Octave Tutorial

Basic operations

- Mathematical operations:
  - Addition (`+`), subtraction (`-`), multiplication (`*`), division (`/`).
- Logical operations:
  - Equality (`==`), inequality (`~=`), greater-than (`>=`, `>`) and lesser-than (`<=`, `<`), logical and (`&&`), logical or (`||`).
- Assignment:
  - `a = 3` or `a = 3;`;
  - Semicolon suppresses the output.
- Constant:
  - *E. g.*: `pi`.
- Matrices and vectors:
  - `A = [1 2; 3 4; 5 6]`: 3×2 matrix;
  - `v = [1; 2; 3]`: column vector;
  - `u = [1 2 3]`: row vector.
- Ranges:
  - `v = 1:0.1:2`: linearly spaced row vector from 1 to 2 (inclusive).
  - The middle argument is optional.
- Std. matrices:
  - `zeros(m,n)`: an $m \times n$ matrix of zeros;
  - `ones(m,n)`: an $m \times n$ matrix of ones;
  - `rand(m,n)`: an $m \times n$ (uniformly) random matrix (between 0 and 1);
  - `randn(m,n)`: an $m \times n$ (normally) random matrix (zero mean and variance equal to one);
  - `eye(n)`: the $n \times n$ identity matrix;
- Functions:
  - `disp()`: print out;
  - `hist()`: histogram;
  - `help`: help for a particular command;
  - `size()`: returns the size of a matrix (as a matrix itself);
  - `length()`: returns the size of the longer dimension.

Moving data around

- `load(<filename>.<ext>)`: loads the file:
  - It is assigned to a variable named `<filename>`.
- `who`: shows all variables loaded in the workspace:
  - `whos`: also shows info on the variables.
- `clear <var>`: removes the variable from the workspace:
  - `clear`: removes all variables.

- `save <filename>.<ext> <var>;`: saves the variable to the informed file:
    - Appending `-ascii`: saves as text.
- `A(i,j)`: returns the element in the `i`-th row, `j`-th column:
    - `A(i,:)`: whole `i`-th row;
    - `A(:,j)`: whole `j`-column;
    - `A([i m],:)`: whole `i`- and `m`-th row;
    - `A = [A, <col_vector>]`: appends `col_vector` to the right (comma is optional);
    - `A(:)`: puts all elements to a column vector;
    - `C = [A B]`: concatenates the matrices horizontally;
    - `C = [A ; B]`: concatenates the matrices vertically.

Computing on data

- `A * B`: matrix multiplication:
    - The number of columns of `A` must match the number of rows of `B`;
- `A .* B`: element-wise multiplication:
    - Both matrices must have the same dimensions;
- `A .^ p`: element-wise exponentiation;
- `A ./ n`: element-wise division:
    - `1 ./ A`: element-wise reciprocal;
- `log(v)`: element-wise logarithm;
- `exp(v)`: element-wise exponential ($\exp\{v_i\}$);
- `abs(v)`: element-wise absolute value;
- `A'`: transpose of `A`;
- `[val, ind] = max(v)`: returns maximum value of `A` and its index;
    - For matrices, it works column-wise;
- `a < 4`: returns a "boolean" matrix, with the same dimensions as `a`;
- `find(a < 3)`: returns a vector with the indices in which the condition is true;
- `[r,c] = find(A >= 7)`: returns vector with the rows (`r`) and columns (`c`) in which the condition is true;
- `magic(n)`: returns an $n \times n$ magic matrix;
- `sum(a)`: returns the sum of elements:
    - `sum(sum(A.*eye(n)))`: sums up the main diagonal of `A`, an $n \times n$ matrix;
    - `sum(sum(A.*flipud(eye(n))))`: sums up the secondary diagonal of `A`, an $n \times n$ matrix;
- `prod(a)`: returns the product of elements;
- `floor(a)`: rounds down the elements;
- `ceil(a)`: rounds up the elements;
- `max(A,[],1)`: returns the maximum values, column-wise (as row vector):
    - `max(A,[],2)`: returns the maximum values, row-wise (as column vector);
    - `max(max(A))` or `max(A(:))`: returns the global maximum.
- `inv(A)`: returns the inverse of the (square) matrix `A`:
    - `pinv(A)`: returns the pseudo-inverse of matrix `A`.

Plotting the data

```
t = [0:0.01:0.98];
y1 = sin(2*pi*4*t);
y2 = cos(2*pi*4*t);
plot(t,y1);
hold on;
plot(t,y2, 'r');
xlabel('time')
ylabel('value')
legend('sin', 'cos')
title('My plot')
cd '<path_to_dir>'; print -dpng 'my_plot.png'
close
```

```
figure(1); plot(t,y1);
figure(2); plot(t,y2)
```

```
subplot(1,2,1); % divides plot in a 1x2 grid
plot(t,y1);
subplot(1,2,2)
plot(t,y2);
axis([0 1 -1 1]) % xmin xmax ymin ymax
```

```
A = magic(5);
imagesc(A), colorbar, colormap gray; % comma-chaining
```

Control statements

Loops

```
v = zeros(10,1);
for i=1:10,
    v(i) = 2^i;
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%
indices = 1:10;
for i=indices,
    v(i) = 2^i;
end;
```

```
i = 1;
while i <= 5,
    v(i) = 100.
    i = i + 1;
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%
i=1;
while true,
    v(i) = 999;
    i = i + 1;
    if i == 6,
```

```
        break;
    end;
end;
```

Conditional

```
if v(1) == 1,
    disp('The value is one.');
elseif v(1) == 2,
    disp('The value is two.');
else
    disp('The value is not one nor two.');
end;
```

Functions

In a `.m` file:

```
function y = squareThisNumber(x)
y = x^2;
```

Returning more than one value:

```
function [y1, y2] = squareAndCubeThisNumber(x)
y1 = x^2;
y2 = x^3;
```

- The function is to be called from inside the directory where the `.m` file is located.

Cost function:

```
function J = costFunctionJ(X, y, theta)
% X is the design matrix, containing the training examples
% y is the class labels

m = size(X,1); % num. of training examples
predictions = X*theta; % hypothesis
sqErrors = (predictions-y).^2; % squared errors

J = 1/(2*m) * sum(sqErrors);
```

Example:

```
X = [1 1; 1 2; 1 3];
y = [1; 2; 3];
theta = [0; 1];

j = costFunctionJ(X, y, theta)
```

Vectorization

Hypothesis:

$$
\begin{aligned}
h_\theta &= \sum_{j=0}^{n} \theta_j x_j \\
&= \boldsymbol{\theta}^T \mathbf{x}
\end{aligned}
$$

Un-vectorized implementation

```
prediction = 0.0;
for j = 1:n+1,
    prediction = prediction + theta(j) * x(j)
end;
```

Vectorized implementation

```
prediction = theta' * x % inner product
```

Gradient descent:

$$\theta_j := \theta_j + \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)}, \quad \forall j : j \in \{1, 2, \dots n\}$$

Vectorized implementation

```
delta = (h(X) - y)' * X;
% X: (m, (n+1)) matrix
% y: (m, 1) vector
theta = theta - alpha * delta'/m;
```