Chapter 6

Fractal Fern

The fractal fern involves 2-by-2 matrices.

The programs fern and finitefern in the exm toolbox produce the Fractal Fern described by Michael Barnsley in Fractals Everywhere [?]. They generate and plot a potentially infinite sequence of random, but carefully choreographed, points in the plane. The command

fern

runs forever, producing an increasingly dense plot. The command

finitefern(n)

generates n points and a plot like Figure 6.1. The command

finitefern(n,'s')

shows the generation of the points one at a time. The command

F = finitefern(n);

generates, but does not plot, n points and returns an array of zeros and ones for use with sparse matrix and image-processing functions.

The exm toolbox also includes fern.jpg, a 768-by-1024 color image with half a million points that you can view with a browser or a paint program. You can also view the file with

F = imread('fern.png');
image(F)
If you like the image, you might even choose to make it your computer desktop background. However, you should really run `fern` on your own computer to see the dynamics of the emerging fern in high resolution.

The fern is generated by repeated transformations of a point in the plane. Let \( x \) be a vector with two components, \( x_1 \) and \( x_2 \), representing the point. There are four different transformations, all of them of the form

\[
x \rightarrow Ax + b,
\]

with different matrices \( A \) and vectors \( b \). These are known as affine transformations. The most frequently used transformation has

\[
A = \begin{pmatrix} 0.85 & 0.04 \\ -0.04 & 0.85 \end{pmatrix}, \quad b = \begin{pmatrix} 0 \\ 1.6 \end{pmatrix}.
\]

This transformation shortens and rotates \( x \) a little bit, then adds 1.6 to its second component. Repeated application of this transformation moves the point up and to the right, heading toward the upper tip of the fern. Every once in a while, one of the other three transformations is picked at random. These transformations move...
the point into the lower subfern on the right, the lower subfern on the left, or the stem.

Here is the complete fractal fern program.

```matlab
function fern
%FERN MATLAB implementation of the Fractal Fern
%Michael Barnsley, Fractals Everywhere, Academic Press, 1993
%This version runs forever, or until stop is toggled.
%See also: FINITEFERN.

shg
clf reset
set(gcf,'color','white','menubar','none', ...
    'numbertitle','off','name','Fractal Fern')
x = [.5; .5];
h = plot(x(1),x(2),'.');
darkgreen = [0 2/3 0];
set(h,'markersize',1,'color',darkgreen,'erasemode','none');
axis([-3 3 0 10])
axis off
stop = uicontrol('style','toggle','string','stop', ...
    'background','white');
drawnow

p = [ .85 .92 .99 1.00];
A1 = [.85 .04; -.04 .85]; b1 = [0; 1.6];
A2 = [.20 -.26; .23 .22]; b2 = [0; 1.6];
A3 = [-.15 .28; .26 .24]; b3 = [0; .44];
A4 = [ 0 0 ; 0 .16];
cnt = 1;
tic
while ~get(stop,'value')
    r = rand;
    if r < p(1)
        x = A1*x + b1;
    elseif r < p(2)
        x = A2*x + b2;
    elseif r < p(3)
        x = A3*x + b3;
    else
        x = A4*x;
    end
    set(h,'xdata',x(1),'ydata',x(2));
cnt = cnt + 1;
drawnow
```
Let’s examine this program a few statements at a time.

shg

stands for “show graph window.” It brings an existing graphics window forward, or creates a new one if necessary.

clf reset

resets most of the figure properties to their default values.

set(gcf,'color','white','menubar','none', ...
     'numbertitle','off','name','Fractal Fern')

changes the background color of the figure window from the default gray to white and provides a customized title for the window.

x = [.5; .5];

provides the initial coordinates of the point.

h = plot(x(1),x(2),'.');

plots a single dot in the plane and saves a handle, h, so that we can later modify the properties of the plot.

darkgreen = [0 2/3 0];

defines a color where the red and blue components are zero and the green component is two-thirds of its full intensity.

set(h,'markersize',1,'color',darkgreen,'erasemode','none');

makes the dot referenced by h smaller, changes its color, and specifies that the image of the dot on the screen should not be erased when its coordinates are changed. A record of these old points is kept by the computer’s graphics hardware (until the figure is reset), but MATLAB itself does not remember them.

axis([-3 3 0 10])
axis off

specifies that the plot should cover the region

\[-3 \leq x_1 \leq 3, \ 0 \leq x_2 \leq 10,\]

but that the axes should not be drawn.
stop = uicontrol('style','toggle','string','stop', ...  
  'background','white');

creates a toggle user interface control, labeled 'stop' and colored white, in the 
default position near the lower left corner of the figure. The handle for the control 
is saved in the variable stop.

drawnow

causes the initial figure, including the initial point, to actually be plotted on the 
computer screen.
The statement

p = [.85 .92 .99 1.00];

sets up a vector of probabilities. The statements

A1 = [.85 .04; -.04 .85]; b1 = [0; 1.6];
A2 = [.20 -.26; .23 .22]; b2 = [0; 1.6];
A3 = [-.15 .28; .26 .24]; b3 = [0; .44];
A4 = [0 0; 0 .16];

define the four affine transformations. The statement

cnt = 1;

initializes a counter that keeps track of the number of points plotted. The statement

tic

initializes a stopwatch timer. The statement

while ~get(stop,'value')

begins a while loop that runs as long as the 'value' property of the stop toggle is 
equal to 0. Clicking the stop toggle changes the value from 0 to 1 and terminates 
the loop.

r = rand;

generates a pseudorandom value between 0 and 1. The compound if statement

if r < p(1)  
  x = A1*x + b1;
elseif r < p(2)  
  x = A2*x + b2;
elseif r < p(3)  
  x = A3*x + b3;
else  
  x = A4*x;
end
picks one of the four affine transformations. Because $p(1)$ is 0.85, the first transformation is chosen 85% of the time. The other three transformations are chosen relatively infrequently.

$$\text{set(h,'xdata',x(1),'ydata',x(2));}$$

changes the coordinates of the point $h$ to the new $(x_1, x_2)$ and plots this new point. But $\text{get(h,'erasemode')}$ is 'none', so the old point also remains on the screen.

$$\text{cnt = cnt + 1;}$$

counts one more point.

$$\text{drawnow}$$
tells MATLAB to take the time to redraw the figure, showing the new point along with all the old ones. Without this command, nothing would be plotted until $\text{stop}$ is toggled.

$$\text{end}$$
matches the $\text{while}$ at the beginning of the loop. Finally,

$$\text{t = toc;}$$

reads the timer.

$$\text{s = sprintf('8.0f points in 6.3f seconds',cnt,t);}$$
$$\text{text(-1.5,-0.5,s,'fontweight','bold');}$$
displays the elapsed time since $\text{tic}$ was called and the final count of the number of points plotted. Finally,

$$\text{set(stop,'style','pushbutton','string','close', ...}$$
$$\text{'callback','close(gcf)')}$$
changes the control to a push button that closes the window.

**Recap**

%% Fern Chapter Recap
%% This is an executable program that illustrates the statements
%% introduced in the Fern Chapter of "Experiments in MATLAB".
%% You can access it with
%% %
%% fern_recap
%% edit fern_recap
%% publish fern_recap
%%
%% Related EXM programs
%%
Exercises

6.1 Fern color. Change the fern color scheme to use pink on a black background. Don’t forget the stop button.

6.2 Flip the fern. Flip the fern by interchanging its x- and y-coordinates.

6.3 Erase mode.
(a) What happens if you resize the figure window while the fern is being generated? Why?

(b) The exm program finitefern can be used to produce printed output of the fern. Explain why printing is possible with finitefern.m but not with fern.m.

6.4 Fern stem.
(a) What happens to the fern if you change the only nonzero element in the matrix A4?
(b) What are the coordinates of the lower end of the fern’s stem?
6.5 **Fern tip.** The coordinates of the point at the upper tip end of the fern can be computed by solving a certain 2-by-2 system of simultaneous linear equations. What is that system and what are the coordinates of the tip?

6.6 **Trajectories.** The fern algorithm involves repeated random choices from four different formulas for advancing the point. If the kth formula is used repeatedly by itself, without random choices, it defines a deterministic trajectory in the \((x, y)\) plane. Modify `finitefern.m` so that plots of each of these four trajectories are superimposed on the plot of the fern. Start each trajectory at the point \((-1, 5)\). Plot o’s connected with straight lines for the steps along each trajectory. Take as many steps as are needed to show each trajectory’s limit point. You can superimpose several plots with

```matlab
plot(...) hold on plot(...) plot(...) hold off
```

6.7 **Sierpinski’s triangle.** Modify `fern.m` or `finitefern.m` so that it produces Sierpinski’s triangle. Start at

\[ x = \left( \begin{array}{c} 0 \\ 0 \end{array} \right). \]

At each iterative step, the current point \(x\) is replaced with \(Ax + b\), where the matrix \(A\) is always

\[ A = \left( \begin{array}{cc} 1/2 & 0 \\ 0 & 1/2 \end{array} \right) \]

and the vector \(b\) is chosen at random with equal probability from among the three vectors

\[ b = \left( \begin{array}{c} 0 \\ 0 \end{array} \right), \quad b = \left( \begin{array}{c} 1/2 \\ 0 \end{array} \right), \text{ and } b = \left( \begin{array}{c} 1/4 \\ \sqrt{3}/4 \end{array} \right). \]