**PROGRAMMING IN MATLAB**

Matlab provides a friendly interactive environment for scientific programming and visualization. While Matlab is not intended as a substitute for programming languages like Fortran and C, it is very helpful for developing and testing of models, as well as, obtaining immediate feedback on directions for solving difficult problems.

In this section we will introduce many useful constructs used over and over again in Matlab programming.

* The program statement *x=input('{some text defining x}')* allows you to interactively input the value of *x* during program execution. For example, in the last worksheet you built vectors of given length with given properties. For programming purposes it is useful to allow someone to input any desired length of a vector at the time of program execution.

n=input(' input an integer n = ');

Then issue the command

v=(1:2:n).^2

* In matlab programming the are several types of loops and conditionals.
  + The first is the *for loop*. The syntax for a *for loop* is
  + for {var} = {a vector of counter values}
  + {statements}

end

for example

for i=1:3

x(i)=i^2

end

produces *x=[1,4,9]*. Here is another example

for i=0:4

for j=1:5

a(i+1,j)=1/(i+j); % note use of semicolon to suppress printing

%and that an index must be positive.

end

a

end

To do these same operations more efficiently in matlab we can use the following: *aa=hilb (5)*

* + The next conditional is the *if statement*. The syntax for an *if statement* has several forms. The simplest is

if {relation} {statements} end

More generally, you can write

if {relation}

{statements}

elseif {relation}

{statements}

else

{statements}

end

* + Here is an example of nested *for loops* and *if statements* First, we use an input statement to select a value of *n*
  + clear % clear the workspace by deleting all variables
  + n=input(' input an integer n = ');

Now we create an *n* by *n* matrix *A*

for i=1:n

for j=1:n

if i< j

A(i,j)=-1;

elseif i> j

A(i,j)=0;

else

A(i,j)=1;

end

end

end

A

As an illustration of Matlab programmin this matrix can also be built as follows:

AA=eye(n)-triu(ones(n),1) %see help on tril and triu

* + Next we have the conditional loop *while*
  + while {relation}
  + {statements}
  + end

Here is an example

j=1

while j <= 10

k(j)=cos(j\*pi);

j=j+1;

end

% to view the matrix type the name of the matrix

k

% compare this with

kk=cos((1:10)\*pi)

* In the *if* and *while* constructs above we have introduced the notation of a *relation*. A *relation* has the general form

{matrix expression} {relational operation} {matrix expression}

* + The relational operations are
  + == equals
  + ~= not equal
  + < less than
  + <= less than or equal
  + > greater than
  + >= greater than or equal
  + The value of a relation is a matrix containing zeros and ones. An entry is *0* if the matrix entry relation is not true and *1* if it is true.
  + For example, if
  + A=[1 2;3 4]; B=[1 2;2 4]

T=A==B

returns the matrix

T=[1 1;0 1]

* + Another useful set of devices used in relational expressions are
  + & % and,
  + | % or

~ % not

These are used on matrices obtained from a relational matrix (as above with all zeros and ones). For example,

T1=[1 1;0 1]; T2=[1 0;0 0]

T=T1&T2

gives the matrix

T1=[1 0; 0 0]

and

T=T1|T2

gives the matrix

T=[1 1;0 1]

Finally,

T=~T1

gives the matrix

T=[0 0;1 0]

* Two useful programming commands that yield matrices consisting of zeros and ones are *any* and *all*. To see help on these in Matlab, type

help any

ANY True if any element of a vector is true.

For vectors, ANY(V) returns 1 if any of the elements of the

vector are non-zero. Otherwise it returns 0. For matrices,

ANY(X) operates on the columns of X, returning a row vector

of 1's and 0's.

Here is a program the does the same as the *any* command (see below), i.e., *w=any(v)* where *v* returns 1 if any of the elements of the vector *v* are non-zero. Otherwise it returns 0.

v(5:2:10)=5:2:10;

w=0;

k=1;

while (k<=length(v)&w==0)

if v(k)~=0

w=1

end

k=k+1;

end

This should be compared with

any(v)

As an example of how you might use the *any* command, let

M=floor(11\*rand(3,4))-1

any(M>=8)

Now consider the *all* command

help all

ALL True if all elements of a vector are true.

For vectors, ALL(V) returns 1 if all of the elements of the

vector are non-zero. Otherwise it returns 0. For matrices,

ALL(X) operates on the columns of X, returning a row vector

of 1's and 0's.

Here is an example

M=floor(11\*rand(3,4))-1

all(M<=8)

* The *find* is also very useful in programming.
* x=floor(10\*rand(1,20))
* I=find(x==3)
* J=find(x<5)
* Programming in Matlab: In Matlab there are 2 types of files: script-files and function files: typically a script-file is just a sequence of commands (a program) while a function file builds a function. script-files can call function files and execute them.
* Lets do an example. You will need to enter the editor!!!! After typing the file (or copying the statements below from netscape and pasting it) into the editor save the file as *test1.m* and go back to the Matlab window. Then type the name of the file, in this case *test1* and hit return. You will be prompted for a mesh size *h*. Enter a value, try several values like *.25*, *.1*, *.05*, *.01* and *.001*. After entering a value for *h* hit *return* or *enter*.
* % this is a test m-file called test1.m
* h=input('mesh size h = '); % inputs a mesh size
* x=(0:h:1); % computes a grid of x values
* lx=length(x); % determines the number of elements in the grid
* y=x.^2; % evaluates the function at the grid points
* int=(h/2)\*( y(1) + 2\*sum(y(2:(lx-1))) + y(lx) ) % trap rule for integral with exact value 1/3

Here is a more complicated example of numerical differentiation and plotting

% this is a test m-file called test2.m

h=input('mesh size h = '); % inputs a mesh size

x=(0:h:2\*pi); % computes a grid of x values

lx=length(x); % determines the number of elements in the grid

y=sqrt(x).\*sin(x); % evaluates the function at the grid points

dy=diff(y)./diff(x); % numerical derivative of the function

plot(x(2:lx),y(2:lx),x(2:lx),dy)

text(x(lx-1),dy(lx-1)-.5,' f''(x)');

text(x(lx),y(lx)-.5,'f(x)')

grid

pause

close

* + The syntax *diff(X,n)* is the *n*-th difference function, so *dy* above is a numerical approximate derivative. Note that in the plot command *dy* has one fewer index since for a vector *z* with *n* components, we have

diff(z)=z(2:n)-z(1:(n-1))

So

diff(y)./diff(x)

has as its entries *(y(x(i+1))-y(x(i)))/(x(i+1)-x(i))* which approximates the derivative of *y* at *x(i+1)*.

* Next we give an example of a program that uses Fourier series to approximate a function. We have built the function in a function file with two choices for the function.This program demonstrates how a Fourier sine series can be used to approximate a function on the interval *(0,1)*. First we will build a file containing the desired function to approximate. Go to the editor and write a file named *fn.m* containing the lines:
* % this is a function file that allows you to choose between two functions
* function y=fn(x)
* global prob
* if prob==1
* y=(1-(1-2\*x).^2);
* elseif prob==2
* y=(1-2\*x).^2;
* end

Save the file (as *fn.m*) and now build an m-file to carry out the approximation. The following should be typed into an m-file and saved as *fourier.m* .

clear

global prob

prob=input(' pick the problem number for fn.m (1 or 2) = ')

N=input('number of terms in Fourier series, N = ')

h=.005;

t=0:h:1;

for n=1:N

a(n)= 2\*h\*fn(t)\*sin(n\*pi\*t)'; % trap rule for the integrals

end

delta=.05;

x=0:delta:1;

p=sin((1:N)'\*x\*pi);

yapp=a\*p;

y=fn(x);

err=norm(y-yapp)\*sqrt(delta) % gives the L2 norm of difference

plot(x,y,x,yapp)

Note that the sine series does not approximate very well a function that is not zero at the ends. Try *N=10, 50, 100, 200*.

* Since Matlab keeps track of all variables in the workspace, it is usually a good idea to begin a Matlab program with statements that delete all the current variables so there won't be any confusion. *clear* and *clear global* delete the variables and global variables in the workspace. Note that Matlab allows the use of global variables. Their use should be kept to a minimum and always give speical names to such variables since they effect all parts of Matlab and will cause considerable trouble if not used properly.

**EXERCISES**

1. Write a short Matlab program to input an integer *n* and build a *n* by *n* matrix with the numbers *1,2,... ,n* on the main diagonal and zeros everywhere else (hint: Look at the command *diag* .
2. Write a short Matlab program to input an integer *n* and build the *n* by *n* matrix *A* with entries *a\_{ij}=3^{i j}*.
3. Write a short Matlab program to compute the first *100* Fibonnaci numbers: *a\_1=1, a\_2=1, a\_n=a\_{n-1}+a\_{n-2}*
4. Write a Matlab m-file to Input an integer *n*, a number *w* and a vector with *n* components, *x*.
5. n=input('n= ');
6. w=input('w = ');
7. x=input(' an n-vector, x = ')

Step two: write a *for loop* program to build a matrix *A* with the given entries. Then, redo the problem the easy way using matlab syntax:

* 1. *a\_{ij}=w^((i-1)(j-1)) Note this not matlab syntax*
  2. *b\_{ij}=1/(i+j-1) Note this not matlab syntax*
  3. *c\_{ij}=x\_i^(j-1) Note this not matlab syntax*

1. Write a program to compute the binomial coefficients *C(n,r)*. Do this by building a matrix *P* which is of size *(n+1)* by *(n+1)* for a given n so that *p\_{ij}* satisfies
2. p\_{i,1}=p\_{1,j}=1,
3. for i+j <= (n+2), p\_{i,j}=p\_{i,(j-1)}+p\_{(i-1),j},
4. for i+j > (n+1), p\_{i,j}=0

You can use loops but it is much easier to use the built-in commands *diag*, *pascal* and *rot90*.

% pascal(k) builds the pascal matrix of size k by k

% rot90(A) rotates the matrix A by 90 degrees

% diag(A) builds a column vector from the diagonal of A

1. Write a program to carry out Euclids algorithm for computing the greatest common divisor of two numbers *a* and *b*. The algorithm is:
   1. input two numbers a and b
   2. compute the remainder of a/b
   3. replace a by b
   4. replace b by the remainder computed in 2)
   5. repeat the second through fourth steps until b is zero
   6. the gcd is the final value of a
2. Redo the Lottery problem we did using Maple.

[RETURN TO MAIN DOC](http://texas.math.ttu.edu/%7Egilliam/mlhtml_1/m4330_ml_1.html)