Tag 7: Datenstrukturen

Version: 6.10.2014

A) DATENSTRUKTUREN

Wenn man komplizierte Datenmengen verwalten möchte, stößt man bei der Verwendung von Variablen der bisher im Kurs eingeführten Typen schnell auf das Problem, dass die Lage unübersichtlich wird. Gerade wenn man Variablen verschiedener Typen aufeinander beziehen muss (z.B. Namen von Versuchstieren mit ihrem Gewicht, ihrem Geschlecht, einer Information, ob sie schon mal Junge bekommen haben, der verwendeten Versuchsbedingung und dem dabei erzielten Ergebnis), muss man sehr sorgfältig arbeiten, damit auch alle Vektoren gleich lang bleiben etc. Wenn man außerdem alle diese Informationen in mehreren Funktionen braucht, muss man jeweils alle Variablen übergeben und darf dabei nicht die Reihenfolge verwechseln. Um solche Aufgaben zu lösen, gibt es deshalb einen praktischeren Weg: die Datenstruktur:

- In einer Struktur kann man mehrere zusammengehörige Variablen in einer einzigen zusammenfassen, auch wenn sie verschiedene Typen besitzen.
- Die Variablen innerhalb der Struktur werden auch als "Felder" bezeichnet.
- Die Struktur kann man bequem zwischen Funktionen übergeben und dort die jeweils relevanten Felder benutzen und evtl. verändern.
- Eine Struktur erhält man durch die Angabe ihrer Felder mit der Syntax

```
strukturname.feldname=Belegung
```

Ein Beispiel: Man kann sich eine Struktur vorstellen wie einen Rucksack: statt alle Bücher, Stifte, Taschentücher, Geldbörse, USB-Stick etc. einzeln in die Uni zu tragen und ständig drauf aufzupassen, ob man noch alles bei sich hat, stopft man alles in den Rucksack und nimmt ihn mit. Wenn man dann im Matlabkurs sitzt, benutzt man aus dem Rucksack den USB-Stick, Stifte und Papier; während man in der Mensa eher den Geldbeutel braucht.

- Wie könnte diese Struktur aussehen?

Der Vorteil von Strukturen ist, dass man sich durch die selbst gewählten Namen sehr aussagekräftige Kombinationen

zusammenstellen kann (die man normalerweise auch Jahre später noch versteht, wenn man noch einmal in sein altes Programm ansieht).

Der Nachteil ist, dass man sie nicht einfach linear durchsuchen kann – z.B. funktioniert der `find`-Befehl nicht zum Durchsuchen aller Felder einer Struktur und man kann auch nicht einfach alle Felder mit einer Schleife durchsuchen.

Aufgaben:

T7A1) Beim Beispiel der Katzen im Tierheim haben wir gesehen, dass man für manche Fragestellungen verschiedene Arten von Daten aufeinander beziehen muss. Wir haben dort mehrere Vektoren bzw. Matrizen benutzt und diese über jeweils gleiche Indizes kombiniert. Diese Vorgehensweise funktioniert, ist aber nicht ungefährlich:

Laden Sie die Daten über die Katzen noch einmal ein [katzen.mat] und sortieren Sie die Katzen nach ihrem Alter. Was ist dann außerdem nötig, damit Sie die anderen Daten über die Katzen noch abfragen können?

T7A2) Um alle Daten beieinander zu halten, führt man in Matlab eine Struktur ein. Eine Struktur besteht aus mehreren Feldern, die verschiedene Datentypen besitzen können. Beispielsweise könnte man für die Verwaltung einer Mäusezucht folgende Struktur benutzen. Die erste Maus erzeugt man mit:

```
maus.ohrmarke='M7777';
maus.kaefig=5;
maus.weiblich=true;
maus.wurfgroessen=[5 8 7];
```

- Lassen Sie sich ausgeben, was gespeichert ist unter `maus`, `maus.kaefig` und `maus.wurfgroessen(2);`
- Um eine weitere Maus hinzuzufügen, muss man dieser einen Index mitgeben:

```
maus(2).ohrmarke='M7735';
maus(2).kaefig=9;
maus(2).weiblich=true;
maus(2).wurfgroessen=[13 4 10 9];
```

- Denken Sie sich ein paar weitere Mäuse aus und lassen Sie sich ausgeben:
  - `maus`
  - Die Käfignummern der Mäuse 1 bis 4

◦ Die Größe des jeweils ersten Wurfs der ersten 3 weiblichen Mäuse.

T7A3) Keine Maus lebt ewig. Schreiben Sie eine Funktion, die bei Angabe der Ohrmarke den richtigen Eintrag in der Struktur maus löscht.

B) CELL ARRAYS

Nah verwandt mit Strukturen sind die sogenannten cell arrays. Mit diesen kann man Listen von Variablen verschiedener Typen verwalten, wobei die einzelnen Einträge nicht durch Namen bezeichnet sind (wie bei Strukturen), sondern durch Indizes. Cell arrays sind also sozusagen Matrizen, in denen verschiedene Datentypen gemischt werden können.

Man kann sie sich als eine Art Lagerraum vorstellen, in dem die Regalbretter durchnummeriert sind und ganz unterschiedliche Dinge enthalten können, die man durch die Nummerierung aber schnell finden kann. Ein Vorteil von Cell Arrays ist, dass man sie (im Gegensatz zu Strukturen) gut linear durchsuchen kann, indem man ihre Cells einfach durchzählt.

Ihr Nachteil ist, dass sie keine sprechenden Namen haben, sondern (genau wie Matrizen) Nummern benutzen, um bestimmte Daten zu adressieren. Außer einem typischen Beispiel für die Anwendung einer Cell-Variablen wollen wir den Spezialfall behandeln, wie man in Matlab Funktionen mit einer variablen Anzahl von Ein- oder Ausgabeparametern programmiert. Hierfür benötigt man die Schlüsselbegriffe *varargin* und *varargout* anstelle von expliziten Angaben der Parameternamen in der Kopfzeile der Funktion. *varargin* und *varargout* sind dann im Programmcode die Namen eines *Cell Arrays* der Ein- bzw Ausgabeparameter. Die Anzahl der beim Funktionsaufruf übergebenen Argumente kann man grundsätzlich in jeder Funktion (auch wenn *varargin* und *varargout* nicht verwendet werden) im Programmtext mit den von Matlab automatisch belegten Variablen *nargin* (number of input arguments) und *nargout* (number of output arguments) abfragen.

Matlab:

Auf ein einzelnes Element eines Cell Arrays wird zugegriffen, indem mit geschweiften Klammern indiziert wird.

- `A{1}=[1 2 3];` % ein cell array wird erzeugt

- `A{2}='A B C D E';` % und um mehr Eintraege ergaenzt.
- `A{3}=logical(0)`
- `A{2,2}=9` % Cell Arrays können auch mehrere Dimensionen haben, so wird aus dem eindimensionalen Cell Array ein zweidimensionales und alle zur kompletten Matrix fehlenden Einträge werden als leere Menge `[ ]` gesetzt.
- `a=A{3,1}(2)` % Sind in einem Cell Array Vektoren oder Matrizen gespeichert, lassen sich auch daraus einzelne Elemente herausgreifen. Dies ist das zweite Element eines in `A{3,1}` gespeicherten Vektors

Aufgaben:

T7B1) Vollziehen Sie folgendes Beispielprogramm zur Verwendung von *cell arrays* aus dem Matlabkurs für Doktoranden nach: [celldemo.m]

T7B2) In einem Praktikum wurde eine EEG-Messung (EEG=Elektroencephalogramm, Messung von Hirnströmen) durchgeführt. Nach der Auswertung (die wir jetzt mal vernachlässigen wollen) wurden die Daten in Matlab in der Datei [experiment1.mat] abgespeichert. Wenn Sie diese Datei laden, werden Sie feststellen, dass sie nur eine Variable vom Typ *cell* (mit 2 Elementen) enthält. In dieser Variablen ist sämtliche Information gespeichert, die zum Experiment gehört.

a) Laden Sie die Datei und schauen sie sich den Inhalt der *Cell-* Variablen an:

- Das erste Element ist eine Struktur, die die Information über das Experiment enthält, wie z.B. den Namen der Originaldatei und Anzahl der Messpunkte.
- Das zweite Element enthält die gemessenen Daten.

b) Plotten Sie die Daten jeder Versuchsperson in einem separaten Plot.

- In jedem Plot soll zusätzlich noch die Bezeichnung der jeweiligen Versuchsperson als Bildüberschrift auftauchen (Befehl: *title*). Diese steht in der Struktur (1. Element der Cell-Variablen) in dem Eintrag Names.
- Tipp: Verwenden sie eine *for*-Schleife, deren Zählvariable als Index für die Daten und für die Bezeichnung benutzt werden kann.

*c) Die Daten geben das EEG Signal in µV wieder und wurden mit

500 Hz aufgezeichnet. Erweitern Sie Ihr Programm durch eine entsprechende Anpassung und Beschriftung der Achsen ihrer Plots.

*d) Erzeugen Sie eine neue `Cell`-Variable, die genauso aufgebaut ist wie die bisherige, die jedoch nur die Daten der eindeutig studentischen Versuchspersonen enthält.

T7B3) Als zweites wollen wir hier einen Spezialfall für die Verwendung von cell arrays behandeln. Schauen Sie sich folgendes Programm an:

```
function [varargout]=testvar(varargin)
%Schleife über alle Eingabeargumente
for i=1:length(varargin)
%die Eingabeargumente werden in einen Vektor geschrieben:
    x(i)=varargin{i};
end
%Schleife über die Ausgabeargumente, nargout ist deren
Anzahl, diese Variable wird automatisch bei jedem
Funktionsaufruf von Matlab gesetzt und kann im
Programmcode verwendet werden:
for i=1:nargout
    varargout{i}=2*x(i);
end
```

- Benutzen Sie diese Funktion mit `[a b c]=testvar(1, 2, 3,4).`
- Probieren Sie auch andere Kombinationen aus Ein- und Ausgabeargumenten aus.
- Dieses Beispiel ist eigentlich ein recht schlechtes, da es ausschließlich für Variablen vom Typ double funktioniert. Erweitern Sie es so, dass alle Variablen aller anderer Datentypen unverändert wieder ausgegeben werden.

C) ANWENDUNGSBEISPIEL: STIMULATION IN EINEM PSYCHOPHYSIKALISCHEN EXPERIMENT

Nun sollen verschiedene Aspekte des in diesem Kurs Gelernten in einer Aufgabe zusammengefasst werden - der Steuerung der Stimulation eines psychophysikalischen Experiments.

T7C1) Ein Beispiel für eine visuelle Stimulation in einem einfachen psychophysikalischen Experiment ist in der Demo [zufallsquadrate.m] gezeigt. Schauen Sie sich dieses Programm an und probieren Sie es aus. Führen Sie schrittweise

folgende Änderungen an dem Programm durch:

a) Bestimmen Sie bei jedem "Sprung" eine zufällige Farbe, die dem neuen Quadrat zugewiesen wird.

b) Lassen Sie die Dauer der Pause als weiteres Argument durch den Benutzer angeben.

c) Nehmen Sie die Benutzerangabe der Pause als Maximalwert und bestimmen Sie jede individuelle Pause zufällig zwischen 0s und diesem Maximalwert.

d) Machen Sie die Benutzereingabe der Pause optional. Wenn der Benutzer einen Pausenwert eingibt, wird dieser benutzt, sonst wird 2sec als defaultwert verwendet.

e) Ändern Sie die Wahrscheinlichkeit der Position von einer Gleichverteilung zu einer Normalverteilung rund um die Mitte der Abbildung.

T7C2) Häufig möchte man die genauen Parameter wissen, die bei einer Stimulation benutzt wurden, entweder um einen Versuch zu reproduzieren, oder um ihn exakt auswerten zu können. Dies ist insbesondere für zufällige Reizungen sehr wichtig. Eine gute Möglichkeit, die verschiedenen Parameter übersichtlich zu speichern, ist die Einführung einer Struktur, die alle wichtigen Angaben enthält. Während das Programm läuft, wird diese Struktur gefüllt und im Anschluss an den Programmablauf abgespeichert. Eine entsprechende Version der Stimulation mit Zufallsquadraten finden Sie in [zufallsquadrate_speicher.m]. Eingeladen und reproduziert werden diese Daten mit [zufallsquadrate_laden.m]. Schauen Sie sich beide Programme an und versuchen Sie nachzuvollziehen, wie mit der Struktur umgegangen wird.

a) Im Moment werden die Parameter immer unter dem Namen `'zufquadparameter.mat'` abgespeichert. Dadurch wird diese Datei bei mehrfachem Aufrufen überschrieben. Ändern Sie die Funktion `zufallsquadrate_speicher` so, dass der Benutzer als Argument einen Dateinamen seiner Wahl angeben kann.

b) Erweitern Sie die Funktion so, dass der Benutzer einen Dateinamen angeben kann (der dann auch zum Speichern verwendet wird) oder nicht. Wenn kein Dateiname angegeben wird, speichert die Funktion die Parameter in die Datei `'standard.mat'`

c) Weisen Sie wie in der letzten Aufgabe dem Quadrat jeweils eine zufällige Farbe zu, die ebenfalls jeweils mit abgespeichert wird.

d) Erweitern Sie Ihre Funktion um ein zusätzliches Argument, mit dem man wählen kann, ob

• man zufällige Farben haben möchte,

• oder ausschließlich rote, blaue und grüne Quadrate.

• Wenn die Option mit nur drei Farben gewählt wird, sorgen Sie dafür, dass diese in zufälliger Reihenfolge auftreten, aber jede der Farben gleich oft gezeigt wird (bzw. eine Farbe einmal häufiger oder seltener auftritt, wenn die Anzahl der Wiederholungen nicht durch 3 teilbar ist.)

e) Passen Sie die Funktion *zufallsquadrate_laden* den Änderungen in *zufallsquadrate_speicher* an.

HAUSAUFGABE:

*T7H1) Für den Matlabkurs für Fortgeschrittene habe ich Demo-Programme geschrieben, die die Benutzung von Strukturen verdeutlichen sollen. Vollziehen Sie die Funktionsweise nach:

[structuredemo.m] benutzt [make_exampledata.m]

T7H2) [CellRespMat.mat] enthält Antworten von 20 Nervenzellen in der Retina, deren Aktivität in einem Experiment aufgezeichnet und in dieser 3D-Matrix abgespeichert wurde. Die erste Dimension der 3D-Matrix entspricht den 20 Zellen, die zweite der Zeit (500 Zeitschritte mit je 1 ms) und die dritte Dimension der Stimuluspräsentation (192 Präsentationen des gleichen Reizes). Z.B: *r=CellRespMat(5,100,45)* ist der 100ste Zeitschritt der Antwort von Zelle 5 bei der 45sten Reizpräsentation. Für jeden Zeitschritt antwortent jede Zelle mit *0* (kein spike) oder *1* (spike). Die Zellen antworten auf 192 Lichtreize (Muster auf dem Bildschirm), die sich mit gleicher Wahrscheinlichkeit nach links oder rechts bewegten. Die Richtung des jeweiligen Bewegungsreizes ist in einem Vektor abgespeichert: [sigma.mat] (*-1* heißt links, *1* rechts).

• Schauen Sie sich die Daten im Array Editor oder als Grafik an. Dafür müssen Sie die Daten auf 2D-Matrizen reduzieren. Der Befehl squeeze macht aus einer "Scheibe" einer 3D-Matrix eine echte 2D-Matrix, z.B. *M=squeeze(CellRespMat(5,:,:))*; enthält alle Zeitpunkte aller Präsentationen der Zelle 5.

• Überlegen Sie sich eine geeignete Art, durch Nutzung von cell Arrays oder Strukturen die Messdaten gemeinsam mit der zugehörigen Stimulation zu speichern, um den Umgang mit den Daten zu erleichtern.

• Hinweise:

◦ Die Art, wie man Daten organisiert, ist immer auch

Geschmacksache - es gibt hier nicht nur eine richtige oder beste Lösung.)

- *sparse()* funktioniert nur für 2D-Matrizen!

*T7H3) (entspricht ungefähr T4H5, kann von der eigenen Lösung dafür modifiziert werden)

Einen Kreis (der eigentlich ein 360-Eck ist), macht man mit Matlab so:

```
degrad =pi/180;
w=0:1:360;
si=sin(w*degrad);
 co=cos(w*degrad);
plot(co,si)
```

Wenn man das ausprobiert, sieht dieser Kreis ziemlich nach einem Ei aus. `axis equal` macht die Achsen quadratisch, so dass es schöner anzuschauen ist. Einen geschlossenen Linienzug wie diesen Kreis kann man mit dem Befehl `fill(x,y,farbdef)` mit einer Farbe ausfüllen. Farben definiert man dabei entweder mit dem auch beim plot-Befehl verwendeten Kürzel (z.B. `y` für yellow etc) oder mit Hilfe der RGB-Kodierung.

• Erzeugen Sie eine Struktur, in der für die verschiedenen Kreise (oder sonstigen n-Ecke) alle relevanten Informationen enthalten sind.

• Zeichnen Sie in gleicher Weise ein Quadrat, ein 3-Eck, ein 7-Eck und ein 25-Eck.

• Probieren Sie aus, wie man die Größe und die Richtung dieser Figuren ändert.

• *) Man muss das Programm nur geringfügig verändern, um einen in einem Zug gezeichneten fünfzackigen Stern zu zeichnen. Wie?

**T7H4) Sternenhimmel (entspricht ungefähr T4H6, baut auf der letzten Aufgabe auf): Sie wollen eine hübschen bunt gefüllten Sternenhimmel mit verschieden großen, bunt gefüllten Sternen (oder wahlweise sonstigen N-Ecken) mit verschieden vielen Ecken vor dunkelblauem Hintergrund darstellen.

a) Überlegen Sie sich eine Struktur, in der Sie alle relevanten Informationen zu ihren Sternen ablegen können.

b) Zeichnen Sie den Sternenhimmel mit mehreren Sternen unterschiedlicher Farbe.

c) Verschönern Sie Ihren Sternenhimmel dadurch, dass er den

ganzen Bildschirm füllt. (Nutzen Sie die Hilfe, um rauszubekommen, wie das geht.)

d) Zeichnen Sie Ihre Sterne an zufällige Orte auf den Himmel.

e) Man kann den Sternenhimmel als eine Art Bildschirmschoner verwenden.

- Zunächst sind die Sterne wie oben beschrieben bunt.
- Anschließend werden die Sterne nach jeweils einer kleinen Pause in zufälliger Reihenfolge gelb.
- Wenn alle Sterne gelb sind, bekommen sie in zufälliger Reihenfolge wieder ihre ursprüngliche Farbe.
- Diese Farbänderung wird solange wiederholt, bis der Benutzer das Programm abbricht.

f) Speichern Sie die Reihenfolge Ihrer Himmelserscheinungen mit Hilfe einer sinnvollen Struktur in einer Datei ab. Schreiben Sie ein Programm, das die Himmelsshow in gleicher Weise wiederholt.

***g) Stellen Sie sicher, dass die verschiedenen Sterne sich nicht überlagern.