Primal/Dual LP Problems (Main Ideas and Examples)

Assume that all primal constraints are **equations** with **non-negative right-hand side**, and all the variables are **non-negative**. Then, we have the following rules for constructing the dual problem

Primal Problem objective	Dual problem		
	Objective	Constraints type	Variable sign
maximization	minimization	$\geq$	unrestricted
minimization	maximization	$\leq$	unrestricted

Key ideas

- Assign a dual variable for each primal (equality) constraint.
- Construct a dual constraint for each primal variable.
- The (column) constraint coefficients and the objective coefficient of the jth primal variable respectively define the left-hand an the right-hand sides of the jth dual constraint.
- The dual objective coefficients equal the right-hand sides of the primal constraint equations.

Example 1

- **Primal problem**

$$\begin{aligned} \text{maximize } z = 5x_1 + 12x_2 + 4x_3 \\ \text{subject to} \quad x_1 + 2x_2 + x_3 \leq 10 \\ 2x_1 - x_2 + 3x_3 = 8 \\ x_1, x_2, x_3 \geq 0 \end{aligned}$$

- Primal in equation form

$$\begin{aligned} \text{maximize } z = 5x_1 + 12x_2 + 4x_3 + 0x_4 \\ \text{subject to} \quad x_1 + 2x_2 + x_3 + x_4 = 10 \\ 2x_1 - x_2 + 3x_3 + 0x_4 = 8 \\ x_1, x_2, x_3, x_4 \geq 0 \end{aligned}$$

- Dual

$$\begin{aligned} \text{minimize } w = 10y_1 + 8y_2 \\ \text{subject to} \quad y_1 + 2y_2 \geq 5 \\ 2y_1 - y_2 \geq 12 \\ y_1 + 3y_2 \geq 4 \\ y_1 + 0y_2 \geq 0 \\ y_1, y_2 \text{ unrestricted} \end{aligned}$$

- **Dual problem**

$$\begin{aligned} \text{minimize } w = 10y_1 + 8y_2 \\ \text{subject to} \quad y_1 + 2y_2 \geq 5 \\ 2y_1 - y_2 \geq 12 \\ y_1 + 3y_2 \geq 4 \\ y_1 \geq 0 \\ y_2 \text{ unrestricted} \end{aligned}$$

Example 2

- **Primal problem**

$$\begin{aligned} \text{minimize } z &= 15x_1 + 12x_2 \\ \text{subject to } \quad x_1 + 2x_2 &\geq 3 \\ 2x_1 - 4x_2 &\leq 5 \\ x_1, x_2 &\geq 0 \end{aligned}$$

- Primal in equation form

$$\begin{aligned} \text{minimize } z &= 15x_1 + 12x_2 + 0x_3 + 0x_4 \\ \text{subject to } \quad x_1 + 2x_2 - x_3 + 0x_4 &= 3 \\ 2x_1 - 4x_2 + 0x_3 + x_4 &= 5 \\ x_1, x_2, x_3, x_4 &\geq 0 \end{aligned}$$

- Dual

$$\begin{aligned} \text{maximize } w &= 3y_1 + 5y_2 \\ \text{subject to } \quad y_1 + 2y_2 &\leq 15 \\ 2y_1 - 4y_2 &\leq 12 \\ -y_1 + 0y_2 &\leq 0 \\ 0y_1 + y_2 &\leq 0 \\ y_1, y_2 &\text{ unrestricted} \end{aligned}$$

- **Dual problem**

$$\begin{aligned} \text{maximize } w &= 3y_1 + 5y_2 \\ \text{subject to } \quad y_1 + 2y_2 &\leq 15 \\ 2y_1 - 4y_2 &\leq 12 \\ y_1 &\geq 0 \\ y_2 &\leq 0 \end{aligned}$$

Example 3

- **Primal problem**

$$\begin{aligned} \text{maximize } z &= 5x_1 + 6x_2 \\ \text{subject to } \quad x_1 + 2x_2 &= 5 \\ -x_1 + 5x_2 &\geq 3 \\ 4x_1 + 7x_2 &\leq 8 \\ x_1 &\text{ unrestricted, } , x_2 \geq 0 \end{aligned}$$

- Primal equation form (here $x_1 = x_1^- - x_1^+$)

$$\begin{aligned} \text{maximize } z &= 5x_1^- - 5x_1^+ + 6x_2 \\ \text{subject to } \quad x_1^- - x_1^+ + 2x_2 &= 5 \\ -x_1^- + x_1^+ + 5x_2 - x_3 &= 3 \\ 4x_1^- - 4x_1^+ + 7x_2 + x_4 &= 8 \\ x_1^-, x_1^+, x_2, x_3, x_4 &\geq 0 \end{aligned}$$

- Dual

$$\begin{aligned} \text{minimize } z &= 5y_1 + 3y_2 + 8y_3 \\ \text{subject to } \quad y_1 - y_2 + 4y_3 &\geq 5 \\ -y_1 + y_2 - 4y_3 &\geq -5 \\ 2y_1 + 5y_2 + 7y_3 &\geq 6 \\ -y_2 &\geq 0 \\ y_3 &\geq 0 \\ y_1, y_2, y_3 &\text{ unrestricted} \end{aligned}$$

- **Dual problem**

$$\begin{aligned} \text{minimize } z &= 5y_1 + 3y_2 + 8y_3 \\ \text{subject to } \quad y_1 - y_2 + 4y_3 &= 5 \\ 2y_1 + 5y_2 + 7y_3 &\geq 6 \\ y_2 &\leq 0 \\ y_3 &\geq 0 \\ y_1 &\text{ unrestricted} \end{aligned}$$

Following the rules listed above, we can use matrix-vector notation to easily find the dual of any linear programming problem (written in standard form).

Primal problem	**Dual problem**
$$\begin{aligned} \text{maximize } & \mathbf{c}^T\mathbf{x} \\ \text{subject to } & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$	$$\begin{aligned} \text{minimize } & \mathbf{b}^T\mathbf{y} \\ \text{subject to } & A^T\mathbf{y} \geq \mathbf{c} \end{aligned}$$
Here A is a $m \times n$ matrix, $\mathbf{c}, \mathbf{x} \in \mathbb{R}^n$, and $\mathbf{b} \in \mathbb{R}^m$. m constraints and n decision variables.	Here $\mathbf{y} \in \mathbb{R}^m$ (dual variable) n constraints and m decision variables.

Primal problem	**Dual problem**
$$\begin{aligned} \text{minimize } & \mathbf{c}^T\mathbf{x} \\ \text{subject to } & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$	$$\begin{aligned} \text{maximize } & \mathbf{b}^T\mathbf{y} \\ \text{subject to } & A^T\mathbf{y} \leq \mathbf{c} \end{aligned}$$

Some useful properties

1. *Any feasible solution to the dual problem gives a bound on the optimal objective function value in the primal problem.*

2. *Understanding the dual problem leads to specialized algorithms for some important classes of linear programming problems.* Examples include the transportation simplex method, the Hungarian algorithm for the assignment problem, and the network simplex method.

3. *The dual can be helpful for sensitivity analysis.* Changing the primal's right-hand side constraint vector or adding a new constraint to it can make the original primal optimal solution infeasible. However, this only changes the objective function or adds a new variable to the dual, respectively, so the original dual optimal solution is still feasible (and is usually not far from the new dual optimal solution).

4. *The dual variables give the shadow prices for the primal constraints.* Suppose you have a profit maximization problem with a resource constraint i. Then the value y_i of the corresponding dual variable in the optimal solution tells you that you get an increase of y_i in the maximum profit for each unit increase in the amount of resource i.

5. *Sometimes the dual is easier to solve.* A primal problem with many constraints and few variables can be converted into a dual problem with few constraints and many variables (the fewer the constraints, the fewer computations required in each iteration of the simplex method).

6. *The dual can be used to detect primal infeasibility.* If the dual is a minimization problem whose objective function value can be made as small as possible, and any feasible solution to the dual gives an upper bound on the optimal objective function value in the primal, then the primal problem cannot have any feasible solutions.

MATLAB and Optimization

1. If you are not familiar with MATLAB please click the following links and watch the tutorial videos:
   (a) Video 1 : Getting Started with MATLAB
   `http://www.mathworks.com/videos/getting-started-with-matlab-68985.html`
   (b) Video 2 : Writing a MATLAB Program
   `http://www.mathworks.com/videos/writing-a-matlab-program-69023.html`

2. Recall the **Klee and Minty** linear programming problem. In its general form, this LP problem is given by

$$\begin{aligned} \max \quad & 2^{n-1}x_1 + 2^{n-2}x_2 + ... + 2x_{n-1} + 1x_n \\ \text{subject to} \quad & 1x_1 + \leq 5 \\ & 4x_1 + 1x_2 \leq 5^2 \\ & 8x_1 + 4x_2 + 1x_3 \leq 5^3 \\ & \vdots \quad \vdots \\ & 2^n x_1 + 2^{n-1}x_2 + ... + 4x_{n-1} + 1x_n \leq 5^n \\ & x_i \geq 0, \ i = 1, ..., n \end{aligned}$$

   The Tableau Simplex Method, starting at $x = (0, 0, ...0)^T$, is known to visit **all extreme points** in this LP.

   We can use the MATLAB built-in solver for linear programming (LP) problems (this is part of the Optimization Toolbox by MathWorks).
   `http://www.mathworks.com/help/optim/ug/linear-programming-algorithms.html`
   We use the function `linprog` in this example.

3. Some things to try:
   (a) Run the MATLAB file (`.m` file) and verify the solution to the Klee and Minty problem for $n = 4$.
   (b) Compare the running time and number of iterations for both methods, **simplex** and **interior-point**: in the `.m` file, modify the line that declares the **method** to be used

```
% Linear Programming Example(LP)
% Goal: use MATLAB command linprog to solve the Klee and Minty problem
% Type help linprog for instructions or visit
% http://www.mathworks.com/help/optim/ug/linprog.html
%
% Test Problem: (n = 4)
% maximize    8*x_1 + 4*x_2 + 2*x_3 + x_4
% subject to     x_1                        <= 5
%              4*x_1 + x_2                  <= 25
%              8*x_1 + 4*x_2 + x_3          <= 125
%             16*x_1 + 8*x_2 + 4*x_3 + x_4 <= 625
%                    x_1 , x_2 , x_3 , x_4 >= 0

% Cost function: recall linprog solves a minimization problem
f = [-8 ; -4; -2; -1];
% Right-hand side:
b = [5; 25; 125; 625];
% Matrix A (constraints) nxn matrix for this LP
A = [1 0 0 0; 4 1 0 0; 8 4 1 0; 16 8 4 1];
% Lower-bounds (non-negativity constraint)
lb = zeros(4,1);

% Optimization solver options: ('simplex'  or 'interior-point')
method = 'interior-point';
options = optimoptions(@linprog,'Algorithm',method);

% Find solution x, f(x), and number of iterations
ub  = [];
Aeq = [];
beq = [];
x0  = [];

% Start timer
tic
[x,fopt,exitflag, output,lambda] = linprog(f,A,b,Aeq,beq,lb,ub,x0,options);
% Stop timer
total_time = toc;
x

fprintf('*****************************************\n\n');
message = strcat(['Optimal point x found. Method used: ', ' ',method]);
fprintf(strcat(message, ' algorithm\n'));
fprintf('f(x) = %f, after %d iterations \n', -fopt,output.iterations)
% Notice that the solution that MATLAB returns must be multiplied by -1
% for our maximization problem
fprintf('Time : %f seconds\n', total_time);
fprintf('*****************************************\n\n');
```

Introduction to MATLAB

MATLAB stands for MATrix LABoratory. It is developed by The Mathworks, Inc. (http://www.mathworks.com). Matlab is an interactive, integrated, environment for numerical computations, symbolic computations, and scientific visualizations. It is a high-level programming language.

- **Quitting Matlab**

To quit Matlab just type ***quit*** in the command window. Caution: if you do this everything that you had typed in the command window will be lost.

- **Runaway or Endless Computation**

A runaway or endless computation happens when you have a program that would not stop of that got stuck. To stop programs like this just use ***ctrl + c.***

- **Help**

To get help just type `help` in the command window and you will have a list of the topics inside help. If you want help in a specific function type `help function_name` and it will give you a short description of the function. (for example, **`help factorial`**). If you feel you need more help click the icon **?** on the Matlab window.

- **Matlab Special Characters**

;	Suppress printing	*	Multiplication
%	Comments	/	Division
+	Addition	\	Solution to A*x=b
-	Subtraction	…	Continue statement on next line

- **Relational Operators**

<	less than	>=	greater than
>	greater than	==	equal
<=	less than or equal	~=	not equal

- **Logical Operators**

& and ~ not | or

- **Constants**

In Matlab you don't need type declaration. To create a constant just type `name=value` where name is the name of the constant and value is the value you want for it. To use a constant you created just type the name of the constant.
To change the value of the constant just type `name=new value`

Examples:

```
A = 3        s = 'hi'        f = 1e-4        alpha = 5.647
```

Also Matlab is *case sensitive*; if you try to use for example the constant `f` but you type `F` it will tell you that the variable does not exist.

- **Operations with Constants**

You can do operation with constants just like you do them in a calculator. You only need to have the constants created before you want to do the operations. Matlab does not give you an error when you want to perform operations on different type of constants. You can also save the result in a new constant.

Example

```
D = A + alpha    in this case  D = 8.6470
E = A*alpha      in this case  E = 16.9410
```

- **Vectors**

To create a vector just type `name=[V1; V2; … Vn]` where name can be any name you want for your vector and `V1, V2, …, Vn` are the values of your vector. You can also create vector by typing `name=[lower:increment:upper]` which creates a vector with values from the lower to upper limits. To access an element in a vector just type `name(index)` where index is the location of the element. To add elements or change a value from the vector just type `name(index)=new value`

Examples

If `V=[1;2;3;4]` then `V(2)` will return `2` or `V(2:4)` will return the elements `2` to `4` in this case `2 3 4`

Let `T=1:10` (if you don't declare any increment the increment will be of one). `T` will have the numbers from one to ten.
If `U=10:-1:1` then `U` will have the numbers from ten to `1`

- **Operations with Vectors**

For any operation you want to perform between vectors the vectors must have the same dimensions, in other words, the same number of rows and columns.
To add or subtract two vectors just type `vector1 + or – vector2.`
Example
If `V1=[1;2;3;4]  V2=[2;4;5;7]  V3=V1+V2  V4=V1-V2` then `V3=[3;6;8;11]` and `V4=[-1;-2;-2;-3]`

The symbol ' works as the transpose operator. Then since, `V1` is a column vector, `V1'` will be a row vector. To multiply vectors remember that their dimensions must agree.

Example

If `V1=[1;2]  V2=[2;4]  V5=V1*V2'` *then* `V5=[ 2  4`
`4  8]`
and `V1'*V2 = 10`

- **Matrices**

You can create matrices in different ways. You can put a colon or a space in between elements and you can put a semicolon or hit the return button to indicate a new row. You always have to end and start a matrix with a bracket.

Example
A=[1 2 3 or A=[1,2,3;4,5,6] or A [1 2 3; 4 5 6] or A=[1,2,3
4 5 6] 4,5,6]
To access an element in a matrix just type the name of the matrix and the index of the element.
Example
`A(1,1)=1, A(2,3)=6`
To access an entire row of a matrix just type *name(row number, :)*
Example
`B=A(1, : )  B = [1 2 3]` will have row one
To access an entire column just type `name( : , column number)`

Example
`C=A( : ,3)` then `C = [3 6]` will have the third column
To obtain a part of the matrix just type `name(row indexes, column indexes)`
Example
```
D=A(1:2,2:3) D = [ 2 3
                   5 6 ]
```

- **Operations with Matrices**

For any operation you want to perform between matrices the matrices must have the appropriate dimensions (as defined in matrix algebra). To add or subtract two matrices just type `matrix1 + or – matrix2.`
Example
If `m1=[1 2; 3 4]  m2=[2 4; 5 7]  m3=m1+m2  m4=m1-m2`
Then `m3=[ 3  6  and  m4=[ -1 -2`
`8 11]  -2 -3]`
To multiply two matrices just type *`matrix1 * matrix2`*.
Example
If `m1=[1 2; 3 4]  m2=[2 4; 5 7]  m5=m1*m2`
Then `m5=[ 12  18`
`26  40 ]`

- **If Statement**

The **if** statement checks if the conditional statement is true or false, if true it will execute the commands if false the statement will not be executed and the program will go to the **elseif** clause of to the **else** clause if neither of this two are present it will go to the end. The format for an *if statement* is:

```
if condition        (condition must include a relational operator)
  Statements
end                 (every if statement must have an end)
```

Example

```
n=3                          n=3
if n<6                       if n<6
  x = n^2;        OR                 x = n^2;
  n = 5;                             n = 5;
end                          elseif n > 10
                                     x = 1;
                                     n = 0;
                             else
                                     x = 0;
                             end
```

- **For Loop**

The **for loop** repeats the group of statements a predetermined fixed number of times. The format for a **for loop** is:

```
for i=limit1:inc:limit2   (where increment (inc) can be
                          positive ornegative. If inc is not defined, the
                          default increment is one)
     Statements           (It will repeat executing
end                       the statements until limit1 reaches limit2)
```

If limit1 is equal to limit2 the statement will still be executed once.

Example

```
for i=1:5                        for i=5:-.5:n
  x=2^i;            OR             x=2^i;
end                              end
```

- **While Loop**

The while loop executes the statements while the condition is true. If the condition is false the statements will not be executed. When doing a while loop always make sure that inside the loop there is a statement that will eventually make the condition false, else you will have a runaway computation. The format for the while loop is:

```
while condition
     Statements
End
```

Example

```
n = 5;
while n <= 15
 x = 2*n;
 n = n+1;        % this ensure that n will eventually be greater than 15
end              %  making the condition false
```

(anything you write right after a % sign will be considered a comment)

- **M-Files**

An M-file is a file where you can put a sequence of statements and save them on a disk. They are called M-files because they must have the file type **".m"** as the last part of their filename. M-files are useful when you need to execute a series of statements at the same time and when you need to edit multiple commands. Inside an m-file you can have if statements, loops and graphs among other things.

```
% Example Simple Newton Method to find x such that
f(x)=0

x = 3;                          % initial point
f = (x^2)-1;                    % Original function
df= 2*x;                        % First derivative

iter = 0;
while iter<100          % maximum number of iterations

    if abs(f)<1e-6,
       break
    end                 % you got the solution

    deltax = -f/df;     % solving the Newton step
    x = x + deltax;     % update

    iter = iter+1;      % updating iterations
    f = x^2-1;          % Evaluate the function at the
                        % current step
    df = 2*x;           % Evaluate the first derivative

end

x                       % To get the last value of x
```

This program will be saved as `newton1.m`, to run the program just type `newton1` at the command window. Make sure the directory in the command window is the same as the directory where you saved your program.

- **Storing Data**

When doing a program it may be necessary to store the value of different variables at iterations.
To do this you can use `fid = fopen( 'filename' ,' Permission' )` opens the file *filename* in the mode specified by *permission*. Permission can be:

'r' read	'r+' read and write (do not create)
'w' write (create if necessary)	'w+' create for read and write
'a' append (create if necessary)	'a+' read and append

This will create and open the file where you will be storing the data. To store the data you will use `fprintf(fid, ' format' ,variables)` where `fid` has been initialized before to be the file you will be using, variables will be the name of the variables you want to save and format can be:

%c	Single character	%f	Fixed-point notation
%i	integer notation	%g	more compact of %e or %f.
%e	Exponential notation	%	String of characters

You can also specify the spacing you want with:

\b	Backspace	\n	New line	\t	Horizontal tab

```
%Example Newton Method with data storing

x = 3;                          % Initial point
f = (x^2)-1;                    % Original function
df= 2*x;                        % First derivative
iter = 0;
fid= fopen('results.txt', 'w');
% fid is the name of the file
fprintf(fid, 'iter\t x\t\t f(x)\n');
% title of the columns in your table
while iter<100          % maximum number of iterations
    if abs(f)<1e-6, break ,end % you got the solution
    deltax = - f/df;            % solving the Newton step
    x         = x + deltax;     % update
    iter      = iter+1;         % updating iterations
    f          = x^2-1;
% Evaluate function at current x
    df=2*x;       % Evaluate first derivative at x
    fprintf(fid, '%i\t %f\t %g\n', iter, x, f);
end
fclose(fid);
```

You can save this program as **`newton2.m`**, to run the program just type *newton2* in the command window. After you run the program your **`results.txt`** file will look something like this:

```
iter    x           f(x)
1       1.666667    1.77778
2       1.133333    0.284444
3       1.007843    0.0157478
4       1.000031    6.1037e-005
5       1.000000    9.31323e-010
```

- **Functions**

A function is a type of M-file and has the format:

```
function [output1,…,outputn] = filename(input1,…, inputn)
```

Functions are useful when you want to make a program more general by being able to change some parameters when you execute the program instead of having to change the context of the program. Also a function can call another function. The name of the function has to be the same as the name of the file.

```
% Example general Newton method

function [iter, sol] = newton3(x)
% input   :
% x        the initial guess
% output  :
% iter :  the number of iterations it took to converge
% sol :  the approximate solution
Iter = 0;
[f,df] = func(x);
fid =  fopen('results.txt', 'w');
fprintf(fid, 'iter\t x\t\t f(x)\n');
%the title of the columns in your table
while iter<100          %maximum number of iterations
    if abs(f)<1e-6, break ,end
                        %you got the solution
    deltax = -f/df;             %solving the Newton step
    x = x+deltax;               %update
    iter = iter+1;              %updating iterations
    [f,df] = func(x);
    %Evaluate function and 1st derivative at x
    fprintf(fid, '%i\t %f\t %g\n', iter, x, f);
end
sol = x;
fclose(fid);
```

This Newton method is getting the function and derivative from an outside function called `func`. The function `func` will be like this:

```
function [f,df] = func(x)
   f  = x.^2 - 1;     % Original function
   df = 2*x ;         % First derivative of the function
end
```

This function needs to be saved under the name **`func.m`** in the same directory as your main program. To run your main program just type

```
[iter, sol] = newton3(x)
```

where `x` can be any value you want to use as the initial guess. You will get the same table of results as in the previous example.

- **Plotting**

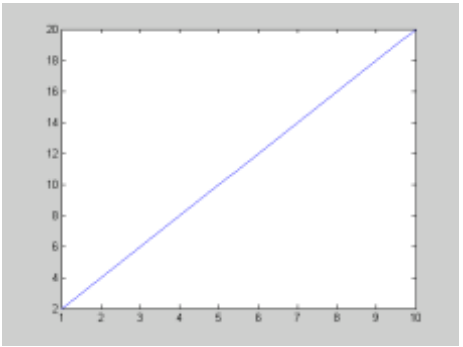
To draw a two dimensional graph just type **`plot(X,Y,S)`** where X is the independent variable, Y is the dependent variable and S is the format of the graph. S can be any of the following characters or a combination of the different columns.

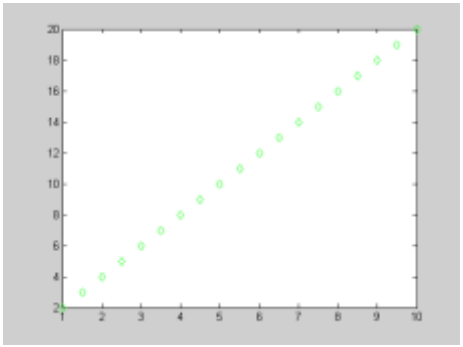
y	yellow	*	star	-	solid
k	black	o	circle	:	dotted
b	blue	x	x-mark	-.	dash dot
r	red	+	plus	--	dashed
g	green	d	diamond		

To see more plotting options type *help plot*

Example:

If `x = [1:.5:10]` and `y=2*x` then
`plot(x,y)` plots a solid blue line. (Blue is the default color for plotting)
`plot(x,y,'gd')` plots green diamonds at each data point but does not connect them.





- **Clearing a Plot**

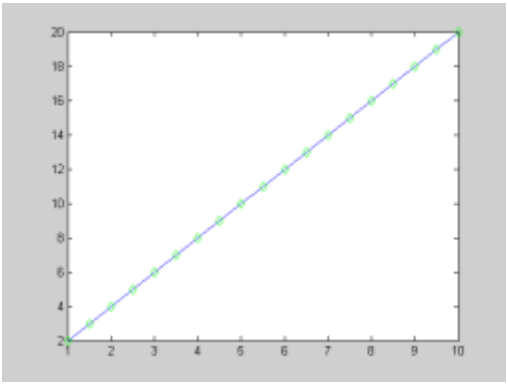
When you use the command plot a figure is created. To clear this figure just type `clf` and the current figure will be cleared.

- **Multiple Plots**

There are three different options for multiple plots. The first option is to have all the plots in one figure. This is a good idea when the two graphs are related and you want to see for example if they intersect. To use this option, type hold on after the first plot command then type the next plot command. Also you can type grid on to add grid lines.

Example:

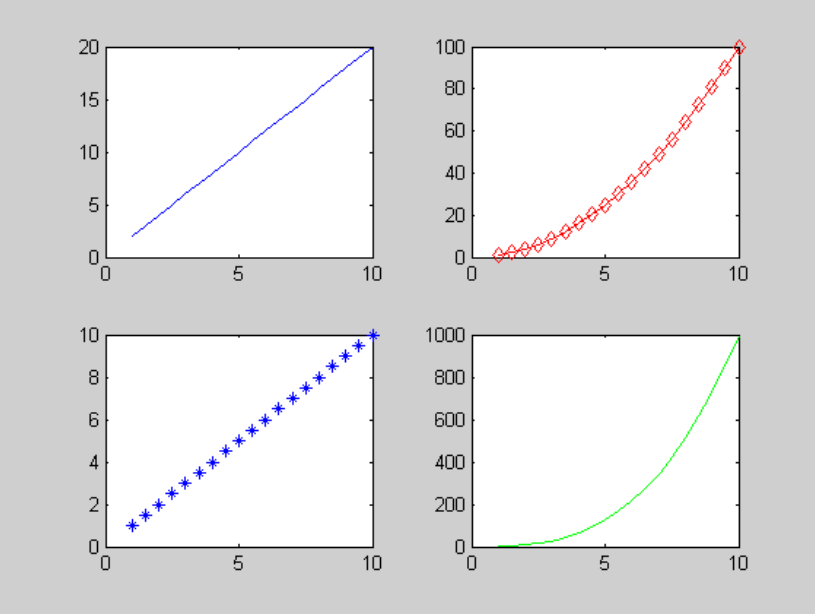
```
plot(x,y)
hold on, grid on
plot(x,y,'gd')
```



The second option is to divide the figure in to several subfigures, in other words to have several small graphs in the same paper. To use this option just type **`subplot(m,n,p)`** this will divide the figure into a `mxn` matrix and p will be the current plot.

Example:

```
subplot(2,2,1); plot(x,y)
subplot(2,2,2); plot(x,x.^2,'rd-')
subplot(2,2,3); plot(x,x,'b*')
subplot(2,2,4); plot(x,x.^3, 'g')
```



The last option is to have different graphs in different figures, in other words you will have each graph in a different paper. To do this just type `figure(n)` where `n` will be the number of the figure you are using currently.

Example:

```
figure(1)

plot(x,y)

figure(2)

plot(x,y,'g+')
```

- **Axis Labels, Titles and Legend**

To add axis labels just type `xlabel('label')` for the x-axis and `ylabel('label')` for the y-axis. To add a title to your graph just type `title('title')`. A legend can be added when you are plotting several graphs on the same plot. Just type `legend('legend1', 'legend2', …, 'legendn')`.

Example:

```
x =[1:.5:5];

plot(x,x.^2)

hold on

plot(x,x.^2,'gd')

legend('graph', 'data points')

xlabel('independent variable')
ylabel('dependent variable')

title('Example of a plot')
```