MATLAB
Eine Einführung

R. Frühwirth

Institut für Hochenergiephysik der
Österreichischen Akademie der Wissenschaften

und

Studiengang Bauingenieurwesen - Baumanagement
fh-campus wien

Seminar am fh-campus wien

20.–21. Februar 2004

Inhaltsverzeichnis

1 Allgemeine Struktur

- MATLAB ist ein Programm der Firma MathWorks Inc. Der Name kommt nicht von „Mathematik“, sondern von „Matrix“, da MATLAB ursprünglich ein Programm zur Vektor- und Matrizenrechnung war. Mittlerweile enthält der Kern noch viele andere Funktionen, von denen wir die wichtigsten kennenlernen werden.
- Der Kern enthält eingebaute Funktionen und Funktionen, die als MATLAB-Quellcode vorliegen.
- Neben dem Kern gibt es noch sogenannte „Toolboxes“, also Werkzeugkisten, die zahlreiche Funktionen für spezielle Anwendungen enthalten. Wir verwenden nur die „Symbolic Toolbox“, die die numerischen Funktionen des Kerns durch symbolische ergänzt.
- Eine Übersicht über sämtliche Toolboxes wird zum Abschluß gegeben.

2 Arbeitsweise

- Wir verwenden MATLAB vorwiegend interaktiv. Das heißt, dass MATLAB die eingegeben Befehle sofort interpretiert und ausführt. Wenn gewünscht, wird das Ergebnis sofort angezeigt.
- Es können vom Benutzer jedoch auch Funktionen und komplette Programme erstellt werden. Soll etwa eine Sequenz von Anweisungen wiederholt ausgeführt werden, genügt es, sie in einer Datei abzuspeichern. Der Typ der Datei muss `.m` sein. Das Eingeben des Dateinamens führt dann die gespeicherte Sequenz aus.
- Soll eine Sequenz wiederholt mit verschiedenen Eingaben ausgeführt werden, empfiehlt es sich, eine Funktion mit Ein- und Ausgabeparametern zu definieren.

3 Anweisungen und Kommentare

- Im interaktiven Modus können Anweisungen eingegeben werden, wenn das MATLAB-Prompt erscheint (`>>`).
- Eine MATLAB-Anweisung kann durch das Zeilende (CR, Enter), durch einen Strichpunkt (`;`) oder durch das Prozentzeichen (`%`) abgeschlossen werden.
- Alle Zeichen hinter dem Prozentzeichen gelten als Kommentar und werden ignoriert.
- Eine MATLAB-Anweisung kann durch die Ellipsis (`...`) in die nächste Zeile fortgesetzt werden.
- Der Strichpunkt verhindert die Anzeige des Resultats der Anweisung.
- Es können auch mehrere Anweisungen in eine Zeile geschrieben werden, getrennt durch Komma oder Strichpunkt. Das Komma erlaubt, der Strichpunkt verhindert die Anzeige des Resultats.

```
>> a=1,b=1;c=3
>> a,b,c
```

4 Tagebuch- und Hilfefunktion

4.1 Tagebuch

- Die Arbeit mit MATLAB kann in einem „Tagebuch“ (diary) aufgezeichnet werden.
- Einschalten des Tagebuchs:

```
>> diary Seminar % Tagebuch einschalten
```

- Ausschalten des Tagebucgs:

```
>> diary off % Tagebuch ausschalten
```

- Die Datei `Seminar` kann anschließend mit einem Texteditor betrachtet werden. Es kann der MATLAB-interne Editor verwendet

werden:

```
>> edit Seminar
```

4.2 Hilfe

- Ist die Arbeitsweise einer Funktion nicht bekannt, kann man die interaktive Hilfe zu Rate ziehen:

```
>> help linspace % Anzeige im Befehlsfenster
>> helpwin linspace % Anzeige in einem neuen Fenster
```

- Ausführliche Dokumentation im Web-Browser:

```
>> helpdesk
```

- Der Pfad einer Funktion kann mit dem Befehl

```
>> which linspace
```

  oder

```
>> which('linspace')
```

  ermittelt werden.

- Ist der Pfad bekannt, kann der Quellcode inspiziert werden:

```
>> edit(which('linspace'))
```

- Stichwortsuche nach Funktionen:

```
>> lookfor Bessel
```

5 Verzeichnisse und Suchpfade

5.1 Verzeichnis- und Dateiverwaltung

- Anzeige des aktuellen Verzeichnisses:

```
>> pwd
```

- Ändern des aktuellen Verzeichnisses:

```
>> cd dir % Gehe in das Verzeichnis dir
>> cd .. % Gehe in das nächsthöhere Verzeichnis
```

- Anzeige der Dateien im aktuellen Verzeichnis:
  ```
  >> dir
  >> ls
  ```
- Anzeige der MATLAB-Dateien im aktuellen Verzeichnis:
  ```
  >> what
  ```
- Anzeige der MATLAB-Dateien in anderen Verzeichnissen:
  ```
  >> what toolbox/matlab/general
  ```
- Ausgabe einer Datei am Bildschirm:
  ```
  >> type test.m
  ```
- Blättern am Bildschirm:
  ```
  >> more on
  >> more off
  ```
- Löschen einer Datei:
  ```
  >> delete test.m
  ```
- Abfrage, ob eine Datei existiert:
  ```
  >> exist test.m
  ```

5.2 Anzeigen und Verändern von Suchpfaden

- Wenn eine Funktion aufgerufen wird, durchsucht MATLAB eine Liste von Verzeichnissen, den sogenannten Suchpfad (search path). Sobald eine Funktion gefunden wird, wird die Suche abgebrochen.
- Eine existierende MATLAB-Funktion kann somit vom Benützer ersetzt werden.
- Anzeige des aktuellen Suchpfades:
  ```
  >> path
  ```
- Hinzufügen von Verzeichnissen:
  ```
  >> addpath dir1 dir2 % am Anfang der Liste
  >> addpath dir3 dir4 -end% am Ende der Liste
  ```
- Entfernen von Verzeichnissen:
  ```
  >> rmpath dir1 dir2
  ```

- Verketten zweier Suchpfade:
  ```
  >> path(p1,p2)
  ```

6 Der MATLAB-Editor

- MATLAB hat einen eingebauten Editor, der mit dem Befehl `edit` aktiviert wird:
  ```
  >> edit test.m % Öffne Datei test.m
  >> edit % Öffne neue Datei
  ```
- Der Editor ist sensitiv auf die MATLAB-Syntax.
- Der Editor dient gleichzeitig als Debugger (siehe unten).

7 Numerische Datentypen und arithmetische Operatoren

7.1 Numerische Konstante

- Numerische Konstante können mit Dezimalpunkt (Festkommaformat) oder im Gleitkommaformat eingegeben werden:
  ```
  >> 1.23
  >> 4.07e6
  >> 2-6.5i % Komplexe Konstante
  ```
- Die Kreiszahl π ist bereits vorprogrammiert:
  ```
  >> pi
  ```
- Die Anzahl der angezeigten Dezimalstellen kann verändert werden:
  ```
  >> format long
  >> pi
  >> format short
  >> pi
  >> format bank
  >> pi
  >> format rat
  >> pi
  ```

- Neben den numerischen Konstanten gibt es noch die Konstanten `NaN` (not a number) und `inf` (unendlich):

```
>> 1/0
>> -1/0
>> 0/0
>> 1/inf
>> 1/nan
>> -inf/inf
```

7.2 Numerische Variable

- Der Name einer Variablen besteht aus maximal 31 Buchstaben oder Ziffern, wobei das erste Zeichen ein Buchstabe sein muss. Das Zeichen „_“ (underscore) ist ab der zweiten Stelle ebenfalls zugelassen. Klein- und Großbuchstaben werden als verschieden betrachtet. Variable können sofort, ohne vorherige Deklaration verwendet werden.

- Numerische Variable können reell oder komplex sein.

- Zuweisung eines numerischen Werts an eine Variable:

```
>> a0=3 % Mit Anzeige
```

- Der zugewiesene Wert wird sofort angezeigt. Die Anzeige wird unterdrückt, wenn die Anweisung durch einen Strichpunkt abgeschlossen wird:

```
>> a0=3; % Keine Anzeige
```

  Nach dem Prozentzeichen können Kommentare angebracht werden, die von MATLAB ignoriert werden.

- Anzeige des Werts einer Variablen:

```
>> a0
```

- Liste aller Variablen:

```
>> who
```

- Ausführliche Information über eine Variable:

```
>> whos a0
>> whos('a0')
```

- Ausführliche Information über alle Variablen:
  ```
  >> whos
  ```
- Abfrage, ob eine Variable existiert:
  ```
  >> exist a0
  >> exist('a0')
  ```
- Reell oder komplex:
  ```
  >> isreal(a0)
  ```
- Löschen einer Variablen:
  ```
  >> clear a0
  >> clear('a0')
  ```
- Löschen aller Variablen:
  ```
  >> clear
  ```

7.3 Arithmetische Operatoren

- Addition:
  ```
  >> b0=a0+4.21
  ```
- Subtraktion:
  ```
  >> c0=b0-18
  ```
- Multiplikation:
  ```
  >> d=b0*c0
  ```
- Division:
  ```
  >> f=32/65
  ```
- Rest der Division:
  ```
  >> r=mod(8,3)
  ```
- Potenzieren und Wurzelziehen:
  ```
  >> z=3^5
  >> z1=z^(1/2)
  >> z1=sqrt(z)
  ```
- Klammern:
  ```
  >> x=1/(2+sqrt(3*(1+13/17)))
  ```

7.4 Arithmetische Ausdrücke

- Ein arithmetischer Ausdruck wird mit Hilfe von arithmetischen Operatoren und von Klammern gebildet. Dazu können noch Funktionen treten, wie etwa die Betragsfunktion oder die weiter unten aufgelisteten elementaren Funktionen.
- Wird ein arithmetischer Ausdruck keiner Variablen zugewiesen, wird das Resultat in der Variablen `ans` abgespeichert.

7.5 Betrag und Vorzeichen

- Betrag:
  ```
  >> ax=abs(x)
  ```
- Vorzeichen:
  ```
  >> sx=sign(x)
  ```

7.6 Primzahlen und Primfaktoren

- Primfaktoren einer ganzen Zahl:
  ```
  >> factor(3124)
  ```
- Alle Primzahlen bis zu einer Zahl:
  ```
  >> primes(100)
  ```
- Primalität:
  ```
  >> isprime(97)
  >> isprime(99)
  ```

7.7 Aufgaben

Aufgabe: Bestimmen Sie die Primfaktorzerlegungen der folgenden Zahlen:

$$2^2+1, 2^4+1, 2^8+1, 2^{16}+1, 2^{32}+1$$

Aufgabe: Berechnen Sie den goldenen Schnitt mit der Formel

$$\varphi = \frac{1+\sqrt{5}}{2}$$

und zeigen Sie, dass

$$\varphi - \frac{1}{\varphi} = 1$$

gilt.

8 Logische Variable und logische Operationen

- Logische Variable haben den Wert 0 (falsch) oder 1 (wahr).
- Sie entstehen als Resultat eine Vergleichsoperation:

```
>> l=(5==5)
>> l=~(5==5)
>> l=(5~=5)
>> l=(5<5)
>> l=(5<=5)
>> l=(5>5)
>> l=(5>=5)
>> islogical(l)
```

- Logische Verknüpfungungen:

```
>> l1&l2 % und
>> l1|l2 % oder
>> xor(l1,l2) % ausschliessendes oder
```

9 Vektoren

9.1 Numerische Vektoren

- Konstante Vektoren werden von eckigen Klammern eingeschlossen.
- Variable Vektoren werden nach den gleichen Regeln bezeichnet wie einfache Variable.
- Es gibt einen leeren Vektor:

```
>> v=[]
```

- Abfrage, ob ein Vektor leer ist:
  ```
  >> isempty(v)
  ```
- Die Komponenten von Zeilenvektoren werden durch Leerzeichen oder Kommas getrennt:
  ```
  >> vz=[1,4,6,10,-5] % Zeilenvektor mit 5 Komponenten
  ```
- Die Komponenten von Spaltenvektoren werden durch Strichpunkte getrennt:
  ```
  >> vs=[1;4;6;10;-5] % Spaltenvektor mit 5 Komponenten
  ```
- Vektoren mit konstanter Schrittlänge:
  ```
  >> v1=1:10
  >> v2=1:3:10
  >> v3=10:-1:1
  >> v4=linspace(0,10,21)
  ```
  Diese Vektoren sind automatisch Zeilenvektoren.
- Auffüllen eines Vektors mit Null:
  ```
  >> o=zeros(1,n) % Zeilenvektor mit n Nullen
  >> o=zeros(n,1) % Spaltenvektor mit n Nullen
  ```
- Auffüllen eines Vektors mit einer Zahl:
  ```
  >> c=3*ones(1,n) % Zeilenvektor mit n mal 3
  >> c=3*ones(n,1) % Spaltenvektor mit n mal 3
  ```
- Durch Transponieren können Zeilen- in Spaltenvektoren (und umgekehrt) umgewandelt werden:
  ```
  >> vzt=vz' % vzt ist ein Spaltenvektor
  >> vst=vs' % vst ist ein Zeilenvektor
  ```
- Verkettung von Vektoren:
  ```
  >> a=[1,2,3]
  >> b=[4,5,6]
  >> c=[a,b,7,8,[9,10]]
  >> d=[a';b';7;8;[9;10]]
  >> e=[a,b'] % Falsch! <<<<<<<<<
  ```

9.2 Logische Vektoren

- Ein logischer Vektor entsteht durch Vergleich zweier Vektoren:
  ```
  >> l=(a<b)
  ```
- Logische Vektoren können zur Auswahl von Komponenten benützt werden.

9.3 Zeichenketten (Strings)

- Zeichenketten (strings) sind Vektoren von Zeichen.
- Konstante Zeichenketten werden in einfaches Hochkomma eingeschlossen:
  ```
  >> s='a1b2C*'
  >> whos s
  >> ischar(s)
  ```
- Verkettung von Zeichenketten:
  ```
  >> s1='ABC!'
  >> s2='135'
  >> s3=[s1 s2]
  ```

9.4 Komponenten von Vektoren

- Komponenten von Vektoren werden mit runden Klammern adressiert, wobei die erste Komponente Index 1 hat.
- Erste Komponente:
  ```
  >> a1=a(1)
  ```
- Komponente i:
  ```
  >> ai=a(i)
  ```
- Vorletzte Komponente:
  ```
  >> aend=a(end-1) % oder
  >> aend=a(length(a)-1)
  ```
- Letzte Komponente:
  ```
  >> aend=a(end) % oder
  >> aend=a(length(a))
  ```

- Komponenten i und j:
  ```
  >> aiundj=a([i,j])
  ```
- Komponenten i bis j:
  ```
  >> aibisj=a(i:j)
  ```
- Alle Komponenten mit ungeraden Indizes:
  ```
  >> aodd=a(1:2:end)
  ```
- Alle Komponenten mit geraden Indizes:
  ```
  >> aeven=a(2:2:end)
  ```
- Alle positiven Komponenten:
  ```
  >> apos=a(a>0)
  ```
- Alle negativen Komponenten:
  ```
  >> apos=a(a<0)
  ```
- Indizes aller Komponenten gleich 0:
  ```
  >> i0=find(a==0)
  >> a(i0)
  ```
- Indizes aller Komponenten ungleich 0:
  ```
  >> i1=find(a~=0)
  >> a(i1)
  ```

9.5 Vektoroperationen

- Vektoraddition:
  ```
  >> c=a+b
  >> a2=a+2
  ```
- Vektorsubtraktion:
  ```
  >> d=a-b
  ```
- Linearkombination:
  ```
  >> e=2.2*a-3.5*b
  ```
- Inneres Produkt oder Skalarprodukt:
  ```
  >> s=dot(a,b)
  ```

- Äußeres Produkt oder Vektorprodukt:
```
>> v=cross(a,b) % Nur für Dimension 3
```
- Komponentenweise Multiplikation:
```
>> p=a.*b
```
- Komponentenweise Division:
```
>> d=a./b
>> bb=1./b
```
- Komponentenweises Potenzieren:
```
>> a2=a.^2
>> aa=2.^a
```
- Betrag:
```
>> na=norm(a)
```

9.6 Weitere Vektorfunktionen

- Länge:
```
>> x=-5:0.1:5
>> l=length(x)
```
- Sortieren:
```
>> xsort=sort(x)
```
- Kleinster und größter Wert:
```
>> minx=min(x) % kleinster Wert
>> maxx=max(x) % größter Wert
```
- Mittelwert:
```
>> m=mean(x)
```
- Median:
```
>> m=median(x)
```
- Varianz und Standardabweichung:
```
>> v=var(x)
>> s=std(x)
```

- Summe aller Komponenten:
```
>> s=sum(a)
>> t=sum(1:100)
```
- Teilsummenvektor:
```
>> cs=cumsum(1:10)
```
- Produkt aller Komponenten:
```
>> nfac=prod(1:n) % n!
>> Cnk=prod(n:-1:n-k+1)/prod(1:k) % Binomialkoeffizient
```
- Differenzenvektor:
```
>> d=diff(x)
>> length(x)
>> length(d)
```
- Trapezsumme:
```
>> I=trapz(x,f)
```
- Komponentenweiser Betrag:
```
>> ax=abs(x)
```
- Komponentenweises Vorzeichen:
```
>> sx=sign(x)
```
- Häufigkeitsverteilung (Histogramm):
```
>> h=hist(x,xh)
```

9.7 Logische Vektorfunktionen

- Test auf `NaN` und `inf`:
```
>> x=1/0
>> isnan(x)
>> isinf(x)
>> isfinite(x)
```
- Test auf Nullvektor:
```
>> z=zeros(1,5)
>> any(z)
```

- Test auf Nullelemente:
  ```
  >> o=ones(1,5)
  >> all(z)
  ```

9.8 Graphische Darstellung

- Vektoren können durch Punkt- und durch Stabdiagramme dargestellt werden.
- Punktdiagramme werden mit der Funktion `plot` erzeugt:
  ```
  >> x=[1 0 2 0 3 0 4 0 5 0 6]
  >> plot(x,'b+')
  ```
- Es kann auch ein Vektor als Funktion eines anderen Vektors gezeichnet werden:
  ```
  >> x=-4:0.5:4
  >> y=x.^2
  >> plot(x,y,'ro-')
  ```
- Das letzte Argument bestimmt die Farbe, das verwendete Symbol und die Art der Verbindungslinie.
- Stabdiagramm werden mit der Funktion `bar` erzeugt:
  ```
  >> figure(1)
  >> bar(x)
  >> figure(2)
  >> bar(x,0.2)
  ```
- Stabdiagramm eines Vektors als Funktion eines anderen Vektors:
  ```
  >> bar(x,y)
  ```

9.9 Aufgaben

Aufgabe: Die zwei Zeilenvektoren $\vec{a}$ und $\vec{b}$ sind gegeben als:

$$\vec{a} = \begin{pmatrix} 1 & 4 & -2 \end{pmatrix}, \quad \vec{b} = \begin{pmatrix} 2 & -3 & 5 \end{pmatrix}$$

Berechnen Sie

$$\vec{c} = 2\,\vec{a} - 3\,\vec{b}, \quad \vec{a} \cdot \vec{b}, \quad \vec{b} \times \vec{c}$$

10 Matrizen

10.1 Aufbau von Matrizen

- Die Matrix ist der grundlegende Datentyp in MATLAB.
- Einfache Variable und Vektoren sind Spezialfälle von Matrizen.
- Konstante Matrizen werden in eckige Klammern eingeschlossen.
- Abfrage, ob eine Matrix leer ist:
  ```
  >> isempty(A)
  ```
- Elemente einer Zeile werden durch Leerzeichen oder Beistriche getrennt, Zeilen werden durch Strichpunkte oder das Zeilenende getrennt:
  ```
  >> A=[1 3 5;2 4 6;3 5 7]
  ```
  oder
  ```
  >> A=[1 2 3
  2 4 6
  3 5 7]
  ```
- Matrizen können auch aus Zeilen- oder Spaltenvektoren zusammengesetzt werden:
  ```
  >> z1=[1 3 5] % erste Zeile
  >> z2=[2 4 6] % zweite Zeile
  >> z3=[3 5 7] % dritte Zeile
  >> A=[a1;a2;a3]
  >> s1=[1;2;3] % erste Spalte
  >> s2=[3;4;5] % zweite Spalte
  >> s1=[5;6;7] % dritte Spalte
  >> A=[s1 s2 s3]
  ```

10.2 Komponenten von Matrizen

- Komponenten von Matrizen werden mit runden Klammern adressiert (Zeilenindex, Spaltenindex).

- Element in Zeile i und Spalte j:
  ```
  >> Aij=A(i,j)
  ```
- Zeile i:
  ```
  >> zeilei=A(i,:)
  ```
- Spalte j:
  ```
  >> spaltei=A(:,j)
  ```
- Untermatrix:
  ```
  >> B=A(i1:i2,j1:j2)
  ```
- Alle Zeilen mit geraden Indizes:
  ```
  >> A1=A(2:2:end,:)
  ```
- Alle Spalten mit ungeraden Indizes:
  ```
  >> A=A(:,1:2:end)
  ```

10.3 Matrixoperationen

- Transposition:
  ```
  >> At=A'
  ```
- Matrixaddition:
  ```
  >> S=A+B
  >> A2=A+2
  ```
- Matrixsubtraktion:
  ```
  >> D=A-B
  ```
- Linearkombination:
  ```
  >> E=2*A-pi*B
  ```
- Matrixmultiplikation:
  ```
  >> C=A*B
  ```
- Kronecker- oder Tensorprodukt:
  ```
  >> T=kron(A,B) % Nur für Dimension 3
  ```
- Inverse Matrix:
  ```
  >> invA=inv(A)
  ```

- Pseudoinverse Matrix:
  ```
  >> pinvA=pinv(A)
  ```
- Komponentenweise Multiplikation:
  ```
  >> p=a.*b
  ```
- Komponentenweise Division:
  ```
  >> d=a./b
  >> bb=1./b
  ```
- Komponentenweises Potenzieren:
  ```
  >> a2=a.^2
  >> aa=2.^a
  ```
- Norm:
  ```
  >> normA=norm(A)
  ```
- Determinante:
  ```
  >> detA=det(A)
  ```
- Rang:
  ```
  >> rangA=rank(A)
  ```

10.4 Eigenwerte und Eigenvektoren

- Eigenwerte:
  ```
  >> eA=eig(A)
  ```
- Eigenvektoren:
  ```
  >> [U,eA]=eig(A)
  ```
- Singulärwertzerlegung:
  ```
  >> [U,S,V] = SVD(A) % A = U*S*V'
  ```

10.5 Weitere Matrixfunktionen

- Dimension:
  ```
  >> [nz,ns]=size(A)
  ```
- Hauptdiagonale:
  ```
  >> d=diag(A)
  ```

- Nebendiagonalen:
  ```
  >> d=diag(A,k)
  ```
- Diagonalmatrix mit gegebener Diagonale:
  ```
  >> D=diag(d)
  ```
- Einheitsmatrix der Ordnung n:
  ```
  >> I=eye(n)
  ```
- Nullmatrix der Ordnung n:
  ```
  >> I=zeros(n)
  ```
- Magisches Quadrat der Ordnung $n \neq 2$:
  ```
  >> M=magic(n)
  ```
- Kleinster und größter Wert:
  ```
  >> min1A=min(A) % Zeilenvektor der Spaltenminima
  >> min2A=min(A,2) % Spaltenvektor der Zeilenminima
  >> minA=min(min(A)) % kleinster Wert
  >> max1A=max(A) % Zeilenvektor der Spaltenmaxima
  >> max2A=max(A,2) % Spaltenvektor der Zeilenmaxima
  >> maxA=max(max(A)) % größter Wert
  ```
- Summe:
  ```
  >> s1=sum(x) % Zeilenvektor der Spaltensummen
  >> s2=sum(x,2) % Spaltenvektor der Zeilensummen
  >> m=sum(sum(A))
  ```
  Analog für das Produkt, den Mittelwert und den Median
- Komponentenweiser Betrag:
  ```
  >> absA=abs(A)
  ```
- Replikation:
  ```
  >> repA=repmat(A,m,n)
  ```
- Lösung eines linearen Gleichungssystems:
  ```
  >> A=[1 -3 4;-2 5 7;3 8 1] % Koeffizientenmatrix
  >> b=[2;-5;4] % rechte Seite
  >> x=A\b % Lösung
  ```

10.6 Graphische Darstellung

- Matrizen können zwei- oder dreidimensional dargestellt werden.
- Jede Spalte wird in einer anderen Farbe dargestellt.
- Zweidimensionales Punktdiagramm mit der Funktion `plot`:

```
>> M=magic(4)
>> plot(M)
```

- Zweidimensionales Stabdiagramm mit der Funktion `bar`:

```
>> bar(M)
```

- Dreidimensionales Stabdiagramm mit der Funktion `bar3`:

```
>> M=magic(4)
>> bar3(M)
```

10.7 Aufgaben

Aufgabe: Erzeugen Sie die magischen Quadrate der Ordnung 3,4,5 und 6. Bestimmen Sie die magische Konstante und die Eigenwerte. Zeigen Sie, dass alle Spalten-, Zeilen- und Diagonalsummen gleich der magischen Konstanten sind. Zeigen Sie, dass die magische Konstante immer ein Eigenwert ist und dass für gerade Ordnung auch 0 ein Eigenwert ist. Bestimmen Sie einen Eigenvektor zur magischen Konstante.

Aufgabe: Die Matrix A ist gegeben durch:

$$A = \begin{pmatrix} 1 & 2 & 2 \\ 4 & 5 & 6 \end{pmatrix}$$

Die Matrix B geht aus A durch Multiplikation der ersten (zweiten) Zeile mit 2 (5) hervor.

Aufgabe: Die Matrix A ist gegeben durch:

$$A = \begin{pmatrix} 1 & 2 & 2 \\ 4 & 5 & 6 \end{pmatrix}$$

Die Matrix B geht aus A durch Addition des Vektors $v = (0\ 1\ 2)$ zu jeder Zeile hervor.

11 Felder (Arrays)

11.1 Numerische Felder

- Analog zu Matrizen können auch Felder mit drei oder mehr Indizes generiert werden:

```
>> M(:,:,1)=[1 3 5;2 4 6;3 5 7]
>> M(:,:,2)=-M(:,:,1)
>> M1=M(1,:,:)
>> squeeze(M1)
>> M2=M(:,2,:)
>> squeeze(M2)
```

- Entfernen von redundanten Dimensionen mit der Funktion `squeeze`.

11.2 Cell Arrays

- Cell Arrays sind Felder, die heterogene Objekte enthalten können.
- Die Objekte werden mit geschwungenen Klammern adressiert:

```
>> a{1}=pi
>> a{2}='abc'
>> a{3}=magic(3)
```

- Alternativ kann folgende Notation verwendet werden:

```
>> a={pi 'abc' magic(3)}
```

- Die Funktion `cell` erzeugt ein leeres Cell Array der gewünschten Größe:

```
>> b=cell(2,3)
```

- Die Funktion `celldisp` zeigt alle Objekte an:

```
>> celldisp(a)
```

- Ist ein Objekt ein Vektor, eine Matrix oder eine Zeichenkette, können Elemente des Objekts auf die übliche Weise adressiert werden:

```
>> a{3}(3,2)
>> a{2}(2)
```

11.3 Strukturen

- Strukturen sind spezielle Felder. Jedes Element hat ein oder mehrere Datenfelder, die über einen Namen adressiert werden:

```
>> student(1).name={'Amanda' 'Breil'}
>> student(1).birth=[27 01 1969]
>> student(1).number=98333558
>> student(2).name={'Christian' 'Dior'}
>> student(2).birth=[12 03 1968]
>> student(2).number=98333559
```

- Alternativ kann die Funktion `struct` verwendet werden:

```
>> student=struct(...
'name',{'Amanda Breil','Christian Dior'},...
'birth',{[27 01 1969],[12 03 1968]},...
'number',{98333558,98333559})
```

- Größere Strukturen können natürlich in Schleifen aufgefüllt werden.

- Ist ein Datenfeld ein Vektor, eine Matrix eine Zeichenkette oder ein Cell Array, können Elemente auf die übliche Weise adressiert werden:

```
>> student(1).birth(3)
>> student(2).name{1}
>> student(2).name{1}(1)
```

- Datenfelder können auch mit den Funktionen `setfield` und `getfield` gelesen bzw. verändert werden.

- Die Namen der Datenfelder können mit der Funktion `filednames` angezeigt werden:

```
>> fieldnames(student)
```

- Entfernen von Datenfeldern:

```
>> student1=rmfield(student,'number')
```

- Ein Datenfeld eines Strukturelements kann wieder eine Struktur sein.

12 Polynome

- Auswertung eines Polynoms:
  ```
  >> c=[3 1 2 1] % Koeffizienten in absteigender Ordnung
  >> y=polyval(c,x)
  ```
- Nullstellen eines Polynoms:
  ```
  >> x=roots(c)
  ```
- Koeffizienten aus den Nullstellen berechnen:
  ```
  >> c=poly(x)
  ```

Aufgabe: Bestimmen Sie die Koeffizienten des Polynoms mit den Nullstellen $1, 2, 3, \ldots, 19, 20$. Berechnen Sie die Nullstellen dieses Polynoms. Bestimmen Sie die relativen Fehler und stellen Sie sie graphisch dar.

13 Elementare Funktionen

- MATLAB kann alle Funktionen berechnen, die auf Taschenrechnern verfügbar sind, die sogenannten elementaren Funktionen.
- Sie umfassen die Winkelfunktionen und ihre Umkehrfunktionen (Arcusfunktionen), die Exponentialfunktionen und ihre Umkehrfunktionen (Logarithmusfunktionen), sowie die Hyperbelfunktionen und ihre Umkehrfunktionen (Areafunktionen).
- Alle elementaren Funktionen akzeptieren neben Skalaren auch Vektoren oder Matrizen als Eingabe und geben in diesem Fall wieder Vektoren oder Matrizen der gleichen Form als Resultat. Zum Beispiel:
  ```
  >> x=[0:pi/20:pi]'
  >> y=sin(x)
  >> z=log10(1:0.5:10)
  ```

13.1 Winkel- und Arcusfunktionen

- Die Argumente der Winkelfunktionen müssen in Radianten angegeben werden. Die Resultate der Arcusfunktionen sind ebenfalls in Radianten.

- Winkelfunktionen:
```
>> y=sin(x)
>> y=cos(x)
>> y=tan(x)
>> y=cot(x)
```
- Arcusfunktionen:
```
>> x=asin(y)
>> x=acos(y)
>> x=atan(y)
>> x=acot(y)
```

13.2 Exponential- und Logarithmusfunktionen

- Exponentialfunktionen:
```
>> y=exp(x) %  Basis e
>> y=a.^x % Basis a
>> y=exp(x*log(a)) %  Basis a
```
- Logarithmusfunktionen:
```
>> y=log(x) % Basis e
>> y=log2(x) % Basis 2
>> y=log10(x) % Basis 10
>> y=log(x)/log(a) % Basis a
```

13.3 Hyperbel- und Areafunktionen

- Hyperbelfunktionen:
```
>> y=sinh(x)
>> y=cosh(x)
>> y=tanh(x)
>> y=coth(x)
```
- Areafunktionen:
```
>> x=asinh(y)
>> x=acosh(y)
>> x=atanh(y)
>> x=acoth(y)
```

13.4 Matrixfunktionen

- Matrixpolynom:
  ```
  >> B=polyvalm(c,A)
  ```
- Matrixwurzel:
  ```
  >> S=sqrtm(A)
  ```
  Ist nicht immer definiert.
- Matrixexponentialfunktion:
  ```
  >> B=expm(A)
  ```
- Matrixlogarithmus:
  ```
  >> A=logm(B)
  ```
  Ist nicht immer definiert.
- Allgemeine Matrixfunktion:
  ```
  >> B=funm(A,'sin')
  >> B=funm(A,'cos')
  >> B=funm(A,'tanh')
  ```

13.5 Aufgaben

Aufgabe: Überprüfen Sie anhand der Werte $x = 1, 1.1, 1.2, \ldots, 2$ die Identität

$$\operatorname{acosh}(x) = \ln(x + \sqrt{x^2 - 1})$$

Aufgabe: Überprüfen Sie anhand der Zahl $z = 1 + i$ die Identität

$$\sin(x + iy) = \sin(x)\cosh(y) + i\,\cos(x)\sinh(y)$$

Aufgabe: Erzeugen Sie mit Hilfe der Funktion `rand` eine symmetrische Matrix A der Dimension 3×3. Berechnen Sie die unitäre Matrix U, deren Spalten die Eigenvektoren von A sind. Überprüfen Sie die folgenden Beziehungen:

$$e^{UAU'} = Ue^AU', \quad \ln(UAU') = U\ln(A)U', \quad \ln(e^A) = A, \quad e^{\ln(A)} = A$$

Wie hängen die Eigenwerte von e^A mit den Eigenwerten von A zusammen?

Aufgabe: Überprüfen Sie anhand der Matrix A aus dem letzten Beispiel die Beziehung

$$\sin(2A) = 2\,\sin(A)\cos(A)$$

Wie hängen die Eigenwerte von $\sin(A)$ mit den Eigenwerten von A zusammen?

14 Höhere Funktionen

Die folgenden höheren Funktionen sind verfügbar:

- Airyfunktionen: `airy`
- Besselfunktionen 1. und 2. Art: `besselj, bessely`
- Modifizierte Besselfunktionen 1. und 2. Art: `besseli, besselk`
- Hankelfunktionen: `besselh`
- Betafunktionen: `beta, betainc, betaln`
- Elliptische Funktionen: `ellipj, ellipke`
- Fehlerfunktionen: `erf, erfc, erfcx, erfinv`
- Exponentialintegral: `expint`
- Gammafunktionen: `gamma, gammainc, gammaln`
- Assoziierte Legendrefunktionen: `legendre`

Aufgabe: Überprüfen Sie für einige ganzzahlige Werte die Beziehung

$$\Gamma(n) = (n-1)!$$

wobei $\Gamma(n)$ die Gammafunktion bezeichnet.

Aufgabe: Überprüfen Sie für einige Werte die Beziehung

$$B(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

wobei $B(a,b)$ die Betafunktion bezeichnet.

15 Eine Auswahl weiterer MATLAB-Funktionen

- Ein-und Ausgabe: `input, disp, fprintf, fscanf, sprintf, fwrite`
- Speichern und Laden des Kontexts: `save, load`
- Zeitmessung: `tic, toc, etime, cputime, clock`
- Fehlermeldung: `error`
- Auswertung einer Zeichenkette als MATLAB-Anweisung: `eval`
- Konversion: `num2str, int2str, str2num, hex2num, hex2dec, dec2hex, bin2dec, dec2bin, base2dec, dec2base`
- String-Funktionen: `findstr, strcmp, strmatch, strcat, strvcat, str2mat`
- Zufallszahlen: `rand, randn`
- Kombinatorik: `nchoosek, perms, permute, randperm`
- Koordinatentransformationen: `cart2pol, cart2sph, pol2cart, sph2cart`
- Kettenbruchentwicklung: `rat`
- Rationale Approximation: `rats`
- Vektoren mit konstanten Differenzen: `linspace`
- Vektoren mit konstanten Quotienten: `logspace`
- Numerische Integration: `quad, quad8, dblquad`
- Trapezregel: `trapz`
- Spiegeln und Drehen von Matrizen: `fliplr, flipud, flipdim, rot90`
- Faktorisierung von Matrizen: `lu, qr, luinc, chol, ...`

- Lösen von linearen Gleichungssystemen: `cgs, bicg, bicgstab, gmres, pcg, qmr`
- Dünn besetzte Matrizen: 45 Funktionen
- Lösen von gewöhnlichen Differenzialgleichungen: `ode45, ode23, ode113, ode15s, ode23s`
- Und viele andere ...!

16 Graphische Darstellung von Funktionen

- Graph einer Funktion mit `fplot`:
```
>> fplot('sin',[-pi pi])
```
- Hinzufügen der Koordinatenachsen:
```
>> hold on
>> plot([-pi pi],[0 0],'k')
>> plot([0 0],[-1 1],'k')
```
- Hinzufügen der Achsenbeschriftung:
```
>> xlabel('x')
>> ylabel('y')
```
- Hinzufügen der Legende:
```
>> legend('sin(x)')
>> ylabel('y')
```
- Graph einer Funktion von zwei Variablen:
```
>> x=linspace(-pi,pi,21);
>> x=linspace(-pi,pi,21);
>> [xx,yy]=meshgrid(x,x);
>> z=sin(xx).*cos(yy);
>> figure(1)
>> surf(xx,yy,z) >> figure(2)
>> contour(xx,yy,z)
```

Aufgabe: Stellen Sie die Funktion $f(x) = (x-1)/(x+1)$ im Intervall $[-3, 5]$ graphisch dar. Zeichnen Sie die Achsen und die Asymptoten ein und fügen Sie eine Legende hinzu.

Aufgabe: Stellen Sie die Funktion $f(x, y) = x^2 - xy - y^2$ im Rechteck $[-2, 2] \times [-2, 2]$ graphisch dar. Zeichnen Sie die Höhenschichtlinien für $z = -3, -2, \ldots, 2, 3$.

17 Kontrollstrukturen

- Kontrollstrukture dienen zur Steuerung des Programmablaufs. Sie treten daher beim interaktiven Arbeiten fast nie auf, umso häufiger aber in Programmen und Funktionen.

17.1 `if ...else ...end`

- Diese Anweisungen dienen zur bedingten Ausführung von Anweisungen:

```
>> if a>0
>>    b=sqrt(a)
>> else
>>    error('a negativ')
>> end
```

- Der `else`-Zweig kann entfallen. Ist die Bedingung des `if`-Zweigs nicht erfüllt, geschieht in diesem Fall überhaupt nichts.

17.2 `for ...end`

- Diese Anweisungen dienen zur Konstruktion von Schleifen mit fester Anzahl von Durchläufen:

```
>> for i=1:100
>>    q(i)=i.^2
>> end
```

- Schleifen sollten vermieden werden, wenn sie durch Vektoroperationen ersetzt werden können. Also besser:

```
>> q=[1:100].^2
```

- Die Schleife kann mit der Anweisung `break` vorzeitig abgebrochen werden.

17.3 `while ...end`

- Diese Anweisungen dienen zur Konstruktion von Schleifen, wenn die Anzahl der Durchläufe vorher nicht bekannt ist:

```
>> % Berechnung aller Fibonacci-Zahlen zwischen 1 und 1000
>> f(1:2)=1;
>> n=3;
>> while f(n-2)+f(n-1)<1000
>>    f(n)=f(n-2)+f(n-1);
>>    n=n+1;
>> end
```

- Die Schleife kann mit der Anweisung `break` vorzeitig abgebrochen werden.

- Mit `while` können auch Endlos-Schleifen konstruiert werden:

```
>> while 1==1 % immer wahr!
>>    disp('hallo')
>> end
```

- Enthält die Endlos-Schleife keine `break`-Anweisung, kann sie nur durch die Eingabe von `Strg-C` abgebrochen werden.

17.4 `switch ...end`

- Diese Anweisungen dienen zur Konstruktion von Fallunterscheidungen:

```
>> switch errorcode
>>    case 1
>>       error('Fehler 1')
>>    case 2
>>       error('Fehler 2')
>>    case 3
>>       error('Fehler 3')
>>    otherwise
>>       error('Unbekannter Fehlercode')
>> end
```

- Der `otherwise`-Zweig kann entfallen. Wird kein passender Fall gefunden, geschieht dann überhaupt nichts.

17.5 `return`

- Diese Anweisung dient zum Rücksprung aus einer Funktion.

17.6 Aufgaben

Aufgabe: Berechnen Sie die Exponentialfunktion der Werte $x = 0, 1e{-}5, 2e{-}5, \ldots, 10$ mit einer Vektoroperation und mit einer Schleife. Vergleichen Sie die Laufzeit mit den Funktionen `tic` und `toc`.

18 Benutzerdefinierte Funktionen

18.1 Inline-Funktionen

- Tritt ein arithmetischer Ausdruck häufig auf, lohnt es sich, dafür eine Inline-Funktion zu erzeugen.
- Deklaration einer Inline-Funktion:
```
>> f=inline('x^2*sin(x)')
```
- Auswertung einer Inline-Funktion:
```
>> f(0)
>> f(1)
>> f(pi/2)
```
- Soll die Inline-Funktion auch Vektoren als Eingabe akzeptieren, müssen alle Operationen punktweise durchgeführt werden:
```
>> g=inline('x.^2.*sin(x)')
>> y=g(1:5)
```
- Eine Inline-Funktion kann auch mehr als ein Argument haben:
```
>> h=inline('sum(x.^2-y.^2)','x','y')
>> h([2 4 6],[1 2 3])
```

18.2 Echte Funktionen

- Lässt sich eine Funktion nicht durch einen einfachen arithmetischen Ausdruck beschreiben, muss sie in einer Datei abgespeichert werden.
- Die erste Zeile der Funktion legt die maximale Zahl der Eingabeparameter und die maximale Zahl der Rückgabeparameter fest:

  ```
  function [a,b]=fun(x,y,z)
  ```

- Die Funktion wird durch den Dateinamen identifiziert, der Funktionsname ist an sich gleichgültig.
- Die Auswertung einer Funktion erfolgt durch die Angabe des (Datei-)Namens und der Eingabeparameter, die in runden Klammern eingeschlossen werden:

  ```
  >> u=sign(pi)
  ```

- Enthält die Zeichenkette `fname` den (Datei-)Namen einer Funktion, kann die betreffende Funktion auch so ausgewertet werden:

  ```
  >> r=feval(fname,par1,par2,...,parn)
  >> u=feval('sign',pi)
  ```

- Eine Funktion kann keinen, einen oder mehrere Eingabeparameter haben:

  ```
  >> n=rand % Zufallszahl
  >> nv=norm(v) % Norm eines Vektors
  >> c=dot(a,b) % Inneres Produkt von zwei Vektoren
  ```

- Eine Funktion kann keinen, einen oder mehrere Rückgabeparameter haben:

  ```
  >> disp(x) % Anzeige eines Feldes
  >> ax=abs(x) % Berag einer Zahl
  >> [xmin,imin]=min(x) % Kleinster Wert und sein Index
  ```

- Viele Funktionen haben verschiedene Funktionalität, je nach der Zahl der Ein- oder der Rückgabeparameter, mit der sie aufgerufen

werden:

```
>> r=rand % Zufallszahl
>> r=rand(n,1) % Spaltenvektor mit Zufallszahlen
>> r=rand(n) % nxn-Matrix von Zufallszahlen
>> hist(x,xh) % Zeichne Histogramm
>> h=hist(x,xh) % Berechne Histogramm
```

- Innerhalb einer Funktion gibt die MATLAB-Funktion `nargin` die Anzahl der Eingabeparameter zurück, mit der die Funktion aufgerufen wurde.
- Innerhalb einer Funktion gibt die MATLAB-Funktion `nargout` die Anzahl der Rückgabeparameter zurück, mit der die Funktion aufgerufen wurde.
- Eine Funktion kann eine beliebige Zahl von Eingabeparametern haben. In diesem Fall muss der letzte Eingabeparameter die Variable `varargin` sein. Diese ist ein Cell Array. Beispiel: `xlabel`.
- Eine Funktion kann eine beliebige Zahl von Rückgabeparametern haben. In diesem Fall muss der letzte Rückgabeparameter die Variable `varargout` sein. Diese ist ein Cell Array. Beispiel: `solve`.
- Abbildung 1 zeigt eine Funktion, die ein bestimmtes Integral mit Hilfe der Simpson-Regel berechnet. Neben arithmetischen Ausdrücken und MATLAB-Funktionen verwendet die Funktion Kontrollanweisungen (`if`, `end`, `return`) und die Funktion `error`, die im Fall von fehlerhafter Eingabe die Berechnung abbricht und eine Fehlermeldung ausgibt.

18.3 Aufgaben

Aufgabe: Definieren Sie eine Inline-Funktion, die den Winkel zwischen zwei Vektoren (in Grad) berechnet.

Aufgabe: Schreiben Sie eine Funktion, die für einen Vektor die Länge, den Mittelwert, den Median und die Standardabweichung zurückgibt. Ist die Eingabe kein Vektor, soll eine Fehlermeldung ausgegeben werden. Testen Sie die Funktion.

```
function I=simpson(x,f)
% Integral mit Simpson-Regel
% Die Laenge der Vektoren x und f muss ungerade sein
nx=length(x)-1;
if mod(nx,2)~=0
error('Die Anzahl der Gitterpunkte muss ungerade sein!')
end
d=diff(x);
if (max(d)-min(d))/mean(d)>1e-12
error('Das Gitter muss regelmäßig sein!')
end
nf=length(f)-1;
if nx =nf
error('Die Listen x und f müssen gleich lang sein!')
end
c=ones(1,nx+1);
c(3:2:nx-1)=2;
c(2:2:nx)=4;
h=mean(d);
I=(h/3)*sum(c.*f);
return
```

Abbildung 1: Die Datei `simpson.m`

19 Programme (Scripts)

- Eine syntaktisch richtige Folge von MATLAB-Anweisungen, die in einer Datei abgespeichert ist, heißt ein MATLAB-Script oder MATLAB-Programm.
- Ein MATLAB-Programm kann durch die Eingabe des Dateinamens ausgeführt werden.

Aufgabe: Erstellen und testen Sie ein Programm, das mit Hilfe der Funktion `simpson` das folgende bestimmte Integral berechnet:

$$I = \int_a^b \frac{\sin(x)}{x}\, dx$$

Die Integrationsgrenzen und die Anzahl der Stützpunkte sollen interaktiv eingegeben werden.

Aufgabe: Die Goldbach'sche Vermutung besagt, dass jede gerade Zahl größer 2 als Summe von zwei Primzahlen darstellbar ist. Diese Darstellung ist in der Regel nicht eindeutig. Bestimmen Sie, auf wieviele verschieden Arten die geraden Zahlen zwischen 4 und 100 als Summe von zwei Primzahlen dargestellt werden können.

20 Interaktiver Debugger

- Die Funktion `dbstop` setzt einen Breakpoint, entweder in eine bestimmte Anweisung oder wenn eine bestimmte Bedingung erfüllt ist.
- Breakpoints können auch direkt im Editor gesetzt werden.
- Hat das Programm an einem Breakpoint angehalten, können alle Variablen angezeigt oder verändert werden.
- Die Funktion `dbclear` löscht einen (oder alle) Breakpoint(s).
- Die Funktion `dbstatus` gibt eine Liste aller Breakpoints aus.
- Die Funktion `dbstack` zeigt die komplette Aufrufsequenz an.
- Mit der Funktion `dbcont` kann die Ausführung des Programms fortgesetzt werden.
- Mit der Funktion `dbquit` wird der Debugger verlassen und die Ausführung des Prgramms abgebrochen.

21 Die MATLAB-Toolboxes

Für mehr Information siehe
`http://www.mathworks.com/products/product_list.jsp`.

- Bioinformatics Toolbox
- Communications Toolbox
- Control System Toolbox
- Curve Fitting Toolbox

- Data Acquisition Toolbox
- Database Toolbox
- Datafeed Toolbox
- Excel Link
- Extended Symbolic Math Toolbox
- Filter Design Toolbox
- Financial Derivatives Toolbox
- Financial Time Series Toolbox
- Financial Toolbox
- Fixed-Income Toolbox
- Fuzzy Logic Toolbox
- GARCH Toolbox
- Genetic Algorithm and Direct Search Toolbox
- Image Acquisition Toolbox
- Image Processing Toolbox
- Instrument Control Toolbox
- LMI Control Toolbox
- Mapping Toolbox
- MATLAB
- MATLAB COM Builder
- MATLAB Compiler
- MATLAB Excel Builder
- MATLAB Link for Code Composer

- MATLAB Report Generator
- MATLAB Runtime Server
- MATLAB Student Version
- MATLAB Web Server
- Model Predictive Control Toolbox
- Model-Based Calibration Toolbox
- μ-Analysis and Synthesis Toolbox
- Neural Network Toolbox
- Optimization Toolbox
- Partial Differential Equation Toolbox
- Robust Control Toolbox
- Signal Processing Toolbox
- Spline Toolbox
- Statistics Toolbox
- Symbolic Math Toolbox
- System Identification Toolbox
- Virtual Reality Toolbox
- Wavelet Toolbox