1 Matrizen und Vektoren

1.1 Eingabe

`M = [0 1 0;`
`1 0 2;`
`1 2 3]`
	Eingabe einer Matrize

`z = [1 2 3 4]`	Zeilenvektor
`s = [1 2 3 4]'`	Spaltenvektor
`x = 5:12`	Erzeugt den Vektor `x = [5 6 7 ... 11 12]`
`x = 0:0.1:1`	Erzeugt den Vektor `x = [0.0 0.1 0.2 ... 0.9 1.0]`
`x = 1:-0.1:0`	Erzeugt den Vektor `x = [1.0 0.9 0.8 ... 0.1 0.0]`

1.2 Zugriff auf Matrixelemente

`M(i,j)`	Element aus Zeile i und Spalte j
`M(i,:)`	Alle Elemente der Zeile i
`M(:,j)`	Alle Elemente der Spalte j
`M(L)`	Alle Elemente aus M, die in der Matrix L den logischen Wert true haben
`M(M>a)=b`	Setzt alle Elemente die grösser als a sind auf b
`I=find(L)`	Index aller Elemente die in der Matrix L den logischen Wert true haben

1.3 Operationen

`A+B`	Matrizen-Addition
`A-B`	Matrizen-Subtraktion
`A*B`	Matrizen-Multiplikation
`A.*B`	elementweise Multiplikation: $a_{ij} \cdot b_{ij}$
`A/B`	Matrizen-Division: $A \cdot B^{-1}$
`A./B`	elementweise Division: a_{ij}/b_{ij}
`A'`	Matrizen-Transponierung und Konjugation
`A.'`	Matrizen-Transponierung (ohne Konjugation)
`A.^x`	elementweise Potenzieren: a_{ij}^x

1.4 Spezielle Matrizen

`Z = zeros(m,n)`	Definition einer $m \times n$-Matrix mit lauter Nullen
`O = ones(m,n)`	Definition einer $m \times n$-Matrix mit lauter Einsen
`N = NaN(m,n)`	Definition einer $m \times n$-Matrix mit lauter NaN's (NaN = Not-a-Number)
`E = eye(n)`	Definition einer $n \times n$-Einheitsmatrix
`R = rand(m,n)`	Definition einer $n \times n$-Matrix mit gleichverteilten Zufallszahlen zwischen 0 und 1
`RN = randn(m,n)`	Definition einer $n \times n$-Matrix mit normalverteilten Zufallszahlen mit Mittelwert 0 und Standardabweichung 1

2 Funktionen

2.1 Matrizen-Dimensionen und Datentypen

`[m,n] = size(A)`	Anzahl Zeilen m und Spalten n einer Matrix
`s = size(A)`	Grössen aller Dimensionen der Matrix A mit beliebiger Dimensionalität
`l = length(A)`	Länge einer Matrix bzw. eines Vektors (grösste Dimension)
`B = double(A)`	Matrix A zu double typecasten
`B = single(A)`	Matrix A zu single (32-Bit floating point) typecasten
`isa(A,'double')`	Datentyp von A überprüfen

2.2 Elementare Mathematische Funktionen

`y = sin(x),`	trigonometrische Funktionen mit dem Argument x in Radiant
`y = cos(x),`	
`y = tan(x)`	
`y = asin(x)`	inverse trigonometrische Funktionen mit dem Rückgabewert y in Radiant
`y = sind(x)`	trigonometrische Funktionen mit dem Argument x in Grad
`y = asind(x)`	inverse trigonometrische Funktionen mit dem Rückgabewert y in Grad
`y = exp(x)`	Exponentialfunktion
`y = log(x)`	naturlicher Logarithmus
`y = log10(x)`	Logarithmus zur Basis 10
`y = sqrt(x)`	quadratische Wurzel
`y = abs(x)`	Absolutwert
`y = round(x)`	Rundung auf die nächste ganze Zahl
`y = floor(x)`	Abrunden auf die nächst kleinere ganze Zahl
`y = ceil(x)`	Aufrunden zur nächst grössere ganze Zahl
`y = conj(x)`	Konjugiert Komplexe Zahl von x

2.3 Funktionen zur Berechnung von Kennwerten

`y = sum(x)`	Summe der Elemente eines Arrays
`ma = max(x)`	grösster Wert in einem Arrays
`mi = min(x)`	kleinster Wert in einem Arrays
`m = mean(x)`	Mittelwert der Elemente eines Arrays
`s = std(x)`	Standardabweichung der Elemente eines Arrays
`v = var(x)`	Varianz der Elemente eines Arrays

2.4 Zeitmessung

`t = tic;`	Start Zeitmessung
`time = toc(t)`	Zeit (time) in Sekunden seit t mit tic erzeugt wurde

2.5 Graphische Funktionen

`figure(n)`	setzt das Plot-Fenster mit Nummer n als aktiv oder erzeugt ein neues Fenster mit dieser Nummer
`plot(x,y,'-o')`	zeichnet die Werte des Vektors y (Ordinate) gegen diejenige des Vektors x (Abszisse), wobei die Punkte linear interpoliert werden und mit Kreisen markiert werden
`plot(x1,y1,x2,y2)`	zeichnet zwei Funktionen in die selbe Graphik
`subplot(1,2,1)`	Plot auf linker Seite des Feldes (1 Zeile, 2 Spalten, 1. Graph)
`histogram(x)`	Erzeugung eines Histogramm-Plots
`stem(x)`	Plotten von diskreten Datensequenzen
`hold on`	Einfrieren der aktuellen Graphik, um zusätzliche Graphen eintragen zu können
`hold off`	Einfrierung aufheben
`axis([xs,xe,ys,ye])`	Festlegung der Skalierung der x- und der y-Achse
`title('Text')`	Titel für die Graphik
`xlabel('Text')`	Beschriftung der x-Achse (Abszisse)
`ylabel('Text')`	Beschriftung der y-Achse (Ordinate)

3 Signale

3.1 Spektrum

`y = fft(x)`	Berechnung der diskreten Fourier-Transformation von x
`x = ifft(y)`	Berechnung der inversen diskreten Fourier-Transformation von y
`y = fftshift(x)`	Verschiebung der "DC-Frequenz" (Frequenz Null) in die Mitte des Spektrums.
`x = ifftshift(y)`	Invertierung der Verschiebung von fftshift().

3.2 Filterung

`w = conv(u,v)`	Berechnung der Faltung von zwei Vektoren u und v
`y = filter(b,a,x)`	Filterung des Eingangssignals x mit dem Filter: $H(z) = \frac{b(1) + b(2)z^{-1} + ... + b(n_b + 1)z^{-n_b}}{a(1) + a(2)z^{-1} + ... + a(n_a + 1)z^{-n_a}}$
`y = cumsum(x)`	Berechnung der kumulativen Summe: $y(n) = \sum_{i=1}^{n} x(n)$

4 Bilder

4.1 Laden und Speichern

`I = imread('f')`	Bild mit dem Filenamen 'f' einlesen
`imwrite(I,'f')`	Bild I im File 'f' speichern

4.2 Anzeigen

`imshow(I)`	Bild I anzeigen
`image(I)`	Bild I anzeigen. Verwendet eine colormap zum Anzeigen von Graubildern.
`imagesc(I)`	Skaliert die Bilddaten I zur vollständigen Farbpalette der aktiven colormap
`colormap(m)`	Jede Reihe in m ist ein RGB-Vektor, welcher eine Farbe definiert.
`colormap('gray')`	erzeugt und aktiviert eine lineare Graustuffen colormap
`I = rgb2gray(RGB)`	konvertiert das Farbbild RGB zum Graubild I

4.3 Filterung

`C = conv2(A,B)`	zweidimensionale Faltung mit A und B
`F = filter2(h,I)`	Bild I mit der zweidimensionalen Matrix h filtern
`F = medfilt2(I,[m n])`	Medianfilter mit m mal n Nachbarschaft

4.4 Spektrum

`F = fft2(I)`	2-dimensionale digitale Fouriertransformation
`I = ifft2(F)`	inverse 2-dimensionale digitale Fouriertransformation

5 Kontrollstrukturen

5.1 for-Schleife

```
for variable = <expression>
     <statements>
end
```

```
for i = 1:10
     y = y + x(i);
end
```

5.2 while-Schleife

```
while <expression>
     <statements>
end
```

```
while i<10
      i = i*2;
end
```

5.3 If-Anweisung

```
if <expression>
     <statements>
elseif <expression>
     <statements>
else
     <statements>
end
```

```
if i>=4
     y = 1;
elseif i<-2
     y = -1;
else
     y = 0;
end
```

5.4 Eigene Funktionen

```
function <outputs>=<name>(<inputs>)
     <statements>
end
```

```
function c = myMultiplication(a,b)
     c = a*b;
end
```