Project proposals

Rules for and grading of programming project

- **Master students** of the Neural & Behavioral class must work on and present a programming project; the project will be graded (not the presentation, see table below). Some degree of assistance and coarse preview of projects before submission by instructor (upon and in order of requests).
- All **other participants** must do the following **in order to get credits for the course:** work on and submit a project, and engage in peer review (see form_projectReview.docx). No assistance before submission. Peer review may be swapped for a presentation (to be negotiated with instructor).
- Master students may *not* swap the presentation for code review, but may additionally perform code review and thus earn bonus points.
- The projects may be one of the proposals listed below or one of your own ideas. Projects based on your own ideas are very welcome!
- Your programming job will be evaluated according to the following criteria:

Criterion	**Max score**
Complexity and scope of project ('difficulty and programming effort')	8
Functionality (program accomplishes job described in project description)	10
Readability of code: compartmentalization, variable names, documentation (help text for function, particularly description of input & output variables, and comments throughout code)	10
Efficiency (speed, usage of memory, vectorized code, preallocation, etc.)	8
Versatility of code: input & output arguments and error-checking and -handling	10
Code review	up to 4
Sum	46+4

- Please note: according to the standard grading scheme only projects with a complexity score of 6 and above can possibly result in the best grade (1.0, provided that all other scores are maximal and no code review has been submitted). Nonetheless: a simple, but well written and documented program may score as well as or even better than a sloppily written complex program.

Guidelines for code

- The 'core' code must reside within a central function, plus other functions, if needed. This function may be called by a script for the presentation (see below). Please give your central function a telling name, not something like 'main.m' or 'start.m'. **Central code in scripts will not be accepted.**
- The central function shall take input arguments and possibly produce output arguments to make the code versatile. For example, the spike threshold detector would certainly need a data array (or a data file name) and a threshold as inputs and return time stamp lists. Checks of input variables must be performed and some precautionary measures against nonsense input should be taken. Try to catch common errors.
- Please do not use Matlab function `input` because it combines the disadvantages of two philosophies of managing user input: it will not allow your code/function to run without user interaction, and the kind of interaction it allows is very limited. In most instances, input

arguments to functions are much better. If your project requires interactivity, consider using pre-fabricated GUIs like `inputdlg, uigetfile`, etc. or even uicontrols (e.g. of the listbox type)

- The purpose of the function and the nature of the input and output variables must be well explained in the help section ('H1 line') of your code (e.g. the size of variables, data type, what kind of data etc.). The function definition (as in e.g. `s=load(filename)` must appear somewhere in the help section.
- Ample documentation of the code (as Matlab comments) should be provided. Place comments on top of the code lines they are supposed to comment, not in the same line as the code itself, and please, please don't let your comment lines overflow a reasonable number of columns (ideally somewhere between column 70 and 80) because this forces me to scroll the screen horizontally all the time, in other words, your comments unnecessarily hard to read.
- In general, make your code as readable as possible. Please try to adhere to a few guidelines, e.g. concerning the naming of variables, as laid out in the **Matlab Programming Style Guidelines** by R. Johnson (\mcourse\doc\MatlabProgramStyleGuide.pdf).
- Please also take a close look at 'Good and bad code examples.pdf' in the \doc directory, which presents two pieces of code which accomplish more or less the same task but differ vastly in readability and partly also in versatility.
- **The points above may be summarized as follows: your final code shall be versatile in the sense that it can handle, without errors, a wide range of input data or situations in which it may be used in real life, and it should be comprehensible by others (e.g. your instructor).**
- Pay attention to efficiency. Use vectorized code where possible and preallocate large variables. If you need to generate large intermediate variables delete them as soon as they are not needed anymore. Think about a memory-saving sequence of computational steps.
- If you make use of any of the diverse Matlab toolboxes, make sure it is also on the Gradschool's laptop used for the presentation (or bring your own laptop).
- In terms of complexity and workload the projects are designed for teams of two. In case you work as a team of three you have to incorporate a few more features (which we will decide upon in individual negotiations)
- Make sure the data which your program will digest or generate during the presentation is of a manageable size so that your demo computations finish in finite time.
- For some of the projects there is already Matlab code around in the internet. Although disseminating code produced by others is in fact a great way to learn Matlab, it goes without saying that the task is NOT to do a simple copy-and-paste job. Please spare me the embarassment of detecting large chunks of verbatim copies of publicly available code (or, for that matter, code from the previous semesters' course) in your solutions.
- For copyright reasons, do not modify code which is part of Matlab, or copy verbatim code snippets, and use such code for your project. Distributing such code is a culpable act.

Presentation of projects

- In order to get credits all members in a team have to share duties during the presentation. **Unannounced no-shows at the project presentation will not get any credit for the course.**
- You have 20 min time, 3-5 min of these should be reserved for questions by others at the end of the presentation. If you can do it in less time, all the better
- **In order to save time it is recommended that you write a short script (m-file) in which input variables are predefined or loaded and the your function(s) is/are called**
- Although this is not a test of your rhetorical skills, please try to be concise

- Depending on the project, you should illustrate the following points
    - scientific background
    - clear delineation of the purpose of the code produced
    - short demonstration of functionality/output
    - overview of code
- If more than one team works on the same project, please coordinate your presentations such that there is little redundancy, that is, the first team could do a thorough introduction/scientific background (if any) whereas the other(s) place more emphasis on the results. That notwithstanding, each team must present their individual code!

Schedule & deadlines

- Any team may switch to a different project post-assignment until the date specified below. **After this date, switching projects and particularly changing teams is not acceptable**.
- I need to have your **project files (i.e. the Matlab code plus data, if any) mailed to me in time** (see schedule below). Late submissions will be penalized. Please compress any voluminous files you attach. If the size of your code and data files exceeds 10 Mb, please make them available to me by other means (download, memory stick, CD).
- I will not accept shrapnels of code in emails that would require me to edit your m-files (e.g. for checking preliminary versions of your code, or last-minute corrections). Any code you send must be attached as complete m-files.

April 13	assignment of teams to projects
April 19	last chance to switch projects or teams
April 26, midnight	submission deadline (submission after this date: 4 points will be subtracted)
April 27	project presentation (submission after the presentation: 8 points will be subtracted)
May 7, noon	non-negotiable deadline for submission of projects (any submission after this date will not be considered)

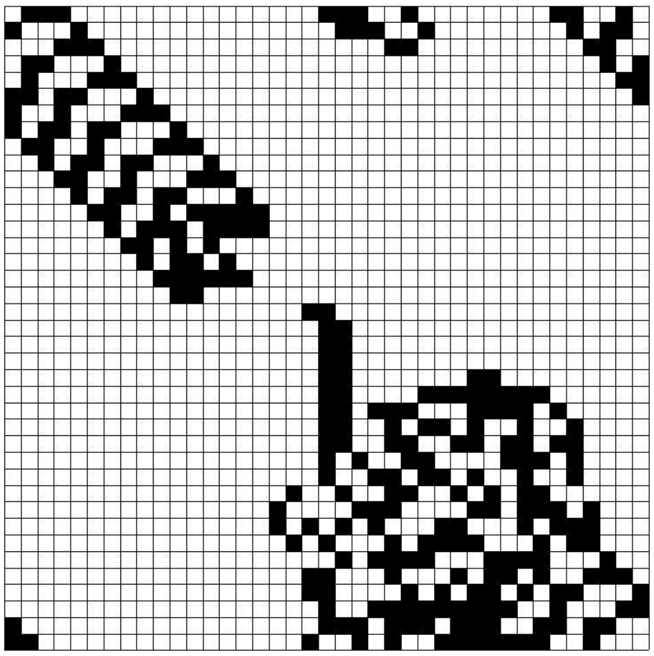
Characterization of the projects

- 4 – 8: programming effort (corresponds to the score for complexity and scope mentioned above)
- + - +++: difficulty of underlying science/analysis concepts

I. Projects of the playful sort

Langton’s ant (7,++)

Imagine a pattern of square 'cells' arranged on a regular lattice in a in 2-dimensional 'world' as in the figure below. There are black and white cells. One of the cells is an 'ant'. The ant moves around and changes its environment by flipping the color of the cells in its path according to the following rules:
1. If it is on a white square, it flips the square’s color to black, then turns right (90°) and advances by one square
2. On a black square, it flips the color to white, turns left (90°) and advances by one square
These simple rules may lead to really surprising patterns!
Details:



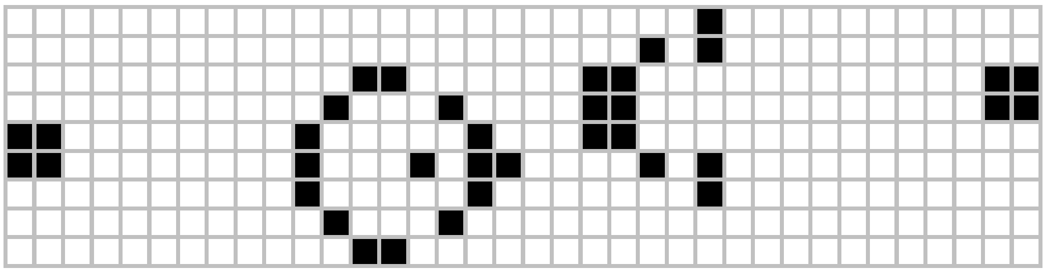
- your program must be able to load (or by other means be initialized) with several test patterns, including an all-white world, a random pattern and a pattern with a filled rectangle (or triangle or square or the like) in the center of the world. There are other interesting starting conditions you may want to include; please consult Wikipedia or other sources.
- boundary conditions are free – the grid may actually be a torus or have impenetrable borders – choose whichever you like.
- your program should run either for a user-defined number of generations or – preferably – indefinitely and be stoppable by the user by some kind of programmed interaction without producing an error. Neither Ctrl-C, which halts running Matlab code but produces an error, nor usage of the pause button in the editor count!

Conway's Game of Life (7,++)

Imagine a pattern of square 'cells' arranged along a regular lattice in a in 2-dimensional 'world' as in the figure below. Some cells are 'alive' (black), others are 'dead' (white). The pattern is not fixed, but shall evolve over time (which is discrete). The rules as to how the pattern changes from one time step to the next are very simple:
1. Any live cell with fewer than two neighbours dies, as if by loneliness.
2. Any live cell with more than three neighbours dies, as if by overcrowding.
3. Any live cell with two or three neighbours lives, unchanged, to the next generation.

4. Any dead cell with exactly three neighbours comes to life.
Please consult e.g. Wikipedia for further details. For an example of a (fairly advanced) implementation you may want to take a look at Matlab function `life`. Details:

- your program must be able to load (or by other means be initialized) with test patterns the behavior of which is known. There are stationary patterns, oscillators, 'gliders', 'guns' (like 'Gosper's Glider Gun pictured below), and so on. Implement at least three of them.
- boundary conditions are free – the grid may actually be a torus or have non-reactive borders (that is, it is surrounded by dead cells that never come alive) – choose whichever you like.
- your program should run either for a user-defined number of generations or – preferably – indefinitely and be stoppable by the user by some kind of programmed interaction without producing an error. Neither Ctrl-C, which halts running Matlab code but produces an error, nor usage of the pause button in the editor count!



pcTunes (8,+++)

Music is a cosmos. A piece of music can be joyful or melancholic, fast or slow, monotonous or of intricate complexity, and so on. Sometimes, I would love to listen for hours to music of the same mood, but how shall I choose the right set of songs among my collection of thousands of tunes? Here is the idea: build a database in which songs are characterized according to a fixed set of criteria (probably more than just the few exemplary ones listed above). Then, write some code which identifies songs that are 'similar to each other'. Obviously, this is not an easy task because of the many different qualities of music. In other words, it is a multi-dimensional problem. Fortunately, there are dimension reduction techniques like principal components analysis (PCA). Imagine all your music pieces exist in an N-dimensional space, N being the number of characteristics of the music pieces. With PCA you can compute the axis (within the N-dim space) along which the variance of your songs is maximal. This axis is the first principal component. The second principal component is orthogonal to the first and oriented such that it maximizes the remaining variance (as good as its gets with the restriction of orthogonality). And so on, up to N. Then, plot the songs in the 2-dimensional space defined by two principal components (almost certainly the first plus the second or third). The spatial arrangement of the songs in this plot should reflect their (dis-)similarity better than a 2-dim plot of one of the original characteristics versus another (say, tempo vs. 'timbre').
The concrete task:

- generate a data base of at least 30 songs with at least four characteristics as mentioned above
- the main code must compute the *principal component coefficients* and *component scores*. You may use function `pca.m` which is part of the statistics toolbox, to that end.
- a 2-dim plot of the component scores must be produced. Each song must be identifiable via a mouse click on the symbol it represents in the plot (see exercise 9.1 for a solution). That is, the band name and title of the song should pop up somewhere, and a short (!) excerpt of the song must be played.

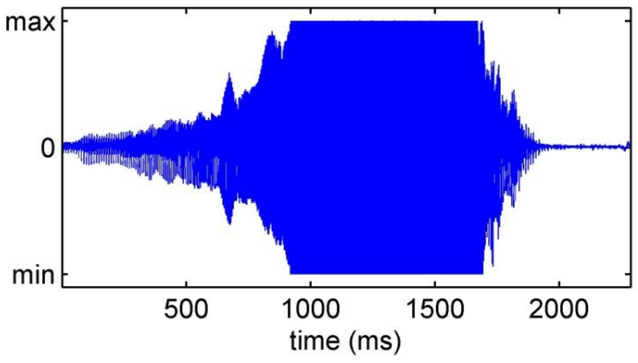
- the user should have a choice of plotting the songs (component scores) for all non-redundant combinations of the first three principal components
- the user must be informed to what degree each PC represents the different qualities by which you rank the various pieces of music. For example, the first PC may be dominated by the characteristic 'speed' because your data base contains songs which vary extremely in this regard. The key to this is the first output argument of pca.m, the component coefficients. Pick the two or three most dominant characteristics and place the information somewhere on the plot, ideally as axis labels, or some other way.

General comments

- you may be familiar with internet sites which let you create personalized music channels. You choose a song or artist you like, and in the following songs in the same category are played. Although the algorithms are diverse and different from what you will produce (I doubt any of them are written in Matlab…) the principal idea is similar.
- you'll have to have (or acquire) a solid understanding of PCA to succeed. I recommend consulting the (very good) Matlab help on principal components analysis
- also, a basic knowledge of graphical user interfaces is required
- this project lends itself to cooperation among teams. If there is more than one team working on this project I would encourage you to agree on a common format of the data base so that you can combine your efforts into a larger data base.
- in developing the code you will have to make a number of partly arbitrary decisions, e.g. whether to normalize the original data before subjecting them to PCA, which categories to choose etc.. Experiment!
- you will probably generate the song data base by listing all the information in a spreadsheet or possibly an m-file. The organization of this data base is crucial. Try to use a variable type which is best suited to 'bundling' the aspects of songs (title, the characteristics described above, etc.). Think of 3000, not only 30 songs, and how you would best succeed in keeping order.
- there have been attempts by groups in the past to find objective criteria to characterize songs, e.g. by computing the frequency power spectrum. You may choose to do so instead of subjectively characterizing the songs, but be warned that funny results likely emerge unless you pre- and/or post-process the data. In this context: the idea of using PCA of objective criteria to describe music has recently been implemented professionally and put into a very enjoyable article (Mauch et al. 2015, in the \projects directory) – peruse it for inspiration, if you like!
- disclaimer: this is not an instigation to use illegally downloaded music!

Loudness war (7, +)

You may have come across the phenomenon that the louder a sound or vocalization, the more attention the emitting agent will usually get. This seems to hold not only in biology (think of frog calls) but also in the music industry: some time in the 1980s the music industry realized that more fun (=sales, from their perspective) can be had with 'louder' CDs: if you have your stereo at a neighbor-friendly volume but nonetheless get blown away by the sheer power of the sound, it seems to make a lasting impression. The trick is to elevate the average level of loudness on the CD. In principle, no objections against that. In practice, though, the problem is that any recording medium or format including CDs, to which we will limit the current discussion, has a limited dynamic range: the span from very silent to very loud that can be adequately represented is limited. So, if the average level

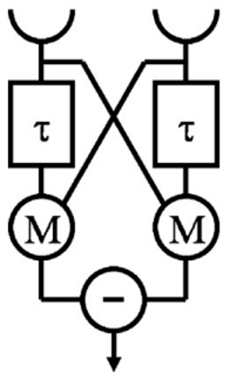


of loudness is more and more inceased, the loudest parts will suffer, and the music will sound distorted. To 'suffer' in digital media means to be clipped (see figure. There is excellent treatment of the topic on the web, including Wikipedia).
Your task is to write a program which simulates and demonstrates the adverse effects of amplifying a signal (=piece of music) beyond the limits of the recording medium. Your program must accept as an input audio files (minimally wav files, possibly others as well) and some measure of how severely the signal is to be clipped (among other input, if necessary). It should then display power spectra of the original and the clipped signal side by side so we can see the effects of clipping, as well as play both versions of the signal one after the other.
Notes:

- audio data can be loaded into Matlab and played e.g. with function `audioplayer`
- make sure that the music pieces to be compared during the presentation are not too long
- you should be familiar with the concept of spectral analysis to some degree
- there is a range of matlab functions performing spectral analysis. `spectrogram` and `periodogram` are among the functions which are useful

II. Models in science

Reichardt motion detector (6-8,++)



The Reichardt motion detector is a simple but elegant model of a motion detection circuit, originally described half a century ago. The essential idea is that the outputs of two photoreceptors are each lowpass filtered, introducing a delay, and then multiplied with the unfiltered signal from the other receptor before being subtracted from each other (see figure on right, taken from A. Borst & T. Euler, Neuron 71, 974-994, 2011).
Your task is to implement such a motion detector and associated code. First, you need to write a function which generates time series (two-column arrays) that represent the light intensities which left and right receptors perceive when light sources move in front of the detector. The light sources and motions shall be i) single point sources of light moving from left to right or vice versa with user-specified levels of speed, and ii) at least one more complicated movement of one light source (e.g. oscillatory) or the movement of more than one light source - use your imagination! The time series should represent the motion in a qualitative way; they need not be correct physical representations of the changes in light intensity (e.g. a clan of fireflies traveling in single file across the detector could be represented by the sine function plus an offset).
The function coding for the detector proper must take as an input argument such a time series (a two-column array) and the cutoff frequency of the lowpass filters, represented by tau in the illustration. The function's output must be the output of the motion detector (a time series of the same length as the input), plus an interpretation of it (e.g. messages like 'movement left', 'movement right', 'no movement', 'multiple movement directions'). The function should be able to run in two different modes: i) 'demo' mode, in which the signals at each stage of the detector (raw input, after filtering, after multiplication, final output) are visualized, and ii) in a 'fast' mode which does no plotting at all and which is optimized in terms of speed and memory usage. This is the basic version (6 points). The project is worth 8 points if you additionally enhance the function producing the two-column time series of light intensities at the receptors: it is supposed to also generate animated graphics of the underlying motion stimuli.

Forest fires (8,++)

Implement a simple model of forest fires as featured in Malamud et al (1998). The program should be able to run in two different modes:

i) Slow mode featuring an animation of the model. That is, one should see the 'forest' growing before a fire, the fire spreadig during a fire, a gap in the forest where trees were consumed, and so on. (If you feel really ambitious, you may try to give the animation a realistic touch, with a 'fire front' leaving behind burned stumps in the middle. This is a suggestion, not a requirement!)

ii) Fast, number-crunching mode without animation to verify the principal statement in the paper, namely, that the frequency of occurrence of fires versus versus fire size (=number of trees burned) follows a power law. The program should produce a double logarithmic plot as featured in the paper. You need not verify all statements in the paper, just pick one set of parameters and show a single distribution.

Malamud B, Morein G, Turcotte DL (1998) Forest fires: an example of self-organized critical behavior. Science 281: 1840-1842.

Side note: the topic of 'self-organized criticality' is also a topic in neuroscience! See e.g. Beggs & Plenz 2003 and 2004 (Journal of Neuroscience) as well as many follow-up studies, in which the phenomenon of 'neuronal avalanches' in cortical networks in vitro is described.

Ants (again)! (8,++)

Implement the model of ant colony clustering as described by Vandermeer et al 2008. Like the Game of Life, Langton’s Ant and the forest fire model above the model is a cellular automaton, that is, a number of 'cells' living on a two-dimensional grid which change (in this case, cluster) according to a set of simple rules (note that the model description and parameters you'll need are somewhat hidden in the paper, mostly in the very last paragraphs of both the main paper and the supplementary information). The program you develop should feature graphical animation of the development of ant colonies as well as depictions of the population density (as in Figure 3, top) and the distribution of nest cluster sizes (as in Figure 4b). Note that because the fine detail of the implementation is not given in the paper, the results of your implementation will likely not reproduce all quantitative results given in the paper (particularly the population density given in Figure 3). However, the qualitative behaviour of the model (clustering, stable populations sizes depending on parameter choices) must of course be reproduced.

Vandermeer J, Perfecto I and Philpott SM. Clusters of ant colonies and robust criticality in a tropical agroecosystem. Nature 451:457-460, 2008

On the occasion: a quick primer for quick animations in Matlab

A number of the projects proposed here require some animation, that is, repeated graphical display of data. The most obvious way to accomplish this is to call a plotting function within a loop:

```
for g=1:nRun
  % compute or update data d
  d=someMagicComputation(whatever);
  % plot
  plot(d);
  drawnow
```

```
end
```

The loop is inevitable, as is the `drawnow` command for fast-running loops (otherwise, Matlab will race on with the computations and not update graphics). However, the `plot` command (or `spy`, `image`, whichever it is) is suboptimal: all plotting functions have a more or less significant 'overhead', householding tasks they have to accomplish when called. To use a silly analogy, it's like opening a new bank account each time you want to transfer money. There is a better way: use the `set` command and update the xdata and/or ydata properties of your plot:

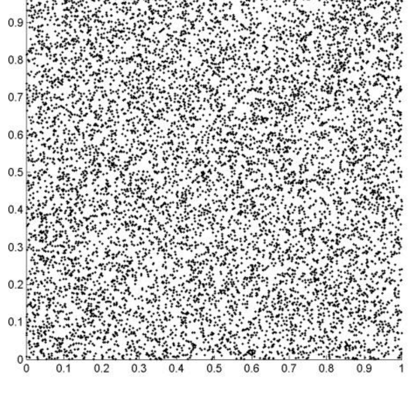
```
% initialize plot, returning handle
ph=plot(d)
for g=1:nRun
  % compute or update data d
  d=someMagicComputation(whatever);
  % update plot
  set(ph,'ydata',d);
  drawnow
end
```

The code above is just an example – **depending on the plot type the syntax is different (e.g. in the case of imagesc, which is the function which lends itself to many projects proposed here), but in all cases you will be rewarded with better performance**.
Last point in this context: the legend and colorbar functions consume an awful lot of computational power – do not use them (or other seemingly simple graphics functions) within a loop unless it is essential for the task at hand. If you are unsure, use the Matlab profiler – it will point to bottlenecks in your code.

Structure from motion (7,++)

Write some code qualitatively replicating the structure-by-motion stimulus used in Lamme et al. 1998. The stimulus is a random dot pattern in which a central square region of dots moves in one direction (by a tiny amount and for a very brief time) and the background of dots moves in the opposite direction, leading to the percept of a square hovering above the background. The figure displaying the animated graphics should feature some GUI elements, minimally a 'start' or 'go' button, and one uicontrol each for the amount and duration of movement.
Notes:

- don't even think of using the code you produce for real psychophysical experiments. For real experiments you will need dedicated hardware and software (for example the psychophysics toolbox for Matlab). The timing precision of graphics in Matlab (and all other software not specifically dedicated to that purpose) is far too poor. Nonetheless, for a qualitative impression it will work.
- generating a proper stimulus is more difficult than may appear at first sight: the task is not simply one of shifting dots. You will e.g. have to take care of the 'gaps' that arise from shifting the central square of dots, etc..



Lamme VAF, Zipser K and Spekreijse H. Figure-ground activity in primary visual cortex is suppressed by anesthesia. PNAS 95:3263-3268, 1998

III. Data analysis

Weather analysis (4-6,+)

Your job is to visualize meteorological data that come in a specific format. File *weather.mat* contains data from three weather stations in Germany collected from January 1991 to October 2007 (freely available at the website of *Deutscher Wetterdienst*). They are in three variables, called `hamburg`, `aachen` and `stuttgart`. Each of them contains 14 columns as described in the table below. Each datum in columns 4-14 is a monthly measure (mean, maximum, minimum or sum) of the parameter in question.

column	**Shortcut used by weathermen**	**Meaning**	**Unit**
1	-	numerical code for station (ignore)	-
2	-	date (yyyymm, e.g. 199204 means April 1994)	-
3	QN	quality level	arbitrary (1-10)
4	TNN	minimum of daily temperature minima	°C
5	TNM	mean of daily temperature minima	°C
6	TMM	mean temperature	°C
7	TXM	mean of daily temperature maxima	°C
8	TXX	maximum of daily temperature maxima	°C
9	SOS	sum of sunshine hours	hours
10	NMM	mean cloud amount	one eighth
11	RSS	sum of precipitation	mm
12	RSX	maximum of daily precipitation	mm
13	FMM	mean wind strength	Beaufort
14	FXX	maximum of wind strength maxima	m/s

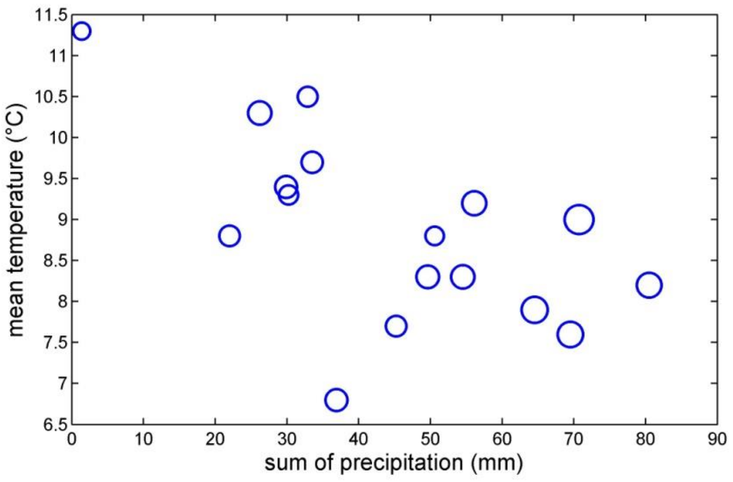
You should write a function which lets the user select any of the three weather stations and then performs the following:

- create a `boxplot` (with notches) depicting the median and monthly variation of one parameter (that is, we want to have twelve boxes in the plot, one for each month). The user should be allowed to choose any of the parameters that make sense for such a plot (that is, all except the first three). Do not forget to label the axes properly.

This is the basic version of the task (4 points).

If you additionally implement the following extension the score will be 5 points:

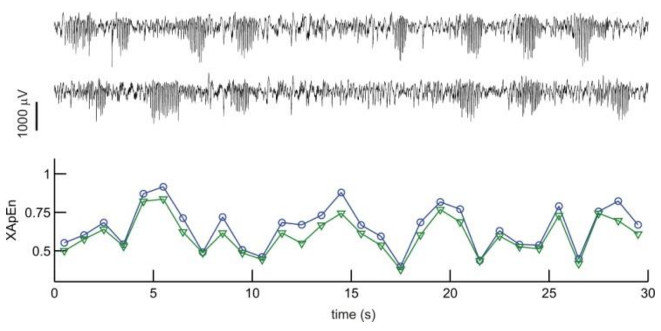
- scatterplot of one parameter versus another for one station and month (all years). A third parameter shall be coded for in terms of the size of the plotted symbols (this is called a 'bubble plot'). An example of such a plot is shown in the figure below (the size of the circles codes for the mean cloud amount). Again, the user must be able to choose all three parameters



With yet another extension the score will be 6 points: your code shall compute the linear regression between the two parameters in the scatter plot (the two axis parameters), add the regression line to the plot and put out the regression parameters (e.g. as text in the plot or on the command line)
Note: the code you produce must be able to handle data not just for the specific time period represented in the data sets provided, but also for longer periods.

Cross approximate entropy analysis of time series (8,+++)

You may be familiar with cross correlation (XC) as a means to assess the similarity of a pair of time series, e.g. two EEG traces sampled from two electrodes. However, XC is only one of a number of diverse methods/algorithms to that end. Another, relatively recently introduced method is cross-approximate entropy (XApEn), which quantifies the predictability of the appearance of common patterns in two time series.



Your task is to implement XApEn analysis for time series (like EEG data). The algorithm is described in these two papers (in directory \mcourse\projects\papers; Hudetz 2002 may be easier to comprehend):

- Pincus SM, Mulligan T, Iranmanesh A, Gheorghiu S, Godschalk M and Veldhuis JD. Older males secrete luteinizing hormone and testosterone more irregularly, and jointly more asynchronously, than younger males. PNAS 93: 14100-14105, 1996.

- Hudetz AG. Effect of volatile anesthetics on interhemispheric EEG cross-approximate entropy. Brain Res 954: 123-131, 2002.

Furthermore, a very detailed, explicit example of how the algorithm works is in XApEn_instruct.doc in directory \mcourse\data\proj_timeSeriesAnalysis.
Data provided for this project are in the same directory. They are in *.mat format compatible with Matlab version 7. Details of the data are described in description_data.rtf.
Additional instructions:

- The data must be pre-processed before submission to XApEn analysis. They should be filtered and downsampled, which is accomplished in `prepareData.m`. Use this m-file for pre-processing and modify it according to your needs.
- In the main function, you should implement an (optional) normalization of the input data, e.g., subtract the mean from each channel and divide it by its standard deviation.
- This note is redundant, but here it is again: it is essential that your function be flexible. The user should be able to specify all major parameters which govern the computations, like the tolerance r, segment length m, and others.
- The figure above depicts results from the analysis of local field potential data (from rats) with XApEn. XApEn was computed for short data segments and plotted below the raw data. This kind of depiction is instructive, but requires some additional programming. For the presentation you need not do it in the same way, but some way of showing the raw data from which XApEn values were computed would certainly aid the audience in getting a 'feeling' for what this complex parameter expresses.

Model of Protein Synthesis (6,++)

Protein Synthesis works in two steps: Transcription and translation. Transcription is the synthesis of a complementary RNA string from a DNA string. Translation is the synthesis of the protein from this RNA string.
Your protein synthesis model shall be able to take a hypothetical bacterial genome, identify the genes and return both the RNA and protein strings. Details:

- Two cell arrays shall be returned, one containing RNA, the other the matching protein strings
- Depending on an input parameter the protein strings should be in one letter code or three letter code
- The algorithm shall identify start codons and end codons by itself

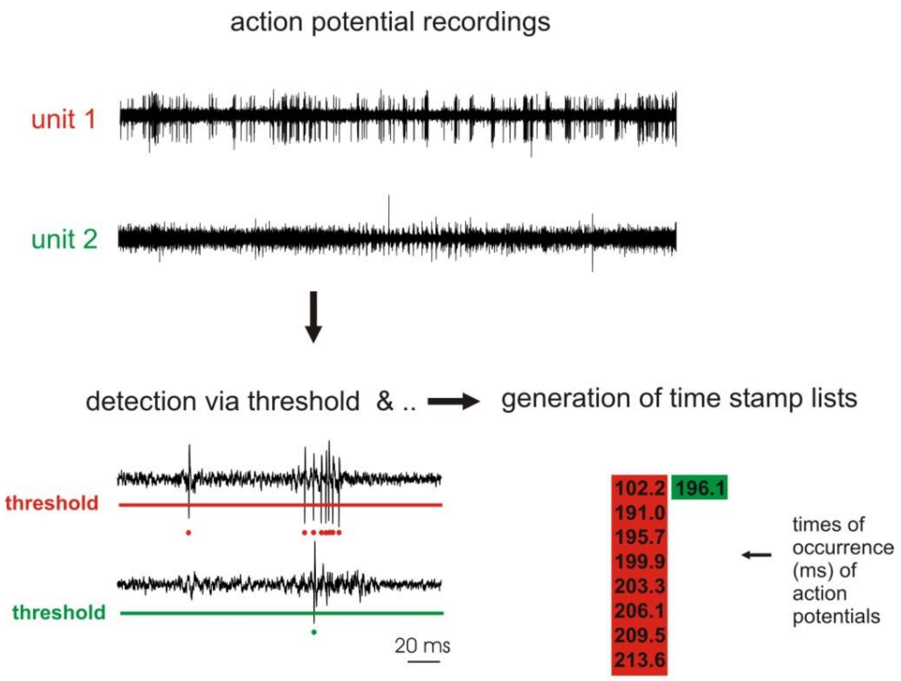
To implement this project you have to have a basic knowledge about transcription and translation. In terms of programming, knowledge about finding subarrays as well as string manipulation will come in handy. You may also consider using one or both of two recently introduced, new data types in Matlab, categorical arrays and strings.

Spike threshold detector and time stamp generator (6,+)

The task is to write a program which detects spikes within a noisy baseline. Details:

- the program should expect at least three inputs: the raw data (e.g. electrophysiological recordings), the sampling interval (in ms or µs) and the threshold(s).

- each column of the raw data represents one channel (time going down the columns). In other words, the function must be able to deal with multi-channel (=multi-column) data
- the program must output the occurrences of action potentials in ms for each channel in the form of a 'time stamp list' (tsl), that is, a column array containing the occurrences. The tsl of the different channels may be placed into a cell array or a struct array


action potential recordings
unit 1
unit 2
detection via threshold & ..
generation of time stamp lists
threshold
threshold
20 ms
102.2
191.0
195.7
199.9
203.3
206.1
209.5
213.6
196.1
times of occurrence (ms) of action potentials

- A spike's time stamp should be defined as the first data point above (or below) threshold
- the program must be able to detect, for each channel, positive-going or negative-going spikes (not both at the same time! One threshold per channel suffices, otherwise things get too complicated). The program should infer the direction of the spikes it should detect in a given channel from the sign of the threshold (so, negative threshold means look for negative-going spikes, positive threshold means look for positive-going spikes).
- data provided for this project are in directory \mcourse\data\proj_spikeThreshDetect. They are in *.mat format compatible with Matlab version 6, but were compressed outside Matlab to save disk space. The file contains two variables, `d` and `si`. `d` is the raw data; each column corresponds to one channel. Note that the data type of `d` is `single`. `si` is the sampling interval in µs.

IV. Graphics

Interactive Neuron (8,++)

The interactive neuron is intended to be a learning tool. It shall feature a graphical user interface. By showing a representative or schematic neuron it guides the user intuitively into the correct section of an encyclopedia. Details:

- Whenever you click on a specific part of the neuron like a spine, the axon hillock or a mitochondrium an explanatory text of this part is shown either in an adjacent text field or in a pop-up box. The selected part should be highlighted, e.g. by a surrounding box.
- The interactive neuron should also support a feature which makes all possible parts that can be activated visible (i.e. a button termed 'show all' or so).

To implement this project you should understand the use of handles both for functions and figures/axes. Furthermore, you should have knowledge about coordinates in figures. It is needed to both locate the mouse cursor and to efficiently issue graphics commands.