**Tag 5: Suchen, Sortieren und Zufallszahlen**

Version vom 4.10.2012

**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(X)* % gibt die Indizes aller Element des Vektors X zurück, die ungleich 0 sind
* *ind = find(X, k)* % gibt die ersten k Elemente des Vektors X zurück, die ungleich 0 sind
* *ind = find(X, k, 'last')* % gibt die letzten k Elemente des Vektors X 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,v] = find(M, ...)* % gibt Zeilen- und Spaltenindizes, sowie die Werte der Elemente der Matrix M zurueck, 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 zurueck.

**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]](http://www.compneuro.uni-oldenburg.de/download/kurs06/fuetterungen.mat" \t "_blank)   
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](http://www.compneuro.uni-oldenburg.de/50246.html), bei dem der gleiche Datensatz auf drei verschiedene Arten durchsucht wird: [[search\_demo.m]](http://www.compneuro.uni-oldenburg.de/download/Matlab_Einfuehrung/2011/downloads/search_demo.m" \t "_blank) benutzt den Datensatz [[VP\_data.mat]](http://www.compneuro.uni-oldenburg.de/download/Matlab_Einfuehrung/2011/downloads/VP_data.mat" \t "_blank)

**B) Zufallszahlen, Wahrscheinlichkeitsverteilungen, Mittelwert und Standardabweichung**

Mit Hilfe von experimentellen Messungen versucht man, allgemeingültige Aussagen zu treffen und Regeln für untersuchte Zusammenhänge aufzustellen. Man variiert einen Parameter (z.B. Menge an Dünger) und beobachtet den dadurch hervorgerufenen Effekt auf eine Messgröße. Dies wäre sehr einfach, wenn grundsätzlich jede Beobachtung immer gleich ausfallen würde, wenn man sie mehrfach wiederholt. In der Realität ist dies allerdings nicht der Fall: Messdaten hängen grundsätzlich zumindest in einem bestimmten Rahmen vom Zufall ab, denn in einem Experiment können niemals alle Zufallsfaktoren ausgeschlossen werden. (z.B. könnte es bei einer Studie über die Wirksamkeit eines Medikaments einen Einfluss haben, wie viel die Patienten geraucht haben oder ob sie gerade Stress hatten.) Auch wenn es eine eindeutige Abhängigkeit zwischen dem variierten Parameter und der gemessenen Größe gibt, werden die Messwerte unterschiedlich ausfallen, sie streuen um den erwarteten Wert.

Wiederholt man ein Experiment in genau gleicher Weise sehr häufig und schaut sich die dabei gewonnen Messwerte an, ergibt sich eine Häufigkeitsverteilung. Diese gibt an, wie häufig ein bestimmter Messwert beobachtet wurde und dient dazu, die Wahrscheinlichkeit dieses Messwertes abzuschätzen.

Für Messdaten (auch künstlich vom Computer erzeugte Zufallszahlen) werden Häufigkeitsverteilungen empirisch ermittelt, um dadurch auf die zugrundeliegende Wahrscheinlichkeitsfunktion zu schließen. Dafür benutzt man Histogramme. Diese teilen den gesamten Wertebereich der Variablen in mehrere Bereiche auf. Das Histogramm gibt für jeden der Bereiche an, wie häufig der Wert der Variable in einer Messung innerhalb des jeweiligen Bereichs lag. Dazu wird für jeden Teilbereich (sogenannte Klassen) ein Rechteck dargestellt, dessen Fläche die gemessene Häufigkeit repräsentiert. In Matlab werden Histogramme mit dem Befehl *hist* erzeugt. Dieser kann vielfältig eingesetzt werden:

|  |  |
| --- | --- |
| *hist(v)* | Teilt den Wertebereich des Vektors *v* in 10 gleich große Klassen ein. Wenn *hist* ohne Ausgabeargument aufgerufen wird, stellt es die Häufigkeit des Auftretens der Klassen als Balkengrafik dar. |
| *h=hist(v)* | Wenn *hist* mit einem Ausgabeargument aufgerufen wird, produziert es keine grafische Ausgabe, sondern gibt den Vektor der Häufigkeiten zurück. (Kann mit mehreren Eingabeargumenten kombiniert werden.) |
| *hist(v,nbins)* | Teilt den Wertebereich des Vektors *v* in *nbins* gleich große Klassen ein. (Mit oder ohne Ausgabeargument verwendbar) |
| *hist(v,centers)* | Benutzt den Vektor *centers* als Mittelpunkte der Klassen, in die die Elemente von *v* aufgeteilt werden. Wenn *hist* ohne Ausgabeargument aufgerufen wird, stellt es die Häufigkeit des Auftretens der Klassen als Balkengrafik dar. (Mit oder ohne Ausgabeargument verwendbar) |
| *[h,xout]=hist(v)* | Wenn *hist* mit zwei Ausgabeargumenten aufgerufen wird, produziert es keine grafische Ausgabe, sondern gibt zwei Vektoren zurück: den Vektor *h* der Häufigkeiten und den Vektor *xout* der Klassenmittelpunkte. (Kann mit mehreren Eingabeargumenten kombiniert werden |

Die meisten biologischen Daten lassen sich durch eine Normalverteilung (auch Gauß-Verteilung genannt)  beschreiben, bei der Messwerte umso häufiger auftreten, je näher sie am Erwartungswert, dem Mittelwert der Verteilung liegen.

Für eine normalverteilte Zufallsvariable x entspricht die Wahrscheinlichkeitsdichte folgender Formel:

ild

wobei μ den Mittelwert und σ die Standardabweichung der Wahrscheinlichkeitsverteilung angibt.

Für normalverteilte Messwerte (oder auch mit einem Zufallsgenerator erzeugte Zufallszahlen) kennt man diese Kennwerte der den Daten zugrunde liegenden Normalverteilung nicht. Man kann sie jedoch näherungsweise aus den Messwerten x1 bis xn ermitteln:

Empirischer Mittelwert:

ild

Empirische Standardabweichung:

ild

Auch wenn es keine schlechte Übung ist, diese Formeln einmal in Matlab umzusetzen, kann man stattdessen auch einfach die Befehle *mean* und *std* benutzen.

**Achtung: Die Berechnung von empirischem Mittelwert und empirischer Standardabweichung macht ausschließlich für NORMALVERTEILTE Werte Sinn!**

Eine andere wichtige Verteilung, die in diesem Kurs auch betrachtet wird, ist die Gleichverteilung, bei der alle Werte in einem bestimmten Bereich mit gleicher Wahrscheinlichkeit auftreten. (Für gleichverteilte Werte ist es absolut sinnlos, Mittelwert und Standardabweichung zu berechnen.)

Bevor wir uns mit der statistischen Auswertung von echten Messdaten beschäftigen, erzeugen wir zunächst einmal selber "Messdaten" mit Matlab, nämlich Zufallszahlen. Diese werden beispielsweise gebraucht, wenn man Experimente plant, in denen bestimmte Reize in zufälliger Reihenfolge präsentiert werden sollen. Eine weitere wichtige Anwendung von Zufallszahlen sind Simulationen biologischer Prozesse. Wenn man Zufallszahlen künstlich erzeugt, ist (im Gegensatz zur Auswertung von Messdaten) die Wahrscheinlichkeitsverteilung bekannt (also die Wahrscheinlichkeit dafür, dass eine Zufallsvariable einen bestimmten Wert annimmt).

Im Rahmen des Kurses erzeugen wir folgende Zufallszahlen:

* *M1=randn(Z,S)* erzeugteine ZxS-Matrix mit normalverteilten Zufallszahlen mit Mittelwert 0 und Standardabweichung 1
* *M2=rand(Z,S)* erzeugteine ZxS-Matrix mit gleichverteilten Zufallszahlen zwischen 0 und 1
* *v=randperm(n)* liefert einen Vektor der ganzen Zahlen von *1* bis *n* in zufälliger Reihenfolge.

**Aufgaben:**

T5B1) Probieren Sie die Funktionen *randn* und rand aus:

* Erzeugen Sie einige Beispiele normalverteilter und gleichverteilter Zufallszahlen: Was passiert, wenn man die gleiche Funktionen mehrfach hintereinander in gleicher Weise aufruft?
* In welchen Bereich liegen die Werte für die beiden Funktionen?

T5B2)  Erzeugen Sie jeweils einen sehr langen Vektor (z.B. 10000 Elemente) mit jeder der beiden Funktionen *rand* und *randn*.

* Schauen Sie sich die jeweilige Verteilung der Zufallszahlen mit dem Befehl *hist* an.
* Was sind die Unterschiede zwischen den beiden Verteilungen?
* Mit *hist(v,n)* teilt *hist* den Vektor *v* in *n* gleich große Bereiche ein. Sehen Sie sich die Verteilungen für verschiedene Werte von *n* an.
* Schätzen Sie aus der Abbildung ab: Was ist der Mittelwert? Was die Standardabweichung?
* Berechnen Sie Mittelwerte, Standardabweichungen, Varianzen, Minima und Maxima Ihrer beiden Vektoren mit *mean, std, var, min* und *max*.

T5B3) Modifizieren Sie Ihre Zufallsvektoren, indem Sie diese

* mit verschiedenen Faktoren multiplizieren
* verschiedene Zahlen hinzuaddieren
* Wie wirken sich diese Änderungen auf die Verteilungen aus?
* Wie wirken sie sich auf Mittelwert, Standardabweichung, Minimum und Maximum aus?

T5B4) Laden Sie den Vektor mit den Anzahlen an Sonnenblumenkernen von 100 Blumen [[sbkerne.mat]](http://www.compneuro.uni-oldenburg.de/download/kurs06/sbkerne.mat" \t "_blank). Berechnen Sie Mittelwert, Varianz und Standardabweichung. Wie habe ich diesen Vektor erzeugt? (Nein, ich habe mich nicht in den Garten gesetzt und gezählt...)

\*T5B5) Setzen Sie die oben angegebenen Formel für die Wahrscheinlichkeitsdichte einer Normalverteilung in Matlab um.

* Schreiben Sie eine Funktion, die drei Eingabeargumente bekommt
  + einen Vektor, der den Definitionsbereich angibt, z.B. *x=-4:0.01:4*,
  + den gewünschten Mittelwert und
  + die gewünschte Standardabweichung
* Als Ausgabeargument soll die Funktion die berechnete Wahrscheinlichkeitsdichteverteilung als Vektor zurück geben
* Die Funktion soll die Wahrscheinlichkeitsdichte außerdem als Kurve grafisch darstellen (bitte mit Titel und beschrifteten Achsen).
* Variieren Sie die Parameter Mittelwert und Standardabweichung. Wie verändern diese die Kurve?

T5B6) Schreiben Sie eine Funktion *wuerfel*, die Ihnen jeweils eine ganze Zahl zwischen 1 und 6 zurückgibt.   
  
T5B7) Benutzen Sie diese Funktion in einer weiteren Funktion *wuerfel\_verteilung*, die als Eingabewert bekommt, wie oft gewürfelt wird, und als Ausgabe die Verteilung (als in einem Vektor gespeichertes Histogramm) der erzielten Würfelergebnisse zurückliefert.

**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 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]](http://www.compneuro.uni-oldenburg.de/download/kurs06/aliasing_effekt.m" \t "_blank)

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]](http://www.compneuro.uni-oldenburg.de/download/Matlab_Einfuehrung/2011/downloads/stimulus.mat" \t "_blank) gespeichert. Die Spannungsantwort (in mV) wird ebenfalls mit 10kHz aufgezeichnet. Sie ist in [[spikes.mat]](http://www.compneuro.uni-oldenburg.de/download/Matlab_Einfuehrung/2011/downloads/spikes.mat" \t "_blank) gespeichert. (Die schnellen, stereotypen Änderungen der Membranspannung, mit denen Nervenzellen kommunizieren, nennt man Aktionspotentiale oder auf Englisch spikes.)

* 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) Hausaufgaben:**

T5H1) Im Programm [[vogelfang.m]](http://www.compneuro.uni-oldenburg.de/download/kurs06/vogelfang.m" \t "_blank) werden drei verschiedene Arten von Zufallszahlen benutzt. Vollziehen Sie dieses Programm nach.   
Nehmen Sie schrittweise folgende Änderungen vor:   
a) Bei Amseln gibt es 60% Weibchen.   
b) Bei Spatzen streut das Gewicht von Weibchen 3 Mal mehr als das Gewicht von Männchen.   
c) Es kommen 25% Meisen und 25% Spatzen in der Gegend vor.   
  
T5H2) In einem psychophysikalischen Experiment sollen einem Versuchstier drei verschiedene Töne in zufälliger Reihenfolge vorgespielt werden, aber jeder Ton soll genau 5 Mal vorkommen.

* Wir kümmern uns erstmal nicht um die Generierung der Töne, sondern nennen sie einfach Bedingung 1, 2 und 3.
* Überlegen Sie sich einen Algorithmus, der die Reizbedingungen in die richtige Reihenfolge bringt und setzen Sie diesen in ein Programm um.
* Testen Sie das Programm, indem Sie es mehrfach laufen lassen. Macht es immer, was es soll? Sind die Ergebnisse jedes Mal gleich?
* Erweitern Sie Ihr Programm so, dass es N (eine beliebige Anzahl) Reize, die M mal (also beliebig oft) vorgespielt werden sollen, in eine Reihenfolge bringt.
* Tipp: Benutzen Sie für diese Aufgabe die Funktion *repmat*. Diese erzeugt eine große Matrix durch mehrfache Wiederholung einer kleineren. Z.B*. B = repmat(A,2,5)* erzeugt eine Matrix *B*, in der insgesamt 10 Kopien der Matrix *A* enthalten sind, wobei *A* zweimal untereinander und fünfmal nebeneinander angeordnet wird. (*B* hat also die doppelte Zeilen- und fünffache Spaltenzahl von *A*.)

\*T5H3) Die Messwerte einer Apparatur ist selbst ohne biologisches Präparat nicht perfekt rauschfrei. Um das Geräterauschen abzuschätzen, wurden im Elektrophysiologie-Praktikum für die Apparatur mit einer Modellzelle (einem elektronischen Schaltkreis, der die Membraneigenschaften einer Nervenzelle nachbaut) 100 Messungen mit dem gleichen Reiz [stimulus1khz.mat] durchgeführt und die Antworten als Matrix unter [[antworten1khz.mat]](http://www.compneuro.uni-oldenburg.de/download/kurs06/antworten1khz.mat" \t "_blank) abgespeichert.

* Schauen Sie sich eine beliebige einzelne Messung zusammen mit dem Reiz an (entsprechend Aufgabe C3).
* Berechnen und plotten Sie in ein neues Grafikfenster den Zeitverlauf der über die 100 Messungen gemittelten Antwort.
* Berechnen und plotten Sie in ein neues Grafikfenster den Mittelwert und die Standardabweichung der jeweils letzten 300ms für jede Messung (Mittelung über die Zeit). Gibt es eine Tendenz? Gibt es Ausreißer?
* Berechnen und geben Sie als Textausgabe im Kommandofenster aus: Sind Mittelwert und / oder Standardabweichung vor, während und nach der Reizung unterschiedlich?

\*T5H4) Üben Sie das Suchen in Daten, indem Sie die Musterlösung [wunschkatze\_jutta.m](http://www.compneuro.uni-oldenburg.de/download/kurs09/tag5/wunschkatze_jutta.m" \t "_blank) (oder Ihre eigene 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.

\*T5H5) Benutzen Sie die Musterlösung [vogeltabelle\_insa.m](http://www.compneuro.uni-oldenburg.de/download/kurs06/vogeltabelle_insa.m" \t "_blank) (oder 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
* dabei sollen 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) Noch eine kleine Grafik-Übung: Nehmen wir an, Sie haben ein paar besonders empfindliche Fische in Ihrem Aquarium, in dem die Messreihe  [[phWerte.mat]](http://www.compneuro.uni-oldenburg.de/download/kurs06/phWerte.mat" \t "_blank) vorgenommen wurde.

* Diese Fische vertragen den Bereich von pH 6.5 bis pH 7.5 gut, sind aber darüber und darunter gefährdet.
* Definieren Sie außerdem in der Messreihe die Abweichungen um mehr als 2 Standardabweichungen als Messfehler.
* Plotten Sie die ph-Werte so, dass sie die Messwerte innerhalb und außerhalb Toleranzbereichs sowie die Messfehler mit Symbolen in drei unterschiedlichen Farben darstellen.

\*T5H7) Die Funktion *hist* kann man auch auf Matrizen anwenden, dann  wird jede Spalte als Histogramm dargestellt. Erzeugen Sie eine Matrix mit zwei gegeneinander verschobenen Verteilungen und stellen diese grafisch dar.