

Chapter 1

Iteration

An investigation of fixed point iterations introduces the assignment statement, for and while loops, the plot function, and the Golden Ratio.

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`. Now, repeatedly execute the statement by using the up-arrow key, followed by the enter or return key. Here is what you get when you start with `x = 3`.

```
x =  
_____
```

Copyright © 2009 Cleve Moler
MATLAB[®] is a registered trademark of The MathWorks, Inc.[™]
August 8, 2009

```

      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. Starting with numbers less than -1 makes things more complicated, but you might want to try it anyway.)

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*.)

$$\begin{aligned}
 X &= \sqrt{1+X} \\
 X^2 &= 1+X \\
 X^2 - X - 1 &= 0 \\
 X &= \frac{1 \pm \sqrt{1+4}}{2} \\
 \phi &= \frac{1 + \sqrt{5}}{2}
 \end{aligned}$$

Figure 1.1. Compute the fixed point by hand.

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 use the quadratic formula. Take a look at the hand calculation shown in figure 1.1. Squaring the equation introduces a second, negative solution. The positive root 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
```

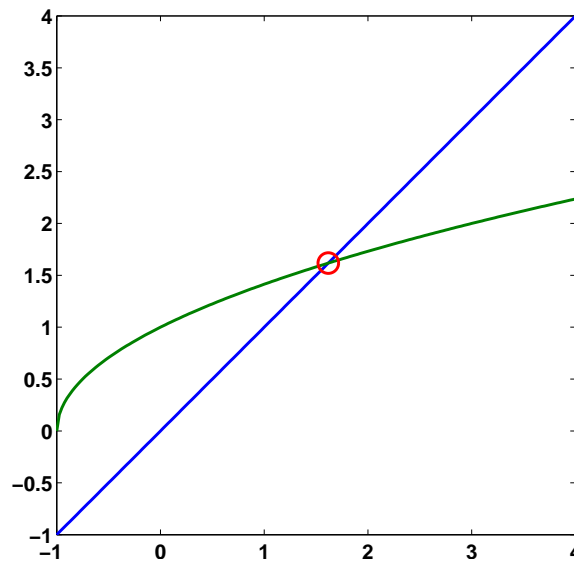


Figure 1.2. A fixed point at $\phi = 1.6180$.

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:.025:4;
```

generates a vector `x` containing the numbers from -1 to 4 in steps of .025. Then the statement

```
plot(x,x,'-',x,sqrt(1+x),'-',phi,phi,'o')
```

produces a figure that has three components. The first two components are graphs of $y = x$ and $y = \sqrt{1+x}$. The `'-'` argument tells the `plot` function to draw solid lines. The last component in the plot is a single point with both x and y -coordinates equal to ϕ . The `'o'` tells the `plot` function to draw a circle. The `plot` function has many variations, including specifying other colors and line types. You can see some of the possibilities with

```
help plot
```

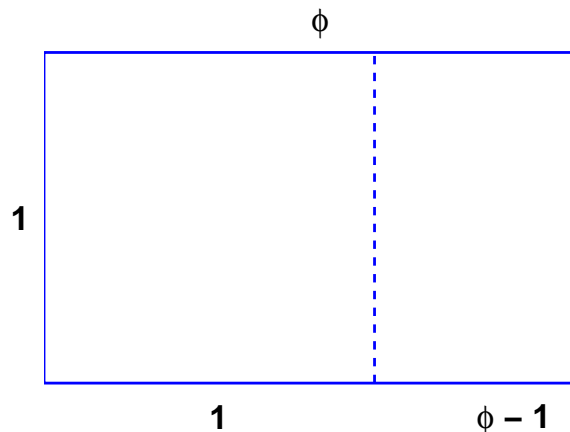


Figure 1.3. *The golden rectangle.*

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 with these fixed point iterations, 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 = Inf;
while abs(x-y) > eps(x)
    y = x;
    x = sqrt(1+x)
end

```

The quantity `Inf` is MATLAB abbreviation of *infinity*. 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.

Exercises

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

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

You can get started with

```
help ^
help sin
```

1.2 Temperature conversation.

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

$c = \textit{something involving } f$

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

$f = \textit{something involving } c$

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

1.5 *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 ϕ 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)

```

1.6 *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?

1.7 *Continued fraction.* Verify that ϕ can be expressed as this infinite continued fraction.

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

If this infinite expression is truncated after n terms, the result is a rational approximation to ϕ . For example, for $n = 4$, the approximation is

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

This is this the same as

$$\phi \approx \frac{5}{3}$$

Verify that this MATLAB program computes two integers p and q whose ratio p/q is a conventional fraction with the same value as the continued fraction truncated after n terms.

```
n = ...
p = 1;
q = 0;
for k = 1:n
    s = p;
    p = p + q;
    q = s;
end
p/q
```

As n is increased, how well does p/q approximate ϕ ?

1.8 Find the numerical solution of the equation

$$x = \cos x$$

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

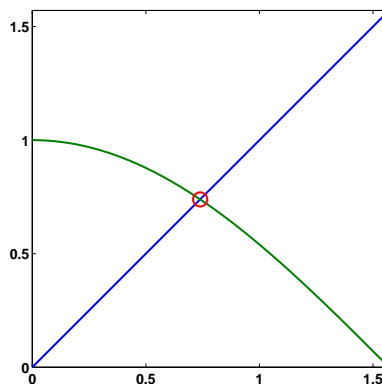


Figure 1.4. Fixed point of $x = \cos(x)$.

1.9 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 solution 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?

$$x = 4.5$$

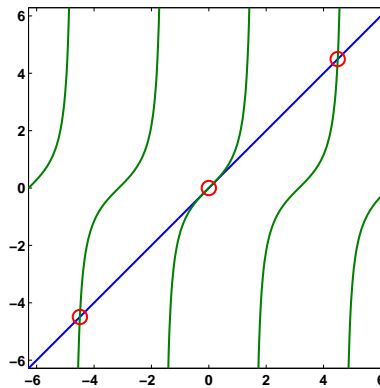


Figure 1.5. Three fixed points of $x = \tan(x)$

```
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
```

1.10 *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
```

1.11 *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.

1.12 *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.

1.13 *Fixed point iterations.*

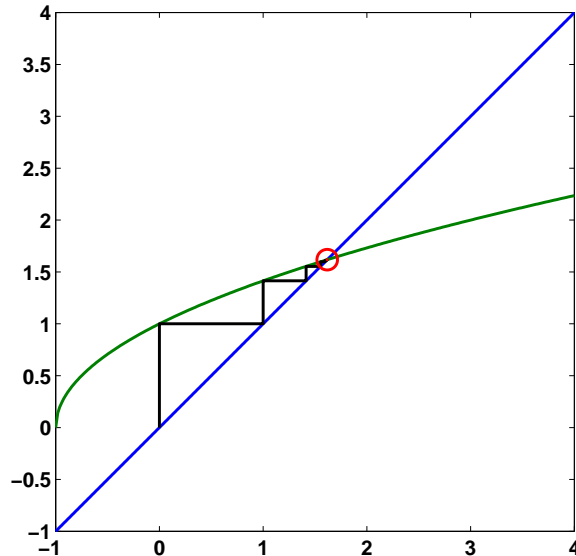


Figure 1.6. *The fixed point iteration.*

Let's examine fixed point iterations more carefully. Figure 1.6 is produced by entering the statement

```
fixedpoint
```

This executes the MATLAB program stored in the file `exm/fixedpoint.m`. In the figure, the straight blue line is the graph of $y = x$. The curved green line is the graph of the function $f(x) = \sqrt{1 + x}$. The staircase black line shows the steps of our iteration alternating back and forth between the two graphs. All three lines converge to a point in the red circle. Click on the magnifying glass with the plus sign in the figure header. Now you can zoom in with mouse clicks on the red circle. As you get closer and closer to the intersection point, the scale changes and the

green line gets straighter, but the overall picture doesn't change otherwise. (If you're reading the printed edition of this book, you can't zoom in on the graphic; you'll have to actually run MATLAB to get a live picture.)

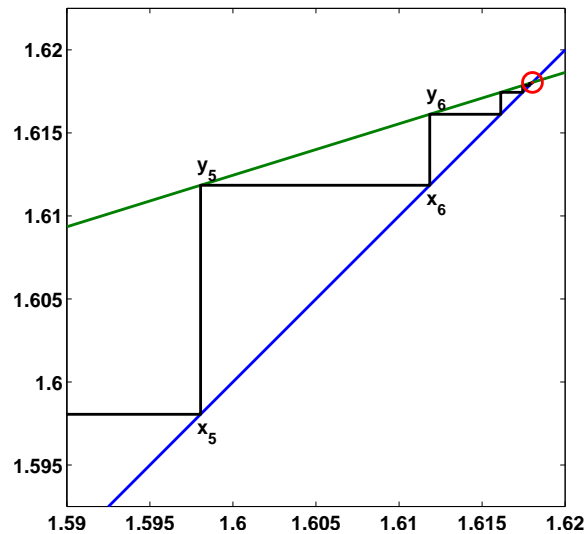


Figure 1.7. *The slope of the green curve is less than one.*

Run `fixedpoint` and zoom in, not on the red circle, but on one of the steps. Figure 1.7 shows the fifth step, from x_5 to x_6 . The blue line is straight and so the object formed with a segment of the blue line as the hypotenuse is actually an equilateral triangle. The green line is not straight, although it is hard to see the curvature at this level of magnification. The slope of one of the green segments is the ratio of the change in y to the change in x . In other words, the slope is the ratio of the length of successive steps of our iteration. The slope for the fifth step is

$$\frac{y_6 - y_5}{x_6 - x_5}$$

The key question is: what are the slopes of these green segments? Here is a MATLAB program that computes these slopes.

```
format short
x = 0;
y = 1;
while x ~= y
    delta = y - x;
    x = y;
    y = sqrt(1 + x);
    slope = (y - x)/delta
end
```

There are several important things to see in this program. The only assignment statement that is not terminated with a semicolon is the assignment to `delta`. Run this program and verify that the output starts with

```
slope = 0.4142
slope = 0.3369
slope = 0.3173
slope = 0.3115
slope = 0.3098
...
```

Each time through the loop, our program computes `delta`, which is the length of the previous step. New values of `x` and `y` are then used to compute `slope`, the ratio of the length of the new step to the length of the previous step. The slopes are all less than one and soon reach a value that is constant to four decimal places. Call this value δ . What is the value of δ ?

The blue line has slope one. The green curve has variable slope, but that slope approaches δ , which is significantly less than one. (In fact, δ is approximately $f'(\phi)$, the value of the derivative of $f(x)$ at the fixed point.)

The function

$$f(x) = \sqrt{1+x}$$

is a *contraction*. For points x_n and x_{n+1} near the fixed point, the function satisfies

$$|y_{n+1} - y_n| = |f(x_{n+1}) - f(x_n)| < \delta |x_{n+1} - x_n|$$

This implies that after n steps

$$|x_{n+1} - x_n| < \delta^n |x_1 - x_0|$$

Use MATLAB to compute δ^{10} and δ^{32} . The output uses an “e” to denote a power of 10 scale factor. For example `8e-6` is $8 \cdot 10^{-6}$. You should find that δ^{10} is less than 10^{-5} , which is the accuracy of `format short`, and that δ^{32} is less than 10^{-16} , which is the accuracy of `format long`.