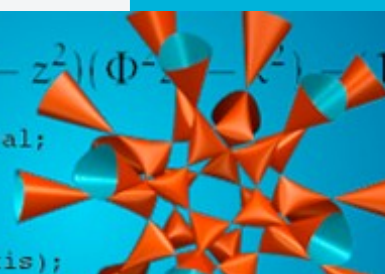


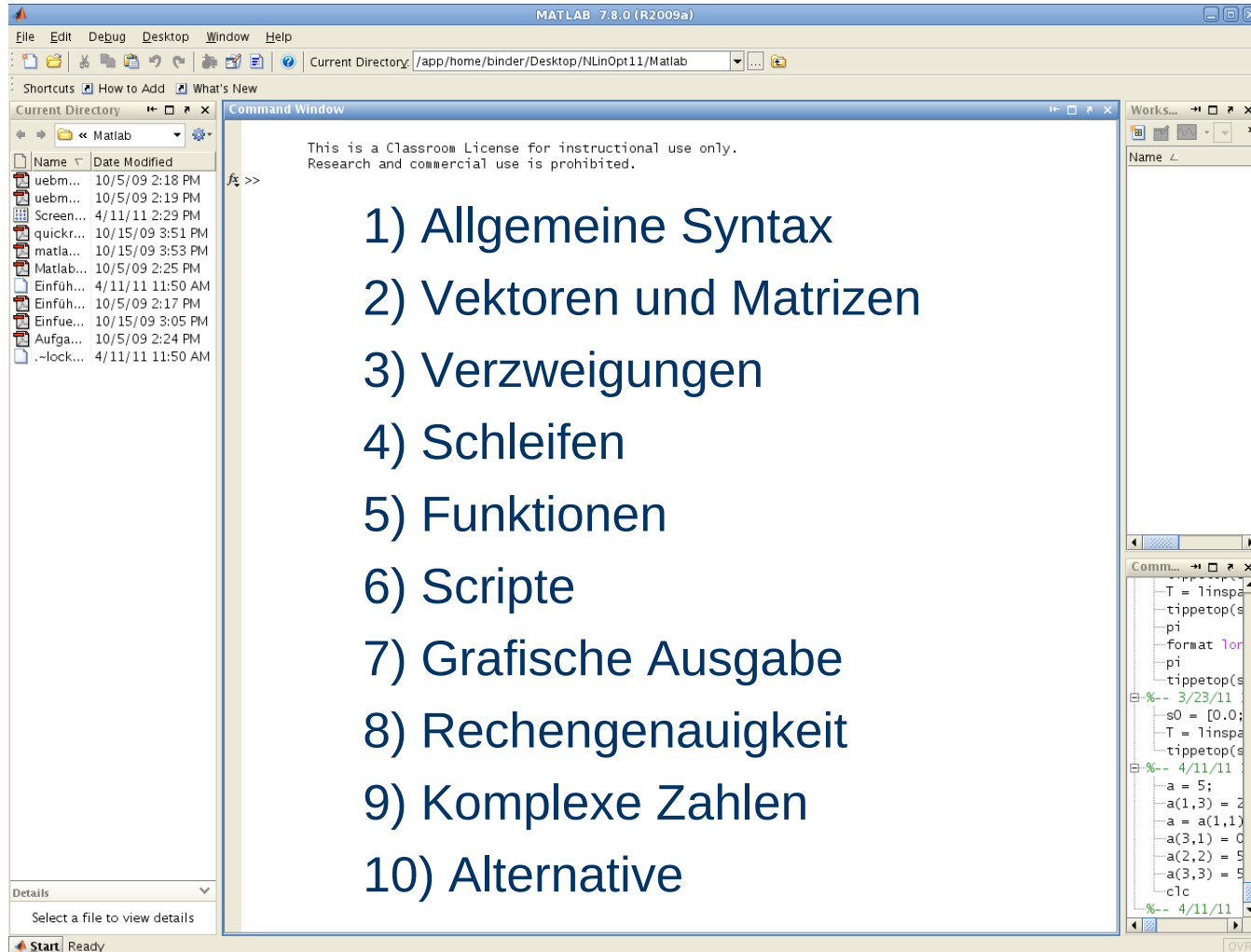
Kurzeinführung in Matlab

Lineare Optimierung
WS 2011/2012
Prof. Dr. E. Kostina

```
1)  $(\Phi^2 x^2 - y^2)(\Phi^2 y^2 - z^2)(\Phi^2 z^2 - x^2) = 1$   
perspective=central;  
spec_s=150.0;  
radius=10.0;  
sextic=rotate(  
    sextic,-0.1,xAxis);
```

A 3D plot of a complex, multi-lobed surface, likely a sextic surface, rendered in orange and red. The surface is composed of many small, triangular facets, creating a faceted appearance. It is set against a dark blue background.

Inhaltsverzeichnis



The image shows the MATLAB 7.8.0 (R2009a) interface. The Command Window displays the text: "This is a Classroom License for instructional use only. Research and commercial use is prohibited." Below this, a list of 10 topics is overlaid in large blue font. The File Explorer on the left shows the current directory as /app/home/binder/Desktop/NLinOpt11/Matlab. The Command Window on the right shows a script with the following code:

```
T = linspace(0, 2, 10);  
tippetop(s) = pi;  
format long g  
pi  
tippetop(s) = pi;  
%-- 3/23/11  
s0 = [0.0; 0.0; 0.0];  
T = linspace(0, 2, 10);  
tippetop(s) = pi;  
%-- 4/11/11  
a = 5;  
a(1,3) = 2;  
a = a(1,1) + i;  
a(3,1) = 0;  
a(2,2) = 5;  
a(3,3) = 5;  
clc  
%-- 4/11/11
```

- 1) Allgemeine Syntax
- 2) Vektoren und Matrizen
- 3) Verzweigungen
- 4) Schleifen
- 5) Funktionen
- 6) Scripte
- 7) Grafische Ausgabe
- 8) Rechengenauigkeit
- 9) Komplexe Zahlen
- 10) Alternative

Allgemeine Syntax

Besonderheiten von Matlab

- MATLAB = MATrix LABoratory
⇒ Matlab fasst alle Variablen als Matrizen auf
- $a = 5;$ % Skalare sind (1,1)-Matrizen
- $a(1,3) = 2;$ % Erweiterung von a zu einem Zeilenvektor $a = (5,0,2)$
- $a = a(1,1);$ % Wiederherstellung von $a = 5$
- $a(3,1) = 0;$ % Erweiterung von a zu einem Spaltenvektor $a = (5,0,0)^T$
- $a(2,2) = 5; a(3,3) = 5;$
 % Erweiterung von a zu einer (3,3)-Matrix mit Fünfen auf
 der Diagonalen
- Das Semikolon am Ende der Zeile unterdrückt die Ausgabe der Eingabe.

Die Matlab-Hilfsfunktion

- Weiß man, welche Funktion man benutzen möchte, erinnert sich aber nicht an die genau Syntax (in welcher Reihenfolge müssen die Parameter übergeben werden?), dann kann man im Command Window die Hilfe dazu aufrufen:

`help fmincon`

- Alternativ gibt es eine Online-Hilfe, die man über die Taskleiste oben aufrufen kann.
- Weiß man nicht, wie die benötigte Matlab-Funktion heißt, kann man die Hilfetexte mit `lookfor` nach einem Stichwort durchsuchen und bekommt alle Funktionen aufgelistet, die passen könnten:

`lookfor 'linear programming'`

Vektoren und Matrizen

Initialisierung von Vektoren und Matrizen

- $u = [1 \ 2 \ 3];$
 $u = [1, 2, 3];$ % Zeilenvektor $u = (1 \ 2 \ 3)$
- $v = [1; 2; 3];$ % Spaltenvektor $v = (1 \ 2 \ 3)^T$
- $M = [1 \ 2 \ 3; 4 \ 5 \ 6];$ % Matrix mit zwei Zeilen und drei Spalten

- $E = \text{eye}(n);$ % (n,n)-Einheitsmatrix
- $N = \text{zeros}(n,m);$ % Nullmatrix mit n Zeilen und m Spalten
- $O = \text{ones}(n,m);$ % Matrix mit n Zeilen und m Spalten gefüllt mit 1

- Vektoren sind Matrizen, in denen eine Dimension gleich 1 ist!

Der :-Operator --- einfache Anwendung

- Zur einfacheren Erzeugung von z.B. Indexvektoren:

$u = 1:3$ ist dasselbe wie $u = [1 \ 2 \ 3]$

$v = (1:3)'$ ist dasselbe wie $v = [1; 2; 3]$

- Zum Aufrufen von Teilvektoren:

$u(1:2)$ liefert die ersten beiden Einträge in u (Zeilenvektor)

$v(2:3)$ liefert die letzten beiden Einträge von v (Spaltenvektor)

$M(2, :)$ liefert die zweite Zeile der Matrix M

$M(1:2, 2:3)$ liefert die Teilmatrix von M , die aus den Einträgen der 1. und 2. Zeile in der 2. und 3. Spalte besteht

- Zusammenfassend: $a:b$ bedeutet “von a bis b ”

$:$ bedeutet “alle Einträge dieser Dimension”

Der :-Operator --- doppelte Anwendung

- Zur zusätzlichen Festlegung einer anderen Schrittweite als 1:

$w = 0:5:15$ ist dasselbe wie $w = [0, 5, 10, 15]$

$M(1:5:16, 1:3:16)$ liefert die Einträge der 1., 6., 11., 16. Zeile in der 1., 4., 7., 10., 13., 16. Spalte von M

- Zusammenfassend: $a:n:b$ bedeutet “von a jedes n-te Element (dieser Dimension) bis b”

Rechenoperationen

- Transponieren einer Matrix:

A' liefert A^T

- Skalarprodukt zweier Vektoren:

$u'v$ liefert das Skalarprodukt $u^T v$ (äquivalent: $\text{dot}(u,v)$)

- Kreuzprodukt (Vektorprodukt zweier Vektoren):

$\text{cross}(u,v)$

- Matrix-Vektor-Multiplikation:

$A*b$ liefert den Vektor Ab

- Matrix-Matrix-Multiplikation:

$A*B$ liefert das übliche Matrixprodukt AB

- Inverse einer Matrix:

$\text{inv}(A)$ liefert das Inverse von A (äquivalent: A^{-1})

Komponentenweise Rechenoperationen

- Ein Punkt vor dem Rechenzeichen bedeutet, dass die Operation komponentenweise ausgeführt wird:

$a.^2$ liefert das Skalarprodukt von a mit sich selbst

$a.^2$ liefert einen Vektor mit Einträgen $a(1)^2, \dots, a(n)^2$

$a*b$ liefert das Skalarprodukt von a mit b

$a.*b$ liefert einen Vektor mit Einträgen $a(1)*b(1), \dots, a(n)*b(n)$

analog für Matrizen

- **Wichtig:** der Anwender hat selbst darauf zu achten, dass die Dimensionen der Matrizen und Vektoren jeweils zueinander passen, sonst erzeugt Matlab eine Fehlermeldung!

Lösung von Gleichungssystemen

- Annahme: Das Gleichungssystem $Ax=b$ ist eindeutig lösbar, d.h.
 $x = A^{-1} * b$
- Möglichkeit 1: $x = \text{inv}(A)*b$;
Inversenberechnung ist oft sehr aufwändig und kann viel Rechenzeit beanspruchen \Rightarrow sollte vermieden werden
- Möglichkeit 2: $x = A \setminus b$;
löst das Gleichungssystem mit Hilfe einer QR-Zerlegung (Gauß-Algorithmus) der Matrix $A \Rightarrow$ gerade bei großen Systemen wesentlich schneller

Verzweigungen

If ... else ...

- Einfache Fallabfrage:

```
if (Bedingung1)
```

```
    Fall 1
```

```
elseif (Bedingung2)
```

```
    Fall 2
```

```
else
```

```
    sonst
```

```
end
```

Beispiel:

```
if (k == 1)
```

```
    disp('k ist 1.');
```

```
elseif (k == 2)
```

```
    disp('k ist 2.');
```

```
else
```

```
    disp('k ist weder 1 noch 2.');
```

```
end
```

Switch ... case ...

- Alternative Fallunterscheidung nach numerischen Werten:

```
switch Variable
    case Wert1
        Fall 1
    case Wert2
        Fall 2
    otherwise
        sonst
end
```

Beispiel:

```
switch k
    case 1
        disp('k ist 1.');
```

```
case 2
        disp('k ist 2.');
```

```
otherwise
        disp('k ist weder 1 noch 2.');
```

```
end
```

Relations- und logische Operatoren

- Relationsoperatoren:

Matlab	Bedeutung
<	<
>	>
<=	≤
>=	≥
==	=
~=	≠

- Logische Operatoren:

Matlab	Bedeutung
~	¬ (nicht)
&&	∧ (und)
	∨ (oder)

- Signalwörter “true” und “false” existieren in Matlab nicht; es gilt: 0 bedeutet “falsch” und jede andere Zahl bedeutet “wahr”

Schleifen

Syntax

- Es gibt for- und while-Schleifen:

```
for index = start(:schritt):ende
    Schleifenkörper
end
```

```
while Bedingung
    Schleifenkörper
end
```

Beispiel:

```
for i = 1:2:n
    a(i) = i^2;
end
```

```
while i < (n/2)
    a(i) = i^2;
    i = i+1;
end
```

- Mit dem Befehl break können Schleifen vorzeitig verlassen werden.

Alternativen

- Schleifen sind sehr rechenaufwändig und sollten möglichst vermieden werden. In Matlab ist das oft ganz einfach:

```
for i=1:n
```

```
    A(2,i) = b(i);
```

ist dasselbe wie

```
A(2,:) = b;
```

```
end
```

```
for i=1:n
```

```
    quad(i) = i^2;
```

ist dasselbe wie

```
quad = (1:n).^2;
```

```
end
```

Funktionen

Vorimplementierte Funktionen

- In Matlab sind viele wichtige Funktionen vorimplementiert:
 - Trigonometrie: sin, cos, tan, asin, acos, atan, sinh, cosh, tanh
 - Gleichungssysteme: \, mldivide, ...
 - Optimierung: fmincon, quadprog, ...
- Viele wichtige Funktionen gibt es aber auch nicht:
 - Newton-Verfahren
 - Simplex-Verfahren

Beispiel (vector.m)

```
% Kommentare vor dem Funktionsaufruf mit einer Beschreibung der  
% Funktion sollten nie fehlen, da diese die Funktion mit "help  
% funktionsname" ausgegeben und mit "lookfor ..." durchsucht  
% werden
```

```
function [vec] = vector(n)
```

```
    vec1 = 1:n;           % Initialisierung mit 1 bis n  
    vec2 = n+1:2*n;      % Initialisierung mit n+1 bis 2n  
    vec = [vec1, vec2];  % Zusammenfügen
```

```
end
```

- Es wird das Argument in den eckigen Klammern (hier: vec) zurückgeliefert und kann auch direkt einer Variablen zugewiesen werden: `var = vector(3)` liefert `var = (1, 2, 3, 4, 5, 6)`

inline-Funktionen

- Die explizite Definition neuer Funktionen ist für einfach Zusammenhänge nicht nötig.
- Einfache Konstruktionen lassen sich direkt übergeben:

```
f = inline('x.^3 + 2*x.^2', 'x');
```

erzeugt die komponentenweise Funktion $f(x) = x^3 + 2x^2$

- Faustregel: Eine inline-Funktion muss sich als ein einziger Ausdruck schreiben lassen, d.h. keine Verzweigungen oder Schleifen o.ä.

Scripte

Erstellung von Scripten

- Scripte funktionieren im Wesentlichen wie Funktionen, haben aber keinen Namen, mit dem sie in anderen Zusammenhängen wiederverwendet werden können, d.h. Jeder Funktionskörper ohne die Anfangszeile `function [ret] = name(param)` und die Endung `end` ist auch ein Script.
- Beispiel von eben:

```
vec1 = 1:n;           % Initialisierung mit 1 bis n
vec2 = n+1:2*n;      % Initialisierung mit n+1 bis 2n
vec = [vec1, vec2];  % Zusammenfügen
```

ist für festes n ein Script; die Ausgabe erfolgt z.B. durch Weglassen den letzten Semikolon.

Script vs. Funktion

- Scripte stehen für sich allein und können keine Parameter übergeben bekommen. Funktionen können aus anderen Funktionen oder Scripten heraus (ggf. mit Parametern) aufgerufen werden.
- Alle in einem Script erzeugten Variablen bleiben auch nach Beendigung des Scripts bestehen. In einer Funktion erzeugte Variablen müssen dafür explizit zurückgegeben und an der aufrufenden Stelle gespeichert werden: soll die Ausgabe von `function [a,b] = fun(c)` weiterverwendet werden, muss man sie explizit abspeichern: `[x,y] = fun(z)`.
- Scripte sammeln also auch nicht mehr benötigte Variablen, die nur den Speicher belasten. Der Befehl `clear` schafft hier Abhilfe.

Grafische Ausgabe

Plots

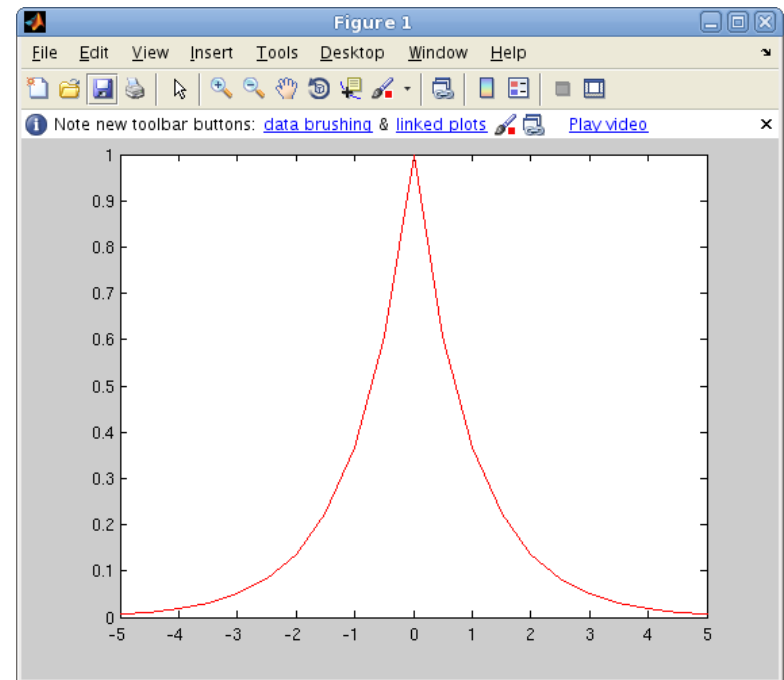
- Funktionsgraphen lassen sich sehr gut mit dem plot-Befehl ausgeben:

```
x = -5:0.5:5;
```

```
f = inline('exp(-abs(x))', 'x');
```

```
plot(x, f(x), '-r')
```

- Der letzte Parameter der plot-Funktion bestimmt das Aussehen der Ausgabe:
'-r' gibt den Graphen in Form einer durchgezogenen roten Linie aus.



Grafikexport und Pixelgrafik

- Mit plot erzeugte Bilder können in diverse Grafikformate exportiert werden. Dies kann entweder durch die im Grafikfenster verfügbare Toolbox geschehen oder mit dem Befehl `saveas`.
- Matlab kann jedes Bild auch als Matrix interpretieren, wobei jedes Pixel einem Matrixeintrag entspricht und jeder Zahlwert einer Farbe (abhängig von der verwendeten Colormap). Umgekehrt kann natürlich auch jede beliebige Matrix ein Bild darstellen.
- Die dazugehörigen Befehle lauten `imread` (Bild als Matrix) und `imagesc` (Matrix als Bild). Beispielbilder zur Demonstration verschiedener Colormaps liefert der Befehl `imageext`.

Rechengenauigkeit

Rechen- und Ausgabegegenauigkeit

- Standardmäßig werden in Matlab Rechenergebnisse mit 4 Nachkommastellen ausgegeben.
- Die interne Rechengenauigkeit ist größer, ca. 16 Nachkommastellen.
- Die Ausgabe und Darstellung der Ergebnisse kann mit dem Befehl **format** geändert werden:

format short

pi = 3.1416

format short e

pi = 3.1416e+00

format long

pi = 3.141592653589793

format long e

pi = 3.141592653589793e+00

(4 Nachkommastellen bzw. 15 Nachkommastellen)

Komplexe Zahlen

Syntax

- Matlab rechnet automatisch mit komplexen Zahlen, wenn Rechenergebnisse nicht mehr reell sind:
`sqrt(-4)` funktioniert und liefert das Ergebnis $0 + 2.0000i$
- Die Eingabe komplexer Zahlen kann auf verschiedene Weise erfolgen:

i. `z = 1.234 + 5.678i;`

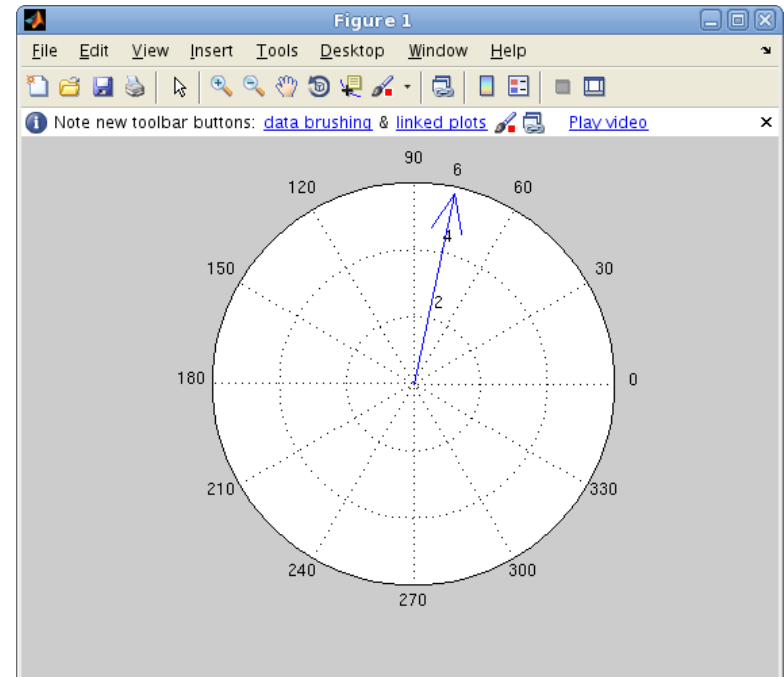
ii. `z = complex(1.234, 5.678);`

iii. `a = 1.234; b = 5.678;`

`z = a+b*1i;` % es muss heißen "1i" und nicht nur "i", um
eine Verwechslung mit Laufindizes in
Schleifen zu vermeiden

Komplexe Konjugation und Polardarstellung

- Die komplex konjugierte Zahl erhält man über den Befehl `conj`:
 $z = 1.234 + 5.678i$; `conj(z)` liefert $1.2340 - 5.6780i$
- Bei Berechnung des Skalarprodukts zweier komplexer Zahlen mit `dot(z1,z2)` bildet Matlab intern automatisch das komplex Konjugierte der ersten Zahl.
- Mit dem Befehl `compass` kann man sich eine Polardarstellung komplexer Zahlen anzeigen lassen.



Alternative

Alternative zu Matlab

- Matlab ist ein kostenpflichtiges Programm, das man unter <http://www.mathworks.com> auch in einer kostengünstigeren Studentenversion beziehen kann.
- Es ist auf den Rechnern des FB12 installiert.
- Ein kostenlose Alternative zu Matlab ist Octave (<http://www.gnu.org/software/octave/>).
- Dieses ist zwar weitgehend kompatibel zu Matlab, aber nicht völlig; d.h. in Octave geschriebene Software läuft meistens auch unverändert unter Matlab, aber eben nicht immer. Es gibt auch nicht für alle Betriebssysteme brauchbare grafische Benutzeroberflächen.