Chapter 4

Matrices

MATLAB *began as a matrix calculator.*

The Cartesian coordinate system was developed in the 17th century by the French mathematician and philosopher René Descartes. A pair of numbers corresponds to a point in the plane. We will display the coordinates in a *vector* of length two. In order to work properly with matrix multiplication, we want to think of the vector as a *column* vector, So

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

denotes the point x whose first coordinate is x_1 and second coordinate is x_2. When it is inconvenient to write a vector in this vertical form, we can anticipate MATLAB notation and use a semicolon to separate the two components,

$$x = (x_1;\ x_2)$$

For example, the point labeled `x` in figure 4.1 has Cartesian coordinates

$$x = (2;\ 4)$$

Arithmetic operations on the vectors are defined in natural ways. Addition is defined by

$$x + y = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \end{pmatrix}$$

Multiplication by a single number, or *scalar*, is defined by

$$sx = \begin{pmatrix} sx_1 \\ sx_2 \end{pmatrix}$$

August 11, 2011

A 2-by-2 *matrix* is an array of four numbers arranged in two rows and two columns.

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix}$$

or

$$A = \begin{pmatrix} a_{1,1} & a_{1,2}; & a_{2,1} & a_{2,2} \end{pmatrix}$$

For example

$$A = \begin{pmatrix} 4 & -3 \\ -2 & 1 \end{pmatrix}$$

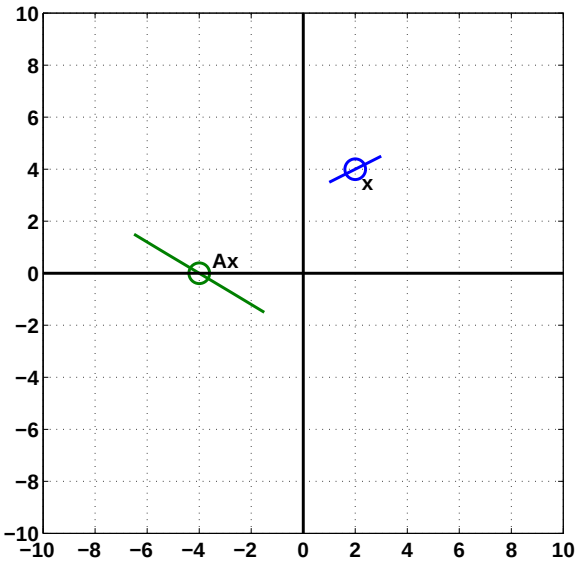

Figure 4.1. *Matrix multiplication transforms lines through x to lines through Ax.*

Matrix-vector multiplication by a 2-by-2 *matrix* A transforms a vector x to a vector Ax, according to the definition

$$Ax = \begin{pmatrix} a_{1,1}x_1 + a_{1,2}x_2 \\ a_{2,1}x_1 + a_{2,2}x_2 \end{pmatrix}$$

For example

$$\begin{pmatrix} 4 & -3 \\ -2 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 4 \end{pmatrix} = \begin{pmatrix} 4 \cdot 2 - 3 \cdot 4 \\ -2 \cdot 2 + 1 \cdot 4 \end{pmatrix} = \begin{pmatrix} -4 \\ 0 \end{pmatrix}$$

The point labeled `x` in figure 4.1 is transformed to the point labeled `Ax`. Matrix-vector multiplications produce *linear* transformations. This means that for scalars s and t and vectors x and y,

$$A(sx + ty) = sAx + tAy$$

This implies that points near x are transformed to points near Ax and that straight lines in the plane through x are transformed to straight lines through Ax.

Our definition of matrix-vector multiplication is the usual one involving the *dot product* of the *rows* of A, denoted $a_{i,:}$, with the vector x.

$$Ax = \begin{pmatrix} a_{1,:} \cdot x \\ a_{2,:} \cdot x \end{pmatrix}$$

An alternate, and sometimes more revealing, definition uses *linear combinations* of the *columns* of A, denoted by $a_{:,j}$.

$$Ax = x_1 a_{:,1} + x_2 a_{:,2}$$

For example

$$\begin{pmatrix} 4 & -3 \\ -2 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 4 \end{pmatrix} = 2 \begin{pmatrix} 4 \\ -2 \end{pmatrix} + 4 \begin{pmatrix} -3 \\ 1 \end{pmatrix} = \begin{pmatrix} -4 \\ 0 \end{pmatrix}$$

The *transpose* of a column vector is a row vector, denoted by x^T. The transpose of a matrix interchanges its rows and columns. For example,

$$x^T = \begin{pmatrix} 2 & 4 \end{pmatrix}$$

$$A^T = \begin{pmatrix} 4 & -2 \\ -3 & 1 \end{pmatrix}$$

Vector-matrix multiplication can be defined by

$$x^T A = A^T x$$

That is pretty cryptic, so if you have never seen it before, you might have to ponder it a bit.

Matrix-matrix multiplication, AB, can be thought of as matrix-vector multiplication involving the matrix$A$ and the columns vectors from B, or as vector-matrix multiplication involving the row vectors from A and the matrix B. It is important to realize that AB is not the same matrix as BA.

MATLAB started its life as "Matrix Laboratory", so its very first capabilities involved matrices and matrix multiplication. The syntax follows the mathematical notation closely. We use square brackets instead of round parentheses, an asterisk to denote multiplication, and `x'` for the transpose of `x`. The foregoing example becomes

```
x = [2; 4]
A = [4 -3; -2 1]
A*x
```

This produces

```
x =
     2
     4
```

```
A =
     4    -3
    -2     1
ans =
    -4
     0
```

The matrices `A'*A` and `A*A'` are not the same.

```
A'*A =
    20   -14
   -14    10
```

while

```
A*A' =
    25   -11
   -11     5
```

The matrix

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

is the 2-by-2 *identity* matrix. It has the important property that for any 2-by-2 matrix A,

$$IA = AI = A$$

Originally, MATLAB variable names were not case sensitive, so `i` and `I` were the same variable. Since `i` is frequently used as a subscript, an iteration index, and `sqrt(-1)`, we could not use `I` for the identity matrix. Instead, we chose to use the sound-alike word `eye`. Today, MATLAB is case sensitive and has many users whose native language is not English, but we continue to use `eye(n,n)` to denote the `n`-by-`n` identity. (The Metro in Washington, DC, uses the same pun – "I street" is "eye street" on their maps.)

2-by-2 Matrix Transformations

The `exm` toolbox includes a function `house`. The statement

```
X = house
```

produces a 2-by-11 matrix,

```
X =
  -6   -6   -7    0    7    6    6   -3   -3    0    0
  -7    2    1    8    1    2   -7   -7   -2   -2   -7
```

The columns of `X` are the Cartesian coordinates of the 11 points shown in figure 4.2. Do you remember the "dot to dot" game? Try it with these points. Finish off by connecting the last point back to the first. The house in figure 4.2 is constructed from `X` by

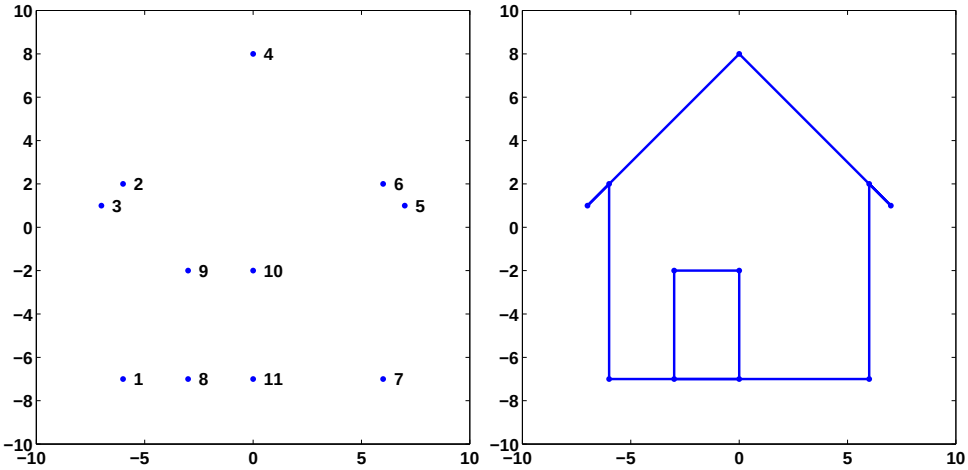

Figure 4.2. *Connect the dots.*

```
dot2dot(X)
```

We want to investigate how matrix multiplication transforms this house. In fact, if you have your computer handy, try this now.

```
wiggle(X)
```

Our goal is to see how `wiggle` works.

Here are four matrices.

```
A1 =
   1/2       0
   0         1

A2 =
   1         0
   0         1/2

A3 =
   0         1
   1/2       0

A4 =
   1/2       0
   0        -1
```

Figure 4.3 uses matrix multiplication `A*X` and `dot2dot(A*X)` to show the effect of the resulting linear transformations on the house. All four matrices are diagonal or antidiagonal, so they just scale and possibly interchange the coordinates. The coordinates are not combined in any way. The floor and sides of the house remain at

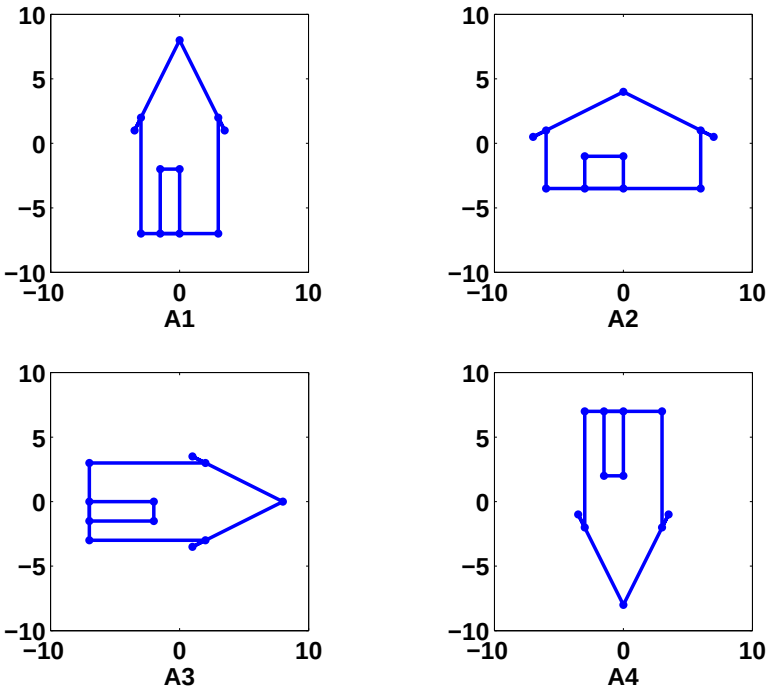

Figure 4.3. *The effect of multiplication by scaling matrices.*

right angles to each other and parallel to the axes. The matrix `A1` shrinks the first coordinate to reduce the width of the house while the height remains unchanged. The matrix `A2` shrinks the second coordinate to reduce the height, but not the width. The matrix `A3` interchanges the two coordinates while shrinking one of them. The matrix `A4` shrinks the first coordinate and changes the sign of the second.

The *determinant* of a 2-by-2 matrix

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix}$$

is the quantity

$$a_{1,1}a_{2,2} - a_{1,2}a_{2,1}$$

In general, determinants are not very useful in practical computation because they have atrocious scaling properties. But 2-by-2 determinants can be useful in understanding simple matrix properties. If the determinant of a matrix is positive, then multiplication by that matrix preserves left- or right-handedness. The first two of our four matrices have positive determinants, so the door remains on the left side of the house. The other two matrices have negative determinants, so the door is transformed to the other side of the house.

The MATLAB function `rand(m,n)` generates an m-by-n matrix with random entries between 0 and 1. So the statement

```
R = 2*rand(2,2) - 1
```

generates a 2-by-2 matrix with random entries between -1 and 1. Here are four of them.

```
R1 =
    0.0323   -0.6327
   -0.5495   -0.5674

R2 =
    0.7277   -0.5997
    0.8124    0.7188

R3 =
    0.1021    0.1777
   -0.3633   -0.5178

R4 =
   -0.8682    0.9330
    0.7992   -0.4821
```

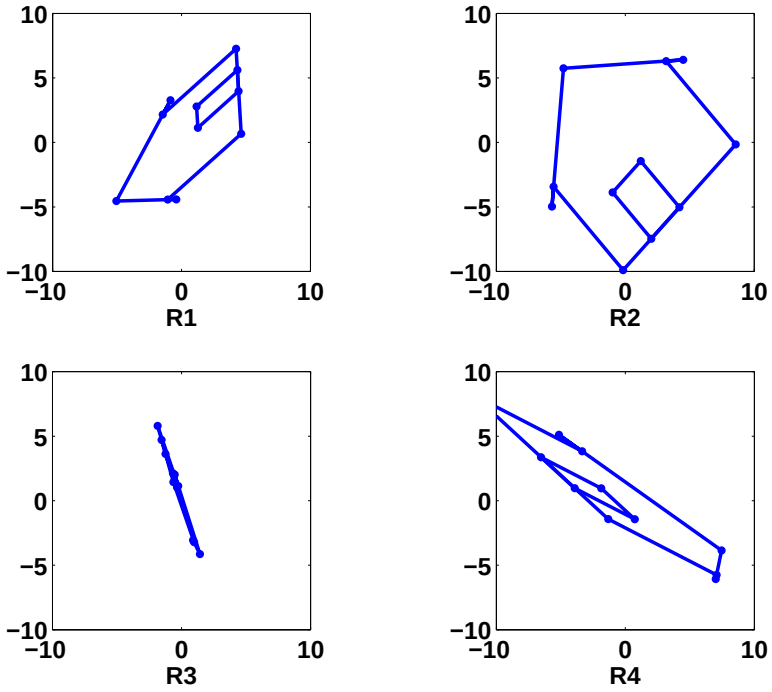

Figure 4.4. *The effect of multiplication by random matrices.*

Figure 4.4 shows the effect of multiplication by these four matrices on the house. Matrices `R1` and `R4` have large off-diagonal entries and negative determinants, so they distort the house quite a bit and flip the door to the right side. The lines are still straight, but the walls are not perpendicular to the floor. Linear transformations preserve straight lines, but they do not necessarily preserve the angles between those lines. Matrix `R2` is close to a rotation, which we will discuss shortly. Matrix `R3` is nearly *singular*; its determinant is equal to 0.0117. If the determinant were exactly zero, the house would be flattened to a one-dimensional straight line.

The following matrix is a *plane rotation*.

$$G(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

We use the letter G because Wallace Givens pioneered the use of plane rotations in matrix computation in the 1950s. Multiplication by $G(\theta)$ rotates points in the plane through an angle θ. Figure 4.5 shows the effect of multiplication by the plane rotations with $\theta = 15°$, $45°$, $90°$, and $215°$.

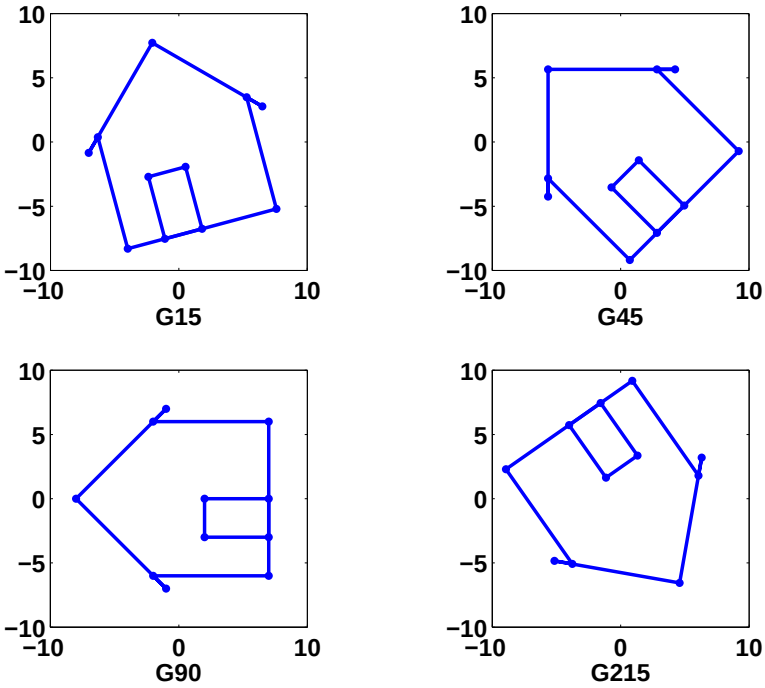

Figure 4.5. *The affect of multiplication by plane rotations though* 15°, 45°, 90°, *and* 215°.

```
G15 =
    0.9659   -0.2588
```

```
    0.2588    0.9659

G45 =
    0.7071   -0.7071
    0.7071    0.7071

G90 =
     0    -1
     1     0

G215 =
   -0.8192    0.5736
   -0.5736   -0.8192
```

You can see that `G45` is fairly close to the random matrix `R2` seen earlier and that its effect on the house is similar.

MATLAB generates a plane rotation for angles measured in radians with

```
G = [cos(theta) -sin(theta); sin(theta) cos(theta)]
```

and for angles measured in degrees with

```
G = [cosd(theta) -sind(theta); sind(theta) cosd(theta)]
```

Our `exm` toolbox function `wiggle` uses `dot2dot` and plane rotations to produce an animation of matrix multiplication. Here is `wiggle.m`, without the Handle Graphics commands.

```
function wiggle(X)
thetamax = 0.1;
delta = .025;
t = 0;
while true
   theta = (4*abs(t-round(t))-1) * thetamax;
   G = [cos(theta) -sin(theta); sin(theta) cos(theta)]
   Y = G*X;
   dot2dot(Y);
   t = t + delta;
end
```

Since this version does not have a stop button, it would run forever. The variable `t` advances steadily by increment of `delta`. As `t` increases, the quantity `t-round(t)` varies between $-1/2$ and $1/2$, so the angle θ computed by

```
theta = (4*abs(t-round(t))-1) * thetamax;
```

varies in a sawtooth fashion between `-thetamax` and `thetamax`. The graph of θ as a function of t is shown in figure 4.6. Each value of θ produces a corresponding plane rotation $G(\theta)$. Then

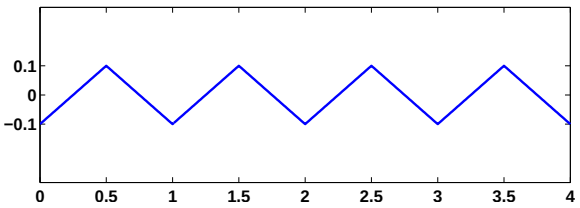

Figure 4.6. *Wiggle angle θ*

```
Y = G*X;
dot2dot(Y)
```

applies the rotation to the input matrix `X` and plots the wiggling result.

Vectors and Matrices

Here is a quick look at a few of the many MATLAB operations involving vectors and matrices. Try to predict what output will be produced by each of the following statements. You can see if you are right by using cut and paste to execute the statement, or by running

```
matrices_recap
```

Vectors are created with square brackets.

```
v = [0 1/4 1/2 3/4 1]
```

Rows of a matrix are separated by semicolons or new lines.

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

There are several functions that create matrices.

```
Z = zeros(3,4)
E = ones(4,3)
I = eye(4,4)
M = magic(3)
R = rand(2,4)
[K,J] = ndgrid(1:4)
```

A colon creates uniformally spaced vectors.

```
v = 0:0.25:1
n = 10
y = 1:n
```

A semicolon at the end of a line suppresses output.

```
n = 1000;
y = 1:n;
```

Matrix arithmetic.

Addition and subtraction, denoted by + and -, are element-by-element. The operation

```
A + B
```

requires `A` and `B` to be the same size, or to be scalars (1-by-1 matrices).

Matrix multiplication, denoted by *, follows the rules of linear algebra. The operation

```
A * B
```

requires the number of columns of `A` to equal the number of row `B`, that is

```
size(A,2) == size(B,1)
```

Remember that `A*B` is usually not equal to `B*A`

```
KJ = K*J
JK = J*K
```

Matrix power is denoted by `^` and is repeated matrix multiplication.

Arrays arithmetic

.

We usually try to distinguish between *matrices*, which behave according to the rules of linear algebra, and *arrays*, which are just rectangular collections of numbers.

Element-by-element operations array operations are denoted by + , - , .* , ./ , . and `.^` . For array multiplication `A.*B` is equal to `B.*A`

```
K.*J
v.^2
```

An apostrophe denotes the transpose of a real array and the complex conjugate transpose of a complex array.

```
v = v'
inner_prod = v'*v
outer_prod = v*v'
Z = [1 2; 3+4i 5]'
Z = [1 2; 3+4i 5].'
```

Further Reading

Of the dozens of good books on matrices and linear algebra, we would like to recommend one in particular.

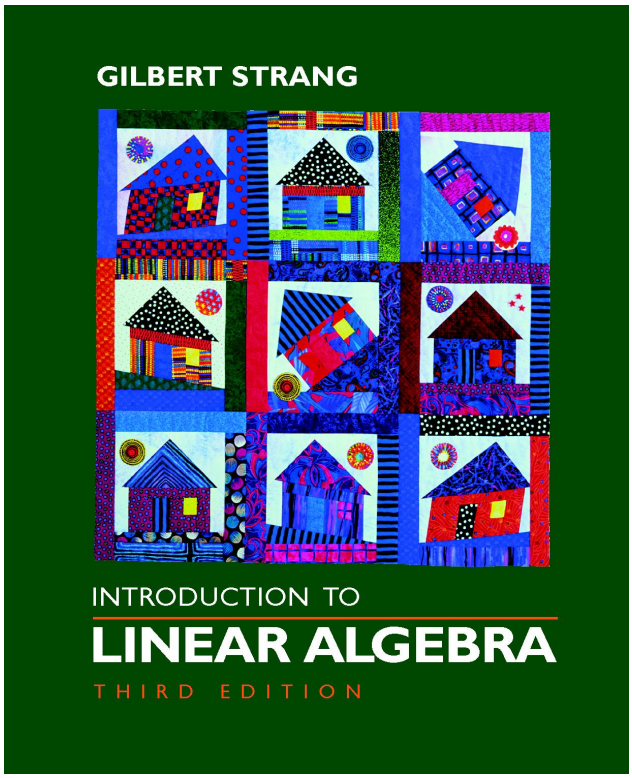

Figure 4.7. *The cover of Gilbert Strang's textbook shows a quilt by Chris Curtis.*

> GILBERT STRANG, *Introduction to Linear Algebra*, Wellesley-Cambridge Press, Wellesley, MA, 2003.
> `http://www.wellesleycambridge.com`

Besides its excellent technical content and exposition, it has a terrific cover. The house that we have used throughout this chapter made its debut in Strang's book in 1993. The cover of the first edition looked something like our figure 4.4. Chris Curtis saw that cover and created a gorgeous quilt. A picture of the quilt has appeared on the cover of all subsequent editions of the book.

Recap

```
%% Recap Matrices Chapter
```

```
%% Recap Matrices Chapter
% This is an executable program that illustrates the statements
% introduced in the Matrices Chapter of "Experiments in MATLAB".
% You can access it with
```

```
%    matrices_recap
%    edit matrices_recap
%    publish matrices_recap

%% Related EXM Programs

%    wiggle
%    dot2dot
%    house
%    hand

%% Vectors and matrices

    x = [2; 4]
    A = [4 -3; -2 1]
    A*x
    A'*A
    A*A'

%% Random matrices

    R = 2*rand(2,2)-1

%% Build a house

    X = house
    dot2dot(X)

%% Rotations

    theta = pi/6   % radians
    G = [cos(theta) -sin(theta); sin(theta) cos(theta)]
    theta = 30  % degrees
    G = [cosd(theta) -sind(theta); sind(theta) cosd(theta)]
    subplot(1,2,1)
    dot2dot(G*X)
    subplot(1,2,2)
    dot2dot(G'*X)

%% More on Vectors and Matrices

% Vectors are created with square brackets.

   v = [0 1/4 1/2 3/4 1]
```

```
% Rows of a matrix are separated by semicolons or new lines.

   A = [8 1 6; 3 5 7; 4 9 2]

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

%% Creating matrices

   Z = zeros(3,4)
   E = ones(4,3)
   I = eye(4,4)
   M = magic(3)
   R = rand(2,4)
   [K,J] = ndgrid(1:4)

%% Colons and semicolons

% A colon creates uniformally spaced vectors.

   v = 0:0.25:1

   n = 10
   y = 1:n

% A semicolon at the end of a line suppresses output.

   n = 1000;
   y = 1:n;


%% Matrix arithmetic.

% Addition and subtraction, + and -, are element-by-element.
% Multiplication, *, follows the rules of linear algebra.
% Power, ^, is repeated matrix multiplication.

   KJ = K*J
   JK = J*K

%% Array arithmetic

% Element-by-element operations are denoted by
% + , - , .* , ./ , .\ and .^ .
```

```
   K.*J
   v.^2

%% Transpose

% An apostrophe denotes the transpose of a real array
% and the complex conjugate transpose of a complex array.

   v = v'
   inner_prod = v'*v
   outer_prod = v*v'
   Z = [1 2; 3+4i 5]'
   Z = [1 2; 3+4i 5].'
```

Exercises

4.1 *Multiplication.*
(a) Which 2-by-2 matrices have $A^2 = I$?
(b) Which 2-by-2 matrices have $A^T A = I$?
(c) Which 2-by-2 matrices have $A^T A = AA^T$?

4.2 *Inverse.* Let

$$A = \begin{pmatrix} 3 & 4 \\ 2 & 3 \end{pmatrix}$$

Find a matrix X so that $AX = I$.

4.3 *Powers.* Let

$$A = \begin{pmatrix} 0.99 & 0.01 \\ -0.01 & 1.01 \end{pmatrix}$$

What is A^n?

4.4 *Powers.* Let

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

What is A^n?

4.5 *Parametrized product.* Let

$$A = \begin{pmatrix} 1 & 2 \\ x & 3 \end{pmatrix} \begin{pmatrix} 4 & 5 \\ 6 & 7 \end{pmatrix}$$

Which elements of A depend upon x? Is it possible to choose x so that $A = A^T$?

4.6 *Product of two symmetric matrices.* It turns out that any matrix is the product of two symmetric matrices. Let

$$A = \begin{pmatrix} 3 & 4 \\ 8 & 10 \end{pmatrix}$$

Express A as the product of two symmetric matrices.

4.7 *Givens rotations.*
(a) What is the determinant of $G(\theta)$?
(b) Explain why $G(\theta)^2 = G(2\theta)$.
(c) Explain why $G(\theta)^n = G(n\theta)$.

4.8 X^8. Find a real 2-by-2 matrix X so that $X^8 = -I$.

4.9 G^T. What is the effect on points in the plane of multiplication by $G(\theta)^T$?

4.10 $\widehat{G}$. (a) What is the effect on points in the plane of multiplication by

$$\widehat{G}(\theta) = \begin{pmatrix} \cos\theta & \sin\theta \\ \sin\theta & -\cos\theta \end{pmatrix}$$

(b) What is the determinant of $\widehat{G}(\theta)$?
(c) What happens if you modify `wiggle.m` to use $\widehat{G}$ instead of G?

4.11 *Goldie.* What does the function `goldie` in the `exm` toolbox do?

4.12 *Transform a hand.* Repeat the experiments in this chapter with

```
X = hand
```

instead of

```
X = house
```

Figure 4.8 shows

```
dot2dot(hand)
```

4.13 *Mirror image.* Find a 2-by-2 matrix R so that

```
dot2dot(house)
```

and

```
dot2dot(R*house)
```

as well as

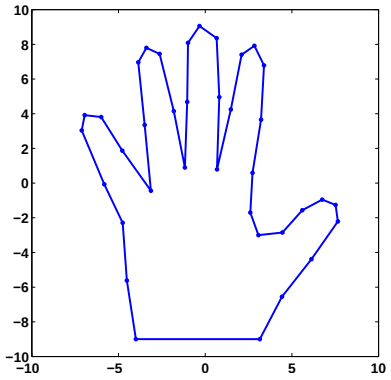

Figure 4.8. *A hand.*

```
dot2dot(hand)
```

and

```
dot2dot(R*hand)
```

are mirror images of each other.

4.14 *Transform your own hand.* Repeat the experiments in this chapter using a plot of your own hand. Start with

```
figure('position',get(0,'screensize'))
axes('position',[0 0 1 1])
axis(10*[-1 1 -1 1])
[x,y] = ginput;
```

Place your hand on the computer screen. Use the mouse to select a few dozen points outlining your hand. Terminate the `ginput` with a carriage return. You might find it easier to trace your hand on a piece of paper and then put the paper on the computer screen. You should be able to see the `ginput` cursor through the paper.

The data you have collected forms two column vectors with entries in the range from -10 to 10. You can arrange the data as two rows in a single matrix with

```
H = [x y]';
```

Then you can use

```
dot2dot(H)
dot2dot(A*H)
wiggle(H)
```

and so on.

You can save your data in the file `myhand.mat` with

```
save myhand H
```

and retrieve it in a later MATLAB session with

```
load myhand
```

4.15 *Wiggler.* Make `wiggler.m`, your own version of `wiggle.m`, with two sliders that control the speed and amplitude. In the initialization, replace the statements

```
thetamax = 0.1;
delta = .025;
```

with

```
thetamax = uicontrol('style','slider','max',1.0, ...
    'units','normalized','position',[.25 .01 .25 .04]);
delta = uicontrol('style','slider','max',.05, ...
    'units','normalized','position',[.60 .01 .25 .04]);
```

The quantities `thetamax` and `delta` are now the *handles* to the two sliders. In the body of the loop, replace `thetamax` by

```
get(thetamax,'value');
```

and replace `delta` by

```
get(delta,'value');
```

Demonstrate your `wiggler` on the `house` and the `hand`.