TEXAS TECH UNIVERSITY

David S. Gilliam
Department of Mathematics
Texas Tech University
Lubbock, TX 79409

806 742-2566
gilliam@texas.math.ttu.edu
http://texas.math.ttu.edu/~gilliam

Mathematics 4330/5344 – # 2 Vectors, Colon, Plotting, Hadamard Operations and Sprintf

1 Introductory Remarks

Much of the material in this lesson comes from the supplemental text for the course "Introducton to Scientific Computing" by C. Van Loan.

The objective in this lesson is to introduce you to various ways of building vectors, functions of vectors, writing "for" loops, building and plotting functions. Several instructional examples are used to motivate the material.

2 Lesson and Examples

1. A vector of x partition of an interval $[a, b]$ is given by

$$a = x_1 < x_2 < \cdots < x_n = b$$

and a vector of values of a function $f(x)$ are given by $y_j = f(x_j)$. A polygonal line plot of the data points (x_j, y_j), $j = 1, 2, \cdots, n$ then gives an approximation to the graph of $y = f(x)$ on $[a, b]$.

2. For example, in matlab we could enter a row vector with three entries as

```
x=[10.1 20.2 30.3]
```

or a column vector as

```
x=[10.1;20.2;30.3]
```

3. If you enter these expression in Matlab the result will define x to be the appropriate right hand side and will then display the result on the screen. If you don't want to display the result simple end the line with a" semi-colon." For example,

   ```
    x=[10.1 20.2 30.3]
   ```

   will set x equal to the right hands side but suppress printing the output.

4. Once x has been defined it can be displayed any time by simply typing `x` and return.

5. To transform a row vector of real numbers x into a column vector of real numbers y simply compute the *transpose* which is given by a "single quote" as in

   ```
    y=x'
   ```

6. A plot with only three terms would not be very informative. Thus we are confronted with the need to build longer vectors. We could, for example, write an 11-vector for the interval $[0, 1]$ as

   ```
    x=[0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1]
   ```

7. Building vectors this way would be a lot of work for larger vectors. For this reason let us first introduce the notion of a "for-loop." The syntax in this simple case would be given by

   ```
   for j=1:11          % for j running from 1 to 11
   x(j)=(j-1)/10;      % compute the jth term
   end                 % ending delimiter
   ```

   First note the use of % which is used to add comments in programs. Matlab ignores everthing on a line following the "percent symbol." Next you see the first use of "colon" notation. The expression

   ```
   J=1:11
   ```

   produces the vector of indices

   ```
   J=[1 2 3 4 5 6 7 8 9 10 11]
   ```

   The for statement simply says: execute the commands following the "for" statement all the way until an "end" is reached for every index $j \in J$.

8. Using loops is easier than entering each element of the vector by hand. Loops can also be used to build a vector of function values. Let us consider the function

   $$f(x) = sin(2\pi x).$$

   Suppose we want to build a vector y whose jth entry is $f(x(j))$. Then we can write

```
for j=1:11
y(j)= sin(2*pi*x(j));
end
```

9. In order to plot the vectors x versus y in Matlab, type

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

10. In the "for" statment above we saw the notation $1 : 11$. This same notation can be used to generate x. Namely, we can write

    ```
    x=0:.1:1
    ```

    More generally, vectors of equispaced elements can be generated using the general format:

    ```
    {beginning number} : {step increment} : {last number}
    ```

    If the step is 1 then it can be omitted

    ```
    {beginning number}  : {last number}
    ```

    so that $x = 1 : 6$ is the vector `x=[1 2 3 4 5  6]`. You can also use a negative incremental step size, if the first number is smaller than the last. For example

    ```
    x=10:-1:1
    ```

    builds the vector `[10 9 8 7 6 5 4 3 2 1]`.

11. If you were to type

    ```
    x=10:1
    ```

    Matlab would return

    ```
    x =
         []
    ```

    where `[\,]` (left brace right brace) represents the empty matrix, i.e., a matrix with no entries. It turns out that the empty matrix is very useful in programming.

12. You could also build the vector x above by using the "linspace" command (type `help linspace` for more information)

    ```
    x=linspace(0,1,11)
    ```

see also “logspace.”

```
x=logspace(.01,1,15);
y=log(x);
plot(x,y,x,y,'*')
```

13. In fact here is a good time to remind you once again of the on-line help: You should look at

    (a) `help who`
    (b) `help whos`
    (c) `help clear`
    (d) `help for`
    (e) `help zeros`
    (f) `help ones`
    (g) `help eye`
    (h) `help ;`
    (i) `help []`
    (j) `help linspace`
    (k) `help logspace`
    (l) `help elfun`
    (m) `help plot`
    (n) `help length`
    (o) `help size`

14. Okay, lets return to plotting and evaluating functions and colon notation. You have now seen how we can use colon notation, rather than loops, to define vectors. This is called *vectorization*. One of the really nice aspects of Matlab is that most builtin functions are built to handle vectorization. Consider the following:

```
x = linspace(0,1,11) ;
y = sin(2*pi*x) ;
plot(x,y)
```

The first command builds a vector x, the next builds a vector y where the *sin* function can be applied to a vector (or matrix). In fact if f is any of the builtin functions in Matlab (see `help elfun`) and A is a matrix

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix}, \quad f(A) = \begin{pmatrix} f(a_{11}) & f(a_{12}) & \cdots & f(a_{1m}) \\ f(a_{21}) & f(a_{22}) & \cdots & f(a_{2m}) \\ \vdots & \vdots & \ddots & \vdots \\ f(a_{n1}) & f(a_{n2}) & \cdots & f(a_{nm}) \end{pmatrix}.$$

15. As a second simple example lets look at a common way of generating “noisy data” sets.

```
x=0:.025:2;
y=sin(pi*x);
yn=y+.25*(rand(size(x)) - .5);
plot(x,y,'--',x,yn)
title(['dashed line: sin(pi*x),' ...
 '  solid line: noisy data'])
xlabel(['noise is random from [-1/8,1/8]'])
```

16. It turns out that there are many ways in which colon notation is used in vector calculations. It is important to try to master these concepts as soon as possible because they greatly facilitate Matlab programming. Here is another example. The goal is to reduce by 1/2 the number of sine function evaluations needed to build $\sin(2 * \pi * x)$ on the interval $[0, 1]$.

```
m=5; n=4*m+1; x=linspace(0,1,n) ;
% [x(1),x(2), ... x(21)],   x(j)-x(j-1)=.05
y=zeros(1,n);
xf=x( 1:(2*m+1) )  ;
% the values [0, .05, ... ,.5]
y(1:(2*m+1))=sin(2*pi*xf);
y( (2*m+2):n )= -y(2:(2*m+1)) ;
% here we are exploiting the
%fact that sin(x+pi)=-sin(x)
plot(x,y)
```

17. The point is this: If `J=[J(1),J(2),...,J(k)]` is a vector of indices of a vector `x=[x(1),x(2),...,x(n)]`, then `w=x(J)` is the vector `w= [x(J(1)),...,x(J(k))]`.

18. More generally, consider the following discussion for a $n \times m$ matrix A:

    (a) to learn the size (i.e, the number of rows and columns) type `[n,m]=size(A)`

    (b) the ith row is `A(i,:)`

    (c) the jth column is `A(:,j)`

    (d) if $1 \le i \le j \le n$ and $1 \le p \le q \le m$, the statement `B=A(i:j,p:q)` gives the matrix
    $$B = \begin{bmatrix} a_{ip} & \cdots & a_{iq} \\ \vdots & \vdots & \vdots \\ a_{jp} & \cdots & a_{jq} \end{bmatrix}$$

    (e) the statement `B=A(i,p:q)` gives the matrix
    $B = \begin{bmatrix} a_{ip} & \cdots & a_{iq} \end{bmatrix}$

(f) the statement `B=A(i:j,p)` gives the matrix

$$B = \begin{bmatrix} a_{ip} \\ \vdots \\ a_{jp} \end{bmatrix}$$

(g) the statement `B=A(i:j,:)` gives the matrix

$$B = \begin{bmatrix} a_{i1} & \cdots & a_{im} \\ \vdots & \vdots & \vdots \\ a_{j1} & \cdots & a_{jm} \end{bmatrix}$$

(h) the statement `B=A(:,p:q)` gives the matrix

$$B = \begin{bmatrix} a_{1p} & \cdots & a_{1q} \\ \vdots & \vdots & \vdots \\ a_{np} & \cdots & a_{nq} \end{bmatrix}$$

19. In order to produce nice looking output from a calculation you can use the *sprintf* command (see also *fprintf* for output to a file).

20. We will only consider a brief development of the syntax for *sprintf* (for a more indepth discussion see a C programming manual on the topic fprintf). The basic syntax looks something like this

```
S = sprintf(FORMAT,A,...)
```

21. Consider a simple example

```
S = sprintf('rho is %5.3f',(1+sqrt(5))/2)
```

(a) The "FORMAT" is a *string* (we will talk about strings later) which for now is just the stuff inside the single quotes. It consists of some words and then the strange looking expression `%5.3f`. This is the crucial part.

(b) The expression `%` is a placemark for a variable (these are always listed at the end after the format string) which in this case is just `(1+sqrt(5))/2`.

(c) The next part `5.3f` is actually three parts:
(1) the `5` says to allow exactly five digits of space,
(2) the `.3` says to display 3 places after the decimal point and
(3) the `f` sets a specific format (flush left).

22. Here is a second example:

```
disp('  n     sum(1:n)   n*(n+1)/2 ')
disp('-----------------------------')
for n=1:10
disp(sprintf(' %3.0f    %5.0f    %5.0f'...
,n,sum(1:n) ,n*(n+1)/2));
end
```

23. Lets consider an example where we begin with the values

```
x = linspace(0,1,11) ;
y = sin(2*pi*x) ;
plot(x,y)
```

with the values $x(k)$ given in degrees instead of radians. This example is taken from [2].

```
% Script File SineTable.m
%
% Prints a short table of sine evaluations.
%
   n = 21;
   x = linspace(0,1,n);
   y = sin(2*pi*x);
   disp(' ')
   disp('  k     x(k)   sin(x(k))')
   disp('------------------------')
   for k=1:21
      degrees = (k-1)*360/(n-1);
      disp(sprintf(' %2.0f     %3.0f     %6.3f'...
         ,k,degrees,y(k)));
   end
   disp( ' ');
   disp('x(k) is given in degrees.')
   disp(sprintf('One Degree = %5.3e Radians',pi/180))
```

The result of executing these commands is

```
  k     x(k)   sin(x(k))
------------------------
  1       0       0.000
  2      18       0.309
  3      36       0.588
  4      54       0.809
  5      72       0.951
  6      90       1.000
  7     108       0.951
  8     126       0.809
  9     144       0.588
 10     162       0.309
 11     180      -0.000
 12     198      -0.309
 13     216      -0.588
 14     234      -0.809
 15     252      -0.951
 16     270      -1.000
 17     288      -0.951
 18     306      -0.809
 19     324      -0.588
 20     342      -0.309
 21     360       0.000

x(k) is given in degrees.
One Degree = 1.745e-02 Radians
```

24. In the SineTable example above there are three variables to print so there are three `%` parts

    ```
    ' %2.0f     %3.0f     %6.3f '
    ```

    Note how the space between terms is determined by the spaces left between the three format terms. Following this we see that there are three values k, $x(k)$ and $\sin(2 * \pi * x(k))$ which are to be printed in place of the `%` placeholders.

25. Notice what happens when the variable is a matrix (or vector).

    ```
    x=(1:5)*2*pi;
    sprintf('   %5.3f',x)
    ```

    Now compare this with

```
x=(1:5)*2*pi;
sprintf('   %5.3f\n',x)
```

The expression `\n` puts in a carriage return linefeed.

26. The last topic in this lesson will be the use of Hadamard (or "dot") operations.

    The Hadamard multiply `.*`, divide `./` and exponentiate `.^` are very useful and powerful syntax.

    ```
       C=A.*B  has entries c(i,j)=a(i,j)b(i,j)

      C=A./B  has entries c(i,j)=a(i,j)/b(i,j)

      C=A.\B  has entries c(i,j)=b(i,j)/a(i,j)

      C=A.^B  has entries c(i,j)=a(i,j)^{b(i,j)}

      C=A.^r  has entries c(i,j)=a(i,j)^r , r a number.

      C=r.^A" has entries c(i,j)=r^{a(i,j)}
    ```

    For example, let

    ```
    A=[2 1;3 -2]
    B=[-4 1;3 2]
    ```

    and compute

    ```
    A.*B
    ```

    and

    ```
    A./B
    ```

    and

    ```
     A.^B
    ```

    and

    ```
     A.^2
    ```

    and

    ```
     2.^B
    ```

This is very useful in evaluating vectorially a function that has products of elementary functions.

27. Suppose for example that you wanted to plot the function $y = x\sin(x^2)$ on the interval $[-4, 4]$. We could use a loop as follows

```
x=-4:.01:4;
lx=length(x);
for j=1:lx
y(j)=x(j)*sin(x(j)^2);
end
plot(x,y)
```

But this is not nearly as efficient as

```
x=-4:.01:4;
y=x.*sin(x.^2);
plot(x,y)
```

28. Here is a final example of multiple plots taken from [2]. The essential new ingredient is the use of *subplot* to generate multiple plot windows.

```
% Script File Polygons
%
% Plots selected regular polygons.

   close all;
% close all the current figure windows
   clc           % clear the screen

   theta = linspace(0,2*pi,361);
   c = cos(theta);
   s = sin(theta);
   k=0;
   for sides = [3 4 5 6 8 10 12 18 24]
        stride = 360/sides;
k=k+1;
subplot(3,3,k)
plot(c(1:stride:361),s(1:stride:361))
xlabel(sprintf(' n = %2.0f',sides));
axis([-1.2 1.2 -1.2 1.2])
axis('square')
% sets the aspect ratio to square
% for better perspective
pause(.01)
 % .01 second pause between each plot
   end
```

ASSIGNMENT 2

1. Build the vector

   ```
   v=[100,95,90, ... ,-95,-100]
   ```

2. Build the vector

   ```
   v=[sin(pi), sin(2 pi), ... , sin(10 pi)]
   ```

3. Build the 1×10 vector

   ```
   v=[0,1,1, ... ,1,1,0]
   ```

4. Build the vector

   ```
   v=[1^2,2^2, ... ,10^2]
   ```

5. Build the vector

   ```
   v=[2,4, ... ,2^8]
   ```

6. Build the vector with complex number entries

   ```
   v=[1+i,1+2i,1+3i, ... ,1+10i]
   ```

   hint: `i= sqrt{-1}` is built into matlab, remember `1+v` adds a one to each entry of `v`.

7. Given the 6×9 matrix

   ```
   A=[1 2 3 4 5 6 7 8 9
   1 1 1 1 1 1 1 1 1
   0 0 0 0 0 0 0 0 0
   9 8 7 6 5 4 3 2 1
   -1 -1 -1 -1 -1 -1 -1 -1 -1
   2 1 2 1 2 1 2 1 1]
   ```

   (a) Build the submatrix `B` consisting of the first and fourth rows and the third thru sixth columns of `A`.
   hint: consider the example

   ```
   A1=fix(10*rand(4))
   S=A1(1:2,3:4)
   ```

   (b) Build the matrix `C` consisting of elements of the even numbered rows of `A`.
   hint: consider the example

```
A1=fix(10*rand(7))
E=A1([2,4,6],:)
```

(c) Reshape the matrix `A` to a new matrix `D` which is `2` by `27`.
hint: consider the example

```
A1=fix(10*rand(3,4))
R=reshape(A1,2,6)
```

(d) First define `AA=A` and then use the empty matrix to omit the first, second and fourth rows of `AA`.
hint: consider the example

```
A1=fix(10*rand(6,4))
A1([1,3,6],:)=[]
```

(e) Build the matrix `F` consisting of the columns of `A` in reverse order.
hint: given the vector

```
v=[1 2 3 4 5]
u=v(5:-1:1)
```

You might also look at `help fliplr`. Here is an example

```
A1=fix(10*rand(6,4))
R1=A1(:,[4:-1:1])
```

(f) Create a matrix `G` which consists of the matrix `A` with all the entries in the even numbered rows replaced by zeros.
hint: consider the example

```
A1=fix(10*rand(6,4))
A1([2 4 6],:)=zeros(3,4)
```

(g) Build a `8` by `11` matrix `H` which is the matrix `A` bordered on all four sides with ones.
hint: consider the example

```
A1=fix(10*rand(2,4))
A2=[1 1 1 1;A1;ones(1,4)]
H1=[ones(4,1) A2 ones(4,1)]
```

8. Build the `1` by `200` vector

```
v=[0,1,0,2,0, ... ,0,99,0,100]
\begin{verbatim}
(hint: you can assign the values of a matrix by
\begin{verbatim} v([1 3 5])=1:3
```

and the values `v(2)` and `v(4)` are automatically set equal to `0`.

9. Given an interval `(a,b)` it is often useful to build a partition consisting of `(n-1)` disjoint intervals `[x(j),x(j+1) ]` of length `1/(n-1)`. This can be accomplished using a partition of `n` points in the interval. A set of such points is obtained from `x(j)=a+(b-a)*((j-1)/(n-1))` where `j` runs from `1` to `(n)`.

   Given the interval `[-7,9]` use Matlab to build this vector `x` when `n=21`.
   hint: consider the example

```
a=1; b=2; n=6; h=(b-a)/(n-1); x1=a + h*(0:(n-1))
```

10. On the same axes plot `sin(j*pi*t)` for `j=1..5`. Use `t=0:.025:1` .
    hint: To plot several graphs on the same axis you will can use several different approachs:

    (a) One approach if there are not to many graphs is to use a single plot command listing each graph as in the following example

```
t=0:.025:1;
plot(t,sin(pi*t), t,sin(2*pi*t), t, ...
sin(3*pi*t), t,sin(4*pi*t), t,sin(5*pi*t))
```

    (b) A second alternative is to use a loop and the command *hold on* (see *help hold*) and when you are done make sure to turn *hold off*.

```
t=0:.025:1;
plot(t,sin(pi*t))
hold on
for j=2:5
plot(t,sin(j*pi*t))
end
hold off
```

    (c) A third method would be to build a matrix with entries $\{\sin(j * pi * t(k)\}_{j=1,k=1}^{5,n}$ where $n = length(t)$ is the length of t. Lets do it Matlab style

```
t=0:.025:1;
lt=length(t);
A=(1:5)'*pi*t;
plot(t,sin(A))
```

11. Here is an exercise concerned with ploting a function with vertical asymptotes. Plot the function $y = \sec(x) = 1/\cos(x)$ on the interval from $-\pi/2$ to $11\pi/2$.
    hint: The following example is taken from [2]:

```
% Plots the function tan(x), -pi/2 <= x <= 9pi/2
  ymax = 10;
  x = linspace(-pi/2,pi/2,40);
  y = tan(x);
```

```
   plot(x,y)
   axis([-pi/2 9*pi/2 -ymax ymax])
   Title('The Tangent Function')
   xlabel('x')
   ylabel('tan(x)')
   hold on
   for k=1:4
      xnew = x+ k*pi;
  plot(xnew,y);
   end
   grid
   hold off
```

12. Plot the rational functions

$$y(x) = \frac{x}{(x^2-4)}, \quad \text{and} \quad y(x) = \frac{x^2}{(x^2-4)}$$

on the interval $[-6, 6]$.
hint: Here is an example showing several ways this can be done for

$$y = \frac{(x-1)^2}{(x-1)(x-3)}.$$

```
x=-6:.01:6;
y=(x-1).^2./((x-3).*(x+2));
plot(x,y)
axis([-6 6 -3 7]);
grid
```

Remark: Notice the *divide by zero* warning. A handy thing to know in Matlab is *eps* which is basically the smallest number bigger than machine precision zero.

```
x=-6:.01:6;
y=(x-1).^2./((x-3).*(x+2)+eps);
plot(x,y)
axis([-6 6 -3 7]);
grid
```

or you can stay away from the asymptotes by plotting in parts

```
d=.001;
x1=-6:.01:(-2-d);   % x values  [-6 , -1.999]
x2=(-2+d):.01:(3-d);% x values  [2.001 , 2.999]
x3=(3+d):.01:6;     % x values  [3.001 , 6]
y1=(x1-1).^2./((x1-3).*(x1+2));
```

```
y2=(x2-1).^2./((x2-3).*(x2+2));
y3=(x3-1).^2./((x3-3).*(x3+2));
plot(x1,y1,x2,y2,x3,y3)
axis([-6 6 -3 7]);
grid
```

13. Use the *sprintf* command to print a table of values of integers k from 1 to 10 and the values of k^2, $\sqrt(k)$, $\log(k)$.

References

[1] *The Matlab Primer,* Kermit Sigmon

[2] *Introduction to scientific computing: a matrix vector approach using Matlab*, Printice Hall, 1997, Charles Van Loan

[3] *Mastering Matlab,* Printice Hall, 1996, Duane Hanselman and Bruce Littlefield

[4] *Advanced Mathematics and Mechanics Applications Using Matlab,* CRC Press, 1994, Howard B. Wilson and Louis H. Turcotte

[5] *Engineering Problem Solving with Matlab*, Printice Hall, 1993, D.M Etter

[6] *Solving Problems in Scientific Computing Using Maple and Matlab,* Walter Gander and Jiri Hrebicek

[7] *Computer Exercises for Linear Algebra*, Printice Hall, 1996, Steven Leon, Eugene Herman, Richard Faulkenberry.

[8] *Contemporary Linear Systems using Matlab*, PWS Publishing Co., 1994, Robert D. Strum, Donald E. Kirk