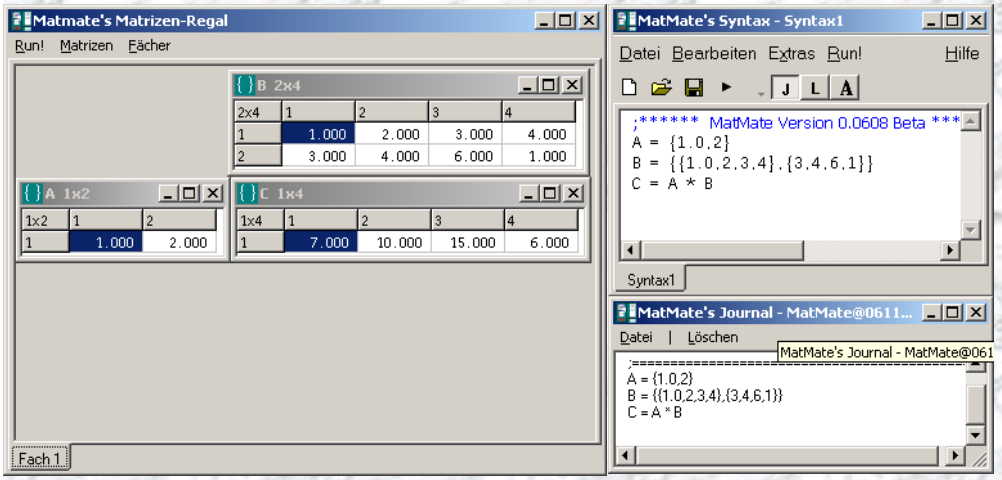
MatMate

Matrizen-Taschenrechner
unter Windows XP

Version: Testversion 0.0610
Okt 2006

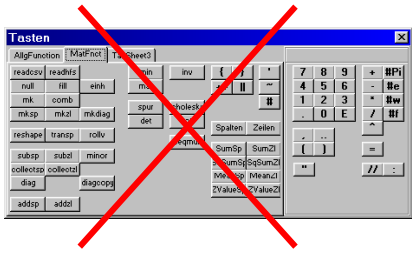


Gottfried Helms

Universität Kassel

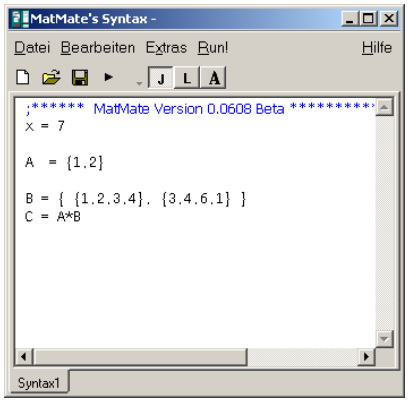
email: helms@uni-kassel.de

MatMate Einführung



MatMate ist für Matrizenrechnen das, was für normales Rechnen ein einfacher "Taschenrechner" ist: ein einfach zu bedienendes Werkzeug auch für nebenbei zu bewältigende Zwischenrechnungen.

Auf eine Simulation der Knöpfcheneingabe wie bei Taschenrechnern üblich wurde (nach einer Probeimplementation) dennoch verzichtet; stattdessen hat man ein Eingabefeld für eine Zahl oder Formel - genauergenommen sogar eine ganze Textdatei.



Der Befehl

```
x=7
```

erzeugt ein Anzeigefeld namens X und zeigt dort die Zahl 7 an.

Der Befehl

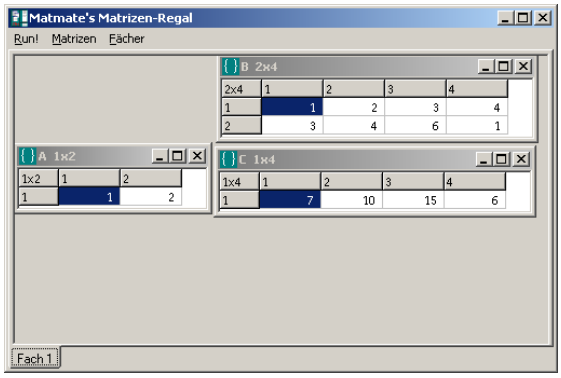
```
y=4*(3+x)^2
```

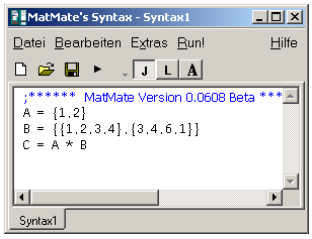
erzeugt ein Anzeigefeld namens A und zeigt dort das Ergebnis der Formel an, unter Einrechnung des X, das soeben vorher berechnet wurde.

Die Befehle

```
A  = {1,2}
B = { {1,2,3,4}, {3,4,6,1} }
C = A*B
```

erzeugen die Matrizen A (einzeilig), B (zweizeilig) und C (zweizeilig) und MatMate zeigt die Matrizen in einem Fach in seinem Anzeigeregal.

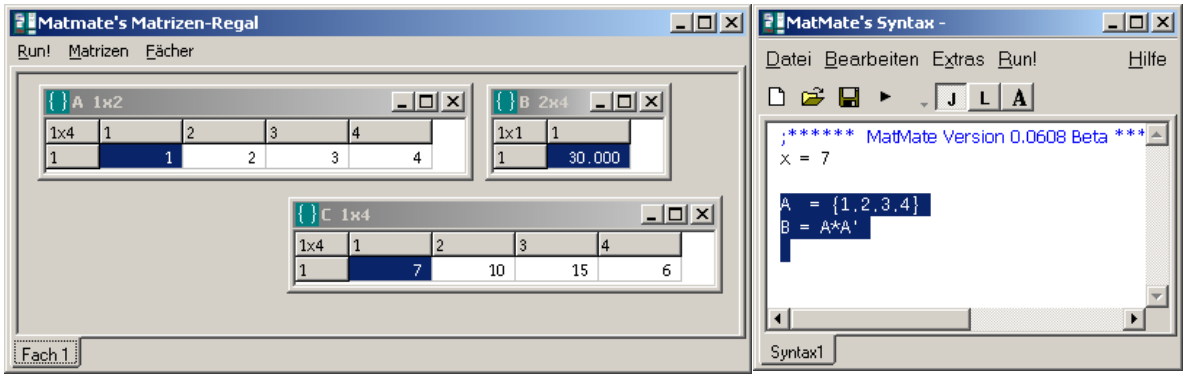




Sie tippen die Befehle so wie sie hier stehen, wie in einer Textverarbeitung. Dabei haben Sie alle Möglichkeiten der Tippfehlerkorrektur, Übernahme von Zeilen aus Ihrer Textverarbeitung via Clipboard etc. Um den Befehl auszuführen lassen Sie die Cursormarke auf der Zeile und drücken das Knöpfchen Run!

Also

```
A= {1,2,3,4}
b= a * a'
```



Markieren Sie beide Zeilen schwarz, dann können sie mit ***Einem*** RUN!-Befehl ausgeführt werden.

Statt des Knopfes RUN! können Sie einfach die Tastenkombination ALT-R drücken - dann brauchen Sie nicht ständig zwischen Maus und Tastatur zu wechseln.

MatMate ist noch nicht vollständig ausprogrammiert; insbesondere bekommen Sie noch keine Fehleranalyse für Tippfehler: was nicht geht, wird meist einfach übergangen. Da es aber bereits jetzt nützlich für einfache Rechnungen wie auch für tiefere Analyse von Matrizenrechnungen ist, möchte ich es jetzt als Beta-/Testversion veröffentlichen.

---------- Rechnen

Es sind alle Standardrechenarten und einige spezielle Funktionen (meist für Untersuchung der Korrelationsanalyse) implementiert. Darunter sind die 4 Grundrechenarten, Potenzierung, Exponential und trigonometrische Funktionen, sowie viele Matrizenfunktionen, bei denen sich ein großer Teil mit dem Extrahieren von Teilmatrizen und Zusammensetzen von Matrizen beschäftigt.

Die Ergebnisse werden in dem 2. Fenster, dem MatMate-"Matrizen-Regal" angezeigt: alle Matrizen haben ein eigenes kleines Windows-Fenster, und alle skalaren Ergebnisse haben eine Zeile in einem gemeinsamen Ausgabefenster.

Unterschiedene elementare Datentypen sind z.Zt:

- Real-(float)- Zahlen (extended precision) (-12.3456E-27)
  Ganz-(integer)-Zahlen (32, in einigen Funktionen 64 Bit)
  Logische (boolean) Werte (TRUE,T,W , FALSE,F)
  Strings ("abcdef")

Unterschiedliche Strukturen sind:

- Skalare (einfache Werte sqrt(14), alle Datentypen)
  Vektoren (Matrizen 1 Zeile oder 1 Spalte, alle Datentypen)
  Matrizen (mehrere Zeilen/Spalten, alle Datentypen)
  Arrays (wie Matrizen)
  Integer-ranges (5..8)
  Real-ranges (1.0..1.5)
  Listen von Werten (1,2,3,4)
  Listen von Ranges (2..5´7..9)

----------Die Formelsprache / die Syntax

1 Befehl pro Zeile

Grundprinzip ist: eine Zeile - ein Berechnungs-Befehl: Cursor setzen - Formel eintippen - Run! - Ausrechnen.

> Am Ende jeder Zeile kann zusätzlich ein beliebiger Kommentar stehen, z.B. weil Sie lange Befehlsfolgen abspeichern und beschreiben wollten. Trennen Sie einen Kommentar durch einen Doppelschrägstrich //, also zwei aufeinanderfolgende / von der Formel ab.

Eine Berechnung ohne Kommentar

```
a = {1,2,3,4}
```

mit Kommentar

```
a = {1,2,3,4}          // eine 1-zeilige Matrix
                       // mit 4 Einträgen anlegen
```

Eine Zeile kann auch fortgesetzt werden; am Ende muß dann ein einzelnes Unterstreichungszeichen stehen (wie in Visual Basic)

```
a = { {1,2}, _
      {3,4}  _
           }          // eine 2x2 Matrix anlegen
```

Zählt als 1 Statement!

Der Cursor muß in eine der auszuführenden Zeile gesetzt werden. Die Zeilengruppe braucht nicht markiert zu werden: die Umgebung wird nach eventuellen Zeilenfortsetzungszeichen durchsucht.

mehrere Befehle

Es können auch mehrere Befehle zusammen ausgeführt werden. Mann muß dann die Markierung auf die gewünschten Zeilen erweitern.

```
a = {1,2,3,4}
b = a * a'
```

Die Markierung überstreicht beide Zeilen; der Cursor kann irgendwo stehen. Run! drücken. Beide Zeilen werden zusammen ausgeführt.

Ist eine Markierung vorhanden, werden alle Zeilen, die von ihr berührt werden i*nclusive evtl vorhandener vorlaufender und nachlaufender Fortsetzungszeilen* ausgeführt. Ein komplexes Beispiel:

```
a = mk (2,2, _
     1,2, _
     3,4)              // eine 2x2 Matrix anlegen
b = mk (2,2, _
     1,3, _
     2,4)              // die Transponierte anlegen
c = a * b - _
          b * a        // Matrixmultiplikationen
```

Alle 3 Befehle, die hier von der Markierung berührt werden, werden komplett gelesen und ausgeführt. Der Cursor braucht in diesem Fall NICHT in einem der Befehle zu stehen. Die Markierung wird nach Ausführung nicht entfernt; Befehle können deshalb iteriert werden, indem man einfach Run! oder die Alt-R-Taste mehrmals drückt.

Ist KEINE Markierung vorhanden, wird die Zeile ausgeführt, in der sich der blinkende Cursor befindet.

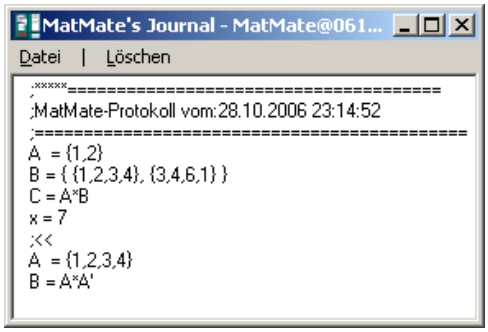
----------einfaches Ändern der Daten einer Matrix

Neben der einfachen Berechnungsweise für Matrizen ist auch die direkte Werteingabe ein praktisches Feature, mit denen Veränderungen im Input einer Formel schnell ausprobiert werden können. (Allerdings erscheinen diese manuellen Änderungen derzeit noch nicht im Journal, und dieses Verfahren wird nicht mehr empfohlen.)

----------Arbeiten mit Daten aus der Literatur

Dadurch, daß die Bedienung des Matrizenrechners über eine Skriptsprache funktioniert, können Sie Matrizen, die Sie aus der Literatur übernommen haben, als Text-/Skript-Eingabe speichern, und Ihre Untersuchungen später mit demselben Datenmaterial reproduzieren oder abwandeln.

---------- Journal



Alle ausgeführten Befehle werden in einem Journal protokolliert. Diese Datei wird anschließend automatisch gespeichert.

Durch die Journal-Funktion können Sie alle Berechnungsschritte reproduzieren, eventuelle Denkfehler untersuchen und verbessern. Insbesondere bei Iterationen ist das Journal vorteilhaft: alle Iterationsschritte sind sequentiell hintereinander aufgezeichnet und können später en bloc wiederholt werden.

Interaktion mit anderen Programmen unter Windows

Es gibt mehrere Schnittstellen:

- Da Sie Befehle/Formeln tippen, haben Sie die Möglichkeit Ihre Arbeiten und Analyseschritte in normalen Textdateien zu spreichern.
- Die berechneten Matrizen können Sie außerdem in Dateien an andere statistische oder mathematische Programme übergeben oder sie in denselben Formaten von dort aus einlesen.
- Die dritte Möglichkeit ist die Windows-Zwischenablage, über die Sie Matrizendaten z.B. direkt an Excel oder Winword übergeben können.

----------(Aktuelles)

- COMP c = <formel>

In der aktuellsten Version kann der Befehlsteil COMP überall entfallen. Die Grafiken in diesem Heft zeigen t.w. noch den Stand der vorigen Version. Die dort dargestellte Syntax funktioniert aber immer noch.

Formel ausrechen:

Neu:

```
A = {1,2,3} * {1,2,3} '
```

Alt:

```
COMP A = {1,2,3} * {1,2,3} '  // NICHT MEHR NÖTIG
```

- Neue Konzepte

Bei der Erweiterung des Programms sind einige Konzepte und Funktionen geändert worden; die alte Funktionsweise ist t.w. noch vorhanden, sodaß man bspw eine Matrix direkt über die geschweiften Klammern (A={1,2,3,4}) oder wie in älteren Versionen über die MkZl, MkSp, Mk-Funktionen anlegen kann (A=mkzl(1,2,3,4))

Ziele / Begrenzungen

----------Tool zur Matrizenrechnung

Das Hauptziel dieses Programms ist die Unterstützung des Selbststudiums der Hintergründe und Vorgänge bei der Matrizenrechnung. Man kann viele Rechenprozesse aus der Literatur nachvollziehen und transparent machen.

Es ist bisher NICHT intendiert, eine vollständige Programmiersprache zu entwickeln. Es ist ebenfalls bisher nicht intendiert, ein symbolisches Algebrasystem zu implementieren. Hierzu gibt es bereits diverse professionelle und semiprofessionelle Angebote, und sogar Freeware-Versionen. MatMate will dazu nicht in Konkurrenz gehen.
Es gibt mittlerweile eine Makro-Sprache.

Darüberhinaus gibt es einen "Demo"-Modus, in dem Skripte ablaufen können, in denen Berechnungen schrittweise (oder timergesteuert) durchgeführt werden. Mithilfe solcher Skripte kann man eine Art "living-Letter" an Kollegen senden oder den Unterricht anreichern.

---------- Anzeigen der Ergebnisse/Ausgabeprotokoll

Der Schwerpunkt liegt -neben der Verfügbarkeit eines billigen Matrizenrechners auf Leistungskurs- und College-Niveau - auf der Sichtbarmachung der Rechenvorgänge. Durch die Präsenz der Matrizenanzeige wird die Matrixalgebra besonders transparent. Die Matrixergebnisse kann man via Clipboard in konventionelle Textverarbeitungsprogramme übernehmen.

Weiter gibt es in der aktuellen Version ein laufendes Ausgaben-Fenster geben, in das Ergebnisse zusammen mit den Formeln/befehlen dokumentiert werden, sodaß matrizenanalytische/statistische Artikel unter Einbeziehung der Operationen und Ergebnisse direkter geschrieben werden können

----------Betaversion

Es handelt sich hier noch um eine Beta-Version, für die noch Listen an zu verbessernden Optionen bestehen.

Insbesondere gibt es noch keine besondere Fehleranalyse; Fehler in der Syntax oder während einer Berechnung führen meist einfach zum Überspringen des betreffenden Befehls, und evtl der Folgebefehle. Programmabstürze sind immerhin in der aktuellen Testphase noch nicht aufgetreten - d.h. Arbeitsergebnisse und Skripte sind noch nicht verloren gegangen. Aber die Gefahr, solch eine Situation bei einem neuen Update einzubauen ist nicht auszuschließen.

Auch sind einige mathematische Verfahren noch nicht vollständig stabilisiert; hier sind meist einfach die von Borland-Delphi gelieferten Methoden übernommen. Matrixexponential und ~logarithmus sind bisher einfache Reihenoperationen ohne Optimierungen. Meist wird hierbei mit Extended-Präzision gearbeitet, es gibt einige Double-Precision-Module. D.h. Genauigkeit bei skalaren numerischen Operationen Zahlen liegt bei ca 15 Dezimalstellen, bei ungünstig konkonfigurierten Matrizen kann dies beliebig schlechter werden.

Die Größe von Arrays/Matrizen wird von 32-bit Integer-Variablen verwaltet; dh. sie dürften praktisch nur durch den bei Ihnen vorhandenen Hauptspeicher begrenzt sein.

Es wird bisher keine Abschätzung des Rechenaufwandes vorgenommen. Wenn Sie also eine 1000 x 1000 Matrix in mit einer Hauptachsenrotation behandeln, dauert das sehr lange.

- ***Keine Garantie für Datenverluste unmittelbar oder mittelbar.***

einfache Beispiele

----------skalarer Rechner

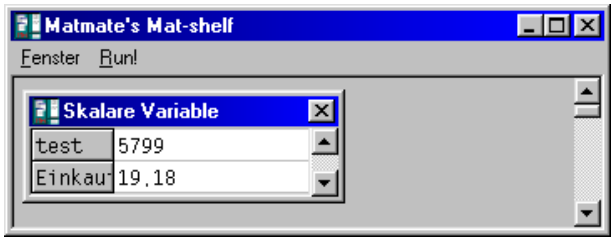
Die Grundrechenarten sowie logarithmische und trigonometrische Funktionen sind implementiert. In den Formeln können Klammern beliebig tief verschachtelt werden. Man kann auf bereits berechnete Ergebnisse zurückgreifen, da jedes Ergebnis einer (skalaren) Variablen (also einem benanntem Anzeigefeld) zugewiesen wird, auf das Sie sich in einer nächsten Formel beziehen können. "Skalar" ist hier die Bezeichnung für einen einzelnen Wert, in Abgrenzung (dieser gewöhnlichen) Rechnung mit einzelnen Zahlen von der Rechnung mit Vektoren, Matrizen und Arrays.

Anzeigen skalarer Ergebnisse

Wenn Sie einen einzelnen Wert so wie auf dem Taschenrechner berechnen wollen, tippen Sie einfach die Formel (Verwenden Sie den Punkt als Dezimal-Trennzeichen):

```
EinkaufSumme = 1.12 + 4.50 + 3.38 + 2.50 + 7.68
```

Im Anzeigefeld "skalare Variable" wird die Einkaufssumme angezeigt.

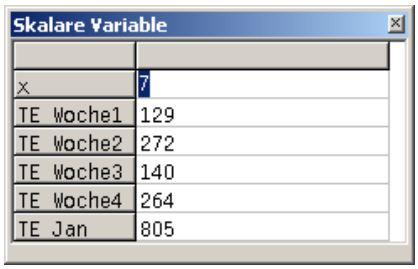


oder

```
// TE_Woche1 : Telefoneinheiten in der Woche Nr...
TE_Woche1= 12+23+45+16+23+8+2
TE_Woche2= 14+13+ 5+ 2+11 + _
                  +23 + 78 +126
TE_Woche3 = 11+23+45+1+1+3+2+6+8+8+9+11+12
TE_Woche4 = 1+5+12+45+67+2+42+64+26

// TE_Jan: Telefoneinheiten für Monat Januar
TE_Jan = TE_Woche1 + TE_Woche2+ _
                   TE_Woche3+TE_woche4
```

Im Anzeigefeld "skalare Werte" werden die Wochensummen und die Summe für Januar angezeigt.



komplexere Formeln

Formeln können Klammern enthalten und beliebig komplex werden. Neben den arithmetischen Rechenarten können auch logische

Vergleiche und eine einfache Form des bedingten Rechnens verwendet werden.

```
x= (a^3 + 3*a*5*(a + 5) + 5^3)^(1/3)
phi2 = 1- cos(phi)²
root = wenn( a>=b, sqrt(a-b) , sqrt(b-a) )
umf = 2*radius* #pi²
```

Sie können in den Formeln #pi und #e, sowie #false und #true als Konstanten angeben.

Beispiel mit Iterationen

Z.B. wollen Sie nachvollziehen, wie mit dem Newton-Verfahren die Wurzel aus 14 berechnet wird. Das übliche Ergebnis berechnen Sie einfach mit der gewöhnlichen Taschenrechnerfunktion.

```
Y = sqrt(14)
```

Es wird eine Anzeigezeile für das Ergebnis erzeugt, und dies erhält den Namen Y. Sie können beliebige Namen bis zur Länge von 16 Buchstaben wählen:

```
standard_Y = sqrt(14)
```

Nun programmieren Sie das Newton-Verfahren. Man nimmt einen Probewert, setzt ihn in die Newton'sche Formel ein, berechnet einen neuen Wert, der wegen der Konstruktion der Formel besser ist als der erste und nimmt diesen dann als neuen Probewert. Das macht man solange, bis der Wert sich nicht mehr (nennenswert) ändert.

```
wert=1 // einen Anfangswert einstellen
```

Führen Sie den obenstehenden Befehl aus. ***wert*** wird angezeigt. Tippen Sie dann die beiden folgenden Befehle und führen Sie sie zusammen aus (Markierung auf beide Zeilen erweitern und Alt-R drücken):

```
altwert = wert
wert = (14 / altwert + altwert) / 2
```

Die Anzeigezeile für ***altwert*** wird eingestellt, und für ***wert*** wird gleich die neue Version ausgerechnet. Wiederholen (iterieren) Sie jetzt diese Befehlsfolge, bis sich ***wert*** nicht mehr wesentlich ändert. Er sollte dann mit ***standard_y*** übereinstimmen. Die verbliebene Ungenauigkeit können Sie sich anzeigen lassen, indem Sie ***wert*** von ***standard_y*** abziehen:

```
chk = wert - standard_y
```

Durch die Anzahl der Iterationen können Sie die Wurzel aus 14 beliebig genau bestimmen. Die Grenzen der maximalen Genauigkeit sind hier die des Betriebssystems, dessen DOUBLE-Genauigkeit verwendet wird, z.B. PC bis ca 16 Dezimalstellen.

----------Matrizenrechner

MatMates Hauptanliegen ist die Unterstützung bei der Matrizenrechnung.

Über die Grundrechenarten hinaus sind auch arithmetische Funktionen für Matrizen implementiert, die nicht nur auf Einzelelementebene, sondern mit der Matrix als Ganzes operieren, wo dies durch die Definition der Funktion Sinn macht.

Nachdem die Matrix-Multiplikation die Matrix als Ganzes behandelt, wird dies auch in allen Funktionen so gehandhabt, die aus der Multiplikation abgeleitet sind, z.B. die exponential- und trigonometrischen Funktionen, deren Definition in einer Summe aus Produkten bzw. Quotienten besteht.

Soll eine solche Funktion sich stattdessen auf die einzelnen Elemente beziehen (z.B. der Logarithmus jedes einzelnen Elements) muß die Matrix temporär als Array verwendet werden; dies wird durch den nachgestellten Array-Operator # eingestellt (Beispiele siehe unten). In der aktuellen Version sind bei den *binären* Operatoren * / und ^ die Varianten *# , /# und ^# hinzugefügt worden, die die entsprechenden Operationen auf den Einzelelementen durchführen so daß Sie die Array-Deklaration in diesen häufigen Fällen nicht mehr brauchen. Bei den Funktionen (z.B. x = sqrt(A) bzw x = sqrt(A#)) ist dies aber weiterhin erforderlich.

Außerdem ist für binäre Operationen zwischen Matrizen und Vektoren eingebaut, daß die Vektoren auf geeignete Dimension expandiert werden, und dann elementweise Operationen durchgeführt werden, wenn die Dimensionen von der Matrix-operation her inkompatibel sind. Expansionen finden nur statt, wenn entweder die Spalten passend sind und genau 1 Zeile vorliegt, oder die Zeilen passend und genau 1 Spalte vorliegt oder es sich um eine 1x1-Matrix handelt.

Anzeigen von Matrizenergebnissen

Alle berechneten Ergebnisse, die in einer Matrix abgespeichert werden, werden sofort angezeigt. Die Form ist z.Zt. ein einfaches dezimales-Fixed-Format; bzw Stringanzeige in der Breite der gegebenen Feldweiteneinstellung dieses kann über den SET-Befehl verändert werden.

Verbergen/Löschen von Matrizen

Haben einige Matrizen nur temporäre Funktion, deren Anzeige nicht interessiert, können sie durch den HIDE- bzw DELETE Befehl scriptgesteuert versteckt bzw. gelöscht werden, wodurch sich das MatMate - MatRegal lichten läßt. Der Hide-Befehl wird auch implizit durchgeführt, wenn Sie das Fenster einer angezeigten Matrix schließen. In der Aktuellen Version gibt es eine zusätzliche Organisationsmöglichkeit in Fächer: nur 1 Fach mit der verwendeten Gruppe von Matrizen wird angezeigt; sie können zwischen den Fächern wechseln.

Ist eine Matrix mit DELETE gelöscht, kann sie nicht mehr als Quelle in einer Formel verwendet werden, sondern muß erst wieder durch eine Zuweisung erzeugt werden. Sie können aber eine Matrix

ausblenden ohne sie zu löschen, wenn Sie das Fenster minimieren oder die HIDE-Funktion verwenden. Durch DISPLAY wird sie wieder angezeigt.

komplexere Formeln

Genau wie skalare Variable können Matrizen in komplexen Formeln verwendet werden. Die Rückgabewerte von Matrixfunktionen sind in der Regel selbst wieder Matrizen, so daß auch alle Matrix-Funktionen beliebig verschachtelt werden können.

```
a = {{4,1},{1,5}}
b = inv(a)* inv( {{5,1},{1,5}}
lad= rot( cholesky( covar2corr(a)),"PCA")
```

Zur Unterstützung der Schreibarbeit gibt es einige Vereinfachungen wie z.B. für die Transposition die postfix-Notation mit dem Apostroph

```
b = a * transp(a)   // oder 2. Schreibweise:
b = a * a'
```

oder den Verkettungsoperator, der Matrizen vertikal aneinanderkettet:

```
b = a || { 1,3 } || null( like(a))
```

oder horizontal:

```
b = {a , {2,5}, einh(like(a)) }
```

Einfaches Beispiel mit Iterationen

Bei Matrizenproblemen muß oft mit Iterationen gearbeitet werden; bspw. kann man auch die Wurzel einer Matrix mit dem Newton-Algorithmus finden, wenn man die Division einer Matrix durch eine andere definiert. In MatMate wird der Divisionsoperator so interpretiert, daß der Divisor durch seine Inverse ersetzt wird, und der Dividend hiermit multipliziert wird. (Dies funktioniert u.U. auch mit nicht-quadratischen Matrizen, wenn eine Pseudoinverse gebildet werden kann). Deshalb kann die Newton-Iteration genau analog zu dem skalaren Beispiel durchgeführt werden:

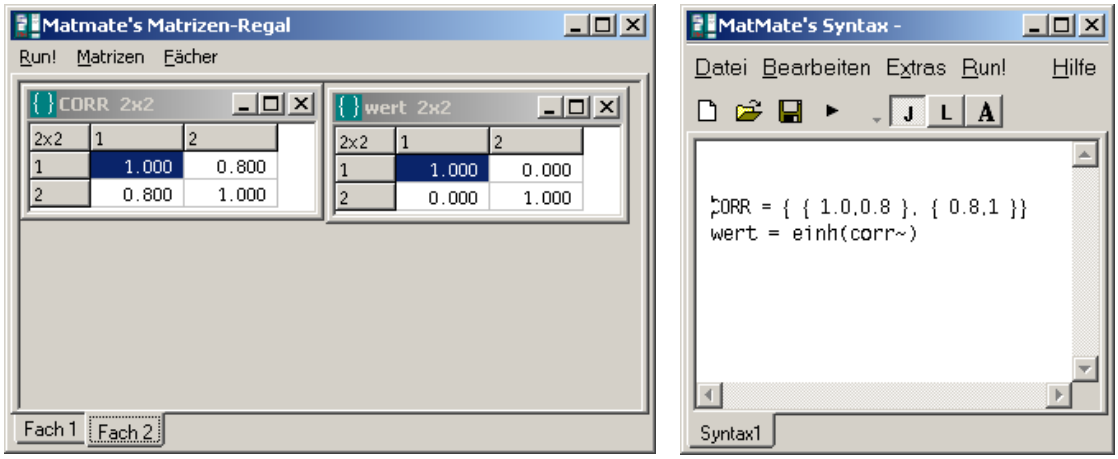
Erzeugen einer Matrix, aus der die Wurzel gezogen werden soll; das ist z.B. grundsätzlich möglich bei einer Korrelations-Matrix:

```
 // eine 2x2-Korrelationsmatrix erzeugen
CORR = { { 1,0.8 },{   0.8,1 }}
```

Nun muß ein Anfangswert für ***WERT*** eingestellt werden:

```
WERT = einh(2) // eine 2x2-Einheitsmatrix
```

Führen Sie den obenstehenden Befehl aus. ***WERT*** wird angezeigt.



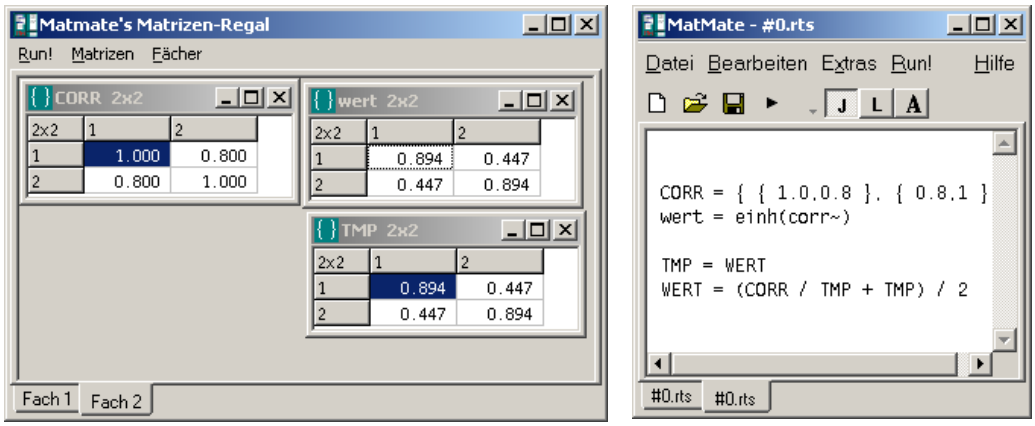
Tippen Sie dann die beiden folgenden Befehle und führen Sie sie zusammen aus (Markierung auf beide Zeilen erweitern und Alt-R drücken):

```
TMP = WERT
WERT = (CORR / TMP + TMP) / 2
```

Die Matrixdivision und -addition, sowie die skalare Division (aller Elemente durch 2) wird durchgeführt, und ein neuer Wert für ***WERT*** berechnet. Wenn Sie die beiden Zeilen mehrfach iterieren, erhalten Sie in ***WERT*** schließlich eine Matrix, für die gilt:

```
CORR = WERT * WERT // nicht transponiert!
```

und die (symmetrische) Wurzel von ***CORR*** darstellt.



(Für die Faktorenanalyse mag interessant sein, daß diese eine Ladungsmatrix darstellt, die für Rotationen zur PCA-Position, Quartimax- oder einer anderen Position verwendet werden kann - und zwar ebensogut wie eine durch die Cholesky-Zerlegung gewonnene Ladungsmatrix. Im Gegensatz zu letzterer Zerlegungsart sind bei dem Newtonverfahren aber keine elementaren Wurzeloperationen notwendig.)

Editieren von Matrizen

Eines der umständlichsten Probleme in Matrizenfähigen Programme ist oft das Editieren: das Herausziehen von Zeilen, Spalten oder Submatrizen bzw. das selektive Updaten von Teilbereichen einer Matrix. Hierzu existiert ein ganzes Set an Utilities in Matmate. Man kann nicht nur Einzelelemente oder Vektoren indizieren:

```
A = randomu(8,8)        // eine 8 x 8 Zufallsmatrix anleg
x = A[2,3]              // element aus Zeile 2, Spalte 3
                        // x ist 1x1-Matrix/Vektor!
```

```
y = v(A[2,3])           // w ist ein skalarer Wert

x = A[2]                // die ganze zweite Zeile
x = A[1..#hi, 3]        // die ganze dritte Spalte

x = A[2..4,3..6]        // die ang. Submatrix
x = A[2´4..5´6, 1..3´7..8]  // listen von Ranges sind
                        //     möglich

// variable Range-definitionen:
myrange = 1..2´7..8
x = A[myrange,2]        // zweite Spalte, Zeiln gem. Range
x = A[4..n,3..m]        // Ausdrücke können verwendet
                        // werden

// indizierung durch Vektoren
x = A[{1,3,5}]          // Zeilen 1,3 und 5

ind = isortsp(A)        // index für sortierten Zugriff
                        // bestimmen
x = A[1..#hi,ind]       // x ist nach spalten sortiert
```

Alle diese Operationen sind auch für Zuweisungen definiert. Lediglich muß bei der Verwendung als Ziel die Matrix geeigneter Größe existieren:

```
B = null(8,8)

B[2,3] = 12
B[4] = A[1]
B[2,ind] = A[1]
B[myrange] = A[myrange]
```

Darüberhinaus gibt es die Möglichkeit Diagonalen oder Subdiagonalen zu extrahieren oder selektiv zu updaten.

```
B = diag(A)             // Spaltenvektor B enthält die
                        // Hauptdiagonale von A
B = diag(A,1)           // Spaltenvektor B enthält die
                        // 1. untere Subdiagonale von A
B = diagcopy(A)         // quadratisches B enthält die
                        // Hauptdiagonale von A
B = mksubdiag(s,diag(A,t))    // s.te subdiagonale des
                        //quadratischen B enthält die t.te
                        // subiagonale von A
```

Horizontales und vertikales Verketten wurde oben bereits beschrieben. Daher können alle alten Funktionen wie

```
collectZl(),collectSp(),
addZl(),addSp()
extractZl(),extractSp()
merge()
```

etc.pp. entfallen und werden nur wegen der Kompatibilität mit alten Skripten weiter unterstützt.

Interaktion mit anderen Programmen

Clipboard

Die Anzeige einer Matrix kann ins Clipboard übernommen werden; dort liegt dann eine Tabulator-getrennte Form vor, die z.B. von Excel oder Winword als Tabellenformat erkannt wird. Die von MatMate verwendeten Dezimalpunkte werden dabei in die für diese Programme gültige Schreibweise mit Kommas umgewandelt. (Menü MATRIX/KOPIEREN…bzw Kontextmenü jeder Matrix) Darüberhinaus gibt es die Möglichkeit, eine Matrix per Formelbefehl in das Clipboard zu schreiben oder daraus zu lesen (csv-Format). Dies kann dann direk in Excel oder Word tabellenmäßig verwendet werden. (beachte: Systemeinstellung "Dezimalpunkt". Es kann aber auch MatMates Voreinstellung angepaßt werden)

Dateien

Matrizen können aus Dateien gelesen und geschrieben werden. Dies geht z.zt. im CSV-Format ("comma separated values"), im Binärformat und Hexadezimal-Textformat, das SPSS® verwendet (Binär: eine double-Speicherstelle pro Element, am Ende einer Zeile ein zusätzliches CrLF, pro Matrixzeile eine Dateizeile) sowie die Ladungsmatrizen meines interaktiven Faktoranalyse-Programms Inside-[R] (z.Zt. nur lesen).

gewöhnliche Probleme

- Einlesen von Matrizen aus Dateien funktioniert nicht

  Der Befehl

  ```
  c = readCSV("meineCSVdatei.txt")
  ```

  funktioniert nicht richtig; c wird nicht angezeigt.

  Mögliche Ursache(n):

  > Die Beispieldaten liegen im Unterverzeichnis "BEISPIELE" zum Programmverzeichnis. Durch einen Datei-Öffnen-Befehl wurde das aktuelle Verzeichnis geändert. MatMate findet dann diese Datei nicht. Geben Sie ggfs. in Ihren Skripten immer den vollständigen Verzeichnispfad an.

- Wie oben, aber c hat falsche Daten

  Mögliche Ursache(n):

  > Das Format der Daten wurde nicht mit MatMate angelegt und kann nicht gelesen werden. Überprüfen Sie die Trennersymbole zwischen den Feldern und die Dezimalzeichen. Fehlen führende Nullen bei Dezimalzahlen (z.B. .1234 kann nicht gelesen werden)?
  > Passen Sie die Optionen für CSV-Daten-einlesen ggfls an.

- Eine Matrix wird nicht angezeigt:

  Der Befehl

  ```
  c = A + B + sqrt(E)
  ```

  funktioniert nicht richtig; c wird nicht angezeigt.

  Mögliche Ursache (n):

  > Eine der Quellmatrizen existiert nicht. E ist nicht quadratisch. A, B und E haben nicht dieselbe Dimensionen. Prüfen Sie diese Möglichkeiten

Komplette Syntax

---------- Allg. Programmbefehle

Außer Formeln gibt es auch einige Befehle, die MatMate ausführen kann. Siehe die aktuellen Ergänzungen im Hilfe-Menü unter "Neues".

- Programmparameter setzen / anzeigen

  Setzt Optionen für das Kopieren in die Zwischenablage. (Anzeige der aktuell verfügbaren Optionen und deren Einstellungen durch SHOW)

```
set ccSchema="office" ccFeldweite=8
show
```

  Startwert für Zufallszahlengenerator setzen

```
set randomstart=41
```

  Die systemvariablen Pfad etc können wie normale Variablen gesetzt und abgefragt werden. Liste: siehe Hilfe/Neues

- Matrix in Datei ausgeben

  gibt die Matrix mFctr in der PCA-rotierten Version in die Betriebssystem-Datei "c:\stat\mmpca.txt" aus. Das Ausgabeformat kann z.B. von SPSS® gelesen werden (Format pbinhex)

```
Matwrite hex("c:\stat\MMPCA.txt") = rot(mFctr,"PCA")

MatWrite <typ>(<matname> [,<option>])=<formel oder vari-
ablenname>


•  Kommagetrenntes Format.
    Option:1 Zeichen Feldtrenner, 2.Zeichen Dezimalpunkt

MatWrite CSV("mymat.txt",feldweite,dezstellen,";,")=<expr>
MatWrite CSV("clip",feldweite,dezstellen,";,")=<expr>
                       // ins Clipboard schreiben

•  Hexadezimales Textformat.
   Interne Darstellung (Double) in 2 ASCII-
   Hexziffern(0..9,A-F) pro byte.
   16 Zeichen pro Feld,
   Zeilenvorschub am Ende
   Kann von SPSS® pbinhex gelesen werden

MatWrite HEX("mymat.hex")=<expr>

•  Binäres Internformat.
   Interne Darstellung (Double)
   8 Byte pro Feld,
   Zeilenvorschub am Ende
   Kann von SPSS® binär gelesen werden

MatWrite BIN("mymat.bin")=<expr>
```

- Formel berechnen

```
 myCorr = covar2corr(x*x'/N)
 myFactors= cholesky(myCorr)
```

- Matrizen anzeigen/ausblenden

```
hide a,myCorr
display myFactors

fach add               // Fach hinzufügen
fach 2                 // Fach 2 anzeigen
fach get a,b,c // die Matrizen a,b,c in das aktuelle Fach
                       // holen
```

- Faktorenmatrix mit Inside-R bzw "Animate" weiterbearbeiten

```
inside-r matrix
animate matrix
```

- Scatterplot anzeigen

```
Zeichne scatterplot= vektorx,vektory      // einfacher
Scatterplot
Zeichne scatterplot= vektorx,matrixy // überlagerter
Zeichne scatterplot= matrixx,matrixy // Scatterplot


Zeichne animate = faktormatrix // rotierende Vektorgrafik
```

- Demo-anzeigen

```
demo "demoskriptdatei"
```

- Matmate-datei als Batch ausführen

```
batch "batchdatei"
```

- Komplette Session speichern/einlesen

```
save "mysession.mms"
get "mysession.mmd"
```

- Programm beenden

```
quit
```

----------Formelzeile / Fortsetzungssymbol

Zeilen können beliebig lang sein (32 KB). Komplexe Befehle können aber auf mehrere Zeilen umgebrochen werden - dann muß an jeder fortgesetzten Zeile ein Fortsetzungszeichen (Leerzeichen-Unterstrich) folgen

```
Matwrite   _  // ein Kommentar darf noch folgen
       hex("c:\statistik\MMPCA.txt")=  _
                rot(myFactors,"PCA")
```

----------Kommentarsymbol

Dem Programmmcode können Kommentare hinzugefügt werden. Hierzu gibt es ggw. zwei Symbole: das Semikolon, der Doppelschrägstrich. Eine Zeile, die mit Semikolon beginnt, wird nicht bearbeitet. Da ds Semikolon aber ggfls für Listenaufzählungen verwendet werden wird, muß ein anderes Symbol für einen Kommentar, der die Zeile abschließt, verwendet werden: der Doppelschrägstrich

```
 a=b+c // so kann ein Kommentar am zeilenende angefügt
          // werden
; dieser Kommentar ist mit einem Semikolon eingeleitet
;h dieser Kommentar wird im Jornal FETT angezeigt
;         (überschrift)
;i dieser Kommentar wird im Journal kursiv angezeigt
;t dieser Kommentar wird im Journal als normaler Text
;t  angezeigt
```

Formelnumerierung

Zeilen können mit [nummer] beginnen; diese Information wird nur im Listing gespiegelt.

```
[1] b = inv(A)
```

----------Namen

Namen in MatMate können bis zu 64 Zeichen lang sein, und nach einem Buchstaben auch Zahlen und Unterstriche enthalten. Groß- und Kleinschreibung wird nicht unterschieden; sie können also Ihre Matrizen- und Variablennamen schönschreiben. Namen können sogar mit Funktionsaufrufen gleichlauten.

> Dies wird aber nicht empfohlen, da eventuell in einer späteren Version mit differenzierteren Variablenzugriffen evtl das gegenwärtig einfache

Unterscheidungsmerkmal "Name (" = Funktion aber "Name " = Variablenname nicht mehr beibehalten werden kann.

---------- Parametertypen und ~konventionen

Die Typprüfung bei Funktionsaufrufen ist noch nicht vollständig; d.h. eine Funktion versucht womöglich eine Operation durchzuführen, die auf den aktuellen Parameter nicht anwendbar ist und bricht die laufende Operation (-sgruppe) mit einer Programmfehlermeldung ab (das Programm selbst wird dabei nicht abgebrochen).

Die Hilfefunktion, die Sie nach drücken der rechten Maustaste erhalten, gibt Kurzhinweise mit Parametertypen.

```
a=abs(G)                // die Absolut-Funktion nimmt
                        //numerische Werte skalar oder
                        // Matrizen
b= einh(4)
b= einh(a~)             // ein skalarer Wert oder eine
                        // Matrixdimension
c= collectsp(a,1..4,7..8)
                        // einer oder mehrere Bereiche
```

----------Zuweisungsoperator

Zuweisungsoperator ist das Gleichheitszeichen:

```
a= null(4,4)
```

Es können mehrere Zuweisungen gleichzeitig angefordert werden:

```
Zeilen,Spalten = 24,12
eigenvektor,eigenvalues = eigensys(A)
```

---------- Konstanten

Ein paar Basiskonstanten sind implementiert

```
#pi  // die Kreiszahl Pi
#e   // die Eulersche Konstante, Basis des natürlichen Log
#true #t #w             //logischen Wahr
#false #f               //logisches Falsch
#hi                     // höchster systeminterner Wert,
                        // für Matrixspalten/Zeilen ver
                        // wendet
b = a[10..#hi]          // von der 10. bis zur höchsten
                        // Zeile von a
```

Beachte die neuen Array-Operatoren:

```
x = a* #pi              // a wird mit Pi multipliziert
aber:
x = a *#pi              // es wird die *Variable* pi
                        // gesucht, um eine elementweise
                        // matrixmultiplikation durchzu-
                        // führen, da Operator *#
```

---------- Zahlen

Zahlen wie in jeder Programmiersprache. Zur Zeit werden alle Zahlen intern im Reellen Format (extended) bzw integer (32 bit) gespeichert; evtl. wird ein extra langzahl-Modul noch eingebaut. Wo Zahlen für Indizes oder Dimensionen etc verwendet werden, wird ein evtl vorhandener Nachkommateil abgeschnitten. Sollen Konstanten als Reelle typen gespeichert werden, muss die erste einen Dezimalpunkt enthalten:

```
123                     // Integerzahl
-123                    // negativ
0.345                   // relle Zahl, Null ist notwendig
123.456                 // relle Zahl
```

```
123.456E42              // relle zahl mal 10 hoch 42
0.123e-14               // relle Zahl mal 10 hoch -14
a = {1,2,3,4}           // integermatrix a
a = {1.0, 2,3,4}        // relle Matrix a
```

----------unäre Operatoren

Während Operationen wie a+b oder a*b als binär bezeichnet werden, (weil zwei Operanden beteiligt sind) bezeichnet man mit "unär" Operationen auf nur einem Operanden. Z.B. sind die Vorzeichen vor Zahlen (oder Variablenbezügen) unäre Operatoren, da sie sich nur auf den nachstehenden Operanden beziehen

```
 a = -12                // unäres Minus
 b = -a                 // wie vor
 c = a * -b             // vor dem Malnehmen wir b negiert
 x = c^-3               // vor dem Potenzieren wird ne-
giert

 f = NOT w              // Negation (wenn w boolesch ist)

 a = 5 !                // fakultätsfunktion
 a = 4.5 !              // wie a=gamma(5.5)

 b=a³                   // 3. Potenz, skalar oder Matrix
 b=a²                   // 2. Potenz, skalar oder Matrix
 b=a°                   // 0. Potenz, skalar oder Matrix
                        // bei Matrix: gibt Einheitsmatrix
                        // der Dimension von a

; # - Array-Operator
 a = sqrt(b#)           // matrix b als array behandeln:
                        // wurzeln aus elementen ziehen,
                        // nicht die Matrixwurzel

; ~ - Like/Dimension-Operator liefert "zeilen,spalten"
 a = null(b~)           // Nullmatrix mit gleicher Dimen-
                        //   sion wie b erstellen

; ' - transposition einer Matrix
 b = a'

; *' - Multiplikation a mit a'
 b = a *'               // b = a * a'
```

----------binäre Operatoren

Hier sind die üblichen Rechenoperatoren vorhanden, sowie eine Zahl an Operatoren für Logik, Matrizen und Strings. Als Operand kann statt einer Konstanten (Zahl etc) oder Variable (Skalar, Matrix) auch wiederum ein Ausdruck stehen, der sinnvollerweise geklammert wird, wenn die implizite arithmetische Prioritätenregelung nicht ausreicht.

```
 c = a + b              // addition
 c = a - b              // subtraktion


 c = a * b              // Multiplikation,
                        // Matrizen: Matrixmultiplikation
 c = a *# b             // Elementweise Multiplikation

 c = a / b              // Division
                        // Matrizen: a * INV(B) mit
                        //  c*b = a
 c = a /# b             // Elementweise Division


 c = a ^ b              // Potenzierung
                        // b darf z.Zt nur skalar sein
 c = a ^# b             // Elementweise Potenzierung
                        // b kann vektor oder Matrix sein
; nur elementweise Operationen
 c = a div b            // Ganzzahldivision
```

```
c = a mod b           // Rest einer Ganzzahlsdivision
c = a over b          // Binomial: (a+b)!/(a!*b!)
c = a ovrmod b        // (a over b) mod b


c = a & b             // logisches UND
c = a | b             // logisches ODER
c = a NAND b          // logisches Nicht-UND
c = a NOR b           // logisches Nicht-Oder
c = a XOR b           // logisches exklusives oder


c = a || b            // Matrix,Strings:
                      //    Horizontale verkettung
c = a IN b            // prüft ob a im Range b liegt

c = a ++ b            // spezielle "Matrixaddition"
                      // experimentell
```

----------allgemeine Funktionen

Diese Funktionen kann man auf Skalare ebenso wie auf Matrizen anwenden. (Soll bei Matrizen die Funktion auf die Einzelelemente angewendet werden, muß die Matrix mit dem #-Arrayoperator in einen Array umgewandelt werden).

```
Typen:                G:skalar oder Matrix,
                      B:skalare boolean,
                      I:skalarer Integer
                      R: skalare reelle Zahl
```

- allgemeine Funktionen

```
;Bei Matrizen: Funktion auf Einzelelemente anwenden:
;z.B.:  c = funktion(G#)     // Array-Operator # ver-
wenden

c = summe(G,...)      // Summe der Argumente
c = produkt(G,...)    // Produkt der Argumente

c = abs(G)            // Absolutwert des Arguments
c = neg(G)            // negativwert des Arguments

c = sqr(G)            // Quadrierung
c = sqrt(G)           // Quadratwurzel
```

- Funktionen\Ganzzahl

```
c = int(G)            // Ganzzahlanteil des Arguments
c = frac(G)           // Dezimalanteil des Arguments
```

- //Funktionen\Trigonometrie (bei Matrizen durch Taylorserie berechnet).

```
c = ln(G)             // natürlicher Logarithmus
c = exp(G)            // natürliche Exponentiation

c = sinh(G)           // hyperbolischer Sinus eines Bo-
genmaßes
c = cosh(G)           // hyperbolischer Cosinus eines
Bogenmaßes

c = sin(G)            // Sinus eines Bogenmaßes
c = cos(G)            // Cosinus eines Bogenmaßes
c = tan(G)            // Tangens eines Bogenmaßes
c = arcsin(G)         // Bogenlänge eines Sinus
c = arccos(G)         // Bogenlänge eines Cosinus
c = arctan(G)         // Bogenlänge eines Tangens
```

Statt eines Matrizennamens oder einer Zahlkonstanten kann natürlich auch ein kompletter Ausdruck stehen, insbesondere können die Funktionen auch geschachtelt auftreten.

```
c = exp ( ln(a+b)+1)
```

----------skalare Funktionen

Diese Funktionen können nur auf Skalare angewendet werden

```
 c = not(B)             // boolsche Negation auch für
                        //   Integer-Zahlen
 c = Fak(R)             // Fakultät-Funktion / Gamma-
Funktion
 c = PrimNr(I)          // die i.te Primzahl
(PrimNr(1)=2),i<6345
```

----------Matrixfunktionen

Die Liste der verfügbaren Funktionen ist je nach Entwicklungsintensität starken Veränderungen unterworfen. Sehen Sie bitte im Kontextmenü (rechte Maustaste) sowie unter "Hilfe/neues" nach.

; Funktionen\Matrix\Erzeugen

; Matrizen aus konstanten Werten erzeugen

```
 c = {wert, wert...}
        // 1-zeilige Matrix erzeugen
 c = { {wert,wert...},mat,mat ...}
        // mehrzeilige Matrix erzeugen
 c = Mk(iZei,iSpa,r,...)
        // Matrix erzeugen (zeilen,spalten,werte...
 c = MkSp(r [,M [,...)
        // Einspaltige Matrix erzeugen (wer-
te,matrizen,...)
 c = MkZl(r [,M [,...)
        // Einzeilige Matrix erzeugen (werte,matrizen...)
 c = MkCorr(r,...)
        // symmetrische M aus oberes Dreieck,sequentiell
 c = MkDiag(r [,M[,...)
        // Diagonalmatrix aus Vektor erzeugen (wer-
te,matrizen...)
```

; Matrizen mit bestimmten Strukturen erzeugen

```
 c = null(Izl,Ispa)
        // Erzeugt Null-Matrix
 c = einh(Idim)
        // Erzeugt Einheitsmatrix
 c = fill(Izl,Ispa,Rwert)
        // Erzeugt Matrix mit einem bestimmten Wert
 c = Seq(rAnf,rEnd [,Istep])
        // Erzeugt Matrix mit einer Sequenz von Werten
 c = Comb(iSpa,iUG,iOG)
        // Erzeugt eine kombinatorische Matrix
        // (spalten, basis=OG-UG ->Basis^Spalten)
 c = RandomN(iZei,iSpa,rMean,rStddev)
        // Matrix mit normalverteilten Zufallszahlen,
zeilenweise
 c = RandomU(iZei,iSpa [,rUG,rOG])
        // Matrix mit uniform verteilten Zufallszahlen,
        // zeilenweise
```

; Matrizen einlesen aus Dateien

```
 c = readHFS(Sdateiname)
        // Matrix aus einer Faktoranalyse-Datei(INSIDE-R)
einlesen
 c = readCSV(Sdateiname)
        // Matrix aus einer Comma-delimited-Datei einle-
sen
 c = ReadHEX(Sdateiname)
        // Matrix aus einer Hex-Binärdatei einlesen
 c = ReadBIN(Sdateiname,iAnz)
        // Matrix aus einer Binärdatei einlesen,
        // iAnz=Anzahl der Spalten
```

; Funktionen\Matrix\Allg

```
•  ; Ergebnis skalare Zahlenwerte:
 c = Zeilen(M)      // Zeilenzahl einer Matrix
 c = Spalten(M)     // Spaltenzahl einer Matrix

 c = Max(M)         // Größter Wert in Matrix
 c = Min(M)         // Kleinster Wert in Matrix

•  ; Ergebnis: Matrizen
 c = Transp(M)      // Transponierte
 c = Ar(M)          // Verwenden einer Matrix als Array
```

; Funktionen\Matrix\Submatrizen

```
 c = Reshape(M,Izl,Isp)
          // Zeilen und Spalten neu dimensionieren
 c = Rollv(M,Icount)
          // Matrixzeilen vertikal rollen

 c = Minor(M,Izl,Isp)      // Minor extrahieren


 c = Diag(MQ)              // Diagonale extrahieren
          // und als 1-sp Vektor ausgeben
 c = DiagCopy(MQ)          // Diagonale in Matrix gleicher
          // Größe kopieren
```

; Funktionen\mehrere Matrizen

```
 c = SeqMult(M1,M2)
           // gewichtetes Addieren benachbarter
           //Elemente einer Matrix (M2:Gewichtsmatrix)
```

; Funktionen\Matrix\Zeilen/Spalten verrechnen

- Ergebnis sind Spalten/Zeilen-Vektoren mit einer Auswertung über die Zeilen/Spalten/komplett einer Matrix

```
// Min/Max berechnen
 c = MinZl(M)           // für jede Zeile
 c = MinSp(M)           // für jede Spalte
 c = Min(M)             // für alle Elemente

// im folgenden jeweils ähnlich Zl,Sp oder Matrix:

 c = Max..(M)           // Maximum
 c = AMin..(M)          // Minimaler Absolutwert
 c = AMax..(M)          // Maximaler Absolutwert

// Spalten-/Zeilenindizes
 c = iMin..(M)          // Index des Minimumwertes
 c = iMax..(M)          // Index des Maximum
 c = iAMin..(M)         // Index des Minimaler Absolutwert
 c = iAMax..(M)         // Index des Maximaler Absolutwert

 c = Sum..(M)           // Summe(n)
 c = Mean..(M)          // Mittelwert(e)
 c = Median..(M)        // Medianwert(e) interpoliert

 c = Sqsum..(M)         // summe der Quadrate
 c = Stddev..(M)        // Standardabweichungen...

 c = Prod..(M)          // Produkt der Werte

; Statistische Tabellen Auswertungen
 c = freq(M)            // Häufigkeitstabelle, M wird nur
                        // Zeilenweise ausgewertet
 c = st_ChiSq           // chisquare einer Kreuztabelle
 c = st_CramersV        // Cramer's V einer Kreuztabelle
```

- Ergebnis sind Matrizen gleicher Größe mit ersetzten Werten

```
//Abweichungen vom Mittelwert (Mittelwert wird 0)
 c = Abw..(M)           //

//Z-Werte (Standardabweichung wird 1)
 c = zvalue..(M)        //

//Normwerte (Quadratsumme wird 1)
```

```
 c = norm..(M)          //

// Zeilen de-korrelieren
 c = UnkorrZl(M)        // Zeilen de-korrelieren
```

; Funktionen\Matrix\Quadratisch

```
 c = Inv(MQ)            // Inverse einer quadr. Matrix


 c = Det(MQ)            // Determinante einer quadr. Ma-
trix
 c = Spur(MQ)           // Spur einer quadr. Matrix
```

; Funktionen\Matrix\Distanz

```
 c = Dist(M1 [,M2'][,iNorm])
                        // Matrix der (Zeilen-)Distanzen
                        // (zu Norm deflt 2)
```

; Funktionen\Matrix\Kovarianz

; Funktionen\Matrix\Kovarianz\Faktorisieren

```
 c = GS(MCOV)           // Cholesky-Zerlegung
 c = Cholesky(MCOV)     // Cholesky-Zerlegung
 c = Decompose(MCOV)    // Vollständige Dekomposition
 c = SQRT(MCOV)         // auch hiermit kann man
                        // faktorisieren
```

; Funktionen\Matrix\Kovarianz\Kommunalitätenschätzung

```
 c = GetIndiv(MCOV)     // 1-Multiple-r² für jede Variable
bestimmen
 c = ComVar(MCOV)       // Common Varianz per dyn. Approx.
 c = ComVarPCA(MCOV)    // Common Varianz per PCA Approx.
 c = ComVarFix(MCOV)    // Common Varianz per fix. Approx.
 c = ComVarPaf(MCOV,IFanz)    // Common Varianz per PAF
                        // Approx.;ANzahl der Faktoren
```

; Funktionen\Matrix\Kovarianz\Rotationen

```
;  c = Rot(LAD,Srottyp)
                        // Spaltenrotationen einer Matrix
                        // (S=Typauswahl)
 c = Rot(LAD,"Tri")
 c = Rot(LAD,"PCA")
 c = Rot(LAD,"Quartimax")
 c = Rot(LAD,"Medimax")
```

; Funktionen\Matrix\Kovarianz\Sonstiges

```
 c = CovToCorr(MCOV)
          // Kovarianz in Korrelation umwandeln
 c = RemVar(MCOV,<ivarnr>)
          // Faktor 1-er Variabler extrahieren
```

; Funktionen\Matrix\R2/R3-Funktionen

```
;Raum-2 wie Komplex (a,b)=a+bi


 c = R2_Add(V2)         // R2-Addition
 c = R2_Sub(V2)         // R2-Subtraktion
 c = R2_Mul(V2)         // R2-Multiplikation
 c = R2_Div(V2)         // R2-Division
 c = R2_ToR3(V2)        // R2 nach R3 Umwandlung

;Raum-3 (a,b,c)= a+b*p+c*q (p,q komplexe 3.Wurzeln aus 1)

 c = R3_Add(V3)         // R3-Addition
 c = R3_Sub(V3)         // R3-Subtraktion
 c = R3_Mul(V3)         // R3-Multiplikation
 c = R3_Div(V3)         // R3-Division
 c = R3_ToR2(V3)        // R3 nach R2 Umwandlung
 c = R3_Norm(V3)        // R3 normieren
```

Referenzen

---------- SPSS®-Formatbeschreibungen

von SPSS ausgeben (SPSS-Steuerbefehle):

- CSV-Format

```
SAVE TRANSLATE
  OUTFILE='FZA_FAC3.csv'
  /TYPE=TAB
  /KEEP S1a188 to s1a207 fak1 fak2 fak3
  .
```

- HEX-Format (für MatMate empfohlen)

```
write cases
  outfile ='f:\temp\simplex\bihex.txt'
  records=1
  /1 f1 to f6 (6rbhex8).
```

- BIN-Format

```
write cases
   outfile="f:\temp\decomp8_SPSS®1.bin"
   records=1
   /1 f1 to f6 (6rb8) .
```

nach SPSS einlesen (SPSS-Steuerbefehle):

- BIN-Format

```
file handle dat2 name="f:\temp\decomp8.bin"
 /recform=fixed
 /lrecl=50
 /mode=image.

data list file=dat2
 /1 f1 to f6 (6rb8) nl (a2).

execute.
```

- HEX-Format (für MatMate empfohlen)

```
data list
  file ='f:\temp\simplex\bihex.txt'
  records=1
  /1 f1 to f6 (6rbhex8).
```

---------- Einlesen aus Inside-[R]

HFS-Files der Inside-[R]-Versionen bis 3.6 können direkt in MatMate eingelesen und von Matmate geschrieben werden. Zeilenbeschränkung bein Inside-R!

---------- *Programmierumgebung für MatMate:*

Borland-Pascal Delphi Version 6. Entwicklerhandbuch

Richedit-Komponente von JEDI.ORG

WIndows 98 und Windows XP

---------- *Copyrights und Markennamen*