Basics of Numerical Methods 4

by **Patrick Boily** (inspired by **Diane Guignard**)

In today's digital age, it's hard to envision a world devoid of data and computers. Yet, the principles of "data science" predate our modern era of digital computation.

Take, for instance, Johannes Kepler's remarkable 16th-century computations. Before the invention of calculus, he analyzed the orbit of Mars based on Tycho Brahe's observations. This monumental effort culminated in the *Laws of Planetary Motion* [6]. Fast forward to the 20th century, where human computers at the *Jet Propulsion Laboratory* painstakingly calculated the number of rockets needed for space missions. These computations often spanned over a week, filling six to eight notebooks with data and intricate formulas [4]. Such endeavours underscore the invaluable contributions of data-based calculations to our scientific legacy.

Modern technology allows us to retrace and even surpass the feats of our predecessors in a mere fraction of their original time. With advancements in quantum computing, big data processing, and artificial intelligence on the horizon, it seems our computational potential knows no bounds – at least from a technical perspective.[1]

This chapter provides an **overview** of the foundational concepts and techniques at the heart of data science: the often-hidden **mathematics underlying data calculations** and data processing. Substantially more details are available in [1, 3].[2]

1: From a sociological and ethical viewpoint, however, the landscape is potentially more complex.

2: Some of the required topological concepts can also be found in [2].

4.1 Basic Concepts

In **scientific computing**, we typically navigate from a **physical problem** (observed phenomenon) to a **computed solution** (algorithm solution) *via* a **mathematical problem** (model) and/or a **numerical problem**, as illustrated in Figure 4.1.

If u is the real solution of the problem and $\hat{u}$ the computed solution, we are often interested in the **computational error**, for obvious reasons: the smaller it is, the more confident we are in exhibiting $\hat{u}$ as a solution.

There are two types of such errors:

- **absolute error:** $|u - \hat{u}|$;
- **relative error:** $\frac{|u-\hat{u}|}{|u|}$.

Sources of Error In practice, it is nearly always the case that the computational error is not 0, i.e., that $u \neq \hat{u}$.

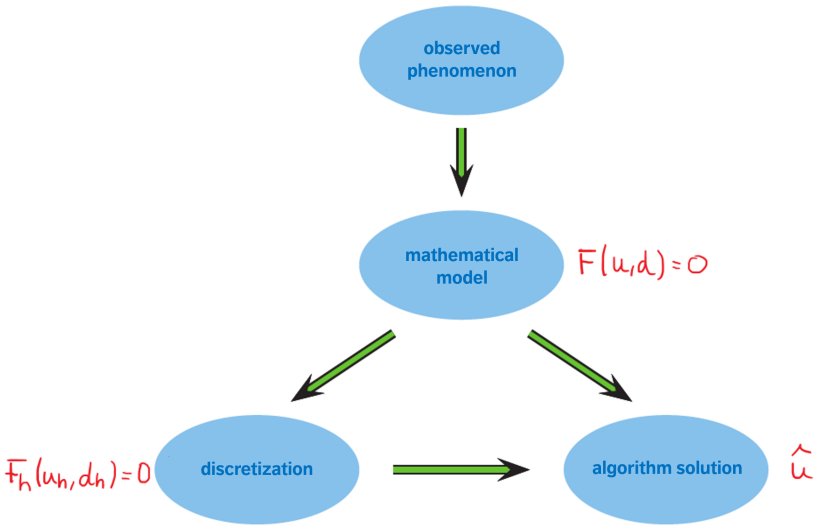


Figure 4.1: Schematics of scientific computing (modified from [1]).

That might prove to be the case due to:

- errors in the mathematical model;
- errors in the input data (e.g., due to measurements);
- **approximation errors**, such as **discretization errors** (in interpolation, differentiation, integration, ...) and **convergence errors** (in iterative methods), and/or
- **round-off errors** due to finite machine precision.

Assessing Numerical Algorithms In theory, there may be multiple ways of solving a problem numerically. In practice, we usually favour algorithms that are:

- **accurate**;
- **efficient** (in terms of CPU runtime, storage requirements, rate of convergence, etc.), as well as
- **robust/reliable/stable** (roughly speaking, computations do not magnify approximation errors).

4.1.1 Round-Off Error

In a computer, a real number x is stored using a **floating point representation**:

$$\mathrm{fl}(x) = (-1)^s \cdot (1.d_1 d_2 \cdots d_t) \cdot 2^e,$$

where

- $s \in \{0, 1\}$ determines the **sign** of x, which is positive if $s = 0$, and negative if $s = 1$;
- $f = d_1 d_2 \ldots d_t$ is the **mantissa** (or fraction) of x in base 2, with $d_i \in \{0, 1\}$, $1 \leq i \leq t$, and
- e is the **exponent**, with $L \leq e \leq U$ for some L, U.

For instance, the floating point representation of -6.5 is

$$\mathrm{fl}(-6.5) = (-1)^1 \cdot (1.101) \cdot 2^2 \quad \Longrightarrow \quad -\left(1 + \frac{1}{2} + \frac{1}{2^3}\right) \cdot 2^2 = -6.5.$$

It is not too difficult to show that the following bound applies on the relative (rounding) error:

$$\frac{|x - \mathrm{fl}(x)|}{|x|} \leq 2^{-(t+1)}.$$

Single vs. Double Precision Different operational systems/computational software use different values of s, e, and f.

	s	e	f	L	U
single (32 bits)	1 bit	8 bits	23 bits	-126	127
double (64 bits)	1 bit	11 bits	52 bits	-1022	1023

In double precision, for instance, we represent numbers as follows:

$$(-1)^s \cdot \left(1 + \sum_{i=1}^{52} \frac{d_i}{2^i}\right) \cdot 2^e \quad \text{with} \quad L = -1022 \leq e \leq 1023 = U.$$

- The **smallest positive number** that can be represented has $s = 0$, $d_i = 0$, and $e = L \implies x_{\min} = 2^{-1022}$;
- the **largest positive number** has $s = 0$, $d_i = 1$, and $e = U \implies x_{\max} = (2 - 2^{-52})2^{1023}$.

We can recover these values (and other parameters) in R.

```
.Machine
```

```
$double.eps
[1] 2.220446e-16

$double.neg.eps
[1] 1.110223e-16

$double.xmin
[1] 2.225074e-308

$double.xmax
[1] 1.797693e+308

$double.base
[1] 2

$double.digits
[1] 53

$double.rounding
[1] 5

$double.guard
[1] 0

$double.ulp.digits
[1] -52

$double.neg.ulp.digits
[1] -53

$double.exponent
[1] 11

$double.min.exp
[1] -1022

$double.max.exp
[1] 1024

$integer.max
[1] 2147483647

$sizeof.long
[1] 4

$sizeof.longlong
[1] 8

$sizeof.longdouble
[1] 16

$sizeof.pointer
[1] 8

$longdouble.eps
[1] 1.084202e-19

$longdouble.neg.eps
[1] 5.421011e-20

$longdouble.digits
[1] 64

$longdouble.rounding
[1] 5

$longdouble.guard
[1] 0

$longdouble.ulp.digits
[1] -63

$longdouble.neg.ulp.digits
[1] -64

$longdouble.exponent
[1] 15

$longdouble.min.exp
[1] -16382

$longdouble.max.exp
[1] 16384
```

Round-off arithmetic can lead to odd behaviour – consider, for instance, the function $f : (0, \infty) \to \mathbf{R}$ defined by

$$f(x) = \frac{(1 + x) - 1}{x}.$$

In theory, we know that $f \equiv 1$ on $(0, \infty)$. In practice, things get messy.

We define the function in R using the following chunk of code.

```
f.test <- function(x){
  ((1+x)-1)/x
}
```

The function evaluates exactly to 1 for $x = 1, 10^{-9}, 10^{-10}$.

```
> f.test(1)
[1] 1
> f.test(0.0000000001)
[1] 1
> f.test(0.00000000001)
[1] 1
```

For smaller values, something strange is happening.

```
> f.test(0.000000000001)
[1] 1.000089
> f.test(0.0000000000001)
[1] 0.9992007
> f.test(0.000000000000001)
[1] 1.110223
> f.test(0.0000000000000001)
[1] 0
```

This phenomenon is know as **cancellation error**. Say we want to compute $f(10^{-16})$. We must first add 10^{-16} and 1 – to do so, we first need to **align the exponents**.

$$
\begin{aligned}
1 &= 1.000000000000000 \times 10^{0} \\
10^{-16} &= 1.000000000000000 \times 10^{-16} \\
&= 0.100000000000000 \times 10^{-15} \\
&= 0.010000000000000 \times 10^{-14} \\
&= 0.001000000000000 \times 10^{-13} \\
&= 0.000100000000000 \times 10^{-12} \\
&= 0.000010000000000 \times 10^{-11} \\
&= 0.000001000000000 \times 10^{-10} \\
&= 0.000000100000000 \times 10^{-9} \\
&= 0.000000010000000 \times 10^{-8} \\
&= 0.000000001000000 \times 10^{-7} \\
&= 0.000000000100000 \times 10^{-6} \\
&= 0.000000000010000 \times 10^{-5} \\
&= 0.000000000001000 \times 10^{-4} \\
&= 0.000000000000100 \times 10^{-3} \\
&= 0.000000000000010 \times 10^{-2} \\
&= 0.000000000000001 \times 10^{-1} \\
&= 0.000000000000000 \times 10^{0}
\end{aligned}
$$

From the perspective of double precision arithmetic, $1 + 10^{-16} = 1$! This explains why $f(10^{-16}) = 0$ in R.[3]

3: In R, the only numbers that are represented **exactly** are the integers and negative powers of 2. More information on round-off error (and error propagation) is available in [3].

4.2 Solving an Equation in 1 Variable

In this section, we will discuss how to solve an equation of the form

$$f(x) = 0$$

numerically, where $f : [a, b] \to \mathbb{R}$ is a (potentially non-linear) **continuous** function. A real number $x^* \in [a, b]$ for which $f(x^*) = 0$ is a **root** (or a **zero**) of the function f; "solving f in $[a, b]$" means finding (at least) one root of f in $[a, b]$.[4]

4: When the context is clear, we will drop "in $[a, b]$" from the conversation.

Iterative Procedures In some cases, we may be able to solve f exactly – if $a \neq 0$, for instance, the linear equation $ax + b = 0$ has exactly one zero at $x^* = -b/a$. In practice, we can usually only hope to solve a continuous f **approximately**, assuming a solution even exists.[5]

5: Not every function has a zero: for instance, $f : \mathbb{R} \to \mathbb{R}$ defined by $f(x) = x^2+1$ does not have a root in $\mathbb{R}$.

In general, we must use an **iterative procedure** in order to zoom in on a root. Given an initial guess x_0, we generate a sequence of **iterates** $x_1, x_2, x_3, \ldots$ which (hopefully) converges to a root x^* of f.

In order to exhibit a candidate x^*, we must stop the iterative process after a **finite number of iterations** n, according to a prescribed **stopping criterion** such as:

- $|x_n - x_{n-1}| \leq \text{tol}$;
- $|x_n - x_{n-1}|/|x_n| \leq \text{tol}$, provided $x_n \neq 0$, or
- $|f(x_n)| \leq \text{tol}$,

where tol is the algorithm's **prescribed tolerance**. We can avoid infinite loops by also prescribing a **maximum number of iterations** $N_{\max}$.

4.2.1 Bisection Method

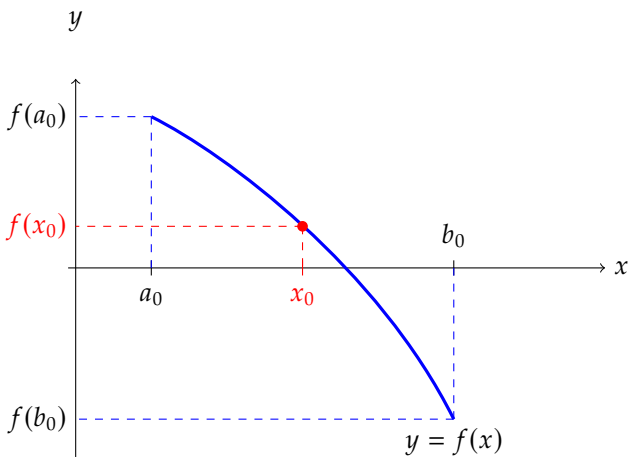
This method is based on the **intermediate value theorem**: if $f \in C([a, b])$ and $f(a)f(b) \leq 0$, then there exists $x^* \in [a, b]$ such that $f(x^*) = 0$.

Let $a_0 = a$, $b_0 = b$ and $x_0 = (a_0 + b_0)/2$. There are three possibilities:

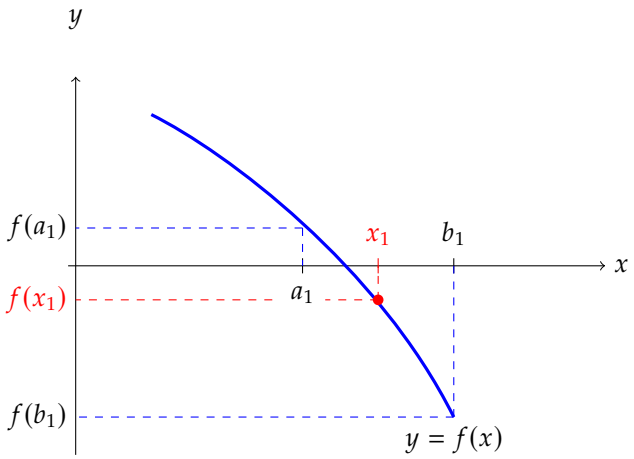
1. $f(x_0) = 0$, in which case $x^* = x_0$ is a root and we are done;
2. $f(a_0)f(x_0) < 0$, in which case f has a root in $[a, x_0]$ and we set $a_1 = a_0$, $b_1 = x_0$, or
3. $f(b_0)f(x_0) < 0$, in which case f has a root in $[x_0, b]$ and we set $a_1 = x_0$, $b_1 = b_0$.

In the latter two cases, we also set $x_1 = (a_1 + b_1)/2$; the **bisection method** re-iterates this process to generate a sequence $\{x_0, x_1, x_2, \ldots\}$, which converges to a root $x^* \in [a, b]$ of f.

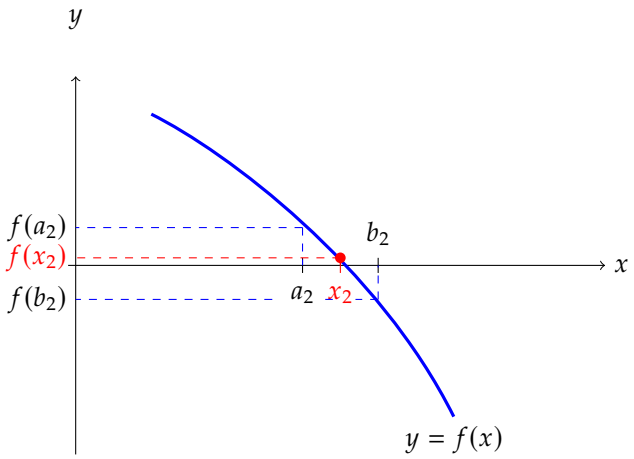
Illustration of the Method Let $f : [a, b] \to \mathbb{R}$ be the continuous function whose graph is displayed on the next page. Let $a_0 = a$, $b_0 = b$ and $x_0 = (a_0 + b_0)/2$; clearly, $f(a_0)f(b_0) < 0$.



We find ourselves in the third case, since $f(b_0)f(x_0) < 0$; as such f has a root in $[x_0, b_0]$. In the next iteration, we set $a_1 = x_0$, $b_1 = b_0$, and $x_1 = (a_1 + b_1)/2$.



We find ourselves in the second case, since $f(a_1)f(x_1) < 0$; as such f has a root in $[a_1, x_1]$. In the next iteration, we set $a_2 = a_1$, $b_2 = x_1$, and $x_2 = (a_2 + b_2)/2$, and so on.



Assume that we would like to use the bisection method to find an approximation x_n of a root x^* satisfying

$$|x_n - x^*| \leq \text{tol}$$

for a given tolerance tol > 0. How large n should be? We can answer this question using the following result.

Theorem: let $f \in C([a,b])$ be such that $f(a)f(b) < 0$. The sequence $\{x_k\}$ generated by the bisection method approximates a root x^* of f with

$$|x_k - x^*| \leq \frac{b-a}{2^{k+1}}, \qquad k \geq 0.$$

Proof: we go through the procedure as illustrated previously; at step k, we have $x^* \in [a_k, b_k]$ and $x_k = (a_k + b_k)/2$. Moreover, $b_k - a_k = \frac{(b-a)}{2^k}$ as we have divided $[a,b]$ in two k times at that point, and so

$$|x_k - x^*| \leq \frac{1}{2}(b_k - a_k) = \frac{b-a}{2^{k+1}},$$

which completes the proof. ■

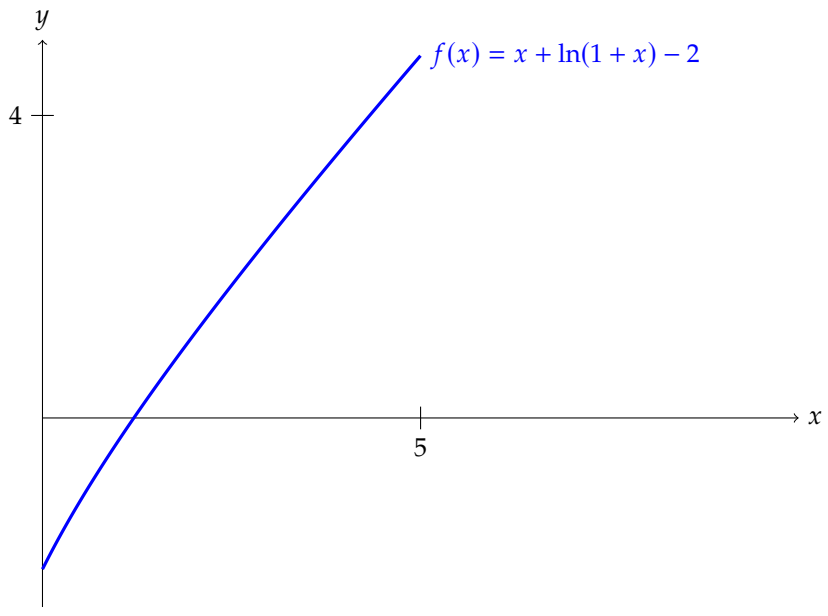
We can guarantee the desired absolute error tolerance if

$$|x_n - x^*| \leq \frac{b-a}{2^{n+1}} < \text{tol},$$

which is to say

$$2^{n+1} \geq \frac{b-a}{\text{tol}} \implies n \geq \log_2\left(\frac{b-a}{\text{tol}}\right) - 1.$$

Example: consider the function $f(x) = x + \ln(1+x) - 2$, $x \in [0,5]$, whose graph is given below.



We can guarantee that the bisection iterate x_n is within tol $= 10^{-4}$ of x^* when $n \geq \log_2(5 \cdot 10^4) - 1 = 14.60964$, which is to say when $n \geq 15$.

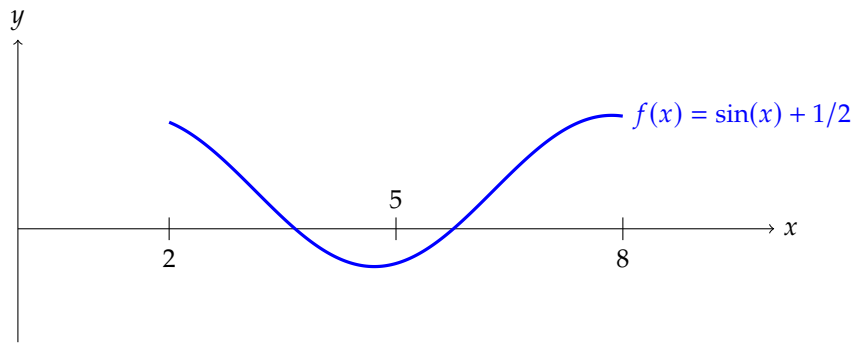
Algorithm: bisection method

Input: continuous f; a, b with $f(a)f(b) < 0$; tol > 0
Output: approximation p of x^*, n

Initialization: $a_0 = a$, $b_0 = b$, $x_0 = \frac{a_0+b_0}{2}$, $n = \lceil \log_2 \left(\frac{b-a}{\text{tol}} \right) - 1 \rceil$;
For $k = 0, 1, 2, \ldots, n-1$ **do**
    If $f(x_k) = 0$ **then**
        $p = x_k$, $n = k$;
    Stop
    If $f(a_k)f(x_k) < 0$ **then**
        $a_{k+1} = a_k$, $b_{k+1} = x_k$;
    Else
        $a_{k+1} = x_k$, $b_{k+1} = b_k$;
    End
    $x_{k+1} = \frac{a_{k+1}+b_{k+1}}{2}$;
End
$p = x_n$.

Comments On the positive side, the bisection method always converges when f has a different sign at a and b, and we have precise control over the error; on the negative side, the convergence is quite slow (the upper bound on the error only halves with each step), and the method fails to be of use if f does not change sign near a root x^*.

Example Throughout this section, we will attempt to find roots of the test function $f(x) = \sin(x) + 1/2$ over the interval $[2, 8]$.



Graphically, we see that there are two roots: $x^* \in (2, 5)$ and $x_* \in (5, 8)$. The function is implemented in R as follows.

```
f.test <- function(x){ sin(x)+1/2 }
```

Can the bisection method find f's roots? Here is an implementation of the method in R.

Bisection method

```
bisection <- function(f, a, b, tol) {
  # initialization
  k <- 0          # 0th iteration
  x <- (a + b)/2  # first iterate (root approximation)
  x_vec <- c(x)

  # max number of iterations for absolute error control
  n <- ceiling(log2((b - a) / tol) - 1)

  # Bisection method
  while (k < n) {

    if (f(x) == 0) {
      break
    } else {
      k <- k + 1

      if (f(a) * f(x) < 0) {
        b <- x
      } else {
        a <- x
      }
    }

    x <- (a + b) / 2
    x_vec <- c(x_vec, x)
  }

  return(list(x=x, k=k, x_vec=x_vec))
}
```

Note that we have not included input checks to the code: we must have $a < b$, tol > 0, $f(a)f(b) < 0$.

We look for x^* in the interval $[2, 5]$, with a tolerance of 0.00005.

```
bisection(f.test, 2, 5, 0.00005)
```

```
$x
[1] 3.665207

$k
[1] 15

$x_vec
 [1] 3.500000 4.250000 3.875000 3.687500 3.593750 3.640625 3.664062
 [8] 3.675781 3.669922 3.666992 3.665527 3.664795 3.665161 3.665344
[15] 3.665253 3.665207
```

What about x_* in the interval $[5, 8]$, with the same tolerance?

```
bisection(f.test, 5, 8, 0.00005)
```

```
$x
[1] 5.759567

$k
[1] 15

$x_vec
 [1] 6.500000 5.750000 6.125000 5.937500 5.843750 5.796875 5.773438
 [8] 5.761719 5.755859 5.758789 5.760254 5.759521 5.759888 5.759705
[15] 5.759613 5.759567
```

The object `x_vec` lists the iterates x_0 to x_{15}: the convergence rate is indeed rather slow.

We can verify that the final iterates are quite close to x^* and x_*.

```
f.test(3.665207)
f.test(5.759567)
```

```
[1] -1.348466e-05
[1] -1.691475e-05
```

Note however that we manually have to separate the problem into two sub-problems in order to capture both roots. If we were to try to find the roots of the test function over a longer interval containing both x^* and x_*, such as [-10,10],[6] the algorithm would find at most one root.

6: We should first verify that $f(-10)f(10) < 0$.

```
bisection(f.test, -10, 10, 0.00005)
```

```
$x
[1] 3.665199

$k
[1] 18

$x_vec
 [1] 0.000000 5.000000 2.500000 3.750000 3.125000 3.437500 3.593750
 [8] 3.671875 3.632812 3.652344 3.662109 3.666992 3.664551 3.665771
[15] 3.665161 3.665466 3.665314 3.665237 3.665199
```

This highlight an important feature of numerical methods in the context of finding roots of a function: they are more useful when we already have a fairly good idea about the location of its roots.

Without the assumption check, the code will still run and might even converge to a root... but not necessarily so. How does the code respond for the test function over $[2, 8]$? Over $[2, 3]$?

4.2.2 Golden Ratio Method

We can also "solve" a continuous function $f : [a, b] \to \mathbb{R}$ by finding a value x^* that **maximizes** f over $[a, b]$ and/or a value x_* that **minimizes** f over $[a, b]$.[7]

7: Admittedly, the word "solve" does some heavy lifting here.

In this new context, the **Golden ratio method** plays an analogous role for **unimodal** continuous functions to that played by the bisection method in the original context.

This method is based on the **max/min theorem**: if $f \in C([a, b])$, then there exist $x^*, x_* \in [a, b]$ such that $f(x^*) \geq f(x) \geq f(x_*)$ for all $x \in [a, b]$.

Say we are seeking the minimal value. If $a = b$, then $x^* = x_* = a = b$, so assume that $a < b$. Let $\varphi = (1 + \sqrt{5})/2$, and set $a_0 = a$ and $b_0 = b$.

1. Set $c = b_0 - (b_0 - a_0)/\varphi$ and $d = a_0 + (b_0 - a_0)/\varphi$. We have

$$\begin{aligned}
\varphi < 2 &\implies \frac{b_0 - a_0}{2} < \frac{b_0 - a_0}{\varphi} \implies b_0 - a_0 < 2\left(\frac{b_0 - a_0}{\varphi}\right) \\
&\implies c = b_0 - \frac{b_0 - a_0}{\varphi} < a_0 + \frac{b_0 - a_0}{\varphi} = d, \\
1 < \varphi &\implies \frac{b_0 - a_0}{\varphi} < b_0 - a_0 \implies a_0 < b_0 - \frac{b_0 - a_0}{\varphi} \text{ and} \\
&\qquad\qquad a_0 + \frac{b_0 - a_0}{\varphi} < b_0,
\end{aligned}$$

   and so $[c, d] \subsetneq [a_0, b_0]$.
2. If $f(c) < f(d)$, set $a_1 = a_0$ and $b_1 = d$.
3. Otherwise, set $a_1 = c$ and $b_1 = b_0$.

The algorithm iterates with this new sub-interval $[a_1, b_1]$, to produce a sequence of nested intervals

$$[a_0, b_0] \subsetneq [a_1, b_1] \subsetneq \cdots [a_k, b_k] \subseteq \cdots$$

That the sequence of sub-intervals converges to the minimizer x_* is guaranteed by the **nested interval theorem** since

$$\lim_{k\to\infty} (b_k - a_k) = \lim_{k\to\infty} \left(\frac{b_0 - a_0}{\varphi^{k+1}}\right) = 0.$$

We can guarantee a desired absolute error tolerance tol after n iterations if

$$b_n - a_n = \frac{b_0 - a_0}{\varphi^{n+1}} \leq \text{tol},$$

which is to say

$$\varphi^{n+1} \geq \frac{b_0 - a_0}{\text{tol}} \implies n \geq \log_\varphi \left(\frac{b_0 - a_0}{\text{tol}}\right) - 1.$$

We learn in introductory calculus classes that a differentiable function reaches its max/min at a point where the derivative is 0 or at a point of the domain where the derivative does not exist.[8]

8: So we could use the bisection method on f' instead, say.

The Golden Ratio method does not require knowledge of the derivative, however!

We implement the method (without checks) as follows.

Golden Ratio method

```
golden.min <- function(f, a, b, tol) {
  # initialization
  phi = (1 + sqrt(5))/2
  k <- 0 # 0th iteration
  c <- b - (b - a)/phi
  d <- a + (b - a)/phi

  a_vec <- c(a) # first iterate (lower endpoint)
  b_vec <- c(b) # first iterate (upper endpoint)

  # max number of iterations for absolute error control
  n <- ceiling(log((b - a) / tol) / log(phi) - 1)

  # Golden Ratio method
  while (k < n) {
      k <- k + 1
      if (f(c) < f(d)) {
        b <- d
      } else {
        a <- c
      }

    c <- b - (b - a)/phi
    d <- a + (b - a)/phi
    a_vec <- c(a_vec, a)
    b_vec <- c(b_vec, b)
    }

  # point estimate for minimizer
  x = (a + b)/2
  fx = f(x)

  return(list(fx=fx, x=x, k=k, a_vec=a_vec, b_vec=b_vec))
}
```

Example In the test function from the previous section, we see that the minimum occurs somewhere in $[4.5, 5]$.

```
golden.min(f.test,2,8,0.00005)
```

```
$fx
[1] -0.5

$x
[1] 4.712396

$k
[1] 24
```

```
$a_vec
 [1] 2.000000 2.000000 3.416408 4.291796 4.291796 4.291796 4.498447
 [8] 4.626165 4.626165 4.674948 4.674948 4.693582 4.705098 4.705098
[15] 4.709497 4.709497 4.711177 4.711177 4.711819 4.712216 4.712216
[22] 4.712216 4.712309 4.712367 4.712367

$b_vec
 [1] 8.000000 5.708204 5.708204 5.708204 5.167184 4.832816 4.832816
 [8] 4.832816 4.753882 4.753882 4.723732 4.723732 4.723732 4.716615
[15] 4.716615 4.713896 4.713896 4.712858 4.712858 4.712858 4.712612
[22] 4.712461 4.712461 4.712461 4.712425
```

From theoretical considerations, we already know that the minimal value of $f(x) = \sin(x) + 1/2$ is indeed $-1/2$.

4.2.3 Fixed Point Iteration Method

Both of the previous algorithms **converge slowly**, in the sense that while they do converge, they typically require an unreasonably large number of iterations to do so.

A root-finding problem $f(x) = 0$ can be transformed into an equivalent **fixed point problem** $g(x) = x$. For instance, if

$$g(x) = x - 2f(x) \quad \text{or} \quad g(x) = x + f^2(x),$$

then $f(x^*) = 0$ if and only if $g(x^*) = x^*$.[9] An input x^* for which $g(x^*) = x^*$ is called a **fixed point** of g.

9: There are infinitely many different formulations for g, as we will see, but not all choices are suitable.

The following theorem gives sufficient conditions under which a function $g : [a, b] \to \mathbb{R}$ has a unique fixed point in $[a, b]$.

Fixed Point Theorem:

1. if $g \in C([a, b])$ and $g(x) \in [a, b]$ for all $x \in [a, b]$, then g has a fixed point in $[a, b]$;
2. if g' exists on (a, b) and if there exists $0 < \rho < 1$ such that

$$|g'(x)| \leq \rho, \quad \forall x \in (a, b),$$

then g has a unique fixed point in $[a, b]$.

Proof: define $\lambda : [a, b] \to \mathbb{R}$ by $\lambda(x) = g(x) - x$. Since $g(a) \geq a$, then $\lambda(a) = g(a) - a \geq a - a = 0$. Since $g(b) \leq b$, $\lambda(b) = g(b) - b \leq b - b = 0$. But g is continuous; according to the the intermediate value theorem, there is thus a $p \in [a, b]$ such that $\lambda(p) = 0$, which is to say $g(p) = p$.

Now suppose $p^*, p_* \in [a, b]$ are two fixed points of g; then

$$|p^* - p_*| = |g(p^*) - g(p_*)|.$$

According to the mean value theorem,[10] if g is differentiable, there is a c between p_* and p^* such that

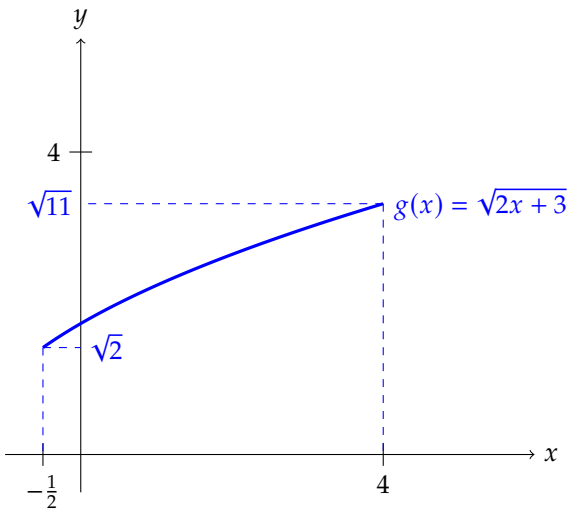
10: See [2] for details.

$$|p^* - p_*| = |g(p^*) - g(p_*)| = |g'(c)| \cdot |p^* - p_*| \leq \rho |p^* - p_*| < |p^* - p_*|.$$

This can only happen if $p^* = p_*$, and so the fixed point is unique. ■

Example Consider the equation $f(x) = x^2 - 2x - 3 = 0$, $x \in \left[-\frac{1}{2}, 4\right]$, and the equivalent fixed point equation $x = g(x) = \sqrt{2x+3}$. Show that g has a unique fixed point, and so that f has a unique root, in $\left[-\frac{1}{2}, 4\right]$.

Solution: any fixed point of g satisfies $x = \sqrt{2x+3} \implies x^2 - 2x - 3 = 0$, and thus is a root of f. Over the interval $\left[-\frac{1}{2}, 4\right]$, g is continuous and increasing, as shown below.



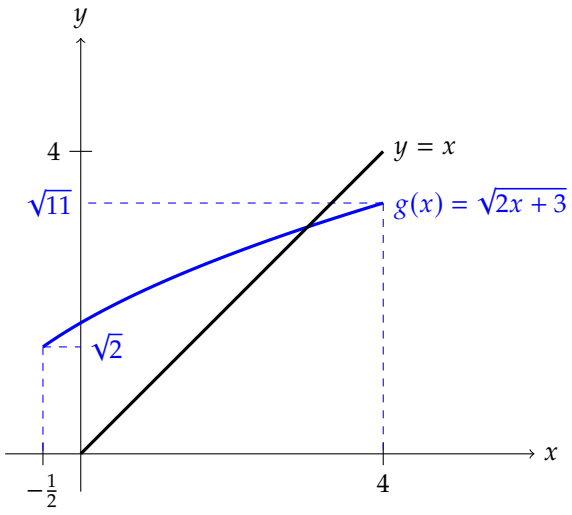
Thus, for any $x \in \left[-\frac{1}{2}, 4\right]$, we have:

$$-\tfrac{1}{2} \le \sqrt{2} \le g\left(-\tfrac{1}{2}\right) \le g(x) \le g(4) \le \sqrt{11} \le 4 \implies g\left(\left[-\tfrac{1}{2}, 4\right]\right) \subseteq \left[-\tfrac{1}{2}, 4\right].$$

Since $g'(x) = \frac{1}{\sqrt{2x+3}}$, then we also have:

$$|g'(x)| \le \frac{1}{\sqrt{2}} < 1, \quad \text{over} \left[-\frac{1}{2}, 4\right].$$

As the assumptions of the theorem are satisfied, g admits a unique fixed point over $\left[-\frac{1}{2}, 4\right]$.



For a given continuous function g on $[a, b]$ and initial iterate x_0, the **fixed point iteration** process reads as:

$$x_k = g(x_{k-1}), \quad k \geq 1.$$

If $\{x_k\}$ converges to some $x_* \in [a, b]$, then x_* is a fixed point of g;[11] indeed,

$$x_* = \lim_{k\to\infty} x_k = \lim_{k\to\infty} g(x_{k-1}) = g\Big(\lim_{k\to\infty} x_{k-1}\Big) = g(x_*).$$

11: Note that the fixed point is not necessarily unique.

Illustration of the Fixed Point Procedure Consider the problem of solving the equation

$$f(x) = x + \ln(1 + x) - 2 = 0, \quad x \in [0, 5],$$

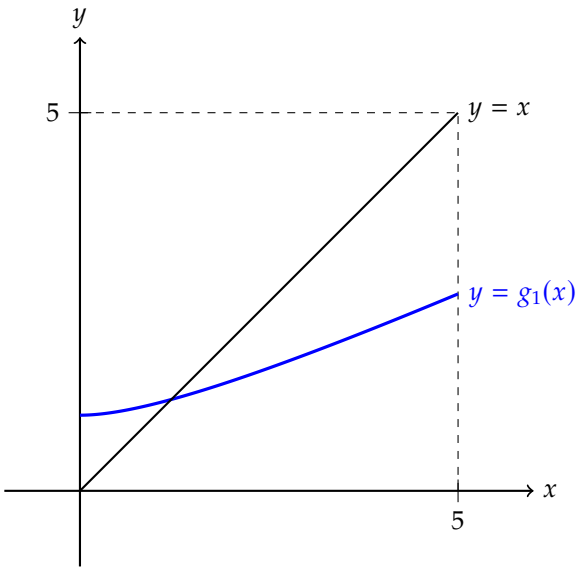
and the three equivalent fixed point equations:

1. $x = g_1(x) = x - \frac{1}{2}[x + \ln(1 + x) - 2]$
2. $x = g_2(x) = 2 - \ln(1 + x)$
3. $x = g_3(x) = e^{2-x} - 1$

We provide a detailed illustration of how the method works on g_1; for g_2 and g_3, we only show the final picture.[12]

12: Is it clear that all the fixed point problems are equivalent to the root-finding problem?

First, we plot the graphs of $y = g_1(x)$ and $y = x$; any intersection of the two curves over the domain $[a, b](= [0, 5])$ must satisfy $g_1(x) = x$ and so is a fixed point of g_1 over the domain.



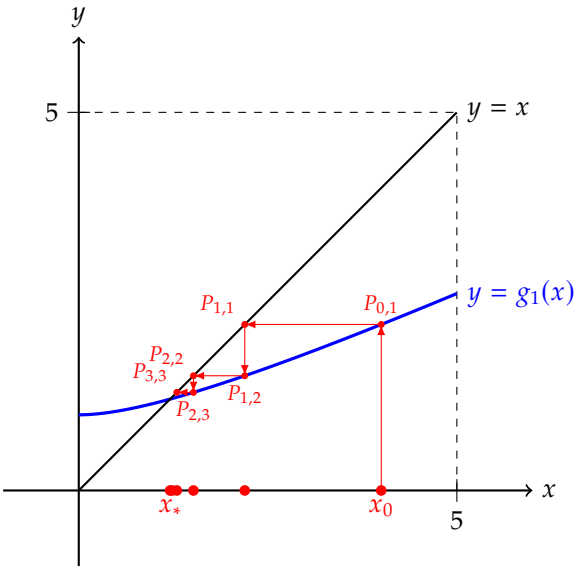
Graphically, we see that there is one such fixed point. How does the procedure find it?

We need an x_0 in the domain; we select $x_0 = 4$, for no particular reason, and we obtain:

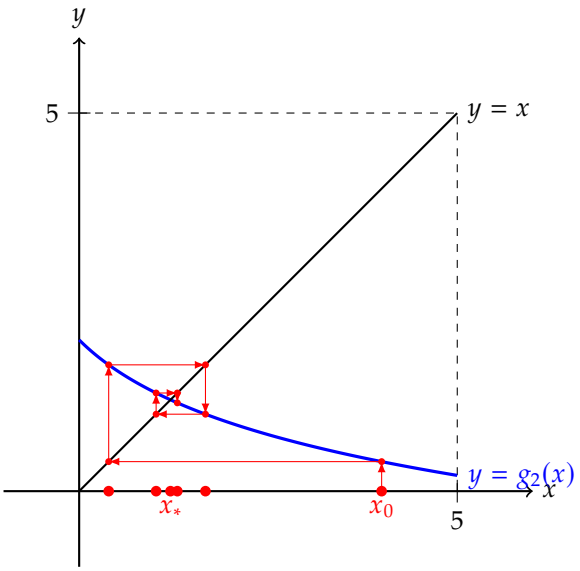
$$x_1 = g_1(x_0) = g_1(4) = 4 - \tfrac{1}{2}[4 + \ln(1 + 4) - 2] \approx 2.195281;$$
$$x_2 = g_1(x_1) = g_1(2.195281) \approx 1.516803;$$
$$x_3 = g_1(x_2) = g_1(1.516803) \approx 1.296907, \quad \text{etc.}$$

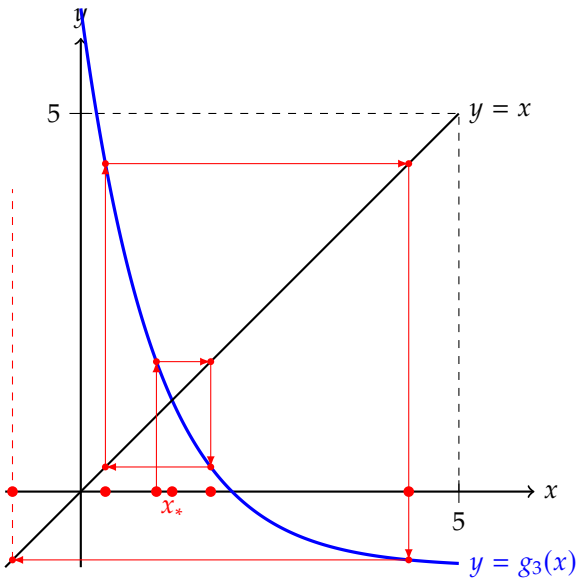
For $k \geq 1$, each x_k plays two roles: it is the $y-$coordinate of a point on the curve $y = g_1(x)$, which becomes the $x-$coordinate of a point on the curve $y = x$, which is then fed back into g_1, and so on.

Graphically, this is represented as a rectangular "curve" which converges to the point $(x_*, x_*) \approx (1.20794, 1.20794)$ in a manner resembling a staircase; the label $P_{i,j}$ represents the point with coordinates (x_i, x_j).



With g_2 and g_3, the fixed point iterations instead take on the following forms.





We see that the method converges for g_1 and g_2, both to the same fixed point x_*, but not for g_3, even though x_* is a fixed point for the latter. Note that $|g_i'(x_*)| < 1$ for $i = 1, 2$, while $|g_3'(x_*)| > 1$.

Convergence So when can we be sure that fixed point iteration converges to a fixed point?

Fixed Point Theorem (Reprise): let $g : [a, b] \to \mathbb{R}$ be a function satisfying hypotheses 1. and 2. of the fixed point theorem of page 193. Then for any initial iterate $x_0 \in [a, b]$, the sequence $\{x_k\}$ defined by

$$x_k = g(x_{k-1}), \quad k \geq 1,$$

converges to the unique fixed point x_* of g in $[a, b]$.

Proof: the original fixed point theorem shows that g has a unique fixed point x_* in $[a, b]$. Let $x_0 \in [a, b]$; we must show that $x_k \to x_*$ as $k \to \infty$.

On the one hand, we have $x_k - x_* = g(x_{k-1}) - g(x_*)$ for all $k \geq 1$. On the other hand, since g is differentiable over (a, b), the mean value theorem implies that

$$g(x_{k-1}) - g(x_*) = g'(c_k)(x_{k-1} - x_*), \quad \text{for some } c_k \text{ between } x_{k-1} \text{ and } x_*.$$

Thus,

$$|x_k - x_*| = |g(x_{k-1}) - g(x_*)| = |g'(c_k)||x_{k-1} - x_*| \leq \rho|x_{k-1} - x_*|,$$

by hypothesis. We then have, recursively,

$$|x_k - x_*| \leq \rho|x_{k-1} - x_*| \leq \rho^2|x_{k-2} - x_*| \leq \cdots \leq \rho^k|x_0 - x_*| \to 0$$

as $k \to \infty$ since $\rho < 1$, which completes the proof. ■

Corollary on the Error Estimates: under the hypotheses of the fixed point theorem, we can show that:

1. $|x_k - x_*| \leq \rho^k \cdot \max\{x_0 - a, b - x_0\}$ for $k \geq 0$;
2. $|x_k - x_*| \leq \frac{\rho^k}{1-\rho} \cdot \max\{x_0 - a, b - x_0\}$ for $k \geq 1$.

Note that the smaller the value $\rho < 1$ is, the faster the sequence converges to the fixed point x_* of g.

Stopping Criterion Ideally, we would like the fixed point procedure to stop whenever the error satisfies $e_k = |x_k - x_*| < \text{tol}$ for some prescribed tolerance $\text{tol} > 0$. However, the exact fixed point x_* is not known; instead, we can use the following **stopping criterion**:

$$|x_{k+1} - x_k| < \text{tol}.$$

The value $r_k = |x_k - g(x_k)| = |x_k - x_{k+1}|$ is the **residual** of the fixed point procedure at step k. Note that $r_k \approx \text{tol}$ does not imply that $e_k \approx \text{tol}$:

$$\begin{aligned} x_k - x_* &= x_k - x_{k+1} + x_{k+1} - x_* \\ &= x_k - x_{k+1} + g(x_k) - g(x_*) \\ &= x_k - x_{k+1} + g'(c_k)(x_k - x_*), \quad \text{for some } c_k \text{ between } x_k \text{ and } x_*, \end{aligned}$$

so that

$$(1 - g'(c_k))(x_k - x_*) = x_k - x_{k+1} \implies e_k = \frac{r_k}{|1 - g'(c_k)|}.$$

If $|g'(c_k)| \ll 1$, then $e_k \approx r_k \approx \text{tol}$; if $g'(c_k) \approx 1$, then it is possible that $e_k \gg \text{tol}$!

The fixed point iteration is summarized in the following algorithm.

Algorithm: fixed point iteration

Input: g with the appropriate properties on $[a, b]$, x_0, tol > 0, $N_{\max}$
Output: approximation p of a fixed point x_* of g, number of iterations n

1 **Initialization:** $x_1 = g(x_0)$, $r_0 = |x_0 - x_1|$, $k = 0$;
2 **While** $r_k >$ tol and $k < N_{\max}$ **do**
3 $k = k + 1$;
4 $x_{k+1} = g(x_k)$;
5 $r_k = |x_k - x_{k+1}|$;
6 **End**
7 $p = x_{k+1}$, $n = k + 1$.

Here is an implementation of the method in R.

Fixed point method

```
fixed_point <- function(g, x0, tol, Nmax) {
  # initialization
  x_old <- x0
  x <- g(x_old)
  res <- abs(x - x_old)

  k <- 1
  x_vec <- c(x0, x)

  # fixed point iteration
  while (res > tol && k < Nmax) {
    k <- k + 1
    x_old <- x
    x <- g(x_old)
    res <- abs(x - x_old)
    x_vec <- c(x_vec, x)

    # tolerance not reached
    if (k == Nmax && res > tol) {
      cat('Nmax iterations reached without
       satisfying the prescribed tolerance\n')
    }
  }

  return(list(x = x, k = k, x_vec = x_vec))
}
```

Example We can find the fixed point x_* of $g(x) = -\cos(x)$ with tol $=$ 0.00005 as follows.

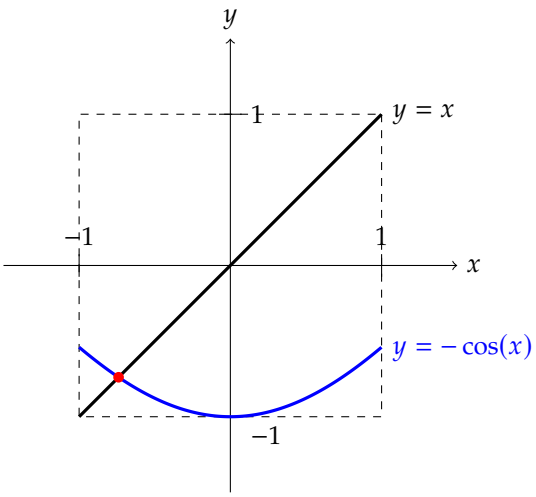
```
g.test <- function(x){ -cos(x) }
fixed_point(g.test, 1, 0.00005, 300)
```

```
$x
[1] -0.7390714

$k
[1] 25

$x_vec
 [1]  1.0000000 -0.5403023 -0.8575532 -0.6542898
 [5] -0.7934804 -0.7013688 -0.7639597 -0.7221024
 [9] -0.7504178 -0.7314040 -0.7442374 -0.7356047
[13] -0.7414251 -0.7375069 -0.7401473 -0.7383692
[17] -0.7395672 -0.7387603 -0.7393039 -0.7389378
[21] -0.7391844 -0.7390183 -0.7391302 -0.7390548
[25] -0.7391056 -0.7390714
```

We can easily verify that the output is at the very least quite near x_*, numerically and graphically.

```
g.test(-0.7390714)+0.7390714
```

```
[1] -2.2984e-05
```



Order of the Method In the proof of the fixed point theorem (reprise), we saw that

$$|g(x_k) - g(x_*)| = |g'(c_k)||x_k - x_*|$$

for some c_k between x_k and x^*.

If g, g' are continuous over $[a, b]$ and $\lim_{k\to\infty} x_k = x_*$, where $g(x_*) = x_*$, then we see that

$$\lim_{k\to\infty} \frac{|x_{k+1} - x_*|}{|x_k - x_*|} = \lim_{k\to\infty} \frac{|g(x_k) - g(x_*)|}{|x_k - x_*|} = \lim_{k\to\infty} |g'(c_k)| = |g'(x_*)|,$$

since $0 \le |x_* - c_k| \le |x_* - x_k| \to 0$ and g' is continuous.

Thus, $|g'(x_*)|$ provides a measure of the **speed of convergence** of the sequence $\{x_k\}$.

Let $x_k \to x_*$ be such that $x_k \neq x_*$ for all k.

1. If
$$\lim_{k\to\infty} \frac{|x_{k+1} - x_*|}{|x_k - x_*|} = \lambda \in (0, 1),$$
then $\{x_k\}$ converges **linearly** to x_*.

2. If
$$\lim_{k\to\infty} \frac{|x_{k+1} - x_*|}{|x_k - x_*|} = 1,$$
then $\{x_k\}$ converges **sublinearly** to x_*.

3. If
$$\lim_{k\to\infty} \frac{|x_{k+1} - x_*|}{|x_k - x_*|} = 0,$$
then $\{x_k\}$ converges **superlinearly** to x_*.

4. Set $\alpha \geq 1$ an integer; if
$$\lim_{k\to\infty} \frac{|x_{k+1} - x_*|}{|x_k - x_*|^\alpha} = \lambda > 0,$$
then $\{x_k\}$ converges to x_* **with order** α; in this case, the value λ is known as the **asymptotic error constant**.[13]

13: If $\alpha = 1$, the convergence is linear and we must have $\lambda < 1$; if $\alpha = 2$, the convergence is **quadratic**.

We say that a fixed point iteration $x_k = g(x_{k-1})$ is **of order** α if $\{x_k\}$ converges to a fixed point x^* with order α. In that case, when x_k is sufficiently close to x^* then we have

$$|x_{k+1} - x_*| \approx \lambda |x_k - x_*|^\alpha.$$

Example Assume that we have two fixed point iterations, one with order $\alpha = 1$ and $\lambda = 0.5$, and the other with order $\alpha = 2$ and $\lambda = 1$. Moreover, suppose that $|x_0 - x^*| = 10^{-1}$. Then we would expect to observe something like the following table.

k	$\lvert x_k - x_*\rvert$	$\alpha = 1, \lambda = 0.5$	$\alpha = 2, \lambda = 1$
0	$\lvert x_0 - x_*\rvert$	0.1	0.1
1	$\lvert x_1 - x_*\rvert$	0.05	0.01
2	$\lvert x_2 - x_*\rvert$	0.025	0.0001
3	$\lvert x_3 - x_*\rvert$	0.0125	0.00000001
⋮	⋮	⋮	⋮

In both cases, $e_k \to 0$; the convergence is quicker in the second case.[14]

14: In practice, this means that we will not need as many iterations of the fixed point procedure before exiting the 'while' loop in the algorithm.

As mentioned above, the exact fixed point x_* is not known, and so we cannot compute the absolute error $e_k = |x_k - x^*|$ exactly. Instead, we estimate the order α of a fixed point iteration with the help of the residual $r_k = |x_k - x_{k+1}|$ and search for the value of α for which that the ratio r_{k+1}/r_k^α converges to a positive constant.

We already know the relationship between r_k and e_k: $r_k = |1 - g'(c_k)|e_k$ for some c_k between x_k and x_*; so if $\dfrac{e_{k+1}}{e_k^\alpha} \to \lambda > 0$, then

$$\lim_{k\to\infty} \frac{r_{k+1}}{r_k^\alpha} = \lim_{k\to\infty} \frac{|1 - g'(c_{k+1})|}{|1 - g'(c_k)|^\alpha} \cdot \frac{e_{k+1}}{e_k^\alpha} = \frac{\lambda}{|1 - g'(x_*)|^{\alpha-1}} > 0.$$

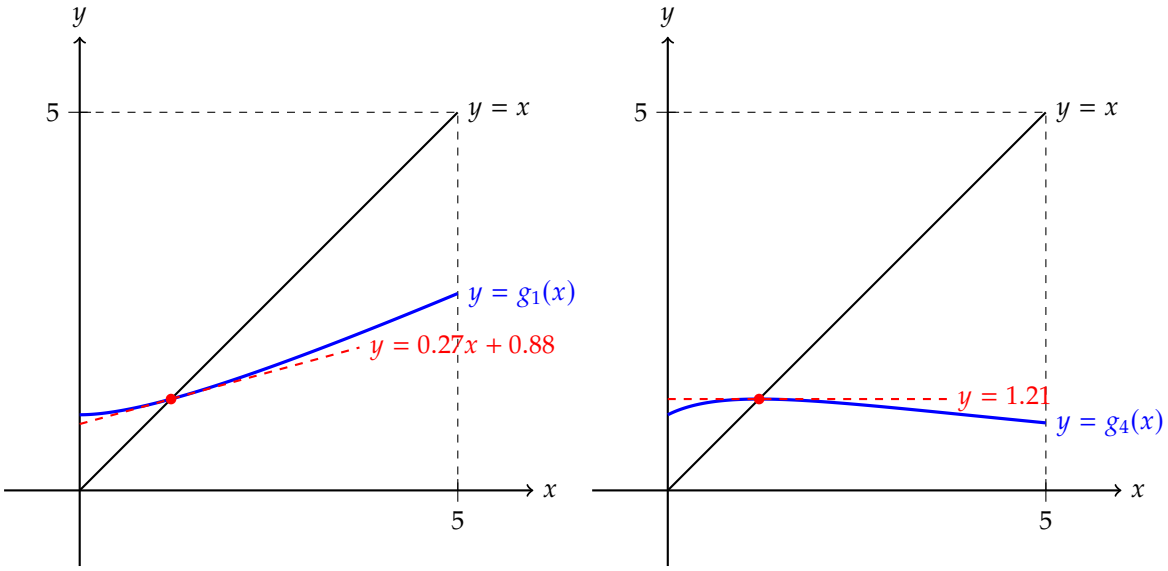
Example Consider again the equation

$$f(x) = x + \ln(1 + x) - 2 = 0, \quad x \in [0, 5],$$

and the equivalent fixed point equations $x = g_i(x)$, $i = 1, 4$, with

$$g_1(x) = x - \frac{1}{2}[x + \ln(1 + x) - 2] \quad \text{and} \quad g_4(x) = \frac{3x + 2 - (1 + x)\ln(1 + x)}{2 + x}.$$

The charts are shown below, with their tangent lines at (x_*, x_*).



In both cases, the derivative at the fixed point falls in $(-1, 1)$, so the fixed point procedure converges for every initial iterate $x_0 \in [0, 5]$; note, however, that $|g_4'(x_*)| < |g_1'(x_*)|$, so we expect the convergence to the fixed point to be of higher order for g_4 than for g_1.

We run the algorithm with $x_0 = 4$ and tol $= 10^{-8}$.

```
g1 <- function(x){x-0.5*(x + log(x+1) - 2)}
x0 = 2
tol = 10^(-8)
Nmax = 1000
fp1 = fixed_point(g1, x0, tol, Nmax)
n1 = length(fp1$x_vec)
```

We compute the residuals, and study the ratios r_{k+1}/r_k:

```
res1 = abs(fp1$x_vec[2:n1] - fp1$x_vec[1:(n1-1)] )
res1[2:(n1-1)]/res1[1:(n1-2)]
```

```
[1] 0.3159122 0.2883925 0.2779797 0.2747903
[5] 0.2738879 0.2736387 0.2735703 ...
```

We see that the sequence of ratios seems to converge to $\lambda_1 = 0.2735... > 0$, and so the fixed point convergence is at least linear. For comparison's sake, we also take a look at the ratios r_{k+1}/r_k^2:

```
res1[2:(n1-1)]/(res1[1:(n1-2)])^2
```

```
[1]   0.5751114   1.6618933   5.5545402  19.7525620
[5]  71.6462604 261.3517413 954.8594146  ...
```

The sequence of ratios does not seem to converge.

If we repeat the above commands for g_4, we find that the fixed point iteration with g_4 is of order 2.

```
g4 <- function(x){(3*x+2-(1+x)*log(1+x))/(2+x)}
x0 = 2; tol = 10^(-8); Nmax = 1000;
fp4 = fixed_point(g4, x0, tol, Nmax)
n4 = length(fp4$x_vec)
```

We compute the residuals, and study the ratios r_{k+1}/r_k:

```
res4 = abs(fp4$x_vec[2:n4] - fp4$x_vec[1:(n4-1)] )
res4[2:(n4-1)]/res4[1:(n4-2)]
```

```
[1] 3.862611e-02 2.290711e-03 5.146737e-06 ...
```

We see that the sequence of ratios seems to converge to $\lambda_4 = 0$. We take a look at the ratios r_{k+1}/r_k^2:

```
res4[2:(n4-1)]/(res4[1:(n4-2)])^2
```

```
[1] 0.04687867 0.07197530 0.07059517 ...
```

These ratios do seem to converge to a non-zero λ_4, so the convergence is at least of order 2. And for r_{k+1}/r_k^3?

```
res4[2:(n4-1)]/(res4[1:(n4-2)])^2
```

```
[1]   0.05689441   2.26150082 968.31804790 ...
```

The sequence of ratios does not seem to converge.

In general, the order of the convergence to a fixed point x_* of g is linked to the order of differentiability of g at x_*.

Theorem: let $g \in C^{\alpha}([a,b])$, $\alpha \geq 1$ an integer, and let $x^* \in [a,b]$ be a fixed point of g, with x_0 sufficiently near x_*. If

$$0 < |g'(x^*)| < 1,$$

then the fixed point iteration $x_k = g(x_{k-1})$, $k \geq 1$, is only of order 1. If

$$g'(x_*) = g''(x_*) = \cdots = g^{(\alpha-1)}(x_*) = 0 \quad \text{and} \quad g^{(\alpha)}(x_*) \neq 0,$$

then the fixed point iteration is of order α.

Proof: we only provide an outline for the case $\alpha > 1$. For any $x, x_0 \in [a,b]$, with x_0 "sufficiently close" to x, we apply **Taylor's theorem** to g,[15] around its fixed point $x_* \in [a,b]$, and write

15: See [2] for details.

$$\begin{aligned} g(x) &= g(x_*) + g'(x_*)(x-x_*) + \frac{1}{2}g''(x_*)(x-x_*)^2 + \cdots + \frac{1}{(\alpha-1)!}g^{(\alpha-1)}(x_*)(x-x_*)^{\alpha-1} + \frac{1}{\alpha!}g^{(\alpha)}(c_x)(x-x_*)^{\alpha} \\ &= g(x_*) + \frac{1}{\alpha!}g^{(\alpha)}(c_x)(x-x_*)^{\alpha}, \end{aligned}$$

for some c_x between x and x_*.[16] When $x = x_k$, we get

16: The mean value theorem is a special case of Taylor's theorem, with $\alpha = 1$.

$$x_{k+1} - x_* = g(x_k) - g(x_*) = \frac{1}{\alpha!}g^{(\alpha)}(c_x)(x-x_*)^{\alpha},$$

where c_k lies between x_k and x_*. Since $x_k \to x_*$, then $c_k \to x_*$ and

$$\lim_{k\to\infty} \frac{|x_{k+1}-x_*|}{|x_k - x_*|^{\alpha}} = \lim_{k\to\infty} \frac{1}{\alpha!}|g^{(\alpha)}(c_k)| = \frac{1}{\alpha!}|g^{(\alpha)}(x_*)|,$$

which is non-zero, by assumption. ■

This explains why some choices of g are better than others; of course, this is of limited applicability as we need to know x_* before we can use this last result to increase the convergence order of the procedure... but if we already know x_*, there is no need to improve the speed of convergence.

4.2.4 Newton's Method

Newton's method is one of the most frequently-used "fast" method for solving **nonlinear equations**, although in many applications, it is often supplanted by task-specific methods, such as **gradient descent methods**.[17]

17: See Chapters 5 and 31, and Section 4.3.2.

We wish to solve the equation $f(x) = 0$, with $f \in C^2([a,b])$. Assume that $x^* \in [a,b]$ is a root of f and let $x_k \in [a,b]$. According to Taylor's theorem, there is a c_k between x^* and x_k such that

$$0 = f(x^*) = f(x_k) + f'(x_k)(x^* - x) + \frac{1}{2}f''(c_k)(x^* - x_k)^2.$$

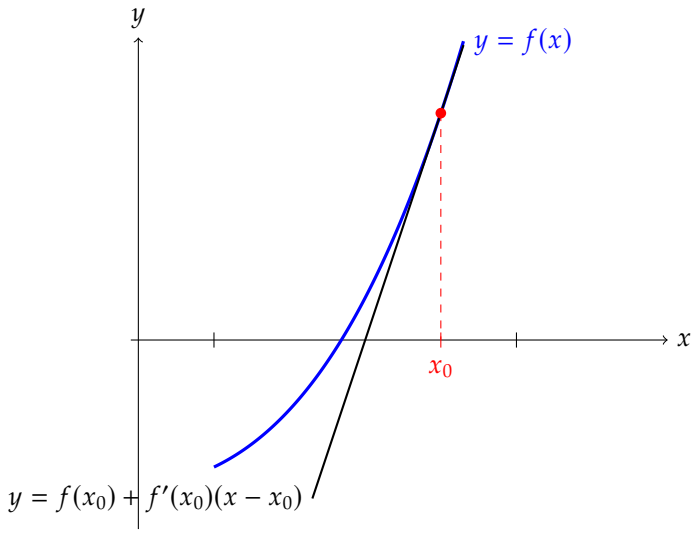
If x_k is near x^*, we expect $|x^* - x_k|$ to be small, so that $|x^* - x_k|^2 \ll |x^* - x_k|$. Moreover, if $f'(x_k) \neq 0$, then

$$0 = f(x^*) \approx f(x_k) + f'(x_k)(x^* - x) \implies x^* \approx x_k - \frac{f(x_k)}{f'(x_k)}.$$

Starting from x_0, **Newton's method** generates the sequence $\{x_k\}$ defined by

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k \geq 0.$$

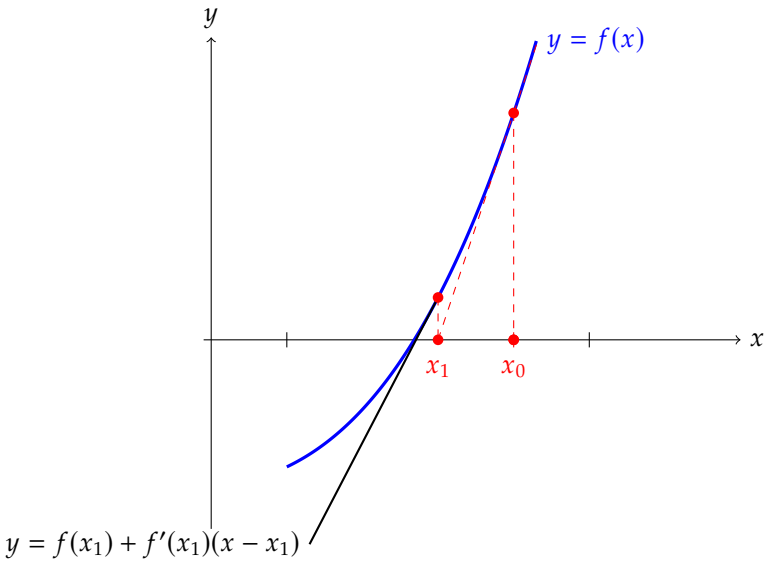
Illustration of the Method Let $f : [a, b] \to \mathbb{R}$ be the C^2 function whose graph is displayed below, and let $x_0 \in [a, b]$ be near x^*. Draw the tangent to f at x_0.



The equation of the tangent is $y = f(x_0) + f'(x_0)(x - x_0)$; the intersection of the line with the $x-$axis at

$$0 = f(x_0) + f'(x_0)(x - x_0) \implies x = x_0 - \frac{f(x_0)}{f'(x_0)},$$

which is exactly the first Newton iterate x_1. Repeat this procedure starting from x_1 to obtain x_2, and so on.



Theorem: let $f \in C^2([a,b])$. If $x^* \in [a,b]$ is such that $f(x^*) = 0$ and $f'(x^*) \neq 0$, then the sequence $\{x_k\}$ generated by Newton's method converges (at least) quadratically to x^* for any x_0 sufficiently near x^*.

Proof: Newton's method can be recast as a fixed point iteration for the function defined by $g(x) = x - \frac{f(x)}{f'(x)}$. At $x = x^*$,

$$g(x^*) = x^* - \frac{f(x^*)}{f'(x^*)} = x^* - \frac{0}{f'(x^*)} = x^*,$$

so x^* is a fixed point of g. But

$$g'(x) = 1 - \frac{[f'(x)]^2 - f(x)f''(x)}{[f'(x)]^2} = \frac{f(x)f''(x)}{[f'(x)]^2} \implies g'(x^*) = \frac{f(x^*)f''(x^*)}{[f'(x^*)]^2} = 0,$$

so the order of convergence is at least $\alpha = 2$ according to the last theorem of Section 4.2.3. ■

Newton's method may not converge if x_0 is too removed from x^*, or if the iterations gets caught in a cycle.

Remark: if $f'(x^*) = 0$, then Newton's method may still converge with order 1. For instance, $f(x) = x^2$ vanishes at $x^* = 0$ and $f'(x^*) = 0$. We then have

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{x_k^2}{2x_k} = \frac{1}{2}x_k = \cdots = \left(\frac{1}{2}\right)^k x_0 \to 0 = x^*$$

as $k \to \infty$, and so $x_k \to x^*$. However, for the equivalent fixed point problem $x = g(x) = x/2$, we have $g(x^*) = 0$ and $g'(x^*) = 1/2 \neq 0$, so the convergence is only linear.

Newton's algorithm is summarized in the following algorithm.

Algorithm: Newton's method

Input: f, f', x_0, tol > 0, $N_{\max}$
Output: approximation p of a root x^* of f, number of iterations n

1 **Initialization:** $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$, $r_0 = |x_1 - x_0|$, $k = 0$;
2 **While** $r_k >$ tol and $k < N_{\max}$ **do**
3 $k = k + 1$;
4 $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$;
5 $r_k = |x_{k+1} - x_k|$;
6 **End**
7 $p = x_{k+1}$, $n = k + 1$.

Here is an implementation of the method in R.

Newton's method

```
newton <- function(f, df, x0, tol, Nmax) {
  x_old <- x0
  x <- x_old - f(x_old) / df(x_old)
  res <- abs(x - x_old)
```

```
  k <- 1
  x_vec <- c(x0, x)

  while(res > tol && k < Nmax) {
    k <- k + 1
    x_old <- x
    x <- x_old - f(x_old) / df(x_old)
    res <- abs(x - x_old)

    x_vec <- c(x_vec, x)

    if(k == Nmax && res > tol) {
      cat('Nmax iterations reached without
       satisfying the prescribed tolerance\n')
    }
  }

  return(list(x=x, k=k, x_vec=x_vec))
}
```

Example We are looking for roots of the function f defined by $f(x) = x^2 - 4$, whose derivative is f' defined by $f'(x) = 2x$.

```
f <- function(x){x^2 - 4}
df <- function(x){2*x}
```

We initialize the algorithm as follows.

```
x0 <- 1
tol <- 1e-5
Nmax <- 100
```

What does Newton's method find?

```
result <- newton(f, df, x0, tol, Nmax)
print(result$x)
```

```
[1] 2
```

```
print(result$k)
```

```
[1] 5
```

```
print(result$x_vec)
```

```
[1] 1.00000 2.50000 2.05000 2.00061 2.00000 2.00000
```

We know, theoretically, that $f(2) = 0$. But $x^* = 2$ is not the only root of f. One of the drawbacks of iterative procedures in the search for roots is that a sequence $\{x_k\}$ converges to one limit (at most).

When we know that there are other roots, we can try playing with the parameters to generate sequences converging to those, but in general that knowledge is not available to us.[18] We can exhibit the other root by using a different x_0.

18: That is, in no small part, exactly why we are looking for roots in the first place.

```
x0 <- -1
result <- newton(f, df, x0, tol, Nmax)
print(result$x)
```

```
[1] -2
```

```
print(result$k)
```

```
[1] 5
```

```
print(result$x_vec)
```

```
[1] -1.00000 -2.50000 -2.05000 -2.00061 -2.00000 -2.00000
```

Newton's method, being of order 2, is usually quite fast, but the function's derivative must be known.

4.2.5 Secant Method

It might be costly to evaluate f'; the **secant method** is a variation of Newton's method where only evaluations of f are needed. The idea is to approximate $f'(x_k)$ by a difference quotient:

$$f'(x_k) = \lim_{x \to x_k} \frac{f(x) - f(x_k)}{x - x_k} \approx \frac{f(x_{k-1}) - f(x_k)}{x_{k-1} - x_k}.$$

The quality of the approximation increases when x_{k-1} is "close" to x_k.

Given initial iterates $x_0 \neq x_1 \in [a, b]$ for which $f(x_0) \neq f(x_1)$, the sequence generated by the secant method is similar to the Newton sequence, but substituting $f'(x_k)$ by $\frac{f(x_{k-1}) - f(x_k)}{x_{k-1} - x_k}$:

$$x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}, \quad k \geq 1.$$

Graphically, we obtain x_2 as the intersection of the $x-$axis with the line joining the points $(x_0, f(x_0))$ and $(x_1, f(x_1))$.

4.3 Systems of Equations

In practice, data problems often give rise to systems of m equations in n unknowns (as opposed to 1 equation in 1 variable). The nature of these systems (**linear** vs. **non-linear**) affects the choice of solution method.[19]

19: Methods derived specifically for linear systems are not easily applicable to non-linear systems, but methods for non-linear systems are usually applicable to linear systems as well.

4.3.1 Linear Systems

In simple linear regression, for instance, we are trying to find the coefficients β_0 and β_1 that "best" fit the data $\{(X_i, Y_i)\}$ in the least square sense: $Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i$, $i = 1, \ldots, n$.

In Chapter 8, we see that the estimators b_0, b_1 are the solutions of

$$n\overline{Y} = n\beta_0 + n\overline{X}\beta_1, \qquad S_{xy} + n\overline{XY} = n\overline{X}\beta_0 + (S_{xx} + n\overline{X}^2)\beta_1.$$

This is a linear system of two equations in two unknowns, which we can re-write in matrix form as $A\boldsymbol{\beta} = \mathbf{c}$. If A is invertible, the estimated solution vector is $A^{-1}\mathbf{c}$.[20]

20: See Chapter 3 for details.

Consider the linear system $A\mathbf{x} = \mathbf{b}$, where A is an $m \times n$ matrix, $\mathbf{x} \in \mathbb{R}^n$, and $\mathbf{b} \in \mathbb{R}^m$. If $m = n$ and A is **invertible**, the system has a unique solution, $\mathbf{x} = A^{-1}\mathbf{b}$.

In practice, we rarely solve the linear system by explicitly computing A^{-1}, especially if n is large.[21]

21: With a computer capable of teraflop speeds, it would take roughly 10^{141} years to compute the inverse of an 100×100 matrix using cofactors or Cramer's rule!

We will briefly discuss two types of methods for solving $A\mathbf{x} = \mathbf{b}$ that do not involve computing A^{-1}: direct methods and iterative methods.

Direct Methods

In theory, a **direct** method finds the exact solution in a finite number of steps; in practice, the solution is "polluted" by round-off error.

Gaussian Elimination and Backward Substitution A linear system may be easy to solve when A has an advantageous structure, such as if it is **upper** (or lower) **triangular**:

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} & a_{1,n} \\ 0 & a_{2,2} & \cdots & a_{2,n-1} & a_{2,n} \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & 0 & 0 & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix}.$$

From the last row $a_{n,n}x_n = b_n$, we obtain $x_n = b_n/a_{n,n}$, assuming that $a_{n,n} \neq 0$.[22]

22: All diagonal entries of a triangular matrix A must be non-zero if A is invertible.

Then, from the penultimate row, we have

$$a_{n-1,n-1}x_{n-1} + a_{n-1,n}x_n = b_{n-1} \implies x_{n-1} = \frac{1}{a_{n-1,n-1}}\left(b_{n-1} - a_{n-1,n}x_n\right),$$

and so on until we reach the first row.

The formal procedure for triangular matrices are provided below.

Algorithm: backward substitution

Input: A upper triangular, $n \times n$, with $a_{i,i} \neq 0$ for all $1 \leq i \leq n$
Output: solution $\mathbf{x}$ of $A\mathbf{x} = \mathbf{b}$

1 **For** $i = n, n-1, \ldots, 1$ **do**

2 $$x_i = \frac{1}{a_{i,i}} \left(b_i - \sum_{j=i+1}^{n} a_{i,j} x_j \right)$$

3 **End**

4 $\mathbf{x} = (x_1, \ldots, x_n)^\top$

Algorithm: forward substitution

Input: A lower triangular, $n \times n$, with $a_{i,i} \neq 0$ for all $1 \leq i \leq n$
Output: solution $\mathbf{x}$ of $A\mathbf{x} = \mathbf{b}$

1 **For** $i = 1, 2, \ldots, n$ **do**

2 $$x_i = \frac{1}{a_{i,i}} \left(b_i - \sum_{j=1}^{i-1} a_{i,j} x_j \right)$$

3 **End**

4 $\mathbf{x} = (x_1, \ldots, x_n)^\top$

In general, the matrix A is not triangular, but it can be brought to a triangular form via **Gaussian elimination**.[23]

23: See Section 3.4.1 for more details.

Example To find the solution of the linear system

$$\begin{cases} x_1 + x_2 + 3x_4 = 4 \\ 2x_1 + x_2 - x_3 + 3x_4 = 1 \\ 3x_1 - x_2 - x_3 + 2x_4 = -3 \\ -x_1 + 2x_2 + 3x_3 - x_4 = 4 \end{cases}$$

we first form the augmented matrix $[A \mid \mathbf{b}]$ and reduce it to its echelon form to obtain

$$\left(\begin{array}{cccc|c} 1 & 1 & 0 & 3 & 4 \\ 0 & -1 & -1 & -5 & -7 \\ 0 & 0 & 3 & 13 & 13 \\ 0 & 0 & 0 & -13 & -13 \end{array}\right).$$

We can read the solution from the reduced matrix directly, *via* backward substitution:

$$\begin{aligned} x_4 &= 13/13 = 1, \\ x_3 &= \frac{1}{3}(13 - 13 \cdot 1) = 0, \\ x_2 &= \frac{1}{-1}(-7 - (-1) \cdot 0 - (-5) \cdot 1) = 2, \\ x_1 &= \frac{1}{1}(4 - 1 \cdot 2 - 0 \cdot 0 - 3 \cdot 1) = -1. \end{aligned}$$

In order to solve a system of n linear equations in n variables, we can show that we need $\mathcal{O}(n^3)$ operations for Gaussian elimination of $[A \mid \mathbf{b}]$, and $\mathcal{O}(n^2)$ operations for backward/forward substitution.[24]

24: We use the "big O" notation $\mathcal{O}(n^k)$ as shorthand for a number of operations $\leq An^k$ for some constant $A > 0$.

LU Factorization If A is invertible, then we can perform Gaussian elimination on it, which also means that it can be factored as

$$A = LU,$$

where L and U are lower and upper square triangular, respectively.[25] In fact, U is the reduced matrix of A (after Gaussian-elimination) and

$$L = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ \ell_{2,1} & 1 & & 0 \\ \vdots & \ddots & \ddots & \\ \ell_{n,1} & \cdots & \ell_{n,n-1} & 1 \end{pmatrix}.$$

25: This assumes that Gaussian elimination can be conducted on A without having to interchange rows, an assumption that we will make throughout this section.

Example In the preceding example, we had

$$A = \begin{pmatrix} 1 & 1 & 0 & 3 \\ 2 & 1 & -1 & 1 \\ 3 & -1 & -1 & 2 \\ -1 & 2 & 3 & -1 \end{pmatrix} \rightsquigarrow U = \begin{pmatrix} 1 & 1 & 0 & 3 \\ 0 & -1 & -1 & -5 \\ 0 & 0 & 3 & 13 \\ 0 & 0 & 0 & -13 \end{pmatrix}.$$

With

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ -1 & -3 & 0 & 1 \end{pmatrix},$$

we indeed have $LU = A$. □

Let $\mathbf{I}_n$ be the $n \times n$ identity matrix, and $\mathbf{M}_n(i, j)$ be the $n \times n$ zero matrix, except in the position (i, j), where the entry is 1. The three types of elementary row transformations that carry A to U can also be written as a left-product of elementary matrices with A:

$$U = E^{(n-1,1)} E^{(n-2,2)} E^{(n-2,1)} \cdots E^{(1,n-1)} \cdots E^{(1,1)} A,$$

where

$$E^{(k,\nu)} = \begin{cases} \mathbf{I_n}[R_i \leftrightarrow R_j], & \nu\text{th operation of step } k \text{ is } R_i \leftrightarrow R_j \\ \mathbf{I_n} + a\mathbf{M}_n(i, j), & \nu\text{th operation of step } k \text{ is } aR_i + R_j \to R_j,\ i > j \\ \mathbf{I_n} + (a-1)\mathbf{M}_n(j, j), & \nu\text{th operation of step } k \text{ is } aR_j \to R_j,\ a \neq 0 \end{cases}$$

Note that $E^{(k,\nu)}$ is always invertible; if no row interchange is required, then $E^{(k,\nu)}$ and $\left[E^{(k,\nu)}\right]^{-1}$ are both lower triangular.

By construction, then

$$A = \left[E^{(1,1)}\right]^{-1} \cdot \left[E^{(1,n-1)}\right]^{-1} \cdots \left[E^{(n-1,1)}\right]^{-1} U = LU,$$

where L is lower diagonal with ones on the diagonal.

Once we have the LU factorization of A, the system $A\mathbf{x} = \mathbf{b}$ can be solved using first forward, then backward substitution:

$$A\mathbf{x} = LU\mathbf{x} = L\mathbf{y} = \mathbf{b}, \quad \text{and then} \quad U\mathbf{x} = \mathbf{y}.$$

The LU factorization approach is particularly useful if we need to solve $A\mathbf{x} = \mathbf{b}$ for different $\mathbf{b}$.[26] It can also be used to speed up determinant computations, since

$$\det(A) = \det(LU) = \det(L)\det(U) = \left(\prod_{i=1}^{n} \ell_{i,i}\right)\left(\prod_{i=1}^{n} u_{i,i}\right) = \left(\prod_{i=1}^{n} u_{i,i}\right).$$

26: We only need $\mathcal{O}(n^3)$ steps for the Gaussian elimination of A once, then $\mathcal{O}(n^2)$ steps for the forward and backward substitution in each system.

Pivoting Strategies When one of the (eventual) **pivot elements** is zero, Gaussian elimination fails because we need access to row interchanges (also known as **pivoting**).[27] But this strategy should also be used when the pivot elements are small in magnitude, relative to the other (reduced) matrix entries, because Gaussian elimination is prone to round-off error.

27: See [7, 5] for details.

Example In exact (symbolic) arithmetic, the matrix form of the linear system

$$\begin{cases} 10^{-20}x_1 + x_2 = 1 \\ x_1 + 2x_2 = 4 \end{cases}$$

reduces to

$$\left(\begin{array}{cc|c} 10^{-20} & 1 & 1 \\ 1 & 2 & 4 \end{array}\right) \rightsquigarrow \left(\begin{array}{cc|c} 10^{-20} & 1 & 1 \\ 0 & 2-10^{20} & 4-10^{20} \end{array}\right),$$

via the row transformation $R_2 - 10^{20}R_1 \to R_2$. Using backward substitution, we then obtain

$$(2-10^{20})x_2 = 4-10^{20} \implies x_2 = \frac{4-10^{20}}{2-10^{20}},$$
$$10^{-20}x_1 = 1-x_2 \implies x_1 = 10^{20}\left(1-\frac{4-10^{20}}{2-10^{20}}\right) = -\frac{2\times 10^{20}}{2-10^{20}};$$

therefore, $x_2 \approx 1$ and $x_1 \approx 2$.

If we are using double precision,[28] we have $2-10^{20} \mapsto -10^{20}$ and $4-10^{20} \mapsto -10^{20}$, and so

$$\left(\begin{array}{cc|c} 10^{-20} & 1 & 1 \\ 1 & 2 & 4 \end{array}\right) \rightsquigarrow \left(\begin{array}{cc|c} 10^{-20} & 1 & 1 \\ 0 & -10^{20} & -10^{20} \end{array}\right),$$

which yields $x_2 = 1$ and $x_1 = 0$. That is problematic!

28: Which is to say, ≈ 16 significant digits.

If we exchange rows 1 and 2 ($R_1 \leftrightarrow R_2$), we obtain instead

$$\left(\begin{array}{cc|c} 1 & 2 & 4 \\ 10^{-20} & 1 & 1 \end{array}\right) \rightsquigarrow \left(\begin{array}{cc|c} 1 & 2 & 4 \\ 0 & 1-2\times 10^{-20} & 1-4\times 10^{-20} \end{array}\right) \mapsto \left(\begin{array}{cc|c} 1 & 2 & 4 \\ 0 & 1 & 1 \end{array}\right),$$

which yields $x_2 = 1$ and $x_1 = 2$.

The elementary matrices in which row interchange are encoded are not lower triangular; an **invertible** matrix A whose Gaussian elimination requires such a transformation does not have an LU decomposition, but it can be decomposed that way up to a **permutation matrix** P:[29]

$$PA = LU.$$

29: A permutation matrix is a matrix whose rows are a permutation of the rows of $\mathbf{I}_n$.

Algorithm: LU factorization with partial pivoting

Input: $n \times n$ matrix $A = (a_{i,j})$
Output: $n \times n$ matrices L, U, P such that $PA = LU$

Initialization: $P = \mathbf{I}_n$;
For $k = 1, 2, \ldots, n-1$ **do**
    Find smallest q such that $|a_{q,k}| = \max_{k \le i \le n} |a_{i,k}|$;
    Exchange rows q and k in A and P;
    For $i = k+1, \ldots, n$ **do**
        Set $a_{i,k} = a_{i,k}/a_{k,k}$;
        For $j = k+1, \ldots, n$ **do**
            Set $a_{i,j} = a_{i,j} - a_{i,k}a_{k,j}$;
        End
    End
End
$L = \mathbf{I}_n$ + strictly lower triangular(A); U = upper triangular(A); P

Once we have P, L, U such that $PA = LU$, then we can solve the system $A\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$ by using

$$A\mathbf{x} = \mathbf{b} \iff PA\mathbf{x} = P\mathbf{b} \iff LU\mathbf{x} = P\mathbf{b},$$

namely, we first solve $L\mathbf{y} = P\mathbf{b}$ using forward substitution, then we solve $U\mathbf{x} = \mathbf{y}$ using backward substitution.

Example Algorithm 6 is implemented in `R` *via* the `Matrix` package's function `lu()`. We use it to find the partial pivoting LU decomposition of

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 4 & 6 \end{pmatrix}.$$

We start by loading the matrix.

```
require(Matrix)
A=t(matrix(c(1,2,3,2,4,5,3,4,6),3,3))
```

We can decompose and extract the factors of the LU decomposition as follows:

```
D <- lu(A)
expand(D)$L
```

```
3 x 3 Matrix of class "dtrMatrix" (unitriangular)
     [,1]      [,2]      [,3]
[1,] 1.0000000         .         .
[2,] 0.6666667 1.0000000         .
[3,] 0.3333333 0.5000000 1.0000000
```

```
expand(D)$U
```

```
3 x 3 Matrix of class "dtrMatrix"
     [,1]     [,2]     [,3]
[1,] 3.000000 4.000000 6.000000
[2,]        . 1.333333 1.000000
[3,]        .        . 0.500000
```

```
expand(D)$P
```

```
3 x 3 sparse Matrix of class "pMatrix"

[1,] . . |
[2,] . | .
[3,] | . .
```

Other (mostly similar) factorizations may be better suited to various types of matrices A:

- for **symmetric** matrices A,[30] we use $A = LDL^\top$, where D is a diagonal matrix;
- for **symmetric positive definite** matrices A,[31] we use the **Cholesky decomposition** $A = MM^\top$;
- it may be possible to take advantage of some **sparse matrices**' structure (such as is the case for **banded matrices**) to greatly increase the speed of the LU decomposition with partial pivoting.

30: $A^\top = A$

31: A symmetric and $\mathbf{x}^\top A\mathbf{x} > 0$ for all $\mathbf{x} \neq \mathbf{0}$.

Matrix Norms A **vector norm** $\|-\| : \mathbb{R}^n \to \mathbb{R}_0^+$ is a function satisfying the following three conditions:

1. $\|\mathbf{x}\| \geq 0$ for all $\mathbf{x} \in \mathbb{R}^n$, and $\|\mathbf{x}\| = 0$ if and only if $\mathbf{x} = \mathbf{0}$;
2. $\|\alpha\mathbf{x}\| = |\alpha|\|\mathbf{x}\|$ for all $\mathbf{x} \in \mathbb{R}^n$, $\alpha \in \mathbb{R}$;
3. $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$.

The 2−norm $\|\mathbf{x}\|_2 = \sqrt{x_1^2 + \cdots + x_n^2}$ is a common example.[32]

32: We can show that all vector norms on $\mathbb{R}^n$ are equivalent, suggesting that there is no real advantage to selecting one over another, in a general setting (although there may be instances where calculations are simpler in one context over another).

Given a vector norm $\|\cdot\|$ on $\mathbb{R}^n$, we can define the **induced matrix norm** $|||-|||$ on the space of $n \times n$ matrices by

$$|||A||| = \sup_{\mathbf{x} \neq \mathbf{0}} \left\{ \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} \right\},$$

where $\mathbf{x}$ ranges over $\mathbb{R}^n$.[33]

33: The properties of vector norms also apply to matrix norms – matrices are the vectors of the space of square matrices, with matrix addition and multiplication by a scalar.

That $|||A||| \geq 0$ is a direct consequence of the definition of the supremum and because $\|A\mathbf{x}\|, \|\mathbf{x}\| \geq 0$ for all $\mathbf{x}$.

If $|||A||| = 0$, then $\|A\mathbf{x}\| \leq 0$ for all $\mathbf{x} \neq \mathbf{0}$; since $\|A\mathbf{x}\| \geq 0$ for all $\mathbf{x} \neq \mathbf{0}$, then $\|A\mathbf{x}\| = 0$ for all $\mathbf{x}$. As $\|A\mathbf{0}\| = 0$, then $\|A\mathbf{x}\| = 0$ for all $\mathbf{x}$. In particular, $\|A\mathbf{e}_k\| = \|A_k\| = 0$ for $1 \leq k \leq n$, so that every column $A_k = \mathbf{0}$; hence $A = \mathbf{O}_{n\times n}$. Conversely, if $A = \mathbf{O}_{n\times n}$, then $\|A\mathbf{x}\| = \|\mathbf{0}\| = 0$ for all $\mathbf{x}$, so that $\|A\mathbf{x}\|/\|\mathbf{x}\| = 0$ for all $\mathbf{x} \neq 0$; hence $|||A||| \leq 0$. Since $|||A||| \geq 0$, we must have $|||A||| = 0$.[34]

34: Properties 2 and 3 are left as exercises.

Theorem: let $|||\cdot|||$ be the matrix norm induced by a vector norm $\|\cdot\|$. Then:

1. $\|A\mathbf{x}\| \leq |||A||| \cdot \|\mathbf{x}\|$ for all A, $\mathbf{x}$;
2. $|||\mathbf{I}_n||| = 1$;
3. $|||AB||| \leq |||A||| \cdot |||B|||$ for all A, B.

Proof: throughout, let A, B be generic $n \times n$ matrices, and $\mathbf{x} \in \mathbb{R}^n$.

1. If $\mathbf{x} = \mathbf{0}$, then the property holds as both sides are 0. Now assume that $\mathbf{x} \neq \mathbf{0}$. By definition,

$$|||A||| \geq \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} \quad \text{for all } \mathbf{x} \neq \mathbf{0};$$

   thus $|||A||| \cdot \|\mathbf{x}\| \geq \|A\mathbf{x}\|$ for all $\mathbf{x} \neq \mathbf{0}$.
2. For any $\mathbf{x} \neq \mathbf{0}$, we have $\|\mathbf{I}_n\mathbf{x}\|/\|\mathbf{x}\| = 1$, so

$$|||\mathbf{I}_n||| = \sup_{\mathbf{x}\neq\mathbf{0}} \left\{ \frac{\|\mathbf{I}_n\mathbf{x}\|}{\|\mathbf{x}\|} \right\} = \sup_{\mathbf{x}\neq\mathbf{0}} \{1\} = 1.$$

3. For any $\mathbf{x} \neq \mathbf{0}$, we see that

$$\|AB\mathbf{x}\| \leq |||A||| \cdot \|B\mathbf{x}\| \leq |||A||| \cdot |||B||| \cdot \|\mathbf{x}\|;$$

   hence

$$|||AB||| = \sup_{\mathbf{x}\neq\mathbf{0}} \left\{ \frac{\|AB\mathbf{x}\|}{\|\mathbf{x}\|} \right\} \leq |||A||| \cdot |||B|||,$$

   which completes the proof.[35] ■

35: See [2] for more information on the supremum.

The ℓ_p vector norm $\|\cdot\|_p$ on $\mathbb{R}^n$ is trivial to compute: for $p \geq 1$, we have

$$\|\mathbf{x}\|_p = \sqrt[p]{|x_1|^p + \cdots + |x_n|^p};$$

for $p = \infty$ we have

$$\|\mathbf{x}\|_\infty = \max_{1\leq k\leq n} |x_k|.$$

It is not as clear how we would compute the corresponding induced matrix norm; we can show that

$$|||A|||_1 = \max_{1\leq j\leq n} \left\{ \sum_{i=1}^{n} |a_{i,j}| \right\};$$

$$|||A|||_\infty = \max_{1\leq i\leq n} \left\{ \sum_{j=1}^{n} |a_{i,j}| \right\};$$

$$|||A|||_2 = \sqrt{\lambda_{\max}(A^\top A)}, \quad \lambda_{\max}(B): \text{ largest eigenvalue of } B.$$

Let $|||\cdot|||$ be an induced matrix norm. The **condition number** of an invertible matrix A under that norm is

$$\kappa(A) = |||A||| \cdot |||A^{-1}|||.$$

Because $AA^{-1} = \mathbf{I}_n$, we have

$$1 = |||\mathbf{I}_n||| = |||AA^{-1}||| \leq |||A||| \cdot |||A^{-1}||| = \kappa(A).$$

When $\kappa(A) \gg 1$, we say that A is **ill-conditioned** under $|||\cdot|||$.

Estimating Error In this section, we estimate the **relative error** between the exact solution of $A\mathbf{x} = \mathbf{b}$ and an approximate solution $\hat{\mathbf{x}}$.[36]

36: This estimation is useful not only to quantify the effect of round-off error in **direct methods**, but also to analyze stopping criteria for **iterative methods**.

Theorem: let A be an invertible $n \times n$ matrix, and let $\mathbf{0} \neq \mathbf{b} \in \mathbb{R}^n$. Let $\mathbf{x} \in \mathbb{R}^n$ be the exact solution to the system $A\mathbf{x} = \mathbf{b}$. Consider a vector norm $\|\cdot\|$ on $\mathbb{R}^n$ and its induced matrix norm $|||\cdot|||$. For any $\hat{\mathbf{x}} \in \mathbb{R}^n$, we have

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \kappa(A) \cdot \frac{\|\mathbf{b} - A\hat{\mathbf{x}}\|}{\|\mathbf{b}\|}.$$

Proof: write

$$\mathbf{b} - A\hat{\mathbf{x}} = A\mathbf{x} - A\hat{\mathbf{x}} = A(\mathbf{x} - \hat{\mathbf{x}}) \implies \mathbf{x} - \hat{\mathbf{x}} = A^{-1}(\mathbf{b} - A\hat{\mathbf{x}});$$

hence

$$\|\mathbf{x} - \hat{\mathbf{x}}\| = \|A^{-1}(\mathbf{b} - A\hat{\mathbf{x}})\| \leq |||A^{-1}||| \cdot \|\mathbf{b} - A\hat{\mathbf{x}}\|.$$

We also have

$$\|\mathbf{b}\| = \|A\mathbf{x}\| \leq |||A||| \cdot \|\mathbf{x}\| \implies \frac{1}{\|\mathbf{x}\|} \leq \frac{1}{\|\mathbf{b}\|}|||A|||.$$

Combining both of these inequalities yields

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq |||A||| \cdot |||A^{-1}||| \cdot \frac{\|\mathbf{b} - A\hat{\mathbf{x}}\|}{\|\mathbf{b}\|};$$

as $\kappa(A) = |||A||| \cdot |||A^{-1}|||$, the proof is complete. ■.

In practice, due to floating point representation, we never really solve the system $A\mathbf{x} = \mathbf{b} \neq \mathbf{0}$;[37] instead, we solve the **perturbed system**

37: If $\mathbf{b} = \mathbf{0}$, the homogeneous system has the exact solution $\mathbf{x} = \mathbf{0}$ and no additional work is needed.

$$(A + \delta A)\hat{\mathbf{x}} = \mathbf{b} + \delta\mathbf{b},$$

where the entries of the $n \times n$ matrix δA and $\delta\mathbf{b} \in \mathbb{R}^n$ are of the order of 10^{-16} those of A and $\mathbf{b}$, respectively.[38]

38: Assuming double precision.

Let $\mathbf{x} \in \mathbb{R}^n$ be the exact solution of the unperturbed system and $\hat{\mathbf{x}}$ that of the perturbed system. Then

$$\mathbf{b} - A\hat{\mathbf{x}} = \mathbf{b} - (\mathbf{b} + \delta\mathbf{b} - \delta A\hat{\mathbf{x}}) = \delta A\hat{\mathbf{x}} - \delta\mathbf{b},$$

and we deduce from the previous theorem that

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \kappa(A)\frac{\|\mathbf{b} - A\hat{\mathbf{x}}\|}{\|\mathbf{b}\|} \leq \kappa(A) \cdot \frac{\|\delta A\|\|\hat{\mathbf{x}}\| + \|\delta\mathbf{b}\|}{\|\mathbf{b}\|}.$$

If $\|\delta A\| \leq \frac{1}{\|A^{-1}\|}$, then we can re-arrange the last equation and write

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \frac{\kappa(A)}{1 - \kappa(A) \cdot \frac{\|\delta A\|}{\|A\|}} \cdot \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}\right).$$

Example If the perturbation δA is $\mathbf{O}_{n\times n}$, then

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \kappa(A) \cdot \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}.$$

For instance, consider the exact and perturbed systems

$$\begin{pmatrix} 1 & 10^{-16} \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1 & 10^{-16} \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 + 10^{-16} \\ 1 \end{pmatrix}.$$

The exact solution of $A\mathbf{x} = \mathbf{b}$ is $\mathbf{x} = (1, 0)^\top$, that of $A\mathbf{x} = \mathbf{b} + \delta\mathbf{b}$ is $(1, 1)^\top$: a tiny perturbation $\delta\mathbf{b}$ has a gigantic effect on the solution. This is due to the fact that A is ill-conditioned. Indeed,

$$A^{-1} = \begin{pmatrix} 0 & 1 \\ 10^{16} & -10^{16} \end{pmatrix}, \quad |||A|||_1 = 2, \quad |||A^{-1}|||_1 = 1 + 10^{16},$$

and so $\kappa_1(A) = 2 + 2 \times 10^{16} \gg 1$.

Since the perturbation δA is $\mathbf{O}_{2\times 2}$, we would expect, in the ℓ_1 vector norm and associated induced matrix norm, to find:

$$\begin{aligned} \frac{\|\mathbf{x} - \hat{\mathbf{x}}\|_1}{\|\mathbf{x}\|_1} &= \frac{\|(0, -1)^\top\|}{\|(1, 0)^\top\|} = 1 \leq \kappa_1(A) \cdot \frac{\|\delta\mathbf{b}\|_1}{\|\mathbf{b}\|_1} \\ &= (2 + 2 \times 10^{16}) \cdot \frac{\|(10^{-16}, 0)^\top\|_1}{\|(1, 1)^\top\|_1} = (2 + 2 \times 10^{16}) \cdot \frac{10^{-16}}{2} = 1 + 10^{-16}, \end{aligned}$$

which is indeed the case. □

Iterative Methods

We can get exact solutions from direct methods, but the process is **computationally expensive** and storage can be prohibitive, especially for **large dense** matrices. In this section, we consider **iterative methods**, which operate in the same spirit as **fixed point iteration**.[39]

39: We will discuss other iterative methods, such as **gradient descent** and its variants in Chapter 31. Other modern approaches include the **generalized minimal residual** and **biconjugate gradient** method, among others.

The problem of solving $A\mathbf{x} = \mathbf{b}$ is equivalent to the problem of solving

$$f(\mathbf{x}) = A\mathbf{x} - \mathbf{b} = \mathbf{0}.$$

We re-write this problem into an equivalent problem

$$\mathbf{x} = g(\mathbf{x}) = T\mathbf{x} + \mathbf{c};$$

given an initial guess $\mathbf{x}_0$, we then compute the iterative sequence

$$\mathbf{x}^{(k+1)} = g(\mathbf{x}^{(k)}) = T\mathbf{x}^{(k)} + \mathbf{c}, \quad k = 0, 1, \ldots$$

The hope is that the sequence converges to the solution $\mathbf{x}^*$ of $A\mathbf{x} = \mathbf{b}$.

Stationary Iteration As was the case for functions of one variables, we can come up with multiple formulations for the **fixed point system**.

One general technique is based on a **splitting** of A: for an invertible matrix P (the **pre-conditioner**), we can write $A = P - (P - A)$:

$$\begin{aligned} A\mathbf{x} = \mathbf{b} &\iff [P - (P - A)]\mathbf{x} = \mathbf{b} \iff P\mathbf{x} = (P - A)\mathbf{x} + \mathbf{b} \\ &\iff \mathbf{x} = P^{-1}(P - A)\mathbf{x} + P^{-1}\mathbf{b} \iff \mathbf{x} = T\mathbf{x} + \mathbf{c}. \end{aligned}$$

The iterative method obtained with this splitting can be written as

$$P\mathbf{x}^{(k+1)} = (P - A)\mathbf{x}^{(k)} + \mathbf{b},$$

or equivalently, upon setting the **residual** $\mathbf{r}^{(k)} = \mathbf{b} - A\mathbf{x}^{(k)}$ **at step** k:

$$\begin{aligned} P\delta\mathbf{x}^{(k+1)} &= \mathbf{r}^{(k)}, \\ \mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \delta\mathbf{x}^{(k+1)}, \quad k = 0, 1, \ldots \end{aligned}$$

This approach is useful when $P\delta\mathbf{x}^{(k+1)} = \mathbf{r}^{(k)}$ is "**much simpler**" to solve than the original system $A\mathbf{x} = \mathbf{b}$, however. This is the case when P is diagonal (**Jacobi**) or triangular (**Gauss-Seidel**).[40]

40: In both cases, we assume that the diagonal entries of A are non-zero, i.e. $a_{i,i} \neq 0$ for $1 \leq i \leq n$.

Jacobi Method In this approach, we use

$$P = \begin{pmatrix} a_{1,1} & & \\ & \ddots & \\ & & a_{n,n} \end{pmatrix} \quad \text{and} \quad P - A = -\begin{pmatrix} 0 & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{n-1,n} \\ a_{n,1} & \cdots & a_{n,n-1} & 0 \end{pmatrix}.$$

The iterative procedure $P\mathbf{x}^{(k+1)} = (P - A)\mathbf{x}^{(k)} + \mathbf{b}$ then reduces to a linear system in which the components of $\mathbf{x}^{(k+1)}$ only depend on the components of $\mathbf{x}^{(k)}$:[41]

$$x_i^{(k+1)} = \frac{1}{a_{i,i}}\left(b_i + a_{i,i}x_i^{(k)} - \sum_{j=1}^{n} a_{i,j}x_j^{(k)}\right), \quad i = 1, \ldots, n.$$

41: They can be computed in **parallel**, which is a non-negligible time saver.

Gauss-Seidel Method In this approach, we use

$$P = \begin{pmatrix} a_{1,1} & & \\ \vdots & \ddots & \\ a_{n,1} & \cdots & a_{n,n} \end{pmatrix} \quad \text{and} \quad P - A = -\begin{pmatrix} 0 & a_{1,2} & \cdots & a_{1,n} \\ & \ddots & \ddots & \vdots \\ & & \ddots & a_{n-1,n} \\ & & & 0 \end{pmatrix}.$$

The iterative procedure $P\mathbf{x}^{(k+1)} = (P - A)\mathbf{x}^{(k)} + \mathbf{b}$ then reduces to a linear system which can be solved by forward substitution:

$$x_i^{(k+1)} = \frac{1}{a_{i,i}}\left(b_i - \sum_{j=1}^{i-1} a_{i,j}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{i,j}x_j^{(k)}\right), \quad i = 1, \ldots, n.$$

Example Consider the system $A\mathbf{x} = \mathbf{b}$ with

$$A = \begin{pmatrix} 3 & -1 & 1 \\ 3 & 6 & 2 \\ 3 & 3 & 7 \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} 1 \\ 0 \\ 4 \end{pmatrix}.$$

We use $\mathbf{x}_0 = (1, 1, 1)^\top$ to compute the first iterate for both the Jacobi and the Gauss-Seidel methods.

In the **Jacobi** method, the first iterate $\mathbf{x}^{(1)}$ solves

$$\begin{pmatrix} 3 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 7 \end{pmatrix} \begin{pmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{pmatrix} = \begin{pmatrix} 0 & 1 & -1 \\ -3 & 0 & -2 \\ -3 & -3 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 4 \end{pmatrix} = \begin{pmatrix} 1 \\ -5 \\ -2 \end{pmatrix},$$

which we can solve directly by substitution:

$$\mathbf{x}^{(1)} = \begin{pmatrix} 1/3 \\ -5/6 \\ -2/7 \end{pmatrix}.$$

In the **Gauss-Seidel** method, the first iterate $\mathbf{x}^{(1)}$ solves

$$\begin{pmatrix} 3 & 0 & 0 \\ 3 & 6 & 0 \\ 3 & 3 & 7 \end{pmatrix} \begin{pmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{pmatrix} = \begin{pmatrix} 0 & 1 & -1 \\ 0 & 0 & -2 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 4 \end{pmatrix} = \begin{pmatrix} 1 \\ -2 \\ 4 \end{pmatrix},$$

which we solve by forward substitution:

$$\begin{aligned} x_1^{(1)} &= 1/3 \\ 3x_1^{(1)} + 6x_2^{(1)} = -2 &\implies x_2^{(1)} = -1/2 \\ 3x_1^{(1)} + 3x_2^{(1)} + 7x_3^{(1)} = 4 &\implies x_3^{(1)} = 9/14. \end{aligned}$$

Convergence and Stopping Criterion We know how to compute iterates in the Jacobi and Gauss-Seidel framework, and, more generally, for an **iteration matrix**

$$T = P^{-1}(P - A).$$

How can we tell if the iteration procedure converges, and if it does, whether it converges to the system's unique solution $\mathbf{x}^*$?

The **error** $\mathbf{e}^{(k+1)}$ **at step** $k+1$ is defined by

$$\mathbf{e}^{(k+1)} = \mathbf{x}^* - \mathbf{x}^{(k+1)}.$$

Recall that $\mathbf{x}^* = T\mathbf{x}^* + \mathbf{c}$. Then

$$\mathbf{e}^{(k+1)} = \mathbf{x}^* - \mathbf{x}^{(k+1)} = T\mathbf{x}^* + \mathbf{c} - T\mathbf{x}^{(k)} - \mathbf{c} = T\left(\mathbf{x}^* - \mathbf{x}^{(k)}\right) = T\mathbf{e}^{(k)}.$$

Thus, for any vector norm $\|\cdot\|$ and induced matrix norm $|||\cdot|||$, we have

$$\|\mathbf{e}^{(k+1)}\| = \|T\mathbf{e}^{(k)}\| \le |||T||| \cdot \|\mathbf{e}^{(k)}\| \le |||T|||^2 \cdot \|\mathbf{e}^{(k-1)}\| \le \cdots \le |||T|||^{k+1} \cdot \|\mathbf{e}^{(0)}\|,$$

and so

$$\lim_{k\to\infty} \|\mathbf{e}^{(k+1)}\| = 0, \quad \text{when } |||T||| < 1.$$

Theorem: if $|||T||| < 1$ for an induced matrix norm, then for any $\mathbf{x}^{(0)}$, the sequence $\{\mathbf{x}^{(k)}\}$ converges to the solution of $\mathbf{x}^*$ of $A\mathbf{x} = \mathbf{b}$. Moreover,

$$\|\mathbf{x}^* - \mathbf{x}^{(k)}\| \le |||T|||^k \cdot \|\mathbf{x}^* - \mathbf{x}^{(0)}\|;$$

the smaller $|||T|||$ is, the faster the convergence to $\mathbf{x}^*$.

At what point in the iteration should we stop? Given a prescribed tolerance tol > 0, the goal is to stop as soon as

$$\|\mathbf{e}^{(k)}\| = \|\mathbf{x}^* - \mathbf{x}^{(k)}\| \leq \text{tol} \quad \text{or} \quad \frac{\|\mathbf{x}^* - \mathbf{x}^{(k)}\|}{\|\mathbf{x}^*\|} \leq \text{tol},$$

the latter assuming $\mathbf{b} \neq \mathbf{0}$. Since the error cannot be computed in practice, as it involves the exact solution $\mathbf{x}^*$, we need to use an **error estimate**.

One possibility is to use the **normalized residual** and stop as soon as

$$\frac{\|\mathbf{r}^{(k)}\|}{\|\mathbf{b}\|} = \frac{\|\mathbf{b} - A\mathbf{x}^{(k)}\|}{\|\mathbf{b}\|} \leq \text{tol}.$$

From a previous theorem, we have

$$\frac{\|\mathbf{x}^* - \mathbf{x}^{(k)}\|}{\|\mathbf{x}^*\|} \leq \kappa(A) \cdot \frac{\|\mathbf{r}^{(k)}\|}{\|\mathbf{b}\|} \leq \kappa(A) \cdot \text{tol};$$

when $\kappa(A)$ is reasonably small, the normalized residual is suitable to use in the stopping criterion.

Another possibility is to use the **increment between two iterates**, and stop as soon as

$$\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\| \leq \text{tol}.$$

In this case, since

$$\begin{aligned}\|\mathbf{x}^* - \mathbf{x}^{(k)}\| &\leq |||T||| \cdot \|\mathbf{x}^* - \mathbf{x}^{(k-1)}\| = |||T||| \cdot \|\mathbf{x}^* - \mathbf{x}^{(k)} + \mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\| \\ &\leq |||T||| \left[\|\mathbf{x}^* - \mathbf{x}^{(k)}\| + \|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|\right].\end{aligned}$$

Thus, provided $|||T||| < 1$, we have

$$\|\mathbf{x}^* - \mathbf{x}^{(k)}\| \leq \frac{|||T|||}{1 - |||T|||} \cdot \|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\| \leq \frac{|||T|||}{1 - |||T|||} \cdot \text{tol}.$$

The incremental stopping criterion would thus be a good choice if $|||T|||$ is not too close to 1.

Implementation The Jacobi (`J`) and Gauss-Seidel (`GS`) methods are implemented in `R` as follows.

Iterative solver

```
iterative_solver <- function(A, b, x0, nmax, tol, method){
  # Check for valid method
  if(!(method %in% c('J', 'GS'))) {
    stop("Unknown method...")
  }

  # Construct preconditioner matrix based on the method
  if(method == 'J') {
    P <- diag(diag(A))
  } else if(method == 'GS') {
    P <- matrix(0, ncol=ncol(A), nrow=nrow(A))
```

```
    for(i in 1:nrow(A)) {
      for(j in 1:ncol(A)) {
        if(i >= j) {
          P[i, j] <- A[i, j]
        }
      }
    }
  }

  # initialization
  b_norm <- norm(b, type="2")
  if(b_norm == 0) {
    b_norm <- 1
  }

  x <- x0
  r <- b - A %*% x
  r_norm <- norm(r, type="2")
  iter <- 0

  # Iteration
  while((r_norm/b_norm > tol) && (iter < nmax)){
    incr <- solve(P, r)
    x <- x + incr
    r <- b - A %*% x
    iter <- iter + 1
    r_norm <- norm(r, type="2")
  }

  return(list(x=x, iter=iter))
}
```

Example The `pracma` library is required to access `norm()`.

```
library(pracma)
```

For instance, we can solve the 4×4 system

$$\begin{pmatrix} 4 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 \\ 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 15 \\ 10 \\ 10 \\ 10 \end{pmatrix}$$

using the Gauss-Seidel method and the normalized residual stopping criterion, with a tolerance of 10^{-5} and $\mathbf{x}_0 = (0, 0, 0, 0)^\top$.

```
A <- matrix(c(4,-1,0,0,-1,4,-1,0, 0,-1,4,-1, 0,0,-1,3), 4, 4)
b <- c(15,10,10,10); x0 <- c(0,0,0,0)
nmax <- 100; tol <- 1e-5; method <- "GS"
result <- iterative_solver(A, b, x0, nmax, tol, method)
```

We can see the solution and number of iterations by calling the two

returned items.

```
result$x
```

```
        [,1]
[1,] 4.999974
[2,] 4.999982
[3,] 4.999991
[4,] 4.999997
```

```
result$iter
```

```
[1] 8
```

This compares very well to the exact solution $\mathbf{x}^* = (5, 5, 5, 5)^\top$.

4.3.2 Non-Linear Systems

The direct method does not generalize to non-linear systems of equations, but the fundamental concept of iterative methods does.[42]

42: We will have more to say on the topic in Chapter 31.

Fixed Point Iteration The ideas of Section 4.2.3 still apply, but they need to be modified somewhat to generalize to non-linear systems of n equations in n unknowns.

Let $f : D \subseteq \mathbb{R}^n \to \mathbb{R}^n$ be a sufficiently differentiable function. We are looking for points $\mathbf{x}^* \in \mathbb{R}^n$ that solve $f(\mathbf{x}) = \mathbf{0}$. In the general case, the system could admit any finite number of solution,[43] an infinite countable set of solutions,[44] or an uncountable set of solutions.[45] There is no simple criterion to determine in which class a given system falls.

43: Not necessarily only 0 or 1.

44: Such as for $\sin x = 0$ over $\mathbb{R}$.

45: Such as for $A\mathbf{x} = \mathbb{0}$ when A is not of full rank.

Example The system

$$f(\mathbf{x}) = \begin{pmatrix} x_1^3 + 2x_1x_2 \\ x_2 + 2x_1^2x_2 \end{pmatrix} = \begin{pmatrix} 8 \\ 13 \end{pmatrix}$$

is equivalent to

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} (8 - x_1^3)/2x_2 \\ 13 - 2x_1^2x_2 \end{pmatrix} = g(\mathbf{x}).$$

Note that there may be multiple ways to transform the system $f(\mathbf{x}) = \mathbf{0}$ into a fixed point problem $g(\mathbf{x}) = \mathbf{x}$, with $g(D) \subseteq D$.

General Fixed Point Theorem: let $g : D \to D$, with D a closed subset of $\mathbb{R}^n$, and $\|\cdot\|$ a vector norm on $\mathbb{R}^n$. If $\exists L < 1$ such that

$$\|g(\mathbf{x}) - g(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|$$

for all $\mathbf{x}, \mathbf{y} \in D$, then g admits a unique fixed point $\mathbf{x}^* \in D$ and the sequence $\mathbf{x}^{(k+1)} = g(\mathbf{x}^{(k)})$ converges to $\mathbf{x}^*$ for all $\mathbf{x}^{(0)} \in D$.

When g meets the condition stated in the theorem, we say it is **contractive on** D; it is not easy to show directly that this property holds. There is a sufficient condition on the **Jacobian matrix** of g at $\mathbf{x}^*$ (see Chapter 2 and the next section on Newton's method) that guarantees that g is contractive in a neighbourhood of $\mathbf{x}^*$:

$$|||Dg(\mathbf{x}^*)||| < 1,$$

assuming that g is at least C^1. In that case, the convergence of the fixed point iterates to $\mathbf{x}^*$ is at least of order 1 (linear).

Newton's Method In Section 4.2.5, we saw that there was a way to avoid directly evaluating the derivative f' in Newton's Method (which can be costly) by using the **secant approximation**.

This is a reasonable approach for equations in one variable, but it is less obvious how we would do so in a multi-dimensional case – this is where the work we put on linear systems will pay off.

In order to apply Newton's method to the system

$$f(\mathbf{x}) = \begin{pmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_n(\mathbf{x}) \end{pmatrix} = \mathbf{0},$$

where $f : D \subseteq \mathbb{R}^n \to \mathbb{R}^n$ is at least C^1, we need to generalize the iterates x_k, the function values $f(x_k)$, and the derivative $f'(x_k)$ to the multi-dimensional case.

The natural way to do this is as follows:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \implies \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - Df(\mathbf{x}^{(k)})^{-1} f(\mathbf{x}^k),$$

for $k \geq 0$, where

$$Df(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_n}{\partial x_n}(\mathbf{x}) \end{pmatrix}.$$

In practice it can be quite costly to invert the matrix not only once, but at every step of the iterative process. We can save time (and increase numerical stability) by re-writing the iteration step as a system of linear equations:

$$Df(\mathbf{x}^{(k)})(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = f(\mathbf{x}^k), \quad \text{for } k \geq 0,$$

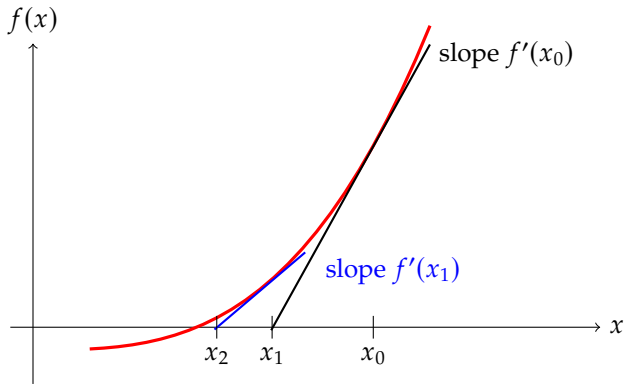
which can be solved using the methods of Section 4.3.1.

Under some regularity conditions on f, the sequence $\{\mathbf{x}^{(k)}\}$ converges quadratically to a solution $\mathbf{x}^*$ of $f(\mathbf{x}^*) = \mathbf{0}$.

Potential problems include the poor choice of the starting point $\mathbf{x}_0$ (at a critical point of f, $\mathbf{x}_0$ entering a cycle); f not being sufficiently differentiable in a neighbourhood of $\mathbf{x}^*$; $\mathbf{x}^*$ not existing; the derivative of f not being continuous at $\mathbf{x}^*$, etc.

4.4 Exercises

1. How must the Golden Ratio method be modified if we are looking for the maximal value of a unimodal continuous function f on $[a, b]$?
2. Is it necessary to use a factor φ in the Golden Ratio method or would any other constant > 1 do the trick?
3. Implement the secant method in R. Test it on this chapter's example functions.
4. Consider the function defined by $f(x) = x^2 - 2$, which has one positive root $x^* = \sqrt{2}$.



   a) Illustrate Newton's method by performing two steps starting at x_0.
   b) Let $\{x_k\}_{k\geq 0}$ be the sequence generated by Newton's method. Write the relationship between x_{k+1} and x_k. Then, compute x_1 and x_2 starting from $x_0 = 2$.
   c) Determine the (exact) order of Newton's method assuming that we start close enough to $x^* = \sqrt{2}$.

5. Let $f(x) = (x+2)(x+1)^2x(x-1)^3(x-2)$. To which zero of f does the bisection method converge when applied on the following intervals?

   a) $[-1.5, 2.5]$ b) $[-0.5, 2.4]$ c) $[-0.5, 3]$ d) $[-3, -0.5]$.

6. Use the bisection method on $[1, 2]$ to find an approximation of $\sqrt{3}$ correct to within 10^{-4}. Indicate which function f you used and report the values of x_0, x_1 and x_2, the final output and the number of iterations.
7. Let $f(x) = x^2 - 2x - 3$. To find a root of f, the following three fixed point method are proposed

   a) $x_k = \dfrac{3}{x_{k-1} - 2}$ b) $x_k = x_{k-1}^2 - x_{k-1} - 3$ c) $x_k = \dfrac{x_{k-1}^2 + 3}{2x_{k-1} - 2}$.

   For each method, compute (if possible) the iterates x_1, x_2, x_3 and x_4 starting from $x_0 = 0$. Report the values you obtain in a table. Which methods seem to be appropriate? Among those, which one seems to converge the fastest?
8. Consider the function $g(x) = \frac{1}{3}\sqrt[3]{x+8}$.
   a) Show that g has a unique fixed point in $[0, 1]$.

b) Assuming that we start from $x_0 = \frac{1}{2}$, find a bound for the number of fixed point iterations needed to achieve 10^{-6} accuracy.

9. Use Newton's method and the secant method with stopping criterion $|x_{k+1} - x_k| \leq 10^{-5}$ to find solutions for the following problems. For Newton's method, use the midpoint of the given interval for x_0 while for the secant method, use the endpoints of the given interval for x_0 and x_1.
   a) $3x - e^x = 0$ for $1 \leq x \leq 2$;
   b) $2x + 5\cos(x) - e^x = 0$ for $-5 \leq x \leq 0$.
10. Recall that a sequence $\{x_k\}$ that converges to some x^* is said to converge with order α and asymptotic error constant λ if
$$\lim_{k\to\infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^\alpha} = \lambda,$$
where we need $\lambda < 1$ if $\alpha = 1$.
   a) Consider the function $f(x) = 1/x - 1/3, x > 0$, which vanishes at $x^* = 3$. Use Newton's method with stopping criterion $|x_{k+1} - x_k| \leq 10^{-4}$ and $x_0 = 1$ to approximate x^*. Determine (numerically) the order α and the asymptotic error constant λ.
   b) Use the secant method to approximate the root of f defined in a) using $x_0 = 0.5$ and $x_1 = 1.5$. Verify that the order of the method is the golden ratio $\alpha = (1 + \sqrt{5})/2$ and determine the value of λ.
11. Suppose that x^* is a zero of multiplicity $m \geq 1$ of a function f of class C^m, namely
$$f(x^*) = f'(x^*) = f''(x^*) = \ldots = f^{(m-1)}(x^*) = 0 \quad \text{and} \quad f^{(m)}(x^*) \neq 0.$$
   a) Show that Newton's method
$$x_{k+1} = g_1(x_k), \quad k \geq 0, \quad \text{where} \quad g_1(x) = x - \frac{f(x)}{f'(x)},$$
converges only linearly (i.e., with order 1) if $m > 1$.
   b) Consider now the *modified Newton's method*
$$x_{k+1} = g_2(x_k), \quad k \geq 0, \quad \text{where} \quad g_2(x) = x - m\frac{f(x)}{f'(x)}.$$
Show that this method converges at least quadratically (i.e., with order ≥ 2) for any m.

   Hint: Write f as
$$f(x) = (x - x^*)^m h(x)$$
for some (unknown) function h with $h(x^*) \neq 0$ and show that $g_1'(x^*) = 1 - 1/m$ and $g_2'(x^*) = 0$.
12. Show that the induced matrix norm is indeed a norm (property 1 has already been proved; finish the job with properties 2 and 3).
13. Consider the $n \times n$ matrix A consisting of 1's on the diagonal and in the first column (every other entry being a 0). Compute $\kappa_1(A), \kappa_\infty(A)$, and $\kappa_2(A)$.

14. If A is a $n \times n$ symmetric positive definitive matrix, show that

$$\kappa_2(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}.$$

15. Solve the linear system

$$\begin{cases} x_1 - x_2 + 3x_3 = -2 \\ x_1 + x_2 = 5 \\ 3x_1 - 2x_2 + x_3 = 4. \end{cases}$$

using Gaussian elimination in its simplest form (i.e., without pivoting) and backward substitution.

16. Let

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 1 & -2 & 2 \\ 2 & 12 & -2 \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} 1 \\ 6 \\ -10 \end{pmatrix}.$$

a) Compute the LU factorization of A, i.e., find a lower triangular matrix L (with ones on the diagonal) and an upper triangular matrix U such that $A = LU$.
b) Solve the system $A\mathbf{x} = \mathbf{b}$ using only forward and backward substitution.

17. Let

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 5 & 3 \\ 4 & 6 & 8 & 0 \\ 3 & 3 & 9 & 8 \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} 1 \\ 0 \\ -2 \\ 2 \end{pmatrix}.$$

a) Find a lower triangular matrix L (with ones on the diagonal), an upper triangular matrix U and a permutation matrix P such that $PA = LU$.
b) Solve the system $A\mathbf{x} = \mathbf{b}$ using the factorization found in a).
c) Compute the determinant of A using the factorization found in a).

18. a) Prove that for any $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n$ we have

$$\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_2, \quad \|\mathbf{x}\|_2 \leq \sqrt{n}\|\mathbf{x}\|_\infty \quad \text{and} \quad \|\mathbf{x}\|_2^2 \leq \|\mathbf{x}\|_\infty \|\mathbf{x}\|_1. \tag{4.1}$$

b) For each inequality in (4.1), find a vector $\mathbf{x}$ for which equality is attained.
c) Prove that for any matrix $A \in \mathbb{R}^{n\times n}$

$$\frac{1}{\sqrt{n}}|||A|||_\infty \leq |||A|||_2 \leq \sqrt{n}|||A|||_\infty.$$

19. Let

$$\mathbf{x} = (1, -3, 2, -1)^T \quad \text{and} \quad A = \begin{pmatrix} -1 & -1 \\ 2 & -2 \end{pmatrix}.$$

a) Compute $\|\mathbf{x}\|_1$, $\|\mathbf{x}\|_2$ and $\|\mathbf{x}\|_\infty$.
b) Compute $|||A|||_1$, $|||A|||_2$ and $|||A|||_\infty$.
c) Compute $\kappa_1(A)$, $\kappa_2(A)$ and $\kappa_\infty(A)$.

20. We say that an $n \times n$ matrix A is strictly diagonally dominant by

row if

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}| \quad \text{for } i = 1, 2, \ldots, n. \tag{4.2}$$

Prove that if A satisfies (4.2) then the Jacobi method applied to $A\mathbf{x} = \mathbf{b}$, $\mathbf{b} \in \mathbb{R}^n$, converges. *Hint: show that* $\|T_J\|_\infty < 1$, *where* T_J *is the iteration matrix for the Jacobi method.*

21. Consider the system $A\mathbf{x} = \mathbf{b}$ with

$$A = \begin{pmatrix} 2 & -1 & 2 \\ -1 & 1 & 0 \\ 0 & 1 & 3 \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} -6 \\ 2 \\ -3 \end{pmatrix}.$$

22. Using $\mathbf{x}^{(0)} = (0, 0, 0)^T$ as initial guess:
    a) find (by hand) the first 2 iterations of the Jacobi method;
    b) find (by hand) the first iteration of the Gauss-Seidel method.
23. We consider the Gauss-Seidel method for solving the linear system $A\mathbf{x} = \mathbf{b}$, where

$$A = \begin{pmatrix} 1 & \alpha \\ -2 & 1 \end{pmatrix}.$$

24. Determine for which values of $\alpha \in \mathbb{R}$ the method converges for any initial guess $\mathbf{x}^{(0)} \in \mathbb{R}^2$ and any right-hand side $\mathbf{b} \in \mathbb{R}^2$.
25. Implement the fixed point algorithm for systems in `R` and solve the system in Section 4.3.2. Is the function g contractive on some neighbourhood D of the fixed point?
26. Implement Newton's algorithm for systems in `R` and solve the system in Section 4.3.2. What is its Jacobian?

Chapter References

[1] U.M. Ascher and C. Greif. *A First Course in Numerical Methods*. SIAM, 2011.

[2] P. Boily. *Analysis and Topology Study Aids*. Data Action Lab.

[3] B. Dionne. *Numerical Analysis*. uOttawa, 2023.

[4] B. Holland. 'Human Computers: The Women of NASA'. In: *History* (May 2003).

[5] W.K. Nicholson. *Linear Algebra with Applications, 3rd Edition*. PWS Publishing Company, 1994.

[6] J. Smékal et al. *Data Science in Physics*. Physics and Data Science.

[7] G. Strang. *Introduction to Linear Algebra*. Wellesley, 2016.