Keith Schneider, 20 July 2006

Matlab and Psychophysics Toolbox Seminar

Part 1. Introduction to Matlab

Variables

Scalars

```
>> a = 1

a =

     1
```

Row vector

```
>> a = [1 2 3 4 5 6]

a =

     1     2     3     4     5     6

>> a = [1,2,3,4,5,6]

a =

     1     2     3     4     5     6
```

Column vector

```
>> a = [1;2;3;4;5;6]

a =

     1
     2
     3
     4
     5
     6

>> a'

ans =

     1     2     3     4     5     6
```

Matrix

```
>> a = [1 2 3 4; 5 6 7 8; 9 10 11 12]

a =

     1     2     3     4
     5     6     7     8
     9    10    11    12
```

Scalars and vectors are just special cases of matricies.

Use the function `size` to determine the size of a variable. (For help on any function, just type "help" and the function name at the Matlab prompt.)

```
>> size(a)

ans =

     3     4

>> [nrows,ncols] = size(a)

nrows =

     3


ncols =

     4
```

Variables can also have three and higher dimensions. A scalar is a 1 x 1 matrix.

Other size-related functions: `ndims` and `length`.

Initialize a variable using `ones` or `zeros` functions.

```
>> a = zeros(2,3)

a =

     0     0     0
     0     0     0
```

Special characters

For more information, type `help ops`.

The percent sign (`%`) is used for comments. Any following text is ignored. When writing scripts and functions files, it is a good idea to comment your code liberally so that you or someone else can later figure out what you were doing (or trying to do).

```
% now we add a and b
c = a + b;
% the above is not an example of a useful comment
```

Semicolon (`;`) at end of expression suppresses printing:

```
>> [nrows,ncols] = size(a);
```

Colon (`:`)

Used as "to" (and "step").

```
>> b = 1:5

b =

     1     2     3     4     5
```

or

```
>> b=1:.5:4

b =

    1.0000    1.5000    2.0000    2.5000    3.0000
3.5000    4.0000

>> b=5:1

b =

   Empty matrix: 1-by-0

>> b = 5:-1:1

b =

     5     4     3     2     1
```

Indexing (subscripts)

Parentheses are used to indicate subscripts, which need to be positive integers, but can be matricies of any size.

```
>> b(2)

ans =

     4

>> b(2:4)

ans =

     4     3     2
```

Same as

```
>> b([2 3 4])

ans =

     4     3     2
```

or

```
>> c=2:4

c =

     2     3     4

>> b(c)

ans =

     4     3     2

>> b([1 3 4])

ans =

     5     3     2
```

What about two-dimensional matricies?

```
>> a = [1 3 5 7; 2 4 6 8; 1 2 3 4]

a =

     1     3     5     7
     2     4     6     8
     1     2     3     4

>> a(6)

ans =

     2
```

The colon used alone means “everything”:

```
>> a(:)'

ans =

     1     2     1     3     4     2     5     6     3
7     8     4

>> row=1; col=2; a(row,col)

ans =

     3

>> a(2,:)

ans =

     2     4     6     8

>> a(:,3)

ans =

     5
     6
     3

>> a([1 3],:)

ans =

     1     3     5     7
```

```
     1     2     3     4

>> a([1 3],[2 4])

ans =

     3     7
     2     4

>> a([1 3],[2 4]) = 0

a =

     1     0     5     0
     2     4     6     8
     1     0     3     0

>> a([1 3],[2 4]) = [1 2 3 4]
???  In an assignment  A(matrix,matrix) = B, the number of
rows in B and the number of elements in the A row index
matrix must be the same.

>> a([1 3],[2 4]) = [1 2; 3 4]

a =

     1     1     5     2
     2     4     6     8
     1     3     3     4
```

Try `flipud(a)`, `fliplr(a)`, `rot90(a)`

also `det inv \`

Math

Type `help arith`.

Operators: `+ - * / ^ .* ./ .^`

Scalar math

```
>> 2*3

ans =

     6

>> 4^1.5

ans =

     8

>> 2*3+1

ans =

     7

>> 2*(3+1)

ans =

     8
```

Vector and scalar math

```
>> a=[1 2 3];
>> 2*a

ans =

     2     4     6

>> a+5

ans =

     6     7     8

>> a^2
```

```
??? Error using ==> ^
Matrix must be square.

>> a.^2

ans =

     1     4     9

>> b = [4 5 6];
>> a*b
??? Error using ==> *
Inner matrix dimensions must agree.

>> a.*b

ans =

     4    10    18

>> c = a'*b

c =

     4     5     6
     8    10    12
    12    15    18

>> repmat(b,[1 3])

ans =

     4     5     6     4     5     6     4     5     6

>> repmat(b,[3 1])

ans =

     4     5     6
     4     5     6
     4     5     6
```

Convert Celcius to Fahrenheit:

```
>> c = 0:10:100

c =
```

```
     0    10    20    30    40    50    60    70    80
90   100

>> f = 9/5*c+32

f =

    32    50    68    86   104   122   140   158   176
194   212

>> plot(c,f,'ro-')

>> help plot

>> plot(f)
```

Also try `semilogx`, `semilogy`, `loglog`

```
figure, hold on

>> figure(2)
>> plot(1:10,'r')
>> hold on
>> plot(2:2:20,'b')

subplot
```

Character strings

```
>> a='hello'

a =

hello

>> size(a)

ans =

     1     5

>> figure(1)
>> xlabel('C^\circ')
>> ylabel('F^\circ')
>> title('Celcius vs. Fahrenheit')

>> a = ['hello';'there']

a =

hello
there

>> a = ['hello';'there';'who']
??? Error using ==> vertcat
All rows in the bracketed expression must have the same
number of columns.

>> a = ['hello';'there';'who  ']

a =

hello
there
who

>> strvcat('hello','there','who')

ans =

hello
there
who

>> a = {'hello','there','who'}
```

```
a =

    'hello'    'there'    'who'

>> a{1}

ans =

hello

>> a{1}(3)

ans =

l

>> a = {1, [1 2], [1 2; 3 4]}

a =

    [1]    [1x2 double]    [2x2 double]

>> a{2}

ans =

     1     2
```

`disp, sprintf, fprintf`

`sprintf` and `fprintf` use the formatted output conventions used in the C programming language (see the help files)

```
>> a = sprintf('Hello %s.  You have %d oranges at %4.2f
each.','Keith',5,pi)

a =

Hello Keith.  You have 5 oranges at 3.14 each.
```

Built-in functions

```
sin, cos, tan, asin, acos, atan, atan2, exp, log, log10,
sqrt

>> x = 0:.1:10*pi;
>> plot(x,sin(x))

round, fix, floor, ceil, sign, rem

>> plot(x,round(2*sin(x)))
>> plot(x,sign(sin(x)))
>> set(gca,'YLim',[-2 2])     % more about plots later

min, max, mean, median, std, sort
sum, prod, cumsum, cumprod
rand, randn, randperm

>> hist(rand(1,10000),100)  % uniform pseudo-random numbers
>> hist(randn(1,10000),100) % normal pseudo-random numbers
```

To get a random integer from 1 to n, use `floor(n*rand)+1`.

```
>> floor(5*rand(1,9))+1

ans =

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

To shuffle the arrangement of an array, use randperm.

```
>> a=1:2:18

a =

     1     3     5     7     9    11    13    15    17

>> b=randperm(9)

b =

     7     5     3     2     4     9     8     6     1

>> a(b)

ans =

    13     9     5     3     7    17    15    11     1
```

Complex numbers

Complex numbers are entered like this:

```
>> 1+1i

ans =

   1.0000 + 1.0000i
```

This is the same as

```
>> a = 1+sqrt(-1)

a =

   1.0    + 1.0000i
```

The variables `i` and `j` are automatically set up to equal `sqrt(-1)`. But, these variables are often used as indicies for loops, so it is best to use `1i` instead.

Another way to do this is with the `complex` function:

```
>> a=complex(1,-2)

a =

   1.0000 - 2.0000i
```

Functions for complex numbers: `angle, abs, conj, real, imag`.

Be careful with the transpose. `a'` is actually the complex conjugate transpose (which is the same as a normal transpose for real numbers).

```
>> a = 1i*[1; 2; 3];
>> a'

ans =

        0 - 1.0000i        0 - 2.0000i        0 - 3.0000i

>> a.'

ans =

        0 + 1.0000i        0 + 2.0000i        0 + 3.0000i
```

Logic

Most Matlab variables are numeric, but there is also a special `logical` type, which can have one of two values, `true` (non-zero) or `false` (zero).

```
>> d = [true false true true false]

d =

     1     0     1     1     0
```

is the same as

```
>> d = logical([1 0 1 1 0])

d =

     1     0     1     1     0
```

Actually, in Matlab, `true` and `false` are functions, parallel to the numeric functions ones and zeros.

```
>> true(2,3)

ans =

     1     1     1
     1     1     1
```

It is possible to use logical variables as a mask to perform "logical indexing" (see `help logical`).

```
>> b = 5:-1:1;
>> b(d)

ans =

     5     3     2
```

In this case, the logical variable that acts as a mask must be the same size as the variable it is indexing. The values indexed by `true` values are returned. This is particularly useful when using logical expressions.

Logical expressions

Relational operators

```
>> help relop

< <= > >= == ~=
```

When using an expression like `a <= b`, `b` must be a scalar or a matrix of the same size as `a`.

```
>> a=1:5

a =

     1     2     3     4     5

>> a>3

ans =

     0     0     0     1     1

>> b=2*ones(1,5)

b =

     2     2     2     2     2

>> a>b

ans =

     0     0     1     1     1
```

Logical operators

Negation: ~

```
>> a = [1 0 1]

a =

     1     0     1

>> ~a
```

```
ans =

     0     1     0
```

Logical comparisons: `&` | `xor`

```
>> a = [1 0 1];
>> b = [0 0 1];
>> a&b

ans =

     0     0     1

>> a|b

ans =

     1     0     1
```

Logical indexing is particularly useful using logical expressions.

```
>> a=1:10;
>> b=-5:4;
>> a(b>1)

ans =

     8     9    10

>> a(b>-1 & a<8)

ans =

     6     7
```

Useful logical functions: `any all find`

The value returned by `any(a)` is `true` if any of the elements of `a` are `true`, and `false` otherwise. `all(a)` is `true` only if all the elements of `a` are `true`.

Find(a) returns the indicies of the true elements of a.

```
>> a=[1 0 1 0 0 1];
>> find(a)

ans =
```

```
     1     3     6
```

This is useful in expressions:

```
>> a = 1:10;

>> b=find(a>5)

b =

     6     7     8     9    10

>> a(b(1:3))

ans =

     6     7     8
```

If you want to know how many elements of a logical matrix `a` are true, use `sum(a)` or `length(find(a))`.

There are lots of "is" identifying functions that return logical values: `isempty isnan isfinite isnumeric islogical isreal`

Flow control

if else elseif end

```
a = 1;
if a > 2
     b=1
else
     b=2
end
```

```
b =

     2
```

for end

Example #1

```
x=1:10;
for i=1:4
     subplot(2,2,i)
     plot(x,x.^i)
     title(sprintf('x^%d',i))
end
```

Example #2

```
a = zeros(1,10);    % always initialize variables
                    % before adding to them in a loop
b = [1 2 3 4 7 8 9 10];
for i=b
     a(i) = i^sqrt(i-1);
end
```

```
>> format short g
>> a
```

while end

switch case otherwise

User-defined functions and script files

In addition to the functions that Matlab provides, you can also write your own functions. Just open a new .m file from the Matlab menu, and make sure the definition of the function appears on the first line. For example:

```
function y=sem(x)
% calculate the standard error of the mean
if ~isempty(x) % you may choose to do some error-checking
               % of the input
     n=length(x);
     y = std(x)/sqrt(n); % make sure that the output
                         % variable(s) gets assigned
                         % somewhere
else
     y=[];
end
```

Save this file (make sure it is in your path), and then you can call this function just like any of the built-in Matlab functions.

```
a=sem(rand(1,100));
```

You can also write script files that perform a series of operations but do not return any values. These script files work exactly as if you had typed the operations into the command window.

Managing the environment

```
who, whos, clear, which

help format
```

Disk operations

```
dir, delete, cd, path

load, save
```

`load` can import ascii (text) data or binary Matlab variables.

More advanced disk operations

`fopen, fread, fwrite, fscanf`, etc. (like their C conventions)

For those of you who want to test your new Matlab skills, I offer the following optional assignment.

Assignment #1

1. Write a function `listprimes(x)` that returns a vector containing all the prime numbers less than or equal to `x` (2 is the smallest prime number, so if `x < 2`, the function should return the empty vector `[]`).

   Hint: Use the method of the Eratosthene's Sieve in which you create an array of numbers from 1 to `x`, and remove the multiples of primes in sequence, i.e., remove the multiples of 2, 3, 5, 7, etc. Those numbers left in the array will be prime. For example:

   ```
   2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 …
   ```

   Remove multiples of 2

   ```
   2 3 0 5 0 7 0 9 0 11 0 13 0 15 0 …
   ```

   Remove multiples of 3

   ```
   2 3 0 5 0 7 0 0 0 11 0 13 0 15 0 …
   ```

   Remove multiples of 5

   ```
   2 3 0 5 0 7 0 0 0 11 0 13 0 0 0 …
   ```

   and so on…

2. Write a function `nprimes(x)` that returns the number of primes less than or equal to `x`.
3. Write a function `probprime(x1,x2)` that returns the probability that a random integer in the range between `x1` and `x2` (inclusive) is prime. On the same graph, plot the the function `1/log(x)` and the probability that a number in the vicinity `x` (i.e. in the range `x` ± `d` for some appropriate choice of `d`) is prime.