Kapitel 5

Mathematik

5.1 Einfache mathematische Funktionen

5.1.1 Arithmetische Operatoren

Mit den in Kapitel 4 definierten arithmetischen Operatoren können alle Grundrechnungsarten durchgeführt werden. Man muss die in Kapitel 4 definierte Priorität der Operatoren beachten und unter Umständen mit runden Klammern die Reihenfolge der Ausführung beeinflussen.

Fälle wo die Reihenfolge oft nicht richtig im numerischen Programm spezifiziert wird, sind in folgender Tabelle angeführt:

MATH	MATLAB
$y = \frac{x}{a+b}$	`y = x ./ (a + b)`
$y = \frac{x}{a} + b$	`y = x ./ a + b`
$y = x^{a+b}$	`y = x .^ (a + b)`
$y = x^a + b$	`y = x .^ a + b`

Man beachte auch die auf den ersten Blick ungewöhnliche Form der Operatoren `.*` (Multiplikation), `./` (Division) und `.^` (Potenz). Der Punkt vor dem Operator teilt MATLAB mit, dass es die Operation elementweise durchführen muss. Damit funktionieren all diese Operationen mit Arrays gleicher Größe oder mit einer Kombination von Arrays und Skalaren. Behandelt man also nicht Probleme der linearen Algebra (z.B. Matrizenmultiplikation) sollte man immer die Punkt-Operatoren für Multiplikation, Division und Potenzierung verwenden.

5.1.2 Mathematische Funktionen und Konstanten

Die Argumente für mathematische Funktionen, siehe z.B. 5.1.5, müssen anders als in der Mathematik immer innerhalb runder Klammern geschrieben werden:

MATH	MATLAB
$y = \sin x$	`y = sin(x)`
$y = \sin \pi x$	`y = sin(pi .* x)`
$y = \sin(a + b)$	`y = sin(a + b)`
$y = e^{-ix}$	`y = exp(-i.*x)`
$y = \sqrt{1 + cosx}$	`y = sqrt(1 + cos(x))`

An den Beispielen sieht man, dass MATLAB die Zahl π (`pi`) aber nicht die Eulersche Zahl e (`exp(1)`, siehe dazu 5.1.3) kennt. MATLAB kennt auch die imaginäre Konstante $i = \sqrt{-1}$ und praktisch alle (sinnvollen) Operationen können sowohl mit rellen als auch mit komplexen Zahlen durchgeführt werden (siehe dazu 5.1.4).

In der letzten Zeile der Tabelle sieht man einen zusammengesetzten Ausdruck. Die Ausführung in MATLAB muss man sich von Innen nach Aussen vorstellen, d.h., das Ergebnis von `cos(x)` wird mit `1` addiert und danach wird daraus die `Wurzel` gezogen. Die Befehle können beliebig verschachtelt sein. Wichtig ist die korrekte Setzung der Klammern und natürlich die Vorschrift, dass die Ergebnisse innerer Operationen formal richtigen Input für die äusseren Operationen liefern müssen.

5.1.3 Exponential Funktion, Logarithmus

Die Exponentialfunktion $\exp(x)$, die in mathematische Schreibweise oft als e^x geschrieben wird, wird in MATLAB durch die Funktion `exp` berechnet. Die Eulersche Zahl e ist in MATLAB nicht definiert. Braucht man sie, dann muss man sie durch `e=exp(1)` selbst definieren. Für weitere Rechnungen braucht man sie aber nicht wirklich, da die Berechnung von e^x in MATLAB besser mit `exp(x)` und nicht mit mit einer `Potenz` von e (`exp(1).^x`) erfolgt.

Die Funktion `expm1` berechnet $\exp(x) - 1$. Dies ist deshalb wichtig, da für kleine Argumente x der Befehl `exp(x)-1` Null liefert, während `expm1(x)` noch korrekte Ergebnisse liefert, da gilt $\{\forall x \to 0 \mid \exp(x) - 1 \propto x\}$.

Die Umkehrfunktion zur Exponentialfunktion $y = e^x$ ist die Logarithmusfunktion $x = \ln y$. Vertauscht man nun x und y so erhält man $y = \ln x$. Dies bezeichnet man als den natürlichen Logarithmus von x. In MATLAB wird der natürliche Logarithmus durch die Funktion `log(x)` berechnet.

Aus ähnlichen Gründen wie bei `expm1` berechnet die Funktion `logp1` die Funktion $\ln(1+x)$. Für kleine Argumente x liefert der Befehl `log(1+x)` Null, während `log1p(x)` noch korrekte Werte liefert, da gilt $\{\forall x \to 0 \mid \ln(x+1) \propto x\}$.

Der Befehl `log` funktioniert für negative Zahlen (komplexes Ergebnis) und für komplexe Zahlen. Interessiert man sich nur für reelle Ergebnisse für positive Zahlen, kann man den Befehl `reallog` verwenden. Dieser liefert für negativen oder komplexen Input eine Fehlermitteilung.

Der natürliche Logarithmus $\ln$ wird auch als Logarithmus zur Basis e bezeichnet

$$\ln x = \log_e x \ .$$

Der Logarithmus zur Basis 10, $\log_{10} x = \log x$, wird in MATLAB mit `log10(x)` berechnet und der Logarithmus zur Basis 2, $\log_2(x)$ wird mit `log2(x)` berechnet. Für Logarithmen zu einer beliebigen Basis a, $\log_a$, muss man sich der Formel

$$\log_a x = \frac{\ln x}{\ln a} \ ,$$

also in MATLAB z.B. für $\log_3 10$:

```
log_a = @(x,a) log(x) ./ log(a);
y = log_a(10,3);
```

Die allgemeine Exponentialfunktion (Potenz)

$$a^x = e^{x \ln a}$$

wird im Normalfall natürlich mit dem Operator `.^` für das `Potenzieren` berechnet. Es stehen aber einige spezielle Befehle zur Verfügung. Der Operator funktioniert natürlich mit komplexer Basis und/oder Hochzahl. Interessiert man sich nur für reelle Ergebnisse für reellen Input kann man den Befehl `realpow(x,y)` für die Berechnung von x^y verwenden.

Die Quadratwurzel, $\sqrt{x} = x^{1/2}$, wird mit `sqrt(x)` berechnet. Interessiert man sich nur für reelle Wurzeln von nicht-negativen Zahlen, kann man den Befehl `realsqrt(x)` verwenden. Zur Berechnung der reellen n-ten Wurzel soll man den Befehl `nthroot(x,n)` verwenden. Am Beispiel von $\sqrt[3]{-3} = (-3)^{1/3}$ sei der Unterschied zwischen `nthroot(-3,3)` und der Potenzfunktion erläutert:

```
nthroot(-3,3) % liefert -1.4422
(-3).^(1/3)   % liefert  0.7211 + 1.2490i
```

Beide Ergebnisse sin korrekt, aber nur mit `nthroot(-3,3)` erhält man die reelle Wurzel. Klarerweise müssen bei negativen x die Potenzen n ungerade ganze Zahlen sein. Sonst bricht die Funktion mit einer Fehlermitteilung ab.

Für die Basis 2 kann man zur Berechnung von 2^x den Befehl `pow2(x)`, bzw. für $a \cdot 2^x$ den Befehl `pow2(a,x)`. Eine solche Spezial-Funktion ist typischerweise schneller als der normale Befehl.

Der Befehl `p=nextpow2(A)` liefert jene Hochzahl p die sicherstellt, dass gilt $2^p \geq A$. Z.B. ist für alle Werte von A zwischen 513 und 1024 $p = 10$.

5.1.4 Komplexe Zahlen

Als imaginäre Einnheit $i = \sqrt{-1}$ ist in MATLAB `i` aber auch `j` vorgesehen. Man sollte diese beiden Zahlen daher nicht als Variable verwenden:

```
z = 3 + 5*i; % komplexe Zahl

i = 5;
z = 3 + 5*i; % liefert 28

clear('i');
z = 3 + 5*i; % komplexe Zahl
```

Komplexe Zahlen können also durch Befehle vom Typ `z = x + i*y` oder mit `z = complex(x,y)` erzeugt werden.

Folgende Befehle sind für komplexe Zahlen $z = x + iy$ hilfreich:

BEZEICHNUNG	MATH	MATLAB
Realteil	$\mathrm{Re}\ z = x$	`real(z)`
Imaginärteil	$\mathrm{Im}\ z = y$	`imag(z)`
Absolutbetrag	$\lvert z\rvert = \sqrt{x^2 + y^2}$	`abs(z)`
Phasenwinkel	$\phi = \arctan(y, x)$	`angle(z)`
Signum	$\mathrm{sign}\ z = z/\lvert z\rvert$	`sign(z)`
Konjugiert komplex	$\overline{z} = x - iy$	`conj(z)`
Konjugiert komplex transponiert	$\overline{z}^T$	`ctranspose(z)`

Der Befehl `sign(x)` liefert für reelle Werte von x den Wert 1 für $x > 0$, -1 für $x < 0$ und 0 für $x = 0$, für komplexe Werte von x liegen die Ergebnisse am Einheitskreis.

Der Befehl `unwrap(p)` korrigiert Phasensprünge in einem Vektor von Phasenwinkeln p die größer als $\pm\pi$ sind. Dabei werden zu p ganzzahlige Vielfache von 2π addiert oder subtrahiert, damit alle Phasensprünge kleiner als π werden.

Die Überprüfung, ob ein Array (eine Zahl) real oder komplex ist, kann mit `isreal(z)` erfolgen. Dies liefert `1`, wenn `z` reell ist, also keinen Imaginärteil besitzt, ansonsten ist das Resultat `0`. Dieses Ergebnis bekommt man auch, wenn der Imaginärteil vorhanden, aber exakt Null ist. Ob eine Zahl keinen Imaginärteil oder einen Imaginärteil, der exakt Null ist, hat, kann man mit

```
~any(imag(z(:)))
```

feststellen. Im Gegensatz zu `isreal(z)` ist hier das Ergebnis `1`, wenn der Imaginärteil vorhanden aber Null ist.

5.1.5 Trigonometrische Funktionen

In 5.1 sind die trigonometrischen Funktionen zusammengefasst. Diese Funktionen sind in MATLAB für verschiedene Argumente implementiert, nämlich `sin(x)` für das Argument `x` in Radiant, bzw., `sind(x)` für das Argument `x` in Grad (Degree). Ausserdem gibt es jeweils die Hyperbelfunktion, also z.B. den Sinus-Hyperbolicus `sinh`.

Die Funktion `Sekans` ist definiert als $1/\cos(x)$ und die Funktion `Kosekans` ist definiert durch $1/\sin(x)$. Sinngemäß gilt das Gleiche für die entsprechenden Hyperbelfunktionen `Sekans-Hyperbolicus` und `Kosekans-Hyperbolicus`.

Die Umkehrfunktionen (Arkus-Funktionen) liefern den Wert in Radiant (`asin`), bzw., in Grad (`asind`). Die Umkehrfunktionen für die Hyperbelfunktionen heissen korrekterweise nicht Arkus-Sinus-Hyperbolicus sondern Area-Sinus-Hyperbolicus. Dieser Umstand wurde in der Tabelle 5.1 aus praktischen Gründen nicht berücksichtigt.

Alle trigonometrischen Funktionen arbeiten natürlich sowohl mit reellen als auch mit komplexen Argumenten.

In Ergänzung zu `atan` gibt es die sehr praktische Funktion `atan2`. Die Funktion `atan(x)` hat ein Argument und liefert das Ergebnis im Intervall $[-\pi/2, \pi/2]$. Die Funktion `atan2(y,x)` hat zwei Argumente und liefert das Ergebnis im Intervall $[-\pi, \pi]$, womit man auch die richtige Zuordnung zum Quadranten erhält. Der Zusammenhang zwischen Zylinderkoordinaten und kartesischen Koordinaten ist $r = \sqrt{x^2 + y^2}$ und $\phi = \arctan(y/x)$. Mit der normalen Funktion `atan` kann man nicht unterscheiden zwichen den Punkten $(x, y) = (1, 1)$ und $(-1, -1)$, da y/x hier immer gleich 1 ist. Diese Ununterscheidbarkeit gilt für alle um den Ursprung gespiegelten Punkte. Das Problem wird bei der getrennten Übergabe von x und y an `atan2(y,x)` gelöst, da mit dieser Information die Zuordung zum richtigen Quadranten und damit die Berechnung des richtigen Winkels leicht ist.

Die Funktion `c=hypot(a,b)` berechnet die Wurzel aus der Summme der Quadrate

$$c = \sqrt{a^2 + b^2} \,,$$

Tabelle 5.1: Trigonometrische Funktionen

BEZEICHNUNG	RADIANT	GRAD	HYPERBOLISCH
Sinus	`sin`	`sind`	`sinh`
Kosinus	`cos`	`cosd`	`cosh`
Tangens	`tan`	`tand`	`tanh`
Kotangens	`cot`	`cotd`	`coth`
Sekans	`sec`	`secd`	`sech`
Kosekans	`csc`	`cscd`	`csch`
UMKEHRFUNKTIONEN			
Arkus-Sinus	`asin`	`asind`	`asinh`
Arkus-Kosinus	`acos`	`acosd`	`acosh`
Arkus-Tangens	`atan`	`atand`	`atanh`
Arkus-Kotangens	`acot`	`acotd`	`acoth`
Arkus-Sekans	`asec`	`asecd`	`asech`
Arkus-Kosekans	`acsc`	`acscd`	`acsch`

wobei sie so geschrieben ist, dass sie in einem viel weiteren Bereich korrekte Ergebnisse liefert als die einfache Umsetzung,

```
hyp = @(a,b) sqrt(a.^2 + b.^2)
```

wobei das Problem durch Überschreiten der größten möglichen Zahl `realmax`, z.B., bei der Operation a^2 besteht, obwohl das Ergebnis noch darstellbar wäre. Die Funktion `hypot` umgeht dieses Problem für große Zahlen.

Für komplexe Zahlen a und b berechnet `hypot` das reelle Ergebnis

$$c = \sqrt{|a|^2 + |b|^2}\,,$$

wobei $|x|$ der Absolutbetrag von x ist (`abs(x)`).

5.1.6 Diskrete Mathematik

5.1.6.1 Primzahlen

Eine Primzahl ist eine natürliche Zahl mit genau zwei natürlichen Teilern, nämlich 1 und sich selbst. Primzahlen sind also 2, 3, 5, 7, 11, usw.. Die fundamentale Bedeutung der Primzahlen für viele Bereiche der Mathematik beruht auf den folgenden drei Konsequenzen aus dieser Definition:

- Primzahlen lassen sich nicht als Produkt zweier natürlicher Zahlen, die beide größer als eins sind, darstellen.
- Lemma von Euklid: Ist ein Produkt zweier natürlicher Zahlen durch eine Primzahl teilbar, so ist bereits einer der Faktoren durch sie teilbar.
- Eindeutigkeit der Primfaktorzerlegung: Jede natürliche Zahl lässt sich als Produkt von Primzahlen schreiben. Diese Produktdarstellung ist bis auf die Reihenfolge der Faktoren eindeutig.

Jede dieser Eigenschaften könnte auch zur Definition der Primzahlen verwendet werden.

Eine natürliche Zahl größer als 1 heißt prim, wenn sie eine Primzahl ist, andernfalls heißt sie zusammengesetzt. Die Zahlen 0 und 1 sind weder prim noch zusammengesetzt.

In MATLABerzeugt der Befehl `p = primes(n)` einen Zeilenvektor p mit allen Primzahlen, die kleiner gleich n sind. Der Befehl `f = factor(n)` liefert die Faktorisierung (Zerlegung in Primzahlen) der natürlichen Zahl n im Zeilenvektor f. Mit dem Befehl `L = isprime(M)` kann man feststellen, ob die Elemente eines Arrays M Primzahlen sind. Das Ergebnis ist ein logisches Feld L der gleichen Größe wie M mit Eines (prim) oder Nullen(nicht prim).

5.1.6.2 Gemeinsame Teiler und Vielfache

Der Befehl `G = gcd(A,B)` erzeugt den "Größten gemeinsamen Teiler" G der entsprechenden Elemente der ganzzahligen Arrays A und B. Nach mathematischer Konvention liefert `gcd(0,0)` den Wert 0 und alle anderen Inputgrößen liefern positive ganze Zahlen für G.

Beim Aufruf `[G,C,D] = gcd(A,B)` liefert der Befehl neben G auch die beiden Arrays C und D, die folgende Gleichung erfüllen

$$A_i C_i + B_i D_i = G_i \ ,$$

wobei der Index i dafür steht, dass dies für alle entsprechenden Elemente in den Arrays gilt.

Der Befehl `L = lcm(A,B)` erzeugt das "Kleinste gemeinsame Vielfache" L der entsprechenden Elemente der ganzzahligen Arrays A und B.

5.1.6.3 Fakultät

Der MATLAB-Befehl `fac=factorial(n)` berechnet n-Fakultät für ganze positive Zahlen inklusive Null, $n \in N_0^+$,

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \ ,$$

wobei in Ergänzung auch gilt $0! = 1$. Für $n \in N^+$ würde natürlich auch

```
fac = prod([1:n])
```

funktionieren, 0-Fakultät wäre hier aber falsch (`prod`). Mit dem Befehl

```
fac1 = cumprod([1:n])
fac0 = [1,cumprod([1:n])]
```

erhält man alle Werte von $1!$ (bzw. $0!$) bis n-Fakultät (`cumprod`). Ist der Input von `factorial` ein Feld `N`, bekommt man ein gleich großes Feld mit der Fakultät jedes Elements von `N`.

Auf Grund der Tatsache, dass der doppeltgenaue Datentyp (`double`) ungefähr 15 Stellen zur Verfügung hat, sind nur Werte bis $21!$ exakt. Darüber hinaus stimmt die Größenordnung und die ersten 15 Stellen.

5.1.6.4 Binomialkoeffizienten

Der MATLAB-Befehl `nchoosek(n,k)` berechnet den Binomialkoeffizienten

$$\binom{n}{k} = \frac{n!}{(n-k)!\,k!}$$

wobei n und k nichtnegative ganze Zahlen sein müssen. In der Kombinatorik bezeichnet der Binomialkoeffizient die Anzahl von möglichen Kombinationen von n Dingen, wobei immer k Dinge ausgewählt werden. Für einen Zeilenvektor v berechnet `nchoosek(n,v)` eine Matrix, wobei in den Zeilen die möglichen Kombinationen der Elemente von v stehen. Die Befehle

```
v = [1:4]
C = nchoosek(v,3)
```

liefern also

$$v = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} \quad , \quad C = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 4 \\ 1 & 3 & 4 \\ 2 & 3 & 4 \end{bmatrix} .$$

Der Befehl `nchoosek(4,3)` würde natürlich die Anzahl der möglichen Kombinationen, also die Anzahl der Zeilen in `C`, liefern (`size(C,1)`).

5.1.6.5 Permutationen

Der MATLAB-Befehl `P = perms(v)` berechnet alle möglichen Permutationen der Elemente des Vektors `v`. Die Befehle

```
v = [1:4]
P = perms(v)
```

liefern damit

$$v = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \quad , \quad C = \begin{bmatrix} 3 & 2 & 1 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \\ 2 & 1 & 3 \\ 1 & 2 & 3 \\ 1 & 3 & 2 \end{bmatrix} \quad ,$$

wobei der Befehl nur praktikabel ist, wenn die Anzahl der Elemente in `v` kleiner als 15 ist. Will man nur die Anzahl der Permutationen wissen, kann man die natürlich einfacher mit `factorial(numel(v))` berechnen. Der Befehl `numel` liefert dabei die Anzahl der Elemente im Vektor `v`.

5.1.6.6 Näherung durch rationale Zahlen

Obwohl alle Fließkommazahlen durch die beschränkte Anzahl von Stellen rationale Zahlen sind, ist es trotzdem manchmal sinnvoll sie durch einfache rationale Zahlen anzunähern, deren Zähler und Nenner jeweils kleine ganze Zahlen sind. Dazu dient der Befehl `rat` So liefert `[n,d]=rat(pi)` die Werte $n = 355$ (nominator, Zähler) und $d = 113$ (denominator, Nenner), also die rationale Zahl $n/d = 355/113$. Diese Berechnung erfolgt mit dem Defaultwert für die relative Genauigkeit von $1 \cdot 10^{-6}$. Will man eine andere Genauigkeit, muss man diese spezifizieren, also z.B. `[n,d]=rat(pi,1.e-8)` und erhält damit $n/d = 104348/33215$. Die weithin bekannte Repräsentation von $\pi \approx 22/7$ erhält man durch `[n,d]=rat(pi,1.e-2)`.

Verwendet man den Befehl `rat` ohne Ausgabeparameter, dann wird der Wert in Form eines Kettenbruchs (”continued fraction”) ausgegeben

$$\pi \approx 3 + \cfrac{1}{7 + \cfrac{1}{16 - \cfrac{1}{294}}} \approx 3 + \cfrac{1}{7 + \cfrac{1}{16}} \approx 3 + \frac{1}{7} \, .$$

Natürlich funktioniert der Befehl auch mit Feldern `X`, `[N,D]=rat(X)` und man erhält als Ergebnis gleich große Felder `N` und `D` mit Zähler bzw. Nenner für die Näherung der jeweiligen Elemente in `X`.

5.2 Platzhalter

Dies ist ein Platzhalter für ein geplantes Kapitel.

In der Zwischenzeit beschränkt sich der Inhalt auf einen Link auf ein MATLAB-Dokument über mathematische Fragen .