FORSCHUNGSZENTRUM JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik
D-52425 Jülich, Tel. (02461) 61-6402

Interner Bericht

Finite Elemente in Scilab: Das Lösen partieller Differentialgleichungen mit Hilfe der FreeFEM-Toolbox

Rainer von Seggern

FZJ-ZAM-IB-2001-03

April 2001

(letzte Änderung: 24.04.2001)

Inhaltsverzeichnis

1 Einleitung **1**

2 Installation der FreeFEM-Toolbox von Scilab **2**
2.1 FREEFEM+ und FreeFEM in Scilab 2
2.2 Die FreeFEM-Programme in Scilab 3
2.3 Zusätzliche Programme 3

3 Die Sprachelemente von FreeFEM **4**
3.1 Datentypen 4
3.2 Typenlisten 4
3.3 Ausführen von FreeFEM-Befehlen 8
3.4 Partielle Differentialgleichungen in FreeFEM 8
3.5 Ergebnisdarstellung 10
3.6 Sichern und Wiederverwenden von Daten in FreeFEM 12

4 Systeme partieller Differentialgleichungen **13**
4.1 Ein elliptisches System mit zwei Gleichungen 13
4.2 Die schwache Formulierung eines Systems von zwei Gleichungen 15
4.3 Ein Beispiel aus der Elastizitätstheorie 16

5 Das Schwarzsche Verfahren **20**

6 Gitteradaption **23**

7 Instationäre Probleme **26**

8 Nichtlineare partielle Differentialgleichungen **30**

Kapitel 1

Einleitung

Für das wissenschaftliche Rechnen werden interaktive Programmierumgebungen immer wichtiger, mit deren Hilfe nahezu unabhängig von der Platform Probleme gelöst, Algorithmen entwickelt und getestet, sowie Demonstrationen zu Unterrichtszwecken entwickelt und vorgeführt werden können. Scilab ist eines dieser Produkte, das ähnlich wie MATLAB insbesondere für numerische Zwecke entwickelt worden ist. Scilab ist frei im Internet bei INRIA in Frankreich erhältlich. (Siehe [2]).
In Scilab ist eine große Zahl von Algorithmen der Numerik entweder direkt Teil der Sprache oder sie stehen als Toolbox zur Verfügung. Die auf der homepage von Scilab angebotene Finite-Elemente-Toolbox FreeFEM zum Lösen von zweidimensionalen partiellen Differentialgleichungen wurde im ZAM unter LINUX installiert und kann seit Anfang des Jahres benutzt werden. In diesem Bericht wird diese Toolbox vorgestellt und ihre Benutzung an Hand unterschiedlicher Beispiele beschrieben. Die Auswahl dieser Beispiele wurde durch die Fähigkeit von FreeFEM bestimmt, konkrete Probleme besonders einfach und übersichtlich mit Hilfe der Methode der Finiten Elemente lösen und die Ergebnisse dann für weitere Rechnungen benutzen zu können. Im Vordergrund standen dabei:

1. das Modellieren mit partiellen Differentialgleichungen
2. das Entwickeln von Testbeispielen für größere FE-Pakete
3. die Demonstration der Methode der Finiten Elemente zu Unterrichtszwecken.

Kapitel 2

Installation der FreeFEM-Toolbox von Scilab

2.1 FREEFEM+ und FreeFEM in Scilab

FREEFEM+ ist eine benutzerfreundliche in C++ geschriebene Software zur Lösung von Systemen partieller Differentialgleichungen (PDGl) in zwei Dimensionen. Die Funktionalität dieses Programms wird in dem FREEFEM+-Handbuch [1] beschrieben. FREEFEM+ läßt sich unabhängig von Scilab benutzen, wobei allerdings die Nachbehandlung der Ergebnisse kaum möglich ist. Dieses Programm ist unter **`/usr/local/bin/freefem+`** installiert und kann mit

```
FreeFem Test
```

für jedes gültige FREEFEM+-Programm **`Test`** aufgerufen werden. Neuere Versionen dieses Programms findet man auf der homepage von Frédéric Hecht, der einer der Autoren ist, unter
`http://www-rocq.inria.fr/Frederic.Hecht/FreeFemPlus.htm`.
Hier liegt auch ein neueres Freefem+-Tutorial als pdf-Datei. (F. Hecht und O. Pironneau vom 28.11.00.) Die FreeFEM-Toolbox von Scilab ist eine Implementation von FREEFEM+ in Scilab und befindet sich auf der homepage von Scilab unter
`http://www-rocq.inria.fr/scilab/contributions.html`.
Für die in diesem Bericht beschriebene Version der FreeFEM-Toolbox wurden verschiedene Änderungen und Erweiterungen durchgeführt, die in den nächsten Abschnitten beschrieben werden. Um die FreeFEM-Toolbox installieren zu können, waren kleinere Änderungen der C++-Programme erforderlich, die nach umfangreicher Fehlersuche durchgeführt werden konnten. Diese Arbeiten und die gesamte Installation wurden von Herrn Niehoff erledigt, der von Herrn Egerer unterstützt wurde. Es handelte sich um folgende Änderungen in zwei Programmen aus dem Verzeichnis
`/usr/local/scilab/FREEFEM/src`:

```
1. In scilink.h:
Zeile 5 alt: #define C2F(a) a##_
Zeile 5 neu: #define C2F(a) a##__
2. In Mesh2.cpp:
Zeile 1095 und 1118 alt: tt1 = ToSwap ? tt1 : Next(tt2);
Zeile 1095 und 1118 neu: if (!ToSwap) tt1 = Next(tt2);
```

Durch Eingabe des Befehls

```
exec('/usr/local/bin/loader.sce',0);
```

in Scilab steht die Toolbox zur Verfügung. Insbesondere sind durch diesen Aufruf auch alle Hilfen zu dieser Toolbox im Scilab help-Fenster zugreifbar.

2.2 Die FreeFEM-Programme in Scilab

In der Datei **/usr/local/scilab/FREEFEM/man/whatis** sind alle Namen der Scilab-Programme der FreeFEM-Toolbox einschließlich einer kurzen Beschreibung aufgeführt. Folgende Programme wurden hinzugefügt:

- **ShowContour ShowLine ShowMesh ShowResult**

Es gibt zwei wesentlich verschiedene Funtionstypen in dieser Toolbox. Der erste Typ bezeichnet Funktionen, die in Scilab auf FreeFEM-Variable wirken. Ihre Wirkungsweise wird in 3.2 beschrieben. Im einzelnen sind dies die folgenden Funktionen:

- **defvar derive div grad laplace**
- **pde_sol pde_varsol rot tr valf**

(Eine Ausnahme ist die Funktion **valf**, die eine reine Scilab-Funktion ist und zur Erzeugung von Randpunkten dient.) Vom zweiten Typ sind Funktionen, die FreeFEM-Befehle absetzen. Hierbei handelt es sich um folgende Funktionen:

- **buildMesh ff_adaptmesh ff_end ff_exec ff_problem ff_var**
- **getffResult getMatrix meshvisu resultvisu**
- **ShowContour ShowLine ShowMesh ShowResult UpdateMesh**

Wird im unteren Teil des help-Fensters von Scilab **FreeFem: Finite element PDE solver** angeklickt, so erscheint im oberen Teil eine Tabelle der FreeFEM-Programme, die mit Hilfe der oben erwähnten whatis-Datei erzeugt wird. Nach Anklicken der einzelnen Programme erscheint die Hilfe in einem extra Fenster. Das gleiche bewirkt ein entsprechender **help**-Befehl in Scilab.

2.3 Zusätzliche Programme

Sehr häufig möchte man Ausdrücke in FreeFEM berechnen und das Ergebnis nach Scilab importieren. Dazu dient die folgende Funktion, die man mit **getf Import** laden muß.

```
function y=Import(Ausdruck)
ff_exec('append(""/tmp/UU"",'+string(Ausdruck)+')');
y=read('/tmp/UU',-1,1);
unix('rm /tmp/UU');
```

Programm 2.1: Import von FreeFEM nach Scilab

Beispielsweise kann man die mittlere quadratische Abweichung der beiden Gitterfunktionen **u** und **uex** auf dem aktiven Gitter in Scilab folgendermaßen berechnen:

```
getf Import
y=Import('sqrt(int()((u-uex)^2))')
```

In spätere Versionen der FreeFEM-Toolbox soll die Funktion **Import** integriert werden.

Kapitel 3

Die Sprachelemente von FreeFEM

3.1 Datentypen

In FREEFEM+ gibt es drei Datentypen:

```
number array function
```

Diese Datentypen werden in FreeFEM durch entsprechende Befehle definiert:

```
ff_var('a',...)
ff_exec('femp0(G) u=...'); Gitterfunktionen der Ordnung p=0
ff_exec('femp1(G) u=...'); Gitterfunktionen der Ordnung p=1
ff_exec('function f(x,y)=...)
```

Die beiden Gitterfunktionen sind durch Werte in den Schwerpunkten der Dreiecke des Gitters **G** beziehungsweise in den Knoten definiert. Beide Funktionen sind für beliebige Werte der beiden Gittervariablen von **G** definiert, wobei die Unstetigkeit der Gitterfunktionen der Ordnung $p = 0$ auf den Rändern der Dreiecke des Gitters zu beachten ist. Innerhalb der Dreiecke des Gitters **G** wird ihr Wert durch entsprechende Interpolation berechnet. Außerhalb des Gitters **G** wird bei Gitterfunktionen der Ordnung $p = 1$ der Wert des nächsten Randpunktes benutzt, während Gitterfunktionen der Ordnung $p = 0$ außerhalb des Gitters **G** Null gesetzt werden. (Siehe [1], 2.1.1 Interpolation.) Gitterfunktionen der Ordnung $p = 1$ sind auf der ganzen Ebene stetig. Innerhalb des Gitters **G** entsprechen die Gitterfunktionen also den Finiten Elementen der Ordnung $p = 0$ und $p = 1$. Um mit diesen Gitterfunktionen in Scilab rechnen zu können, werden formale FreeFEM-Variable eingeführt, die in Scilab durch Typenlisten dargestellt werden. (Siehe 3.2). Rechnungen mit FreeFEM-Variablen in Scilab sind symbolisch und dienen primär zur Formulierung der partiellen Differentialgleichungen in Form von Zeichenketten, die in Lösungsfunktionen eingesetzt werden. Einzelheiten werden in den nächsten Abschnitten beschrieben. (Siehe insbesondere 3.4). Für Gitterfunktionen der Ordnung $p = 1$ lassen sich die Funktionswerte in den Knoten des Gitters **G** mit Hilfe der FreeFEM-Funktion **getffResult** nach Scilab importieren. Gitterfunktionen der Ordnung $p = 0$ sind in den Knoten und auf den Rändern der Dreiecke im allgemeinen unstetig; als Funktionswert in den Knoten wird das arithmetische Mittel der Funktionswerte in den umgebenen Dreiecken zurückgegeben. Außerdem können alle diese Werte mit dem Befehl **ff_exec('save(...)')** gesichert werden.

3.2 Typenlisten

In Scilab gibt es neben den üblichen Listen den Datentyp **tlist**. Solche Typenlisten haben als erstes Argument einen Typenvektor, der ihren Namen und die Namen ihrer Elemente enthält. Die FreeFEM-Toolbox benutzt zwei Typenlisten mit den Namen **ffeq** für FreeFEM-Variable und **border** für die Definition von Randpunkten des Gitters. Folgendes Beispiel zeigt das Rechnen

mit FreeFEM-Variablen. Verschiedene ein- und zweistellige Operationen mit diesen Variablen wurden durch Operatorerweiterung (overloading) definiert. Insbesondere wurde die display-Funktion für FreeFEM-Variablen entsprechend verändert.

```
--> u=defvar(['u1';'u2']) // Scilabeingabe
FreeFEM-variable:         // Scilabausgabe
! u1 !
! u2 ! // Spaltenvektor
--> u*u
FreeFEM-variable:
u1*u1+u2*u2 // Skalarprodukt
--> div(u)
FreeFEM-variable:
dx(u1)+dy(u2) // Divergenz
--> grad(u)
FreeFEM-variable:
! dx(u1) dy(u1) !
! dx(u2) dy(u2) ! // Jacobimatrix
```

Beispiel 3.1: Das symbolische Rechnen mit FreeFEM-Variablen in Scilab.

Im nächsten Beispiel wird demonstriert, wie man in FreeFEM

- Stückweise definierte Ränder einführt
- Innere Ränder benutzt
- Gitterfunktionen einführt und
- ihre Werte in den Knoten ausgibt.

In einem Gebiet Ω wird ein Dreiecksgitter **G** eingeführt, dessen Eckpunkte auf dem Rand von Ω liegen; dabei soll der Rand von **G** so durchlaufen werden, daß **G** links vom Rand liegt. Die Ränder von Ω und **G** stimmen im allgemeinen nicht überein. Die mit Hilfe der Funktion **valf** berechneten Randpunkte von **G** können auf beliebigen Kurvenstücken liegen, die sich nicht schneiden. Alle Randstücke zusammen müssen einen geschlossene Rand bilden. Innere Ränder müssen in Knotenpunkten (auch inneren !) beginnen und enden und dürfen sich nicht schneiden. Sie werden von Dreieckskanten des Gitters gebildet und dienen hauptsächlich dazu, das Gebiet Ω so aufzuteilen, daß Gitterfunktionen der Ordnung $p = 0$ eingeführt werden können, die nur längs dieser inneren Ränder unstetig sind.
Bei der Konstruktion der Gitter ist unter anderem darauf zu achten, daß nicht alle Dreiecke zwei oder sogar drei Eckpunkte auf dem äußerem Rand haben.

```
m=3;
deff('[x,y]=Rf(t)','x=cos(t); y=sin(t);');
Rand = tlist(['border';'K1';'xA';'K2'],...
             list(valf(Rf,linspace(0,%pi,m)),1),...
             list('x=t; y=0;',-1,1,m,1),...       // Innerer Rand
             list(valf(Rf,linspace(%pi,2*%pi,m)),1));
buildMesh(Rand,'G');
ff_exec('femp0(G) h0=x+y'); ff_exec('femp1(G) h1=x+y');
[Knoten,Dreiecke,h0s]=getffResult('h0'); h1s=getffResult('h1');
  xset('window',0);
  xset('wdim',400,400);
  xbasc(0);
ShowMesh(Knoten,Dreiecke,1);
xarc(-1,1,2,2,0,360*64);
xtitle('Das Gitter G');
  xset('window',1);
  xset('wdim',400,400);
  xbasc(1);
ShowLine('h0',[-1;-1],[1;1],100,1);
xtitle('ShowLine(''h0'',[-1;-1],[1;1],100,1);');
  xset('window',2);
  xset('wdim',400,400);
  xbasc(2);
ShowLine('h1',[-1;-1],[1;1],100,1);
xtitle('ShowLine(''h1'',[-1;-1],[1;1],100,1);');
disp([h0s h1s]); // Die arithmetischen Mittel
                 // bzw. Funktionswerte in den Knoten
```

Beispiel 3.2: Gitterfunktionen der Ordnung $p = 0$ und $p = 1$.

Im obigen Beispiel ist Ω der Einheitskreis in dem ein einfaches Dreiecksgitter **G** eingeführt wird. Der mit xA bezeichnete x-Achsenabschnitt bildet einen inneren Rand, so daß die x-Achse also kein Dreieck von **G** schneiden kann.

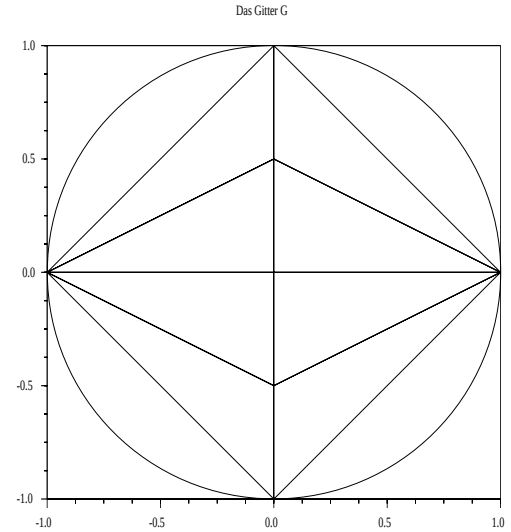

Abbildung 3.1: Die Triangulierung von Ω

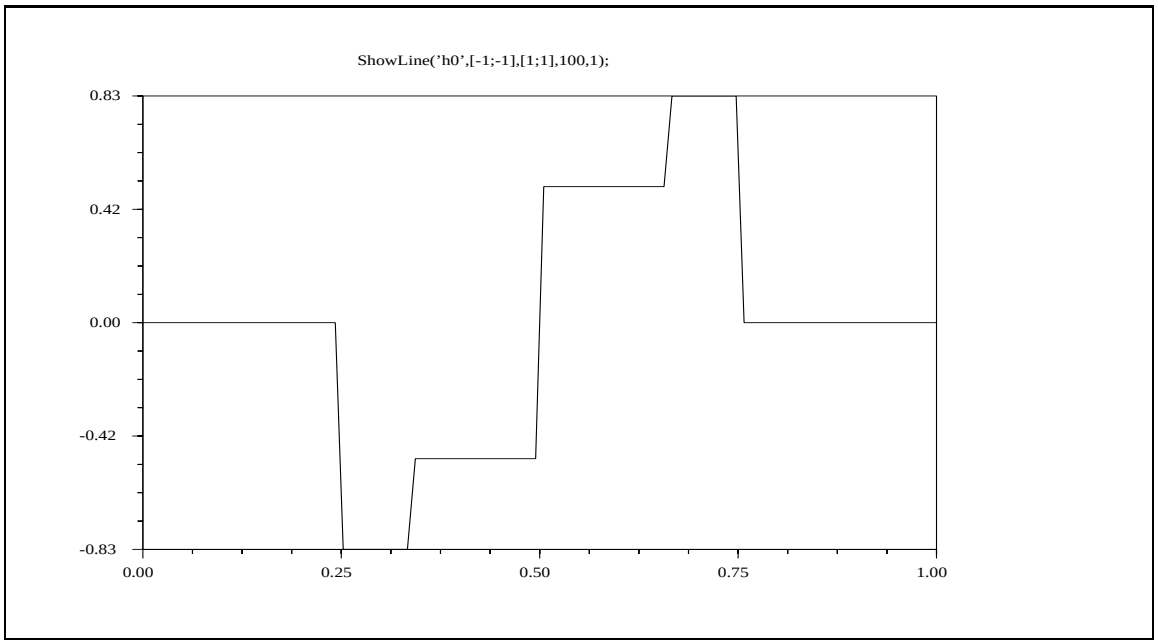

Abbildung 3.2: Die Gitterfunktion der Ordnung $p = 0$

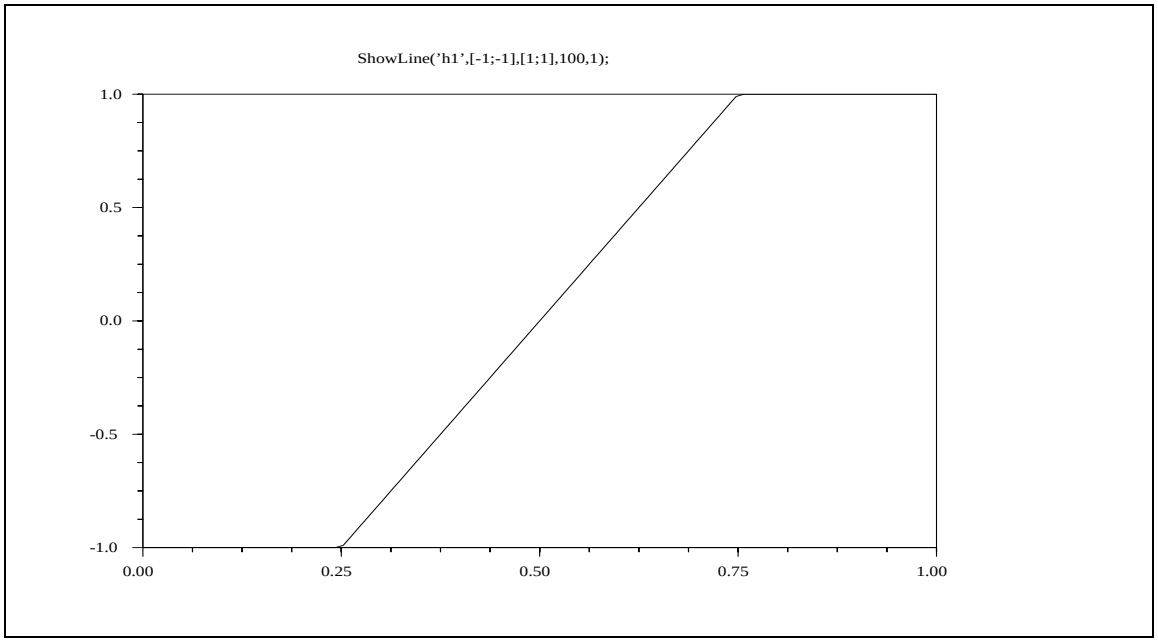

Abbildung 3.3: Die Gitterfunktion der Ordnung $p = 1$

In der obigen Abbildung 3.2 wird der Graph der Gitterfunktion $h_0(x, y)$ der Ordnung $p = 0$ längs der Winkelhalbierenden des ersten und dritten Quadranten dargestellt. Die Funktionswerte in den Dreiecken sind durch die Werte der definierenden Funktion in ihren Schwerpunkten definiert. Die Funktionswerte der Gitterfunktionen der Ordnung $p = 1$ dagegen werden in den Dreiecken durch lineare Interpolation aus den Funktionswerten der definierenden Funktion in den Eckpunkten gewonnen.
Man beachte auch die unterschiedliche Fortsetzung der beiden Gitterfunktionen außerhalb des definierenden Gitters.

3.3 Ausführen von FreeFEM-Befehlen

In Scilab werden FreeFEM-Befehle unter anderem durch **`ff_exec(Text)`** und **`ff_problem(Textmatrix,b)`** abgesetzt. Dabei ist **`Text`** eine Zeichenkette, die einen zuläßigen FreeFEM-Befehl enthält und **`Textmatrix`** eine aus zuläßigen FreeFEM-Befehlen gebildete Zeichenkettenmatrix, die mit Hilfe der FreeFEM Funktionen **`pde_sol`** und **`pde_varsol`** gebildet werden kann. **`b`** ist eine Boolsche Variable, deren Wert keinen Einfluß zu haben scheint - man kann sie auch weglassen.
Eine andere Möglichkeit FreeFEM-Ausdrücke in Scilab mit Hilfe der Funktiom **`Import`** zu berechnen, wurde schon in 2.3 beschrieben.

3.4 Partielle Differentialgleichungen in FreeFEM

In FreeFEM können partielle Differentialgleichungen (PDGl) zweiter Ordnung in der Ebene mit den verschiedenen Randbedingungen gelöst werden. In diesem Abschnitt wird die Formulierung der allgemeinen linearen partiellen Differentialgleichungen zweiter Ordnung in FreeFEM beschrieben. In späteren Kapiteln werden auch instationäre und nichtlineare PDGl zweiter Ordnung mit zwei Ortsvariablen behandelt. Die folgenden Gleichungen zeigen die allgemeine lineare PDGl zweiter Ordnung in der Ebene sowie deren kompaktere Darstellung mit Hilfe der Operatoren **`grad`** und **`div`**. (Wie in FreeFEM ist **`grad(u)`** für eine skalare Funktion $u(x,y)$ ein Spaltenvektor.)

$$L(u) := \alpha(x,y)u(x,y) + \sum_{i=1}^{2} v_i(x,y)D_i(u) - \sum_{i=1}^{2}\sum_{j=1}^{2} D_i(K_{ij}(x,y)D_j(u)) = f(x,y)$$

$$L(u) = \alpha(x,y)u(x,y) + grad(u)^T\vec{v}(x,y) - div(K(x,y)grad(u)) = f(x,y)$$

In der zweiten Formel bezeichnet $\vec{v}(x,y)$ ein gegebenes Vektorfeld mit den Komponenten $v_i(x,y)$ und $K(x,y)$ eine gegebene Matrix mit den Elementen $K_{ij}(x,y)$. Die Differentialgleichung ist elliptisch, wenn die Matrix $K(x,y)$ in Ω positiv definit ist. Auf die Eigenschaften der Koeffizientenfunktionen, die bei den verschiedenen Randbedingungen die eindeutige Existenz von Lösungen $u(x,y)$ sicherstellen und deren Eigenschaften soll hier nicht eingegangen werden.
Die nächste Gleichung zeigt die zugehörige schwache Formulierungen der Differentialgleichung mit geeigneten Testfunktionen $w(x,y)$, die insbesondere auf dem Rand $\partial\Omega$ des Definitionsgebietes Ω verschwinden, für homogene Dirichlet-Randbedingungen. (Siehe [3], Kapitel 6 und [4], Kapitel 1-3, 6.)

$$\int_\Omega \left\{ \left(\alpha(x,y)u(x,y) + grad(u)^T\vec{v}(x,y) \right) w(x,y) + grad(u)^T K(x,y) grad(w) \right\} dxdy$$

$$= \int_\Omega f(x,y)w(x,y)dxdy \quad \forall\, w(x,y)$$

Um die oben eingeführten Differentialgleichungen in FreeFEM mit geeigneten Randbedingungen zu lösen, kann man wie in 3.2 beschrieben formale FreeFEM-Variable einführen. Die Vorgehensweise wird im folgenden Programm im einzelnen beschrieben. Zu beachten ist dabei, daß **grad(u)** in FreeFEM ein Spaltenvektor ist und daß in den FreeFEM-Funktionen **pde_sol** und **pde_varsol** eine bestimmte Reihenfolge der Funktionen zu beachten ist. Beispielsweise

```
pde_sol(u,u*al+grad(u)*v-div(grad(u)'*K),f)   und nicht
pde_sol(u,al*u+v*grad(u)-div(K*grad(u)),f).
```

```
m=10;
getf Import;
Rand = tlist(['border';'a';'b';'c';'d'],...
list('x = t; y = 0',0,1,m,1),...       //a
list('x = 1; y = t',0,1,m,1),...       //b
list('x = 1 - t; y = 1',0,1,m,1),...   //c
list('x = 0; y = 1 - t',0,1,m,1));     //d
buildMesh(Rand,'G');

ff_exec('femp1(G) uex=exp(x+y)');
ff_exec('femp0(G) K0=exp(x^2+y^2)');
ff_exec('function f(x,y)=-2*(x+y+1)*exp(x^2+y^2+x+y)+...
                         (x*y+x+y)*exp(x+y)');
ff_exec('function v1(x,y)=y');ff_exec('function v2(x,y)=x');
ff_exec('function al(x,y)=x*y');

u=defvar('u');uex=defvar('uex');w=defvar('w');
f=defvar('f(x,y)');
K=defvar(['K0' '0';'0' 'K0']);al=defvar('al(x,y)');
v=defvar(['v1(x,y)';'v2(x,y)']);

ff_problem(['solve(G,u) {',...
            pde_sol(u,u*al+grad(u)*v-div(grad(u)'*K),f),...
            'on(a,b,c,d) u=uex;};'],%t);
Import('sqrt(int(G)((u-uex)^2))')  // Fehlernorm

ff_problem(['varsolve(G,0) A(u,w) with {',...
   pde_varsol('A','G',u*al*w+grad(u)*v*w+grad(u)'*K*grad(w)-f*w),...
            '+on(a,b,c,d)(w)(u=uex)};'],%t);
Import('sqrt(int(G)((u-uex)^2))')  // Fehlernorm
```

Programm 3.1: Allgemeine lineare partielle Differentialgleichung der Ordnung 2.

Bemerkungen:

1. Ist K eine skalare FreeFEM-Variable, so kann statt $div(grad(u) * K)$ auch $laplace(u) * K$ benutzt werden, obwohl beide Ausdrücke nur für konstante K identisch sind. Bei der Formulierung der PDGl ist also Vorsicht geboten.
2. Die Funktion $dnu(u)$ ist in FreeFEM die konormale Ableitung von u in Richtung der äußeren Normalen $\vec{n}$. (Siehe [3], Kapitel 6). Diese Tatsache ist insbesondere bei Problemen mit unstetigen Materialeigenschaften wichtig, weil so z.B. die Stetigkeit des Wärmeflusses beim Überschreiten innerer Ränder in FreeFEM automatisch erfüllt wird.

3.5 Ergebnisdarstellung

Zur Ergebnisdarstellung stehen in FreeFEM folgende Funktionen zur Verfügung:

- **`getffResult`**
- **`ShowContour`**
- **`ShowLine`**
- **`ShowMesh`** $\sim$ **`meshvisu`**
- **`ShowResult`** $\sim$ **`resultvisu`**

Die Programme **`Show...`** wurden neu aufgenommen. Die Programme **`ShowMesh`** und **`ShowResult`** sind modifizierte Formen von **`meshvisu`** und **`resultvisu`**, in denen veränderte Plotbefehle und keine globalen Variablen verwendet werden.
Im folgenden Programm werden verschiedene Anwendungen dieser Programme durchgeführt. Hier wird insbesondere das Programm **`ShowResult`** vorgestellt, während in den Beispiele meistens Höhenlinienplots gezeigt werden, die man mit **`ShowContour`** erhält. Die beiden Programme **`ShowResult`** und **`resultvisu`** sind insbesonde für Temperaturdarstellungen geeignet, da sie die Funktionswerte mit Hilfe von Farbabstufungen darstellen, die bei der schwarz-weiß Wiedergabe zu Grauabstufungen werden.

```
m=21;
Rand = tlist(['border';'K1';'K2';'xA'],...
list('x = cos(t); y = sin(t)',0,%pi,m,1),...
list('x = cos(t); y = sin(t)',%pi,2*%pi,m,1),...
list('x = t; y = 0;',-1,1,m,1));
buildMesh(Rand,'K');
ff_exec('femp1(K) u=x^2+y^2');
ff_exec('femp0(K) c=(y>0)*y^2+(y<0)*(1-y^2)');
[Knoten,Dreiecke,us]=getffResult('K,u');
cs=getffResult('K,c');
  xset('wdim',500,500);
  xsetech([0,0,1/2,1/2]);
ShowMesh(Knoten,Dreiecke,1);
  xsetech([1/2,0,1/2,1/2]);
ShowLine('c',[0;-1],[0;1],100,1)
  xset('colormap',hotcolormap(254)); // Temperaturfarben:
                                     // weiss/hellgelb=heiss
                                     // bis dunkelrot/schwarz=kalt
  xsetech([0,1/2,1/2,1/2]);
ShowResult(Knoten,Dreiecke,us);
  xsetech([1/2,1/2,1/2,1/2]);
ShowResult(Knoten,Dreiecke,cs);
```

Programm 3.2: Verschiedene Ergebnisdarstellungen

In den folgenden Ergebnisdarstellungen ist insbesondere die Darstellung der unstetigen Gitterfunktion $c(x, y)$ bemerkenswert. Es wird ein Schnitt durch den Graphen dieser Funktion längs der y-Achse und Darstellungen der Funktionen $u(x, y)$ und $c(x, y)$ mit Hilfe von Farbabstufungen gezeigt, die hier als Grauabstufungen erscheinen.

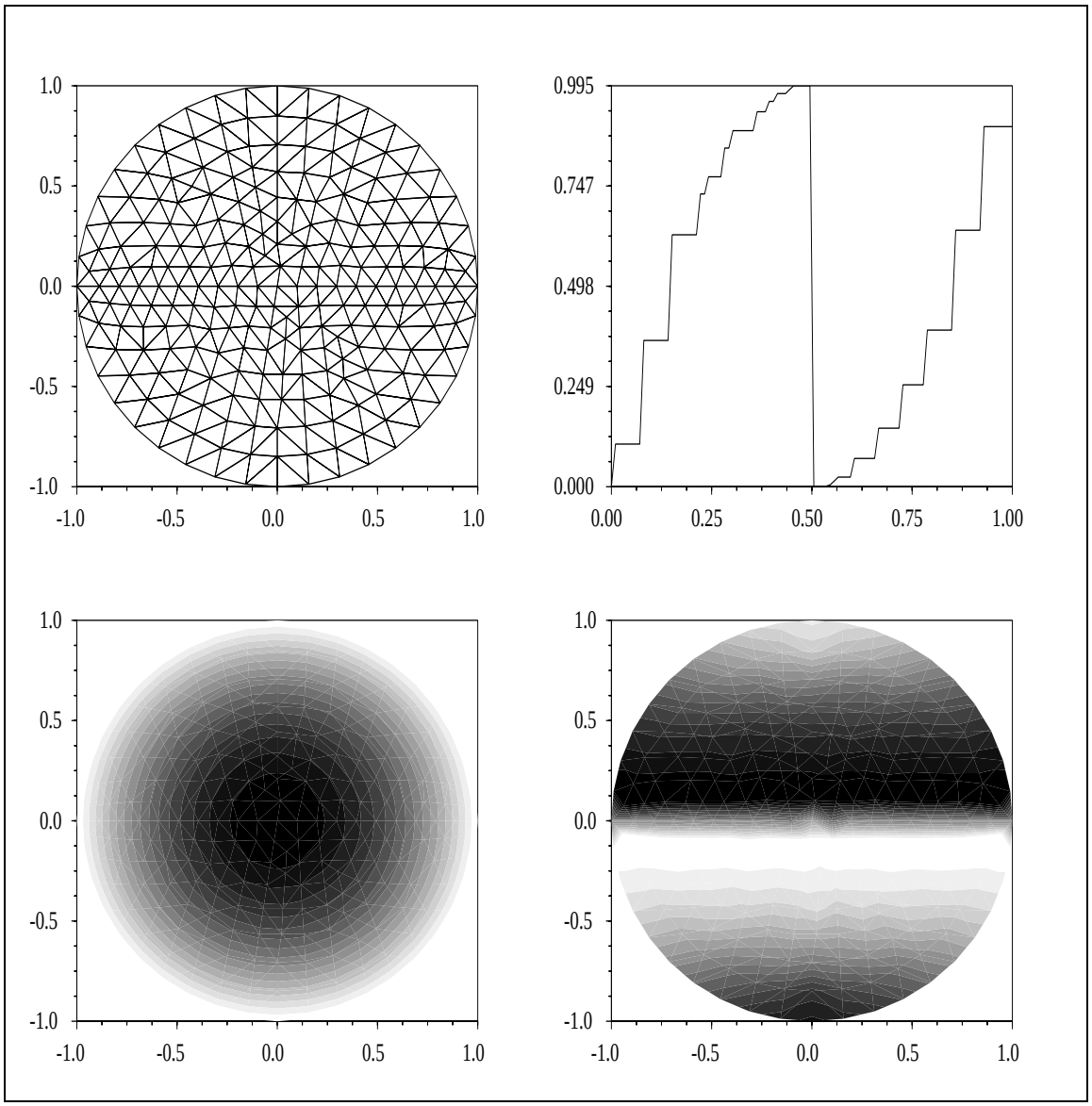

Abbildung 3.4: Verschiedene Ergebnisdarstellungen

3.6 Sichern und Wiederverwenden von Daten in FreeFEM

Im Handbuch [1], Abschnitt 8.3 wird beschrieben, wie man in FREEFEM+ insbesondere das einmal erzeugte Gitter speichern und in anderen Anwendungen benutzen kann. In FreeFEM ist das Abspeichern mit dem Befehl **`ff_exec('savemesh(Dateiname,Name)')`** zwar möglich, das Einlesen mit **`ff_exec('mesh Name=readmesh(Dateiname)')`** führt jedoch zum Absturz von Scilab. (Für alle in [1], Abschnitt 8.3 beschriebenen Suffixe. Dabei muß **`Dateiname`** in der Form `""Dateiname""` eingegeben werden.) Während derselben Scilabsitzung ist es jedoch möglich, in einem getrennten Programm berechnete Gitter und FreeFEM-Variable in anderen Programmen mit denselben Namen zu benutzen. Natürlich dürfen in diesen Programmen nicht andere Objekte mit diesen Namen bezeichnet werden. Dieses ist insbesondere wichtig, wenn man Parameterstudien durchführen will, die sich auf identische Gitter beziehen, da jeder Aufruf **`buildMesh(Rand,'Name')`** ein verändertes Gitter ergibt. Beim neuen Aufruf eines Programms mit unverändertem Rand kann man **`buildMesh(Rand,'Name')`** auskommentieren, um auf identischen Gittern Vergleichsrechnungen durchführen zu können.
Diese Vorgehensweise zeigen folgende kleine Programme:

```
m=20;
getf Import
Rand = tlist(['border';'K1';'K2';'xA'],...
list('x = cos(t); y = sin(t)',0,%pi,m,1),...
list('x = cos(t); y = sin(t)',%pi,2*%pi,m,1),...
list('x = t; y = 0;',-1,1,m,1));
buildMesh(Rand,'K');
ff_exec('femp1(K) f=exp(x+y)')
al=1;ff_var('al',string(al));
ff_problem(['solve(K,u) pde(u) -laplace(u)*al=1; ';...
            'on(K1,K2) u=0;']);
```

Programm 3.3: 1. Programm

```
Import('f(0,1)')             // Altes f
ff_exec('femp1(K) f=x^2+y^2')
Import('f(0,1)')             // Neues f
al=2;ff_var('al',string(al));
ff_problem(['solve(K,v) pde(v) -laplace(v)*al=1; ';...
            'on(K1,K2) v=0;']);
Import('u(0,0)')             // Loesung fuer al=1
Import('v(0,0)')             // Loesung fuer al=2
```

Programm 3.4: 2. Programm

Kapitel 4

Systeme partieller Differentialgleichungen

4.1 Ein elliptisches System mit zwei Gleichungen

In diesem Abschnitt wird ein inhomogenes System von zwei linearen partiellen Differentialgleichungen betrachtet, wobei die Inhomogenität mit Hilfe der vorgegebenen Lösungsfunktionen berechnet wurde. Auf diese Weise läßt sich die Vorgehensweise zum Lösen des Systems in FreeFEM übersichtlich beschreiben. Außerdem erhält der Benutzer einen Eindruck von der Approximationsgüte der benutzten Finiten Elemente erster Ordnung. Das betrachtete System hat folgende Gestalt:

$$\begin{aligned} \triangle u + u_{xx} + v_{xy} &= RS_1(x,y) \\ \triangle v + v_{yy} + u_{xy} &= RS_2(x,y)\ , \end{aligned}$$

wobei die beiden rechten Seiten RS_1 und RS_2 so gewählt wurden, daß

$$u(x,y) = v(x,y) = x(1-x)y(1-y)$$

Lösungen sind. Randbedingungen müssen natürlich hiermit verträglich gewählt werden.
Im folgenden wird das System im Einheitsquadrat mit homogenen Dirichlet-Randbedingungen gelöst.

```
m=10;
Rand = tlist(['border';'a';'b';'c';'d'],...
list('x=0; y=1-t',0,1,m,1),...                  //a
list('x=t; y=0',0,1,m,1),...                    //b
list('x=1; y=t',0,1,m,1),...                    //c
list('x=t; y=1',1,0,m,1));                      //d
buildMesh(Rand,'Omega');
[Knoten,Dreiecke]=getffResult();
  xset('window',0);
  xset('wdim',400,400);
  xbasc(0);
  ShowMesh(Knoten,Dreiecke,5);
```

Programm 4.1: Das Integrationsgebiet mit Gitter

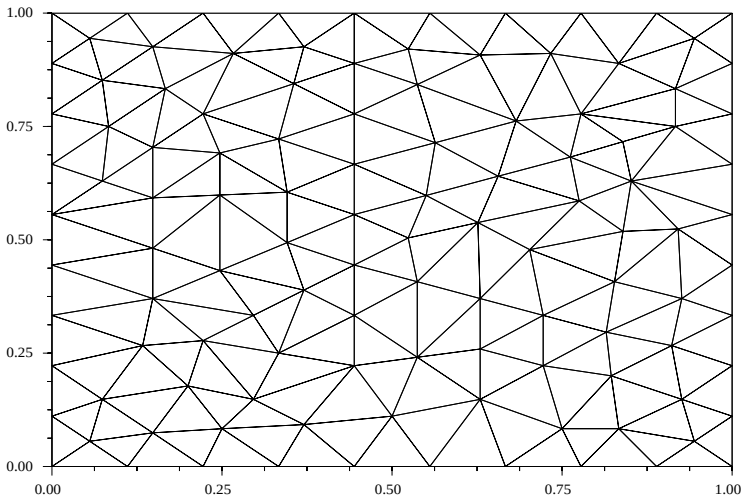

Abbildung 4.1: Die Triangulierung von $Omega$

```
ff_exec('function RS1(x,y) = 2*x^2-4*x+4*y^2-6*y+4*x*y+1');
ff_exec('function RS2(x,y) = 2*y^2-4*y+4*x^2-6*x+4*x*y+1');
ff_problem(['solve(Omega,u,v) {'                                ;...
            'pde(u) laplace(u)+dxx(u)+dxy(v)=RS1(x,y);'         ;...
            'on(a,b,c,d)   u=0;'                                ;...
            'pde(v) laplace(v)+dyy(v)+dyx(u)=RS2(x,y);'         ;...
            'on(a,b,c,d)   v=0;'                                ;...
            '}'],%t);
```

Programm 4.2: Das System in FreeFEM

Bemerkung:
In diesem Kapitel werden nicht die Funktionen **pde_sol** und **pde_varsol** benutzt, damit die unterschiedlichen Formulierungen der Differentialgleichungen deutlicher werden.

4.2 Die schwache Formulierung eines Systems von zwei Gleichungen

Insbesondere für Probleme der Elastizitätstheorie bei denen Systeme von partiellen Differentialgleichungen auftreten, sind schwache Formulierungen von Bedeutung, die historisch aus dem sogenannten Prinzip der virtuellen Arbeit hergeleitet wurden. (Siehe [1], Kapitel 7). In dem folgenden Programm wird die schwache Formulierung des Systems aus dem vorigen Abschnitt zur Lösung mit FreeFEM benutzt. Dabei ist zu beachten, daß für Testfunktionen $w(x, y)$, die auf dem Rand des betrachteten Gebietes verschwinden, die Greensche Formel für den Laplaceoperator folgende Identität liefert: (Siehe [3], Kapitel 5)

$$-\int_\Omega \triangle uw \; dxdy = \int_\Omega grad(u)^T grad(w) \; dxdy$$

```
u1=defvar('u1');
u2=defvar('u2');
ff_problem(['varsolve(Omega,0) A(u1,w1,u2,w2) with {'          ;...
            'L1=dx(u1)*dx(w1)+dy(u1)*dy(w1);'                  ;...
            'L2=dx(u2)*dx(w2)+dy(u2)*dy(w2);'                  ;...
            'L3=dx(u1)+dy(u2);'                                ;...
            'L4=dx(w1)+dy(w2);'                                ;...
            'A=int()(-L1-L2-L3*L4'                             ;...
            '-RS1(x,y)*w1-RS2(x,y)*w2)'                        ;...
            '+on(a,b,c,d)(w1)(u1=0)'                           ;...
            '+on(a,b,c,d)(w2)(u2=0);'                          ;...
            '}'],%t);
```

Programm 4.3: Die schwache Formulierung des PDGl-Systems

```
UpdateMesh(Rand,'Omega1');
ff_exec('femp1(Omega1) uu1 = u1');
ff_exec('femp1(Omega1) uex = x*(1-x)*y*(1-y)');
getf Import
Fehler=Import('sqrt(int(Omega1)((uu1-uex)^2))');
xset("fpf",' ');
xset('window',1);
xset('wdim',400,400);
xbasc(1);
ShowContour('uu1',[0;0],1,1,50,6,1);
xtitle('Fehler = '+string(Fehler));
```

Programm 4.4: Der Höhenlinienplot der Funktion $u1(x, y)$

Bemerkungen:

1. Wegen eines Fehlers in FreeFEM muß nach der Benutzung von **`varsolve`** mit **`UpdateMesh`** ein neues Gitter eingeführt werden. Hier ist das neue mit dem alten Gitter identisch. Der Fehler tritt bei den Funktionen **`print`** und **`append`** auf. Letztere wird in der FreeFEM-Funktion **`ShowContour`** benutzt.
2. Da die Methode der Finiten Elemente immer auf die schwache Formulierung führt, und die Benutzung der Funktion **`solve`** in allen getesteten Beispielen schneller war, sollte sie falls möglich vorgezogen werden.
3. Bei verschiedenen Randbedingungen sind zusätzliche Randintegrale zu berücksichtigen. (Siehe [1], Kapitel 7).

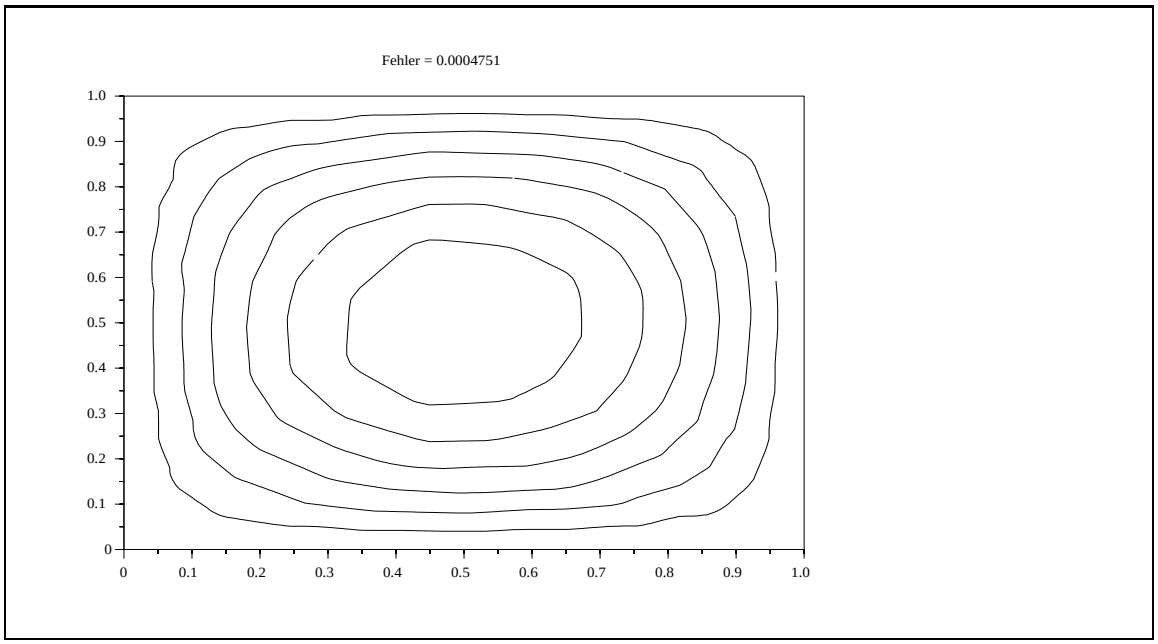

Abbildung 4.2: Höhenlinien der Funktion $u1(x, y)$

4.3 Ein Beispiel aus der Elastizitätstheorie

Für ebene Spannungszustände erhält man mit Hilfe des Hookeschen Gesetzes einen linearen Zusammenhang zwischen Spannungen und Verzerrungen, die mit Hilfe der Verschiebungen $u(x, y)$ und $v(x, y)$ ausgedrückt werden können. Nach Einführung der sogenannten Laméschen Konstanten:

$$\lambda := \frac{\nu E}{1-\nu^2} \text{ und } G := \frac{E}{2(1+\nu)} ,$$

(E ist der Elastizitätsmodul und ν die Poissonsche Zahl)
läßt sich der Zusammenhang zwischen Spannungen und Verzerrungen folgendermaßen darstellen:

$$\begin{pmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{pmatrix} = \begin{pmatrix} \lambda/\nu & \lambda & 0 \\ \lambda & \lambda/\nu & 0 \\ 0 & 0 & G \end{pmatrix} \begin{pmatrix} \partial u/\partial x \\ \partial v/\partial y \\ \partial u/\partial y + \partial v/\partial x \end{pmatrix}$$

Es gelten folgende Gleichgewichtsgleichungen der Kräfte in x- und y-Richtung:

$$\begin{aligned} \frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} &= -f_1 \\ \frac{\partial \sigma_y}{\partial y} + \frac{\partial \tau_{xy}}{\partial x} &= -f_2 \end{aligned}$$

Setzt man die Ausdrücke für die Spannungen in diese Gleichungen ein, so erhält man ein gekoppeltes System von zwei partiellen Differentialgleichungen für die beiden Verschiebungen $u(x, y)$ in x-Richtung und $v(x, y)$ in y-Richtung, die sogenannten Laméschen Differentialgleichungen:

$$\begin{aligned} -G\triangle u - (\lambda+G)(\frac{\partial^2 u}{\partial^2 x} + \frac{\partial^2 v}{\partial x \partial y}) &= f_1 \\ -G\triangle v - (\lambda+G)(\frac{\partial^2 v}{\partial^2 y} + \frac{\partial^2 u}{\partial x \partial y}) &= f_2 \ . \end{aligned}$$

Diese Differentialgleichungen sind mit entsprechenden Randbedingungen zu lösen. Bei der Formulierung dieser Randwertprobleme in FreeFEM ist darauf zu achten, daß die Funktion $dnu(\cdot)$ die

konormale Ableitung ist. In dem hier vorliegendem Fall also die Funktionen

$$dnu(u) = (\lambda + 2G)d_1 + Gd_2 = \frac{\lambda}{\nu}d_1 + Gd_2$$

$$dnu(v) = (\lambda + 2G)d_2 + Gd_1 = \frac{\lambda}{\nu}d_2 + Gd_1$$

mit der äußeren Normalen $\vec{n} = (n_1, n_2)^T$ und $\vec{d} := (u_x n_1, u_y n_2)^T$. Auf Rändern parallel zur y-Achse stimmt $dnu(u)$ also mit der Spannung σ_x überein.
In dem folgenden Beispiel wird eine rechteckige Lochscheibe behandelt, die auf zwei gegenüberliegenden Seiten mit entgegengesetzt gleichen Kräften parallel zur Scheibe belastet wird. Unter Ausnutzung der Symmetrien des Problems erhält man folgende FreeFEM-Formulierung.
Die Ergebnisse stimmen qualitativ mit den in [5], Seite 95, angegebenen Werten für die Spannungen in den Punkten $(0, 1/2)^T$ und $(0, 1)^T$ überein. Die lokalen Maxima für σ_x liegen jeweils in der Nähe der angebenen Punkte; ihre Werte haben bei der angegebenen Diskretisierung relative Fehler von 25 und 100%. Die Unabhängigkeit dieser Werte vom Elastizitätsmodul E und der Poissonschen Zahl ν ist sehr gut zu beobachten.
Bei diesen Rechnungen ist zu beachten, das die Funktionswerte von Gitterfunktionen, in deren Definition Ableitungen anderer Gitterfunktionen benutzt werden, nicht mit Hilfe der FreeFEM-Funktion **`getffResult`** importiert werden können. (Im Beispiel ist das die Gitterfunktion **`sx`**)

```
m=50;l=10;ff_var('l',string(l));
Rand = tlist(['border';'a';'b';'c';'d';'e'],...
list('x=0; y=1-t',0,0.5,m,1),...                        //a
list('x=0.5*cos(t); y=0.5*sin(t)',%pi/2,0,m,1),...      //b
list('x=0.5+t; y=0',0,l-0.5,m,1),...                    //c
list('x=l; y=t',0,1,10,1),...                           //d
list('x=t; y=1',l,0,m,1));                              //e

buildMesh(Rand,'Lochscheibe');

E = 1; nu = 0.29;
ff_var('E',string(E));
ff_var('nu',string(nu));
ff_var('lam',string(E*nu/(1-nu^2)));
ff_var('G',string(E/(2*(1+nu))));
u=defvar('u');
v=defvar('v');sx=defvar('sx');

ff_problem(['solve(Lochscheibe,u,v) {'                                  ;...
            'pde(u) -laplace(u)*G-dxx(u)*(lam+G)-dxy(v)*(lam+G)=0;';...
            'on(a)   u=0;'                                              ;...
            'on(d)   dnu(u)=1;'                                         ;...
            'pde(v) -laplace(v)*G-dyy(v)*(lam+G)-dxy(u)*(lam+G)=0;';...
            'on(c)   v=0;}'],%t);
ff_exec('femp1  sx =(lam/nu)*(dx(u)+nu*dy(v));');
xset('window',1);
xset('wdim',400,400);
xset("fpf",' ');
xbasc(1);
ShowContour('sx',[0.005;0.505],0.003,0.003,50,30,1);
```

Programm 4.5: Die Laméschen Differentialgleichungen

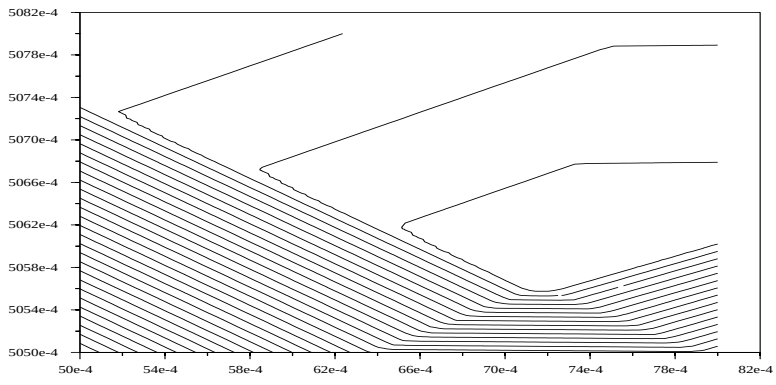

Abbildung 4.3: Höhenlinien der Funktion $\sigma_x(x, y)$ in der Nähe des Punktes $(0, 1/2)^T$

Im Falle des hier vorliegenden Gitters liegt ein lokales Maximum der Spannung σ_x im Punkt $(0.0072, 0.5061)^T$ und hat dort den Wert 3.2. Im Punkt $(0, 1)^T$ hat σ_x den maximalen Wert von 1.6, ist also etwa doppelt so groß wie der in [5], Seite 95, angegebene Wert.

Bemerkungen:

1. Bei der Formulierung der Randwertprobleme muß auf die Regularität der zu lösenden linearen Gleichungssysteme geachtet werden. Beim obigen Beispiel wird das durch Dirichlet-Randbedingungen für u und v erreicht. In vielen Fällen ist eine Lösung mit FreeFEM problematisch. Im obigen Beispiel ist die durch die schwache Formulierung implizierte natürliche Randbedingung auf dem Kreisbogen möglicherweise Ursache zusätzlicher Fehler.
2. In FreeFEM wird mit linearen Finiten Elementen gerechnet. Auf eine genauere Fehleranalyse insbesondere von Approximationen der Ableitungen der Verschiebungen u und v kann hier nicht eingegangen werden.

Kapitel 5

Das Schwarzsche Verfahren

Das Schwarzsche Verfahren ist ein iteratives Verfahren zur Lösung von allgemeinen Problemen in Gebieten, die sich als Vereinigung einfacherer Gebiete darstellen lassen die sich überlappen. Dieses Verfahren spielt insbesondere in der parallelen Numerik eine große Rolle. (Siehe [1], 7.2 und [4], 9.3.6). Das folgende Beispiel zeigt die iterative Lösung der Poissonschen Gleichung mit homogenen Dirichlet-Randbedingungen in einem Gebiet Ω, das sich als Vereinigung eines Rechteckes Ω_1 mit dem Einheitskreis Ω_2 darstellen läßt. Hierbei wird gleichzeitig das do-Schleifenkonzept von FreeFEM vorgestellt, bei dem insbesondere die Möglichkeit der Wiederbenutzung der Faktorisierung der Systemmatrizen zur Zeitoptimierung hervorzuheben ist. (Siehe [1], Kapitel 6 und 7).

```
m=2;
Rand1 = tlist(['border','a','a1','b','c','d']          ,...
        list('x=t; y=0;',0,1,5*m,1)                    ,...  //a
        list('x=t; y=0;',1,2,5*m,1)                    ,...  //a1
        list('x=2; y=t;',0,1,5*m,1)                    ,...  //b
        list('x=t; y=1;',2,0,10*m,1)                   ,...  //c
        list('x=0; y=t;',1,0,5*m,1));                        //d
Rand2 = tlist(['border','e','e1']                      ,...
        list('x=cos(t); y=sin(t);',0,%pi/2,5*m,1)      ,...  //e
        list('x=cos(t); y=sin(t);',%pi/2,2*%pi,25*m,1));     //e1
buildMesh(Rand1,'Om1');
buildMesh(Rand2,'Om2');
```

Programm 5.1: Die überlappenden Gebiete

```
ff_exec('femp1(Om1) v=0');
n=5;ff_var('n',string(n));
ff_problem(['for i=0 to n do {'         ,...
            'solve(Om2,V) with A(i) {'  ,...
            'pde(V) -laplace(V)=1;'     ,...
            'on(e)  V=v;'               ,...
            'on(e1) V=0;};'             ,...
            'solve(Om1,v) with B(i) {'  ,...
            'pde(v) -laplace(v)=1;'     ,...
            'on(a,d)    v=V;'           ,...
            'on(a1,b,c) v=0;};};']);
```

Programm 5.2: Der FreeFEM Aufruf

In der folgenden Abbildung werden Höhenlinien der Gitterfunktionen V und v, sowie der Differenz $V - v$, jeweils nach 5 Iterationen des Schwarzschen Verfahrens dargestellt. Man achte insbesondere auf die Funktion V in $\Omega_1 \cap \Omega_2$ und ihre Fortsetzung außerhalb des Einheitskreises Ω_2 mit Hilfe ihrer Werte auf dem Kreisrand.

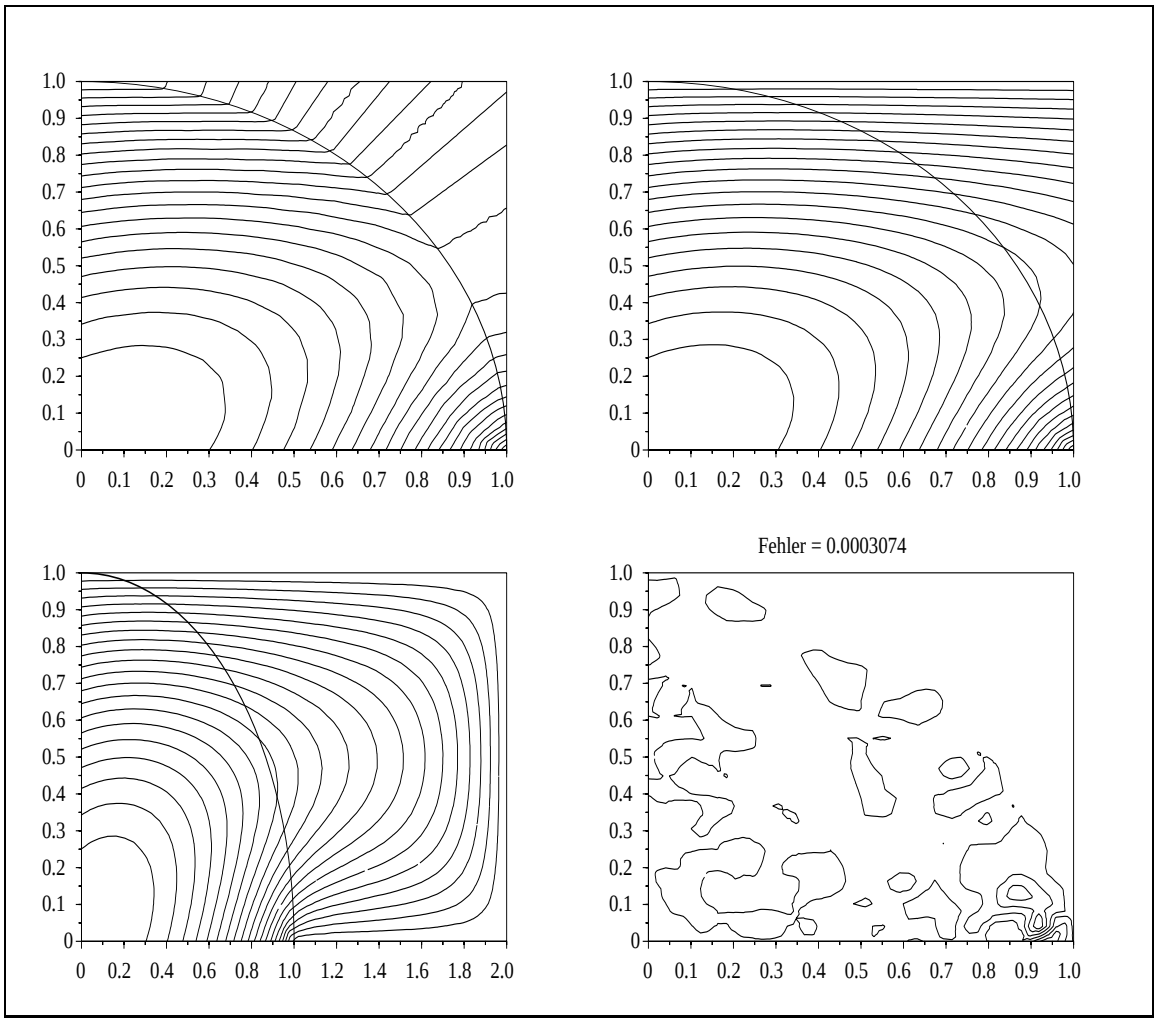

Abbildung 5.1: Von oben links bis unten rechts: V in $\Omega_1 \cap \Omega_2$, v in $\Omega_1 \cap \Omega_2$, v in Ω_1 und die Differenz $V - v$ in $\Omega_1 \cap \Omega_2$; jeweils nach 5 Iterationen. Der Fehler ist die mittlere quadratische Abweichung der Differenz in $\Omega_1 \cap \Omega_2$.

In der nächsten Abbildung werden Höhenlinien der Lösung der Poissonschen Gleichung in $\Omega_1 \cup \Omega_2$ mit homogenen Dirichlet-Randbedingungen gezeigt. Wegen der einfachen Möglichkeit, in sehr allgemeinen Gebieten mit Hilfe der Funktion **`buildMesh`** Dreiecksgitter einzuführen, ist das Schwarzsche Verfahren in FreeFEM hauptsächlich zum Testen paralleler Algorithmen und für pädagogische Zwecke von Interesse.

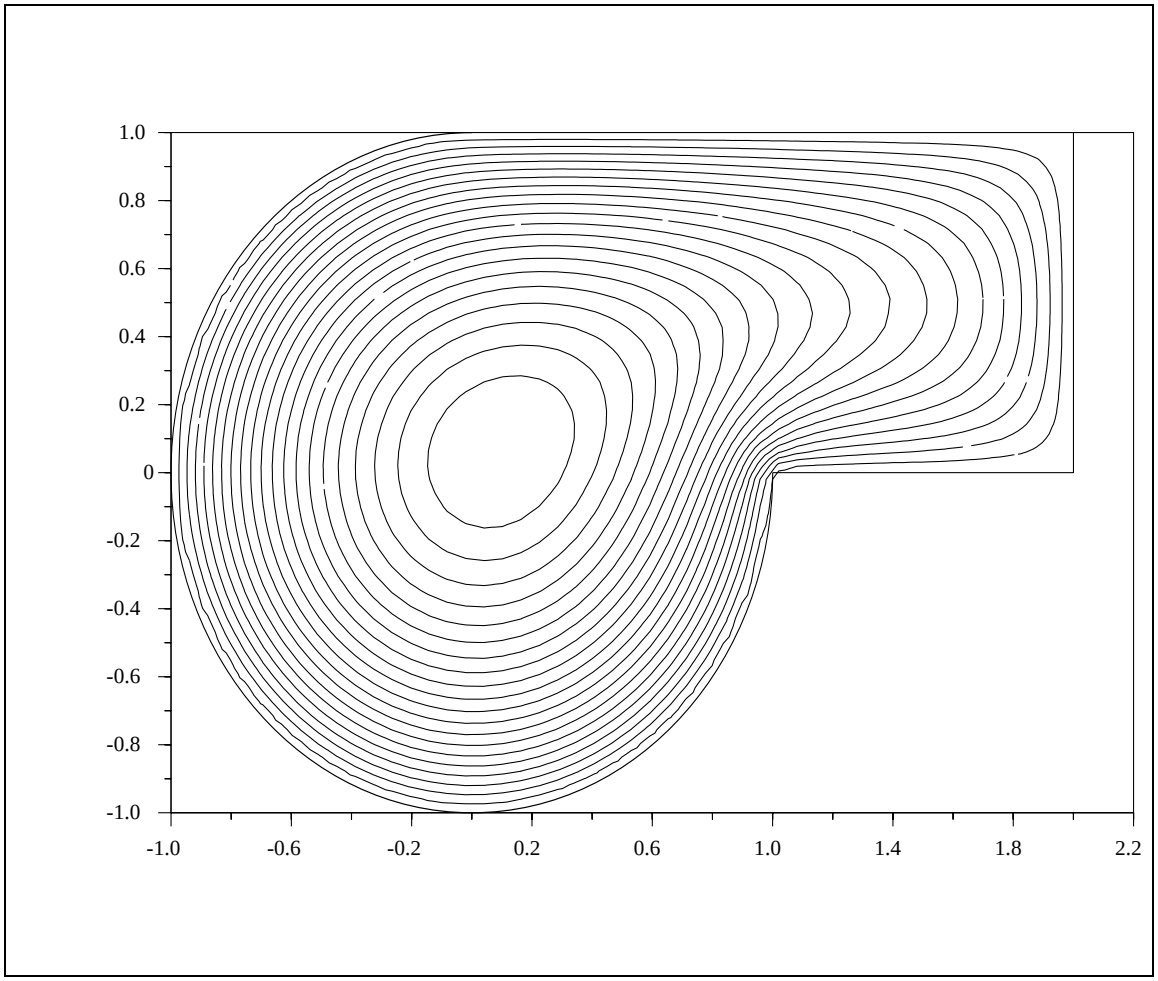

Abbildung 5.2: Die Lösung der Poissonschen Gleichung im Gebiet $\Omega_1 \cup \Omega_2$

Kapitel 6

Gitteradaption

In FreeFEM gibt es die Möglichkeit, vorhandene Gitter mit Hilfe der FreeFEM-Funktion **`ff_adaptmesh`** an Extremalstellen vorgegebener Funktionen und ihrer Ableitungen zu verfeinern. Damit kann man Gitter iterativ einem Problem anpassen. (Siehe [1], 4.4). Die Einzelheiten der Benutzung von **`ff_adaptmesh`**, insbesondere die Bedeutung der Parameter werden in [1] nicht beschrieben. In der Scilab Toolbox FreeFEM erhält man durch den Aufruf

```
help ff_adaptmesh
```

eine Auflistung dieser Parameter, aber keine näheren Hinweise. Trotz dieser unangenehmen Einschränkungen kann man mit Hilfe der Beispiele aus [1] ähnliche Testbeispiele durchrechnen.
Das erste Beispiel zeigt die adaptive Verfeinerung eines Gitters im Einheitskreis mit Hilfe einer Funktion, die auf der y-Achse und für $x = -1$ große Werte annimmt.

```
m=30;
Rand = tlist(['border';'K'],...
        list('x=cos(t); y=sin(t);',0,2*%pi,m,1));
buildMesh(Rand,'K');
Fehler=0.01;
ff_exec('femp1(K) u=exp(2*x)/(0.001+y^2)-5/(0.001+(1+x)^2)');
ff_adaptmesh('mesh KA = adaptmesh  (K,u)',verbosity=5,...
               abserror=1,nbjacoby=2,err=Fehler, nbvx=5000, ...
               omega=1.8,ratio=1.8, nbsmooth=3,splitpbedge=1, ...
               maxsubdiv=5,rescaling=1);
```

Programm 6.1: Gitteradaption im Kreis

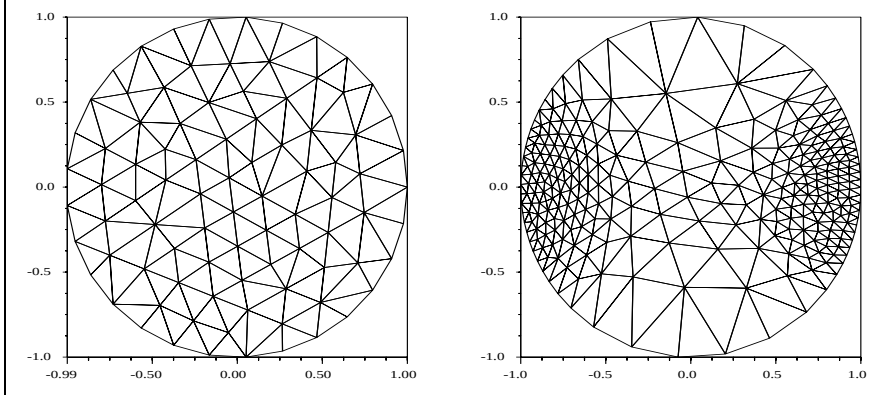

Abbildung 6.1: Ausgangsgitter **`K`** und mit der Gitterfunktion **`u`** adaptiertes Gitter **`KA`**

Ein bekanntes Beispiel der adaptiven Verfeinerung von Gittern ist die Lösung der Poissonschen Gleichung mit homogenen Dirichlet-Randbedingungen in einem L-förmigen Gebiet, da der Gradient der Lösung in der einspringenden Ecke singulär ist.
Folgendes FreeFEM-Programm illustriert die Vorgehensweise:

```
Rand = tlist(['border';'a';'b';'c';'d';'e';'f'],...
list('x = t; y = 0',0,1,6,1),...                //a
list('x = 1; y = t',0,0.5,4, 1),...             //b
list('x = 1 - t; y = 0.5',0,0.5,4,1),...        //c
list('x = 0.5; y = t',0.5,1,4,1),...            //d
list('x = 1 - t; y = 1',0.5,1,4,1),...          //e
list('x = 0; y = 1 - t',0,1,6,1));              //f
buildMesh(Rand,'R'); [Knoten,Dreiecke]=getffResult();

Fehler = 0.1;  Koeff  = 0.1^(1./5.);
for i=1:4
  ff_problem('solve(u) pde(u) laplace(u) = 1; ...
              on(a,b,c,d,e,f) u = 0;');
  xset('window',i);xset('wdim',700,500);
  xbasc(i);xsetech([0,0,1,0.5]);
  ShowMesh(Knoten,Dreiecke,1);
  xsetech([0,0.5,1,0.5]);
  xset("fpf",' ');
  ShowContour('u',[0;0],1,1,50,10,1);
  xpoly([0.5;0.5;1],[1;0.5;0.5],'lines')
  Fehler=Fehler*Koeff;
  ff_adaptmesh('mesh R = adaptmesh  (R,u)',verbosity=5,...
               abserror=1,nbjacoby=2,err=Fehler, nbvx=5000, ...
               omega=1.8,ratio=1.8, nbsmooth=3,splitpbedge=1, ...
               maxsubdiv=5,rescaling=1);
  if (i<4) then
  [Knoten,Dreiecke]=getffResult();
  end
end
ff_problem('solve(u) pde(u) laplace(u) = 1; ...
            on(a,b,c,d,e,f) u=0;');
[Knoten,Dreiecke]=getffResult();
xset('window',5);xset('wdim',700,500);
xbasc(5);xsetech([0,0,0.5,0.5]);
ShowMesh(Knoten,Dreiecke,1);
xsetech([0.5,0,0.5,0.5]);
ShowMesh(Knoten,Dreiecke,1,[0.35 0.35 0.55 0.55]);
xsetech([0,0.5,1,0.5]);
ShowContour('u',[0;0],1,1,50,10,1);
xpoly([0.5;0.5;1],[1;0.5;0.5],'lines')
```

Programm 6.2: Lösung der Poissonschen Gleichung im L-Gebiet

Bemerkung:
Mit Hilfe des Befehls **`scilab -nw -f Progr.sci > Ausgabe`** kann man Scilab im Batch-Modus aufrufen und die FreeFEM-Ausgabe in die Datei Ausgabe umlenken.
Dazu muß die FreeFEM-Toolbox am Anfang des Programms **`Progr.sci`** mit dem Aufruf **`exec('/usr/local/bin/loader.sce',0)`** geladen werden. Am Ende des Programms muß Scilab mit **`exit`** beendet werden.

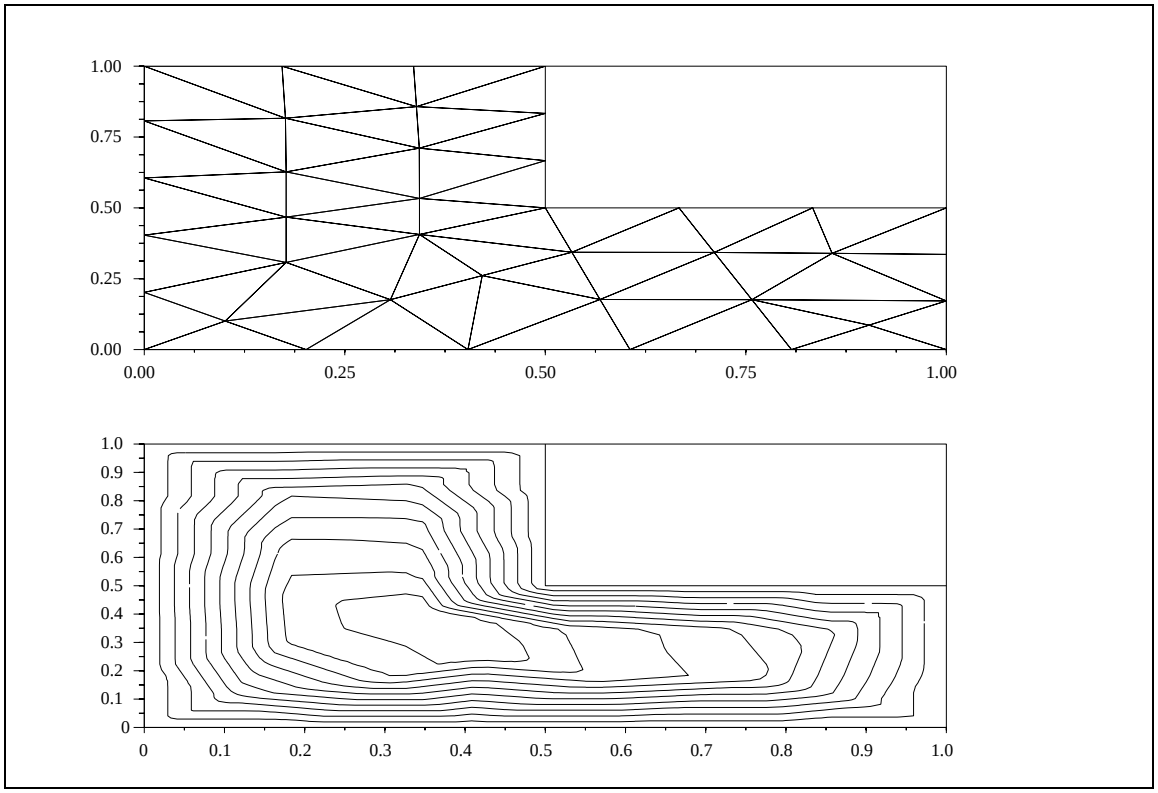

Abbildung 6.2: Lösung mit dem Ausgangsgitter

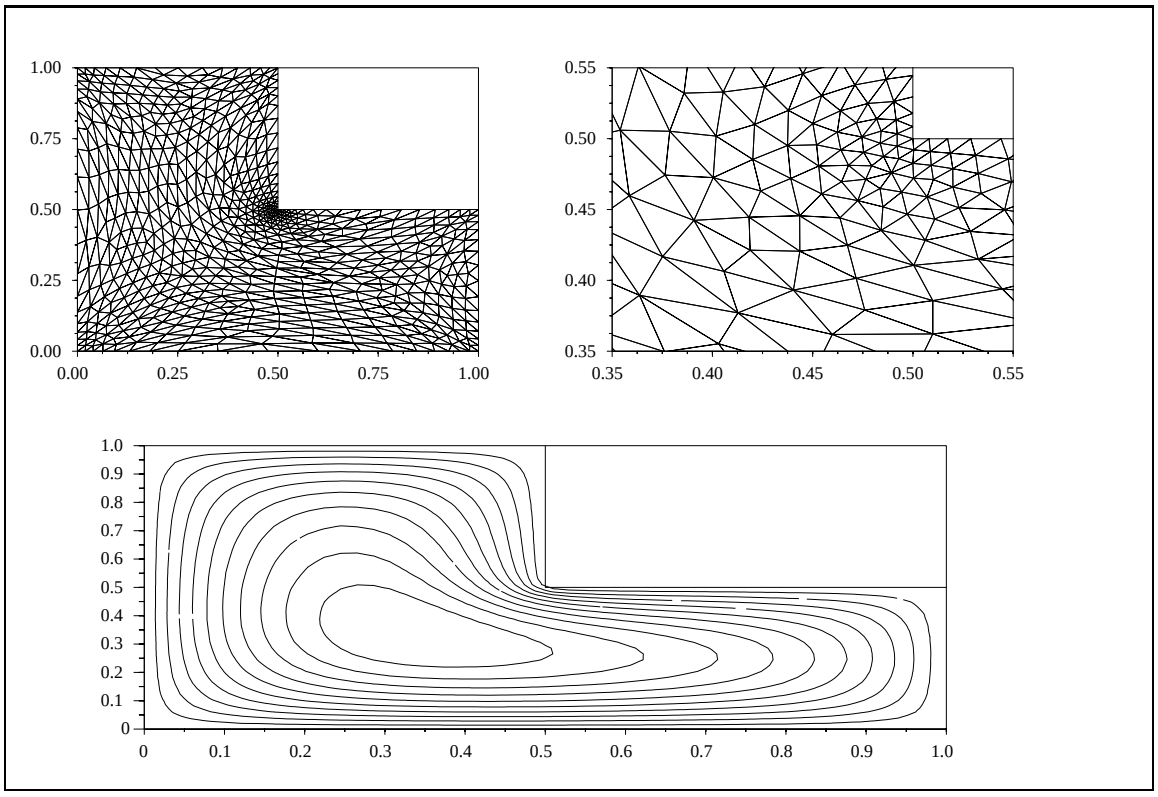

Abbildung 6.3: Lösung nach vier Gitteradaptionen

Kapitel 7

Instationäre Probleme

In diesem Kapitel wird die numerische Lösung von instationären Problemen mit Hilfe von FreeFEM behandelt. Als Testbeispiel wählen wir die Wärmeleitungsgleichung $u_t = \triangle u$ mit der Anfangsbedingung $u(x, y, 0) = 1$ in einem Quadrat $Q = [-1, 1] \times [-1, 1]$.
Die Lösung $u(x, y, t)$ soll homogene Dirichlet-Randbedingungen erfüllen.
Die Lösung dieses Problems kann man als Produktdarstellung aus der entsprechenden eindimensionalen Wärmeleitungsgleichung herleiten. Die Lösung des eindimensionalen Problems erhält man mit Hilfe der Laplacetransformationsmethode als unendliche Reihe. Mit Hilfe des folgenden Scilabprogramms kann man Partialsummen dieser Reihe berechnen.

```
function u=Heat(x,t,n)
//  Partialsummen der Reihendarstellung der Loesung u(x,t)
//  der Waermeleitungsgleichung fuer Anfangswerte u(x,0)=1
//  und Randwerte u(-1,t)=u(1,t)=0 im Intervall [-1,1].
//  Die entsprechende Loesung im Quadrat [-1,1]x[-1,1]
//  ist u(x,t)*u(y,t).
n=0:n;
u=(-1).^n.*exp(-(2*n+1).^2*(%pi)^2*t/4).*cos((2*n+1)*%pi*x/2)./(2*n+1);
u=u($:-1:1);u=sum(u);
u=4*u/%pi;
```

Programm 7.1: Lösung der eindimensionalen Wärmeleitungsgleichung in Scilab

Bemerkung: Da die Rand- und Anfangsbedingungen unverträglich sind, muß man für kleine Zeiten in der Nähe des Randes mit langsamer Konvergenz der unendlichen Reihe rechnen.

Diskretisiert man die Differentialgleichung nur bezüglich der Zeitvariablen t und bezeichnet $u(x, y, n\Delta t)$ mit $u_n(x, y)$, so erhält man beispielsweise folgendes Gleichungssystem:

$$\frac{u_{n+1}(x, y) - u_n(x, y)}{\Delta t} = \theta \triangle u_{n+1}(x, y) + (1 - \theta) \triangle u_n(x, y) \ ,$$

das mit den vorgegebenen Randbedingungen für $n = 0, 1, 2, \cdots$ gelöst werden muß. Der Parameter θ kann aus dem Intervall $[0, 1]$ gewählt werden. Für $\theta = 1$ und $\theta = 1/2$ erhält man beispielsweise Varianten des impliziten Eulerverfahrens und des Crank-Nicolson-Verfahrens, die im folgenden in FreeFEM benutzt werden. (Siehe [3], Kapitel 11).
Bei instationären Problemen muß man beachten, daß die rechte Seite der Differentialgleichung, die mit Hilfe der FREEFEM+-Funktion **solve** in FreeFEM gelöst werden soll, keine Operatoren wie z.B. **laplace** enthalten darf. Also kommt bei der Benutzung von **solve** nur das implizite Eulerverfahren in Frage. Für die anderen Verfahren mit $\theta < 1$ muß die schwache Formulierung der Differentialgleichungen benutzt werden, also die FREEFEM+-Funktion **varsolve**.

Als erstes Beispiel wird die Wärmeleitungsgleichung mit Hilfe des impliziten Eulerverfahrens gelöst. Dieses Verfahren ist in FreeFEM einfach zu programmieren und ist etwa doppelt so schnell wie das folgende Crank-Nicolson-Verfahren. Die Faktorisierung der Systemmatrix wird bei Benutzung der Funktion **`solve`** nur einmal am Anfang der Iteration durchgeführt.

```
m=10;
Rand = tlist(['border';'a';'b';'c';'d'],...
list('x = t; y = -1',-1,1,m,1),...
list('x = 1; y = t',-1,1,m,1),...
list('x = t; y = 1',1,-1,m,1),...
list('x = -1; y = t',1,-1,m,1));
buildMesh(Rand,'Q');
[Knoten,Dreiecke]=getffResult();
T=0.3;n=50;dt=T/n;
ff_var('dt',string(dt));
ff_var('i',0);
ff_var('n',string(n));
ff_exec('femp1 u=1');
ff_problem(['for i=0 to n-1 do {'                ;...
            'solve(Q,u) with A(i) {'             ;...
            'pde(u) u-laplace(u)*dt = u;'        ;...
            'on(a,b,c,d) u = 0;'                 ;...
            '};};'],%t);
disp('Exakte Loesung: u(0,0,0.3)=0.3682109');
getf Import; u0=Import('u(0,0)');
  xset('window',1);
  xset('wdim',800,400);
  xbasc(1)
  xsetech([0,0,0.5,1]);
  ShowMesh(Knoten,Dreiecke,1);
  xsetech([0.5,0,0.5,1]);
  ShowContour('u',[-1;-1],2,2,50,5,1);
  xtitle('u(0,0,'+string(T)+')='+string(u0))
```

Programm 7.2: Das implizite Eulerverfahren

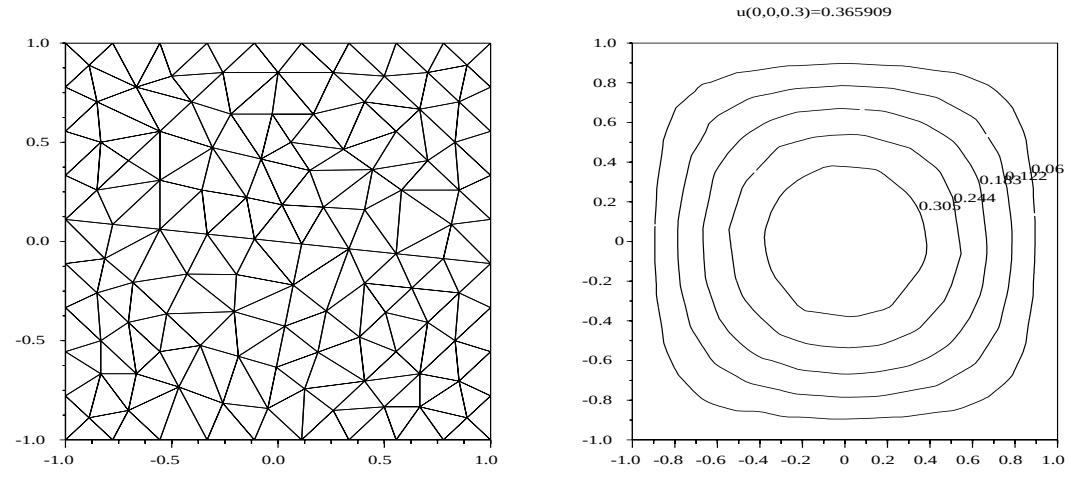

Abbildung 7.1: Ausgangsgitter **`Q`** und die Lösung **`u(x,y,T)`** beim impliziten Eulerverfahren

Mit Hilfe der schwachen Formulierung des obigen Differentialgleichungssystems kann das Crank-Nicolson-Verfahrens in FreeFEM zur Lösung instationärer Probleme benutzt werden. Dabei ist wieder die in 4.2 erwähnte Greensche Formel zu beachten.

```
m=10;
Rand = tlist(['border';'a';'b';'c';'d'],...
list('x = t; y = -1',-1,1,m,1),...
list('x = 1; y = t',-1,1,m,1),...
list('x = t; y = 1',1,-1,m,1),...
list('x = -1; y = t',1,-1,m,1));
buildMesh(Rand,'Q');[Knoten,Dreiecke]=getffResult();
T=0.3;n=50;dt=T/n;
theta=0.5;
ff_var('dt',string(dt));ff_var('theta',string(theta));
ff_var('n',string(n));
ff_exec('femp1 ua=1');
u=defvar('u');uu=defvar('uu');w=defvar('w');
ff_problem(['for i=0 to n-1 do {'                                ;...
       'varsolve(Q,i) A(u,w) with      {'                        ;...
       'A = int(Q)((u-ua)*w+ dt*theta*(dx(u)*dx(w)+dy(u)*dy(w))'  ;...
       '            +dt*(1-theta)*(dx(ua)*dx(w)+dy(ua)*dy(w)))'   ;...
       '            +on(a,b,c,d)(w)(u=0)}'                        ;...
       ' ua = u;               };']);

disp('Exakte Loesung: u(0,0,0.3)=0.3682109');
[us] = getffResult('u');
UpdateMesh(Rand,'Q1');
ff_problem('solve(Q1,uu1) pde(uu1) uu1=u; on(a,b,c,d) uu1=u;',%t);
ff_exec('femp1(Q1) uu=u');
getf Import
uu0=Import('uu(0,0)');uu10=Import('uu1(0,0)');
disp([uu0 uu10])
disp('max(us)= ');disp(max(us));
  xset('window',1);
  xset('wdim',800,400);
  xbasc(1)
  xsetech([0,0,0.5,1])
  ShowMesh(Knoten,Dreiecke,1);
  xsetech([0.5,0,0.5,1])
  ShowContour('uu',[-1.2;-1.2],2.4,2.4,50,5,1);
  xpoly([-1 1 1 -1 -1],[-1 -1 1 1 -1],'lines')
  xtitle('u(0,0,'+string(T)+')='+string(uu10))
```

Programm 7.3: Das Crank-Nicolson-Verfahren

Bemerkungen:

1. Die Gitterfunktion **uu1** ist nicht mit **uu** identisch. Sie wurde zu Testzwecken eingeführt.
2. Die analytische Lösung der Wärmeleitungsgleichung wurde mit Hilfe der Scilabfunktion **Heat** berechnet.
3. Auch beim Crank-Nicolson-Verfahren ist nur am Anfang der Iteration eine Faktorisierung erforderlich. (Siehe [3], 11.3.1). Wie diese Tatsache in FREEFEM+ im Einzelnen ausgenutzt wird, ist nicht ersichtlich.

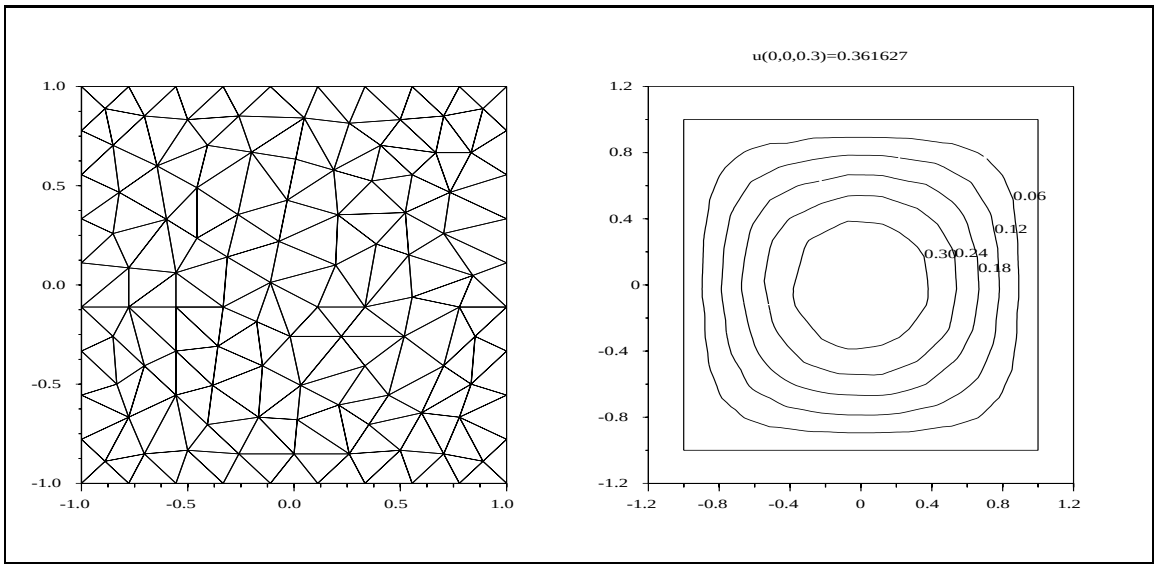

Abbildung 7.2: Ausgangsgitter `Q` und die Lösung `u(x,y,T)` beim Crank-Nicolson-Verfahrens

Kapitel 8

Nichtlineare partielle Differentialgleichungen

In diesem Kapitel wird am Beispiel einer konstruierten nichtlinearen partiellen Differentialgleichung die iterative Lösung in FreeFEM durchgeführt. In konkreten Anwendungen ist die Wahl eines konvergenten Verfahrens das Hauptproblem. Eine Formulierung der Iteration als Fixpunktiteration führt dabei manchmal zum Ziel. (Siehe z.B. [4], Kapitel 6.3).
Im folgenden wird eine nichtlineare Differentialgleichung im Einheitskreis konstruiert, deren Lösung in Polarkoordinaten durch $u_{ex}(r) = r^4$ gegeben ist. In kartesischen Koordinaten hat die Differentialgleichung für $u(x, y)$ und reelle α folgende Form:

$$-\triangle u + 16\sqrt{u} + \alpha\, u = \alpha\,(x^2 + y^2)^2$$

Mit Hilfe der vorgegebenen Lösung $u_{ex}(\sqrt{x^2 + y^2})$ lassen sich z.B. Robinsche Randbedingungen $\frac{\partial u}{\partial n_L} + u = 5$ bestimmen, die im folgenden benutzt werden. (Siehe [3], Kapitel 6, wo die schon in 3.4 benutzte konormale Ableitung in Richtung der äußeren Normalen eingeführt wird.)

```
m=50;
Rand = tlist(['border';'R'],...
        list('x=cos(t); y=sin(t);',0,2*%pi,m,1));
buildMesh(Rand,'K');
n=20;ff_var('n',string(n));
al=100;ff_var('al',string(al));
ff_exec('femp1 ua=1');
ff_exec('femp1 uex=(x^2+y^2)^2');
ff_problem(['for i=0 to n-1 do {'                                  ;...
            'solve(K,u) with A(i) {'                               ;...
            'pde(u) -laplace(u)+u*al =-16*sqrt(abs(ua))+al*uex;'   ;...
            'on(R) dnu(u)+u = 5;'                                  ;...
            'ua = u;'                                              ;...
            'append(""Fehler"",sqrt(int(K)((u-uex)^2)));'
            '};};'],%t);
F=read('Fehler',-1,1);
xset('font',2,4)
plot2d([0:n-1]',log(F));
clear u ua F
unix('rm Fehler')
```

Programm 8.1: Iteration zur Lösung einer nichtlinearen Gleichung

Mit Hilfe des Faktors α in obiger Differentialgleichung kann man das Konvergenzverhalten des Iterationsverfahrens studieren.

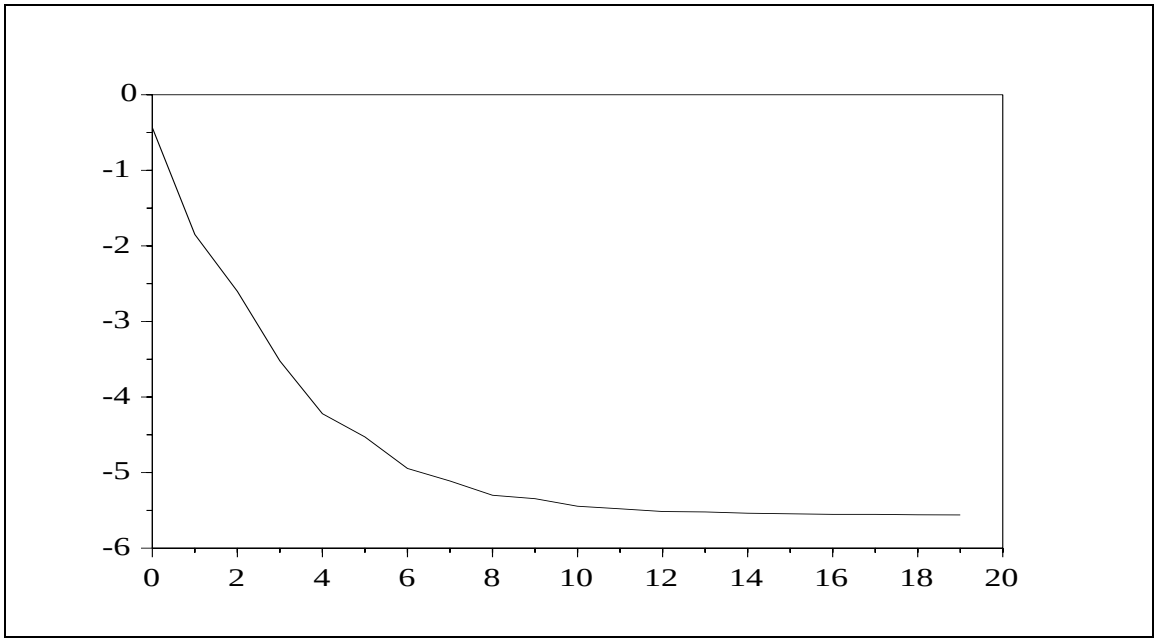

Abbildung 8.1: Mittlerer quadratischer Fehler (logarithmisch) in Abhängigkeit von der Iterationszahl

Literaturverzeichnis

[1] D. Bernadi, F. Hecht, K. Ohtsuka, O. Pironneau.
FREEFEM+ for Macs, PCs, Linux.
December 25, 1998.
Abgelegt unter: **`/usr/local/freefem+/doc/FreeFem.pdf`**

[2] R. von Seggern, H. Niehoff, M. Vaeßen.
Scilab - eine interaktive Programmierumgebung für numerische Anwendungen.
FZJ-ZAM-TKI-0367 vom 08.02.01. (Technische Kurzinformation).

[3] Alfio Quarteroni, Alberto Valli.
Numerical Approximation of Partial Differential Equations.
Springer-Verlag, Berlin 1994.

[4] Brigitte Lucquin, Olivier Pironneau.
Introduction to Scientific Computing.
John Wiley & Sons Chichester 1998.

[5] S. P. Timoshenko, J. N. Goodier.
Theory of Elasticity.
McGraw-Hill Book Company, Third Edition 1970.