

CHAPTER 1

Computation with *MATLAB*

1.1. Data in *MATLAB*

The basics: Matrices and vectors

Setting vectors and matrices with internal *MATLAB* commands

Simple functions and plots

Reading data in *MATLAB*

1.2. Mathematical operations with matrices

Elementary operations: Addition, subtraction, and so forth

Matrix multiplication

Multiplication on an element-by-element basis

1.3. Symbolic math toolbox

1.4. Programming in *MATLAB*

Script files

Function files

1.5. Summary

1.6. Problems

1.7. References

1. *Computation with MATLAB*

This set of lecture notes describes numerical methods for solving a variety of problems in the hydrological sciences. The computational engine used for the work is *MATLAB*, a commercially available software package. This first chapter introduces the basics of working with *MATLAB*.

1.1. Data in *MATLAB*

The basics: Matrices and vectors

MATLAB operates on *matrices*, which are arrays of numbers. That is, a matrix is a "table" of numbers with, say, N rows and M columns. For example, if N=3 and M=2, a matrix A has dimension 3x2. Assignment of such a matrix in *MATLAB* can be accomplished as follows. Type in a "[" to let *MATLAB* know that you are going to input a matrix. Type the numbers for the first row of the matrix (say "1" and "2") and then type a ";" to let *MATLAB* know that you have reached the end of a row. Type the next two rows in the same way and end with a "]".

```
A=[1 2;3 4;5 6]
```

After typing this in, *MATLAB* echoes the contents of the matrix, A.

```
A =
     1     2
     3     4
     5     6
```

A *vector* of numbers is simply a matrix with one of the dimensions equal to one. For example, you might have data on air temperatures taken at noon every day over a week. These can be represented as a vector in *MATLAB*.

```
T=[12.1 13.6 9.5 8.2 10.4 11.7 11.9]
```

Typing this line into *MATLAB* results in:

```
T =
    12.1000    13.6000     9.5000     8.2000    10.4000    11.7000    11.9000
```

As defined, T is a "row vector". It can be converted to a column vector by taking its *transpose*, an action that is accomplished in *MATLAB* by placing a single quote after the matrix or vector. That is, to define the transpose of T, we type

```
T'
ans =
    12.1000
    13.6000
     9.5000
     8.2000
    10.4000
    11.7000
    11.9000
```

11.9000

Setting vectors and matrices with internal MATLAB commands

Special vectors and matrices can be constructed with built-in *MATLAB* statements. For example, it often is useful to construct a vector that covers a certain range and has evenly-spaced entries. This would be the case where we want to examine a sequence of regularly spaced data where we did not explicitly have the independent variable recorded. If, for example, we had daily temperatures recorded for a year and wanted to plot the data versus a time variable, we can form such a variable as follows.

```
t=1:1:365;
```

Note that we put a semicolon at the end of this assignment statement. This tells *MATLAB* not to echo the 365-element vector of times.

Other *MATLAB* functions that are useful in setting vectors and matrices are `zeros` (sets a matrix with all zero elements), `ones` (sets a matrix with elements equal to one), and `rand` (sets a matrix with elements chosen using a pseudo-random-number generator for a uniform distribution on [0,1]). Try typing `a=rand(3,4)` and `b=ones(1,10)` to get the feel of these utilities. [The command `randn` generates numbers from a normal distribution.]

Simple functions and plots

MATLAB also has many built-in functions. Thus, we can form a sine function for a seasonal variable using the following *MATLAB* statement.

```
t=1:5:365;
temp=15*sin(2*pi*t/365)+12;
```



We can look at the result by using *MATLAB* graphics capabilities (Figure 1.1).

```
plot(t,temp,'+') ;
xlabel('Time, days');
ylabel('Smoothed temperature, degrees C')
```

The series of commands above actually produce a figure with default characteristics on line widths, font size, and so forth. Figure 1.1 and other figures in these lecture notes are modified to make them more easily readable. *MATLAB* commands allow specification of a variety of aspects of figures [Box 1.1].

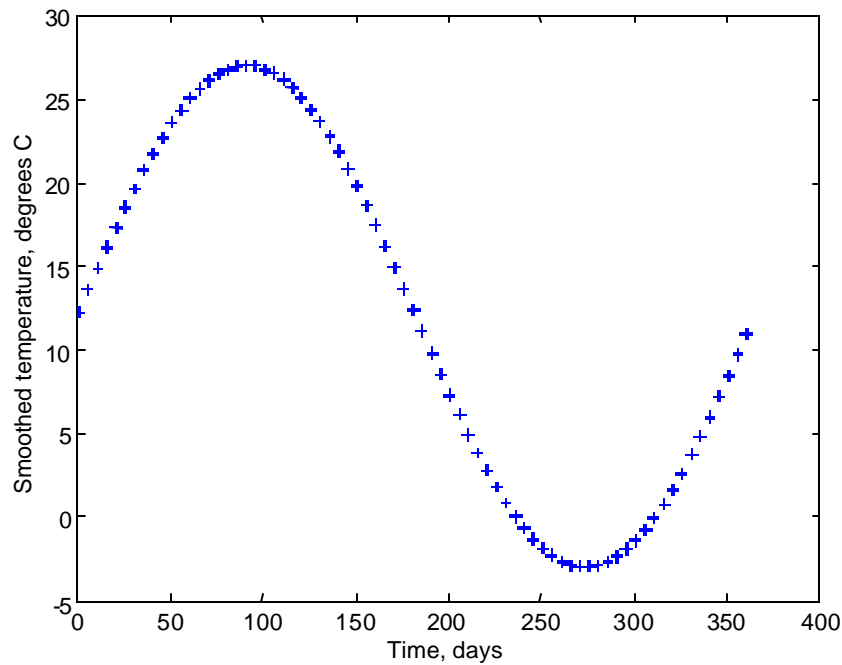


Figure 1.1. Example of a simple graph produced in *MATLAB*.

One of the most important things to note about *MATLAB* as you learn to use it is the extensive on-line HELP facility. Typing `help` by itself gives a series of choices from which you can refine your search. Typing `help item` gives help on the item chosen. For example to get help with the *MATLAB* plot utility¹,

```
help plot
```

```
PLOT Plot vectors or matrices.
```

```
PLOT(X,Y) plots vector X versus vector Y. If X or Y is a matrix,
then the vector is plotted versus the rows or columns of the matrix,
whichever line up.
```

```
PLOT(Y) plots the columns of Y versus their index.
```

```
If Y is complex, PLOT(Y) is equivalent to PLOT(real(Y),imag(Y)).
```

```
In all other uses of PLOT, the imaginary part is ignored.
```

```
Various line types, plot symbols and colors may be obtained with
PLOT(X,Y,S) where S is a 1, 2 or 3 character string made from
the following characters:
```

y	yellow	.	point
m	magenta	o	circle
c	cyan	x	x-mark
r	red	+	plus
g	green	-	solid
b	blue	*	star

¹ Reprinted with permission from The Mathworks, Inc.

w	white	:	dotted
k	black	-.	dashdot
		--	dashed

For example, `PLOT(X,Y,'c+')` plots a cyan plus at each data point.

`PLOT(X1,Y1,S1,X2,Y2,S2,X3,Y3,S3,...)` combines the plots defined by the (X,Y,S) triples, where the X's and Y's are vectors or matrices and the S's are strings.

For example, `PLOT(X,Y,'y-',X,Y,'go')` plots the data twice, with a solid yellow line interpolating green circles at the data points.

The `PLOT` command, if no color is specified, makes automatic use of the colors specified by the axes `ColorOrder` property. The default `ColorOrder` is listed in the table above for color systems where the default is yellow for one line, and for multiple lines, to cycle through the first six colors in the table. For monochrome systems, `PLOT` cycles over the axes `LineStyleOrder` property.

`PLOT` returns a column vector of handles to `LINE` objects, one handle per line.

The X,Y pairs, or X,Y,S triples, can be followed by parameter/value pairs to specify additional properties of the lines.

See also `SEMILOGX`, `SEMILOGY`, `LOGLOG`, `GRID`, `CLF`, `CLC`, `TITLE`, `XLABEL`, `YLABEL`, `AXIS`, `AXES`, `HOLD`, and `SUBPLOT`.

The `lookfor` command is often useful for finding what might be available on a given topic. For example, you might want to know what is available in *MATLAB* for interpolation².

```
lookfor interpolation
```

```
ICUBIC 1-D cubic Interpolation.
INTERP1 1-D interpolation (table lookup).
INTERP1Q Quick 1-D linear interpolation..
INTERP2 2-D interpolation (table lookup).
INTERP3 3-D interpolation (table lookup).
INTERP4 2-D bilinear data interpolation.
INTERP5 2-D bicubic data interpolation.
INTERP6 2-D Nearest neighbor interpolation.
INTERPFT 1-D interpolation using FFT method.
INTERPN N-D interpolation (table lookup).
SPLINE Cubic spline interpolation.
NTRP113 Interpolation helper function for ODE113.
NTRP15S Interpolation helper function for ODE15S.
NTRP23 Interpolation helper function for ODE23.
NTRP23S Interpolation helper function for ODE23S.
NTRP45 Interpolation helper function for ODE45.
NDGRID Generation of arrays for N-D functions and interpolation.
```

² Reprinted with permission from The Mathworks, Inc.

PDEARCL Interpolation between parametric representation and arc length.

After finding the items available, you can use the `help` command to get more information (e.g., `help INTERP1`).

Reading data into MATLAB

How about getting data into *MATLAB* without typing it directly from the keyboard? Suppose you have data from a laboratory experiment in which water flows through a column of sand. A chemical tracer (for example sodium chloride) is introduced as a pulse into the inflow end of the column and the breakthrough curve, chloride concentration as a function of time at the outflow end of the column, is observed. The result is a data file listing number of pore volumes that have been eluted (dimensionless surrogate for time) and relative concentration (measured concentration divided by the concentration of the pulse input) of the effluent water. The data look like this



```
0.0700 0.02
0.1400 0.02
0.2100 0.02
0.2700 0.02
0.3400 0.02
0.4100 0.02
0.4800 0.02
0.5500 0.02
0.6200 0.02
0.6800 0.000
0.7500 0.0000
0.8200 0.0000
0.8900 0.02
0.9600 0.1
1.0300 0.44
1.0900 0.78
1.1600 0.66
1.2300 0.3
1.3000 0.14
1.3700 0.06
1.4400 0.02
1.5000 0.02
```

Suppose these data are in a file called "bt.dat". Use the `load` command to get the data into *MATLAB*.

```
load bt.dat
```

Note that the files to be loaded into *MATLAB* *must* have an extension unless the extension is "mat". For example, a file named "bt.dat" can be loaded by `load bt.dat`; similarly, "bt.q" or "bt.xxx" can be loaded by typing the full name. If the file is named "bt.mat", the command `load bt` will work. Now we can set "pv" (for number of pore volumes) equal to the first column of the file and "rc" (for relative concentration) equal to the second column of the file by typing

```
pv=bt(:,1);
```

Note that the colon notation is used to select entire columns of the matrix. That is `bt(:,1)` means "bt(all elements, column 1)". In the same way, we set "rc" (for relative concentration) equal to the second column in the data file "bt".

```
rc=bt(:,2);
```

To look at the breakthrough curve (Figure 1.2):

```
plot(pv,rc,'or',pv,rc,'b')
xlabel('Dimensionless time, pore volumes')
ylabel('Reduced concentration, c/c_0')
```

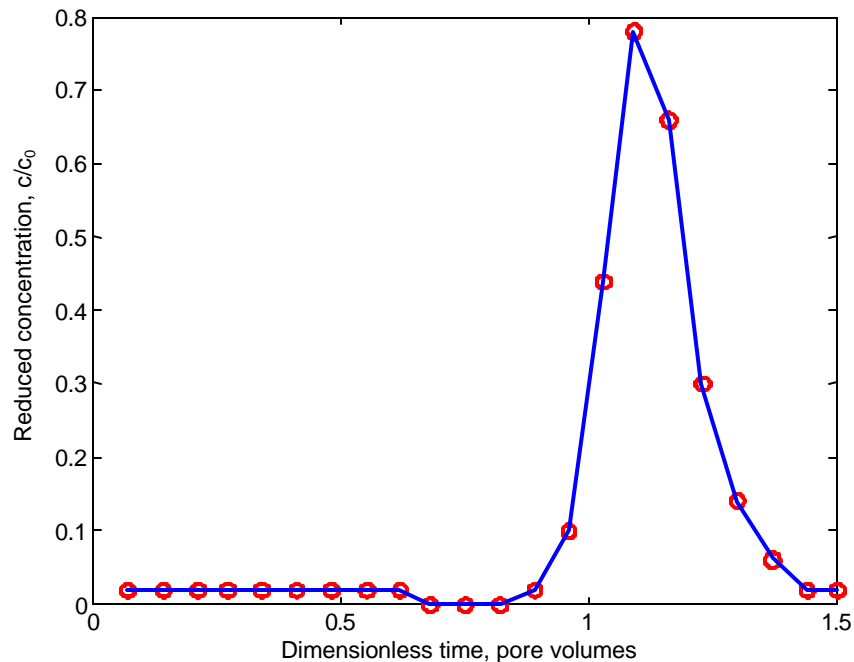


Figure 1.2. Plot of data from a laboratory column experiment.

You can load data into *MATLAB* as long as the data are in an array containing only numbers (e.g., no table headings) and containing no blank spaces (e.g., zeroes must be typed explicitly in the file, not left as blanks). (See the *MATLAB* functions `TBLREAD`, `TDFREAD`, and `XLSREAD` for other options to get data into the program.)

1.2. Mathematical operations with matrices

MATLAB is a powerful engine for data analysis and modelling primarily because of its design for performing matrix calculations. All standard mathematical operations are supported. You just have to remember that matrices on which you are operating must be "conformable" for that particular operation.

Elementary operations: Addition, subtraction, and so forth

Matrices (vectors as a special case) are conformable for addition (and subtraction) as long as they are the same size. Thus, if

```
A=[1 2 3; 4 5 6]
```

```
A =
     1     2     3
     4     5     6
```

and

```
B=[7 8 9; 10 11 12]
```

```
B =
     7     8     9
    10    11    12
```

then the sum is what you would expect:

```
A+B
```

```
ans =
     8    10    12
    14    16    18
```

as is the difference:

```
A-B
```

```
ans =
    -6    -6    -6
    -6    -6    -6
```

Multiplication by a scalar is also straightforward.

```
2*A
```

```
ans =
     2     4     6
     8    10    12
```

All of the functions available in *MATLAB* can be applied to matrices. These include sin, cos, exp, log, and others. For example,

```
sqrt(A)
```

```
ans =
    1.0000    1.4142    1.7321
    2.0000    2.2361    2.4495
```

Note that these functions operate on each element of the matrix.

Matrix multiplication

Two matrices, say C of dimension n by m and D of dimension k by j, are conformable for multiplication if m=k. Matrices A and B above are *not* conformable for multiplication because A is

2x3 and B is also 2x3. (That is, "m" is 3 and "k" is 2.) If you try to multiply the two, *MATLAB* gives an error message.

```
A*B
```

```
??? Error using ==> *
Inner matrix dimensions must agree.
```

Now the *transpose* of B is a 3x2 matrix. (In *MATLAB* the transpose is formed by placing a single quotation mark (') after the matrix.)

```
BT=B'
```

```
BT =
     7     10
     8     11
     9     12
```

By the rules of matrix multiplication, A and BT are conformable for multiplication.

```
A*BT
```

```
ans =
     50     68
    122    167
```

The case of multiplication of a matrix and a vector is of particular importance as systems of equations are represented in this form. Note that in this case, the vector length must equal the number of rows in the matrix. The form of a set of equations is $Ax=b$ where A is a matrix of (known) coefficients, x is the unknown vector, and b is a vector of known quantities. The system of equations

$$3x + 3y = 9$$

$$2x - 2y = -2$$

is represented in matrix notation as

$$\begin{bmatrix} 3 & 3 \\ 2 & -2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 9 \\ -2 \end{bmatrix}$$

Our main interest in such matrix equations is in solving them -- determining values for x and y that satisfy the equations. We will explore this topic in some detail in a later lecture. For now, just accept that the "backslash" operator in *MATLAB* does the trick.

```
A=[ 3 3; 2 -2];
b=[ 9 -2]';
xy=A\b
```

```
xy =
     1
     2
```

Multiplication on an element-by-element basis

Often we need to operate on a vector element by element. *MATLAB* uses a period at the end of a vector to indicate operation on an element-by-element basis. For example, suppose we want to take the vector of concentrations that we defined above and square each element. Note that the command `c2=c*c` will not work because the "*" implies matrix multiplication and the only time a matrix is conformable with itself for multiplication is when it is square. (Note that, even in the case of a square matrix, `A*A` is *not* equivalent to squaring each element! Try it. Define a square matrix by typing, say, `A=rand(3,3)` and examine `A` and `A*A`.) To have *MATLAB* square the elements of `c`, we type `c.*c` noting that the dot after the first `c` specifies calculation element by element. Alternatively, we can type `c.^2`, using *MATLAB*'s carat notation for exponentiation.

```
c2=c(12:18).*c(12:18)
```

```
c2 =
      0
  0.0004
  0.0100
  0.1936
  0.6084
  0.4356
  0.0900
```

Note that we looked at the result for only elements 12 to 18 of the vector by indicating the range 12 to 18 in parentheses after `c`. This is standard notation in *MATLAB*; `A(n1:n2,m1:m2)` indicates the submatrix with rows `n1` to `n2` and columns `m1` to `m2` of the original matrix.

Another example of the use of element-by element computation is finding the reciprocal of the elements of a vector. Take the vector `pv` from the breakthrough-curve data. Suppose we want to find $1/pv_i$ for $i=12$ to 18. As it turns out, we can't use `1/pv` because division for matrices is not defined. Also, `inv(pv)` -- read "inverse of `pv`"-- won't work because the inverse of a matrix is not the same as inverting element by element. So again, we use the "dot" at the end of a vector to denote the element-by-element operation.

```
oneoverpv=(ones(size(pv(12:18)))./pv(12:18))'
```

```
oneoverpv =
  1.2195    1.1236    1.0417    0.9709    0.9174    0.8621    0.8130
```

Note that we used the *MATLAB* `size` function to specify a vector of ones with the same length as the set of "pv's" that we want to invert. We also placed a dot at the end of the vector of ones to indicate that we wanted to take the inverse of each element of the vector.

As a final example, consider calculating the difference between successive values of `rc` of the breakthrough data that we loaded earlier (i.e., Δrc) divided by the difference in successive `pv` values (Δpv). The *MATLAB* `diff` function calculates these differences. For example,

```
diff(rc)'
```

```
ans =
```

```
Columns 1 through 7
    0         0         0         0         0         0         0
Columns 8 through 14
    0    -0.0200         0         0    0.0200    0.0800    0.3400
Columns 15 through 21
    0.3400   -0.1200   -0.3600   -0.1600   -0.0800   -0.0400         0
```

We use the dot at the end of the numerator to calculate $\Delta rc/\Delta pv$ on a term by term basis:

```
(diff(rc)./diff(pv))'
```

```
ans =
Columns 1 through 7
    0         0         0         0         0         0         0
Columns 8 through 14
    0   -0.3333         0         0    0.2857    1.1429    4.8571
Columns 15 through 21
    5.6667   -1.7143   -5.1429   -2.2857   -1.1429   -0.5714         0
```

1.3. Symbolic math toolbox

The student version of *MATLAB* includes the symbolic mathematics toolbox. This tool performs mathematical operations – differentiation, integration, equation solving, etc. – on symbols. That is, the toolbox operates on mathematical symbols (the "x" of algebra!) and not on numbers. Although we concentrate on *numerical* methods in this series of lectures, access to the symbolic math toolbox will be useful. For example, the symbolic math toolbox can be used to obtain the exact analytical solution to some of the problems we will attack with numerical methods, allowing evaluation of errors in the numerical (by definition, approximate) solution. Our concentration, however, will be very strongly on numerical computation. We present just a few basics here.

To manipulate symbols, *MATLAB* first must be informed about what variables are to be treated symbolically. Such variables are declared using `syms`. For example, suppose we want to work with the Manning equation,

$$Q = \frac{1}{n} S^{1/2} \left(\frac{A}{p} \right)^{2/3} A.$$

We would type the declaration

```
syms Q n S A p
```

to make all of the variables in the equation symbolic. Now suppose that we want to solve this equation for the wetted perimeter, p . We can use the `solve` command:

```
p=solve('Q=(1/n)*sqrt(S)*(A/p)^(2/3)*A',p)
```

which results in the following.

```
p =
```

```
[ 1/Q^2/n^2*(Q*n*S^(3/2)*A)^(1/2)*A^2]
[-1/Q^2/n^2*(Q*n*S^(3/2)*A)^(1/2)*A^2]
```

Note that both the positive and negative solutions are given. To make the solution clearer (the positive solution, which is the one of interest), we use the `pretty` command:

```
pretty(p(1)).
```

Try this example to see how the solution looks in a more traditionally readable form.

The *MATLAB* toolbox can also be used to solve differential equations. Consider steady, one-dimensional flow of groundwater between two drains. The head remains constant at 5 m in one drain and at 10 m in the other. The aquifer, with transmissivity T , is being recharged at a constant rate, w , along its length. The (linearized) equation describing the case is:

$$\begin{aligned}\frac{d^2 h}{dx^2} &= -\frac{w}{T} \\ h(0) &= 5 \\ h(100) &= 10.\end{aligned}$$

The solution to the equation can be gotten with the following commands.

```
syms h T w
h=dsolve('D2h=-w/T','h(0)=5','h(100)=10','x')
```

This command results in the following response.

```
h =
-1/2*w/T*x^2+1/20*(1000*w+T)/T*x+5
```

Again, the `pretty` command puts the solution into a more standard symbolic format. To determine the solution for $w/T=0.002$ per meter, simply insert the numerical value at the appropriate place.

```
h=dsolve('D2h=-0.002','h(0)=5','h(100)=10','x')
```

```
h =
-1/1000*x^2+3/20*x+5
```

The command: `ezplot(h,[0 100])` shows graphically how head in the aquifer varies with distance (Figure 1.3).

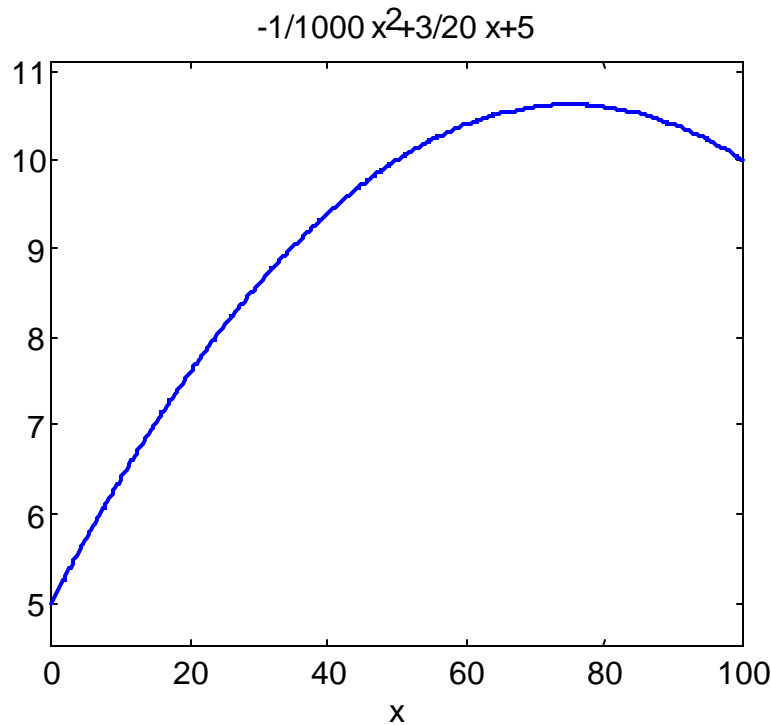


Figure 1.3. Solution to differential equation from the symbolic math toolbox.

1.4. Programming in *MATLAB*

To utilize the full power of *MATLAB*, you will need to learn to write "m-files". These files are programs to execute a sequence of calculations and operations. For example, you might want to have a program to take a set of data and perform some statistical calculation. The files can contain any legitimate *MATLAB* command. The files can have any legal name with the extension "m". *MATLAB* looks for files with an "m" extension when the prefix is typed, interprets the commands, and executes them in sequence.

Script files

The first type of m-file that you will use is a straight script file. Script files are, literally, a list of commands that are interpreted and executed. To illustrate, suppose you want to create a program to load a set of concentration-time data (defining a breakthrough curve) and calculate the mean travel time as $\sum c_i t_i / \sum c_i$. Using the *MATLAB* editor/debugger (or your favorite word processor), you type in the following:

```
% Calculate mean travel time
%
load bt.dat
t=bt(:,1);c=bt(:,2);
tbar=sum(t.*c)/sum(c)
```

and save it in file `tbar.m`. (Note that lines in the file preceded by a "%" are comments; *MATLAB* ignores them.) Then, from within *MATLAB*, type `tbar` and return.

```
tbar
```

```
tbar =  
1.0796
```

The 'tbar' code is a particularly simple m-file, but you should be able to appreciate how easy it is to program more complicated tasks in *MATLAB*. For example, the script below calculates daily estimates of potential evapotranspiration using the Hamon (1961) method [Box 1.2].



```
% Hamon calculation of PET in mm/day  
% inputs are (1) latitude (degrees) and (2) a file name with data:  
%     Julian day  
%     Temperature (Celsius)  
% output is daily values of PET  
  
fname=input('name of file listing Julian day and T in Celsius? ');  
fid=fopen(fname,'r');  
data=fscanf(fid,'%g');  
data=reshape(data,2,length(data)/2);data=data';  
status=fclose(fid);  
J=data(:,1);T=data(:,2);    % data are J, Julian day, and T, temperature  
  
phi=input('latitude in degrees? ');    % latitude  
delta=0.4093*sin((2*pi/365)*J-1.405); % solar declination  
omega_s=acos(-tan(2*pi*phi/360).*tan(delta)); % sunset hour angle  
Nt=24*omega_s/pi;    % hours in day  
a=0.6108;b=17.27;c=237.3;  
es=a*exp(b*T./(T+c));    % saturation vapor pressure  
E=(2.1*(Nt.^2).*es)./(T+273.3);    % Hamon PET  
i_cold=find(T<=0);E(i_cold)=0;  
  
plot(J,E)  
xlabel('Time, Julian day')  
ylabel('Hamon PET, mm/day')
```

Executing the code with one year of data for a site in central Virginia shows the annual cycle of potential evapotranspiration with day-to-day variability introduced by temperature fluctuations (Figure 1.4).

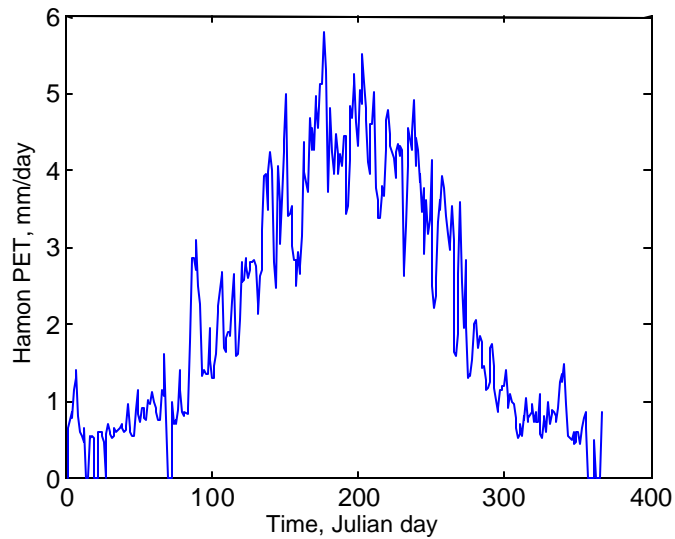


Figure 1.4. Potential evapotranspiration estimated using Hamon's method.

Function files

The other type of m-file, which is used more frequently than a script file, is a function file. A function file operates on vectors or matrices specified in the call and returns variables defined in the function statement. The *MATLAB* help on this topic illustrates the concept nicely³.

help function

FUNCTION Function M-files.

New functions may be added to MATLAB's vocabulary if they are expressed in terms of other existing functions. The commands and functions that comprise the new function must be put in a file whose name defines the name of the new function, with a filename extension of '.m'. At the top of the file must be a line that contains the syntax definition for the new function. For example, the existence of a file on disk called STAT.M with:

```
function [mean,stdev] = stat(x)
n = length(x);
mean = sum(x) / n;
stdev = sqrt(sum((x - mean).^2)/n);
```

defines a new function called STAT that calculates the mean and standard deviation of a vector. The variables within the body of the function are all local variables. See SCRIPT for procedures that work globally on the workspace.

See also ECHO, SCRIPT.

³ Reprinted with permission from The Mathworks, Inc.

Function files are especially useful for doing computations to be applied to different data sets or for different values of parameters in an equation. As an example of a case where we want a function to operate on various data sets, consider the code below for doing simple linear regression on a set of x-y data.

```
function [m,b,sm,sb,r2]=linreg(x,y)
%
% Simple linear regression of variable y on variable x
% y=mx+b
% syntax: [m,b,sm,sb,r2]=linreg(x,y)
% m is estimated slope; sm is standard error of estimated slope
% b is estimated intercept; sb is standard error of estimated intercept
% r2 is the coefficient of variation
% Solution to normal equations using unweighted least squares:
% m=(n*sum(x*y)-sum(x)*sum(y))/(n*sum(x^2)-(sum(x))^2)
% b=mean(y)-m*mean(x)
%
[k1,k2]=size(x);
n=max(k1,k2);
m=(n*sum(x.*y)-sum(x)*sum(y))/(n*sum(x.^2)-sum(x)^2);
b=mean(y)-m*mean(x); % regression parameters
r=(y-b-m*x); % residuals
s2=sum(r.*r)/(n-2); % mean squared error
sm=sqrt(s2*(1/n+mean(x)^2)/sum((x-mean(x)).^2)); % standard error
sb=sqrt(s2/sum((x-mean(x)).^2)); % standard error
numerator=sum(x.*y)-(sum(x)*sum(y)/n).^2;
denominator=(sum(x.^2)-(sum(x)^2/n))*(sum(y.^2)-(sum(y)^2/n));
r2=numerator/denominator; % r-squared
y_hat=m*x+b; % estimated y from regression
%
% 95% prediction confidence intervals based on large values of n
factor= sqrt(1+1/n+(x-mean(x)).^2/sum((x-mean(x)).^2));
y_upper=y_hat+2.306*sqrt(s2)*factor;
y_lower=y_hat-2.306*sqrt(s2)*factor;
d=max(x)-min(x);
xi=min(x):d/50:max(x);
yh=m*xi+b;
yyl=spline(x,y_lower,xi);
yyu=spline(x,y_upper,xi);
plot(x,y,'+r',xi,yh,xi,yyu,'g',xi,yyl,'g','MarkerSize',8)
xlabel('x')
ylabel('y')
text(0.2*mean(x),0.8*max(y),['r^2=' num2str(r2)])
```

As an example of a function file to compute results with different model parameters, consider the case of dispersion of a non-reactive contaminant in a homogeneous aquifer with average pore velocity v . A mass m of contaminant is assumed to be injected instantaneously into an extensive aquifer of thickness b (Figure 1.5). The equation governing the flow and dispersion (determined by the dispersion coefficients, D_x and D_y , of the contaminant and the analytical solution to the equation for this case are (Wilson and Miller 1978):

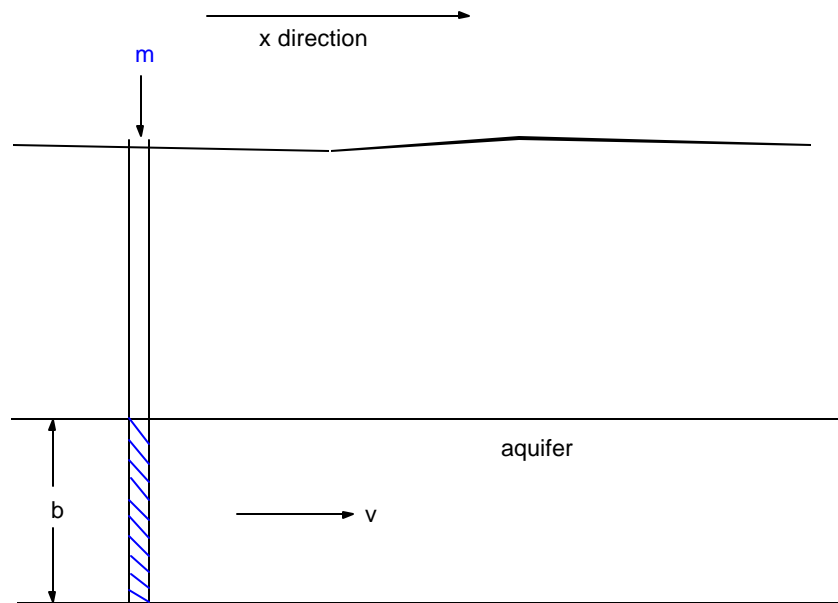


Figure 1.5. Schematic of contaminant pulse injection into an aquifer.

Equation:
$$\frac{\partial c}{\partial t} = D_x \frac{\partial^2 c}{\partial x^2} + D_y \frac{\partial^2 c}{\partial y^2} - v \frac{\partial c}{\partial x}$$

Solution:
$$c(x, y, t) = \frac{m/b}{4\pi t \sqrt{D_x D_y}} \exp\left(-\frac{(x-vt)^2}{4D_x t} - \frac{y^2}{4D_y t}\right)$$

Note that m, b, v, D_x , and D_y are passed to the function file as parameters. This solution is sometimes referred to as a "Gaussian puff" because the cloud spreads out in the form of a Gaussian distribution. The *MATLAB* function file below computes the solution and also demonstrates one of the visualization tools available in *MATLAB*. The solution is contoured at a series of times and each "frame" is saved using `getframe`. After executing `G_puff` the command `movie(M)` can be used to display the sequence of frames (Video 1.1).

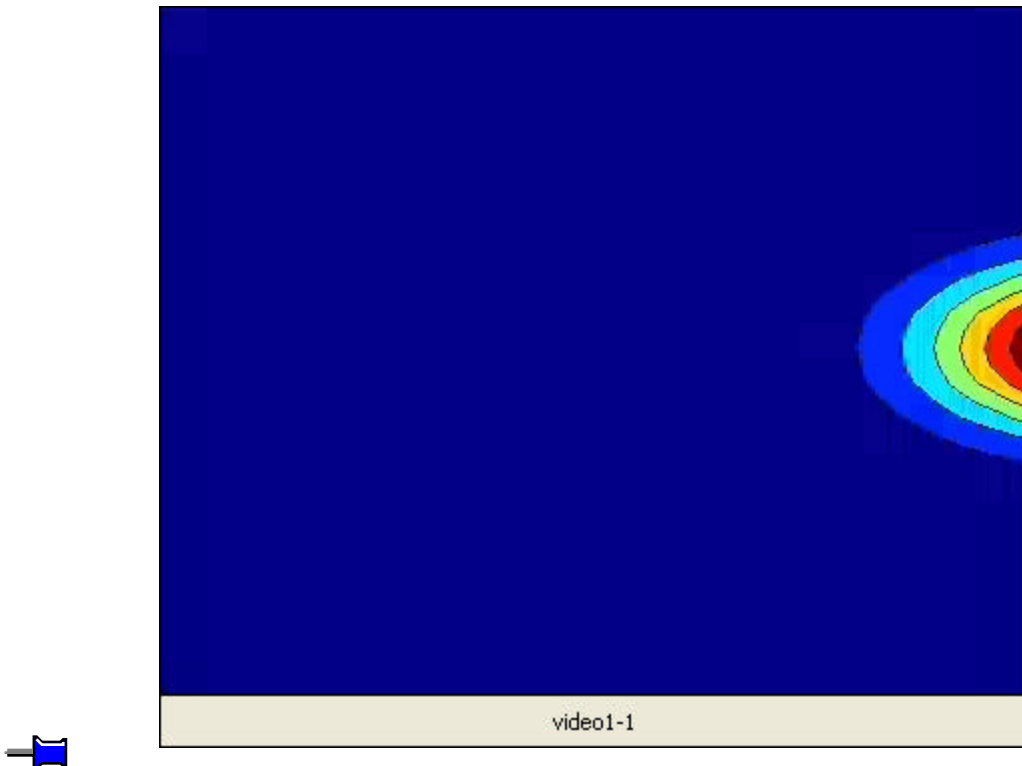
```
function M=G_puff(m,Dx,Dy,v,b)
% Gaussian puff over a 100-meter length
% syntax: M=G_puff(m,Dx,Dy,v,b)
% m=mass; v=average pore velocity; b=aquifer thickness
% Dx=axial dispersion coeff.; Dy=transverse dispersion coeff.;

[X, Y]=meshgrid(-5:2.5:100, -50:2.5:50);
tmax=110/v;
t=tmax/20:tmax/20:tmax;
n=length(t);
```

```

axis tight
set(gca, 'nextplot', 'replacechildren');
for k=1:n
e=exp(-(X-v*t(k)).^2/(4*Dx*t(k))-Y.^2/(4*Dy*t(k)));
c=(m/b)/(4*pi*t(k)*sqrt(Dx*Dy))*e;
contourf(c)
set(gca, 'Visible', 'off');
M(k)=getframe;
end

```




Video 1.1. Transport of a contaminant in an aquifer.

1.5. Summary

Throughout the series of lectures presented in these notes, *MATLAB* is used for computation and for programming. You should work to familiarize yourself with this software. Read these notes, read the documentation (available with the student edition of *MATLAB*, which is recommended strongly), use the on-line help facility in the program liberally, and practice working with *MATLAB* by writing m files.

1.6. Problems

The problems below are designed to introduce work with *MATLAB* and to do some basic data manipulation and simple programming. The data below are precipitation in mm for the Southern Piedmont of the U.S. (Karl et al. 1994). The rows represent years from 1960 through 1989.

	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
	133.8	157.6	120.9	84.5	87.5	71.3	130.0	112.7	100.1	55.2	28.0	48.1
	70.4	153.6	115.7	136.7	91.7	143.1	96.4	159.4	35.6	44.8	54.2	127.1
	139.2	100.0	124.7	94.4	63.3	136.0	96.4	77.0	121.5	35.6	130.1	73.1
	92.0	78.5	127.1	69.3	80.2	106.5	87.5	53.9	114.0	4.7	135.7	70.3
	142.7	125.5	129.5	118.6	51.9	90.1	170.1	166.0	84.4	178.4	49.6	111.8
	48.9	104.7	159.1	81.7	46.6	157.4	151.6	89.5	68.2	64.3	44.1	15.8
	130.5	109.0	69.8	67.7	116.2	75.6	77.5	94.7	130.9	74.9	40.1	78.5
	62.5	86.9	67.3	55.3	127.3	73.8	115.7	176.8	64.2	40.4	68.0	123.0
	111.9	19.4	79.4	68.4	107.9	125.4	138.9	69.0	34.3	92.6	107.1	69.3
	64.8	84.8	103.6	112.5	84.8	107.2	120.5	123.6	105.2	37.6	37.1	114.4
	58.5	79.4	121.5	66.6	75.9	67.9	136.5	140.7	50.1	125.8	63.6	75.3
	100.1	116.5	133.8	73.8	153.0	87.5	147.7	137.7	108.9	171.9	69.5	43.7
	107.8	108.1	75.3	58.0	145.9	181.2	117.4	86.1	84.0	89.2	125.7	140.2
	97.9	114.1	144.2	126.6	109.0	172.1	85.7	101.5	72.9	60.5	27.9	146.6
	122.4	90.2	78.6	66.4	126.9	91.1	91.4	139.6	121.0	14.4	61.5	132.5
	137.9	108.9	185.7	66.0	141.0	90.3	213.7	71.7	182.5	58.7	71.6	94.5
	84.0	36.5	98.0	24.2	146.3	144.2	85.4	66.2	116.3	184.6	77.2	107.6
	68.3	31.9	145.1	52.5	60.7	84.0	54.8	114.9	111.8	134.6	103.4	89.7
	199.0	22.2	111.0	101.5	116.5	95.6	140.7	116.0	54.9	28.9	75.8	76.3
	148.8	157.4	84.4	134.4	116.3	131.2	95.9	93.8	208.8	77.9	109.4	35.6
	120.7	42.6	211.3	68.3	86.3	80.3	82.5	53.5	128.2	88.4	72.6	29.9
	17.1	98.0	58.5	45.2	75.9	102.4	146.0	123.2	72.5	82.8	24.0	139.6
	120.3	131.1	57.3	114.4	93.7	170.0	118.4	93.9	67.0	88.2	74.0	106.4
	68.4	126.5	176.7	147.2	81.6	79.2	51.5	79.7	77.2	98.0	119.4	172.8
	95.1	137.5	148.7	119.5	142.1	62.9	207.8	113.0	19.7	57.0	52.4	49.2
	92.4	124.4	26.5	28.9	104.3	85.5	158.1	166.1	17.3	115.3	213.2	31.9
	33.8	50.3	67.4	28.7	67.8	36.4	83.5	192.1	35.7	85.7	118.3	101.6
	153.5	98.3	117.9	107.2	56.3	116.3	73.7	75.8	193.0	32.1	98.2	70.5

85.7	44.5	60.7	71.7	86.7	73.2	102.7	109.1	109.3	74.1	96.7	22.2
53.6	112.3	131.4	102.9	135.7	172.8	178.8	99.6	131.1	121.9	71.1	80.3

1. Read the data into *MATLAB*.
2. Calculate the total annual ppt for each year and plot these data versus year. (Hint: use the *MATLAB* help facility to check on the **sum** command. Recall that you can perform operations on the transpose of a matrix.)
3. Calculate the mean monthly ppt for each month and plot the values using a bar chart. (Hint: check on the **mean** and **bar** commands.) Plot the monthly means on a regular plot along with "error bars" showing the standard deviations of the monthly means. (Hint: check the **errorbar** command and the **std** command.) How different are the means and medians? Hint: check the **median** command.)
4. Plot all monthly precipitation values consecutively. (You may want to examine the **reshape** command.) Is a seasonal pattern evident in the data?
5. Write an m-file program to: (a) query for a year (see the **input** command); (b) for the selected year, calculate and display the minimum, maximum, and mean monthly precipitation (see the **min** and **max** commands); (c) make a stem plot of the data (see the **stem** command), labelling the axes and placing an appropriate title (see the **xlabel**, **ylabel** and **title** commands).
6. Write a function file that accepts an $N \times M$ matrix (such as the precipitation file) as an argument, finds the maximum correlation coefficient (absolute value) between any two columns of the data, reports that value as output, and presents a scatter plot of the two columns of data with maximal correlation. Apply the code to the precipitation file and briefly comment on any interpretation of the result. (You may want to use **corrcoef** and **eye**.)
7. Use the symbolic math toolbox with Manning's equation to solve for channel width given the following information: $S=0.036$, $n=0.02$, $Q=2 \text{ m}^3\text{s}^{-1}$, channel width= w (the "unknown"), and water depth= $w/2.5$.

1.7. References

- Haith, D. A. and L. L. Shoemaker, Generalized watershed loading functions for stream flow nutrients. *Water Res. Bull.*, 23: 471-478, 1987.
- Hamon, W.R., Estimating Potential Evapotranspiration, *J. Hydraul. Div., ASCE*, 87: 107-120, 1961.
- Karl, T.R., D.R. Easterling, and P.Ya.Groisman, United States historical climatology network - National and regional estimates of monthly and annual precipitation. pp. 830-905. In: T.A. Boden, D.P. Kaiser, R.J. Sepanski, and F.W. Stoss (eds.) *Trends '93: A Compendium of Data on Global Change*, ORNL/CDIAC-65. Carbon Dioxide Information Analysis Center, Oak Ridge National Laboratory, Oak Ridge, Tenn., USA, 1994.
- Wilson, J. and P. Miller, Two-dimensional plume in uniform groundwater flow, *J. Hydraul. Div., ASCE*, 104: 503-514, 1978.

Box 1.1. Controlling attributes of a plot

Properties of graphs can be changed in *MATLAB* using the editor in the Figure window itself. Clicking the "edit plot" icon invokes the editing mode from which various Figure properties can be changed. Alternatively, plots can be edited from the command line by setting properties. Many properties can be set within the basic graphics commands. For example, the commands listed below produce larger symbol sizes and larger fonts relative to the defaults.

```
plot(t,temp,'+', 'MarkerSize',8);  
xlabel('Time, days','FontName','helvetica','FontSize',14);  
ylabel('Temperature, degrees C','FontName','helvetica','FontSize',14)
```

Use the "Help Desk" facility in *MATLAB* to learn more about how to edit Figure properties.

Box 1.2. Estimating potential evapotranspiration (PET) using temperature data

One of the simplest estimates of potential evaporation is presented by Hamon (1961). Following Haith and Shoemaker (1987), Hamon's estimate of potential evaporation is:

$$E_t = \frac{2.1 H_t^2 e_s}{T_t + 273.3}$$

E_t = evaporation on day t [mm day⁻¹]

H_t = average number of daylight hours per day during the month in which day t falls

e_s = saturated vapor pressure at temperature T [kPa]

T_t = temperature on day t [° C]

e_s can be calculated from temperature: $e_s = 0.2749 \times 10^8 \exp \left[\frac{-4278.6}{T_t + 242.8} \right]$. H_t can be calculated by using the maximum number of daylight hours on day t , N_t , which is equal to $\frac{24 w_s}{p}$, where w_s is the sunset hour angle of day t , $w_s = \arccos(-\tan f \tan d)$, where f is the latitude and d is the solar declination given by $d = 0.4093 \sin \left(\frac{2p}{365} J - 1.405 \right)$. On days when $T_t \leq 0$, Haith and Shoemaker set $E=0$.