Eine kurze Einführung in **scilab**

von Werner Schabert
April 2003

Lehrstuhl für Angewandte Mathematik mit Schwerpunkt Numerik
Universität Augsburg

Inhaltsverzeichnis

1 Rechenoperationen und mathematische Funktionen

Da es sich bei **scilab** um eine mathematische Programmiersprache handelt, beherrscht **scilab** alle Grundrechenarten, inklusive der gängigen Rechenregeln (wie z.B. Punkt vor Strich). Ein Großteil der gebräuchlichsten mathematischen Funktionen sind bereits implementiert, zusätzlich können auch neue Funktionen in Scilab erstellt werden (mehr dazu in Abschnitt 5). Die Umsetzung der einzelnen Rechenoperationen bzw. Funktionen kann man aus Tabelle 1 entnehmen.

Operation	Umsetzung in **scilab**
$5 + 6$	5 + 6
$7 - 9$	7 - 9
$8 \cdot 6$	8 * 6
$13 : 3$	13/3
7^3	7^3
e^5	exp(5)
sinus(a)	sin(a)
cosinus(a)	cos(a)
$log_e(15)$	log(15)
$log_{10}(43)$	log10(43)
$log_2(11)$	log2(11)

Tabelle 1: Rechenopertationen in **scilab**

Bei der cosinus und sinus Funktion ist dabei zu beachten, dass der Winkel im Bogenmaß angegeben werden muss. Auch besitzt **scilab** nur runde Klammern () und keine eckigen [] bzw. geschweiften {} Klammern. Als Beispiel sei hier die einfache Rechnung $sin([20 * (70 - 3/5) + 5]^2)$ angegen, die in **scilab** so aussieht

```
-->sin((20*(70-3/5)+5)^2)
  ans  =

    0.3100191
```

Wie man sieht, kann man **scilab** im einfachsten Fall lediglich als Taschenrechner benutzen. Zusätzlich bietet aber **scilab** noch eine Vielzahl von anderen Anwendungsarten, die nun in den einzelnen Abschnitten vorgestellt werden.

2 Variablen, Vektoren und Matrizen

Neben der Durchführung von Rechenoperationen mit Zahlen ist es auch möglich diese Ergebnisse zu Speichern, oder ganz allgemein Zahlen in Variablen zu speichern. In **scilab** stehen dabei drei verschiedene Speichergrößen zur Verfügung: einfache Variablen (enthalten nur eine Zahl), Vektoren (können n verschiedene Zahlen enthalten) und Matrizen (können $m \cdot n$ verschiedene Zahlen enthalten).

Die einfachste Speichergröße ist die Variable. Variablen enthalten nur ein Element und werden durch

Variablenname = Zahl

Variablenname = Rechenoperation

belegt. Also z.B.

```
-->a=5
 a  =

    5.

-->b=exp(13)
 b  =

    442413.39

-->hans=5*log2(30)
 hans  =

    24.534453

-->hans=5*log2(30);

-->

-->hans
 hans  =

    24.534453
```

Eine Eigenart von **scilab** ist es, dass nach Definition einer Variablen, der Variablenname und der Inhalt angezeigt werden, wie man an obigem Beispiel sehen kann. Die Ausgabe des Variablennamens und -inhalts kann man aber auch Unterdrücken, indem man einen „;“ am Ende der Definition hinzufügt. Will man schließlich den Inhalt einer Variablen wissen, so braucht man nur den Namen der Variablen in der Kommandozeile eigeben.

Oft ist es nützlich, mehrere Zahlen unter einem Namen abzuspeichern. **scilab** bietet hierfür Vektoren an. In **scilab** lassen sich im Gegensatz zu anderen reinen Programmiersprachen Zeilen- und Spaltenvektoren unterscheiden. Ein Vektor läßt sich dabei auf mehrere Arten mit Zahlenwerten besetzen:

- Komponentenweise (nur Zeilenvektor): Vektor(1)=a1, Vektor(2)=a2, Vektor(3)=a3, ...
- Im Ganzen (Zeilenvektor): Vektor=[a1,a2,a3,...]
- Im Ganzen (Spaltenvektor): Vektor=[a1;a2;a3;...]
- Soll der Vektor nur Elemente enthalten, die der Größe nach geordnet sind und jeweils einen festen Abstand voneinander haben, so läßt sich ein solcher Vektor (nur Zeilenvektor) durch

  Vektor=Anfangswert:Abstand:Endwert

  erzeugen. Ist der Abstand lediglich 1, so kann er auch weggelassen werden. Der Befehl ist dann

  Vektor=Anfangswert:Endwert

Wie man sieht unterscheidet sich die Definition eines Spalten- von der eines Zeilenvektors nur durch die Verwendung von „,“ bzw. „;“. Beispiele hierfür sind

```
-->a(1)=5
 a  =

    5.

-->a(2)=13
 a  =

!   5.  !
!   13. !

-->a(3)=9
 a  =

!   5.  !
!   13. !
!   9.  !

-->Monate=[1,2,3,4,5,6,7,8,9,10,11,12]
 Monate  =

         column  1 to 11

!   1.    2.    3.    4.    5.    6.    7.    8.    9.    10.    11. !

         column 12

!   12. !

-->d=[1;2]
 d  =
```

```
!   1. !
!   2. !

-->c=1:0.5:3
 c  =

!   1.    1.5    2.    2.5    3. !

-->c=1:5
 c  =

!   1.    2.    3.    4.    5. !
```

Auch hier kann man die Ausgabe der Vektorinhalte durch das Hinzufügen eines „;“ unterbinden. Schließlich kann man auch noch mehrere Vektoren in eine Matrix zusammenfassen. Die Beschreibung einer Matrix kann dabei wieder auf mehrere Arten beschreiben:

- Komponentenweise: Matrix(1,1)=a11, Matrix(1,2)=a12, Matrix(2,1)=a21, Matrix(2,2)=a22, allgemein Matrix(Zeile,Spalte)=Zahl
- Im Ganzen: Matrix=[a11,a12;a21,a22]

Bei der zweiten Definitionsart (im Ganzen), werden die Elemente einer Zeile durch „,“ und die Zeilen durch „;“ voneinander getrennt. Genau betrachtet handelt es sich bei einem Vektor um einen Spezialfall der Matrix. In **scilab** lassen sich deshalb auch Zeilen- und Spaltenvektoren unterscheiden. Ein Spaltenvektor kann dabei nur Auch hier wieder zwei Beispiele

```
-->a(1,1)=1
 a  =

    1.

-->a(1,2)=2
 a  =

!   1.    2. !

-->a(2,1)=3
 a  =

!   1.    2. !
!   3.    0. !

-->a(2,2)=4
 a  =

!   1.    2. !
```

```
!   3.    4. !

-->b=[1,2;3,4]
 b  =

!   1.    2. !
!   3.    4. !
```

Der Zugriff auf Vektoren und Matrizen erfolgt dabei auch auf zwei Arten. Will man auf ein einzelnes Element zugreifen, so gibt man einfach die Indizes an,

a=Vektor(Komponente)
b=Matrix(Zeile,Spalte)

Will man aufeinanderfolgende Elemente eines Vektors in einen neuen Vektor schreiben, so kann man dies durch

b=a(Anfangsindex:Endindex)

erreichen. Bei Matrizen läßt sich auch auf ähnliche Weise auf Teilmatrizen zugreifen. Der Zugriff lautet

Matrix2=Matrix1(Anfangszeile:Endzeile,Anfangsspalte:Endspalte)

Beispiele

```
-->a=[1,2,3,4,5,6]
 a  =

!   1.    2.    3.    4.    5.    6. !

-->b=a(2:4)
 b  =

!   2.    3.    4. !

-->D=[1,2,3,4;5,6,7,8;9,10,11,12]
 D  =

!   1.    2.     3.     4.  !
!   5.    6.     7.     8.  !
!   9.    10.    11.    12. !

-->E=D(2:3,1:3)
 E  =

!   5.    6.     7.  !
!   9.    10.    11. !
```

Schließlich lassen sich bei Variablen, Vektoren und Matrizen auch alle Rechenoperationen aus Abschnitt 1 durchführen. So lassen sich zwei Variablen/Vektoren/Matizen addieren und subtrahieren. Wird eine mathematische Funktion auf einen Vektor/eine Matrix angewandt, so entsteht ein neuer Vektor/neue Matrix, dessen Komponenten das Ergebnis der Anwendung der Funktion auf die einzelnen Komponenten des Vektors/Matrix enthalten.

Beispiele

```
-->a=[1,2,3,4]
 a  =

!   1.    2.    3.    4. !

-->a=[1,2,3,4];

-->b=[5,6,7,8];

-->a+b
 ans  =

!   6.    8.    10.    12. !

-->M=[1,2,3,4;5,6,7,8];

-->N=[1,1,1,1;2,2,2,2];

-->M-N
 ans  =

!   0.    1.    2.    3. !
!   3.    4.    5.    6. !

-->exp(a)
 ans  =

!   2.7182818    7.3890561    20.085537    54.59815 !
```

Die Multiplikation und Division lassen sich jedoch nicht so leicht durchführen. Es treten hier mehrere Fälle auf.

- Variable/Zahl * Vektor/Matrix: Hier werden die einzelnen Komponenten des Vektors/Matrix mit der Variablen/Zahl multipliziert.
- Vektor1 * Vektor2: hier lassen sich drei Fälle unterscheiden, ist Vektor1 ein Zeilenvektor und Vektor2 ein Spaltenvektor und besitzen zusätzlich beide Vektoren die gleiche Anzahl von Elementen, so entspricht das Produkt dem Skalarprodukt. Ist Vektor1 ein Spaltenvektor mit n und Vektor2 ein Zeilenvektor mit m Elementen, so ergibt das Produkt eine $n \times m$ Matrix. Bei allen anderen Kombinationen tritt eine Fehlermeldung auf.

- Vektor * Matrix: Dies liefert nur ein Ergebnis, wenn der Vektor ein Zeilenvektor ist und die Matrix genauso viele Zeilen hat wie der Vektor Elemente. Das Ergebnis ist ein Zeilenvektor, der dem Vektor-Matrizen-Produkt aus der linearen Algebra entspricht.
- Matrix * Vektor: Dies liefert nur ein Ergebnis, wenn der Vektor ein Zeilenvektor ist und die Matrix genauso viel Spalten besitzt wie der Vektor Elemente hat. Das Ergebnis ist ein Spaltenvektor, der dem Matrizen-Vektor-Produkt aus der linearen Algebra entspricht.
- Matrix1 * Matrix2: Dies liefert nur ein Ergebnis, wenn die Anzahl der Spalten der Matrix1 der Anzahl der Zeilen der Matrix2 entspricht. Das Ergebnis entspricht dem Matrizen-Produkt aus der linearen Algebra.
- Vektor1 .* Vektor2 : Hier entsteht ein Vektor dessen Komponenten die Prokukte der entsprechenden Komponenten von Vektor1 und Vektor2 sind.
- Vektor1 ./ Vektor2 : Hier entsteht ein Vektor dessen Komponenten das Ergebnis der Division der entsprechenden Komponente von Vektor1 durch die entsprechende Komponente von Vektor2 sind.
- Analog zum letzten Punkt gibt es auch Matrix1 .* Matrix2 und Matrix1 ./ Matrix2, die das gleiche liefern wie zuvor nur jetzt mit den Komponenten der Matrizen.

3 Abfragen - if then else select

Oft kommt es in einem Programm vor, daß in Abhängigkeit von einem Wert verschiedene Befehle ausgeführt werden müssen. **scilab** enthält für diesen Fall die Abfrageroutine if-then-elseif-else, die wie folgt aufgebaut ist:

```
if Bedingung 1 then
   Befehl 1
   ...
   Befehl n
elseif Bedingung 2 then
   Befehl 1
   ...
   Befehl m
else
   Befehl 1
   ...
   Befehl k
end
```

Hierbei kann das thenäuch weggelassen werden. Die Ausdrücke "Bedingung 1ünd "Bedingung 2ßtehen dabei für eine logische Abfrage, die im allgemeinen

aus einem Vergleich bestehen. **scilab** besitzt fünf Vergleichsoperatoren: <, <=, ==, >= und >. Diese Vergleiche können durch die Operatoren "&" (und) und "|" (oder) verbunden werden. Folgendes kleines **scilab** Programm zeigt die Verwendung der Abfragen:

```
a=5;
b=10;

if a==b | b>9
  a=a^2;
  b=b^2;
elseif a<b & b<9
  a=b-a;
else
  a=a+b;
end

a  =

    25.
 b  =

    100.
```

Hängen vom Wert einer Variablen sehr viele Entscheidungen ab, z.B. ob sie kleiner 1 oder 2 oder 3 oder ... ist, so kann eine Umsetzung der Abfragen durch eine if-then-else Kette sehr umständlich sein. Unter **scilab** kann diese Abfrage durch den Befehl "select" sehr vereinfacht werden. Der Aufbau ist dabei

```
select Variable
case Wert 1 then
   Befehl 1
   ...
   Befehl n
case Wert 2 then
   Befehl 1
   ...
   Befehl m
else
   Befehl 1
   ...
   Befehl k
end
```

Folgendes Beispiel zeigt die Umsetzung des "select" Befehles:

```
a=2;
```

```
select a
  case 1
    b=a^2;
  case 2
    b=a;
  else
    b=a+7;
end

 b  =

    2.
```

4 Schleifen - loops

Soll eine Folge von Befehlen öfters nacheinander wiederholt werden, so kann man anstatt die Befehle mehrmals zu schreiben, diese durch eine Schleife abarbeiten lassen. In **scilab** existieren zwei Arten von Schleifen, die **for** und die **while** Schleife.

Die **for** Schleife ist dann von Nutzen, wenn man bereits weiß, wie oft ein Folge von Befehlen wiederholt werden soll. Sie wird durch die Befehle

```
for zaehler=beginn:schrittweite:ende
   Befehl 1
   Befehl 2
   ...
end
```

erzeugt. Hierbei bezeichnet die Variable *zaehler* den Zähler, der zählt, wie oft die Schleife durchlaufen werden soll. Mit *beginn* wird festgelegt, mit welcher Zahl die Zählung beginnen soll, und mit *end* bis zu welcher Zahl gezählt werden soll. Die Schrittweite mit der gezählt werden soll ist dabei durch *schrittweite* definiert. Ist die Schrittweite lediglich eins, so kann sie auch weggelassen werden und man kann verkürzt

```
for zaehler=beginn:ende
```

schreiben. Der *zaehler* wird nach dem Durchlaufen der Schleife gelöscht. Folgendes Beispiel zeigt die Berechnung der linearen Funktion $y = 5 * x + 13$ an den Punkten $x = 1, 1.5, 2, ..., 9.5, 10$:

```
-->for x=1:0.5:10
-->y(x)=5*x+13;
-->end
```

```
-->e
 e  =

!   20.5 !
!   25.5 !
!   30.5 !
!   35.5 !
!   40.5 !
!   45.5 !
!   50.5 !
!   55.5 !
!   60.5 !
!   63.  !
```

Die **while** Schleife ist hingegen von Nutzen, wenn die genaue Anzahl der Wiederholungen nicht vorab bekannt ist sondern durch eine Abbruchbedingung bestimmt wird. Der Syntax lautet

```
while Abbruchbedingung
   Befehl 1
   Befehl 2
   ...
end
```

Bei der Abbruchbedingung können auch mehrere Bedingungen durch und "&" oder oder "|" verknüpft werden. Folgendes Beispiel prüft, wie oft eine Zahl hintereinander durch zwei ganzzahlig geteilt werden kann.

```
-->j=128;

-->i=0;

-->while j>1
-->j=j/2;
-->i=i+1;
-->end

-->i
 i  =

    7.
```

Schließlich kann sowohl die **for** als auch die **while** schleife jederzeit durch den Befehl **break** unterbrochen werden. Das Programm wird dann nach der Schleife fortgesetzt.

5 Funktionen

Wird eine immer gleiche Folge von Befehlen an mehreren Stellen des Programmes verwendet, so bietet **scilab** die Möglichkeit, diese Befehle zu einer Funktion zusammenzufassen. Die Abarbeitung der Befehlsfolge geschieht dann einfach durch den Aufruf der Funktion. Da Funktionen oft Werte von Variablen, Vektoren oder Matrizen verändern, müssen zum einen Werte an die Funktion übergeben werden und zum anderen definiert werden, welche Werte von der Funktion zurückgegeben werden sollen. Die Definition einer Funktion erfolgt dabei durch

funktion [E1, E2, ..., E n]=Funktionsname(Ü1, Ü2, ..., Ü m)

Befehle

endfunction

Wobei die Ei's für die Ergebniswerte und die Üi's für die Übergabewerte stehen. Der Aufruf der Funktion erfolgt durch

[var1, var2, ..., var n]=Funktionsname(wert1, wert2, ..., wert m)

Hierbei müssen beim Aufruf die Übergabewerte nicht die gleichen Namen haben, wie die Übergabewerte bei der Definition der Funktion. Gleiches gilt für die Ergebniswerte. Beim Aufruf der Funktion wird von jedem Übergabewert eine Kopie angelegt und nur diese Kopie wird von der Funktion verwendet bzw. verändert. Das heißt, daß die Übergabewerte nach dem Beenden der Funktion dieselben Werte haben wie vor dem Funktionsaufruf, auch wenn sie in der Funktion verändert wurden. Will man nun einen Übergabewert durch die Funktion verändern, so muss man den Übergabewert nochmal bei den Ergebniswerten aufführen. Eine Funktion läßt sich jederzeit durch den Befehl **return** verlassen.
Beispiel

```
-->function [e,f]=tausch1(c,d)
-->e=d;
-->f=c;
-->endfunction

-->function tausch2(c,d)
-->g=c;
-->c=d;
-->d=g;
-->endfunction

-->a=15;

-->b=11;

-->tausch2(a,b)
```

```
-->a
 a  =

    15.

-->b
 b  =

    11.

-->[a,b]=tausch1(a,b)
 b  =

    15.
 a  =

    11.
```

Anders als bei anderen Programmiersprachen, in denen alle Variablen, die in der Funktion benutzt werden, an die Funktion übergeben werden oder in der Funktion definiert werden müssen, können in **scilab** auch Variablen/Vektoren/Matizen in der Funktion verwendet werden, die weder an die Funktion übergeben noch in der Funktion definiert wurden. Dies funktioniert allerdings nur, wenn eine Variable/Vektor/Matrix mit dem gleichen Namen bereits im Hauptprogramm existiert. In diesem Fall wird dann die Variable/Vektor/Matrix aus dem Hauptprogramm verwendet.

6 Grafikbefehle

Zur Ausgabe von Graphik ist es wichtig zunächst einmal ein Graphikfenster zu öffnen. Dies geschieht mit dem Befehl

xset("window",n)

wobei n für die Nummer des Graphikfensters steht. Die Angabe der Nummer ist wichtig, da oft mehrere Bilder gleichzeitig auf dem Bildschirm sind und man zwischen diesen hin und herschalten muß. Existiert das Fenster mit der Nummer n noch nicht, so wird es durch **xset** geöffnet. Existiert es bereits, so wird lediglich zu Fenster n gewechselt. Dieses Wechseln ist wichtig, da **scilab** alle Plots in das Graphikfenster macht, das gerade ausgewählt wurde, ohne den Fensterinhalt zu löschen.

Für die graphische Ausgabe von Graphen steht der Befehl **plot2d** zur Verfügung. Der Vollständige Aufruf lautet

plot2d(x,y,style=[s])

Hierbei befinden sich x-Koordinaten der zu plottenden Punkte in der Variable x und die y-Koordinaten in der Variable y. Für x und y lassen sich drei Fälle unterscheiden.

- x und y sind Vektoren: Durch plot2d wird ein Graph gezeichnet.
- x ist ein Vektor und y eine Matrix: Durch plot2d werden pro Spalte der Matrix y ein Graph gezeichnet, wobei die x-Koordinaten aus dem Vektor x und die y-Koordinaten aus einer Spalte von y genommen werden. Es ist zu beachten, dass die Anzahl der Spalten von y der Länge von x entspricht.
- x und y sind Matrizen: Durch plot2d wird pro Spalte von y ein Graph gezeichnet. Dabei werden die x-Koordinaten aus einer Spalte von x und die y-Koordinaten aus der gleichen Spalte von y genommen. Es ist zu beachten, dass x und y Matrizen von gleicher Dimension sind, d.h. gleiche Anzahl von Zeilen und Spalten haben.

Die Option *style*, die auch weggelassen werden kann, gibt vor, wie die Graphen geplottet werden. s ist ein Vektor dessen Länge der Anzahl der Spalten von y entspricht. Dabei definiert das i-te Element von s den Graph für die i-te Spalte von y. Ist $s(i) > 0$, so werden die Punkte (x,y) durch eine Linie mit der Farbe $s(i)$ verbunden. Ist hingegen $s(i) < 0$, so werden lediglich die Punkte (x,y) gezeichnet und mit Kreuzen, Kreisen, Dreiecken, ... (abhängig von $s(i)$) markiert. Folgendes Beispiel zeichnet zwei Linien, wobei eine Gerade mit einer durchgezogenen Linie gezeichnet wird, während bei der anderen nur Punkte gesetzt werden.

```
-->t=1:10;

-->t=t';

-->x=5*t-3;

-->y=-4*t+6;

-->plot2d(t,[x y],style=[3 -3])

-->legends(['style 3','style -3'],[3 -3],2)
```

Das resultierende Bild sieht dann so aus:

Zusätzlich wurde hier der Befehl *legends* verwendet. Mit diesem Befehl kann man das gezeichnete Bild mit einer Legende versehen. Der Befehl is folgendermaßen strukturiert:

> legends(['Bezeichnung für das erste element',...,'Bezeichnung für das letzte Element'], [Farbe des ersten Elements ... Farbe des letzten Elements],position)

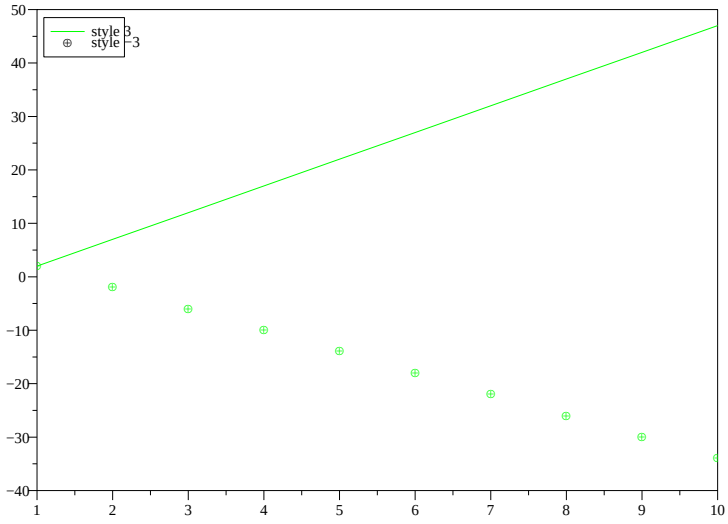

Abbildung 1: Plot von zwei Linien mit verschiedenen Stielen.

Die Farben der einzelnen Legendeneinträge entsprechen denen des *plot2d* befehls, d.h. positive Zahl = Farbe, negative Zahl = Markierung. Die Position der Legende wird mit dem Parameter *position* angegeben, dabei gilt

- 1 : rechts oben
- 2 : links oben
- 3 : links unten
- 4 : rechts unten
- 5 : Freies setzen mit der Maus

Neben der Ausgabe von einem Bild pro Graphikfenster ist es auch möglich mehrere Bilder in ein Graphikfenster zu zeichnen. Mit dem Befehl

```
subplot(i,j,n)
```

wird das Graphikfenster in $i * j$ Teilfenster unterteilt, wobei die Teilfenster in i Zeilen zu jeweils j Teilfenstern angeordnet werden. Mit n wird das Teilfenster ausgewählt, in das gezeichnet werden soll. Das Plotten erfolgt dabei wie gehabt mit **plot2d**. Bild 2 zeigt das Zeichnen von zwei Linien in jeweils ein eigenes Teilfenster, das mit fogenden Befehlen erzeugt wird:

```
-->t=1:10;

-->x=5*t-3;

-->y=-4*t+6;

-->subplot(1,2,1)

-->plot2d(t,x)

-->subplot(1,2,2)

-->plot2d(t,y)
```

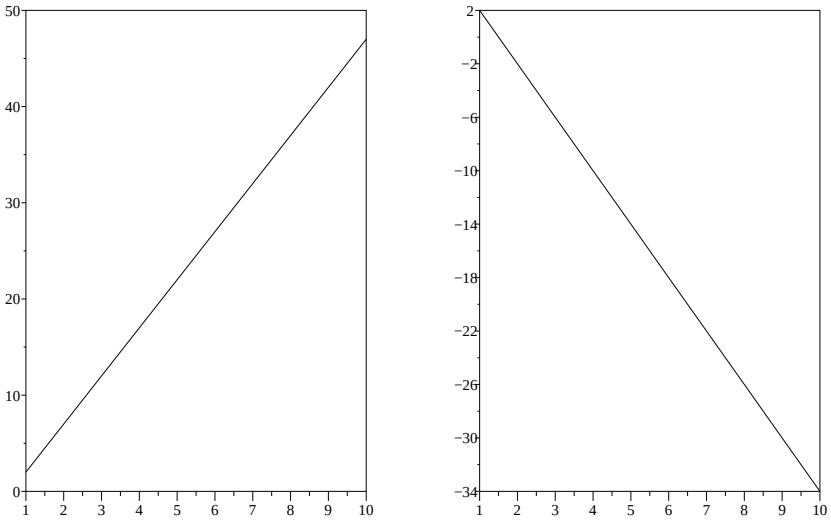

Abbildung 2: Plot von zwei Linien in jeweils ein Teilfenster.

Schließlich kann man jedem Plot auch noch einen Titel geben und die Achsen beschriften. Dies erfolgt mit

xtitle('Titel','x-Achsenbeschriftung','y-Achsenbeschriftung')

Kombiniert mit **subplot** kann somit auch jedem Teilfenster eine eigene Beschriftung gegeben werden wie folgendes Beispiel demonstriert (siehe Bild 3):

```
-->t=1:10;

-->x=5*t-3;
```

```
-->y=-4*t+6;

-->subplot(1,2,1)

-->plot2d(t,x)

-->xtitle('Linker Plot')

-->subplot(1,2,2)

-->plot2d(t,y)

-->xtitle('Rechter Plot')
```

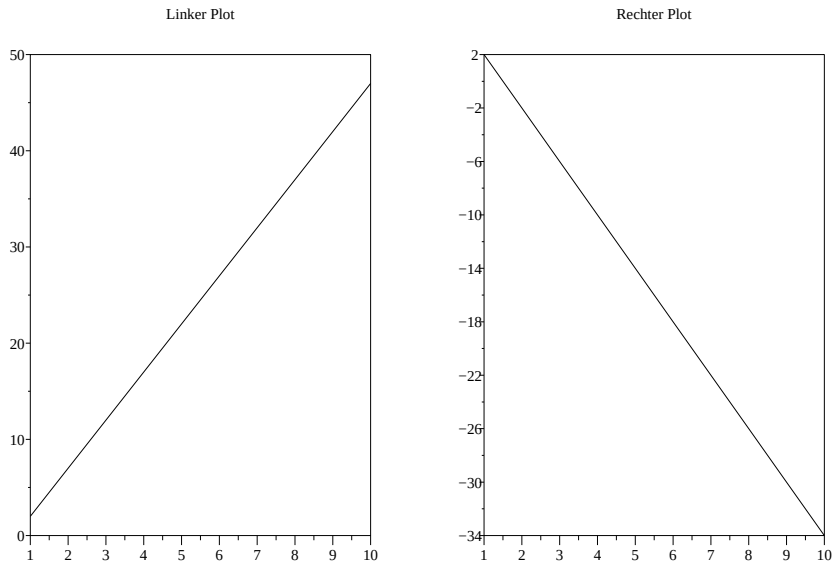

Abbildung 3: Plot von zwei Linien in jeweils ein Teilfenstern mit Überschriften.