

Farida Suharleni, M.Si

# **Modul Praktikum**

# **Pengantar Ilmu Komputer**



## KATA PENGANTAR

*Alhamdulillah*, segala puji bagi Allah *'Azza wa Jalla* yang telah memberikan hidayah dan *inayah*-Nya sehingga penulis dapat menyelesaikan buku petunjuk praktikum “PENGANTAR ILMU KOMPUTER”. Modul ini disusun untuk membantu mahasiswa mengimplementasikan algoritma yang ditemui dalam beberapa mata kuliah Pengantar Ilmu komputer I, Program Komputer I, Analisis Numerik, Persamaan Diferensial Biasa, Persamaan Diferensial Parsial, Statistika, dan mata kuliah lain ke dalam suatu bahasa program.

Modul ini memuat tentang proses pengimplementasian algoritma melalui *software* Matlab. Dengan beberapa fungsi khusus yang sudah “*build in*” dalam Matlab Library, mahasiswa diharapkan dapat dengan mudah membuat program dalam bahasa nonprosedural yang bersifat singkat dan lugas namun dapat mengatasi masalah–masalah kompleks dalam Matematika.

Atas terselesaikannya penulisan modul ini, penulis juga menyampaikan terima kasih kepada yang terhormat:

1. Dekan Fakultas SAINTEK UIN-Maliki Malang;
2. Ketua Jurusan Matematika;
3. Pihak-pihak yang terlibat secara langsung maupun tidak langsung dalam penyusunan modul ini.

Demi perbaikan kualitas modul ini, kritik dan saran senantiasa penulis nantikan. Semoga modul ini memberikan manfaat bagi mahasiswa dan pembaca pada umumnya. *Amin ya Rabb*.

## DAFTAR ISI

Kata Pengantar .....	i
Daftar Isi .....	ii
Daftar Gambar .....	iii
Daftar Tabel .....	iii
MODUL 1 [Praktikum 1 – 4]	
Pengenalan Matlab, Vektor dan Matriks .....	1
Pendahuluan .....	1
Operasi Lembar Kerja .....	4
1. Memulai Matlab .....	4
2. Beberapa Fungsi Dalam Matlab .....	5
3. Matriks .....	7
3.1. Penjumlahan dan Pengurangan .....	19
3.2. Perkalian Matriks .....	20
3.3. Persamaan Linier dalam Matriks .....	22
3.4. Transposisi .....	23
3.5. Operasi Elemen-per-Elemen .....	25
3.6. Fungsi Elemen-per-Elemen .....	27
MODUL 2 [Praktikum 5 – 7]	
GRAFIK DAN SUARA .....	31
1. Plot 2-Dimensi .....	31
2. Lebih Jauh Mengenai Plot .....	34
3. Plot 3-Dimensi .....	40
3.1. Plot Garis .....	40
3.2. Plot Permukaan .....	42
3.3. Plot Kontur .....	44
4. Suara .....	46
MODUL 3 [Praktikum 8 – 11]	
M-FILE DAN PEMROGRAMAN MATLAB .....	47
1. Membuat M-File .....	47
2. M-File Sebagai Skrip Program .....	48
3. M-File Sebagai Fungsi .....	52
4. Display dan Input .....	55
5. Control Statement .....	56
5.1. Statement if ... elseif ... else ... end .....	56
5.2. Statement switch ... case .....	58
5.3. Statement for ... end .....	58
5.4. Statement while ... end .....	61
5.5. Statement break dan return .....	62
5.6. Statement continue .....	65
6. Operator Perbandingan dan Logika .....	66

## DAFTAR GAMBAR

Gambar 1 Menu dan simbol utama Matlab .....	2
Gambar 2 Matlab Help .....	3
Gambar 3 Lembar kerja untuk menjalankan fungsi-fungsi Matlab .....	4
Gambar 4 Jendela figure .....	31
Gambar 5 Contoh plot: kurva $Y = X^3$ .....	33
Gambar 6 Hasil plot dengan “hold on” .....	34
Gambar 7 Pembagian area plot dengan “subplot” .....	35
Gambar 8 Contoh plot semi-logaritmik .....	38
Gambar 9 Contoh penggunaan subplot .....	39
Gambar 10 Contoh plot dengan <i>command</i> “polar” .....	40
Gambar 11 Contoh plot 3-dimensi dengan <i>command</i> “plot3” .....	41
Gambar 12 Contoh penggunaan “plot3” .....	42
Gambar 13 Hasil plot dengan “mesh” dan “surf” .....	43
Gambar 14 Plot 3-dimensi dari fungsi $\sin(r) / r$ .....	44
Gambar 15 Contoh plot kontur .....	45
Gambar 16 Jendela editor M-file .....	48
Gambar 17 Memilih direktori untuk menjalankan M-file .....	49
Gambar 18 Contoh plot 4 kurva parabola dengan “for” .....	60
Gambar 19 Contoh plot 4 kurva dengan “while” .....	62

## DAFTAR TABEL

Tabel 1 .....	10
Tabel 2 .....	17
Tabel 3 .....	17
Tabel 4 .....	23
Tabel 5 .....	24
Tabel 6 .....	32
Tabel 7 .....	33
Tabel 8 .....	34
Tabel 9 .....	37
Tabel 10 .....	44
Tabel 11 .....	46
Tabel 12 .....	66
Tabel 13 .....	66
Tabel 14 .....	67

# MODUL 1 [Praktikum 1 – 4]

## PENGENALAN MATLAB, VEKTOR DAN MATRIKS

### Pendahuluan

MATLAB (MATrix LABoratory) adalah bahasa tingkat tinggi dan interaktif yang memungkinkan untuk melakukan komputasi secara intensif. MATLAB telah berkembang menjadi sebuah *environment* pemrograman yang canggih yang berisi fungsi-fungsi *build in* untuk melakukan pengolahan sinyal, aljabar linear dan kalkulasi matematis lainnya. MATLAB juga berisi *toolbox* yang berisi fungsi – fungsi tambahan untuk aplikasi khusus. Penggunaan MATLAB meliputi bidang :

- \* Matematika dan Komputasi
- \* Pembentukan Algorithm
- \* Akuisisi Data
- \* Pemodelan, simulasi dan Pembuatan Prototype
- \* Analisis Data, Explorasi, dan Visualisasi Grafik
- \* Keilmuan dan Bidang Rekayasa

Beberapa keunggulan Matlab terletak pada:

#### 2. Aspek Komputasional

- \* Analisa matrik dan manipulasinya
- \* Reduksi data dan pengolahan data statistik
- \* FFT, statistic korelasi dan kovarian
- \* Pendukung matrik “sparse”
- \* Fungsi trigonometri dan beberapa fungsi kompleks lainnya
- \* Fungsi Bessel, beta, dan fungsi kepadatan lainnya
- \* Persamaan diferensial linier dan nonlinier

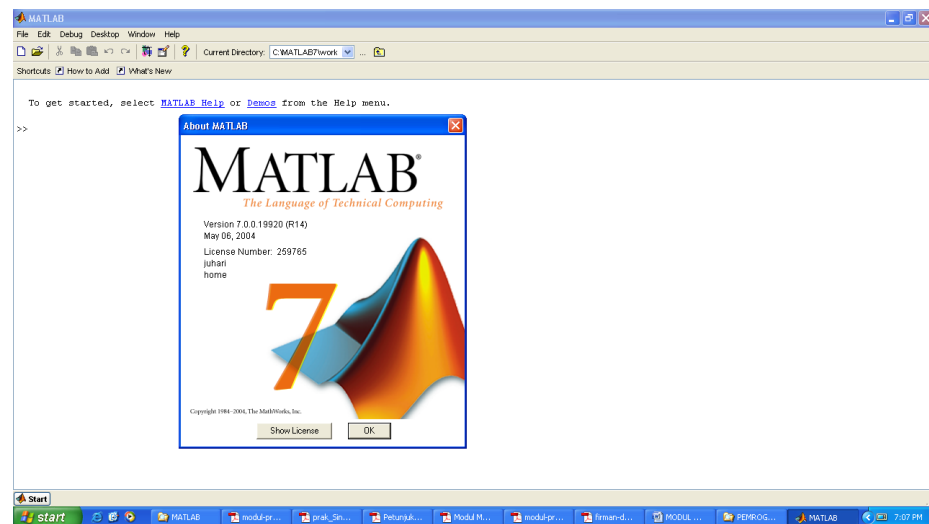
#### 3. Aspek Grafik dan Visualisasi

- \* 2-D scatter, grafik garis, poligon dan mesh, counter, grafik polar, dan plot histogram
- \* 3-D scatter, grafik garis, poligon, mesh dan plot “wireframe”

- ✳ Grafik dengan variasi permukaan disertai dengan animasi gambar dan suara

#### 4. Aspek Pemrograman

- ✳ Struktur control (FOR, WHILE dan IF)
- ✳ Manipulasi string
- ✳ Input file berupa ASCII dan biner
- ✳ Debugging
- ✳ Dapat berinteraksi dengan bahasa pemrograman C

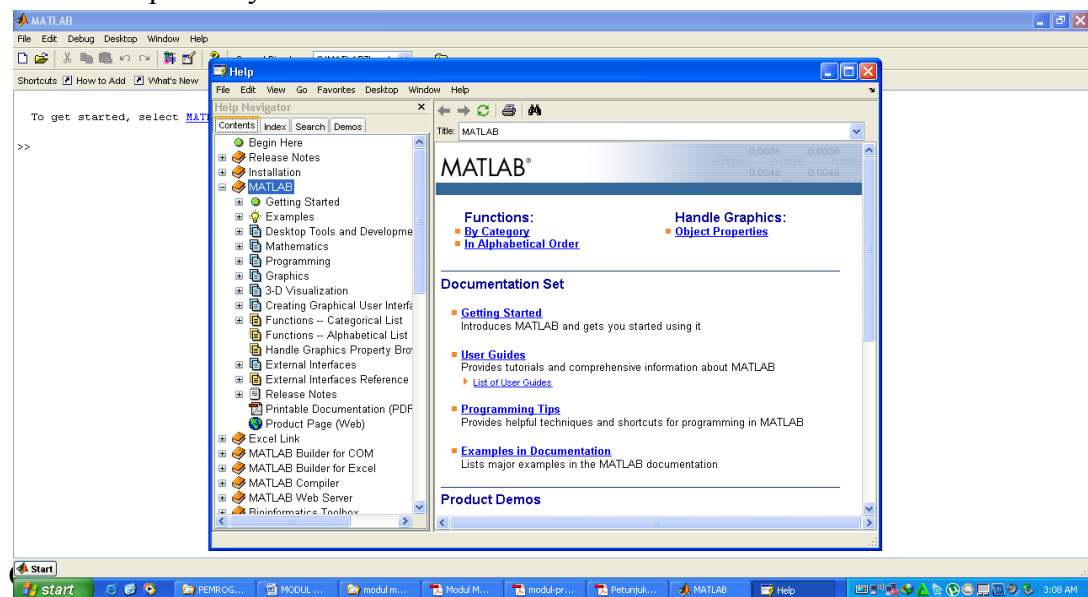


Disamping sebagai compiler, Matlab juga dilengkapi dengan Toolbox software yang merupakan satu-satu kecanggihan Matlab. Software ini mencakup berbagai masalah – masalah besar dalam teknologi tingkat tinggi, di antaranya adalah:

- ✳ **Control system Toolbox**, merupakan kumpulan fungsi-fungsi Matlab untuk pemodelan, analisis dan desain system kontrol otomatis.
- ✳ **Financial Toolbox**, merupakan software untuk menyelesaikan beberapa masalah keuangan dari masalah yang sederhana sampai masalah yang cukup kompleks.
- ✳ **Fuzzy Logic Toolbox**, merupakan software untuk mengembangkan desain “fuzzy” dari tahap “setup” sampai diagnose.

- \* **Signal Processing Toolbox**, merupakan tools untuk menyelesaikan masalah besar dalam analisa bispektral, model signal linier dan nonlinier, transformasi FFT dan DCT serta visualisasi spectrum.
- \* **Spectral Analysis Toolbox**, merupakan tools untuk menganalisa signal dengan menggunakan “cumulant” atau spectral dengan order tinggi.
- \* **Image Processing Toolbox**, merupakan software khusus dalam matlab untuk desain filter, analisa citra, manipulasi warna dan lain-lain yang berkenaan dengan visualisasi citra.
- \* **Statistics Toolbox**, merupakan software yang menangani masalah-masalah stokastik.
- \* **System Identification Toolbox**, merupakan software untuk melakukan aktifitas desain system dinamis yang berdasarkan pada input dan output data.

Selanjutnya silahkan pelajari sendiri tentang Matlab dengan menekan tombol F1 pada keyboard !

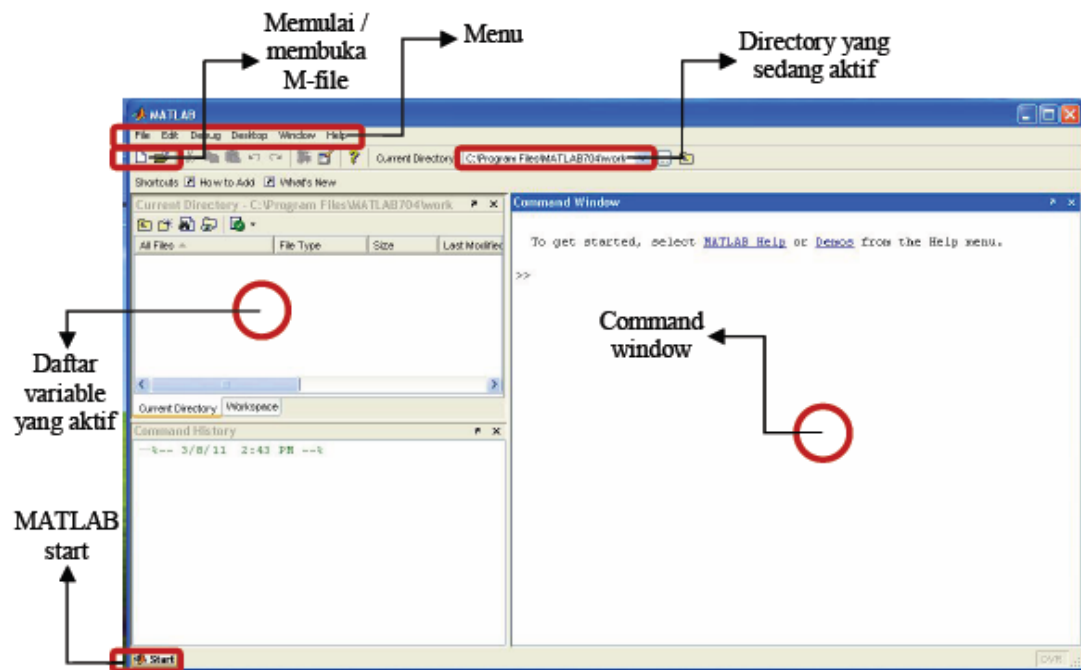


### Tujuan Khusus Praktikum :

Mahasiswa dapat memahami kegunaan software Matlab dan aplikasinya untuk vektor dan matriks

#### 1. Memulai Matlab

Setelah melakukan instalasi MATLAB pada PC, perhatikan icon MATLAB pada tampilan desktop kemudian “doubleclick” pada icon tersebut. Selanjutnya akan muncul tampilan seperti pada gambar berikut ini.



bagian penting di dalam MATLAB, antara lain :

a. Jendela perintah (Command Window)

Pada command window, semua perintah matlab dituliskan dan dieksekusi. Kita dapat menuliskan perintah perhitungan sederhana, memanggil fungsi, mencari informasi tentang sebuah fungsi dengan aturan penulisannya (help), demo program, dan sebagainya. Setiap penulisan perintah selalu diawali dengan prompt '>>'.

b. Jendela ruang kerja (Workspace)

Jendela ini berisi informasi penggunaan variabel di dalam memori MATLAB.

c. Jendela history (Command History)



Jendela ini berisi informasi tentang perintah yang pernah dituliskan sebelumnya. Kita dapat mengambil kembali perintah dengan menekan tombol panah ke atas atau mengklik perintah pada jendela histori, kemudian melakukan copypaste ke command window.

## 2. Beberapa Fungsi Dalam Matlab

Beberapa fungsi yang dapat dipakai dalam Matlab untuk menyelesaikan beberapa masalah dapat diringkas dalam penjelasan berikut:

### a. Fungsi pengatur umum

- >> *help* fungsi : untuk mengetahui petunjuk pemakaian suatu fungsi
- >> *type* file.m : untuk menampilkan isi dari M-File
- >> *pwd* : untuk mengetahui subdirektori aktif
- >> *cd* a\ data : memindahkan suatu direktori aktif ke direktori lain yaitu A dalam subdirectori data
- >> *dir* : untuk menampilkan isi direktori
- >> *!ren* file 1.txt file 1.m : merubah nama file1.txt menjadi file 1.m

### b. Fungsi pengatur variabel dan areal kerja

- >> *save* filename : untuk menyimpan variabel dalam file.mat
- >> *load* filename : untuk memanggil data yang disimpan dalam file.mat
- >> *clear* : untuk menghapus variabel terdefinisi
- >> *pack* : untuk memampatkan pemakaian memory lembar kerja
- >> *size(A)* : untuk mengetahui ordo matrik A
- >> *max(A)* atau *min(A)* : untuk mengetahui nilai terbesar dan terkecil dari elemen matrik A
- >> *length(A)* : menginformasikan bilangan terbesar dari ordo matrik A
- >> *clc*: membersihkan layar lembar kerja

c. Operator numerik dan matrik

>>  $\pm$  : penjumlahan dan pengurangan

>>  $*$ ,  $^$  : perkalian dan perpangkatan

>>  $/$ ,  $\backslash$  : pembagian kanan untuk bilangan dan pembagian kiri untuk matrik dan vektor

>>  $'$  : transpose vektor atau matrik

d. OPERATOR array

>>  $\pm$  : penjumlahan dan pengurangan

>>  $.*$ ,  $.^$  : perkalian dan perpangkatan

>>  $./$ ,  $.\backslash$  : pembagian kanan untuk bilangan dan pembagian kiri untuk matrik dan vektor

>>  $'$  : transpose vektor atau matrik

Penambahan titik dalam operator array disebabkan adanya operasi sederetan bilangan dalam waktu yang bersamaan. Contoh array  $x = 0:0.1:10$

e. Operator logika dan relasional

>>  $<$ ,  $\leq$  : lebih kecil dan lebih kecil sama dengan

>>  $>$ ,  $\geq$  : lebih besar dan lebih besar sama dengan

>>  $=$  : sama atau ekuivalen

>>  $\sim$  : tidak sama atau tidak ekuivalen

>>  $\&$ ,  $|$ ,  $\sim$  : dan, atau, tidak

f. Penulisan fungsi matematika

>>  $abs(x)$  : fungsi untuk menghasilkan nilai absolut dari x

>>  $sign(x)$  : fungsi untuk menghasilkan nilai -1 jika  $x < 0$ , 0 jika  $x = 0$  dan 1 jika  $x > 0$

>>  $exp(x)$  : untuk menghasilkan nilai eksponensial natural,  $e^x$

>>  $log(x)$  : untuk menghasilkan nilai logaritma natural x,  $\ln x$

>>  $log10(x)$  : untuk menghasilkan nilai logaritma dengan basis 10,  $x \log_{10}$

>>  $sqrt(x)$  : untuk menghasilkan akar dari nilai x,  $\sqrt{x}$

>>  $rem(x,y)$  : untuk menghasilkan nilai modulus (sisa pembagian) x terhadap y

g. Fungsi M-file

>> *disp* ('karakter') : menampilkan karakter (string)

>> *num2str* : mengkonversi numerik menjadi string

>> *input* : meminta user memberikan input

>> *pause* : menghentikan program sampai user menekan <ENTER>

>> *pause(n)* : berhenti selama n detik

### 3. Matriks

Terdapat 3 jenis format data di MATLAB yaitu skalar, vektor dan matriks.

- \* Skalar, ialah suatu bilangan tunggal
- \* Vektor, ialah sekelompok bilangan yang tersusun 1-dimensi. Dalam MATLAB biasanya disajikan sebagai vektor-baris atau vektor-kolom
- \* Matriks, ialah sekelompok bilangan yang tersusun dalam segi-empat 2-dimensi. Di dalam MATLAB, matriks didefinisikan dengan jumlah baris dan kolomnya. Di MATLAB terdapat pula matriks berdimensi 3, 4, atau lebih, namun dalam buku ini kita batasi hingga 2-dimensi saja.

Sebenarnya, semua data bisa dinyatakan sebagai matriks. Skalar bisa dianggap sebagai matriks satu baris – satu kolom (matriks  $1 \times 1$ ), dan vektor bisa dianggap sebagai matriks 1-dimensi: satu baris – n kolom, atau n baris – 1 kolom (matriks  $1 \times n$  atau  $n \times 1$ ). Semua perhitungan di MATLAB dilakukan dengan matriks, sehingga disebut MATrix LABoratory.

Matriks didefinisikan dengan kurung siku ([ ]) dan biasanya dituliskan baris-per-baris. Tanda koma (,) digunakan untuk memisahkan kolom, dan titik-koma (;) untuk memisahkan baris. Kita juga bisa menggunakan spasi untuk memisahkan kolom dan menekan Enter ke baris baru untuk memisahkan baris.

Contoh vektor-baris dan vektor-kolom:

```
>> vektor1=[3,5,7]
```

```
vektor1 =
```

```
3 5 7
```

```
>> vektor2=[2;4;6]
vektor2 =
2
4
6
```

Berikutnya kita coba contoh berikut untuk mendefinisikan matriks  $3 \times 3$ .

```
>> matriks1=[10 20 30
40 50 60
70 80 90]
>> matriks2=[10 20 30; 40 50 60; 70 80 90]
```

Terlihat bahwa **matrix1** dan **matrix2** isinya sama, karenanya kita bisa menekan Enter untuk membuat baris baru, ataupun menggunakan titik-koma.

Kita juga bisa mendefinisikan matriks elemen per elemen.

```
>> mat(1,1)=100; mat(1,2)=200; mat(2,1)=300;
>> mat(2,2)=400
mat =
100 200
300 400
```

Kita sekarang akan mencoba menggabungkan variabel yang ada untuk membentuk matriks baru.

```
>> gabung1=[vektor2 matriks1]
gabung1 =
2 10 20 30
4 40 50 60
6 70 80 90
>> gabung2=[vektor1; matriks2]
```

```
gabung2 =
3 5 7
10 20 30
40 50 60
70 80 90
```

Kita harus ingat bahwa matriks gabungan harus memiliki jumlah baris dan kolom yang valid sehingga membentuk persegi panjang.

Untuk mengetahui ukuran atau dimensi dari matriks yang ada, kita bisa gunakan command **size** dan **length**. **size** umumnya digunakan untuk matriks 2-dimensi, sementara **length** untuk vektor.

```
>> length(vektor1)
```

```
ans =
```

```
3
```

```
>> size(matrix1)
```

```
ans =
```

```
3 3
```

Menunjukkan panjang **vektor1** ialah 3 elemen, dan ukuran **matrix1** ialah 3-baris 3-kolom (3×3). Kita juga bisa menyimpan keluaran command dalam variabel baru.

```
>> panjang=length(vektor2)
```

```
panjang =
```

```
3
```

```
>> [jml_baris,jml_kolom]=size(gabung5)
```

```
jml_baris =
```

```
3
```

```
jml_kolom =
```

```
6
```

Sementara itu, untuk menghitung jumlah elemen dari suatu matriks, kita menggunakan command **prod**. Misalkan untuk matriks **gabung5**, jumlah elemennya ialah;

```
>> jml_elemen=prod(size(gabung5))
jml_elemen =
    18
```

Berbagai matriks khusus yang kerap kita gunakan dalam perhitungan bisa dibuat secara efisien dengan command yang telah ada di MATLAB.

**Tabel 1**

<b>ones(n)</b>	membuat matriks satuan (semua elemennya berisi angka 1) berukuran $n \times n$ .
<b>ones(m,n)</b>	membuat matriks satuan berukuran $m \times n$ .
<b>zeros(n)</b>	membuat matriks nol (semua elemennya berisi angka 0) berukuran $n \times n$ .
<b>zeros(m,n)</b>	membuat matriks nol berukuran $m \times n$ .
<b>eye(n)</b>	membuat matriks identitas berukuran $n \times n$ (semua elemen diagonal bernilai 1, sementara lainnya bernilai 0).
<b>rand(n), rand(m,n)</b>	membuat matriks $n \times n$ , atau $m \times n$ , berisi bilangan random terdistribusi uniform pada selang 0 s.d. 1.
<b>randn(n), randn(m,n)</b>	membuat matriks $n \times n$ , atau $m \times n$ , berisi bilangan random terdistribusi normal dengan mean = 0 dan varians = 1. Command ini kerap kita gunakan untuk membangkitkan derau putih gaussian.

`[]` matriks kosong, atau dengan kata lain matriks  $0 \times 0$ ; biasa digunakan untuk mendefinisikan variabel yang belum diketahui ukurannya.

Untuk memperdalam pemahaman, mari kita lihat contoh di bawah ini.

```
>> clear
```

```
>> mat_1=5*ones(2,4)
```

```
mat_1 =
```

```
     5     5     5     5
     5     5     5     5
```

```
>> mat_2=zeros(2,4)
```

```
mat_2 =
```

```
     0     0     0     0
     0     0     0     0
```

```
>> mat_3=[eye(4) -ones(4)]
```

```
mat_3 =
```

```
 1  0  0  0  -1  -1  -1  -1
 0  1  0  0  -1  -1  -1  -1
 0  0  1  0  -1  -1  -1  -1
 0  0  0  1  -1  -1  -1  -1
```

```
>> bil_acak_uniform=rand(1,10)
```

```
bil_acak_uniform =
```

```
Columns 1 through 7
```

```
0.9501 0.2311 0.6068 0.4860 0.8913 0.7621 0.4565
```

```
Columns 8 through 10
```


```
0.0185 0.8214 0.4447
```

```
>> gaussian_noise=randn(5,1)
gaussian_noise =
    -0.4326
   -1.6656
    0.1253
    0.2877
   -1.1465
```

Misalkan kita ingin membangkitkan 20 buah bilangan acak gaussian dengan mean = 5 dan varians = 3.

```
>> mu=5; %Nilai mean
>> varians=3; %Nilai variansi

>> bil_acak_gaussian= sqrt(varians)*randn(1,20) + mu
bil_acak_gaussian =
```

 Setiap kali kita menggunakan *command* **rand** dan **randn**, kita akan selalu **Tips** mendapatkan nilai keluaran yang berbeda. Hal ini merupakan salah satu sifat bilangan acak.

Dalam vektor ataupun matriks, indeks digunakan untuk menunjuk satu/beberapa elemen dari vektor/matriks. Indeks dituliskan di dalam tanda kurung ( ) dengan pola umum sebagai berikut.

Untuk vektor:

**nama\_vektor( indeks )**

Untuk matriks:

**nama\_matriks( indeks\_baris , indeks\_kolom )**

Dalam suatu vektor, elemen pertama diberi indeks = 1, sementara dalam matriks, indeks menunjukkan nomor baris dan nomor kolom dari elemen yang ingin ditunjuk. Untuk lebih jelasnya perhatikan contoh berikut ini.



```
>> clear
>> vektor_ini = [1 3 5 7 9];
>> vektor_itu = [9; 8; 7; 6; 5];
>> matrix = [10 20 30; 40 50 60; 70 80 90];
>> vektor_ini(1)
ans =
     1
>> vektor_itu(2)
ans =
     8
>> matrix(1,2)
ans =
    20
>> [matrix(1,1) matrix(1,2) matrix(1,3)]
ans =
    10    20    30
```

Kita juga bisa mengambil beberapa baris dan kolom sekaligus dari suatu matriks dengan operator titik-dua (:). Dalam hal ini tanda titik-dua berarti **“sampai dengan”**.

Misalkan untuk mengambil elemen ke-1 sampai ke-3 dari **vektor\_ini**

```
>> vektor_ini(1:3)
ans =
     1     3     5
```

Mengambil elemen ke-3 sampai ke-5 dari **vektor\_itu**

```
>> vektor_itu(3:5)
ans =
     7
     6
```

5

Mengambil elemen baris ke-1 sampai ke-2, kolom ke-2 sampai ke-3 dari **matrix**

```
>> matrix(1:2, 2:3)
```

```
ans =
```

```
    20    30
    50    60
```

Dalam hal lain tanda titik-dua bisa berarti “seluruhnya”.

Misalkan untuk mengambil seluruh elemen dari **vektor\_ini**

```
>> vektor_ini(:)
```

```
ans =
```

```
    1    3    5    7    9
```

Mengambil seluruh baris dan kolom dari **matrix**

```
>> matrix(:, :)
```

```
ans =
```

```
    10    20    30
    40    50    60
    70    80    90
```

Mengambil seluruh elemen di baris ke-1 dari **matrix**

```
>> matrix(1, :)
```

```
ans =
```

```
    10    20    30
```

Mengambil seluruh elemen di kolom ke-2 dari **matrix**

```
>> matrix(:, 2)
```

```
ans =
```

```
    20
    50
```

80

Mengambil seluruh elemen di kolom ke-2 dan ke-3 dari **matrix**

```
>> matrix(:,2:3)
```

```
ans =
```

```
    20    30
```

```
    50    60
```

```
    80    90
```

Dengan menggunakan indeks, kita bisa mengubah nilai elemen matriks yang telah ada.

```
>> vektor_ini(1)=1000
```

```
vektor_ini =
```

```
    1000     3     5     7     9
```

```
>> vektor_itu(2:4)=[-1; -1; -1]
```

```
vektor_itu =
```

```
     9
```

```
    -1
```

```
    -1
```

```
    -1
```

```
     5
```

```
>> matrix(3,:)=100*ones(1,3)
```

```
matrix =
```

```
    10    20    30
```

```
    40    50    60
```

```
   100   100   100
```

Deret bilangan merupakan hal yang kerap kita temui dalam pengolahan data, terutama berkaitan dengan plot data dan proses iterasi (perhitungan berulang-ulang). Misalkan kita memiliki data tegangan suatu baterai pada setiap

menit selama 1 jam. Dalam menyajikan data “waktu”, kita harus membuat vektor berisi deret. Kita tentunya bisa melakukannya secara manual seperti ini:

```
>> time=[1, 2, 3, 4, ..., 60]
```

Tetapi akan lebih efisien jika deret diciptakan menggunakan operator titik-dua. Formulasnya ialah:

**deret = nilai\_awal : inkremen : nilai\_akhir**

Inkremen harus bilangan bulat positif atau negatif

Khusus untuk inkremen = 1:

**deret = nilai\_awal : nilai\_akhir**

Sehingga kita bisa tuliskan

```
>> time=1:60
```

Sekarang kita akan berlatih menggunakan operator titik-dua untuk membuat deret berikut:

**x** = 0, 100, 200, 300, 400, ... , 2200, 2300

**y** = -10, -9.5, -9, -8.5, ... -0.5, 0, 0.5, ... , 9, 9.5, 10

**z** = 10, 9.95, 9.9, 9.85, 9.8, 9.75, ... , 1, 0.95, 0.9, ... , 0.05, 0

```
>> x=0:100:2300;
```

```
>> y=-10:0.5:10;
```

```
>> z=10:-0.05:0;
```



Bedakan operator titik-dua untuk manipulasi indeks matriks dengan **Penting!** operator titik-dua untuk membuat deret. Untuk membedakannya ingatlah selalu bahwa indeks selalu berada di dalam tanda kurung ( )

Di dalam MATLAB, pembuatan deret juga bisa dilakukan dengan *command* berikut ini.

**Tabel 2**

<b>linspace(a,b,n)</b>	membuat vektor baris berisi <b>n</b> titik yang terpisah merata secara linier antara <b>a</b> dan <b>b</b> .
<b>logspace(a,b,n)</b>	membuat vektor baris berisi <b>n</b> titik yang terpisah merata secara logaritmik antara <b>10<sup>a</sup></b> dan <b>10<sup>b</sup></b> . <i>Command</i> ini biasa digunakan untuk menghitung respon frekuensi suatu sistem.

Contoh:

```
>> linspace(0,10,11)
```

```
ans =
```

```
0 1 2 3 4 5 6 7 8 9 10
```

```
>> logspace(0,2,10)
```

```
ans =
```

```
Columns 1 through 7
```

```
1.0000 1.6681 2.7826 4.6416 7.7426 12.9155 21.5443
```

```
Columns 8 through 10
```

```
35.9381 59.9484 100.0000
```

Terdapat beberapa *command* yang bisa digunakan untuk menukar, merotasi, dan menyusun kembali elemen matriks.

**Tabel 3**

<b>fliplr(A)</b>	menukar posisi elemen matriks <b>A</b> secara melintang, yaitu sebelah kiri ditukar dengan sebelah kanan.
<b>flipud(A)</b>	menukar posisi elemen matriks <b>A</b> secara membujur, yaitu

	sebelah atas ditukar dengan sebelah bawah.
<b>rot90(A)</b>	merotasi posisi elemen matriks <b>A</b> berlawanan arah jarum jam sejauh $90^\circ$ .
<b>reshape(A,m,n)</b>	menyusun ulang elemen matriks <b>A</b> menjadi berukuran <b>m</b> $\times$ <b>n</b> . Harus diingat bahwa jumlah elemen <b>A</b> harus sama dengan <b>m</b> $\times$ <b>n</b>

### Contoh

```
>> A=[0:3; 4:7]
```

```
A =
```

```
    0    1    2    3
    4    5    6    7
```

```
>> fliplr(A)
```

```
ans =
```

```
    3    2    1    0
    7    6    5    4
```

```
>> flipud(A)
```

```
ans =
```

```
    4    5    6    7
    0    1    2    3
```

```
>> rot90(A)
```

```
ans =
```

```
    3    7
    2    6
    1    5
    0    4
```

```
>> reshape(A,1,8)
ans =
      0  4  1  5  2  6  3  7
>> reshape(A,4,2)
ans =
      0  2
      4  6
      1  3
      5  7
```

### 3.1. Penjumlahan dan Pengurangan

Penjumlahan dua matriks,  $\mathbf{A+B}$ , dan selisih dua matriks,  $\mathbf{A-B}$ , terdefinisi jika  $\mathbf{A}$  dan  $\mathbf{B}$  berukuran sama. Namun demikian, penjumlahan/pengurangan juga bisa dilakukan antara matriks dengan skalar. Untuk jelasnya mari kita praktekkan contoh berikut ini.

```
>> A=[0 1;2 3];
>> B=[4 5;6 7];

>> Jumlah=A+B, Selisih=A-B, Tambah50=A+50
Jumlah =
      4  6
      8 10

Selisih =
     -4 -4
     -4 -4

Tambah50 =
     50 51
     52 53
```

### 3.2. Perkalian Matriks

Perkalian matriks, misalkan  $C = AB$ , terdefinisi jika jumlah kolom di  $A$  sama dengan jumlah baris di  $B$ . Selain itu, perkalian juga bisa dilakukan antara matriks dengan skalar.

Kita akan lanjutkan contoh sebelumnya.

```
>> A, B
```

```
A =
```

```
0 1
2 3
```

```
B =
```

```
4 5
6 7
```

```
>> MultAB=A*B, MultBA=B*A
```

```
MultAB =
```

```
6 7
26 31
```

```
MultBA =
```

```
10 19
14 27
```



**Tips** Ketika mengalikan dua matriks, maka matriks hasil perkalian dihitung berdasarkan formula baku. Misalkan  $C=AB$ ;  $A$  dan  $B$  matriks  $2 \times 2$ , sehingga hasilnya  $C$  juga  $2 \times 2$ .

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$



di mana:	$c_{11} = a_{11}b_{11} + a_{12}b_{21}$
	$c_{12} = a_{11}b_{12} + a_{12}b_{22}$
	$c_{21} = a_{21}b_{11} + a_{22}b_{21}$
	$c_{22} = a_{21}b_{12} + a_{22}b_{22}$

Contoh berikutnya ialah perkalian dua vektor, yang juga mengikuti aturan perkalian matriks, karena vektor sesungguhnya sama dengan matriks 1-dimensi.

```
>> x=[3 2 1], y=[100;10;1]
```

```
x =
```

```
    3    2    1
```

```
y =
```

```
   100
```

```
    10
```

```
     1
```

```
>> z1=x*y, z2=y*x
```

```
z1 =
```

```
   321
```

```
z2 =
```

```
   300  200  100
```

```
    30   20   10
```

```
     3    2    1
```

Selain perkalian di atas, dikenal pula perkalian vektor, yaitu: “dotproduct” (atau disebut juga *inner-product*), dan “cross-product”.

<b>dot(x,y)</b>	menghitung <i>dot-product</i> dari vektor <b>x</b> dan <b>y</b>
<b>cross(x,y)</b>	menghitung <i>cross-product</i> dari vektor <b>x</b> dan <b>y</b>

**Tips**

*Dot-product* dan *cross-product* dihitung berdasarkan formula baku.

Misalkan terdapat dua vektor  $\mathbf{x} = (x_1 \ x_2 \ x_3)$  dan

$\mathbf{y} = (y_1 \ y_2 \ y_3)$ , maka:

$$\text{dot-product: } \mathbf{x} \cdot \mathbf{y} = x_1y_1 + x_2y_2 + x_3y_3$$

$$\text{cross-product: } \mathbf{x} \times \mathbf{y} = (x_2y_3 - x_3y_2 \ x_3y_1 - x_1y_3 \ x_1y_2 - x_2y_1)$$

Perlu diingat bahwa hasil *dot-product* berupa skalar, sementara hasil *cross-product* berupa vektor.

### 3.7. Persamaan Linier dalam Matriks

Kita sering menemui persamaan linier dengan beberapa variabel. Di dalam aljabar, solusi persamaan tersebut bisa ditemukan, salah satunya dengan menggunakan matriks. Misalkan kita tinjau sistem persamaan linier dengan variabel  $x_1$  dan  $x_2$ .

$$x_1 - 2x_2 = 32$$

$$12x_1 + 5x_2 = 7$$

Dalam bentuk matriks bisa kita tuliskan:

$$\begin{pmatrix} 1 & -2 \\ 12 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 32 \\ 7 \end{pmatrix} \Leftrightarrow \mathbf{AX} = \mathbf{B}$$

Dalam MATLAB kita tuliskan:

```
>> A=[1 -2;12 5]; B=[32;7];
```

```
>> X=inv(A)*B
```

```
X =
```

```
6.0000
```

-13.0000

Sehingga kita dapatkan solusi  $x_1 = 6$  dan  $x_2 = -13$ . Atau kita juga bisa mendapatkan solusi tersebut dengan operator pembagian terbalik:

```
>> X=A\B
```

```
X =
```

```
6.0000
```

```
-13.0000
```

Sebagai bahan latihan, cobalah Anda pecahkan persamaan linier dengan tiga variabel berikut ini.

$$x + 2y + 3z = 2$$

$$4x + 5y + 6z = -5,5$$

$$7x + 8y - 9z = -49$$

### 3.8. Transposisi

Salah satu operasi yang penting dalam matriks ialah transposisi, dituliskan dalam MATLAB dengan operator petik tunggal ( ' ) dan titik-petik ( .' ). Operasi ini mempertukarkan baris dan kolom dari suatu matriks atau vektor.

**Tabel 4**

<b>petik tunggal ( ' )</b>	operasi transposisi untuk matriks berisi bilangan riil, atau transposisi dan konjugasi untuk matriks kompleks.
<b>titik-petik ( .' )</b>	operasi transposisi tanpa konjugasi. Untuk matriks riil, operator ini memberi hasil yang sama dengan petik tunggal

Mari kita praktekkan contoh berikut ini untuk memahami kedua operator di atas.

```
>> Mat_riil=[1 0; 3 5], Mat_kompleks=[1+2i 3i; 1 2+3i]
```

```
Mat_riil =
```

```
1    0
```

```
3    5
```

```
Mat_kompleks =
```

```
1.0000 + 2.0000i
```

```
0 + 3.0000i
```

```

1.0000          2.0000 + 3.0000i

>> Transp_riil=Mat_riil',Transp_kompleks=Mat_kompleks'
Transp_riil =
     1     3
     0     5
Transp_kompleks =
 1.0000 - 2.0000i   1.0000
      0 - 3.0000i   2.0000 - 3.0000i

>> Transp_riil2=Mat_riil.'
Transp_riil2 =
     1     3
     0     5

>> Transp_kompleks2=Mat_kompleks.'
Transp_kompleks2 =
 1.0000 + 2.0000i   1.0000
      0 + 3.0000i   2.0000 + 3.0000i

```

### 3.9. Operasi Elemen-per-Elemen

Di dalam MATLAB, operasi matematik juga bisa dilakukan elemen-per-elemen. Dalam hal ini matriks atau vektor yang terlibat harus berukuran sama. Operasi yang bisa dilakukan ialah perkalian/pembagian, penjumlahan/pengurangan, serta pangkat. Operator yang digunakan diawali dengan tanda “titik” (kecuali penjumlahan/pengurangan), yaitu:

**Tabel 5**

+ -	Tambah dan kurang (elemen-per-elemen)
.* ./ \	Kali, bagi, bagi terbalik (elemen-per-elemen)
.^	Pangkat (elemen-per-elemen)

Operasi penjumlahan/pengurangan matriks secara definit sudah dilakukan elemen-per-elemen, sehingga + dan - tidak diawali "titik".

Sekarang kita coba praktekkan contoh di bawah ini.

```
>> A=[1 -2;1 5]; B=[7 5; 2 0];
```

```
>> A+B
```

```
ans =  
      8      3  
      3      5
```

```
>> A.*B
```

```
ans =  
      7     -10  
      2      0
```

```
>> B./A
```

```
ans =  
      7.0000   -2.5000  
      2.0000         0
```

```
>> B.^2
```

```
ans =  
      49      25  
      4       0
```

```
>> A.^B
```

```
ans =  
      1     -32  
      1       1
```

```
>> 2.^B
```

```
ans =
    128    32
     4     1
```

Perhatikan bahwa hasil operasi juga berupa matriks berukuran sama dengan **A** dan **B**.

Pada contoh berikutnya kita coba operasi antar vektor.

```
>> a = [3 2 1]; b = [4 5 6];
>> c = [10 20 30]'; d = [5 10 15]';
```

```
>> a.*b
ans =
    12    10     6
```

```
>> c.*d
ans =
    50
   200
   450
```

```
>> a.*c
??? Error using ==> .*
Matrix dimensions must agree.
```

Perhatikan bahwa ukuran **a** dan **c** tidak cocok sehingga muncul pesan error (**a** berukuran  $1 \times 3$  sementara **c**  $3 \times 1$ ).

```
>> b.^a, c./d+2
ans =
    64    25     6
ans =
     4
```

```

4
4

>> c./2.*d.^2
ans =
    125
   1000
   3375

```

Ingat, operasi pangkat selalu dilakukan lebih dulu, diikuti perkalian/pembagian, kemudian penjumlahan/pengurangan.

### 3.10. Fungsi Elemen-per-Elemen

Semua fungsi matematik yang berlaku pada skalar (lihat kembali subbab 2.4), berlaku pula untuk matriks/vektor secara elemen-per- elemen. Pada contoh kali ini, kita akan mencoba beberapa contoh sederhana, kemudian kita coba pula dua kasus perhitungan dengan memanfaatkan berbagai fungsi yang telah kita pelajari.

```

>> n=-3:3
n =
   -3   -2   -1    0    1    2    3

>> abs(n), sign(n)
ans =
    3    2    1    0    1    2    3
ans =
   -1   -1   -1    0    1    1    1

>> round(n./2), floor(n./2), ceil(n./2)

```

```

ans =
    -2    -1    -1     0     1     1     2
ans =
    -2    -1    -1     0     0     1     1
ans =
    -1    -1     0     0     1     1     2

>> rem(n,3)
ans =
     0    -2    -1     0     1     2     0

```

Contoh kasus:

Anda ditugasi membuat tabel trigonometri: sinus dan cosinus untuk sudut-sudut istimewa:  $0^\circ$ ,  $30^\circ$ ,  $45^\circ$ ,  $60^\circ$ ,  $90^\circ$ , ...,  $360^\circ$ . Dalam tugas ini akan digunakan pula *command* **sort** untuk mengurutkan data dan **disp** untuk menampilkan isi variabel di layar.

Mula-mula, kita definisikan **x** sebagai sudut-sudut istimewa, berupa sudut kelipatan  $30^\circ$  mulai  $0^\circ$  hingga  $360^\circ$ . Kemudian kita tambahkan empat sudut istimewa:  $45^\circ$ ,  $135^\circ$ ,  $225^\circ$ , dan  $315^\circ$ , lalu kita urutkan isi vektor **x**.

```

>> clear
>> x=0:30:360;
>> x=[x 45 135 225 315];

>> x=sort(x)
x =
Columns 1 through 13
     0    30    45    60    90   120   135   150   180   210   225   240   270

Columns 14 through 17
   300   315   330   360

```



$x$  dalam satuan derajat kita ubah menjadi  $t$  (radian), karena perhitungan trigonometri dilakukan dalam satuan radian.

```
>> t=x.*pi/180;
>> y1=sin(t); y2=cos(t);
```

Selanjutnya kita buat matriks tiga kolom bernama **tabel** berisi: sudut, sin, dan cos.

```
>> tabel=[x;y1;y2]';
>> judul='      sudut      sin      cos';
```

Ingat, vektor  $x$ ,  $y1$ , dan  $y2$  berupa satu baris; padahal kita ingin menampilkannya memanjang ke bawah berupa kolom, jadi perlu dilakukan transposisi.

```
>> disp(judul), disp(tabel)
      sudut      sin      cos
           0           0      1.0000
  30.0000      0.5000      0.8660
  45.0000      0.7071      0.7071
  60.0000      0.8660      0.5000
  90.0000      1.0000      0.0000
 120.0000      0.8660     -0.5000
 135.0000      0.7071     -0.7071
 150.0000      0.5000     -0.8660
 180.0000      0.0000     -1.0000
 210.0000     -0.5000     -0.8660
 225.0000     -0.7071     -0.7071
 240.0000     -0.8660     -0.5000
 270.0000     -1.0000     -0.0000
 300.0000     -0.8660      0.5000
```

315.0000	-0.7071	0.7071
330.0000	-0.5000	0.8660
360.0000	-0.0000	1.0000

## MODUL 2 [Praktikum 5 – 7]

### GRAFIK DAN SUARA

Salah satu keunggulan MATLAB ialah kemampuannya dalam menampilkan/mengolah grafik dan suara dengan *command* yang sederhana dan fleksibel. Pada bab ini ini kita akan belajar mengenai visualisasi data (plot grafik 2-dimensi dan 3-dimensi), serta penyuaran.

#### Tujuan Khusus Praktikum :

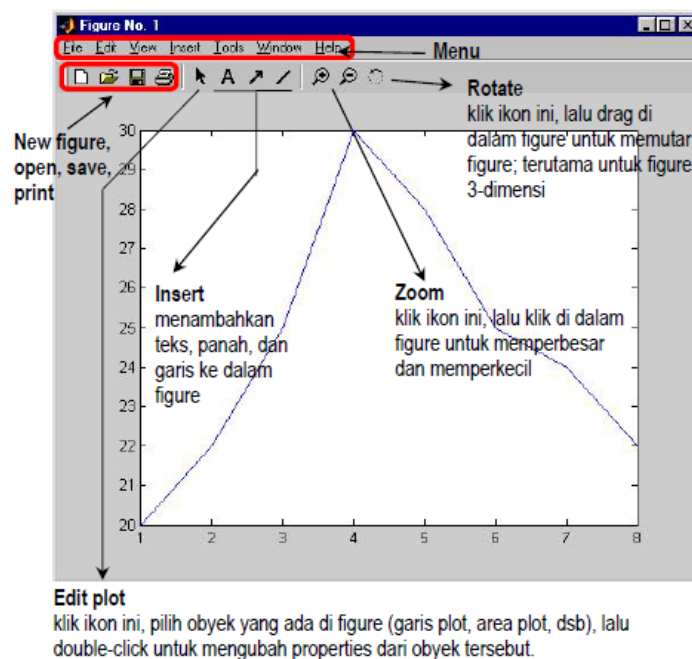
Mahasiswa dapat menjalankan fungsi-fungsi Matlab untuk menyelesaikan masalah visualisasi

### 1. Plot 2-Dimensi

Untuk memvisualisasi data secara 2-dimensi ataupun 3-dimensi, kita menggunakan berbagai *command* plotting; di mana *command* yang paling dasar ialah plot. Anda bisa praktekkan contoh berikut ini.

```
>> x = 1:8; y=[20 22 25 30 28 25 24 22];  
>> plot(x,y)
```

Akan muncul *window* baru berisi figure hasil plotting. Perhatikan kegunaan dari ikon yang ada.



**Gambar 4**  
**Jendela figure**

Seperti yang Anda lihat, titik (1,20), (2,22), (3,25), (4,30), dst... terhubung dengan garis lurus. Sekarang Anda bisa coba untuk membalik urutan sintaks dan mengamati grafik yang dihasilkan!

```
>> plot(y,x)
```

Setiap gambar di *figure window*, bisa Anda print melalui menu **File→Print** (Ctrl+P), atau Anda simpan **File→Save** (Ctrl+S), ataupun Anda ekspor sebagai file JPG, EMF, BMP, dsb dengan **File→Export**.

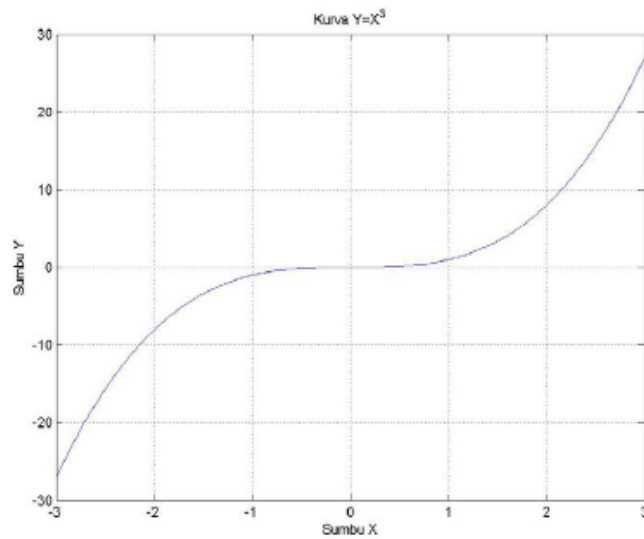
Untuk menambahkan judul, label, dan grid ke dalam hasil plot Anda, digunakan *command* berikut ini.

**Tabel 6**

<b>xlabel</b>	memberi label pada sumbu-x
<b>ylabel</b>	memberi label pada sumbu-y
<b>title</b>	memberi judul di atas area plot
<b>grid on</b>	memunculkan grid di dalam area plot
<b>grid off</b>	menghapus grid

Sekarang mari kita lihat contoh plot yang lain. Kita akan memplot kurva  $y = x^2$  pada rentang  $x = -3$  hingga  $x = +3$ .

```
>> clear
>> x=-3:0.1:3; %inkremen=0.1 agar kurva terlihat mulus
>> y=x.^3;
>> plot(x,y)
>> xlabel('Sumbu X'), ylabel('Sumbu Y')
>> title('Kurva Y=X^3')
>> grid on
```



**Gambar 5**  
**Contoh plot: kurva  $Y = X^3$**

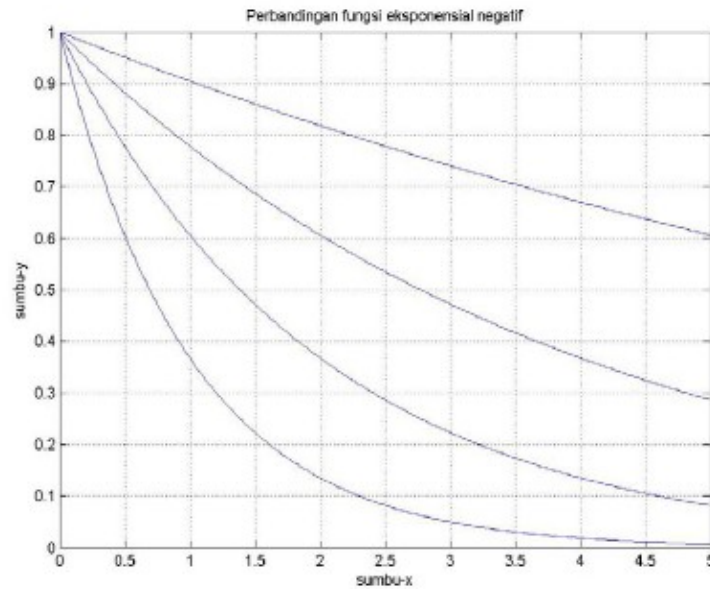
Ketika Anda menggunakan *command plot*, gambar sebelumnya di *figure window* akan terhapus. Lalu bagaimana jika kita ingin memplot beberapa fungsi dalam satu figure sekaligus? Dalam hal ini kita bisa gunakan *command hold*.

**Tabel 7**

<b>hold on</b>	untuk ‘menahan’ gambar sebelumnya supaya tak terhapus ketika ditimpa gambar baru
<b>hold off</b>	untuk menonaktifkan <i>command hold</i>

Berikut ini contoh memplot beberapa kurva eksponensial negatif sekaligus.

```
>> clear
>> x=linspace(0,5,500);
>> y1=exp(-x); plot(x,y1);
>> grid on
>> hold on
>> y2=exp(-0.5*x); plot(x,y2);
>> y3=exp(-0.25*x); plot(x,y3);
>> y4=exp(-0.1*x); plot(x,y4);
>> xlabel('sumbu-x'), ylabel('sumbu-y')
>> title('Perbandingan fungsi eksp onensial ...
negatif')
```



**Gambar 6**  
**Hasil plot dengan “hold on”**

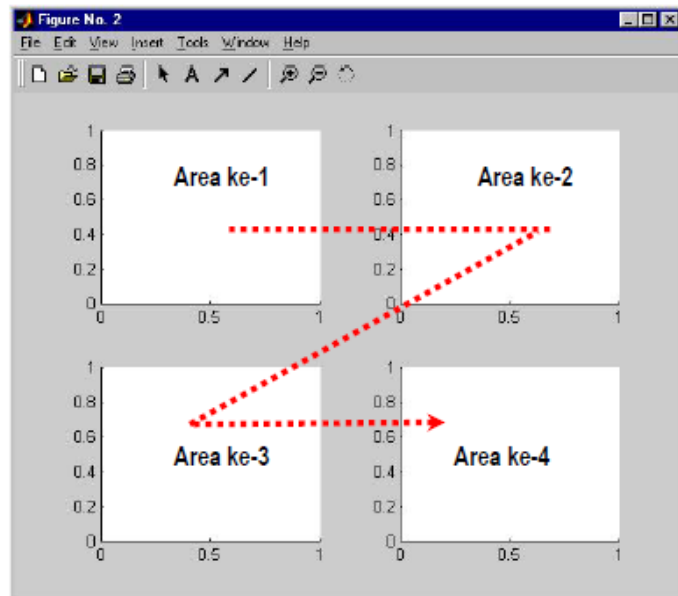
## 2. Lebih Jauh Mengenai Plot

Anda mungkin ingin memplot beberapa fungsi dalam beberapa *figure window* yang terpisah, atau membagi satu *window* menjadi sejumlah area plot, ataupun mengatur properties dari plot yang akan digambar. Beberapa *command* di bawah ini bisa digunakan untuk tujuan tersebut.

**Tabel 8**

<b>figure</b>	menciptakan <i>figure window</i> baru yang kosong dan siap untuk di-plot
<b>figure(k)</b>	untuk ‘menduduki’ <i>figure window</i> nomor-k
<b>subplot(m,n,k)</b>	membagi <i>figure window</i> menjadi m-baris $\times$ n-kolom area plot yang terpisah, dan menduduki area ke-k
<b>clf</b>	“clear figure”, mengosongkan <i>figure window</i> yang sedang ‘diduduki’

Misalkan *figure window* berikut dibagi menjadi 2-baris  $\times$  2-kolom dengan **subplot**. Perhatikan urutan nomor area dari kiri-atas ke kanan-bawah.



**Gambar 7**  
Pembagian area plot dengan “subplot”

**Tabel 8 (lanjutan)**

**plot(x,y,'string')** menciptakan plot 2-dimensi dari vektor **x** versus vektor **y**, dengan property yang ditentukan oleh **string**, sebagai berikut:

Warna	Jenis Garis	Jenis Point
<b>b</b> biru	- utuh	. titik
<b>g</b> hijau	: titik-titik	<b>o</b> lingkaran
<b>r</b> merah	-. titik-strip	<b>x</b> tanda ×
<b>c</b> biru muda	-- putus-putus	+ tanda +
<b>m</b> ungu		* tanda *
<b>y</b> kuning		<b>s</b> bujur sangkar
<b>k</b> hitam		<b>d</b> permata
<b>w</b> putih		<b>v</b> segitiga ke bawah
		^ segitiga ke atas
		< segitiga ke kiri
		> segitiga ke kanan
		<b>p</b> segilima
		<b>h</b> segienam

Misalkan:

**plot(x,y,'r-')** memplot **x** versus **y** dengan garis utuh warna merah

**plot(x,y,'k\*')** menempatkan tanda \* warna hitam untuk setiap titik **x** versus **y**.

**plot(x,y,'g--s')** memplot dengan garis putus-putus warna hijau dan menempatkan tanda bujur sangkar di setiap titik **x** versus **y**.

Perlu diingat bahwa **'string'** dalam plot bersifat opsional. Apabila tidak dituliskan maka digunakan garis utuh warna biru.

**Tabel 8 (lanjutan)**

<b>plot(x1,y1,'string1',x2,y2,'string2',x3,y3,'string3', ... )</b>	menciptakan sejumlah plot sekaligus dalam satu area plot: <b>x1</b> versus <b>y1</b> dengan property <b>string1</b> , <b>x2</b> versus <b>y2</b> dengan property <b>string2</b> , dan seterusnya
<b>legend('ket1','ket2','ket3', ...)</b>	menambahkan legenda ke dalam plot yang telah dibuat; <b>ket1</b> untuk plot pertama, <b>ket2</b> untuk plot kedua, dan seterusnya
<b>axis off</b>	menghilangkan tampilan sumbu koordinat pada plot
<b>axis on</b>	menampakkan kembali sumbu koordinat
<b>axis([x_awal x_akhir y_awal y_akhir])</b>	membuat tampilan area plot pada batas-batas nilai <b>x = x_awal</b> hingga <b>x_akhir</b> , dan nilai <b>y = y_awal</b> hingga <b>y_akhir</b>
<b>axis equal</b>	mengubah skala sumbu-x dan sumbu-y menjadi sama
<b>axis square</b>	mengubah bentuk area plot menjadi bujur sangkar

Berbagai fungsi yang berkaitan dengan plot di atas, berlaku pula untuk plot diskrit, plot logaritmik dan plot dalam koordinat polar.



Tabel 9

<b>stem( ... )</b>	sama dengan <b>plot( ... )</b> , tetapi menampilkan <b>y</b> sebagai data diskrit
<b>semilogy( ... )</b>	sama dengan <b>plot( ... )</b> , kecuali sumbu-y menggunakan skala logaritmik (basis 10)
<b>semilogx( ... )</b>	sama dengan <b>plot( ... )</b> , kecuali sumbu-x menggunakan skala logaritmik
<b>loglog( ... )</b>	sama dengan <b>plot( ... )</b> , tetapi sumbu-x dan sumbu-y menggunakan skala logaritmik
<b>polar(theta,rho,'string')</b>	membuat plot dalam koordinat polar dari sudut <b>theta</b> (satuan radian) versus radius <b>rho</b> , dengan property ditentukan oleh <b>string</b>

Kini saatnya mencoba berbagai *command* di atas dalam contoh berikut ini.

Pertama, kita akan mencoba memplot kurva eksponensial negatif seperti pada contoh subbab 5.1 secara lebih efisien.

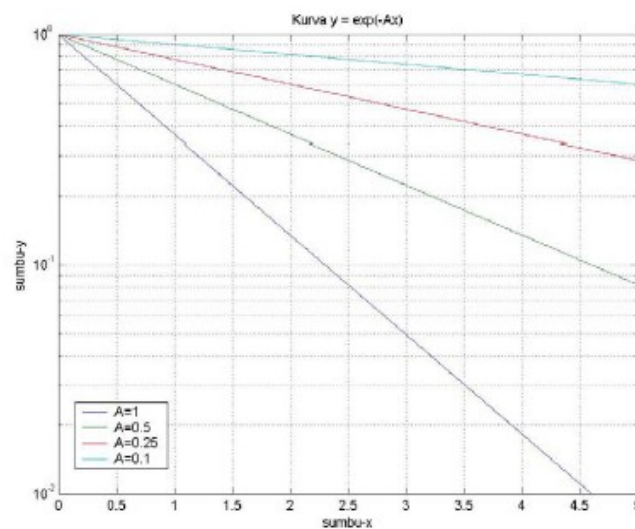
```
>> clear
>> x=linspace(0,5,500);
>> y1=exp(-x); y2=exp(-0.5*x); y3=exp(-0.25*x);
>> y4=exp(-0.1*x);
>> plot(x,y1,x,y2,x,y3,x,y4)
>> grid on
>> xlabel('sumbu-x'), ylabel('sumbu-y')
>> title('Kurva y = exp(-Ax)')
>> legend('A=1','A=0.5','A=0.25',' A=0.1')
```

Kemudian, kita coba memplot kurva tersebut dalam skala semilogaritmik

```
>> figure
>> semilogy(x, y1, x, y2, x, y3, x, y4)
>> grid on
>> xlabel('sumbu-x'), ylabel('sumbu-y')
>> title('Kurva y = exp(-Ax)')
>> legend('A=1', 'A=0.5', 'A=0.25', 'A=0.1')
```

Misalkan kita ingin menyempitkan area plot pada  $y = 1$  hingga  $10^2$  saja, maka:

```
>> axis([0 5 1e-2 1])
```



**Gambar 8**  
**Contoh plot semi-logaritmik**

Dalam contoh kedua, kita akan memplot gelombang sinus, cosinus, kotak, dan gigi gergaji dengan melibatkan *command subplot*.

```
>> figure
>> t=0:0.05:10;

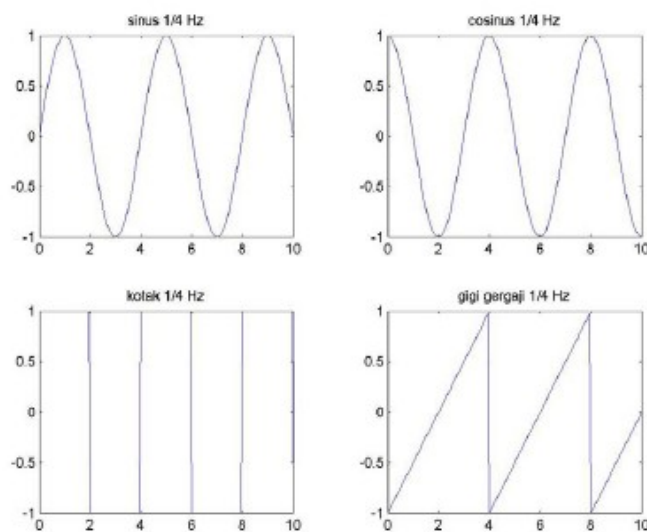
>> sinus=sin(2*pi*0.25*t);
>> cosinus=cos(2*pi*0.25*t);
>> kotak=square(2*pi*0.25*t);
>> gigi=sawtooth(2*pi*0.25*t);

>> subplot(2,2,1);
>> plot(t,sinus), title('sinus 1/4 Hz')

>> subplot(2,2,2);
>> plot(t,cosinus), title('cosinus 1/4 Hz')

>> subplot(2,2,3);
>> plot(t,kotak), title('kotak 1/4 Hz')

>> subplot(2,2,4);
>> plot(t,gigi), title('gigi gergaji 1/4 Hz')
```



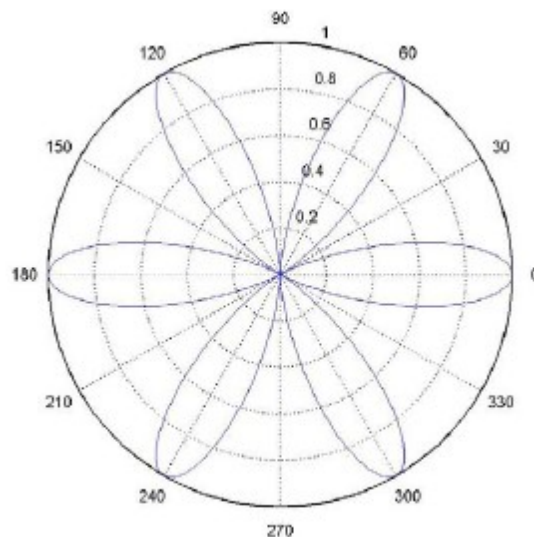
**Gambar 9**  
**Contoh penggunaan subplot**

Dalam contoh ketiga, kita akan mencoba memplot suatu fungsi matematis dalam koordinat polar. Diinginkan plot fungsi:

$$= \sin^2(3\theta)$$

dalam MATLAB dituliskan

```
>> figure
>> theta=linspace(0,2*pi,500);
>> rho=(cos(theta.*3)).^2;
>> polar(theta,rho);
```



**Gambar 10**  
Contoh plot dengan *command* “polar”

### 3. Plot 3-Dimensi

Dalam subbab ini akan dibahas tiga macam plot 3-dimensi: plot garis, plot permukaan (surface), dan plot kontur.

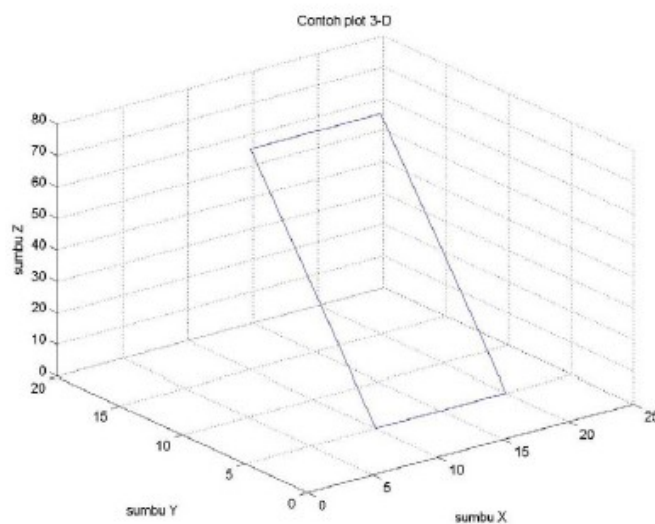
#### 3.1. Plot Garis

Mari kita mulai dengan plot garis di dalam ruang 3-dimensi. Ini mirip dengan plot 2-dimensi, tetapi kali ini kita gunakan *command* `plot3(...)`, dan dibutuhkan vektor *z*, untuk dimensi ketiga.

```

>> X = [10 20 20 10 10];
>> Y = [5 5 15 15 5];
>> Z = [0 0 70 70 0];
>> plot3(X,Y,Z); grid on;
>> xlabel('sumbu X'); ylabel('sumbu Y');
>> zlabel('sumbu Z');
>> title('Contoh plot 3-D');
>> axis([0 25 0 20 0 80])

```



**Gambar 11**  
**Contoh plot 3-dimensi dengan *command* “plot3”**

Perhatikan bahwa *command* **label**, **title**, **grid**, **axis**, **hold**, dan **subplot** juga berlaku di sini. Anda juga bisa merotasi gambar 3- dimensi tersebut dengan cara men-klik ikon rotate dan dragging mouse di atas gambar.

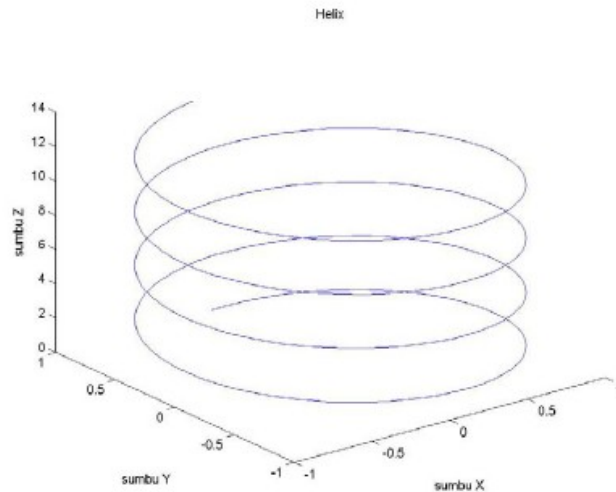
Sekarang kita coba contoh yang lain untuk menggambarkan helix.

```

>> t=0:0.1:25;
>> X=sin(t); Y=cos(t); Z=0.5*t;
>> plot3(X,Y,Z)

```

```
>> xlabel('sumbu X'); ylabel('sumbu Y');
>> zlabel('sumbu Z');
>> title ('Helix');
```



**Gambar 12**  
Contoh penggunaan “plot3”

### 3.2. Plot Permukaan

Sementara itu, untuk plot permukaan (surface) dalam ruang 3- dimensi digunakan *command* **mesh** atau **surf**. Contoh berikut ini menggambarkan fungsi dua variabel  $z = x^2 + y^2$

Caranya ialah:

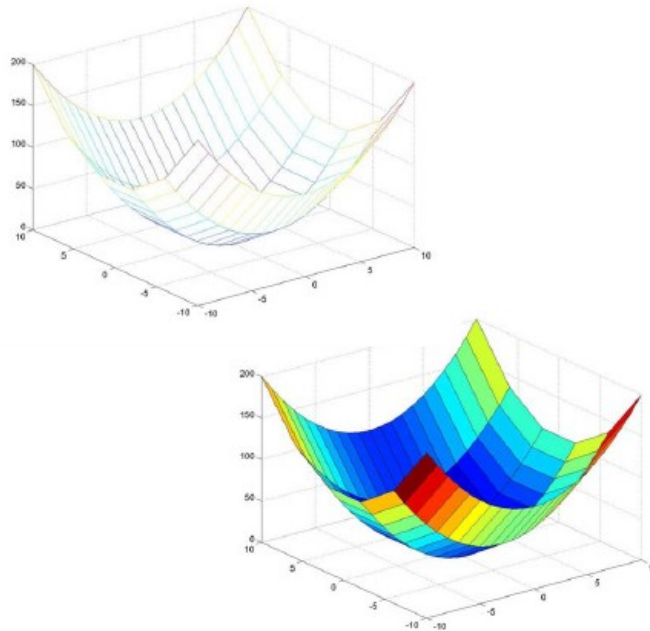
- 1) Definisikan batas-batas nilai x dan y yang akan diplot
- 2) Gunakan *command* **meshgrid** untuk “mengisi” bidang-XY dengan jalinan titik
- 3) Hitunglah fungsi 3-dimensi untuk jalinan titik tersebut
- 4) Buatlah plot dengan *command* **mesh** atau **surf**.

Sebagai contoh:

```
>> batas_x = -10:1:10; batas_y = -10:4:10;  
>> [X,Y] = meshgrid(batas_x,batas_y);  
>> Z = X.^2 + Y.^2;  
>> mesh(X,Y,Z);
```

Kini Anda mendapatkan plot 3-dimensi. Kini cobalah

```
>> surf(X,Y,Z);
```

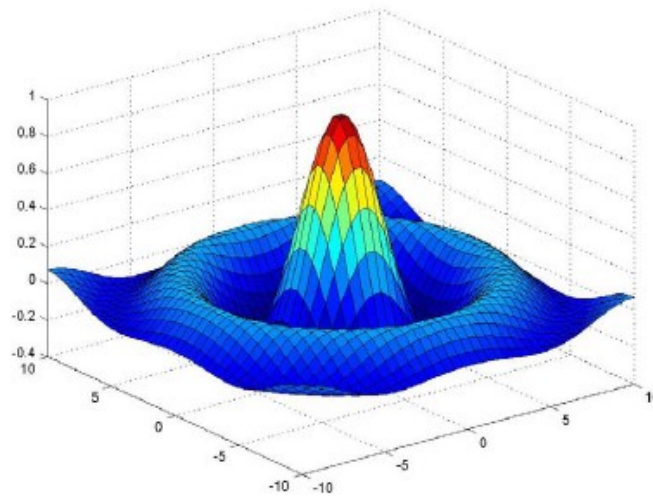


**Gambar 13**  
Hasil plot dengan “mesh” dan “surf”

Amatilah perbedaan hasil antara **mesh** dan **surf** ! Anda juga bisa menambahkan “label” dan “title” seperti plot pada umumnya.

Sekarang kita coba contoh yang lain untuk memplot fungsi 3- dimensi

```
>> x = linspace(-10,10,40); y = x;
>> [X,Y] = meshgrid(x,y);
>> R = sqrt(X.^2+Y.^2);
>> Z = sin(R)./(R+eps);
>> surf(X,Y,Z);
```



**Gambar 14**  
Plot 3-dimensi dari fungsi  $\sin(r) / r$

di sini kita menggunakan variabel **eps**, untuk mencegah perhitungan  $0/0$  ketika  $R = 0$ .

### 3.3. Plot Kontur

Fungsi dua variabel, misalkan  $z = f(x, y)$  bisa kita gambarkan konturnya dalam dua dimensi dengan *command* berikut ini:

**Tabel 10**

<b>contour(X,Y,Z)</b>	menggambar kontur dari nilai di <b>Z</b> dengan 10 level. Elemen <b>Z</b> diterjemahkan sebagai level-level di atas bidang (x,y)
<b>C = contour(X,Y,Z)</b>	menghitung matriks kontur <b>C</b>
<b>contour(X,Y,Z,n)</b>	menggambar kontur dengan <i>n</i> level

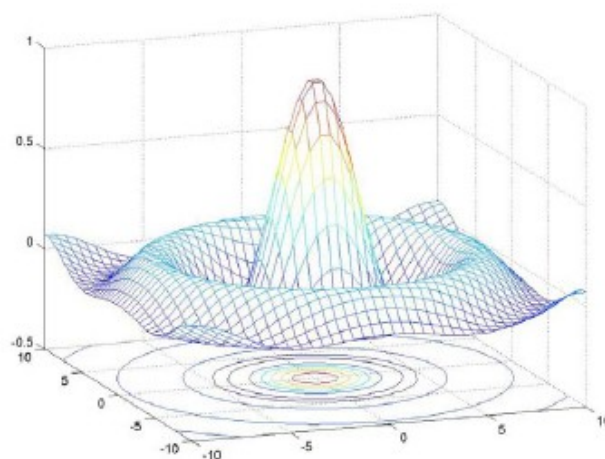
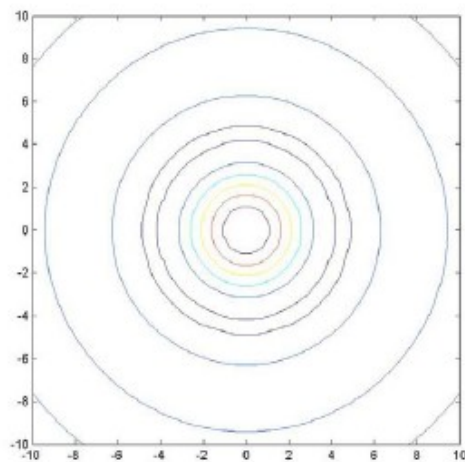


<b>contour( ... , 'string')</b>	menggambar kontur dengan <i>property</i> yang ditentukan oleh string (lihat Tabel 8)
<b>clabel(C)</b>	menuliskan angka pada garis-garis kontur untuk menunjukkan level
<b>meshc(X,Y,Z)</b>	menggambar permukaan seperti pada <i>command</i> <b>mesh</b> , dan juga memplot kontur pada dasar grafik.

Mari kita gambarkan kontur dari fungsi  $\sin(r)/r$  di atas, lalu bandingkan dengan plot permukaannya:

```
>> figure; contour(X,Y,Z);
```

```
>> figure; meshc(X,Y,Z);
```



**Gambar 15**  
Contoh plot kontur

## 5. Suara

Untuk menyuarkan suatu vektor, ataupun membaca dan menyimpan file audio berformat WAV, digunakan *command* berikut ini:

**Tabel 11**

<b>[x,Fs] = wavread('nama_file')</b>	
	membaca file WAV dan menyimpannya dalam vektor <b>x</b> , serta mengembalikan frekuensi sampling <b>Fs</b> dari file tersebut. <i>Command</i> ini juga bisa membaca file WAV multi kanal
<b>wavwrite(x,Fs,'nama_file')</b>	
	menuliskan file WAV dari vektor <b>x</b> dengan frekuensi sampling <b>Fs</b>
<b>sound(x,Fs)</b>	
	menyuarkan vektor <b>x</b> dengan frekuensi sampling <b>Fs</b>
<b>soundsc(x,Fs)</b>	
	sama seperti sintaks sebelumnya, namun vektor <b>x</b> terlebih dahulu diskalakan pada selang $-1 \leq \mathbf{x} \leq +1$

File yang akan dibaca harus tersimpan di direktori **Matlab\work**, atau Anda harus merinci drive, direktori dan nama file jika file tersimpan di direktori lain.

Sebagai gambaran, marilah kita dengarkan suara berikut ini.

Pertama, suara pitch 400 Hz berdurasi 2 detik.

```
>> Fs=8000;           %frekuensi sam pling 8 kHz
>> t=0:1/Fs:2;       %sinyal berdur asi 2 detik
>> frek=400;         %frekuensi sin yal 400 Hz
>> m=cos(2*pi*frek*t);
>> sound(m, Fs);     %suara dari m
>> wavwrite(m, Fs, 'tone_400Hz.wav'); ...
%Menyimpan vektor m ke dalam file
```

Berikutnya, memperdengarkan suara helikopter yang ada di file **heli.wav**.

```
>> [x,Fs]=wavread('heli.wav'); %Me mbaca file heli.wav
>> sound(x, Fs);     %Suara helikopter
```

## **MODUL 3 [Praktikum 8 – 11]**

### **M-FILE DAN PEMROGRAMAN MATLAB**

Pada bab-bab yang lalu, Anda telah belajar berinteraksi dengan MATLAB menggunakan *command window*. Sekarang, katakanlah Anda harus mempergunakan sederetan *command* secara berulang-ulang di dalam sesi MATLAB yang berbeda. Akan sangat repot jika Anda harus mengetikkan *command* tersebut secara manual di *command window* setiap kali Anda butuhkan. Namun dengan M- file, deretan *command* tersebut bisa Anda simpan dalam bentuk skrip teks. Kapan saja Anda butuhkan, skrip tersebut bisa dijalankan/dieksekusi secara otomatis dengan cara mengetikkan nama M-file yang bersangkutan di *command window*.

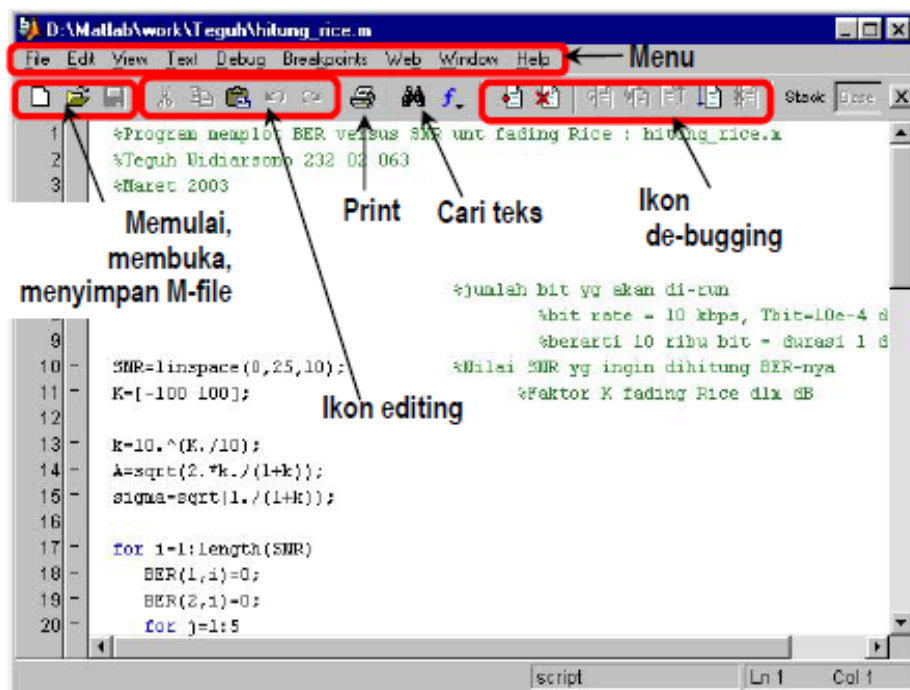
Kali ini kita akan belajar mengenal M-file dengan contoh sederhana. Namun demikian perlu diketahui bahwa MATLAB sebenarnya merupakan bahasa pemrograman umum, seperti halnya Basic, C, Java, Pascal, Fortran, dll. Sehingga dalam bab ini kita akan menitikberatkan pada pelajaran pemrograman komputer.

#### **Tujuan Khusus Praktikum :**

*Mahasiswa dapat menulis program dalam programming editor dan dapat menjalankannya dalam lembar kerja Matlab*

#### **1. Membuat M-File**

Untuk menuliskan skrip M-file, Anda bisa mulai dengan membuka file baru. Caranya ialah melalui menu di *main window*: File Open atau File New M-file; atau dengan mengklik ikon yang ada di jendela utama. Sebuah jendela editor akan terbuka seperti gambar berikut ini.



**Gambar 16**  
**Jendela editor M-file**

Dengan editor ini, kita bisa membuka sejumlah M-file, melakukan editing, ataupun mencoba menjalankannya dan melakukan debugging (mencari kesalahan di dalam skrip).

Sementara itu, untuk menyimpan M-file, Anda bisa lakukan dengan menu: **File**→**Save** atau **File**→**Save As**; ataupun dengan mengklik ikon yang ada.

Namun demikian, sebenarnya Anda juga bisa menuliskan M-file dengan sebarang editor teks, seperti MS Word, Notepad, dll.; yang penting Anda menyimpan file tersebut dengan ekstensi **\*.m**.

## 2. M-File Sebagai Skrip Program

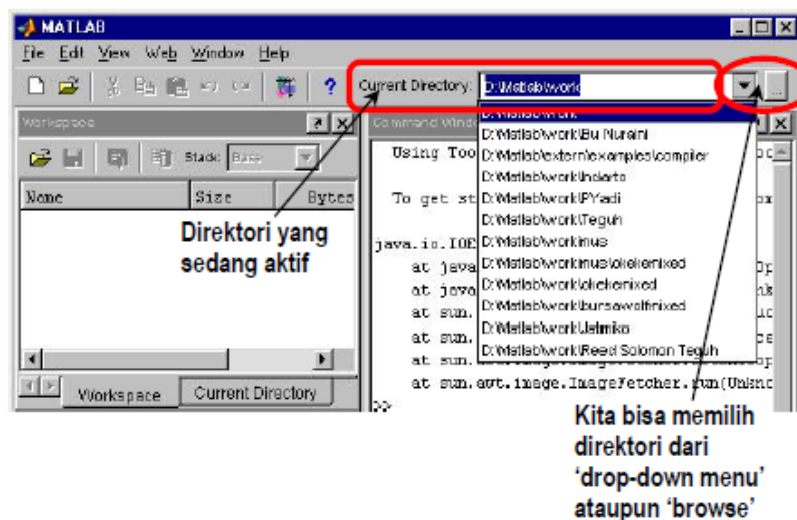
Pada bagian ini, kita akan menggunakan M-file untuk menjalankan sederetan *command* yang kita tuliskan sebagai skrip. Mari kita mulai dengan skrip sederhana untuk menghitung rata-rata dari lima bilangan. File ini kita namakan **rata\_rata.m**.

Bukalah M-file baru lalu ketikkan skrip berikut ini.

```
% Program sederhana untuk menghitung
% rata-rata 5 bilangan:
% rata_rata.m
a = 50;
b = 100;
c = 150;
d = 200;
e = 250;
% Menghitung dan menampilkan rata-rata
hasil = (a + b + c + d + e)/5;
hasil
```

Teks yang diawali tanda “%” menunjukkan komentar, dan tidak akan dieksekusi oleh MATLAB.

Simpanlah file ini di dalam direktori **Matlab\work** dengan nama **rata\_rata.m**. Sekarang cobalah jalankan dari *command window*. Sebelumnya pastikan bahwa direktori menunjuk ke **Matlab\work**. Perhatikan “Current Directory” yang ada di jendela utama MATLAB. Kita bisa mengubah direktori yang sedang aktif melalui drop-down menu ataupun melalui *browse*.



**Gambar 17**  
Memilih direktori untuk menjalankan M-file

```

>> clear
>> rata_rata
hasil =
150
>> whos
Name           Size           Bytes  Cl ass
-----
a              1x1             8  do uble array
ans            1x1             8  do uble array
b              1x1             8  do uble array
c              1x1             8  do uble array
d              1x1             8  do uble array
e              1x1             8  do uble array
hasil          1x1             8  do uble array

```

Grand total is 7 elements using 56 bytes

Perhatikan bahwa:

- \* Di dalam M-file, setiap *command* diakhiri dengan titik-koma supaya hasil perhitungan di tiap baris tidak ditampilkan di *command window*. Kecuali pada hasil perhitungan yang ingin kita tampilkan, tidak diakhiri titik-koma.
- \* Variabel yang didefinisikan di dalam M-file akan disimpan oleh MATLAB ketika M-file telah dieksekusi.

Di dalam editor, skrip yang kita tuliskan akan memiliki warna tertentu:

- \* hijau untuk komentar
- \* hitam untuk variabel dan *command*
- \* biru untuk statement pemrograman.

Sekarang, marilah kita mencoba M-file lain untuk menghitung sisi miring suatu segi tiga siku-siku dengan formula pythagoras, menghitung luasnya, dan kelilingnya.

```
% Program menghitung segi-3 siku-siku: segi3.m
% Untuk menghitung sisi miring, lu as, dan keliling

% Mendefinisikan sisi siku-siku se gitiga
Sisi_A = 3;
Sisi_B = 4;

% Menghitung sisi miring
Sisi_C = sqrt(Sisi_A^2 + Sisi_B^2)

% Menghitung luas segitiga
Luas = 1/2* Sisi_A * Sisi_B

% Menghitung keliling
Keliling = Sisi_A + Sisi_B + Sisi_C
```

Lalu simpan dengan nama **segi3.m**.

Sekarang kita panggil M-file tersebut

```
>> segi3
Sisi_C =
5
Luas =
6
Keliling =
12
```

Sekarang Anda bisa mencoba sendiri membuat program yang lebih menantang, seperti menghitung dan memplot fungsi 2 ataupun 3-dimensi dengan M-file.

### 3. M-File Sebagai Fungsi

Sebagai skrip program, jika kita ingin mengubah/mengatur parameter masukan program, maka harus kita lakukan di dalam editor. Padahal seringkali kita harus menjalankan satu program/algoritma berulang kali dengan nilai masukan yang berbeda-beda, misalkan dalam proses iterasi atau optimasi. Untuk keperluan ini, kita bisa menuliskan M-file sebagai suatu fungsi spesifik sesuai kebutuhan kita.

Dalam setiap fungsi terdapat tiga unsur:

- \* Parameter masukan; dalam hal ini kita sebut sebagai “argumen input”. Jumlah parameter (argumen) tersebut bisa sebarang (satu, dua, sepuluh, atau tidak ada argumen input sama sekali). Jenis argumen pun sebarang (variabel, bilangan ataupun teks).
- \* Proses di dalam program; berupa sederetan *command* untuk menjalankan suatu algoritma tertentu.
- \* Parameter keluaran; atau “argumen output” yang jumlah dan jenisnya sebarang.

Deklarasi fungsi di M-file harus dilakukan pada baris awal dengan sintaks:

```
function [argumen output] = nama_fungsi(argumen input)
```

Sebagai contoh awal, kita akan membuat fungsi untuk menghitung sisi miring, luas, dan keliling segitiga; seperti program yang ada pada contoh sebelumnya.

```
%Fungsi untuk menghitung segi-3 si ku-siku: segitiga.m
```

```
%Untuk menghitung sisi miring, luas, dan keliling
```

```
function [Sisi_C,Luas,Kll] = segitiga(Sisi_A,Sisi_B)
```



```

% Menghitung sisi miring
Sisi_C = sqrt(Sisi_A^2 + Sisi_B^2) ;

% Menghitung luas segitiga
Luas = 1/2* Sisi_A * Sisi_B;

% Menghitung keliling
Kll = Sisi_A + Sisi_B + Sisi_C;

```

Lalu simpan dengan nama “**segitiga.m**”.  
Sekarang Anda panggil fungsi tersebut.

```

>> clear
>> [Hyp,Area,Circum]=segitiga(12,1 6)
Hyp =
20
Area =
96
Circum =
48

```

Dari contoh sederhana tersebut, ada beberapa hal yang perlu kita perhatikan:

- \* Dalam fungsi **segitiga**, terdapat dua argumen input (**Sisi\_A**, **Sisi\_B**), dan tiga argumen output (**Sisi\_C**, **Luas**, **Kll**).
- \* Ketika dipanggil di *command window*, kita bisa menggunakan nama argumen input/output yang berbeda dengan di M-file, namun urutannya tidak berubah. Di dalam contoh, argumen **Sisi\_A** dan **Sisi\_B** kita isi dengan bilangan, sementara argumen **Sisi\_C**, **Luas**, dan **Keliling** kita panggil dengan **Hyp**, **Area**, dan **Circum**.

Sekarang kita lihat dengan *command whos*:

```
>> whos
Name          Size          Bytes  Class

Area          1x1            8      double array
Circum        1x1            8      double array
Hyp           1x1            8      double array

Grand total is 3 elements using 24 bytes
```

Terlihat bahwa variabel yang dideklarasikan di dalam fungsi tidak disimpan, melainkan dimusnahkan ketika suatu fungsi selesai dijalankan. Yang ada di sana hanyalah variabel yang telah dideklarasikan di *command window* untuk menyimpan nilai output. Hal ini merupakan salah satu perbedaan utama antara skrip program dengan fungsi.



**Penting!** Ketika membuat fungsi dengan M-file, nama file harus sama dengan nama fungsi yang dideklarasikan dalam sintaks `function ...`.  
Aturan penamaan M-file sama dengan penamaan variabel! Lihat kembali aturan tersebut di subbab 2.2

Perlu diperhatikan bahwa fungsi yang telah kita buat pada dasarnya sama dengan fungsi yang telah ada di MATLAB, semisal fungsi **sin(x)** ataupun **sqrt(x)**. Misalkan kita memanggil fungsi tanpa menyebutkan argumen output, maka keluaran akan disimpan di **ans**.

#### 4. Display dan Input

Adakalanya kita membutuhkan interaksi dengan pengguna program untuk memasukkan parameter tertentu di awal/tengah program. Dalam hal ini kita bisa menggunakan cara sederhana dengan *command input*. Sementara *command disp* digunakan untuk menampilkan teks di layar.

Misalkan kita akan membuat program untuk menghitung jumlah kombinasi team basket yang mungkin dari sejumlah mahasiswa.

```
% Program menghitung kombinasi : hit_komb.m
% untuk menghitung jumlah kombinasi
% dari sejumlah populasi

% Menampilkan judul program
clc;
disp('Menghitung Kombinasi');
disp('-----');

% Meminta masukan dari user
n = input('Berapa jumlah mahasiswa yang ada? ');
r = input('Berapa jumlah personel satu team? ');

% Menghitung kombinasi
kombinasi = factorial(n)/factorial(r)/factorial(n-r);

% Menampilkan keluaran
disp('Jumlah kombinasi yang ada = ',kombinasi);
```

Kita coba jalankan program tersebut:

```
>> hit_komb
```

```
Menghitung Kombinasi
-----
```

Berapa jumlah mahasiswa yang ada? : 8

Berapa jumlah personel satu team? : 5

Jumlah kombinasi yang ada =

56

## 5. Control Statement

Seperti halnya bahasa program pada umumnya, kita bisa mengendalikan arah program dengan berbagai cara, berupa percabangan arah program berdasarkan kondisi tertentu, ataupun loop (perhitungan berulang) ketika kita melakukan iterasi.

### 5.1. Statement `if ... elseif ... else ... end`

Ini merupakan statement untuk percabangan program berdasarkan satu/beberapa kondisi tertentu. Sintaks yang digunakan dalam MATLAB meliputi:

```
if kondisi
    Command yang dijalankan jika kondisi dipenuhi
end
```

```
if kondisi
    Command yang dijalankan jika kondisi dipenuhi
else
    Dijalankan jika kondisi tidak dipenuhi
end
```

```

if kondisi1
    Command yang dijalankan jika kondisi1 dipenuhi
elseif kondisi2
    Dijalankan jika kondisi2 dipenuhi
elseif kondisi3
    Dijalankan jika kondisi3 dipenuhi
elseif ...
    ...dst...
else
    Dijalankan jika kondisi manapun tidak dipenuhi
end

```

Selain itu, dimungkinkan pula membuat pernyataan **if** di dalam pernyataan yang lain (disebut *nested-if*), misalkan:

```

if kondisi1
    command1
    if kondisiA
        commandA
    else
        commandB
    end
else
    command2
end

```



**Penting!**

jangan keliru menuliskan **elseif** dan **else if**, karena keduanya berbeda. Yang pertama untuk menguji kondisi alternatif setelah kondisi di **if** terdahulu tak dipenuhi; tetapi yang kedua berarti **nested-if**.

## 5.2. Statement switch ... case

Sebagai alternatif dari statement **if ... elseif ... else ... end**, kita bisa menggunakan statement **switch**. Sintaksnya ialah:

```

Switch nama_variabel
case{kondisi1,kondisi2,...}
    Dijalankan jika kondisi1 atau kondisi2 dst...
    dipenuhi
case{kondisiA,kondisiB,...}
    Dijalankan jika kondisiA atau kondisiB dst...
    dipenuhi
Case(...)
    ...dst...
default
    Dijalankan jika kondisi manapun tidak dipenuhi
end

```

## 5.3. Statement for ... end

Statement ini digunakan untuk loop/perhitungan berulang. Sintaks yang digunakan dalam MATLAB ialah:

```

for variabel = nilai_awal : inkremen : nilai_akhir
    Command untuk dijalankan
end

```

Adapun sintaks yang digunakan untuk membatasi loop mirip dengan yang kita pakai untuk membuat deret (lihat kembali subbab 3.5). Misalkan untuk menampilkan bilangan kelipatan 3 dari 30 sampai 100.

```
for k = 30:3:100
    k
end
```

Hasilnya ialah:

```
k =
    30
k =
    33
k =
    ...
k =
    99
```

Sementara untuk nilai inkeremen = 1, cukup dituliskan nilai awal dan akhir. Misalkan untuk mendaftar bilangan bulat dari -10 hingga 10 dan menyimpannya dalam satu vektor.

```
Vektor=[];
for k = -10:10 %dalam hal ini inkremen = 1
    Vektor = [Vektor k];
end
Vektor
```

Menghasilkan:

```
Vektor =
Columns 1 through 13
-10  -9  -8  -7  -6  -5  -4  -3  -2  -1  0  1  2
```

Columns 14 through 21

3 4 5 6 7 8 9 10

Atau untuk memplot kurva parabola:

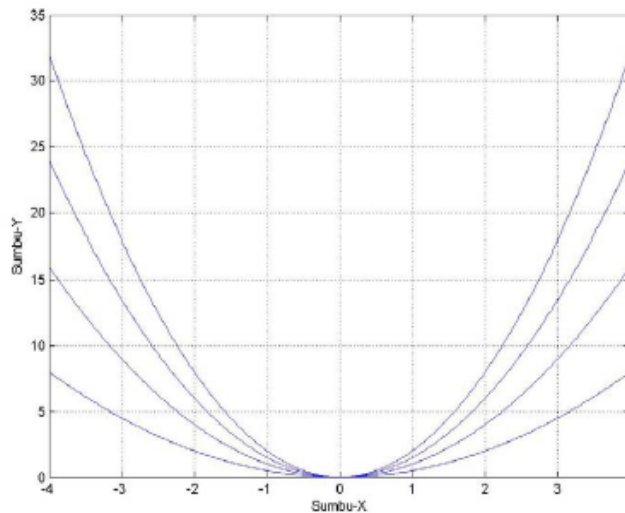
$$y = Ax^2$$

$$y = Ax^2$$

dengan berbagai nilai parameter  $A$ , yaitu 0,5 , 1 , 1,5 , dan 2. Dalam hal ini indeks vektor  $A$  kita iterasi dari 1 hingga indeks terakhir.

```
figure;
x = linspace(-4,4,500); % mendefinisikan nilai x
A = 0.5:0.5:2; % mendefinisikan vektor A
for i = 1:length(A)
    y = A(i) * x.^2;
    plot(x,y);
    hold on;
end
grid on;
```

Menghasilkan:



**Gambar 18**  
Contoh plot 4 kurva parabola dengan “for”



Perhatikan bahwa setiap selesai satu loop, **variabel** (dalam contoh di atas ialah *i*) akan otomatis mengalami inkremen. Demikian seterusnya hingga **nilai\_akhir** (yaitu `length(A)`) tercapai dan program dilanjutkan ke baris selanjutnya.

#### 5.4. Statement `while ... end`

Alternatif dari sintaks loop ialah berikut ini

```
while   kondisi
        Command untuk dijalankan jika kondisi dipenuhi
end    %keluar dari loop jika kondisi tidak dipenuhi
```

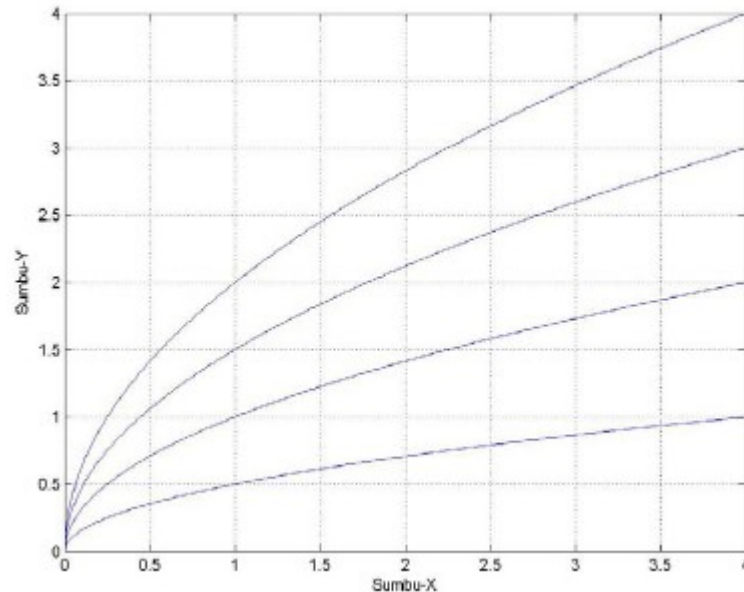
Misalkan untuk memplot fungsi akar kuadrat

$$y = Bx^{1/2}$$

dengan berbagai nilai parameter *B*.

```
figure;
x=linspace(0,4,500);
A=0.5:0.5:2;
i=1;
while i <= length(A)
    y = A(i) * x.^(1/2);
    plot(x,y); hold on;
    i=i+1;
end
grid on;
```

Menghasilkan:



**Gambar 19**  
Contoh plot 4 kurva dengan “while”

### 5.5. Statement break dan return

Ketika kita sudah berada dalam suatu loop, kita bisa keluar dengan **break** tanpa menunggu **nilai\_akhir** tercapai, atau tanpa menunggu kondisi loop tidak dipenuhi lagi. Sementara, **return** digunakan untuk keluar dari fungsi yang sedang berjalan. Berikut ini gambarannya dalam kasus penentuan apakah suatu bilangan bersifat prima atau tidak.

Algoritma yang akan digunakan ialah sebagai berikut:

- \* User memasukkan satu bilangan bulat positif **N** sebagai argumen input.
- \* Apabila **N** bukan bilangan bulat positif, maka perhitungan tidak dilanjutkan, dan digunakan return untuk keluar.
- \* **N** kita coba-coba bagi dengan 2, 3, 4, 5, ... dst. dengan loop. Apabila satu waktu ditemukan **N** habis terbagi, berarti **N** bukan bilangan prima. Selanjutnya kita langsung keluar loop dengan break dan menampilkan hasilnya di layar.

- \* Apabila **N** tidak pernah habis dibagi oleh 2, 3, 4, ... ,  $N/2$  (sampai loop selesai), maka **N** pasti bilangan prima. Selanjutnya kita tampilkan di layar dan program selesai.
- \* Untuk mengetahui apakah **N** habis terbagi atau tidak, kita bisa menggunakan fungsi **rem(N,pembagi)**.

```
% Fungsi untuk menentukan sifat pr ima suatu bilangan:
% apa_prima.m
% function apa_prima(N)

% N : bil. bulat positif yang dima sukkan oleh user

if N <= 0 %Jika N bilangan negatif
    disp('Masukan harus bilangan b ulat positif');
    return; %Perhitungan tidak dilanjutkan
end

% Membulatkan N kalau-kalau N bukan bil. bulat
N = round(N);

prima = 1; %Kita anggap N bil prima pd awal program
%flag 'prima' kita set jadi satu.

for i = 2:floor(N/2)
    if rem(N,i) == 0
        prima=0;
    % ternyata N tidak prima,
    % flag 'prima' kita set jadi nol
        break; % Keluar dari loop
    end
end
```

```
% Menampilkan hasil:
if prima == 0
    disp(N), disp('bukan bilangan  prima!');
else
    disp(N), disp('adalah bilangan  prima!');
end
```

Simpanlah fungsi ini dengan nama **apa\_prima.m** di dalam direktori **Matlab\work**.

Kita coba jalankan fungsi di atas pada *command window*:

```
>> apa_prima(37)
    37
adalah bilangan prima!
```

```
>> apa_prima(27)
    27
bukan bilangan prima!
```

```
>> apa_prima(-27)
Masukan harus bilangan bulat positif
```

Perlu diingat bahwa fungsi **apa\_prima** di atas tidak memiliki argumen keluaran, karena hasil perhitungan langsung kita tampilkan di layar menggunakan **disp**, sehingga hasil tersebut tidak bisa disimpan dalam variabel.

## 5.6. Statement *continue*

Statement *continue* digunakan untuk memaksa program untuk langsung menuju iterasi berikutnya dari suatu loop, tanpa mengeksekusi *command* yang masih ada di bawahnya.

Sebagai contoh, kita akan membuat fungsi untuk mengumpulkan bilangan tak nol dari suatu vektor.

```
% Fungsi untuk mengumpulkan bilangan
% tak nol di dalam vektor
% hit_taknol.m

function y = hit_taknol(x)
% x : vektor masukan
% y : vektor berisi bilangan tak nol dari x

y = [];
for i=1:length(x)
    if x(i)==0
        continue
    else
        y=[y x(i)];
    end
end
end
```

Sekarang kita coba:

```
>> x = [0 0 2 -3.6 0 0 0 3 0 -0.6 10 0 0 0];

>> y = hit_taknol(x)
y =
    2.0000    -3.6000     3.0000    -0.6000    10.0000
```

## 6. Operator Perbandingan dan Logika

Seperti yang kita lihat pada subbab 6.5 Control Statement, kita harus bisa menuliskan kondisi dalam bahasa MATLAB untuk menciptakan percabangan program ataupun loop. Untuk keperluan ini kita mungkin harus membandingkan dua variabel (sama atau tidak, lebih besar atau lebih kecilkah?), mengevaluasi apakah suatu variabel memenuhi satu dari sejumlah syarat, dan sebagainya.

Untuk membandingkan dua variabel digunakan operator berikut ini:

**Tabel 12**

<	>	lebih kecil, lebih besar
<=	>=	lebih kecil atau sama dengan, lebih besar atau sama dengan
==	~=	sama dengan, tidak sama dengan

Sementara untuk mengevaluasi logika, digunakan fungsi dan operator:

**Tabel 13**

<b>and(A,B)</b> atau <b>A &amp; B</b>	operasi logika AND antara <b>A</b> dan <b>B</b>
<b>or(A,B)</b> atau <b>A   B</b>	operasi logika OR
<b>xor(A,B)</b>	operasi logika XOR
<b>not(A)</b> atau <b>~A</b>	operasi logika NOT pada <b>A</b>

**A** dan **B** di sini bisa berupa skalar, vektor, maupun matriks, asalkan ukuran **A** dan **B** sama.

Adapun tabel kebenaran yang digunakan pada setiap operasi logika tersebut ialah sebagai berikut:

Tabel 14

A	B	A & B	A   B	xor(A,B)	~A
nol	nol	0	0	0	1
nol	bukan nol	0	1	1	1
bukan nol	nol	0	1	1	0
bukan nol	bukan nol	1	1	0	0

Perlu diperhatikan bahwa operasi logika memiliki prioritas untuk dihitung lebih dahulu, kemudian diikuti operasi aritmatika, lalu operasi perbandingan.

Untuk menambah pemahaman, mari kita praktekkan contoh di bawah ini di *command window*:

```
>> A = [1 2 0 -1 -2]; B = [1 0 0 0 5];
```

```
>> C = and(A,B)
```

```
C =
```

```
1 0 0 0 1
```

```
>> D=A|B|C
```

```
D =
```

```
1 1 0 1 1
```

```
>> E = xor(~A,B)
```

```
E =
```

```
1 0 1 0 1
```

Sekarang, mari kita mencoba membuat fungsi untuk menentukan suatu tahun termasuk kabisat atau tidak. Jangkauan tahun yang bisa dihitung ialah 1900 hingga 2500. Kita ketahui bahwa tahun kabisat terjadi pada tahun-tahun berkelipatan 4, kecuali tahun akhir abad; namun untuk tahun akhir abad berkelipatan 400 termasuk kabisat pula.

```

% Fungsi untuk mengetahui tahun kabisat atau tidak
% iskabisat.m

function hasil = iskabisat(thn)

% thn : merupakan masukan bilangan bulat positif
% hasil = 1 jika kabisat, 0 jika tidak

if thn<1900 | thn>2500
    disp('Tahun yang valid: 1900 - 2500');
    hasil=[];
    return
end

if rem(thn,4)==0 & (rem(thn,100)~= 0|rem(thn,400)==0)
    hasil=1;
else
    hasil=0;
end

```

Pada fungsi tersebut, terdapat dua control statement “if”:

- \* `if thn<19 00 | thn>2500`  
 Berarti jika variabel **thn** kurang dari 1900 ATAU lebih dari 2500, *command* di dalam “if” tersebut akan dijalankan.
- \* `if rem(th n,4)==0 & ...`  
`(rem(thn, 100)~=0|rem(thn,400)==0)`  
 Berarti jika variabel **thn** habis dibagi 4 DAN logika `(rem(thn, 100)~=0|rem(thn,400)==0)` bernilai 1 (true), maka *command* setelah “if” akan dijalankan.



Perlu diperhatikan bahwa logika  $(\text{rem}(\text{thn}, 100) \sim= 0 \mid \text{rem}(\text{thn}, 400) == 0)$  akan bernilai 1 bila **thn** bukan tahun abad (kelipatan 100); ataupun kalau tahun abad haruslah kelipatan 400.

Sekarang kita bisa coba:

```
>> iskabisat(2005), iskabisat(1972 )
ans =
     0
ans =
     1
```

Fungsi ini hanya bisa mengolah masukan skalar. Lalu bagaimana kalau diinginkan masukan berupa vektor atau matriks? Kita bisa ubah fungsinya menjadi berikut ini:

```
% Fungsi untuk mengetahui tahun ka bisat atau tidak
% iskabisat.m

function hasil = iskabisat(thn)

% thn : merupakan masukan bilangan bulat positif
% hasil = 1 jika kabisat, 0 jika t idak

if sum(sum(thn<1900 | thn>2500))~= 0
    disp('Tahun yang valid: 1900 - 2500');
    hasil=[];
    return
end

hasil = rem(thn,4)==0 & ...
(rem(thn,100)~=0|rem(thn,400)==0);
```

Sekarang kita bisa coba untuk menentukan tahun kabisat antara 1980 hingga 1990.

```
>> iskabisat(1980:1990)
```

```
ans =
```

```
1 0 0 0 1 0 0 0 1 0 0
```



Jurusan Matematika  
Fakultas Sains Dan Teknologi  
Universitas Islam Negeri Maulana Malik Ibrahim Malang