UNIVERSITÄT
DES
SAARLANDES

Lehrstuhl für Physikalische Chemie und Didaktik

Kurs „MATLAB für Naturwissenschaftler“
im Sommersemester 2019

Till Biskup, Matthias Marquardt, Johann Seibert, Christopher Kay

— Aufgabenblatt 5 vom 30.04.2019 —

Vorbemerkung

Alle für die Bearbeitung der folgenden Aufgaben notwendigen Dateien werden Ihnen auf der zum Kurs gehörigen Internetseite bereitgestellt:

`https://www.till-biskup.de/de/lehre/matlab/ss2019/`

Sie finden die Dateien beim Material zur Lektion 7. Alle Dateien befinden sich in einem einzigen ZIP-Archiv `daten.zip`, das Sie herunterladen und entpacken müssen. Am Besten entpacken Sie sich all diese Daten in ein eigenes Verzeichnis und arbeiten dann auch in diesem Verzeichnis mit MATLAB.

Aufgabe 5—1 (Einlesen von Daten aus einfachen Textdateien)

Die mit Abstand einfachste Variante des Datenimportes ist die MATLAB-Routine `load`. Auch wenn sie bei Textdateien auf solche Dateien beschränkt ist, die ausschließlich Zahlenkolonnen enthalten und eine identische Anzahl von Spalten für jede Zeile aufweisen, gibt es ausreichend viele Geräte, die diese primitivste Form des Datenexports unterstützen.

Hinweis: Das vielleicht größte Problem beim Export von Daten in dieser Form ist das vollständige Fehlen jeglicher weiterer Information über die Daten.

Versuchen Sie, mit dem Befehl `load` die Datei `simple.txt` zu laden und stellen Sie anschließend das Ergebnis dar. Beachten Sie dafür, welche Dimension der Rückgabewert von `load` hat und nutzen Sie geschickt Ihr Wissen (`:`-Operator), um die zweite gegen die erste Spalte zu plotten.

Haben Sie aufgrund der Form der Daten und der Werte der *x*-Achse eine Idee, um was für eine Art von (spektroskopischen) Daten es sich handeln könnte? (Anmerkung: Das sind reale Daten...)

Aufgabe 5—2 (Einlesen von Daten mit Kopfzeilen)

Besitzen die Textdateien mit den Daten zusätzliche Kopfzeilen, Sie sind aber (zumindest für's Erste) lediglich an den Daten interessiert, dann können Sie – vorausgesetzt, die Zahl der Kopfzeilen ist immer konstant – mit der MATLAB-Routine `importdata` sehr weit kommen.

Versuchen Sie, mit dem Befehl `importdata` die Datei `daten-mit-header.txt` zu laden und stellen Sie anschließend das Ergebnis dar. Hier ist erst einmal wichtig herauszufinden, wie viele Kopfzeilen die Datei enthält, und welches Zeichen Sie als Trenner (*delimiter*) eingeben müssen. Spielen Sie hier ein wenig mit den Werten, bis Sie eine vernünftige Rückgabe bekommen.

Aus dem Dateikopf können Sie weitere Informationen entnehmen, um welche Art Daten es sich handelt. Entsprechend können Sie weiterhin auch korrekte Achsenbeschriftungen vornehmen.

Aufgabe 5—3 (Einlesen von Daten mit Komma als Dezimaltrenner)

Einige Standard-UV/vis-Spektrometer des Herstellers Shimadzu beherrschen zwar den Export als ASCII-Textdatei, verwenden allerdings statt des Punktes als Dezimaltrenner das Komma.[1] Diese Dateien können Sie mit keiner der bisher genannten MATLAB-Routinen (`load`, `importdata`) einlesen. Hier sind Sie stattdessen vollständig auf das Schreiben einer eigenen Funktion und den Rückgriff auf „Low-level"-Routinen angewiesen.

Schauen Sie sich die Datei `uvvis.txt` mit einem Texteditor, z.B. dem MATLAB-Editor, an und überlegen Sie, wie Sie diese Datei importieren können. Sie sind hier auf die entsprechenden „Low-level"-Routinen `fopen`, `fclose` und `fgetl` bzw. `fgets` angewiesen.

Schreiben Sie eine *robuste* Funktion zum Einlesen von UV/vis-Daten, die sich an der in Listing 1 definierten Schnittstelle orientiert. Die Ausgabe, `data`, soll hier eine Matrix mit zwei Spalten (Wellenlänge und Intensität) sein, der Eingabeparameter, `filename`, ist eine Zeichenkette (*string*).

Listing 1: Schnittstelle einer Funktion zum Einlesen vom UV/vis-Daten

```
data = loadUVVisData(filename)
```

Achten Sie auf eine grundlegend robuste Programmierung, indem Sie `try-catch`-Konstrukte verwenden. Das ist essentiell, um bei Fehlern in der Verarbeitung durch nicht geschlossene Dateien nicht in größere Schwierigkeiten mit Ihrem Datei- und Betriebssystem zu kommen, die sich im Zweifel nur durch einen Neustart des gesamten Systems beheben lassen.

Gehen Sie in zwei Schritten vor: Zunächst importieren Sie den Inhalt der gesamten Datei (am Besten in ein `cell array`) und schließen Sie nachfolgend die Datei wieder. Diesen gesamten Block sollten Sie in ein `try-catch`-Konstrukt verpacken, da er der wirklich kritische Bereich ist. Anschließend können Sie dann sowohl die Ersetzung von Komma durch Punkt als auch die Umwandlung des Textes in Zahlen durchführen.

Hinweis: Das Einlesen dieser Daten ist Ihnen bereits beim vorherigen Aufgabenblatt begegnet. Allerdings ist der dortige Code insofern eher ungeeignet für ernsthafte Anwendungen, als dass er auf jegliches Abfangen von Problemen beim Lesen der Datei verzichtet. Trotzdem können Sie dort sehen, wie man die Konversion der Daten (Ersetzen von Komma durch Punkt und anschließendes Konvertieren in Zahlen) durchführen könnte.

Aufgabe 5—4 (Lesen und Schreiben von Textdateien)

Eine immer wiederkehrende Aufgabe, für die MATLAB keine wirklich einfache Funktion bereitstellt, ist das schlichte Einlesen von Textdateien in ein `cell array` und das Schreiben beliebigen Textes (meist in einem `cell array` vorliegend) in eine Textdatei.

Beheben Sie dieses Manko von MATLAB, in dem Sie Sie zwei Funktionen zum Lesen und Schreiben von Textdateien, `readTextFile` und `writeTextFile`, erstellen, die jeweils den nachfolgend definierten Schnittstellen gehorchen:

Listing 2: Schnittstelle einer Funktion zum Einlesen vom Textdateien

```
fileContents = readTextFile(filename)
```

[1] Die Vorherrschaft des Punktes als Dezimaltrenner im Programmierkontext kommt hauptsächlich daher, dass er in den USA üblich ist und von dort viele Programme kommen. Tatsächlich ist das Komma als Dezimaltrenner weltweit betrachtet deutlich weiter verbreitet...

Listing 3: Schnittstelle einer Funktion zum Schreiben von Textdateien

```
writeTextFile(filename,fileContents)
```

Hierbei sei `filename` eine Zeichenkette (*string*) und `fileContents` ein `cell array`.

Achten Sie auch hier wieder auf eine grundlegend robuste Programmierung, indem Sie `try-catch`-Konstrukte verwenden. Wie Sie wissen, ist das essentiell, um bei Fehlern in der Verarbeitung durch nicht geschlossene Dateien nicht in größere Schwierigkeiten mit Ihrem Datei- und Betriebssystem zu kommen, die sich im Zweifel nur durch einen Neustart des gesamten Systems beheben lassen.

Testen Sie anschließend Ihre beiden Funktionen, indem Sie eine beliebige zur Verfügung stehende Textdatei (ggf. auch einfach eine der Funktionen selbst) mit der ersten Funktion einlesen und mit der zweiten Funktion dann in eine neue Datei schreiben.

Aufgabe 5—5 (Einfaches Parsen von Textdateien)

Nachdem Sie in der vorangegangenen Aufgabe eine generische Routine zum Einlesen von Textdateien erzeugt haben, können Sie diese nun nutzen, um eine Parameterdatei von Bruker-EPR-Daten (die mit einem Bruker EMX-Spektrometer erzeugt wurden) einzulesen und anschließend zu parsen.

Versuchen Sie, die Datei `bruker.par` einzulesen und anschließend zu parsen. Schauen Sie sich dafür zunächst diese Datei mit einem Texteditor, z.B. dem MATLAB-Editor, an. Nach dem erfolgreichen Einlesen wundern Sie sich ggf., dass Sie statt einem `cell array` mit mehreren Elementen lediglich eines mit nur einem Element vorfinden. Das liegt am (zugegebenermaßen hinreichend seltsamen) Format dieser Datei. Das „Geheimnis“ sind die Zeilenenden. Sie können sich hier dadurch behelfen, dass Sie den String in ihrem `cell array` mit der Funktion `strsplit` zerlegen und als Trenner (*delimiter*) „`\r`“ verwenden. Beachten Sie hier weiterhin, dass `strsplit` als erstes Argument zwingend eine Zeichenkette (*string*) erfordert.

Der Aufruf von `strsplit` könnte etwa wie folgt aussehen, wenn wir einmal davon ausgehen, dass sich in der Variable `fileContents` der Inhalt der eingelesenen Datei in einem `cell array` befindet.

Listing 4: Beispiel eines Aufrufs von `strsplit`

```
lines = strsplit(fileContents{1},'\r');
```

In `lines`, das wiederum ein `cell array` ist, befinden sich nun die einzelnen Zeilen Ihrer Ursprungsdatei. Nun können Sie jedes einzelne Element dieses `cell arrays` wiederum parsen und entsprechende Schlüssel-Wert-Paare erzeugen, wie Sie das in der Präsentation zur Lektion gesehen haben.

Hinweis: Das Parsen dieser Datei ist die Voraussetzung für die Verarbeitung der Bruker-Binärdateien. Anderweitig kann man nicht auf die Dimension der Daten schließen, die wiederum essentiell für das Verarbeiten der eingelesenen Binärdatei `bruker.spc` ist.

Aufgabe 5—6 (Import von Bruker-EPR-Daten)

Mit all den vorangegangenen Vorarbeiten können wir nun tatsächlich daran denken, Bruker-Dateien, die mit einem Bruker EMX-Spektrometer erzeugt wurden, einzulesen.

Eine wichtige Information, die Sie in der Bruker-Dokumentation zum Dateiformat finden, lautet, dass es sich beim Binärformat um „*4 byte floating point*“ handele. Darüber hinaus können Sie davon ausgehen, dass die Datei ausschließlich Binärdaten in diesem Format enthält, Sie also die Datei in einem Schwung einlesen können.

Zum Einlesen der Daten müssen folgende Schritte in genau dieser Reihenfolge abgearbeitet werden:

- Parsen der `PAR`-Datei
- Einlesen der `SPC`-Datei
- Verarbeiten der eingelesenen Binärdaten

Versuchen Sie, eine Einleseroutine zu schreiben, die Ihnen die Daten in `bruker.par` bzw. `bruker.spc` einliest. Die Dimension der Daten bekommen Sie aus dem Feld `RES` in der `PAR`-Datei. Wären die Daten nicht ein- sondern zweidimensional, beinhaltete die `PAR`-Datei noch ein zusätzliches Feld `REY` für die zweite Dimension.[2]

Stellen Sie anschließend die Resultate grafisch dar.

Zusatzaufgabe: Versuchen Sie, durch „informed guessing“ die beiden weiteren Parameter in der `PAR`-Datei herauszufinden, die Ihnen die Information über die x-Achse liefern, und stellen Sie die Daten entsprechend auf dieser x-Achse korrekt dar.

[2] Netterweise sind beide Felder in der Spezifikation des Dateiformates, wie man sie von Bruker bekommen kann, nicht aufgeführt. Aber „informed guessing“ und Kenntnis des durchgeführten Experimentes ist mitunter zielführend.