Basic MATLAB

Running MATLAB in Athena

```
athena% add matlab
athena% matlab
```

Creating matrices and vectors

```
>> rowvector=[10 20]
rowvector =
   10    20

>> rowvector=[10; 20]
columvector =
   10
   20

>> matrix=[10 20; 30 40]
matrix =
   10    20
   30    40

>> zeromatrix=zeros(3)
zeromatrix =
   0    0    0
   0    0    0
   0    0    0

>> zerorowvector=zeros(1,3)
zerorowvector =
   0    0    0
```

Some operation of matrices/vectors. The “dot” modifier

```
>> rowvector + [1 2]
an =
   11    22

>> rowvector * 2
ans =
   20    40

>> matrix ^ 2
ans =
   700   1000
   1500  2200

>> matrix .^ 2
ans =
   100    400
   900   1600
```

For operations that are naturally defined on matrices the dot (.) makes the operation to apply to every matrix element, overriding the default behavior.
Note that rowvector ^ 2 doesn’t make sense!

ans =
  100   400

The inverse of every element of a matrix.

ans =
  0.1000   0.0500
  0.0333   0.0250

The inverse of the matrix.

ans =
 -0.2000   0.1000
  0.1500  -0.0500

Other way of obtaining the same result.

ans =
 -0.2000   0.1000
  0.1500  -0.0500

Transposing a row vector.

ans =
  10
  20
  30

Transposing a matrix.

ans =
  10   30
  20   40

Summing the components of a vector.

ans =
   60

What would sum(matrix) give us?.... Check the help!

>> help sum

SUM Sum of elements.
   For vectors, SUM(X) is the sum of the elements of X. For
   matrices, SUM(X) is a row vector with the sum over each
   column. For N-D arrays, SUM(X) operates along the first
   non-singleton dimension.

   SUM(X,DIM) sums along the dimension DIM.

   Example: If X = [0 1 2
                   3 4 5]

   then sum(X,1) is [3 5 7] and sum(X,2) is [ 3
                                             12];

   See also PROD, CUMSUM, DIFF.

Overloaded methods
   help sym/sum.m
>> sum(matrix)
ans =
   40   60

Remember to check the help system often! It is really easy! If you know the command that you want to obtain some info about it is as easy as typing `help command` where command is the command that you are interested in.

**Exercise**: Write a line that would compute the norm of a row vector.

**Checking dimensions**

>> size(matrix)
ans =
   2    2

>> size(rowvector)
ans =
   1    3

>> length(vector)
ans =
   3

**Accessing different parts of a matrix. The “colon” modifier**

>> matrix(1,1)
ans =
   1

>> matrix(2,:)
ans =
   30   40

>> matrix(:,1)
ans =
   10
   30

>> matrix(:,:)
ans =
   10   20
   30   40

**Functions**

If we create a file named `f.m` in the current working directory with this code...

```matlab
%f.m
function y=f(x,a)
y=a*x^2;
```

Obs: Lines that start with `%` are just comments; the semicolon instructs MATLAB *not* to output the result of this evaluation.
Then, from the command window we can just evaluate the function f...

\[
\text{>> } f(3,4) \\
\text{ans } = 36
\]

Just remember to name your .m file with the name of the function you are creating. Note that the way we have written our function, it can also be applied to matrices (but not to vectors… why?).

\[
\text{>> } f(\text{matrix},4) \\
\text{ans } = \\
\begin{array}{cc}
2800 & 4000 \\
6000 & 8800
\end{array}
\]

**Symbolic math**

\[
\text{>> } a=g^2 \\
\text{??? Undefined function or variable 'g'.}
\]

\[
\text{>> clear} \\
\text{>> syms g h} \\
\text{>> a=g*h} \\
a = g*h \\
\text{>> a^2} \\
\text{ans } = \\
g^2*h^2
\]

\[
\text{EDU» diff(a^2,g)} \\
\text{ans } = \\
2*g*h^2
\]

\[
\text{EDU» g=1; h=2; a} \\
a = g*h \\
\text{EDU» eval(a)} \\
\text{ans } = \\
2
\]

**Plotting**

\[
\text{>> t=0:0.1:10;} \\
\text{We create a vector } t \text{ that consists of all the values between 0 and 10 with a 0.1 spacing.} \\
\text{>> y=sin(t);} \\
\text{We compute the sin of all the points in } t. \\
\text{>> plot(t,y);} \\
\text{And now we just plot one vector vs the other!}
\]

After executing the last command we will get a new window with this plot.
Here is a more complicated plot, note that if you want to plot two curves in one figure you have to provide x and y vectors for the two functions that you want to plot.

\[
\text{>> plot(t, y, t, sqrt(1-y.^2));}
\]

**ODE Solver**

EDU» help ode23

ODE23 Solve non-stiff differential equations, low order method.

\[ [T,Y] = \text{ODE23('F',TSPAN,Y0) with TSPAN = [T0 TFINAL] integrates the system of differential equations } y' = F(t,y) \text{ from time T0 to TFINAL with initial conditions Y0. 'F' is a string containing the name of an ODE file. Function F(T,Y) must return a column vector. Each row in solution array Y corresponds to a time returned in column vector T.} \]

Let's first create a function that computes the right handside of the ODE \[ y' = F(t,y). \]

%f.m

\[
\text{function } dy=f(t,y) \\
\text{dy}=t.^2;
\]

\[
\text{>> ode23('f', [0 1], 3)}
\]

And we automatically get a plot of the solution.

If we want to keep the output of the simulation for later processing (plotting for instance)...

\[
\text{>> [t,y]=ode23('f', [0 1], 3);} \\
\text{>> plot(t,-y)};
\]

It is quite easy to solve multi-dimensional systems as well. But ode23 wants us to express F as row vectors

%f.m

\[
\text{function } dy=f(t,y) \\
\text{dy}=[t.^2; -y(2)];
\]

\[
\text{>> ode23('f', [0 1], [3 3])}
\]
Matlab code 1

% filename: mm.m

kl=1e3; % units 1/(Ms)
k_1=1; % units 1/s
k2=0.05; % units 1/s
E0=0.5e-3; % units M
options=[];

[t y]=ode23('mmfunc',[0 100],[1e-3 0 0],options,kl,k_1,k2,E0);

S=y(:,1);
ES=y(:,2);
E=E0-ES;
P=y(:,3);
plot(t,S,'r',t,E,'b',t,ES,'g',t,P,'c');

% filename: mmfunc.m

function dydt = f(t,y,flag,kl,k_1,k2,E0)
% [S] = y(1), [ES] = y(2), [P] = y(3)
dydt = [-kl*E0*y(1)+(kl*y(1)+k_1)*y(2);
k1*E0*y(1)-(k1*y(1)+k_1+k2)*y(2);
k2*y(2)];

Matlab code 2

% filename: hasty.m

alpha=50;
gamma=20;
sigma1=1;
sigma2=5;
options=[];

[t1 y1]=ode23('hastyfunc',[0 0],[0],options,alpha,gamma,sigma1,sigma2);
[t2 y2]=ode23('hastyfunc',[0 0],[1],options,alpha,gamma,sigma1,sigma2);
plot(t1,y1(:,1),'b',t2,y2(:,1),'r');

% filename: hastyfunc.m

function dydt = f(t,y,flag,alpha,gamma,sigma1,sigma2)
% [x] = y(1)
dydt = [alpha*y(1)^2/(1+(1+sigma1)*y(1)^2+sigma2*y(1)^4)-gamma*y(1)+1];