Introductory Numerical Analysis

Thomas Trogdon
University of Washington
`trogdon@uw.edu`

Contents

Preface

My preface

Chapter 0

Matlab basics

MATLAB is a scripting language without types. That means a few things.

1. There is no need to declare variables before you assign values to them. This, for example, is something that is necessary in JAVA.
2. A variable initially defined to be a scalar than then be assigned to be an array and then assigned to be a function. This can be convenient but also makes it tougher to catch bugs.
3. Unless a specific environment is loaded, MATLAB will do all of its calculations in floating point arithmetic. In a language like PYTHON an expression like `1/4` will return `0` because `0` is the closest integer and `1` and `4` are integers. MATLAB will return `0.25`.

MATLAB code is stored in `.m`-files, or m-files for short. These are also referred to as MATLAB scripts. It is good practice to include the following at the top of every script that your write:

```
clear all;  close all;
```

This will help ensure that

1. The ability of your code to run is not affected by other previously set variables.
2. If you close MATLAB, your script will behave the same when the next time you open it.

MATLAB will also print the output of any line if you do not append a semicolon `;` at the end of the line. For example, executing the following produces output to the Command Window

```
x = 10
```

```
x =

    10
```

while

```
x = 10;
```

produces no output. In order to debug your code you will want to have nearly every line end with a semicolon. And then if the script does not run as expected you can remove one or two semicolons at a time to monitor the output.

The following code uses a *for loop* to add up all the positive integers that are less than or equal to n

```
n = 10;
SUM = 0; % using capital letters because sum() is a built-in function
for i = 1:n
    SUM = SUM + i;
end
SUM
n*(n+1)/2  % known answer to check
```

```
SUM =

    55


ans =

    55
```

Another type of loop is the *while loop*. Here is an example of performing the same sum as above using a while loop.

```
n = 10;
SUM = 0;
i = 0;
while i < n
    i = i + 1;
    SUM = SUM + i;
end
SUM
```

```
SUM =

    55
```

0.1 ▪ Using Matlab with some examples from calculus

Because Part I of this text concerns numerical linear algebra, there are few opportunities introduce the reader to the plotting functionality in MATLAB. So, we take some time to review some theorems from calculus and illustrate them using MATLAB.

0.1.1 ▪ Demonstrations from differential calculus

Definition 0.1. *A function f defined on a set X has limit L at x_0*

$$\lim_{x \to x_0} f(x) = L$$

if, given any real number $\epsilon > 0$, there exists a real number $\delta > 0$ such that

$$|f(x) - L| < \epsilon, \quad \textit{whenever} \quad x \in X \quad \textit{and} \quad 0 < |x - x_0| < \delta.$$

Definition 0.2. *Let f be a function defined on a set X of real numbers and $x_0 \in X$. Then f is continuous at x_0 if*

$$\lim_{x \to x_0} f(x) = f(x_0).$$

The function

$$f(x) = \begin{cases} \cos(\pi/x), & x \neq 0, \\ 1 & x = 0, \end{cases}$$

is not continuous at $x = 0$. To show this, let $x_n = 1/n$. If f is continuous then $\lim_{n\to\infty} f(1/n) = 1$

```
f = @(x) cos(pi./x);
ns = linspace(1,10,100);
plot(ns,f(1./ns))
xlabel('n'); ylabel('f(1/n)') %label axes
```

The plot that results is shown in Figure 1

Definition 0.3. *Let $C^n[a,b] = \{f : [a,b] \to \mathbb{C} \mid f^{(n)}$ exists and is continuous$\}$.*

We use the notation $C[a,b] := C^0[a,b]$.

Theorem 0.4 (Intermediate Value Theorem). *Let $f \in C[a,b]$. Assume $f(a) \neq f(b)$. For every real number y, $f(a) \leq y \leq f(b)$, there exists $c \in [a,b]$ such that $f(c) = y$.*

```
x = linspace(-3,3,100);
f = @(x) sin(x);
c = @(x) 0*x+.1;
plot(x,f(x),'k')
hold on
plot(x,c(x)) %every value between f(-3) and f(3) is attained at least once
```

The plot that results is shown in Figure 2

Definition 0.5. *Let f be a function defined on an open interval containing x_0. The function f is differentiable at x_0 if*

$$f'(x_0) = \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0}$$

exists. In which case, $f'(x_0)$ is the derivative of $f(x)$ at x_0. If f has a derivative at each point in a set X then f is said to be differentiable on X.

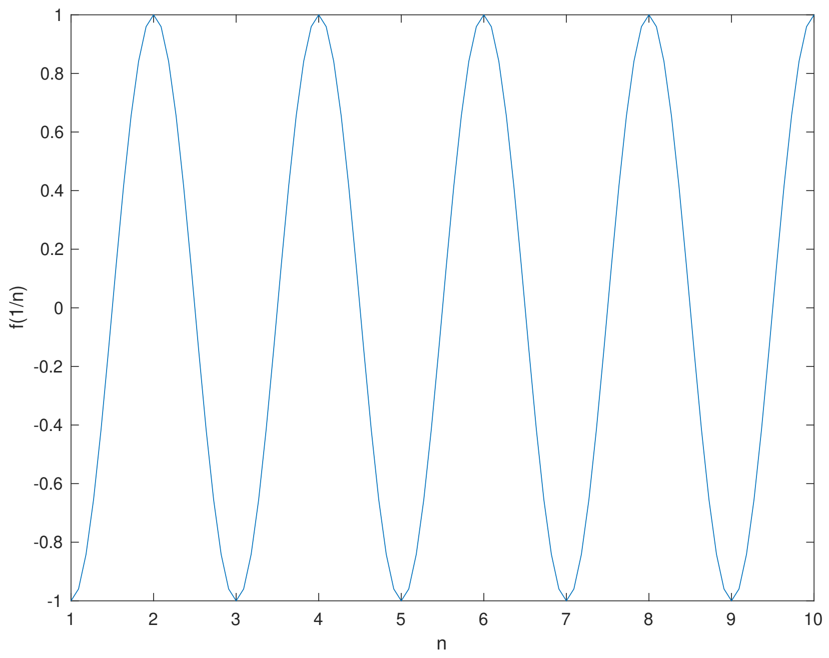

Figure 1. *A plot of the function $f(1/n)$*

```
format long %to see more digits
f = @(x) sin(x); df = @(x) cos(x);
x = .0001; x0 = 0;
(sin(x)-sin(x0))/(x-x0)-cos(x0)
```

```
ans =

    -1.666666582522680e-09
```

Here are some of the most important theorems from single-variable calculus:

Theorem 0.6. *If a function f is differentiable at x_0, it is continuous at x_0.*

Theorem 0.7 (Rolle's Theorem). *Suppose $f \in C[a,b]$ and f is differentiable on $[a,b]$. If $f(a) = f(b)$, the a number c in (a,b) exists with $f'(c) = 0$.*

Theorem 0.8 (Mean Value Theorem). *Suppose $f \in C[a,b]$ and f is differentiable on $[a,b]$. There exists a point $c \in (a,b)$ such that*

$$f'(c) = \frac{f(b) - f(a)}{b - a}.$$

Theorem 0.9 (Extreme Value Theorem). *If $f \in C[a,b]$, then $c_1, c_2 \in [a,b]$ exist with $f(c_1) \le f(x) \le f(c_2)$, for all $x \in [a,b]$. Furthermore, if f is differentiable on $[a,b]$ then c_1, c_2 are either the endpoints (a or b) or at a point where $f'(x) = 0$.*

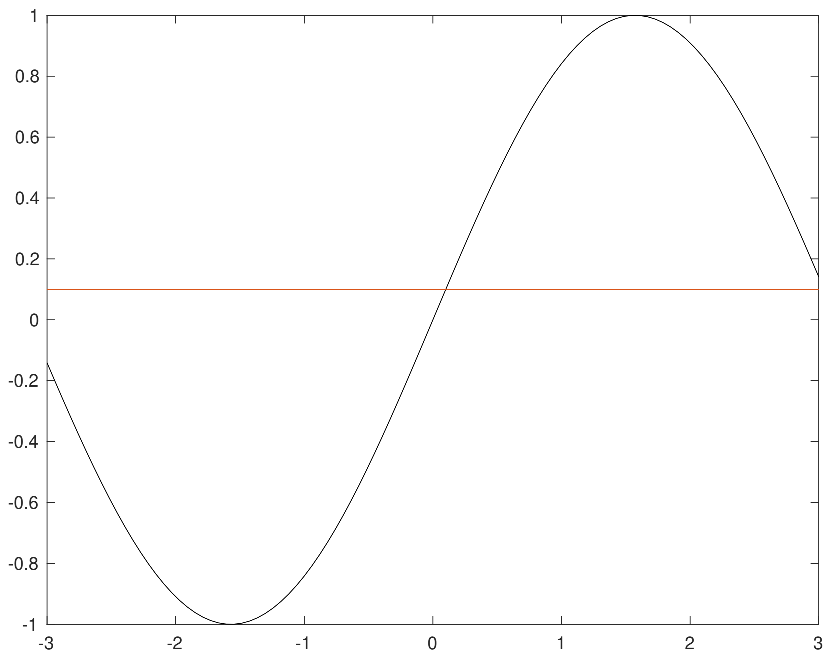

Figure 2. *A demonstration of the intermediate value theorem for $f(x) = \sin(x)$.*

This theorem states that both the maximum and minimum values of $f(x)$ on a closed interval $[a, b]$ must be attained within the interval (at points c_2 and c_1). A more involved theorem is the following:

Theorem 0.10. *Suppose $f \in C[a, b]$ is n-times differentiable on (a, b). If $f(x) = 0$ at $n + 1$ distinct numbers $a \leq x_0 < x_1 < \cdots < x_n \leq b$, then a number $c \in (x_0, x_n)$, (and hence in (a, b)) exists with $f^{(n)}(c) = 0$.*

Consider the fourth degree polynomial $f(x) = 8x^4 - 8x^2 + 1$:

```
f = @(x) 8*x.^4-8*x.^2+1; % has 4 zeros on (-1,1)
dddf = @(x) 8*4*3*2*x; % must have 1 zero on (-1,1)
x = linspace(-1,1,100);
plot(x,dddf(x))
```

The plot that results is shown in Figure 3

Theorem 0.11 (Taylor's Theorem). *Suppose $f \in C^n[a, b]$, and that $f^{(n+1)}$ exists on $[a, b]$, and $x_0 \in [a, b]$. For every $x \in [a, b]$, there exists a number $\xi(x)$ between x_0 and x with*

$$f(x) = P_n(x) + R_n(x),$$

where

$$P_n(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n,$$

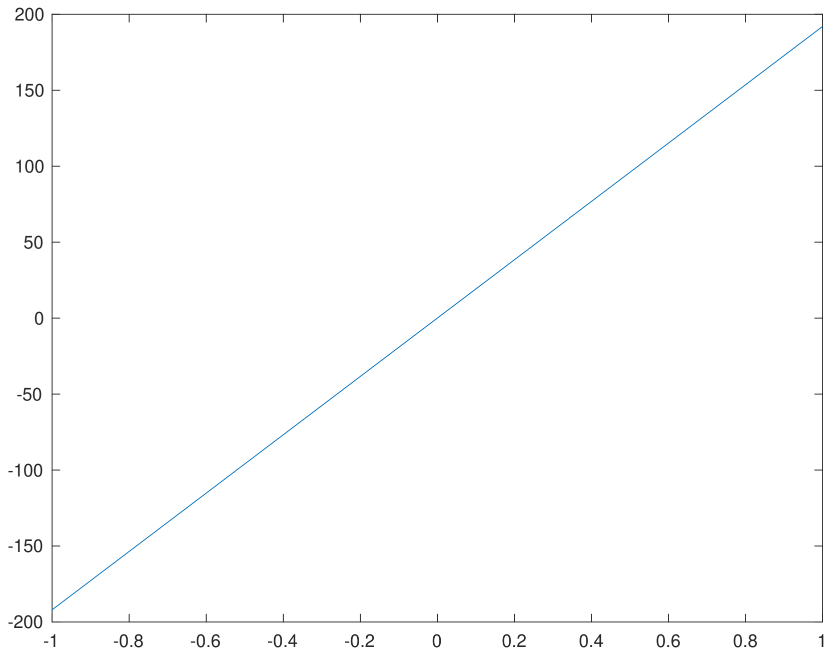

Figure 3. *A demonstration of Theorem 0.10 with* $f(x) = 8x^4 - 8x^2 + 1$.

and

$$R_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - x_0)^{n+1}.$$

0.1.2 ▪ Demonstrations from integral calculus

Definition 0.12. *The Riemann integral of the function* f *defined on the interval* $[a, b]$ *is the following limit (if it exists):*

$$\int_a^b f(x)dx = \lim_{\max \Delta x_i \to 0} \sum_{i=1}^n f(\bar{x}_i)\Delta x_i,$$

where the numbers $x_0, x_1, \dots, x_n$ *satisfy* $a = x_0 \le x_1 \le \dots \le x_n = b$, $\Delta x_i = x_i - x_{i-1}$ *for* $i = 1, 2, \dots, n$. *And* $\bar{x}_i$ *is an arbitrary point in the interval* $[x_{i-1}, x_i]$.

Let's choose the points x_i to be evenly spaced: $x_i = a + i\frac{b-a}{n}$ and $\bar{x}_i = x_i$. Then we have $\Delta x_i = \frac{b-a}{n}$ and

$$\int_a^b f(x)dx = \lim_{n\to\infty} \frac{b-a}{n} \sum_{i=1}^n f(x_i).$$

```
f = @(x) exp(x);
n = 10; a = -1; b = 1;
x = linspace(a,b,n+1); % create n + 1 points
```

```
x = x(2:end); % take the last n of these points
est = (b-a)/n*sum(f(x)) % evaluate f at these points and add them up
actual = exp(b)-exp(a) % the actual value
abs(est-actual)
```

```
est =

   2.593272082493666


actual =

   2.350402387287603


ans =

   0.242869695206063
```

Now choose and $\bar{x}_i = \dfrac{x_i + x_{i-1}}{2}$ to be the midpoint. We still have $\Delta x_i = \frac{b-a}{n}$ and

$$\int_a^b f(x)dx = \lim_{n\to\infty} \frac{b-a}{n} \sum_{i=1}^n f(\bar{x}_i).$$

```
f = @(x) exp(x);
n = 10; a = -1; b = 1;
x = linspace(a,b,n+1); % create n + 1 points
x = x(2:end); % take the last n of these points
x = x - (b-a)/(2*n); % shift to the midpoint
est = (b-a)/n*sum(f(x)) % evaluate f at these points and add them up
actual = exp(b)-exp(a) % the actual value
abs(est-actual)
```

```
est =

   2.346489615388305


actual =

   2.350402387287603


ans =

   0.003912771899298
```

Theorem 0.13 (Weighted Mean Value Theorem). *Suppose $f \in C[a, b]$, the Riemann integral of g exists on $[a, b]$, and $g(x)$ does not change sign on $[a, b]$. Then there* *exists a number c in* (a, b) *with*

$$\int_a^b f(x)g(x)dx = f(c)\int_a^b g(x)dx.$$

Part I

Numerical linear algebra

Chapter 1

Systems of equations

Before we start combining mathematics with programming in MATLAB we need to learn some of the mathematics!

1.1 ▪ A linear system of equations

Consider the following basic problem of fitting a line to data:

Problem 1.1. *Find the line that passes through the coordinates* $(0, 1)$ *and* $(-1, 2)$.

To solve this problem we start with our general form for a line

$$y = f(x) = ax + b,$$

and then enforce the conditions $f(0) = 1$ and $f(-1) = 2$. This gives

$$\begin{aligned} a \cdot 0 + b &= 1, \\ a \cdot (-1) + b &= 2. \end{aligned}$$

This is a *linear system* of two equations for the two unknowns a, b. Below we will make the term linear precise and we will understand when we can solve such a system. We will also extend our understanding to systems of n equations for n unknowns. And even go further and generalize this to systems m linear equations with n unknowns!

Back to the problem at hand, we see that the first equation tells us $b = 1$ and the second equation gives $-a + 1 = 2$, or $a = -1$. So the line

$$y = -x + 1,$$

passes through the coordinates $(0, 1)$ and $(-1, 2)$.

1.1.1 ▪ Vectors

A vector can be understood as an $n \times 1$ array of numbers:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}.$$

We will also use the shortened notation $\mathbf{v} = (v_j)_{j=1}^n$ to represent the same vector. We refer to the v_j's as the entries, or components, of $\mathbf{v}$. Unless otherwise stated, for a vector $\mathbf{v}$, v_j will refer to its entries. Sometimes we will encounter *row vectors.* A row vector is a $1 \times n$ array of numbers.

Some more notation before we proceed:

- $\mathbb{R}$ and $\mathbb{C}$ refer to all real and complex numbers[1], respectively.
- To state that a is a real number, we write $a \in \mathbb{R}$ (and similarly $a \in \mathbb{C}$) if a is complex).
- $\mathbf{v}$ is a vector with n entries, all being real numbers, we say $v \in \mathbb{R}^n$ (and similarly $\mathbf{v} \in \mathbb{C}^n$) if $\mathbf{v}$ has complex entries).
- We use the notation $\sum_{j=1}^n v_j = v_1 + v_2 + \cdots + v_n$.

Vectors represent an extension of our usual real (or complex) number system and so we should give some rules for addition, subtraction, and multiplication.

Definition 1.2. *We use the following rules for manipulating vectors* $\mathbf{u} = (u_j)_{j=1}^n, \mathbf{v} = (v_j)_{j=1}^n$ *in* $\mathbb{R}^n$*:*

1. $\mathbf{u} + \mathbf{v} = (u_j + v_j)_{j=1}^n$,
2. $\mathbf{u} - \mathbf{v} = (u_j - v_j)_{j=1}^n$,
3. $a\mathbf{v} = (av_j)_{j=1}^n$ *for* $a \in \mathbb{R}$*, and*
4. $\mathbf{u} = \mathbf{v}$ *if and only if* $\mathbf{u} - \mathbf{v} = \mathbf{0}$ *where* $\mathbf{0} \in \mathbb{R}^n$ *is the vector of all zeros.*

These rules state that everything just happens entry-by-entry. The last statement is equivalent to $u_j = v_j$ for every j. Note that enforcing that a vector with n entries is zero is actually enforcing n conditions! We also see the convenience of the notation introduced above.

The rules we are used to for scalar multiplication and addition carry over:

Theorem 1.3. *Suppose* $\mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ *and* $a, b \in \mathbb{R}$*. Then*

- $(\mathbf{u} + \mathbf{v}) + \mathbf{w} = \mathbf{u} + (\mathbf{v} + \mathbf{w})$ *(Associative, vector addition)*
- $\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$ *(Commutative, vector addition)*
- $a(\mathbf{u} + \mathbf{v}) = (a\mathbf{u}) + (a\mathbf{v})$ *(Distributive, vector addition)*
- $(a + b)\mathbf{u} = a\mathbf{u} + b\mathbf{u}$ *(Distributive, scalar addition)*
- $\mathbf{u} + \mathbf{0} = \mathbf{u}$ *(Additive identity)*
- $\mathbf{u} + (-\mathbf{u}) = \mathbf{0}$ *(Additive inverse)*

Example 1.4. Let $\mathbf{v} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \mathbf{w} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$. Then

[1] A complex number z is equal to $z = x + \mathrm{i}\,y$ where $x, y \in \mathbb{R}$ and $\mathrm{i} = \sqrt{-1}$. While these numbers may feel strange and artificial, they are extremely helpful in simplifying computations, especially on a computer.

- $\mathbf{v} + \mathbf{w} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$, and
- $2\mathbf{w} = \begin{bmatrix} 2 \\ -2 \end{bmatrix}$.

Then Problem 1.1, a system of two equations, can be summarized using one *vector equation.* To see how to do this first note that

$$\begin{bmatrix} a \cdot 0 \\ a \cdot (-1) \end{bmatrix} = a \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} b \\ b \end{bmatrix} = b \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

So then we need to enforce that

$$a \begin{bmatrix} 0 \\ -1 \end{bmatrix} + b \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

This conversion of the problem does not actually help us solve it! We need to introduce matrices before this conversion really becomes useful.

1.1.2 ▪ Matrices

A matrix can be understood as an $m \times n$ array of numbers

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & \vdots \\ \vdots & & \ddots & \\ a_{m1} & \cdots & & a_{mn} \end{bmatrix}.$$

And just like the case of vectors we use the notation $A = (a_{ij})_{\substack{1 \le i \le m \\ 1 \le j \le n}}$ to quickly refer to the matrix. In this index notation we always take the top index (i in this case) to refer to the rows of the matrix and the bottom index (j) to refer to columns. Unless otherwise stated, for a matrix A, a_{ij} will refer to its entries.

Example 1.5. Write down the matrix A given by

$$A = \left(\frac{1}{i+j} \right)_{\substack{1 \le i \le 3 \\ 1 \le j \le 4}}.$$

The matrix A is then given by

$$A = \begin{bmatrix} \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{bmatrix}.$$

Similar to vectors, if A is an $m \times n$ matrix of real numbers, we write $A \in \mathbb{R}^{m \times n}$ (similarly $A \in \mathbb{C}^{m \times n}$ if A has complex entries). And we have an analogous set of rules.

Definition 1.6. *We use the following rules for manipulating matrices* $A = (a_{ij})_{\substack{1 \le i \le m \\ 1 \le j \le n}}, B = (b_{ij})_{\substack{1 \le i \le n \\ 1 \le j \le m}}$ *in* $\mathbb{R}^{m \times n}$*:*

1. *$A+B=(a_{ij}+b_{ij})_{\substack{1\le i\le m,\\ 1\le j\le n}}$*
2. *$A-B=(a_{ij}-b_{ij})_{\substack{1\le i\le m,\\ 1\le j\le n}}$*
3. *$cA=(ca_{ij})_{\substack{1\le i\le m\\ 1\le j\le n}}$ for $c\in\mathbb{R}$, and*
4. *$A=B$ if and only if $A-B=\mathbf{0}$ where $\mathbf{0}\in\mathbb{R}^{m\times n}$ is the matrix of all zeros.*

As with vectors, addition, multiplication and subtraction are simply taken entry-by-entry. Note that matrix sizes must be the same for them to added or subtracted.

Example 1.7. Let $A=\begin{bmatrix}1&2&0\\-1&0&1\end{bmatrix}$ and $B=\begin{bmatrix}1&0&0\\-1&0&-1\end{bmatrix}$. Then

$$\begin{aligned}A+B&=\begin{bmatrix}2&2&0\\-2&0&0\end{bmatrix},\\ 3A&=\begin{bmatrix}3&6&0\\-3&0&3\end{bmatrix}.\end{aligned}$$

Matrix-vector multiplication

We now return to this equation

$$a\begin{bmatrix}0\\-1\end{bmatrix}+b\begin{bmatrix}1\\1\end{bmatrix}=\begin{bmatrix}0&1\\-1&1\end{bmatrix}\begin{bmatrix}a\\b\end{bmatrix}=\begin{bmatrix}1\\2\end{bmatrix}.$$

The first equality can be taken as a definition of the matrix-vector product and it motivates the general definition below. But before we discuss this in greater detail we introduce some notation for columns of a matrix. Suppose $A\in\mathbb{R}^{m\times n}$. The vectors $\mathbf{a}_1,\mathbf{a}_2,\cdots,\mathbf{a}_n\in\mathbb{R}^m$ are the columns of A, i.e.,

$$A=\begin{bmatrix}\mathbf{a}_1&\mathbf{a}_2&\cdots&\mathbf{a}_n\end{bmatrix}.$$

Definition 1.8. *Suppose $A\in\mathbb{R}^{m\times n}$ and $\mathbf{v}\in\mathbb{R}^n$ then we define*

$$A\mathbf{v}=\sum_{j=1}^{n}v_j\mathbf{a}_j.$$

In other words, the matrix-vector product $A\mathbf{v}$ gives a weighted sum of the columns of A.

Example 1.9.

$$\begin{bmatrix}0&1\\-1&1\end{bmatrix}\begin{bmatrix}-1\\1\end{bmatrix}=(-1)\begin{bmatrix}0\\-1\end{bmatrix}+(1)\begin{bmatrix}1\\1\end{bmatrix}=\begin{bmatrix}1\\2\end{bmatrix}.$$

But this is not typically how one computes this by hand. By hand, most students find it easier to use the rule

Theorem 1.10. *For $A \in \mathbb{R}^{m\times n}$ and $v \in \mathbb{R}^n$*

$$A\mathbf{v} = \left(\sum_{j=1}^{n} a_{ij}v_j\right)_{i=1}^{m}.$$

Example 1.11. To compute

$$\mathbf{w} = \underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 2 & 3 \end{bmatrix}}_{2\times 3} \underbrace{\begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}}_{3\times 1}$$

we first note that the “inner dimensions” match and therefore we have a well-defined product. The dimension of the output is calculated by

$$(2\times 3)\times(3\times 1) \longrightarrow (2\times \cancel{3})\times(\cancel{3}\times 1) \longrightarrow 2\times 1.$$

So, $\mathbf{w} \in \mathbb{R}^2$. By Theorem 1.10 we see that

$$w_1 = 1\cdot 1 + 0\cdot 0 + (-1)\cdot(-1) = 2.$$

This is computed by multiplying the first row of the matrix times the vector and adding. Similarly, the second entry is given by

$$w_2 = 1\cdot 1 + 2\cdot 0 + 3\cdot(-1) = -2.$$

We conclude that $\mathbf{w} = \begin{bmatrix} 2 \\ -2 \end{bmatrix}$. Let's now go back and verify that this gives the same as our definition:

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 2 & 3 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} = 1\begin{bmatrix} 1 \\ 1 \end{bmatrix} + 0\begin{bmatrix} 0 \\ 2 \end{bmatrix} + (-1)\begin{bmatrix} -1 \\ 3 \end{bmatrix} = \begin{bmatrix} 2 \\ -2 \end{bmatrix}.$$

Computing a matrix-vector product via Theorem 1.10 is simpler by hand because you can work entry-by-entry and you do not have to think about all of the columns at once. But for a computer, thinking about all the columns is actually a simpler way of doing things! This highlights something interesting that we'll encounter again, many times, in this book:

- Something that is hard for human computation may be “easy” for a computer.

1.1.3 ▪ Matrix-matrix product

While it is not immediately clear why we will ever need to develop a way to multiply two matrices, we define it here. It turns out this is absolutely critical!

Definition 1.12. *Suppose $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{\ell\times m}$. Then*

$$BA = \begin{bmatrix} B\mathbf{a}_1 & B\mathbf{a}_2 & \cdots & B\mathbf{a}_n \end{bmatrix}.$$

Note that

$$\underbrace{B}_{\ell\times m}\underbrace{A}_{m\times n},$$

so the output should be

$$(\ell\times m)\times(m\times n)\longrightarrow(\ell\times \cancel{m})\times(\cancel{m}\times n)\longrightarrow \ell\times n.$$

This is confirmed by noting that $B\mathbf{a}_j\in\mathbb{R}^\ell$ and we have n of these vectors. To actually compute the matrix-matrix product by hand, it is often easiest to fill out each column of the resulting matrix at a time, by doing a matrix-vector product.

Example 1.13.

$$\begin{bmatrix}1 & 0 & -1\\ 1 & 2 & 3\end{bmatrix}\begin{bmatrix}1 & 1\\ 0 & 1\\ -1 & 1\end{bmatrix}=\begin{bmatrix}2 & 0\\ -2 & 6\end{bmatrix}$$

The summary of the operations is given below.

Theorem 1.14. *Suppose A, B, C are matrices of the appropriate dimensions*[2] *so that the following combinations are defined. Suppose, $a, b\in\mathbb{R}$. Then*

- $A+B=B+A$ *(Commutative, matrix addition)*
- $(A+B)+C=A+(B+C)$ *(Associative, matrix addition)*
- $A+\mathbf{0}=A$ *(Additive identity)*
- $A+(-A)=\mathbf{0}$ *(Additive inverse)*
- $c(dA)=(cd)A$ *(Associative, scalar multiplication)*
- $c(A+B)=(cA)+(cB)$ *(Distributive, scalar multiplication) item* $(c+d)A=(cA)+(dA)$ *(Distributive, scalar multiplication)*
- $(AB)C=A(BC)$ *(Associative, matrix multiplication)*
- $A(B+C)=(AB)+(AC)$ *(Distributive, matrix multiplication)*
- $(B+C)A=(BA)+(CA)$ *(Distributive, matrix multiplication)*
- $c(AB)=A(cB)=(cA)B$ *(Compatibility)*
- $IA=A=AI$ *(Multiplicative identity)*
- $A\mathbf{0}=\mathbf{0}=\mathbf{0}A$ *(Zero matrix)*

1.2 ▪ Creating and using vectors and matrices in Matlab

This section is the first example of what will become common in this text. Right after some new mathematical concepts are introduce, we realize them using MATLAB code.

[2]The dimensions may vary from line to line.

1.2.1 ▪ Vectors

Here is the creation of a row vector `v` and a (column) vector `w`.

```
v = [1,2,3]; % row vector
w = [1;2;3]; % column vector
v
w
```

```
v =

     1     2     3


w =

     1
     2
     3
```

Now, let us create these vectors and then grab specific entries or groups of entries. The last command here add up all the entries in a vector. The output that follows gives the result from lines 3-6.

```
1  v = [1,2,3]; % row vector
2  w = [1;2;3]; % column vector
3  w(1)
4  v(1:2)
5  w(2:3)
6  sum(w)
```

```
ans =

     1


ans =

     1     2


ans =

     2
     3


ans =

     6
```

1.2.2 ▪ Matrices

Now, we will create a matrix by manually specifying every entry. MATLAB uses the semicolon ; as a line break.

```
M = [1,2,3,4;5,6,7,8;9,10,11,12;12,14,15,16]
M(1,4) % get the (1,4) element
M(2:3,2:3)% get the "middle" block of the matrix
M(3,:) % Get row 3
sum(M)
```

```
M =

     1     2     3     4
     5     6     7     8
     9    10    11    12
    12    14    15    16


ans =

     4


ans =

     6     7
    10    11


ans =

     9    10    11    12


ans =

    27    32    36    40
```

For the last example in this section we confirm the multiplication in Example 1.13.

```
A = [1,0,-1; 1,2,3];
B = [1,1; 0,1; -1,1];
A*B
```

```
ans =

     2     0
    -2     6
```

1.3 ▪ Regular Gaussian elimination

Before we begin our discussion of Gaussian elimination we define some important concepts

Definition 1.15.

- *The diagonal of a matrix* $A \in \mathbb{R}^{m\times n}$ *is the entries* a_{ii} *for* $i = 1, 2, \dots, \min\{m, n\}$.
- *A matrix* L *is said to be* lower triangular *if* $\ell_{ij} = 0$ *for all* $i < j$.
- *A matrix* U *is said to be* upper triangular *if* $u_{ij} = 0$ *for all* $i > j$.
- *An upper/lower-triangular matrix is said to be* special *if all the diagonal entries are ones.*

We use an example to illustrate (regular) Gaussian elimination. Consider the linear system of equations

$$\begin{aligned} x + 4y + z &= 2, \\ 2x + 12y + z &= 7, \\ x + 2y + 4z &= 3. \end{aligned} \tag{1.1}$$

We know that this system is equivalent to the vector equation

$$\begin{bmatrix} 1 & 4 & 1 \\ 2 & 12 & 1 \\ 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 2 \\ 7 \\ 3 \end{bmatrix}.$$

This last system is represented in shorthand by the *augmented matrix*

$$\left[\begin{array}{ccc|c} 1 & 4 & 1 & 2 \\ 2 & 12 & 1 & 7 \\ 1 & 2 & 4 & 2 \end{array}\right].$$

$$\begin{aligned} x + 4y + z &= 2 \\ 2x + 12y + z &= 7 \\ x + 2y + 4z &= 3 \end{aligned} \qquad \Leftrightarrow \qquad \left[\begin{array}{ccc|c} 1 & 4 & 1 & 2 \\ 2 & 12 & 1 & 7 \\ 1 & 2 & 4 & 2 \end{array}\right]$$

Now, on the left, we multiply the first equation by -2 and add it to the second equation

$$\begin{array}{rc} & (-2)\cdot(x + 4y + z = 2) \\ + & 2x + 12y + z = 7 \\ \hline & 0x + 4y - z = 3 \end{array}$$

We write the system out and convert it to our augmented matrix

$$\begin{aligned} x + 4y + z &= 2 \\ 0x + 4y - z &= 3 \\ x + 2y + 4z &= 3 \end{aligned} \qquad \Leftrightarrow \qquad \left[\begin{array}{ccc|c} 1 & 4 & 1 & 2 \\ 0 & 4 & -1 & 3 \\ 1 & 2 & 4 & 2 \end{array}\right]$$

Then, the important observation is that the new augmented matrix can be found from the old by taking $(-2)\cdot$ (row one) + (row two) and replacing row two with this: $-2R_1 + R_2 \to R_2$. We call this an *elementary row operation.* Now, we eliminate x from the last equation:

$$\begin{array}{rr} & (-1)\cdot(x+4y+z=2) \\ + & x+2y+4z=3 \\ \hline & 0x-2y+3z=1 \end{array}$$

$$\begin{aligned} x+4y+z&=2 \\ 0x+4y-z&=3 \\ 0x-2y+3z&=1 \end{aligned} \quad\Leftrightarrow\quad \left[\begin{array}{ccc|c} 1 & 4 & 1 & 2 \\ 0 & 4 & -1 & 3 \\ 0 & -2 & 3 & 1 \end{array}\right]$$

The last step of (regular) Gaussian elimination for this system is to eliminate y from the last equation

$$\begin{array}{rr} & (1/2)\cdot(4y-z=3) \\ + & -2y+3z=1 \\ \hline & 0y+5/2z=5/2 \end{array}$$

$$\begin{aligned} x+4y+z&=2 \\ 0x+4y-z&=3 \\ 0x+0y+\frac{5}{2}z&=\frac{5}{2} \end{aligned} \quad\Leftrightarrow\quad \left[\begin{array}{ccc|c} 1 & 4 & 1 & 2 \\ 0 & 4 & -1 & 3 \\ 0 & 0 & \frac{5}{2} & \frac{5}{2} \end{array}\right]$$

This is the last step of Gaussian elimination.

Definition 1.16. *An elementary row operation of type 1 is of the form $aR_j + R_i \to R_i$ for $i \neq j$.*

Definition 1.17. *Regular Gaussian elimination is the process by which a matrix (square or rectangular) is reduced to upper triangular form using only row operations of type I where $i > j$.*

This definition states that in regular Gaussian elimination we can use only rows above to introduce zeros.

Algorithm 1: Regular Gaussian elimination `rGE`

```
Result: An upper triangular matrix, or failure
Input: $A \in \mathbb{R}^{n\times n+j}$, $j \geq 0$
begin
    for $j = 1$ to $n-1$ do
        if $a_{jj} = 0$ then
            $A$ is not regular;
            return failure;
        end
        for $i = j+1$ to $n$ do
            set $\ell_{ij} = a_{ij}/a_{jj}$;
            perform $-\ell_{ij}R_j + R_i \to R_i$ on $A$;
        end
    end
end
```

We now need to understand why introducing zeros in the matrix, i.e. elimination, is beneficial. For the system

$$\begin{aligned} x + 4y + z &= 2 \\ 0x + 4y - z &= 3 \\ 0x + 0y + \frac{5}{2}z &= \frac{5}{2} \end{aligned}$$

we know that the solutions of this are the same as the solutions of the original system (1.1). We first determine z:

$$\frac{5}{2}z = \frac{5}{2} \Rightarrow z = 1,$$

then determine y

$$4y - z = 3 \Rightarrow y = 1,$$

and, lastly, determine x

$$x + 4y + z = 2 \Rightarrow x = -3.$$

This process of working backward up the matrix is referred to as *back substitution.* To properly describe this process algorithmically, consider breaking up the final augmented matrix

$$\left[\; U \mid \mathbf{c} \;\right] = \left[\begin{array}{ccc|c} 1 & 4 & 1 & 2 \\ 0 & 4 & -1 & 3 \\ 0 & 0 & \frac{5}{2} & \frac{5}{2} \end{array}\right].$$

Then the following algorithm is back substitution.

Algorithm 2: Back substitution `Backsub`

```
Result: The solution of $U\mathbf{x} = \mathbf{c}$
Input: An augmented matrix $\left[\begin{array}{c|c} U & \mathbf{c} \end{array}\right]$, $U \in \mathbb{R}^{n\times n}$ upper triangular and
       $u_{jj} \neq 0$ for all $j$
begin
    set $x_n = c_n/u_{nn}$;
    for $i = n-1$ to $1$ with increment $-1$ do
        set $x_i = \frac{1}{u_{ii}}\left(c_i - \sum_{j=i+1}^{n} u_{ij}x_j\right)$;
    end
end
```

Remark 1.18. *We return to the example linear system from Problem 1.1, now written in augmented form,*

$$\left[\begin{array}{cc|c} 0 & 1 & 1 \\ -1 & 1 & 2 \end{array}\right].$$

It is clear that regular Gaussian elimination will immediately fail on this matrix. We will introduce more row elementary row operations soon to deal with this system.

So, we now give a name to matrices for which regular Gaussian elimination succeeds.

Definition 1.19. *A matrix A is called regular if regular Gaussian elimination succeeds in transforming it to an upper-triangular matrix with non-zero diagonal entries.*

1.4 ▪ The LU factorization

Now consider the matrix product (by divine inspiration!)

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & -\frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 1 & 4 & 1 \\ 0 & 4 & -1 \\ 0 & 0 & \frac{5}{2} \end{bmatrix} = \begin{bmatrix} 1 & 4 & 1 \\ 2 & 12 & 1 \\ 1 & 2 & 4 \end{bmatrix} = A.$$

This is called an *LU* factorization of the matrix A. Note that the first matrix in the product is lower triangular with ones on the diagonal. How is this useful? To really understand this we need to discuss matrix inverses.

Definition 1.20 (Matrix inverse). *The inverse of an $n\times n$ square matrix A is another matrix B such that*

$$BA = I = I_n = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix}.$$

We refer to $I = I_n$ as the $n \times n$ identity matrix and we denote $B = A^{-1}$.

Example 1.21. For a 1×1 matrix $A = \left[a\right]$, where $a \in \mathbb{R}$, it is clear that $A^{-1} = \left[1/a\right]$.

But note that for

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix},$$

it is no longer clear what A^{-1} is!

Multiplication by the identity matrix does not change the vector.

Proposition 1.22. *For $\mathbf{v} \in \mathbb{R}^n$, $I_n\mathbf{v} = \mathbf{v}$.*

So, suppose $A = LU$ and we want to solve

$$A\mathbf{x} = \mathbf{b} \tag{1.2}$$

for a vector $\mathbf{b}$. Substitute,

$$LU\mathbf{x} = \mathbf{b}.$$

Multiply by L^{-1}

$$L^{-1}LU\mathbf{x} = U\mathbf{x} = L^{-1}\mathbf{b}.$$

So, we have the system $U\mathbf{x} = L^{-1}\mathbf{b}$ and we can now multiply by U^{-1}

$$U^{-1}U\mathbf{x} = \mathbf{x} = U^{-1}L^{-1}\mathbf{b}.$$

This entirely oversimplifies this whole procedure. How do we compute L^{-1}, U^{-1}? And then we have to multiply by them? This is something that is accomplished all at once.

Theorem 1.23. *Suppose $U \in \mathbb{R}^{n\times n}$ is upper triangular with non-zero diagonal entries. Then computing $U^{-1}\mathbf{c}$ gives the exact same result as solving $\left[\begin{array}{c|c} U & \mathbf{c} \end{array}\right]$ with back substitution.*

This implies that (1) we never need to actually find out what U^{-1} is and (2) we already have an algorithm for handling this computation.

For this procedure to be successful we need to have an analogous procedure to handle L. Since L is lower-triangular, we can solve with forward substitution.

Algorithm 3: Forward substitution `Forsub`.

```
Result: The solution of $L\mathbf{c} = \mathbf{b}$
Input: An augmented matrix $\left[\begin{array}{c|c} L & \mathbf{b} \end{array}\right]$, $L \in \mathbb{R}^{n\times n}$ lower triangular and $\ell_{jj} = 1$
       for all $j$
begin
    set $c_1 = b_1$;
    for $i = 2$ to $n$ do
        set $c_i = b_i - \sum_{j=1}^{i-1} \ell_{ij} c_j$;
    end
end
```

Theorem 1.24. *Suppose $L \in \mathbb{R}^{n\times n}$ is lower triangular with ones on the diagonal. Then computing $L^{-1}\mathbf{b}$ gives the exact same result as solving $\left[\begin{array}{c|c} L & \mathbf{b} \end{array}\right]$ with forward substitution.*

With this in hand we can now describe the use of the LU factorization — if we have it.

Algorithm 4: Solve a system using an LU factorization

```
Result: The solution of $LU\mathbf{x} = \mathbf{b}$
Input: $U \in \mathbb{R}^{n\times n}$ upper triangular and $u_{jj} \neq 0$ for all $j$ and a special
       lower-triangular matrix $L \in \mathbb{R}^{n\times n}$
begin
    set $\mathbf{c} = \texttt{Forsub}(\left[\begin{array}{c|c} L & \mathbf{b} \end{array}\right])$;
    set $\mathbf{x} = \texttt{Backsub}(\left[\begin{array}{c|c} U & \mathbf{c} \end{array}\right])$;
end
```

Before describing how to compute the LU factorization, we pause to note that when we applied regular Gaussian elimination to the augmented system $\left[\begin{array}{c|c} A & \mathbf{b} \end{array}\right]$ and reduced it to $\left[\begin{array}{c|c} U & \mathbf{c} \end{array}\right]$, the forward substitution step is not needed (indeed, it is actually encoded within regular Gaussian elimination!).

Algorithm 5: Solve a system using regular Gaussian elimination

```
Result: The solution of $A\mathbf{x} = \mathbf{b}$
Input: $A \in \mathbb{R}^{n\times n}$, regular.
begin
    set $\left[\begin{array}{c|c} U & \mathbf{c} \end{array}\right] = \texttt{rGE}(\left[\begin{array}{c|c} A & \mathbf{b} \end{array}\right])$;
    set $\mathbf{x} = \texttt{Backsub}(\left[\begin{array}{c|c} U & \mathbf{c} \end{array}\right])$;
end
```

1.5 ▪ Computing the LU factorization

If we look back at `rGE` as defined in Algorithm 1 we see that quantities ℓ_{ij} are defined in the process. These are exactly the (strictly) lower-triangular entries of L. We will work with an example to see that this is indeed the case. Note that in Algorithm 1, ℓ_{ij} is defined but $-\ell_{ij}$ is used in the row operation. So, we collect ℓ_{ij}'s that are used in (1.1):

$$
\begin{aligned}
(i,j) &= (2,1) \Rightarrow \ell_{21} = 2,\\
(i,j) &= (3,1) \Rightarrow \ell_{21} = 1,\\
(i,j) &= (3,2) \Rightarrow \ell_{32} = -1/2.
\end{aligned}
$$

One can then check that

$$
\begin{bmatrix} 1 & 4 & 1 \\ 2 & 12 & 1 \\ 1 & 2 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & -1/2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 4 & 1 \\ 0 & 4 & -1 \\ 0 & 0 & \frac{5}{2} \end{bmatrix}.
$$

But this is a long way from actually establishing that this is how we choose the entries. To see this is the case we introduce *elementary matrices.*

Definition 1.25. *An elementary matrix is formed by applying an elementary row operation to the identity matrix.*

Example 1.26. For the row operation $aR_1 + R_2 \to R_2$ on a 3×3 matrix we have

$$\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} \xrightarrow{aR_1+R_2\to R_2} \begin{bmatrix}1 & 0 & 0\\ a & 1 & 0\\ 0 & 0 & 1\end{bmatrix} = E.$$

Here E is the elementary matrix associated to the row operation $aR_1 + R_2 \to R_2$ on a 3×3.

Next consider our example matrix, for which we do the initial row operation $(-2)R_1 + R_2 \to R_2$

$$\begin{bmatrix}1 & 4 & 1\\ 2 & 12 & 1\\ 1 & 2 & 4\end{bmatrix} \xrightarrow{(-2)R_1+R_2\to R_2} \begin{bmatrix}1 & 4 & 1\\ 0 & 4 & -1\\ 1 & 2 & 4\end{bmatrix}.$$

Then consider multiplication by the associated elementary matrix

$$\begin{bmatrix}1 & 0 & 0\\ -2 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 4 & 1\\ 2 & 12 & 1\\ 1 & 2 & 4\end{bmatrix} = \begin{bmatrix}1 & 4 & 1\\ 0 & 4 & -1\\ 1 & 2 & 4\end{bmatrix}.$$

So, we see that this elementary matrix applies the row operation! Continuing through all three row operations, we find

$$\underbrace{\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & \frac{1}{2} & 1\end{bmatrix}}_{E_3}\underbrace{\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ -1 & 0 & 1\end{bmatrix}}_{E_2}\underbrace{\begin{bmatrix}1 & 0 & 0\\ -2 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}}_{E_1}\underbrace{\begin{bmatrix}1 & 4 & 1\\ 2 & 12 & 1\\ 1 & 2 & 4\end{bmatrix}}_{A} = \underbrace{\begin{bmatrix}1 & 4 & 1\\ 0 & 4 & -1\\ 0 & 0 & \frac{5}{2}\end{bmatrix}}_{U}. \tag{1.3}$$

1.5.1 ▪ The inverse of an elementary matrix of type 1

An elementary row operation of type 1, $aR_j + R_i \to R_i$ does not change R_j. To reverse, or undo, this operation we should add $-aR_j$ to R_i. Or, in other words, the elementary row operation $-aR_j + R_i \to R_i$ is inverse to $aR_j + R_i \to R_i$ and this then implies that, for example,

$$\begin{bmatrix}1 & 0 & 0\\ a & 1 & 0\\ 0 & 0 & 1\end{bmatrix}^{-1} = \begin{bmatrix}1 & 0 & 0\\ -a & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$$

which is easily verified by seeing that

$$\begin{bmatrix}1 & 0 & 0\\ -a & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ a & 1 & 0\\ 0 & 0 & 1\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}.$$

This discussion shows us that

$$E_1^{-1} = \begin{bmatrix}1 & 0 & 0\\ 2 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}, \quad E_2^{-1} = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 1 & 0 & 1\end{bmatrix}, \quad E_3^{-1} = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & -\frac{1}{2} & 1\end{bmatrix}.$$

At this point we can work abstractly with (1.3) to solve for an expression for A

$$\begin{aligned}
E_3E_2E_1A &= U, \\
E_3^{-1}E_3E_2E_1A &= E_3^{-1}U, \\
E_2E_1A &= E_3^{-1}U, \\
E_2^{-1}E_2E_1A &= E_2^{-1}E_3^{-1}U, \\
E_1A &= E_2^{-1}E_3^{-1}U, \\
E_1^{-1}E_1A &= E_1^{-1}E_2^{-1}E_3^{-1}U, \\
A &= \underbrace{E_1^{-1}E_2^{-1}E_3^{-1}}_{L}U.
\end{aligned}$$

In principle, computing the matrix L requires multiplying three 3×3 matrices. If we were working with a 4×4 matrix we'd be looking at multiplying six 4×4 matrices! This is too much work (for us or for a computer) and that is part of the magic of Gaussian elimination:

$$E_1^{-1}E_2^{-1}E_3^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & -\frac{1}{2} & 1 \end{bmatrix}.$$

It is important to note that this is not how matrix-matrix multiplication works in general, but because of the specific structure of these matrices we can just fill in entrys to compute the product of the inverses!

Algorithm 6: *LU* factorization `LU`

```
**Result:** $L$ and $U$ such that $LU = A$, or failure
**Input:** $A \in \mathbb{R}^{n \times n+j}$, $j \geq 0$
begin
    set $L = I_m$;
    set $U = A$;
    for $j = 1$ to $n - 1$ do
        if $u_{jj} = 0$ then
            $A$ is not regular;
            return failure;
        end
        for $i = j + 1$ to $n$ do
            set $\ell_{ij} = u_{ij}/u_{jj}$;
            perform $-\ell_{ij}R_j + R_i \to R_i$ on $U$;
        end
    end
end
```

Chapter 2

Pivoting

2.1 ▪ Pivoting

In terms of solving linear systems, the previous sections give us all the elementary tools we need to deal with regular matrices. But what if the matrix is not regular? We go back to Problem 1.1 to find a matrix that is not regular but one that we were able to solve. A new elementary row operation is introduced.

Definition 2.1. *An elementary row operation of type 2, denoted $R_i \Leftrightarrow R_j$, corresponds to the interchange of rows i and j.*

Example 2.2.

$$\left[\begin{array}{cc|c} 0 & 1 & 1 \\ -1 & 1 & 2 \end{array}\right] \xrightarrow{R_1 \leftrightarrow R_2} \left[\begin{array}{cc|c} -1 & 1 & 2 \\ 0 & 1 & 1 \end{array}\right]$$

And to understand that a row interchange in an augmented matrix will not change the solution of the associated linear systems of equations we just note that it only corresponds to reordering the equations

$$\begin{aligned} 0a + b &= 1 \\ -a + b &= 2 \end{aligned} \qquad \Leftrightarrow \qquad \left[\begin{array}{cc|c} 0 & 1 & 1 \\ -1 & 1 & 2 \end{array}\right]$$

$$\begin{aligned} -a + b &= 2 \\ 0a + b &= 1 \end{aligned} \qquad \Leftrightarrow \qquad \left[\begin{array}{cc|c} -1 & 1 & 2 \\ 0 & 1 & 1 \end{array}\right]$$

Definition 2.3. *A permutation matrix is the product of any number of elementary matrices associated to row operations of type 2.*

Example 2.4. For a 2×2 matrix

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \xrightarrow{R_1 \leftrightarrow R_2} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

With the addition of row operations of type 2 we expand the class of matrices that can be reduced to upper-triangular form.

Definition 2.5. *A square matrix is called* nonsingular *if it can be reduced to upper-triangular form with non-zero diagonal entries, using elementary row operations of type 1 & 2.*

The following is self-explanatory, but we include it to be clear.

Definition 2.6. *A square matrix is called* singular *if it not nonsingular.*

We have argued that type 1 and type 2 row operations do not change the solution(s) of a linear system. And we know that back substitution succeeds. This implies that we can $\underline{\text{always}}$ find a solution of a linear system $A\mathbf{x} = \mathbf{b}$ for any choice of $\mathbf{b}$ if A is nonsingular. And since we find only one solution we have proved part (2) of the following:

Theorem 2.7.

1. *If $n \times n$ linear system $A\mathbf{x} = \mathbf{b}$ has a unique solution for $\underline{\text{every}}$ choice of right-hand side $\mathbf{b}$ then A is nonsingular.*

2. *If $A \in \mathbb{R}^{n\times n}$ is nonsingular then $A\mathbf{x} = \mathbf{b}$ for $\underline{\text{every}}$ choice of right-hand side vector $\mathbf{b}$.*

We will prove part (1) below which shows that a matrix A being nonsingular is the *de facto* condition for being able to solve a $n \times n$ linear system.

Let us now demonstrate the process of solving a linear system with a nonsingular matrix. Consider

$$\begin{aligned} 2y + 3z + w &= 1, \\ x + 3y + z + w &= -1, \\ x - y - 5z + w &= 0, \\ x + y + z + w &= 0. \end{aligned}$$

This is converted to the augmented matrix

$$\left[\begin{array}{cccc|c} 0 & 2 & 3 & 1 & 1 \\ 1 & 3 & 1 & 1 & -1 \\ 1 & -1 & -5 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{array}\right]$$

Then we proceed with Gaussian elimination

$$\left[\begin{array}{cccc|c} 0 & 2 & 3 & 1 & 1 \\ 1 & 3 & 1 & 1 & -1 \\ 1 & -1 & -5 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{array}\right] \xrightarrow{R_1 \leftrightarrow R_2} \left[\begin{array}{cccc|c} 1 & 3 & 1 & 1 & -1 \\ 0 & 2 & 3 & 1 & 1 \\ 1 & -1 & -5 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{array}\right]$$

$$\left[\begin{array}{cccc|c} 1 & 3 & 1 & 1 & -1 \\ 0 & 2 & 3 & 1 & 1 \\ 1 & -1 & -5 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{array}\right] \xrightarrow{-R_1 + R_3 \to R_3} \left[\begin{array}{cccc|c} 1 & 3 & 1 & 1 & -1 \\ 0 & 2 & 3 & 1 & 1 \\ 0 & -4 & -6 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{array}\right]$$

$$\left[\begin{array}{cccc|c} 1 & 3 & 1 & 1 & -1 \\ 0 & 2 & 3 & 1 & 1 \\ 0 & -4 & -6 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{array}\right] \xrightarrow{-R_1 + R_4 \to R_4} \left[\begin{array}{cccc|c} 1 & 3 & 1 & 1 & -1 \\ 0 & 2 & 3 & 1 & 1 \\ 0 & -4 & -6 & 0 & 1 \\ 0 & -2 & 0 & 0 & 1 \end{array}\right]$$

$$\left[\begin{array}{cccc|c} 1 & 3 & 1 & 1 & -1 \\ 0 & 2 & 3 & 1 & 1 \\ 0 & -4 & -6 & 0 & 1 \\ 0 & -2 & 0 & 0 & 1 \end{array}\right] \xrightarrow{2R_2 + R_3 \to R_3} \left[\begin{array}{cccc|c} 1 & 3 & 1 & 1 & -1 \\ 0 & 2 & 3 & 1 & 1 \\ 0 & 0 & 0 & 2 & 3 \\ 0 & -2 & 0 & 0 & 1 \end{array}\right]$$

$$\left[\begin{array}{cccc|c} 1 & 3 & 1 & 1 & -1 \\ 0 & 2 & 3 & 1 & 1 \\ 0 & 0 & 0 & 2 & 3 \\ 0 & -2 & 0 & 0 & 1 \end{array}\right] \xrightarrow{R_2 + R_4 \to R_4} \left[\begin{array}{cccc|c} 1 & 3 & 1 & 1 & -1 \\ 0 & 2 & 3 & 1 & 1 \\ 0 & 0 & 0 & 2 & 3 \\ 0 & 0 & 3 & 1 & 2 \end{array}\right].$$

We need to perform yet another row interchange

$$\left[\begin{array}{cccc|c} 1 & 3 & 1 & 1 & -1 \\ 0 & 2 & 3 & 1 & 1 \\ 0 & 0 & 0 & 2 & 3 \\ 0 & 0 & 3 & 1 & 2 \end{array}\right] \xrightarrow{R_3 \leftrightarrow R_4} \left[\begin{array}{cccc|c} 1 & 3 & 1 & 1 & -1 \\ 0 & 2 & 3 & 1 & 1 \\ 0 & 0 & 3 & 1 & 2 \\ 0 & 0 & 0 & 2 & 3 \end{array}\right],$$

and this finalizes the transformation to upper-triangular form. Then we proceed with back substitution:

$$\begin{aligned} 2w = 3 &\Rightarrow w = \frac{2}{3}, \\ 3z + w = 2 &\Rightarrow z = \frac{1}{6}, \\ 2y + 3z + w = 1 &\Rightarrow y = -\frac{1}{2}, \\ x + 3y + z + w = -1 &\Rightarrow x = -\frac{7}{6}. \end{aligned}$$

It should be clear that this is not a true "algorithm" at this point because there was an implicit choice made. In the first step, we swapped the first and second rows. We could have swapped the first and third rows instead. And even if we do not "need" to swap, we could anyway. The "simplest" algorithm is that when a zero is encountered that would make regular Gaussian elimination fail, we interchange the first row below that has a non-zero entry in that column. This is stated more precisely in the following algorithm.

Algorithm 7: nonsingular Gaussian elimination GE

```
Result: An upper triangular matrix, or failure
Input: $A \in \mathbb{R}^{n \times n+j}$, $j \geq 0$
begin
    for $j = 1$ to $n-1$ do
        if $a_{kj} = 0$ for all $k \geq j$ then
            $A$ is singular;
            return failure;
        end
        find the smallest $k$ such that $a_{kj} \neq 0$ then
            perform $R_k \leftrightarrow R_j$ on $A$;
        end
        for $i = j+1$ to $n$ do
            set $\ell_{ij} = a_{ij}/a_{jj}$;
            perform $-\ell_{ij}R_j + R_i \to R_i$ on $A$;
        end
    end
end
```

2.2 ▪ Pivoting strategies

The choice of which row to swap, and if to swap it, is called the pivoting strategy. The most commonly used method used is called *partial pivoting*. And to fully understand the need for this we present a simple numerical example.

Example 2.8. Consider the linear system

$$\begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 1-\epsilon \end{bmatrix}.$$

We can verify that the solution of this is given by

$$x = \frac{\epsilon}{\epsilon - 1}, \quad y = \frac{\epsilon - 1 - \epsilon^2}{\epsilon - 1}.$$

```
eps = 1e-8;
format long
A = [eps,1;1,1];
b = [1;1-eps];
U = [A,b];
U(2,:) = U(2,:) - (U(2,1)/U(1,1))*U(1,:);
x2 = U(2,3)/U(2,2);
x1 = 1/U(1,1)*(U(1,3) - U(1,2)*x2);

x1_real = eps/(eps-1);
x2_real = (eps-1-eps^2)/(eps-1);
```

```
err1 =

   -1.220446039250313e-08
```

```
err2 =

     0
```

We encounter an error that is significant! This is due to rounding errors — computers typically working with floating point arithmetic — due to inexact arithmetic.

But if we interchange the rows of the system first, we see significantly better rounding errors!

```
A = [1,1;eps,1];
b = [1-eps;1];
U = [A,b];
U(2,:) = U(2,:) - (U(2,1)/U(1,1))*U(1,:);
x2 = U(2,3)/U(2,2);
x1 = 1/U(1,1)*(U(1,3) - U(1,2)*x2);

err1 = x1-x1_real
err2 = x2-x2_real
```

```
err1 =

    -1.722921957828615e-16


err2 =

     0
```

The heuristics that produce the partial pivoting are that we want to maximize the diagonal entries in our upper-triangular matrix.

Algorithm 8: Gaussian elimination with partial pivoting `GEpp`

```
Result: An upper triangular matrix, or failure
Input: $A \in \mathbb{R}^{n\times n+j}$, $j \geq 0$
begin
    for $j = 1$ to $n-1$ do
        if $a_{kj} = 0$ for all $k \geq j$ then
            $A$ is singular;
            return failure;
        end
        find $k$ such that $|a_{kj}| \geq |a_{\ell j}|$ for all $\ell \geq j$ then
            perform $R_k \leftrightarrow R_j$ on $A$;
        end
        for $i = j+1$ to $n$ do
            set $\ell_{ij} = a_{ij}/a_{jj}$;
            perform $-\ell_{ij}R_j + R_i \to R_i$ on $A$;
        end
    end
end
```

2.3 ▪ The permuted LU factorization

Once pivoting is involved, we will no longer just factor $A = LU$, we will see that we actually end up factoring $PA = LU$ for a permutation matrix P. Let us see how this works with the matrix

$$\begin{bmatrix} 0 & 2 & 3 & 1 \\ 1 & 3 & 1 & 1 \\ 1 & -1 & -5 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

First, write

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{will become } P} \begin{bmatrix} 0 & 2 & 3 & 1 \\ 1 & 3 & 1 & 1 \\ 1 & -1 & -5 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{will become } L} \underbrace{\begin{bmatrix} 0 & 2 & 3 & 1 \\ 1 & 3 & 1 & 1 \\ 1 & -1 & -5 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}}_{\text{will become } U}.$$

And we need some facts to proceed:

1. If P is the elementary matrix associated with $R_i \leftrightarrow R_j$ then $PP = I$
2. If P is the elementary matrix associated with $R_i \leftrightarrow R_j$ then multiplication on the right, AP, interchanges columns i and j of A.

Let P_1 be the elementary matrix associated with $R_1 \leftrightarrow R_2$. Then because $P_1P_1 = I$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 3 & 1 \\ 1 & 3 & 1 & 1 \\ 1 & -1 & -5 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} P_1P_1 \begin{bmatrix} 0 & 2 & 3 & 1 \\ 1 & 3 & 1 & 1 \\ 1 & -1 & -5 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

Then using fact (2)

$$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}0&2&3&1\\1&3&1&1\\1&-1&-5&1\\1&1&1&1\end{bmatrix}=\begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&3&1&1\\0&2&3&1\\1&-1&-5&1\\1&1&1&1\end{bmatrix}.$$

But we want the first matrix on the right-hand side to be upper triangular. Multiplying both sides by P_1 get us there

$$P_1\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}0&2&3&1\\1&3&1&1\\1&-1&-5&1\\1&1&1&1\end{bmatrix}=P_1\begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&3&1&1\\0&2&3&1\\1&-1&-5&1\\1&1&1&1\end{bmatrix},$$
$$\begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}0&2&3&1\\1&3&1&1\\1&-1&-5&1\\1&1&1&1\end{bmatrix}=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&3&1&1\\0&2&3&1\\1&-1&-5&1\\1&1&1&1\end{bmatrix}.$$

This might seem like more work than is really necessary for this step but it is really what is needed to deal with what happens further along in the process. Now we can perform the row operations $-R_1 + R_3 \to R_3$ and $-R_1 + R_4 \to R_4$ in the last matrix, accounting for this in the first matrix on the right-hand side, exactly as in the case of the LU factorization:

$$\begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}0&2&3&1\\1&3&1&1\\1&-1&-5&1\\1&1&1&1\end{bmatrix}=\begin{bmatrix}1&0&0&0\\0&1&0&0\\1&0&1&0\\1&0&0&1\end{bmatrix}\begin{bmatrix}1&3&1&1\\0&2&3&1\\0&-4&-6&0\\0&-2&0&0\end{bmatrix}.$$

We proceed with the next two row operations

$$\begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}0&2&3&1\\1&3&1&1\\1&-1&-5&1\\1&1&1&1\end{bmatrix}=\begin{bmatrix}1&0&0&0\\0&1&0&0\\1&-2&1&0\\1&-1&0&1\end{bmatrix}\begin{bmatrix}1&3&1&1\\0&2&3&1\\0&0&0&2\\0&0&3&1\end{bmatrix}.$$

And to finish the upper triangularization, we need to interchange the last two rows. So let P_2 be the elementary matrix for $R_3 \leftrightarrow R_4$, giving

$$\begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}0&2&3&1\\1&3&1&1\\1&-1&-5&1\\1&1&1&1\end{bmatrix}=\begin{bmatrix}1&0&0&0\\0&1&0&0\\1&-2&1&0\\1&-1&0&1\end{bmatrix}P_2P_2\begin{bmatrix}1&3&1&1\\0&2&3&1\\0&0&0&2\\0&0&3&1\end{bmatrix}.$$

Absorbing P_2 into neighboring matrices gives

$$\begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}0&2&3&1\\1&3&1&1\\1&-1&-5&1\\1&1&1&1\end{bmatrix}=\begin{bmatrix}1&0&0&0\\0&1&0&0\\1&-2&0&1\\1&-1&1&0\end{bmatrix}\begin{bmatrix}1&3&1&1\\0&2&3&1\\0&0&3&1\\0&0&0&2\end{bmatrix}.$$

Like in our first step, we have broken the lower-triangular structure of the first factor on the right-hand side. So, we solve this problem by multiplying through by P_2

$$P_2\begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}0&2&3&1\\1&3&1&1\\1&-1&-5&1\\1&1&1&1\end{bmatrix}=P_2\begin{bmatrix}1&0&0&0\\0&1&0&0\\1&-2&0&1\\1&-1&1&0\end{bmatrix}\begin{bmatrix}1&3&1&1\\0&2&3&1\\0&0&3&1\\0&0&0&2\end{bmatrix},$$

$$\underbrace{\begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}}_{P}\underbrace{\begin{bmatrix}0&2&3&1\\1&3&1&1\\1&-1&-5&1\\1&1&1&1\end{bmatrix}}_{A}=\underbrace{\begin{bmatrix}1&0&0&0\\0&1&0&0\\1&-1&1&0\\1&-2&0&1\end{bmatrix}}_{L}\underbrace{\begin{bmatrix}1&3&1&1\\0&2&3&1\\0&0&3&1\\0&0&0&2\end{bmatrix}}_{U}.$$

It takes quite a bit of work to prove that this procedure will always result in an LU product on the right. But once you do, a result of this fact is the following.

Theorem 2.9. *A matrix $A \in \mathbb{R}^{n\times n}$ is nonsingular if and only if it has a factorization $PA = LU$ where P is a permutation matrix, L is special lower triangular and U is upper triangular with non-zero diagonal entries.*

In code, and especially in Matlab, it is unnecessary to construct P as a matrix. This permutation matrix P is actually captured by the following vector of integers

$$P=\begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}\leftrightarrow p=\begin{bmatrix}2&1&4&3\end{bmatrix}$$

were the integer gives, for each column, the location of the non-zero entry. Note that Matlab easily lets us apply permutations using this notation:

```
P = [0,1,0,0;
     1,0,0,0;
     0,0,0,1;
     0,0,1,0];
p = [2,1,4,3];
V = [1,1;2,2;3,3;4,4]
P*V
V(p,:)
```

```
V =

     1     1
     2     2
     3     3
     4     4


ans =

     2     2
```

```
         1        1
         4        4
         3        3


ans =

         2        2
         1        1
         4        4
         3        3
```

We see that constructing P and applying to a matrix gives the same result as simply indexing with p. The $PA = LU$ factorization is summarized below. While we will work with the transpose more later, we define the transpose for a vector here:

Definition 2.10. *The transpose of a vector*

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix},$$

is a row vector with the same entries

$$\mathbf{v}^T = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}.$$

Conversely, the transpose of a row vector is a vector with the same entries

Algorithm 9: The $PA = LU$ factorization with partial pivoting `PLU`

```
Result: A permutation $\mathbf{p}$, a special lower-triangular matrix $L \in R^{n\times n}$ and a
        nonsingular matrix $U \in \mathbb{R}^{n\times n+j}$, or failure
Input: $A \in \mathbb{R}^{n\times n+j}$, $j \geq 0$
set $U = A$;
set $L = I_n$;
set $\mathbf{p} = \begin{bmatrix} 1 & 2 & \cdots & n \end{bmatrix}^T$;
begin
    for $j = 1$ to $n - 1$ do
        if $u_{kj} = 0$ for all $k \geq j$ then
            $A$ is singular;
            return failure;
        end
        find $k$ such that $|u_{kj}| \geq |u_{\ell j}|$ for all $\ell \geq j$ then
            perform $R_k \leftrightarrow R_j$ on $U$;
            perform $R_k \leftrightarrow R_j$ on $L$;
            perform a interchange of columns $k$ and $j$ of $L$;
            perform $R_k \leftrightarrow R_j$ on $\mathbf{p}$;
        end
        for $i = j + 1$ to $n$ do
            set $\ell_{ij} = u_{ij}/u_{jj}$;
            perform $-\ell_{ij}R_j + R_i \to R_i$ on $U$;
        end
    end
end
```

2.4 ▪ Reducing memory requirements in $PA = LU$

Let us to understand how large a matrix we can actually compute with. At the writing of these notes, a high-end laptop will typically come with 16 gigabytes (GBs) of ram. While we will get more into the the details of this in Chapter 4, a floating point number is represented using 64 bits — 64 0's and 1's. There are 8,589,934,592 bits in one GB. So, one GB can store 134,217,728 floating point numbers. This translates to 2,146,483,648 numbers that can be stored on such a laptop. A square $n \times n$ matrix has n^2 entries. So,

$$\sqrt{2,146,483,648} \approx 46,340.95\ldots.$$

So, if we ignore all of the memory requirements of an operating system we can store one 46340×46340 matrix in memory on the laptop. As Algorithm 9 is set up, we have to store 3 matrices! So the size of the matrix is cut by a factor of $\sqrt{3}$ (recall n^2 entries!). This would become even worse if the P matrix was not encoded as a row vector.

On this laptop Algorithm 9 can support a matrix of size at most 26754×26754. We can rework the algorithm to overwrite A as it proceeds so that the size of the matrix is not cut by $\sqrt{3}$. To describe this algorithm we introduce matrix indexing notation that is similar to MATLAB's. For a vector $\mathbf{v} = (v_j)_{j=1}^n$ we us $\mathbf{v}_{i:j}$ to denote the subvector of $\mathbf{v}$ consisting of entries i through j. For a matrix $A = (a_{ij})_{\substack{1\leq i\leq m \\ 1\leq j\leq n}}$ we use

$$A_{i:j,k:\ell}$$

to refer to the submatrix of A consisting of rows i through j and columns k through ℓ if $i = j$ or $k = \ell$ we use $A_{i,k:\ell}$ or $A_{i:j,k}$, respectively.

Algorithm 10: The memory optimized $PA = LU$ factorization with partial pivoting GEpp.

```
Result: A $n \times (n+1)$ matrix and a permutation $\mathbf{p}$, or failure
Input: $A \in \mathbb{R}^{n\times n}$
set $\mathbf{p} = \begin{bmatrix}1 & 2 & \cdots & n\end{bmatrix}^T$;
begin
    for $j = 1$ to $n-1$ do
        if $a_{kj} = 0$ for all $k \geq j$ then
            $A$ is singular;
            return failure;
        end
        find $k$ such that $|a_{kj}| \geq |a_{\ell j}|$ for all $\ell \geq j$ then
            perform $R_k \leftrightarrow R_j$ on $A$;
            perform $R_k \leftrightarrow R_j$ on $\mathbf{p}$;
        end
        for $i = j+1$ to $n$ do
            set $a_{ij} = a_{ij}/a_{jj}$;
            perform $-a_{ij}R_j + R_i \to R_i$ on $A_{1:n,j+1:n}$;
        end
    end
end
```

This algorithm cleverly exploits the fact that every entry of A in which Gaussian elimination eliminates an entry corresponds to an entry of L that is filled in. Furthermore, the entries of L should not be involved in any sort of row operation going forward, so the row operations are just performed on the last $n - j$ columns. We demonstrate how the output is used with some simple MATLAB code. Suppose we apply this algorithm to a matrix A:

```
A = randn(7)
```

```
A =

    1.6360   -0.0549    1.8089    0.9421    0.4128   -0.1318    0.4400
   -0.4251   -1.1187   -1.0799    0.3005   -0.9870    0.5954   -0.6169
    0.5894   -0.6264    0.1992   -0.3731    0.7596    1.0468    0.2748
   -0.0628    0.2495   -1.5210    0.8155   -0.6572   -0.1980    0.6011
   -2.0220   -0.9930   -0.7236    0.7989   -0.6039    0.3277    0.0923
   -0.9821    0.9750   -0.5933    0.1202    0.1769   -0.2383    1.7298
    0.6125   -0.6407    0.4013    0.5712   -0.3075    0.2296   -0.6086
```

And then suppose A is modified and outputted, along with p, by Algorithm 10:

```
[Anew,p] = GEpp(A)
```

```
Anew =
   -2.0220   -0.9930   -0.7236    0.7989   -0.6039    0.3277    0.0923
    0.4857    1.4573   -0.2418   -0.2678    0.4703   -0.3975    1.6850
    0.0311    0.1924   -1.4520    0.8422   -0.7289   -0.1317    0.2741
   -0.8091   -0.5890   -0.7444    2.0577   -0.3415   -0.1988    1.7112
   -0.2915   -0.6285    0.1127   -0.1961    0.8943    0.8684    1.6653
    0.2102   -0.6244    0.7429   -0.3209   -0.1503    0.4428    1.0118
   -0.3029   -0.6461   -0.0179    0.3184   -0.1016    0.4997   -0.3683


p =

     5
     6
     4
     1
     3
     2
     7
```

The factors L and U can be reconstructed simply[3].

```
L = eye(length(Anew)) + tril(Anew,-1)
U = triu(Anew)
```

```
L =

    1.0000         0         0         0         0         0         0
    0.4857    1.0000         0         0         0         0         0
    0.0311    0.1924    1.0000         0         0         0         0
   -0.8091   -0.5890   -0.7444    1.0000         0         0         0
   -0.2915   -0.6285    0.1127   -0.1961    1.0000         0         0
    0.2102   -0.6244    0.7429   -0.3209   -0.1503    1.0000         0
   -0.3029   -0.6461   -0.0179    0.3184   -0.1016    0.4997    1.0000


U =

   -2.0220   -0.9930   -0.7236    0.7989   -0.6039    0.3277    0.0923
         0    1.4573   -0.2418   -0.2678    0.4703   -0.3975    1.6850
         0         0   -1.4520    0.8422   -0.7289   -0.1317    0.2741
         0         0         0    2.0577   -0.3415   -0.1988    1.7112
         0         0         0         0    0.8943    0.8684    1.6653
         0         0         0         0         0    0.4428    1.0118
         0         0         0         0         0         0   -0.3683
```

Lastly, we can check to make sure, using `p`.

[3]Note that this is not necessary to do either forward substitution or backward substitution!

```
A(p,:) - L*U
```

```
ans =

   1.0e-15 *

         0         0         0         0         0         0         0
         0         0         0   -0.0139   -0.0278         0         0
         0         0         0         0         0         0         0
         0    0.1180         0   -0.3331   -0.0555         0   -0.1110
         0    0.1110   -0.0278         0         0         0   -0.1665
         0         0   -0.2220   -0.0555         0         0   -0.1110
         0         0    0.0555         0         0         0         0
```

This last snippet tells us that to compute PA we evaluate `A(p,:)`.

2.5 ▪ Using GEpp

We now describe how use the output of `GEpp` to solve a linear system. The benefit of the storage both L and U in one matrix is that if you think closely about how back and forward substitution algorithms work, you will notice that

`Forsub([L,b]) = Forsub([Anew,b])`

for any vector `b` and similarly

`Backsub([U,c]) = Backsub([Anew,c]).`

So, the solution procedure is given below.

```
A = randn(7);
b = ones(7,1);
[Anew,p] = GEpp(A);
c = Forsub([Anew,b(p)]);
x = Backsub([Anew,c])
```

```
x =

    1.1908
    1.6383
   -0.2902
   -0.0049
    0.0773
    1.7849
   -0.0917
```

We test this against the output of MATLAB's built in function to solve linear systems.

```
x_real = A \ b; % Matlab's notation for solve Ax=b
x_real - x % test our solution
```

```
ans =

   1.0e-15 *

    0.4441
    0.2220
   -0.1665
    0.0893
   -0.0971
    0.4441
   -0.0833
```

2.6 ▪ Gauss-Jordan elimination

Gauss-Jordan elimination is the process by which an augmented matrix is first reduced to upper-triangular form, and then the left-most square submatrix is reduced to the identity matrix. In order to accomplish this, we need our last row operation type.

Definition 2.11. *An elementary row operation of type 3, denoted $aR_i \to R_i$, corresponds to multiplication of row i by a non-zero scalar a.*

We now demonstrate Gauss-Jordan elimination on a 3×3 matrix, where we augment the matrix with the identity matrix. Note that the textbook reserves the term Gauss-Jordan elimination for the case when the matrix is augmented like this with the identity matrix. I will use this term more generally.

$$\left[\begin{array}{ccc|ccc} 1 & 4 & 1 & 1 & 0 & 0 \\ 2 & 12 & 1 & 0 & 1 & 0 \\ 1 & 2 & 4 & 0 & 0 & 1 \end{array}\right] \xrightarrow{-2R_1+R_2\to R_2} \left[\begin{array}{ccc|ccc} 1 & 4 & 1 & 1 & 0 & 0 \\ 0 & 4 & -1 & -2 & 1 & 0 \\ 1 & 2 & 4 & 0 & 0 & 1 \end{array}\right]$$

$$\left[\begin{array}{ccc|ccc} 1 & 4 & 1 & 1 & 0 & 0 \\ 0 & 4 & -1 & -2 & 1 & 0 \\ 1 & 2 & 4 & 0 & 0 & 1 \end{array}\right] \xrightarrow{-R_1+R_3\to R_3} \left[\begin{array}{ccc|ccc} 1 & 4 & 1 & 1 & 0 & 0 \\ 0 & 4 & -1 & -2 & 1 & 0 \\ 0 & -2 & 3 & -1 & 0 & 1 \end{array}\right]$$

$$\left[\begin{array}{ccc|ccc} 1 & 4 & 1 & 1 & 0 & 0 \\ 0 & 4 & -1 & -2 & 1 & 0 \\ 0 & -2 & 3 & -1 & 0 & 1 \end{array}\right] \xrightarrow{\frac{1}{2}R_2+R_3\to R_3} \left[\begin{array}{ccc|ccc} 1 & 4 & 1 & 1 & 0 & 0 \\ 0 & 4 & -1 & -2 & 1 & 0 \\ 0 & 0 & \frac{5}{2} & -2 & \frac{1}{2} & 1 \end{array}\right]$$

Then, we divide the last row by the value on the diagonal

$$\left[\begin{array}{ccc|ccc} 1 & 4 & 1 & 1 & 0 & 0 \\ 0 & 4 & -1 & -2 & 1 & 0 \\ 0 & 0 & \frac{5}{2} & -2 & \frac{1}{2} & 1 \end{array}\right] \xrightarrow{\frac{2}{5}R_3\to R_3} \left[\begin{array}{ccc|ccc} 1 & 4 & 1 & 1 & 0 & 0 \\ 0 & 4 & -1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -\frac{4}{5} & \frac{1}{5} & \frac{2}{5} \end{array}\right]$$

The $(3, 3)$ entry is then used to eliminate entries above the diagonal

$$\left[\begin{array}{ccc|ccc} 1 & 4 & 1 & 1 & 0 & 0 \\ 0 & 4 & -1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -\frac{4}{5} & \frac{1}{5} & \frac{2}{5} \end{array}\right] \xrightarrow{R_3+R_2\to R_2} \left[\begin{array}{ccc|ccc} 1 & 4 & 1 & 1 & 0 & 0 \\ 0 & 4 & 0 & -\frac{14}{5} & \frac{6}{5} & \frac{2}{5} \\ 0 & 0 & 1 & -\frac{4}{5} & \frac{1}{5} & \frac{2}{5} \end{array}\right]$$

$$\left[\begin{array}{ccc|ccc} 1 & 4 & 1 & 1 & 0 & 0 \\ 0 & 4 & 0 & -\frac{14}{5} & \frac{6}{5} & \frac{2}{5} \\ 0 & 0 & 1 & -\frac{4}{5} & \frac{1}{5} & \frac{2}{5} \end{array}\right] \xrightarrow{-R_3+R_1\to R_1} \left[\begin{array}{ccc|ccc} 1 & 4 & 0 & \frac{9}{5} & -\frac{1}{5} & -\frac{2}{5} \\ 0 & 4 & 0 & -\frac{14}{5} & \frac{6}{5} & \frac{2}{5} \\ 0 & 0 & 1 & -\frac{2}{5} & \frac{1}{5} & \frac{2}{5} \end{array}\right].$$

Right now, we could use row 2 to eliminate the $(1,2)$ entry. But we will follow the more algorithmic method of first dividing then then eliminating. So, we divide the second row by 4.

$$\left[\begin{array}{ccc|ccc} 1 & 4 & 0 & \frac{9}{5} & -\frac{1}{5} & -\frac{2}{5} \\ 0 & 4 & 0 & -\frac{14}{5} & \frac{6}{5} & \frac{2}{5} \\ 0 & 0 & 1 & -\frac{2}{5} & \frac{1}{5} & \frac{2}{5} \end{array}\right] \xrightarrow{\frac{1}{4}R_2 \to R_2} \left[\begin{array}{ccc|ccc} 1 & 4 & 0 & \frac{9}{5} & -\frac{1}{5} & -\frac{2}{5} \\ 0 & 1 & 0 & -\frac{7}{10} & \frac{3}{10} & \frac{1}{10} \\ 0 & 0 & 1 & -\frac{2}{5} & \frac{1}{5} & \frac{2}{5} \end{array}\right].$$

The last step of Gauss-Jordan elimination is to eliminate the $(2,1)$ entry

$$\left[\begin{array}{ccc|ccc} 1 & 4 & 0 & \frac{9}{5} & -\frac{1}{5} & -\frac{2}{5} \\ 0 & 1 & 0 & -\frac{7}{10} & \frac{3}{10} & \frac{1}{10} \\ 0 & 0 & 1 & -\frac{2}{5} & \frac{1}{5} & \frac{2}{5} \end{array}\right] \xrightarrow{-4R_2 + R_1 \to R_1} \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & \frac{23}{5} & -\frac{7}{5} & -\frac{4}{5} \\ 0 & 1 & 0 & -\frac{7}{10} & \frac{3}{10} & \frac{1}{10} \\ 0 & 0 & 1 & -\frac{2}{5} & \frac{1}{5} & \frac{2}{5} \end{array}\right].$$

We say that Gauss-Jordan elimination succeeds if the left block becomes the identity matrix.

Theorem 2.12.

- *Gauss-Jordan elimination succeeds on* $\left[\begin{array}{c|c} A & I \end{array}\right]$ *for* $A \in \mathbb{R}^{n\times n}$ *if and only if* A *is nonsingular.*
- *If Gauss-Jordan elimination succeeds on* $\left[\begin{array}{c|c} A & I \end{array}\right]$ *it produces* $\left[\begin{array}{c|c} I & B \end{array}\right]$ *where* $BA = I$*, or* $B = A^{-1}$*.*

The first part of this theorem follows from the fact that a matrix is nonsingular if and only if we can reduce it to upper-triangular form, with non-zero diagonal entries, with type 1 and 2 row operations. Then we can divide by the diagonal entries and elimination what is above the diagonal. The second part follows from the fact that all row operations can be expressed as elementary matrices. Let $E_1, E_2, \ldots, E_k$ be the row operations that are sequentially performed:

$$E_k E_{k-1} \cdots E_1 \left[\begin{array}{c|c} A & I \end{array}\right] = \left[\begin{array}{c|c} I & B \end{array}\right].$$

Then it follows that both $E_k E_{k-1} \cdots E_1 A = I$ and $E_k E_{k-1} \cdots E_1 = B$.

To compute the inverse of a nonsingular matrix $A \in \mathbb{R}^{n\times n}$, apply Gauss–Jordan elimination to $\left[\begin{array}{c|c} A & I \end{array}\right]$ and take the augmented matrix!

It also follows from this that if A is nonsingular then applying Gauss-Jordan elimination is effectively the same as computing

2.7 ▪ Using the inverse matrix

Suppose that we use Gauss–Jordan elimination to compute the inverse A^{-1} for a nonsingular matrix A. Then if we want to solve

$$\begin{aligned} A\mathbf{x} &= \mathbf{b}, \\ \underbrace{A^{-1}A}_{I}\mathbf{x} &= A^{-1}\mathbf{b}, \\ \mathbf{x} &= A^{-1}\mathbf{b}. \end{aligned}$$

Definition 2.13. *Let $\mathbf{e}_j$ be the j column of the identity matrix I.*

In the previous definition, the dimension of the vector will be inferred from context. Returning to the context Theorem 2.7(1), we see that if we can solve $A\mathbf{x} = \mathbf{b}$ for every choice of $\mathbf{b}$, we can then, in particular, solve

$$A\mathbf{x}_j = \mathbf{e}_j, \quad \text{for} \quad j = 1, 2, \ldots, n, \tag{2.1}$$

To solve all of these systems at once, we could use the augmented matrix

$$\left[\begin{array}{c|ccc} A & \mathbf{e}_1 & \mathbf{e}_2 & \cdots \mathbf{e}_n \end{array}\right],$$

put A in upper triangular form using row operations of type 1 and 2, and the preform back substitution. It can be seen that this gives the same result as Gauss-Jordan elimination — x_j is the jth column of the inverse A^{-1}! This tells us the following.

Theorem 2.14. *For a square nonsingular matrix A,*

$$A^{-1}A = I = AA^{-1}.$$

Note that this does not prove Theorem 2.7(1)! We will do so in Section 2.9.

2.8 ▪ Complexity theory

At this point, one should be wondering why we waited so long to find a method to compute the inverse of a matrix, spending all this time on the $PA = LU$ factorization, if all we want to do is solve $A\mathbf{x} = \mathbf{b}$.

Why compute $PA = LU$ instead of A^{-1}?

The answer to this question we need to discuss the complexity of the operations we have described. We do this using FLOPs (floating point operations). We count addition, multiplication, subtraction and division as a FLOP.

Example 2.15 (Matrix-vector multiplication). The multiplication $A\mathbf{v}$, $A \in \mathbb{R}^{n\times n}$, $\mathbf{v} \in \mathbb{R}$ requires n^2 multiplications and $n(n-1) = n^2 - n$ additions.

So, if we know the inverse of matrix, we need to perform $2n^2 - n$ FLOPs. We now compare this with forward and back substitution

Example 2.16 (Forward substitution). Suppose L is a special lower-triangular matrix ($\ell_j j = 1$ for all j). Forward substituion to solve $L\mathbf{c} = \mathbf{b}$ requires $\frac{n^2-n}{2}$ additions and $\frac{n^2-n}{2}$ multiplications.

Example 2.17 (Back substitution). Suppose U is an upper-triangular matrix with non-zero diagonal entries. Back substituion to solve $U\mathbf{x} = \mathbf{c}$ requires $\frac{n^2-n}{2}$ additions and $\frac{n^2+n}{2}$ multiplications/divisions.

From these we conclude that it requires $2n^2 - n$ FLOPs to solve $A\mathbf{x} = \mathbf{b}$ with the $PA = LU$ factorization.

This is the same amount of "work" that is required to use A^{-1}. From a computational perspective, having the $PA = LU$ factorization is just as good as having the actual inverse!

But then we just make the clear observation that computing the inverse takes a lot more work than computing the $PA = LU$ factorization — twice as much, actually — and so in practice the inverse of a matrix is almost never calculated.

2.9 ▪ Singular matrices and determinants

We first discuss what happens when we perform row operations on a singular matrix. If $A \in \mathbb{R}^{n\times n}$ is singular then at some point we must encounter a situation where row interchanges do not allow us to put a non-zero entry on the diagonal of a matrix. That means the matrix U at this intermediate stage must be of the form

$$\begin{bmatrix} u_{11} & u_{1,2} & \cdots & \cdots & & & & \cdots & u_{1,n} \\ 0 & u_{22} & u_{2,3} & \cdots & & & & \cdots & u_{2,n} \\ 0 & 0 & \ddots & & & & & & \vdots \\ \vdots & \vdots & & & \ddots & & & & \\ 0 & 0 & \cdots & 0 & u_{k,k} & u_{k,k+1} & \cdots & \cdots & u_{k,n} \\ 0 & 0 & \cdots & 0 & 0 & \boxed{0} & u_{k+1,k+2} & \cdots & u_{k+1,n} \\ 0 & 0 & \cdots & 0 & 0 & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & u_{n,k+2} & \cdots & u_{n,n} \end{bmatrix}.$$

The zero that is highlighted is on the diagonal. We can continue to apply row operations on the lower right block:

$$\begin{bmatrix} u_{k+1,k+2} & \cdots & u_{k+1,n} \\ \vdots & & \vdots \\ u_{n,k+2} & \cdots & u_{n,n} \end{bmatrix}.$$

But we note that his has more rows than columns! And if we put this into upper-triangular form, there must be a zero in the (n, n) entry.

Proposition 2.18. *A singular matrix A can be reduced to upper triangular form with elementary row operations of type 1 and type 2 and the resulting upper-triangular matrix must have a row of all zeros.*

Consider the matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

we wish to check if it is nonsingular. First, suppose $a \neq 0$. Then we do no need to interchange rows. We perform the one needed row operation

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \xrightarrow{R_2 - \frac{c}{a}R_1 \to R_2} \begin{bmatrix} a & b \\ 0 & d - \frac{c}{a}b \end{bmatrix}$$

Then the matrix is nonsingular if $d - \frac{c}{a}b \neq 0$ or since $a \neq 0$ this is equivalent to $ad - bc \neq 0$. Similarly, if $a = 0$, then either $d = 0$ and the matrix is singular, or if $d \neq 0$, one can interchange rows and do the reduction. The same condition $ad-bc \neq 0$ for nonsingularity follows! This scalar quantity $ad-bc$ is the determinant of a general 2×2 matrix and we will see that it is nonzero if and only if the matrix is nonsingular. But first, we work on the general definition for the determinant of an $n \times n$ matrix.

Theorem 2.19. *Associated with every square matrix, there exists a uniquely defined scalar quantity, known as its* determinant *and denoted* $\det A$*, that obeys the following axioms:*

1. *Row operations of type 1 do not change the determinant.*
2. *Row operations of type 2 change the determinant by a factor of* -1.
3. *Row operation of type 3 with a scalar* a *multiplies the determinant by the same factor* a
4. *The determinant of an upper-triangular matrix* U *is equal to the product of the diagonal entries:* $\det U = u_{11}u_{22}\cdots u_{nn}$.

We have some immediate consequences:

- $\det I = 1$
- Suppose that A has a row of all zeros, say row i. Then axiom (3) states that

$$A \xrightarrow{aR_i \to R_i} \tilde{A},$$
$$\det A \xrightarrow{aR_i \to R_i} a \det \tilde{A}.$$

  But then since row i is a row of all zeros, $\tilde{A} = A$, and we choose $a = 0$ to see that $\det A = 0$ if A has a row of all zeros!

- If $A = LU$ then $\det A = \det U$. This follows from axiom (1).
- If $PA = LU$ then $\det A = (-1)^k \det U$ where k is the number of row interchanges that are performed.
- $\det A = 0$ if and only if A is singular. This follows from Proposition 2.18.

We continue with some more involved properties.

Proposition 2.20. *Suppose* $A, B \in \mathbb{R}^{n\times}$ *then*

$$\det(AB) = \det A \det B.$$

Proof. First, suppose A is nonsingular. Then Gauss-Jordan elimination shows that we can express A as a product of elementary matrices of type 1,2 & 3, $A = E_k E_{k-1} \cdots E_1$.

We can then see that

$$\begin{aligned} I &\xrightarrow{R_i+aR_j\to R_i} E_1, \\ 1 = \det I &\xrightarrow{R_i+aR_j\to R_i} \det E_1 = 1, \\ I &\xrightarrow{R_i\leftrightarrow R_j, j\neq i} \det E_2, \\ 1 = \det I &\xrightarrow{R_i\leftrightarrow R_j, j\neq i} \det E_2 = -1, \\ I &\xrightarrow{aR_i\to R_i} \det E_3, \\ 1 = \det I &\xrightarrow{aR_i\to R_i} \det E_3 = a. \end{aligned}$$

So if we perform the row operations associated with $E_1, E_2, \dots, E_k$ to the identity matrix to form A, we see that

$$\det A = \det E_k \det E_{k-1} \cdots \det E_1.$$

Then if we consider AB, we can interpret it as applying the associated row operations that combine to create A, it follows that

$$\det(AB) = \det(E_k E_{k-1} \cdots E_1 B) = \det A \det B.$$

Proposition 2.21. *Suppose A is nonsingular, then* $\det A^{-1} = \dfrac{1}{\det A}$.

Now, to finally prove Theorem 2.7(1). If $A\mathbf{x} = \mathbf{b}$ can be solved for every choice then in reviewing (2.1), we see that in defining

$$X = \begin{bmatrix}\mathbf{x}_1 & \mathbf{x}_2 & \cdots x_n\end{bmatrix} \tag{2.2}$$

we have

$$AX = I \Rightarrow \det A \det X = 1.$$

This implies that $\det A \neq$ and therefore A is nonsingular.

We now illustrate the computation of determinants.

Example 2.22. Compute the determinant of

$$A = \begin{bmatrix} 1 & 4 & 1 \\ 2 & 12 & 1 \\ 1 & 2 & 4 \end{bmatrix}.$$

In reviewing the calculations in Section 2.6 (or in the previous chapter) we see that this matrix is reduced to upper triangular form

$$U = \begin{bmatrix} 1 & 4 & 1 \\ 0 & 4 & -1 \\ 0 & 0 & \frac{5}{2} \end{bmatrix},$$

using only row operations of type 1. Thus $\det A = \det U = (1)(4)(5/2) = 10$.

Example 2.23. Compute the determinant of

$$A = \begin{bmatrix} 0 & 2 & 3 & 1 \\ 1 & 3 & 1 & 1 \\ 1 & -1 & -5 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

In reviewing the calculations in Section 2.1 we use row operations of type 1 and 2 to get to

$$U = \begin{bmatrix} 1 & 3 & 1 & 1 \\ 0 & 2 & 3 & 1 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

Since we used two interchanges we find

$$\det A = (-1)^2 \det U = (1)(2)(3)(2) = 12.$$

Note that we did not use row operations of type 3 in these preceding examples. We can indeed do that, but often that complicates things because scaling used has to be accounted for.

Remark 2.24. *You may have encountered other methods of computing determinants. These methods actually require more computation! But they are typically easier to perform by hand when variables are present in the matrix under consideration. When we introduce eigenvalues we will need to introduce other methods to evaluate determinants.*

Chapter 3

Iterative methods

3.1 ▪ Simple scalar iteration

Suppose that $|m| < 1$ and consider the iteration

$$x_0 = 0$$
$$x_{k+1} = mx_k + c.$$

If $\lim_{k\to\infty} x_k$ exists, say, $\lim_{k\to\infty} x_k = x$ then

$$x = mx + c.$$

So, then $x = \frac{c}{1-m}$! Notice here that $|m| < 1$ is sufficient to ensure that this equation makes sense. Let's take a moment to show that this limit exists.

$$\begin{aligned}
x_0 &= 0 \\
x_1 &= c \\
x_2 &= mc + c \\
x_3 &= m^2c + mc + c \\
x_4 &= m^3c + m^2c + mb + c \\
&\vdots \\
x_k &= m^{k-1}c + m^{k-2}c + \cdots + mc + c.
\end{aligned}$$

Then

$$x_k = \frac{1-m^k}{1-m}c \overset{k\to\infty}{\longrightarrow} \frac{c}{1-m}.$$

3.2 ▪ Matrix iteration

We can attempt to do the same steps with a matrix M, and a vector $\mathbf{b}$. But what is an analogous condition to $|m| < 1$? Well, if $|m| < 1$ if and only if $m^k \to 0$ as $k \to \infty$. And that is exactly what is needed in the above proof.

Definition 3.1. *An $n \times n$ matrix M is* convergent *if*

$$\lim_{k\to\infty} (M^k)_{ij} = 0,$$

for all $1 \leq i, j \leq n$.

So, suppose that M is convergent and consider the iteration

$$\mathbf{x}^{(0)} = \mathbf{0},$$
$$\mathbf{x}^{(k+1)} = M\mathbf{x}^{(k)} + \mathbf{c}.$$

Again suppose that $\lim_{k\to\infty} \mathbf{x}^{(k)} = \mathbf{x}$ for any $\mathbf{b}$. Indeed, this will hold if the matrix is convergent. Then we have

$$\mathbf{x} = M\mathbf{x} + \mathbf{c}$$

or, equivalently,

$$(I - M)\mathbf{x} = \mathbf{c}.$$

So, we can then solve $(I - M)\mathbf{x} = \mathbf{c}$ for any choice of $\mathbf{c}$ and $(I - M)$ must be nonsingular. We summarize this in the following theorem.

Theorem 3.2. *If M is convergent then $\lim_{k\to\infty} \mathbf{x}^{(k)} = \mathbf{x}$ and $(I - M)\mathbf{x} = \mathbf{c}$.*

Proof. To come!

We now illustrate this.

```
n = 10;
M = .3*(rand(n,n)-.5);
A = eye(n) - M;
c = rand(n,1);
y = zeros(n,1);
format long

y = M*y+c
```

```
y =

   0.100750511921236
   0.507848830829537
   0.585609125701878
   0.762887095910741
   0.082962649110544
   0.661596193082714
   0.516979014706213
   0.171048017525447
   0.938557864331842
   0.590483177142572
```

```
y = M*y+c
```

```
y =

   0.077727154976476
   0.363806457179105
   0.557369511603846
   0.826806199241477
   0.255422901714552
   0.654633360078442
   0.700934934901354
   0.112068576471026
   1.177176246248708
   0.583426758737004
```

```
y = M*y+c
```

```
y =

   0.082842695914941
   0.294356706610472
   0.540812014758675
   0.870628718117166
   0.267063702748974
   0.615778654434824
   0.709714248726689
   0.127461627133065
   1.174512522775538
   0.615045985569920
```

```
y = M*y+c
```

```
y =

   0.079412489145411
   0.281314175491463
   0.541724244583188
   0.868548783529157
   0.269134760081902
   0.607213140925380
   0.711363265789150
   0.141891164485511
   1.165261731102694
   0.619361991943737
```

Then we can check how close we are to a solution:

```
A*y-c
```

```
ans =
   0.001218592337001
  -0.000229910292611
  -0.000121057628151
   0.002162997640271
  -0.000325319008234
   0.001295842952326
  -0.000896903506264
  -0.003470421629003
   0.006721034684748
   0.000220713676622
```

3.3 ▪ Jacobi's Algorithm

Consider the linear system

$$A\mathbf{x} = \mathbf{c}, \quad A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 5 \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

Since A has no non-zero entries off the diagonal we say that A is a diagonal matrix. We apply the simple iteration from the previous example, with disasterous results!

```
A = diag([1,2,3,4,5]);
M = eye(5) - A;
c = [1,1,1,1,1]';
y = zeros(5,1);
format long

for i = 1:5
    y = A*y + c
end
```

```
y =

     1
     1
     1
     1
     1


y =

     2
     3
     4
     5
     6
```

```
y =

     3
     7
    13
    21
    31


y =

     4
    15
    40
    85
   156


y =

     5
    31
   121
   341
   781
```

This is clearly not converging, but the solution is easily obtained by row operations of type 3:

$$\mathbf{x} = \begin{bmatrix} 1 \\ 1/2 \\ 1/3 \\ 1/4 \\ 1/5 \end{bmatrix}.$$

Any "good" algorithm should work on a nonsingular diagonal matrix and Jacobi's algorithm is a modification of the simple iteration that fixes this issue. To describe this algorithm, we decompose $A \in \mathbb{R}^{n\times n}$ is a simple manner:

$$A = L + D + U.$$

Here L is the strictly lower-triangular of A, D is just the diagonal of A and U is the strictly upper-triangular part of A. Note that here, L, U <u>do not have any</u> relation to the LU factorization. Our goal is to solve

$$\begin{aligned} A\mathbf{x} &= \mathbf{b} \\ (L+D+U)\mathbf{x} &= \mathbf{b} \\ D^{-1}(L+D+U)\mathbf{x} &= D^{-1}\mathbf{b} \\ \mathbf{x} + D^{-1}(L+U)\mathbf{x} &= D^{-1}\mathbf{b}. \end{aligned}$$

This last equation is of the form:

$$(I - M)\mathbf{x} = \mathbf{c}, \quad M = -D^{-1}(L + U), \quad \mathbf{c} = D^{-1}\mathbf{b}.$$

Then Jacobi's algorithm is just the simple iteration applied to this equation:

$$\begin{aligned} \mathbf{x}^{(0)} &= \mathbf{0}, \\ \mathbf{x}^{(k+1)} &= -D^{-1}(L+U)\mathbf{x}^{(k)} + D^{-1}\mathbf{b}, \quad k = 0, 1, 2, \ldots . \end{aligned}$$

If we write out the entries of the iteration we see

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

Here we use the convention that $\sum_{j=i}^{k} = 0$ if $i > k$.

Algorithm 11: Jacobi's algorithm.

```
Result: An approximate solution of $A\mathbf{x} = \mathbf{b}$ or failure
Input: $A \in \mathbb{R}^{n\times n}$ with non-zero diagonal entries, $\mathbf{b} \in \mathbb{R}^n$, an error tolerance $\epsilon$
       and a maximum number of steps $K > 0$
decompose $A = L + D + U$;
set $\mathbf{y} = \mathbf{0}$, $\mathbf{z} = \mathbf{0}$;
begin
    for $k = 1$ to $K$ do
        for $i = 1$ to $n$ do
            set $z_i = \frac{1}{d_{ii}}\left(b_i - \sum_{j=1}^{i-1} \ell_{ij} y_j - \sum_{j=i+1}^{n} u_{ij} y_j\right)$;
        end
        if $\max_{i=1}^{n} |y_i - z_i| < \epsilon$ then
            return $\mathbf{z}$;
        end
        set $\mathbf{y} = \mathbf{z}$;
    end
    return Failed to converge;
end
```

This algorithm might appear very complicated, but the MATLAB implementation is actually quite simple.

```
A = diag([1,2,3,4,5]) + .1*randn(5);
L = tril(A,-1);
U = triu(A,+1);
LpU = L + U; % L + U
D = diag(A);  % this is just a vector
b = [1,1,1,1,1]';
y = zeros(5,1);
```

and then one iteration is given by:

```
% compute y = -D^{-1}(L + U)y + D^{-1}b
y = LpU*y;  % does y = (L+U)*y
y = (b-y)./D % does y = D^{-1}(y- b)
```

```
y =

   1.121717334469232
   0.525299224393024
   0.348722648710539
   0.238094431031120
   0.203743873727282
```

The full algorithm is then given by

```
K = 20; eps = 1e-5;
y = zeros(5,1);
for i = 1:K
    z = LpU*y;  % does y = (L+U)*y
    z = (b-z)./D; % does y = D^{-1}(y- b)
    if max(abs(y-z)) < eps
        y = z
        break;
    end
    y = z;
end
i
y
```

```
i =

     6


y =

   1.205185617349236
   0.609134473234669
   0.335089112402153
   0.299768194236575
   0.200559560326893
```

We then check the error

```
A*y-b
```

```
ans =

   1.0e-05 *

   0.040172683357653
   0.084002826383767
  -0.005611569531272
   0.127622150825069
  -0.006015806019999
```

3.4 ▪ The Gauss-Seidel algorithm

We can improve on Jacobi's algorithm while still using the $A = L+D+U$ decomposition. Jacobi's algorithm leveraged the fact that we can easily invert a diagonal matrix, or rather, solving $D\mathbf{x} = \mathbf{b}$ when D is a diagonal matrix. But we know that back substitution allows us to solve $(U + D)\mathbf{x} = \mathbf{b}$ because $U + D$ is an upper-triangular matrix. Gauss-Seidel uses this.

$$\begin{aligned} \mathbf{x}^{(0)} &= \mathbf{0}, \\ \mathbf{x}^{(k+1)} &= -(U+D)^{-1}L\mathbf{x}^{(k)} + (U+D)^{-1}\mathbf{b}, \quad k = 0, 1, 2, \ldots. \end{aligned}$$

If we write out the entries of the iteration, it looks $\underline{\text{very}}$ similar to Jacobi's algorithm but the iteration indices on the right-hand side are modified.

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

We again use the convention that $\sum_{j=i}^{k} = 0$ if $i > k$. Note that while the right-hand side depends on the iteration $k+1$, which we are in the process of determining, it only depends on values that have been determined, provided that $i = n, n-1, n-2, \ldots$.

Algorithm 12: Gauss-Seidel algorithm.

```
Result: An approximate solution of $A\mathbf{x} = \mathbf{b}$ or failure
Input: $A \in \mathbb{R}^{n\times n}$ with non-zero diagonal entries, $\mathbf{b} \in \mathbb{R}^n$, an error tolerance $\epsilon$
       and a maximum number of steps $K > 0$
decompose $A = L + D + U$;
set $\mathbf{y} = \mathbf{0}$, $\mathbf{z} = \mathbf{0}$;
begin
    for $k = 1$ to $K$ do
        for $i = n$ to $1$ with steps of $-1$ do
            set $\mathbf{z} = \mathbf{y}$;
            set $z_i = \frac{1}{d_{ii}}\left(b_i - \sum_{j=1}^{i-1} \ell_{ij} z_j - \sum_{j=i+1}^{n} u_{ij} z_j\right)$;
        end
        if $\max_{i=1}^{n} |y_i - z_i| < \epsilon$ then
            return $\mathbf{z}$;
        end
        set $\mathbf{y} = \mathbf{z}$;
    end
    return Failed to converge;
end
```

When we implement this in MATLAB we use matrix products and `Backsub` to simplify the calculations.

```
format long
A = diag([1,2,3,4,5]) + .1*randn(5);
UpD = triu(A);
L = tril(A,-1);
b = [1,1,1,1,1]';
y = zeros(5,1);
```

And one iteration is given by the following.

```
y = L*y;
y = Backsub([UpD,b-y])
```

```
y =

   1.013662166771282
   0.466425148512682
   0.333718136696037
   0.258290303921671
   0.202366035768882
```

The convergence is evident after just a couple of iterations.

```
y = L*y;
y = Backsub([UpD,b-y])
```

```
y =

   1.006409315749220
   0.435189514755413
   0.343010156594949
   0.238502700950548
   0.220848441572283
```

```
y = L*y;
y = Backsub([UpD,b-y])
```

```
y =

   1.006164937363233
   0.435365413217977
   0.342461609754539
   0.238489278211545
   0.221465576386166
```

```
A*y-b
```

```
ans =

   1.0e-04 *

                   0
  -0.153900509562988
  -0.067695517094180
   0.038155862203126
   0.086718538330199
```

3.4.1 ▪ Comparision with Jacobi's algorithm

We use the following code to compare the performance of Jacobi's algorithm to that of the Gauss-Seidel algorithm.

```
A = diag([1,2,3,4,5]) + .1*randn(5);
L = tril(A,-1);
U = triu(A,+1);
D = diag(A);
LpU = L + U; % L + U
UpD = triu(A);
b = [1,1,1,1,1]';

y_gs = zeros(5,1);
y_j = zeros(5,1);
x = A\b;  % solves Ax = b

T = 10;
err_gs = zeros(1,T);
err_j = zeros(1,T);

for i = 1:T
    y_j = LpU*y_j;
    y_j = (b-y_j)./D ;
    y_gs = L*y_gs;
    y_gs = Backsub([UpD,b-y_gs]);
    err_gs(i) = max(abs(y_gs-x));
    err_j(i) = max(abs(y_j-x));
end
hold off
semilogy(1:T,err_gs,'r')
hold on
grid on
semilogy(1:T,err_j,'b')
legend('Gauss-Seidel','Jacobi')
xlabel('Iteration number')
```

The output is displayed in Figure 3.1

3.5 ▪ Convergence

We have given examples when Jacobi's algorithm and the Gauss-Seidel algorithm work. But understanding when these algorithms work in general is important. Given $A \in \mathbb{R}^{n\times n}$ with non-zero diagonal entries decompose $A = L + D + U$ as before and define

$$M_{\mathrm{J}} = -D^{-1}(L+U), \quad M_{\mathrm{GS}} = -(D+U)^{-1}L.$$

It follows that if M_{J} is a convergent matrix then Jacobi's algorithm converges, and similarly, if M_{GS} is a convergent matrix then the Gauss-Seidel algorithm converges. But these conditions are really difficult to check! The next definition will help us give a condition that is simpler to check.

Definition 3.3. *$A \in \mathbb{R}^{n\times n}$ is said to be diagonally dominant if*

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|, \quad \text{for all} \quad i \leq i \neq n.$$

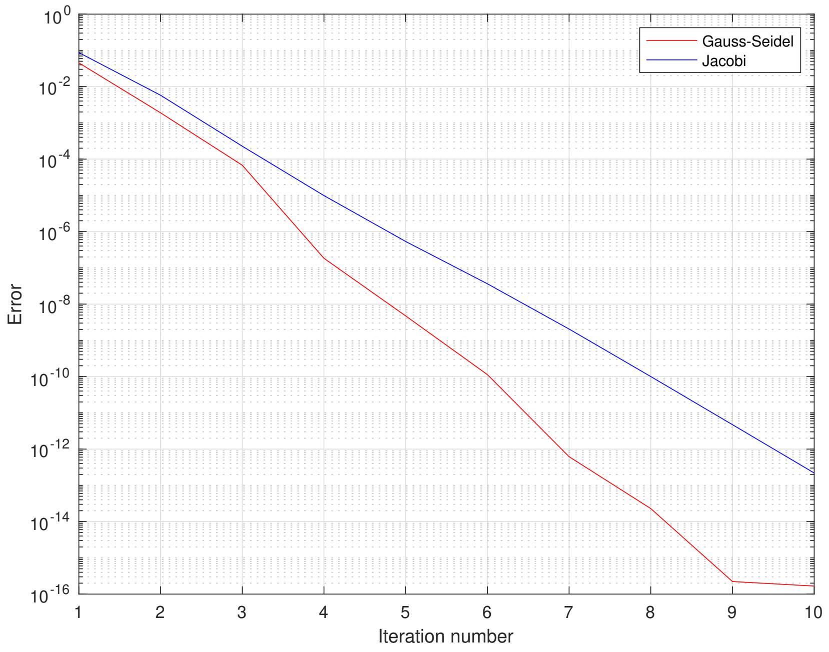

Figure 3.1. *A comparision of the errors in produced by the Jacobi's algorithm and the Gauss-Seidel algorithm*

A diagonally dominant matrix is a matrix whose diagonal entries, in absolute value, are larger than the sum of the absolute of the other entries in the row.

Example 3.4. The following matrix is diagonally dominant

$$\begin{bmatrix} 3 & 1 & 0 \\ 0 & -4 & 2 \\ 1 & 2 & 5 \end{bmatrix}$$

while the following matrix is not

$$\begin{bmatrix} 3 & 1 & 2 \\ 0 & -4 & 2 \\ 1 & 2 & 5 \end{bmatrix}.$$

Theorem 3.5. *Jacobi's algorithm and the Gauss-Seidel algorithm converge if A is diagonally dominant.*

Note that the algorithm may still converge if the matrix is not diagonally dominant, but it is just not guaranteed by this theorem.

Chapter 4

Rounding errors

sddsfsdf

Part II

Approximation theory

Chapter 5

Interpolation

To come!

Bibliography