Maple-Kurzreferenz (Version 3.1 – J.Berwanger & F.Wünsch)

Hier sind (fast) alle Befehle anhand einzelner, konkreter Beispiele zusammengefasst, welche in dem einwöchigen Kurs besprochen werden.

Programmstart, Strukturierung und Eingabemodi:

1. Erstellen Sie ein neues Dokument immer im **Worksheet Mode** und nicht im Document Mode. Ersteres ermöglicht eine **klare Trennung** von **logischem Input** und **begleitenden Text.**
2. Geben Sie **logischen Input** immer im **Mathe-Modus** und **erläuternden Text** im **Text-Modus** ein (Umschalten in den Mathe- bzw. Textmodus mittels **„Strg“+„m“** bzw. **„Strg“+„t“** oder mittels des Navigationsmenüs).
3. **Strukturieren Sie Ihr Worksheet** mittels Sections und Subsections übersichtlich und **fügen Sie erklärenden Text ein**. Mittels des Navigationsmenüs oder mittels **„Strg“+„.“**(Punkt) wird eine neue Section bzw. Subsection angelegt. Mittels **„Strg“+„,“**(Komma) wird aus der aktuellen Section bzw. Subsection herausgesprungen, und durch anschließendes Drücken von **„Strg“+„.“**(Punkt) kann eine neue aufgemacht werden.

Grundlegendes:
Neustart, Hilfe, Paketaufruf, Rechengenauigkeit und Dateneigenschaften

Befehl	Erklärung
restart;	**Setzt Maple** mit allen Variablen **zurück**
x:='x';	**Setzt** die Variable ***x* zurück**
%; %%; bzw. ***%%%;***	Referenziert den letzen, vorletzten bzw. drittletzten Befehl
?befehl;	Liefert ausführliche **Hilfe** und Beispiele **zu** einem **Befehl**
with(plots);	*plots*-Paket wird zum Maple-Kernsystem dazugeladen
eq:=x=y+z;	Gibt der Gleichung *x=y+z* den Namen *eq*
rhs(glg); bzw. ***lhs(glg);***	**Extrahiert** die **rechte** (*rhs)* bzw. **linke Seite** (*lhs)* einer Gleichung oder Zuweisung
I^2;	Berechnet das Quadrat der **imaginären Zahl *I***
sin (1/4 * Pi); bzw. ***sin (0.25 * Pi);***	Bestimme Wert **ohne** bzw. **mit numerischer Berechnung** und Rundung (letzteres sollte vermieden werden)
evalf(x,N);	**Berechnet *x* numerisch auf N Dezimalstellen** genau (N ist optional und standardmäßig auf 10)
Digits:=50;	**Stellt global** die **Genauigkeit** auf 50 ein. (Standard: 10)
evalb (7 < 3);	**Berechnet** einen **logischen Ausdruck**. Ausgabe: *true/false*
a:=x^2+2*x+1; ***subs(x=17,a);***	**Ersetzte temporär** *x* und gebe Ausdruck *a* aus
assume(x,real); ***additionally(x>-1);***	**Belege** Variable ***x* mit Eigenschaften**
about(x);	**Gebe Eigenschaften** der Variable *x* **aus**

Datentypen:
Bereiche, Folgen Listen, Mengen, Arrays, Vektoren und Matrizen

Befehl	Erklärung
exp(sqrt(2*x+t));	**Algebraischer Ausdruck**
glg:=x=y+z;	**Gleichung** welche auf dem Namen *glg* abgelegt ist
intervall:=0..2*Pi;	**Bereich** mit dem Namen *intervall*
folge:=1,2,5,sqrt(2);	**Folge** mit dem Namen *folge*
seq(n^2,n=1..5);	**Erzeugt** die **Folge gemäß** einer **Vorschrift**: Hier: *1,4,9,16,25*
[1,3,x,I,exp(8)];	**Liste**, ohne Zuweisung auf Parkvariable. Reihenfolge ist **fest**
[seq(n^2,n=1..5)];	**Erzeugt** die **Liste** *1,4,9,16,25*
menge1:={2,3,y,sqrt(3)};	**Menge** mit dem Namen *menge1. Reihenfolge* **nicht fest**
erg[2];	**Adressiert** das **2. Element** einer Folge, Liste und Menge (hier mit dem Namen *erg*)
arr1:=array(1..4,1..2,[[1,2],[3,4],[5,6],[7,8]]);	**Array** mit **vier Zeilen** und **zwei Spalten** mit dem Namen *arr1* (es wird zeilenweise belegt!)
arr1[3]; ***arr2[2,2];***	**Adressiert Elemente** eines ein- bzw. Mehrdimensionalen **Arrays**
Vector[column]([vx,vy,vz]); bzw. ***<vx,vy,vz>;*** ***Vector[row]([vx,vy,vz]);*** bzw. ***<vx|vy|vz>;***	Erzeugen eines **Spalten-** und. **Zeilenvektors** mit den Einträgen *vx,vy* und *vz*
A:=Matrix(3,2,[[1,2],[2,3],[3,4]]);	Erzeugt **Matrix mit 3 Zeilen und 2 Spalten**

Funktionen vs. Ausdrücke:
Ausdrucks- und Funktionsdefinition, Kovertierung (Ausdruck – Funktion)

Befehl	Erklärung
a:=2*x+20;	**Definiert** einen **Ausdruck** und legt ihn auf *a* ab
subs(x=10,a);	**Wertet** den **Ausdruck temporär** an der Stelle *x=10* **aus**.
a:=subs(x=10,a);	**Wertet** den **Ausdruck dauerhaft** an der Stelle *x=10* **aus**. (da a **rekursiv** überschrieben wird)
f:=x->x^5; bzw. ***f(x):=x^5;*** (nicht in Maple V!)	**Definiert** eine **Funktion** *f* welche *x* auf *x^5* abbildet (=Abbildungsvorschrift)
f(21);	**Wertet** die **Funktion** *f* an der Stelle 21 **aus**.
f:=x->piecewise(x<0,0,x>0,x^2);	**Definiert** eine **Funktion stückweise**
y:=(x,alpha,v)->tan(alpha)*x-v/(x^2);	**Funktionsdefinitions mehrerer Variablen**
a:=2*x+20*y; ***f:=unapply(a,x,y);***	**Definiere** einen **Ausdruck** *a* und mache anschließend mittels *unapply* aus dem **Ausdruck** *a* eine **Funktion** *f*, welche (hier) von *x* und *y* abhängt
f(x,y);	**Mache aus** der **Funktion** zweier Variabler *f* einen **Ausdruck** (Werte also die Funktion an der „allgemeinen“ Stelle *(x,y)* aus)

Vereinfachung von Ausdrücken:
simplify, factor, collect, combine, expand* und *convert

Befehl	Erklärung
simplify((a+b)^2+(a-b)^2);	"**Allgemeine**" (unspezifische / unkontrollierte) **Vereinfachung**, welche oft schnell weiter hilft
factor(6*x^2 + 18*x - 24);	**Faktorisierung** eines Ausdrucks (funktioniert nur bei Ganzzahlen automatisch)
collect (f, x*(x+1)+y*(x+1),x);	**Zusammenfassen** der Terme mit der gleichen Potenz einer Variablen (hier *x*)
combine(exp(sin (x) * cos (y)) * exp(cos (x) * sin (y));	**Zusammenfassen** von Termen "unter Ausnutzung von Rechenregeln"
expand((x+1)*(x+2));	"**Expandiere**" einen Ausdruck ("to distribute products over sums")
convert(exp(I*z),trig);	**Bringe Ausdruck in eine andere Form** (hier: *cos(z) +I*sin(z)*)

Reihen, Summen, Produkte und Grenzwerte: ***series, sum, product,* und *limit***	
Befehl	**Erklärung**
series(exp(x), x=0, 4);	**Reihenentwicklung** der ersten 4 Glieder von *exp(x)* um *x=0*
convert(reihe, polynom);	Jetzt ist der O-Term(höhere Entwicklungsterme) weg
sum(x^n,n=0..infinity);	Berechne **Summe** von *x^n* von 0 bis unendlich
product((n^2-2)/(n^2+4),n=0..10);	Berechne das **Produkt** für *n* von 0 bis 10
limit(sin(x)/x,x=0);	**Grenzwert** des an der Stelle *x=0*
limit(1/x , x=0); (ergibt *undefined*) ***limit(1/x , x=0 , left);*** ***limit(1/x , x=0 , right);***	**Rechts**- bzw. **linksseitiger Grenzwert** nach Ausgabe *undefined*
assume(b>0); ***limit(exp(-b*x) , x=infinity);***	Dieser Grenzwert wäre ohne „Hilfe“ nicht zu lösen, da er hier vom Vorzeichen von b abhängt

Analytisches und numerisches Lösen von Gleichungen und Gleichungssystemen: ***solve, fsolve, allvalues, assign* und *RootOf*-Notation**	
Befehl	**Erklärung**
implicitplot({x^2+2/y^2=2,y^2-x^2=1},x=-2..2, y=-2..2);	Visualisierung der Lösungsmenge mit **implizitem Plot**
sol1:=solve(a*x^2+b*x+c=0,x);	**Löst eine Gleichung** nach *x* auf und speichert Ergebnis als Folge *erg1, erg2* auf Namen *sol1* ab
a:=sol1[1]; ***b:=sol1[2];***	**Ablegen der Lösung** 1 bzw. 2 auf *a* bzw. *b*
solve({x^2-1=0,x>0},x);	**Lösen einer Gleichung mit Nebenbedingung**
sol2:=solve({x+2*y-2*z=5,3*x-y+z=-2,-4*x+y-7*z=0},{x,y,z});	**Lösen eines linearen Gleichungssystems;** Lösung als Menge von **nicht zugewiesenen** Ausdrücken: *{z=3, x=5, y=2}*
assign(sol2);	**Zuweisung** von x,y *und* z auf die Werte 3,5 und 2
sol3:=solve({x^2+2/y^2=2,y^2-x^2=1},{x,y});	**Löse nichtlineares Gleichungssystem**; Ergebnis: Folge von Lösungs-Mengen *{x1,y1}, {x2,y2}, ...*
p3:=allvalues(sol3[3]);	Erzwingt **explizites Ausrechnen** von bestimmten Lösungen(hier der Lösung 3) (sonst oft nur in der Form *RootOf( )*) und legt es auf *p3* ab.
fsolve({x^2+2/y^2=2,y^2-x^2=1},{x,y});	**Numerische Lösung** von **Gleichungssystemen**; i.d.R. wird nur *eine* beliebige Lösung ausgegeben
fsolve({x^2+2/y^2=2,y^2-x^2=1},{x=0,y=0.1});	Suche **numerische Lösung in der Nähe** von *{x=0, y=0.1}* (alternativ auch Bereiche möglich: x=0..1,y=0..1)

Differenzieren und Integrieren von Ausdrücken und Funktionen: ***diff,* D-Operator und *int***	
Befehl	**Erklärung**
diff(x^2,x);	**Erste Ableitung** eines **Ausdrucks**; Ergebnis: **Ausdruck**
diff(sin(x),x$2);	**Zweite Ableitung** eines **Ausdrucks**; Ergebnis: **Ausdruck**
f:=x->x-x^2; ***D(f);***	**Erste Ableitung** einer **Funktion;** Ergebnis: **Funktion**
D(f)(x);	**Erste Ableitung** einer **Funktion;** Ergebnis: **Ausdruck**
(D@@2)(f); bzw. ***(D@@2)(f)(x)***	**Zweite Ableitung** einer **Funktion;** Ergebnis: **Funktion** bzw. **Ausdruck**
y:=(x,alpha,v)->tan(alpha)*x-v/(x^2);	Funktion dreier Variabler
diff(y(x,alpha,v),v$2);	**Zweite Ableitung** eines **Ausdrucks** nach *v*; Ergebnis: **Ausdruck**
(D[1]@@2)(y); bzw. ***(D[1]@@2)(y)(x,alpha,v);***	**Zweite Ableitung einer Funktion** nach der zweiten Variablen *alpha;* Ergebnis: **Funktion** bzw. **Ausdruck**
int(sin(x)^2,x);	**Unbestimmtes Integral** (*int* erwartet **Ausdruck**)
int(x^n,x=a..b);	**Bestimmtes Integral**
assume(a>0); ***int(exp(-a*x),x=0..infinity);***	Lösbar nur unter Bedingung, da Vorzeichen entscheidend
int(sin(sqrt(f(x)+1)+1),x=0..Pi); ***evalf(%);***	Falls analytische Lösung nicht möglich (Output=Input); **numerisch** mittels *evalf(int(...));*

Graphiken in Maple (z.T. erst nach Eingabe von *with(plots);* verfügbar): ***plot, plot3d, animatecurve, animate, animate3d, implicitplot, implicitplot3d, display* und *spacecurve***		
	Befehl	**Erklärung**
'	***plot(sin(x),x=0..4*Pi,style=point,color=black, numpoints=100);***	**Plottet 1D Ausdruck**. (Alles nach dem Ausdruck ist **optional**)
	plot([seq([n,rand(100)()],n=1..10)],style=point);	**Plotte** zufällige **(x,y)-Wertepaare** als „listlist“
	plot([x^2,1/x,x^3],x=0..2,scaling=constrained);	**Mehrere Kurven in einen Plot** (Option *scaling* erzwingt gleiche Skalierung für x- und y-Achse)
	plot([sin(2*t),cos(t),t=0..2*Pi]);	**Parameterplot** (trägt *x=sin(t)* gegen *y=cos(t)* auf)
	plot3d(sin(x)*cos(y),x=-3..3, y=-3..3, view = [0 .. 20, -10 .. 10, -20 .. 20]);	**Plottet 2D Ausdruck**. (*x=-3..3* und *y=-3..*3 definiert **ausgewerteten Bereich** und Option *view* definiert **angezeigten Bereich**)
with(plots);	***animatecurve(sin(x), x=-Pi..Pi,frames=50);***	**Plottet/animiert sich aufbauend Kurve** in **50 Einheiten**
	animate(sin(x-t),x=0..20,t=0..8*Pi,frames=50);	**Plottet zeitlich variierenden 1D Ausdruck** in **50 Einzelbildern**
	animate3d(x*cos(y*t),x=-3..3,y=-3..3, t=1..2);	**Plottet zeitlich variierenden 2D Ausdruck**
	spacecurve([sin(t),cos(t),t/20],t=0..12*Pi);	**Erzeugt** eine **3D-Raumkurve**
	implicitplot(x^2+2/y^2=9, x=-3..3, y=-3..3);	**Plot** einer **impliziten Gleichung**
	implicitplot([x^2+2/y^2=9, x^2-y^2=1],x=-3..3, y=-3..3);	**Plot** eines **impliziten Gleichungssystems**
	lsg := cos(x)+cos(y)-cos(x)*cos(y) = (1/10)*c; ***a := seq(implicitplot(subs(c = i, lsg), x = -3 .. 3, y = -3 .. 3), i = 0 .. 10);*** ***display([a], insequence = true)***	**Erzeugen mehrerer Plots** einer **impliziten Gleichung** *lsg* für verschiedene *c* Werte (Option ***insequence=true* stellt einzelne Plots nacheinander dar→Movie**)
	f := x^3+y^3+z^3+1 = (x+y+z+1)^3: ***implicitplot3d(f, x=-2..2,y=-2..2,z=-2..2);***	**Plot** einer **3D impliziten Gleichung**
	implicitplot3d(r=1, r=0..1.2, phi=0..2*Pi, theta=0..Pi, coords=spherical);	**Plot** einer **3D impliziten Gleichung** (hier mit **Kugelkoordinaten)**

Lösen von Differentialgleichungen ohne & mit Nebenbedingungen: *dsolve* und *dsolve(*DGL,*numeric)*

Befehl	Erklärung
dsolve(diff(y(x),x)=y(x),y(x)); bzw. ***dsolve(D(y)(x)-y(x)=0,y(x));***	**Allgemeine Lösung** einer **DGL 1.Ordnung**; Lösung: *y(x)=_C1*exp(x)*
dsolve(diff(y(x),x$2)+y(x)=0,y(x)); bzw. ***dsolve((D@@2)(y)(x)=-y(x),y(x));***	**Allgemeine Lösung** einer **DGL 2.Ordnung**; Lösung: *y(x)=_C1*cos(x)+_C2*sin(x)*
dgl:=(D@@2)(y)(x)+y(x)=0; ***dsolve({dgl,y(0)=17,D(y)(0)=2}, y(x));***	**Spezielle Lösung** einer **DGL 2.Ordnung** mit **Nebenbedingungen**
dsolve(dgl,y(x),implicit);	**Implizite Lösung** einer **DGL 1.Ordnung**
dgls:=D(x)(t)=a11*x(t)+a12*y(t), ***D(y)(t)=a21*x(t)+a22*y(t);*** ***dsolve({dgls},{x(t),y(t)});***	**Allgemeine Lösung** eines **Systems von DGLs**
anfb:= anf:=x(0)=1,D(x)(0)=0; ***dsolve({dgls,anfb},{x(t),y(t)});***	**Lösung** eines **Systems von DGLs** mit **Nebenbedingungen**
dgl:=(D@@2)(x)(t) + D(x)(t)/4 + x(t)=0; ***anf:=x(0)=1,D(x)(0)=0;*** ***sol:=dsolve({dgl,anf},x(t),numeric);*** ***sol(1.0);***	**Numerische Lösung** einer **DGL** (hier gedämpfte harmonisches Pendel); **Ergebnis** ist eine **Maple-Funktion:** Aufruf für die Zeit *t=1.0* mittels *sol(1.0)* Ergebnis: [t=1.0, x(t)=0.5757, dx(t)/dt=-0.7447]
seq([rhs(sol(i/10)[1]),rhs(sol(i/10)[2])],i=0..200): ***plot([%]);***	**Plot** der **numerischen Lösung** mittels Wertepaare-Erzeugung via *seq()*

Programmieren: *if-, while- & for-*Schleifen, Prozeduren, Argumentenliste, *break & RETURN()* und *print*

Befehl	Erklärung
if j<0 then ***Befehle#1;*** ***elif j=0 then*** ***Befehle#2;*** ***else*** ***Befehle#3;*** ***fi;*** bzw. ***end fi;***	Beispielshafte ***if*-Schleife:** Hierbei sind die *elif*- und die *else*-Abfragen optional! Anstatt den *j<0* und *j=0* Abfragen können selbstverständlich beliebige Bedingungen gesetzt werden
i:=1.1; ***while i<10 do*** ***Befehle;*** ***od***; bzw. ***end do;***	Beispielshafte ***while*-Schleife:**
for i from -2 by 5 to 18 do ***print(„Hallo Welt!“);*** ***od:*** bzw. ***end do;***	Beispielshafte ***for*-Schleife:** Die Schrittweite (hier *by 5*) ist optional und standardmäßig auf 1 gesetzt!
myproc := proc(x) ***local i,j;*** ***j:=x*i;*** ***end;***	Beispielshafte **Prozedur** (hier mit dem Namen *myproc*)**.** In diesem Beispiel werden *i* und j als **lokale Variablen** definiert (Alternative: ***global***) und *j:=x*i;* berechnet und ausgegeben.
plot('myproc(x)',x=-1..1);	**Plotten von Prozeduren nur mit ″ möglich**
test := proc() # kein Argument ist möglich. ***local i;*** ***if nargs <> 2 then*** ***print("Falsche Anzahl vonArgumente");*** ***RETURN(); fi;*** ***if not type(args[2],nonnegint) then print("...");*** ***RETURN(); fi;*** ***print(sum(args[1]^j,j=0..args[2]));*** ***end:***	Unter ***args*** bzw. ***nargs*** sind die **Argumente der Argumentenliste** bzw**. die Anzahl der Argumente abgespeichert**. Auf diese kann dann innerhalb einer Prozedur zugegriffen werden.
break; bzw. ***RETURN();***	Die Befehle brechen die aktuelle Schleife bzw. Prozedur ab und springen aus dieser heraus.
not, ***or*** und ***and***	**Verknüpfungen bei komplexen Bedingungen**
print(„Hallo Welt!“); ***a:=10: print(a):***	Der *print()* Befehl gibt seinen Inhalt unabhängig von seinem Kommandoabschluß aus.

Lineare Algebra (*with(LinearAlgebra)* und Vektoranalysis (*with(VectorCalculus)*): *Add, DotProduct, CrossProduct, VectorNorm, Normalize, Transpose, Multiply, MatrixInverse, Determinant, Eigenvalues, Eigenvectors, whattype, LinearSolve, VectorField, Gradient, Divergence, Nabla, Curl* und *ScalarPotential*

	Befehl	Erklärung
with(LinearAlgebra);	***v+w;*** bzw. ***Add(v,-w);*** bzw. ***Vector(v+w);***	**Vektoraddition/-subtraktion** zweier Vektoren
	v.w; bzw. ***DotProduct(v,w);***	**Skalarprodukt** zweier Vektoren
	v &x w; bzw. ***CrossProduct(v,w);***	**Kreuzprodukt** zweier Vektoren
	VectorNorm(v,Euclidean);	**Euklidische Norm**(=Länge) eines **Vektors**
	ev := Normalize(v,Euclidean);	Normalisierung(Länge=1) eines **Vektors**
	Transpose(v); bzw. ***Transpose(A);***	**Transponieren** eines **Vektors** bzw. einer **Matrix**
	A+B; bzw. ***Add(A,-B);*** bzw. ***Matrix(A+B);***	**Matrixaddition/-subtraktion** zweier Matrizen
	A.B; bzw. ***Multiply(A,B);*** bzw. ***Matrix(A.B);***	**Matrixmultiplikation** zweier Matrizen
	Ai:=MatrixInverse(A);	**Inverse** einer **Matrix**
	Determinant(A);	**Determinante** einer **Matrix**
	Eigenvalues(A);	**Eigenwerte** einer **Matrix** als **Spaltenvektor**
	eigenw,eigenv:=Eigenvectors(A);	***Eigenvektoren einer Matrix – Teil A:*** ***1.Vektor=Spaltenvektor mit Eigenwerten*** ***2.Matrix=Spalten sind dazugehörige Eigenvektoren***
	eigenw[2]; eigenv[1 .. 5, 2]	**Adressierung** des **Eigenwerts und -vektors 2**
	Eigenvectors(A, output = list);	***Eigenvektoren einer Matrix – Teil B:*** ***Ausgabe gemäß der Struktur:*** ***[EW1, Multiplizität, {EV1,EV2, ...}],…***
	eval(v); bzw. ***eval(A);***	**Ausgeben** eines **Vektors** bzw. einer **Matrix**
	whattype(v); whattype(A);	**Eigenschaftsabfrage:** ***Vector[row/column]*** oder ***Matrix***
	w:=LinearSolve(A,v);	**Lösung lineares Gleichungssystem: A*w=v**
with(VectorCalculus);	***V:=(x,y)-> exp(-(x^2+y^2));***	**Definition** eines **Skalarfelds** (Gaußfunktion)
	F:=VectorField(<x^2+y,y^2-z,z^2*x>, ***'cartesian'[x,y,z]);***	**Definition** eines **Vektorfelds** in **kartesischen Koordinaten**
	grad1:=Gradient(V(x,y),[x,y]);	**Gradient** eines **Skalarfelds**
	Divergence(F); bzw. ***Nabla.F;***	**Divergenz** eines **Vektorfelds**
	Curl(F);	**Rotation** eines **Vektorfelds**
	F := VectorField(evalm(q1*q2/(4*Pi*epsilon) ***1/(x^2+y^2+z^2)^(3/2))*<x,y,z>,*** ***'cartesian'[x,y,z]);*** ***curl(F);*** (da kommt =0 heraus) ***V:=ScalarPotential(F);***	*ScalarPotential* **erzeugt Skalarpotential zu Vektorfeld,** sofern **curl(F)==0;** (andernfalls keine Rückgabe)