

# Bisection and Fixed-Point Iterations

## 1 The Bisection Method

- bracketing a root
- running the bisection method
- accuracy and cost

## 2 Fixed-Point Iterations

- computing fixed points
- geometric interpretation
- a criterion for convergence

MCS 471 Lecture 3  
Numerical Analysis

Jan Verschelde, 27 August 2021

# Bisection and Fixed-Point Iterations

## 1 The Bisection Method

- **bracketing a root**
- running the bisection method
- accuracy and cost

## 2 Fixed-Point Iterations

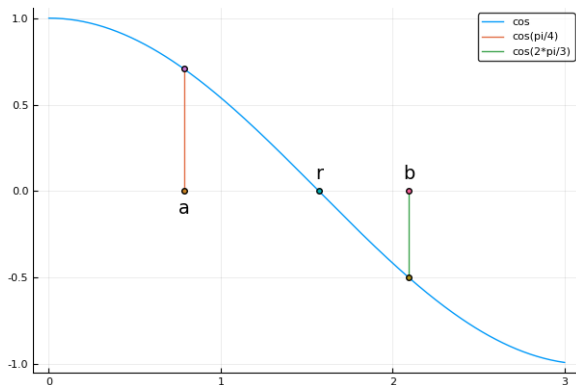
- computing fixed points
- geometric interpretation
- a criterion for convergence

# the bisection method

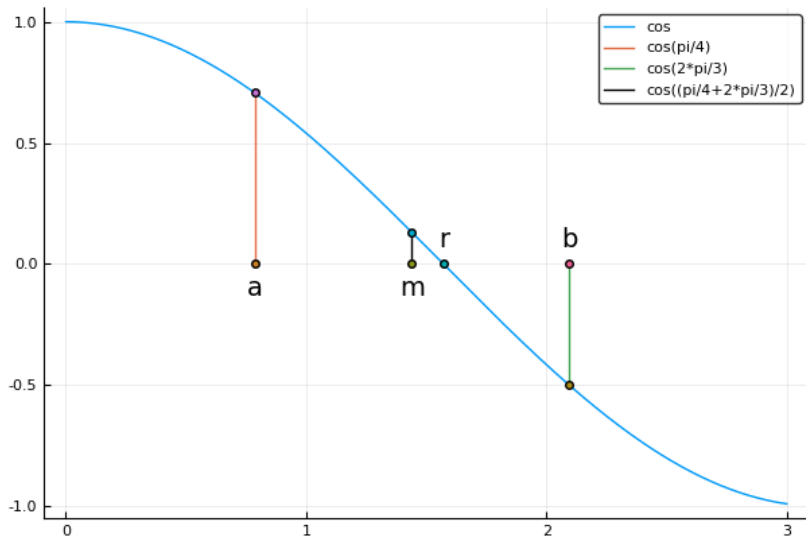
## Theorem (corollary of the intermediate value theorem)

Let  $f$  be a continuous function on  $[a, b]$ .

If  $f(a)f(b) < 0$ , then there is an  $r$ ,  $a < r < b$ , such that  $f(r) = 0$ .



bisect  $[a, b]$ ,  $m = (a + b)/2$



## halving the interval

```
 $m := (a + b)/2$   
if  $f(a)f(m) < 0$   
  then  $b := m$   
  else  $a := m$ 
```

Stop bisectioning if one of the following conditions is met:

- for a given  $\delta > 0$ :  $|b - a| < \delta$ ; or
- for a given  $\epsilon > 0$ :  $|f(m)| < \epsilon$ ; or
- for a given  $N$ : number of bisections  $\geq N$ .

# Bisection and Fixed-Point Iterations

## 1 The Bisection Method

- bracketing a root
- **running the bisection method**
- accuracy and cost

## 2 Fixed-Point Iterations

- computing fixed points
- geometric interpretation
- a criterion for convergence

# Julia function

using Printf # to format floating-points numbers

"""

Applies the bisection method to the function  $f$  on  $[a,b]$ ,  
 $f$  is assumed to be continuous and  $f(a)*f(b) < 0$ .

Stops when  $|f(a)| < \text{eps}$  or  $|f(b)| < \text{eps}$  or  $|b-a| < \text{eps}$ .

Returns the left and right bound of the final interval  
enclosing the root, and a Boolean reporting failure.

Failure is reported when the accuracy requirement is  
not satisfied in  $N$  steps; otherwise  $\text{fail} = 0$  on return.

Example :

```
(a,b,fail) = bisect(cos,pi/4,2*pi/3,1.0e-4,100)
```

"""

```
function bisect(f::Function,
```

```
    a::Float64,b::Float64,eps::Float64,N::Int)
```

```
    println("running the bisection method...")
```

```
    println("-----")
```

```
    println("      a          b          m          | f(m) |          | b-a |      ")
```

```
    println("-----")
```

## the loop

```
for i = 1:N
    m = (a+b)/2
    fm = f(m)
    if fm*f(a) < 0
        b = m
    else
        a = m
    end
    stra = @sprintf("%4.3f", a)
    strb = @sprintf("%4.3f", b)
    strm = @sprintf("%4.3f", m)
    stra_fm = @sprintf("%.2e", abs(fm))
    strabma = @sprintf("%.2e", abs(b-a))
    println("  $stra  $strb  $strm  $stra_fm  $strabma")
end
```

## applying the stopping criteria

```
        if (abs(fm) < eps) | ((b-a) < eps)
            fail = false;
            stri = string(i)
            println("succeeded after $stri steps")
            return [a, b, fail]
        end
    end
    strN = string(N)
    println("failed requirements after $strN steps")
    fail = true
    return [a, b, fail]
end

result = bisection(cos, pi/4, 2*pi/3, 1.0e-4, 100)
strres = string(result)
println("The result : $strres")
```

# running the function

```
$ julia bisection.jl  
running the bisection method...
```

| a     | b     | m     | f(m)     | b-a      |
|-------|-------|-------|----------|----------|
| 1.440 | 2.094 | 1.440 | 1.31e-01 | 6.54e-01 |
| 1.440 | 1.767 | 1.767 | 1.95e-01 | 3.27e-01 |
| 1.440 | 1.604 | 1.604 | 3.27e-02 | 1.64e-01 |
| 1.522 | 1.604 | 1.522 | 4.91e-02 | 8.18e-02 |
| 1.563 | 1.604 | 1.563 | 8.18e-03 | 4.09e-02 |
| 1.563 | 1.583 | 1.583 | 1.23e-02 | 2.05e-02 |
| 1.563 | 1.573 | 1.573 | 2.05e-03 | 1.02e-02 |
| 1.568 | 1.573 | 1.568 | 3.07e-03 | 5.11e-03 |
| 1.570 | 1.573 | 1.570 | 5.11e-04 | 2.56e-03 |
| 1.570 | 1.572 | 1.572 | 7.67e-04 | 1.28e-03 |
| 1.570 | 1.571 | 1.571 | 1.28e-04 | 6.39e-04 |
| 1.571 | 1.571 | 1.571 | 1.92e-04 | 3.20e-04 |
| 1.571 | 1.571 | 1.571 | 3.20e-05 | 1.60e-04 |

```
succeeded after 13 steps
```

```
The result : [1.57076, 1.57092, 0.0]
```

```
$
```

# the Jupyter notebook plotcosbisection.ipynb

jupyter plotcosbisection (autosaved) Logout

File Edit View Insert Cell Kernel Widgets LaTeX\_envs Help Not Trusted Julia 1.6.2

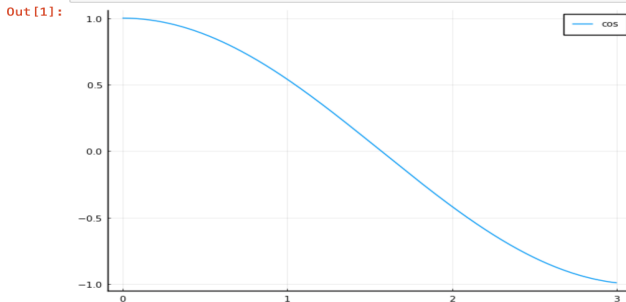
LaTeX\_envs: Refresh rendering of labels, equations and citations Read bibliography and generate references section

LaTeX\_envs: Some configuration options (toggle toolbar)

## The Bisection Method

We illustrate the application of the bisection methods with some plots.

```
In [1]: using Plots; pyplot()
plot(cos, 0.0:0.01:3.0, label="cos")
```



Now we add to this plot...

## a first exercise

**Exercise 1:** Consider  $f(x) = e^x - 8x = 0$ .

Apply three steps with the bisection method to find an approximation for a root of  $f(x) = 0$  inside the interval  $[a, b] = [-2, 3]$ .

- 1 Make a plot of  $f(x) = 0$  to illustrate the method marking the end points of the intervals  $[a_1, b_1](= [a, b])$ ,  $[a_2, b_2]$ , and  $[a_3, b_3]$  that contain the root of  $f$ .
- 2 Make a table with the numerical intermediate results of three steps of the bisection method.

The numbers in the table must be formatted with four significant decimal places after rounding, in scientific notation; for example, 2 is represented as  $2.0000\text{E}+00$ .

# Bisection and Fixed-Point Iterations

## 1 The Bisection Method

- bracketing a root
- running the bisection method
- accuracy and cost

## 2 Fixed-Point Iterations

- computing fixed points
- geometric interpretation
- a criterion for convergence

# accuracy and cost of the bisection method

In each step, the interval is halved.

Starting with  $[a, b]$ , after  $N$  steps, the length of the interval is

$$\frac{|b - a|}{2^N}.$$

Take the middle  $m$  of the last interval as the approximation for the root  $r$ , then

$$|r - m| \leq \frac{|b - a|}{2^{N+1}}.$$

Cost of the algorithm: one new function evaluation per step.

**Exercise 2:** Make the code for the function `bisect` more efficient by storing the function values.

# predicting the accuracy of the bisection method

## Exercise 3:

We want to find a root of  $f(x) = x^3 - 7x + 1$  in the interval  $[2, 3]$ .

- 1 Does the bisection method apply to this problem? Justify.
- 2 What is the accuracy of executing 20 steps of the bisection method?

Justify without running the bisection method.

# Bisection and Fixed-Point Iterations

## 1 The Bisection Method

- bracketing a root
- running the bisection method
- accuracy and cost

## 2 Fixed-Point Iterations

- computing fixed points
- geometric interpretation
- a criterion for convergence

# fixed-point iterations

## Definition

The number  $r \in \mathbb{R}$  is *a fixed point* of the function  $g$  if  $g(r) = r$ .

We will reformulate the root finding problem  $f(r) = 0$  into a fixed point computation of  $g(r) = r$ .

Example  $f(x) = x^3 + x - 1 = 0$  leads to  $g(x) = 1 - x^3 = x$ .

A fixed-point iteration applies a simple formula:

$$x_{k+1} = g(x_k), \quad k = 0, 1, \dots$$

starting at an initial guess  $x_0$  for the fixed point.

If the fixed-point iteration converges, then  $x_\infty$  is a fixed point.

# a Julia function – specification

using Printf # to format floating-point numbers

"""

Applies the fixed-point iteration to a given function g.

ON ENTRY :

|       |                                                                                                    |
|-------|----------------------------------------------------------------------------------------------------|
| g     | a function in one variable                                                                         |
| x0    | initial guess for the fixed-point iteration                                                        |
| maxit | upper bound on the number of iterations                                                            |
| tol   | tolerance on the $\text{abs}(g(x) - x)$ where $x$ is the current approximation for the fixed point |

ON RETURN :

|       |                                                                |
|-------|----------------------------------------------------------------|
| x     | the current approximation for the fixed point                  |
| numit | the number of iterations done                                  |
| fail  | true if the accuracy requirement was not met, false otherwise. |

EXAMPLE :

```
g(x) = 1-x^3
(x, numit, fail) = fixedpoint(g,0.5,10,1.0e-4)
```

"""

```
function fixedpoint(g::Function,x0::Float64,maxit::Int,tol::Float64)
```

## implementation of the function `fixedpoint`

```
strit = @sprintf("%3d", 0)
strx0 = @sprintf("%.4e", x0)
println("$strit : $strx0")
xprevious = x0
xnext = xprevious
for i=1:maxit
    xnext = g(xprevious)
    strit = @sprintf("%3d", i)
    strxi = @sprintf("%.4e", xnext)
    error = abs(xnext - xprevious)
    strerr = @sprintf("%.2e", error)
    println("$strit : $strxi : $strerr" )
    if error < tol
        return (xnext, i, false)
    end
    xprevious = xnext
end
return (xnext, maxit, true)
end
```

# the main program

```
"""
Calls the fixed point iteration on three examples.
"""
function main()
    x0 = 0.5
    maxit = 15
    tol = 1.0e-8
    g1(x) = 1 - x^3
    println("running a fixed-point iteration on 1 - x^3 ...")
    (endpt, numit, fail) = fixedpoint(g1, x0, maxit, tol)
    g2(x) = (1 - x)^(1/3)
    println("running a fixed-point iteration on (1 - x)^(1/3) ...")
    (endpt, numit, fail) = fixedpoint(g2, x0, maxit, tol)
    g3(x) = (1 + 2*x^3)/(1 + 3*x^2)
    println("running a fixed-point iteration on (1 + 2*x^3)/(1 + 3*x^2) ...")
    (endpt, numit, fail) = fixedpoint(g3, x0, maxit, tol)
end

main()
```

## running with $g(x) = 1 - x^3$

running a fixed-point iteration on  $1 - x^3 \dots$

```
0 : 5.0000e-01
1 : 8.7500e-01 : 3.75e-01
2 : 3.3008e-01 : 5.45e-01
3 : 9.6404e-01 : 6.34e-01
4 : 1.0405e-01 : 8.60e-01
5 : 9.9887e-01 : 8.95e-01
6 : 3.3761e-03 : 9.95e-01
7 : 1.0000e+00 : 9.97e-01
8 : 1.1544e-07 : 1.00e+00
9 : 1.0000e+00 : 1.00e+00
10 : 0.0000e+00 : 1.00e+00
11 : 1.0000e+00 : 1.00e+00
12 : 0.0000e+00 : 1.00e+00
13 : 1.0000e+00 : 1.00e+00
14 : 0.0000e+00 : 1.00e+00
15 : 1.0000e+00 : 1.00e+00
```

running with  $g(x) = \sqrt[3]{1-x}$

running a fixed-point iteration on  $(1-x)^{(1/3)} \dots$

```
0 : 5.0000e-01
1 : 7.9370e-01 : 2.94e-01
2 : 5.9088e-01 : 2.03e-01
3 : 7.4236e-01 : 1.51e-01
4 : 6.3631e-01 : 1.06e-01
5 : 7.1380e-01 : 7.75e-02
6 : 6.5901e-01 : 5.48e-02
7 : 6.9863e-01 : 3.96e-02
8 : 6.7045e-01 : 2.82e-02
9 : 6.9073e-01 : 2.03e-02
10 : 6.7626e-01 : 1.45e-02
11 : 6.8665e-01 : 1.04e-02
12 : 6.7922e-01 : 7.42e-03
13 : 6.8454e-01 : 5.32e-03
14 : 6.8074e-01 : 3.81e-03
15 : 6.8346e-01 : 2.73e-03
```

running on  $g(x) = (1 + 2x^3)/(1 + 3x^2)$

$$x = 1 - x^3$$

$$3x^3 + x = 1 + 2x^3$$

$$(3x^2 + 1)x = 1 + 2x^3$$

$$x = (1 + 2x^3)/(1 + 3x^2)$$

running a fixed-point iteration on  $(1 + 2x^3)/(1 + 3x^2)$  ...

0 : 5.0000e-01

1 : 7.1429e-01 : 2.14e-01

2 : 6.8318e-01 : 3.11e-02

3 : 6.8233e-01 : 8.51e-04

4 : 6.8233e-01 : 6.19e-07

5 : 6.8233e-01 : 3.28e-13

# Bisection and Fixed-Point Iterations

## 1 The Bisection Method

- bracketing a root
- running the bisection method
- accuracy and cost

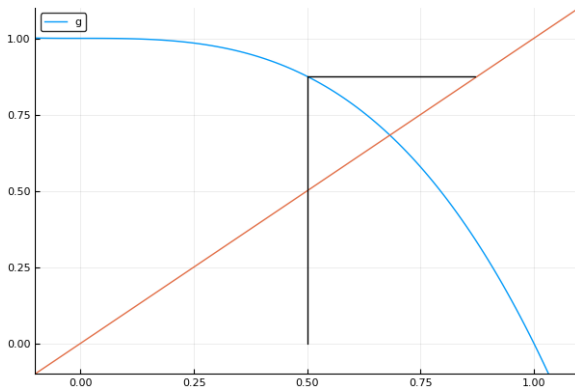
## 2 Fixed-Point Iterations

- computing fixed points
- **geometric interpretation**
- a criterion for convergence

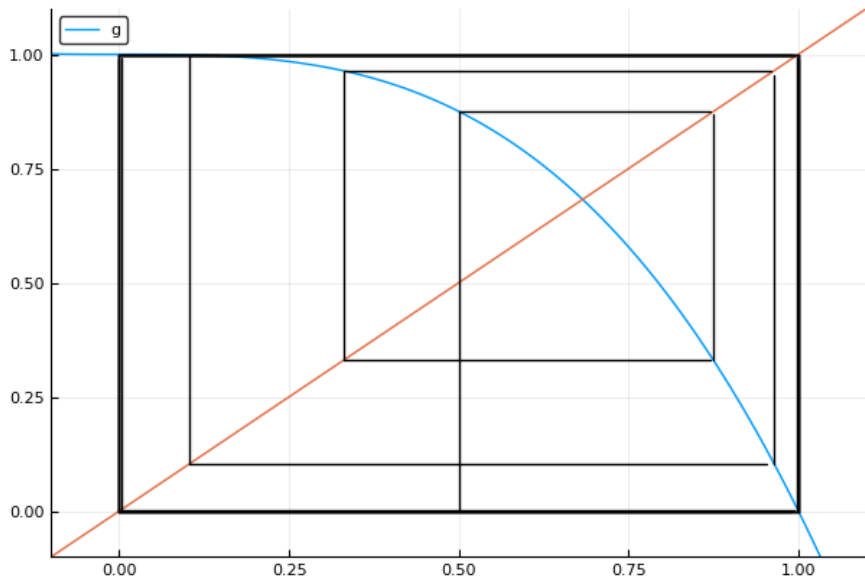
# drawing a cobweb diagram

Two steps in executing  $x_1 = g(x_0)$ :

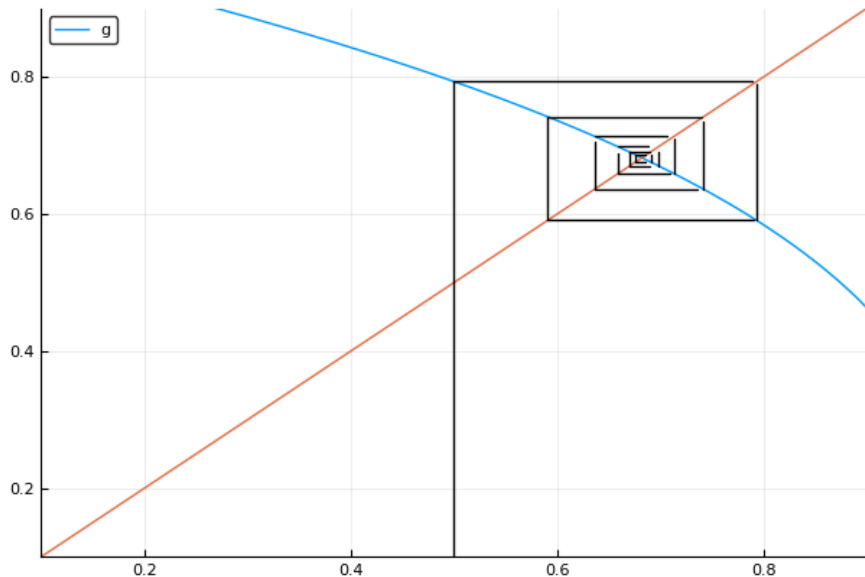
- 1 evaluate  $g$ :  $y = g(x_0)$ , draw a vertical line
- 2 eliminate  $y$ :  $x_1 = y$ , draw a horizontal line



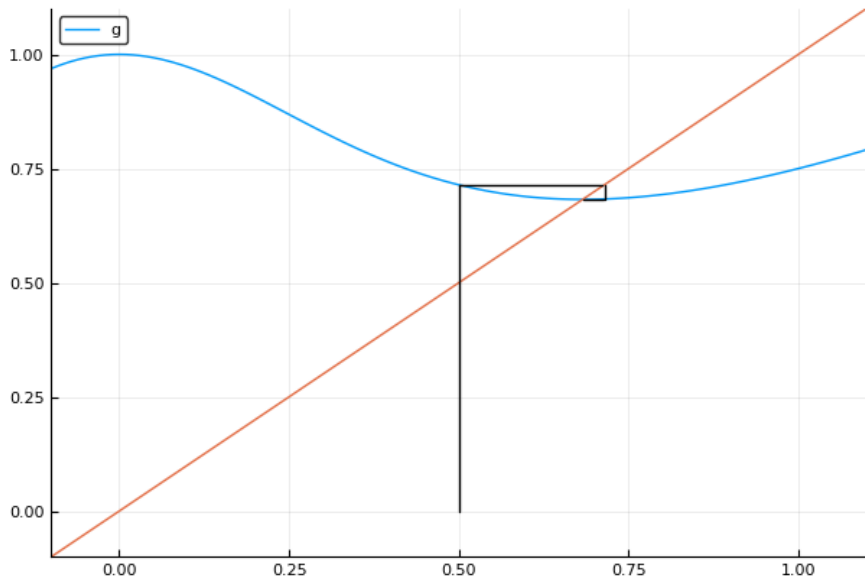
# cobweb diagram for $g(x) = 1 - x^3$



# cobweb diagram for $g(x) = \sqrt[3]{1-x}$



# cobweb diagram for $g(x) = (1 + 2x^3)/(1 + 3x^2)$



## the Jupyter notebook `cobweb.ipynb`

The posted Jupyter notebook `cobweb.ipynb` combines

- 1 numbers computed by the fixed-point iterations;
- 2 a plot of the cobweb diagram; and *most importantly*,
- 3 text to document the numbers and the plot.

On a fresh Julia installation, install `Plots` and `PyPlot`:

```
julia> import Pkg; Pkg.add("Plots")  
julia> import Pkg; Pkg.add("PyPlot")
```

Before doing homework, download the programs and notebooks at <http://www.math.uic.edu/~jan/mcs471> and execute the programs.

# Bisection and Fixed-Point Iterations

## 1 The Bisection Method

- bracketing a root
- running the bisection method
- accuracy and cost

## 2 Fixed-Point Iterations

- computing fixed points
- geometric interpretation
- a criterion for convergence

## a criterion for convergence

If the fixed-point iteration  $x_{k+1} = g(x_k)$  converges to  $x_\infty$ , where  $x_\infty$  is a fixed point:  $x_\infty = g(x_\infty)$ , then the error  $x_\infty - x_{k+1}$  can be written as

$$\begin{aligned}x_\infty - x_{k+1} &= g(x_\infty) - g(x_k) \\&= \frac{g(x_\infty) - g(x_k)}{x_\infty - x_k} (x_\infty - x_k)\end{aligned}$$

For sufficiently close  $x_k$  to  $x_\infty$ ,  $\lim_{x_k \rightarrow x_\infty} \left( \frac{g(x_\infty) - g(x_k)}{x_\infty - x_k} \right) = g'(x_\infty)$  and we have:

$$|x_\infty - x_{k+1}| = |g'(x_\infty)| |x_\infty - x_k|.$$

### Theorem

*If  $x_{k+1} = g(x_k)$  converges to  $x_\infty$ , then  $|g'(x_\infty)| < 1$ .*

If at a fixed point  $r = g(r)$ :  $|g'(r)| > 1$ , then  $x_{k+1} = g(x_k)$  diverges.

## a fourth exercise

### Exercise 4:

The equation  $x = 2 \sin(x)$  has two fixed points: 0 and 1.895.

- 1 Compute the rate of convergence (or divergence) of the fixed-point iteration  $x_{k+1} = 2 \sin(x_k)$ , when  $x_k$  is close to the fixed point 0 or close to 1.895.
- 2 For  $x_0 = 0.5$ , compute  $x_1$ ,  $x_2$ , and  $x_3$ . Draw a cobweb diagram.

The assigned homework that will be collected is at

<http://www.math.uic.edu/~jan/mcs471/homework.html>.