Einführung in MATHEMATICA

G. Reents
Institut für Theoretische Physik
Universität Würzburg

Inhalt:

1. Simple Arithmetik 1
2. Die wichtigsten Funktionen 2
3. Graphik I 3
4. Variable und selbstdefinierte Funktionen 5
5. Listen 7
6. Einlesen, Speichern und 'Fitten' von Daten 8
7. Erste MATHEMATICA–Programme 11
8. Schleifen, 'If'-Abfragen, Rekursive Definitionen 11
9. init.m und Protokoll 13
10. Differentiation, Integration, Taylorreihen 13
11. Vektoren, Matrizen, Eigenwerte 15
12. Gleichungen lösen 16
13. Benutzung von 'Packages' 17
14. Graphik II 18
15. Graphik III , Interpolation 19
16. Muster, Ordnen, Sortieren 20
17. Unterprogramme, Pure Functions, Spezielles 21

Stichwortverzeichnis 23

Das Kommando *math* startet eine *MATHEMATICA*-Sitzung

In[..]:= Quit beendet die Sitzung

1 Simple Arithmetik

```
In[1]:= 4 + 7
Out[1]= 11
In[2]:= 3  4.2                    Leerstelle = Multiplikation
Out[2]= 12.6                      Dezimalzahlen mit Dezimalpunkt
In[3]:= 2*3*4                     Auch * = Multiplikation
Out[3]= 24
In[4]:= 24/8                      / = Division
Out[4]= 3
In[5]:= 2^3                       ^ = Potenz
Out[5]= 8
In[6]:= (3+4)^2 - 2 (3+1)
Out[6]= 41
In[7]:= % / 5                     % = das letzte Resultat
Out[7]= 41/5
In[8]:= N[%]                      N[x] = numerischer Wert von x
Out[8]= 8.2
In[9]:= 22.4 / (6.023 10^23)
Out[9]= 3.71908 10^-23
In[10]:= 3^-2*36+4                'Potenz' vor 'Punkt' vor 'Strich'
Out[10]= 8
```

2 Die wichtigsten Funktionen

```
In[11]:= Sqrt[16]
```
Sqrt[x] = *Wurzel von x*

```
Out[11]= 4
```

```
In[12]:= N[ Pi/2 ]
```
Pi = *MATHEMATICA - Form von* π

```
Out[12]= 1.5708
```

```
In[13]:= Sin[%]
```
Sin[x] = $\sin x$

```
Out[13]= 1.
```
1. = 1.0

> Alle in *MATHEMATICA* vorhandenen Funktionen, Prozeduren,
> Konstanten, ... beginnen mit einem Großbuchstaben.
> Funktionsargumente stehen jeweils in eckigen Klammern [ ] .

```
In[14]:= Cos[Pi/4]
```

```
Out[14]= Sqrt[2]/2
```
Unterschied: exakte und numerische Werte

```
In[15]:= Exp[0.5]
```
Exp[x] = *Exponentialfunktion von x*

```
Out[15]= 1.64872
```

```
In[16]:= Ln[%]
```

```
Out[16]= Ln[1.64872]
```

```
In[17]:= ?L*
```

> Hilfen in *MATHEMATICA* erhält man durch '?'
> In[..]:= ?* ⟶ 'Alles' wird aufgelistet.
> In[..]:= ?L* ⟶ Alle Funktionen, die mit L beginnen.
> In[..]:= ?Log ⟶ Informationen zu Log[x].
> In[..]:= ??Log ⟶ Mehr Informationen zu Log[x].

```
In[17]:= ?Ln
Global`Ln
```
Das Symbol 'Ln' wurde schon benutzt

```
In[17]:= ?Log
Log[z] gives the natural logarithm of z (logarithm to base E). ...
```

```
In[17]:= Log[ Out[15] ]
```
Alle während einer Sitzung erzeugten In[n] *und* Out[n] *sind wieder aufrufbar*

```
Out[17]= 0.5
```

```
In[18]:= Sqrt[-1]

Out[18]= I

In[19]:= ?I
I represents the imaginary unit Sqrt[-1]

In[19]:= Log[1 + 2 I]

Out[19]= Log[1 + 2 I]

In[20]:= Log[1. + 2 I]

Out[20]= 0.804719 + 1.10715 I

In[21]:= Sqrt[5.] // Log          Ist äquivalent zu Log[ Sqrt[5.] ]

Out[21]= 0.804719

In[22]:= ?Arc*
ArcCos ArcCot ArcCsc ArcSec ArcSech ArcSin ArcSinh ArcTan ArcTanh
ArcCosh ArcCoth ArcCsch

In[22]:= ArcTan[2.]

Out[22]= 1.10715

In[23]:= Re[ Out[20] ]            Re[z] = Realteil von z

Out[23]= 0.804719

In[24]:= Im[ %20 ]                Im[z] = Imaginärteil von z ;  %n = Out[n]

Out[24]= 1.10715
```

3 Graphik I

```
In[..]:= Plot[Sin[x],{x,0,2 Pi}]

In[..]:= Plot[Cos[t],{t,-5,5}]

In[..]:= Plot[ { Sin[x], x, x - x^3/6 }, {x,0,Pi}]
                                  Alle eingefügten Leerzeichen sind optional

In[..]:= Plot[{Sin[x],x,x-x^3/6},{x,0,Pi}, PlotRange -> {-3,3}]

In[..]:= Plot[{Sin[x],x,x-x^3/6},{x,0,Pi}, PlotRange -> {-3,3},
                                           AspectRatio -> Automatic]
```

```
In[..]:= ?Bessel*
BesselI BesselJ BesselK BesselY

In[..]:= ?BesselJ
BesselJ[n, z] gives the Bessel function of the first kind J(n, z).

In[..]:= Plot[{BesselJ[0,x],BesselJ[1,x],BesselJ[2,x]},{x,0,10}]

In[..]:= Plot[Mod[x,1],{x,0,4}]                Mod[x,y]  = x modulo y

In[..]:= Plot[Sign[Mod[x,1] - 0.5],{x,0,4}]    Sign[x]  = Vorzeichen von x
```

> Die wichtigsten Plot-Befehle:
> `Plot[f, {x, xmin, xmax}]`
> `Plot3D[f, {x, xmin, xmax}, {y, ymin, ymax}]`
> `ParametricPlot[{fx, fy}, {t, tmin, tmax}]` und `ParametricPlot3D[...]`
> `ContourPlot[f, {x, xmin, xmax}, {y, ymin, ymax}]`

```
In[..]:= Plot3D[ BesselJ[0,Sqrt[x^2+y^2]], {x,-9,9}, {y,-9,9} ]

In[..]:= Plot3D[ BesselJ[0,Sqrt[x^2+y^2]], {x,-9,9}, {y,-9,9},
                                                  PlotPoints -> 40 ]

In[..]:= ParametricPlot[ {Sin[5t], Sin[6t]}, {t,0,2Pi} ]

In[..]:= ParametricPlot[ {Log[1+t] Cos[t], Log[1+t] Sin[t]}, {t,0,8Pi},
                                            AspectRatio -> Automatic ]

In[..]:= ParametricPlot3D[ { Cos[t], Sin[t], t/6 }, {t,0, 6Pi} ]

In[..]:= ParametricPlot3D[ {Sin[t] Cos[u], Sin[t] Sin[u], Cos[t]},
                                                {t,0,Pi}, {u,0,2Pi} ]

In[..]:= ContourPlot[ 2 Exp[-(x^2+y^2)/2] + Exp[-(x-3)^2 -y^2],
                                    {x,-3,5}, {y,-4,4}, PlotPoints ->60]

In[..]:= Options[Plot]   -->   Liste von Optionen zu Plot[..]

In[..]:= Show[%% , ContourSmoothing -> True]
```

`Show[...]` *zeigt Graphikobjekte mit den entsprechenden Optionen*

> *Aufgabe :*
> *Überlegen Sie sich eine Parametrisierung von zwei sich berührenden Kugeln, und erzeugen Sie mit* `ParametricPlot3D[...]` *davon ein 3-dimensionales Bild.*

4 Variable und selbstdefinierte Funktionen

```
In[..]:= n = 40!
```
`k!` = `Factorial[k]` = *k - Fakultät*

```
Out[..]= 815915283247897734345611269596115894272000000000
```

```
In[..]:= N[n]
```

```
Out[..]= 8.15915 10^47
```

```
In[..]:= a = % / 10^47
```

```
Out[..]= 8.15915
```

```
In[..]:= N[a,14]
```
Der numerische Wert von a mit 14 Stellen

```
Out[..]= 8.159152832479
```

```
In[..]:= N[a,16]
```

```
Out[..]= 8.15915283247898
```

```
In[..]:= N[a,20]
```

```
Out[..]= 8.15915283247898
```
Bei 16 ist Schluß. Falls nicht anders spezifiziert, werden Zahlen mit Dezimalpunkt intern mit 16 Stellen verarbeitet

```
In[..]:= N[Pi,40]
```

```
Out[..]= 3.1415926535897932384626433832795028841972
```
π *mit 40 Stellen*

```
In[..]:= Precision[Pi]
```

```
Out[..]= Infinity
```
`Infinity` = + *Unendlich*

```
In[..]:= Precision[1.0]
```

```
Out[..]= 16
```
Eventuell anders bei einem anderen Computertyp

```
In[..]:= x = a n
```

```
Out[..]= 6.65718 10^48
```

```
In[..]:= Clear[a,n,x]
```
`Clear[a,n,...]` *löscht die Belegung der Variablen* `a,n,...`

```
In[..]:= q = (a + b)^3
```

```
Out[..]= (a + b)^3
```

```
In[..]:= q
```

```
Out[..]= (a + b)^3
```

```
In[..]:= q = Expand[q]
```
`Expand[..]` *multipliziert aus*

```
Out[..]= a^3 + 3 a^2 b + 3 a b^2 + b^3
```

```
In[..]:= Factor[q]                    Factor[..] faktorisiert
```

```
In[..]:= b = -1
```

```
In[..]:= q                            ⟶  -1 + 3 a - 3 a^2 + a^3
```

```
In[..]:= Plot[ q, {a,.5,1.5} ]        Auch solche Ausdrücke lassen sich plotten
```

Definition von Funktionen

```
In[..]:= ?@
```

```
Es ist zu empfehlen, in Variablennamen keine Großbuchstaben zu verwenden.
In[..]:= ?@  zeigt alle benutzten Symbole, die keine Großbuchstaben
                                                          enthalten.
In[..]:= Clear["@"]  löscht die Belegung aller dieser Symbole.
In[..]:= Remove["@"]  'beseitigt' alle diese Variablen.
In[..]:= Clear["Global`*"]  löscht die Belegung von allen
                            (benutzerdefinierten) Variablen.
```

```
In[..]:= Clear["@"]
```

```
In[..]:= f[x_] = Exp[-0.25 x^2] Cos[4.5 x]
```

x_ ist ein 'Platzhalter' mit dem Namen x. Der Unterstreichungsstrich bei x_ ist wichtig. Er darf nur auf der linken Seite der Gleichung auftauchen.

```
In[..]:= f[2]                         ⟶  -0.335186
```

```
In[..]:= Plot[ f[x], {x,-Pi,Pi} ]
```

```
In[..]:= fermi[e_,b_]:= 1/(Exp[ b (e-1) ] + 1)
```

```
In[..]:= Plot[ { fermi[e,8], fermi[e,16], fermi[e,Infinity] }, {e,0,1.7} ]
```

Aufgabe :
Die spektrale Verteilung im Planckschen Strahlungsgesetz ist durch

$$u(\nu, T) = \frac{8\pi h\nu^3}{c^3} \frac{1}{e^{\frac{h\nu}{kT}} - 1}$$

gegeben. Plotten Sie eine Kurvenschar $\{ u(\nu, T_1), u(\nu, T_2), u(\nu, T_3) \}$ *, an der das Wiensche Verschiebungsgesetz deutlich wird.*

5 Listen

```
In[..]:= l1 = {2,3,4}
```
Eine Liste von 3 Zahlen. Listen werden mit mit geschweiften Klammern geschrieben.

```
In[..]:= Head[l1]
```
⟶ `List` ; *jedes Ding hat einen Kopf.* `l1` *ist eine Liste.*

```
In[..]:= lq = l1^2
```
⟶ `{4, 9, 16}` ; *fast alle Funktionen haben das Attribut* `Listable` .

```
In[..]:= Log[l1] // N
```
⟶ *Liste der Logarithmen*

```
In[..]:= Attributes[Log]
```
⟶ { `Listable, Protected` }

```
In[..]:= Table[ 2, {10} ]
```
`Table[..]` *macht Listen*

```
In[..]:= Table[ i^3, {i,5} ]
```
⟶ `{1, 8, 27, 64, 125}`

```
In[..]:= Table[ 1/j, {j,3,8} ]
```
⟶ $\{\frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}, \frac{1}{7}, \frac{1}{8}\}$

```
In[..]:= Table[ x, {x,0,2,0.25} ]
```
⟶ `{0, 0.25, 0.5, ..., 2.}`

> `Table[ expr , {...} ]` erzeugt Listen. Die geschweifte Klammer mit der allgemeinen Form `{i, imin, imax, di}` heißt 'iterator'. Verkürzungen der allgemeinen Form sind möglich.

```
In[..]:= Range[0, 2, 0.25]
```
⟶ `{0, 0.25, 0.5, ..., 2.}` ; *Listen mit äquidistanten Zahlen auch mit* `Range[...]`.

```
In[..]:= liste = Table[ {x, Gamma[x]}, {x, 2, 4, 0.05} ];
```

> Ein Semikolon ";" nach einem Ausdruck unterdrückt das nächste `Out[..]`. Die Variablen haben dennoch die zugewiesenen Werte.

```
In[..]:= Short[liste]
```
⟶ *Kurzform von* `liste`

```
In[..]:= ListPlot[liste]
```
`ListPlot[...]` *plottet Daten*

```
In[..]:= letters = Table[ FromCharacterCode[j], {j,122,97,-1} ]

Out[..]= {z, y, x, w, ...}
```

```
In[..]:= Sort[letters]
```
`Sort[..]` *sortiert; auch Zahlen.*

```
In[..]:= letters[[3]]
```
⟶ `x` , *das 3. Element der Liste* `letters`. *Teile von Listen oder Ausdrücken mit doppelten eckigen Klammern* `[[..]]` .

```
In[..]:= Length[ liste ]          ⟶ 41 ;  die Länge der Liste liste
In[..]:= liste[[20]]              ⟶ {2.95, 1.91077}
In[..]:= liste[[20,1]]            ⟶ 2.95
In[..]:= liste[[20,2]]            ⟶ 1.91077
In[..]:= abc = Reverse[ letters ]          ⟶ {a, b, c, ... }
In[..]:= abc[[{2,4,6}]]           ⟶ {b, d, f}
In[..]:= abc[[-1]]                ⟶ z ;  negativer Index = Zählung von hinten
In[..]:= Permutations[ {1,2,3} ]  ⟶ alle Permutationen von {1,2,3}
```

Aufgabe :
Erzeugen Sie für $n = 1$ *bis* 100 *Listen mit* $\ln n!$ *, mit der Näherung aus der Stirling-Formel* $\ln\left(\sqrt{2\pi n}\frac{n^n}{e^n}\right)$ *, Listen der Differenzen, der relativen und der prozentualen Abweichungen. Plotten Sie alle Ergebnisse mit* `ListPlot[..]` .

6 Einlesen, Speichern und 'Fitten' von Daten

```
In[..]:= Remove["@"]              'Beseitigt' alle Variablen ohne Großbuchstaben
In[..]:= f[x_]:= 1.0 x^4 - 2.0 x^2 + 0.5
In[..]:= ?f                       ⟶ Definition von f[x]
In[..]:= p1 = Plot[ f[x], {x,-1.5,1.5} ]
In[..]:= r:= Random[ Real, {-0.1,0.1} ]
```

`Random[]` *(ohne Argument) liefert eine reelle Zufallszahl zwischen 0 und 1.* `Random[type,range]` *ist die allgemeinere Form.*

```
In[..]:= daten = Table[ {x, f[x]+r}, {x, -1.5, 1.5, 0.05} ]
In[..]:= p2 = ListPlot[daten]
In[..]:= Show[p1,p2]              Show[..] kann Graphikobjekte kombinieren
In[..]:= g[x_] = Fit[daten, { 1, x, x^2, x^3, x^4}, x ]
                                  Fit[..] berechnet einen linearen 'least square' Fit
In[..]:= Plot[{f[x],g[x]},{x,-1.5,1.5}]
In[..]:= daten >> daten.dat       MATHEMATICA-Objekte lassen sich mit
                                  >> 'filename' speichern
```

```
In[..]:= ! ls
```

> " ! Kommando " bewirkt, daß das Kommando ans Betriebssystem geht.
> " !!filename " bringt den Inhalt von 'filename' auf den Bildschirm.

```
In[..]:= !! daten.dat

In[..]:= daten2 = << daten.dat                    <<  = Einlesen

In[..]:= daten == daten2                  ==  ist das Gleichheitszeichen in Gleichungen

Out[..]= True

In[..]:= Save["random.def", r]    MATHEMATICA-Definitionen lassen sich speichern
                                  mit  Save["filename", symb1, symb2, ... ] .

In[..]:= Remove[r]

In[..]:= << random.def            Auf diese Weise kann man ganze MATHEMATICA-
                                  Programme einlesen


In[..]:= Remove["@"]

In[..]:= f[x_] := Exp[0.5 - 1.0 x^2]

In[..]:= p1 = Plot[f[x],{x,-2,2}]

In[..]:= r := Random[Real, {-0.05, 0.05}]

In[..]:= SeedRandom[1771]                 setzt den Zufallszahlengenerator in
                                          einen definierten Anfangszustand

In[..]:= daten = Table[{x, Abs[f[x] + r]}, {x,-2,2,.02}];        Abs[x] = |x|

In[..]:= p2 = ListPlot[daten]

In[..]:= Show[p1,p2]

In[..]:= FortranForm[6.0916 10^-6]    ⟶ Out[..]//FortranForm= 6.0916e-6

In[..]:= paare = Map[FortranForm,daten,{2}]      wendet FortranForm auf daten
                                                 an, und zwar auf das zweite Level

In[..]:= liste = TableForm[paare]

In[..]:= OutputForm[liste] >> list.dat        list.dat könnte ein Datenfile sein,
                                              das bei einer Messung erzeugt wurde

In[..]:= data = ReadList["list.dat",{Number,Number}]
                                  ReadList[...] macht aus Daten eine MATHE-
                                  MATICA-Liste; hier eine Liste von Zahlenpaaren
```

```
In[..]:= logy[{x_, y_}] := {x, Log[y]}

In[..]:= logdata = Map[logy, data]
```

Die 2. Komponente aller Paare von `data` *wird logarithmiert*

```
In[..]:= linfit[x_] = Exp[ Fit[ logdata, {1, x^2 }, x ] ]

In[..]:= p3 = Plot[ linfit[x], {x,-2,2} ]

In[..]:= Show[p2,p3]
```

Schlechter Fit. Diese Methode wird im MATHEMATICA-Handbuch empfohlen

```
In[..]:= testf[x_]:= Exp[ a - b x^2 ]

In[..]:= x[i_]:= data[[i,1]] ; y[i_]:= data[[i,2]]
```

> Durch Semikolon (;) getrennt kann man mehrere MATHEMATICA-Befehle aneinanderfügen.

```
In[..]:= Length[data]
```

$\longrightarrow$ `201`

```
In[..]:= qsum = Sum[ (testf[ x[i] ] - y[i])^2, {i, 201} ] ;
```

`Sum[...]` *summiert. Iterator wie bei* `Table[...]`

```
In[..]:= Plot3D[ qsum, {a,.25,.75}, {b,.8,1.2}, ViewPoint -> { -1.3, 2.4, .3} ]

In[..]:= mini = FindMinimum[ qsum, {a, .5}, {b, 1.} ]

Out[..]= {0.160063, {a -> 0.504624, b -> 1.00775}}
```

Das Ergebnis enthält den minimalen Wert und eine Liste von 'Regeln' mit den Werten der Variablen am Minimum

```
In[..]:= regel = mini[[2]]
```

$\longrightarrow$ *Die Liste der Regeln*

```
In[..]:= goodfit[x_] = testf[x] /. regel
```

> `expr /. rules` wendet eine Regel oder eine Liste von Regeln auf `expr` an. " /. " ist die Kurzform von `ReplaceAll[..]`.

```
In[..]:= p4 = Plot[ goodfit[x], x,-2,2 ]

In[..]:= Show[p2,p4]
```

Aufgabe :
Sie finden in Ihrem home directory eine Datei mit dem Namen `testdaten.dat` .
Sie enthält x-y-Werte, die sich mit der Funktion $f(x) = \dfrac{a\,x^3}{e^{x/b} - 1}$ *fitten lassen.*
Verwenden Sie die hier beschriebene Methode, um die optimalen Parameter a und b zu finden.

7 Erste MATHEMATICA–Programme

`unix>` **vi** `null1.m` *(`unix>` ist der Systemprompt)*

```
(* null1.m     Uebung mit FindRoot *)

Clear[ "@" ]          (* So schreibt man Kommentare in MATHEMATICA *)
f[x_] := Tan[ x/4 ]
gl = f[x] == 1
r = FindRoot[ gl, {x, 3.1} ]
pi = x /. r
```

```
In[..]:= << null1.m
Out[..]= 3.14159
```

`FindRoot[...]` *findet numerische Lösungen von Gleichungen*

`unix>` **vi** `null2.m`

```
Clear[ "@"]
f[x_] := Sin[x] - x/2 ;        eq = f[x] == 0
zero[start_] := FindRoot[ eq, {x, start} ]
```

```
In[..]:= << null2.m
In[..]:= zero[.5]              ⟶  {x -> 0.}
In[..]:= zero[1.5]             ⟶  {x -> 1.89549}
In[..]:= zero[-1.5]            ⟶  {x -> -1.89549}
```

Aufgabe :
Schreiben Sie ein Programm, das beim Aufruf eine Liste der ersten 20 Nullstellen der Besselfunktion $J_0(x)$ ausgibt.

8 Schleifen, ‘ If ’-Abfragen, Rekursive Definitionen

```
In[..]:= Remove["@"]
In[..]:= Do[ Print[ " Hello world "], {10} ]
In[..]:= s1 = "   " ;  s2 = "   ASCII-Code    "        s1 und s2 sind Strings
In[..]:= Do[ Print[ s1, FromCharacterCode[j], s2, j ], {j, 65, 75} ]
In[..]:= For[ i=1, i < 11, i = i + 1, Print[i] ]
In[..]:= ?For
```

```
In[..]:= For[ i=1, i < 11, i++, Print[i, "   ", i^2] ]
```

```
Schleifen :
Do[ expr, { iterator } ]
For[ start, test, incr, body ]
While[ test, body ]
Nest[ f, expr, n ]
NestList[ f, expr, n ]
FixedPoint[ f, expr ]
```

```
In[..]:= test := (a = Random[]; a < 0.8)
In[..]:= test                    ⟶  True oder False
In[..]:= While[ test, Print[a] ]

In[..]:= Nest[ Sin, t, 3 ]       ⟶ Sin[Sin[Sin[t]]]
In[..]:= Nest[ Sin, 1.5, 30 ]
In[..]:= NestList[ Sin, 1.5, 30 ]

In[..]:= f[x_] := N[Cos[x], 20]
In[..]:= FixedPoint[ f, .100000000000000000]   ⟶  0.73908513321516064164
```

Die 'If'- Konstruktion

```
In[..]:= If[ Random[] < 0.5, Print[ " Zahl < 0.5 "] ]
In[..]:= beep:= Print["\^G"] ; beepbeep:= ( beep; Pause[1]; beep )
In[..]:= If[ Random[] < .5, beep, beepbeep ]
In[..]:= theta[x_]:= If[ x <= 0, 0, 1 ]
In[..]:= Plot[ theta[x], {x,-1,1} ]
```

Rekursive Definitionen

```
In[..]:= p[1] = 1. ; p[n_]:= p[n-1]*(365 - n)/365 ; w[n_]:= 1 - p[n]
In[..]:= Do[ Print["   w(", k, ") = ", Round[100*w[k] ], " %" ], {k, 2, 50} ]

In[..]:= pr[1] = 1.
In[..]:= pr[n_]:= pr[n] = pr[n-1]*(365 - n)/365
In[..]:= wr[n_]:= 1 - pr[n]
```

Diese Konstruktion bewirkt, daß die Funktion `pr[..]` *sich an alle vorher berechneten Werte "erinnert".*

```
In[..]:= Timing[ Table[ w[k], {k, 40, 50} ] ]
In[..]:= Timing[ Table[ wr[k], {k, 40, 50} ] ]
```

`Timing[..]` *liefert CPU-Zeit*

9 init.m und Protokoll

`unix>` **vi** `init.m` — *Das File* `init.m` *wird bei jedem Start von* `MATHEMATICA` *gelesen und ausgeführt*

```
<< /usr/local/mathematica/StartUp/init.m
<< protokoll.m
```

`unix>` **vi** `protokoll.m`

```
Print[" "] ; Print[" "]
answer = InputString[ "Protokollfile anlegen? (j,n) "]
If[ answer == "j", datum = Date[];
                   protoname = StringJoin["protokoll_",
                                          ToString[ datum[[3]] ],"_",
                                          ToString[ datum[[2]] ],".m"
                                         ];
                   AppendTo[ $Echo, OpenAppend[ protoname ] ];
                   Print[" "] ; Print[" "];
                   Print["  Protokollfile ","`", protoname, "'",
                          " geoeffnet" ];
                   AppendTo[ $Output, protoname ];
   ]
```

Aufgabe :
Erzeugen Sie ein File mit dem Namen `end.m` *. Es wird am Ende jeder MATHEMATICA-Sitzung nach dem* `Quit` *gelesen und ausgeführt. Der Inhalt sollte sein: eine Abfrage, ob das Protokollfile gelöscht werden kann,*
Ausführen von `DeleteFile[ protoname ]` *falls die Antwort* `"ja"` *ist.*

10 Differentiation, Integration, Taylorreihen

```
In[..]:= Clear["@"]
```

```
In[..]:= D[ Exp[ x^2 ], x ]                          −→  die Ableitung bezüglich x
```

```
In[..]:= Integrate[ 1/(x^4 - 1), x]                  −→  das unbestimmte Integral
```

```
In[..]:= D[ %, x ]
```

```
In[..]:= Simplify[ % ]
```

```
In[..]:= Integrate[ Exp[c*x], {x, a, b} ]
```

```
In[..]:= Integrate[ Log[x], {x, 1, 3} ]
```

```
In[..]:= Integrate[ Sin[ Sin[x] ], {x, 0, 1} ]   −→  ohne Erfolg
```

```
In[..]:= NIntegrate[ Sin[Sin[x]], {x, 0, 1} ]    −→  numerisch geht's
```

```
In[..]:= Integrate[ x*y*Exp[-(x^2 + y^2) ], {x,0,2}, {y,x,1} ]

In[..]:= Integrate[ 1/x^2, {x,-1,1} ]          ⟶  ?
```

```
unix> vi vol.m
```

```
(*  Das Volumen einer n-dimensionalen Kugel  *)

arg[n_]:= Table[ {x[i],0,Sqrt[r^2 - Sum[ x[j]^2, {j,i-1}] ]}, {i, n} ]
Unprotect[Power] ;  (r^2)^(1/2) = r ;  Protect[Power]
vol[n_]:= 2^n Apply[ Integrate, Join[ {1}, arg[n] ] ]

(* Apply[ operator, expr ]  ersetzt  Head[ expr ]  durch  operator *)
```

```
In[..]:= << vol.m

In[..]:= Join[ {1}, arg[3] ]                   ⟶  o.k. ?

In[..]:= vol[2]                                ⟶  Pi r^2  wäre richtig

In[..]:= Table[ vol[k], {k, 5} ]

In[..]:= reihe = Series[ Tan[x], {x, 0, 10} ]
```

```
Series[ expr, {x, x0, n} ]  erzeugt eine Taylorreihe von expr bzgl.
der Variablen x um den Punkt x0, und zwar bis zur Ordnung n.

Normal[..] macht aus dem Ergebnis von Series[..] ein Polynom.
```

```
In[..]:= Sum[ ( D[ Tan[a], {a,j} ] /. a -> 0 )*x^j / j!, {j,0,10} ]

In[..]:= Normal[ reihe ]

In[..]:= Timing[ r1 = Series[ Exp[ Sin[x] ], {x,0,50} ]; ]

In[..]:= Timing[ r2 = Series[ Exp[ Sin[1.0 x] ], {x,0,50} ]; ]
```

Falls ohnehin nur die numerischen Werte interessieren, sollte man das von vornherein berücksichtigen.

Aufgabe :
Erzeugen Sie von $f(x) = e^{\sin x}$ *eine Taylorreihe um* $x = 0$ *, die* $f(x)$ *für* $|x| \leq \pi$ *mit einer Genauigkeit von* 0.001 *approximiert.*

11 Vektoren, Matrizen, Eigenwerte

```
In[..]:= Clear["@"]

In[..]:= v = {x, y, z}

In[..]:= r = Table[ x[j], {j, 3} ]

In[..]:= r2 = Array[ x, 3 ]

In[..]:= m = Array[ a, {3, 3} ]

In[..]:=  MatrixForm[m]

In[..]:= b = Table[ 1/(5 + i - j), {i,3}, {j,3} ]

In[..]:= MatrixForm[b]

In[..]:= d = DiagonalMatrix[ {d1, d2, d3} ]

In[..]:= e = IdentityMatrix[3]

In[..]:= t = Transpose[b]

In[..]:= Det[b]

In[..]:= invb = Inverse[b]

In[..]:= v.v

In[..]:= r.v                    ⟶  Skalarprodukte

In[..]:= m.v                    v  wird hier als Spaltenvektor interpretiert,
In[..]:= v.m                    und hier als Zeilenvektor.

In[..]:= m.b                    Matrixprodukt m mal b

In[..]:= invb.b

In[..]:= Dimensions[v]          ⟶  {3}

In[..]:= Dimensions[m]          ⟶  {3,3}

In[..]:= Eigenvalues[b]

In[..]:= N[%]

In[..]:= Chop[%]

In[..]:= Eigenvalues[ N[b] ]
```

Aufgabe :
Konstruieren Sie eine tridiagonale $n \times n$-Matrix ($n = 2, \ldots, 10$) mit Hauptdiagonale einheitlich gleich 2.0 und Nebendiagonalen einheitlich gleich -1.0 . Bestimmen Sie die Eigenwerte $\lambda(n, j)$ ($j = 1, 2, \ldots, n$) dieser Matrix und bestätigen Sie, daß $\lambda(n, j)$ gegeben ist durch den numerischen Wert von

$$4 \sin^2 \frac{j\,\pi}{2(n+1)} \ , \quad j = 1, 2, \ldots, n \ .$$

Dabei ist angenommen, daß die Eigenwerte $\lambda(n, j)$ der Größe nach geordnet sind.

12 Gleichungen lösen

```
In[..]:= FindRoot[ Cos[x] == x, {x, 1} ]

In[..]:= FindRoot[ {Cos[a*x] == x, -a Sin[a*x] == -1}, {x,1}, {a,1} ]

In[..]:= Plot[20^x*NProduct[ j/(j + x), {j,20} ], {x,1,4} ]

In[..]:= FindRoot[20^x*NProduct[j/(j + x), {j,20}] == 10 , {x,3.5} ]
                                                    → Fehlermeldung

In[..]:= FindRoot[20^x*NProduct[j/(j + x), {j,20}] == 10 , {x,3,4} ]
```

```
FindRoot[lhs == rhs, {x, x0}]  findet numerische Lösungen von Gleichungen
Solve[eqns, vars]  versucht, exakte Lösungen von Gleichungen zu finden
NSolve[eqns, vars]  findet numerische Lösungen algebraischer Gleichungen
LinearSolve[m, b]  löst das Gleichungssystem  m.x == b
DSolve[eqn, y[x], x]  löst Differentialgleichungen
NDSolve[eqns, y, {x, xmin, xmax}]  findet numerische Lösungen von
Differentialgleichungen
```

```
In[..]:= Solve[ a x^2 + b x + c == 0, x ]

In[..]:= Solve[ ArcSin[x] == a, x]

In[..]:= Solve[ { Exp[x y] == 2, x + 2 y == 1 }, {x,y} ]

In[..]:= Solve[ x^5 - 5 x^2 + 1 == 0 ,x ]

In[..]:= NSolve[ x^5 - 5 x^2 + 1 == 0 ,x ]

In[..]:= DSolve[ y''[x] == y[x], y[x], x]

In[..]:= DSolve[ { y''[x] == y[x], y[0]==1,y'[0]==2 }, y[x], x]

In[..]:= f[x_] = y[x] /. %[[1]]

In[..]:= DSolve[ y''[x] == Cos[x]*y[x],y[x],x]

In[..]:= NDSolve[ { y''[x]==Cos[x]*y[x], y[0]==1, y'[0]==1 }, y[x], {x,0,3} ]

In[..]:= g[x_] = y[x] /. %[[1]]

In[..]:= Plot[ g[x], {x,0,3} ]
```

Aufgabe :
Lösen Sie die Differentialgleichung $x\,y' = y + x^2\cos x + \frac{1}{2}$, $y(\pi) = -\frac{1}{2}$ *einmal mit* `DSolve[..]` *und zum anderen mit* `NDSolve[..]` *für* $|x| \leq 5\pi$ *. Wie groß ist in diesem Bereich die Differenz zwischen numerischer und exakter Lösung.*

13 Benutzung von 'Packages'

```
In[..]:= Needs[ "Statistics`NonlinearFit`" ]

In[..]:= ?Nonl*

In[..]:= data = ReadList["testdaten.dat", {Real, Real}]

In[..]:= NonlinearFit[ data, a x^3 /(Exp[x/b] - 1), {x}, {a, b} ]

In[..]:= f[x_]= a x^3 /(Exp[x/b] - 1) /. %

In[..]:= f[0]=f[0.]=0

In[..]:= Show[ Plot[ f[x], {x,0,60}, DisplayFunction -> Identity ],
               ListPlot[ data, DisplayFunction -> Identity ],
               DisplayFunction -> $DisplayFunction ]

In[..]:= ??Random

In[..]:= Needs[ "Statistics`Master`" ]

In[..]:= ??Normal*

In[..]:= ??NormalDistribution

In[..]:= NormalDistribution

In[..]:= ??NormalDistribution

In[..]:= ??Random

In[..]:= Random[ NormalDistribution[0,1] ]

In[..]:= Timing[ Table[ Random[ NormalDistribution[0,1] ], {1000} ]; ]

In[..]:= gaussdata = Table[ Random[ NormalDistribution[0,1] ], {10000} ];

In[..]:= Short[ gaussdata ]

In[..]:= xmin = Min[ gaussdata ]

In[..]:= xmax = Max[ gaussdata ]

In[..]:= ListPlot[ gaussdata ]

In[..]:= BinCounts

In[..]:= ??BinCounts

In[..]:= dx = (xmax - xmin)/51

In[..]:= bins = BinCounts[ gaussdata, {xmin, xmax, dx} ]

In[..]:= Needs["Graphics`Graphics`"]

In[..]:= BarChart[ bins, PlotRange -> All ]
```

Aufgabe :
Nach dem Aufruf von `Needs["Graphics`Graphics`"]` *steht auch eine Prozedur* `GeneralizedBarChart[..]` *zur Verfügung. Informieren Sie sich über die Syntax dieser Prozedur und ergänzen Sie die Liste* `bins` *in geeigneter Weise, so daß Sie* `GeneralizedBarChart[..]` *anwenden können.*

14 Graphik II

```
unix> vi .cshrc
```

> *# Folgende Zeile bei den* alias*-Einträgen hinzufügen : #*
>
> ```
> alias   motifps   /usr/local/mathematica/Bin.HP9000S700/Display/motifps
> ```

```
unix> source .cshrc
```

```
In[..]:= pl1 = ParametricPlot3D[{ 2 Abs[Cos[a]]*Sin[a]*Cos[p],
                                  2 Abs[Cos[a]]*Sin[a]*Sin[p],
                                  2 Abs[Cos[a]]*Cos[a] }, {a,0,Pi},{p,0,2Pi} ]
```

> Display[" *filename*", *graphics*] schreibt die in *graphics* enthaltene Information in das File *filename* . Mit motifps *filename* bringt man sie wieder auf den Bildschirm.

```
In[..]:= Display[ "spheres.mathps", pl1 ]
```

```
unix> motifps spheres.mathps
```

```
In[..]:= Plot[ Evaluate[ Table[ LegendreP[n,x], {n,3} ] ], {x,-1,1},
               PlotStyle ->
               { RGBColor[1,0,0], RGBColor[0,1,0], RGBColor[0,0,1] } ]
```

```
In[..]:= Plot[Evaluate[Table[LegendreP[n, x], {n, 3}]], {x, -1, 1},
              Frame -> True, PlotStyle ->
       { Dashing[{}], Dashing[{0.003, 0.003}], Dashing[{0.006, 0.006}] }]
```

```
In[..]:= Plot[HermiteH[6,x], {x,-3,3}, AxesLabel->{"x","Hermite6[x]"} ]
```

```
In[..]:= Show[ %,Frame->True,FrameLabel->{"Hermite Polynom 6. Ordnung", ""} ]
```

```
In[..]:= Needs[ "Graphics`PlotField`" ]
```

```
In[..]:= ?Plot*
```

```
In[..]:= ?PlotGradientField
```

```
In[..]:= PlotGradientField[ x^2 -y^2, {x,-2,2}, {y,-2,2} ]
```

```
In[..]:= PlotVectorField[ { -y ,x }, {x,-2,2}, {y,-2,2} ]
```

> *Aufgabe :*
> *Plotten Sie das Magnetfeld, das von zwei geraden parallelen Leitern erzeugt wird. Die Ströme in den beiden Leitern sollen einmal gleich sein, das andere Mal entgegengesetzt gleich sein.*

15 Graphik III , Interpolation

```
In[..]:= Directory[]                    ⟶  das aktuelle directory
In[..]:= FileNames["*rofil*"]           ⟶  {profile.dat}
In[..]:= !! profile.dat                 Meßdaten des Profils einer Absorptionslinie
In[..]:= data = ReadList[ "profile.dat", {Real,Real} ]
In[..]:= pl1 = ListPlot[ data ]
In[..]:= ListPlot[ data, PlotJoined -> True ]
In[..]:= Show[ pl1, Graphics[ Line[ {{-4,10},{4,10}} ] ] ]
In[..]:= Show[ Graphics[ Line[ data ] ] ]
In[..]:= Show[ Graphics[ Line[ data ] ], Axes->True ]
In[..]:= ?Interpol*
In[..]:= ??Interpolation
In[..]:= inter = Interpolation[ data ]
In[..]:= pl2 = Plot[ inter[x] , {x,-4,4}, PlotRange->{0,10.5} ]
In[..]:= Plot[ Evaluate[ D[inter[x],x] ], {x,-4,4}, PlotRange->All]
In[..]:= Plot[ Evaluate[ D[inter[x],{x,2}]],{x,-4,4},PlotRange->All]
In[..]:= Plot[ Evaluate[ D[inter[x],{x,3}]],{x,-4,4},PlotRange->All]
In[..]:= Show[ pl2, pl1, Graphics[ Line[{ {-4,10}, {4,10} }] ] ]
In[..]:= Integrate[ (10 - inter[x]), {x,-4,4} ]          Fehlanzeige
In[..]:= NIntegrate[ (10 - inter[x]), {x,-4,4} ]
In[..]:= NIntegrate[10 - inter[x], {x, -4, 4}, AccuracyGoal -> 3]
In[..]:= Plot[inter[x], {x,2.0,4}]
In[..]:= polyfit[x_] = Fit[ data, Table[ x^j, {j,0,10} ], x ]
In[..]:= Needs["Statistics`Master`"]
In[..]:= NonlinearFit[ data, a + b/( (x/c)^2 +1), x , {a,b,c} ]
In[..]:= f1[x_] = a + b/( (x/c)^2 +1) /. %
In[..]:= Show[ Plot[f1[x], {x,-4,4} ], pl1 ]
In[..]:= NonlinearFit[ data, a - b Exp[-c x^2], x, {a,b,c} ]
In[..]:= f2[x_] = a - b Exp[-c x^2] /. %
In[..]:= Show[ Plot[ f2[x], {x,-4,4} ], pl1 ]
```

16 Muster, Ordnen, Sortieren

```
In[..]:= f[a] + f[b] /. f[x_] -> x^2
In[..]:= liste = { 1, 2, f[a], g[b], x^n, f[b], 3.4, Sin[p]^2}
In[..]:= Position[ liste, f[x_] ]                    alle Plätze mit f[irgendwas]
In[..]:= Cases[ liste, _^2]                          alle Fälle mit Potenz 2
In[..]:= Count[ liste, _^_ ]                 Anzahl aller Fälle mit irgendeiner Potenz
In[..]:= DeleteCases[ liste, _[b] ]                  irgendwas[b] fällt raus

In[..]:= Select[ liste, IntegerQ]                    IntegerQ[ ganzer Zahl ] = True
In[..]:= ?*Q
In[..]:= Select[ liste, NumberQ ]                    NumberQ[ Zahl ] = True
In[..]:= Select[ liste, !NumberQ[#]& ]       Ausrufezeichen (!) = logische Verneinung

In[..]:= term = Expand[ 3*(1 + x)^3 * (1 - y - x)^2 ]
In[..]:= Expand[ (x * y)^n ]
In[..]:= PowerExpand[ % ]
In[..]:= Factor[ term ]
In[..]:= FactorTerms[ term ]                         klammert Zahlenfaktoren aus
In[..]:= PolynomialQ[ term, x ]                      gibt True, wenn term = Polynom in x
In[..]:= Collect[ term, y ]                          ordnet nach Potenzen von y
In[..]:= Variables[ term ]
In[..]:= Coefficient[ term, x^2 y^2]                 der Koeffizient von x^2 y^2
In[..]:= Coefficient[ term, y^2 ]
In[..]:= Coefficient[ term, x, 5 ]                   der Koeffizient von x^5
In[..]:= CoefficientList[ term, x ]                  die Koeffizienten der x-Potenzen
In[..]:= Table[ x^j,{j,0,Exponent[term,x]}]
In[..]:= Expand[ %.%% ] == term
In[..]:= Apart[ 1/(x^4 - 1) ]                       Partialbruchzerlegung
In[..]:= Together[ % ]                               bringt auf den Hauptnenner
In[..]:= ExpandDenominator[ % ]                      multipliziert Nenner aus
In[..]:= Expand[ Sin[x]^2 + Sin[2x]^2 ]
In[..]:= Expand[ % , Trig -> True ]                          Potenzen verschwinden
In[..]:= Expand[ Sin[a x] Cos[b x]^2, Trig -> True ]         Produkte verschwinden
In[..]:= Factor[ % , Trig -> True ]                          die Umkehrung
In[..]:= Sort[ { Sqrt[2], 2, (4 - 2 Sqrt[2])/2 } ]           stimmt das Ergebnis ?
In[..]:= N[ % ]                                              numerisch wohl kaum!
In[..]:= Sort[ % ]
```

17 Unterprogramme, Pure Functions, Spezielles

Unterprogramme

```
Unterprogramme mit lokalen Variablen :
Block[{x, y, ...}, expr]    und    Block[{x = x0, ...}, expr]    bzw.
Module[{x, y, ...}, expr]   und    Module[{x = x0, ...}, expr]
```

```
In[..]:= !! euler.m
```

```
euler::usage = "
euler[f, x0, y0, xende (, h )] integriert numerisch die Differential-
gleichung  y' = f(x,y)  mit der Anfangsbedingung y(x0) = y0 von x0 bis
xende. Es verwendet das Eulerverfahren mit der Schrittweite h. Ohne An-
gabe dieses Parameters wird h auf |xende - x0|/100 gesetzt."

      euler[ f_, x0_, y0_, xende_ ]:=
            euler[ f, x0, y0, xende, Abs[(xende - x0)/100] ] ;
      euler[ f_, x0_, y0_, xende_, h_ ]:=
            Block[ {n, eulerstep, xn, yn, nh, k, liste},
                  n = Round[ N[ Abs[ (xende - x0)/h ] ] ] ;
                  nh = N[ h Sign[ (xende -x0) ] ] ;
                  eulerstep[ {xn_, yn_} ]:= ( k = nh*f /. {x->xn,y->yn} ;
                                              N[ {xn + nh, yn + k} ] )  ;
                  liste = NestList[ eulerstep, {x0,y0}, n ]  ;
                  ListPlot[ liste, PlotJoined -> True, PlotRange -> All,
                                   PlotStyle -> Thickness[0.001] ]
                  ]
```

```
In[..]:= << euler.m
In[..]:= ?euler
In[..]:= p1 = euler[ (y + x^2 Cos[x] +1/2)/x, Pi, -1/2, 5 Pi ]
In[..]:= p2 = Plot[ x Sin[x] -1/2 , {x,-5Pi,5Pi}, PlotStyle->Thickness[.001] ]
In[..]:= Show[ p2, p1 ]
In[..]:= p3 = euler[ (y+x^2 Cos[x] + 1/2)/x, Pi, -1/2, 5Pi, .001 ]
In[..]:= p4 = euler[ (y+x^2 Cos[x] + 1/2)/x, Pi, -1/2, -5Pi, .001]
In[..]:= Show[ p3, p4, p2 ]
```

Pure Functions, Spezielles

```
pure functions sind Operatoren
Format :
operator := expr[ # ]&    bzw.   operator:= expr[ #1, #2, ... ]&
```

```
In[..]:=  liste = { 1, 2, f[a], g[b], x^n, f[b], 3.4, Sin[p]^2 }

In[..]:= Map[ NumberQ, liste ]

In[..]:= Map[ !NumberQ, liste ]                    so geht's nicht

In[..]:= notnum:= !NumberQ[#]&                     die Negation von NumberQ

In[..]:= Map[ notnum, liste ]

In[..]:= Select[ liste, notnum ]

In[..]:= fermi := 1/(Exp[ #1( #2 - 1)] + 1)&       Funktionen lassen sich auch als
                                                   Operatoren definieren

In[..]:= fermi[b,e]

In[..]:= Plot[ fermi[8,e], {e,0,3} ]

In[..]:= Nest[ 1/(1 + #)&, x, 3 ]                  so macht man Kettenbrüche
```

Zu dem Sortierproblem

```
In[..]:= Sort[ {2, Sqrt[2], (4 - 2 Sqrt[2])/2} ]      falsche Ordnung ?

In[..]:= ?Sort                                        was ist eine ordering function ?

In[..]:= 2 < Sqrt[2]

In[..]:= FullForm[ % ]

In[..]:= kleiner:= N[#1] < N[#2] &                    ordnet nach dem numerischen Wert

In[..]:= Sort[ {2, Sqrt[2], (4 - 2 Sqrt[2])/2}, kleiner ]
```

jetzt stimmt's

Stichwortverzeichnis

Abs 9
AccuracyGoal 19
All 17
Apart 20
AppendTo 13
Apply 14
ArcTan 3
Array 15
AspectRatio 3
Attributes 7
Axes 19
AxesLabel 18
BarChart 17
BesselJ 4
BinCounts 17
Block 21
Cases 20
Chop 15
Clear 5
Coefficient 20
CoefficientList 20
Collect 20
ContourPlot 4
ContourSmoothing 4
Cos 2
Count 20
D 13
Dashing 18
Date 13
DeleteCases 20
DeleteFile 13
Det 15
DiagonalMatrix 15
Dimensions 15
Directory 19
Display 18
DisplayFunction 17
Do 11
DSolve 16
Eigenvalues 15
Evaluate 18
Exp 2
Expand 5
ExpandDenominator 20
Exponent 20
Factor 6
FactorTerms 20
Factorial 5
FileNames 19
FindMinimum 10
FindRoot 11
Fit 8
FixedPoint 12
For 11
FortranForm 9
Frame 18
FrameLabel 18
FromCharacterCode 7
FullForm 22
GeneralizedBarChart 17
Graphics 19
Head 7
HermiteH 18
I 3
IdentityMatrix 15
If 12
Im 3
Infinity 5
InputString 13
IntegerQ 20
Integrate 13, 14
Interpolation 19
Inverse 15
Join 14
LegendreP 18
Length 8
Line 19
LinearSolve 16
ListPlot 7
Log 2
Map 9, 10
MatrixForm 15
Max 17
Min 17
Mod 4
Module 21
N 1, 5
NDSolve 16

Needs 17
Nest 12
NestList 12
NIntegrate 13
NonlinearFit 17
Normal 14
NormalDistribution 17
NProduct 16
NSolve 16
NumberQ 20
OpenAppend 13
Options 4
OutputForm 9
ParametricPlot 4
ParametricPlot3D 4
Pause 12
Permutations 8
Pi 2
Plot 3, 4
Plot3D 4
PlotGradientField 18
PlotJoined 19
PlotPoints 4
PlotRange 3
PlotStyle 18
PlotVectorField 18
PolynomialQ 20
Position 20
PowerExpand 20
Precision 5
Print 11
Protect 14
RGBColor 18
Random 8
Range 7
Re 3
ReadList 9
Real 9
Remove 6, 8
Reverse 8
Round 12
Save 9
SeedRandom 9
Select 20
Series 14
Short 7
Show 4
Sign 4
Simplify 13
Sin 2
Solve 16
Sort 7, 20
Sqrt 2
StringJoin 13
Sum 10
Table 7
TableForm 9
Tan 14
Thickness 21
Timing 12
ToString 13
Together 20
Transpose 15
Trig 20
True 9
Unprotect 14
Variables 20
ViewPoint 10
While 12
$DisplayFunction 17
$Echo 13
! 9, 20
!! 9
-> 10
. 15
/. 10
// 3
; 7, 10
<< 9
== 9
>> 8
? 2
[[ ]] 7, 8
22
% 1
%n 3
& 22
@ 6