Einführung in MATLAB

Übungen Musterklassifikation

Jan Moringen, Christian Lang

Angewandte Informatik / CoR–Lab
Universität Bielefeld

{jmoringe,clang}@techfak.uni-bielefeld.de

WS 2009/2010

Was ist MATLAB?

- MATLAB steht für “MATrix LABoratory”
- eine höhere Programmiersprache (Interpreter) mit imperativen, funktionalen und objektorientierten Elementen
- für mathematische (numerische) Berechnungen und Visualisierung von Daten
- entwickelt und vertrieben von “The Mathworks”[1]
- proprietär (spezielle “Student Version” ca. $ 89)
- Standard in universitärer und industrieller Forschung

“Für die Belange von Ingenieuren und Wissenschaftlern bietet MATLAB den größten Funktionsumfang und ist das höchst entwickelte Programm seiner Klasse.”

IEEE Spectrum magazine

[1] `www.mathworks.de`

Was ist MATLAB?

Was kann MATLAB?

- viele grundlegende und höhere mathematische Funktionen, Transformationen, usw.
- viele Erweiterungen in Form von “Toolboxes” für spezielle Aufgabengebiete

Wofür kann MATLAB sinnvoll genutzt werden?

- Technische Berechnungen aller Art
- Modellierung und Simulation
- Rapid Prototyping, Test von Algorithmen
- Analyse und Explorieren von Daten, Visualisierung
- Bildverarbeitung, Mustererkennung, ...

Was ist MATLAB?

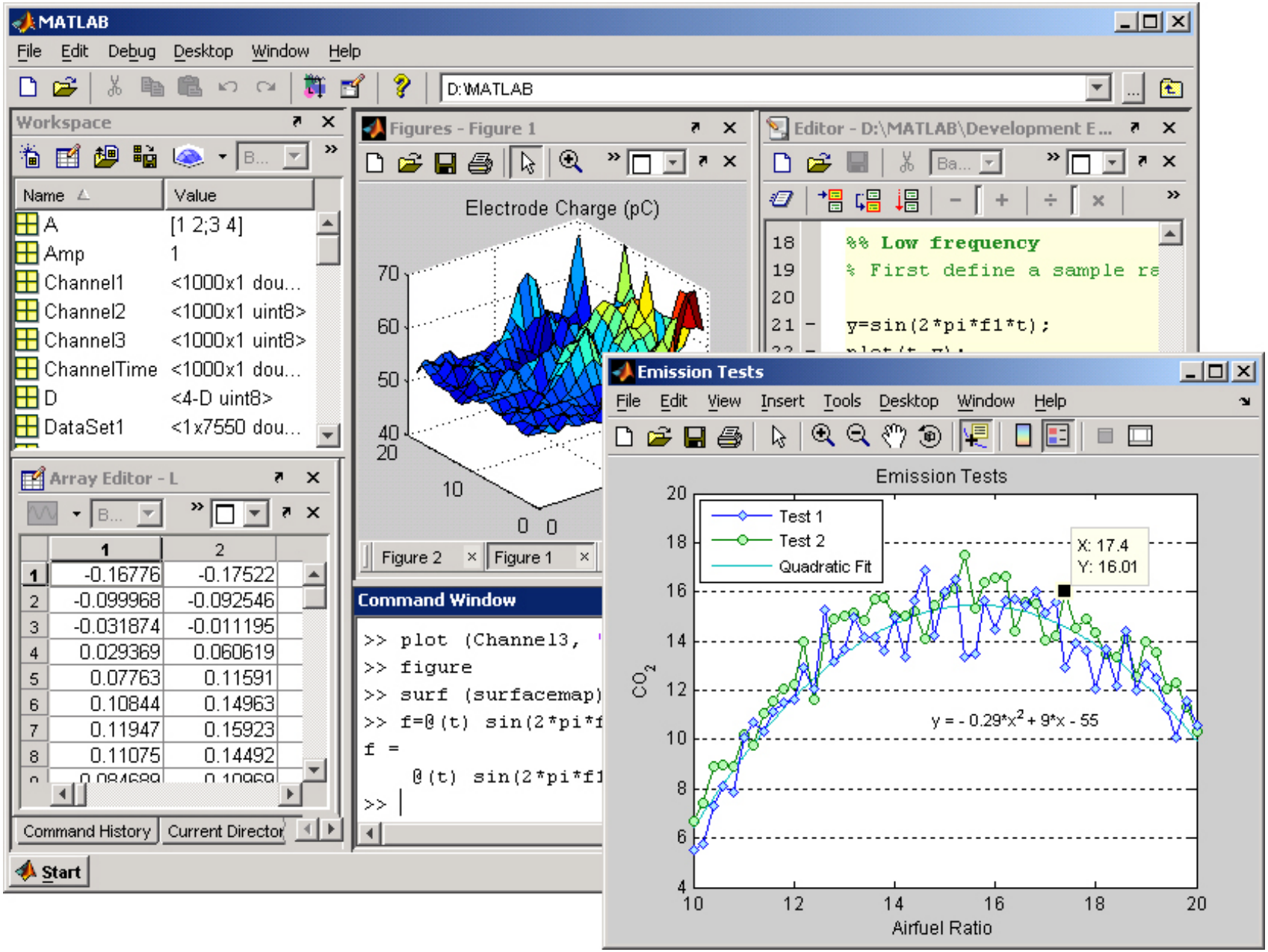
Wann ist MATLAB nicht so gut geeignet?

- Laufzeit- und Speichereffizienz sind sehr wichtig
- tief verschachtelte Rekursion wird benötigt
- Das zu lösende Problem läßt sich nicht gut auf MATLAB–Strukturen abbilden
- Lizenzgebühren sollen vermieden werden

Alternative: “Octave”[2]

- nicht effizienter als MATLAB
- Open Source
- bietet ca. 50% der MATLAB–Funktionalität
- für unsere Zwecke aber meist ausreichend

[2] `www.octave.org`


MATLAB
File Edit Debug Desktop Window Help
D:\MATLAB
Workspace
Name Value
A [1 2;3 4]
Amp 1
Channel1 <1000x1 dou...
Channel2 <1000x1 uint8>
Channel3 <1000x1 uint8>
ChannelTime <1000x1 dou...
D <4-D uint8>
DataSet1 <1x7550 dou...
Array Editor - L
Command History
Current Director
Start
Figures - Figure 1
Electrode Charge (pC)
Figure 2
Figure 1
Command Window
>> plot (Channel3,
>> figure
>> surf (surfacemap)
>> f=@(t) sin(2*pi*f
f =
@(t) sin(2*pi*f1
>>
Editor - D:\MATLAB\Development E...
18 %% Low frequency
19 % First define a sample ra
20
21 - y=sin(2*pi*f1*t);
Emission Tests
File Edit View Insert Tools Desktop Window Help
Emission Tests
Test 1
Test 2
Quadratic Fit
X: 17.4
Y: 16.01
y = - 0.29*x^2 + 9*x - 55
CO2
Airfuel Ratio

MATLAB in der Technischen Fakultät

MATLAB starten

- normaler Start:
  `$ /vol/matlab/latest/bin/matlab`
- MATLAB kann auch ohne GUI gestartet (sinnvoll bei Ressourcen–Beschränkung, z.B. bei Remote–Login):
  `$ /vol/matlab/latest/bin/matlab -nosplash -nojvm`

Dokumentation

- unter `/vol/lehre/Musterklassifikation/WS0910/Doku`: MATLAB–Primer, Getting Started Guide, diese Einführung
- `www.mathworks.com/access/helpdesk/help/techdoc`
- in MATLAB: `help <functionsname>` oder `helpwin`

Die Matrix ...

Die grundlegende Datenstruktur in MATLAB ist die Matrix:
$\Rightarrow$ Alles in MATLAB ist eine Matrix!

Einfache Beispiele

```
a = 5          % a ist ein Skalar
a(1)           % kann aber auch als einelementiger Vektor
a(1,1)         % oder 1x1-Matrix
a(1,1,1,1)     % oder mehrdimensionales Array benutzt werden

b = [1 2 3]     % ein Zeilenvektor mit 3 Elementen
c = [1; 2; 3]   % ein Spaltenvektor mit 3 Elementen

A = [1 2 3; 4 5 6]    % eine 2x3-Matrix
B = [1 2; 3 4; 5 6]   % eine 3x2-Matrix
C = [1,2;3,4;5,6]     % auch m"oglich: "," anstatt " "
```

Datentypen

Numerische Datentypen

- Standard: Float–Typ `double` (auch `single` Typ vorhanden)
- explizites Casting nach Integer–Typen möglich (`int8`, `uint8`, `int16`, ..., `uint64`)
- Boolscher Datentyp `logical`
- Float- und Integer–Typen beinhalten auch komplexe Zahlen
- nähere Informationen mit `help datatypes` in MATLAB

Zeichenketten

- Strings sind Arrays vom Typ `char`
- siehe `help char` bzw. `help strings`

Datentypen

strukturierte Daten

- Zusammenfassung verschiedener Daten mit benannten Feldern (ähnlich zu structs in C)
- Erzeugen explizit mit `struct` oder implizit durch Einführen eines Feldnamens (z.B. `a.b = 4; a.c = 'yes';`)

Cellarrays

- Matrizen, die Elemente verschiedenen Typs beinhalten
- Erzeugen explizit mit `cell` oder implizit durch Wertzuweisung
- zwei verschiedene Zugriffsarten auf Matrixelemente:
  - `A(x,y)` ... Zugriff auf eine Zelle
  - `A{x,y}` ... Zugriff auf den *Inhalt* einer Zelle

Ein- und Ausgabe

Bildschirm und Tastatur

- Ein Semikolon nach einer Anweisung unterdrückt die Ausgabe des Ergebnisses
- Matrixeingabe kann sich über mehrere Zeilen erstrecken
- Textausgabe mit `disp` bzw. `error` für Fehlermeldungen

Dateien

- Ähnlich zu C mit Funktionen `fopen`, `fprintf`, `fscanf`, usw.
- mit `fprintf` ist auch Bildschirmausgabe möglich
- Tastatureingabe mit `input`
- Spezialfunktionen zum laden und speichern von Bildern usw. (`imread`, `imwrite`, ...)

Operatoren

- Arithmetische Operatoren
  - Matrix–Operatoren `+ - * / \ ^ '`
  - elementweise Operatoren `.* ./ .\ .^ .'`
    - `A'` ... komplex konjugiert transponiert
    - `A.'` ... elementweise transponiert (nicht konjugiert)
    - `A/B` und `A\B` ... rechts/links "Matrixdivision" → siehe Doku
  - siehe Dokumentation: "Arithmetic Operators"
- Vergleichsoperatoren
  - `< > <= >= == ~=`
  - arbeiten auf numerischen Matrizen
  - für String–Vergleiche: `strcmp`
  - siehe Dokumentation: "Relational Operators"
- Logische Operatoren
  - Matrixoperatoren: and: `&`, or: `|` und not: `~`, Funktion `xor`
  - bei Skalaren auch `&&` und `||` für "short–circuit"
  - Vorsicht, wenn ein Operand die leere Matrix `[]` ist!
  - bitweise: Funktionen `bitand`, `bitor`, `bitcmp`, `bitxor`
  - siehe Dokumentation: "Logical Operators"

Program Control Statements

if und switch

```
% if statement
if a == b
  disp('a ist gleich b')
elseif a == c
  disp('a ist ungleich b, aber gleich c')
else
  disp('a hat irgendeinen anderen Wert')
end

% switch statement
switch var
  case 0
    disp('Null')
  case 1
    disp('Eins')
  otherwise
    error('Unglueltiger Wert')
end
```

Program Control Statements

for und while

```
% for statement
for a = 1:10  % zaehlt von 1 bis 10 in Einerschritten
  for b = -20:4:20  % von -20 bis 20 in Viererschritten
    count = [1 3 5 7 11 13 17 19];
    for c = count  % fuer alle Elemente in "count"
      fprintf('a = %d  b = %d  c = %d\n',a,b,c);
    end
  end
end

% while statement
x = 10;
while x > 0
  y(x) = 2^x
  x = x-1;
end

% "continue" und "break" wie in C++
```

MATLAB Workspace und Skripte

Workspace

- alle Variablen, mit denen gearbeitet wurde
- Einführen neuer Variablen einfach durch erstmalige Wertzuweisung (keine gesonderte Deklaration)
- `who` und `whos`: Anzeige aller bekannten Variablen
- `clear`: Löschen einiger oder aller Variablen
- Laden und speichern mit `load` und `save`

Skripte

- Textdateien mit Endung `.m`, Aufruf über Dateiname
- Arbeiten im globalen Workspace, so, als ob die einzelnen Anweisungen nacheinander von Hand eingegeben wurden

Arbeitsverzeichnis und Suchpfad

Arbeitsverzeichnis

- Ähnlich zu einer Unix–Konsole gibt es in auch MATLAB ein aktuelles Arbeitsverzeichnis
- `pwd`: Zeigt das aktuelle Arbeitsverzeichnis an
- `ls`: Zeigt den Inhalt des Arbeitsverzeichnisses an
- `cd`: Arbeitsverzeichnis wechseln
- Angaben von Dateinamen (z.B. für `load` und `save`) können immer absolut oder relativ zum Arbeitsverzeichnis sein

Suchpfad

- Suchpfad zum Lokalisieren von Funktionen in `.m`–Dateien
- `path`: Zeigt den aktuellen Suchpfad an
- `addpath`: Fügt ein Verzeichnis zum Suchpfad hinzu

Funktionen

Funktionsdefinitionen

- definiert in einer `.m`–Datei mit Schlüsselwort `function`
- pro Datei nur eine Funktion plus Hilfsfunktionen
- Übereinstimmung von Funktions- und Dateinamen
- Hilfetext mittels Kommentar am Anfang der Datei

Variablen in Funktionen

- Variablen sind lokal (nicht im globalen Workspace)
- Zwei Spezialfälle:
  - `global`: Zugriff auf eine Variable aus dem Workspace
  - `persistent`: eine “statische” Variable, die ihren Wert über mehrere Funktionsaufrufe beibehält

Beispiel

```
% Hilfetext wie er mit "help sqr" angezeigt wird
function s = sqr(a)
  s = a^2;

% Hilfetext wie er mit "help restDiv" angezeigt wird
function [q,r] = restDiv(a,b)
  q = floor(a/b);
  r = mod(a,b);
```

Anzahl der Argumente

- Es können auch weniger als die angegeben Ein- und Ausgabeargumente übergeben werden
- `nargin` und `nargout`: Anzahl der Ein-/Ausgabeargumente
- Fehler bei Zugriff auf nicht–vorhandene Argumente

Arbeiten mit Matrizen

Auswahl von Untermatrizen

Mit dem : – Operator können Submatrizen ausgewählt werden

Beispiele

```
A = rand(4,5);   % 4x5-Zufallsmatrix
A(:,1)           % gesamte erste Spalte
A(3,:)           % gesamte dritte Zeile
A(3,1:5)         % auch die gesamte dritte Zeile,
                 % mit expliziter Bereichsangabe
A(3,1:end)       % wieder die gesamte dritte Zeile,
                 % "end" steht fuer den hoechsten Index
A(:,2:4)         % die zweite bis vierte Spalte
A(3:end,4:end)   % 2x2-Submatrix rechts unten
A(1:end-1,:)     % alles ausser die letzte Zeile
A(:,1:2:end)     % nur jede zweite Spalte
A([1 4 9])       % explizite Angabe bestimmter Elemente
```

Arbeiten mit Matrizen

Geschwindigkeit

Matrixoperationen sind viel schneller als handgeschriebene Schleifen und sollten — wo immer möglich — bevorzugt verwendet werden!

Beispiel

Es soll die Zeile `k` der `mxn`–Matrix `A` auf Null gesetzt werden:

```
% Schleife: langsam
for i=1:n
  A(k,i) = 0;
end

% Matrixoperation: schnell
A(k,:) = 0;
```

Arbeiten mit Matrizen

implizite Vergrößerung

- Matrizen werden bei Zugriff auf ihre Elemente wenn möglich automatisch vergrößert, falls ein Index eigentlich außerhalb der Matrix liegt
- stets “rechteckige” Struktur, neue Elemente werden mit Null, Leerzeichen oder der leeren Matrix initialisiert

Beispiel

```
A(2,3) = 5;   % Erzeugt eine 2x3-Matrix mit einer 5 an der
              % angegebenen Stelle und Nullen sonst
A(4,2) = 2;   % A ist jetzt eine 4x3-Matrix
b = rand(1,5);  % b ist ein 5-dimensionaler Zeilenvektor
A(1,:) = b;   % Fehler: Dimensionen passen nicht (4 <-> 5)
A(1,1:5) = b;  % Jetzt geht es: A wird um eine fuenfte
               % Spalte erweitert, dann wird b zugewiesen
```

Arbeiten mit Matrizen

Nützliche Funktionen:

- `size`: Liefert die Größe einer Matrix
- `find`: Elemente mit bestimmten Eigenschaften finden
- `ind2sub` und `sub2ind`: Konvertieren zwischen den üblichen mehrdimensionalen Matrix–Indizes und linearen Indizes (wie sie z.B. `find` verwendet)
- `reshape`: Form einer Matrix bzw. eines Vektors verändern
- `repmat`: Eine große Matrix aus vielen Kopien einer kleinen Matrix zusammensetzen
- `sum`: Summe der Elemente einer Matrix
- `eye`, `zeros`, `ones` und `rand`: Einheits-, Null-, Einer- und Zufallsmatrix erzeugen
- `sparse`: dünnbesetzte Matrizen (speichereffizient)
- `A'` bzw. `A.'`: Matrix `A` transponieren
- `char`, `strcat`, `strvcat`, `strcmp`, `strfind`, usw. für Strings

Cellarrays

Matrizen sind immer “rechteckig”!

- das kann zu ungewünschten Effekten führen
- Beispiel: Array von Strings
- jedes Matrixelement ist ein Buchstabe
  $\longrightarrow$ alle Strings müssen gleich lang sein!

Array von Strings

```
>> A = strvcat('Hallo','Welt')
A =      % Die Matrix A enthaelt zwei Strings:
Hallo    % 'Hallo' in der ersten Zeile
Welt     % 'Welt ' (mit Leerzeichen!) in der zweiten Zeile

>> size(A)
ans =
     2     5  % A ist also eine 2x5-Matrix
```

Cellarrays

“Hierarchische” Matrizen

- Jedes Element eines Cellarrays (Zelle) ist eine Art “Zeiger” auf eine beliebige Datenstruktur
- Zugriff auf die Zelle (den “Zeiger”) wie gehabt mit `(...)`
- Zugriff auf den Inhalt mit `{...}`

Cellarray von Strings

```
>> A = {'Hallo';'Welt'}
A =     % Die Matrix A enthaelt zwei Zellen:
    'Hallo'   % Zugriff auf den Inhalt mit "A{1}"
    'Welt'    % Zugriff auf den Inhalt mir "A{2}"

>> size(A)
ans =
     2     1   % Jetzt ist A also eine 2x1-Matrix
```

Noch ein Beispiel

Das geht natürlich auch mit beliebigen anderen Daten:

```
>> A{1,1} = 'daten.txt';
>> A{1,2} = 1;
>> A{2,1} = rand(1,5);
>> A{2,2} = rand(10,4);
>> A   % A ist eine 2x2-Matrix
A =
    'daten.txt'      [           1]
    [1x5 double]     [10x4 double]
>> A(2,1)  % Die Zelle A(2,1) enthaelt eine 1x5-Matrix
ans =
    [1x5 double]
>> A{2,1}  % A{2,1} ist diese 1x5-Matrix (Zell-Inhalt)
ans =
    0.8147    0.9058    0.1270    0.9134    0.6324
```

Debugging

Debugging von Matlab–Skripten und -Funktionen

- Haltepunkt–basiert, ähnlich wie gdb
- `dbstop at <lineno> in <m-file>`: Haltepunkt setzen
- `dbcont`: Weiter bis zum nächsten Haltepunkt
- `sbstep`: Eine weitere Zeile ausführen
- `dbstack`: Aufruf–Stack anzeigen
- `dbstatus`: Alle Haltepunkte anzeigen
- `dbclear all`: Alle Haltepunkte löschen
- `dbclear at <lineno> in <m-file>`: einen bestimmten Haltepunkt löschen
- während des Debuggens kann auf lokale Funktionsvariablen zugegriffen werden

Visualisierung

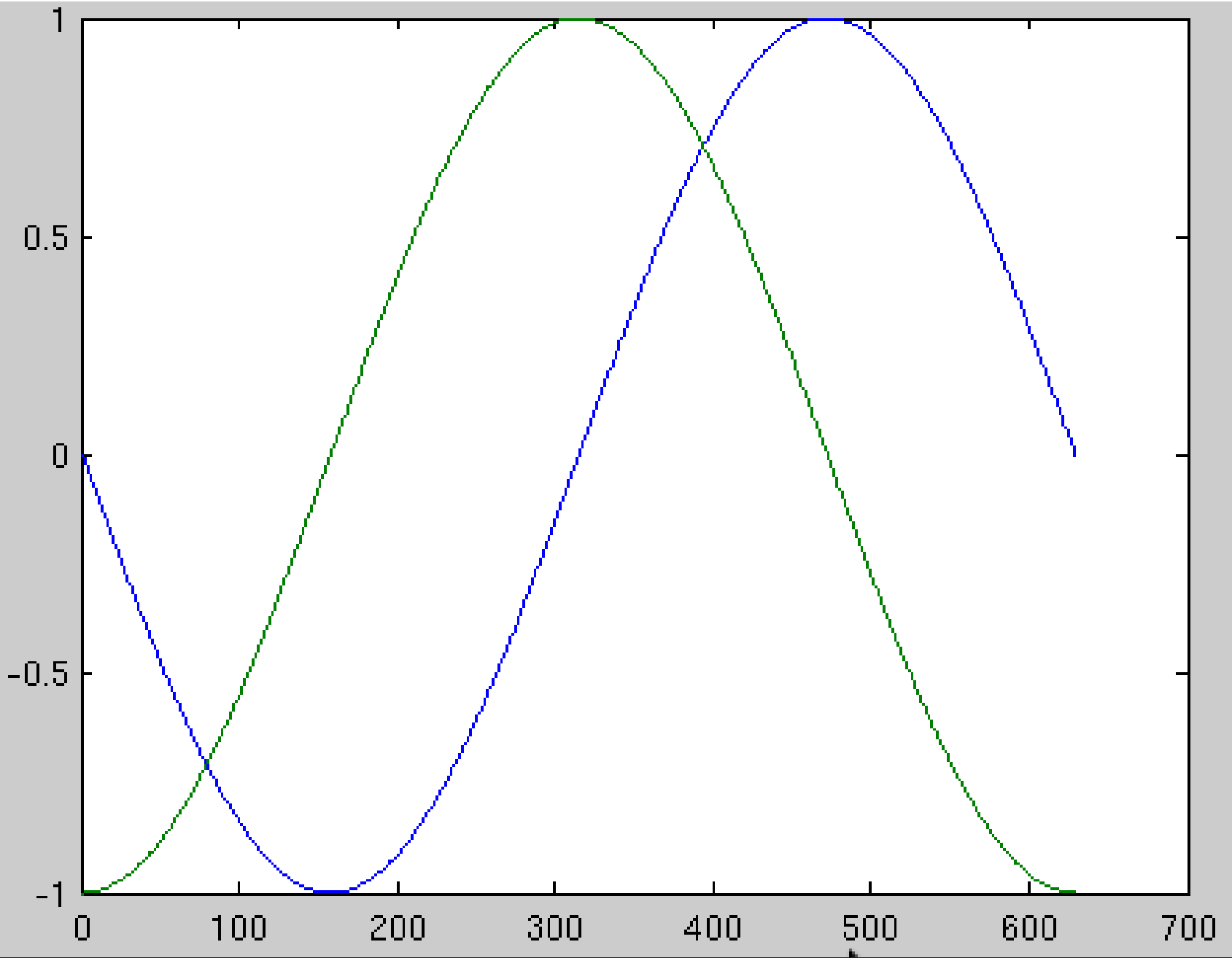
Datenvisualisierung in MATLAB

- `plot`: Plotten von 2D–Daten
- `plot3`: Plotten von 3D–Daten
- `figure`: neues Ausgabefenster bzw. Auswahl des Ausgabefenster für nachfolgende Plot–Anweisungen
- `subplot`: mehrere Plots im gleichen Fenster
- `hold on/off`: vorherige Ausgabe überschreiben
- noch viel mehr Möglichkeiten → umfangreiche Doku

Beispiel

```
range = -pi:0.01:pi;   % Wertebereich festlegen
x(:,1) = sin(range);   % Funktionswerte berechnen
x(:,2) = cos(range);
plot(x);               % Die Daten anzeigen
```

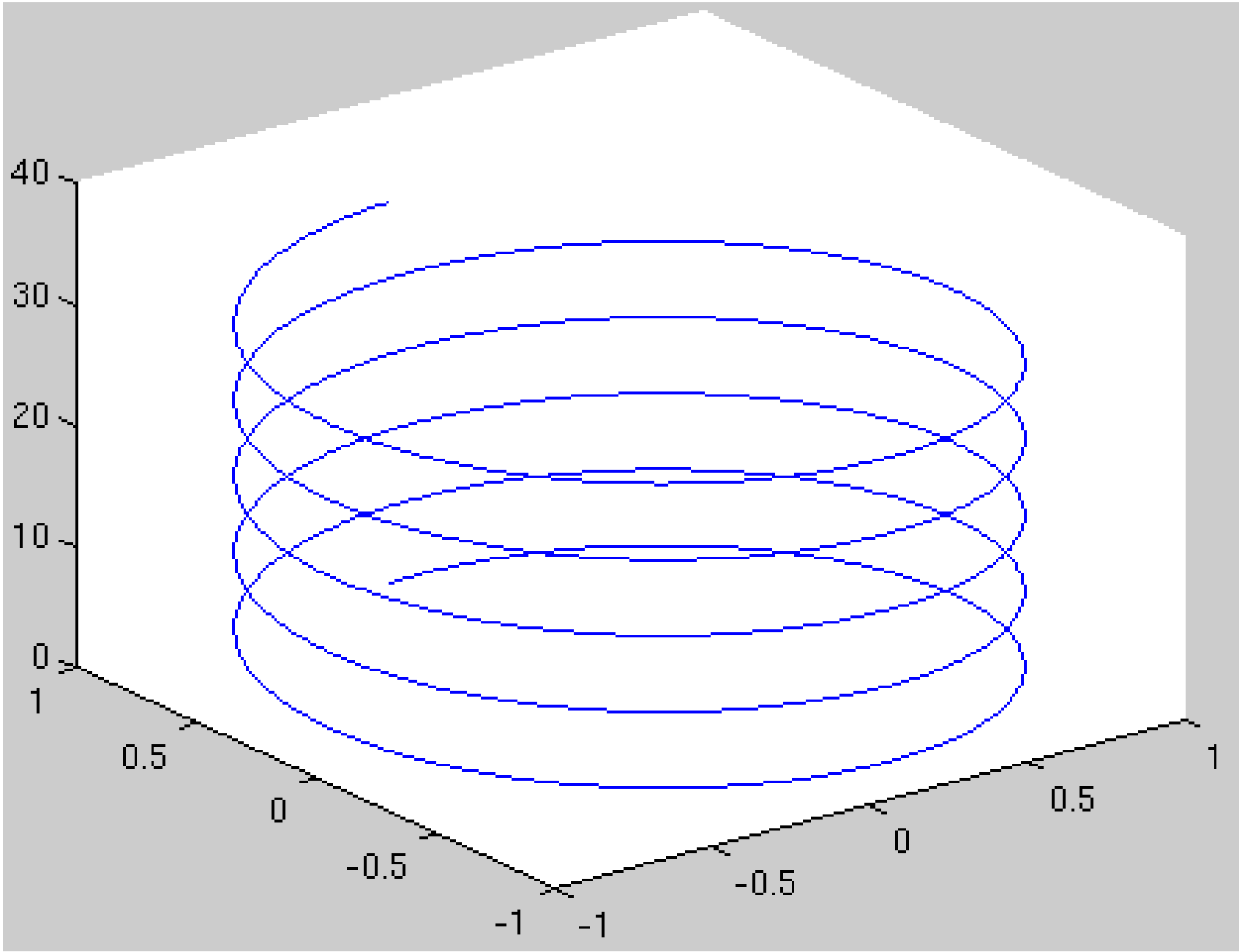
2D–Plot



Noch ein Beispiel

```
t = 0:0.01:10*pi;
plot3(sin(t),cos(t),t);
```

3D–Plot



Viel Erfolg und Spaß
bei der Arbeit!