Tag 5: Umgang mit Daten

Version: 23.8.2016

A) SUCHEN UND SORTIEREN

Wissenschaftlichen Datenauswertung erfordert es häufig, Daten nach bestimmten Kriterien zu durchsuchen, um dann nur denjenigen Teil der Daten zur Analyse zu verwenden, für den bestimmte Bedingungen erfüllt sind. Prinzipiell lassen Daten sich sehr gut (und in jeder beliebigen Programmiersprache) mit Hilfe von logischen Operatoren, Fallunterscheidungen und Schleifen durchsuchen.

Matlab stellt außerdem ein spezielles Konzept zur Verfügung, die logische Indizierung. Hierbei wendet man eine Matrix (oder einen Vektor) vom Typ *logical* als Indizes auf eine Matrix (oder einen Vektor) gleicher Größe an. Als Ergebnis erhält man einen Vektor, in dem nur diejenigen Elemente der ursprünglichen Matrix enthalten sind, die den logischen *true* in der Index-Matrix entsprechen. Z.B.:

```
a=[1 9 6; -9 7 0];

indi=logical([0 1 0; 1 0 1]);

b=a(indi)
```

liefert *b=[9;-9;0]*.

Sehr häufig erzeugt man die Matrix aus *logicals*, indem man einen Vergleichsoperator auf die ursprüngliche Matrix anwendet. Z.B. *ag0=a>0*

Speziell zum Suchen stellt Matlab den speziellen Befehl *find* zur Verfügung, mit dem man Matrizen nach bestimmten Kriterien durchsuchen kann. *find* gibt die Indizes derjenigen Elemente eines Vektors oder einer Matrix zurück, die ungleich 0 sind. Kombiniert man find mit logischen Operatoren, lassen sich Daten auch nach komplizierten Mustern sehr effizient durchsuchen (zumindest mit etwas Übung...). Syntax von find:

ind=find(v)	Gibt alle Indizes des Vektors *v* zurück, die ungleich 0 sind.
ind=find(v,k)	Gibt die ersten *k* Indizes des Vektors *v* zurück, die ungleich 0 sind.
ind=find(v,k,'last')	Gibt die letzten *k* Indizes des Vektors *v* zurück, die ungleich 0 sind.
[row,col]=find(M,...)	Gibt Zeilen- und Spaltenindizes der Elemente der Matrix *M* zurück, die ungleich 0 sind.
[row,col]=find(M,...)	Gibt Zeilen- und Spaltenindizes und Werte der Elemente der Matrix *M* zurück, die ungleich 0 sind.

Es kommt häufig vor, dass man Daten nicht in der ursprünglichen Reihenfolge belassen möchte, sondern nach bestimmten Kriterien sortieren. Im Prinzip geht auch das "per Hand" mit Fallunterscheidungen und Schleifen - das kann aber schnell ziemlich viel Arbeit werden. Deshalb gibt es in Matlab die beiden praktischen Befehl *sort* und *sortrows*.

sort sortiert die Daten unabhängig für jede Zeile (oder Spalte) einer Matrix. Bei *sortrows* wird die Matrix entsprechend einer Spalte sortiert, indem die Zeilen jeweils beibehalten werden.

vs1=sort(v1)	sortiert die Elemente eines Vektors *v1* in aufsteigender Reihenfolge
ms1=sort(m,1)	sortiert die Elemente jeder Spalte der Matrix *m* in aufsteigender Reihenfolge(unabhängig voneinander)
ms2=sort(m,2)	sortiert die Elemente jeder Zeile der Matrix *m* in aufsteigender Reihenfolge(unabhängig voneinander)
msd=sort(m,1,'descend')	sortiert die Elemente jeder Spalte der Matrix *m* in absteigender Reihenfolge (unabhängig voneinander)
[ms1,index]=sort(m,1)	gibt zusätzlich zur sortierten Matrix die Indizes zurück
mr1=sortrows(m1,n)	sortiert die Zeilen der Matrix *m1* gemäß ihrer Einträge in der n- ten Spalte in aufsteigender Reihenfolge.
mdesc1=sortrows(m1,-n)	sortiert die Zeilen der Matrix *m1* gemäß ihrer Einträge in der *n*- ten Spalte in absteigender Reihenfolge.
mr_n1=sortrows(m1,[n,1])	sortiert die Zeilen der Matrix *m1* ihrer Einträge in der *n*-ten Spalte in aufsteigender Reihenfolge. Bei gleichen Werten in der *n*-ten Spalte werden diese Zeilen entsprechend der 1. Spalte sortiert.
[mr1,index]=sortrows(m1,n)	gibt zusätzlich einen Vektor der Indizes zurück.

Aufgaben:

T5A1) Probieren Sie logische Indizierung aus:

m=[0 1; 5 9; 3 0] ind_m=logical([0 1; 0 0; 1 0]) m_teil=m(ind_m)

Erzeugen Sie mit logischer Indizierung einen Vektor, der nur diejenigen Elemente von *m* enthält, die größer als 2 sind.

T5A2) Probieren Sie den *find* Befehl aus:

v=[1 7 5 0 8 0 3 1] ind_ungleich_0=find(v) % Suche der Indizes von Elementen ungleich 0 v_ungleich_0=v(ind_ungleich_0) % Schreibe Werte der Elemente ungleich 0 in einen Vektor ind_gleich_1=find(v==1) % suche Indizes der Elemente, die gleich 1 sind

m=[0 1; 5 9; 3 0] [row_not0,col_not0]=find(m) % Matrix-Art des Suchens lin_ind_not0=find(m) % Suchen mit linearer Indizierung m_not0=m(lin_ind_not0) % Ausgabe der Elemente ungleich 0 als Vektor [row_1,col_1]=find(m==1)

Erzeugen Sie mit *find* einen Vektor, der nur diejenigen Elemente von *m* enthält, die größer als 2 sind.

T5A3) Laden Sie folgende Beobachtungsreihe: [fuetterungen.mat]

In dieser Matrix wurden an 31 Tagen eines Monats eingetragen, wie groß die Niederschlagsmenge des Tages in mm war (1. Spalte) und wie oft ein Meisenweibchen sein Nest zum Füttern angeflogen hat (2. Spalte). Finden Sie heraus:

- An welchen Tagen dieses Monats ist kein Regen gefallen?
- An welchen Tagen hat die Meise häufiger als 50 Mal gefüttert?
- An welchen Tagen ohne Niederschlag hat die Meise häufiger als 50 Mal gefüttert?
- Wie häufig ist es vorgekommen, dass die Meise an zwei aufeinanderfolgenden Tagen zusammen über 100 Mal gefüttert hat?

T5A4) Sortieren Sie die Matrix *fuetterungen* einmal nach der Anzahl Fütterungen und einmal nach der Niederschlagsmenge.

* T5A5) Falls Sie sich ein Beispiel für Suche in Messdaten ansehen wollen, hier ein Beispiel aus dem Fortgeschrittenen-Matlabkurs, bei dem der gleiche Datensatz auf drei verschiedene Arten durchsucht wird: [search_demo.m] benutzt den Datensatz [VP_data.mat]

B) GRAFISCHE DARSTELLUNG VON DATEN

Daten graphisch darzustellen ist eine extrem wichtige Aufgabe bei der wissenschaftlichen Datenauswertung. Der wichtigste Befehl hierfuer ist der bereits bekannte *plot* Befehl. In diesem Abschnitt wollen wir einige zusätzliche Möglichkeiten der grafischen Darstellung betrachten.

Eine wichtige Voraussetzung zum Verständnis von Grafik-Formaten ist die Farbkodierung, mit der man die in einer Grafik genutzten Farben genau

festlegen kann. Farben werden in der sogenannten RGB-Kodierung dargestellt, d.h. sie werden durch einen Vektor *farbe=[rot gruen blau]* definiert. In Matlab werden zur Kodierung für rot, grün und blau Werte zwischen 0 und 1 angegeben, z.B. ist *[1 0 0]* rot, *[0 0 0]* schwarz, *[1 1 0]* gelb, [0.6 0 0.6] dunkel-violett. So lassen sich beliebige Farben mischen. (Siehe Aufgabe T5B6)

Ein wichtiges Grafik-Format ist jpg. jpg-Dateien koennen in Matlab eingeladen und bearbeitet werden. Darin ist jeder Punkt durch einen Vektor von drei Farbwerten kodiert (allerdings etwas unterschiedlich zum in Matlab üblichen Format). Bei zweidimensionalen Abbildungen ergibt sich also eine 3 dimensionale Matrix der Größe MxNx3 (M und N ist die Anzahl Pixel in vertikaler und horizontaler Richtung). (Siehe Aufgabe T5B7)

Man kann Matlab-Abbildungen abspeichern und erneut in Matlab öffnen. Die einfachste Möglichkeit dazu ist, den Menüpunkt ">File >Save as" zu benutzen. Der Standard ist dann, dass die Abbildung mit der Endung *.fig* abgespeichert wird - dabei handelt es sich um ein Format, was nur von Matlab vernünftig verarbeitet werden kann, dort aber beim erneuten Einladen alle Eigenschaften der Abbildung beibehält.

Wenn man die Abbildungen mit einem anderen Programm benutzen will, z.B. um sie in ein mit Word geschriebenes Protokoll einzubinden, muss man ein anderes Dateiformat benutzen als *.fig*. Dazu kann man wie oben beschrieben abspeichern, muss jedoch das gewünschte Dateiformat auswählen und die Endung der Datei entsprechend anpassen.

T5B1) Häufig verarbeitet man mehr Daten als man sinnvoll in einer Abbildung unterbringen kann. Man kann eine Abbildung mit dem Befehl *subplot* in mehrere kleine Abbildungen teilen. Mit *subplot(2,3,5)* erzeugt man eine Matrix aus kleinen Abbildungen mit 2 Zeilen und 3 Spalten, wobei das 5. Fenster aktiv ist. In diesem kann der *plot*-Befehl (und alle dazugehörigen Formatierungen und Beschriftungen) normal verwendet werden. Um ein anderes der Abbildungsteile zu aktivieren, wird wiederum der Befehl *subplot* mit der entsprechenden Angabe für die aktuelle Nummer genutzt (z.B. *subplot(2,3,1))*. Generieren Sie eine Abbildung bestehend aus 4 subplots, in denen für *x=-5:0.1:5* die Kurven *x.^1* bis *x.^4* dargestellt sind.

T5B2) Matlab kann auch mehrere Grafikfenster gleichzeitig benutzen. Eine neue Abbildung bekommt man mit *figure*. Mit *figure(1)* und *figure(2)* springt man zwischen den aktiven Fenstern. *close(1)* schließt das zuerst erzeugte Fenster, *close all* schließt alle Grafikfenster. *clf* löscht den Inhalt des aktuellen Fensters.

Öffnen Sie zwei Grafikfenster plotten Sie im für den x-Vektor *0.1:0.1:10* im ersten Fenster die Quadratwurzel, im zweiten Fenster das Quadrat von *x*. Probieren Sie die oben genannten Befehle aus.

*T5B3) Kuchengrafiken sind eine beliebte Darstellungsmöglichkeit für Prozentanteile. Um die (gerade noch) aktuelle Verteilung der Sitze des Oldenburger Stadtrates darzustellen, würde man z.B. schreiben

sitze=[17, 14, 10, 3, 2, 1, 1, 1, 1];
pie(sitze)
Wenn man es noch etwas hübscher haben möchte, kann man z.B. auch eine der folgenden Möglichkeiten benutzen:

pie3(sitze)

pie(prozente,{'SPD','Gruene','CDU','Linke','FWBFO','FDP','Piraten','WFO', 'NPD'})

pie3(prozente,[0 1 0 0 0 0 0 0 0]).

T5B4) Wir haben vor einiger Zeit schon mal den Befehl *imagesc* benutzt, um uns den Inhalt einer Matrix farbkodiert anzeigen zu lassen. Benutzen Sie ihn, um sich die Verteilung einzelliger Algen in einer Petrischale anzusehen, die als Matrix *population* in [algen.mat] gespeichert ist.

Was bedeuten in dieser Abbildung die beiden Achsen?

In den Vektoren *x* und *y* sind Abstände der Messbereiche in x und y Richtung der Petrischale gespeichert. Mit *imagesc(x,y,population)* bekommt man aussagekräftige Achsenbeschriftungen.

Der Befehl *colorbar* erzeugt eine Legende für die Farbkodierung der Werte.

T5B5) Dieselbe Algenverteilung lässt sich auch dreidimensional als "Gebirge" darstellen. Der Befehl dafür ist *surf(population)*. Um aussagekräftige Achsen zu bekommen, muss jedem Punkt der Matrix ein x- und ein y-Wert zugeordnet werden (die Werte der Matrix *population* werden auf der z-Achse aufgetragen). Benutzen Sie dafür die mit in der Datei abgespeicherten Matrizen *xmatrix* und *ymatrix*. Der Befehl *colorbar* funktioniert auch hier. Probieren Sie den alternativen Befehl *mesh* aus. Was ist der Unterschied zu *surf*?

T5B6) Stellen Sie den Vektor *0:1:10* mit einer durchgezogenen dunkelroten Linie dar, indem Sie die RGB-Kodierung verwenden.

*) Wenn Sie Lust auf ausgefeilte Grafik haben: Versehen Sie auf der Linie die einzelnen Punkte mit türkis gefüllten Kreisen (hierfür braucht man *line properties*, siehe T1C7) oder suchen Sie sich eine besonders hübsche Farbkombination für Linien und Marker aus.

T5B7) Schauen Sie sich die Arbeitsweise des Skripts [meisenshow.m] an. Hierzu brauchen Sie außerdem die Datei [bmeise.jpg]

- Wie unterscheidet sich die Farbkodierung dieser jpg-Datei von der in Matlab üblichen?
- Teilen Sie die Matrix in drei Teile, um sich die Rot-, Grün- und Blau-Anteile des Bildes einzeln anzusehen.
- *) Erweitern Sie die Show um eine Meise mit zwei Köpfen

C) DATENAUFNAHME UND SPARSE MATRIZEN

Die meisten biologischen Daten ändern sich nicht schrittweise, sondern kontinuierlich (z.B. Ph-Wert, Körpertemperatur, Membranpotential einer Nervenzelle etc.). Wenn man solche Daten aufzeichnet, steht man vor dem Problem, dass eine kontinuierliche Aufzeichnung grundsätzlich nicht möglich ist. Spätestens wenn ein Computer benutzt wird, um die Daten zu speichern, müssen sie diskretisiert werden - es ist nur in bestimmten Zeitschritten eine Aufzeichnung möglich und durch die begrenzte (wenn auch sehr hohe) Genauigkeit der Darstellung von Nachkommazahlen sind auch die Messwerte nicht zu 100% kontinuierlich messbar. Grundsätzlich gilt: je größer die Abtastrate ist (also je mehr Messungen pro Zeiteinheit vorgenommen werden), desto genauer ist die Messung - aber auch desto "kostspieliger", denn es fallen mehr Daten an, die gespeichert und verarbeitet werden müssen (und häufig wird die Messhardware deutlich teurer, wenn man hohe Abtastraten erreichen will.)

Für große zweidimensionale Matrizen, bei denen ein hoher Prozentsatz der Elemente Nullen sind, bietet Matlab einen speziellen Datentyp an, die *sparse* Matrizen. Diese benötigen im Fall von hauptsächlich mit Nullen gefüllten Matrizen wesentlich weniger Speicherplatz als normale Matrizen bestehend aus dem Datentyp *double*, bei denen jede *0* mit 32bit Speicherplatz kodiert wird. Bei *sparse* Matrizen werden nur diejenigen Elemente, die ungleich 0 sind, gemeinsam mit ihren Zeilen- und Spalten-Indizes in der Matrix abgespeichert.

Eine *sparse* Matrix wird erzeugt mit dem Befehl *sparse*, z.B.

```
A=[0 0 0; 0 9.5 0; 1 0 0];
S1=sparse(A)
```

ergibt:

```
S1 =  (3,1) 1.0000  (2,2) 9.5000
```

Mit dem Befehl *full* wird eine *sparse* Matrix in eine normale zweidimensionale Matrix des Typs double übersetzt: *F=full(S1)*ergibt die gleiche Matrix, die oben als *A* definiert wurde.

Mit sparse Matrizen lassen sich alle Matrix-Manipulationen (z.B. Indizierung, Anfügen von Zeilen oder Spalten etc) und die meisten Operationen mit der gleichen Syntax ausführen, wie bei den normalen Matrizen (z.B. *e=S1(2,2); S1=[S1; 0 0 7]; S2=2*S1*).

Nützliche Funktionen für den Umgang mit *sparse* Matrizen:

issparse(s)	Test, ob eine Matrix sparse ist
find	finde Elemente ungleich 0 (wie sonst auch)
nonzeros	Werte der Elemente ungleich 0
speye	generiert sparse Identitätsmatrix
spfun	wendet Funktion auf Elemente ungleich 0 an.

sprand(S)	erzeugt eine sparse Matrix mit der gleichen Struktur wie bei Matrix *S* aber gleichverteilten zufälligen Elementen
sprandn(S)	erzeugt eine sparse Matrix mit der gleichen Struktur wie bei Matrix *S* aber normalverteilten zufälligen Elementen
spones	Ersetzt die Elemente ungleich 0 mit einsen
spy	Visualisierung des *sparse*-Musters

T5C1) Lassen Sie eine Parabel zeichnen: *x=-5:1:5, y=x.^2, plot(x,y)*

- Was fällt Ihnen an dieser Parabel auf? Machen Sie die einzelnen Punkte sichtbar.
- Machen Sie die Schritte kleiner und plotten Sie die Parabel erneut.
- Erstellen Sie einen gemeinsamen plot, bei dem die oben erzeugten Werte mit einzelnen Punkten und die Werte für *x_fein=-5:0.01:5* mit einer durchgezogenen Linie in eine Abbildung gezeichnet werden.

T5C2) Wenn eine kontinuierliche Größe gemessen wird, ist die Wahl der Abtastrate wichtig - zu geringe Abtastraten können zu grundlegend falschen Ergebnissen führen. Dieser Effekt soll mit folgendem Skript verdeutlicht werden: [aliasing_effekt.m]

T5C3) Schauen Sie sich den Effekt einmal für echte Daten an: Im Rahmen eines Elektrophysiologie-Praktikums werden Intrazellulärmessungen des Membranpotentials einer Nervenzelle des Blutegels vorgenommen. Die Zelle wird mit einer zeitlichen Auflösung von 10000 Punkten/Sekunde (10kHz) mit elektrischem Strom gereizt. Der Zeitverlauf dieses Stroms (in nA) ist in [stimulus.mat] gespeichert. Die Spannungsantwort (in mV) wird ebenfalls mit 10kHz aufgezeichnet. Sie ist in [spikes.mat] gespeichert. Das Neuron antwortet auf diesen Strompuls mit einer Depolarisation des Membranpotentials (also mit positiveren Spannungen), die mit sogenannten "spikes" überlagert ist. (Mit diesen immer ähnlich aussehenden Ausschlägen der Spannung kommunizieren Nervenzellen miteinander). Die Daten wurden mit einer Frequenz von 10000 Datenpunkten pro Sekunde aufgenommen.

- Laden Sie die Dateien ein und plotten Sie sie übereinander in zwei *subplots* einer Abbildung.
- Passen Sie die Zeitachsen in den Plots so an, dass Sekunden dargestellt werden und beschriften Sie die Achsen.
- Erzeugen Sie einen Vektor, in den nur jeder 10. Punkt des Vektors spikes übernommen wird (dies entspricht einer Aufnahme mit 1kHz).
- Erzeugen Sie einen weiteren Vektor, in den nur jeder 100. Punkt des Vektors spikes übernommen wird (dies entspricht einer Aufnahme mit 100Hz).

- Öffnen Sie ein neues Graphikfenster, und legen Sie darin 3 untereinander liegende subplot-Fenster an.
- Zeichnen Sie jeweils einen der drei Vektoren mit den Aufnahmen bei 10kHz, 1kHz und 100Hz in die drei Fenster ein.
- Wie unterscheiden sich die Zeitverläufe?
- Benutzen Sie die Lupen-Funktion und die Darstellung einzelner Punkte auf dem Graphen, um sich den Zeitverlauf eines einzelnen Aktionspotentials bei den drei Vektoren anzusehen.
- Schauen Sie sich auch einen Zeitausschnitt an, bei dem "nichts" passiert, also der Messwert um einen festen Wert fluktuiert.
- Berechnen Sie für alle drei Vektoren Mittelwert und Standardabweichung der ersten 200ms der Messung.

T5C4) Erzeugen Sie eine übersichtliche Matrix, in der viele Nullen, aber auch einige andere Elemente enthalten sind.

- Erzeugen Sie unter einem anderen Namen eine *sparse* Matrix aus ihrer ursprünglichen Matrix.
- Probieren Sie aus, ob Sie mit dem Befehl *full* die ursprüngliche Matrix herausbekommen.
- Führen Sie einige Matrix-Operationen sowohl auf der *sparse*- als auch auf der *full*-Version der Matrix aus.
- Überprüfen Sie, ob am Ende beide Matrizen den gleichen Inhalt darstellen.

T5C5) Mit *sparse* Matrizen lässt sich im Arbeitsspeicher und beim Abspeichern in Dateien Platz sparen. Jedoch benutzt Matlab beim Speichern in Dateien ohnehin eine geschickte Art der Komprimierung, so dass sich hier der Unterschied zwischen *sparse* und normaler Matrix nicht so stark auswirkt:

- Matlab Erzeugen Sie eine wirklich große Matrix (mindestens 1000x1000) bestehend aus Nullen.
- Ersetzen Sie an zufälligen Stellen 1% der Nullen mit Zufallszahlen.
- Erzeugen Sie aus dieser Matrix eine sparse-Matrix mit einem anderen Namen.
- Speichern Sie beide Matrizen jeweils einzeln in eine Datei.
- Vergleichen Sie die Dateigrößen der beiden Dateien.
- Erzeugen Sie eine gleichgroße Matrix, die komplett mit Zufallszahlen gefüllt ist. Und erzeugen Sie auch für diese eine *sparse* Matrix-Variante.
- Speichern Sie beide Zufalls-Matrizen einzeln in Dateien.
- Was fällt Ihnen bei den vier Dateigrößen auf?

D) HAUSAUFGABE

T5H1) Eine kleine Übung zum Umgang mit Matrizen und Graphik: Laden Sie die Matrix picturematrix.mat herunter.

- Stellen Sie die Matrix mit *imagesc* graphisch dar.
- Kopieren Sie nur den Teil, der das Haus enthält, in eine neue Matrix und stellen Sie es in einem neuen Graphikfenster dar.
- Erzeugen Sie eine dritte Matrix, die nur einen der Sterne enthält und stellen Sie auch diese in einem neuen Fenster graphisch dar.
- *) Warum ändert der Stern seine Farbe? Wie könnte man das verhindern?

T5H2) ABGABE! 4 Punkte In Ihrem Aquarium wurde die Messreihe [phWerte.mat] des Aquarienwassers vorgenommen.

- Ihre Fische vertragen den Bereich von pH 6.5 bis pH 7.5 gut, sind aber darüber und darunter gefährdet.
- Wenn Sie sich die Daten ansehen, werden Sie sehen, ob es in diesem Datensatz offensichtliche Messfehler gibt. Überlegen Sie sich eine gute Methode, diese automatisiert zu finden.
- Plotten Sie die ph-Werte so, dass sie die Messwerte innerhalb und außerhalb Toleranzbereichs sowie die Messfehler mit Symbolen in drei unterschiedlichen Farben darstellen.

T5H3) ABGABE! 6 Punkte Wie oben handelt es sich bei [spikedaten_kurz.mat] um intrazelluläre Messungen des Membranpotentials eines Blutegelneurons. Es wurden 10 Antworten dieser Zelle auf einen jeweils gleichen Reiz aufgezeichnet, einen Strompuls, dessen Zeitverlauf (in nA) im gleichzeitig gespeicherten Vektor *stimulus* abgelegt ist.

- Schreiben Sie ein Skript, um sich alle Antwortspuren gemeinsam mit dem Zeitverlauf der Reizung anzusehen (das darf auch nacheinander sein, aber ein menschlicher Betrachter sollte die Reizung und die Antwort sinnvoll aufeinander beziehen können).
- Legen Sie eine Schwelle fest, mit der sich die Aktionspotentiale finden lassen. (Falls Sie keine Idee haben, wie man das automatisch machen könnte, legen Sie "per Augenmaß" einen Wert fest.)
- Achtung: jeweils am Ende des Strompulses gibt es in der Antwortspur ein Artefakt. Dieses wollen wir möglichst nicht als Aktionspotential erkennen.

**T5H4) Bonuspunkte möglich: Schreiben Sie eine Funktion, die als Eingabeargumente die Matrix mit den Messdaten der letzten Aufgabe und einen Schwellwert bekommt und als Ausgabe einen Vektor liefert, wie viele Spikes in den einzelnen Durchläufen jeweils ausgelöst wurden. (Die Schwierigkeit an dieser Aufgabe ist, dass Blutegel-spikes eine zeitliche Länge von mehreren Millisekunden besitzen aber jeweils nur einmal gezählt werden sollen).

*T5H5) Benutzen Sie Ihre eigene Lösung zur Aufgabe T4H7 um eine Matrix gefangener Vögel zu erstellen. Sortieren Sie diese so, dass, zuerst die Amseln, dann die Rotkehlchen und zuletzt die Meisen in der Matrix stehen, wobei innerhalb dieser Gruppen jeweils zuerst die Männchen und dann die Weibchen aufgelistet werden. Die Tiere einer Art und eines Geschlechts sollen jeweils nach dem Gewicht sortiert sein.

*T5H6) Üben Sie das Suchen in Daten, indem Sie Ihre Lösung für das Wunschkatzen-Programm von Tag 3 (T3H4) so erweitern, dass es die Daten (Geschlecht, Alter, Farben) sämtlicher gefundener Wunschkatzen ausgibt. Sortieren Sie diese Angaben nach dem Alter.

*T5H7) Für Graphik-Freunde: Stellen Sie die Kuchengraphiken aus T5B3 in den zu den Parteien passenden Farben dar. (Dazu werden Sie in die Hilfe sehen müssen...)