Matlab for PhD students – Basics 3 – WS 2010/11

Loops

Very often, blocks of commands have to be repeated in (almost) the same way several times. Loops are used to control repetitive algorithms.

For-Loops

- Idea: For-loops are (usually) count loops. They are used, when a command block needs to be repeated N times and the number N is known beforehand. Moreover, they are very convenient to use when an algorithm should be done for several specified values of a parameter.

- General syntax:
```
for vector
     commands
end
```

- Examples: (see `for_loop_script.m`)

```
%% example 1: stupid display of counter
for t=1:5
    counter=t                      % just display the counter
end

%% example 2: sum positive values of a vector
% this example is the long version of writing
% total_sum=sum([-8 19 88 0 -18])  ;-)
pos_sum=0;                         % define variables used in loop body
total_sum=0;
for v=[-8 19 88 0 -18];            % use for with a pre-defined
                                   % vector
    if v>0                         % counter can be used in
                                   % conditions
        pos_sum=pos_sum+v;         % sum positive values only
    end
    total_sum=total_sum+v;         % sum all values
end

%% example 3: fill a vector element-by-element
v=zeros(1,50);                     % pre-define a vector
for z=1:length(v)                  % visit each element of the vector
    v(z)=rand(1,1)*z;              % asign a value to the current
                                   % element
end
```

- Good to know:
    - Even though a vector (e.g `t=1:5`) is used in the for-statement, in each repetition of the commands the variable has a scalar value corresponding to the N-th element. E.g. in the first repetition t==1, in the second t==2… in the last t==5.
    - The most common vector used in for-loops are used for counting (e.g. `i=1:10` or `even_num=2:2:22`), but it is also possible to pre-define

vectors (e.g. v=[78; 9; -0.5] and use their values sequentially during the repetitions)
    - You should not change the value of the counter variable in the body of the loop. The value will be overwritten anyway when the next repetition starts.

While-Loops

- Idea: While-loops are the more general case to repeat a block of statements than for-loops. The program body of while-loops is repeated as long as a certain condition is true. You do not need to know beforehand, how often the commands will be repeated.
- General syntax:

```
define t
while condition, depending on t
     commands
     change t
end
```

- Example: (see `while-loop-script.m` and `factorial_demo.m for comparison of for-loops and while loops)`

```
%% example 1: stupid example to count with a while loop
c=1                 % define counter
while c<10          % test condition
    c=c+1           % increase counter (change test variable)
end

%% example 2: let the user guess a random integer number between
% 1 and 20 and generate a vector of guesses

   z=randi(20,1);       % random integer number between 1 and 20
   guess=0;             % define the test variable to make sure that
                        % the condition will initially be true
   c=0;                 % define a counter to count how long the user
                        % needs
   while(guess~=z)      % repeat as long as the random number is not
                        % guessed
       guess=input('Guess a number between 1 and 20; ');  % new test
                                                           % variable
       all_guesses=[all_guesses,guess];    % concatenate the new
                                           % guess to the list of guesses
end
sprintf('Congratulations, you needed %g trials to guess the
number!',length(all_guesses))
```

- Good to know:
    - A while-loop only works if the variable used in the condition is defined beforehand
    - The variable used in the condition must be changed (at least under certain conditions) in the body of the while loop. Otherwise, the loop will never terminate.
    - In case you produced an infinite loop, you can interrupt the program with ctrl-c.

    - You can always use while-loops instead of for-loops. See program for comparison. (However, for-loops are often easier to program and will always terminate.)

Nested loops

- Idea: You can use loops within loops. This is used quite often to generate index pairs to visit matrix elements.
- Example:

```
%% generate a matrix with random numbers
M=randn(17,25);         % generate a matrix
C=0;                    % define the counter for the
                        % elements >0.8
[rows,cols]=size(M);    % number of rows and columns

%% visit all elements of a random matrix and count the
% elements >0.8
for r=1:rows            % outer loop visits rows
    for c=1:cols        % within each row visit each column
        if M(r,c)>0.8   % count elements >0.8
            C=C+1;
        end             % end of if
    end                 % end of loop across columns
end                     % end of loop across rows

%% user output
sprintf('There were %g elements >0.8',C)
```

Searching in data

- Idea: Very often, you need to find subsets of data that match certain conditions. There are (at least) three ways in Matlab to do so, which we will compare for one example. of a test person doing a psychophysical experiment.
- Example: In our example we take the (fake) data of a test person doing a psychophysical experiment:
    - Column 1: 10 different stimulus conditions, named 1 to 10, presented in random order.
    - Column 2: corresponding subject response: 1 if the subject responded correctly, 0 if he did not.
    - Column 3: corresponding reaction time

    (if you want to see how I generated this fake data: `make_VP_data.m`)

  The tasks for this example are:
    1. What is the total percentage of correct detections?
    2. What is the percentage of correct detections for stimulus condition 3?
    3. Are the reaction times different for correct versus incorrect responses?

  See `search_demo.m !!!`
- Loops: Loops are the classic way (the “C way”) to search in data. The idea is to loop through all elements of a data set and test if a certain condition is met. This way of searching works well in Matlab, but it is not the fastest and most elegant way.
- Logical matrix operations: This is the “Matlab way” to search. It is the most elegant way, but maybe also the way which needs most thoughts. The idea is

to apply logical operations to entire vectors or matrices and use only those sub-parts for which the condition is true. The main method here is logical indexing. If a vector of logical values is applied as index to a vector of the same length, a vector of those elements that correspond to “true” is extracted.
Syntax of logical indexing:

```
V=[6 9 18 35 4];      % define a vector
L=logical([1 0 0 1 1]);    % define a vector of logicals of the
                      % same length
Vshort=V(L)           % apply the vector of logicals as
                      % indices. Vshort is [6 35 4].
```

- find command: This is the high-level-function way to search (maybe the “excel-way”). Find is a really powerful command, but also a good source of chaos if you are not sure you know what you are doing. Please note: find usually gives back the indices, not the values of the elements you search for!
  Syntax of find:

```
ind = find(X)                   % gives back indices of all
                                % nonzero elements of vector X
ind = find(X, k)                % gives back indices of the
                                % first k nonzero elements of
                                % vector X
ind = find(X, k, 'first')       % gives back indices of the
                                % first k nonzero elements of
                                % vector X
ind = find(X, k, 'last')        % gives back indices of the
                                % last k nonzero elements of
                                % vector X
[row,col] = find(M, ...)        % gives back row and column
                                % indices of nonzero elements
                                % in matrix M
[row,col,v] = find(M, ...)      % gives back row and column
                                % indices and values of nonzero
                                % elements in matrix M
```

Sorting data

- Idea: Very often, you want to sort data according to specific criteria, e.g. from small to large numbers. However, there are two different ways how you can sort a matrix: Either you sort the rows or columns independently of each other. Or the matrix is sorted according to a reference column, keeping the rows intact. (There is no “sortcolumns” command in Matlab. If you want to sort according to a row, you have to transpose the matrix.)
- sort command: Sorts the columns of a matrix independently of each other

```
B = sort(A)                  % sorts the elements of vector A
                             % or each column of matrix A
B = sort(A,dim)              % for each column of matrix A for
                             % dim=1 or each row for dim=2
B = sort(...,mode)           % mode can be ‘ascend’ or ‘descend’
[B,IX] = sort(A,...)         % also returns indices
```

- sortrows command: Sorts a matrix based on a reference column, keeping the rows intact.

```
B = sortrows(A)              % sorts the rows of matrix or column
                             % vector A in ascending order based
                             % on the first column of the matrix
```

```
B = sortrows(A,columns)      % based on the columns specified in
                             % the vector column. If an element
                             % of column is positive, the MATLAB
                             % software sorts the corresponding
                             % column of matrix A in ascending
                             % order; if an element of column is
                             % negative, MATLAB sorts the
                             % corresponding column in descending
                             % order.
sortrows(A,[2 -3])           % sorts the rows of A first in
                             % ascending order for the second
                             % column, and then by descending
                             % order for the third column.
[B,index] = sortrows(A,...)  % also returns an index vector
```

- issorted command: returns a logical 1 (true) if all elements are in sorted order

```
TF = issorted(A)             % A is considered to be sorted if A
                             % and sort(A) are equal.
TF = issorted(A, 'rows')     % A is considered to be sorted if A
                             % and sortrows(A) are equal.
```

Homework

*   means "optional"
** means "optional and difficult"

Please work through the script and the programs `for_loop_script.m`, `while_loop_script.m`, `nested_loops_script.m`, `factorial_demo.m`, and `search_demo.m` first.
If you have not done so on Thursday, also take a look at `aliasing_effect.m` and `if_demo.m`

1. **Loops**
   a. Write a function, which reverses the order of elements of a user-defined vector. (E.g. [19 7 30] becomes [30 7 19]).

   b. Modify example 2 in `while_loop_script.m` by telling the user after each guess if the random number is smaller or bigger than the number he / she guessed.

   c. As mentioned before, loops can slow down computations considerably. Test this effect by using the Matlab stop watch:

   ```
   tic              % turns on the stop watch
   statements
   toc              % turns off the stop watch and generates
                    % a command window output telling the
                    % time needed to process the statements.
   ```

   Use tic and toc to test, how long it takes Matlab to generate a 1000-by-1000 Matrix of random numbers
   - When the matrix is generated in one line
   - When the matrix is generated with a for-loop
   - When the matrix is generated with a while-loop
   - When the matrix is generated with nested loops

d. Measurement of reaction times very often show skewed distributions. Take a look at the reaction times (in ms) measured for a test person: `rt_VP5.mat`.
    - Plot a histogram with more than 10 bins to see the distribution. Why is this not a normal distribution?
    - Calculate the mean and the median of the test person. Why are they so different?
    - You will find one very large value in the data. Remove this data and calculate mean and median again.
    - In the data set `rt_all.mat` 180 reaction times of 24 test persons were measured. Calculate the means and medians for all test persons and look at their distributions.

e. Population growth is a topic with a long tradition in biological science. Here, we will have a look at the very simplest linear model for exponential growth:
   Imagine a population of P flies in a laboratory, which is counted every day. After a while we will find out that – on average - every day 20% of the flies eclose (fertility f=0.2) and 10% of the flies die (mortality m=0.1). Based on these numbers, we can predict the daily population growth:
   $dP_t = f*P_t - m*P_t = (f-m)*P_t$.
   Therefore, the population size P_{t+1} at day t+1 depends on the population size of the day before in the following way:
   $P_{t+1} = P_t + dP_t = P_t + (f-m)*P_t = (1+f-m)*P_t$

    - Write a function that gets mortality, fertility and number of animals on day 1 and the number of days to be calculated as inputs. The function should calculate the population size for each day and store these numbers as a vector. Display this vector graphically and give it back as output variable. (Test your function e.g. with fertility f=0.2, mortality m=0.1 and size P=500 on day 1 and 200 calculated days)
    - Write a function that gets a threshold value as additional input parameter and calculates the population growth until this threshold is reached.
    - Add tests for correct user inputs to the function.
    - Write a script that systematically loops through several parameter values for the fertility by calling the function for each of these combinations. Display the effect of fertility on population size graphically by plotting the number of animals e.g. on day 200 against the fertility values.

f. * The Kniffel task: write a program that tosses 5 dice and allows the user up to two times to put back as many of the dice into the dice cup as he / she wishes and toss them again. Write a nice user dialogue and sort the dice by their numbers.

g. ** `spikedaten_kurz.mat` (on the course homepage) contains the intracellularly recorded membrane potential of a leech neuron. The cell responses (in mV) to 10 presentations of the same current stimulus (in nA) were recorded 10 times with a sample rate of 10000 values/sec.

The neuron responds with action potentials (fast depolarizations of the membrane potential).
Write a function that gets a matrix with membrane potential recordings and calculates for every row the number of action potentials generated by the neuron. Tips:

- First display the data graphically and look at it
- Define “by eye” a voltage threshold to find the action potentials
- Caution! At the end of the stimulation an artifact occurs that looks almost like an action potential but is much shorter. Try not to count these artifacts.
- The difficulty of this task is that the voltage stays over the threshold for some time, but you want to count each action potential only once.
- If you want to think even harder, think about a better way to find action potentials than a fixed voltage threshold determined by eye.

2. Searching and sorting

a. Load `fuetterungen.mat` from the course homepage. The matrix shows for 31 days the amount of rain (in mm, 1. column) and the number of times a female bird visits the nest for feeding the young birds (2. column).
   1. Sort the matrix fuetterungen once according to the number of feedings and once according to the amount of rain.
   2. Find out
   - Which days of this month were without rain?
   - On which days did the bird feed more than 50 times?
   - On which days without rain did the bird feed more than 50 times?
   - How often did the bird feed on two consecutive days together more than 100 times?

b. Exercise 2b of day 2 repeated: Use `stimulus.mat` and `antwort.mat` downloaded from the homepage.
   - Calculate the mean and the standard deviation of the responses before the stimulus changes.
   - Determine the time when the cell model responds to the stimulus by using a threshold: The response starts, when a data point differs by more than two standard deviations from the average response before stimulation.
   - Calculate the time difference between the stimulus change and the response onset.
   - * Think about a useful criterion to determine the offset of the response and calculate the time between stimulus offset and response offset.

   Write your program as a function with useful input and output parameters.

c. * A little graphics exercise: The data set `phWerte.mat` shows the pH measurements of a fish tank. The fish in your tank are happy with the range pH 6.5 to 7.5, but are in danger when this range is left. Find the values that are inside and those that are outside of this range. Moreover, define all measurements that deviate by more than 3 standard deviations from the mean measurement as measurement errors. Plot your data set with three differently colored symbols for the three categories of data points.