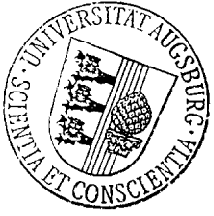


S-Plus:

INHALT:

- **Einleitung**
 - Historie
 - Einordnung und Abgrenzung zu anderen Paketen
 - Tutorial
- **Datentypen und Datenmanipulation**
 - Übersicht
 - Hierarchie von Datentypen und Operatoren
 - Konstruktoren und Selektoren der Datentypen
 - Mathematische und statistische Funktionen
- **Programmierung**
 - Kontrollstrukturen
 - Definition von Funktionen
 - Rekursion, Iteration
 - Objektorientierte Programmierung
- **Graphik**
 - Low- und Highlevel Plots
 - Visualisierung von 1- und mehrdimensionalen Daten
 - Untersuchung und Vergleich von Verteilungen
- **Statistische Modelle**



HISTORIE:

1976-1980:

Version 1

Honeywell Mainframe

Fortran Subroutinen

Entwicklung bei AT&T:

- Becker, Chambers, Wilks
- Chambers, Hastie
- Cleveland

1980-1984:

Version 2

Portierung auf UNIX

Interface Language

1988-1991:

Version 3

UNIX: Umstellung auf C

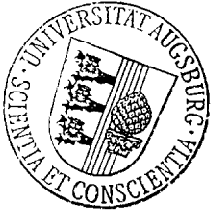
Eigene S-Funktionen & Objekte

1991-heute:

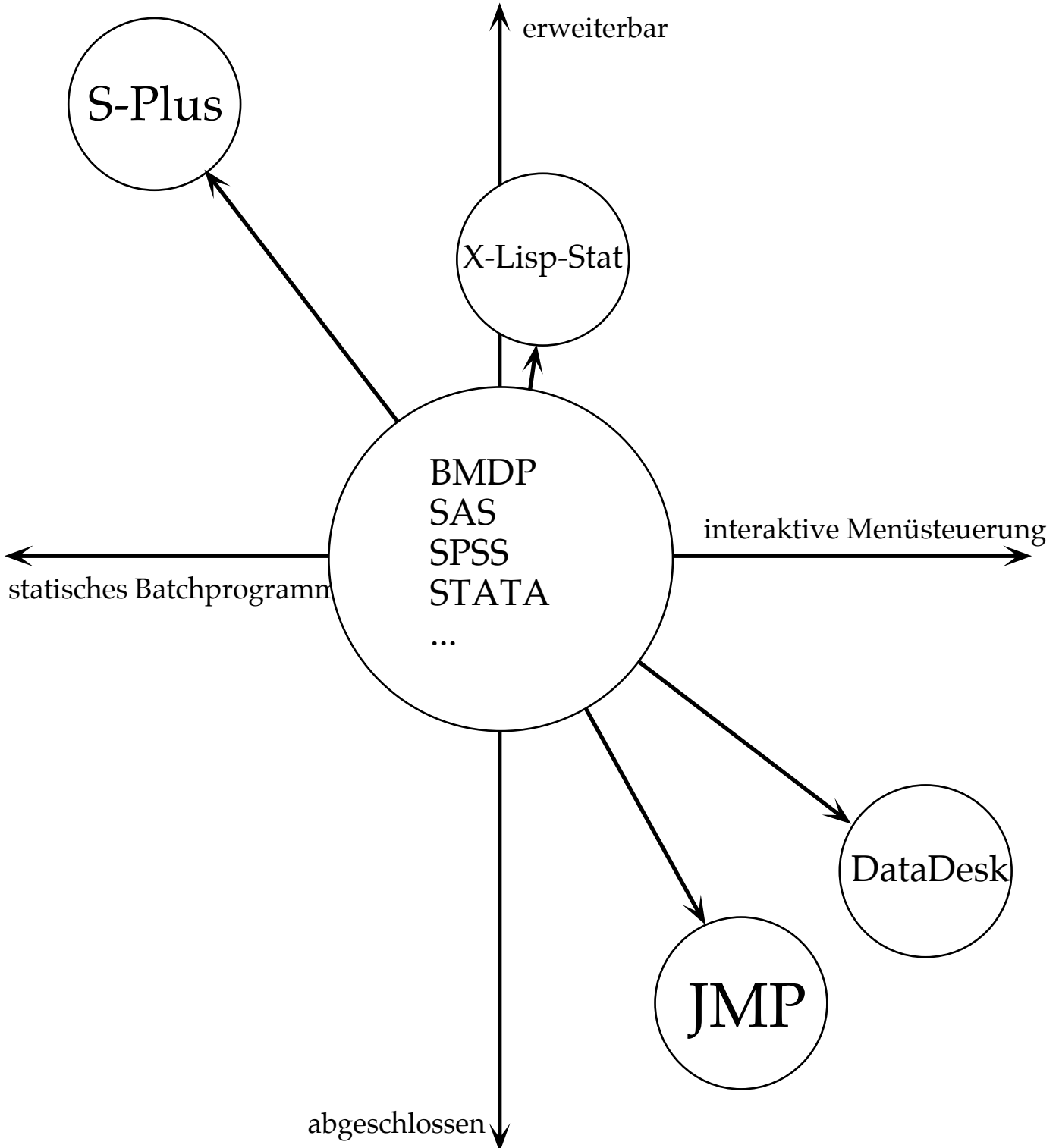
Version 3.2

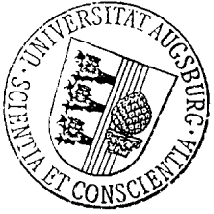
MS-Windows Version

Objektorientierung



EINORDNUNG UND ABGRENZUNG:





LITERATUR:

The NEW S Language, a Programming Env. for Data Anal. and Graphics

Becker R., Chamber J., Wilks A., Wadsworth & Brooks, 1988

Statistical Models in S

Chamber J., Hastie T., Wadsworth & Brooks, 1992

Numerical Reliability of Data Analysis Systems

Sawitzki, G., Computational Statistics & Data Analysis 1994

Classes and Methods in S, I: Recent Developments, II: Future Directions

Chambers J., Computational Statistics, 1993, 8: 167--196

S-Plus für WINDOWS, User's Manual Vol. 1 & 2

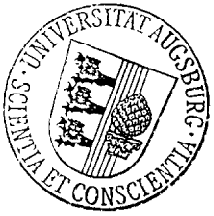
Statistical Sciences Inc.

S-Plus Statistical Analysis in S-Plus

Statistical Sciences Inc.

S und S-Plus, Eine Einführung in Programmierung und Anwendung

Süselbeck, B., Gustav Fischer Verlag, 1993



TUTORIAL:

Starten von S-Plus:

Unix: `golf% Splus`
 `S-PLUS : Copyright (c) 1988,1993 Statistical Sciences, Inc.`
 `S : Copyright AT&T.`
 `Version 3.2 Release 1 for Sun SPARC, SunOS 4.x : 1993`
 `Working data will be in .Data`
 `>`
 `>...`

Windows: Doppelklick auf S-Plus Icon

```
S-PLUS : Copyright (c) 1988,1994 MathSoft, Inc.  
S : Copyright AT&T.  
Version 3.2 Release 1 for MS Windows 3.1 : 1994  
Working data will be in .Data  
>  
>...
```

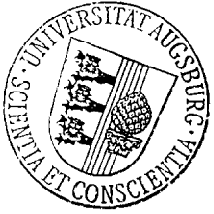
Verlassen von S-Plus:

```
Immer:     >  
           >...  
           > q()
```

Einfache Ausdrücke:

```
> 4 + 5  
[1] 9  
> 7 * 2  
[1] 14  
> sin(pi)  
[1] 1.224606e-16  
> Hallo  
Error: Object "Hallo" not found  
>
```

Benutzung wie einen Taschenrechner



TUTORIAL:

Hilfesystem:

```
> help(hist)
Plot a Histogram
```

DESCRIPTION:

Creates a histogram on the current graphics device.
Several options are available.

USAGE:

```
hist(x, nclass=<<see below>>, breaks=<<see below>>,
     plot=T, probability=F, ...)
```

REQUIRED ARGUMENTS:

x: numeric vector of data for histogram. Missing values
(NAs) are allowed.

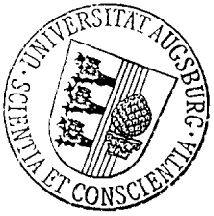
OPTIONAL ARGUMENTS:

nclass: recommendation for the...

Unter Windows: Aufruf des üblichen Windows-Hilfesystems.

Ein Beispiel:

```
> Wahl <- scan( what = 0 ) <- Einlesen der Daten
1: 42
2: 36
3: 7
4: 6
5: 4
6: 5
7: <- Leere Eingabe
> Wahl <- unformatierte Ausgabe
[1] 42 36 7 6 4 5
> eines Objektes
```



TUTORIAL:

Beispiel ... :

```
> length( Wahl )
[1] 6
> mean( Wahl )
[1] 16.66667
> min( Wahl )
[1] 4
> max( Wahl )
[1] 42
> median( Wahl )
[1] 6.5
> summary( Wahl )
Min. 1st Qu. Median Mean 3rd Qu. Max.
  4      5.25     6.5 16.67  28.75   42
```

Berechnung von:

<- Länge
Mittelwert
Minimum
Maximum
Median

...

Zusammenfassung.

```
>
> Wahl <- c(42,36,7,6,4,5)
> Wahl
[1] 42 36  7  6  4  5
```

<- alternative Zuweisung

...

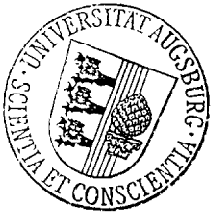
und Namensinformation

```
>
> names(Wahl) <- c("Union", "SPD", "Gruene", "FDP", "PDS", "Sonstige")
> Wahl
  Union SPD Gruene FDP PDS Sonstige
    42  36     7   6   4     5
```

```
>
> Wahl90 <- c(43.8, 33.5, 3.8, 11.0, 2.4, 5.5)
Wahl90
[1] 43.8 33.5  3.8 11.0  2.4  5.5
```

```
>
> names( Wahl90 ) <- names( Wahl) <- Kopieren der Namen
> names( Wahl90 )
[1] "Union"    "SPD"      "Gruene"   "FDP"      "PDS"      "Sonstige"
```

```
> Wahl90
  Union SPD Gruene FDP PDS Sonstige
 43.8 33.5  3.8  11  2.4     5.5
```

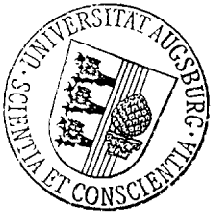
Datentypen und Datenmanipulation:

ÜBERSICHT:

- Elementare Datentypen:
 - Skalare \Leftrightarrow Vektoren
 - Listen
 - Strukturierte Datentypen:
 - Matrizen
 - Arrays
 - Zeitreihen
 - Kategorien
 - Attribute strukturieren Datentypen:
 - mode
 - + null
 - + logical
 - + numeric
 - + complex
 - + character

} atomar
 - + list
 - + function
 - + graphics

} zusammengesetzt
 - length
 - names
 - dimnames
 - dim
 - ... (da erweiterbar)
- } strukturierende Attribute
- } haben alle Objekte



DATENTYPEN:

Erfragen aller Attribute:

```
> names( attributes( Wahl ) )  
[1] "names"
```

Erfragen der Ausprägungen der Attributen:

```
> attr(Wahl, "length")  
[1] 6
```

```
> attr(Wahl, "mode")  
[1] "numeric"
```

Ändern der Ausprägungen von Attributen:

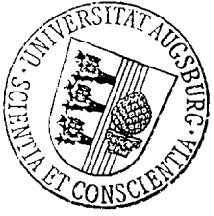
```
> attr( Wahl, "mode") <- "character"  
(oder: mode( Wahl) <- "character")
```

```
> Wahl  
Union  SPD  Gruene  FDP  PDS  Sonstige  
"42"   "36"  "7"      "6"  "4"  "5"
```

```
> length( Wahl ) <- 8  
> Wahl  
Union  SPD  Gruene  FDP  PDS  Sonstige  
42    36      7     6   4      5 NA NA
```

Typstest, Typkonversion und Typkonstruktion:

```
> is.mode( x )  
> as.mode( x )  
> mode( x )
```



DATENTYPEN:

Spezielle Typen:

- Fehlende Werte:

```
> array(dim=c(2,2))
      [,1] [,2]
[1,]   NA   NA
[2,]   NA   NA

> is.na(1)
[1] F

> is.na(NA)
[1] T
```

- undefinierte Werte:

```
> is.nan(0/0)
[1] T

> is.nan(NA)
[1] F
```

- Unendliche Werte:

```
> is.finite(1)
[1] T

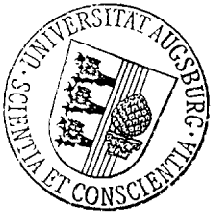
> is.infinite(1/0)
[1] T
```

- Konstanten:

```
> pi
[1] 3.1415926535897931

> 1 == 1
[1] T

> 1 == 1.000001
[1] F
```



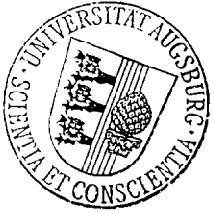
OPERATOREN:

Hierarchie:

Operator	Name	Wertigkeit
\$	Element, Auswahl	HOCH ↑ ↓ TIEF
[][Indizierung, Element	
^	Potenzierung	
-	Vorzeichen	
:	Sequenzoperator	
%name%	spezieller Operator	
* /	Multiplikation, Division	
+ -	Addi-, Subtraktion	
< > <= >= == !=	Vergleich	
!	Negation	
& &&	log., bin. Und und Oder	
<- _ ->	Zuweisung	

Spezielle Operatoren:

- Operatoren sind generisch, d.h. sie können auf jeden Datentyp verallgemeinert werden. (siehe OOP)
- Mit %name% können neue Operatoren für eigene Datentypen definiert werden.



KONSTRUKTOREN & SELEKTOREN:

Vektoren:

```
> 1:5
[1] 1 2 3 4 5

> 1.234:5.678
[1] 1.234 2.234 3.234 4.234 5.234

> seq( from=2, by=0.25, to=4)
[1] 2.00 2.25 2.50 2.75 3.00 3.25 3.50 3.75 4.00

> x <- c(16,3,1967)
> names(x) <- c("Tag", "Monat", "Jahr")
> x
  Tag Monat  Jahr
   16     3 1967

> rep(x,2)
  Tag Monat  Jahr Tag Monat  Jahr
   16     3 1967  16     3 1967

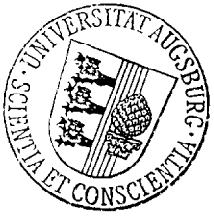
> x[1]
  Tag
   16

> x[2:3]
  Monat  Jahr
     3 1967

> x[-2]
  Tag  Jahr
   16 1967

> x[x<20]
  Tag  Monat
   16     3

> x[names(x) == "Tag"]
  Tag
   16
```



KONSTRUKTOREN & SELEKTOREN:

Listen:

```
> as.list( c(1,2,3) )
[[1]]:
[1] 1

[[2]]:
[1] 2

[[3]]:
[1] 3

> x <- list( list( 1, 2 ), 3, c( 4, 5 ), "sechs" )
> names(x) <- c( "Liste", "Zahl", "Vektor", "String" )

> x[[3]]
[1] 4 5

> x[[4]]
[1] "sechs"

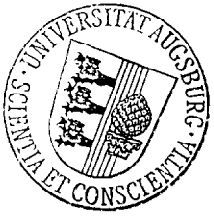
> x[3]
$Vektor:
[1] 4 5

> is.list(x[3])
[1] T

> is.list(x[[3]])
[1] F

> is.vector(x[[3]])
[1] T

> x$Vektor
[1] 4 5
```



KONSTRUKTOREN & SELEKTOREN:

Matrizen:

```
> matrix( 1, 3, 3 )
      [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    1
[3,]    1    1    1

> matrix( c( 1, 2, 3) , 3, 3)
      [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    2    2
[3,]    3    3    3

> matrix( c( 1, 2, 3), 3, 3, byrow=T)
      [,1] [,2] [,3]
[1,]    1    2    3
[2,]    1    2    3
[3,]    1    2    3

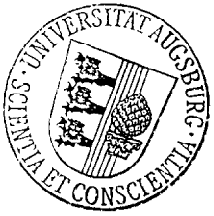
> diag( c(1, 2, 3), 3, 3)
      [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    2    0
[3,]    0    0    3

> x <- c( 1, 2, 3)

> cbind( x , x, x)
      x x x
[1,] 1 1 1
[2,] 2 2 2
[3,] 3 3 3

> rbind( x, x, x)
      [,1] [,2] [,3]
x      1    2    3
x      1    2    3
x      1    2    3

> is.matrix(rbind( x, x, x))
[1] T
```



KONSTRUKTOREN & SELEKTOREN:

Matrizen:

```
> x <- diag( c(1, 2, 3), 3, 3)
> nrow(x)
[1] 3

> ncol(x)
[1] 3

> dim(x)
[1] 3 3

> x[1]
[1] 1

> x[2]
[1] 0

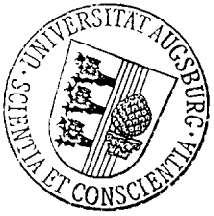
> x[5]
[1] 2

> x[2,2]
[1] 2

> x[2,]
[1] 0 2 0

> x[,3]
[1] 0 0 3

> x[ matrix( rep(1:3, rep(2,3) ), 3, 2, byrow=T) ]
[1] 1 2 3
```



KONSTRUKTOREN & SELEKTOREN:

Felder, Arrays:

- Verallgemeinerung von Matrizen
=> Behandlung analog (Dimensionalität > 2)

Zeitreihen:

```
> ts( 100:45, start=c(1990,3), frequency=12, end=c(1994,10))
      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1990:      100  99  98  97  96  95  94  93  92  91
1991:   90  89  88  87  86  85  84  83  82  81  80  79
1992:   78  77  76  75  74  73  72  71  70  69  68  67
1993:   66  65  64  63  62  61  60  59  58  57  56  55
1994:   54  53  52  51  50  49  48  47  46  45
```

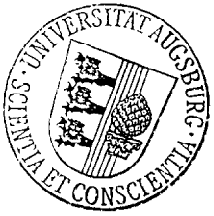
Kategorien, Faktoren:

```
> category( c("eins", "eins", "drei", "zwei", "eins", "drei") )
[1] 2 2 1 3 2 1
attr(,"levels"):
[1] "drei" "eins" "zwei"
> cut(1:10,2)
[1] 1 1 1 1 1 2 2 2 2 2
attr(,"levels"):
[1] "0.91+ thru 5.50" "5.50+ thru 10.09"
```

Data Frames (Datensätze):

```
> x <- data.frame( Alter=c(44,48,39,55),
+ Geschlecht = c("m", "m", "w", "w"), Raucher=c(F,T,T,F),
+ row.names = c("Meier", "Maier", "Mayer", "Mayr") )
> x
      Alter Geschlecht Raucher
Meier    44          m  FALSE
Maier    48          m   TRUE
Mayer    39          w   TRUE
Mayr     55          w  FALSE

> x$Alter
[1] 44 48 39 55
```



OPERATOREN:

Vektoren:

```
> x<-10:1
> any(x>6)
[1] T

> all(x>6)
[1] F

> sum(x)
[1] 55

> cumsum(x)
[1] 10 19 27 34 40 45 49 52 54 55

> diff(x)
[1] -1 -1 -1 -1 -1 -1 -1 -1 -1

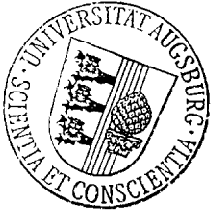
> sort(x)
[1] 1 2 3 4 5 6 7 8 9 10

> order(x)
[1] 10 9 8 7 6 5 4 3 2 1

> x[order(x)]
[1] 1 2 3 4 5 6 7 8 9 10

> rev(x)
[1] 1 2 3 4 5 6 7 8 9 10

> match(x,x+9)
[1] 10 NA NA NA NA NA NA NA NA NA
```



OPERATOREN:

Listen:

```
> x <- list( c(1, 2), 3, c(4, 5, 6, 7))
> lapply( x, mean)
[[1]]:
[1] 1.5

[[2]]:
[1] 3

[[3]]:
[1] 5.5

> sapply(x, mean)
[1] 1.5 3.0 5.5

> unlist(x)
[1] 1 2 3 4 5 6 7

> append(x,8)
[[1]]:
[1] 1 2

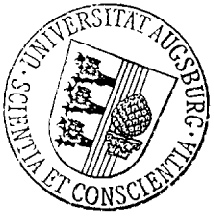
[[2]]:
[1] 3

[[3]]:
[1] 4 5 6 7

[[4]]:
[1] 8

> y <- append(x, 9)
> z <- replace(y, 4, 8)
> z[[4]]
[1] 8
```

append und replace gelten auch für Vektoren !



OPERATOREN:

Matrizen:

```
> x <- matrix( c(1,0,1,1), 2, 2 )
> x
      [,1] [,2]
[1,]    1    1
[2,]    0    1

> t(x)
      [,1] [,2]
[1,]    1    0
[2,]    1    1

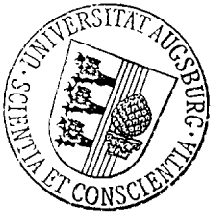
> solve(x)
      [,1] [,2]
[1,]    1   -1
[2,]    0    1

> x %*% solve(x)
      [,1] [,2]
[1,]    1    0
[2,]    0    1

> eigen(x)
$values:
[1] 1 1

$vectors:
      [,1] [,2]
[1,] -1e+00  1
[2,]  3e-18  0

> scale(x)
      [,1] [,2]
[1,] 0.7071068 NA
[2,] -0.7071068 NA
```



OPERATOREN:

Felder, Arrays:

```
> x <- array( c(0,1,1,1,2,3,4,5), dim=c(2,2,2) )  
> x
```

```
, , 1  
  [,1] [,2]  
[1,]  0   1  
[2,]  1   1
```

```
, , 2  
  [,1] [,2]  
[1,]  2   4  
[2,]  3   5
```

```
> apply( x, 1, sum)  
[1]  7 10
```

```
> sweep( x, 1, apply(x, 1, mean), "-")
```

```
, , 1  
  [,1] [,2]  
[1,] -1.75 -0.75  
[2,] -1.50 -1.50
```

```
, , 2  
  [,1] [,2]  
[1,] 0.25 2.25  
[2,] 0.50 2.50
```

```
> aperm(x, c(3,2,1) )
```

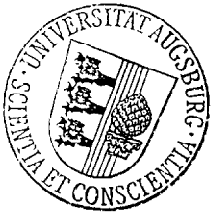
```
, , 1  
  [,1] [,2]  
[1,]  0   1  
[2,]  2   4
```

```
, , 2  
  [,1] [,2]  
[1,]  1   1  
[2,]  3   5
```

apply, sweep und aperm

können auch auf Matrizen
angewendet werden.

(aperm \Leftrightarrow t)



OPERATOREN:

Kategorien:

```
> Pet <- factor(c("Cat", "Dog", "Cat", "Dog", "Cat", "Cat"))
> Food <- factor(c("Dry", "Dry", "Dry", "Wet", "Wet", "Wet"))
> table(Pet, Food)
      Dry Wet
Cat   2   2
Dog   1   1
```

Zeitreihen:

```
> x <- ts( 1:24, start=c(1993,1), frequency=12,
+end=c(1994,12) )

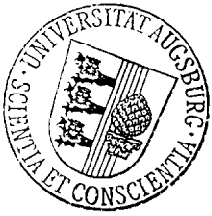
> x
      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1993:   1   2   3   4   5   6   7   8   9  10  11  12
1994:  13  14  15  16  17  18  19  20  21  22  23  24

> lag( x, 12)
      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1992:   1   2   3   4   5   6   7   8   9  10  11  12
1993:  13  14  15  16  17  18  19  20  21  22  23  24

> x+lag(x, 12)
      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1993:  14  16  18  20  22  24  26  28  30  32  34  36

> fft( 1:8 )
[1] 36+0.000i -4+9.656i -4+4.000i -4+1.656i -4+0.000i
[6] -4-1.656i -4-4.000i -4-9.656i

> fft( fft( 1:8 ),inverse=T ) / 8
[1] 1+0.0000e+00i 2-4.4408e-16i 3+4.4408e-16i 4+4.4408e-16i
[5] 5+0.0000e+00i 6-4.9303e-32i 7-4.4408e-16i 8+9.8607e-32i
```



MATHEMATISCHE FUNKTIONEN:

- elementweise Funktionen:

- + Grundrechenarten: $+$, $-$, $*$, $/$, $^$
- + unäre Funktionen: \sin , \cos , \tan
 asin , acos , atan
 \sinh , \cosh , \tanh
 asinh , acosh , atanh
 \exp , \log , \log_{10}
 $\sqrt{\quad}$, abs , Γ
 int , floor , $\operatorname{ceiling}$

- Vektorfunktionen:

- + `vecnorm`

- Matrixfunktionen:

- + `kroneker` Kronecker-Produkt
- + `solve(mat, vec)` Lösung eines LGS
- + `qr` QR-Zerlegung
- + `svd` Singulärwertzerlegung
- + `eigen` Eigenwerte, -vektoren

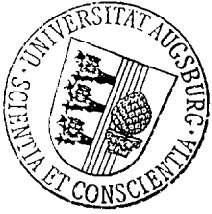
- Integration, Differentiation:

- + `integrate`

```
> integrate( sin, 0, pi)
$integral:
[1] 2 ...
```

- + `D`

```
> D( expression( 3*x^2 ), "x")
3 * (2 * x)
```



MATHEMATISCHE FUNKTIONEN:

- Optimierung:

- + Wurzelberechnung:

```
> polyroot( c(-2,0,1) )  
[1] 1.414214+0i -1.414214+0i
```

- + lokale Extrema:

```
> peaks( c(1,1,1,1,3,1,1,1) )  
[1] F F F F T F F F
```

- + lokale Extrema einer stetigen Funktion:

```
> optimize(sin, interval = c(0, pi), maximum = T)  
$maximum:  
[1] 1.570799 ...
```

- + lokale Extrema einer Funktion mehrerer Variablen:

nlminb	nichtlineare Minimierung
ms	Summenminimierung

- + nichtlineare kleinste Quadrate Probleme:

nls, nlregb

- + Rucksackproblem:

napsack

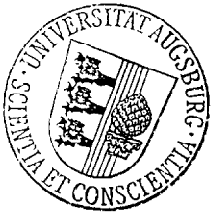
- Interpolation:

- + Lineare Interpolation, kubische Splineinterpolation

approx	eindimensionale Interpolation
--------	-------------------------------

interp	zweidimensionale Interpolation
--------	--------------------------------

spline	kubische Splineinterpolation
--------	------------------------------



STATISTISCHE FUNKTIONEN:

- Statistiken:

+ mean, median, var,
min, max, sort,
quantile, summary

- Verteilungsfunktionen:

+ Zufallszahlen: r Verteilung

+ Verteilungsfunktion: p Verteilung

+ Dichtefunktion: d Verteilung

+ Quantilsfunktion: q Verteilung

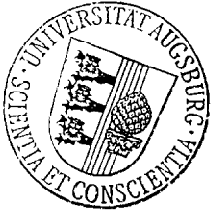
Verteilungen:

beta, binom, cauchy, chisq,
exp, f, gamma, geom,
hyper, lnorm, logis, nbinom,
norm, pois, stab, t,
unif, weibull, wilcox.

Beispiel:

```
> runif(10,0,1)
[1] 0.32991 0.25600 0.36541 0.61216 0.71185 0.02275
[7] 0.45730 0.80087 0.40367 0.68491

> mean( runif (10000,0,1) )
[1] 0.5027383
```



Programmierung:

KONTROLLSTRUKTUREN:

- Reihung:

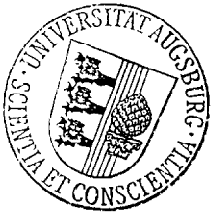
```
> x <- { t <- 4 * atan(1)
+ s <- paste( "Nicht vergessen: Pi = ", t)
+ s
+ }
> x
[1] "Nicht vergessen: Pi = 3.14159265358979"
```

- Verzweigung:

```
> if( alpha != 0 )
+ {
+   ( x^alpha - 1 ) / alpha
+ }
+ else
+ {
+   log( x )
+ }
```

- Mehrfachverzweigung:

```
> x <- pi
> switch( mode(x),
+ numeric    = print("Numerische Variable"),
+ character  = print("String"),
+ print("Weder numerisch noch alphanumrisch")
+ )
[1] "Numerische Variable"
```



WIEDERHOLUNGEN:

- Implizit:

```
> a <- 0
> for ( i in 1:10 )
+ {
+     a <- a + i
+ }
> a
[1] 55

> ( 10 * ( 10 + 1 ) ) / 2
[1] 55
```

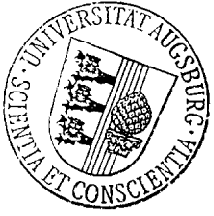
- Explizit:

- + while

```
> f <- cos
> unten <- 0
> oben <- pi
> while( (oben - unten) > 0.01 )
+ {
+   if( f( (oben + unten) / 2 ) > 0 )
+     unten <- (oben + unten) / 2
+   else
+     oben <- (oben + unten) / 2
+ }
> oben * 2
[1] 3.141593
```

- + repeat

```
> repeat
...
> until( (oben - unten) < 0.01 )
```



FUNKTIONSDEFINITION:

- Deklaration:

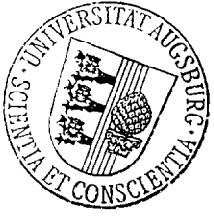
```
> x <- function()  
+ pi  
  
> x  
function()  
pi  
  
> x()  
[1] 3.141593
```

- Parameterübergabe, -rückgabe und Standardparameter:

```
> twice <- function( x = 1 )  
+ {  
+   return( 2 * x )  
+ }  
  
> twice()  
[1] 2  
  
> twice(3)  
[1] 6
```

- Unbestimmte Anzahl von Parametern:

```
> Summe <- function(...)  
+ {  
+   e <- 0  
+   for( x in list(...) )  
+     e <- e + x  
+ }  
> Summe(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  
[1] 55
```



FUNKTIONSDEFINITION:

- Parametersubstitution:

```
> f <- function( eins=1, zwei=2, drei=3, vier=4)
+ {
+ print(eins)
+ print(zwei)
+ print(drei)
+ print(vier)
+ invisible(NULL)
+ }

> f(vier=1, dr=2,, 3)
[1] 1
[1] 3
[1] 2
[1] 1
```

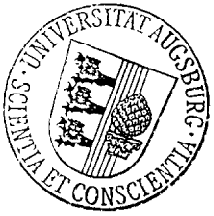
Regeln:

1. Ersetze alle Parameter, deren Name exakt übereinstimmt
2. Ersetze dann alle Präfixe, die übereinstimmen
3. Alle unbenannten Parameter werden ihrer Reihenfolge nach ersetzt

- Lokale Variablen:

```
> f <- function()
+ {
+ a <- 10
+ print(a)
+ invisible(a)
+ }
> a <- 100
> f()
[1] 10

> a
[1] 100
```



ITERATION VS. REKURSION:

- Iteration:

```
> fak <- function( n )  
+ {  
+   p <- 1  
+   for ( i in 1:n )  
+     p <- p * i  
+   return(p)  
+ }  
  
> fak(5)  
[1] 120
```

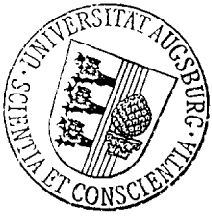
- Rekursion:

```
> fak <- function( n )  
+ {  
+   if( n == 0 )  
+     1  
+   else  
+     n * fak( n - 1 )  
+ }  
  
> fak(5)  
[1] 120
```

- Unterschiede:

- | | |
|----------------------|-----------------|
| + Rekursion ist | + Iteration ist |
| - langsamer | - schneller |
| - speicherintensiver | - "billiger" |
| - eleganter | - "eckiger" |

↔
Entscheidung je nach Problem



OBJEKTORIENTIERTE PROGRAMMIERUNG (OOP):

Vorgehensweise:

- Problemstellung

Darstellung von komplexen Zahlen in Polarkoordinaten.

- Definition der Datenstruktur

```
> p <- list( Winkel = 60, Radius = 10)
> p
$Winkel:
[1] 60

$Radius:
[1] 10
```

- Definition der Klasse

```
> class(p) <- "polar"
> attr(p, "class")
[1] "polar"
```

- Definition eines Konstruktors

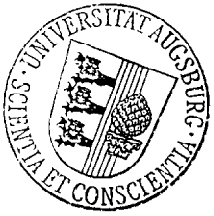
```
> polar <- function( x, y) {
+ p <- list( Winkel = x, Radius = y)
+ class(p) <- "polar"
+ p }
```

- Definition von Methoden

```
> ".*.polar" <- function( p1, p2 )
+ polar( p1$Winkel + p2$Winkel, p1$Radius * p2$Radius )
> a <- polar(10, 10)
> b <- polar(10, 10)
> a * b
$Winkel:
[1] 20

$Radius:
[1] 100

attr(, "class"):
[1] "polar"
```



DEBUGGING:

- **Tracer:**

```
> trace(fak)
> fak(4)
On entry:          fak(4)
On entry:          fak(n - 1)
On entry:          fak(n - 1)
On entry:          fak(n - 1)
On entry:          fak(n - 1)
[1] 24
> untrace(fak)
```

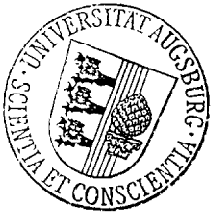
- **Browser:**

```
> fak <- function( n ) {
+ browser()
+ if ( n == 0 )
+ }
> fak(4)
Called from: fak(4)
b(2)> ?
1: n
b(2)> n
[1] 4
b(2)> 0
...
b(6)> 0
[1] 24
```

- **Traceback:**

```
...      n * fac( n - 1 )
...
> fak(4)
Error in fak(4): couldn't find function "fac"
Dumped

> traceback()
Message: couldn't find function "fac"
2: fak(4)
1:
```



Graphik:

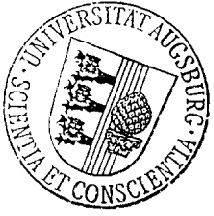
LOWLEVEL PLOTS:

Lowlevel Plots zeichnen auf ein bereits existierendes Koordinatensystem, z.B:

- `points(x, y)`
- `lines(x, y)`
- `polygon(x, y)`
- `segments(x1, y1, x2, y2)`
- `arrows(x1, y1, x2, y2)`
- `text(x, y, label)`
- `abline(a, b)` $(y = a + b * x)$
- `locator(n)`
- `identify(x, y, labels)`

Das Kommando `frame()` initialisiert das aktuelle Plotting-Device.

Highlevel Plots können oft mit der Option `add=T` zu einer Lowlevelfunktionalität umgewandelt werden.



HIGHLEVEL PLOTS:

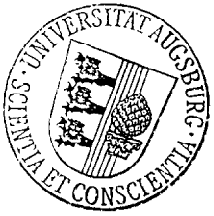
Highlevel Plots erzeugen ein neues Koordinatensystem.
Sie sind aus lowlevel Funktionen zusammengesetzt.

z.B:

- `plot(x [, y])`
- `tsplot(x)`
- `image(x, y, z)`
- `contour(x, y, z)`
- `barplot(x)`
- `dotchart(x)`
- `pie(x)`
- `boxplot(x)`
- `hist(x)`

Das Kommando `legend(...)` ermöglicht das Hinzufügen einer Legende.

Alle Plotkommandos können mit zahlreichen Parametern modifiziert werden. Mit dem Befehl `par(...)` werden Parameter global für graphische Ausgaben gesetzt.



GRAPHISCHE PARAMETER:

- **Plottyp:**

type = "l" Linien
"p" Punkte
"b" Linien und Punkte (ausgespart)
"o" Linien und Punkte (überlagert)
"h" vertikale Linien
"s" Treppenstufen
(Punkte werden in City-Block-Metrik verbunden)
"n" Ausgabe wird unterdrückt

- **Punkttyp:**

pch = 1:18 vordefinierte Typen
"char" benutzereigener Buchstabe "char"
cex = n Größe der verwendeten Zeichen

- **Linien:**

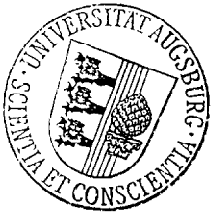
lty = 1:n vordefinierte Typen verschiedener Straffur
lwd = n Definition der Linienbreite

- **Farbe:**

col = 1:n vordefinierte Farbstufen, je nach gewählter
Palette
"name" vordefinierte Farbe mit Namen "name"
#RRGGBB Spezifikation der Farbe durch je zwei hexa-
dezimale Stellen für: rot, grün und blau

- **Seitenlayout:**

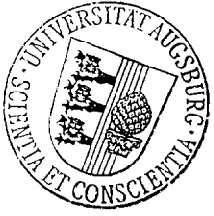
mai = c(unten, links, oben, rechts)
Rand des Plots
mfrow = c(z, s) Aufteilung der Ausgabeseite in z Zeilen
und s Spalten, d.h. z * s Plots!



GRAPHISCHE PARAMETER:

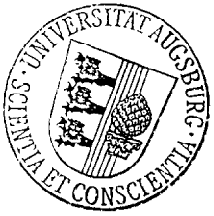
- Achsen:

<code>axes =</code>	<code>T F</code>	Zeichnen der Achsen ein- bzw. ausschalten
<code>xaxs =</code>	<code>"r"</code>	Datenbereich plus 4%
	<code>"i"</code>	Datenbereich ohne Erweiterung
	<code>"e"</code>	Erweiterung der Achsen auf ganze Achsen- teilung
	<code>"s"</code>	Tickmarks an Anfang und Ende, jedoch nur minimal nötige Erweiterung
	<code>"d"</code>	Achsen des letzten Plots werden beibehalten
	<code>" "</code>	Zurücksetzen der Standardeinstellung
<code>yaxs =</code>	<code>...</code>	Analog
<code>log =</code>	<code>"y"</code>	logarithmische Skalierung der y-Achse
	<code>"x"</code>	logarithmische Skalierung der x-Achse
	<code>"xy"</code>	logarithmische Skalierung beider Achsen
<code>xlim =</code>	<code>c(xmin, xmax)</code>	Skalierung der x-Achse
<code>ylim =</code>	<code>...</code>	analog
<code>xlab =</code>	<code>"title"</code>	Beschriftung der x-Achse
<code>ylab =</code>	<code>"title"</code>	Beschriftung der y-Achse
<code>main =</code>	<code>"title"</code>	Überschrift
<code>tck =</code>	<code>val</code>	Tickmarklänge in <code>val %</code> der Plotgröße Vorzeichen "-": nach außen, "+": nach innen
<code>las =</code>	<code>0</code>	Text senkrecht zu den Tickmarks
	<code>1</code>	Text horizontal
	<code>2</code>	Text parallel zu den Tickmarks
<code>at =</code>	<code>c(p1, p2, ...)</code>	Position der Tickmarks
<code>labels =</code>	<code>c("str1", "str2", ...)</code>	Beschriftung der Tickmarks. Kann nur in Verbindung mit <code>at</code> benutzt werden!
<code>axis(</code>	<code>1 2 3 4, ...)</code>	Nachträgliches Zeichnen der Achsen



VISUALISIERUNG VON MEHRDIMENSIONALEN DATEN:

- **Pairs:**
`pairs(mat)` Scatterplotmatrix für jede Kombination von Spalten in `mat`
- **Starplot:**
`stars(mat)` Jede Zeile aus `mat` wird in Form eines Sternes dargestellt
- **Chernoff Faces:**
`faces(mat)` Jede Zeile aus `mat` wird in Form eines Gesichts dargestellt. Jede Spalte, bzw. Variable, assoziiert dabei mit einem Element des Gesichts
- **Brushing:**
`brush(mat)` Interaktive Scatterplotmatrix, d.h. die einzelnen Beobachtungen in einem Scatterplot können markiert werden, wobei sie auch in allen anderen Scatterplots markiert werden.
- **Spin:**
`spin(mat)` Erweiterung des `brush`-Kommandos um einen dreidimensionalen Rotationsplot. Auch hier existiert die Interaktivität.



UNTERSUCHUNG UND VERGLEICH VON VERTEILUNGEN:

- Histogramm:

```
hist( x )
```

Histogramm für x

Optionen:

- `prob = T` Darstellung in Y so, daß die Gesamtfläche 1 ergibt.
- `nclass = n` Anzahl der Klassen.
- `breaks = ...` Klassengrenzen.

✓ Mit Histogrammen kann man "Löcher" finden

- Geglättetes Histogramm – Dichteschätzer:

```
plot( density( x ), type = "l" )
```

Optionen:

- `n = N` Anzahl Stützstellen.
- `window = "typ"` Art des verwendeten Gewichtungsfensters.
- `width = b` Breite des Fensters.

✓ Dichteschätzer können Multimodalität entdecken

- Boxplots:

```
boxplot( mat )
```

Für jede Zeile aus `mat` wird ein Boxplot in einem gemeinsamen Koordinatensystem gezeichnet.

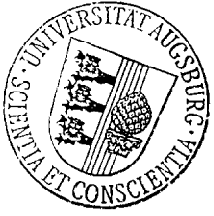
✓ Boxplots sind ideal, um Verteilungen zu vergleichen

- Quantil-Quantil-Plots:

```
qqprob( x )
```

Scatterplot der Quantile aus `prob` gegen die empirischen Quantile von x .

✗ QQ-Plots sollen die Verteilung von Daten prüfen



Statistische Modelle:

DATEN IN MODELLEN:

- Datenhaltung allgemein:

<code>ls</code>	Auflisten aller Objekte
<code>objects</code>	Begrenzung der Auswahl möglich
<code>rm</code>	Löschen von Objekten

- Datenhaltung in Dataframes:

Ein Dataframe ist eine spezielle Form der Datenrepräsentation, in der Daten verschiedenen Typs in Form einer Matrix zusammengefaßt werden.

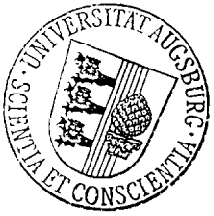
Dataframes sind ein eigener Datentyp in S-Plus:

- `as.data.frame` (Konversion)
- `is.data.frame` (Test)
- `data.frame` (Erzeugung)

werden analog zu anderen Typen bereitgestellt.

Beispiel:

```
> auto.dat <- data.frame( auto.stats )
> is.data.frame( auto.dat )
[1] T
> names( attributes( auto.dat ) )
[1] "names"      "row.names"  "class"
```



FORMELN IN MODELLEN:

Darstellung der Formeln zur statistischen Modellierung durch S-Plus Objekte, z.B:

- mathematische Darstellung

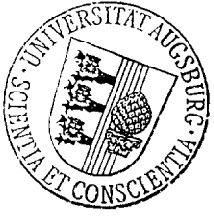
$$Fuel = \alpha + Weight \beta_1 + Disp. \beta_2$$

- Formeldarstellung in S-Plus

$$Fuel \sim 1 + Weight + Disp.$$

Syntax von Formeln:

- $T \sim F$ T wird durch F modelliert
- $+$, $-$, $/$ stehen für die Rechenarten
- $A : B$ Alle Interaktionen von A und B werden modelliert
- $A * B$ $= A + B + A : B$
- $(A+B) ^m$ Alle Interaktionen bis zum Grad m
- $. \sim .$ $.$ steht für einen kompletten Ausdruck
- -1 Modell ohne Konstante



MODELLIERUNGSFUNKTIONEN:

Ein Modell besteht immer aus einer Formel und den dazugehörigen Daten:

- allgemein:

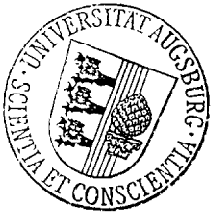
`modell(Formel, Daten)`

- Beispiel „lineares Modell“:

```
lm(Fuel ~ 1 + Weigth + Disp., auto.dat)
```

Aktuell verfügbare Modellfunktionen:

- `lm` lineares Modell
- `aov` Varianzanalyse
- `varcomp` Varianzkomponentenmodelle
- `glm` Generalisiertes lineares Modell
- `gam` Generalisiertes additives Modell
- `loess` lokales Regressions-Modell
- `tree` CART



BEISPIEL:

Hinzunahme einer Variablen:

```
> add1( ozone.lm, ~ . + wind + radiation )  
Single term additions
```

Model:

```
ozone ~ temperature
```

	Df	Sum of Sq	RSS	Cp
<none>			37.74698	39.13219
wind	1	5.83962	31.90736	33.98517
radiation	1	3.83904	33.90793	35.98575

Wegnahme einer Variablen:

```
> drop1(lm(ozone ~ radiation + temperature + wind, data=air))  
Single term deletions
```

Model:

```
ozone ~ radiation + temperature + wind
```

	Df	Sum of Sq	RSS	Cp
<none>			27.84808	29.93018
radiation1	4.05928	31.90736	33.46893	
temperature	1	17.48174	45.32982	46.89140
wind	1	6.05985	33.90793	35.46950

Automatische Selektion von Variablen:

```
> step( ozone.lm, ~ . + radiation + temperature + wind,  
trace=F)
```

Call:

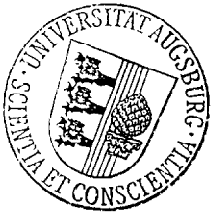
```
lm(formula = ozone ~ temperature + wind + radiation, data=air)
```

Coefficients:

(Intercept)	temperature	wind	radiation
-0.2973296	0.05004432	-0.07602195	0.002205541

Degrees of freedom: 111 total; 107 residual

Residual standard error (on weighted scale): 0.5101592



BEISPIEL:

Vorhersage:

```
> ozone.lm <- lm( ozone ~ temperature, data=air )
> summary( air$temperature )
  Min. 1st Qu. Median  Mean 3rd Qu.  Max.
    57     71     79 77.79    84.5    97
> newtemp <- 100:110
> predict( ozone.lm, data.frame( temperature = newtemp ) )
   1     2     3     4     5     6     7     8
4.810359 4.880723 4.951086 5.02145 5.091813 5.162176 5.23254 5.302903
   9    10    11
5.373267 5.44363 5.513994
```

Konfidenzintervalle:

```
> pre <- predict( ozone.lm , se.fit = T )

> names( pre )
[1] "fit"          "se.fit"       "residual.scale" "df"

> lim <- pointwise( pre )

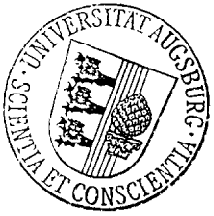
> names( lim )
[1] "upper" "fit"    "lower"

> plot( air$temperatur, air$ozone )

> abline( ozone.lm )

> lines( sort( air$temperature ),
         lim$lower[ order( air$temperatur ) ] )

> lines( sort( air$temperature ),
         lim$upper[ order( air$temperatur ) ] )
```

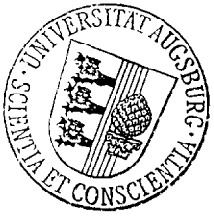


PRAKTIKUMSAUFGABEN:

1. Gegeben seien folgende Wahlergebnisse:

Partei	Jahr 1994	1990	1987
CDU/CSU	41.5	43.8	44.3
SPD	36.4	33.5	37.0
FDP	6.9	11.0	9.1
Grüne	7.3	4.9	8.3
PDS	4.4	2.4	-.-
Sonstige	3.5	4.4	1.2

- Lesen Sie die Ergebnisse in einzelne Variablen: `Wahl94`, `Wahl90`, `Wahl87` ein!
Ordnen Sie den Zahlen die entsprechenden Parteinamen zu.
- Stellen Sie die Wahlergebnisse in einem Piechart dar!
- Passen Sie die Reihenfolge der Parteien ihrem Anteil in '90 an!
- Ordnen Sie den Parteien die üblichen Farben zu (PDS: ?).
- Stellen Sie die Differenzen zwischen den einzelnen Wahlen in einem Barchart dar! Auch hier sollten Reihenfolge und Farbe stimmen.
- Erstellen Sie eine Datei mit den obigen Wahlergebnissen, und lesen Sie diese Datei mittels der `scan` Funktion ein.
- Die Funktion `lsfit` berechnet eine kleinste Quadrat Geradenanpassung an Daten (siehe Helpfile). Berechnen Sie anhand der gegebenen Daten, die lineare Approximation der Ergebnisse der Union.
Stellen Sie das gewonnene Ergebnis graphisch dar, und berechnen Sie, wann die Partei unter die 5% Hürde fällt!



PRAKTIKUMSAUFGABEN:

- h) Fassen Sie die drei Variablen zu einer Matrix zusammen!
- i) Berechnen Sie möglichst effizient das durchschnittliche Ergebnis jeder Partei der letzten drei Wahlen.
- j) In einem Jahr addieren sich die Ergebnisse nicht zu 100%. Suchen Sie das fehlerhafte Jahr mit einer geeigneten Funktion!
- k) Stellen Sie die Verläufe der einzelnen Parteien mittel eines `tsplot` dar. Ist es sinnvoll, die Daten dafür in den Typ `ts` umzuwandeln?

2.

- a) Erzeugen Sie jeweils Objekte vom Typ
 - Vektor
 - Liste
 - Matrix
 - Array.Testen Sie mit der "is." Funktion, ob diese Objekte jeweils nur einer Klasse angehören, und stellen Sie eine Ergebnismatrix der Art

Vektor	is.vector T	is.list ?	...
Liste	?	T	
...			

auf! Interpretieren Sie das Ergebnis!

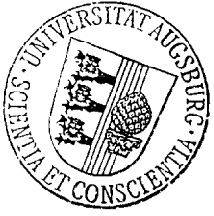
- b) Der Befehl

```
> dim(iris)
```

liefert

```
[1] 50  5  3.
```

Für eine statistische Analyse benötigen Sie jedoch alle 3 Sorten in einer Matrix. wandeln Sie die Daten auf dreierlei Weise um!



PRAKTIKUMSAUFGABEN:

3. Die Arbeitslosenzahlen des ehemaligen Bundesgebiets sind in folgender Tabelle aufgeführt:

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1991:										1599	1618	1731
1992:	1875	1863	1768	1747	1704	1715	1828	1822	1784	1830	1885	2025
1993:	2257	2288	2223	2197	2148	2166	2326	2315	2288	2359	2410	2510
1994:	2740	2740	2640	2590	2506	2478	2570	2531	2452			

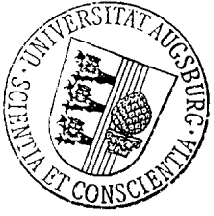
- Lesen Sie die Daten in eine Variable vom Typ `ts` ein.
 - Berechnen Sie für die letzten drei Jahre eine lineare Trendfunktion, und stellen Sie das Ergebnis graphisch dar.
 - Führen Sie die Berechnung aus b) jeweils für die einzelnen letzten drei Jahre durch. Wählen Sie auch hier eine geeignete Darstellung.
 - Die Funktion `loess` berechnet (im Gegensatz zu `lsfit`) eine lokale Regressionskurve. Experimentieren Sie mit dem Parameter `span`, um eine möglichst gute Anpassung für die Daten zu erlangen.
4. Sie wollten folgende Funktion ausführen:

```
> sum(list(1,2,3))
```

erhielten jedoch leider folgende Meldung:

```
Error in sum(list(1,2,3)):Numeric summary undefined for mode "list"  
Dumped
```

- Schreiben Sie eine Funktion, die
 - Eine Liste als Argument nimmt, und deren Summe ausgibt.
 - Einen Fehler meldet, wenn das Argument keine Liste ist.
 - Dito, wenn ein Element nicht atomar und numerisch ist.
- Lösen Sie das Problem rekursiv!
- Erweitern Sie die Funktion so, daß sie auch auf Listen von Listen arbeitet!



PRAKTIKUMSAUFGABEN:

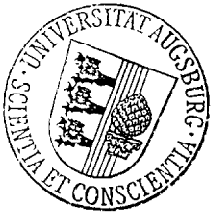
5. Betrachtet sei die Klasse der komplexen Zahlen in der Polarkoordinatendarstellung:
 - a) Erweitern Sie den Konstruktor so, daß er auch auf Vektoren arbeitet.
 - b) Erstellen Sie eine entsprechende Methode für
 1. Die Multiplikation
 2. Die Addition
 3. Die Ausgabe (print)

6. Betrachten Sie die Variablen `lottery.payoff` und `lottery.number`.
 - a) Stellen Sie beide Größen jeweils in einem Boxplot und einem Histogramm dar.
 - b) Stellen Sie den Gewinn unterteilt nach der führenden Stelle der Gewinnnummer (3 = 003 etc) mittels Boxplots dar. Hinweis: `split` erleichtert das Leben.
 - c) Untersuchen Sie mittels eines Dichteschätzers die Verteilung der gezogenen Nummern.

Kombinieren Sie Dichteschätzer und Histogramm.

Variieren Sie sowohl die Klassen des Histogramms, als auch die Breite des Dichteschätzers.

7. Der Datensatz `voice.five` enthält das Spectrum des Wortes "five".
 - a) Stellen Sie die Daten mittels `persp` dar. Wählen Sie eine geeignete Ansicht.
 - b) Benutzen Sie die Funktion `image` um die Daten darzustellen. Welchen Effekt hat die Variation der Farbskala, bzw. des Darstellungsbereiches in `z`?
 - c) Verwenden Sie auch `contour`, um die Daten darzustellen. Welche Funktion hat der Parameter `nlevels`?
 - d) Diskutieren Sie die drei Verfahren aus a)-c)!



LÖSUNGEN ZU PRAKTIKUMSAUFGABEN:

```
1. a) > Wahl87 <- scan( what = 0 )
      1: 44.3
      2: 37.0
      3: 9.1
      4: 8.3
      5: NA
      6: 1.2
      7:
      > names(Wahl87) <- c("Union", "SPD", "FDP", "Gruene",
+ "PDS", "Sonstige")
      > Wahl87
      Union SPD FDP Gruene PDS Sonstige
      44.3  37 9.1   8.3  NA   1.2

90,94 analog

b) > pie( Wahl87 )
Error: Missing values not allowed
Dumped
Error in pie
> pie( replace(Wahl87, 5, 0) )

c)  i) neu eintippen
     ii) > Wahl87[ rev( order( Wahl90 ) ) ]

d) > pie( Wahl90, col=c(1,2,3,4,7,6) )

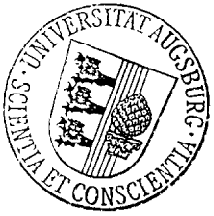
e) > barplot( Wahl94[ rev( order( Wahl90 ) ) ]-
+ Wahl90[ rev( order( Wahl90 ) ) ], col=c(1,2,3,4,7,6) )

f) > write( t( cbind( Wahl94, Wahl90, Wahl87 ) ), "tmp",
+ ncolumns = 3 )
```

Inhalt der Datei:

```
41.5 43.8 44.3
36.4 33.5 37
6.9 11 9.1
7.3 4.9 8.3
4.4 2.4 NA
3.5 4.4 1.2
```

```
> matrix( scan("tmp", what=c(0,0,0)) , 6, 3, byrow=T)
      [,1] [,2] [,3]
[1,] 41.5 43.8 44.3
[2,] 36.4 33.5 37.0
[3,]  6.9 11.0  9.1
[4,]  7.3  4.9  8.3
[5,]  4.4  2.4  NA
[6,]  3.5  4.4  1.2
```



LÖSUNGEN ZU PRAKTIKUMSAUFGABEN:

```
1. g) > times <- c( 1987, 1990, 1994 )
> votes <- rev( c( 41.5, 43.8, 44.3 ) )
> plot( times, votes, type="l" )
> abline( lsfit( times, votes) )
> lsfit( times, votes )$coef
  Intercept          X
    858.1608 -0.4094595

> (5-858.1608) / -0.4094595
[1] 2083.627

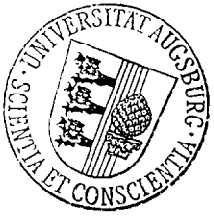
h) > Wahl <- cbind( Wahl94, Wahl90, Wahl87 )
> Wahl
      Wahl94 Wahl90 Wahl87
Union    41.5   43.8   44.3
SPD     36.4   33.5   37.0
FDP      6.9   11.0    9.1
Gruene   7.3    4.9    8.3
PDS      4.4    2.4    NA
Sonstige 3.5    4.4    1.2

i) > apply(Wahl, 1, mean)
Union    SPD FDP  Gruene PDS Sonstige
43.2 35.63333 9 6.833333 NA 3.033333

j) > apply( Wahl, 2, sum )
Wahl94 Wahl90 Wahl87
  100    100    NA

> apply( Wahl, 2, sum, na.rm=T )
Wahl94 Wahl90 Wahl87
  100    100   99.9

k) > tsplot( rev( Wahl[1,]), rev(Wahl[2,]), ...)
```



LÖSUNGEN ZU PRAKTIKUMSAUFGABEN:

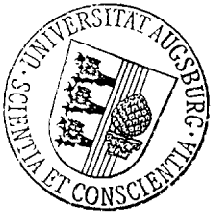
2. a)

	is.vector	is.list	is.matrix	is.array
Vektor	T	F	F	F
Liste	T	T	F	F
Matrix	F	F	T	T
Array	F	F	F	T

```
b) > dim( iris )
[1] 50  4  3
> flow <- rbind( iris[,,1], iris[,,2], iris[,,3] )
> dim(flow)
[1] 150  4
> flow[1, 1] == iris[1, 1, 1]
Sepal L.
      T
> flow[150, 4] == iris[50, 4, 3]
Petal W.
      T
> flow[150, 1] == iris[50, 1, 3]
Sepal L.
      T

> flow <- apply( iris, 2, rbind )
> flow[150, 4] == iris[50, 4, 3]
Petal W.
      T
> flow[1, 1] == iris[1, 1, 1]
Sepal L.
      T
> flow[150, 4] == iris[50, 4, 3]
Petal W.
      T
> flow[150, 1] == iris[50, 1, 3]
Sepal L.
      T

> flow <- matrix( aperm( iris, c( 1, 3, 2) ), 150, 4)
> flow[1, 1] == iris[1, 1, 1]
[1] T
> flow[150, 4] == iris[50, 4, 3]
[1] T
> flow[150, 1] == iris[50, 1, 3]
[1] T
```



LÖSUNGEN ZU PRAKTIKUMSAUFGABEN:

```
3. a) > tmp <- ts( scan( "test.dat", what=0 ),
+ start=c(1991,10), frequency=12, end=c(1994,9))
> tmp
```

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1991:										1599	1618	1731
1992:	1875	1863	1768	1747	1704	1715	1828	1822	1784	1830	1885	2025
1993:	2257	2288	2223	2197	2148	2166	2326	2315	2288	2359	2410	2510
1994:	2740	2740	2640	2590	2506	2478	2570	2531	2452			

```
b) > tsplot( tmp )
> abline( lsfit( seq( 1991+10/12,1994+9/12,1/12 ),
+ tmp) )
```

```
c) > abline( lsfit( seq( 1991+10/12,1992+9/12,1/12 ),
+ tmp[1:12] ), col=2 )
> abline( lsfit( seq( 1992+10/12,1993+9/12,1/12 ),
+ tmp[13:24] ), col=3 )
> abline( lsfit( seq( 1993+10/12,1994+9/12,1/12 ),
+ tmp[25:36] ), col=4 )
```

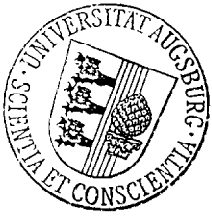
```
d) > tsplot( tmp )
> lines( lowess( tmp, f=0.5 ), col=2 )
> lines( lowess( tmp, f=0.3 ), col=3 )
> lines( lowess( tmp, f=0.4 ), col=4 )
```

```
4. a) 1. > suml1
function(l)
{
    s <- 0
    for(i in 1:length(l))
        s <- s + l[[i]]
}
```

2. Test mit `is.list`

3. Test mit `is.atomic` und `length(...)==1` (wg. vector)

```
b) > suml
function(l)
{
    if(length(l) == 1)
        l[[1]]
    else l[[1]] + suml(l[2:length(l)])
}
```



LÖSUNGEN ZU PRAKTIKUMSAUFGABEN:

```
4. c) > suml2
function(l)
{
  if(length(l) == 1)
    if(is.list(l[[1]]))
      suml2(l[[1]])
    else l[[1]]
  else if(is.list(l[[1]]))
    suml2(l[[1]]) + suml2(l[2:length(l)])
  else l[[1]] + suml2(l[2:length(l)])
}
```

5. a) Test der Eingabedaten auf

- i) Typ: `is.numeric`
- ii) gleiche Länge: `length(Winkel) == length(Radius)`

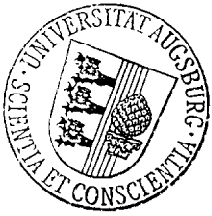
Ansonsten ist der Konstruktor identisch mit dem aus dem Kurs.

b) i) Analog zur Folie in der Vorlesung. Eine Erweiterung des Konstruktors ist bereits erfolgt – Achtung: Multiplikation und Addition auf Vektoren wird implizit vorausgesetzt!

- ii)
 - Umwandlung in Koordinatendarstellung
 - Addition
 - Rückkonversion in Polarkoordinaten

Hier empfiehlt es sich evtl. allgemeine Konversionsfunktionen zu erstellen.

iii) Zeilenweise Ausgabe der Winkel und Radien mittels `cat`.



LÖSUNGEN ZU PRAKTIKUMSAUFGABEN:

```
6 a) > par( mfrow=c(2,2) )
      > hist( lottery.payoff )
      > hist( lottery.number )
      > boxplot( lottery.payoff )
      > boxplot( lottery.number )

b) > boxplot( split( lottery.payoff, lottery.number%%100) )

c) > hist( lottery.number, prob=T )
      > lines( density( lottery.number, width=250 ) )
      > lines( density( lottery.number, width=400 ), col=2 )

etc.
```

```
7 a) > persp( voice.five )
      > persp( voice.five, eye=c(10000, 1/6, 8) )

b) > image( voice.five )
      > range( voice.five$z )
      [1] 0.013888 4.597048
      > image( voice.five, zlim=c(0,5) )
      > image( voice.five, zlim=c(0,10) )
      > image( voice.five, zlim=c(2,5) )

c) > contour( voice.five )
      > contour( voice.five, nlevels=10 )
      > contour( voice.five, nlevels=10, labex=0 )

d) Je nach Art der Analyse und der Daten sind die
    Darstellungen zu wählen.
```