

Chapter 2

Fibonacci Numbers

Fibonacci numbers introduce vectors, functions and recursion. An exercise introduces graphics user interfaces.

Leonardo Pisano Fibonacci was born around 1170 and died around 1250 in Pisa in what is now Italy. He traveled extensively in Europe and Northern Africa. He wrote several mathematical texts that, among other things, introduced Europe to the Hindu-Arabic notation for numbers. Even though his books had to be transcribed by hand, they were widely circulated. In his best known book, *Liber Abaci*, published in 1202, he posed the following problem:

A man put a pair of rabbits in a place surrounded on all sides by a wall. How many pairs of rabbits can be produced from that pair in a year if it is supposed that every month each pair begets a new pair which from the second month on becomes productive?

Today the solution to this problem is known as the *Fibonacci sequence*, or *Fibonacci numbers*. There is a small mathematical industry based on Fibonacci numbers. A search of the Internet for “Fibonacci” will find dozens of Web sites and hundreds of pages of material. There is even a Fibonacci Association that publishes a scholarly journal, the *Fibonacci Quarterly*.

If Fibonacci had not specified a month for the newborn pair to mature, he would not have a sequence named after him. The number of pairs would simply double each month. After n months there would be 2^n pairs of rabbits. That’s a lot of rabbits, but not distinctive mathematics.

Let f_n denote the number of pairs of rabbits after n months. The key fact is that the number of rabbits at the end of a month is the number at the beginning

of the month plus the number of births produced by the mature pairs:

$$f_n = f_{n-1} + f_{n-2}.$$

The initial conditions are that in the first month there is one pair of rabbits and in the second there are two pairs:

$$f_1 = 1, \quad f_2 = 2.$$

The following MATLAB function, stored in the M-file `fibonacci.m`, produces a vector containing the first `n` Fibonacci numbers.

```
function f = fibonacci(n)
% FIBONACCI Fibonacci sequence
% f = FIBONACCI(n) generates the first n Fibonacci numbers.
f = zeros(n,1);
f(1) = 1;
f(2) = 2;
for k = 3:n
    f(k) = f(k-1) + f(k-2);
end
```

With these initial conditions, the answer to Fibonacci's original question about the size of the rabbit population after one year is given by

```
fibonacci(12)
```

This produces

```
1
2
3
5
8
13
21
34
55
89
144
233
```

The answer is 233 pairs of rabbits. (It would be 4096 pairs if the number doubled every month for 12 months.)

Let's look carefully at `fibonacci.m`. It's a good example of how to create a MATLAB function. The first line is

```
function f = fibonacci(n)
```

The first word on the first line says `fibonacci.m` is a function, not a script. The remainder of the first line says this particular function produces one output result,

`f`, and takes one input argument, `n`. The name of the function specified on the first line is not actually used, because MATLAB looks for the name of the M-file, but it is common practice to have the two match. The next two lines are comments that provide the text displayed when you ask for `help`.

```
help fibonacci
```

produces

```
FIBONACCI Fibonacci sequence
f = FIBONACCI(n) generates the first n Fibonacci numbers.
```

The name of the function is in uppercase because historically MATLAB was case insensitive and ran on terminals with only a single font. The use of capital letters may be confusing to some first-time MATLAB users, but the convention persists. It is important to repeat the input and output arguments in these comments because the first line is not displayed when you ask for `help` on the function.

The next line

```
f = zeros(n,1);
```

creates an `n`-by-1 matrix containing all zeros and assigns it to `f`. In MATLAB, a matrix with only one column is a column vector and a matrix with only one row is a row vector.

The next two lines,

```
f(1) = 1;
f(2) = 2;
```

provide the initial conditions.

The last three lines are the `for` statement that does all the work.

```
for k = 3:n
    f(k) = f(k-1) + f(k-2);
end
```

We like to use three spaces to indent the body of `for` and `if` statements, but other people prefer two or four spaces, or a tab. You can also put the entire construction on one line if you provide a comma after the first clause.

This particular function looks a lot like functions in other programming languages. It produces a vector, but it does not use any of the MATLAB vector or matrix operations. We will see some of these operations soon.

Here is another Fibonacci function, `fibnum.m`. Its output is simply the n th Fibonacci number.

```
function f = fibnum(n)
% FIBNUM Fibonacci number.
% FIBNUM(n) generates the nth Fibonacci number.
if n <= 1
    f = 1;
```

```

else
    f = fibnum(n-1) + fibnum(n-2);
end

```

The statement

```

fibnum(12)

```

produces

```

ans =
    233

```

The `fibnum` function is *recursive*. In fact, the term *recursive* is used in both a mathematical and a computer science sense. The relationship $f_n = f_{n-1} + f_{n-2}$ is known as a *recursion relation* and a function that calls itself is a *recursive function*.

A recursive program is elegant, but expensive. You can measure execution time with `tic` and `toc`. Try

```

tic, fibnum(24), toc

```

Do *not* try

```

tic, fibnum(50), toc

```

Now compare the results produced by `goldfract(6)` and `fibonacci(7)`. The first contains the fraction 21/13 while the second ends with 13 and 21. This is not just a coincidence. The continued fraction is collapsed by repeating the statement

```

p = p + q;

```

while the Fibonacci numbers are generated by

```

f(k) = f(k-1) + f(k-2);

```

In fact, if we let ϕ_n denote the golden ratio continued fraction truncated at n terms, then

$$\frac{f_{n+1}}{f_n} = \phi_n.$$

In the infinite limit, the ratio of successive Fibonacci numbers approaches the golden ratio:

$$\lim_{n \rightarrow \infty} \frac{f_{n+1}}{f_n} = \phi.$$

To see this, compute 40 Fibonacci numbers.

```

n = 40;
f = fibonacci(n);

```

Then compute their ratios.

```
f(2:n)./f(1:n-1)
```

This takes the vector containing $f(2)$ through $f(n)$ and divides it, element by element, by the vector containing $f(1)$ through $f(n-1)$. The output begins with

```
2.000000000000000
1.500000000000000
1.666666666666667
1.600000000000000
1.625000000000000
1.61538461538462
1.61904761904762
1.61764705882353
1.61818181818182
```

and ends with

```
1.61803398874990
1.61803398874989
1.61803398874990
1.61803398874989
1.61803398874989
```

Do you see why we chose $n = 40$? Use the up arrow key on your keyboard to bring back the previous expression. Change it to

```
f(2:n)./f(1:n-1) - phi
```

and then press the Enter key. What is the value of the last element?

The population of Fibonacci's rabbit pen doesn't double every month; it is multiplied by the golden ratio every month.

It is possible to find a closed-form solution to the Fibonacci number recurrence relation. The key is to look for solutions of the form

$$f_n = c\rho^n$$

for some constants c and ρ . The recurrence relation

$$f_n = f_{n-1} + f_{n-2}$$

becomes

$$\rho^2 = \rho + 1.$$

We've seen this equation in the chapter on the Golden Ratio. There are two possible values of ρ , namely ϕ and $1 - \phi$. The general solution to the recurrence is

$$f_n = c_1\phi^n + c_2(1 - \phi)^n.$$

The constants c_1 and c_2 are determined by initial conditions, which are now conveniently written

$$\begin{aligned} f_0 &= c_1 + c_2 = 1, \\ f_1 &= c_1\phi + c_2(1 - \phi) = 1. \end{aligned}$$

One of the exercises asks you to use the MATLAB backslash operator to solve this 2-by-2 system of simultaneous linear equations, but it is may be easier to solve the system by hand:

$$c_1 = \frac{\phi}{2\phi - 1},$$

$$c_2 = -\frac{(1 - \phi)}{2\phi - 1}.$$

Inserting these in the general solution gives

$$f_n = \frac{1}{2\phi - 1}(\phi^{n+1} - (1 - \phi)^{n+1}).$$

This is an amazing equation. The right-hand side involves powers and quotients of irrational numbers, but the result is a sequence of integers. You can check this with MATLAB.

```
n = (1:40)';
f = round((phi.^(n+1) - (1-phi).^(n+1))/(2*phi-1))
```

The `.^` operator is an element-by-element power operator. It is not necessary to use `./` for the final division because `(2*phi-1)` is a scalar quantity. Roundoff error prevents the results from being exact integers, so the `round` function is used to convert floating point quantities to nearest integers. The resulting `f` begins with

```
f =
     1
     2
     3
     5
     8
    13
    21
    34
```

and ends with

```
5702887
9227465
14930352
24157817
39088169
63245986
102334155
165580141
```

Exercises

2.1 *Waltz*. Which Fibonacci numbers are even? Why?

2.2 *Backslash*. Use the MATLAB *backslash* operator to solve the 2-by-2 system of simultaneous linear equations

$$\begin{aligned}c_1 + c_2 &= 1, \\c_1\phi + c_2(1 - \phi) &= 1\end{aligned}$$

for c_1 and c_2 . You can find out about the backslash operator by taking a peek at the Linear Equations chapter, or with the commands

```
help \
help slash
```

2.3 *Logarithmic plot*. The statement

```
semilogy(fibonacci(18),'-o')
```

makes a logarithmic plot of Fibonacci numbers versus their index. The graph is close to a straight line. What is the slope of this line?

2.4 *Execution time*. How does the execution time of `fibnum(n)` depend on the execution time for `fibnum(n-1)` and `fibnum(n-2)`? Use this relationship to obtain an approximate formula for the execution time of `fibnum(n)` as a function of n . Estimate how long it would take your computer to compute `fibnum(50)`. Warning: You probably do not want to actually run `fibnum(50)`.

2.5 *Overflow*. What is the index of the largest Fibonacci number that can be represented *exactly* as a MATLAB double-precision quantity without roundoff error? What is the index of the largest Fibonacci number that can be represented *approximately* as a MATLAB double-precision quantity without overflowing?

2.6 *Slower maturity*. What if rabbits took two months to mature instead of one? The sequence would be defined by

$$\begin{aligned}g_1 &= 1, \\g_2 &= 1, \\g_3 &= 2\end{aligned}$$

and, for $n > 3$,

$$g_n = g_{n-1} + g_{n-3}$$

- Modify `fibonacci.m` and `fibnum.m` to compute this sequence.
- How many pairs of rabbits are there after 12 months?
- $g_n \approx \gamma^n$. What is γ ?
- Estimate how long it would take your computer to compute `fibnum(50)` with this modified `fibnum`.

2.7 *Mortality*. What if rabbits took one month to mature, but then died after six months. The sequence would be defined by

$$d_n = 0, \quad n \leq 0$$

$$d_1 = 1,$$

$$d_2 = 1$$

and, for $n > 2$,

$$d_n = d_{n-1} + d_{n-2} - d_{n-7}$$

- Modify `fibonacci.m` and `fibnum.m` to compute this sequence.
- How many pairs of rabbits are there after 12 months?
- $d_n \approx \delta^n$. What is δ ?
- Estimate how long it would take your computer to compute `fibnum(50)` with this modified `fibnum`.

2.8 *Rabbits*. Simulate Fibonacci's rabbit pen. If you are not interested in the details of MATLAB graphical user interfaces, then just experiment with `rabbits` without trying to understand the program itself. However, if you are interested in MATLAB Handle Graphics, then `rabbits` is a good starting point.

Let me explain the program. First of all, if you have a running MATLAB handy, enter

```
type rabbits
```

so you can see the entire program. The preamble of `rabbits` is

```
function rabbits(action)
% RABBITS Fibonacci's rabbit pen.
% Create immature pushbuttons that age after one click.
```

The first line indicates that the function has no output arguments and at most one input argument. The input argument, if it exists, is named `action`. The two comment lines provide the response to `help rabbits`.

If you call `rabbits` with no arguments, then `nargin`, which stands for “number of input arguments”, is zero and so you enter this section.

```
if nargin == 0
    clf
    shg
    uicontrol('style','text','fontsize',12,'fontweight','bold', ...
        'units','normal','position',[.47 .94 .06 .04])
    action = 'mature';
end
```

The commands `clf` and `shg` clear the current figure window and bring it forward if it is not already. The call to `uicontrol` creates a Handle Graphics `text` object and sets two of its font attributes. The coordinate system to be used within the

figure window is normalized, so the window is given unit height and width. The text object has its lower left corner at (.47,.94) and its size .06-by-.04, so it is centered horizontally near the top of the window and occupies a small fraction of the overall window. The final statement in this section sets the missing input argument named `action` to the string `'mature'`.

The function then switches on the value of `action`. If `action` is `'mature'`, which it is on the first entry, then the following section of `rabbits` is executed.

```
switch action
case 'mature'
    f = get(gcf,'position');
    p = [.90*f(3:4).*rand(1,2) 24 24];
    g = [.7 .7 .7];
    c = 'rabbits(''immature'')';
    uicontrol('style','pushbutton','position',p,'background',g, ...
        'callback',c,'enable','off');
    set(gcbo,'enable','off')
```

The variable `f` is assigned a four-element vector giving the location and size of the current figure window. Since `units` is not specified, the results are in pixels. The default figure window size, `f(3:4)`, is typically 560 pixels wide and 420 pixels high, but you can resize the window. Computers with large screens might have larger default figure windows. The variable `p` is assigned a four-element vector whose first two elements are random fractions of the window size and whose second two elements specify a 24-by-24 pixel portion of the window. The variable `g` is set to a 3-element color vector describing the shade of gray that results from having red, green and blue components equal to 70% of their full value. The variable `c` is assigned a string which, when subsequently evaluated, will result in a call to `rabbits` with an argument of `'immature'`. The `uicontrol` statement creates a new Handle Graphics object, this time a pushbutton with a gray background. The button is placed in position `p` within the figure window. Whenever the pushbutton is pushed, the callback string `c` will be evaluated, resulting in another call to `rabbits`. This is not exactly recursion, since `rabbits` does not call itself directly. Instead, `rabbits` posts a callback to the graphics system and eventually exits. Later, when a mouse click occurs, the graphics system calls `rabbits` again. The final statement disables `gcbo`, which stands for “get current callback object”. Disabled buttons created are visible, but do not respond to mouse clicks.

If, on the other hand, `action` is `'immature'`, then the following section of `rabbits` is executed.

```
case 'immature'
    R = imread('rabbit.jpg');
    g = get(gcbo,'position');
    p = [g(1:2)-12 48 48];
    c = 'rabbits(''mature'')';
    set(gcbo,'cdata',R,'position',p,'callback',c,'enable','off');
end
```

The first statement reads the jpeg image `rabbit.jpg` that is available in the `exm` toolbox. This size of this image is 47-by-45 pixels. The variable `g` is set to the position, in pixels within the figure, of `gcbo`, which is the button that initiated this callback. The variable `p` is set to a four-element vector obtained by moving `g` twelve pixels down and to the left and increasing its size to 48-by-48. The variable `c` is the callback string, `rabbits('mature')`. The final statement in this section sets the pushbutton's color data to be the image `R`, resets the button's position and callback string, and disables the button. In contrast to the `mature` section of program, this section does not create a new graphics object. It just modifies four of the properties of the button that was pushed.

The last three statements of `rabbits` are executed each time the function is called.

```
b = findobj(gcf,'style','pushbutton');
if ~any(any(char(get(b,'enable')) == 'n'))
    set(b,'background','w','enable','on')
end
set(findobj(gcf,'style','text'),'string',length(b))
```

The variable `b` is set to a vector of handles of all the pushbuttons that have been created. The `if` statement is pretty tricky. It checks the `enable` state of all the pushbuttons. If all of the buttons are disabled, they are all turned on. By this time you have probably forgotten that this exercise has something to do with Fibonacci numbers. The last line of `rabbits` counts the number of pushbuttons that have been created and updates the text object that was initialized a long time ago with the information needed to explain what this program is doing.