

Contents

1	Introduction to Machine Learning	3
1.1	Supervised and Unsupervised Learning	3
2	Linear Regression	4
2.1	Linear Regression with one variable	4
2.1.1	Cost Function	4
2.1.2	Gradient Descent	4
2.1.3	Gradient Descent for linear regression	5
2.2	Multivariate Linear Regression	5
2.2.1	Multiple Features	5
2.2.2	Gradient Descent for Multiple Variables	5
2.3	Normal equation	6
3	Logistic Regression	8
3.1	Classification and Representation	8
3.1.1	Classification	8
3.1.2	Hypothesis Representation	8
3.1.3	Decision Boundary	9
3.2	Logistic Regression Model	9
3.2.1	Cost Function	9
3.3	Multiclass Classification	10
3.3.1	One-vs-all	10
3.4	The Problem of Overfitting	10
3.4.1	Regularized Cost Function	11
3.4.2	Regularized Linear Regression	11
4	Neural Networks	13
4.1	Intutions	14
4.1.1	OR Function	14
4.1.2	Important Note	14
4.2	Multiclass Classification	15
4.3	Cost Function	15
4.4	Backpropagation Algorithm	15

4.5	Gradient Checking	16
5	Octave/MATLAB Commands	17

Chapter 1

Introduction to Machine Learning

Machine Learning is undeniably one of the most influential and powerful technologies in today's world. It is a tool for turning information into knowledge. In the past 50 years, there has been an explosion of data. This mass of data is useless unless we analyse it and find the patterns hidden within. Machine learning techniques are used to automatically find the valuable underlying patterns within complex data that we would otherwise struggle to discover. The hidden patterns and knowledge about a problem can be used to predict future events and perform all kinds of complex decision making.

There are multiple forms of Machine Learning; supervised, unsupervised, semi-supervised and reinforcement learning. Each form of Machine Learning has differing approaches, but they all follow the same underlying process and theory. I'll cover supervised and unsupervised learning for my report.

1.1 Supervised and Unsupervised Learning

The most basic thing to remember is that we already know what our correct output should look like in Supervised Learning. But, we have little or no idea about what our results should look like.

Supervised Learning:

- Classification: Spam/Not-spam.
- Regression: Predicting age.

Unsupervised Learning:

- Clustering: Grouping based on different variables.
- Non Clustering: Finding structure in chaotic environment.

Chapter 2

Linear Regression

2.1 Linear Regression with one variable

Regression being a part of Supervised Learning is used for estimating data (Real-valued output).

2.1.1 Cost Function

This function measures the performance of a Machine Learning model for given data.

Hypothesis: $h_{\theta}(x) = \theta_0 + \theta_1 x$

Parameters: θ_0, θ_1

Cost Function:

$$J(\theta_0, \theta_1) = 1/2m \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \quad (2.1)$$

Goal: Minimize cost function with θ_0, θ_1 as parameters.

2.1.2 Gradient Descent

Basic idea:

- Start with some θ_0, θ_1
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we end up at minima.

Algorithm: repeat until convergence:

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_j} \quad (2.2)$$

(for $j = 0, 1$, here).

Intuition: If α is too small, descent can be slow and if too large, descent may fail to converge or even diverge. Gradient descent can converge to a local minimum, even with fixed learning rate α . As we approach local minimum, gradient descent will automatically take smaller steps. So, no need to decrease α over time.

2.1.3 Gradient Descent for linear regression

Combining gradient descent algorithm with linear regression model, we get:

$$j = 0 : \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_0} = 1/2 \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \quad (2.3)$$

$$j = 1 : \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_1} = 1/2 \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)} \quad (2.4)$$

Now, we can repeat 2.3 and 2.4 until convergence to obtain the minima.

"Batch" gradient descent: Each step of gradient descent uses all the training examples. For eq. "m" batches in equation 2.1.

2.2 Multivariate Linear Regression

Linear regression involving more than one variable. For eq., Predicting price of a house based on parameters "Plot Area", "No. of Floors", "Connectivity with markets", etc.

2.2.1 Multiple Features

The multivariable form of the hypothesis is as follows:

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \dots + \theta_n x_n. \quad (2.5)$$

This hypothesis function can be concisely represented as:

$$h_{\theta}(x) = \theta^T x \quad (2.6)$$

where, θ^T is a $1 \times n$ matrix consisting of $\theta_0, \theta_1, \theta_2, \dots, \theta_n$.

2.2.2 Gradient Descent for Multiple Variables

Gradient descent formula for Multiple variable will be similar to that of single variable.

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \quad (2.7)$$

Repeating this equation until convergence will give the minima. ¹

¹ $x_0 = 1$ in equation 2.7

Feature Scaling

Feature Scaling is used to reduce the number of iterations in Gradient Descent. Basic idea of feature scaling is to bring all the features on the same scale. (in general we try to approximate every feature in the range $-1 < x_i < 1$)

Reducing the number of iteration doesn't mean making computation of each step easier. And also it does not effect comtational efficiency of Normal Equation.

Mean Normalisation

Mean Normalisation makes features to have approximately zero mean.

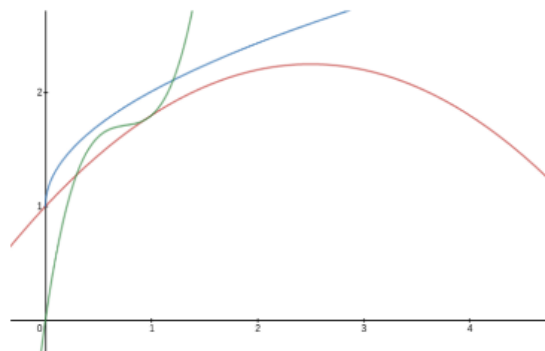
Learning Rate

If α is too small: slow convergence.

if α is too large: $J(\theta)$ may not decrease on every iteration, or may not converge.

Polynomial Regression

Selecting proper polynomial for fitting data is very important.



Red: Quadratic

Blue: Square root funtion $\theta_0 + \theta_1 x + \theta_2 \sqrt{x}$

Green: Cubic function

2.3 Normal equation

Normal Equation is a method to solve for θ_T analytically, by creating a $m \times (n + 1)$ matrix X and another $m \times 1$ matrix Y .²

²Every element of first column of matrix X is 1 and other are the feature's coefficient

Mathematically θ is given as:

$$\theta = (X^T X)^{-1} X^t y \quad (2.8)$$

Gradient Descent	Normal Equation
Need to choose α	No need to choose α
Needs many iteration	Don't need to iterate
Works well with large n	Slow for large n

Reasons for non-invertibility of $X^T X$

- Redundant features (linear dependence) ³
- Too many features ($m \leq n$)

³Eg. Using both m^2 & $(feet)^2$ features

Chapter 3

Logistic Regression

3.1 Classification and Representation

3.1.1 Classification

The classification problem is just like the regression problem, except that the values we now want to predict take on only a small number of discrete values. For now, we'll discuss binary classification problem.

3.1.2 Hypothesis Representation

We may use our old regression algorithm by classifying data on the basis of a threshold. But it will have very poor performance.

We will introduce "Sigmoid Function", also called the "Logistic Function":

$$h_{\theta}(x) = g(\theta^T x) \quad (3.1)$$

$$z = \theta^T x \quad (3.2)$$

$$g(z) = \frac{1}{1 + e^{-z}} \quad (3.3)$$

This is how the Sigmoid Function looks like:



Figure 3.1: Sigmoid Function 3.3

3.1.3 Decision Boundary

The decision boundary is the line that separates the area where $y=0$ and where $y=1$. It is similar to the decision boundary for linear regression, the only difference is distribution of values (linear and sigmoid)

3.2 Logistic Regression Model

3.2.1 Cost Function

Cost function for logistic regression looks like:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_{\theta}(x^{(i)}), y^{(i)}) \quad (3.4)$$

$$\text{Cost}(h_{\theta}(x), y) = -\log(h_{\theta}(x)) \quad \text{if } y = 1$$

$$\text{Cost}(h_{\theta}(x), y) = -\log(1 - h_{\theta}(x)) \quad \text{if } y = 0$$

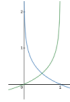


Figure 3.2: Cost Function

Simplified Cost Function

This cost function can be compressed into a single function:

$$\text{Cost}(h_{\theta}(x), y) = -y \log(h_{\theta}(x)) - (1 - y) \log(1 - h_{\theta}(x)) \quad (3.5)$$

A vectorised implementation is:

$$h = g(X\theta) \\ J(\theta) = \frac{1}{m} \cdot (-y^T \log h - (1 - y)^T \log(1 - h))$$

Vectorised implementation for Gradient Descent:

$$\theta := \theta - \frac{\alpha}{m} X^T (g(X\theta) - y)$$

3.3 Multiclass Classification

3.3.1 One-vs-all

This approach is when data has more than two categories. We divide our problem into n^1 binary classification problems, in each one, we predict the probability considering one of the category to be +ve and all other to be -ve. Repeating this for all other categories will finally give us all the decision boundaries.

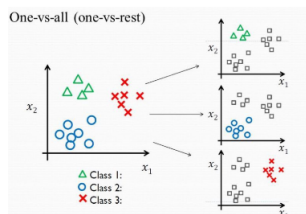
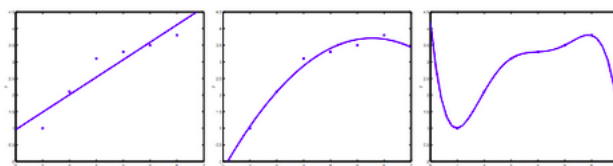


Figure 3.3: One vs all classification method

3.4 The Problem of Overfitting

Consider the problem of predicting y from $x \in \mathbb{R}$. The leftmost figure below shows the result of fitting a $y = \theta_0 + \theta_1 x$ to a dataset. We see that the data doesn't really lie on straight line, and so the fit is not very good.



Instead, if we had added an extra feature x^2 , and fit $t = \theta_0 + \theta_1 x + \theta_2 x^2$, then we obtain a slightly better fit to the data (See middle figure). Naively, it might seem that the more features we add, the better. However, there is also a danger in adding too many features: The rightmost figure is the result of fitting a 5th order polynomial $y = \sum_{j=0}^5 \theta_j x^j$. We see that even though the fitted curve passes through the data perfectly, we would not expect this to be a very good predictor of, say, housing prices (y) for different living areas (x). Without formally defining what these terms mean, we'll say the figure on the left shows an instance of **underfitting**—in which the data clearly shows structure not captured by the model—and the figure on the right is an example of **overfitting**.

¹ n = no of categories in dataset

How to address this issue?

1. Reduce the number of features:
 - Manually select which features to keep.
 - Use a model selection algorithm.²
2. Regularisation:
 - Keep all the features, but reduce the magnitude of parameters θ_j .
 - Regularization works well when we have a lot of slightly useful features.

3.4.1 Regularized Cost Function

To solve this problem of overfitting, we can eliminate the influence of $\theta_3 x^3$ and $\theta_4 x^4$. Without actually getting rid of these features or changing the form of our hypothesis, we can instead modify our cost function:

$$J_{\theta} = \min \text{ of } \left[\frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^2 \theta_j^2 \right] \quad (3.6)$$

These extra terms will inflate the cost of extra parameters.

The λ is called the **regularisation parameter**/ It determines how much the costs of our theta parameters are inflated.

3.4.2 Regularized Linear Regression

Repeat {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

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

}

The term $\frac{\lambda}{m} \theta_j$ performs our regularization. With some manipulation our update rule can also be represented as:

²we'll cover it later

$$\theta_j := \theta_j(1 - \alpha \frac{\lambda}{m}) - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})x_j^{(i)}$$

The first term in the above equation, $\alpha \frac{\lambda}{m}$ will always be less than 1. Intuitively you can see it as reducing the value of $j\theta_j$ by some amount on every update.

Normal Eequation

This will be the non-iterative approach for regularazation.

To add in regularization, we'll just add another term:

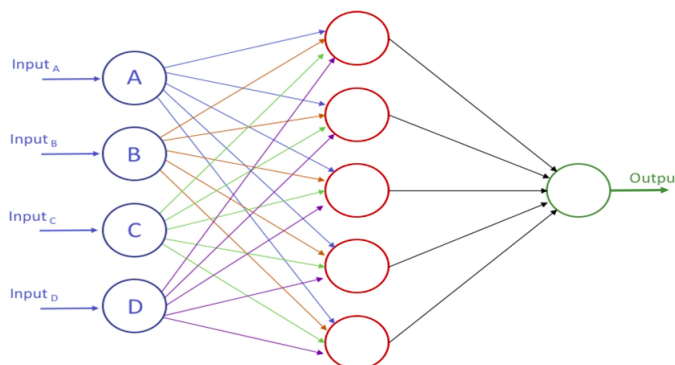
$$\theta = X^T X + \lambda.L^{-1}X^T y \tag{3.7}$$

where, L is $(n + 1) \times (n + 1)$ matrix with 0 at the top lest, 1's down the diagonal and all other element '0'.

Chapter 4

Neural Networks

At a very simple level, neurons are basically computational units that take inputs (**dendrites**) as electrical inputs (called "spikes") that are channeled to outputs (**axons**). In our model, our dendrites are like the input features $x_1...x_n$, and the output is the result of our hypothesis function. In this model our x_0 input node is sometimes called the "bias unit." It is always equal to 1.



Our input nodes (layer 1), also known as the "input layer", go into another node (layer 2), which finally outputs the hypothesis function, known as the "output layer". We can have intermediate layers of nodes between the input and output layers called the "hidden layers."

These "hidden layer" nodes are called as "activation units". The values for each activation nodes are represented as:

$$\begin{array}{c} x_0 \quad a_1^{(2)} \\ x_1 \rightarrow a_2^{(2)} \rightarrow h_{\theta}(x) \\ x_2 \quad a_3^{(2)} \\ x_3 \end{array}$$

$$a_1^{(2)} = g(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3) \quad (4.1)$$

$$a_2^{(2)} = g(\theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3) \quad (4.2)$$

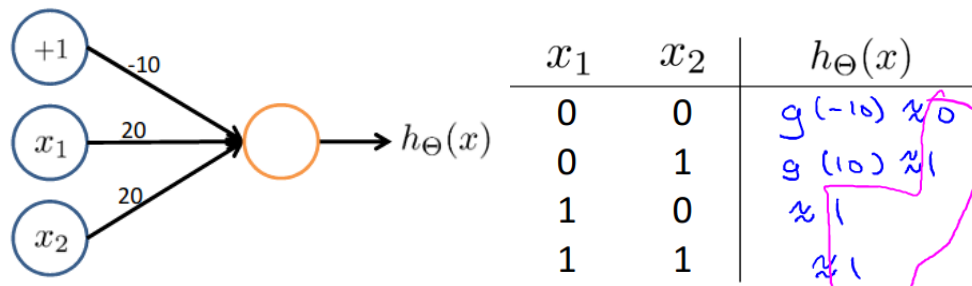
$$a_3^{(2)} = g(\theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3) \quad (4.3)$$

$$h_\theta(x) = a_1^{(3)} = a_1^{(2)} = g(\theta_{10}^{(2)} x_0 + \theta_{11}^{(2)} x_1 + \theta_{12}^{(2)} x_2 + \theta_{13}^{(2)} x_3) \quad (4.4)$$

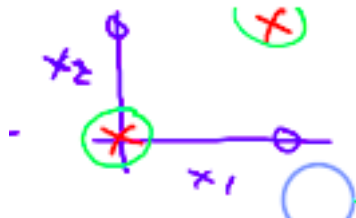
From these equations, we can conclude that we will get a matrix for each layer to calculate the weight for the second layer.

4.1 Intutions

4.1.1 OR Function



4.1.2 Important Note



For any prediction which involve a straight line as decision boundary, we can represent it with a neural network without any hidden layer but otherwise we'll have to include few hidden layers. An important point to note is that we can represent almost any distribution with cenrain arrangement of neural network.

4.2 Multiclass Classification

To classify data into multiple classes, we'll have to define out set of resulting classes as y :

$$y^{(i)} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix},$$

4.3 Cost Function

Let's first define a few variables that we'll need to use:

- L = total number of layers in the network
- numbers of units (non counting bias unit) in layer 1
- K = numbet of output unit/classes

Cost function for neural networks will be slightly more complicated as it involves few other factors of ' K ' and ' L ' defined earlier.

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log((h_{\theta}(x^{(i)}))_k) + (1 - y_k^{(i)}) \log(1 - (h_{\theta}(x^{(i)}))_k) + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\theta_{j,i}^{(L)})^2 \quad (4.5)$$

Note:

- the double sum simply adds up the logistic regression costs calculated for each cell in the output layer.
- the triple sum simply adds up the squares of all the individual θ s in the entire network.
- the i in the triple sum does not refer to training example i .

4.4 Backpropagation Algorithm

Given training set $(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})$

- Set $\Delta_{i,j}^{(l)} := 0$ for all (l, i, j) , (hence you end up having a matrix full of zeros)

For training example $t=1$ to m :

1. Set $a^{(1)} := x^{(t)}$
2. Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \dots, L$

Gradient Computation

3. Using $y^{(t)}$, compute $\delta^{(L)} = a^{(L)} - y^{(L)}$
4. Compute $\delta^{(l)}$
5. $\Delta^{(L)} := \Delta^{(L)} + \delta^{(l+1)}(a^{(l)})^T$

Hence we update our new Δ matrix.

- $D_{i,j}^{(l)} := \frac{1}{m} \Delta_{i,j}^{(l)} + \lambda \Theta_{i,j}^{(l)}$, if $j \neq 0$.
- $D_{i,j}^{(l)} := \frac{1}{m} \Delta_{i,j}^{(l)}$ if $j = 0$

4.5 Gradient Checking

Gradient checking will assure that our backpropagation works as intended. We can approximate the derivative of our cost function with:

$$\frac{\partial}{\partial \Theta} J(\Theta) \approx \frac{J(\Theta + \epsilon) - J(\Theta - \epsilon)}{2\epsilon}$$

And for multiple features,

$$\frac{\partial}{\partial \Theta_j} J(\Theta) \approx \frac{J(\Theta_1, \dots, \Theta_j + \epsilon, \dots, \Theta_n) - J(\Theta_1, \dots, \Theta_j - \epsilon, \dots, \Theta_n)}{2\epsilon}$$

4.6 Putting it Together

First, pick a network architecture; choose the layout of your neural network, including how many hidden units in each layer and how many layers in total you want to have.

- Number of input units = dimension of features $x^{(i)}$
- Number of output units = number of classes

- Number of hidden units per layer = usually more the better (must balance with cost of computation as it increases with more hidden units)
- Defaults: 1 hidden layer. If you have more than 1 hidden layer, then it is recommended that you have the same number of units in every hidden layer.

Training a Neural Network

1. Randomly initialize the weights
2. Implement forward propagation to get $h_{\theta}(x^{(i)})$ for any $x^{(i)}$
3. Implement the cost function
4. Implement backpropagation to compute partial derivatives
5. Use gradient checking to confirm that your backpropagation works. Then disable gradient checking.
6. Use gradient descent or a built-in optimization function to minimize the cost function with the weights in theta.

Chapter 5

Octave/MATLAB Commands

Basic Operations

```
octave:1> a = pi
a = 3.1416
octave:2> disp(sprintf('6 decimals: %0.6f', a))
6 decimals: 3.141593
octave:3> a
a = 3.1416
octave:4> format long
octave:5> a
a = 3.141592653589793
octave:6> format short
octave:7> a
a = 3.1416
octave:8> v = 1:0.1:2
v =

    1.0000    1.1000    1.2000    1.3000    1.4000    1.5000
    1.6000    1.7000    1.8000    1.9000    2.0000

octave:9> v = 1:0.1:2
v =

Columns 1 through 8:

    1.0000    1.1000    1.2000    1.3000    1.4000    1.5000
    1.6000    1.7000

Columns 9 through 11:

    1.8000    1.9000    2.0000
```

```

octave:10> v = 1:6
v =

    1    2    3    4    5    6

octave:11> zeros(1,3)
ans =

    0    0    0

octave:12> rand(1,3)
ans =

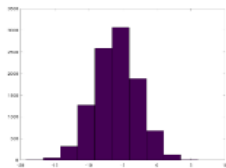
    0.43623    0.76554    0.23635

octave:13> randn(1,3)
ans =

    0.5602642   -0.0043628    0.1344922

octave:14> w = -6 + sqrt(10)*(randn(1,10000))
octave:15> hist(w)

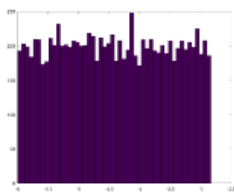
```



```

octave:1> w = -6 + sqrt(10)*(rand(1,10000));
octave:2> hist(w,50)

```



Moving Data around

```
octave:1> A = [1,2;3,4;4,5]
A =

     1     2
     3     4
     4     5

octave:2> size(A)
ans =

     3     2

octave:3> sz = size(A)
sz =

     3     2

octave:4> size(sz)
ans =

     1     2

octave:5> size(A,1)
ans = 3
octave:6> size(A,2)
ans = 2
octave:7> length(A)
ans = 3
octave:8> length([1,2,3,4,5])
ans = 5
octave:9>
octave:9>
octave:9> pwd
ans = /home/sahasra
octave:10> cd /home/sahasra/
octave:11> pwd
ans = /home/sahasra
octave:12> ls
Android          Documents      Music          Public          Videos
AndroidStudioProjects Downloads      MyPaint        snap
Desktop          examples.desktop Pictures        Templates
octave:13> who
Variables in the current scope:
```

```
A      ans    sz
```

```
octave:14> whos
```

```
Variables in the current scope:
```

Attr	Name	Size	Bytes	Class
	A	3x2	48	double
	ans	1x13	13	char
	sz	1x2	16	double

```
Total is 21 elements using 77 bytes
```

```
octave:15> clear
```

```
octave:16> whos
```

```
octave:17> A = [1,2;3,4;5,6]
```

```
A =
```

```
1    2
3    4
5    6
```

```
octave:18> A(3,2)
```

```
ans = 6
```

```
octave:19> A(2,:) 
```

```
ans =
```

```
3    4
```

```
octave:20> A(:,2)
```

```
ans =
```

```
2
4
6
```

```
octave:21> A([1,3], :)
```

```
ans =
```

```
1    2
5    6
```

```
octave:22> A([2,3], :)
```

```
ans =
```

```

      3      4
      5      6

octave:23> A(:,2) = [10;11;12]
A =

      1      10
      3      11
      5      12

octave:24> A = [A, [5;6;7]]
A =

      1      10      5
      3      11      6
      5      12      7

octave:25> A(:)
ans =

      1
      3
      5
     10
     11
     12
      5
      6
      7

octave:26> A
A =

      1      10      5
      3      11      6
      5      12      7

octave:27> B = [45;46;47]
B =

     45
     46
     47

```

```
octave:28> C = [A,B]
C =
```

```
    1    10     5    45
    3    11     6    46
    5    12     7    47
```

Computing on Data

```
octave:1> A = [1 2;3 4;5 6]
A =
```

```
    1     2
    3     4
    5     6
```

```
octave:2> B = [11 12; 13 14; 15 16]
B =
```

```
    11    12
    13    14
    15    16
```

```
octave:3> C = [1 1; 2 2]
C =
```

```
    1     1
    2     2
```

```
octave:4> A*C
ans =
```

```
     5     5
    11    11
    17    17
```

```
octave:5> A .* B % A .* B gives element wise operation
ans =
```

```
    11    24
    39    56
    75    96
```

```
octave:6> 1 ./ A
ans =
```

```

    1.00000    0.50000
    0.33333    0.25000
    0.20000    0.16667

octave:7> v = [1;2;3]
v =

    1
    2
    3

octave:8> log(v)
ans =

    0.00000
    0.69315
    1.09861

octave:9> exp(v)
ans =

    2.7183
    7.3891
   20.0855

octave:10> abs([-1; 2; -3])
ans =

    1
    2
    3

octave:11> A
A =

    1    2
    3    4
    5    6

octave:12> A' % A' = A transpose
ans =

    1    3    5
    2    4    6

```



```

octave:13> val = max([1;2;3;6;7])
val = 7
octave:14> max(A)
ans =

    5    6

octave:15> A
A =

    1    2
    3    4
    5    6

octave:16> a = [1;4;6;7;9]
a =

    1
    4
    6
    7
    9

octave:17> a < 3
ans =

    1
    0
    0
    0
    0

octave:18> find(a<3)
ans = 1
octave:19> A = magic(3) % Magic Square
A =

    8    1    6
    3    5    7
    4    9    2

octave:20> [r,c] = find(a >= 7)
r =

    4

```

```

5

c =

    1
    1

octave:21> a
a =

    1
    4
    6
    7
    9

octave:22> a = a'
a =

    1    4    6    7    9

octave:23> sum(a)
ans = 27
octave:24> rand(3)
ans =

    0.272471    0.059338    0.757392
    0.414497    0.174242    0.354694
    0.811891    0.935437    0.956667

octave:25> A
A =

    8    1    6
    3    5    7
    4    9    2

octave:26> max(A,[],1)
ans =

    8    9    7

octave:27> max(A,[],2)
ans =

```

```

8
7
9

octave:28> max(max(A))
ans = 9
octave:29> A
A =

    8    1    6
    3    5    7
    4    9    2

octave:30> pinv(A)
ans =

    0.147222   -0.144444    0.063889
   -0.061111    0.022222    0.105556
   -0.019444    0.188889   -0.102778

octave:31> temp = pinv(A)
temp =

    0.147222   -0.144444    0.063889
   -0.061111    0.022222    0.105556
   -0.019444    0.188889   -0.102778

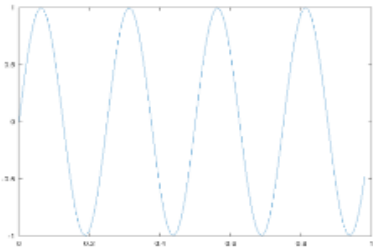
octave:32> temp * A
ans =

    1.0000e+00    2.0817e-16   -3.1641e-15
   -6.1062e-15    1.0000e+00    6.2450e-15
    3.0531e-15    4.1633e-17    1.0000e+00

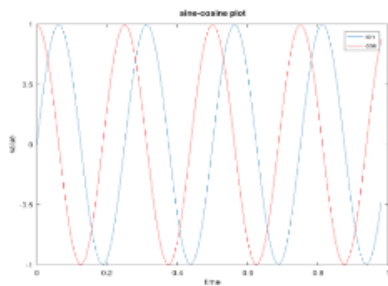
octave:33> % this is the 3x3 Identity matrix,
           % not having exact values beacuse of variable overflow

octave:1> t = [0:0.01:0.98];
octave:2> y1 = sin(2*pi*4*t);
octave:3> plot(t,y1)

octave:4> y2 = cos(2*pi*4*t);
octave:5> plot(t,y1);
octave:6> hold on;
```



```
octave:7> plot(t, y2, 'r');
octave:8> xlabel('time')
octave:9> ylabel('value')
octave:10> legend('sin', 'cos')
octave:11> title('sine-cosine plot')
octave:12> print -dpng 'myPlot.png'
```

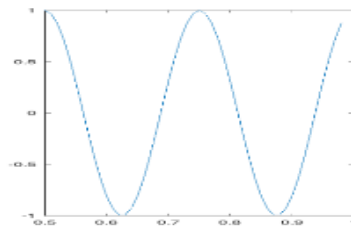
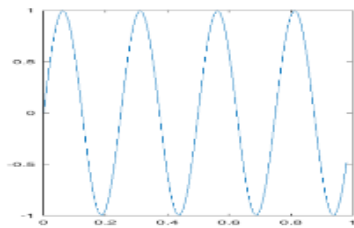


```
octave:13> close
octave:14> figure(1); plot(t, y1);
octave:15> figure(2); plot(t, y2);
octave:16> subplot(1,2,1); % Divides plot a 1x2 grid
octave:17> plot(t,y1);
octave:18> subplot(1,2,2)
octave:19> plot(t,y2);
octave:20> axis([0.5 1 -1 1])
```

Control Statements: for, while, if, else-if ...

```
octave:1> v = zeros(10,1)
v =

    0
    0
```



```

0
0
0
0
0
0
0
0
0
0

octave:2> for i=1:10,
> v(i) = 2^i;
> end;

octave:3> v
v =

    2
    4
    8
   16
   32
   64
  128
  256
  512
 1024

octave:4> i=1;
octave:5> while i <= 5,
> v(i) = 100;
> i = i+1;
> end;
octave:6> v
v =
```

```

100
100
100
100
100
  64
128
256
512
1024

octave:7> i = 1;
octave:8> while true,
> v(i) = 999;
> i = i+1;
> if i == 6,
>   break;
> end;
> end;

octave:9> v
v =

  999
  999
  999
  999
  999
   64
  128
  256
  512
 1024

```