

Chapter 2

Fibonacci Numbers

Fibonacci numbers introduce vectors, functions and recursion.

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 puts 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*.

A simulation of Fibonacci’s problem is provided by our `exm` program `rabbits`. Just execute the command

```
rabbits
```

and click on the pushbuttons that show up. You will see something like figure 2.1.

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

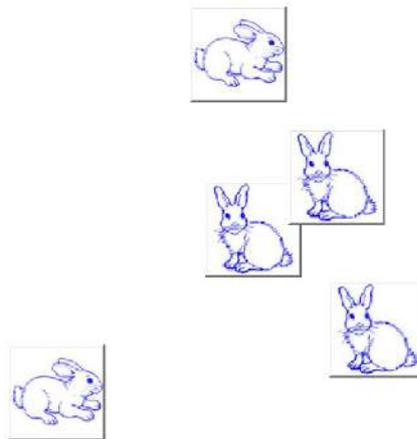


Figure 2.1. *Fibonacci's rabbits.*

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 a file `fibonacci.m` with a `.m` suffix, 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 file with a `.m` suffix that contains the function, 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. In mathematics, the relationship $f_n = f_{n-1} + f_{n-2}$ is a *recursion relation*. In computer science, 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
```

Fibonacci Meets Golden Ratio

The Golden Ratio ϕ can be expressed as an infinite continued fraction.

$$\phi = 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \dots}}}$$

To verify this claim, suppose we did not know the value of this fraction. Let

$$x = 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \dots}}}$$

We can see the first denominator is just another copy of x . In other words.

$$x = 1 + \frac{1}{x}$$

This immediately leads to

$$x^2 - x - 1 = 0$$

which is the defining quadratic equation for ϕ ,

Our `exm` function `goldfract` generates a MATLAB string that represents the first n terms of the Golden Ratio continued fraction. Here is the first section of code in `goldfract`.

```
p = '1';
for k = 2:n
    p = ['1 + 1/(' p ')'];
end
display(p)
```

We start with a single '1', which corresponds to $n = 1$. We then repeatedly make the current string the denominator in a longer string.

Here is the output from `goldfract(n)` when $n = 7$.

```
1 + 1/(1 + 1/(1 + 1/(1 + 1/(1 + 1/(1 + 1/(1))))))
```

You can see that there are $n-1$ plus signs and $n-1$ pairs of matching parentheses.

Let ϕ_n denote the continued fraction truncated after n terms. ϕ_n is a rational approximation to ϕ . Let's express ϕ_n as a conventional fraction, the ratio of two integers

$$\phi_n = \frac{p_n}{q_n}$$

```
p = 1;
q = 0;
for k = 2:n
    t = p;
    p = p + q;
    q = t;
end
```

Now compare the results produced by `goldfract(7)` 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 for the Golden Ratio 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

$$\phi_n = \frac{f_n}{f_{n-1}}$$

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

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

To see this, compute 40 Fibonacci numbers.

```
n = 40;
f = fibonacci(n);
```

Then compute their ratios.

```
r = 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 $n = 40$? Compute

```

phi = (1+sqrt(5))/2
r - phi

```

What is the value of the last element?

The first few of these ratios can also be used to illustrate the rational output format.

```

format rat
r(1:10)
ans =
    2
    3/2
    5/3
    8/5
    13/8
    21/13
    34/21
    55/34
    89/55

```

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

An Analytic Expression

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

$$c\rho^n = c\rho^{n-1} + c\rho^{n-2}$$

Dividing both sides by $c\rho^{n-2}$ gives

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

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 may be easier to solve the system by hand:

$$\begin{aligned} c_1 &= \frac{\phi}{2\phi - 1}, \\ c_2 &= -\frac{(1 - \phi)}{2\phi - 1}. \end{aligned}$$

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

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

Recap

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

%% Related EXM Programs
%
%   fibonacci.m
%   fibnum.m
%   rabbits.m

%% Functions
% Save in file sqrt1px.m
%
%   function y = sqrt1px(x)
%   % SQRT1PX Sample function.
%   % Usage: y = sqrt1px(x)
%
%       y = sqrt(1+x);

%% Create vector
n = 8;
```

```
f = zeros(1,n)
t = 1:n
s = [1 2 3 5 8 13 21 34]

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

%% Recursion
% function f = fibnum(n)
% if n <= 1
%     f = 1;
% else
%     f = fibnum(n-1) + fibnum(n-2);
% end

%% Tic and Toc
format short
tic
    fibnum(24);
toc

%% Element-by-element array operations
f = fibonacci(5)'
fpf = f+f
ftf = f.*f
ff = f.^2
ffdf = ff./f
cosfpi = cos(f*pi)
even = (mod(f,2) == 0)
format rat
r = f(2:5)./f(1:4)

%% Strings
hello_world
```

Exercises

2.1 *Rabbits*. Explain what our `rabbits` simulation demonstrates. What do the different figures and colors on the pushbuttons signify?

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

2.3 *Primes*. Use the MATLAB function `isprime` to discover which of the first 40 Fibonacci numbers are prime. You do not need to use a `for` loop. Instead, check out

```
help isprime
help logical
```

2.4 *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
```

2.5 *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?

2.6 *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)`.

2.7 *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?

2.8 *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`.

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

$$\begin{aligned}d_n &= 0, \quad n \leq 0 \\d_1 &= 1, \\d_2 &= 1\end{aligned}$$

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`.

2.10 *Hello World*. Programming languages are traditionally introduced by the phrase "hello world". An script in `exm` that illustrates some features in MATLAB is available with

```
hello_world
```

Explain what each of the functions and commands in `hello_world` do.

2.11 *Fibonacci power series*. The Fibonacci numbers, f_n , can be used as coefficients in a power series defining a function of x .

$$\begin{aligned}F(x) &= \sum_{n=1}^{\infty} f_n x^n \\&= x + 2x^2 + 3x^3 + 5x^4 + 8x^5 + 13x^6 + \dots\end{aligned}$$

Our function `fibfun1` is a first attempt at a program to compute this series. It simply involves adding an accumulating sum to `fibonacci.m`. The header of `fibfun1.m` includes the help entries.

```
function [y,k] = fibfun1(x)
% FIBFUN1 Power series with Fibonacci coefficients.
% y = fibfun1(x) = sum(f(k)*x.^k).
% [y,k] = fibfun1(x) also gives the number of terms required.
```

The first section of code initializes the variables to be used. The value of `n` is the index where the Fibonacci numbers overflow.

```
\excise
\emph{Fibonacci power series}.
The Fibonacci numbers,  $f_n$ , can be used as coefficients in a
power series defining a function of  $x$ .
\begin{eqnarray*}
F(x) & = & \sum_{n = 1}^{\infty} f_n x^n \\
& = & x + 2 x^2 + 3 x^3 + 5 x^4 + 8 x^5 + 13 x^6 + \dots
\end{eqnarray*}
Our function #fibfun1# is a first attempt at a program to
compute this series. It simply involves adding an accumulating
sum to #fibonacci.m#.
The header of #fibfun1.m# includes the help entries.
\begin{verbatim}
function [y,k] = fibfun1(x)
% FIBFUN1 Power series with Fibonacci coefficients.
% y = fibfun1(x) = sum(f(k)*x.^k).
% [y,k] = fibfun1(x) also gives the number of terms required.
```

The first section of code initializes the variables to be used. The value of `n` is the index where the Fibonacci numbers overflow.

```
n = 1476;
f = zeros(n,1);
f(1) = 1;
f(2) = 2;
y = f(1)*x + f(2)*x.^2;
t = 0;
```

The main body of `fibfun1` implements the Fibonacci recurrence and includes a test for early termination of the loop.

```
for k = 3:n
f(k) = f(k-1) + f(k-2);
y = y + f(k)*x.^k;
if y == t
return
end
```

```

    t = y;
end

```

There are several objections to `fibfun1`. The coefficient array of size 1476 is not actually necessary. The repeated computation of powers, x^k , is inefficient because once some power of x has been computed, the next power can be obtained with one multiplication. When the series converges the coefficients $f(k)$ increase in size, but the powers x^k decrease in size more rapidly. The terms $f(k)*x^k$ approach zero, but huge $f(k)$ prevent their computation.

A more efficient and accurate approach involves combining the computation of the Fibonacci recurrence and the powers of x . Let

$$p_k = f_k x^k$$

Then, since

$$f_{k+1} x^{k+1} = f_k x^k + f_{k-1} x^{k-1}$$

the terms p_k satisfy

$$\begin{aligned} p_{k+1} &= p_k x + p_{k-1} x^2 \\ &= x(p_k + x p_{k-1}) \end{aligned}$$

This is the basis for our function `fibfun2`. The header is essentially the same as `fibfun1`

```

function [yk,k] = fibfun2(x)
% FIBFUN2 Power series with Fibonacci coefficients.
% y = fibfun2(x) = sum(f(k)*x.^k).
% [y,k] = fibfun2(x) also gives the number of terms required.

```

The initialization.

```

    pkm1 = x;
    pk = 2*x.^2;
    ykm1 = x;
    yk = 2*x.^2 + x;
    k = 0;

```

And the core.

```

    while any(abs(yk-ykm1) > 2*eps(yk))
        pkp1 = x.*(pk + x.*pkm1);
        pkm1 = pk;
        pk = pkp1;
        ykm1 = yk;
        yk = yk + pk;
        k = k+1;
    end

```

There is no array of coefficients. Only three of the p_k terms are required for each step. The power function \wedge is not necessary. Computation of the powers is incorporated in the recurrence. Consequently, `fibfun2` is both more efficient and more accurate than `fibfun1`.

But there is an even better way to evaluate this particular series. It is possible to find an analytic expression for the infinite sum.

$$\begin{aligned}
 F(x) &= \sum_{n=1}^{\infty} f_n x^n \\
 &= x + 2x^2 + 3x^3 + 5x^4 + 8x^5 + \dots \\
 &= x + (1+1)x^2 + (2+1)x^3 + (3+2)x^4 + (5+3)x^5 + \dots \\
 &= x + x^2 + x(x + 2x^2 + 3x^3 + 5x^4 + \dots) + x^2(x + 2x^2 + 3x^3 + \dots) \\
 &= x + x^2 + xF(x) + x^2F(x)
 \end{aligned}$$

So

$$(1 - x - x^2)F(x) = x + x^2$$

Finally

$$F(x) = \frac{x + x^2}{1 - x - x^2}$$

It is not even necessary to have a `.m` file. A one-liner does the job.

```
fibfun3 = @(x) (x + x.^2)./(1 - x - x.^2)
```

Compare these three `fibfun`'s.