CSE5301 Neuro-Fuzzy Computing

Tutorial/Assignment 1: Basic computational tools and concepts

---

About this tutorial

The objective of this tutorial is to introduce you to:

- the computational package MATLAB that we will be using in our unit,
- basic concepts of linear algebra, namely, vectors and matrices, operations that can be performed on them, and their geometric interpretation.
- simple neural networks

It is important to realise that vectors and related points from a n-dimensional space, which are describe by a collection of n values (numbers), represent signals or stimuli exciting **neurons** through their **synapses**, whereas matrices typically describe synaptic strengths or weights.

1.1 Introduction to MATLAB

MATLAB is a widely adopted package which integrates computation, data analysis, visualization, and programming in one environment. It is available on many PCs and Unix/Linux workstations around the campus. You can also purchase a MATLAB Student Version, in particular, from

`http://www.mathworks.com/academia/student_version/`

1.1.1 Easy beginning

Depending on the computational platform that you work with, to **invoke** MATLAB you either click on the MATLAB icon, or type in `matlab` in the terminal window. It should bring out the MATLAB **command window** in which you can see a prompt `>>`. That is the place where you type in commands that MATLAB will interpret immediately. Start with

```
>> 2*2
```

and the response will be:

```
ans =
   4
```

Now, you are ready to get help that will explain all the complexity (or simplicity) of MATLAB operations. Type in

```
help +
```

to get a long list of operators and special characters.

Any programming language consists, at the bottom of its structure, of **constants**, **variables** and **operators**. The assignment operator '=' and resulting **assignment statement** is the most fundamental one. As an example, you can write the following simple assignment statements:

```
a = 2e2
b = a*a/5
```

where `a` and `b` are variables. The statements assign a constant, and the result of evaluation of an expression value to two variables, `a` and **b**, respectively. You do not have to declare variables, nor specify their type.

Once we have mastered a 2×2 example we can proceed to produce the whole multiplication table. Type in the following commands and study the results.

Create a 10-component row vector:
`a = 1:10`
Transpose it to get a column vector
`a'`
Multiply a row vector by a column vector (it is an inner product – will discus it below)
`b = a*a'`
Multiply two row vectors component-wise
`c = a.*a`
Multiply a column vector by a row vector (it is an outer product – will discus it below)
`d = a'*a`

Use the `help` command with: `colon`, `punct` and `paren` to see lots of useful explanations.
Note in particular that colon ':' is used to produce a sequence of numbers (arithmetic progression) in the form `initial:step:final`

1.1.2 A diary file

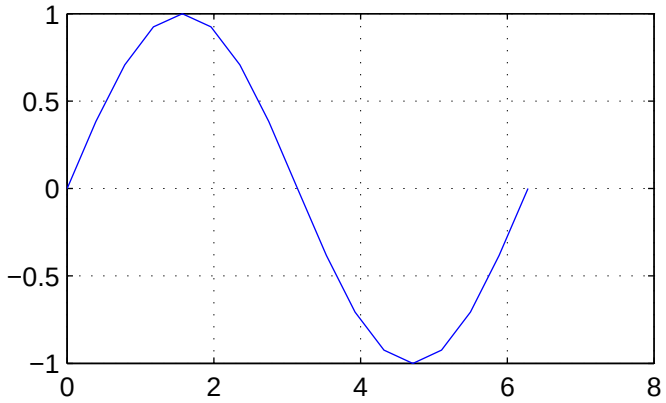
Writing statements in a command window seems to be a convenient way of proceeding for a beginner. The obvious drawback is, however, that our typing has not been saved. This can be accomplished by opening a diary file.
To begin with get `help` on three commands: `pwd`, `cd` and `diary`.

First establish the name of your **current directory** (folder) using the `pwd` command, and then go to your **favourite directory** using the `cd` command. Alternatively you can navigate in the `current directory` window to your favourite directory. Now type in the command:
`diary my1st.txt`
From now on everything you write goes into the file `my1st.txt`. We will use this file later on to produce our first MATLAB script. In order to be able to use a diary file we have to close it using the command `diary off`.

1.1.3 The first plot

We are ready to produce our first plot. Let us try to plot a sinusoidal function. Define a set of points for which the function is to be evaluated:
`x = 2*pi*(0:16)/16`
where `pi` is a pre-defined well known constant $\pi = 3.141596...$. Note that angles are in radians. Now calculate

```
y = sin(x)
```
and here the plot is coming:
```
plot(x, y)
```
Add also the grid typing in
```
grid on
```
The result should look similar to the following plot:



1.1.4 Formation of vectors and matrices

A **matrix** is a rectangular $r \times c$ table of numbers. This is a trivial statement, but we have to start somewhere and learn how we can input such a table. One way is to use square bracket, commas and semicolons. Write in a command window:
```
a = [ 2, 3, 4 ; 0.1, .5, .8]
```
and
```
size(a)
```
to get the size of the matrix, that is, the number of **rows** and **columns**.
We can also write statements like this:

```
b = [2:2:8
     7:-1:4
     5 2 7 3]
```

to specify a 3 by 4 matrix.
We can refer to any element from the matrix, or any submatrix using the following notation. To get the element from a 2nd row and a 3rd column type in:
```
b(2, 3)
```
To get the complete 2nd row type in:
```
b(2, :)
```
To get the complete 3rd column type in:
```
b(:,3)
```
Check that
```
b(:)
```
produces a long column vector formed from the column-wise scan of the matrix `b`. You can try many other possible commands referring to submatrices.

Note many functions performed by the colon ':' operator.

Exercise 1. 1
In **your report**

1. list all functions of the colon operator encountered by now.
2. Input a matrix
$$W = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ -1 & 1 & 3 & 1 \end{bmatrix}$$
3. What does `W(:)` produce?

In **your report** list all functions of the colon operator encountered by now. Give examples.

An n-dimensional **vector** is represented by either a column $(n \times 1)$ or row $(1 \times n)$ matrix. More importantly, a vector can be thought of to be equivalent to a **point** in an n-dimensional space. Let us specify a point in a 2-dimensional (2-D) space:
```
v = [3 2]
```
To visualize such a point we use a plot command
```
plot(v(1), v(2), 'd'), grid on
```
That should produce a figure with a diamond shape representing the point `v`. To **plot a vector** representing the point we need a collection of x and y coordinates. The following plot command will do the trick (try `help plot` to get details clarified).
```
plot([0 v(1)], [0 v(2)], '-', v(1), v(2), 'd'), grid on
```
adding the following command
```
axis([0 5 0 4])
```
would move the point to the centre of the plot.

1.2 Vectors and matrices

In this section we refresh our knowledge of basic facts from the linear algebra and interpretation of these facts in MATLAB.

1.2.1 The length of a vector (aka vector norm or magnitude)

There is usually a bit of a terminological confusion regarding terms size and length of a vector.

The **size** of an n-dimensional vector is equal to the number of its components, which is n.
A MATLAB example:
```
vn = 0.9:0.2:2.1 , n = max(size(vn))
```

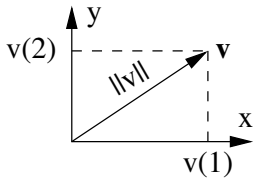
The **length** of a vector is a geometric concept. If a vector in an n-dimensional spaceis specified by the following n components:
$$\mathbf{v} = [v_1,\ v_2 \ldots v_n]$$

then recalling Pythagoras the vector **length** (also known as the Euclidian **norm** or vector magnitude) is defined in the following way:

$$||\mathbf{v}|| = \sqrt{v_1^2 + v_2^2 + \ldots + v_n^2} = \sqrt{\sum_{i=1}^{n} v_i^2} \tag{1}$$

Note that in literature a vector is often denoted with bold, lower case characters, e.g. $\mathbf{v}$ and its ith component by v_i, accordingly. In MATLAB we do not use bold characters. Instead we can use any informative names.
Let us do some MATLAB examples with vectors. Try first:

```
v = [3 4], vL = sqrt(3^2 + 4^2)
```



and it should produce 5 (recall a right-angle triangle with sides 3, 4, 5).

A bit more complicated example:

```
b = 0.5:0.3:2.7
Lb1 = sqrt(sum(b.^2))
Lb2 = norm(b)
```

Analyze the above statements carefully to be sure that you understand all its minute details and verify that `Lb1 == Lb2`. Use `help` when needed.

Note the difference between the **assignment operator** '=' and the **equality operator** '=='.

Note also that, if we **multiply a vector by a constant**, its length is increased by the same factor.

Exercise 1. 2
Confirm the above statement with two examples.
□

1.2.2 Adding vectors

A sum of two vectors is a vector with respective components summed together. To write a formal expression let us assume that we have two n-dimensional vectors

$$\mathbf{u} = [u_1,\ u_2, \ldots, u_n] \quad \text{and} \quad \mathbf{v} = [v_1,\ v_2, \ldots, v_n]$$

then their sum can be written as:

$$\mathbf{s} = \mathbf{u} + \mathbf{v} = [u_1 + v_1,\ u_2 + v_2, \ldots, u_n + v_n] \tag{2}$$

Let us plot in MATLAB a well-known parallelogram illustrating addition of 2-dimensional vectors. We take this opportunity to introduce MATLAB scripts.

1.2.3 MATLAB scripts

A MATLAB script is a text file containing commands in the order of their execution, that is MATLAB program. We can use any text editor to create such a file, A native editor is provided by MATLAB. To invoke it, select from the pull down menu:

File → New → M-file

It should open a new editor window named `Untitled.m`. Extension `.m` is a must for MATLAB scripts.

The next thing we need to do is to change the name to something meaningful, say, `plt2vec.m`. To do this, from the **editor** pull down menu select:

File → Save As...

and in the dialog window Save file as: type in the file name, in this case, `plt2vec.m`.
From now on if you type in the name `plt2vec` in the MATLAB command window the script, that is, commands in it will be executed.

Type in in the editor window the following commands:

```
%  plt2vec.m          % is a comment mark
%       21 February 2005
% A program to demonstrate addition of 2-D vectors

 clear        % a good idea to start with
 u = [3 2] ;  % semicolon suppresses the printing
 v = [2 5] ;
 s = u + v ;  %  s = [s(1) s(2)] is the sum vector
 % an illustrative plot of vector addition:
 figure(1)  % specify the figure number
 %  triple dot is  a line continuation mark
 plot([0 u(1)], [0 u(2)], '-', u(1), u(2), '*', ... % u
      [0 v(1)], [0 v(2)], '-', v(1), v(2), '*', ... % v
      [0 s(1)], [0 s(2)], '-', s(1), s(2), '*', ... % s
      [u(1) s(1)], [u(2) s(2)], '--', [v(1) s(1)], [v(2) s(2)], '--')
  % the last line plots other sides of the parallelogram
  grid on
  title('Adding two vectors')
  xlabel('x (or 1st) axis')
  ylabel('y (or 2nd) axis')
  axis([0 6 0 8])
  text(2.1, 1.7, 'u')
  text(1.2, 3, 'v')
  text(3.1, 4.5, 's')
  print -f1 -djpeg plt2vec
  % print -f1 -depsc2 plt2vec
```

Once the script is ready, **save** it.
To run the script, you can either type in the command window `plt2vec`, or select from the editor pull-down menu:

Debug → Run

Fix all the reported errors (might not be easy!) and after that the resulting plot should look similar to the one in Figure 1.

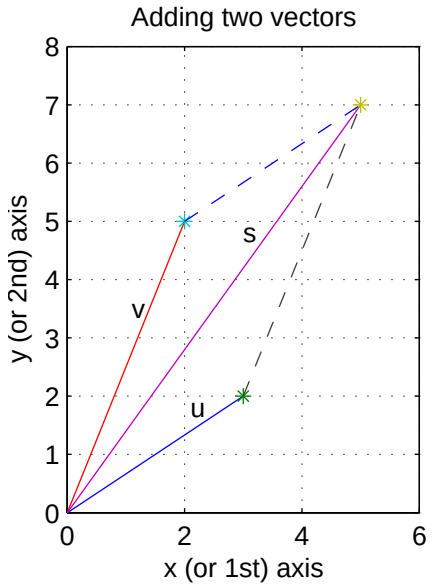


Figure 1: Plotting the sum of two vectors

The `print` command generates a plot `plt2vec.jpg` in a jpeg format. It will be stored in your current directory. Other formats, e.g., colour postscript `epsc2` can be also selected. Use help for details.

Exercise 1. 3
Produce a plot similar to the one in Figure 1 with two vectors being added forming an **obtuse angle**.
□

1.2.4 Inner product of two vectors (aka "dot" product)

Let us assume that we have two n-dimensional **row vectors** specified as follows:

$$\mathbf{u} = [u_1,\ u_2, \ldots, u_n] \quad \text{and} \quad \mathbf{v} = [v_1,\ v_2, \ldots, v_n]$$

then their **inner product is a single number** specified as a **sum of products** of corresponding components:

$$p = \mathbf{u} \cdot \mathbf{v}^T = u_1 \cdot v_1 + u_2 \cdot v_2 + \ldots + u_n \cdot v_n = \sum_{i=1}^{n} u_i \cdot v_i \tag{3}$$

where $[\ ]^T$ denotes a transpose operation (an apostrophe in MATLAB), so that $\mathbf{v}^T$ is a column vector,

To calculate the inner product in MATLAB we must multiply **a row vector by a column vector**.
Try

```
u = [2 3 5], v = [3 1 2], p = u*v'
```

The result should be 19.

There are many interesting **properties of the inner product** that we will be using in our classes. Some of them are listed below.

- The inner product can be calculated as the product of the vector norms (lengths) and the cosine of the angle between vectors:

$$p = \mathbf{v} \cdot \mathbf{u} = ||\mathbf{v}|| \cdot ||\mathbf{u}|| \cdot \cos(\alpha) \tag{4}$$

  The above expression can be used to calculate the angle between two vectors, which, in a multidimensional space might be difficult to imagine.

$$\cos(\alpha) = \frac{\mathbf{v}}{||\mathbf{v}||} \cdot \frac{\mathbf{u}}{||\mathbf{u}||} \quad , \quad \text{or} \quad \alpha = \arccos(\frac{\mathbf{v}}{||\mathbf{v}||} \cdot \frac{\mathbf{u}}{||\mathbf{u}||}) \tag{5}$$

  Try for the `v` and `u` vectors specified above:

```
alpha = acos(u*v'/(norm(u)*norm(v)))
```

  The resulting angle in in radians. To convert it to degrees write:

```
alpha*180/pi
```

  The result should be $\approx 34.5^o$.

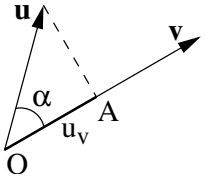
  Note also that in eqn (5) the fraction $\frac{\mathbf{v}}{||\mathbf{v}||}$ denotes a vector of the length equal to 1 (a **unity vector**) pointing in the same direction as the vector $\mathbf{v}$.

- **Orthogonal vectors**

  Note from eqn (4) that if the angle between vectors is 90^o (vectors are orthogonal) that their **inner product is equal to zero** $(\cos 90^o = 0)$.

- **Projection of a vector on another vector**

  Final application of an inner product is calculation of a projection of a vector, say, $\mathbf{u}$ on another vector, say $\mathbf{v}$. With reference to the sketch below we have:



$$u_v = ||\mathbf{u}|| \cdot cos(\alpha) = \frac{\mathbf{u} \cdot \mathbf{v}}{||\mathbf{v}||} \tag{6}$$

  Projection u_v of a vector $\mathbf{u}$ on a vector $\mathbf{v}$ is equal to the inner product of the vector $\mathbf{u}$ and the unity vector $\frac{\mathbf{v}}{||\mathbf{v}||}$.

  Note that if vectors $\mathbf{u}$ and $\mathbf{v}$ are orthogonal then the projection $u_v = 0$

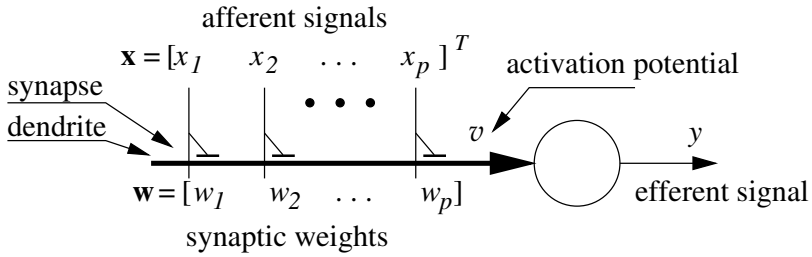
Exercise 1. 4

- Check the projection property of eqn (6) in MATLAB, that is, that two ways of calculating the projection give the same result.
- Produce in MATLAB a plot similar to that in eqn (6) (hint: calculate the $\overline{OA}$ vector).

□

1.2.5 Modelling a simple neuron

Consider a simple model of a neuron as in the following dendritic diagram:



The input vector $\mathbf{x}$ represents p values of a pattern exciting neuronal synapses represented by small triangles in the figure. the pattern values are arranged in a column vector, $\mathbf{x}$. Each synapse is characterised by a synaptic strength parameter, or **weight**. The individual weights are collected into a **weight vector** of size p. Now the activation signal v is calculated as an inner product of the input and weight vectors:

$$v = \mathbf{w} \cdot \mathbf{x}$$

In other words, the activation potential is a **linear combination** of input signals (afferents) and weight parameters. That is really the simplest, but powerful thing that we can do with input signals. Note that in a simplest neuron model the output signal (efferent) y is just proportional to the activation potential:

$$y = a \cdot v$$

where in particular, the proportionality constant often referred to as the gain, can be unity $(a = 1)$.

When a specific weight is **negative**, the synapse is called **inhibitory**. Conversely, if the weight is **positive**, the related synapse is called **excitatory**. When a weight is zero the related synapse does not exist.

Exercise 1. 5
Practice the above expression with MATLAB: specify p, $\mathbf{w}$ and $\mathbf{x}$, and calculate v and y.
□

1.2.6 Multiplication of a vector by a matrix

As you might remember from your MATH001 unit the product of a matrix and a vector is formed on a **row by column** basis. In order to examine details of such an operation let us assume that an $r \times c$ matrix W is composed of r row vectors as follows

$$W = \begin{bmatrix} w_{11} & \dots & w_{1c} \\ \vdots & \ddots & \vdots \\ w_{r1} & \dots & w_{rc} \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_r \end{bmatrix}$$

Now, if

$$\mathbf{v} = [v_1 \dots v_c]^T$$

is a column vector of size c, then the resulting product $\mathbf{u}$ is a column vector of size r formed from inner products of the matrix rows and the vector $\mathbf{v}$:

$$\mathbf{u} = W \cdot \mathbf{v} = \begin{bmatrix} w_{11} & \dots & w_{1c} \\ \vdots & \ddots & \vdots \\ w_{r1} & \dots & w_{rc} \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ \vdots \\ v_c \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_r \end{bmatrix} \cdot \mathbf{v} = \begin{bmatrix} \mathbf{w}_1 \cdot \mathbf{v} \\ \vdots \\ \mathbf{w}_r \cdot \mathbf{v} \end{bmatrix} = \begin{bmatrix} u_1 \\ \vdots \\ u_r \end{bmatrix} \tag{7}$$

It does sound simple, doesn't it? Just remember that you form **inner products** of rows and column(s) and that the resulting vector $\mathbf{u}$ consists of those inner products.

To practice it let us start with a 3 by 2 matrix W (capital letters will be typically used to name matrices)

```
W = [1 2 ; 3 1 ; 2 3]
```

and vector $\mathbf{v}$ of size 3

```
v = [2 1]'
```

The product

```
u = W*v
```

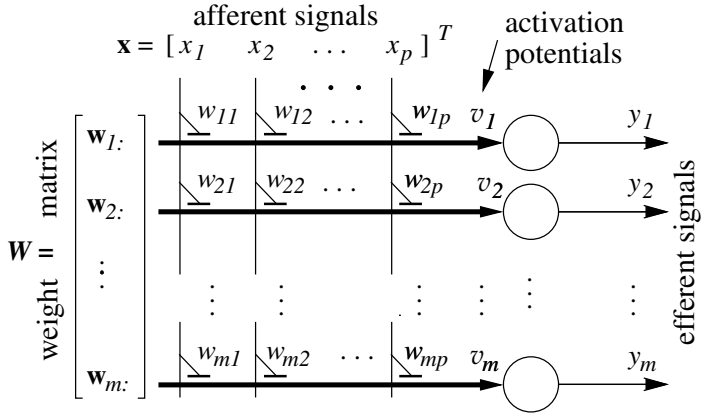
is a column vector of size 3.

Exercise 1. 6
Calculate a product of a 2×3 matrix by a suitable vector
□

Remember that if an $r \times c$ matrix acts on a vector, it changes the vector dimensionality from c to r.

1.2.7 A simple multi neuron model

Let us consider a collection of neurons excited by the same input pattern as in the following figure:



Note that now we have a whole $m \times p$ matrix of weights, W, one row per neuron. The p-dimensional input pattern, or p input signals (afferents) go to all m neurons, supplying synapses in a column with the same signal. The **vector of activation potentials**

$$\mathbf{v} = [v_1 \dots v_m]$$

can be simply calculated as a product of the weight matrix, W and an input vector $\mathbf{x}$:

$$\mathbf{v} = W \cdot \mathbf{x}$$

In a linear model, the output vector (efferents) y is proportional to the activation potential vector, v

$$\mathbf{y} = a \cdot \mathbf{v}$$

Exercise 1. 7
Specify p, m, W and $\mathbf{x}$, and calculate the vector of activation potentials and efferents.
□

1.2.8 Outer product of two vectors

The inner product of two vectors (row by column) produces a number, the outer product (column by row) produces a matrix and plays an important role in forming associative memories.

Let us assume that we have a **column** $(r \times 1)$ vector $\mathbf{u}$ and a **row** $(1 \times c)$ vector $\mathbf{v}$. Then their outer product, which will be a $r \times c$ matrix A, is defined as follows

$$A = \mathbf{u} \cdot \mathbf{v} = \begin{bmatrix} u_1 \\ \vdots \\ u_r \end{bmatrix} \cdot \begin{bmatrix} v_1 & \dots & v_c \end{bmatrix} = \begin{bmatrix} u_1 v_1 & \dots & u_1 v_c \\ \vdots & \ddots & \vdots \\ u_r v_1 & \dots & u_r v_c \end{bmatrix} \tag{8}$$

As in all matrix/vector multiplications we multiply rows by columns, only this time we have r rows and c columns, therefore, the result is a $r \times c$ matrix. Note that the outer product matrix consists of all possible products of components of two vectors. We can say that two vectors are "**stored**" in the outer product matrix.

As a simple example let us try the following calculation

```
u = [2 ; 3], v = [4 2 1], A = u*v
```

This will produce a matrix

```
A =  8  4  2
    12  6  3
```

Finally we will demonstrate how to "read out" a vector stored in an outer product matrix. This will be achieved by multiplying the matrix by the appropriately transposed vector. Following notation of eqn (8) we have

$$A \cdot \mathbf{v}^T = \mathbf{u} \cdot \mathbf{v} \cdot \mathbf{v}^T = ||\mathbf{v}||^2 \cdot \mathbf{u}$$

This can be re-written in a form of an important property:

$$\text{If} \quad A = \mathbf{u} \cdot \mathbf{v} \quad \text{and} \quad g = ||\mathbf{v}||^2 \,, \quad \text{then} \quad A \cdot \mathbf{v}^T = g \cdot \mathbf{u} \tag{9}$$

Exercise 1. 8
Verify the "read-out" property of the outer product with MATLAB examples.
□

1.2.9 More MATLAB commands

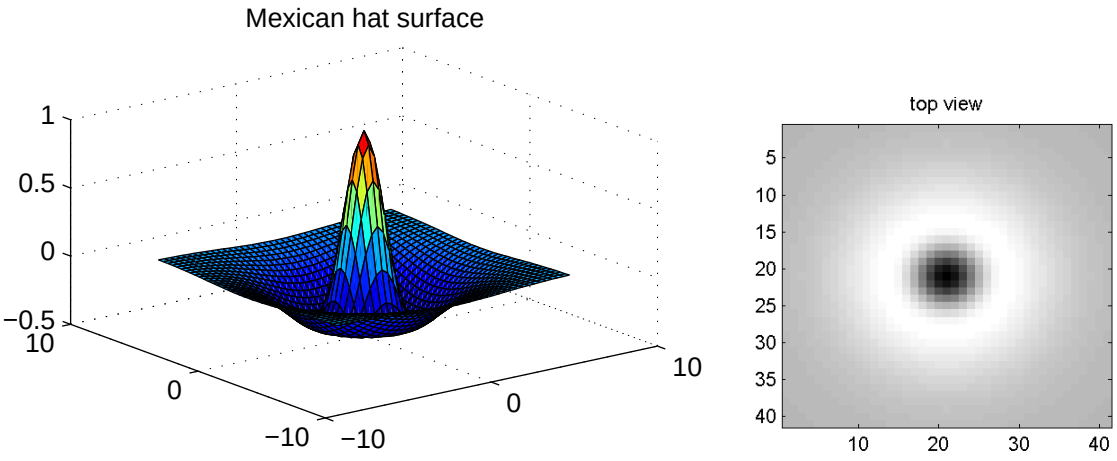
If you have ever heard of **complex numbers** you would be pleased to know that MATLAB knows about them too. Just try to type `j` or `i` or `sqrt(-1)` to get the idea what is available. We will try not to use complex numbers too often, but on occasion you can get MATLAB warnings regarding them.

The first plot we produced was about a function (sin) of a single variable x. Such a function is visualised as a line in **2-D space**. MATLAB can also do functions of two variables. Such a function is visualised as a surface in a **3-D space**.
As an example we consider a "Mexican hat" that we will be often referred to in our course. The function is obtained as a difference between two bell-shaped Gaussian functions.

```
x1 = -2:2
x2 = -3:3
[X Y] = meshgrid(x1, x2)
R2 = X.^2 + Y.^2
sgp = 2, sgm = 18, am = 0.4
mxH = (exp(-R2/sgp) - am*exp(-R2/sgm))/(1-am)
figure(1)
surf(x1, x2, mxH), grid on, title('Mexican hat surface')
figure(2)
imagesc(mxH), title('top view')
```

Exercise 1. 9

1. Modify the values of the program variables to get a plot similar to the following:



2. Produce your own plot of an interesting surface.

□

1.3 Basic properties of typical activation functions

Exercise 1. 10

1. Derive analytical expressions (pen-and-paper method) for derivatives $\frac{d\varphi}{dv}$ of the following activation functions $y = \varphi(v)$:

   Unipolar sigmoidal:

   $$\varphi(v) = \frac{1}{1 + e^{-\beta v}}$$

   Hyperbolic tangent:

   $$\varphi(v) = \tanh(\beta v)$$

   Compare it with the sigmoidal function.

   Gaussian:

   $$\varphi(v) = \exp\left(-\frac{v^2}{2\sigma^2}\right)$$

   Rational:

   $$\varphi(v) = \frac{v}{\beta + |v|}$$

2. Plot the derivatives using MATLAB.

□

1.4 Perceptron

Exercise 1. 11
Study the perceptron learning law as demonstrated in the following MATLAB M-files:
`$CSE5301/Mtlb/perc.m`
and
`$NNET/nndemos/demop1.m, demop4.m, demop5.m, demop6.m`.
where
`$CSE5301 = http://www.csse.monash.edu.au/courseware/cse5301`
`$NNET = $MATLAB/toolbox/nnet`
□

Exercise 1. 12
Write a MATLAB M-file, similar to `perc.m` and the other demo files, which implements the perceptron learning law:

- use 4-D augmented input patterns ($p = 4$)

- plot the projections of input patterns and separation planes in all three 2-D projection planes, $x_1 - x_2$, $x_2 - x_3$, $x_3 - x_1$.

  Use the subplot command.

  Plot also in the above figure the paths of the tip of the normalised weight vector during training.

- Use the modified learning rule as described in the lecture notes.

Clearly indicate sections of code that you wrote yourself.
□

Exercise 1. 13
Implement a non-trivial logic function of your choice of three variables: $y = f(x_1, x_2, x_3)$ using a single perceptron. Give a **boolean expression** describing your function. Derive the **weight vector** using two methods:

- by working out the equation of the decision plane,
- by training using the perceptron learning law.

Remember that a logic function of three variables may be visualised by a 3-D cube, each vertex being labeled with the value of function. You must have a logic function which values are linearly separable. This is equivalent to bisecting the cube with a plane with zeros and ones of the function on the opposite side of the plane.
□

In **your submission** include answers/solutions to all exercises.