Universität zu Köln

Prof. Dr. Angela Kunoth
Max Glembock, Laslo Hunhold

Matlab Präsenzaufgaben — WS 2017/2018

Übungsblatt 9

Ausgabe: 13.12.2017

Matlab in der Praxis

Nachdem wir uns auf den vorherigen Übungsblättern mit grundlegenden und bereits fortgeschrittenen Matlab-Funktionen auseinander gesetzt haben, möchten wir uns auf diesem Blatt an ein praxisbezogeneres Beispiel wagen. Das Ziel dieses Blattes ist es, dass wir gemeinsam einen *Rohdatenverarbeiter* entwickeln, mit dessen Hilfe wir Aktien- bzw. Indexverläufe graphisch via Matlab darstellen können. Die meisten Funktionen, die wir hierzu brauchen, sind uns bereits vertraut. Wir möchten uns aus Gründen der Einheitlichkeit auf eine Darstellung des DAX festlegen. Eine `csv`-Datei mit historischen DAX-Werten können wir unter der URL `https://de.finance.yahoo.com/quote/%5EGDAXI/history` herunterladen. Dort findet sich am Tabellenkopf eine Verknüpfung mit der Bezeichnung „Daten herunterladen“, die den Download startet.
Haben wir die Datei auf unseren Rechnern, so merken wir beim Betrachten mit Excel, dass diese trotz der angeblichen Aufbereitung noch ein wenig chaotisch ausschaut. Die angegebenen Daten der Handelstage, die Öffnungs-, Hoch-, Tief-, Schluss- und adjustierten Schlusskurse sowie auch die Tageshandelsvolumina befinden sich nur von Kommata getrennt in der Excel-Spalte `'A'`. Wir werden im Folgenden eine Möglichkeit kennen lernen, mit dieser Unordnung umzugehen.
Zunächst möchten wir uns mit dem Importieren der Daten in Matlab beschäftigen. Bei einer bereits geöffneten Datei (siehe Übungsblatt 8, `fopen`) kann man ihre Daten mit dem Befehl

```
data = textscan(fileID, formatSpec);
```

einlesen. Hierbei werden die Daten in einem Cell-Array `data` gespeichert. Die zu lesende Datei wird mit der `fileID` identifiziert. Unter `formatSpec` wird eingestellt, wie die einzelnen Textstücke der zu lesenden Datei zu interpretieren sind, d.h. ob die Textstücke bspw. als Zahlen oder Buchstaben interpretiert und verwaltet werden sollen. Die unterschiedlichen Argumente hierfür können in der Matlab-Dokumentation eingesehen werden. Ein **zusätzliches Argument** für `textscan` ist `'delimiter'`, worauf folgend ein Zeichen als String eingetragen werden kann, was beim Einlesen der Daten als Trennzeichen interpretiert wird. Ebenfalls kann durch das Argument `'HeaderLines'` und darauf folgend einen positiven Integer-Wert die Anzahl der obersten Zeilen festgelegt werden, die Matlab beim Einlesen überspringen soll. Somit haben wir das Handwerkszeug für den ersten Schritt unserer Arbeit kennen gelernt.
Wir müssen nun noch ein paar Voraussetzungen für unsere Modellierung treffen: Bei der graphischen Darstellung des DAX-Verlaufs ist es ausreichend, wenn wir aus der heruntergeladenen Tabelle nur die Daten der Handelstage sowie die dazugehörigen Schlusskurse in Matlab importieren. Man beachte, dass die Verteilung der Handelstage *nicht uniform* ist. Das bedeutet, dass wir nur Kurse von den Tagen haben, an denen die Börse geöffnet hatte, und keine Kurse von Wochenenden oder Feiertagen. Bei unserem späteren Plot wird Matlab diese fehlenden Tage daher nicht berücksichtigen.

Aufgabe 34 (Rohdatenverarbeiter – Teil 1)
Erstellen Sie eine Funktion `Rohdatenverarbeiter(filename)`. Diese Funktion werden wir im Verlauf des Blattes Stück für Stück ausbauen. Der Name `table.csv` der heruntergeladenen Datei soll dabei als String über `filename` eingegeben werden können. Öffnen Sie die `csv`-Datei über `fopen` mit Leserechten und importieren Sie die Daten über den `textscan`-Befehl. Als `formatSpec` tragen Sie `'%s %*s %*s %*s %*s %s %*s %*s'` und als `delimeter` ein Komma (`','`) ein.[1] Achten Sie darauf, die Überschrift nicht einzulesen. Schließen Sie anschließend die Datei wieder über `fclose`.

Haben wir die Datei erfolgreich eingelesen, so ist in unserem Workspace nun ein neues 1×2-Cell-Array gespeichert, über das wir die Handelstage sowie die DAX-Schlusskurse in einem chronologisch absteigenden Format einsehen können. Leider liegen die Daten der Handelstage in einem String-Array und somit in einem

[1] Durch diese Kombination wird bewirkt, dass beim Einlesen der Daten jedes gelesene Komma als eine Trennung gehandhabt wird, weshalb beim eingetragenen `formatSpec` auch insgesamt sieben Formate integriert sind (für jede Spalte ein Format). Durch die *-Sterne wird Matlab kenntlich gemacht, dass die entsprechende Spalte nicht eingelesen werden soll. Somit lesen wir nur die erste Spalte mit den Handelstagen sowie die fünfte Spalte mit den Schlusskursen ein.

für unsere Zwecke schlecht zu handhabenden Format vor. Eine Möglichkeit, solche Daten handhabbarer zu machen, bietet die Funktion `datenum`, die jedem Datum im String-Format eine eindeutige Nummer (beginnend beim 0. Januar 0000) zuordnet:

```
DateNumbers = datenum(DateStrings, formatIn);
```

Hierbei werden in einem Vektor `DateNumbers` die Zahlenwerte für die Daten gespeichert. `DateStrings` bezeichnet das String-Array mit den zu transformierenden Daten-Strings. Unter `formatIn` kann als String ein Format eingetragen werden in dem das Datum interpretiert werden soll. Diese Eintragung ist optional, eine vorab richtige Format-Einstellung kann jedoch eine erhebliche Einsparung an Rechenzeit zur Folge haben. Haben Sie bspw. das String-Datum `'01/03/2000'` im europäischen Datenstandard vorliegen, so empfiehlt es sich für `formatIn` den String `'dd/mm/yyyy'` einzugeben. Eine ausführliche Erklärung zu den unterschiedlichen Formaten erhalten Sie durch einen Blick in die entsprechende Matlab-Dokumentation.

Aufgabe 34 (Rohdatenverarbeiter – Teil 2)
Erstellen Sie einen Vektor `DateNumbers`, in dem Sie die Handelstage aus dem String-Array Ihres Cell-Arrays als Zahlen abspeichern. Nutzen Sie hierbei die Funktion `datenum`. Achten Sie auf die Eingabe eines richtigen Interpretationsformats `formatIn`, testen Sie jedoch auch eine Transformation ohne ein angegebenes Format.

Wichtig:

- Achten Sie darauf, dass Sie beim Zugriff auf das String-Array innerhalb des Cell-Arrays eine Doppelindizierung verwenden müssen. Ist das String-Array im Cell-Array also an Position `{1,1}` und Sie möchten innerhalb des String-Arrays auf Position `(3,4)` zugreifen, so geschieht durch den Zugriff `data{1,1}(3,4)`.
- Verwenden Sie, wenn möglich, bei der Transformation Ihrer Datenwerte keine `for`-Schleifen, sondern den Doppelpunkt-Operator. Hierdurch ergibt sich ebenfalls eine enorme Zeitersparnis.

Nun haben wir einen zusätzlichen Vektor vorliegen, in dem die Daten der Handelstage als Zahlen abgespeichert sind. Zu jedem Handelstag des DAX gibt es ferner einen eindeutigen Schlusskurs, welcher sich noch im Cell-Array `data` befindet. Diese Tagesschlusskurse sind ebenfalls in einem String-Array abgespeichert, so dass wir sie für die Handhabung zunächst noch in ein Zahlen-Array umcasten müssen. Matlab bietet eine ganze Reihe von Cast-Funktionen mit denen man Strings in Zahlen, Hexadezimalzahlen oder Matrizen umwandeln kann. In unserem Fall möchten wir Strings in Double-Werte umwandeln. Der Befehl dazu lautet:

```
X = str2double('str');
```

In `X` wird dabei der Double-Wert gespeichert, der vorab im String `'str'` übergeben wurde. Alternativ kann auch ein ganzes String-Array eingesetzt werden.

Aufgabe 34 (Rohdatenverarbeiter – Teil 3)
Erstellen Sie einen Vektor `Values`, in dem Sie die Zahlenwerte der Schlusskurse speichern. Nutzen Sie zum Casten die Funktion `str2double`.

Die Handelstage sowie die Schlusskurse liegen nun chronologisch aufsteigend in Zahlenvektoren vor. Wenn wir nun durch `plot(DateNumbers, Values)` einen Plot erstellen, erkennen wir bereits den bekannten DAX-Verlauf. Wir werden uns nun mit einer angemessen Gestaltung dieses Plots beschäftigen. Insbesondere möchten wir die `x`-Achse so gestalten, dass dort für jedes Jahr der 1. Januar angegeben wird. Um die Zahlenwerte der Handelstage in Matlab wieder als Daten zu interpretieren, gibt es die Funktion

```
DateMatrix = datevec(DateNumbers);
```

Dabei ist `DateMatrix` eine $m \times 6$-Matrix für einen m-dimensionalen Vektor `DateNumbers`. Die Matrix `DateMatrix` besitzt 6 Spalten, da in ihr das Jahr, der Monat, der Tag, die Stunde, die Minute und die Sekunde des jeweils transformierten Zahlenwertes gespeichert werden. Ein Datum im String-Format kann auch direkt durch den Befehl

```
DateMatrix = datevec(DateString, formatIn);
```

transformiert werden. Das Argument `formatIn` hat die gleiche Bedeutung wie bereits erklärt. Umgekehrt kann `DateMatrix` durch

```
DateNumbers = datenum(DateMatrix);
```

auch wieder in Zahlen transformiert werden.

Aufgabe 34 (Rohdatenverarbeiter – Teil 4)
Erstellen Sie mit dem Befehl `datevec` einen Vektor `x_dates`, in dem Sie jeweils die Zahlenwerte für den 1. Januar jeden Jahres zwischen dem Startdatum und dem Enddatum speichern. An der Position `1` soll der Vektor das Startdatum und an der Position `end` das Enddatum enthalten. Lassen Sie den 1. Januar des Jahres **nach** dem Startdatum sowie den 1. Januar des Jahres **vor** dem Enddatum weg, um spätere Überschneidungen in der Darstellung zu vermeiden.

Wir haben somit einen Vektor mit Zahlenwerten respektive Daten erstellt, die wir auf der `x`-Achse als Ticks (Einheiten) angezeigt haben möchten. Um diese nun auch anzeigen zu lassen, müssen wir eine neue `figure` kreieren und über den `axis`-Befehl spezielle Einstellungen vornehmen. Zusätzlich möchten wir auch noch kleinere Ticks sowohl auf der `x`- als auch auf der `y`-Achse angezeigt bekommen. Der Befehl inkl. Initialisierung für die Achseneinstellungen lautet:

```
my_axes = axes('parent', figure, 'YMinorTick', 'on',
'XMinorTick', 'on', 'XTick', x_dates);
```

Hierbei wird über `'parent'` das entsprechende `figure`-Fenster ausgewählt, in welchem die Achseneinstellungen gelten sollen. Über `'YMinorTick'` bzw. `'XMinorTick'` sowie das jeweils darauffolgende `'on'`, wird die Darstellung von kleineren Ticks zwischen unseren Daten aktiviert. Die Daten selbst werden durch `'XTick'` und der darauffolgenden Angabe des Vektors `x_dates` als Einheiten auf die `x`-Achse gesetzt.
Damit diese Einstellungen auch bei weiteren Formatierungen des Plots erhalten bleiben, muss zusätzlich der Befehl `hold(my_axes, 'all')` verwendet werden. Da unsere Werte aus `x_dates` jedoch zum jetzigen Zeitpunkt noch Zahlenwerte sind, müssen wir diese abermals umwandeln. Dazu gibt es den Befehl `datetick`:

```
datetick(tickaxis, dateFormat)
```

Hierbei ist für `tickaxis` entweder `'x'` oder `'y'` einzusetzen, je nachdem, für welche Achse die übergebenen Ticks umgestellt werden sollen. Für `dateFormat` muss das gewünschte Datumsformat als String eingetragen werden. Wir entscheiden uns für das europäische Standardformat `'dd.mm.yyyy'`. Eine zusätzliche Eingabe des Arguments `'keepticks'` sorgt dafür, dass alle angegebenen Ticks erhalten bleiben.

Aufgabe 34 (Rohdatenverarbeiter – Teil 5)
Erstellen Sie nun einen Plot mit den Zahlenwerten von `DateNumbers` auf der `x`-Achse und den Zahlenwerten von `Values` auf der `y`-Achse. Formatieren Sie die `x`-Achse gemäß den obigen Einstellungen. Fügen Sie ferner eine angemessene Beschriftung der `y`-Achse sowie einen Titel hinzu. Aktivieren Sie die Gitternetzlinien im Plot über `grid on`.

Zum Abschluss müssen wir noch eine Justierung der Daten auf der `x`-Achse vornehmen, so dass diese auch gut und vollständig lesbar sind. Hierzu möchten wir die externe Funktion `rotateticklabel(axes, rot)` verwenden, die man unter der URL `http://www.mathworks.com/matlabcentral/fileexchange/8722-rotate-tick-label` kostenlos herunterladen kann. Als Funktionsargumente sind die zu bearbeitenden Achsen `axes` sowie der Rotierungsgrad `rot` einzutragen. In unserem Fall erhalten wir für `axes=my_axes` und `rot=60` eine gut lesbare Formatierung der `x`-Achse. Unser Ergebnis sieht nun aus wie folgt:

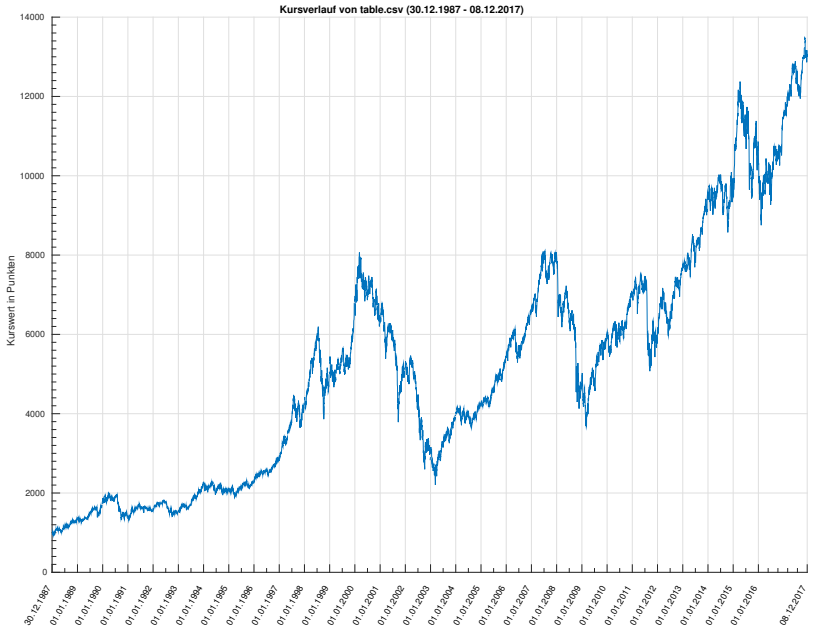

Abbildung 1.1: *Darstellung des DAX durch den Rohdatenverarbeiter*

Bemerkung: Die Ideen und Teile der Umsetzung für das konstruierte Programm entstammen der Seminararbeit *Verarbeitung von Rohdaten einer Zeitreihe* von Thilo Schmidtmann aus dem Seminar „Datenkompression und -analyse“ von Frau Prof. Dr. Kunoth aus dem WS 2013/14. Vielen Dank für die Bereitstellung!