koeff2*(x2pkte(ecke1) - x2AMRfoo)
+ koeff3];
    uconstAMR = [uconstAMR; u(AMRandup(j)) * ones(size(xhh1up))];
end

%Projection
bool_hAMRa = (uhAMR < ua);
uhAMR(bool_hAMRa) = ua;
bool_hAMRb = (uhAMR > ub);
uhAMR(bool_hAMRb) = ub;

uoptAMR = 2.0*pi*pi * sin(pi*x1AMR) .* sin(pi*x2AMR);
%   Projection
bool_optAMRa = (uoptAMR < ua);
uoptAMR(bool_optAMRa) = ua;
bool_optAMRb = (uoptAMR > ub);
uoptAMR(bool_optAMRb) = ub;

dAMR = uoptAMR - uhAMR;
dAMR = [dAMR(:, 1); dAMR(:, 2); dAMR(:, 3)];
LinfAMR = norm(dAMR, inf);
```

```
dcAMR = uoptAMR - uconstAMR;
dcAMR = [dcAMR(:, 1); dcAMR(:, 2); dcAMR(:, 3)];
LinfcAMR = norm(dcAMR, inf);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Auf A_a und A_b ist |uopt - uh| = 0, da  uopt = uh (= ua oder ub)
%   => Linf-Norm muss nur noch auf I berechnet werden!!!


uhIlow = proj(corner1(Ilow));
uhIlow = [uhIlow; proj(corner2(Ilow))];
uhIlow = [uhIlow; proj(corner3(Ilow))];

koeff1 = ( proj(corner1(Ilow)) - proj(corner2(Ilow)) ) / H;
koeff2 = ( proj(corner2(Ilow)) - proj(corner3(Ilow)) ) / H;
koeff3 = proj(corner1(Ilow));

x1midlow = x1pkte(corner1(Ilow)) + 2.0*(H/3.0);
x2midlow = x2pkte(corner1(Ilow)) + H/3.0;

uhIlow = [uhIlow; koeff3 - koeff1* 2.0*(H/3.0) - koeff2* H/3.0];


uhIup = proj(corner1(Iup));
uhIup = [uhIup; proj(corner2(Iup))];
uhIup = [uhIup; proj(corner3(Iup))];

koeff1 = ( proj(corner3(Iup)) - proj(corner2(Iup)) ) / H;
koeff2 = ( proj(corner1(Iup)) - proj(corner3(Iup)) ) / H;
koeff3 = proj(corner1(Iup));

x1midup = x1pkte(corner1(Iup)) + H/3.0;
x2midup = x2pkte(corner1(Iup)) + 2.0*(H/3.0);
```

```
uhIup = [uhIup; koeff3 - koeff1* (H/3.0) - koeff2* 2.0*(H/3.0)];


uoptIlow = 2.0*pi*pi * sin(pi*x1pkte(corner1(Ilow))').* sin(pi*x2pkte(corner1(Ilow))');
uoptIlow = [uoptIlow; 2.0*pi*pi * sin(pi*x1pkte(corner2(Ilow))').*
sin(pi*x2pkte(corner2(Ilow))')];
uoptIlow = [uoptIlow; 2.0*pi*pi * sin(pi*x1pkte(corner3(Ilow))').*
sin(pi*x2pkte(corner3(Ilow))')];
uoptIlow = [uoptIlow; 2.0*pi*pi * sin(pi*x1midlow').*
sin(pi*x2midlow')];

uoptIup = 2.0*pi*pi * sin(pi*x1pkte(corner1(Iup))').* sin(pi*x2pkte(corner1(Iup))');
uoptIup = [uoptIup; 2.0*pi*pi * sin(pi*x1pkte(corner2(Iup))').*
sin(pi*x2pkte(corner2(Iup))')];
uoptIup = [uoptIup; 2.0*pi*pi * sin(pi*x1pkte(corner3(Iup))').*
sin(pi*x2pkte(corner3(Iup))')];
uoptIup = [uoptIup; 2.0*pi*pi * sin(pi*x1midup').*
sin(pi*x2midup')];

dcIlow = 2.0*pi*pi * sin(pi*x1pkte(corner1(Ilow))').*
sin(pi*x2pkte(corner1(Ilow))') - u(Ilow);
dcIlow = [dcIlow; 2.0*pi*pi * sin(pi*x1pkte(corner2(Ilow))').*
sin(pi*x2pkte(corner2(Ilow))') - u(Ilow)];
dcIlow = [dcIlow; 2.0*pi*pi * sin(pi*x1pkte(corner3(Ilow))').*
sin(pi*x2pkte(corner3(Ilow))') - u(Ilow)];
dcIlow = [dcIlow; 2.0*pi*pi * sin(pi*x1midlow').*
sin(pi*x2midlow') - u(Ilow)];

dcIup = 2.0*pi*pi * sin(pi*x1pkte(corner1(Iup))').* sin(pi*x2pkte(corner1(Iup))')
- u(Iup);
dcIup = [dcIup; 2.0*pi*pi * sin(pi*x1pkte(corner2(Iup))').* sin(pi*x2pkte(corner2(Iup))')
- u(Iup)];
dcIup = [dcIup; 2.0*pi*pi * sin(pi*x1pkte(corner3(Iup))').* sin(pi*x2pkte(corner3(Iup))')
- u(Iup)];
dcIup = [dcIup; 2.0*pi*pi * sin(pi*x1midup').* sin(pi*x2midup')- u(Iup)];

dI = [uoptIlow - uhIlow; uoptIup - uhIup];
```

```
dcI = [dcIlow; dcIup];

LinfI = norm(dI, inf);

LinfcI = norm(dcI, inf);

L_inf = max(LinfI, LinfAMR);
Linfconst = max(LinfcI, LinfcAMR);
```

findMaxStep.m

```
% ------------------------------------
% ----- FIND THE MAXIMAL DISCONTINOUS STEP --------
% ------------------------------------
%
% ---------------------------------
%  CALLED BY: constrained.m
%
%  NEEDED INPUT:
%       - N (from gridinit.m)
% ---------------------------------
%  Author: Rene Simon
%  Date: 06.04.2004
% ---------------------------------


maximal=0;
max_index=0;
for mu=2:N
    for nu=2:N
        i=(mu-1)*(N+1)+nu;
        j=(mu-2)*6*N+3*nu-5;
        vec=[u(1+j),u(3*N+2+j), u(3*(N+1)+j), u(6*N+j), u(6*N+5+j), u(9*N+4+j)];

        if ((max(vec)-min(vec))>maximal)
```

```
            max_index=i;
            maximal=max(vec)-min(vec);
        end;
    end;
end;
```

Literaturverzeichnis

[1] C. Meyer, A. Rösch, *Superconvergence properties of optimal control problems*, Preprint 017-2003, Institut für Mathematik, Technische Universität Berlin, 2003.

[2] C. Meyer, *Dokumentation zum free- und constrained-Code*.

[3] A. Rösch, *Error estimates for linear-quadratic problems with control constraints*, Preprint 740-2002, Institut für Mathematik, Technische Universität Berlin, 2002. .

[4] E. Casas, F. Tröltzsch, *Error estimates for linear-quadratic, elliptic control problems* in: Analysis and Optimization of Differential Systems, V. B. et al, ed., Boston, 2003, Kluwer Academic Publishers, 89–100.

[5] P. Grisvard, *Elliptic problems in nonsmooth domains*, Pitman, Boston-London-Melbourne, 1985.

[6] P. Ciarlet, *Basic error estimates for elliptic problems*, in Finite Element Methods, vol. II des *Handbook of Numerical Analysis*, North-Holland, 1991, 17–352.

[7] K. Malanowski, *Convergence of approximations vs. regularity of solutions for convex, control-constrained optimal control problems*, Appl.Math.Opt., 8 (1981), 69–95.

[8] D. Braess, *Finite Elemente*, Springer Verlag, Berlin Heidelberg, 1992.

[9] J. Bonnans, E. Casas, *An extension to Pontryagin's principle for state constrained optimal control of semilinear elliptic equation and variational inequalities*, SIAM J. Control and Optimization, 33 (1985), 274–298.

[10] F. Tröltzsch, *Optimalsteuerung bei partiellen Differentialgleichungen*, Vorlesungsskript, TU Berlin.

[11] L. C. Evans, *Partial Differential Equations*, Graduate Studies in Mathematics Vol. 19, American Mathematical Society, Providence, Rhode Island, 1998.

[12] P. Raviart, J. Thomas, *Introduction à l'Analyse Numérique des Equations aux Dérivées Partielles*, Masson, Paris, 1992.

[13] J.L. Lions, *Optimal Control of Systems Governed by Partial Differential Equations*, Springer Verlag, Berlin Heidelberg New York, 1971.

[14] A. Rösch, F. Tröltzsch, *Partielle Differentialgleichungen*, Vorlesungsskript, TU Berlin

Danksagung

Herrn Dr. A. Rösch danke ich für die Überlassung des interessanten Themas und die sehr gute Betreuung während der Anfertigung dieser Arbeit.

C. Meyer danke ich für die hilfreiche Begleitung meiner Arbeit und für die vielen guten Ideen und Denkanstöße, insbesondere bei der programmiertechnischen Umsetzung.

Ich danke meinen Eltern, ohne deren Unterstützung diese Arbeit nicht zustandegekommen wäre.

Persönliche Erklärung

Die selbständige und eigenhändige Anfertigung versichere ich an Eides statt.

Berlin, den 8. Juli 2004 René Simon