# MATLAB in the Biosciences - Exercise Solutions

Sven Mesecke

Sep 2013

**Exercise 1**

```
% there are several ways for creating these arrays
% 1 - squared brackets
A = [1 3 5; 3 6 3; 9 8 5]
% 2 - functions for array creation
A = rand(3)
A = magic(3)

b = [3; 4: 8] % 3 x 1 - column vector!

A * b(1)
A * b % matrix multiplication
A.*b % element-wise multiplication

B = [b b b] % horizontal concatenation

A = A.*B

A(2,1)
A(3,:)
A(2:3,2:3)
```

---

**Exercise 2**

```
t = linspace(0,2*pi,100); % use linspace to get n equidistant numbers
tnew = sin(t);
tc = asin(tnew);
t == tc % you could use `all(t==tc)` to check whether all are equal
notequalpos = find(t~=tc);
```

---

**Exercise 3**

```
% use the documentation to find the functions needed
max(tnew)
min(tnew)
abs(tnew)

sqrt(t)
log10(t)
```

---

**Exercise 4**

```
str = 'Sven'

upper(str)
lower(str)

cstr = {str, 'EMBL'}
```

---

**Exercise 5**

*interactive*

1. Use the `Import Tool` (double-click the csv or Excel file) to preview the file. Rename the column headers to *time*, *R*, *L* and *RL*
2. Generate script, name script `importdata.m`
3. Import data as *Column Vectors*
4. Plot the data by using the *Plot* dialog in the *Workspace Browser*
5. Add all data to the plot (*Add Data*), modify `LineWidth` and other properties like Markers, Colors, labels, title etc. according to you taste
6. Once done, generate code (*File -> Generate Code*) and save the corresponding function as `createfigure.m`
7. Save the figure as a png file using either the (i) *File->Export Setup*, (ii) `print` command or (iii) the `export_fig()` function from the MATLAB File Exchange

---

**Exercise 6**

```
%% import the data using autogenerated code
% assign the columns to data vectors
[time, R, L, RL] = importdata('RLBinding.csv',1, 17)

%% plot using the autogenerated code from the plot tools
createfigure(time, R, L, RL) % the exact function signature
% depends on the generated code and your sequence of steps
```

---

**Exercise 7**

```
%% get a list of csv files in the current folder
files = dir('*.csv');
% calculate the best number of rows and columns for
% the subplot
nfiles = length(files);
nrows = round(sqrt(nfiles));
ncols = ceil(sqrt(nfiles));

%% loop through all files

for ii = 1:nfiles
    %% import the data using autogenerated code
    % assign the columns to data vectors
    [time, R, L, RL] = importdata(files(ii).name,1, 17)

    %% plot using the autogenerated code from the plot tools
    subplot(nrows,ncols,ii), createfigure(time, R, L, RL)
    % the exact function signature
    % depends on the generated code and your sequence of steps

end
```

Then click on the publish button and a HTML report is automatically generated. Change the format by clicking on the arrow below *Publish*

---

**Exercise 8**

1. in MATLAB:

```
% the ODE model
drdt = - kf*r*l;
dldt = - kf*r*l;

drldt = + kf*r*l;
```

2.

```
function rhs = oderhs(t,x)
% wrap a function header around the previous code block and
% save as oderhs.m

kf = 1;
% extract the input vector into the state variables
r = x(1);
l = x(2);
rl = x(3);

% ODE system
drdt = - kf*r*l;
dldt = - kf*r*l;
drldt = + kf*r*l;

% any MATLAB ODE solver expects a column vector as output argument
rhs = [drdt; dldt; drldt];
```

3.

```
function [t,x] = odeeuler(t0, tf, x0, h)
% solves an ODE system using the Euler method
% if you want it to be a general function that could be
% used for any ODE system, specify the ODE rhs function
% as an input argument:
% function [t,x] = odeeuler(@oderhs, t0, tf, x0, h)

% create the vector of time points
% at which to solve
t = t0:h:tf;
nsteps = length(t);
% preallocate the solution array
x = zeros(nsteps,3);
x(1,:) = x0;

% loop from 2 to the number of wanted solution points
for ii = 2:nsteps
```

```
    x(ii,:) = x(ii-1,:) + h*oderhs(t(ii),x(ii-1,:));
end
```

4.

```
x0 = [10 8 0];
[t,x] = odeeuler(0, 20, x0, 0.1);
% extract into column vectors
Rs = x(:,1);
Ls = x(:,2);
RLs = x(:,3);

% we could be using the createfigure function we created earlier,
% or just call plot()
plot(t,Rs,'r-d', t,Ls,'g-o', t,RLs,'b-x')
```

---

**Exercise 9**

1.

```
function obj = objfunction(parameter)

[time, R, L, RL] = importdata(files(ii).name,1, 17);
% put the dataset into a single matrix so that we simply
% do a matrix subtraction
data = [R L RL];
ndata = length(time);

% simulate the model using the parameter
% instead of our own ODE solver, we'll use one
% of the in-built solvers and define the model rhs as
% a nested function to be able to pass in the parameter
% (matlab ODE solvers can only use functions that accept
% exactly two arguments)

[t, x] = ode15s(@odefunc,time, data(1,:));
res = data-x;
% calculate the least-squares term
% when using lsqnonlin in the calling function,
% simply return the column-vectorized residual
% (here: res)
obj = 1/(ndata-1)*sum(sqrt((res(:)).^2));
```

```
% nested ode function
function rhs = odefunc(t,x)
    % the 'parameter' variable in the parent function
    % is visible in this nested function (see the
    % section on scope)
    kf = parameter;
    % extract the input vector into the state variables
    r = x(1);
    l = x(2);
    rl = x(3);

    % ODE system
    drdt = - kf*r*l;
    dldt = - kf*r*l;
    drldt = kf*r*l;
end

% when using nested functions the main function needs
% to be closed with 'end'
end
```

2.

```
function bestparameter = fitparameter()

[time, R, L, RL] = importdata(files(ii).name,1, 17);
data = [R L RL]; % this dataset (and time) is/are
% visible in objfunction

% when we use lsqnonlin (available in the optimization
% toolbox, we need to return only a vector of the
% residuals)
% the third and fourth argument are the lower and upper
% bounds on the argument that should be minimized
[bestparameter] = lsqnonlin(@objfunction, p0, 0, Inf);

function res = objfunction(parameter)

    ndata = length(time);
    [t, x] = ode15s(@odefunc,time, data(1,:));
    res = data-x;
    % convert the ndata x 3 matrix to a column vector
    % using the : operator
    res = res(:);
```

```
    function rhs = odefunc(t,x)
        kf = parameter;
        % extract the input vector into the state variables
        r = x(1);
        l = x(2);
        rl = x(3);

        % ODE system
        drdt = - kf*r*l;
        dldt = - kf*r*l;
        drldt = kf*r*l;
    end
end
end
```

---

**Exercise 10**

```
files = dir('*.tif');

for ii = 1:length(files)
    iminfo = imfinfo(files(ii).name);
    % allocate the array imstack
    imstack = zeros(iminfo(1).Height, iminfo(1).Width,length(iminfo);

    % read in the tif into imstack using imread
    for jj = 1:length(iminfo)
        imstack(:,:,jj) = imread(files(ii).name,'Index',jj);
    end

    % extract the fluorescent images into another image stack
    imstackf = imstack(:,:,1:end/2);

    % maximum intensity projection
    imm = max(imstackf,[],3);

    % convert it to (0,1) intensity range
    img = mat2gray(imm);

    % median filtering to remove noise
    img = medfilt2(img);

    % sharpen the image
    img = imsharpen(img);
```

```
    % calculate the threshold
    threshold = graythresh(img);

    % segment the image using the threshold
    imbw = im2bw(img, threshold);

    % identify the objects
    imobjs = bwlabel(imbw);

    % extracting properties of the objects
    improps = regionprops(imobjs,'all');
    ncells = length(improps);

    % the number of cells is the number of identified objects
    sprintf('This image contains %d cells', ncells)
end
```

---