Eine kurze Einführung in *Octave* und *Matlab*

Dieses Dokument ist eine kurze Einführung in den Gebrauch von *Octave* oder *Matlab*. Die wichtigsten, absolut notwendigen Begriffe und Konzepte werden vorgestellt. Für eine vertiefte Einführung gibt es ausführlichere Dokumente im PDF-Format. Alle folgenden Beispiele und Erklärungen sind für *Octave* in einer UNIX Umgebung formuliert. Die Befehle für *Matlab* sind praktisch identisch.

1 Starten von *Octave*

Zuerst sollte eine *shell* geöffnet werden. Dies ist Unix Fenster in dem Befehle Zeile für Zeile ausgeführt werden können. Das Resultat in der ersten Zeile kann folgendermassen aussehen.

shell
```
user@hostname:~/matlab/myworks>
```

In dieser Befehlszeile der Shell können nun einzelne Befehle eingegeben werden. Als Beispiel zeigt der Befehl `pwd` das aktuelle Verzeichnis im System.

shell
```
user@hostname:~/matlab/myworks>pwd
/home/user/matlab/myworks
user@hostname:~/matlab/myworks>
```

Mit dem Befehl `cd` kann man sich im Verzeichnissbaum bewegen: mit `cd ..` um eine Stufe nach oben und mit `cd NameVerzeichniss` in das entsprechende Unterverzeichniss. Sehen Sie sich das folgende Beispielan:

shell
```
user@hostname:~/matlab/myworks>cd ..
user@hostname:~/matlab>
user@hostname:~/matlab>cd myworks
user@hostname:~/matlab/myworks>
```

Um eine Liste aller Dateinamen im aktuellen Verzeichnis zu erhalten kann der Befehl `ls` verwendet werden.

shell
```
user@hostname:~/matlab/myworks>ls
bisection.m    octave.tex       mygraphic.m
f01.m          fcn.m            myfigure.pdf
user@hostname:~/matlab/myworks>
```

Vor dem Starten von *Octave* sollten Sie das das gewünschte Arbeitsverzeichnis zum aktuellem Verzeichnis machen, durch einen geeigneten `cd` Befehl. In den folgenden Beispielen wählen wir immer `.../matlab/myworks` als Arbeitsverzeichnis.

Um *Octave* zu starten müssen Sie auf der Befehlszeile `octave` eingeben. Das Resultat wird vergleichbar zum Untenstehenden sein.

shell

```
user@hostname:~/matlab/myworks>octave
GNU Octave, version 3.8.2
Copyright (C) 2014 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "x86_64-unknown-linux-gnu".

Additional information about Octave is available at http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/get-involved.html

Read http://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.

octave:1>
```

Die letzte Zeile zeigt an, dass *Octave* bereit ist für den ersten Befehl. Die Nummerierung der Befehle wird automatisch nachgeführt. Die Befehle `cd`, `ls` und `pwd` sind auch unter *Octave* noch verfügbar.

Um *Octave* zu verlassen genügt es den Befehl `quit` auf der Befehlszeile einzugeben.

2 Installation und Dokumentation

Eine gute Installation von *Octave* besteht aus zwei Teilen: dem Grundpaket *Octave* und den Ergänzungen im Paket *Octave-Forge*. Diese Zweiteilung hat auch dazugeführt, dass Dokumentation und Installation aus mehreren Teilen besteht.

2.1 Dokumentation

Dokumentation zu *Octave* finden Sie auf der Webseite:
`staff.ti.bfh.ch/sha1` → *Octave information.*
Als Grundlage können Sie die Notizen von Griffiths verwenden.

2.2 Installation

- ***Schulrechner***
  Auf den durch die Informatikdienste unterhaltenen Linux-Rechnern an der TI ist Octave überall installiert. Für Windows sollte die Instalation bald erfolgen.
- ***Linux*** *auf Privatrechner*
  Fast alle Linux Distributionen beinhalten Octave und Octave-Forge. Dies ist sicher der einfachste Weg diese Pakete zu installieren. Sie können aber auch direkt die Source-Pakete kompilieren und installieren.
- ***Windows*** *auf Privatrechner*
  Auf der obigen Webseite finden Sie Hinweise zu Installation.

3 Einzelne Befehle ausführen in *Octave*

Für einfachste Berechnungen können Octave–Befehle Zeile für Zeile in der Shell eingegeben und ausgeführt werden. Dazu muss der Befehl auf eine Zeile getippt werden und anschliessend wird durch die `RETURN`-Taste der Befehl an Octave zur Berechnung übergeben. Das Resultat der Rechnung wird auf den folgenden Zeilen ausgegeben. Unten finden Sie, kommentarlos, einige Beispiele.

```
octave:1> 3 + 4*5
ans = 23
octave:2> 1/(12+1)
ans = 0.076923
octave:3> sqrt(13)
ans = 3.6056
octave:4>
```

Wird eine Zeile durch einen Strichpunkt abgeschlossen, so wird der Befehl ausgeführt, aber das Resultat nicht angezeigt. Dies kann sehr nützlich sein um den Bildschirm nicht mit länglichen Zwischenresultaten zu füllen.

In Octave können Variablen mit einem ***Namen*** *versehen werden, es kann mit symbolischen Variablen gerechnet werden. Als Namen zulässig sind Zeichenketten bestehend aus Gross- und Kleinbuchstaben und Ziffern (ab der zweiten Stelle). Einige mathematische Konstanten sind bereits vordefiniert: die Zahl* π *(*`pi`*) die Euler'sche Zahl* e *(*`e`*) oder die imaginäre Zahl* j *(*`i`*). Die Anweisung* `format long` *(resp.* `format short`*) führt dazu, dass Zahlen mit vielen (resp. wenigen) Dezimalstellen angezeigt werden. Untersuchen Sie die untenstehenden Beispiele und Resultate.*

```
octave:4> e
e = 2.7183
octave:5> pi
pi = 3.1416
octave:6> format long
octave:7> pi
pi = 3.14159265358979
octave:8> a = 3
a = 3
octave:9> x = 3 ;
octave:10> y = 4 ;
octave:11> z = x + i*y ;
octave:12> z*z
ans =  -7 + 24i
```

4 Arbeiten mit Script–Dateien

Während der Octave–Session werden die Befehle Zeile für Zeile abgearbeitet, die Resultate bestimmt und angezeigt, sofern gewünscht. Nach Abschluss der Session sind sowohl Eingaben wie Ausgaben verloren. Damit dieselbe Sequenz von Befehle mehrfach ausgeführt werden kann, muss ein ***Script–File*** *erstellt werden. Es handelt sich um normale Text–Dateien die mit einem Editor Ihrer Wahl bearbeitet werden können. Der Name der Datei soll sorgfältig gewählt werden, vorzugsweise zeigt der Name an, was mit den Befehlen berechnet wird. Der Filename muss mit* `.m` *abgeschlossen werden, damit der Befehl von Octave erkannt wird. Deshalb spricht man manchmal auch von M–Files. Unten finden Sie den Inhalt des M-Files* `triangle.m` *in dem die Länge der Hypothenuse eines rechtwinkligen Dreiecks mit den Kathethen 4 und 5 berechnet wird. Die ersten Zeilen sind Kommentare und Bemerkungen zum Code, eingeleitet durch das erste Zeichen % in jeder Zeile. Octave ignoriert Kommentare.*

```
% Ceci est le fichier script 'triangle.m'
% Ce programme elementaire calcule l'hypotenuse et les deux angles non droits
% d'un triangle rectangle dont les cathetes sont 5 et 4.

a = 5 ; % premiere cathete
b = 4 ; % deuxieme cathete
hyp = sqrt(4*4 + 5*5)
angle1 = atan(b/a)*180/pi
angle2 = atan(a/b)*180/pi
controle = angle1 + angle2
```

Um diesen Befehl in Octave auszuführen müssen Sie den Namen des Files eintippen, ohne die Erweiterung `.m`*.*

```
octave:11> triangle
hyp = 6.4031
angle1 = 38.660
angle2 = 51.340
controle = 90
```

Der Befehl `help triangle` *zeigt die ersten Kommentarzeilen des Funktionsfiles an.*

```
octave:12> help triangle
triangle is the file: /home/cip/matlab/myworks/triangle.m

Ceci est le fichier script 'triangle.m'
Ce programme elementaire calcule l'hypotenuse et les deux angles non droits
d'un triangle rectangle dont les cathetes sont 5 et 4.
```

Generell kann mit dem Befehl `help` *oder* `help ...` *auf die eingebaute Hilfe von Octave zugegriffen werden. Als Beispiel untersuchen Sie:*

```
octave:14> help atan
atan is a built-in mapper function

 -- Mapping Function:  atan (X)
     Compute the inverse tangent of each element of X.
```

Auf Unix-Systemen kann mit `help -i ...` *eine erweiterte Hilfefunktion verwendet werden.*

5 Vektoren und Matrizen

*Die Programmiersprachen Octave und Matlab (**Mat**rix **Lab**oratory) basieren ihre Berechnungen wesentlich auf den Konzepten und Notationen der Vektor- und Matrizenrechnung. Die meisten Befehle haben somit eine mehr oder wenige enge Beziehung zu Vektoren (Listen von Zahlen) oder Matrizen (Listen von Listen von Zahlen). Ein Vektor wird in Octave dargestellt als Liste von Zahlen, begrenzt durch eckige Klammern ([...]). Für einen Zeilenvektor werden die Einträge durch Kommas oder Leerstellen getrennt. Für einen Spaltenvektor sind die Einträge durch Strichpunkte zu trennen. Eine Matrix wird sehr ähnlich dargestellt, wobei die Zeilen durch Strichpunkte zu trennen sind.*

Alle üblichen Vektor- und Matrixoperationen sind in Octave möglich, illustriert durch die untenstehenden, selbstdokumentierenden Beispiele.

```
octave:1> u = [-1 2 4] ;
octave:2> v = [1.5 2 -1]
v =
   1.5000   2.0000  -1.0000

octave:3> u + v          % somme de deux vecteurs de meme dimension
ans =
  0.50000  4.00000  3.00000

octave:4> v'             % vecteur transpose
ans =
   1.5000
   2.0000
  -1.0000

octave:5> a = u*v'       % produit scalaire des deux vecteurs u et v
a = -1.5000
```

```
octave:6> A = [-1 1 2 ; 3 -1 1 ; -1 3 4]
A =
  -1   1   2
   3  -1   1
  -1   3   4

octave:7> det(A)          % determinant de la matrice A
ans = 10.000

octave:8> A*u'            % produit matriciel de A par u
ans =
   11
   -1
   23
```

Octave ist ein sehr effizientes Werkzeug um Systeme von linearen Gleichungen zu lösen, schliesslich wurden Matlab und Octave mit diesem Typ Anwendungen als Ziel entwickelt. Als Beispiel untersuchen wir ein System von 3 Gleichungen für 3 Unbekannte:

$$\begin{array}{rcrcrcr} -x_1 & + & x_2 & + & 2x_3 & = & 10 \\ 3x_1 & - & x_2 & + & x_3 & = & -20 \\ -x_1 & + & 3x_2 & + & 4x_3 & = & 40 \end{array}$$

Dieses System kann mit Hilfe der Matrizennotation dargestellt werden in der Form $\mathbf{A} \cdot \vec{x} = \vec{b}$.

$$\begin{bmatrix} -1 & 1 & 2 \\ 3 & -1 & 1 \\ -1 & 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 10 \\ -20 \\ 40 \end{bmatrix}$$

Die Lösung des System erfolgt nun durch die einfache Anweisung `x = A\b` . *Untersuchen Sie die untenstehenden Zeilen.*

```
octave:9> A = [-1 1 2 ; 3 -1 1 ; -1 3 4] ;
octave:10> b = [10 -20 40]' ;
octave:11> x = A\b
x =
    1.0000
   19.0000
   -4.0000
```

6 Definition von Funktionen

Die meisten der Grundfunktion sind in Octave bereits vordefiniert: `sin(x)`, `cos(x)`, `exp(x)`, `log(x)`, *usw. Die Argumente der Funktionen sind immer mit runden Klammern einzufassen.*

In Anwendungen muss man oft mit Modifikationen und Kombinationen der Standardfunktionen arbeiten. Somit ist es in Octave möglich neue Funktionen zu deklarieren und konstruieren. Hierzu muss mit einem Texteditor ein Funktionsfile erstellt werden. Das Vorgehen ist sehr ähnlich zu Skript–Dateien.

Als Beispiel untersuchen wir die Funktion $f(x) = 5\,\cos(2\,\pi\,x)\,e^{-x}$ *für verschiedene Werte des Argumentes* x. *Für Octave erstellen wir mit einem Editor ein Funktionsfile mit der Definition der Funktion. Als Beispiel wählen wir den Funktionsnamen* `f01` *und somit den Filenamen* `f01.m` . *Den Inhalt des Files finden Sie unten.*

```
function y = f01(x)
  y = 5*cos(2*pi.*x).*exp(-x) ;
end
```

Es ist zu beachten, dass der Name des Files (hier `f01.m`) und der Name der Funktion in der ersten Zeile (`f01`) übereinstimmen müssen.

In der zweiten Zeile des Codes für die obige Funktion haben Sie sicher die unübliche Notation (`.*`) für die Multiplikation bemerkt. Dies ist durch die vektorielle Struktur von Octave bedingt. Jede Funktion sollte auch mit Vektoren als Argumente ausgeführt werden können: der Funktionswert sollte für jede Komponente des Vektors bestimmt werden. Der Punkt vor dem Multiplikationszeichen bewirkt, dass die Operation auf jedes Element des Vektors angewandt wird. Untersuchen Sie das untenstehende Beispiel.

```
octave:13> f01(1.234)
ans = 0.14609

octave:14> x = [1 1.1 1.2 1.3] ;
octave:15> f01(x)
ans =
   1.83940   1.34649   0.46537  -0.42108
```

Wird eine neue Funktion nur sehr wenig verwendet, so ist es auch möglich die Berechnung direkt auszuführen, ohne ein Funktionsfile zu erstellen. Dazu ist zuerst das Argument x der Funktion zu erzeugen und dann die Berechnungsformel einzugeben. Das folgende Beispiel zeigt dieses Vorgehen.

```
octave:16> x = 1.2 ;
octave:17> y = 5*cos(2*pi*x)*exp(-x)
y = 0.46537

octave:18> x = [1.2 1.3 1.4 1.5] ;
octave:19> y = 5*cos(2*pi.*x).*exp(-x)
y =
   0.46537  -0.42108  -0.99751  -1.11565
```

Die Konstruktion von Funktionen mehrerer Variablen erfolgt analog, Beispiele werden im folgenden Abschnitt verwendet um Graphiken zu erstellen.

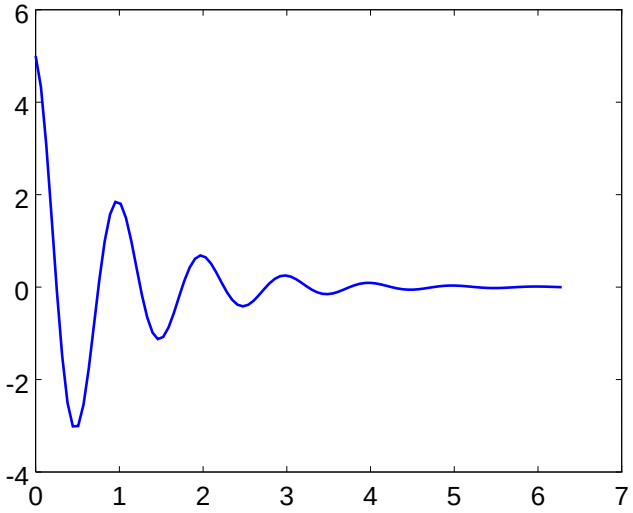
7 2D und 3D Graphiken

Um eine Funktion einer Variablen graphisch darzustellen geht man üblicherweise folgendermassen vor: man wählt ein Intervall $[a\,,\,b]$ auf dem der Graph zu zeichnen ist, dann werden die Funktionswerte in diesem Intervall bestimmt und auf der y–Achse aufgetragen. Theoretisch wären alle y–Werte für die x–Werte im Intervall zu bestimmen. Effektiv wählt man aus dem Intervall $[a\,,\,b]$ eine endliche Anzahl von repräsentativen Werten aus und berechnet die zugehörigen y–Werte. Meistens werden die x–Werte im Intervall gleichmässig verteilt von a bis b.

Das Vorgehen in Octave entspricht exakt der obigen Beschreibung. Zuerst werden n gleichverteilte x–Werte erzeugt durch den Befehl `x=linspace(a,b,n)`. Anschliessend werden die zugehörigen y–Werte berechnet und dann mittels des Befehls `plot()` die Graphik erzeugt. Als konkretes Beispiel untersuchen wir die obige Funktion `f01` auf dem Intervall $[0\,,\,2\pi]$ mit 100 gleichverteilten x–Werten.

```
octave:20> x = linspace(0, 2*pi, 100) ;
octave:21> y = f01(x) ;
octave:22> plot(x,y)
octave:23> print('graphe01.png')
```

Die letzte Zeile erzeugt eine Datei `graphe01.png` im PNG Format.



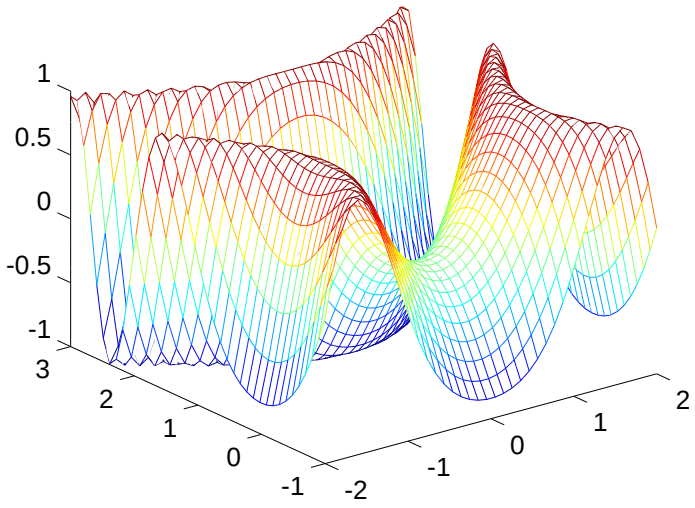
Verschiedenste Optionen für die Graphik können gewählt werden: Titel der Graphik, Achsenbeschriftungen, Hintergrundgitter, Farben, ... Konsultieren die die übliche Dokumentation um mehr zu erfahren.

Für Graphen von Funktionen zweier Variablen wird analog vorgegangen, allerdings muss zuerst das 2D–Gitter der unabhängigen Variablen erstellt werden. Das ist aufwendiger als für ein einfaches Intervall. Als Beispiel untersuchen wir die Funktion $f(x, y) = \sin(x^2 - y^2)$ *auf dem Rechteck* $[-2\,,\,2] \times [-1\,,\,3]$, *wobei in jeder Richtung 40 gleichverteilte Punkte gewählt werden. Zuerst wird das neue Funktionsfile* `f02.m` *erstellt:*

```
% Contenu du fichier-fonction f02.m
function z = f02(x,y)
z = sin(x.^2 - y.^2) ;
end
```

Anschliessend wird das Gitter und der Graph der Funktion erzeugt und dargestellt mittels der folgenden Octave–Befehle:

```
octave:24> x = linspace(-2,2,40) ;
octave:25> y = linspace(-1,3,40) ;
octave:26> [xx,yy] = meshgrid(x,y) ;
octave:27> z = f02(xx,yy) ;
octave:28> mesh(x,y,z)
```



8 Berechnungen mit Polynomen

Mathematisch sind Polynome einer Variablen eindeutig bestimmt durch die Koeffizienten. Beispielsweise ist das Polynom $p(x) = x^3 - 5\,x - 3$ bestimmt durch die Koeffizienten 1 (für x^3), 0 (für x^2), -5 (für x) und -3 . Für Octave sind die Koeffizienten geordnet nach fallenden Exponenten anzugeben. Für das obige Beispiel erhalten wir:

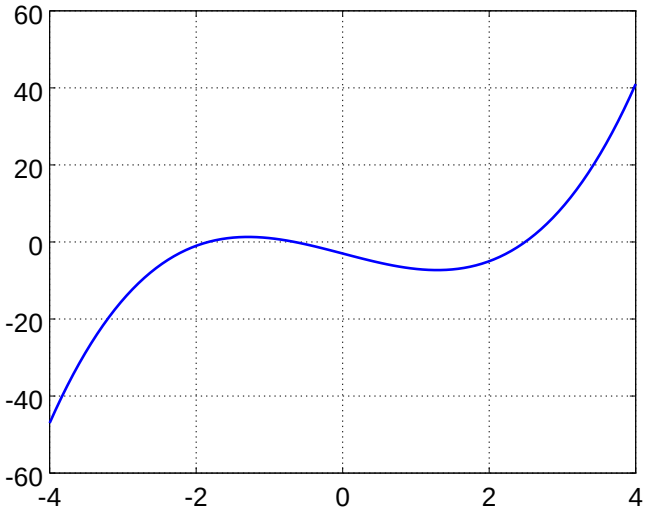
```
octave:30> p = [1 0 -5 -3] ;
```

Um die Werte eines Polynoms für ein bestimmtes Argument zu bestimmen kann der Befehl `polyval()` *verwendet werden. Es können ein oder mehrere Argumente untersucht werden, wie das untenstehende Beispiel zeigt.*

```
octave:31> x = 1.45 ;
octave:> polyval(p,x)
ans = -7.2014
octave:32> x = [1.4 1.5 1.6 1.7] ;
octave:33> polyval(p,x)
ans =
  -7.2560  -7.1250  -6.9040  -6.5870
```

Der Graph dieses Polynoms kann mit den Befehlen des vorangehenden Abschnitts erstellt werden.

```
octave:34> x = linspace(-4,4,100) ;
octave:35> y = polyval(p,x) ;
octave:36> plot(x,y)
octave:37> grid on
```



Es gibt einige spezielle Befehle für Polynome, als Beispiel sei der Befehl `roots()` *erwähnt. Mit ihm können Nullstellen bestimmt werden:*

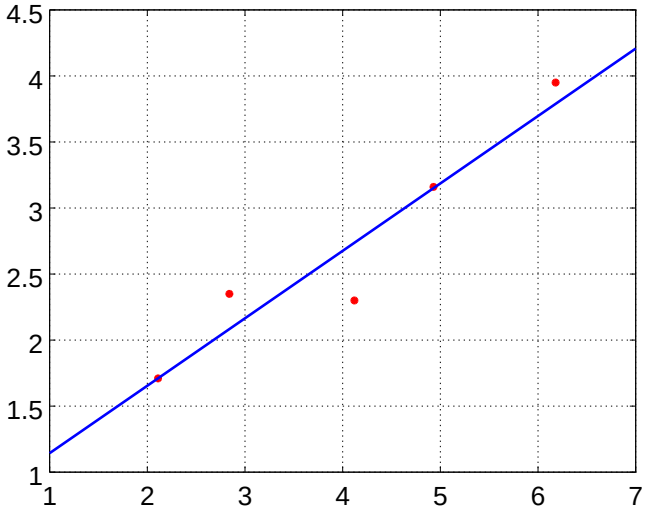
```
octave:39> roots(p)
ans =
   2.49086
  -1.83424
  -0.65662
```

Die ***lineare Regression*** *ist eine der nützlichsten numerischen Anwendungen von Polynomen. Ist eine Liste von Punkten in der Ebene gegeben (oder durch Messungen im Labor entstanden), so sucht man die Gleichung der Geraden,*

die so gut wie möglich durch diese Punkte geht. Eine Gerade ist beschrieben durch die Gleichung $y = a\,x + b$, und es sind somit die beste möglichen Werte der Parameter a und b zu bestimmen. Der Befehl `polyfit()` führt genau diese Berechnungen aus. Um ihn zu verwenden müssen die Liste der x– und y–Werte als Vektoren vorliegen und der Grad des Polynoms (hier 1) als Argumente übergeben werden. Es könnten auch Polynome höheren Grades eingepasst werden. Untersuchen Sie das folgende Beispiel:

```
octave:41> xp = [2.11 2.84 4.12 4.93 6.18] ;
octave:41> yp = [1.71 2.35 2.30 3.16 3.95] ;
octave:42> p = polyfit(xp,yp,1)
p =
  0.51058  0.63331

octave:43> x = linspace(1,7,50) ;
octave:44> y = polyval(p,x) ;
octave:45> plot(xp,yp,'r*',x,y)
octave:46> grid on
```



9 Gleichungen

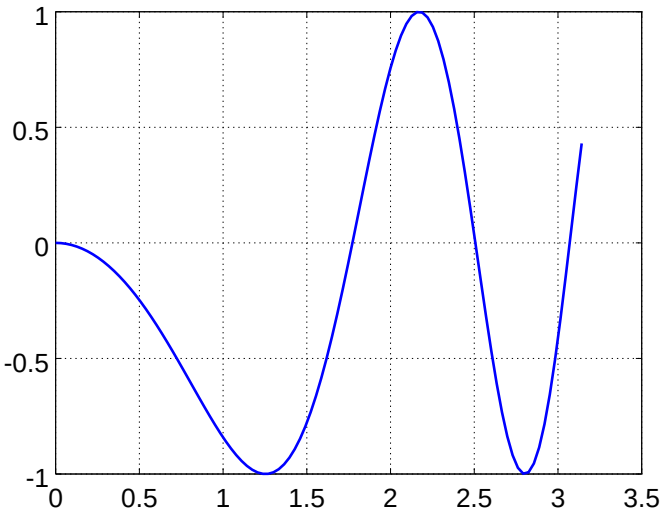
Die Programmiersprachen Matlab und Octave wurden in erster Linie für Matrizen– und Vektor–Berechnungen entwickelt. Folglich sind speziell effizient im Umgang mit linearen Systemen von Gleichungen, geschrieben in Matrizenform. Die weitergehende Dokumentation gibt viel mehr Information über den Umgang mit linearen Systemen, auch für sehr grosse Systeme.

9.1 Eine nichtlineare Gleichung mit einer Variablen

Falls eine Gleichung nicht linear ist, so müssen andere Methoden eingesetzt werden. In Octave ist ein gutes Verfahren im Befehl `fsolve()` implementiert.

Um eine Gleichung mit einer Unbekannten zu lösen, ist diese zuerst in die Form $f(x) = 0$ umzuschreiben, wobei x die zu bestimmende Grösse repräsentiert. Somit wird das Lösen einer Gleichung umgeschrieben zum Problem: finde die Nullstelle. Das verwendete Verfahren ist ein iterativer Algorithmus und folglich sind ein (möglichst guter) Startwert anzugeben.

Wir untersuchen das folgende Beispiel: bestimme die Nullstellen der nichtlinearen Funktion $f(x) = \sin(x^2 + \pi)$ im Intervall $[0\,,\,\pi]$. Der Graph der Funktion ist unten dargestellt.



Wir stellen fest, dass eine Nullstelle in der Nähe von $x = 1.7$ liegt, eine zweite nach bei $x = 2.5$ und eine dritte in der Nähe von $x = 3$. Nun werden diese drei Nullstellen mit erheblich grösserer Genauigkeit bestimmt. Zuerst ist ein Funktionsfile mit der entsprechenden Funktion zu erstellen und als `f03.m` abzuspeichern.

```
function y = f03(x)
y = sin(x.^2 + pi) ;
end
```

Die Nullstellen sind anschliessend durch die folgenden Befehle bestimmbar.

```
octave:46> format long
octave:47> fsolve('f03',1.7)
ans = 1.77245385090552

octave:48> f03(ans)
ans =  -2.44921270764475e-16

octave:49> fsolve('f03',2.5)
ans = 2.50662827463096

octave:48> f03(ans)
ans =  1.90437563721974e-13

octave:49> fsolve('f03',3)
ans = 3.06998012383947
```

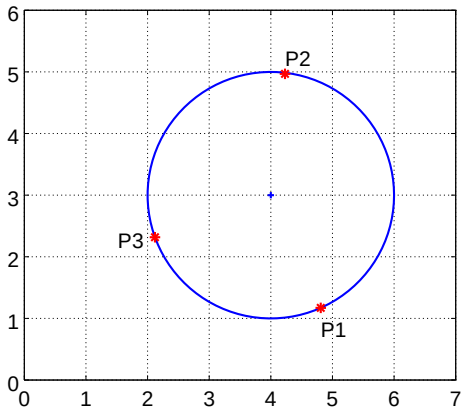
Bei der ersten Nullstelle wird als Kontrolle noch der Wert der Funktion an der Nullstelle bestimmt. Man stellt fest, dass effektiv eine Nullstelle gefunden wurde, bis auf unvermeidbare Rundungsfehler.

9.2 Systeme von nichtlinearen Gleichungen mit mehreren Unbekannten

In diesem Abschnitt untersuchen wir das folgende, illustrative Beispiel: drei beliebige Punkte in einer Ebene bestimmen eindeutig einen Kreis, sofern die Punkte nicht auf einer Geraden liegen. Gesucht ist die Kreisgleichung. Dieses Problem führt auf drei Gleichungen mit den drei Unbekannten x_0, y_0 und R. Die Gleichung eines Kreises mit Mittelpunkt bei $(x_0\,,\,y_0)$ und Radius R ist gegeben durch

$$(x - x_0)^2 + (y - y_0)^2 = R^2$$

Diese Gleichung muss erfüllt sein für die drei gegebenen Punkte $(x_k\,,\,y_k)$ wobei $k = 1, 2, 3$. Wegen der Quadrate ist dies kein lineares System von Gleichungen.



Untersuchen wir ein numerisches Beispiel, illustriert durch die obige Graphik: $P_1(4.81,\ 1.171)$, $P_2(4.23,\ 4.968)$ *und* $P_3(2.12,\ 2.317)$ *. Schreiben wir die drei Gleichungen aus, so erhalten wir*

$$\begin{array}{rcrcrcl} (4.81-x_0)^2 & + & (1.171-y_0)^2 & - & R^2 & = & 0 \\ (4.23-x_0)^2 & + & (4.986-y_0)^2 & - & R^2 & = & 0 \\ (2.12-x_0)^2 & + & (2.317-y_0)^2 & - & R^2 & = & 0 \end{array}$$

Nun werden die drei Zeilen als Komponenten eines Vektors aufgefasst und wir erhalten eine vektorwertige Funktion, deren Nullstellen zu suchen sind. In Octave führt dies zu einem Funktionsfile `f06.m` *welches die drei Werte der Funktion als Vektor berechnet. Die Unbekannten sind als Komponenten eines Vektors* $\vec{x}$ *geschrieben, hier* `x(1)`*,* `x(2)` *und* `x(3)`*.*

```
% Fichier-fonction f06.m
function z = f06(x)
z = zeros(3,1) ;
z(1) = (4.81 -x(1)).^2 + (1.171 - x(2)).^2 - x(3).^2 ;
z(2) = (4.23 -x(1)).^2 + (4.986 - x(2)).^2 - x(3).^2 ;
z(3) = (2.12 -x(1)).^2 + (2.317 - x(2)).^2 - x(3).^2 ;
end
```

Da ein iterativer Algorithmus verwendet wir, müssen gute Startwerte für die drei Unbekannten angegeben werden. Wir verwenden den Schwerpunkt der drei gegebenen Punkte: $S(3.72,\ 2.82)$ *und den geschätzten Radius* 1.8 *. Das System wird dann durch den folgenden Befehl gelöst:*

```
octave:50> fsolve('f06', [3.72 2.82 1.8])
ans =
  3.99983639831334
  2.99941877091002
  1.99987006160751
```

Für ein gegebenes nichtlineares System von Gleichungen ist es leicht möglich, dass der iterative Algorithmus nicht konvergiert. Dann kann dem Befehl `fsolve()` *mehr Information zur Verfügung gestellt werden: die Jacobi–Matrix der partiellen Ableitungen (siehe Analysis Kurs). Für unser Beispiel ist ein zweites Funktionsfile* `J06.m` *zu erstellen mit folgendem Inhalt:*

```
% Matrice jacobienne relative aux fonctionS f06.m
function z = J06(x)
z = [-2*4.81+2.*x(1) -2*1.171+2.*x(2) ;
     -2*4.23+2.*x(1) -2*4.986+2.*x(2) ;
     -2*2.12+2.*x(1) -2*2.317+2.*x(2)] ;
end
```

Nun kann das System gelöst werden durch

```
octave:51> [X, INFO, MSG] = fsolve(['f06' ; 'J06'], [3.72 2.82 1.8])
X =
  3.99983639831334
  2.99941877091002
  1.99987006160709

INFO = 1
MSG = solution converged within specified tolerance
```

Falls der Startwert nicht gut genug gewählt ist, kann es auch sein, dass der Algorithmus keine Lösung findet, wie das untenstehende Beispiel zeigt.

```
octave:52> [X, INFO, MSG] = fsolve(['f06' ; 'J06'], [3.7 2.8 1.5])
X =
  3.70000000000000
  2.80000000000000
  1.50000000000000

INFO = 3
MSG = iteration is not making good progress
```

10 Übungen

Übung 10.1 Berechnen Sie das exakte Volumen V und die Oberfläche S (ohne Grundfläche) der Cheopspyramide. Eine Seite der quadratischen Grundfläche B hat eine Länge von 227 m und die Pyramide eine Höhe h von 135 m. Wie lang müsste eine Mauer der Höhe 1 m und Breite 0.5 m sein, damit sie dasselbe Volumen hat?

Übung 10.2 Untersuchen Sie die Funktion $f(t) = \sin(\omega t)e^{-\alpha t}$, mit $\omega = 10\pi$ und $\alpha = 2$. Zeichnen Sie den Graphen der Funktion $f(t)$ auf dem Intervall $0 \leq t \leq 1$. Zeichnen Sie in derselben Graphik auch die beiden Hüllkurven $g(t) = \pm e^{-\alpha t}$.

Übung 10.3 Untersuchen Sie die Funktion $f(t) = e^{-t^2}\left[2 - \sin(8t^2)\right]$.

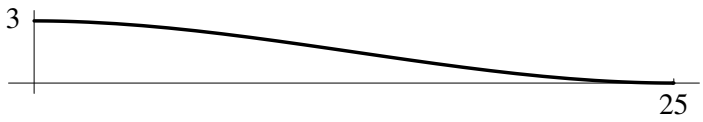
1. Für $N = 20$ und $N = 100$, erzeugen Sie die N Werte der Funktion $f(t)$ auf dem Intervall $[0, 2]$, d.h.

   $$E = \{f(0), f(\Delta t), f(2\Delta t), \ldots, f(N\Delta t)\}$$

   wobei $\Delta t = \frac{2}{N}$.

2. Stellen Sie das obige Resultat graphisch dar. Verwenden Sie nicht nur die Funktion `plot()` sondern auch `stem()`.

Übung 10.4 Beim Landeanflug auf einen Flughafen kann die Flugbahn eines Flugzeugs beschrieben werden durch ein Polynom vom Grad 3 $p(x) = a\,x^3 + b\,x^2 + c\,x + d$. Die Steigung dieser Kurve ist gegeben durch $p'(x) = 3\,a\,x^2 + 2\,b\,x + c$.



Beim horizontalen Abstand von 25 km vom Flughafen fliegt das Flugzeug horizontal auf einer Höhe von 3000 m. Anschliessend folgt es der Kurve $p(x)$ um am richtigen Punkt exakt horizontal aufzusetzen.

Bestimmen Sie die Werte der Koeffizienten a, b, c und d des Polynoms $p(x)$ und stellen Sie die Kurve graphisch dar mit *Octave*.
Tipp: System von Gleichungen.

Übung 10.5 Die Funktion $f(x,t) = e^{-0.1t}\sin(x-vt)$ stellt eine sinussförmige Welle dar, die sich mit der Geschwindigkeit v in x–Richtung bewegt und die gedämpft ist bezüglich der Zeit t. Erzeugen Sie eine Graphik für $0 \leq x \leq 10$ und $0 \leq t \leq 20$. Wählen Sie verschiedene Werte für v, z.B. $v = 1$.

---

11 Lineare Regression für den Widerstand als Funktion der Temperatur

11.1 Erläuterung der Problemstellung

Der elektrische Widerstand von Kupfer ist sehr stark von der Temperatur T abhängig. Diese Temperaturabhängigkeit kann gemessen werden und auch theoretische Überlegungen zeigen, dass für kleine Temperaturbereiche eine affine Funktion entstehen muss. Somit ist der Widerstand R gegeben als Funktion der Temperatur T durch

$$R(T) = p_1 \cdot T + p_2$$

Ziel dieses Abschnittes ist es die Werte von p_1 und p_2 aus den in einem File gespeicherten Messdaten herauszuholen, die notwendigen Berechnungen auszuführen und die Daten zu visualisieren.

11.2 Lesen der Daten aus einer Datei und Anzeigen der Daten

Die Datei `copper.dat` enthält die Messwerte der Temperatur T und des Widerstandes R in zwei Spalten. Die erste Spalte zeigt, dass die Temperatur von $20°C$ anwächst auf $80°C$. Der Widerstand wächst von $4.36\ \Omega$ auf $5.37\ \Omega$.

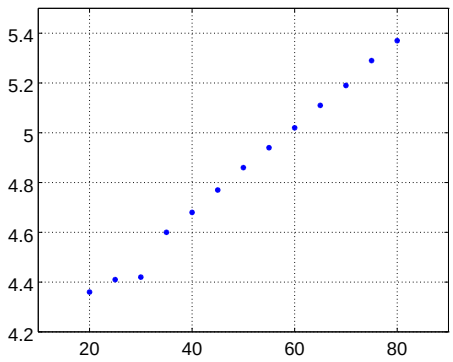
copper.dat

```
20 4.36
25 4.41
30 4.42
35 4.60
40 4.68
45 4.77
50 4.86
55 4.94
60 5.02
65 5.11
70 5.19
75 5.29
80 5.37
```

Der untenstehende Code liest die Daten aus der Datei und zeigt sie in einer einfachen Graphik an.

```
data = dlmread('copper.dat'); % read the data
k = length(data);
T = data(:,1); R = data(:, 2);

disp(sprintf('there are %d numbers of lines with data',k))
clf
plot(T,R,'*');          %% generate a basic plot
axis([10 90 4.2 5.5])
```



Das Resultat zeigt, dass die Daten richtig gelesen wurden.

11.3 Ausführen der linearen Regression mit dem Befehl `polyfit()`

Um den obigen Widerstand als lineare Funktion der Temperatur zu schreiben kann der Befehl `polyfit()` *verwendet werden, siehe auch Abschnitt 8, ab Seite 8. Gesucht sind die optimalen Werte des Vektors* $\vec{p} = (p_1, p_2)$ *sodass*

$$R(T) \approx p_1\, T + p_2$$

Das Resultat der Octave–Zeile

```
p = polyfit(T,R,1)
```

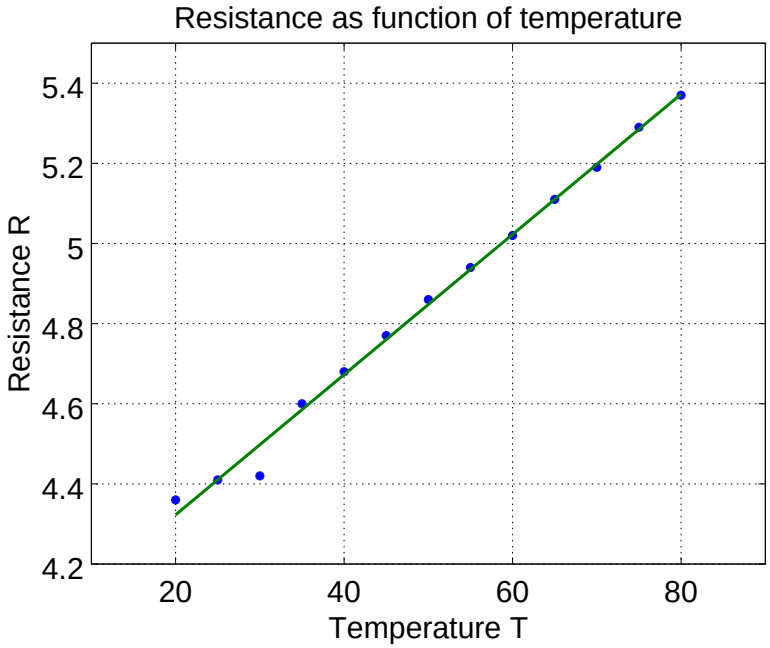
zeigt, dass

$$R(T) \approx 0.017495 \cdot T + 3.972967$$

11.4 Visualisierung

Nun sollten noch die gemessenen Punkte und die Regressionsgerade in einem Plot dargestellt werden, inklusive Titel und Beschriftung der Achsen. Das Resultat wird auch als PostScript Datei abgespeichert für eine eventuelle Weiterverwendung in einem Messbericht. Die resultierende Graphik ist unten gezeigt.

```
p = polyfit(T,R,1)
Rfit = polyval(p,T);            %% compute the fitted resistances
plot(T,R,'*',T,Rfit);           %% generate a basic plot
title('Resistance as function of temperature')
grid on
axis([10,90,4,5.6])
xlabel('Temperature T');
ylabel('Resistance R');
```



11.5 Berechnung mit `LinearRegression()`

Um weitergehende Information über die Regressionsgerade zu erhalten kann der Befehl[1] `LinearRegression()` *verwendet werden. Da die gesuchte Gerade die Form*

$$R(T) = p_1 \cdot T + p_2 \cdot 1$$

[1] Die Funktion `LinearRegression.m` ist teil des Packages `optim` oder muss muss zuerst in Ihr Arbeitsverzeichniss kopiert werden.

hat, muss zuerst die Matrix

$$\mathbf{F} = \begin{bmatrix} T_1 & 1 \\ T_2 & 1 \\ T_3 & 1 \\ \vdots & \vdots \\ T_n & 1 \end{bmatrix}$$

konstruiert werden. Dann können sowohl die geschätzten Werte von p_1 und p_2 bestimmt werden, als auch die Standardabweichung dieser geschätzten Werte.

```
F = ones(k,2);
F(:,1) = T;
[p,y_var,r,p_var] = LinearRegression(F,R);
values = p
standarddev = sqrt(p_var)
```

Die Werte von p_1 und p_2 stimmen selbstverständlich mit den Werten des vorangehenden Abschnitts überein. Die geschätzten Standardabweichungen führen auf

$$\begin{aligned} p_1 &= 0.017495 \pm 0.00040 \\ p_2 &= 3.972967 \pm 0.02143 \end{aligned}$$

Aufgrund der vorliegenden Standardabweichungen der Werte sind oben zu viele Ziffern angegeben und das Resultat wird konsistenter dargestellt als

$$p_1 = 0.0175 \pm 0.0004 \qquad \textit{und} \qquad p_2 = 3.97 \pm 0.02$$

11.6 Regression durch eine Parabel

Um eine allgemeine Parabel der Form

$$R(T) = p_1 \cdot T^2 + p_2 \cdot T + p_3 \cdot 1$$

an die gegebenen Daten anzupassen muss zuerst die Matrix

$$\mathbf{F} = \begin{bmatrix} T_1^2 & T_1 & 1 \\ T_2^2 & T_2 & 1 \\ T_3^2 & T_3 & 1 \\ \vdots & \vdots & \vdots \\ T_n^2 & T_n & 1 \end{bmatrix}$$

konstruiert werden. Dann führt der Code

```
F = ones(k,3);
F(:,1)=(T.^2); F(:,2)=T;
[p,y_var,r,p_var] = LinearRegression(F,R);
values = p
standarddev = sqrt(p_var)
Rquad = F*values;
plot(T,R,'*',T,Rfit,T,Rquad);
```

zu den Resultaten

$$\begin{aligned} p_1 &= 3.1968 \cdot 10^{-6} \pm 2.5373^{-5} \\ p_2 &= 0.0172 \pm 0.003 \\ p_3 &= 3.9798 \pm 0.006 \end{aligned}$$

Man erkennt, dass

- *Der quadratische Term $p_1 \cdot T \approx 3.1968 \cdot 10^{-6} \cdot T$ ist sehr klein im Vergleich zu den beiden anderen Termen.*
  - *Die Werte der Temperatur T sind von der Grössenordnung $T \approx 80 \approx 100 = 10^2$.*
  - *Der quadratische Term ist von der Grössenordnung $p_1 \cdot T^2 \approx 10^{-6} \cdot 10^{2 \cdot 2} = 10^{-2}$*
  - *Der lineare Term ist von der Grössenordnung $p_2 \cdot T \approx 10^{-2} \cdot 10^2 = 1$*
  - *Der konstante Term ist von der Grössenordnung $p_3 \approx 4$*
- *Die Standardabweichung von p_1 ist erheblich grösser als der Wert von p_1.*
- *Die Werte des konstanten und linearen Beitrags haben im Vergleich mit der obigen Regressionsgerade nicht erheblich geändert, einzig deren Standardabweichung hat neue Werte der selben Grössenordnung.*

Die obigen Beobachtungen bestätigen, dass die Wahl einer Geraden als approximative Kurve eine gute Wahl ist. Diese Tatsache kann noch zusätzlich unterstützt werden durch eine Graphik erzeugt durch den obigen Code. Gezeigt werden die Rohdaten, die Regressionsgerade und die Regressionsparabel. Es sind visuell keine Unterschiede feststellbar.

12 Regression angewandt auf ein Wärmeleitungsproblem

12.1 Erläuterung der Problemstellung

Taucht man einen sehr kalten Metallkörper in ein grosses, warmes Wasserbad mit Temperatur T_0 ein, so wird sich der Metallklotz langsam erwärmen und nach langer Zeit auch die Temperatur T_0 haben. Das Abkühlgesetz von Newton besagt, dass die Temperatur T des Körpers gegeben ist als Funktion der Zeit durch

$$T(t) = T_0 - c \cdot e^{-\alpha\, t}$$

für geeignete Konstanten c und α. Mit dem Wert von α kann der Wärmekapazität der Körpers bestimmt werden. Ein echter Datensatz soll entsprechend analysiert werden.

12.2 Lesen und Anzeigen der Daten

Die Daten wurden durch ein Messprogramm abgespeichert in der Datei `Messdaten.prn` und die ersten Zeilen sind unten gezeigt.

Messdaten.prn

```
-0.120000004768372      61.7662492      8.01224287
2.95999991893768        61.758956       8.02274545
4.87999999523163        61.7564654      8.00599555
6.9099999666214         61.7493545      7.96234964
8.93999993801117        61.752114       7.99515274
10.8700000047684        61.7550227      8.23236488
...
```

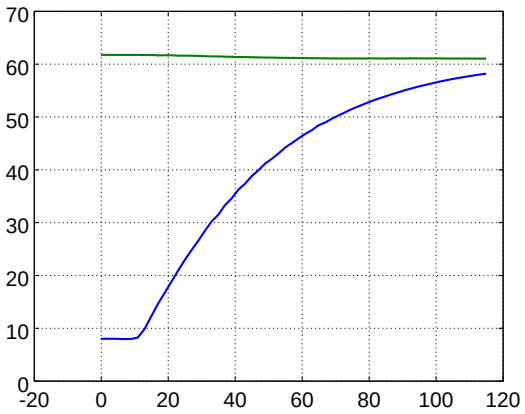
Die erste Spalte liefert die Zeit t, die zweite die Temperatur T_0 des Wasserbades und die dritte die Temperatur T des Metallblocks. Analog zum vorangehenden Abschnitt sind zuerst die Daten zu lesen und eine erste Graphik zu erzeugen.

```
data = dlmread('Messdaten.prn');
k = length(data);
disp(sprintf('there are %d numbers of lines with data',k))
t  = data(:,1);
T  = data(:,3);
T0 = data(:,2);

plot(t,T,t,T0)
```

Die resultierende Graphik ist unten gezeigt und wir stellen einige Effekte fest:

- *Die Temperatur des Wasserbades ist fast konstant. Für die weiteren Berechnungen werden wir sie folglich durch ihren Mittelwert über alle Messwerte ersetzen.*
- *Die Temperatur $T(t)$ des Blocks hat die richtige qualitative Form, ausser während den die ersten ca. 13 Sekunden. In dieser Anfangsphase war der Block noch nicht ins Bad eingetaucht und hat somit seine Temperatur gehalten. Somit ist diese Anfangsphase aus dem Datensatz zu entfernen.*



12.3 Vorbereitung der Daten

Zuerst ist die mittlere Temperatur T_0 des Wasserbades zu bestimmen. Anschliessend berechnet man ab welchem Punkt die Zeitwerte t grösser als 13 sind. Nur diese Punkte werden berücksichtigt für die weitere Analyse. Die Berechnungen werden graphisch überprüft.

```
T0 = mean(T0);     %% replace the tepmerature of the bath by its average value

maxIndex = max(find(t<13)); %% find the maximal index for which t<13
t = t(maxIndex:end);        %% examine only t>13
T = T(maxIndex:end);
plot(t,T0-T)                %% visual verification of the result
```

Das Gesetz von Newton besagt, dass

$$\begin{aligned} T(t) &= T_0 - c\,e^{-\alpha\,t} \\ T_0 - T(t) &= c\,e^{-\alpha\,t} \\ y = \ln(T_0 - T(t)) &= \ln(c) - \alpha\,t \end{aligned}$$

Tragen wir somit $y = \ln(T_0 - T(t))$ als Funktion von t auf, so sollten wir eine Gerade mit Steigung $-\alpha$ erhalten. Dies wird bestätigt durch den untenstehenden Code und die resultierende Graphik.

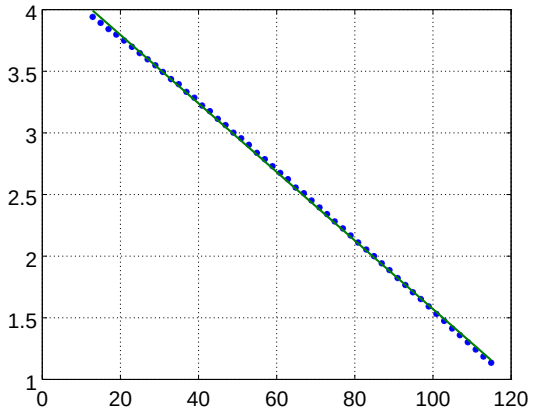
```
y = log(T0-T);
plot(t,y)
```

12.4 Lineare Regression und Visualisierung der Resultate

Die Parameter dieser Geraden können wie im vorangehenden Abschnitt durch eine lineare Regression bestimmt werden. Die Qualität der Regression kann graphisch überprüft werden.

```
F = ones(length(t),2);
F(:,1) = t;
[p,y_var,r,p_var] = LinearRegression(F,y);
values = p
variances = sqrt(p_var)

yfit = F*p;
plot(t,y,'*',t,yfit);
```



Die numerischen Werte ergeben

$$\alpha \approx 0.02780 \pm 0.0009 \qquad \textit{und} \qquad \ln(c) \approx 4.351 \pm 0.006$$

Aus diesen Werten kann nun die ursprüngliche Kurve rekonstruiert werden durch

$$T(t) = T_0 + c\, e^{-\alpha\, t} = T_0 + e^{\ln(c)}\, e^{-\alpha\, t}$$

Die resultierende Graphik zeigt, dass die Berechnungen recht gute Resultate liefern.

```
Tfit = T0-exp(values(2))*exp(values(1)*t);

plot(t,T,'*',t,T0*ones(size(t)),t,Tfit);
title('Temperatures as function of time')
grid on
axis([0,120,0,70])
xlabel('Time t'); ylabel('Temperatures T and T0');
print('./../HeatFit.png');
```