MATLAB für gewöhnliche Differentialgleichungen

January 2020

1 Introduction

Empfehlung:
https://www.tu-ilmenau.de/fileadmin/media/simulation/Lehre/Vorlesungsskripte/MATLAB/Matlab_fuer_Ingenieure.pdf

1.1 Vektoren und Matrizen

Alle Rechnunge in Matlab basieren auf Vektoren und Matrizen. Matlab unterscheidet einen Spalten- und einen Zeilenvektor.

Man definiert Vektoren und Matrizen in eckigen Klammern:

```
>> v = [1 2 3 4 5];
```

definiert einen Zeilenvektor (leere Zeichen zwischen Vektorelemente)

```
>> u = [1, 2, 3, 4, 5];
```

definiert auch einen Zeilenvektor (Komma zwischen Vektorelemente)

```
>> u = [1; 2; 3; 4; 5];
```

definiert auch einen Spaltenvektor (Semikolon zwischen Vektorelemente)

```
>> u = [1 2 3; 4 5 6; 7 8 9; 10 11 12];
```

definiert eine 4×3 Matrix mit Zeilen [1 2 3], [4 5 6], [7 8 9], [10 11 12].

```
>> u = ones(r,s);
```

definiert eine Matrix mit 1 als Einträge und mit Dimensionen r Zeilen und s Spalten. Wenn eine Dimension gleich 1 ist, wird es ein Vektor (Spalten oder Zeilen).

```
>> u = zeros(r,s);
```

definiert eine Matrix mit 0 als Einträge und mit Dimensionen r Zeilen und s Spalten. Wenn eine Dimension gleich 1 ist, wird es ein Vektor (Spalten oder Zeilen).

```
>> u = 1:n;
```

definiert eine arithmetische Folge von 1 bis n mit Schrittweite 1 und speichert sie in den Vektor u.

```
>> u = a:h:b;
```

definiert eine arithmetische Folge von a bis b mit Schrittweite h und speichert sie in den Vektor u. Die Schrittweite kann auch negativ sein, wenn $a > b$ ist; dann erhält man eine abnehmende Folge von Zahlen.

Bemerkung: Semikolon am Ende der Befehle verursacht, dass das Ergebnis des Befehls nicht ausgeschrieben wird (aber das Ergebnis wird ausgerechnet und in eine Variable zugeordnet, falls wir das in dem Befehl tun, in diesem Fall u.)

1.2 Operationen

```
>> c = a + b;
>> c = a - b;
```

Summe (oder Differenz) von Zahlen, Vektoren oder Matrizen - Matlab kann das selbst unterscheiden. Die Dimensionen müssen passen, wie man es aus Algebra kennt.

```
>> c = a * b;
```

Ein Produkt von zwei Zahlen, Vektoren oder Matrizen. Die Dimensionen müssen passen (Spalten, Zeilen).

```
>> c = a / b;
```

Wenn a und b Zahlen sind, dann ist das "a durch b".

Mit Vektoren und Matrizen kann man folgendes tun:

```
>> c = a.*b;
>> c = a./b;
```

In diesem Fall geht es um eine Element-weise Multiplikation oder Dividierung.

```
>> c = a^2;
>> c = a.^2;
```

Das erste ist einfach eine Zahl a hoch 2, das zweite ist ein Vektor oder eine Matrix a element-weise hoch 2.

```
>> c = sqrt(a);
```

Die zweite Wurzel von a.

1.3 Skripte und Funktionen

In Matlab kann man den Code auf zwei verschiedenen Weisen schreiben.

In einem **Skript** schreibt man einfacht die Folge von Befehle, und diese werden dann nach Auslösen des Programms ausgeführt. Zum Beispiel

```
a = 1;
b = 10;
c = a + b;
```

definiert die Variable a mit dem Wert 1, Variable b mit dem Wert 10, und Variable c als Ergebnis der Operation $a + b$. Diese Art vom Programm fängt gleich mit Befehle an, es beinhaltet keinen Kopf.

In einer **Funktion** muss man vor der Folge von Befehle zuerst einen Kopf schreiben:

```
function [c,d] = name(a,b)
a = 1;
b = 10;
c = a + b;
d = a * b;
```

Das erste Wort ist immer "function", danach kommt die Liste von "Output Parameters" in eckigen Klammern, dann ein =, danach der Name von der Funktion (man kann das wählen), und zum Schluss die Listen von "Input Parameters" in runden Klammern. Die Werte von den Input Parameters werden von dem Hauptprogramm in diese Funktion geholt, sonst würde sie diese Funktion nicht kennen und könnte also die erfordeten Operationen nicht ausführen. Die Werte von c und d werde aus dieser Funktion ausgeholt.

1.4 Definition von eigenen Funktionen

Haben wir zum Beispiel folgendes Hauptprogramm (Skript):

```
a = 1;
b = 10;
c = a + b;
d = eigenefunktion(b,c);
```

wobei wir möchten, dass "eigenefunktion" b durch c dividiert. Dann können wir eine separate Funktion schreiben als:

```
function [d] = name(b,c)
d = b/c;
```

Hier werden die Werte von b und c als Input geholt, dann wird das Befehl $d = b/c$ ausgeführt, und das Ergebnis von d wird aus der Funktion rausgeholt und dann im Hauptprogramm in die Variable d zugeordnet.

Die Variablen in "eigenefunction" müssen sich nicht gleich nenne wie im Hauptprogramm. Folgendes würde auch gut funktioneren:

```
function [z] = name(x,y)
z = x/y;
```

Es werden nämlich nicht die Namen von Variablen, sondern nur ihre Werte in und aus der Funktion übergeben. Was aber wichtig ist, ist die ORDNUNG, in diesem Fall die Ordnung von "b, c" als Inputparameters im Hauptprogramm, denn so werden sie dann zu den Funktionsvariablen x, y (nur für die Funktion bekannt) zugeordnet.

1.5 Matlab-Funktionen für die Lösung von gewöhnlicher Differentialgleichungen

Generelles Befehl:

```
[T,Y] = solver(odefun,tspan,y0)
```

Solvers: ode45, ode23, ode113, ode15s, ode23s, ode23t, and ode23tb
odefun: Funktion mit der rechten Seite von der DGL
tspan: Zeitbereich für die Lösung
y0: Anfangswert

1.5.1 Eine Differentialgleichung

Betrachten wir die Differentialgleichung 1. Ordnung (siehe Datei gewDGL1.m)

$$y'(t) = ay(t)(1 - y(t))$$

mit Wachstumsparameter $a = 3.2$, gewünschtem Zeitintervall für $t \in (0, T] = (0, 5]$ für die Lösung, und mit der Anfangsbedingung $y(0) = 0.1$.

Um diese DGL mit Hilfe von Matlabfunktionen zu lösen, brauchen wir zuerst, die rechte Seite der DGL zu definieren. Das müssen wir in einem separaten Programm als Funktion machen:

```
function [f] = myfunction(t,y)
f = 3.2*y*(1-y);
```

Aufruf im Kommandofenster (oder Skript, oder Funktion):

```
[t,sol]=ode45(@myfunction,[0 5], 0.1);
```

Graph der Lösung:

```
plot(t,sol,'r','LineWidth',1.5)
xlabel('Time t','FontSize',14)
ylabel('Solution','FontSize',14)
set(gca,'FontSize',14)
```

Eine andere Möglichkeit ist, die rechte Seite direkt in ode45 zu definieren (siehe Datei gewDGL2.m):

```
[t,sol]=ode45(@(t,y) 3.2*y*(1-y),[0 5], 0.1);
```

oder (siehe Datei gewDGL3.m)

```
rechteseite = @(t,y) 3.2*y*(1-y);
[t,sol]=ode45(rechteseite,[0 5], 0.1);
```

1.5.2 Differentialgleichungssysteme

Van-der-Pol-Schwinger (siehe Datei gewDGL4.m):

$$\dot{y}(t) = \begin{pmatrix} \dot{y}_1(t) \\ \dot{y}_2(t) \end{pmatrix} = \begin{pmatrix} y_2(t) \\ \mu(1-y_1(t)^2)y_2(t) - y_1(t) \end{pmatrix}$$

mit Anfangsbedingung $y(0) = [2; 0]$ und Parameter $\mu = 1$ oder $\mu = 1000$.

Funktion mir der rechten Seite:

```
function dydt=myfunction2(t,y,mu) % mit weiterer Parameteruebergabe
dydt=[y(2);mu*(1-y(1)^2)*y(2)-y(1)];
```

und das Hauptprogramm:

```
tspan = [0 50];
mu = 1; % es geht um eine nicht-steife DGL, wir verwenden ode45
[t1,y1]=ode45(@myfunction2,tspan,[2;0],[],mu); % oder

mu = 1000; % es geht um eine steife DGL, wir verwenden ode15s
[t1000,y1000]=ode15s(@myfunction2,tspan,[2;0],[],mu);
```

Oder definieren wir die rechten Seiten direkt im Solveraufruf:

```
tspan = [0 50];
mu = 1; % es geht um eine nicht-steife DGL, wir verwenden ode45
[t1,y1]=ode45(@(t,y)  [y(2);mu*(1-y(1)^2)*y(2)-y(1)],tspan,[2;0],[],mu); % oder

mu = 1000; % es geht um eine steife DGL, wir verwenden ode15s
[t1000,y1000]=ode15s(@(t,y)  [y(2);mu*(1-y(1)^2)*y(2)-y(1)],tspan,[2;0],[],mu);
```

1.6 Weitere Beispiele

Beispiel 1.

$$\begin{aligned} y_1' &= y_2 y_3 \\ y_2' &= -y_1 y_3 \\ y_3' &= -0.51 y_1 y_2 \end{aligned}$$

mit $y_1(0) = 0, y_2(0) = 1, y_3(0) = 1, t \in [0, 12]$.

Lösung. Funktion mit den rechten Seiten:

```
function dy = rigid(t,y)
dy = zeros(3,1);    % Spaltenvektor
dy(1) = y(2) * y(3);
dy(2) = -y(1) * y(3);
dy(3) = -0.51 * y(1) * y(2);
```

und das Hauptprogramm:

```
[T,Y] = ode45(@rigid,[0 12],[0 1 1]);
plot(T,Y(:,1),'-',T,Y(:,2),'-.',T,Y(:,3),'.')
```

Beispiel 2.

$$y'' = \frac{A}{B} t y$$

mit $y(0) = 0, y'(0) = 0.1, t \in [0, 5]$. Diese Aufgabe kann auf ein DGL-System 1. Ordnung umgeschrieben werden:

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= \frac{A}{B} y_1 \end{aligned}$$

Lösung. Funktion mit den rechten Seiten:

```
function dydt = odefcn(t,y,A,B)

dydt = zeros(2,1);
dydt(1) = y(2);
dydt(2) = (A/B)*t.*y(1);
```

und das Hauptprogramm:

```
A = 1;
B = 2;
tspan = [0 5];
y0 = [0 0.01];
[t,y] = ode45(@(t,y) odefcn(t,y,A,B), tspan, y0);

plot(t,y(:,1),'-o',t,y(:,2),'-.')
```