

OCTAVE und GNUPLOT für die Auswertung von Messdaten

Michael Zeising
michael@michaels-website.de

Stand: 29. April 2008



Copyright © 2006-2008 Michael Zeising

Kopieren, Verbreiten und/oder Modifizieren ist unter den Bedingungen der *GNU Free Documentation License*, Version 1.2 oder einer späteren Version, veröffentlicht von der *Free Software Foundation*, erlaubt. Es gibt keine unveränderlichen Abschnitte, keinen vorderen Umschlagtext und keinen hinteren Umschlagtext.

Copyrights

OCTAVE © by John W. Eaton

MATLAB ® is a registered trademark by The MathWorks, Inc.

Inhaltsverzeichnis

1	Graphen plotten mit GNUPLOT	4
1.1	Einfaches Plotten	4
1.2	Daten aus einer Datei plotten	5
1.3	Aussehen des Graphen anpassen	5
1.4	Sonderzeichen	6
1.5	Zwei y-Achsen	7
1.6	Glättung	8
1.7	Kurvenanpassung	8
1.8	Plots exportieren	12
1.9	Skriptdateien	13
1.10	Anwendungsbeispiele	14
2	Numerische Berechnungen in OCTAVE	15
2.1	Skalare, Vektoren und Matrizen	15
2.2	Der Workspace	16
2.3	Bereiche	16
2.4	Arithmetik	17
2.5	Funktionen	17
2.6	Eingebaute Hilfe	18
2.7	Plotten	18
2.8	Import/Export von Daten	19
2.9	3D-Plotten	19
2.10	.m-Dateien	20
2.11	Eigene Funktionen	20
2.12	Genauigkeit und Ausgabeformat	21
2.13	Kurzer Überblick über den Funktionsumfang von OCTAVE	21
2.14	Anwendungsbeispiele	22

1 Graphen plotten mit GNUPLOT

GNUPLOT ist ein Funktionsplotter, also ein Programm zur grafischen Darstellung von Funktionen und Daten. Es besitzt keine grafische Benutzeroberfläche, sondern wird von der Kommandozeile aus bedient. GNUPLOT stammt aus der Unix-Welt, läuft aber auf allen gängigen Betriebssystemen¹.

1.1 Einfaches Plotten

Nach dem Start meldet sich GNUPLOT mit dem Prompt

```
gnuplot>
```

und erwartet nun Befehle.

Die Funktion $f(x) = x^2$ können wir mit folgenden Befehlen darstellen:

```
gnuplot> f(x) = x**2
gnuplot> plot f(x)
```

Der Befehl `plot` hat (unter anderem) folgende Syntax:

```
plot Funktion title "Beschriftung"
      oder
plot Funktion notitle
```

Er kann beliebig viele Funktionen gleichzeitig plotten:

```
gnuplot> g(x) = sqrt(x)
gnuplot> plot f(x) title "Parabel", g(x) title "Wurzel"
```

Wenn keiner angegeben wird, wählt GNUPLOT eigenständig einen Bereich, in dem es die Funktion darstellt. In unserem Fall wirkt die Wurzel im Gegensatz zu Parabel wie eine Gerade. Wir müssen also mit `set xrange` und `set yrange` einen kleineren Ausschnitt festlegen.

Die Syntax der Befehle lautet

```
set xrange [x:x̄]
      bzw.
set yrange [y:ȳ]
```

In unserem Fall ist es sinnvoll, die beiden Funktionen im Bereich $[-1, 1]$ zu plotten:

```
gnuplot> set xrange [-1:1]
gnuplot> replot
```

¹wobei die Windows-Portierung nur im Zusammenhang mit Octave und der *Octave Workshop* Oberfläche zu empfehlen ist; siehe <http://www.math.mcgill.ca/loisel/octave-workshop/>

1.2 Daten aus einer Datei plotten

Dem `plot`-Befehl kann statt einer Funktion auch ein Dateiname übergeben werden. In diesem Fall ist ein weiterer Parameter möglich, nämlich

`using Spalten`

Er legt eine durch `:` getrennte Liste der Spalten fest, die für den Plot verwendet werden sollen. Um also die ersten beiden Spalten der Datei `daten.dat` als x- und y-Werte zu plotten ist der folgende Befehl nötig:

```
plot "daten.dat" using 1:2 title "Meine Daten"
```

Da die Daten aber meistens nur in zwei Spalten vorliegen, kann der Parameter weggelassen werden. GNUPLOT nutzt dann alle.

1.3 Aussehen des Graphen anpassen

Linienstil

Dem Befehl `plot` kann zur Anpassung der Linienform und -farbe die Option

`with Linientyp`

übergeben werden. Zunächst kann man grundsätzlich zwischen den Linientypen `lines`, `points`, `linespoints`, `dots`, `impulses` und noch vielen mehr wählen. Zusätzlich lässt sich mit

`lw Stärke lt Farbe/Form`

zum einen die Stärke, zum anderen die Farbe bei Linien bzw. die Form bei Punkten angeben. Der Befehl

```
gnuplot> plot f(x) with points lt 11
gnuplot> plot g(x) with lines lw 3 lt 7
```

plottet also $f(x)$ mit blauen Dreiecken und $g(x)$ mit einer orangefarbenen Linie der Stärke 3.

Welche Möglichkeiten GNUPLOT bietet kann dem Testplot entnommen werden. Er wird mit

```
gnuplot> test
```

erzeugt.

Gitternetz

Das Gitternetz kann mit

```
gnuplot> set grid
```

eingeschaltet werden.

Logarithmische Skalierung

Mit dem Befehl

```
set log Achse(n)
```

kann die Skalierung für eine oder mehrere Achsen auf logarithmisch umgeschaltet werden.

```
gnuplot> set log xy
```

skaliert also die x- und y-Achse logarithmisch.

Fehlerbalken

Häufig hat man zusätzlich zur x- und y-Spalte einer Messreihe eine Fehlerspalte gegeben. Handelt es sich um die absoluten Fehler der y-Werte in der richtigen Dimension, lassen sie sich leicht als Fehlerbalken darstellen:

```
gnuplot> plot "messreihe.dat" using 1:2:3 with yerrorbars
```

Es werden also drei Spalten für den Plot herangezogen und die dritte als Fehlerbalken in y-Richtung dargestellt.

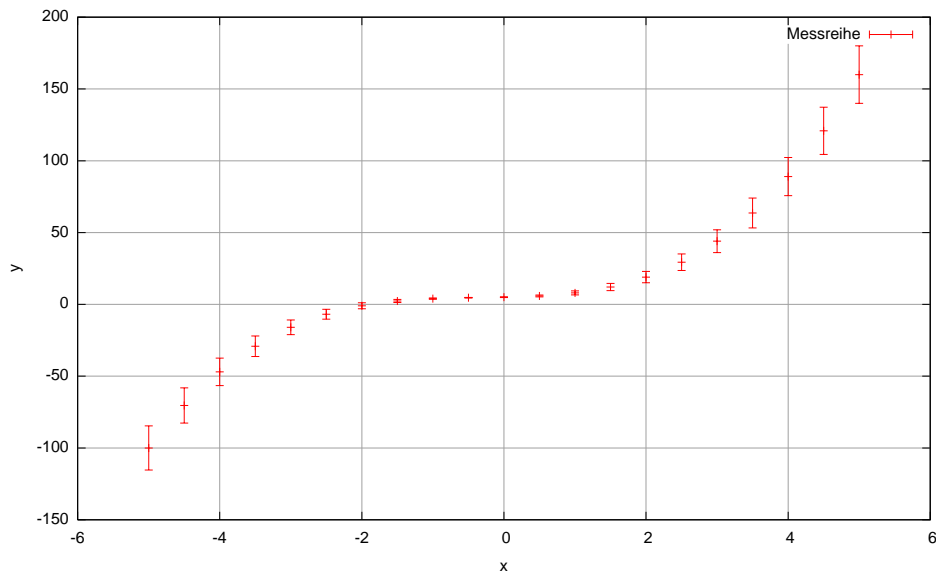


Abbildung 1: Fehlerbalken

1.4 Sonderzeichen

Beim Umgang mit Sonderzeichen ist zu beachten, dass GNUPLOT den selben Zeichensatz verwendet, wie der Editor bzw. die Konsole in der man arbeitet. Bei Problemen kann der Zeichensatz einfach festgelegt werden. Für ISO 8859-15 etwa:

```
gnuplot> set encoding iso_8859_1
```

Sonderzeichen, die nicht über die Tastatur eingegeben werden können, wie z.B. das Ångström-Zeichen können anhand ihres Oktalwertes im jeweiligen Zeichensatz angegeben werden. Das Zeichen Å hat im Zeichensatz ISO 8859-15 den Oktalwert 305, kann also wie folgt verwendet werden:

```
gnuplot> set xlabel "x [\305]"
```

Das Ergebnis wäre in diesem Fall die Achsenbeschriftung x [Å]. Eine Übersicht über den Zeichensatz erhält man z.B. auf der *manual page* zu `iso_8859_1`.

1.5 Zwei y-Achsen

Oft sollen zwei Reihen verschiedener Dimension in einem Koordinatensystem dargestellt werden um eine Beziehung zu verdeutlichen. In diesem Fall sind zwei y-Achsen notwendig. Wir stellen z.B. eine Dämpfung a aus `a.dat` und eine Konzentration c aus `c.dat` in Abhängigkeit von der Zeit t dar.

```
gnuplot> set xlabel "t [s]"
gnuplot> set ylabel "a [dB]"
gnuplot> set y2label "c [Mol]"
gnuplot> set y2tics
gnuplot> set grid

gnuplot> plot "a.dat" with lines title "Daempfung" axes x1y1,
             "c.dat" title "Konzentration" axes x1y2
```

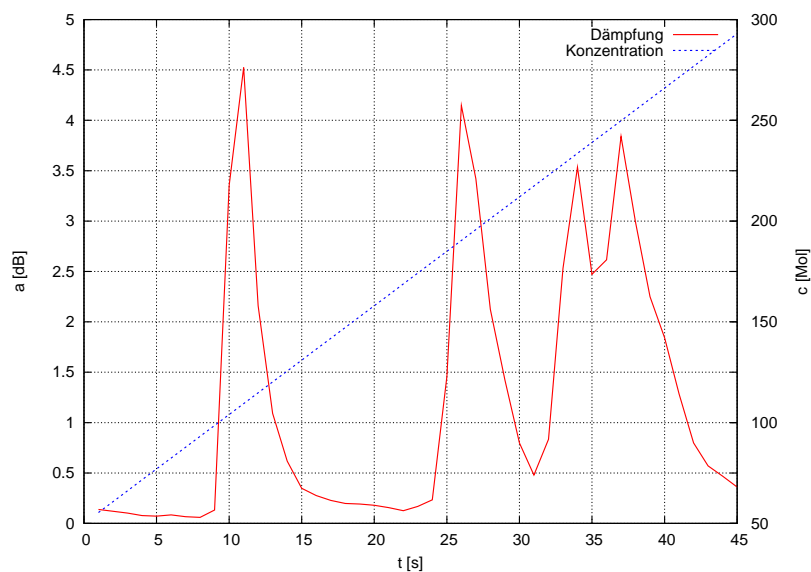
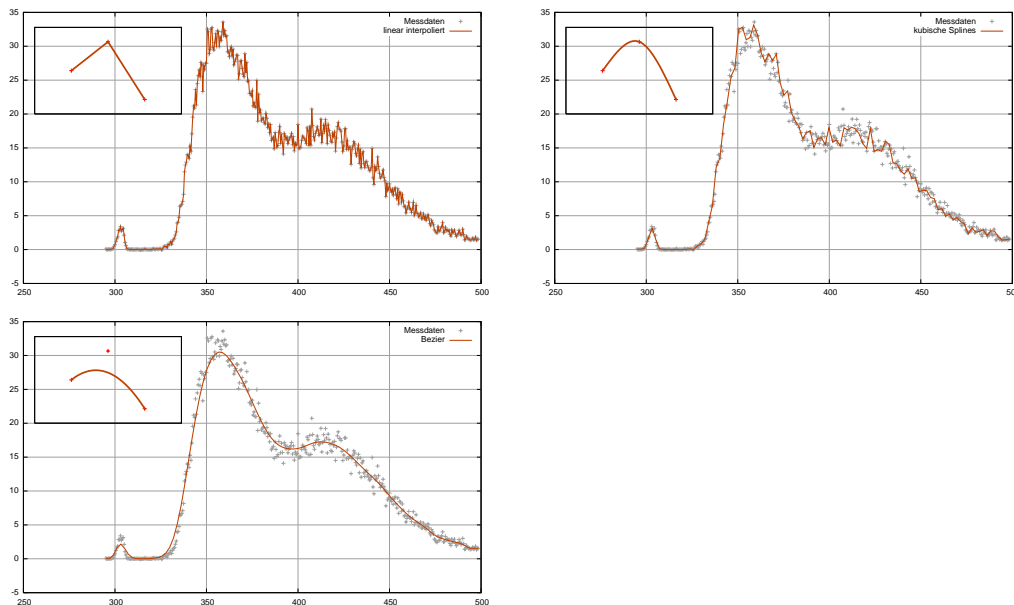


Abbildung 2: Zwei y-Achsen

1.6 Glättung

Eine einfache Methode, mit der man eine Kurve optisch aufbessern kann, ist die Glättung. GNUPLOT unterstützt hierfür verschiedene Interpolationsverfahren. Die gewünschte Methode wird dem `plot`-Befehl mit `smooth Methode` angehängt:

```
gnuplot> plot "data.dat" with lines smooth csplines
```



Hier wurden Messdaten einmal linear interpoliert (`smooth unique`), einmal mit kubischen Splines (`smooth csplines`) und einmal mit Bézierkurven (`smooth bezier`).

1.7 Kurvenanpassung

Die fortgeschrittene Methode ist die Kurvenanpassung. Hier wird eine Modellkurve solange variiert, bis ihre Abweichung von den Messpunkten minimal ist (*Methode der kleinsten Quadrate*).

GNUPLOT stellt hierfür den Befehl `fit` zu Verfügung. Die Syntax lautet

```
fit [x : x̄] [y : ȳ] Funktion Daten using Spalten via Variablen
```

- Grenzen:** die ersten beiden Ausdrücke sind optional. Mit ihnen kann der Bereich eingegrenzt werden, der für die Anpassung verwendet wird.
- Funktion:** die Modellfunktion, deren Parameter angepasst werden sollen
- Daten:** die Eingabedatei, z.B. "messwerte.dat"
- Spalten:** eine durch `:` getrennte Liste der Spalten, die aus der Datei verwendet werden soll; üblicherweise `1:2`
- Variablen:** eine durch `,` getrennte Liste der Variablen, die für die Optimierung verwendet werden sollen

Die Eingabe

```
gnuplot> f(x) = m*x + t
gnuplot> fit f(x) "daten.dat" using 1:2 via m,t
```

nähert also die Geradengleichung an die Daten in `daten.dat` an, wobei die ersten beiden Spalten als x - bzw. y -Vektor verwendet werden und die Parameter m und t variiert werden. GNUPLOT liefert dabei detaillierte Informationen zum Ergebnis der Ausgleichsrechnung:

Final set of parameters		Asymptotic Standard Error	
=====		=====	
m	= 0.992153	+/- 0.03224	(3.25%)
t	= -1.42523	+/- 1.866	(130.9%)

Hinweis: Die Funktion `fit` nutzt den *Levenberg-Marquardt-Algorithmus*, nähert sich also iterativ einer Lösung an. Dabei ist nicht gesichert, dass der Algorithmus konvergiert, also zu einer Lösung kommt. Es kann u.U. notwendig sein, Startwerte für die Parameter zu setzen, damit das Verfahren schneller bzw. überhaupt konvergiert. Beim obigen Beispiel hätte man also den Parameter `m` mit dem Befehl `m = 1` initialisieren können.

Lineare Regression

Im einfachsten Fall kann die Punktwolke durch eine Gerade genähert werden. Variiert wird dann

$$f(x) = mx + t$$

bzw. in GNUPLOT-Schreibweise

```
gnuplot> f(x) = m*x + t
```

Um die Kurve anzupassen und sie zusammen mit den Messdaten zu plotten, sind folgende Anweisungen nötig:

```
gnuplot> fit f(x) "daten.dat" using 1:2 via m,t
gnuplot> plot "daten.dat" title "Messung", f(x) title "Anpassung"
```

Allometrische Anpassung

Die klassische Allometrieformel lautet

$$f(x) = a \cdot x^b$$

bzw.

```
gnuplot> f(x) = a*x**b
```

Exponentielle Anpassung

Ein exponentieller Anstieg ist gegeben durch

$$f(x) = y_0 + Ae^{\frac{x-x_0}{t}}$$

y_0 : y -Verschiebung

x_0 : Zentrum

A : Amplitude

also

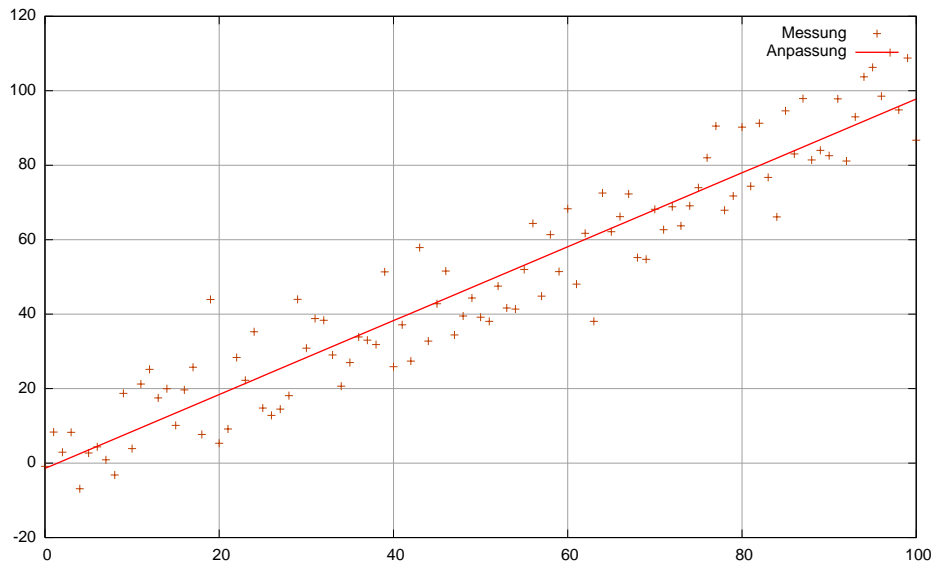


Abbildung 3: Lineare Regression

```
gnuplot> f(x) = y0 + A*exp((x-x0)/t)
```

Einen exponentiellen Abfall erreichen wir mit einem negativen Exponenten:

```
gnuplot> f(x) = y0 + A*exp(-(x-x0)/t)
```

Gauß-Anpassung

Die Modellfunktion für eine Gauß-Verteilung lautet

$$f(x) = y_0 + \frac{A}{w\sqrt{\frac{\pi}{2}}} e^{-2\left(\frac{x-x_c}{w}\right)^2}$$

x_c : Zentrum
 $A > 0$: Fläche
 $w > 0$: Halbwertsbreite
 y_0 : y-Verschiebung

```
gnuplot> f(x) = y0 + (A/(w*sqrt(pi/2)))*exp(-2*((x - xc)/w)**2)
```

Hier ist es notwendig, gute Startwerte für die Parameter zu wählen, um ein gutes Ergebnis zu erreichen und um sicherzustellen, dass der Algorithmus konvergiert. Es genügt im allgemeinen, die Parameter y_0 und x_c zu setzen.

```
gnuplot> f(x) = y0 + (A/(w*sqrt(pi/2)))*exp(-2*((x - xc)/w)**2)
gnuplot> y0 = 0
gnuplot> xc = 420
gnuplot> fit f(x) "daten.dat" using 1:2 via y0,xc,w,A
gnuplot> plot "daten.dat" t "Messung", f(x) t "Anpassung"
```

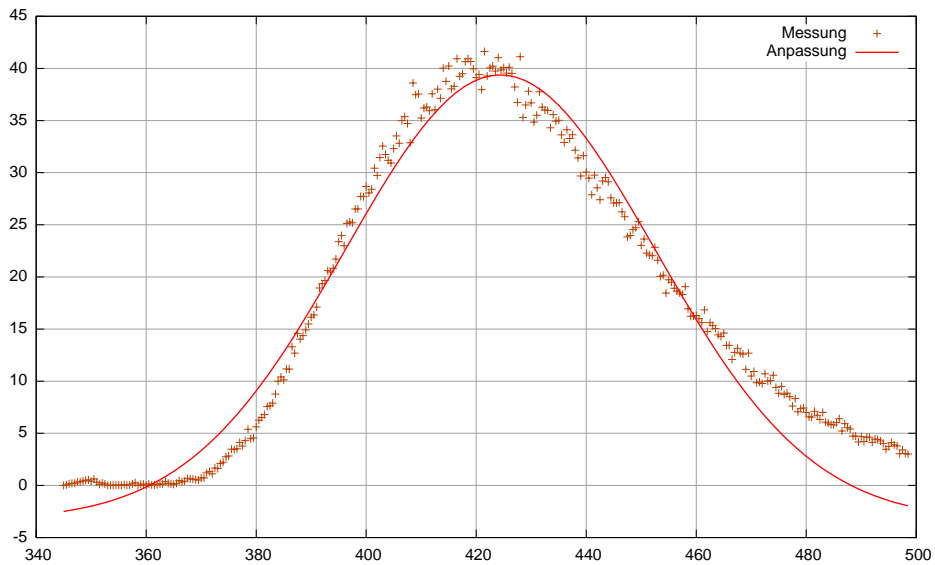


Abbildung 4: Anpassung durch Gaußfunktion

Lorentz-Anpassung

Die Modellfunktion einer Lorentz-Verteilung lautet

$$f(x) = y_0 + \frac{2A}{\pi} \cdot \frac{w}{4(x - x_c)^2 + w^2}$$

x_c : Zentrum
 $A > 0$: Fläche
 $w > 0$: Halbwertsbreite
 y_0 : y-Verschiebung

```
gnuplot> f(x) = y0 + 2*A/pi * w/(4*(x-xc)**2 + w**2)
```

Boltzmann-Anpassung

Hier lautet die Funktion

$$f(x) = A_2 + \frac{A_1 - A_2}{1 + e^{\frac{x-x_0}{dx}}}$$

A_1 : Startwert
 A_2 : Endwert
 x_0 : Zentrum
 dx : Zeitkonstante

```
gnuplot> f(x) = A2+(A1-A2)/(1+exp((x-x0)/dx))
```

Anpassung durch Polynome

Lässt sich keine geeignete Modellfunktion finden, kann man die Wolke auch durch Polynome beliebigen Grades annähern, wie z.B. $f(x) = ax^3 + bx^2 + cx + d$.

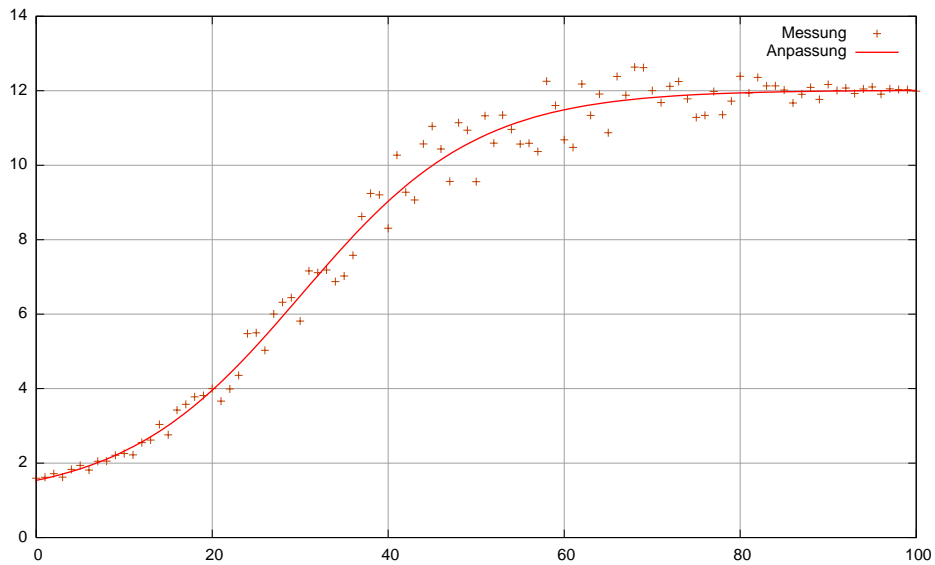


Abbildung 5: Anpassung durch Boltzmann-Funktion

1.8 Plots exportieren

Mit dem Befehl

```
set terminal Terminal
```

kann die Art der Ausgabe festgelegt werden. Voreingestellt ist der Terminal `x11`, d.h. der Graph wird in einem Fenster dargestellt.

Welche Terminals, also Ausgabeformate möglich sind, kann man mit

```
gnuplot> set terminal
```

anzeigen. Bei Ausgaben, die eine Datei erzeugen, kann der Dateiname mit

```
set output Dateiname
```

festgelegt werden.

Export als ASCII-Tabelle

Statt eine Funktion darzustellen, kann man ihre Werte auch in eine Datei schreiben um sie evtl. in anderen Programmen nutzen zu können:

```
gnuplot> set terminal table
gnuplot> set output "werte.dat"
gnuplot> plot f(x)
```

Die Werte werden somit als ASCII-Tabelle, also als lesbarer Text in `werte.dat` gespeichert.

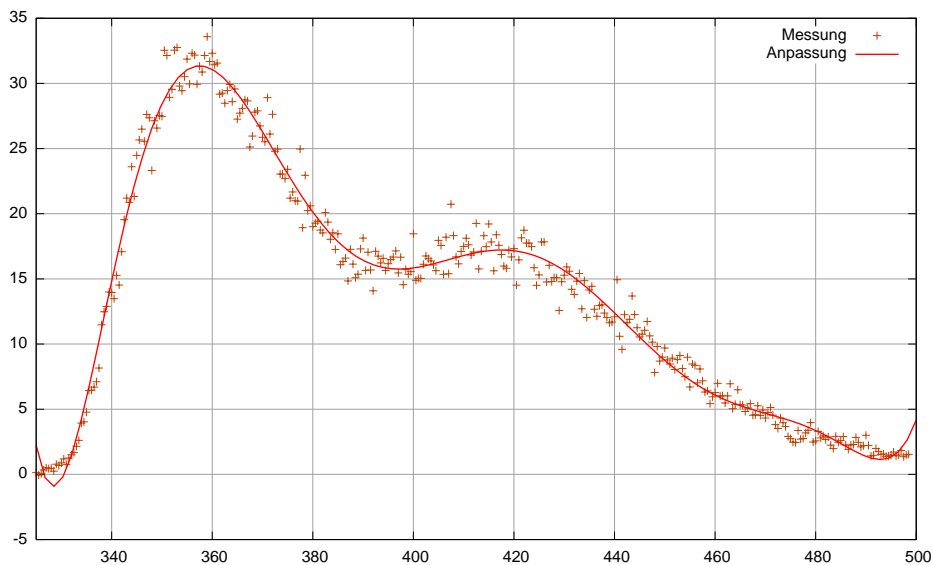


Abbildung 6: Anpassung durch ein Polynom 8. Grades

Export als Grafik

Die Ausgabe als PDF-Datei liefert die beste Qualität. Die Datei kann wiederum in ein andere PDF-Datei eingebunden (vgl. dieses Dokument) oder separat gedruckt werden.²

```
gnuplot> set terminal pdf enhanced
gnuplot> set output "bild.pdf"
gnuplot> replot
```

Das PNG-Format eignet sich wohl am besten für die Office-Umgebungen, liefert allerdings nur mäßige Qualität.

```
gnuplot> set terminal png
gnuplot> set output "bild.png"
gnuplot> replot
```

1.9 Skriptdateien

GNUPLOT-Befehle lassen sich natürlich auch in Skriptdateien zusammenfassen. Man kann sie dann mit

```
gnuplot Dateiname
```

ausführen.

Soll der Plot normal angezeigt und nicht als Bild gespeichert werden, muss man den Schalter `-persist` anhängen, damit das Fenster nicht sofort wieder geschlossen wird.

²Der Terminal `pdf` steht bei manchen Distributionen nicht zur Verfügung. In diesem Fall kann der Terminal `postscript` genutzt werden und die Ausgabe per `ps2pdf` o.ä. umgewandelt werden.

```
gnuplot -persist Dateiname
```

Solche Skriptdateien kann man auch nutzen, um lediglich Funktionen zu definieren, z.B. komplizierte Modellfunktionen für die Kurvenanpassung:

```
gauss(x) = y0 + (A/(w*sqrt(pi/2)))*exp(-2*((x - xc)/w)**2)
lorentz(x) = y0 + 2*A/pi * w/(4*(x-xc)**2 + w**2)
boltzmann(x) = A2+(A1-A2)/(1+exp((x-x0)/dx))
```

In GNUPLOT kann man dieses Skript dann mit

```
gnuplot> load funktionen.plt
```

ausführen und somit die Funktionen definieren.

1.10 Anwendungsbeispiele

Extrapolation einer Kurve

Oft möchte man monotone Bereiche einer Kurve durch Geraden nähern. Hierzu führen wir eine lineare Regression im entsprechenden Intervall durch:

```
gnuplot> g(x) = A*x + B
gnuplot> fit [35:40] g(x) "messung.dat" using 1:2 via A,B
gnuplot> plot "messung.dat" w lines t "Messung", g(x) t "Extrapolation"
```

Den Schnittpunkt mit der x-Achse erhalten wir aus den Parametern der Gerade:

```
gnuplot> print -B/A
```

2 Numerische Berechnungen in OCTAVE

Mit GNUPLOT können wir Funktionen oder Daten darstellen und Graphen manipulieren. Für komplexere Berechnungen dient OCTAVE, ein numerisches Computeralgebrasystem. Es ist weitgehend kompatibel zu MATLAB, im Gegensatz dazu jedoch freie Software. OCTAVE wurde ursprünglich für Chemie-Studenten an der UNIVERSITY OF WISCONSIN entwickelt.

2.1 Skalare, Vektoren und Matrizen

Die wichtigste Datenstruktur ist die Matrix. Selbst Skalare werden intern als 1x1 Matrix behandelt. Wir definieren z.B. die Variable `meinSkalar` mit dem Wert 0.5: ³

```
octave> meinSkalar = .5
```

OCTAVE antwortet mit

```
meinSkalar = 0.50000
```

Will man diese Ausgabe unterdrücken (übersichtlicher!), so kann man dies mit einem Semikolon am Ende der Zeile bewirken.

Ein Vektor ist eine Matrix mit nur einer Zeile bzw. Spalte:

```
octave> zeilenVektor = [ 2 4 8 ]
zeilenVektor =

  2  4  8
```

Um einen Spaltenvektor zu erzeugen, trennt man die Werte durch Semikolons:

```
octave> spaltenVektor = [ 2; 4; 8 ]
spaltenVektor =

  2
  4
  8
```

Der Übergang zu echten Matrizen liegt nahe.

```
octave> Matrix3x3 = [ 1 2 3; 4 5 6; 7 8 9 ]
Matrix3x3 =

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

Um Matrizen oder Vektoren zu transponieren, wird ein Apostroph benutzt:

```
octave> Matrix3x3 '
ans =

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

³wie in Programmiersprachen üblich kann die Null vor dem Komma weggelassen werden

Auf die Elemente der Matrix kann durch Indizes (von 1 aufwärts) zugegriffen werden:

```
octave> Matrix3x3(2, 3)
ans = 6
```

Hier wurde das Ergebnis nicht in einer Variablen gespeichert. Die Ausgabe erfolgt dann in `ans` (für *answer*).

Um mehr als ein Element auszuwählen, geben wir Bereiche an. Wir wählen z.B. den zweiten und dritten Wert in der zweiten Zeile aus:

```
octave> Matrix3x3(2, 2:3)
ans =
    5    6
```

Wir erhalten einen Zeilenvektor mit den gewünschten Werten. Lässt man die Grenzen des Bereichs weg, erhält man den gesamten Bereich, also die ganze Zeile:

```
octave> Matrix3x3(2, :)
ans =
    4    5    6
```

2.2 Der Workspace

Informationen über alle bereits definierten Variablen, den sog. *Workspace*, erhält man mit dem Befehl `whos`.⁴ Der Befehl gibt außerdem Aufschluss darüber, wieviel Speicherplatz die Variablen belegen.

```
octave> whos

*** local user variables:

  Prot Name              Size      Bytes  Class
  ==== =====
  rwd Matrix3x3          3x3        72  matrix
  rw- __nargin__         1x1         8  scalar
  rwd meinSkalar         1x1         8  scalar
  rwd spaltenVektor     3x1        24  matrix
  rwd zeilenVektor      1x3        24  matrix

Total is 17 elements using 136 bytes
```

2.3 Bereiche

Es ist oft notwendig, einen Vektor mit Zahlen in einem bestimmten Bereich zu füllen. Hierfür gibt es die folgende Notation:

```
octave> x = 5 : 15
x =
```

⁴`__nargin__` ist eine spezielle Variable, die immer definiert ist

```
5 6 7 8 9 10 11 12 13 14 15
```

Per Vorgabe wird hierbei immer um 1 erhöht, was sich natürlich auch ändern lässt:

```
octave> x = 1 :.25: 2
x =
 1.0000  1.2500  1.5000  1.7500  2.0000
```

2.4 Arithmetik

Alle arithmetischen Operation werden grundsätzlich als Matrix- bzw. Vektoroperationen verstanden. Das heißt:

```
octave> [1 2 3] * [1; 2; 3]
ans = 14
```

Möchte man eine Operation elementweise durchführen, muss man dem Operator einen Punkt voranstellen. Das betrifft insbesondere die Division, da sie für Matrizen nicht definiert ist und so in OCTAVE eine besondere Bedeutung hat:

```
octave> [1 2 3] .* [1 2 3]
ans =
```

```
1 4 9
```

```
octave> [1 2 3] ./ [1 2 3]
ans =
```

```
1 1 1
```

Natürlich muss man beachten, dass man elementweise Operationen nur auf gleichförmige Objekte anwenden kann:

```
octave> [1 2 3] .* [1; 2; 3]
error: product: nonconformant arguments (op1 is 1x3, op2 is 3x1)
error: evaluating binary operator `.*'
```

2.5 Funktionen

Wie auch in der Mathematik spielen Funktionen in OCTAVE eine wichtige Rolle. OCTAVE stellt eine riesige Zahl an eingebauten Funktionen aus den Bereichen Statistik, Signalanalyse, lineare Algebra, Finanzwesen uvm. bereit. Der Aufruf ist denkbar einfach. Hier der Sinus⁵:

```
octave> sin(1)
ans = 0.84147
```

oder die Determinante einer Matrix:

```
octave> A = [-2 3 0 1; 1 3 -3 4; 0 6 -1 4; 2 1 7 4]
A =
```

⁵Achtung: die Winkelfunktionen in OCTAVE erwarten ihr Argument im Bogenmaß!

```
-2  3  0  1
 1  3 -3  4
 0  6 -1  4
 2  1  7  4
```

```
octave> det(A)
ans = -201.00
```

Funktionen können natürlich auch verschachtelt werden:

```
octave> xu = [ 1 0 -1 ];
octave> xv = [ 0 1 -1 ];
octave> norm(cross(xu,xv))
ans = 1.7321
```

Hier wurde $\|x_u \times x_v\|$ berechnet, mit $x_u = (1, 0, -1)$ und $x_v = (0, 1, -1)$. Das Ergebnis entspricht $\sqrt{3}$.

2.6 Eingebaute Hilfe

Mit `help Funktionsname` lässt sich zu jeder Funktion eine Hilfeseite anzeigen. Hier die Hilfe zum Sinus:

```
octave> help sin
sin is a built-in mapper function

-- Mapping Function:  sin (X)
   Compute the sine of each element of X.
```

Wie man sieht, kann die Funktion auch Vektoren⁶ verarbeiten.

2.7 Plotten

Man kann auch aus OCTAVE heraus Graphen plotten. OCTAVE nutzt hierfür wiederum GNU-PLOT. Der Funktionsumfang ist hier aber etwas eingeschränkt.

Wir plotten z.B. den Kosinus in $[-2\pi, 2\pi]$:

```
octave> x = -2*pi : .1 : 2*pi;
octave> y = cos(x);
octave> title "Darstellung einer Winkelfunktion"
octave> xlabel "x"
octave> ylabel "cos(x)"
octave> plot(x, y, ";Kosinus;")
```

Die Funktion `plot` erwartet also den x- und y-Vektor. Als drittes Argument kann eine Formatbeschreibung übergeben werden. Hier wird die Linie beschriftet; genaueres erfährt man mit `help plot`. Mit `title`, `xlabel` und `ylabel` können Überschriften gesetzt und Achsen beschriftet werden.

⁶hier im Sinne von Zahlenreihe (vgl. `array`)

2.8 Import/Export von Daten

Messdaten werden für gewöhnlich im ASCII-Format, also als Klartext spaltenweise in Textdateien abgelegt. Solche Dateien können mit

```
load Dateiname
```

geladen werden:

```
octave> load "daten.dat"
octave> whos

*** local user variables:

  Prot Name          Size          Bytes  Class
  ==== =====          =====
  rw-  __margin__    1x1             8  scalar
  rwd  daten         408x2          6528  matrix

Total is 817 elements using 6536 bytes
```

Der Aufruf von `whos` zeigt uns, dass der Inhalt der Datei nun als Matrix unter dem Namen `daten` verfügbar ist. Um den Zugriff zu vereinfachen können wir die Werte auf die gewohnten Vektoren `x` und `y` verteilen:

```
octave> x = daten(:, 1);
octave> y = daten(:, 2);
```

Mit

```
save Dateiname Variable1 Variable2 ...
```

kann der Inhalt von Variablen in eine Textdatei gespeichert werden. Übergibt man nur den Dateinamen, wird der gesamte Workspace, also alle momentan definierten Variablen in der Datei gespeichert.

Will man das übliche Spaltenformat erzeugen, muss man die `x`- und `y`-Werte als Spaltenvektoren zu einer Matrix verbinden und diese speichern:

```
octave> x = -5 : 5;
octave> y = x.^2;
octave> A = [x' y'];
octave> save "parabel.dat" A
```

2.9 3D-Plotten

Für die Darstellung von dreidimensionalen Funktionen steht die Funktion `mesh` zur Verfügung. Dabei ist zu beachten, dass die Vektoren der beiden Laufvariablen zunächst mit `meshgrid` auf Matrizen *gestreckt* werden müssen⁷:

⁷`meshgrid` erzeugt dabei aus $\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$ die Matrix $\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$

```
x = -2 : .1 : 2;
y = -2 : .1 : 2;
[xx, yy] = meshgrid(x, y);
z = sin(xx.^2 - yy.^2);
xlabel "x"
ylabel "y"
zlabel "z"
mesh(x, y, z)
```

2.10 .m-Dateien

Anstatt eine Reihe von Befehlen immer wieder einzutippen, kann man sie auch in einer .m-Datei speichern und diese dann in OCTAVE aufrufen. Hierzu wechselt man mit `cd` (*change directory*) in das Verzeichnis, in dem die Datei abgelegt ist und gibt dann den Dateinamen des Skripts ohne Endung ein.

2.11 Eigene Funktionen

Solche .m-Dateien sind aber etwas vielseitiger als einfache Skripte, wie wir sie von GNUPLOT kennen. Sie bieten die Möglichkeit, eigene Funktionen zu definieren.

Die sinc-Funktion (Kardinalsinus) ist zwar bereits in OCTAVE vorhanden, wir schreiben sie aber als Beispiel für eine eigene Funktion neu. Es gilt:

$$\text{sinc}(x) := \begin{cases} \frac{\sin(x)}{x} & \text{für } x \neq 0 \\ 1 & \text{für } x = 0 \end{cases}$$

Der Rumpf einer Funktion in OCTAVE lautet

```
function Rückgabewert = Funktionsname (Parameter)

endfunction
```

Der *Funktionsname* muss die Funktion eindeutig kennzeichnen, der *Rückgabewert* ist ein Skalar, ein Vektor oder eine Matrix, die uns die Funktion nach ihrem Aufruf liefert. Der oder die *Parameter* sind die Eingabewerte, mit denen die Funktion arbeitet. Für unsere Funktion lautet dieser Rumpf also

```
function y = meinsinc (x)

endfunction
```

Innerhalb der Funktion muss der Wert `y` nun definiert werden. Wir müssen eine Fallunterscheidung treffen; es gilt also in OCTAVE zu formulieren:

$$\text{Wenn } x = 0 \text{ ist, dann ist } y := 1; \text{ ansonsten ist } y := \frac{\sin(x)}{x}.$$

Das entspricht⁸:

⁸der Operator `==` vergleicht zwei Werte, während `=` einen Wert zuweist!

```

if x == 0
    y = 1;
else
    y = sin(x)./x;
endif

```

Diesen Code speichern wir zusammen mit dem Rumpf in `meinsinc.m`. Ab jetzt ist die neue Funktion in OCTAVE verfügbar und kann genauso wie `sinc` verwendet werden.

2.12 Genauigkeit und Ausgabeformat

OCTAVE rechnet immer mit der maximal möglichen Genauigkeit (üblicherweise *double*, also 15 bis 16 Dezimalstellen). Bei der Ausgabe werden der Übersicht halber aber nur 4 Stellen angezeigt. Mit dem Befehl `format long` kann auf eine genauere Darstellung umgeschaltet werden:

```

octave> pi
pi = 3.1416
octave> format long
octave> pi
pi = 3.14159265358979

```

oder auch

```

octave> format bit
octave> pi
pi = 0100000000001001001000011111101101010100010001000010110100011000

```

Außerdem ist zu beachten, dass viele Probleme nicht direkt gelöst sondern nur numerisch genähert werden können. So liefert OCTAVE zum Beispiel folgende Ausgabe:

```

sin(pi)
ans = 1.2246e-16

```

Dieser Wert liegt an der Grenze der oben genannten Genauigkeit und kann daher als 0 interpretiert werden.

Prinzipiell kann in OCTAVE auch mit Unendlichkeit gerechnet werden. Für ∞ steht der besonderer Zahlenwert `Inf`:

```

octave> x = 1/0
x = Inf
octave> 3 * x
ans = Inf

```

2.13 Kurzer Überblick über den Funktionsumfang von OCTAVE

Lösen eines linearen Gleichungssystems

Der Befehl

```

octave> x = A\b

```

löst das Gleichungssystem $A\vec{x} = \vec{b}$.

Inverse, Zerlegungen und Eigenwerte

```
octave> B = inv(A)
```

berechnet die Inverse zur Matrix A .

```
octave> [L,U,P] = lu(A)
octave> [Q,R] = qr(A)
octave> R = chol(A)
```

berechnet die Dreieckszerlegung $LU = PA$, die QR-Zerlegung $QR = A$ oder die Cholesky-Zerlegung von A .

```
octave> E = eig(A)
octave> [V,D] = eig(A)
```

berechnet entweder nur die Eigenwerte von A oder eine Diagonalmatrix D mit den Eigenwerten und eine Matrix V , die die Eigenvektoren enthält ($AV = VD$).

2.14 Anwendungsbeispiele

Hintergrund subtrahieren

Bei vielen Messverfahren wird vor der eigentlichen Messung zunächst ein Hintergrund aufgenommen. Bei der Auswertung muss dieser dann von der Messreihe subtrahiert werden.

Wir importieren also beide Messreihen aus einer ASCII-Datei:

```
octave> load "background.dat"
octave> load "messung.dat"
```

Anschließend kopieren wir die geladenen Daten in geeignete Vektoren:

```
octave> x = messung(:,1);
octave> y = messung(:,2);
octave> yb = background(:,2);
```

Nun können wir den Hintergrund von der Messreihe abziehen und speichern das Ergebnis in yf :

```
octave> yf = y - yb;
```

Die Daten können dann wieder in einer ASCII-Datei gesichert werden:

```
octave> A = [yf x];
octave> save "messung_f.dat" A
```

Fourier-Transformation

Um das Spektrum einer Messreihe zu berechnen nutzen wir ihre Fourier-Transformierte. Genauer ihren Betrag, da wir uns nicht für komplexwertige Anteile interessieren.

Computer berechnen die diskrete Fourier-Transformation mit Hilfe der *Fast Fourier Transform (FFT)*, einem Algorithmus, der sich gegenseitig aufhebende Terme nicht berechnet und daher wesentlich schneller ist als die direkte Berechnung.

OCTAVE stellt hierfür die Funktion `fft(A)` zur Verfügung. Sie berechnet die FFT für jede Spalte der Matrix A .

Im folgenden werden wir das Fourier-Spektrum eines Kardinalsinus berechnen und darstellen.

Zunächst die Ausgangsfunktion im Zeitbereich:

```
octave> t = -20 : .2 : 20;  
octave> y = sinc(t);
```

Wir berechnen mit `fft` die Fourier-Transformierte, verschieben das Ergebnis mit `fftshift` und berechnen mit `abs` den Betrag der entstehenden Funktion:

```
octave> Y = abs(fftshift(fft(y)));
```

Nun legen wir eine geeignete Frequenzachse von $-\frac{f_A}{2}$ bis $\frac{f_A}{2}$ an:

```
octave> n = length(t);  
octave> f = [-n/2 : n/2-1]/n;
```

Anschließend können wir das Spektrum darstellen:

```
octave> xlabel "f/f_A"  
octave> ylabel "Y(f)"  
octave> plot(f, Y, ";Fourier-Spektrum des sinc;")
```

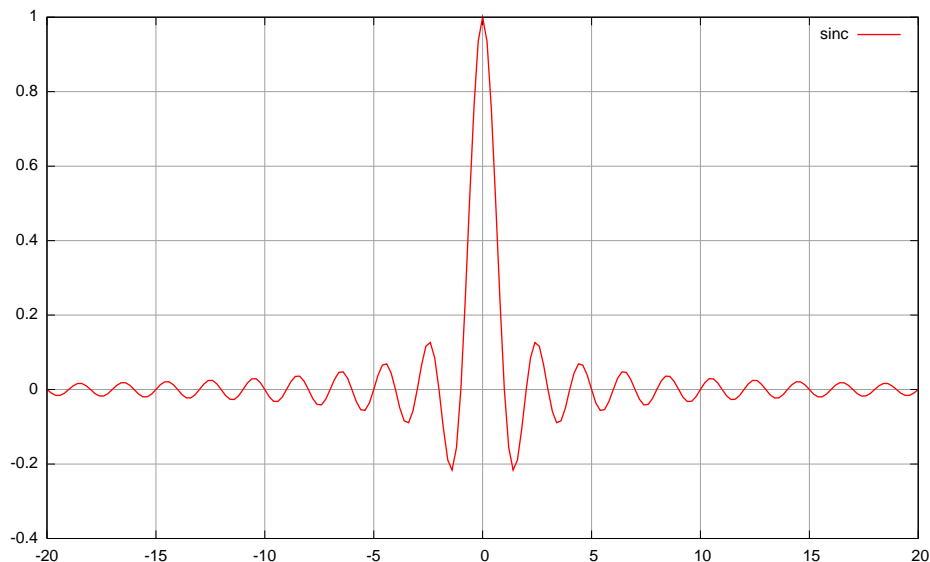


Abbildung 7: Die sinc-Funktion im Zeitbereich

Spektrum einer Messreihe

Da wir die Fourier-Transformierte einer Reihe im allgemeinen als das Frequenzspektrum von Messwerten interpretieren, ist der negative Frequenzbereich für uns uninteressant.

Wir schreiben also eine Funktion, die das Fourier-Spektrum einer Messreihe `y` berechnet und das genannte Spiegelbild dabei löscht. Unsere Funktion `spectrum` hat folgende Form:

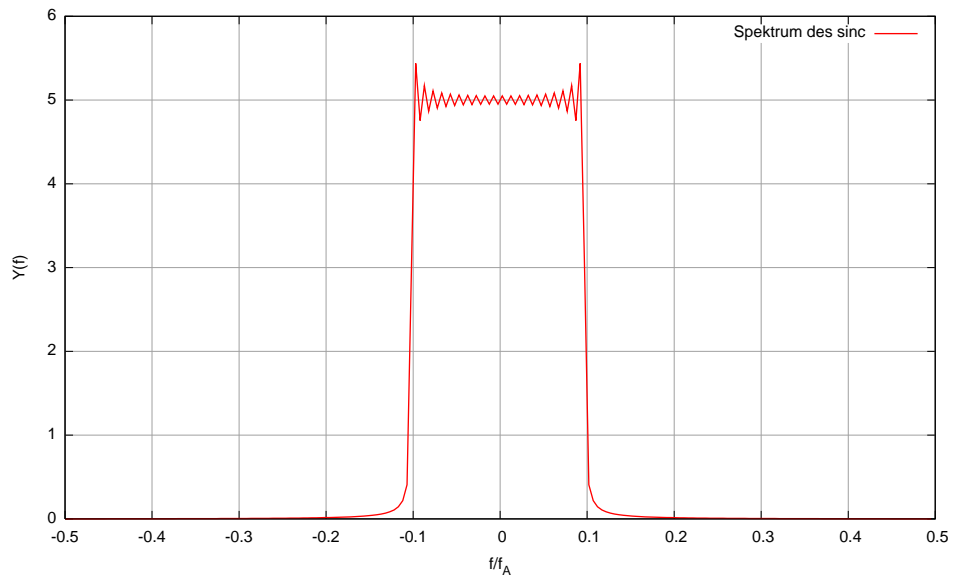


Abbildung 8: Das Fourier-Spektrum der sinc-Funktion

```
function [f, Y] = spectrum( y )
    n = length(y);
    Y = abs(fftshift(fft(y)));
    Y = Y(fix(n/2) : n-1);
    f = [0 : n/2-1]/n;
end function
```

Die Funktion lässt sich dann problemlos auf ASCII-Daten anwenden:

```
octave> load "daten.dat"
octave> y = daten(:, 2);
octave> [f, Y] = spectrum(y);
octave> plot(f, Y)
```