**Tag 9: Datenstrukturen**

**A) Datenstrukturen**

**B) Cell Arrays**

**C) Anwendungsbeispiel: Stimulation in einem psychophysikalischen Experiment**

**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 dann 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 sieht). 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.

T9A1) Beim Beispiel der Katzen im Tierheim haben wir gesehen dass es nötig sein kann, verschiedene Arten von Daten so bereitzuhalten, dass man sie aufeinander beziehen kann. Wir haben dort mehrere Vektoren bzw. Matrizen benutzt. Diese Vorgehensweise funktioniert, ist aber nicht ungefährlich: Laden Sie die Daten über die Katzen noch einmal ein: [katzen.mat]

Sortieren Sie die Katzen nach ihrem Alter. Was ist dann außerdem nötig, damit Sie die anderen Daten über die Katzen noch abfragen können?

T9A2) 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];

a) 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.

\*b) Irgendwann segnen Mäuse das Zeitliche. Schreiben Sie eine Funktion, die bei Angabe der Ohrmarke den richtigen Eintrag in der Struktur maus löscht.

\*T9A3) Denken Sie sich eine Struktur für die Katzendaten aus, und setzen Sie die Vektoren in die Struktur um (natürlich nicht durch Eintippen aller Daten!).

**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.

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

A{1}=[1 2 3]; A{2}='A B C D E'; A{3}=logical(0)

erzeugt ein eindimensionales Cell Array.

Cell Arrays können auch mehrere Dimensionen haben,

A{2,2}=9

So wird aus dem eindimensionalen Cell Array ein zweidimensionales und alle zur kompletten Matrix fehlenden Einträge werden als leere Menge [ ] gesetzt.

Sind in einem Cell Array Vektoren oder Matrizen gespeichert, lassen sich auch daraus einzelne Elemente herausgreifen:

a=A{3,1}(2) ist das zweite Element eines in A{3,1} gespeicherten Vektors

Sehr praktisch an Cell Arrays ist, dass man sie (im Gegensatz zu Strukturen) gut linear durchsuchen kann, indem man ihre Cells einfach durchzählt.

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 ermöglichen es einer Funktion eine variable Anzahl von Eingabe- oder Ausgabeargumente zu haben. 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.

T9B1) Das EEG-Experiment: In einem Praktikum wurde eine EEG-Messung (EEG=Elektroencephalogramm) 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. Verwenden Sie hierzu eine for-Schleife. 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 die Zählvariable der Schleife als Index für die Daten *und* für die Bezeichnung.

\*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 sich eine neue Cell-Variable, die genauso aufgebaut ist wie die bisherige, die jedoch nur die Daten der eindeutig studentischen Versuchspersonen enthält.

T9B2) 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**

Am Schluss des Kurses sollen verschiedene Aspekte des Gelernten in einer Aufgabe zusammengefasst werden - der Stimulation eines psychophysikalischen Experiments.

T9C1) 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.

T9C2) 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 diese zweite Option gewählt wird, sorgen Sie dafür, dass diese drei Farben 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.)