1 Matlabeinführung

MatLab (MATrix LABoratory) ist eine kommerzielle Software für wissenschaftliche Berechnungen mit Schwerpunkten auf numerischen Rechnungen und Implementierungen. Matlab ist grundsätzlich pflegeleicht und benötigt keine zusätzlichen Deklarationen von Variablen. Grafische Darstellungen sind unkompliziert und werden etwa in 2D mittels Polygonzügen umgesetzt. MatLab lässt sich sehr gut für die Softwareentwicklung einsetzen, es kann von der Rechengeschwindigkeit aber nicht mit klassischen Methoden wie C oder Fortran mithalten. Matlab steht über remote-Verbindung auf den Uniservern zur Verfügung.

1.1 Grundsätzliches

Variablendefinitionen werden mittels Gleichheitszeichen umgesetzt, etwa

```
>> x = pi/2
x =
   1.5708e+00
>> sin(x)
ans =
     1
```

Sollen die Ausgaben unterdrückt werden, so wird ein Semikolon ans Ende der Eingabe gesetzt

```
>> x = pi/2;
>> sin(x)
ans =
     1
```

Arbeiten lässt sich mittels des Command-windows und direkten Ein- sowie Ausgaben oder mittels eines *.m-files. In einem *.m-file können viele Befehle aufgelistet werden, welche nacheinander abgearbeitet werden. Ausgeführt wird das *.m-file aus dem Command-window, hier werden auch mögliche Ausgaben bzw. Fehlermeldungen angezeigt. Weiter kann ein *.m-file aber auch als Routine (function) definiert werden, siehe später Abschnitt 1.3.

1.2 Vektoren und Matrizen

Definition von 1- und 2D-Feldern mittels eckiger Klammern:

```
>> A = [1 2 3;4 5 6]
A =
     1 2 3
     4 5 6
>> b = [8;9]
b =
     8
     9
>> c = [pi exp(1)]
c =
    3.1416 2.7183
```

Multiplikation von Vektoren und Matrizen mittels *. Stimmen die Dimensionen nicht, so wird gemeckert

```
>> c*b
ans =
   49.5973
>> b*c
ans =
   25.1327 21.7463
   28.2743 24.4645
>> A*b
Error using *
Incorrect dimensions for matrix multiplication. Check that the
number of columns in the first matrix matches the number of
rows in the second matrix. To perform elementwise multiplication,
use '.*'.
```

Transponieren mittels ':

```
>> A'
ans =
     1 4
     2 5
     3 6
>> A'*b
ans =
    44
    61
    78
```

Sollen Rechenoperationen (etwa Quadrieren, siehe Beispiel unten) auf alle einzelnen Einträge eines Vektors oder einer Matrix ausgeführt werden, so muss ein Punkt vor der Rechenoperaiton eingefügt werden

```
>> A.^2
ans =
     1 4 9
    16 25 36
>> b.^2
ans =
    64
    81
```

1.3 Funktionen

Funktionen lassen sich im Command window oder mittels eines *.m-files definieren. Die Definition im Command window erfolgt mittels des @-Symbols

```
>> f = @(x) sin(x)
f =
  function_handle with value:
    @(x)sin(x)
>> f(0)
ans =
     0
>> f(pi)
ans =
   1.2246e-16
>> f(pi/2)
ans =
     1
```

Eine Funktion in Abhängigkeit zweier Variabler:

```
>> u = @(x,y) sin(x).*cos(y);
>> [X,Y] = meshgrid(linspace(0,2*pi,101),linspace(0,2*pi,201));
>> U = u(X,Y);
>> mesh(X,Y,U)
```

Wird eine komplexe Prozedur nötig, so bieten sich *.m-files an. Das *.m-file trägt den Namen der Funktion und die erste Zeile lautet „function RUECKGABE(N)=FUNKTIONSNAME(EINGABEWERT(E))“. Ein Beispiel:

```
function fac = fakultaet(n)
% Berechnet n!
% Genutzt wird eine rekursive Funktion
if n==1
    fac = 1;
else
    fac = n*fakultaet(n-1);
end
```

Kommentare werden mittels %-Teichen eingefügt. Aufgerufen wird die Prozedur im Command window

```
>> fakultaet(3)
ans =
     6
>> fakultaet(6)
ans =
   720
```

Eine andere Implementierung:

```
function fac = fakultaet2(n)
% Berechnet n!
% Genutzt wird eine for-Schleife
fac = 1;
for i=2:n
    fac = fac*i;
end
```

If- und for-Schleifen sind damit auch erklärt. Bleiben noch while-Schliefen:

```
function fac = fakultaet3(n)
% Berechnet n!
% Genutzt wird eine while-Schleife
fac = n;
i = n-1;
while i>1
    fac = fac*i;
    i = i-1;
end
```

1.4 Grafische Darstellungen

Grafische Darstellungen werden mittels Vektoren als Polygonzüge umgesetzt

```
>> x = linspace(0,2*pi,100);
>> plot(x,f(x))
```

Dabei wurde zunächst ein Vektor x definiert, mit 100 äquidistanten Einträgen zwischen 0 und 2π. Geplottet wird der Polygonzug der x-Werte und der Werte $f(x)$. Nützliche Optionen sind

```
>> plot(x,f(x),'-.')
>> plot(x,f(x),'x')
>> plot(x,f(x),'-x')
>> plot(x,f(x),'linewidth',1)
```

Zwei oder mehr grafische Darstellung werden durch Kommata getrennt

```
>> plot(x,f(x),x,cos(x),x,1./(x+1),'linewidth',1)
```

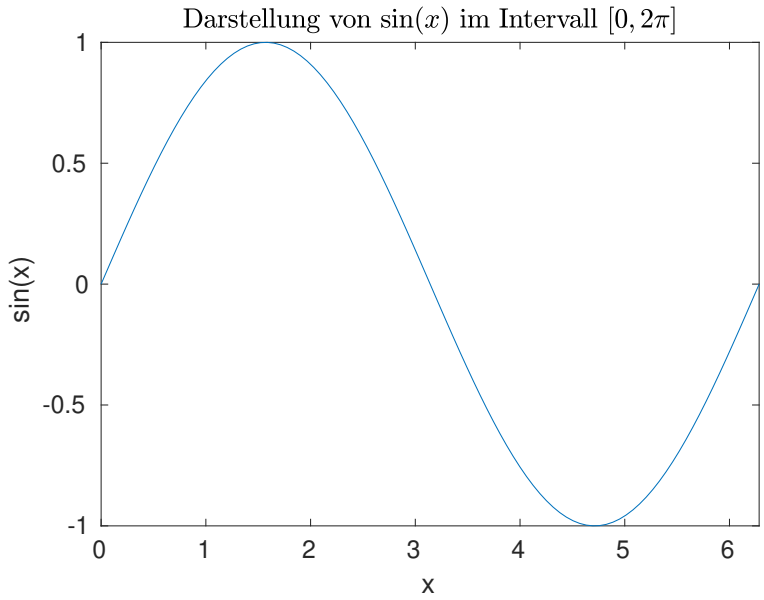
Für die grafische Darstellung wichtig sind auch

```
>> xlabel('x')
>> ylabel('sin(x)')
>> title('Darstellung von $\sin(x)$ im Intervall $[0,2\pi]$','Interpreter','latex')
>> set(gca,'xlim',[0 2*pi],'ylim',[-1 1],'fontsize',12)
```

Für den Titel wurde die Option „Interpreter“ „latex“ genutzt, um π zu setzen, siehe später die LaTeX-Einführung. Die Bezeichnung gca steht für „get current axis“.

Export von Grafiken als *.eps-files mittels

```
>> set(gcf,'paperpositionmode','auto')
>> print('-depsc','-loose','DATEINAME');
```


Darstellung von sin(x) im Intervall [0, 2π]
1
0.5
0
-0.5
-1
sin(x)
0
1
2
3
4
5
6
x

1.5 Lösung von Gleichungssystemen

Zur Lösung von Gleichungssystemen lässt sich der Backslash nutzen

```
>> A = [1 2 3;4 5 6;7 8 1];
>> b = [1 3 9]';
>> x = A\b
x =
   -0.1667
    1.3333
   -0.5000
```

Lineare Ausgleichprobleme lassen sich ebenfalls mittels Backslash lösen:

```
>> A = [1 2;3 4;5 6]
A =
     1 2
     3 4
     5 6
>> b = [1 3 9]';
>> x = A\b
x =
    3.6667
   -1.6667
>> A*x-b
ans =
   -0.6667
    1.3333
   -0.6667
>> norm(ans,2)
ans =
    1.6330
```

1.6 Lösung von Anfangswertprobleme

Gelöst werden soll ein Anfangswertproblem

$$\begin{aligned} y'(x) &= f(x, y(x)), \qquad a \leq x \leq b, \\ y(a) &= y_0 \end{aligned}$$

mit $f : [a, b] \times \mathbb{R}^n \to \mathbb{R}^n$. Gesucht ist die Lösung $y : [a, b] \to \mathbb{R}^n$, vgl. Existenz- und Eindeutigkeitssatz von Picard-Lindelöf. Ist eine analytische Lösung nicht zugänglich oder zu schwierig zu bestimmen, so lassen sich numerische Routinen zur Approximation der Lösung nutzen. Dazu werden das Intervall $[a, b]$ diskretisiert

$$a = x_0 < x_1 < x_2 < \ldots < x_N = b$$

und Approximationen

$$Y_i \approx y(x_i)$$

bestimmt. Die Abstände in x-Richtung

$$h_i = x_{i+1} - x_i$$

werden Schrittweiten genannt. Bei der Anwendung von expliziten Einschrittverfahren werden die Approximationen iterativ in der Form

$$\begin{aligned} Y_0 &= y_0, \\ Y_{i+1} &= Y_i + h_i \Phi(x_i, Y_i, h_i), \qquad i = 0, \ldots, N-1, \end{aligned}$$

bestimmt. Bei impliziten und Mehrschrittverfahren hängt die Verfahrensfunktion Φ von weiteren Variablen ab.

In Matlab wird zunächst die rechte Seite der Differentialgleichung als Funktion definiert, typischerweise als „rhs(x,y,...)“ für right hand side der Differentialgleichung. Anschließend lässt sich die Lösung des Anfangswertproblems etwa mittels der Routinen ode45 (nichtsteife Systeme) oder ode15s (steife Systeme) approximieren.

Wir betrachten das Anfangswertproblem (Räuber-Beute-Modellierung)

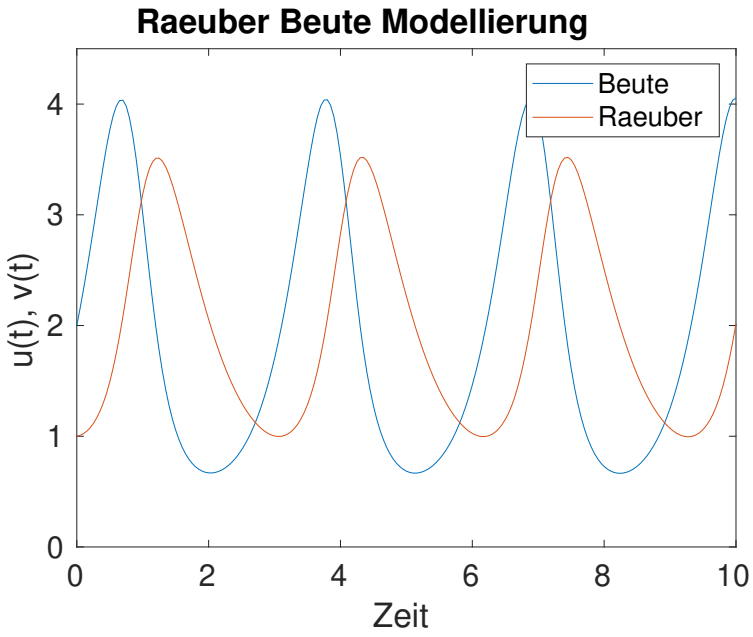
$$\begin{pmatrix} \dot{y}(t) \\ \dot{z}(t) \end{pmatrix} = \begin{pmatrix} ay(t) - by(t)z(t) \\ cy(t)z(t) - dz(t) \end{pmatrix}, \quad t \in [0, 10],$$
$$\begin{pmatrix} y(0) \\ z(0) \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

mit $a = 3$, $b = 1.5$, $c = 0.8$ und $d = 1.5$.

MatLab-Implementierung:

```
function raeuberbeute

h = 0.05;
tfin = 10;
T = 0:h:tfin;

a = 3;
b = 1.5;
c = 0.8;
d = 1.5;

q0 = [2 1];

[T,Q] = ode45(@(t,q) rhs(t,q,a,b,c,d), T, q0);

plot(T,Q)
xlabel('Zeit')
ylabel('u(t), v(t)')
title('Raeuber Beute Modellierung')
legend('Beute','Raeuber','location','northeast')
set(gcf,'paperpositionmode','auto')
set(gca,'xlim',[0 tfin],'ylim',[0 4.5],'fontsize',15)
```

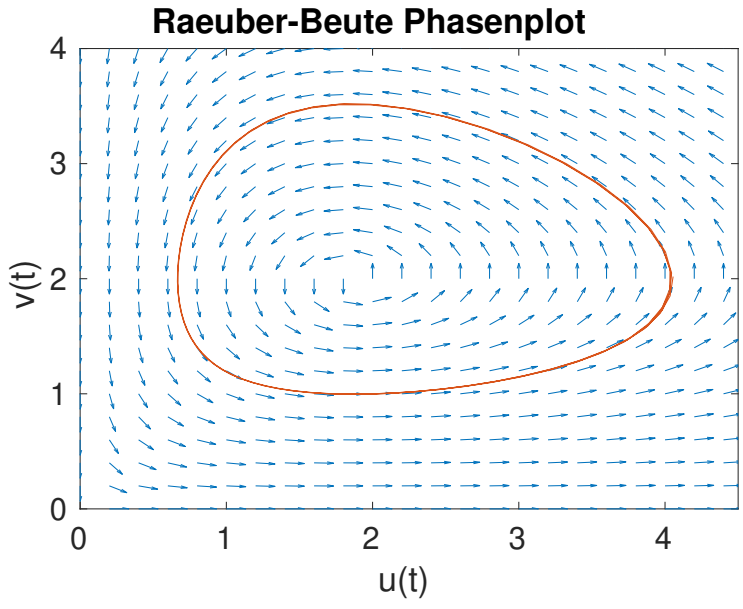
```
print('-depsc','-loose','rb1');

function dq = rhs(t,q,a,b,c,d)
dq = q;
dq(1) = q(1)*(a-b*q(2));
dq(2) = q(2)*(c*q(1)-d);
```



Im Folgenden *.m-file wird der Phasenplot angezeigt.

```
function raeuberbeute2

h = 0.05;
tfin = 10;
T = 0:h:tfin;

a = 3;
b = 1.5;
c = 0.8;
d = 1.5;

q0 = [2 1];

[T,Q] = ode45(@(t,q) rhs(t,q,a,b,c,d), T, q0);
```

```
[u,v] = meshgrid(0:0.2:4.5, 0:0.2:4);
for i=1:size(u,1)
    for j=1:size(u,2)
        tmp = rhs(0,[u(i,j) v(i,j)],a,b,c,d);
   % Skalierung der Pfeile, damit diese nicht zu lange sind
        tmp = tmp/norm(tmp,2)/7.5;
        du(i,j) = tmp(1);
        dv(i,j) = tmp(2);
    end
end
quiver(u,v,du,dv,0)
hold on
plot(Q(:,1),Q(:,2))
hold off
xlabel('u(t)')
ylabel('v(t)')
title('Raeuber-Beute Phasenplot')
set(gcf,'paperpositionmode','auto')
set(gca,'xlim',[0 4.5],'ylim',[0 4],'fontsize',15)
print('-depsc','-loose','rb2');

function dq = rhs(t,q,a,b,c,d)
dq = q;
dq(1) = q(1)*(a-b*q(2));
dq(2) = q(2)*(c*q(1)-d);
```



1.7 Umformung von Differentialgleichungen höherer Ordnung

Numerische Verfahren zur Lösung von Anfangswertproblemen sind für Differentialgleichungen erster Ordnung konzipiert. Liegt eine Differentialgleichung höherer Ordnung vor, so muss diese in ein System erster Ordnung umgeformt werden. Dazu sei

$$y^{(k)} = f(x, y, y', y'', \ldots, y^{(k-1)})$$

mit $y : [a, b] \to \mathbb{R}$ gegeben. Nun wird $z \in \mathbb{R}^k$ eingeführt mit

$$z_1(x) = y(x),\ z_2(x) = y(x)',\ z_3(x) = y(x)'',\ \ldots,\ z_k = y^{(k-1)}(x)$$

und also

$$\frac{\mathrm{d}}{\mathrm{d}x} \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_{k-1} \\ z_k \end{pmatrix} = \begin{pmatrix} z_2 \\ z_3 \\ \vdots \\ z_k \\ f(x, z_1, z_2, \ldots, z_k) \end{pmatrix}.$$

Beispiel: Gelöst werden soll

$$\begin{aligned} &y''(x) = -y(x), \qquad x \in [0, 2\pi], \\ &y(0) = 0,\ y'(0) = 1. \end{aligned}$$

Umformung ergibt das System erster Ordnung

$$\begin{aligned} &\begin{pmatrix} z_1(x) \\ z_2(x) \end{pmatrix}' = \begin{pmatrix} z_2(x) \\ -z_1(x) \end{pmatrix}, \qquad x \in [0, 2\pi], \\ &z(0) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \end{aligned}$$

MatLab-Implementierung:

```
function dglsys

[X,Y] = ode45(@(t,q) rhs(t,q),[0 2*pi], [0 1]);

plot(X,Y(:,1),X,sin(X))
xlabel('$x$','Interpreter','latex');
ylabel('Approx. $U_k$ und Lsg. $u(x)$','Interpreter','latex');
title('Numerische Approximation')
set(gcf,'paperpositionmode','auto')
legend({'Approx. $U_k$','Loesung $u(x)$'},'Interpreter','latex')
set(gca,'xlim',[0 2*pi],'ylim',[-1 1],'fontsize',17)
print('-depsc','-loose','dglsyslsg');

plot(X,Y(:,1)-sin(X))
xlabel('$x$','Interpreter','latex');
ylabel('Fehler $U_k-u(x_k)$','Interpreter','latex');
title('Approximationsfehler')
set(gcf,'paperpositionmode','auto')
set(gca,'xlim',[0 2*pi],'ylim',[-2.E-4 2.E-4],'fontsize',17)
print('-depsc','-loose','dglsyserr');

function dq = rhs(t,q)
dq = q;
dq(1) = q(2);
dq(2) = -q(1);
```

Numerische Lösung sowie Abweichung zur analytischen Lösung: