Technische Universität Hamburg-Harburg **SS 2008**
Institut für Numerische Simulation
Prof. Dr. W. Mackens

MATLAB-Ganz-Kurz-Einführung (2 Stunden)

Juni 2008

1 Ziele:

A. Elementares Umgehen mit MATLAB

   1. `lookfor, what, help, Tab-Taste, demo`
   2. Interaktionen
   3. Script files (M-Files und Funktionen)

B. Variablen, Vektoren, Matrizen, Gleichungssysteme, Eigenwertaufgaben

C. Programm-Ablaufkontrolle

D. Plotten und Graphik

E. „Funktionenfunktionen“

F. Anfangswertaufgabenlösung

2 Elementares Umgehen mit MATLAB

2.1 lookfor, what, help, Tab-Taste, demo

Daten werden in Speichern gehalten, die durch Namen ansprechbar sind.

Datenspeicher werden alloziert, indem Variableninhalte definiert werden. So definiert

```
a= 7.123
```

einen Speicher für eine Gleitpunktzahl (Real), die hinfort durch den Namen `a` ansprechbar ist.

Es gibt weitere Datenstrukturen: Komplexe Zahlen, Vektoren, Matrizen, Arrays, Strukturen,...

Zur Manipulation der Daten stellt MATLAB furchtbar viele Befehle bereit.

Um sich im Befehlsdschungel zurechtfinden zu können, gibt es eine Reihe von „Orientierungsbefehlen“:

- `lookfor` „Wort“ sucht nach allen Befehlen, in deren Beschreibung das Wort „Wort“ vorkommt.
- `what` ist ähnlich, sucht aber weniger
- `help` „Befehlsname“ erklärt die Benutzung des Befehls mit dem Namen „Befehlsname“
- Drückt man nach der Eingabe einer Zeichenkette die Tab-Taste, so versucht MATLAB, die Zeichenkette zu einem Befehl zu ergänzen.
- Der Befehl `demo` eröffnet den Zugang zu (wirklich) stundenlanger Beschäftigung mit Demo-Programmen für alle möglichen einfachen und schwierigen Aufgaben. Wenn man sich in ein neues Anwendungsgebiet von MATLAB einarbeiten möchte, macht es Sinn, die zugehörigen Demo-Programme genau zu studieren.

2.2 Interaktionen

- Befehle können benutzt werden, um Daten zu definieren
  `a = [1,2,3] % a ist ein Zeilenvektor.`
  oder um sie zu manipulieren:
  ```
  a=a'; % Die Transposition ist das Hochkomma; a ist nun ein Spaltenvektor
  b = 2*a; % Nimmt den Vektor mit 2 mal.
  n = norm(a); % Berechnet die 2-Norm
  w= a'*a; % ist das innere Produkt von a mit sich selbst.
  M = a*b'; % M ist ein (ausmultipliziertes) dyadisches Produkt
  s = sin(a);% s ist der Vektor der Sinus-Werte der Komponenten von a
  ```
- Ein Semikolon hinter dem Befehl unterdrückt die Ausgabe des Ergebnisses.
- mit den `Pfeil rauf und runter` Tasten kann man in den letzten Befehlen browsen.
- Angabe eines Befehlsanfanges und Betätigen der Pfeiltasten sucht den letzten Befehl mit dem angegebenen Anfang.
- MATLAB unterscheidet zwischen Klein- und Großbuchstaben
- MATLAB benutzt alle Klammersorten `( ), [ ],` { } und unterscheidet zwischen ihnen.
- MATLAB wird beendet mit den Befehlen `quit` oder `exit`.
- Die Ergebnisse einer MATLAB-Sitzung können mit
  `save` "Filename"
  in das File `Filename.mat` gespeichert werden und mit
  `load` „Filename"
  wieder geladen werden.

2.3 Script files, Funktionen

Längere Operationsfolgen wird man abspeichern wollen, insbesondere wenn diese spezielle Algorithmen für klare Aufgabenstellungen beinhalten.

Dafür stehen zwei verschiedene File-Sorten zur Verfügung

Script M-files haben keinen In- oder Out-Put. Die Befehle operieren auf den Variablen des Arbeitsplatzes von dem sie aufgerufen wurden.

Function M-Files haben eine Funktionsdefinitionszeile, können Ein- und Ausgabe-Argumente haben. Sie arbeiten auf ihren internen Variablen, die nach außen nicht sichtbar sind (es sei denn, sie seien als `global` deklariert).

3 Variablen, Vektoren, Matrizen, Gleichungssysteme, Eigenwertaufgaben

3.1 Vordefinierte Variablen

Es gibt in MATLAB vordefinierte Variablen (die man aus praktischen Erwägungen nicht überschreiben sollte) .

`ans` letzte Antwort

`eps` Maschinengenauigkeit

`i,j` $\sqrt{-1}$, imaginäre Einheit

`inf` ∞, Unendlich

`NaN` Not a Number

`pi` $\pi = 3.14159265...$

3.2 Reelle und komplexe Variablen

Variablennamen beginnen mit einem Buchstaben und haben maximal `namelengthmax` Zeichen aus Buchstaben, Zahlen, und „Underscores“. Dabei werden große und kleine Buchstaben unterschieden.
`Dies_ist_also_eine_1ste_gueltige_Variable`
ist also ein gültiger Name für eine Variable, und das folgende
`Dies_ist_Also_eine_1ste_gueltige_Variable`
ist der Name für eine zweite davon verschiedene. Warum?

Was durch einen Namen bezeichnet wird, wird durch die Zuweisung entschieden.
Der Befehl
`Komplexe_Zahl = 5`
wird auf dem mit `Komplexe_Zahl` ansprechbaren 8-Byte-Speicher die Zahl 5 als „doppelt genaue“ Gleitpunkzahl (entsprechend einer Dezimalmantissenlänge von etwa 16) speichern.
Durch
`Ganze_Zahl = pi+ i*eps`
werden dagegen über den Namen `Ganze_Zahl` zwei 8-Byte-Speicher für den Realteil π und den Imaginärteil $2.220446049250313e-016$ (auf meinem Laptop) einer komplexen Zahl (mit den Rechenregeln für komplexe Zahlen) ansprechbar sein[1]. Bei der Ausgabe, die durch Aufruf von
`Ganze_Zahl`
erfolgt, sieht man auf dem Bildschirm

```
Ganze_Zahl =
            3.141592653589793 + 0.0000000000000000i
```

da Real- und Imaginärteil gemeinsam skaliert werden.
Isoliert man den Imaginärteil durch
`imag(Ganze_Zahl),`
so erhält man

```
ans =
     2.220446049250313e-016.
```

Ausgabeart und -länge lassen sich übrigens mit dem Befehl
`format`
steuern. Wie, wird einem nach Eingabe von
`help format`
erzählt.

3.3 Vektoren und Matrizen

MATLAB kommt von „MATrix LABoratory“, und daher ist der Umgang mit Matrizen und Vektoren hierin ganz einfach, weil MATLAB diese Datenstrukturen und ihre mathematischen Verknüpfungen besonders unterstützt.
Einen vierdimensionalen Zeilenvektor mit den Elementen 4,3,2,1 gibt man z.B. so ein:

```
v =[ 4 3 2 1];
```

oder so

```
v = [4,3,2,1];
```

oder auch so[2]

```
v = 4:-1:1;
```

Allgemein ist p:q:r der Vektor aller Zahlen p, p+q, 2+2q ..., die kleiner oder gleich `r` sind.
Lässt man bei solchen Aufzählungen das Inkrement q fort, wählt MATLAB dafür den Wert 1.
Durch

```
k = 4:10
```

ist z.B. also der Vektor

```
k =
    4  5  6  7  8  9  10
```

[1] Für eine komplexe Zahl den Namen „ganze Zahl“ zu wählen und umgekehrt wird ein Programm nicht unbedingt besser verstehbar machen. Hier haben wir damit nur demonstriert, dass die Namensgebung in MATLAB nicht durch die Inhalte der Speicher bestimmt wird.

[2] Für alle in diesem kurzen Anriss gegebenen Befehle gilt: „Es gibt viele weitere Befehlsmöglichkeiten, dasselbe Resultat zu erzielen. So ist auch die Liste der Definitionsmöglichkeiten eines Vektors mit den hier angegebenen drei Befehlen bei weitem noch nicht erschöpft. Welche Generierung jeweils günstiger ist, hängt von den Umständen ab - etwa der Art, in der eingehende Daten gespeichert sind.

beschrieben.
Hat der 6-dimensionale Vektor nur in der 6. Komponente ein zu spezifizierendes Element und ist sonst Null, so kann man schreiben:
`a(6) =4`
um zu erhalten
```
a=
   0  0  0   0  0  4
```
Vektoren sind nichts anderes als Matrizen, deren eine Dimension gleich eins ist. Eine 4x3-Matrix kann man eingeben wie einen Vektor, nur dass ein Zeilenvorschub durch eine Semikolon hervorgerufen wird:
`A = [ 1 2 3 ; 2 3 4 ; 3 4 5 ; 4 5 6 ]`
ergibt

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{pmatrix}.$$

Elemente von `arrays` (das sind indizierte Größen, wie Vektoren, Matrizen oder Speicher mit noch mehr Indizes) spricht man durch Angabe der gewünschten Indizes in runden Klammern an. So greift der Aufruf
`a(3)`
auf das dritte Element von a zu und zewigt es an, wogegen die Zuweisung
`A(2,3)= 7`
das Element A_{23} zu 7 ändert:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 7 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{pmatrix}.$$

3.4 Spezial-Matrizen

Für häufig gebrauchte Matrizen gibt es Spezialbefehle. Wir listen hier zunächst die häufiger gebrauchten auf

`zeros` `zeros(4,5)` erzeugt eine 4X5- Null-Matrix, `zeros(6)` erzeugt eine 6x6-Nullmatrix.

`ones` erzeugt analog Matrizen aus lauter Einsen.

`eye` erzeugt Einheitsmatrizen.

`rand` erzeugt gleichverteilte Zufalls-Matrizen

`randn` erzeugt normalverteilte Zufalls-Matrizen

`gallery` gibt Zugriff auf einen Satz von Test-Matrizen

`linspace` erzeugt einen Vektor aus äquidistant verteilten Punkten

3.5 Matrix-Vektor-Manipulationen

Alle üblichen Matrix-Vektormanipulationen sind in MATLAB leicht möglich. Multipliziert wird mit dem `*` -Operator, wobei die Partner die üblichen Dimensionsanforderungen erfüllen müssen.

Beim Produkt
`C= A*B`
einer (m,n)-Matrix A und einer (p,q)-Matrix B, müssen die „inneren Dimensionen“ n und p in

$$(m, n) \times (p, q)$$

übereinstimmen, wenn eine (m,q)-Matrix `C` dadurch erklärt sein soll.

Für die Multiplikation $A \cdot B$ und die Exponentiation A^3 sollen hier gleich noch zwei Varianten angegeben werden, die elementweise wirken.

Während für `x= [ 1 2 3 4]` und y = [0 1 0 3] die Operation
`z = x*y`
überhaupt nicht definiert ist, ist
`z = x.*y`
sehr wohl erklärt und bedeutet die komponentenweise Multiplikation
`z(i) = x(i)*y(i), i=1,2,3,4;` mit dem Ergebnis `z = [0 2 0 12]`.

Analog ist
`x^2 ,`
die zweite Potenz von x, nur für quadratische Matrizen x erklärt, wogegen `x.^2`
einfach wieder elementweise ausgeführt wird.
Und während die Division zweier Vektoren nicht erklärt ist, ist die elementweise Division,
`z= y./x`
durchaus sinnvoll, solange x keine Null-Elemente hat.

Da die Addition und Subtraktion sowieso elementweise ausgeführt werden, sind Punktversionen hier natürlich nicht nötig.

Wenn man nun noch weiß, dass man die Transposition einfach durch das Apostroph erhält und die Inverse von `A` durch `inv(A)`, kann man eigentlich alle üblichen Matrix-Vektorausdrücke bilden. Dabei verwendet mn `inv` aber bitte nur zu ganz besonderen Anlässen[3], weil man `x =inv(A)* b` viel schneller als Lösung des linearen Gleichungssystems `Ax=b` bestimmen kann.

Natürlich wird man nun wissen wollen, wie man denn lineare Gleichungssysteme löst, und wir schreiben sogleich den Standardbefehl hierzu auf.

Es sollte an dieser Stelle aber angemerkt werden, dass Sie in Ihrem SBachelor-tudium wegen einer relativ knappen Mathematikausbildung leider nur wenige der vielen heute bekannten - und daher auch in MATLAB realisierten Methoden - zur Lösung linearer Systeme kennen lernen konnten. Im Masterprogramm können Sie immerhin weitere Vorlesungen wählen, die Ihnen einige dieser Methoden näher bringen[4].

Ein Gleichungssysteme $Ax = b$ mit einer regulären (n,n)-Matrix A sowie einem n-dimensionalen Spaltenvektor b wird einfach gelöst durch `x = A\ b`.
Ausgeführt wird dabei im Normalfall der Gauss-Algorithmus. Wenn das Gleichungssysten Besonderheiten aufweist, wird aber auch schon einmal vom Normalweg abgewichen.

Z.B. löst dieser Befehl auch das Ausgleichsproblem

$$\|Ax - b\|_2 = \min,$$

wenn das System $Ax = b$ überbestimmt ist[5].

Die weiteren Ihnen bekannten Operationen der linearen Algebra listen wir hier einfach auf und überlassen es Ihnen, ihre genaue Bedeutung und Anwendung über `help` herauszufinden.

`inv, lu, qr, norm, det`

3.6 Untermatrizen und Blockmatrizen

Matrizen können **einerseits** aus anderen Matrizen als Block-Matrizen definiert werden. So ensteht aus den Matrizen

$$A = \begin{pmatrix} 1 & 2 & -3 \\ 2 & 3 & 4 \end{pmatrix}, B = \begin{pmatrix} 10 & 11 \\ 12 & 13 \\ 14 & 15 \end{pmatrix} \text{ und } C = \begin{pmatrix} 100 \\ 200 \\ 300 \end{pmatrix}$$

durch

[3] Da Sie die vermutlich nicht kennen, bleibt besser die Devise bestehen: „Wer Marizen invertiert, ist doof!“

[4] Obwohl dies eigentlich zu spät ist, denn dort sollen Sie ja in Ihren Ingenieurvorlesungen moderne Rechentechnik anwenden können und nicht diese erst lernen, nachdem Sie in Ihren Fachvorlesungen die Mathematik des letzten oder vorletzten Jahrhunderts verwenden mussten.
Die Politik hat mit dem Bologna-Prozess der Ausbildung mal wieder einen argen Bärendienst erwiesen.

[5] Und zwar nach den Regeln der Kunst mit der QR-Zerlegung und nicht etwa mit den rundungsfehlersensitiven Normalgleichungen.

`M = [[A; [0 0 0]],B,C]`
die Matrix

$$M = \begin{pmatrix} 1 & 2 & -3 & 10 & 11 & 100 \\ 2 & 3 & 4 & 12 & 13 & 200 \\ 0 & 0 & 0 & 14 & 15 & 300 \end{pmatrix}.$$

Umgekehrt können aus Matrizen auch Teilmatrizen entnommen werden: So ergibt
`W = M([1,3],[2,4,6])`
die Teiluntermatrix der ersten und dritten Zeile und der zweiten, vierten und sechsten Spalten von M, also

$$W = \begin{pmatrix} 2 & 10 & 100 \\ 0 & 14 & 300 \end{pmatrix}.$$

Will man **alle** Zeilen oder **alle** Spalten von der p-ten bis zur q-ten, so kann man statt einer Aufzählung aller entsprechenden Indizes auch einfach `p:q` schreiben. Wille man alle von Anfang bis Ende, so schreibt man nur `:` .

Der Index `end` ist der jeweils letzte Index der Dimension, in der er benutzt wird. Möchte ich z.B.die quadratische (3,3)-Submatrix in der rechten unteren Ecke einer Matrix P mit unbekannter Zeilen- und Spaltenzahl (beide aber größer oder gleich 3), so kann ich diese ansprechen über `P(end-2:end,end-2:end)`.

3.7 Eigenwertaufgaben

Es werden hier nur die Hauptbefehle einfach angegeben.

`eig:` `d = eig(A)` liefert im Vektor `d` die Eigenwerte von A zurück. `[V,D] = eig(A)` legt die Eigenwerte von `A` in der Diagonale der Diagonalmatrix `D` und die zugehörigen Eigenvektoren von `A` in den entsprechenden Spalten der Matrix `V` ab (,wenn `A` diagonalisierbar ist).

`eig:` `e= eig(A,B)` bzw. `[V,D] = eig(A,B)` führen analoge Rechnungen für das verallgemeinerte Eigenwertproblem `Ax =` λ `Bx` aus.

`eigs:` Findet wenige Eigenwerte (mit zugehörigen Eigenvektoren) in vorgegebener Region.

4 Programm-Ablaufkontrolle

MATLAB verfügt wie jede Programmiersprache über das übliche Repertoire von Ablauf-Kontrollbefehlen.

Für die logische Verzweigung gibt es die üblichen `if, else,elseif` - Konstrukte.

```
if expression1
     statements1
elseif expression2
    statements2
else
    statements3
end
```

wobei die „expressions“ als Resultate jeweils die logischen Werte 1 (wahr) oder 0 (falsch) haben und die nachfolgenden „statements“ bei „wahr“ausgeführt und bei „falsch“ übersprungen werden. In den „expressions“ kommen wiederum verschiedenste Vergleichs- und logische Verknüpfungsoperatoren zum Einsatz.

Für Schleifen stehen die üblichen `for`-Schleifen zur Verfügung.

```
for k=1:3:8
statements(k)
end
```

bedeutet z.B. dass die von der Variable k abhängigen statements für in Schritten von 3 wachsendes k ausgeführt wird, solange k die Zielzahl 8 nicht überschreitet. Als Schleifenkontrollvektoren (in der letzten Schleife der Vektor `1:3:8`) können auch allgemeine Vektoren verwendet werden. Mit

```
kv = [ 1   4  7  1   -1]
```
würde nach

```
for k=kv
statements(k)
end
```

die Statements

```
staments(1), statements(4), statements(7), statements(1), statements(-1)
```

ausgeführt werden.

Einen Überblick über die Programmflusskontrolle gibt der erste Absatz der nach

```
help lang
```

ausgedruckten Hilfe.

5 Plotten und Grafik

Die Möglichkeiten des Plottens von Funktionen gehen fast in's Unermessliche.

Einen Funktionsgraphen der Sinus-Funktuion auf dem Intervall von Null bis 2π findet man etwa so

```
x = linspace(0,2*pi,51) % Teile [0,2π] äqidistant in 50 Teile.
y = sin(x) % Berechne die Funktionswerte für den ganzen Vektor x.
plot(x,y) % Plottet Funktion durch Verbindung der Punkte (x(i),y(i)), i=1,..,n.
```

Weitere mit Plotaufgaben verbundenen Befehle sind
`plottools, semilogx, semilogy, loglog, plotyy, plot3, grid, title, xlabel, ylabel, axis, axes, hold, legend, subplot, scatter.`

Einen weitgehenden Übreblick erhält man durch
`help graph2d`
`help graph3d`
und
`help specgraph`

6 „Funktionenfunktionen“

Häufig werden Funktionen selbst wieder Werte (in Vektorräumen; häufig in den reellen oder komplexen Zahlen) zugeordnet. Einfache Beispiel dafür sind

$$\begin{array}{rcl} f & \mapsto & f(1) \\ f & \mapsto & f'(2)+3f(3) \\ f & \mapsto & \int_0^1 f(t)dt \end{array}$$

Will man solche Funktionen von Funktionen programmtechnisch realisieren, wird man die jeweils im Anwendgsfall konkret vorliegende Funktion seinem Programm zu Kenntnis bringen wollen. In diesem Fall muss der Funktionsname der abzubildenden Funktion als „Functionhandle“ vorliegen.

Beispiel: Es soll die durch ein Programm wie

```
function erg = quadrat(x)
erg = x*x;
```

gegebene Funktion an einer Stelle 2.3 numerisch differenziert werden. Das Differentiationsprogramm dazu soll so geschrieben werden, das der Name der Funktion noch variabel bleibt und auch die Stelle , an der differenziert wird (hier also 2.3), erst bei Anwendung dem Programm bekannt gegeben werden muss.

In MATLAB kann der Aufruf des Differentiationsprogramm `MyDiffer` (habe ich nun mal so genannt, weil ich's schön finde!) zur Differentiation der Funktion `quadrat` an der Stelle 2.3 dann z.B. so aussehen

```
wert =MyDiffer(@quadrat,2.3)
```

`@quadrat` ist hierbei ein „Funktionshandle". Durch das @ wird MATLAB angezeigt, dass dies keine gewöhnliche Variable ist sondern der Name eines zu findenden und aufzurufenden Unterprogrammes.

Die Funktionsfunktion MyDiffer wird natürlich geschrieben werden müssen, ohne dass der Name der später als Parameter zu spezifizierenden Funktion `quadrat` bekannt ist. Im Gegenteil soll hier ja jeder seine eigenen Funktionsnamen der von ihm jeweils neu geschriebenen Funktionen eingeben können. Das gilt auch für die Stelle der Ableitungsauswertung.

Die Lösung aus dem Dilemma sieht so aus:

```
function resultat = MyDiffer(fun, p)
resultat = (feval(fun,p+1e-8)-feval(fun,p))/1e-8;
```

Der Befehl `feval(fun,x)` wertet eine Funktion, deren Name auf der Variable `fun` übergeben wird an der Stelle `x` aus.]

7 Anfangswertaufgabenlösung

Mit MATLAB löst man im Bereich der gewöhnlichen Differentialgleichungen im Standardfall Anfangswertaufgaben für explizite Systeme von Differentialgleichungen erster Ordnung.

$$\begin{aligned} y_1'(t) &= f_1(t, y_1, \ldots, y_n), \\ &\vdots \qquad\quad \vdots \\ y_n'(t) &= f_n(t.y_1, \ldots, y_n) \end{aligned}$$

Beispiel: Gegeben sei das Volterra-Lotka-System

$$\begin{aligned} y_1'(t) &= 0.1 \cdot y_1(t) - 0.2 \cdot y_1(t) \cdot y_2(t), \\ y_2'(t) &= -0.2 \cdot y_2(t) + 0.1 \cdot y_1(t) \cdot y_2(t). \end{aligned}$$

mit Anfangsbedingungen

$$y_1(0) = 1, \quad y_2(0) = 0.1$$

Dann wird dieses System einfach wie folgt in MATLAB beschrieben:

```
function erg = VolLot(t,y) %Eingabe t und y; Ausgabe erg
erg(1)= 0.1*y(1)-0.2*y(1)*y(2); % Rechte Seite der 1. Dgl
erg(2)= -0.2*y(2)+0.1*y(1)*y(2);% Rechte SEite der 2. Dgl
erg=erg';% komponentenweise Erzeugung produziert Zeilenvektoren.
         % ode45 unten erwartet Spaltenvektor.
```

FERTIG! das war schon alles.

Ein Integrationsaufruf hierzu wäre

```
[t,y] = ode45(@VolLot, [0,10], [1,0.1]);
```

```
plot(t,y);
```

und sorgt für die Integration der Differentialgleichung von $t = 0$ bis $t = 10$ (Erste Klammer), mit den in der zweiten Klammer angegebenen Anfangswerten.

Anstatt viel mehr Raffiniertes anzustellen, soll kurz erklärt werden, wie man eine Differentialgleichung höherer Ordnung in ein System erster Ordnung umschreibt.

Wir nehmen dazu mal die Aufgabe

$$y^{(4)}(t) + \sin(y^{(3)}(t) \cdot t) + y^{(2)}(t) \cdot y(t) = \exp(-t)$$

Sie ist VIERTER Ordnung. Deshalb wählen wir die folgenden VIER aufsteigenden Ableitungen von y beginnend bei der NULLTEN als Funktionen des zu bestimmenden Systems erster Ordnung[6]:

$$\begin{aligned} y_1(t) &:= y(t), \\ y_2(t) &:= y'(t), \\ y_3(t) &:= y^{(2)}(t), \\ y_4(t) &:= y^{(3)}(t). \end{aligned}$$

und finden nun unser gewünschtes System

$$\begin{aligned} y_1'(t) &= y'(t) &&= y_2(t), \\ y_2'(t) &= (y'(t))' &&= y_3(t), \\ y_3'(t) &= (y^{(2)}(t))' &&= y_4(t), \\ y_4'(t) &= (y^{(3)}(t))' &&= y^{(4)}(t) = -(\sin(y^{(3)}(t) \cdot t) + y^{(2)}(t) \cdot y(t)) + \exp(-t) \end{aligned}$$

Indem man jetzt noch in der letzten Gleichung die Ableitungen von y durch die entsprechenden neuen y_i-Funktionen ersetzt, gelangt man zu

$$\begin{aligned} y_1'(t) &= y_2(t), \\ y_2'(t) &= y_3(t), \\ y_3'(t) &= y_4(t), \\ y_4'(t) &= -(\sin(y_4(t) \cdot t) + y_3(t) \cdot y_1(t)) + \exp(-t) \end{aligned}$$

Wie dies sein muss, hängt die rechte Seite nur von t und den Komponentenfunktionen $y_1, \ldots, y_4$ ab.

Weitere wichtige Befehle:

```
other ODE solvers: ode23, ode113, ode15s, ode23s, ode23t, ode23tb
implicit ODEs: ode15i
options handling: odeset, odeget
output functions: odeplot, odephas2, odephas3, odeprint
```

.....

Wir könnten beliebig lange weitermachen, haben aber nur eine Doppelstunde zur Verfügung.

[6]Der Großdruck soll andeuten, dass man bei einer DGL der Ordnung n auch eine Vektorfunktion $(y_1(t), \ldots, y_n(t))^T$ mit n Komponentenfunktionen wählt.