School of Engineering
Winterthur
Zürcher Hochschule für Angewandte Wissenschaften

Numerical Solution of the Heat Equation with Matlab

Rebekka Axthelm & Martin Loeser

Winterthur 2011

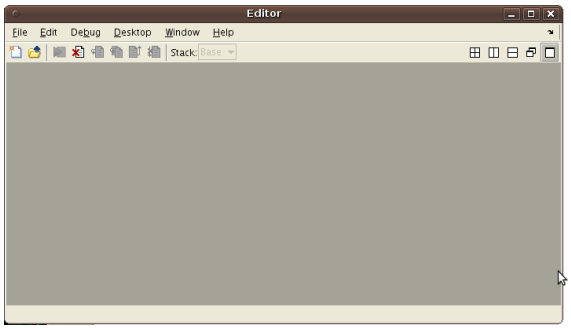
Tableofcontents

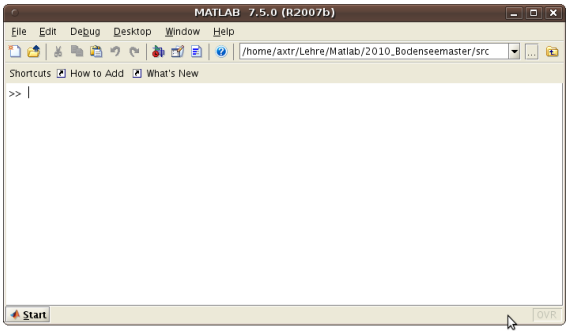
–1–
Eine Einführung in MatLab

1.1 Erste Schritte

Wie wir Matlab aufrufen hängt vom System ab, auf dem wir arbeiten. Unter Windows finden wir das Programm in der Programmliste und klicken es einfach an unter Linux führt der Befehl `matlab` im xterm zum Ziel. Es öffnet sich die Matlab–Konsole und je nachdem wie das Programm zuletzt verlassen wurde öffnet sich zusätzlich ein Editor.

Das Programm hat im Wesentlichen diese beiden Bedienebenen. Zum einen ist das die Matlab Konsole, wo Matlabbefehle eingegeben werden und zum anderen der Editor, indem Matlabdateien editiert werden. Es kann auch ein x–beliebiger Editor verwendet werden. Es empfiehlt sich aber den hauseigenen zu verwenden, da dieser auch über einige Verarbeitungsmöglichkeiten verfügt.





Öffnet sich der Editor beim Aufruf nicht automatisch so können Sie diesen manuell aufrufen, indem Sie über die Menueleiste

`Desktop` → `Editor`

aktivieren.

Aufgabe 1 Starten des Programms

Starten Sie Matlab.

Wichtig! Sie sollten immer wissen "wo" Sie sind. Wenn Sie auf eine Datei zugreifen wollen so muss sich diese im aktuellen Verzeichnis befinden oder aber Sie müssen den kompletten Pfad der Datei angegeben. Haben Sie Matlab aus einem xterm oder einer Eingabeaufforderung gestartet so ist das lokale Verzeichnis eben dieses, aus dem Sie das Programm gestartet haben, andernfalls ist es das unter Matlab eingestellte Standardverzeichnis. Es empfiehlt sich den Berg zum Propheten zu tragen: Rufen Sie Matlab dort auf, wo Ihre Dateien gespeichert sind oder aber wechseln Sie mittels `cd` das lokale Verzeichnis in der Matlabkonsole entsprechend.

1: Einige Systembefehle:

`pwd`	Zeigt den aktuellen Pfad an.
`dir`	Listet den Inhalt des aktuellen Verzeichnisses auf.
`cd <pfad>`	Wechselt das lokale Verzeichnis nach `<pfad>`.
`delete <datei>`	Löscht die Datei <datei>.
`edit <datei>`	öffnet/erzeugt die Datei `<datei>` im Editor.

1.2 Variablen und Operatoren

Bleiben wir zunächst in der Konsole.

2: Variablen und Felder belegen:

```
>> a=2
a =
   2
```

```
>> V=[1 2 3 4]
V =
   1 2 3 4
```

```
>> a=1/5
a =
   0.2000
```

```
>> length(V)
ans =
   4
```

```
>> V=1:1:4
V =
   1 2 3 4
```

```
>> V(5)=5
V =
   1 2 3 4 5
```

```
>> V=0:2:7
V =
   0 2 4 6
```

```
>> length(V)
ans =
   5
```

Aufgabe 2 Variablen und Felder

Sie dürfen einmal die obigen Definitionen und Anweisungen in der Konsole eingeben.

3: Weitere Systembefehle:

`who`	Zeigt alle definierten Größen an.
`whos`	Zeigt alle definierten Größen mit ihren Eigenschaften an.
`clear <Variable>`	Löscht die Variable <Variable>.
`clear all`	Löscht alle definierten Variablen.
`quit, exit`	Matlab verlassen.

4: Noch was:

`V(3:6)`	Zugriff auf 3. bis 6. Komponente des Feldes `V`
`S='bla'`	Zeichenketten oder einzelne Zeichen werden in Hochkomma gesetzt.
`;`	Ein Semikolon am Ende einer Anweisung verhindert die "Antwortausgabe" `ans` in der Konsole.

Aufgabe 3 Erste Schritte

Erzeugen Sie ein Feld `V` mit 10 Komponenten, so dass `V(i)=i`. Speichern Sie die Komponenten 3 bis 7 in einem Feld `W`.

Lösung auf Seite 26

Achtung! Bei der Definition von Feldern wie wir sie eben besprochen haben erhalten wir eigentlich 1x`N`–Matrizen, also Felder als Zeilenvektoren. Wir können auch Felder als Spaltenvektoren definieren, indem wir ein Hochkomma `'` anhängen, was so viel bedeutet wie "transponiert". Oder wir definieren die Einträge mit Semikolons getrennt:

`[1 2 3]`	Zeilenvektor der Länge `N`
`[1 2 3]'`	Spaltenvektor der Länge `N`
`[1;2;3]`	Spaltenvektor der Länge `N`

Jedes Feld kann mit `'` transponiert werden. Wenn wir im Folgenden von Vektoren sprechen so meinen wir immer Zeilenvektoren, andernfalls kennzeichnet ein Hochkomma die andere Situation.

5: Mehrdimensionale Felder - Matrizen:

`[[1;2] [3;4]]`	Matrix bestehend aus den beiden Spaltenvektoren `[1;2]` und `[3;4]`. Also $\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$
`[1 2 ; 3 4]`	Matrix bestehend aus Zeilenvektoren `[1 2]` und `[3 4]`. Also $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

Eine 3x2x2–Matrix:

```
A(1,:,:) = [[1 2];[2 3]];
A(2,:,:) = [[2 3];[4 5]];
A(3,:,:) = [[1 0];[0 1]];
```

6: Felder durch Felder beschreiben:

```
>> V=[3 5]
V =
   3 5
```

```
>> W=[4 6]
W =
   4 6
```

```
>> A=[V W]
A =
   3 5 4 6
```

```
>> A=[V' W']
A =
   3   4
   5   6
```

```
>> S=[3;5]
S =
   3
   5
```

```
>> T=[4;6]
T =
   4
   6
```

```
>> B=[S T]
B =
   3   4
   5   6
```

Aufgabe 4 Matrizen

Probieren Sie selbst die obigen Definitionen aus.

Konventionen:

`V, W`	Vektor der Länge `N`
`A, C`	`NxN`–Matrix
`B`	`MxN`–Matrix

`k`	$\in \mathbb{N}_0$
`a, b`	$\in \mathbb{R}$

7: M–Files I (Programmdatei):

Wir erzeugen eine Programmdatei durch

```
>>edit first.m
```

Eine leere Datei namens `first.m` öffnet sich im Editor. Sie könne alle Definitionen und Anweisungen, die Sie in die Konsole geschrieben haben nun einfach in die Datei eintragen und diese dann durch den Aufruf

```
>>first
```

in der Matlabkonsole ausführen.

Tipp: Setzen Sie in die erste Zeile ein

```
clear all
```

so werden bei jedem Neustart Ihres Programms alle Variablen gelöscht und neu definiert. Sie beseitigen damit eine unangenehme Fehlerquelle.

Durch ein vorangestelltes % leiten Sie eine Kommentarzeile ein.

```
% Von hier bis zum Ende der Zeile wird alles ignoriert
```

Es empfiehlt sich, die Programme, die Sie schreiben, von vornherein gut zu kommentieren. Glauben Sie mir, Sie kommen nach zwei Wochen Ihren eigenen Gedankengängen nicht mehr auf die Spur ...

Aufgabe 5 Matlabdatei

Erzeugen Sie eine Matlabdatei `first.m`. Definieren Sie in der Datei verschiedene Felder und Matrizen gemäß des obigen Beispiels. Führen Sie Ihr Programm `first` aus.

8: arithmetische Operatoren (+ - * / ^):
Operationen zwischen Variablen:
Die Operationen `+,-,*,/,^` zwischen Variablen verwenden wir wie es im mathematischen Sinne üblich ist.

Operationen zwischen Feldern und Variablen:
Die Operationen `+,-,*` zwischen einer Variablen und einem Feld werden komponentenweise ausgeführt. Bei der Division `/` gilt das Gleiche, es ist lediglich darauf zu achten, dass eine Variable nicht duch ein Feld geteilt werden kann.

`V+a`	Der Wert `a` wird auf jede Komponente von `V` addiert.
`V-a`	Von `V` wird komponentenweise der Wert `a` subtrahiert.
`a-V`	`=-(V-a)`
`V/a`	Jede Komponente mit `a` dividiert.

Operationen zwischen Feldern:
Die Operationen `+,-,*` zwischen Feldern verwenden wir wie es im mathematischen Sinne üblich ist, wobei auf Zeilen– bzw. Spaltenform der Vektoren zu achten ist.

`V*W'`	Skalarprodukt von zwei gleichgroßen Vektoren
`B*A`	Matrix–Multiplikation
`V*A , B*V'`	Matrix–Vektor–Multiplikation
`B^k`	$\underbrace{B \cdots B}_{\text{k mal}}$

9: Beispiele von Multiplikationen zwischen Feldern:

Vektor–Vektor Multiplikation:

```
>> V=[1 2];

>> W=[2 4];

>> V*W'
ans =
  10

>> V'*W
ans =
   2   4
   4   8
```

Matrix–Matrix Multiplikation:

```
>> A=[[1 2];[0 1]]
A =
   1   2
   0   1
>> B=[[1 0];[0 2]]
B =
   1   0
   0   2
>> A*B
ans =
   1   4
   0   2
```

Matrix–Vektor Mutliplikation:

```
>> V=[1 2]
V =
   1   2
>> A*V'
ans =
   5
   2
```

10: besondere arithmentische Operatoren (.* .^ ./):

Wird der Operator mit einem vorangestelltem '.' bei Operationen zwischen Feldern verwendet, so erfolgt die entsprechende Verknüpfung komponentenweise.

`V.^b`	`(V(1)^b,...,V(N)^b)`
`V.*W`	`(V(1)*W(1),...,V(N)*W(N))`
`A./B`	$\begin{pmatrix} \frac{A_{11}}{B_{11}} & \dots & \frac{A_{1N}}{B_{1N}} \\ \vdots & \ddots & \vdots \\ \frac{A_{N1}}{B_{N1}} & \dots & \frac{A_{NN}}{B_{NN}} \end{pmatrix}$

Aufgabe 6 Variablen und Felder

Fügen Sie Ihrem Programm `first` folgendes hinzu:

1. Es sei `X` ein Feld mit `DIM` Koordinaten (äquidistant) in $[0, 1]$. Also `X=(0,.....,1)` mit `DIM` Einträgen.
2. Belegen Sie ein Feld `Y`, welches genau die Werte von `X` enthält. Zeichnen Sie den entsprechenden Grafen mit der `plot`–Funktion: `>>plot(X,Y)`
3. Belegen Sie ein Feld `W=X^2` und zeichnen Sie den Grafen zum obigen dazu mittels: `>>plot(X,Y,X,W)`

Lösung auf Seite 26

11: Einige vordefinierte Funktionen:

`abs(a)`	$\lvert a\rvert$	`sum(V)`	$\sum_{i=1}^{N} V(i)$
`sqrt(a)`	$\sqrt{a}$	`sum(A)`	$\left(\sum_{i=1}^{N} A(i,j)\right)_j$
`exp(a)`	e^a	`floor(a)`	rundet ab.
`log(a)`	$\ln a$	`ceil(a)`	rundet auf.
`log2(a)`	$\log_2 a$	`fix(a)`	rundet betragsmäßig ab.
`sin(a)`	$\sin a$	`round(a)`	rundet betragsmäßig auf.

Aufgabe 7 Euklidische Norm

Berechnen Sie die Euklidische Norm des Vektors

$$V = \begin{pmatrix} 2 \\ 4 \end{pmatrix}.$$

Lösung auf Seite 26

12: Hilfe, Dokumentation und Demos:

`help <begriff>`	Bietet eine Kurzhilfe zum Suchbegriff (falls vorhanden)
`doc`	Öffnet die Matlab–Dokumentation
`demo`	Öffnet die Matlab–Demos

Aufgabe 8 Hilfetext verwenden

Sie glauben – zu Recht – dass es eine Funktion zur Berechnung der Euklidischen Norm bei Matlab gibt. Finden Sie sie mithilfe der "help"-Funktion und berechnen Sie diese für obiges V direkt.

Berechnen Sie des weiteren für

$$A = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$$

die Normen $\|A\|_1$, $\|A\|_2$ und $\|A\|_\infty$.

Lösung auf Seite 27

13: spezielle Matrixformen:

`diag(V)`	ist eine Diagonalmatrix mit `V` als Diagonale
`diag(V(1:N-k),k)`	ist eine Matrix deren `k`–te obere Nebendiagonale mit den ersten (`N-k`)–ten Werten von `V` belegt wird.
`diag(V(1:N-k),-k)`	ist eine Matrix deren `k`–te untere Nebendiagonale mit den ersten (`N-k`)–ten Werten von `V` belegt wird.
`diag(A)`	ist ein Vektor der die Diagonalelemente von `A` enthält
`size(B)`	liefert einen Vektor `[M N]`, der die Anzahl der Zeilen (`M`) und Spalten (`N`) von `B` enthält.
`ones(N)`	liefert eine `NxN`–Matrix mit Einsen als Einträgen.
`ones(M,N)`	liefert eine `MxN`–Matrix mit Einsen als Einträgen.
`zeros(N)`	liefert eine `NxN`–Matrix mit Nullen als Einträgen.
`zeros(M,N)`	liefert eine `MxN`–Matrix mit Nullen als Einträgen.

Aufgabe 9 Matrizen

Es sei `DIM` eine zu definierende Variable. Editieren Sie für diese Aufgabe eine Datei `matrizen.m`.

1. Erzeugen Sie eine Matrix der Größe DIM x DIM mit Nulleinträgen.

2. Erzeugen Sie eine Diagonalmatrix der Größe DIM x DIM mit 2–en auf der Diagonale

3. Erzeugen Sie die Matrix
$$\begin{pmatrix} 0 & 3 & 0 \\ 0 & 0 & 3 \\ 0 & 0 & 0 \end{pmatrix}$$

4. Erzeugen Sie die Matrix
$$\begin{pmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix}$$

Lösung auf Seite 26

14: Weitere vorimplementierte Funktionen:

`A\V'`	"Division von links mit `A`" liefert die Lösung `W` der Gleichung $$AW^T = V^T \quad \Leftrightarrow \quad W^T = A^{-1}V^T$$
`A/V`	"Division von rechts mit `A`" liefert die Lösung `W` der Gleichung $$WA = V \quad \Leftrightarrow \quad W = VA^{-1}$$

Aufgabe 10 Lösen eines Gleichungssystems

1. Berechnen Sie die Lösung `X` des Gleichungssystems
$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

2. Berechnen Sie den Fehler in der Euklidischen Norm
$$\mathrm{err} = \|X - X_{\mathrm{ex}}\|_2$$
mit
$$X_{\mathrm{ex}} = \frac{1}{3}\begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix}.$$

Lösung auf Seite 27

1.3 For– und If–Else–Anweisung

15: logische und Vergleichsoperatoren:

logische Operatoren (& | ~):

&, |	logisches UND, ODER
~	Negation

Vergleichsoperatoren (== ~= <= >= < >):

==, ~=	gleich, nicht gleich
>, <	größer, kleiner
>=, <=	größer oder gleich, kleiner oder gleich

Aufgabe 11 Aussagenlogik

Überprüfen Sie einmal folgende Situationen:

$$
\begin{array}{lllll}
a = 1 & \Rightarrow & \sim a = & a = 2 & \Rightarrow \quad \sim a = \\
a = 0 & \Rightarrow & \sim a = & b = 2 & \Rightarrow \quad a \mid b = \\
 & & & & \Rightarrow \quad a \mid \sim b = \\
 & & & & \Rightarrow \quad \sim a \mid b = \\
 & & & & \Rightarrow \quad \sim (a \mid b) = \\
 & & & & \Rightarrow \quad a \,\&\, b = \\
 & & & & \Rightarrow \quad \sim a \,\&\, b = \\
 & & & & \Rightarrow \quad a \,\&\, \sim b =
\end{array}
$$

Und wie sieht's aus mit

$$((a \,\&\, \sim (a \mid b)) \mid \sim a) \,\&\, b = ?$$

Lösung auf Seite 28

16: If–Else– und Switch–Anweisungen:

```
if <ausdruck>
  <anweisung>
elseif <ausdruck>
  <anweisung>
else
  <anweisung>
end
```

```
switch <switch_ausdruck>
  case <case_ausdruck>
    <anweisung>
  case {<case_ausdruck1>, ...}
    <anweisung>
  otherwise
    <anweisung>
end
```

Der Ausdruck beinhaltet den Wahrheitsgehalt einer Aussage, wobei es sich bei der Aussage auch um die Verknüpfung mehrerer Aussagen mit den logischen oder den Vergleichsoperatoren handeln kann.

Hinter einem <case_ausdruck> verbirgt sich die Abfrage "if <switch_ausdruck> == <case_ausdruck>". Ein <switch_ausdruck> darf auch eine Zeichenkette beinhalten.

17: Beispiele von Switch–Anweisungen:

```
mode = 2;
switch mode
   case 1
     display('mode ist 1');
   case {2,3}
     display('mode ist 2 oder 3');
   otherwise
     display('unbekannter mode');
end
```

```
was_soll_ich_tun='was_rechnen';
switch was_soll_ich_tun
   case 'was_rechnen'
     display('ich rechne was');
   case 'nix','gar_nix'
     display('ich rechne nix');
   otherwise
     display('?');
end
```

Mehrere case_ausdrücke sind in {}–Klammern zu setzen und mit Kommata zu trennen.

Aufgabe 12 If–Else– und Switch–Anweisungen

Spielen Sie die obigen Beispiele einmal durch. Editieren Sie am besten eine Datei `second.m` dazu. Die Anweisung `display('bla')` gibt einfach die Zeichenkette `bla` an der Matlabkonsole aus. Sie dürfen gerne den Hilfetext zu dieser Anweisung (`>>help display`) lesen, um weitere Benutzmöglichkeiten zu erfahren.

18: For– und While–Schleife:

```
for variable = <ausdruck>
   <anweisung>
end
```

Unter einem <ausdruck> verstehen wir hier in der allgemeinsten Form

```
 Start: Step: End
```

Oder

```
 Start: End
```

Die Schrittweite ist dann automatisch 1.

```
while <ausdruck>
   <anweisung>
end
```

<ausdruck> bedeutet hier das gleiche wie bei der If–Abfrage.

19: Beispiele für For– und While–Schleifen:

```
for i = 1:N
  for j = 1:N
    A(i,j) = 1/(i+j-1);
  end
end
```

```
while error<tol
  solve_the_equation
  error=error_of_solution
end
```

Eine For–Schleife darf auch rückwärts laufen und in eine Zeile geschrieben werden:

```
 for S = 1.0: -0.1: 0.0, do_something, end
```

Dort, wo wir bisher eine neue Zeile begonnen haben deutet nun ein Komma auf das insgeheime Vorhaben hin.

Aufgabe 13 For–Schleife

Versuchen Sie mal das:

```
for i=0:-0.1:-1, display(i),end
```

und das:

```
for i=0:0.1:-1, display(i),end
```

oder das:

```
for i=0:0.1:-1, display(i),end
```

Wo ist was und warum falsch?

Aufgabe 14 Summation

Schreiben Sie ein Programm (`summe.m`), welches zu vorgegebenem `N` die Summe

$$S = \sum_{i=1}^{N} i$$

mittels einer For–Schleife berechnet. Prüfen Sie Ihr Ergebnis mit dem "kleinen Gauß":

$$\sum_{i=1}^{N} i = \frac{N(N+1)}{2}$$

Lösung auf Seite 28

Aufgabe 15 Fakultät

Schreiben Sie eine while–Schleife, die die Fakultät

$$n!$$

berechnet. Erzeugen Sie dazu eine neue Datei fak.m Lösung auf Seite 28

1.4 Strukturierte Programmierung

20: M–Files II (Funktionsdatei):

Datei fun.m:

```
function [V,W] = fun(X,Y)
% Hilfetext: Der hier aufgefuehrte Text
% erscheint, wenn man in der Konsole
%   >>help fun
% aufruft.
V = X.^2 + Y;
W = X.^3;
```

Die Funktion erwartet als Eingangsparameter `X` und `Y`. `X, Y` dürfen jeweils skalare Werte sein oder auch Felder; im Grunde alle Formen, die innerhalb der Funktion in den Rechenoperationen berücksichtig sind (beachte .^)
Die Funktion `fun` gibt zwei Variabeln oder Felder `V` und `W` zurück.

Variabeln und Felder sind immer lokal definiert. Das heiśt, wenn wir eine Variable `X` im Hauptprogramm definiert haben und eine mit dem gleichen Namen in der Funktion `fun` so stören diese sich gegenseitig nicht; sie wissen gar nichts voneinander. Wir können also die Werte von `X` in `fun` verändern ohne, dass die Werte des Feldes dessen Namen wir an `fun` übergeben haben verändert werden.

Aufgabe 16 Funktionen und lokale Variablen

Schreiben Sie eine vereinfachte Funktion `fun` wie oben, wobei Sie `Y` und `W` weglassen.. Rufen Sie diese innerhalb eines Programms `third.m` auf. Setzen Sie in der Programmdatei `X=0:0.1:1`, `Y=fun(X)` und plotten Sie `Y` über `X` mittels `plot(X,Y)`.

So. Jetzt verändern Sie mal die Werte von `X` innerhalb der Funktion `fun`, etwa alles 0, und schauen nach verlassen der Funktion, welche Werte `X` dann wieder hat? Sollte sich nichts verändert habe, das heiśt es sollten immer noch Werte von 0 bis 1 enthalten sein.

Lösung auf Seite 29

Wollen wir hingegen eine Variable oder ein Feld global bekanntbegen so müssen wir es entsprechend deklarieren:

21: Globale Variablen:

`global X Y`	deklariert die Variablen/Felder `X` und `Y` global. Wo immer diese bekannt sein sollen, müssen sie entsprechend deklariert werden.

22: Beispiel für globale Variablen:

In der Hauptdatei `main.m`:

```
...
global DIM A B
A=0; B=1; DIM=10;
Y=funct;
...
```

In der Funktionsdatei `funct.m`:

```
function F=funct
global DIM A B
X=A:1/DIM:B;
F=X.^2;
```

Aufgabe 17 Globale Variablen

Schreiben Sie ein Programm `fourth.m` und eine Funktionsdatei `funct.m` wie im obigen Beispiel.

Als Übergabeparameter an eine Funktion kann auch ein Funktionsname verwendet werden. Wir betrachten das mal an einem einfachen Beispiel. Sie wollen näherungsweise den Wert eines Integrals berechnen für verschiedene Integranten und streben eine recht profane Quadraturformel an (hier geht's ja jetzt nur um's Prinzip!):

$$\int_0^1 f(x)\,dx \approx h \sum_{i=1}^{N-1} f(x_i)\,, \text{wobei} \quad x_1 = 0\,,\ x_N = 1$$

23: Zeiger auf Funktionen:

Die Funktionsdateien:

Datei `fun.m`:

```
function Y=fun(X)
% fun liefert als Funktionswert
% das Quadrat der eingelesenen
% Variablen
Y = X.^2
```

Datei `quadr.m`

```
function Q=quadr(fun,a,b,N)
% quadr berechnet das Integral
der
% Funktion fun von x=a bis b
mittels
% einer Quadraturformel mit N
% aequidistanten Stuetzstellen

h=(b-a)/N;
X=a:h:b;
F=function(X);

Q=h*sum(F(1:N-1));
```

Aufruf in der Programmdatei:

```
quadr(@fun,0,1,100);
```

1.5 Grafische Darstellung

24: Plot Beispiele: aus Abbildung 1.5

```
myfirstplot=figure(1);

subplot(2,1,1)
x= 0:0.1:2*pi;
plot(x,sin(x),'g*')
axis([0 2*pi -1.1 +1.1])

subplot(2,1,2)
x = 0:0.01:2;
y = 5*x.*exp(-3*x);
yn = y + 0.02*randn(size(x));
plot(x,y,'-',x,yn,'ro');
xlabel('x (arbitrary units)');
ylabel('y (arbitrary units)');
title('Plot of y=5*x*exp(-3*x)+noise');
legend('true y','noisy y');
```

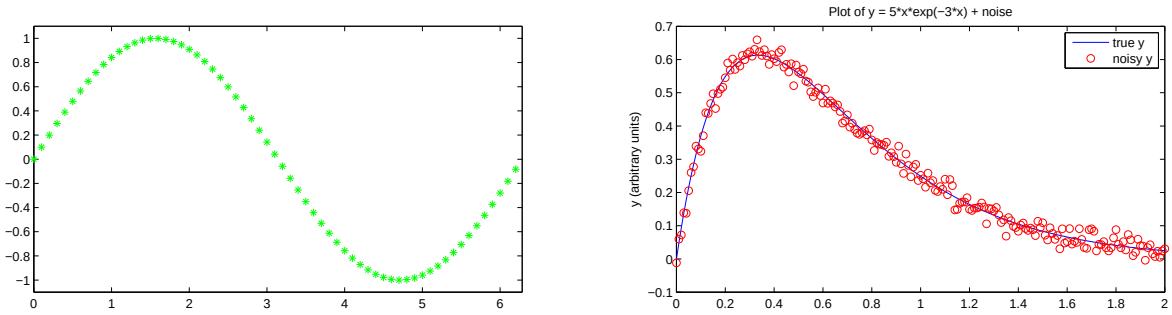


Abbildung 1: Plot Beispiele 1D

25: Plot Beispiele: aus Abbildung 1.5

```
mylastplot=figure(4);
[X,Y] = meshgrid([-2:.1:2]);
Z = X.*exp(-X.^2-Y.^2);

subplot(1,3,1)
surf(X,Y,Z)
title('surf: 3-D colored surface')

subplot(1,3,2)
contour3(X,Y,Z)
title('contour3: 3-D contour plot')

subplot(1,3,3)
contourf(X,Y,Z)
title('contourf: filled contour plot')
```

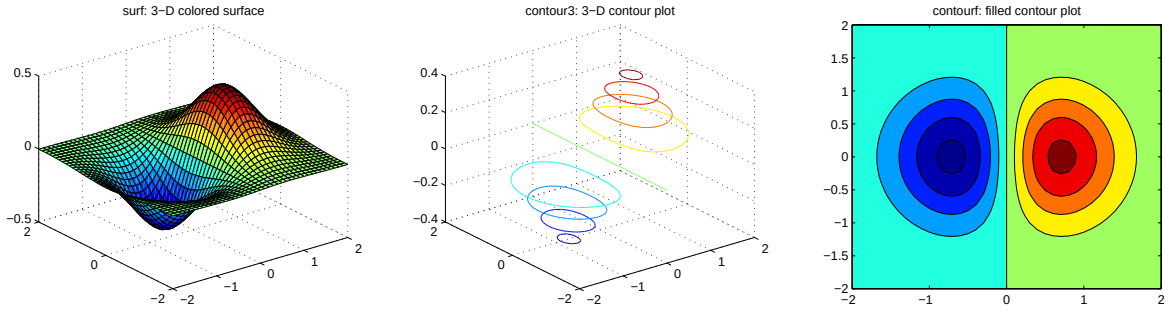


Abbildung 2: Plot Beispiele 3D

Aufgabe 18 Plot Beispiele

Auf

`N:\users\staff\axtr\unix\public\MNT\Matlab\PlotExamples`

finden Sie obige und andere Beispiele zum ausprobieren.

1.6 Schreiben und Lesen von Dateien

Matlab bietet eine umfangreiche sammlung von Einlese- und Schreibroutinen für Dateien in den verschiedensten Formaten. An dieser Stelle sind nur einige wenige Beispiele aufgef uhrt, damit Sie den Einstieg finden können. Sie werden zusätzlich des öfteren darauf verwiesen werden, sich weitere Details aus den Matlab–Hilfetexten zu besorgen.

Wir werden zwischen speziellen `mat`–Dateien und Datendateien unterscheiden. Die C–Programmierer unter Ihnen werden sich darüber erfreuen, dass die Dateiverwaltung auch bei unter Matlab mithilfe der Funktionen `scanf` und `fprintf` bewerkstelligt werden kann.

26: In eine Datei schreiben:

Der Befehl

```
> save mydata.mat
```

speichert alle definierten Variablen in die `mat`–Datei `mydata.mat`. Der Aufruf von

```
> load mydata
```

lädt die gespeicherten Variablen wieder.

Genau wie bei der Programmiersprache C können die Funktionen `fprintf` und `fscanf` verwendet werden.

```
> fid=fopen(<dateiname>,<permissions>);
> fscanf(fid,<format>,<size>);
```

oder

```
> fprintf(fid,<format>,...);
> fclose(fid);
```

Zur genaueren Beschreibung empfehle ich abermals die Hilfeseite `> help fprintf`, etc. oder zum einfachen Gebrauch das Beispiel auf Seite 24.

27: Daten aus einer Datei einlesen:

Der Befehl `load` kann auch dazu verwendet werden, einen gespeicherten Datensatz aus einer Datei einzulesen.

```
> Q=load('datei.dat')
```

liest den Inhalt der Datei `datei.dat` ein und weist diesen der Variable `Q` zu. Jede Zeile des Dateiinhalts muss aus gleich vielen Spaltenzahlen bestehen. Jedes einzulesende Zeichen muss eine Zahl sein.

Tipp: Es gibt eine Vielzahl an Gebrauchmöglichkeiten der `load`–Funktion, die unter `> help load` nachzulesen sind.

28: Beispiel für den Gebrauch der Befehle `load` und `save` für `mat`–Dateien:

```
> A=[[1 2];[3 4]];
> S=2/3;
> wort='hallo';
> whos
 Name  Size  Bytes  Class   Attributes

 A     2x2      32  double
 S     1x1       8  double
 wort  1x5      10  char
> save mydata.mat
> clear
> whos
> load mydata.mat
> whos
 Name  Size  Bytes  Class   Attributes

 A     2x2      32  double
 S     1x1       8  double
 wort  1x5      10  char
```

29: Einlesen einer Datei mit `fscanf` und `fgets`:

```
> fid=fopen('datei.dat','r');
> S=fgets(fid);
> Q=fscanf(fid,'%d',[2,3]);
> fclose(fid);
> Q

Q =

 1 3 5
 2 4 6
> S

S =

 ich bin eine Zeichenkette
```

Der Inhalt der Datei im obigen Beispiel war

```
ich bin eine Zeichenkette
1 3 5
2 4 6
```

Die gleiche Datei lesen wir mit einem anderen Befehl ein, nämlich mit `importdata`. Die Funktion sucht den Inhalt der Datei nach bekannten Strukturen, wie sie etwa bei bestimmten Bildformaten gegeben sind. Sie sammelt Klassen von Zeichenstrukturen und belegt entsprechende Felder. Im folgenden Beispiel wird eine Struktur A mit dem Inhalt der Datei belegt. Die Komponente `textdata` beinhaltet die Zeichenkette in der ersten Zeile der Datei und die Komponente `data` die 2×3–Matrix mit den gespeicherten integer Werten.

30: Einlesen einer Datei mit `importdata`

```
A=importdata('bla.dat');
> A
A =

   data: [2x3 double]

   textdata: 'ich bin eine Zeichenkette'
> A.data
ans =

   1 2 3

   4 5 6
> A.textdata
ans =

   'ich bin eine Zeichenkette'
```

Eine ausührliche Beschreibung zu den vielseitigen Möglichkeiten in Matlab Dateien verschiedenster Formate einzulesen und herauszuschreiben finden Sie beim Aufruf von

```
> doc help
```

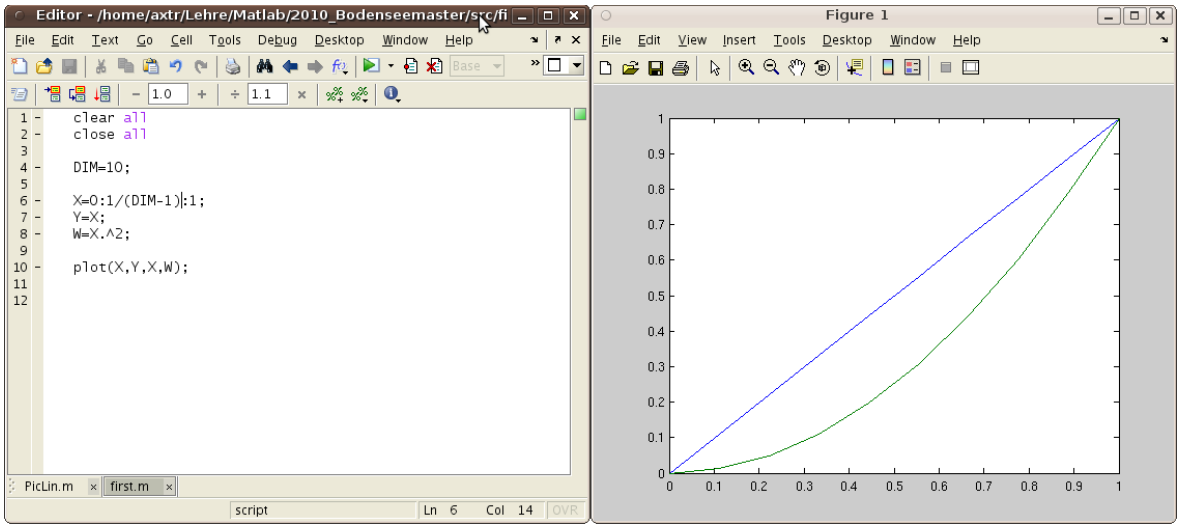
unter `MATLAB`→`Functions - By Category`→`File I/O`

1.7 Lösungen

Lösung 3

```
>> V=1:1:10
V =
   1 2 3 4 5 6 7 8 9 10
>> W=V(3:7)
W =
   3 4 5 6 7
```

Lösung 6



Tipp: Zum Befehl `clear all` zu Beginn Ihrer Programmdatei fügen Sie am besten noch den Befehl `close all`. Damit werden bei Programmneustart alle offenen Fenster geschlossen und neu erzeugt.

Lösung 9

Datei `matrizen.m`:

```
DIM=4;

A0 = zeros(DIM);
A1 = diag(2*ones(DIM,1));
A2 = diag(3*ones(1,DIM-1),1);
A3 = diag(-1*ones(1,DIM-1),-1);
```

Lösung 7

```
>> V=[2 4]';
>> sqrt(sum(V.^2))
```

```
ans =
   4.472
```

Lösung 8

Der Aufruf

```
>> help norm
```

liefert unter anderem

```
NORM(X,P) is available for matrix X only if P is 1, 2, inf or 'fro'.
NORM(X,2)=NORM(X)
```

Also

```
>> norm(V)
ans =
   4.4721
```

Die Matrix:

```
>> A=[[1 2];[0 1]];
A =
   1 2
   0 1
>> norm(A,2)
ans =
 2.4142
>> norm(A,1)
ans =
   3
>> norm(A,inf)
ans =
   3
```

Lösung 10

1. ```
   >> X=A\V
   X =
      0.6667
      -0.3333
      0.3333
   ```
   ```

2.
```
>> Xex=[2;-1;1]/3
Xex =
   0.6667
   -0.3333
   0.3333
>> err=norm(X-Xex)
err =
   1.3597e-16
```

Lösung 11

$$
\begin{array}{llllll}
a = 1 & \Rightarrow & \sim a = 0 & a = 2 & \Rightarrow & \sim a = 0 \\
a = 0 & \Rightarrow & \sim a = 1 & b = 2 & \Rightarrow & a \mid b = 1 \\
 & & & & \Rightarrow & a \mid \sim b = 0 \\
 & & & & \Rightarrow & \sim a \mid b = 1 \\
 & & & & \Rightarrow & \sim (a \mid b) = 0 \\
 & & & & \Rightarrow & a \,\&\, b = 0 \\
 & & & & \Rightarrow & \sim a \,\&\, b = 1 \\
 & & & & \Rightarrow & a \,\&\, \sim b = 0
\end{array}
$$

$$((a \,\&\, \sim (a \mid b)) \mid \sim a) \,\&\, b = 1$$

Lösung 14

```
N=12;                          % Summengrenze
S=0;                           % Startwert fuer die Summation

for i=0:N
    S = S + i;
end

KG = N*(N+1)/2;                % der kleine Gauss

display(sprintf('sum(1:%d)=%d, KG=%d',N,S,KG));
```

Lösung 15

```
n  = 6;                    % zu berechnende Fakultaet
no = n;                    % fuer die Ausgabe merken
nf = 1;                    % Startwert fuer das Produkt

while n>0
    nf = nf*n;
    n  = n-1;
end

display(sprintf('%d! = %d',no,nf));
```

Lösung 16

Datei `third.m`:

```
X=0:0.1:1;
Y=fun(X);

plot(X,Y);
```

Funktionsdatei `fun.m`:

```
function Y=fun(X)

X=diag(ones(length(X)));
Y=X.^2;
```

–2–
Numerische Berechnung der Wärmeleitungsgleichung: Teil I

2.1 Das kontinuierliche Problem: stationäre Wärmeleitungsgleichung

Wenn alle Einflussfaktoren der Wärmeleitungsgleichung zeitunabhängig sind, so wird sich irgendwann ein stationärer Zustand für die Wärmeverteilung einstellen. Es gilt dann $T_t = 0$. Wir gehen von genau dieser Situation aus und betrachten daher das folgende stationäre Problem:

Es sei $I = [0, 1] \subset \mathbb{R}$. Gesucht ist eine Funktion $T : I \to \mathbb{R}$, die zu gegebener Funktion $q : I \to \mathbb{R}$ die Gleichung

$$-T''(x) = q(x) \qquad \text{in } I$$

mit den Randwerten

$$T(0) = T_l$$
$$T(1) = T_r$$

zu $T_l, T_r \in \mathbb{R}$ erfüllt.

2.2 Das diskrete Problem: Finite–Differenzen Methode

Die Diskretisierung besteht im Wesentlichen darin, dass wir die auftretenden Ableitungen durch Differenzenquotienten approximieren. Dazu zerlegen wir sunächst das Intervall I in $N - 1$ gleichgroße Teilintervalle, so dass

$$I = \bigcup_{i=1}^{N-1} I_i \quad \text{mit} \quad I_i = [x_i, x_{i+1}]$$

gilt, wobei $x_1 = 0$ und $x_N = 1$ ist. Funktionsauswertungen an den Stützstellen x_i kennzeichnen wir mit fettgedruckten Buchstaben, das heißt es gelte

$$\mathbf{T}_i = T(x_i) \quad \text{und} \quad \mathbf{q}_i = q(x_i) \quad \text{für} \quad i = 1, \ldots, N\,.$$

Alle Funktionswerte an den Stützstellen fassen wir in jeweils einem Vektor zusammen:

$$\mathbf{T} = (\mathbf{T}_1, \ldots, \mathbf{T}_N)$$
$$\mathbf{q} = (\mathbf{q}_1, \ldots, \mathbf{q}_N)$$

Wir beginnen nun mit der Berechnung der Differenzenquotienten für $T''(x)$. Betrachten wir zunächst nur eine Ableitung. Um einen Differenzenquotienten von $T(x)$ an der Stelle x_i aufzustellen gibt es drei Möglichkeiten:

Vorwärtsdifferenzenquotient:

$$T'(x_i) \approx T'_+(x_i) = \frac{T(x_i + h) - T(x_i)}{h} = \frac{T(x_{i+1}) - T(x_i)}{h} = \frac{1}{h}\left(\mathbf{T}_{i+1} - \mathbf{T}_i\right)$$

Rückwärtsdifferenzenquotient:

$$T'(x_i) \approx T'_-(x_i) = \frac{T(x_i) - T(x_i - h)}{h} = \frac{T(x_i) - T(x_{i-1})}{h} = \frac{1}{h}\left(\mathbf{T}_i - \mathbf{T}_{i-1}\right)$$

Mittelwert aus Vorwärts– und Rückwärtsdifferenzenquotienten:

$$T'(x_i) \approx T'_\pm(x_i) = \frac{1}{2}\left(T'_+(x_i) + T'_-(x_i)\right) = \frac{1}{2h}\left(\mathbf{T}_{i+1} - \mathbf{T}_{i-1}\right)$$

wobei

$$h = \frac{1}{N-1}\,.$$

Wecher der Differenzenquotienten gewählt wird ist situationsabhängig. Will man nur die erste Ableitung approximieren so wählt man $T'_\pm(x_i)$ für die inneren Knoten ($i = 2, \ldots, N-1$), $T'_+(x_i)$ für den linken Randknoten $i = 1$ und $T'_-(x_i)$ für den rechten Randknoten $i = N$.

Wir benötigen zwei Ableitungen. Für die erste Ableitung verwenden wir den Vorwärtsdifferenzenquotienten und für die zweite Ableitung den Rückwärtsdifferenzenquotienten:

Rückwärtsdifferenzenquotient für die zweite Ableitung:

$$T''(x_i) \approx \frac{T'(x_i) - T'(x_{i-1})}{h}$$

Vorwärtsdifferenzenquotient für die erste Ableitung:

$$\begin{aligned}
&\approx \frac{\frac{T(x_{i+1}) - T(x_i)}{h} - \frac{T(x_i) - T(x_{i-1})}{h}}{h} \\
&= \frac{T(x_{i-1}) - 2\,T(x_i) + T(x_{i+1})}{h^2} \\
&= \frac{1}{h^2}\left(\mathbf{T}_{i-1} - 2\,\mathbf{T}_i + \mathbf{T}_{i+1}\right)
\end{aligned}$$

Für alle Knoten $i = 1, \ldots, N$ können wir diesen Ausdruck in Matrix–Vektor Schreibweise formulieren:

$$\frac{1}{h^2}\begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & \ddots & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix}\begin{pmatrix} \mathbf{T}_1 \\ \vdots \\ \vdots \\ \vdots \\ \mathbf{T}_N \end{pmatrix} = \begin{pmatrix} \mathbf{q}_1 \\ \vdots \\ \vdots \\ \vdots \\ \mathbf{q}_N \end{pmatrix}$$

Sie haben sicher bemerkt, dass wir an den Randknoten nicht ganz sauber gearbeitet haben. Das macht aber nichts. Sie erinnern sich, dass wir noch Randwerte fordern. Die müssen wir jetzt noch in unser System einbauen, womit wir die Randsituation ohnehin völlig neu gestalten. Wir können uns also guten Gewissens diesen kleinen Lapsus verzeihen.

2.3 Randwerte für das diskrete Problem: Dirichlet Randwerte

Wir haben die zweite Ableitung aus unserem kontinuierlichen Problem diskretisiert und fordern nun zusätzlich Randbedingungen, nämlich $T(0) = T_l$ und $T(1) = T_r$. Wir zwingen unser Gleichungssystem dazu diese Bedingung einzuhalten, indem wir die Matrix entsprechend modifizieren. Das diskrete Pendant zu unserem kontinuierlichen Problem lautet dann:

Gesucht sind die Funktionswerte $\mathbf{T}_i = T(x_i), i = 1, \ldots, N$ mit

$$\frac{1}{h^2}\begin{pmatrix} h^2 & 0 & \cdots & \cdots & 0 \\ -1 & 2 & -1 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & -1 & 2 & -1 \\ 0 & \cdots & \cdots & 0 & h^2 \end{pmatrix}\begin{pmatrix} \mathbf{T}_1 \\ \vdots \\ \vdots \\ \vdots \\ \mathbf{T}_N \end{pmatrix} = \begin{pmatrix} T_l \\ \mathbf{q}_2 \\ \vdots \\ \mathbf{q}_{N-1} \\ T_r \end{pmatrix}. \tag{1}$$

Auf diese Weise kann die Lösung gar nicht anders, als die Dirichlet Randwerte zu erfüllen.

–3–
Aufgabe I: Wärmeleitung mit Dirichletrand

3.1 Die Aufgabenstellung

In den folgenden Schritten werden wir die Wärmeleitgleichung für den stationären Fall lösen. Dazu betrachten wir einen Stab mit der Länge L, der links und rechts eine vorgegebene Temperatur T_l beziehungsweise T_r hat. Der Mantel des Stabes ist perfekt isoliert. Es fliesst somit keine Wärme über die Isolation ab. Dies führt dazu, dass wir die Temperatur nur in Längsrichtung des Stabes betrachten und reduzieren das Modell daher auf eine Dimension. In der folgenden Gleichung bezeichnet T die örtliche Temperaturverteilung, q die Wärmequelle und κ den Wärmeleitkoeffizienten:

$$-\kappa T''(x) = q, \tag{2}$$
$$T(0) = T_l, \tag{3}$$
$$T(L) = T_r. \tag{4}$$

3.1.1 Grundgerüst

Als Grundgerüst für die Berechnung erzeugen sie ein m-File, das alle nötigen Funktionen für die Berechnung enthalten wird.

```
function solveheatequation(Len,Dim,Tl,Tr,Kappa,Csource)
 ...
 end
```

Speichern sie das File unter dem entsprechenden Namen ab und rufen sie es mit einem Parametersatz im Command-Fenster auf.

3.1.2 Diskretisierung

Erstellen sie unterhalb der vorherigen Funktion (im gleichen m-File) eine weitere Funktion, die ihnen die Gitterweite h und die Diskretisierungspunkte aus der Länge und Dimension berechnet.

```
function [grid,h,npts]=discretize(length,dim)
 input: length of domain (length, 1 × 1),
```

```
    number of intervals (dim, 1 × 1)
output: grid points (grid, (dim + 1) × 1),
  distance (h, 1 × 1) and number of grid points (npts, 1 × 1)
```

3.1.3 Quellterm

Erstellen sie eine weitere Funktion, in der sie eine konstante Wärmequelle an jedem Punkt vorgeben.

```
function [source]=specifysource(npts,kappa,constval)
input: number of grid points (npts,1 × 1),
  coefficient kappa (kappa,1 × 1),
  constant source (constval,1 × 1)
output: source term (source,(dim + 1) × 1)
```

3.1.4 Matrix

Erstellen sie eine Funktion, in der sie die Matrix aus den finiten Differenzen erzeugen.

```
function [mat]=calcmatrix(h,dim)
input: h (1 × 1), dim (1 × 1)
output: matrix from finite difference scheme (mat,(dim + 1) × (dim + 1))
```

3.1.5 Einbinden der Randbedingungen

Ergänzen sie die Matrix und die rechte Seite mit den Randbedingungen.

```
function [rhs,mat]=addboundary(mat,source,tleft,tright,dim)
 input: mat ((dim + 1) × (dim + 1)), source ((dim + 1) × 1),
  temperature on left side (tleft, 1 × 1),
  temperature on right side (tright, 1 × 1),
  dim (1 × 1)
output: right handside (rhs, (dim + 1) × 1),
  mat (dim + 1) × (dim + 1)
```

3.1.6 Lösen des Gleichungssystem

Rufen sie nun die obigen Funktionen in der Hauptfunktion

```
solveheatequation(Len,Dim,Tl,Tr,Kappa,Csource)
```

in der richtigen Reihenfolge auf. Lösen sie das Gleichungssystem mit der berechneten Matrix und rechter Seite auf und sie erhalten die Temperaturverteilung im Stab.

3.1.7 Darstellen der Lösung

Als nächster Schritt stellen sie die berechnete Lösung als Funktion des Ortes dar. Als Kontrolle können sie zusätzlich noch die analytische Lösung plotten. Ausserdem können sie den Fehler zwischen der numerischen und analytischen Lösung berechnen. Bilden sie dazu die euklidische Norm des Fehlers und teilen sie ihn durch die Anzahl Punkte, an denen die Lösung berechnet wurde:

$$\frac{1}{N}||T_{true} - T_{num}||_2. \tag{5}$$

Was passiert, wenn sie die Distanz zwischen den Punkten verändern?

3.1.8 Weitere Spielereien...

Zusätzlich können sie auch noch den Wärmefluss berechnen und darstellen. Schlagen sie dazu die Definition nach und erweitern sie ihre Berechnung damit.
Im weiteren können sie ihre Wärmequelle in der entsprechenden Funktion verändern. Was beobachten sie?

3.2 Lösungen

Das einfachste Beispiel ist gegeben durch die Wahl von $q = 0$. Wir erhalten als Lösung für Randwerte 21° und 100° eine Temperaturverteilung wie sie in Abbildung 3 dargestellt ist. Das ist intuitiv korrekt, denn wir suchen etwas dessen zweite Ableitung verschwindet ($-T'' = 0$). Die Lösung ist also erwartungsgemäß eine lineare Funktion. Etwas anders verhält es sich, wenn wir einen Quellterm hinzuziehen, also $q \neq 0$ gilt. Sagen wir etwas erhitzt den Stab in einem Bereich in der Mitte. Wir definieren einmal ein entsprechendes q, wobei bei wir auf die physikalische Bedeutung und Größe mal eben gar keine Rücksicht nehmen wollen:

$$q(x) := \begin{cases} 1 & \text{falls } x \in [0.4, 0.6] \\ 0 & \text{sonst} \end{cases}$$

Abbildung 4 zeigt die resultierende Lösung, wobei wir an den Stabenden von einer Raumtemperatur von 21° ausgehen, einen Graf des Flusses T_x und der des Quellterms q.

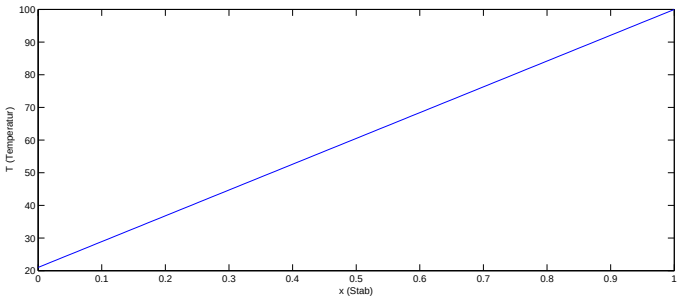


Abbildung 3: $\kappa = 1$, $q = 0$, $T_l = 21$, $T_r = 100$

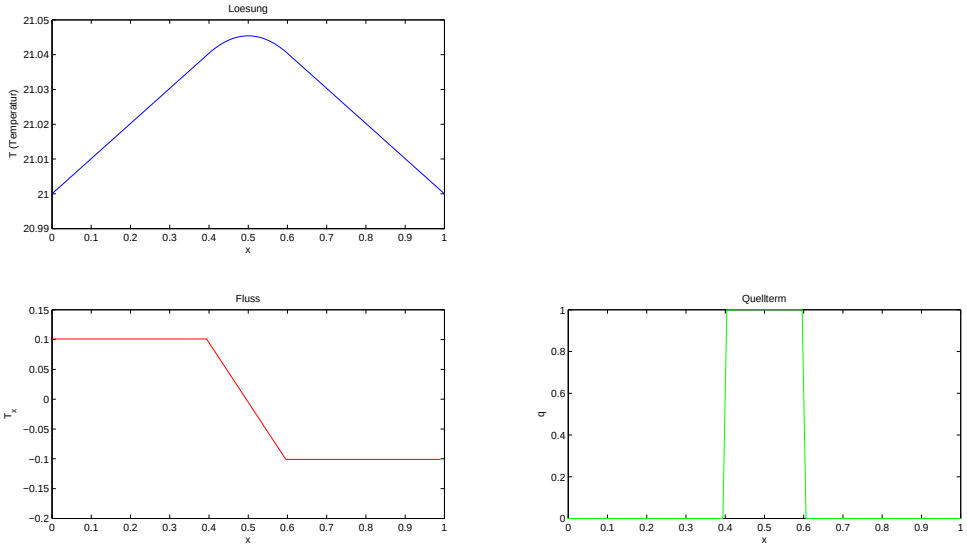


Abbildung 4: $\kappa = 1$, $q = 1$ in $[0.4 : 0.6]$ sonst 0, $T_l = T_r = 21$

–4–
Numerische Berechnung der Wärmeleitungsgleichung: Teil II

Wir betrachten nun eine Situation, die der in Kapitel 2 ganz ähnlich ist mit dem Unterschied, dass wir an einem der beiden Ränder des Intervalls I eine andere Situation fordern. Nehmen wir den linken Rand $x = 0$ (Wir können genausogut den rechten Rand wählen). Wir schreiben nicht eine konkrete Temperatur dort vor, wie es bei der Dirichlet Randbedingung der Fall ist sondern wir schreiben vor, wie der Fluss dort aussieht. Wir legen also einen Wert für $T'(0)$ fest. Eine solche Randbedingung heißt Neumann Randwert. Das Problem formuliert sich dann als

Es sei $I = [0, 1] \subset \mathbb{R}$. Gesucht ist eine Funktion $T : I \to \mathbb{R}$, die zu gegebener Funktion $q : I \to \mathbb{R}$ die Gleichung

$$\begin{aligned} -T''(x) &= q(x) \quad \text{in } I \\ T'(0) &= N_l \qquad \text{auf } \{0\} \\ T(1) &= T_r \qquad \text{auf } \{1\} \end{aligned} \tag{6}$$

zu $N_l, T_r \in \mathbb{R}$ erfüllt.

4.1 Randwerte für das diskrete Problem: Neumannrandwerte

Genau wie wir beim Problem mit Dirichlet Randwerten vorgegangen sind tun wir das nun auch für die neue Form von Randwerten. Wir zerlegen unser Intervall I in $N - 1$ Teilintervalle I_i mit $I = \bigcup_{i=1}^{N-1} I_i$. Wir halten uns an die Notationen in Kapitel 2. Dann beschreibt die folgende Matrix–Vektor Darstellung

$$\frac{1}{h^2} \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & \ddots & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} \mathbf{T}_1 \\ \vdots \\ \vdots \\ \vdots \\ \mathbf{T}_N \end{pmatrix} = \begin{pmatrix} \mathbf{q}_1 \\ \\ \vdots \\ \\ \mathbf{q}_N \end{pmatrix}. \tag{7}$$

für alle Knoten $i = 1, \ldots, N$ die Diskretisierung der Gleichung $-T'' = q$, wobei wir wie in der vorherigen Zituation in der ersten und letzten Zeile nicht ganz korrekt liegen. Aber auch in diesem Fall macht das nichts, denn wir werden wiederum genau diese Zeilen durch die Randbedingungen ersetzen.

Nun müssen noch die Randwerte in das System eingebaut werden. Den Dirichlet Randwert bei $x = 1$ erhalten wir genauso wie in (1).

$$\begin{aligned} & T(x_N) = T_r \\ \Leftrightarrow \quad & \mathbf{T}_N = T_r \end{aligned} \tag{8}$$

Bleibt also noch die Frage nach dem Neumannrand. Wir gehen dabei genauso naiv vor: Wir fordern, dass $T'(0) = N_l$ ist. Diskret sieht das dann so aus:

$$\begin{aligned} & \frac{T(x_2) - T(x_1)}{x_2 - x_1} = N_l \\ \Leftrightarrow \quad & \frac{1}{h^2}(-h\,\mathbf{T}_1 + h\,\mathbf{T}_2) = N_l \end{aligned} \tag{9}$$

Wir bauen (8) in die letzte und (9) in die erste Zeile des LGS (7) ein und erhalten damit das vollständig diskretisierte Gleichungssystem

$$\frac{1}{h^2}\begin{pmatrix} -h & h & 0 & \cdots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 2 & -1 \\ 0 & \cdots & 0 & 0 & h^2 \end{pmatrix}\begin{pmatrix} \mathbf{T}_1 \\ \vdots \\ \vdots \\ \vdots \\ \mathbf{T}_N \end{pmatrix} = \begin{pmatrix} N_l \\ \mathbf{q}_2 \\ \vdots \\ \mathbf{q}_{N-1} \\ T_r \end{pmatrix}. \tag{10}$$

–5–
Aufgabe II: Wärmeleitung mit Neumannrand

5.1 Zusatzaufgabe zu Aufgabe 1 in Kapitel 3.1

Führen Sie eine globale Variable `Boundary` ein, die ein 2×1 Feld mit zwei Zeichen darstellt. Das erste Zeichen (`Boundary(1)`) ist `'N'` für Neumann– und `'D'` für Dirichletrand am linken Rand. Das zweite Zeichen (`Boundary(2)`) ist `'N'` für Neumann– und `'D'` für Dirichletrand am rechten Rand.

Berechnen Sie die diskrete Lösung von Problem 6 mit $T_x(0) = 0, T(1) = 21$ und $q(x) = 1$ in $[0.4 : 0.6]$ sonst 0.

Eine Darstellung der Lösung ist in Abbildung 5 gegeben.

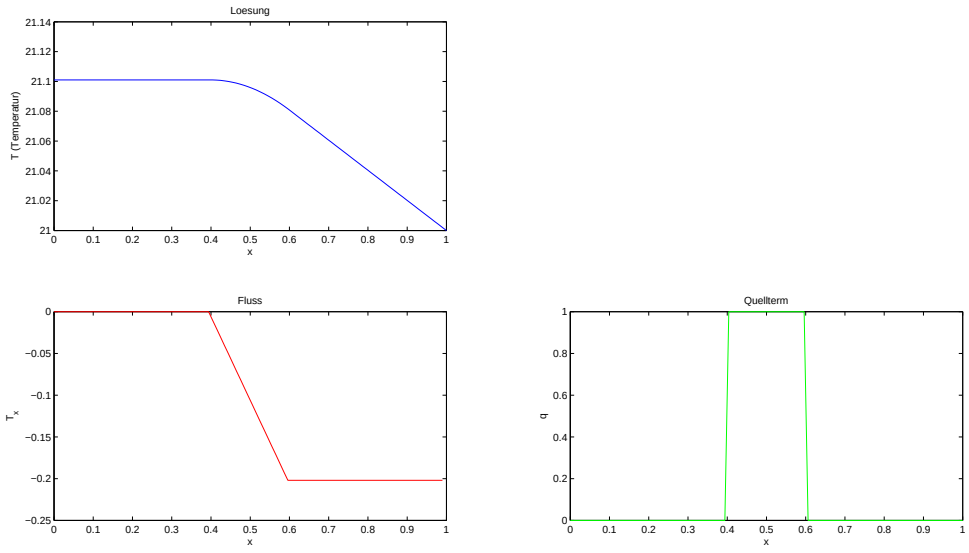


Abbildung 5: $\kappa = 1$, $q = 1$ in $[0.4 : 0.6]$ sonst 0, $n_l = 0$, $T_r = 21$

A Alternative Software

A.1 Scilab

A.1.1 Where to get

URL:

http://www.scilab.org

download:

http://www.scilab.org/products/scilab/download

A.1.2 Syntaxunterschiede zu Matlab

Introduktion:

http://www.scilab.org/content/download/1104/10840/file/introscilab.pdf

Manual:

http://www.scilab.org/product/man

Unterschiede zu Matlab

Topic	Scilab	Matlab
Dateiendung	`.sci`	`.m`
Programmaufruf	`>>exec datei.sci`	`>> datei`

A.2 Octave

A.2.1 Where to get

URL:

http://www.gnu.org/software/octave

Windows Installer:

http://www.gnu.org/software/octave

→ Octave_Windows - MinGW

→ → Octave 3.2.4 for Windows MinGW32 Installer

→ → → Octave-3.2.4_i686-pc-mingw32_gcc-4.4.0_setup.exe

Installationhinweis:

Wärend des setups kommt irgendwann (im 4. Schritt) die Aufforderung `Choose Componentes`. An der stelle sollte man ein Häkchen bei `gnuplot` machen. Dieses Grafikprogramm wird für den `plot`–Befehl verwendet. Wer `gnuplot` ohnehin schon installiert hat braucht das nicht mehr; da genügt dann die Standardinstallation.

A.2.2 Syntaxunterschiede zu Matlab

Manual:

http://www.gnu.org/software/octave/doc/interpreter

Unterschiede zu Matlab

Topic	Octave	Matlab
Dateiendung	`.m`	`.m`
Ende einer Funktionsdefinition	`endfunction`	`end`

B Online Dokumentation von Matlab

Tutorial von mathworks

helpdesk: http://www.mathworks.com

Skripte der ETH

Matlabkurs

Ingenieur–Tool