UNIVERSITÄT
DES
SAARLANDES

MATLAB
für Naturwissenschaftler

7. Datenein- und -Ausgabe

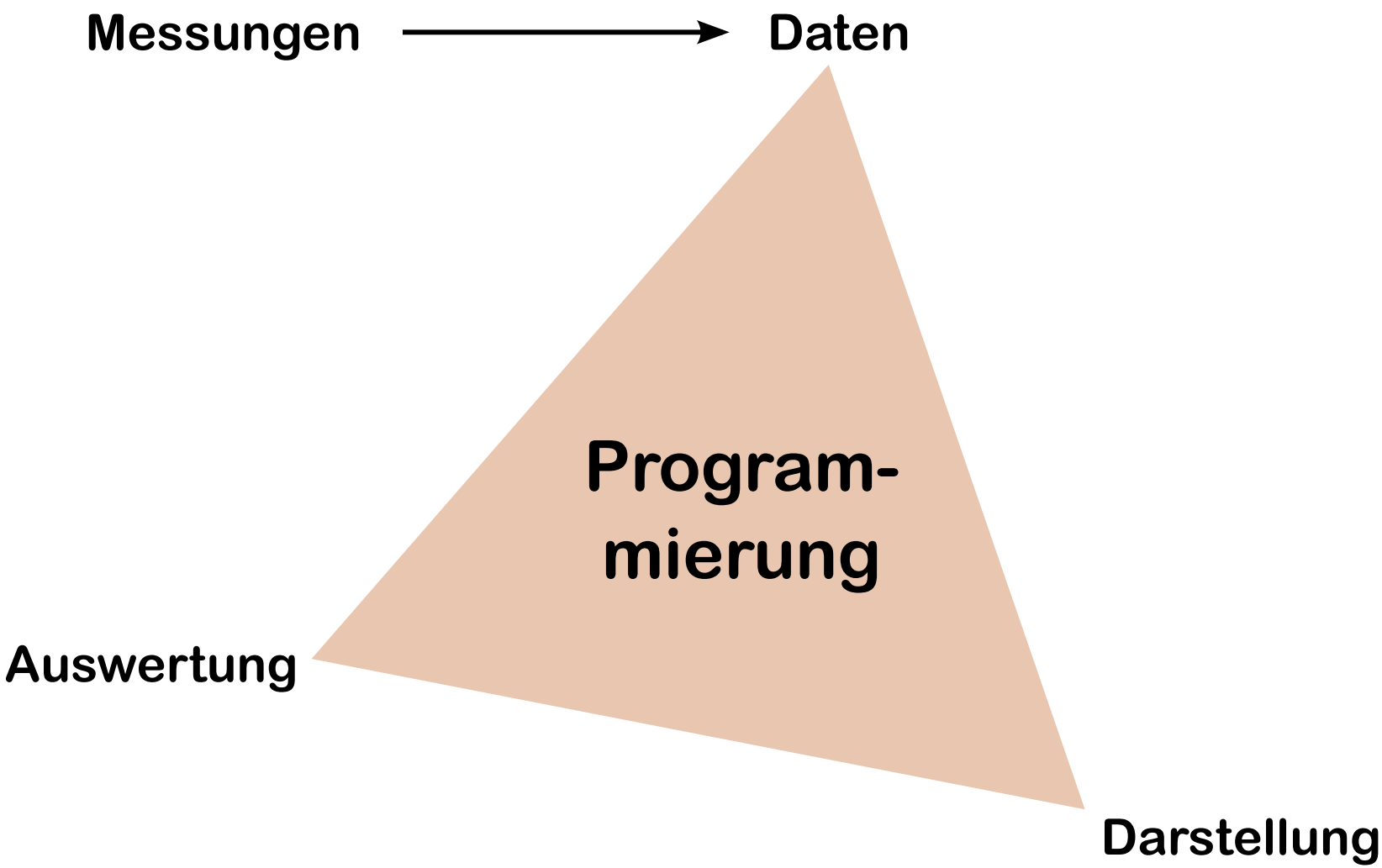
Till Biskup

Lehrstuhl für Physikalische Chemie und Didaktik
Universität des Saarlandes

Übersicht

Motivation

Wir haben Daten gemessen



Warum sich mit Datenein- und -Ausgabe beschäftigen?

- Wir haben Daten gemessen...
  - ...und wollen diese Daten auswerten.
  - Idealerweise liegen die Daten bereits elektronisch vor.
  - Oft müssen spezielle Formate importiert werden.
- Wir haben Daten ausgewertet...
  - ...und wollen die Ergebnisse der Auswertung speichern.
  - ...und wollen die Parameter der Auswertung speichern.

☛ Daten liegen i.d.R. in Formaten vor,
die nicht direkt von MATLAB verarbeitet werden können.

☛ Auswertungen sollten nicht im MATLAB-internen Format gespeichert werden (Unabhängigkeit).

Daten importieren

Voraussetzung für alles Weitere

Listing 1: ASCII-Export eines LS45-Fluoreszenzspektrometers

```
PE FL                    SPECTRUM    ASCII       PEDS        1.60
  -1
1A-01.SP
13/05/02
13:55:40.00
13/05/02
13:56:11.00


400.000000
1
#DATA
270.000000  7.640000
270.500000  7.712395
271.000000  7.775626
271.500000  7.830389
272.000000  7.877049
272.500000  7.925246
273.000000  7.979488
273.500000  8.035135
274.000000  8.080773
274.500000  8.139876
275.000000  8.216758
275.500000  8.314536
```

Daten importieren

- Messdaten liegen meist in elektronischer Form vor
  - Oft proprietäre Formate
  - Meist Möglichkeit des Exports in ein „Standardformat“ (notfalls reiner Text)
- Datenimport in MATLAB
  - MATLAB unterstützt eine Reihe an Formaten.
  - Möglichkeiten des Imports: GUI/Kommandozeile
- Routine zum Einlesen von Messdaten
  - Routinevorgang: Daten werden häufig eingelesen
  - Eigene Routine für speziellen Datentyp oft hilfreich

Von MATLAB unterstützte Formate

- MATLAB unterstützt nativ eine Reihe von Formaten.
  - (einfache) Textdateien
  - Standardformate (XML, CDF, HDF, netCDF, ...)
- Selbstgeschriebene Importroutinen für weitere Formate
  - Viele Geräte speichern in proprietären Formaten.
  - MATLAB: „low-level“-Funktionen zum Rohimport

Hinweis zu Textdateien (ASCII)

- MATLAB akzeptiert nur den Punkt als Dezimaltrenner.
  - Selbstgeschriebene Importroutine für Dateien mit Komma

Von MATLAB unterstützte Formate

Format	Funktion
MATLAB	`load`
Text (ASCII)	`load`, `dlmread`, `csvread`
Spreadsheet	`xlsread`
XML	`xmlread`
CDF, HDF, netCDF, ...	diverse
Bilder	`imread`
Audio	`audioread`
Video	`VideoReader`

- ☛ Details in der MATLAB-Dokumentation

Möglichkeiten des Datenimports

- Grundsätzlich zwei Herangehensweisen
  1. Grafisch über die MATLAB-GUI
  2. Über die Kommandozeile (Befehlsaufruf)
- Grafisch über die MATLAB-GUI
  - „Intuitiv“
  - Gut geeignet für Einzelfälle
  - Nicht automatisierbar
- Über die Kommandozeile (Befehlsaufruf)
  - Für Routineaufgaben
  - Bei Verwendung in Skripten/Funktionen
  - Automatisierbar

Konkretes Beispiel: Import von Textdateien

- Ausgangspunkt
  - Textdateien mit Messdaten (z.B. ASCII-Export)
  - Punkt als Dezimaltrenner
- Zwei Funktionen
  1. `load`
  2. `importdata`

Wiederholung: Dezimaltrennzeichen

- MATLAB akzeptiert nur den Punkt als Dezimaltrenner.
  - Selbstgeschriebene Importroutine für Dateien mit Komma
  - Zeilenweises Einlesen
  - Ersetzen des Punktes durch Komma

Import von Textdateien (ASCII): load

- Voraussetzungen
  - Datei enthält *ausschließlich* Zahlenwerte
  - Jede Reihe muss die identische Anzahl Spalten aufweisen

Listing 2: Beispiel einer mit load einlesbaren Textdatei

```
1 270.000000  7.640000
2 270.500000  7.712395
3 271.000000  7.775626
4 271.500000  7.830389
5 272.000000  7.877049
```

Listing 3: Aufruf der load-Funktion

```
data = load('filename');
```

Import von Textdateien (ASCII): `importdata`

- Einsatzgebiet
  - Textdateien mit Kopfzeilen
- Voraussetzung
  - Länge des Dateikopfes ist bekannt
  - Daten: Identische Spaltenzahl für jede Reihe
  - Daten: Punkt als Dezimaltrennzeichen
- Parameter der Funktion
  - Dateiname
  - *Optional*: Trennzeichen für die einzelnen Datenspalten
  - *Optional*: Zahl der Kopfzeilen der Datei

Import von Textdateien (ASCII): importdata

Listing 4: Beispiel einer mit `importdata` lesbaren Textdatei

```
1  PE FL                    SPECTRUM    ASCII       PEDS        1.60
2    -1
3  1A-01.SP
4  13/05/02
5  13:55:40.00
6  13/05/02
7  13:56:11.00
8
9
10 400.000000
11 1
12 #DATA
13 270.000000  7.640000
14 270.500000  7.712395
15 271.000000  7.775626
16 271.500000  7.830389
17 272.000000  7.877049
18 272.500000  7.925246
19 273.000000  7.979488
20 273.500000  8.035135
```

Import von Textdateien (ASCII): importdata

Listing 5: Aufruf der importdata-Funktion

```
1 % Aufruf ohne optionale Parameter
2 data = importdata('filename');
3
4 % Aufruf mit Spaltentrennzeichen (hier: Tabulator)
5 data = importdata('filename','\t');
6
7 % Aufruf mit Spaltentrennzeichen und Anzahl Kopfzeilen
8 data = importdata('filename','\t',13);
```

- Wichtige Hinweise
  - Zahl der Kopfzeilen kann nur *gemeinsam* mit dem Spaltentrennzeichen angegeben werden.
  - Rückgabeparameter (`data`) ist ein `struct`
    Felder: `data`, `textdata`, `colheaders`

„Low-level“-Routinen

- Einsatzgebiete
  - Textdateien mit Komma als Dezimaltrennzeichen
  - Dateien, die nicht mit anderen MATLAB-Routinen lesbar sind
- Was bedeutet „low level“?
  - Direkte Operation auf Dateisystemebene
  - Einzelne Routinen für Öffnen, Lesen, Schließen
  - Große Verantwortung des Nutzers
- Routinen
  - `fopen`/`fclose` – Datei öffnen/schließen
  - `fgets`/`fgetl` – Textdatei zeilenweise lesen
  - `fread` – Binärdatei (byteweise) lesen

„Low-level“-Routinen: Ein einfaches Beispiel

Listing 6: Einlesen einer Textdatei über „Low-level“-Routinen

```
% Open file
fid = fopen('textfile.txt');

% Initialise loop variable
k = 0;

% Read content of file line by line
while ~feof(fid)
    % Increment loop variable
    k = k+1;
    % Read line of the textfile and store it in cell array
    fileContents{k} = fgetl(fid);
end

% Close file
fclose(fid);
```

„Low-level“-Routinen: Verantwortung des Nutzers

- Geöffnete Dateien immer auch schließen
  - Die Zahl der gleichzeitig geöffneten Dateien in einem Betriebssystem ist begrenzt.
  - Andere Programme können auf geöffnete Dateien nicht oder nur partiell zugreifen.
- Robusten Code schreiben
  - Mögliche Fehler abfangen (`try...catch`).
  - Dafür sorgen, dass in *jedem* Fall der Befehl `fclose` ausgeführt wird.

☛ Im Zweifel hilft nur noch ein Rechnerneustart...

Ein Plädoyer für offene Datenformate

- Daten sind die Währung (experimenteller) Wissenschaft
  - Rohdaten immer aufbewahren
  - Auswertungen dokumentieren
  - Daten in zukunftssicheren Formaten abspeichern
- Kriterien für ein geeignetes Datenformat
  - quelloffen und lizenzfrei
  - ausreichend dokumentiert
  - verlustfrei (auch bzgl. numerischer Genauigkeit)
  - portabel
  - maschinenlesbar und parsbar

☛ Auswahl bzw. Entwicklung geeigneter Datenformate ist kein trivialer, aber wichtiger Aspekt von Wissenschaft.

Von MATLAB unterstützte Formate

- MATLAB unterstützt nativ eine Reihe von Formaten.
  - (einfache) Textdateien – nur Zahlenwerte
  - Standardformate (XML, CDF, HDF, netCDF, ...)
- Selbstgeschriebene Exportroutinen für weitere Formate
  - z.B.: Textdateien mit Text (und Zahlen)
  - MATLAB: „low-level“-Funktionen zum Rohexport

Hinweis zu Textdateien (ASCII)

- MATLAB kann nur Zahlenwerte als Textdateien exportieren
  - Selbstgeschriebene Exportroutine für andere Inhalte

Von MATLAB unterstützte Formate

Format	Funktion
MATLAB	`save`
Text (ASCII)	`csvwrite`, `dlmwrite`
Spreadsheet	`xlswrite`
XML	`xmlwrite`
CDF, HDF, netCDF, ...	diverse
Bilder	`imwrite`
Audio	`audiowrite`
Video	`VideoWriter`

- ☛ Details in der MATLAB-Dokumentation

„Low-level“-Routinen

- Einsatzgebiete
  - Formate, die nicht nativ von MATLAB unterstützt werden
  - Bsp.: Textdateien, die keine/nicht nur Daten enthalten
- Was bedeutet „low level“?
  - Direkte Operation auf Dateisystemebene
  - Einzelne Routinen für Öffnen, Schreiben, Schließen
  - Große Verantwortung des Nutzers
- Routinen
  - `fopen`/`fclose` – Datei öffnen/schließen
  - `fprintf` – Text (zeilenweise) schreiben
  - `fwrite` – Binärdaten schreiben

„Low-level“-Routinen: Ein einfaches Beispiel

Listing 7: Schreiben einer Textdatei über „Low-level“-Routinen

```
1  % Cell array with contents
2  toFile = {...};
3
4  % Open file for writing
5  fid = fopen('textfile.txt','w+');
6
7  % Write content of cell array "contents" line by line
8  for line = 1:length(toFile)
9      % Write line to the textfile
10     fprintf(fid,'%s\n',toFile{line});
11 end
12
13 % Close file
14 fclose(fid);
```

☛ Das ist kein robuster Code!

„Low-level“-Routinen: Verantwortung des Nutzers

- Geöffnete Dateien immer auch schließen
  - Die Zahl der gleichzeitig geöffneten Dateien in einem Betriebssystem ist begrenzt.
  - Andere Programme können auf geöffnete Dateien nicht oder nur partiell zugreifen.
- Robusten Code schreiben
  - Mögliche Fehler abfangen (`try...catch`).
  - Dafür sorgen, dass in *jedem* Fall der Befehl `fclose` ausgeführt wird.

☛ Im Zweifel hilft nur noch ein Rechnerneustart...

Daten exportieren

„Low-level“-Routinen

Plädoyer: Wozu eine Routine zum Textdateien schreiben?

- Dokumentation der Verarbeitungsschritte
  - Bei komplexeren (mehrschrittigen) Auswertungen
  - Parameter (z.B. für Regressionen) in Datei schreiben
  - Alles, was das Programm „kennt“, kann es mitschreiben.
- ☛ Viel weniger fehleranfällig als händisches Notieren...
- ☛ Idealerweise maschinenlesbares Format

- Zusätzliche Informationen zu den Daten
  - Im Dateikopf, ggf. als Kommentar
  - U.a. Spaltenbezeichnungen bei mehrspaltigen Daten
- ☛ Idealerweise MATLAB-kompatibles Format

Parsen von Textdateien

Dem Computer zu einem Verständnis der Inhalte verhelfen

Die Ausgangslage

- Parameter/Informationen oft elektronisch vorhanden
  - In separater Textdatei
  - Am Anfang von Datendateien
- Automatisiertes Auslesen ist Trumpf
  - Werden die Parameter vom Gerät selbst geschrieben, sind sie meist weniger fehleranfällig als das Laborbuch...
- Parameter mitunter essentiell für korrekten Import

Was man gerne hätte...

- Automatisches Einlesen der Parameter
- „Verständnis“: Datenverarbeitung parameterabhängig

Parsen von Textdateien

Voraussetzungen und Vorgehen für Parameterdateien

Voraussetzungen

- Strukturierte Ablage der Parameter (Muster)
- Eineindeutigkeit (idealerweise)

Vorgehen für Parameterdateien/-Blöcke

- Zeilenweises Einlesen
- Aufteilung des eingelesenen Strings
  - Parametername
  - Wert
- Ablage in (hierarchischer) Struktur erlaubt Zugriff
  - „Verständnis“ durch das Programm

Beispiel für eine (einfache) Parameterdatei

Listing 8: Ausschnitt aus einer Parameterdatei für Bruker-EPR-Daten

```
1  DOS  Format
2  ANZ 1024
3  MIN -162.095703
4  MAX 194.904297
5  JSS 0
6  GST 3325.000000
7  GSI 150.000000
8  JUN G
9  JON weber
10 JDA 28.Jul.2014
11 JTM 09:36
12 JRE c:\winepr\tpu\superhq_2014_03_24.cal
13 JEX field-sweep
14 JNS 5
15 JSD 3
```

☛ Sofort Schlüssel-Wert-Paare erkennbar

Bausteine für einen Parser

Listing 9: Ausschnitt aus einer Routine zum Parsen der Parameterdatei

```
% Read text file contents, returning cell array
fileContent = readTextfile('<filename>');

% Assign empty structure for key-value pairs
parameters = struct();

% Parse contents line by line
for line = 1:length(fileContent)
    % Split line at first whitespace character
    [key,value] = strtok(fileContent{line});

    % Add "key" as field to structure "parameters"
    % with "value" as content of this field
    % "strtrim" gets rid of leading/trailing whitespace
    parameters.(strtrim(key)) = strtrim(value);
end
```

☛ Die Realität ist etwas komplizierter...

Allgemeines Vorgehen

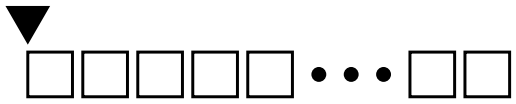
- Lineare Abarbeitung des Textes
  - Trennzeichen müssen bekannt sein (Leerzeichen etc.)
  - Aktuelle Ebene wird in Variable abgelegt.
  - Erlaubt Bearbeitung sehr großer Dateien
    (nicht durch den vorhandenen Arbeitsspeicher limitiert)
- Erster Schritt beim Übersetzen von Quellcode

☛ Parsen ist allgemein viel mächtiger,
aber dann auch anspruchsvoller zu programmieren.

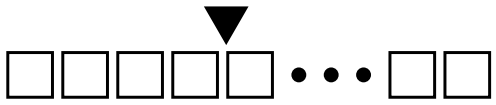
Arbeit mit Binärdateien

Mächtig, aber nichts für den Anfang

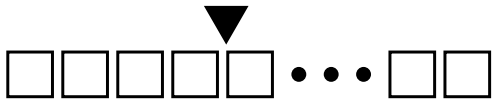
fid = fopen('...')



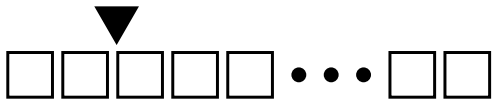
data = fread(fid,4)



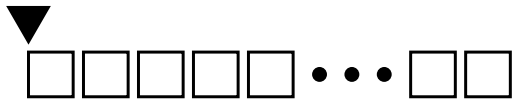
4 = ftell(fid)



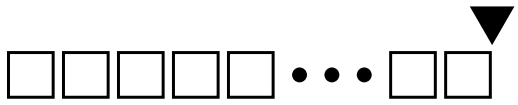
fseek(fid,-2)



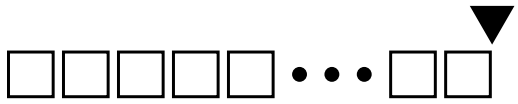
frewind(fid)



data = fread(fid,'inf')



true = feof(fid)



▼ position in file

□ one byte (= 8 bit)

Arbeit mit Binärdateien

Mächtig, aber nichts für den Anfang

- Alle Daten sind (heutzutage) letztlich binär abgelegt.
  - Grundsätzlich Zugriff auf Byte-Ebene möglich
  - Erlaubt maximale Freiheit in der Art der Codierung.
  - Binärcode wesentlich kompakter als Text
  - Datentypen direkt abspeicherbar
- Grundsätzliches Vorgehen
  - Datei öffnen (`fopen`)
  - Datenstrom lesen/schreiben (`fread`/`fwrite`)
  - Datei schließen (`fclose`)
- Funktionen für das Bewegen im Datenstrom
  - Vorwärts springen (`fseek`)
  - Zurück zum Dateianfang (`frewind`)
  - Position in Datei (`ftell`)
  - Test auf Dateiende (`feof`)

Arbeit mit Binärdateien

Mächtig, aber nichts für den Anfang

Schwierigkeit mit Binärdateien

- Einlesen nur möglich bei Kenntnis des Aufbaus
  - Komplexere Datentypen sind größer als ein Byte.
  - Kenntnis des Aufbaus erlaubt blockweises Einlesen und ggf. Konvertieren in entsprechende Datentypen.

Ausnahme: Reine Datendateien

- Ausschließlich numerische Binärdaten
  - Standardformat: IEEE
  - Wird von MATLAB (und den meisten Sprachen) unterstützt
- Gute Wahl für numerische Daten
  - Plattformunabhängig, universell unterstützt, kompakt
  - Nachteil: Parameter müssen anderweitig abgelegt werden

Ausblick

Wie geht's weiter?

...Zeit für eigene praktische Arbeit...

Vorschau: Grafiken

- Formatierung von Abbildungen
- Plot-Befehle in MATLAB
- Abbildungen exportieren