Chapter 1

Iteration

Iteration is a key element in much of technical computation. Examples involving the Golden Ratio introduce the MATLAB *assignment statement,* `for` *and* `while` *loops, and the* `plot` *function.*

Start by picking a number, any number. Enter it into MATLAB by typing

`x =` *your number*

This is a MATLAB *assignment statement*. The number you chose is stored in the *variable* `x` for later use. For example, if you start with

```
x = 3
```

MATLAB responds with

```
x =
     3
```

Next, enter this statement

```
x = sqrt(1 + x)
```

The abbreviation `sqrt` is the MATLAB name for the square root function. The quantity on the right, $\sqrt{1+x}$, is computed and the result stored back in the variable `x`, overriding the previous value of `x`.

Somewhere on your computer keyboard, probably in the lower right corner, you should be able to find four arrow keys. These are the *command line editing* keys. The up-arrow key allows you to recall earlier commands, including commands from

October 4, 2011

previous sessions, and the other arrows keys allow you to revise these commands. Use the up-arrow key, followed by the enter or return key, to iterate, or repeatedly execute, this statement:

```
x = sqrt(1 + x)
```

Here is what you get when you start with `x = 3`.

```
x =
     3
x =
     2
x =
    1.7321
x =
    1.6529
x =
    1.6288
x =
    1.6213
x =
    1.6191
x =
    1.6184
x =
    1.6181
x =
    1.6181
x =
    1.6180
x =
    1.6180
```

These values are 3, $\sqrt{1+3}$, $\sqrt{1+\sqrt{1+3}}$, $\sqrt{1+\sqrt{1+\sqrt{1+3}}}$, and so on. After 10 steps, the value printed remains constant at `1.6180`. Try several other starting values. Try it on a calculator if you have one. You should find that no matter where you start, you will always reach `1.6180` in about ten steps. (Maybe a few more will be required if you have a very large starting value.)

MATLAB is doing these computations to accuracy of about 16 decimal digits, but is displaying only five. You can see more digits by first entering

```
format long
```

and repeating the experiment. Here are the beginning and end of 30 steps starting at $x = 3$.

```
x =
     3
```

```
x =
     2
x =
   1.732050807568877
x =
   1.652891650281070

    ....

x =
   1.618033988749897
x =
   1.618033988749895
x =
   1.618033988749895
```

After about thirty or so steps, the value that is printed doesn't change any more. You have computed one of the most famous numbers in mathematics, ϕ, the *Golden Ratio*.

In MATLAB, and most other programming languages, the equals sign is the assignment operator. It says compute the value on the right and store it in the variable on the left. So, the statement

```
x = sqrt(1 + x)
```

takes the current value of `x`, computes `sqrt(1 + x)`, and stores the result back in `x`.

In mathematics, the equals sign has a different meaning.

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

is an *equation*. A solution to such an equation is known as a *fixed point*. (Be careful not to confuse the mathematical usage of *fixed point* with the computer arithmetic usage of *fixed point*.)

The function $f(x) = \sqrt{1 + x}$ has exactly one fixed point. The best way to find the value of the fixed point is to avoid computers all together and solve the equation using the quadratic formula. Take a look at the hand calculation shown in figure 1.1. The positive root of the quadratic equation is the Golden Ratio.

$$\phi = \frac{1 + \sqrt{5}}{2}.$$

You can have MATLAB compute ϕ directly using the statement

```
phi = (1 + sqrt(5))/2
```

With `format long`, this produces the same value we obtained with the fixed point iteration,

```
phi =
   1.618033988749895
```

$$X = \sqrt{1+X}$$
$$X^2 = 1 + X$$
$$X^2 - X - 1 = 0$$
$$X = \frac{1 \pm \sqrt{1+4}}{2}$$
$$\varphi = \frac{1+\sqrt{5}}{2}$$

Figure 1.1. *Compute the fixed point by hand.*

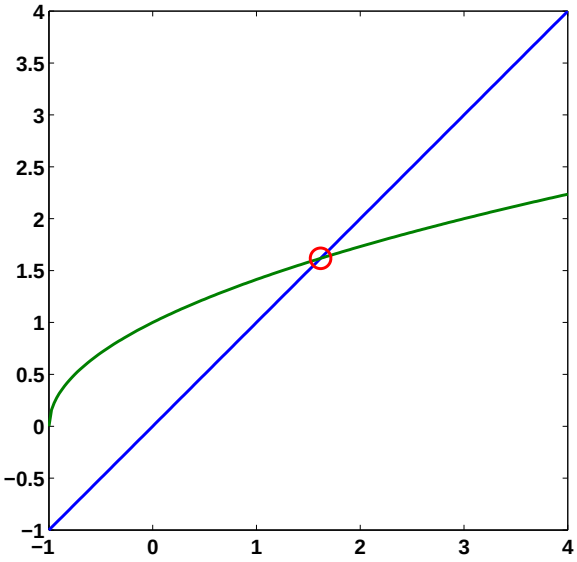

Figure 1.2. *A fixed point at $\phi = 1.6180$.*

Figure 1.2 is our first example of MATLAB graphics. It shows the intersection of the graphs of $y = x$ and $y = \sqrt{1+x}$. The statement

```
x = -1:.02:4;
```

generates a vector `x` containing the numbers from -1 to 4 in steps of .02. The statements

```
y1 = x;
y2 = sqrt(1+x);
plot(x,y1,'-',x,y2,'-',phi,phi,'o')
```

produce a figure that has three components. The first two components are graphs of x and $\sqrt{1+x}$. The `'-'` argument tells the `plot` function to draw solid lines. The last component in the plot is a single point with both coordinates equal to ϕ. The `'o'` tells the `plot` function to draw a circle.

The MATLAB `plot` function has many variations, including specifying other colors and line types. You can see some of the possibilities with

```
help plot
```

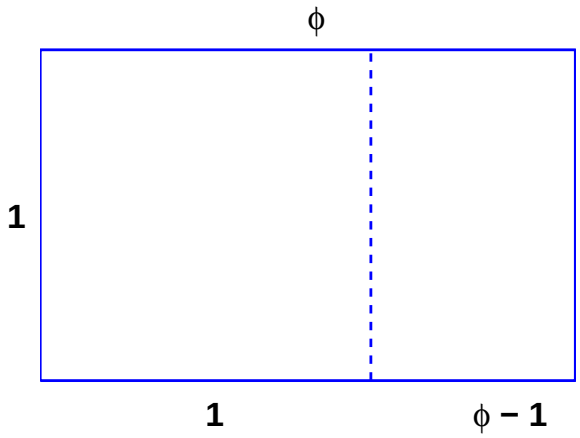

Figure 1.3. *The golden rectangle.*

The Golden Ratio shows up in many places in mathematics; we'll see several in this book. The Golden Ratio gets its name from the golden rectangle, shown in figure 1.3. The golden rectangle has the property that removing a square leaves a smaller rectangle with the same shape. Equating the aspect ratios of the rectangles gives a defining equation for ϕ:

$$\frac{1}{\phi} = \frac{\phi - 1}{1}.$$

Multiplying both sides of this equation by ϕ produces the same quadratic polynomial equation that we obtained from our fixed point iteration.

$$\phi^2 - \phi - 1 = 0.$$

The up-arrow key is a convenient way to repeatedly execute a single statement, or several statements, separated by commas or semicolons, on a single line. Two more powerful constructs are the `for` loop and the `while` loop. A `for` loop executes a block of code a prescribed number of times.

```
x = 3
for k = 1:31
   x = sqrt(1 + x)
end
```

produces 32 lines of output, one from the initial statement and one more each time through the loop.

A `while` loop executes a block of code an unknown number of times. Termination is controlled by a logical expression, which evaluates to `true` or `false`. Here is the simplest `while` loop for our fixed point iteration.

```
x = 3
while x ~= sqrt(1+x)
   x = sqrt(1+x)
end
```

This produces the same 32 lines of output as the `for` loop. However, this code is open to criticism for two reasons. The first possible criticism involves the termination condition. The expression `x ~= sqrt(1+x)` is the MATLAB way of writing $x \neq \sqrt{1+x}$. With exact arithmetic, `x` would never be exactly equal to `sqrt(1+x)`, the condition would always be true, and the loop would run forever. However, like most technical computing environments, MATLAB does not do arithmetic exactly. In order to economize on both computer time and computer memory, MATLAB uses *floating point* arithmetic. Eventually our program produces a value of `x` for which the floating point numbers `x` and `sqrt(1+x)` are exactly equal and the loop terminates. Expecting exact equality of two floating point numbers is a delicate matter. It works OK in this particular situation, but may not work with more complicated computations.

The second possible criticism of our simple `while` loop is that it is inefficient. It evaluates `sqrt(1+x)` twice each time through the loop. Here is a more complicated version of the `while` loop that avoids both criticisms.

```
x = 3
y = 0;
while abs(x-y) > eps(x)
   y = x;
   x = sqrt(1+x)
end
```

The semicolons at the ends of the assignment statements involving `y` indicate that no printed output should result. The quantity `eps(x)`, is the spacing of the floating point numbers near `x`. Mathematically, the Greek letter ϵ, or *epsilon*, often represents a "small" quantity. This version of the loop requires only one square root calculation per iteration, but that is overshadowed by the added complexity of the code. Both `while` loops require about the same execution time. In this situation, I prefer the first `while` loop because it is easier to read and understand.

Help and Doc

MATLAB has extensive on-line documentation. Statements like

```
help sqrt
help for
```

provide brief descriptions of commands and functions. Statements like

```
doc sqrt
doc for
```

provide more extensive documentation in a separate window.

One obscure, but very important, `help` entry is about the various punctuation marks and special characters used by MATLAB. Take a look now at

```
help punct
doc punct
```

You will probably want to return to this information as you learn more about MATLAB.

Numbers

Numbers are formed from the digits `0` through `9`, an optional decimal point, a leading `+` or `-` sign, an optional `e` followed by an integer for a power of 10 scaling, and an optional `i` or `j` for the imaginary part of a complex number. MATLAB also knows the value of π. Here are some examples of numbers.

```
42
9.6397238
6.0221415e23
-3+4i
pi
```

Assignment statements and names

A simple assignment statement consists of a name, an = sign, and a number. The names of variables, functions and commands are formed by a letter, followed by any number of upper and lower case letters, digits and underscores. Single character names, like `x` and `N`, and anglicized Greek letters, like `pi` and `phi`, are often used to reflect underlying mathematical notation. Non-mathematical programs usually employ long variable names. Underscores and a convention known as camel casing are used to create variable names out of several words.

```
x = 42
phi = (1+sqrt(5))/2
Avogadros_constant = 6.0221415e23
camelCaseComplexNumber = -3+4i
```

Expressions

Power is denoted by `^` and has precedence over all other arithmetic operations. Multiplication and division are denoted by `*`, `/`, and `\` and have precedence over addition and subtraction, Addition and subtraction are denoted by `+` and `-` and

have lowest precedence. Operations with equal precedence are evaluated left to right. Parentheses delineate subexpressions that are evaluated first. Blanks help readability, but have no effect on precedence.

All of the following expressions have the same value. If you don't already recognize this value, you can ask Google about its importance in popular culture.

```
3*4 + 5*6
3 * 4+5 * 6
2*(3 + 4)*3
-2^4 + 10*29/5
3\126
52-8-2
```

Recap

```
%% Iteration Chapter Recap
% This is an executable program that illustrates the statements
% introduced in the Iteration chapter of "Experiments in MATLAB".
% You can run it by entering the command
%
%    iteration_recap
%
% Better yet, enter
%
%    edit iteration_recap
%
% and run the program cell-by-cell by simultaneously
% pressing the Ctrl-Shift-Enter keys.
%
% Enter
%
%   publish iteration_recap
%
% to see a formatted report.

%% Help and Documentation
%   help punct
%   doc punct

%% Format
    format short
    100/81
    format long
    100/81

    format short
```

```
    format compact

%% Names and assignment statements
    x = 42
    phi = (1+sqrt(5))/2
    Avogadros_constant = 6.0221415e23
    camelCaseComplexNumber = -3+4i

%% Expressions
   3*4 + 5*6
   3 * 4+5 * 6
   2*(3 + 4)*3
   -2^4 + 10*29/5
   3\126
   52-8-2

%% Iteration
% Use the up-arrow key to repeatedly execute
   x = sqrt(1+x)
   x = sqrt(1+x)
   x = sqrt(1+x)
   x = sqrt(1+x)

%% For loop
   x = 42
   for k = 1:12
      x = sqrt(1+x);
      disp(x)
   end

%% While loop
   x = 42;
   k = 1;
   while abs(x-sqrt(1+x)) > 5e-5
      x = sqrt(1+x);
      k = k+1;
   end
   k

%% Vector and colon operator
   k = 1:12
   x = (0.0: 0.1: 1.00)'

%% Plot
   x = -pi: pi/256: pi;
   y = tan(sin(x)) - sin(tan(x));
```

```
   z = 1 + tan(1);
   plot(x,y,'-', pi/2,z,'ro')
   xlabel('x')
   ylabel('y')
   title('tan(sin(x)) - sin(tan(x))')

%% Golden Spiral
   golden_spiral(4)
```

Exercises

1.1 *Expressions.* Use MATLAB to evaluate each of these mathematical expressions.

43^2	-3^4	$\sin 1$
$4^{(3^2)}$	$(-3)^4$	$\sin 1^\circ$
$(4^3)^2$	$\sqrt[4]{-3}$	$\sin \frac{\pi}{3}$
$\sqrt[4]{32}$	$-2^{-4/3}$	$(\arcsin 1)/\pi$

You can get started with

```
help ^
help sin
```

1.2 *Temperature conversion.*
(a) Write a MATLAB statement that converts temperature in Fahrenheit, `f`, to Celsius, `c`.

`c =` *something involving* `f`

(b) Write a MATLAB statement that converts temperature in Celsius, `c`, to Fahrenheit, `f`.

`f =` *something involving* `c`

1.3 *Barn-megaparsec.* A *barn* is a unit of area employed by high energy physicists. Nuclear scattering experiments try to "hit the side of a barn". A *parsec* is a unit of length employed by astronomers. A star at a distance of one parsec exhibits a trigonometric parallax of one arcsecond as the Earth orbits the Sun. A *barn-megaparsec* is therefore a unit of volume – a very long skinny volume.

A barn is 10^{-28} square meters.
A megaparsec is 10^6 parsecs.
A parsec is 3.262 light-years.
A light-year is $9.461 \cdot 10^{15}$ meters.
A cubic meter is 10^6 milliliters.
A milliliter is $\frac{1}{5}$ teaspoon.

Express one barn-megaparsec in teaspoons. In MATLAB, the letter `e` can be used to denote a power of 10 exponent, so $9.461 \cdot 10^{15}$ can be written `9.461e15`.

1.4 *Complex numbers.* What happens if you start with a large negative value of `x` and repeatedly iterate

```
x = sqrt(1 + x)
```

1.5 *Comparison.* Which is larger, π^ϕ or ϕ^π?

1.6 *Solving equations.* The best way to solve

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

or

$$x^2 = 1 + x$$

is to avoid computers all together and just do it yourself by hand. But, of course, MATLAB and most other mathematical software systems can easily solve such equations. Here are several possible ways to do it with MATLAB. Start with

```
format long
phi = (1 + sqrt(5))/2
```

Then, for each method, explain what is going on and how the resulting `x` differs from `phi` and the other `x`'s.

```
% roots
help roots
x1 = roots([1 -1 -1])

% fsolve
help fsolve
f = @(x) x-sqrt(1+x)
p = @(x) x^2-x-1
x2 = fsolve(f, 1)
x3 = fsolve(f, -1)
x4 = fsolve(p, 1)
x5 = fsolve(p, -1)

% solve (requires Symbolic Toolbox or Student Version)
help solve
help syms
syms x
x6 = solve('x-sqrt(1+x)=0')
x7 = solve(x^2-x-1)
```

1.7 *Symbolic solution.* If you have the Symbolic Toolbox or Student Version, explain what the following program does.

```
x = sym('x')
length(char(x))
for k = 1:10
   x = sqrt(1+x)
   length(char(x))
end
```

1.8 *Fixed points.* Verify that the Golden Ratio is a fixed point of each of the following equations.

$$\phi = \frac{1}{\phi - 1}$$

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

Use each of the equations as the basis for a fixed point iteration to compute ϕ. Do the iterations converge?

1.9 *Another iteration.* Before you run the following program, predict what it will do. Then run it.

```
x = 3
k = 1
format long
while x ~= sqrt(1+x^2)
   x = sqrt(1+x^2)
   k = k+1
end
```

1.10 *Another fixed point.* Solve this equation by hand.

$$x = \frac{1}{\sqrt{1 + x^2}}$$

How many iterations does the following program require? How is the final value of x related to the Golden Ratio ϕ?

```
x = 3
k = 1
format long
while x ~= 1/sqrt(1+x^2)
   x = 1/sqrt(1+x^2)
   k = k+1
end
```

1.11 *cos(x).* Find the numerical solution of the equation

$$x = \cos x$$

in the interval $[0, \frac{\pi}{2}]$, shown in figure 1.4.

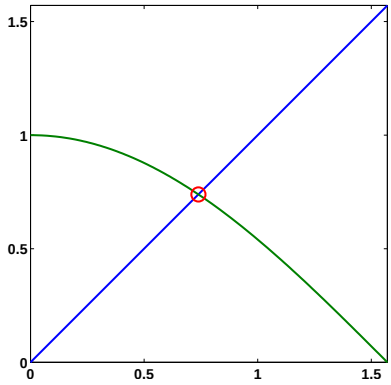

Figure 1.4. *Fixed point of x = cos(x).*

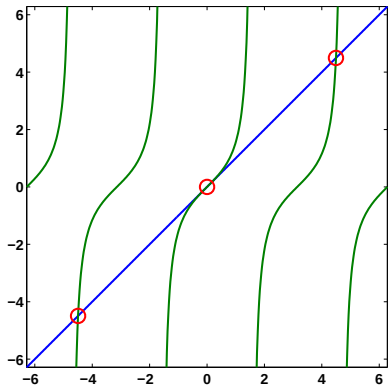

Figure 1.5. *Three fixed points of x = tan(x)*

1.12 *tan(x).* Figure 1.5 shows three of the many solutions to the equation

$$x = \tan x$$

One of the solutions is $x = 0$. The other two in the plot are near $x = \pm 4.5$. If we did a plot over a large range, we would see solutions in each of the intervals $[(n - \frac{1}{2})\pi, (n + \frac{1}{2})\pi]$ for integer n.
(a) Does this compute a fixed point?

```
x = 4.5
```

```
for k = 1:30
   x = tan(x)
end
```

(b) Does this compute a fixed point? Why is the " `+ pi`" necessary?

```
x = pi
while abs(x - tan(x)) > eps(x)
   x = atan(x) + pi
end
```

1.13 *Summation.* Write a mathematical expression for the quantity approximated by this program.

```
s = 0;
t = Inf;
n = 0;
while s ~= t
   n = n+1;
   t = s;
   s = s + 1/n^4;
end
s
```

1.14 *Why.* The first version of MATLAB written in the late 1970's, had `who`, `what`, `which`, and `where` commands. So it seemed natural to add a `why` command. Check out today's `why` command with

```
why
help why
for k = 1:40, why, end
type why
edit why
```

As the `help` entry says, please embellish or modify the `why` function to suit your own tastes.

1.15 *Wiggles.* A glimpse at MATLAB plotting capabilities is provided by the function

```
f = @(x) tan(sin(x)) - sin(tan(x))
```

This uses the '`@`' sign to introduce a simple function. You can learn more about the '`@`' sign with `help function_handle`.

Figure 1.6 shows the output from the statement

```
ezplot(f,[-pi,pi])
```

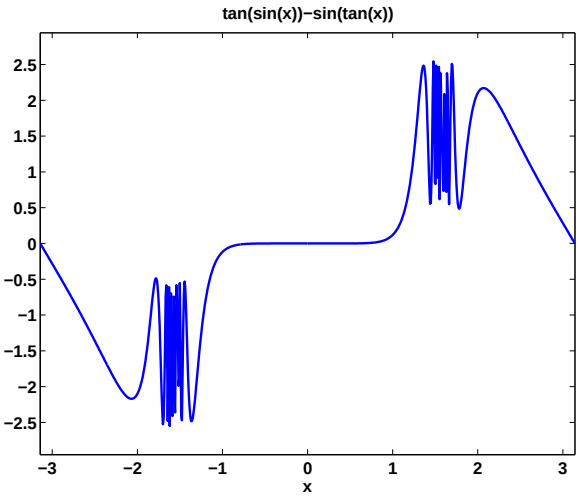

Figure 1.6. *A wiggly function.*

(The function name `ezplot` is intended to be pronounced "Easy Plot". This pun doesn't work if you learned to pronounce "z" as "zed".) You can see that the function is very flat near $x = 0$, oscillates infinitely often near $x = \pm\pi/2$ and is nearly linear near $x = \pm\pi$.

You can get more control over the plot with code like this.

```
x = -pi:pi/256:pi;
y = f(x);
plot(x,y)
xlabel('x')
ylabel('y')
title('A wiggly function')
axis([-pi pi -2.8 2.8])
set(gca,'xtick',pi*(-3:1/2:3))
```

(a) What is the effect of various values of `n` in the following code?

```
x = pi*(-2:1/n:2);
comet(x,f(x))
```

(b) This function is bounded. A numeric value near its maximum can be found with

```
max(y)
```

What is its analytic maximum? (To be precise, I should ask "What is the function's *supremum*?")

1.16 *Graphics.* We use a lot of computer graphics in this book, but studying MATLAB graphics programming is not our primary goal. However, if you are curious, the

script that produces figure 1.3 is `goldrect.m`. Modify this program to produce a graphic that compares the Golden Rectangle with TV screens having aspect ratios 4:3 and 16:9.

1.17 *Golden Spiral*

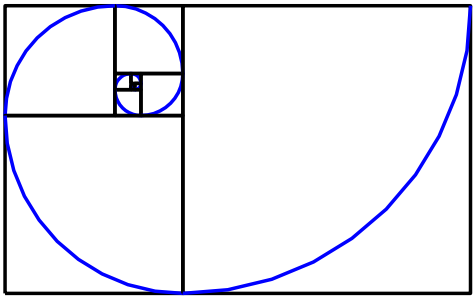

Figure 1.7. *A spiral formed from golden rectangles and inscribed quarter circles.*

Our program `golden_spiral` displays an ever-expanding sequence of golden rectangles with inscribed quarter circles. Check it out.