Modellierung und Programmierung

Dr. Martin Riplinger

6.2.2013

Arbeiten mit Dateien

spezielles Matlab-Binärformat (Endung: `.mat`)

`save(Dateiname, [Var1,...,VarN])`	speichert die Variablen `Var1` bis `VarN` (bzw. alle im Workspace enthaltenen) in die Datei `Dateiname.mat`
`load(Dateiname, [Var1,...,VarN])`	lädt die Variablen `Var1` bis `VarN` (bzw. alle enthaltenen) aus der Datei `Dateiname.mat` in den Workspace

Textdateien

`fopen, fprintf, fscanf, fclose`	(nahezu) analog zu C, siehe Matlabhilfe für Details.

Binärdateien

`fopen, fread, fwrite, fclose`	(nahezu) analog zu C, siehe Matlabhilfe für Details.

verschiedene Dateiformate, z.B. xls, bmp, png, jpg, ... und (einfache) Textdateien

`importdata(Dateiname,[Delimiter],[Kopfzeilen])`	importiert zahlreiche Formate, wobei `Delimiter` das Spaltentrennzeichen festlegt und die `Kopfzeilen` beim Einlesen übersprungen werden

Fallstricke

Achtung: In Matlab ist fast alles erlaubt!

Führt man das folgende Skript in Matlab aus, so erhält man keine Fehlermeldung und auch der Code Analyzer gibt keine Warnungen aus!

```
sum = 0;
for i = 1:10
    sum = sum + i;
end
pi = 1;
```

Folgende **Probleme** treten anschließend auf:

1. `»sum([1 2])`
   `Index exceeds matrix dimensions.`
   Merke: Funktionen werden durch Variablen mit gleichem Namen überdeckt!
2. `»z=1+2*i`
   `z=21.`
   Merke: Möchte man mit komplexen Zahlen arbeiten, so muss man auf `i` oder `j` als Laufindex für Schleifen verzichten!
3. `»sin(pi)`
   `ans=0.8415.`
   Merke: Selbst Konstanten wie π können durch beliebige Werte überschrieben werden!

Matlab-Profiler und Code Analyzer

Code Analyzer

Während der Eingabe wird eine Syntax-Prüfung des Codes durchgeführt, auffällige Stellen werden unterschlängelt. Zudem werden Syntax-Fehler mit roten und Warnungen durch orangefarbene Balken in der Scrollleiste markiert.

`mlintrpt`	Anzeige des Code Analyzer Reports für alle Dateien im `current folder`. (alternativ Klick auf „Analyze Code“)
`mlintrpt('Dateiname')`	Anzeige des Code Analyzer Reports für die Datei `Dateiname.m`. (alternativ „Show Code Analyzer Report“ im Editor)

Matlab-Profiler

- Suche nach Optimierungspotential, meist ist nur ein kleiner Teil des Programms für lange Rechenzeiten verantwortlich.
- Debugging des Matlab-Codes

`profile on`	Profiling starten.
`profile viewer`	Profile Summary anzeigen. (alternativ: „Run and Time“ im Editor)

geschicktes Programmieren in Matlab: Binomialkoeffizient

langsame Variante:

```
function [ binom ] = binom_l( n,k )

if (n < 0 || k < 0)
    disp(['Bitte zwei positive,'...
          'Zahlen eingeben']);
    binom = NaN;
elseif n >= k
    if k > n/2, k = n-k; end
    zaehler = 1;
    nenner = 1;
    for l = n-k+1:n
        zaehler = zaehler * l;
    end

    for l = 1:k
        nenner = nenner * l;
    end
    binom = zaehler / nenner;
else
    binom = 0;
end

end
```

schnellere Variante:

```
function [ binom ] = binom_s( n,k )

if (n < 0 || k < 0)
    disp(['Bitte zwei positive,'...
          'Zahlen eingeben']);
    binom = NaN;
elseif n >= k
    if k > n/2, k = n-k; end
    binom = prod(((n-k+1):n)./(1:k));
else
    binom = 0;
end

end
```

Bemerkung: Die schnellere Variante ist zudem stabiler, da für große k die Variablen `zaehler` und `nenner` nicht über alle Schranken wachsen.

geschicktes Programmieren in Matlab: Funktionsauswertungen

langsame Variante:

```
function [x,y] = funkauswert_l( t )
x = zeros(1,length(t));
y = zeros(1,length(t));

for l = 1:length(t)
    x(l) = t(l)^2;
    y(l) = log2(t(l));
end

end
```

schnelle Variante:

```
function [x,y] = funkauswert_s( t )

x = t.^2;
y = log2(t);

end
```

geschicktes Programmieren in Matlab: Vandermonde-Matrix

$$V := \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ 1 & x_3 & x_3^2 & \dots & x_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^{n-1} \end{pmatrix}$$

häufige Anwendung: Beschreibung einer Polynom-Interpolation.

langsame Variante:

```
function [ V ] = vander_l( x )

for l = 1:length(x)
    for m = 1:length(x)
        V(l,m) = x(l)^(m-1);
    end
end

end
```

schnelle Variante:

```
function [ V ] = vander_s( x )

n = length(x);
x = x(:); %x ist nun Spaltenvektor
V = ones(n);
for l = 1:n-1
    V(:,l+1) = x.*V(:,l);
end

end
```

Bemerkung: Die Matlab-Funktion `vander` berechnet eine modifizierte Version der Vandermonde-Matrix.

geschicktes Programmieren in Matlab

Um die Geschwindigkeit der Funktionen zu testen, wird mit dem Matlab-Profiler das folgende Skript analysiert:

```
B1 = ones(100);
B2 = ones(100);
for n = 1:100
    for k = 1:n
        B1(n,k) = binom_l(n, k);
        B2(n,k) = binom_s(n, k);
    end
end

t = 0:0.00001:1;
[f1,f2] = funkauswert_s(t);
[g1,g2] = funkauswert_l(t);

x = 0.1:0.001:1;
v1 = vander_l(x);
v2 = vander_s(x);
```

geschicktes Programmieren in Matlab: Ergebnis des Profilers

Profile Summary

Generated 04-Feb-2013 09:25:33 using cpu time.

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
startscript	1	0.681 s	0.009 s	
vander_l	1	0.564 s	0.564 s	
binom_l	5050	0.049 s	0.049 s	
funkauswert_l	1	0.032 s	0.032 s	
binom_s	5050	0.021 s	0.021 s	
vander_s	1	0.004 s	0.004 s	
funkauswert_s	1	0.003 s	0.003 s	

Self time is the time spent in a function excluding the time spent in its child functions.

allgemeine Tipps:

- (Große) Matrizen sollten vor ihrer Verwendung mit der **maximal benötigten** Größe vorbelegt werden (z.B. mit `zeros(M,N)`).
- `For`-Schleifen sollten (falls möglich) durch **Vektorisierung des Codes** vermieden werden. Dies führt zu effizienterem und übersichtlicherem Code.
- Ausgaben im Command Window sowie grafische Ausgaben benötigen viel Zeit und sollten daher nur mit Bedacht eingesetzt werden.
- Löschen nicht mehr benötigter Variablen (mit `clear Name`) gibt dadurch belegten Speicher wieder frei.
- Jeder Aufruf von Skripten oder Funktionen, die in separaten Dateien gespeichert sind, benötigt zusätzlich Zeit. Wird eine Funktion in einer `for`-Schleife häufig aufgerufen, so erhält man schnelleren Code, indem man den Inhalt der Funktion (entsprechend angepasst) in die Schleife kopiert.
  Achtung: Dadurch verschlechtert sich die Lesbarkeit sowie die Wartung des Codes, daher nur in geschwindigkeitskritischen Stellen verwenden!
- `Function Handles (@)`, welche im Wesentlichen mit dem Konzept von Funktionszeigern in C übereinstimmen, können oft gewinnbringend eingesetzt werden.

Graphiken mit Matlab

Graphiken können direkt in Matlab erstellt werden!

`figure`	erzeugt ein neues Figure
`hold on`	schützt ein Fenster vor Überschreiben
`hold all`	schützt ein Fenster vor Überschreiben, die Linienfarbe wird automatisch gewechselt
`legend(String1,String2,...)`	fügt der Grafik eine Legende hinzu
`axis equal`	Achseneinheit in alle Richtungen gleich lang
`set(...) / get(...)`	Eigenschaften setzen/anzeigen lassen
`gcf / gca`	aktuelles Figure- bw. Achsen-Handle

2D-Plots:

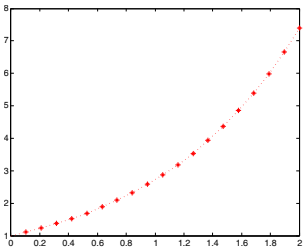
`plot(x,y)`	plottet den Vektor y gegen den Vektor x
`plot(y)`	plottet den Vektor y gegen dessen Indizes
`subplot(m, n, zaehler)`	erstellt ein Figure für `m` $\times$ `n` - Subplots, $1 \leq$ `zaehler` $\leq$ `m` $\cdot$ `n` bezeichnet das aktuelle Fenster, wobei zeilenweise gezählt wird.

Allgemeine Syntax:

`plot(x,y,'FMS',...,'Eigenschaft', 'Wert')`

- FMS = Farbe Marker Linientyp

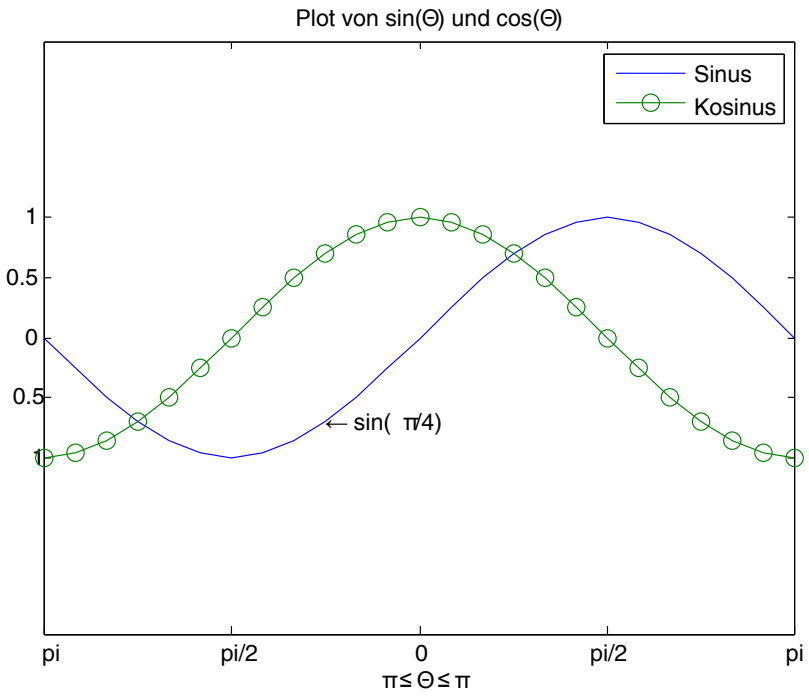
  z.B. `'r*:'` zeichnet rote gepunktete Linie mit *-Marker
- Eigenschaft = `'LineWidth'`, `'MarkerSize'`,...



2D-Plot: Beispiel

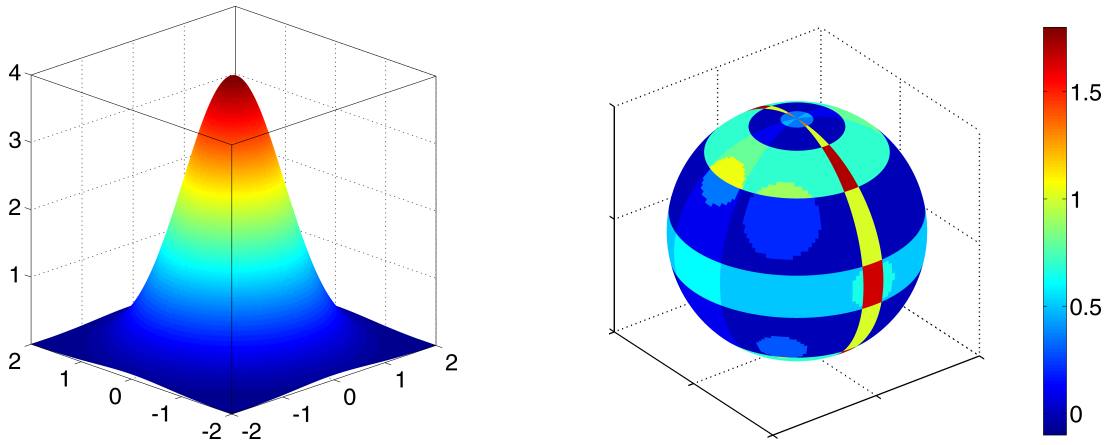
```
1  theta = linspace(-pi,pi,25);
2  sinus = sin(theta);
3  kosinus = cos(theta);
4
5  %figure
6  hold all;
7  plot(theta,sinus);
8  plot(theta,kosinus,'-o','MarkerSize',8);
9
10 set(gca,'XTick',-pi:pi/2:pi);
11 set(gca,'XTickLabel',{'-pi','-pi/2','0','pi/2','pi'});
12 xlabel('-\pi \leq \Theta \leq \pi');
13 set(gca,'YTick',-1:0.5:1);
14 title('Plot von sin(\Theta) und cos(\Theta)');
15 text(-pi/4,sin(-pi/4),'\leftarrow sin(-\pi/4)',...
16      'HorizontalAlignment','left') ;
17 legend('Sinus','Kosinus');
18 axis([-pi pi -1 1]);
19 axis equal;
20 box;
```

2D-Plot: Beispiel



- Der Plot Editor bietet zahlreiche Möglichkeiten der Nachbearbeitung.
- Bei Speicherung als `.fig` ist eine Nachbearbeitung auch später noch möglich.
- Export in alle Standardformate möglich: PNG, JPG, EPS, PDF, ...

3D Grafiken: Beispiele



Beispiel: Leastsquare

Gegeben: N Messwerte $(x_1, y_1), \ldots, (x_N, y_N)$, die als $N \times 2$-Matrix in einer Textdatei (ohne Kommentare) gespeichert sind.

Aufgabenstellung: Berechne die zugehörige Bestgerade mit Hilfe der Methode der kleinsten Quadrate (vergleiche Aufgabe 2 auf Übungsblatt 6).

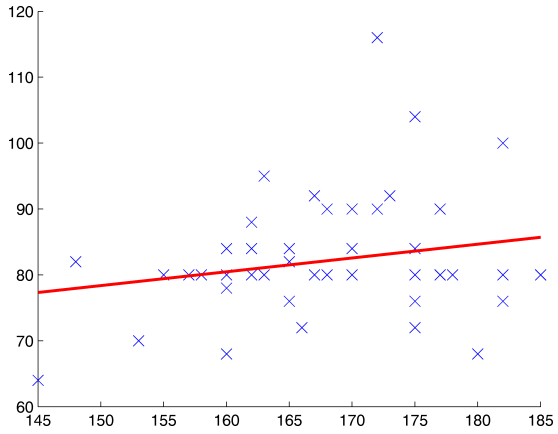
Ziele: Schreibe eine Funktion, die,

- falls nur ein Rückgabewert angefordert wird, nur die Steigung der Geraden berechnet und diese zurückgibt,
- falls zwei Rückgabewerte angefordert werden, die Steigung und den y-Achsenabschnitt der Geraden zurückgibt,
- die Punktewolke und die berechnete Gerade plottet, sofern der Parameter `Plotschalter` den Wert 1 besitzt.

Beispiel: Leastsquare

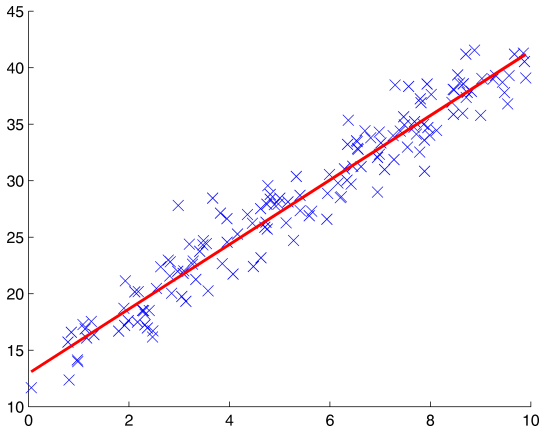
```
1  function [ a,b ] = leastsquare( Dateiname, Plotschalter )
2  daten = importdata(strcat(Dateiname,'.dat'));
3  [N,M] = size(daten);
4  if M ~= 2 disp('Falsche Dimension!'); end;
5  tquer = sum(daten(:,1))/N;
6  yquer = sum(daten(:,2))/N;
7  %alterantiv: s = sum(daten)/N; tquer2 = s(1); yquer2 = s(2);
8  a = (daten(:,1)-tquer)'*(daten(:,2)-yquer)/sum((daten(:,1)-tquer).^2);
9
10 if nargout == 2 || (nargin == 2 && Plotschalter == 1)
11     b = yquer-a*tquer;
12     if nargin == 2 && Plotschalter == 1
13         figure('Name','leastsquare')
14         hold all
15         plot(daten(:,1),daten(:,2),'x','MarkerSize',10)
16         tmin = min(daten(:,1));
17         tmax = max(daten(:,1));
18         plot([tmin;tmax],[a*tmin+b;a*tmax+b],'r','LineWidth',2);
19     end
20 end
21
22 end
```

Beispiel: Leastsquare



Datensatz: leastsquare.dat
(siehe Homepage, Übung)

```
leastsquare('leastsquare',1);
```



weiterer Datensatz

```
leastsquare('testdaten',1);
```

Erstellen von Filmen mit Matlab

`F = getframe(H)`	Kreieren eine Bildabfolge, Figure H wird in eine spezielle Struktur F (Frame) gespeichert.
`movie(F,n)`	*n*-maliges Abspielen des Frames F
`Obj = VideoWriter(Dateiname);`	Erstellung des Objekts `Obj`, um Videos in eine `.avi`-Datei zu schreiben
`open(Obj);`	Öffnen der Datei, welche `Obj` zugewiesen ist.
`writeVideo(Obj,F)`	Schreibt den Frame in die `Obj` zugewiesenen Datei.
`close(Obj);`	Schließen der Datei, welche `Obj` zugewiesen ist.

Beispiel: (primitive) Simulation eines Aktienkurses

Teil 1 des Skriptes: Parameter setzen und Werte berechnen.

```
%% Parameter
startwert = 100;
tage = 30;
anzahl = 20;
max_schwank = 0.2*startwert;
tendenz = 0.01;

%% Berechnung
wert = zeros(anzahl,tage+1);
wert(:,1) = startwert;
for k = 1:tage
    wert(:,k+1) = wert(:,k)+2*(rand(anzahl,1)-0.5+tendenz)*max_schwank;
end
disp(['geschätzter Endwert: ', num2str(mean(wert(:,tage+1)),'%.2f')]);
```

Beispiel: (primitive) Simulation eines Aktienkurses

Teil 2 des Skriptes: Movie erzeugen und als `aktie.avi` speichern.

```
%% Figure und Movie
figure;
hold on;
axis([1 tage startwert-5*max_schwank startwert+10*max_schwank])
t = linspace(1,tage+1,tage+1);
col = lines(anzahl);

mov(anzahl*(tage-1)) = struct('cdata',[],'colormap',[]);
hilf = 0;
for l = 1:anzahl
    for k = 2:tage
        plot(t(k-1:k),wert(l,k-1:k),'LineWidth',2,'color',col(l,:));
        hilf = hilf + 1;
        mov(hilf) = getframe(gcf);
    end
end

writerObj = VideoWriter('aktie.avi');
open(writerObj);
writeVideo(writerObj,mov);
close(writerObj);
```

Beispiel: (primitive) Simulation eines Aktienkurses