22. Elimination for matrices

When working with matrices, it is often very helpful, sometimes even crucial, that they are of a special stair-like form, with zeros in the lower-left corner. Here are some typical examples.

$$\begin{pmatrix} 13 & 23 & 0 \\ 0 & 0 & 31 \\ 0 & 0 & 0 \end{pmatrix} \qquad \begin{pmatrix} 13 & 23 & 31 & -13 \\ 0 & -13 & 0 & 26 \\ 0 & 0 & 14 & 26 \end{pmatrix} \qquad \begin{pmatrix} 13 & 23 & 0 & -13 \\ 0 & 23 & 13 & 0 \\ 0 & 0 & 31 & -13 \\ 0 & 0 & 0 & 31 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

How do we recognize the proper shape? The first non-zero entry in each row must come after the first non-zero entry of the previous row.

Definition 22.1.
Let $A = (a_{i,j})_{i,j=1}^{n,m}$ be an $n \times m$ matrix. We say that this matrix is in the **row-echelon form** if the following condition is true:
For every $i = 2, \dots, n$ and $j = 2, \dots, m$, if $a_{i,j} \neq 0$ then there must be $k \in \{1, \dots, m-1\}$ such that $a_{i-1,k} \neq 0$.

We can visualize a stair-like shape drawn into the matrix so that there are zeros under it. It is possible to also have some zeros above, but not where the steps start.

$$\begin{pmatrix} 13 & -13 & 0 & 14 & 23 & 0 \\ 0 & 0 & 13 & 0 & 31 & -13 \\ 0 & 0 & 0 & 0 & 23 & 7 \\ 0 & 0 & 0 & 0 & 0 & -13 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

We actually prefer it when the stairs are regular and go along the diagonal, in other words if all steps have the width of number, like in the first two examples above. This is not always possible, and a matrix has to be special to be reducable to such a pleasant form. We actually have a name for such matrices when they are square: These are called regular or nonsingular matrices. They have many nice properties, for instance they are invertible, have non-zero determinant, they have full rank, and when they come from a system of linear equations then this system always has a unique solution.

A good news is that in numerical analysis we almost exclusively work with regular matrices, which will make our work easier. Indeed, we will soon forcus on systems of linear equations, and we usually consider systems that have unique solutions.

However, we often need to attach some extra columns on the right to create an extended matrix, and we would still want such an extended matrix to reduce into a nice shape, for instance like the first example above. Such matrices do not have a generally accepted name, so we will call them "extended regular matrices" here.

We can now state our main focus for this chapter: We want to transform extended regular matrices into a row-echelon form. The main tool for this is the Gaussian elimination. The reader is probably familiar with this method, and may want to skip (or skim) the next session where we introduce it. We will address it there in full generality for all matrices of any dimension.

22a. Introducing elimination

Elimination is a process where we gradually transform the given matrix into the desired chape by applying row operations. Row operations are certain mathematical moves that apply to rows of matrices. This is the key concept. Elimination sees every matrix as a collection of rows and ma-

nipulates them as units. Of course, these operations are then executed by working with individual entries, but elimination does not care about it, it is a question of implementation.

Operations with matrix rows are just like operations with vectors. When we multiply a vector by a number, we simply multiply each entry by this number. We add vectors by adding individual coordinates and creating a new vector from the outcomes. It is exactly the same with matrix rows. When we work with rows, we can see it as a paraller process where certain thing is done in all columns simultaneously.

There are three **basic row operations**:

- We can exchange two rows;
- we can multiply or divide a row by a non-zero number;
- we can subtract a multiple of one row from another row.

Note that the multiplicative constant in the third operation can be negative as well, so we can also think of adding a multiple of one row to another.

There are also corresponding column operations, but these are not as popular, cause troubles in many applications and we will not work with them here.

The key stage in the elimination process is to take a given matrix and using row operations change it into another matrix where the first row and the first column (or more) are according to our expectations.

22a.1 Basic stage in elimination.

If you are new to this, it may be a good idea to read the following general description without worrying too much about details, proceed to the example and then return to this general description again.

Consider an $n \times m$ matrix $(a_{i,j})$. The key location for elimination is the upper left corner, we will call it the "pivot position". The number there is called a **pivot**, and we will assume for now that it is not zero. Thus it will be the beginning of a step in the resulting reduced matrix, and we want to create zeros under it using row operations. The other entries in those rows will also change, but we do not have any plans for them, they do not play any role in our decision making.

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{pmatrix} \mapsto \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ 0 & b_{2,2} & \cdots & b_{2,m} \\ \vdots & \vdots & & \vdots \\ 0 & b_{n,2} & \cdots & b_{n,m} \end{pmatrix}.$$

How do we achieve this? Consider some row i for $i > 1$. The first entry is $a_{i,1}$, and we want to change it into zero using row operations. We are not allowed to simply multiply a row by zero, so instead we use the pivot row and subtract it from row i so many times that $a_{i,1}$ disappears. How many times? We use the multiplicative constant $l_{i,1} = \frac{a_{i,1}}{a_{1,1}}$. Then the new number in row i and the first column will be

$$a_{i,1} - l_{i,1}a_{1,1} = a_{i,1} - \frac{a_{i,1}}{a_{1,1}}a_{1,1} = 0,$$

exactly as needed. But the same operation is also performed simultaneously in all the other columns, so the new entries in row i will now be

$$b_{i,j} = a_{i,j} - l_{i,1}a_{1,j}.$$

Not that the multiplication of the pivot row by $l_{i,1}$ is an inside part of this row subtracting operation, a preparatory step done somewhere on the side, it does not change the pivot row itself. The pivot row never changes while it is a pivot row.

When we apply this to all rows $i = 2, \ldots, n$, we arrive at the form outlined above. Now the first column and the first row are in their final form, so we will disregard them (from our attention)

and only focus on the remaining matrix

$$\begin{pmatrix} b_{2,2} & \cdots & b_{2,m} \\ \vdots & & \vdots \\ b_{n,2} & \cdots & b_{n,m} \end{pmatrix}.$$

Note that $b_{2,2}$ is one down and one to the right from the upper-left corner $a_{1,1}$, so it is an ideal starting point for another step. Assuming that it is not zero, we can proclaim it to be our new pivot, and simply apply the reduction procedure outlined above to this new matrix.

If we continue like this, eventually the process meets its natural end and we obtain a reduced matrix.

Example 22a.a: Consider the matrix

$$\begin{pmatrix} 2 & -2 & -6 & 2 \\ 1 & 3 & 0 & 1 \\ 2 & -8 & -9 & 3 \end{pmatrix}.$$

In the upper right corner we see a non-zero number, that will be our pivot.

To obtain zero below it in the second row, we will multiply the first row by $l_{2,1} = \frac{1}{2}$ and subtract. This operation is applied to all columns simultaneously, so we can imagine that the first row after multiplication by $\frac{1}{2}$ is $(1, -1, -3, 1)$, we then subtract it from the second row. The fact that we applied a row operation to a matrix is often indicated by a tilde.

$$\begin{pmatrix} 2 & -2 & -6 & 2 \\ 1 & 3 & 0 & 1 \\ 2 & -8 & -9 & 3 \end{pmatrix} \sim \begin{pmatrix} 2 & -2 & -6 & 2 \\ 1-1 & 3-(-1) & 0-(-3) & 1-1 \\ 2 & -8 & -9 & 3 \end{pmatrix} = \begin{pmatrix} 2 & -2 & -6 & 2 \\ 0 & 4 & 3 & 0 \\ 2 & -8 & -9 & 3 \end{pmatrix}.$$

As we mentioned in the general outline, when we multiplied the first row by $\frac{1}{2}$, we did it on the side to help us with the subtracting operation, we did not really change the first row in the matrix.

Similarly we subtract from the third row the first row multiplied by $l_{3,1} = \frac{2}{2} = 1$, that is, we just subtract the first row from the third.

$$\begin{pmatrix} 2 & -2 & -6 & 2 \\ 0 & 4 & 3 & 0 \\ 2 & -8 & -9 & 3 \end{pmatrix} \sim \begin{pmatrix} 2 & -2 & -6 & 2 \\ 0 & 4 & 3 & 0 \\ 0 & -6 & -3 & 1 \end{pmatrix}.$$

The first column and row are now finished, we will ignore them and focus on the 2×3 submatrix in the lower right corner. The number in its upper left corner (four) is not zero, so it will serve as a pivot. We want to create a zero under it, so we subtract from the third row the second row multiplied by $l_{3,2} = \frac{-6}{4} = -\frac{3}{2}$. This in effect means that we should multiply the second row by $\frac{3}{2}$ and then add it to the third row. We obtain

$$\begin{pmatrix} 2 & -2 & -6 & 2 \\ 0 & 4 & 3 & 0 \\ 0 & -6 & -3 & 1 \end{pmatrix} \sim \begin{pmatrix} 2 & -2 & -6 & 2 \\ 0 & 4 & 3 & 0 \\ 0 & 0 & \frac{3}{2} & 1 \end{pmatrix}.$$

Now the second column and second row are done, we ignore them and move our attention to the 1×2 sub matrix in the lower right corner. This is already in the right shape (the non-zero pivot $\frac{3}{2}$ does not have any non-zero numbers under it), so we conclude that this elimination is done.
△

We just saw a typical elimination process. Effectively, we deal with progressively smaller matrices, shifting our attention towards the lower right corner of the matrix. This procedure has one weak point, namely the pivot. What if at a certain stage we look at the pivot position and the number there is zero?

We scan the numbers below, and with a bit of luck we will find a non-zero number there. Then we simply switch two rows so that this non-zero number appears in the pivot position, and we are set to go. This is called "pivoting" and we will talk more about it later. We will also see soon that

in numerical analysis we usually deal with special matrices for which there always is some non-zero candidate for a pivot.

However, in general it could happen that at some stage we restrict our attention to some submatrix and all numbers in its first column are zero. Then the response is very simple: We simply proclaim this column of zeros to be already in the finished form and disregard it, moving our attention to a matrix that is one column less wide, but we keep all the rows as before. By the way, the fact that this happens actually tells us something very important about this matrix, for instance if we were solving a system of linear equations, then this would tell us that this system does not have a unique solution.

Example 22a.b: Consider the matrix

$$\begin{pmatrix} 2 & -2 & -6 & 2 \\ 1 & -1 & -3 & 8 \\ 2 & -2 & -8 & 3 \end{pmatrix}.$$

In the upper right corner we see a non-zero number, that will be our pivot. Actually, the first column is just like in the previous example, so we do exactly the same row operations: We subtract the first row $l_{2,1} = \frac{1}{2}$ times from the second row and $l_{3,1} = \frac{2}{2} = 1$ times from the third row. However, numbers in the other columns did change, so we do have to recalculate the new matrix.

$$\begin{pmatrix} 2 & -2 & -6 & 2 \\ 1 & -1 & -3 & 8 \\ 2 & -2 & -8 & 3 \end{pmatrix} \sim \begin{pmatrix} 2 & -2 & -6 & 2 \\ 0 & 0 & 0 & 7 \\ 0 & 0 & -2 & 1 \end{pmatrix}.$$

The first column and row are now finished, we will ignore them and focus on the 2×3 submatrix in the lower right corner

$$\begin{pmatrix} 0 & 0 & 7 \\ 0 & -2 & 1 \end{pmatrix}.$$

There are only zeros in the first column, so we consider it done and shift our attention to the 2×2 submatrix on the right.

$$\begin{pmatrix} 2 & -2 & -6 & 2 \\ 0 & 0 & 0 & 7 \\ 0 & 0 & -2 & 1 \end{pmatrix}$$

We see a zero in the pivot place, but there is a non-zero number below it, so we switch the second and the third rows to move it to the right place.

$$\begin{pmatrix} 2 & -2 & -6 & 2 \\ 0 & 0 & 0 & 7 \\ 0 & 0 & -2 & 1 \end{pmatrix} \sim \begin{pmatrix} 2 & -2 & -6 & 2 \\ 0 & 0 & -2 & 1 \\ 0 & 0 & 0 & 7 \end{pmatrix}$$

Now we have a non-zero pivot and we can start the next stage of elimination. Since there are only zeros under the pivot, we are actually done. We disregard the third column and the second row and pass to the 1×1 submatrix in the lower right corner. This minimatrix is already in its reduced form, so we stop elimination.

Note that pivots now do not follow along the diagonal, which is something that may be troublesome in some situations.

△

This is the basic elimination as performed by millions of students around the world. In fact, we are not really interested in this general form. As we will see below, we will be focusing on a special version that is in some way simpler, but on the other hand it incorporates some aspects needed in numerical analysis. Still, for the sake of completeness we will state the general algorithm. Since pivots need not follow the diagonal in general, we will need two pointers to show its position, a row-index and a column index.

Algorithm 22a.2.
⟨Gaussian elimination for reducing matrix to row-echelon form⟩
Given: matrix $A = (a_{i,j})_{i,j=1}^{n,m}$ of real numbers.

0. Set $k = 1$, $K = 1$.

1. If $a_{i,K} = 0$ for all $i = k, \dots, n$, then go to step **3**.
Otherwise: If $a_{k,K} = 0$, then choose any i such that $a_{i,K} \neq 0$ and then exchange rows k and i.
Continue with step **2**.

2. For $i = k+1, \dots, n$ do the following:
If $a_{i,K} \neq 0$, let $l_{i,k} = \frac{a_{i,K}}{a_{k,K}}$, set $a_{i,K} = 0$ and for $j > K$ replace $a_{i,j}$ with $a_{i,j} - l_{i,k}a_{k,j}$.
If $k < n$, increase k by one and go to step **3**.
Otherwise the algorithm stops.

3. If $K < m$, then increase K by one and go back to step **1**.
Otherwise the algorithm stops.
The **output** is the matrix $(a_{i,j})_{i,j=1}^{n,m}$.
△

When applied to (extended) regular matrices, this algorithm can be simplified. There always is a suitable candidate for a pivot, so no column skipping is necessary. Consequently, pivots lie on the diagonal and just one pointer for the pivot row and pivot column is needed. Moreover, extended regular matrices are never "tall" (the number of columns is never smaller than the number of rows), which simplifies the stopping condition. We simply go row by row until we reach the last one.

22b. Elimination: error and computationalcomplexity

Having got familiar with the Gaussian elimination, we can introduce it as a new numerical method and treat it in the usual way.

First we note that this method is absolutely reliable. There is never a situation that elimination could not handle, and we always arrive at a reduced matrix.

Which brings us to the main concern for numerical methods: the **error of the method**. Here the answer is rather unusual: There is none Recall that when investigating the error of some method, we always assume that all calculations are done precisely, and the error then arises from the fact that numerical procedures usually calculate something else than the actual desired object. For instance, numerical integration does not calculate the integral itself, but the area of some other object. Similarly, a root-finding procedure does not calculate the root, but some approximation. In contract, the elimination aims to produce a reduced matrix, and it does just that.

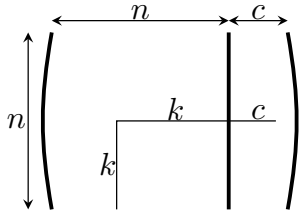
This is a rare occasion in numerical analysis. Gaussian elimination is sometimes called a "finite method", because it reaches its target in a finite number of steps, unlike methods for numerical integration or root finding that are also capable of producing a precise answer, but only if we let them make infinitely many steps. So it seems that the section on error of the method, which is usually the most extensive part, is very short this time.

To make up for it, we will ask a brand new question: We just noted that elimination finishes its work after finitely manys teps. How much work must be done before that? As we will see, this is a very important question, because it is related to the practical question fo how long we should wait for the answer.

We are in fact dabbling in another field, namely the algorithm theory, where this question is labeled **computational complexity**. The problem of how long an algorithm runs can get rather complex. For starters, are we interested in the worst case scenario, or some sort of average runtime? To make things more interesting, people even worry about the fact that computer addition and subtraction takes markedly longer than multiplication, and division is even worse. Moreover, practical implementation can sometimes exchange speed for extra storage requirements, so people also worry about memory complexity of algorithms.

However, this is not that kind of book, so we will keep things simple and ask the following question: Given an extended regular $n \times (n+c)$ matrix, how many basic algebraic operations do we have to perform before the matrix gets reduced by elimination, assuming that we never get lucky like suddenly realizing that we already have zeros where we need them?

Our setup is simple: We have an $n \times n$ regular matrix that was extended by c columns, with $c = 0$ also possible. Since elimination goes by stages, we start by exploring a specific stage. Since the main matrix is regular, pivots in stages go regularly along the diagonal, so in general we face $k \times (k+c)$ matrices, where the left $k \times k$ part comes from the original square matrix and on the right we see last rows of the added c columns. We will now explore one such stage.



Since row exchanges are not considered operations, we can assume that we did it if necessary, so there is a non-zero pivot ready for us. We will need to create zeros in $k-1$ rows below the pivot row, and for each row we follow the same procedure. Consequently, it is enough to investigate the number of operations for one row operation, and then multiply this by $k-1$.

When we want to change the leading term $a_{i,k}$ of a certain row i into zero, we start by determining the factor $l_{i,k} = \frac{a_{i,k}}{a_{k,k}}$, so it costs us one division. Then we simply write zero instead of $a_{i,k}$, this costs us nothing. However, then we have to go through the remaining column, there is $k-1+c$ of them, and each time we replace $a_{i,j}$ with $a_{i,j} - l_{i,k}a_{k,j}$. This requires two operations, multiplication and subtraction. The total for one row operation therefore is

$$1 + (k-1+c) \cdot 2.$$

The whole stage costs

$$(k-1)(1+2(k-1+c)) = (k-1)(2k-1+2c) = 2k^2 - (3-2c)k + (1-2c)$$

operations.

To reduce a matrix with n rows, we have to apply this stage to matrices of all sizes, starting with $k = n$ and ending with $k = 1$. Then we do no work, which the formula above confirms, so this seems to make sense. The total number of operations is therefore

$$\begin{aligned}\sum_{k=1}^{n} 2k^2 - (3-2c)k + (1-2c) &= 2\sum_{k=1}^{n} k^2 - (3-2c)\sum_{k=1}^{n} k + (1-2c)\sum_{k=1}^{n} 1 \\ &= 2 \cdot \tfrac{1}{6}n(n+1)(2n+1) - (3-2c) \cdot \tfrac{1}{2}n(n+1) + (1-2c) \cdot n \\ &= \left(\tfrac{2}{3}n^3 - \tfrac{1}{2}n^2 - \tfrac{1}{6}n\right) + c(n^2-n).\end{aligned}$$

Remarkably, this expression always produces a natural number for $n \in \mathbb{N}$ and $c \in \mathbb{N}_0$.

We are actually interested only in the asymptotic growth of this expression as n goes to infinity. Then we can ignore unimportant terms, for instance right at the start we could argue that $k-1 \sim k$, or in the sumation formulas we could argue that $n(n+1)(2n+1) \sim 2n^3$ and $n(n+1) \sim n^2$. This would definitely make our work easier, but we could not simplify totally, because there is the problem of c. We do not know how large c is relative to n, so we cannot ignore it safely. We therefore prefered to make a precise calculation and now we can focus on its behaviour. We cannot really make any comparison between the first and the second term due to the uncertainity about c, but we can simplify in each of the brackets. We get

$$\left(\tfrac{2}{3}n^3 - \tfrac{1}{2}n^2 - \tfrac{1}{6}n\right) + c(n^2-n) \approx \tfrac{2}{3}n^3 + cn^2.$$

Actually, we have just proved the following statement.

Theorem 22b.1.
Gaussian elimination applied to an $n \times (n+c)$ matrix requires at most $\left(\tfrac{2}{3}n^3 - \tfrac{1}{2}n^2 - \tfrac{1}{6}n\right) + c(n^2-n)$ operations.
For n large this is asymptotically $\tfrac{2}{3}n^3 + cn^2$.

In numerical analysis we usually use floating point numbers, then these would be floating point operations, or flops for short.

The actual growth depends on how c compares to n. In some applications it is comparable, as we will see in section 22f. However, our major focus will be on solving system of equations, then $c = 1$. Other applicatins use $c = 0$, which is similar in that c is bounded, not dependent on n. Then we have an important simplified result.

Fact 22b.2.
The computational complexity of GE when reducing an $n \times n$ matrix or an $n \times (n + 1)$ matrix is

$$\tfrac{2}{3}n^3 + O(n^2).$$

This is the result that people commonly remember: Gaussian elimination costs about $\frac{2}{3}n^3$ operations. Now is it a lot or reasonably little? The good news is that this is a polynomial time, which in theory of algorithm means a tractable problem. Once there are no geometric sequences or factorials, algorithm people are happy. On the other hand, people in industry and science often work with matrices where n can be of order 10^4 to 10^5, which is quite a lot, and in extreme cases matrices with n about 10^{12} an be seen, and then one really asks how long such an elimination would take.

It turns out that the cubic growth is unpleasantly large. To put this into a perspective, I type this on a fairly recent PC with Maple installed. The kind of matrices one meets in school are done immediately, square matrices with $n = 100$ take less than 10 seconds. For $n = 1000$ the runtime climbs to about 2 hours, which is quite a hike. If I wanted to reduce a matrix with $n = 10^6$, my experiments and calculations indicate that it would take about $20,000$ years. I hope I will be forgiven for not verifying this prediction. This is definitely something to take into account when thinking of elimination in actual real life applications.

Note that if we tried the modification of elimination without division (see Remark), when integer entries are kept, the computational complexity would increase to $n^3 + O(n^2)$.

22c. Numerical stability

The second traditional question when introducing new numerical method is how sensitive they are to errors in real-life computations with floating-point numbers, that is, their numerical stability. As usual, there are two sources of errors. Some of the initial entries in the matrix may already have an error, for instance if square roots appear there then computer has to round them. The second source comes from the usual rounding problem when performing operations in floating point numbers. What happens to errors during elimination?

We readily observe that numbers are used repeatedly in the algorithm. If there are any errors in the first row, they are spread in the first stage of the algorithm to the rest of the matrix. Then we use the second row which already has errors of its own plus the ones caused by the first row and spread it into the rows below. And this is done again and again, at the end the last row is the proud recipient of all errors from the previous rows, possibly enlarged by all the operations that went into making the last row. The more rows a matrix has, the more times the errors are used, so for very large matrices they may accumulate to a significant amount. So intuitively, elimination does not seem exactly safe.

For a closer look, we imagine that all entries in a matrix have relative errors vbounded by some ε. we recall that after one stage, a new entry is determined by expressions of the form

$$a_{i,j} - l_{i,k}a_{k,j}.$$

Recalling the results from chapter 3, we see that just the multiplication typically doubles the relative error. By the time we get to the last row, these multiplications cause the original relative

error to grow as high as $n\varepsilon$, where n is the number of rows. Recall that when we multiply a relative error by 10, we loose one digit of precision, so for a matrix of size $n = 1000$ we could easily loose three digits of precision through elimination.

We ignored subtractions so far, and there we actually do not have any control at all, we know that subtraction is very dangerous. To make things worse, during elimination we again repeat it many times before we are through.

However, so far we were just talking about impact of the original error and the situation is far worse. With every floating operation we likely introduce brand new errors due to round-off procedures built into the multiplication/division procedures, and there is also another source of trouble if we happen to add or subtract numbers of very different magnitudes. Last but not least, we may also run into troubles when very large numbers start appearing during elimination.

In short, elimination has a potential to get significantly influenced by numerical errors, enough to make the results worthless if we get really unlucky. Our analysis was actually very crude, but the sad outcome is valid, and it will be enough to guide us.

When we focus on one row operation, we see the pivot row multiplied by some $l_{i,k}$ and subtracted from the target row. We cannot really influence how close or how different the numbers will be during this operation, so it is fairly pointless to worry about it. But there is one thing that we can control. The errors in the pivot row get multiplied by $l_{i,k}$ and then added to errors in the target row. Thus it is to our advantage if we can make coefficients $l_{i,k}$ as small as possible during elimination, preferably smaller than one in absolute value. Now for a certain stage with pivot row k and pivot column K, these coefficients are determined as

$$l_{i,k} = \frac{a_{i,k}}{a_{k,K}}.$$

We see that these coefficients are small if the pivots are large. We will use this observation below to develop pivoting strategies.

Surprisingly enough, this simple reasoning provides us with the strongest weapon against numerical errors. Of course, it would require a significantly more sophisticated analysis to confirm that the recommendation on pivot size and consequences described below are in fact effective.

There is another recommendation some people propose. When a matrix has rows that feature significantly different numbers in magnitude (like one row featuring two-digit numbers and the other featuring millions), it makes it more prone for behaving badly with errors. Some authors recommend that before we start elimination, we should divide each row by its largest term, ending up with a matrix whose entries are all of magnitude at most 1, and with a 1 on every row. This may reduce the danger of wastly different rows, but by dividing one can introduce errors into the matrix that otherwise would not be there. So not everyone thinks this is a good idea.

In the next chapter we will introduce some tools that will offer more insight into error propagation. We will learn that some matrices are more prone to trouble than others, so success of any measures that we take depends to a large extent on what kind of matrix we face.

22d. Pivoting

Recall the basic setting. We are given a square regular $n \times n$ matrix A, and we create an extended matrix (call it C) by attaching c columns to the right of A. The matrix C thus has the dimension $n \times (n + c)$. We apply Gaussian elimination to it, and because A is regular, the last pivot position is (n, n), that is, still within the matrix A. We therefore never use the extra columns of C in our decision making and in any significant role, the entries to the right of A are just passive players in row operations.

Recall that pivoting refers to the act of choosing a certain number for a pivot. Sometimes we do **elimination without pivoting**. This means what it says, we are not allowed to exchange rows. Obviously, there are cases when we end up with zero in the place where we need a pivot, which means that elimination fails. There are problems where this makes sense, and we will see this in

the next chapter, see section . People sometimes call this version "GE".

However, in most applications we do not want elimination to fail, and we already introduced a very simple tool for that inspired by the least effort philosophy. If the first number we see in the pivot position is not zero, we keep it. Only when forced do we start looking down the column, and usually we grab the first non-zero number that comes up. For regular matrices we always succeed. This pivoting strategy does not have a name, we can call it a "forced pivoting", or perhaps minimal pivoting here.

However, this is not the only possible approach. We observed in the previous section that we would really appreciate if the coefficients $l_{i,k}$ were small, that is, if the pivots were as large as possible. This can be easily arranged.

Full pivoting works as follows. When we restrict our attention to a certain submatrix of C, we scan all entries in the columns that belong to A, that is, we ignore the extended columns, and choose the largest number (in absolute value). This we bring into the pivot position using a row swap and most likely also a column swap.

Full pivoting is the most powerful technique for dealing with numerical errors, and it can be proved that Gaussian elimination with full pivoting is a numerically stable method, which intuitively means that the numerical errors never accumulate so much that they would make answers too unreliable.

However, we pay a price. Exchanging columns causes troubles in many applications. There are ways to handle this, but people usually find that the extra work is not worth it.

There is also the problem of the time it takes. As we go through stages, we have to compare elements from materices of sizes $n \times n$, ... ,1×1. Asymptotically, this requires

$$n^2 + (n-1)^2 + \cdots + 2^2 + 1^2 \sim \frac{1}{3}n^3$$

comparisons, which is quite a lot. The time spent on one comparison is not much, but as we commented above, n^3 grows uncomfortably fast. In short, there are good reasons why people prefer a somewhat less powerul alternative.

Partial pivoting is somewhat simpler. When we want to find a candidate for a pivot, we only scan the pivot column of the submatrix we just work on. We move the largest number (in absolute value) into the pivot position by a row swap, which is a nice convenient operation and causes no trouble. Gaussian elimination with partial pivoting strategy is sometimes abbreviated "GEPP", other people call it **GEM**. It is the most popular elimination algorithm in numerical analysis.

Algorithm 22d.1.
⟨GEPP/GEM: Gaussian elimination with partial pivoting for extended regular matrices⟩
Given: a matrix $C = (c_{i,j})_{i,j=1}^{n,n+c}$ of real numbers, where $c \geq 0$.

0. Set $k = 1$.

1. If $c_{i,k} = 0$ for all $i = k, \ldots, n$, then the square submatrix $C = (c_{i,j})_{i,j=1}^{n,n}$ is not regular. The algorithm fails and stops.
Otherwise, do the "pivoting": Among rows $i = k, \ldots, n$ choose the row k' that has the largest possible value of $|c_{i,k}|$. If $k' \neq k$, exchange rows k and k'.
The (new) number $c_{k,k}$ is called a "pivot". Continue with step **2**.

2. For $i = k+1, \ldots, n$ do the following:
If $c_{i,k} \neq 0$, let $l_{i,k} = \frac{c_{i,k}}{c_{k,k}}$, set $c_{i,k} = 0$ and for $j > k$ compute $c_{i,j} = c_{i,j} - l_{i,k}c_{k,j}$.
If $k < n$, increase k by one and go back to step **1**.
Otherwise the algorithm stops.

Output: the matrix $(c_{i,j})_{i,j=1}^{n,n+c}$.

△Note: The output has the form

$$\begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,n-1} & c_{1,n} & \cdots & c_{1,n+c} \\ 0 & c_{2,2} & \cdots & c_{2,n-1} & c_{2,n} & \cdots & c_{2,n+c} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & c_{n-1,n-1} & c_{n-1,n} & \cdots & c_{n-1,n+c} \\ 0 & 0 & \cdots & 0 & c_{n,n} & \cdots & c_{n,n+c} \end{pmatrix}.$$

Note that the algorithm could have ended when $k = n - 1$, because the last iteration does not modify the matrix. But the last step does have its use, it checks whether there is a non-zero number at position $c_{n,n}$, that is, it decides whether the matrix is extended regular.

Partial pivoting is a generally accepted compromise. It obviously requires less effort than full pivoting. In fact, to decide on pivots we need to make (asymptotically)

$$(n-1) + (n-2) + \cdots + 2 + 1 \sim \frac{1}{2}n^2$$

comparisons, which would be negligible compared to the $\frac{2}{3}n^3$ cost of GEM itself, even if comparisons took much longer than algebraic operations. Moreover, we do not have to worry about column switching. Partial pivoting also benefits us in another way. When reducing a large matrix, the entries have a tendency to grow, which may cause trouble. Partial pivoting helps to slow this growth. However, this all comes as a cost, Gaussian elimination is no longer numerically stable with just partial pivoting.

The good news is that matrices for which partial pivoting does not help are extremely rare and probability of encountering one tends to zero as we increase n. However, given that we do not know in advance whether our matrix will be nice or not, this is not something that we accept easily in real life applications. In chapter 24 we will look closer that numerical stability and develop tools that will allow us to recognize troublesome matrices. For the bad ones we have to use more powerful techniques, for instance the full pivoting.

Example 22d.a: We return to example 22a.a.

$$\begin{pmatrix} 2 & -2 & -6 & 2 \\ 1 & 3 & 0 & 1 \\ 2 & -8 & -9 & 3 \end{pmatrix}.$$

We will do elimination with partial pivoting.

In the first step we scan the first column. We see that no candidate is larger than the 2 in the upper left corner, so we keep it as our pivot. he first stage thus proceeds exactly as in the first example.

$$\begin{pmatrix} 2 & -2 & -6 & 2 \\ 1 & 3 & 0 & 1 \\ 2 & -8 & -9 & 3 \end{pmatrix} \sim \begin{pmatrix} 2 & -2 & -6 & 2 \\ 0 & 4 & 3 & 0 \\ 0 & -6 & -3 & 1 \end{pmatrix}.$$

For the second stage, we scan the last two rows of the second column. We see that -6 in the thrird row is larger than the -4 in the second row, so we switch rows.

$$\begin{pmatrix} 2 & -2 & -6 & 2 \\ 0 & 4 & 3 & 0 \\ 0 & -6 & -3 & 1 \end{pmatrix} \sim \begin{pmatrix} 2 & -2 & -6 & 2 \\ 0 & -6 & -3 & 1 \\ 0 & 4 & 3 & 0 \end{pmatrix}.$$

Now we subtract from the third row the second row multiplied by $\frac{4}{-6} = -\frac{2}{3}$. We obtain

$$\begin{pmatrix} 2 & -2 & -6 & 2 \\ 0 & -6 & -3 & 1 \\ 0 & 4 & 3 & 0 \end{pmatrix} \begin{pmatrix} 2 & -2 & -6 & 2 \\ 0 & -6 & -3 & 1 \\ 0 & 0 & 1 & \frac{2}{3} \end{pmatrix}.$$

△

These are not the only possible pivoting strategies. An interesting alternative called "rook pivoting" appeared around the end of the 20th century. When looking for the right candidate for position (k,k), we start scanning the curent submatrix and take the first candidate that dominates (in absolute value) other entries in its row and column in the active submatrix. As usual, we ignore the extended columns in this decision making. When we find such a number, we move it to the pivot place using a row swap and column swap.

In the worst case, we have to always scan the whole submatrix, making this as demanding as full pivoting. However, it can be proved that on average, the amount of work is about three times that of partial pivoting, and rook pivoting does provide a numerically stable algorithm.

Remarkably, there is a pivoting strategy that goes in exactly the opposite direction. Sometimes we scan the active column and choose the smallest non-zero candidate. We do this when we start with an integer-valued matrix and planon doing all th eelimination with fractions instead of floating point numbers. Choosing the smallest (non-zero) candidate help us to avoid fractions if possible and keep denominators small. However, there is a price to pay, numbers in the eliminated matrix tend to grow faster, which can be very unpleasant especially for large matrices.

This reverse strategy of choosing the smallest candidate is particularly effective if we are to do elimination by hand.

Example 22d.b: We return to example 22a.a.

$$\begin{pmatrix} 2 & -2 & -6 & 2 \\ 1 & 3 & 0 & 1 \\ 2 & -8 & -9 & 3 \end{pmatrix}.$$

When we do elimination by hand and see 1 among the candidates, we rejoice and quickly move it into the pivot position before somebody steals it.

$$\begin{pmatrix} 1 & 3 & 0 & 1 \\ 2 & -2 & -6 & 2 \\ 2 & -8 & -9 & 3 \end{pmatrix}.$$

Now we do the first stage of elimination easily, we subtract the double of the first row from the two rows below.

$$\begin{pmatrix} 1 & 3 & 0 & 1 \\ 2 & -2 & -6 & 2 \\ 2 & -8 & -9 & 3 \end{pmatrix} \sim \begin{pmatrix} 1 & 3 & 0 & 1 \\ 0 & -8 & -6 & 0 \\ 0 & -14 & -9 & 1 \end{pmatrix}.$$

We can see that numbers in the matrix and indeed larger compared to our prevous attempt.

There are two candidates for the second stage of elimination and the smaller one is already in place. We would be tempted to divide the second row by -2, but we do not know for what purpose this elimination is made, so we will stick with the official elimination algorithm where we do not do such things.

We subtract the second row multiplied by $\frac{-8}{-14} = \frac{4}{7}$ from the second row.

$$\begin{pmatrix} 1 & 3 & 0 & 1 \\ 0 & -8 & -6 & 0 \\ 0 & -14 & -9 & 1 \end{pmatrix} \sim \begin{pmatrix} 1 & 3 & 0 & 1 \\ 0 & -8 & -6 & 0 \\ 0 & 0 & \frac{3}{2} & 1 \end{pmatrix}.$$

The fraction in the last row is actually less frinedly than the fraction in our previous attempts, which shows that the smallest candidate pivoting may actually backfire. Doing elimination by hand allows us to mix strategies judiciously and ajdust to situation.

△

This brings us to an interesting alternative that deserves its own subsection.

22d.2 Remark Euclidean pivoting:

There is a demand for "fraction-free elimination", where the given matrix has integer entries and we want this to be true throughout the elimination. This is possible, but it comes at the expense of speed. The actual increase in complexity depends on sophistication of the method, but generally we cannot do better than asymptotically $O(n^4)$ operations, which is one order worse than the standard Gaussian elimination.

Here we will show one interesting approach, and as usual it is enough to figure out one specific stage. To this end, imagine a regular matrix A, we want to reduce its first column.

We start by emplying the smallest candidate pivoting, that is, we move the smallest non-zero element from the first column into the pivot place. Next, we subtract multiples of the pivot row from the rows below it, but now we are only allowed to use integers as multiplicative coefficients. This means that we will get zeros under the pivot only if we get lucky. Generally we aim for something else: We choose those coefficients in such a way that the numbers that appear below the pivot are as close to zero as possible. We can imagine that we are repeatedly subtracting or adding the pivot row until we are (moderately) happy with the outcome.

There are two possible outcomes now. If we are really lucky, then all terms below the pivot are zero, and then this stage is complete. Otherwise there are some non-zero numbers there, and we repeat this stage, looking for candidates for a new pivot.

Note that due to the way we created the numbers below the pivot, they must all be smaller than the current pivot (in absolute value). This means that we are guaranteed to find a new pivot, smaller than the previous one. We repeat the elimination step and look below the pivot again. The numbers there must again be smaller than the pivot, and if some are not zero, they will provide us with yet a new pivot.

Since the pivot gets closer to zero in every repetition and it only attains integer value, sooner or later all candidates must be zero, that is, this stage must end.

Example 22d.c: We return to example 22d.b.

$$\begin{pmatrix} 2 & -2 & -6 & 2 \\ 1 & 3 & 0 & 1 \\ 2 & -8 & -9 & 3 \end{pmatrix}.$$

The new method calls for using the smallest non-zero candidate for pivot, which is exactly what we did in that example. We also subtracted double the pivot row, which also fits with our new approach. We therefore repeat the first stage.

$$\begin{pmatrix} 2 & -2 & -6 & 2 \\ 1 & 3 & 0 & 1 \\ 2 & -8 & -9 & 3 \end{pmatrix}. \sim \begin{pmatrix} 1 & 3 & 0 & 1 \\ 0 & -8 & -6 & 0 \\ 0 & -14 & -9 & 1 \end{pmatrix}.$$

Now we start the second stage. For the pivot we should take the smallest candidate, which we already have. Now we want to subtract the second row from the third one so many times that -14 changes into a number that is as close to zero as possible. If we subtract once, we obtain $-14-(-8)=-6$. If we subtracct twice, we obtain $-14-2\cdot(-8)=2$. This is preferable, so we subtract the second row twice from the third.

$$\begin{pmatrix} 1 & 3 & 0 & 1 \\ 0 & -8 & -6 & 0 \\ 0 & -14 & 3 & 1 \end{pmatrix} \sim \begin{pmatrix} 1 & 3 & 0 & 1 \\ 0 & -8 & -6 & 0 \\ 0 & 2 & 3 & 1 \end{pmatrix}.$$

Do we have zeros below the pivot? No, so we have to repeat this stage. Now the smallest candidate happens to be in the third row, so we switch.

$$\begin{pmatrix} 1 & 3 & 0 & 1 \\ 0 & -8 & -6 & 0 \\ 0 & 2 & 3 & 1 \end{pmatrix} \sim \begin{pmatrix} 1 & 3 & 0 & 1 \\ 0 & 2 & 3 & 1 \\ 0 & -8 & -6 & 0 \end{pmatrix}.$$

We are lucky, now we just add the second row fur times to the third one.

$$\sim \begin{pmatrix} 1 & 3 & 0 & 1 \\ 0 & 2 & 3 & 1 \\ 0 & -8 & -6 & 0 \end{pmatrix} \sim \begin{pmatrix} 1 & 3 & 0 & 1 \\ 0 & 2 & 3 & 1 \\ 0 & 0 & 6 & 4 \end{pmatrix}.$$

Our fraction-free elimination is complete.

△

We really had to repeat one stage so indeed, this type of elimination takes more work. I this particular example numbers did not grow significantly larger compared to our previous attempts at this matrix, but that was to be expected, it is just a small matrix.

Sometimes when doing this by hand, people prefer to subtract rows so that only non-negative numbers appear below the pivot. This usually adds to the number of repetitions one has to do, but with exception of the first run of every stage, in all the following ones only deal with non-negative numbers, which may be more pleaant for some.

The example above was too simple to show important aspects of this type of pivoting, so we will show another one.This time we will focus only on the pivot column.

Example 22d.d: Consider a matrix with the active column

$$\begin{pmatrix} 15 \\ 21 \\ 30 \\ 42 \end{pmatrix}.$$

We will apply the new pivoting here. We already see the smallest candidate in the first row, so we start thinking about the row subtractions. We will definitely want to subtract the pivot row twice from the third one.

If we subtract it once from the second, we get 6, and if we subtractct twice, we get -9. The first one is preferable. If we subtract the pivot row twice from the last, we get 12, while subtracting it three times we get -3, which is preferable. We have our plan.

$$\begin{pmatrix} 15 \\ 21 \\ 30 \\ 42 \end{pmatrix} \sim \begin{pmatrix} 15 \\ 6 \\ 0 \\ -3 \end{pmatrix}.$$

Not all numbers below the 15 are zero, so we repeat this stage. We move the smallest candidate up:

$$\begin{pmatrix} 15 \\ 6 \\ 0 \\ -3 \end{pmatrix} \sim \begin{pmatrix} -3 \\ 6 \\ 0 \\ 15 \end{pmatrix}.$$

We see that we are lucky and easily achieve zeros where needed.

$$\begin{pmatrix} -3 \\ 6 \\ 0 \\ 15 \end{pmatrix} \sim \begin{pmatrix} -3 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

If we preferred positive numbers to negative, we would subtract 15 just twice from the last row in the previous stage, obtaining

$$\begin{pmatrix} 15 \\ 21 \\ 30 \\ 42 \end{pmatrix} \sim \begin{pmatrix} 15 \\ 6 \\ 0 \\ 12 \end{pmatrix}.$$

Not the leading candidate for pivot is in the second row.

$$\begin{pmatrix} 15 \\ 6 \\ 0 \\ 12 \end{pmatrix} \sim \begin{pmatrix} 6 \\ 15 \\ 0 \\ 12 \end{pmatrix}.$$

We subtract the first row from the second twice, the last row is obvious.

$$\begin{pmatrix} 6 \\ 15 \\ 0 \\ 12 \end{pmatrix} \sim \begin{pmatrix} 6 \\ 3 \\ 0 \\ 0 \end{pmatrix}.$$

Finally we see a very nice pvot in the second row and complete the elimination.

$$\begin{pmatrix} 6 \\ 3 \\ 0 \\ 0 \end{pmatrix} \sim \begin{pmatrix} 3 \\ 6 \\ 0 \\ 0 \end{pmatrix} \sim \begin{pmatrix} 3 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

We see that dislike of negative numbers cost us an extra iteration of this stage.
△

The number 3 that we were left with is closely related to the numbers in the original column. In fact, it is their gratest common divisor, and the reduction that we were showing here is exactly the Euclidean algorithm for finding it. We could therefore call this modified method the Euclidean elimination, or perhaps Euclidean pivoting. When we apply it to a suitable matrix, we can use Euclidean elimination so tolve systems of linear dophantine equations.

22e. Gauss-Jordan elimination

Many students learn elimination at school in a somewhat different way. I was among them. The main idea of Gaussian elimination as we introduced it above can be expressed by saying that we create zeros under pivots. In high school I was taught to also create zeros above pivots, such a matrix is said to be in a **reduced row-echelon form**. I was also told to turn pivots into ones by dividing rows. This can be helpful in some situations. Our algorithm for Gaussian elimination can be easily modified to produce matrices of this alternative form.

Example 22e.a: For the last time we return to example 22a.a

$$\begin{pmatrix} 2 & -2 & -6 & 2 \\ 1 & 3 & 0 & 1 \\ 2 & -8 & -9 & 3 \end{pmatrix}.$$

We apply the Gauss-Jordan elimination to this matrix, namely the version with forced (minimal) pivoting to save some time.

In the upper right corner we see a non-zero number, that will be our pivot. First we create one in place of the pivot, which we achieve by dividing the first row by two.

$$\begin{pmatrix} 2 & -2 & -6 & 2 \\ 1 & 3 & 0 & 1 \\ 2 & -8 & -9 & 3 \end{pmatrix} \sim \begin{pmatrix} 1 & -1 & -3 & 1 \\ 1 & 3 & 0 & 1 \\ 2 & -8 & -9 & 3 \end{pmatrix}.$$

Now we easily see that we need to subtract the first row once from the second row and twice from the third row.

$$\begin{pmatrix} 1 & -1 & -3 & 1 \\ 1 & 3 & 0 & 1 \\ 2 & -8 & -9 & 3 \end{pmatrix} \sim \begin{pmatrix} 1 & -1 & -3 & 1 \\ 0 & 4 & 3 & 0 \\ 0 & -6 & -3 & 1 \end{pmatrix}.$$

We pass tot he second stage. We have a non-zero number 4 in place of the new pivot, so we keep it and change it into one.

$$\begin{pmatrix} 1 & -1 & -3 & 1 \\ 0 & 4 & 3 & 0 \\ 0 & -6 & -3 & 1 \end{pmatrix} \sim \begin{pmatrix} 1 & -1 & -3 & 1 \\ 0 & 1 & \frac{3}{4} & 0 \\ 0 & -6 & -3 & 1 \end{pmatrix}.$$

Now we add this row six times to the last ro, and also add it once to the first row.

$$\begin{pmatrix} 1 & -1 & -3 & 1 \\ 0 & 1 & \frac{3}{4} & 0 \\ 0 & -6 & -3 & 1 \end{pmatrix} \sim \begin{pmatrix} 1 & 0 & -\frac{9}{4} & 1 \\ 0 & 1 & \frac{3}{4} & 0 \\ 0 & 0 & \frac{3}{2} & 1 \end{pmatrix}.$$

This completes the second stage. We move on to the third pivot place and see a non-zero number $\frac{3}{2}$ there. For the last time we create a one in the pivot place.

$$\begin{pmatrix} 1 & 0 & -\frac{9}{4} & 1 \\ 0 & 1 & \frac{3}{4} & 0 \\ 0 & 0 & \frac{3}{2} & 1 \end{pmatrix} \sim \begin{pmatrix} 1 & 0 & -\frac{9}{4} & 1 \\ 0 & 1 & \frac{3}{4} & 0 \\ 0 & 0 & 1 & \frac{2}{3} \end{pmatrix}.$$

Now we add the last row to the first one $\frac{9}{4}$ times and subtract it from the second row $\frac{3}{4}$ times. We obtain the final form of the matrix.

$$\begin{pmatrix} 1 & 0 & -\frac{9}{4} & 1 \\ 0 & 1 & \frac{3}{4} & 0 \\ 0 & 0 & 1 & \frac{2}{3} \end{pmatrix} \sim \begin{pmatrix} 1 & 0 & 0 & \frac{5}{2} \\ 0 & 1 & 0 & -\frac{1}{2} \\ 0 & 0 & 1 & \frac{2}{3} \end{pmatrix}.$$

△

Algorithm 22e.1.
⟨GJE: Gauss-Jordan elimination with partial pivoting for extended regular matrices⟩
Given: a matrix $C = (c_{i,j})_{i,j=1}^{n,n+c}$ of real numbers, where $m \geq 0$.

0. Set $k = 1$.

1. If $c_{i,k} = 0$ for all $i = k, \dots, n$, then the square submatrix $C = (c_{i,j})_{i,j=1}^{n,n}$ is not regular. The algorithm fails and stops.
Otherwise, do the "pivoting": Among rows $i = k, \dots, n$ choose the row k' that has the largest possible value of $|c_{i,k}|$. If $k' \neq k$, exchange rows k and k'.
The (new) number $c_{k,k}$ is called a "pivot". Continue with step **2**.

2. For $j > k$ let $c_{k,j} = \frac{c_{k,j}}{c_{k,k}}$. Set $c_{k,k} = 1$.
For $i \neq k$ do the following:
If $c_{i,k} \neq 0$, for $j > k$ compute $c_{i,j} = c_{i,j} - l_{i,k}c_{k,j}$ and then set $c_{i,k} = 0$.
If $k < n$, increase k by one and go back to step **1**.
Otherwise the algorithm stops.
The **output** is the matrix $(c_{i,j})_{i,j=1}^{n,n+c}$.

△Note: The output has the form

$$\begin{pmatrix} 1 & 0 & \cdots & 0 & 0 & c_{1,n+1} & \cdots & c_{1,n+c} \\ 0 & 1 & \cdots & 0 & 0 & c_{2,n+1} & \cdots & c_{2,n+c} \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 & 0 & c_{n-1,n+1} & \cdots & c_{n-1,n+c} \\ 0 & 0 & \cdots & 0 & 1 & c_{n,n+1} & \cdots & c_{n,n+c} \end{pmatrix}.$$

This algorithm so similar to Gaussian elimination that we naturally expect all that we said about numerical stability to apply also to this algorithm.

We can also make a similar analysis of computational complexity. To this end we again consider some $n \times n$ matrix extended by c columns, and focus on a certain stage. Since we are also worrying about number above the pivot, we must work with all rows, so we look at an $n \times (k + c)$ matrix.

We assume that we have a suitable pivot in place and proceed with the sage. First we divide the pivot row by the pivot. This requires $(k-1)+c$ divisions. The we use the 1 that we now have at the pivot place to create zeros both above and below. This means that we have to take care of $n-1$ rows, and we do not have to prepare any multiplicative coefficients. IN one row we replace $(k-1)+C$ numbers, each time needing one multiplication and one subtraction, that is, two operations. The total for one stage is therefore

$$(k-1+c)+(n-1)\cdot 2\cdot(k-1+c)=(2n-1)(k-1+c).$$

Going through all the stages then requires the following number of operations:

$$\begin{aligned}\sum_{k=1}^{n}(2n-1)(k-1+c) &= (2n-1)\sum_{k=1}^{n}k+(2n-1)(c-1)\sum_{k=1}^{n}1\\ &= (2n-1)\tfrac{1}{2}n(n+1)+(2n-1)(c-1)n\\ &= \left(n^3-\tfrac{3}{2}n^2+\tfrac{1}{2}n\right)+(2n^2-n)c\\ &\sim n^3+2cn^2.\end{aligned}$$

Theorem 22e.2.
Gauss-Jordan elimination applied to an $n\times(n+c)$ matrix requires at most $\left(n^3-\frac{3}{2}n^2+\frac{1}{2}n\right)+(2n^2-n)c$ operations.
For large n this is assymptoticaly n^3+2cn^2.

Again, we make a special version for the popular situation when n changes but c is the same.

Fact 22e.3.
The computational complexity of GJE when reducing an $n\times n$ matrix or an $n\times(n+1)$ matrix is

$$n^3+O(n^2).$$

We see that GE and GJE have both complexity of asymptotic order $\Theta(n^3)$, but closer look reveals that GJE is slower by about half, which can be an important factor.

We used the Gauss-Jordan elimination to easily solve systems of equations and find inverse functions. It really works well when matrices are small, but we will soon see that for really large systems we have better approaches. We will dedicate a special chapet to solving systems of equations, and look at two applications in the next section.

22f. Determinant, Inverse matrix

We conclude this chapter with two applications of elimination. We start with determinant. Given a square matrix A, there are several ways to find its determinant (definition, expansion with respect to some row or column), but by far the most efficient is to use elimination. We apply the Gaussian elimination to A, and if there is a case when a pivot cannot be found, the determinant is zero. Otherwise the algorithm provides an upper triangular matrix whose determinant is found simply by mutiplying the entries on the diagonal. Unfortunately, this does not give the determinant of A. We know that the determinant of a matrix changes when we exchange rows or multiply a row by a constant. The easiest way to deal with this problem, is simply to keep track of all changes we do during the algorithm.

We can introduce a constant, say K, and set it to 1 at the start. Every time we exchange rows, we set $K:=-K$. At the end we find the determinant of the resulting matrix by multiplying numbers on the diagonal, divide this number by K and we have the determinant of A. Computational complexity of this extra work is at most n, so this can be ignored. We see that determining determinant using Gaussian notation has asymptotic complexity $\frac{2}{3}n^3$.

If we use the special version for integer matrices, we also have to do $K := K \cdot a_{k,k}$ every time we multiplied some row by the pivot as outlined above. This may result in about n^2 operation, which is again negligible.

Now we look at the problem of finding an **inverse matrix** to a given square matrix A. The procedure is well-known. We extend the matrix A by attaching a unit matrix of appropriate size and then use elimination to change A into E_n. This makes the E_n on the right into A^{-1}. Symbolically,

$$(A|E_n) \mapsto (E_n|A^{-1}).$$

If the algorithm fails to find a pivot at some stage, then A is not regular and there is no A^{-1}.

The Gauss-Jordan algorithm is just made for this. We also see that the number of columns m by which we extend is the same as n. Thus we can substitute $m = n$ in Theorem 22e.2 and arrive at the following conclusion.

Corollary 22f.1.
Determining an inverse matrix using the Gauss-Jordan elimination requires (asymptotically as $n \to \infty$) $3n^3 + O(n^2)$ operations.

The order n^3 is the best one can get, but we will see later that we can pass from 3 to a smaller number, see Remark 23a.5.

It should be noted that numerical instability discussed above applies to these applications as well, so one has to cautious.

23. Solving systems of linear equations by elimination

Consider a system of linear equations

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,m}x_m &= b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,m}x_m &= b_2 \\ \vdots \qquad &= \vdots \\ a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,m}x_m &= b_n \end{aligned}$$

We assume that the reader knows the matrix notation $A\vec{x} = \vec{b}$ of such a problem, where $A = (a_{i,j})$ is the matrix of the system, $\vec{b} \in \mathbb{R}^n$ is the vector of right hand sides and $\vec{x} \in \mathbb{R}^m$ is the unknown.

In lienar algebra we learn a general procedure for solving such a system, including the need to introduce parameters when such a system is underdetermined. However, in numerical analysis we do not really meet such systems, in scientific and enginering applications the matrix A is typically square and regular. This significantly simplifies the situation, in particular it means that such systems have unique solutions, which makes them a suitable target for numerical procedures. We will therefore follow the usual approach and assume that our matrices are square and regular.

23a. Solving systems of linear equation using Gaussian elimination

Students often learn the following procedure to solve systems of linear equations.

1. Form the extended matrix $(A|\vec{b})$.
2. Use elimination to turn it into the form $(E_n|\vec{x}_0)$.
3. The vector $\vec{x}_0$ is the solution.

The second step was called the Gauss-Jordan elimination in the previous chapter 22, so we know how much work we have to do: asymptotically it is n^3 operations.

This procedure works fine for us, but that was because we only did it by hand for small matrices. For larger ones there is a faster way.

Imagine that we use the cheaper Gaussian elimination that only costs about $\frac{2}{3}n^3$ operations, doing something like this:

$$(A|\vec{b}) \mapsto (U|\vec{d}),$$

where U is an upper triangular matrix. What next? This matrix represents the following system.

$$\begin{aligned} u_{1,1}x_1 + u_{1,2}x_2 + \cdots + u_{1,n}x_n &= d_1 \\ u_{2,2}x_2 + \cdots + u_{2,n}x_n &= d_2 \\ \vdots \\ u_{n-2,n-2}x_{n-2}u_{n-1,n-2}x_{n-1} + u_{n-2,n}x_n &= d_{n-2} \\ u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n &= d_{n-1} \\ u_{n,n}x_n &= d_n \end{aligned}$$

We can see that the last equation can be easily solved, $x_n = \frac{d_n}{u_{n,n}}$. Once we know x_n, the second last equation has only one unknown and we can solve for it, $x_{n-1} = \frac{d_{n-1} - u_{n-1,n}x_n}{u_{n-1,n-1}}$. This now allows us to solve the second last equation for x_{n-2} and so on, until we find x_1. We go from the last to the first, that's why it's called the **back substitution**.

Algorithm 23a.1.
⟨Solving an upper triangular system by back substitution⟩
Given: a system $U\vec{x} = \vec{d}$, where the matrix U is square, regular and upper triangular.

1. Find the solution $\vec{x}_0$ using the formulas

$$\begin{aligned} x_n &= \frac{d_n}{u_{n,n}} \\ x_{n-1} &= \frac{d_{n-1} - u_{n-1,n}x_n}{u_{n-1,n-1}} \\ x_{n-2} &= \frac{d_{n-2} - u_{n-2,n}x_n - u_{n-2,n-1}x_{n-1}}{u_{n-2,n-2}} \\ &\vdots \\ x_1 &= \frac{d_1 - u_{1,n}x_n - u_{1,n-1}x_{n-1} - \cdots - u_{1,2}x_2}{u_{1,1}} \end{aligned}$$

In general,

$$x_k = \frac{1}{u_{k,k}}\Big(d_k - \sum_{i=k+1}^{n} u_{k,i}x_i\Big).$$

△

How much work does it take? The general formula shows clearly how many operations it takes to evaluate x_k, there are $n-k$ multiplications, one less addition, but then we subtract the sum so we are back at $n-k$, and one division. We sum it all up,

$$\sum_{k=1}^{n}(2(n-k)+1) = -2\sum_{k=1}^{n}k + (2n+1)\sum_{k=1}^{n}1 = -2\tfrac{1}{2}n(n+1) + (2n+1)n = n^2.$$

Before we make it into a theorem, we observe that a similar procedure would work with a lower triangular matrix L. The system $L\vec{x} = \vec{d}$ then has the form

$$\begin{aligned} l_{1,1}x_1 &= d_1 \\ l_{2,1}x_1 + l_{2,2}x_2 &= d_2 \\ &\vdots \\ l_{n,1}x_1 + l_{n,2}x_2 + \cdots + l_{n,n}x_n &= d_n \end{aligned}$$

and we easily solve for x_1, then x_2 and so on, this is the **forward substitution**.

Algorithm 23a.2.
⟨Solving a lower triangular system by forward substitution⟩
Given: A system $L\vec{x} = \vec{d}$, where the matrix L is square, regular and lower triangular.

1. Find the solution $\vec{x}_0$ using the formulas

$$\begin{aligned} x_1 &= \frac{d_1}{l_{1,1}} \\ x_2 &= \frac{d_2 - l_{2,1}x_1}{l_{2,2}} \\ x_3 &= \frac{d_3 - l_{3,1}x_1 - l_{3,2}x_2}{l_{3,3}} \\ &\vdots \\ x_n &= \frac{d_n - l_{n,1}x_1 - l_{n,2}x_2 - \cdots - l_{n,n-1}x_{n-1}}{l_{n,n}} \end{aligned}$$

In general,

$$x_k = \frac{1}{l_{k,k}}\Big(d_k - \sum_{i=1}^{k-1} u_{k,i}x_i\Big).$$

△

Number of operations is calculated just like for the back substitution.

Fact 23a.3.
A system $A\vec{x} = \vec{b}$ whose matrix is upper triangular, resp. lower triangular can be solved using back substitution, resp. forward substitution with computational complexity n^2.

Since n^2 is negligible compared to the cost $\frac{2}{3}n^3$ of Gaussian elimination, for larger matrices it is better not to use the Gauss-Jordan elimination (with complexity n^3) and instead use the following algorithm.

Algorithm 23a.4.
⟨solving systems of linear equations by elimination⟩
Given: a system $A\vec{x} = \vec{b}$, where A is a square regular matrix.

1. Use Gaussian elimination to change the extended matrix $(A|\vec{b})$ into an upper triangular matrix $(U|\vec{d})$.

2. Solve the system $U\vec{x} = \vec{d}$ using back substitution.
△

Note that back and forward substitution is not just less work, but it is also very simple to implement, essentially it is just two nested loops.

Example 23a.a: We return to example 22a.a.

$$\begin{pmatrix} 2 & -2 & -6 & 2 \\ 1 & 3 & 0 & 1 \\ 2 & -8 & -9 & 3 \end{pmatrix}.$$

We can see it as the matrix of the system

$$\begin{aligned} 2x - 2y - 6z &= 2 \\ x + 3y \qquad &= 1 \\ 2x - 8y - 9z &= 3. \end{aligned}$$

In example we in fact found the solution, namely $x = \frac{5}{2}$, $y = -\frac{1}{2}$, $z = \frac{2}{3}$. ow we try the new way of solving this system.

In example 22a.a we reduced the matrix as follows:

$$\left(\begin{array}{ccc|c} 2 & -2 & -6 & 2 \\ 1 & 3 & 0 & 1 \\ 2 & -8 & -9 & 3 \end{array}\right) \sim \left(\begin{array}{ccc|c} 2 & -2 & -6 & 2 \\ 0 & 4 & 3 & 0 \\ 0 & 0 & \frac{3}{2} & 1 \end{array}\right).$$

This coresponds to the system of equations

$$\begin{aligned} 2x - 2y - 6z &= 2 \\ 4y + 3z &= 0 \\ \tfrac{3}{2}z &= 1. \end{aligned}$$

From the third equation we obtain $z = \frac{2}{3}$. Substituting this into the second we obtain $y = -\frac{3}{4}z = -\frac{1}{2}$. Finally, substituting into the first equation we get $x = 1 + y + 3z = 1 - \frac{1}{2} + 2 = \frac{5}{2}$. We obtained the right answer.

△

23a.5 Remark: Note that similar reasoning can be used to find an inverse matrix. In order to find a matrix X such that $AX = E_n$ we are actually simultaneously solving systems of linear equations $A\vec{x}^j = \vec{e}_j$, where by $\vec{x}^j$ we now denoted the jth column of X. Thus we can get an inspiration from our previous work and suggest the following procedure.

1. First we use GE to change $(A|E_n)$ into $(U|D)$, where U is upper triangular.

2. We now treat every column of matrix D as a right hand side vector of a system of linear equations and use back substitution to determine corresponding column of the matrix $X = A^{-1}$. Thus the general formula reads

$$x_{k,l} = \frac{1}{u_{k,k}}\Big(d_{k,l} - \sum_{i=k+1}^{n} u_{k,i}x_{i,l}\Big).$$

How many operations does it take? We use Theorem 22b.1 with $c = n$ and see that this requires $\frac{5}{3}n^3 + O(n^2)$ operations.

Back substitution requires n^2 operations, but that is for only one column and we need to handle n of them. Thus altogether this brings another n^3 operations for the grand total of $\frac{8}{3}n^3 + O(n^2)$. We saved $\frac{1}{3}n^3$ operations compared to the GJE way, which is quite a bit.

It should be noted that if A itself is already triangular, then the number of operations is markedly smaller. It is easy to show that an inverse matrix to an upper trinagular matrix is again upper triangular (and an analogous statement applies to lower triangular matrices). Thus we need to find only half of the terms. We can find direct formulas for them. Indeed, the desired equality $AX = E_n$ becomes for an upper triangular A the following system of equations:

$$\begin{aligned}
&a_{1,1}x_{1,1} = 1,\\
&a_{2,2}x_{2,2} = 1, \quad a_{1,1}x_{1,2} + a_{1,2}x_{2,2} = 0\\
&a_{3,3}x_{3,3} = 1, \quad a_{2,2}x_{2,3} + a_{2,3}x_{3,3} = 0, \quad a_{1,1}x_{1,3} + a_{1,2}x_{2,3} + a_{1,3}x_{4,3} = 0\\
&\vdots
\end{aligned}$$

Now we can go by rows with back substitution, obtaining

$$\begin{aligned}
&x_{1,1} = \frac{1}{a_{1,1}},\\
&x_{2,2} = \frac{1}{a_{2,2}}, \quad x_{1,2} = -\frac{1}{a_{1,1}}a_{1,2}x_{2,2}\\
&x_{3,3} = \frac{1}{a_{3,3}}, \quad x_{2,3} = -\frac{1}{a_{2,2}}a_{2,3}x_{3,3}, \quad x_{1,3} = -\frac{1}{a_{1,1}}(a_{1,2}x_{2,3} + a_{1,3}x_{3,3})\\
&\vdots\\
&x_{k,k} = \frac{1}{a_{k,k}}, \quad x_{j,k} = -\frac{1}{a_{j,j}}(a_{j,j+1}x_{j+1,k} + a_{j,j+2}x_{j+2,k} + \cdots + a_{j,k-1}u_{k-1,k} + a_{j,k}x_{k,k})\\
&\qquad\qquad \text{for } j \text{ from } k-1 \text{ to } j = 1.
\end{aligned}$$

Calculating $x_{k,k}, x_{k-1,k}, \ldots, x_{1,k}$, that is, the kth column of X, requires

$$1 + 2 + 4 + \cdots + (2k-2) = 1 + 2\sum_{j=1}^{k-1} j = 1 + (k-1)k$$

operations, the total for the whole matrix is

$$\sum_{k=1}^{n}[k^2-k+1] = \tfrac{1}{6}n(n+1)(2n+1) - \tfrac{1}{2}n(n+1) + n = \tfrac{1}{3}n^3 + \tfrac{2}{3}n.$$

We see that the complexity is $\frac{1}{3}n^3 + O(n^3)$, which is even less than GE; moreover, we have direct formulas.

△

23b. Solving systems of linear equations by LUP factorization

Sometimes there is a need to solve the same system repeatedly, with different right hand sides $\vec{b}$. If we know all of them at the start, we can extend the system matrix A by all of them and solve all those systems at the same time.

However, sometimes the right hand sides come one at a time. We definitely would not like to do $\frac{2}{3}n^3$ operations every time a new $\vec{b}$ comes up. Is there a way to do some work beforehand to save operations later? One possibility follows from simple algebra, because the solution is given by $\vec{x} = A^{-1}\vec{b}$ (assuming that A is regular, but this usually works out in applications). Determining A^{-1} costs n^3 operations, and when somebody comes up with a new $\vec{b}$, we just do $\vec{x}_0 = A^{-1}\vec{b}$, which will costs $2n^2 + O(n)$ operations. Definitely better than repeating GE.

However, there is a better approach. Note that the steps made in elimination depend only on the matrix A, so with a new right hand side we would perform exactly the same row operations, just applying them to new vectors $\vec{b}$. If we could store the information about operations, with a new right hand side we would not have to do another $\frac{2}{3}n^3$ operations, we would just apply all those row operations to numbers b_i, which is, asymptotically, about n^2 operations.

What do we really need? After we do one run $(A|\vec{b}) \mapsto (U|\vec{d})$, we obviously need to remember U so that we can do the necessary back substitution again. We also need to know which row operations changed $\vec{b}$ into $\vec{d}$ so that we can repeat this with some new $\vec{b}$, and this information is contained in the constants $l_{i,k}$ from the Gaussian elimination (see 22d.1). Indeed, the fact that $l_{i,k} = a \neq 0$ means that we are supposed to subtract the kth row a-times from the ith row, so we just do it with entries in the vector $\vec{b}$. We just need to remember in which order we have to do those operations, and GE says that we should take $k = 1, \dots, n$ in sequence and for each of them do operations corresponding to $l_{i,k}$ for $i > k$.

Note that the numbers $l_{i,k}$ do not store information about row exchanges. For now we will therefore restrict our attention only to matrices that can be reduced by elimination without any row switches; that is, at all stages we find a non-zero number at the pivot location.

Example 23b.a: We again return to example 22a.a, where we reduced the given matrix as

$$\left(\begin{array}{ccc|c} 2 & -2 & -6 & 2 \\ 1 & 3 & 0 & 1 \\ 2 & -8 & -9 & 3 \end{array}\right) \sim \left(\begin{array}{ccc|c} 2 & -2 & -6 & 2 \\ 0 & 4 & 3 & 0 \\ 0 & 0 & \frac{3}{2} & 1 \end{array}\right).$$

Scanning through that example we can see that we did not switch any rows during the elimination, so the whole process is captured in coefficients $l_{2,1} = \frac{1}{2}$, $l_{3,1} = 1$, and $l_{3,2} = -\frac{3}{2}$.

Now imagine that we are asked to solve the system

$$\begin{aligned} 2x - 2y - 6z &= -2 \\ x + 3y \qquad &= -2 \\ 2x - 8y - 9z &= 1. \end{aligned}$$

Since the left-hand sides are exactly the same as in the original system from example 22a.a, reducing

the new matrix

$$\begin{pmatrix} 2 & -2 & -6 & -2 \\ 1 & 3 & 0 & -2 \\ 2 & -8 & -9 & 1 \end{pmatrix}$$

would follow in the same way. We will therefore save time and just apply those operations to the column of right-hand sides. We have to be careful about the order. First we start with $l_{i,1}$ and go through all possible i in sequence, then we apply $l_{i,2}$. We have to subtract the first entry $\frac{1}{2}$ times from the second and once from the third. Then we have to add the new second entry $\frac{3}{2}$ times to the third.

$$\begin{pmatrix} -2 \\ -2 \\ 1 \end{pmatrix} \sim \begin{pmatrix} -2 \\ -1 \\ 1 \end{pmatrix} \sim \begin{pmatrix} -2 \\ -1 \\ 3 \end{pmatrix} \sim \begin{pmatrix} -2 \\ -1 \\ \frac{3}{2} \end{pmatrix}.$$

This is the reduced right-hand side, we recycle the reduced left-hand sides and obtain the system

$$\begin{aligned} 2x - 2y - 6z &= -2 \\ 4y + 3z &= -1 \\ \tfrac{3}{2}z &= \tfrac{3}{2}. \end{aligned}$$

The back substitution then provides us with

$$z = 1, \qquad y = \tfrac{1}{4}(-1 - z) = -1, \qquad x = -1 + y + 3z = 1.$$

This is actually the correct solution for the new system. It seems that this idea works.
△

The numbers $l_{i,k}$ just beg to be stored in a matrix. For the above example it would be

$$\begin{pmatrix} & & \\ \frac{1}{2} & & \\ 1 & -\frac{3}{2} & \end{pmatrix}.$$

Now if somebody told me to code the fact that the kth row is important and I subtract it from the ith row, I'd have called it $l_{k,i}$. Why are indices the other way around? There are good reasons for this that will be soon revealed. For now we observe that when we perform an elimination and we suspect that we would have to work with the same system again, we would want to store the resulting upper triangular matrix U and the coefficients $l_{i,k}$. By a remarkable coincidence, these fit exactly to the place where U has zeros, so we can store both information in one handy matrix. Moreover, we can store these $l_{i,k}$ on the go as we create zeros during the elimination, which is very convenient.

However, there is more. The matrix formed out of $l_{i,k}$ is obviously unfinished, we can complete it by putting 1 on the diagonal and zeros above it, creating a lower-triangular matrix. Now comes the surprise.

Theorem 23b.1.
Let A be an $n \times n$ matrix that can be reduced through elimination without row exchanges into an upper triangular matrix U, with row operations coded by $l_{i,k}$. Defining $l_{i,i} = 1$ for all $i = 1, \dots, n$ and $l_{i,k} = 0$ for all $1 \leq i < k \leq n$ we create a lower triangular matrix L.
Then $LU = A$.

Proof: (outline) We know from linear algebra that when we have an $n \times m$ matrix A and we want to subtract the k row l times from the row i, we can achieve this by multiplying A from the left by a matrix that is created by putting $-l$ into the unit matrix E_n at position (i, k):

$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & & \\ & & 1 & & \vdots \\ & -l & & & \\ 0 & & & & 1 \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,m} \\ & & & \\ \vdots & & & \vdots \\ & & & \\ a_{n,1} & a_{n,2} & \dots & a_{n,m} \end{pmatrix} = \begin{pmatrix} a_{1,1} & \overline{a_{1,2}} & \dots & a_{1,m} \\ & & & \\ \vdots & & -l\times & \vdots \\ & & & \\ a_{n,1} & a_{n,2} & \dots & a_{n,m} \end{pmatrix}$$

Such a matrix is called an elementary matrix.

If we denote the matrix corresponding to operation $l_{i,k}$ as $L_{i,k}$, we can capture the whole elimination as

$$U = L_{n,n-1} \cdot Ln, n-2 \cdot Ln-1, n-2 \cdots L_{3,2} \cdot L_{n,1} \cdots L_{3,1} \cdot L_{2,1} A.$$

Now we multiply this eqaution from the left by inverses of the elementary matrices, and obtain

$$L_{2,1}^{-1} \cdot L_{3,1}^{-1} \cdots L_{n,1}^{-1} \cdot L_{3,2}^{-1} \cdots Ln-1, n-2^{-1} \cdot Ln, n-2^{-1} \cdot L_{n,n-1}^{-1} \cdot U = A.$$

It is not hard to show that the inverse matrix of an elementary matrix can be created very easily, we just change the sign at l. This means that while the matrix $L_{i,k}$ has $-l_{i,k}$ at position (i,k), the inverse matrix $L_{i,k}^{-1}$ has $l_{i,k}$ at position.

In general, multiplyinh elementary matrices can change their contents in varuous ways. However, one can show that when we multiply them in exactly the order outlined above, then the entries $l_{i,k}$ are added to those already there, so the whole product creates exactly the matrix L described in the statement. That is, the above formula reads

$$LU = A.$$

This concludes the outline.

□

One popular sport in matrix theory is decompositions (or factorizations), when we express the given matrix as a product of special matrices. There are many useful factorizations, and the result above inspires us to make the following definition.

Definition 23b.2.
Consider a real $n \times n$ matrix A. We say that $n \times n$ matrices L, U are an **LU factorization** of A if U is an upper-triangular matrix, L is a lower-triangular matrix whose diagonal entries are all 1, and $A = LU$.

The above theorem has an immediate consequence.

Corollary 23b.3.
If Gaussian elimination without pivoting applied to A succeeds, then A admits an LU-factorization.
Moreover, the matrices L, U from Gaussian eliminations are the right ones.

Careful reading of the proof above shows that the relationship in fact goes two ways. If we have an LU-factorization $A = LU$, then $l_{i,k}$ define an elimination process that reduces A to U. Since we know that for some matrices we are forced to switch rows during elimination, it follows that there are matrices for which there is no LU-factorization. On the other hand, if a singular matrix A does have an LU-factorization, then it has infinitely many of them. However, for a regular matrix there is only one LU-factorization possible.

Example 23b.b: We return again to example 22a.a. Putting together our results we easily check that

$$\begin{pmatrix} 2 & -2 & -6 \\ 1 & 3 & 0 \\ 2 & -8 & -9 \end{pmatrix} . = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ 1 & -\frac{3}{2} & 1 \end{pmatrix} . \left.\begin{pmatrix} 2 & -2 & -6 \\ 0 & 4 & 3 \\ 0 & 0 & \frac{3}{2} \end{pmatrix}\right| .$$

△

It should be noted that some authors do not require that the diagonal entries of L should be all one. This gives them more freedom, in particular we do not have uniqueness even for regular matrices. However, this is exactly what some other people do not like. The way we defined LU-factorization here seems the more popular choice.

The fact that elimination leads to an algebraic identity is not just a curio. Recall our main motivation: We wanted to save work when solving a system of equations repeatedly with new right-hand sides. We did find a way, but so far it is a programmer's solution: We have to redo a list of operations. The LU-factorization allows us to approach this problem using mathematical notation and familiar procedures.

To this end, consider a regular matrix A for which we know its LU-factorization $A = LU$. We want to solve the system $A\vec{x} = \vec{b}$. We substitute: $(LU)\vec{x} = \vec{b}$, that is, $L(U\vec{x}) = \vec{b}$.

We can denote $\vec{y} = U\vec{x}$ and the system now reads $L\vec{y} = \vec{b}$. Since L is lower-triangular, we can solve this system using forward substitution at the cost of n^2 operations.

Now we know $\vec{y}$, and $U\vec{x} = \vec{y}$, where U is upper triangular. This means that we can determine $\vec{x}$ using back substitution, another n^2 operations. Symbolically:

$$(L|\vec{b}) \mapsto \vec{y}, \qquad (U|\vec{y}) \mapsto \vec{x}.$$

This process costs $2n^2$ operations, which is much cheaper than using the gaussian elimination again.

We will not formalize this process yet, first we will address the rpoblem that some matrices need not have an LU-factorization.

Example 23b.c: Consider $A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$. We have a nice candidate for the pivot in the first column, but without pivoting we cannot move it to the right place and the algorithm fails.

Now we try it algebraically. Is it possible to have factorization

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ l & 1 \end{pmatrix} \begin{pmatrix} a & b \\ 0 & d \end{pmatrix} ?$$

The upper left corner of A forces $a = 0$, then for the lower left corner we obtain the equation $l \cdot 0 + 0 = 1$ and obviously no choice of l can do this.

△

Note that the above matrix is "nice" as far as the usual requirements on a matrix go: It is real, regular, even symmetric. What more can one ask? And still it did not help. The question of existence for LU factorization has been studied and there is a theoretical answer.

Theorem 23b.4.
Let A be a real $n \times n$ matrix. If its rank is n and its first k leading principal minors are non-zero, then it has an LU factorization.
Let A be a real regular $n \times n$ matrix. It has an LU factorization if and only if all its leading principal minors are non-zero.

Given that evaluating those leading principals takes about n^4 operations, it is actually easier to just try elimination. An even better idea is to figure out a way to have all the benefits with just some minor modification of our situation.

Recall that when we want to switch two rows in a matrix A, we can see it a multiplication PA, where P is a permutation matrix. In fact, we obtain it easily by taking the identity matrix E_n and switch the rows in it in the way we want to switch them in A. This allows us to capture the general process of elimination.

Given a regular matrix A, we can always reduce it to an upper triangular matrix U by applying elementary matrices as we outlined above, but once in a while we insert a permutation matrix $P_{i,j}$ that switches rows i and j. Symbolically,

$$U = L \cdot L \cdots L \cdot P \cdot L \cdots L \cdot P \cdots L \cdot A.$$

This makes the situation somewhat more complicated, but it can be analyzed, and in particular one can explore what happens when we try to pull the permutation matrices out of the product. In the end we arive at the formula $LU = PA$, where L is again a lower triangular matrix based on the row operations, but in this case some moficiations have to be made to account for the permutations, it is not just coefficients $l_{i,k}$ sitting at their places.

We can interpret the resulting formula as follows: We somehow guess beforehand what row exchanges have to be made during elimination, and capture them in the permutation matrix P. Then PA is a pre-processed matrix that can be reduced by elimination without any row exchanges, hence it has its LU-factorization $PA = LU$.

Example 23b.d: Consider the matrix $A = \begin{pmatrix} 0 & 6 & 5 \\ 4 & -3 & 2 \\ 4 & -1 & 1 \end{pmatrix}$.

Gaussian elimination would start by looking at the first column and finding the need to exchange rows. We have two fours to choose from and the one in the third row dominates its row more, so that will be our new pivot. After the first stage we obtain the matrices

$$U = \begin{pmatrix} 4 & -1 & 1 \\ 0 & -2 & 1 \\ 0 & 6 & 5 \end{pmatrix}, \; L = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

We see that the second stage has no need of pivoting, and the third one is just the closing step, so we predict that just one row exchange will be enough. Namely, we want to exchange the first and the third row, which can be realized using the permutation matrix

$$P = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}.$$

Indeed,

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

Now we apply Gaussian elimination to the latter matrix.

$$\begin{pmatrix} 4 & -1 & 1 \\ 4 & -3 & 2 \\ 0 & 6 & 5 \end{pmatrix} \sim \begin{pmatrix} 4 & -1 & 1 \\ 0 & -2 & 1 \\ 0 & 6 & 5 \end{pmatrix} \sim \begin{pmatrix} 4 & -1 & 1 \\ 0 & -2 & 1 \\ 0 & 0 & 8 \end{pmatrix}.$$

We first subtracted the first row from the second, so $l_{2,1} = 1$. Then we added the second three times to the third, so $l_{3,2} = -3$. This determines L, and U is just the last matrix in elimination. We have the following.

$$U = \begin{pmatrix} 4 & -1 & 1 \\ 0 & -2 & 1 \\ 0 & 0 & 8 \end{pmatrix}, \qquad L = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & -3 & 1 \end{pmatrix}.$$

It is easy to check that

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

△

Definition 23b.5.
Consider a real $n \times n$ matrix A. We say that $n \times n$ matrices L, U and a permutation matrix P are an **LUP factorization** of A if U is an upper-triangular matrix, L is a lower-triangular matrix whose diagonal entries are all 1, and $PA = LU$.

This factorization does not fail us.

Fact 23b.6.
For every square matrix A there exists an LUP factorization.

This factorization can be again found using Gaussian elimination. However, this time we have to work more on it. We have to keep track of all row permutations, this is done by introducing a matrix P. At the beginning we set $P = E_n$ and every time we exchange some rows, we do exactly the same to matrix P. and the end it is the right matrix P for the factorization.

However, this is not all, row exchanges also require some adjusting to the matrix L that we are slowly building. We will show a simplified algorithm that is taylored for regular matrices, and we include partial pivoting. It also works for singular matrices and produces one possible LU-factorization, but the resultign matrix U is not the one that we expect to see from regular Gaussian elimination, as we do not skip columns in this algorithm (cf Remark 23b.9).

Algorithm 23b.7.
⟨LUP factorization of a matrix⟩
Given: an $n \times n$ matrix $A = (a_{i,j})_{i,j=1}^{n}$ of real numbers.

0. Let $U = A$ and $L = P = E_n$ (unit matrix). Set $k = 1$.

1. If $u_{i,k} = 0$ for all $i \geq k$, then go to step **3**.
Otherwise:
Among the rows $i = k, \ldots, n$ choose the row k' that has the largest possible value of $|u_{i,k}|$. If $k' > k$, then exchange rows k and k' in matrices U and P; in the matrix L exchange the first $k-1$ entries of rows k and k'.
Continue with step **2**.

2. For $i = k+1, \ldots, n$ do the following: Let $l_{i,k} = \frac{u_{i,k}}{u_{k,k}}$, set $u_{i,k} = 0$ and for $j > k$ compute $u_{i,j} = u_{i,j} - l_{i,k} u_{k,j}$.

3. If $k < n$, increase k by one and go back to step **1.**
Otherwise the algorithm stops.

Output: matrices P, L, U.
△

How do we use it in solving systems of equations? We have the following situation.

$$A\vec{x} = \vec{b} \iff PA\vec{x} = P\vec{b} \iff L(U\vec{x}) = P\vec{b}.$$

This suggests the following procedure.

Algorithm 23b.8.
⟨solving systems of linear equations using LUP factorization⟩
Given: a system $A\vec{x} = \vec{b}$, where A is a regular square matrix.

1. Find the LUP factorization $LU = AP$.

2. Using the forward substitution, solve the system $L\vec{y} = P\vec{b}$ for $\vec{y}$.

3. Using the back substitution, solve the system $U\vec{x} = \vec{y}$ for $\vec{x}$.
△

How much work does it take? Preparing the LUP factorization takes as many operations as GE, that is, $\frac{2}{3}n^3 + O(n^2)$. For every $\vec{b}$ we then have to do the following:

We have to multiply $P\vec{b}$. This is nominally n^2 operations, but note that P is a permutation matrix, so in fact there is no need to multiply, we just permute the entries in $\vec{b}$. If you do want to do it using multiplications, note that there is only one non-zero number in every row, so it will take only n multiplications.

Then we need to do the back and forward substitution for n^2 each.

Conclusion: Preparation takes $\frac{2}{3}n^3 + O(n^2)$ operations, thereafter every solution to a new system takes $2n^2 + O(n)$ operations.

Compared to the idea with A^{-1} we saved $\frac{1}{3}n^3$ operations.

This is the perfect opportunty to explain why LUP factorization is done for regular matrices. Its main application is exactly as we just saw, for solving systems of equations. If the original matrix A were not regular, then we could do the LUP factorization (see below), but the resulting matrix U would be singular as well. And then the back substitution would fail and the whole nice procedure we just developed would be useless. So we'd better stick with regular matrices.

Since we are using modifications of Gaussian elimination, practical problems with errors and overflows are exactly the same, so one has to be careful. Plus there is the problem of unfriendly systems that we keep refering to chapter 24.

23b.9 Remark: .

As noted above, LUP factorization is traditionally used with regular matrices. What happens if we apply it to a singular matrix?

At the beginning of each stage we start by obtaining a good pivot. With a regular matrix and pivoting allowed, getting a non-zero pivot was guaranteed. Without pivoting we either had success or a total failure. With a singular matrix, a third possibility appears: We may have a column where all $u_{i,k}$ for $i \geq k$ are zero. What next?

This is the point where we realize that aims of general Gaussian elimination and LUP factorization differ. The elimination wants to have the tightest form of the matrix available, so it shifts its attention one column to the right and keeps the working row.

On the other hand, LUP factorization cares only about having zeros under the diagonal, which is true in the case we are just discussing, so we can take it as the usual successful stage and move on, passing to the next column and also to the next row. This means that, compared to the usual elimination, we have one less row to worry about. That is exactly the algorithm we used above. Let's see how this works in practice.

Consider the matrix $\begin{pmatrix} 1 & -1 & 2 \\ 1 & -1 & 3 \\ -2 & 2 & 3 \end{pmatrix}$. The first step of Gaussian elimination leads to $U = \begin{pmatrix} 1 & -1 & 2 \\ 0 & 0 & 1 \\ 0 & 0 & 5 \end{pmatrix}$ and we store the coefficients $l_{2,1}$, $l_{3,1}$ into $L = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ -2 & 0 & 1 \end{pmatrix}$.

We pass to the second column and second row and we see that there are no candidates for our pivot. The simplified algorithm above is happy with the shape and passes to $u_{3,3}$. Since it is the last row, nothing more is done, the algorithm stops. We check that indeed

$$\begin{pmatrix} 1 & -1 & 2 \\ 1 & -1 & 3 \\ -2 & 2 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ -2 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & -1 & 2 \\ 0 & 0 & 1 \\ 0 & 0 & 5 \end{pmatrix},$$

so we found a correct factorization.

The traditional Gaussian elimination would skip the second column and move its attention to the third, but still looking at the second row. $u_{2,3} \neq 0$, so we easily do the last step, subtract the second row from the third five times (so $l_{3,2} = 5$) and obtain $U = \begin{pmatrix} 1 & -1 & 2 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$. Again, we check that

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

This incidentally shows that a singular matrix can really have more LUP factorizations.

As we noted above, LUP factorization is traditionally used for regular matrices, but it is good to know that if we encounter a singular one, we can still factorize it.

△

Just out of curiosity, this is a modification of LUP factorization that for singular matrices does the best shape, that is, U is in reduced row echelon form.

Algorithm 23b.10.
⟨LUP factorization of a matrix⟩
Given: an $n \times n$ matrix $A = (a_{i,j})_{i,j=1}^{n}$ of real numbers.

0. Let $U = A$ and $L = P = E_n$ (unit matrix). Set $k = 1$, $K = 1$.

1. If $u_{i,K} = 0$ for all $i \geq k$, then go to step **3**. Otherwise:
Among the rows $i = k, \dots, n$ with non-zero $u_{i,K}$, choose the row k' that has the largest $|u_{i,K}|$. If $k' > k$, then exchange rows k and k' in matrices U and P; in the matrix L exchange the first $k-1$ entries of rows k and k'.
Continue with step **2**.

2. For $i = k+1, \dots, n$ do the following: Let $l_{i,k} = \frac{u_{i,K}}{u_{k,K}}$, set $u_{i,K} = 0$ and for $j > K$ compute $u_{i,j} = u_{i,j} - l_{i,k}u_{k,j}$.
Increase k by one and continue with step **3**.

3. If $K < n$, increase K by one and go back to step **1.**
Otherwise the algorithm stops.

Output: matrices P, L, U.
△

23c. Checking on and improving solution

When I was a schoolboy, math teachers would try to impress on me the need to check my answers when solving equations. We now learned some procedures for solving systems of equations and given that they are expected to feature errors when used on computers, this solution checking seems like a very good idea.

How does it work? Given $A\vec{x} = \vec{b}$ we try to find the solution $\vec{x}_0$, but due to limitations of computers we find some other vector $\vec{x}_a$ instead. What should we think about it?

The natural approach is to substitute into the left hand side of the equation and compare the outcome $A\vec{x}_a$ with what should be there, that is, $\vec{b}$. The difference tells us something about the quality of our "solution".

Definition 23c.1.
Consider a system of linear equations $A\vec{x} = \vec{b}$. Let $\vec{x}_g$ be a candidate for solution. By its **residual** (or residual error) we mean the vector $\vec{r} = \vec{b} - A\vec{x}_a$.

What does the residual tell us about the quality of our solution? We are interested in the error we made, that is, in the vector $\vec{E}_x = \vec{x}_0 - \vec{x}_a$. Can we say anything about this vector? In the world of real numbers we would like to say that this error is "small", but what does this mean in the world of vectors? The obvious candidate to look at is the magnitude of this vector, but to know something about it we would have to first find some connection between this error and the information that we have, that is, the residual. There is one.

$$\vec{r} = A\vec{x}_0 - A\vec{x}_a = A(\vec{x}_0 - \vec{x}_a) = A\vec{E}_x.$$

In an ideal world we could simply determine the error by calculating $\vec{E}_x = A^{-1}\vec{r}$. However, we now know that we would most likely find the inverse matrix using elimination, which is the same process that produced errors in $\vec{x}_a$ in the first place, so we would not trust such a calculation any more that we trust $\vec{x}_a$.

So we give up the idea of finding the error precisely and ask a softer, yet important question: If wwe are lucky and the residual is small, can we then hope that also the error is small?

This is actually a pretty loaded question. The error is a vector, but that should not be such a bit problem, we all know how to figure out the size (magnitude) of a vector. However, we would need more. To see that, let's pretend for a moment that we have just numbers (one equation, one unknown). The formula above then reads $E_x = a^{-1}r$. From that we would conclude that $|E_x| = |a^{-1}| \cdot |r|$, so indeed, if the residual is small, we obtain also a bound on the size of the error.

Unfortunately, when we pass to our real problem, we encounter the formula $\vec{E}_x = A^{-1}\vec{r}$. If we try to translate this into the language of "sizes", we know what to do with vectors, but who ever heard about magnitude of a matrix? This is such a good question that it deserves its own chapter. We will explore this question and at the end arrive at some interesting answers, including the problem we just have here. So for conclusion of this discussion you will have to go to chapter .

However, there is something we can do now, so the work was not in vain. Note that the error that we made is determined by the equation $A\vec{E}_x = \vec{r}$, which is exactly the same system we just solved, just with a different right hand side. We learned how to solve systems repeatedly with less work, so we do it here, determine $\vec{E}_x$ and then, theoretically, we could recover $\vec{x}_0 = \vec{x}_a + \vec{E}_x$. Unfortunately, we made some errors in solving for $\vec{E}_x$ as well, but as we will see in chapter , this new error is very likely smaller than the original one. Thus we obtain a better approximation of the solution, and we can repeat this whole process until we are happy with the outcome.

Algorithm 23c.2.
⟨iterative improvement of solution⟩
Given: A system $A\vec{x} = \vec{b}$, where A is a square regular matrix.

0. Find a solution $\vec{x}$ of the system $A\vec{x} = \vec{b}$,

1. Determine the residual $\vec{r} = \vec{b} - A\vec{x}$. Solve the system $A\vec{E}_x = \vec{r}$.
If the error $\vec{E}_x$ is not sufficiently small, do the correction $\vec{x} := \vec{x} + \vec{E}_x$ and go back to step **1**.
△

What does it mean "sufficiently small"? Usually there is some tolerance ε given by a customer, so a good place to stop is when $\|\vec{E}_x\| < \varepsilon$. We know that this error has some error of its own, but in most cases this works reasonably well.

Another way to determine when to stop is when $\vec{E}_x$ has entries comparable in size to the inherent roundoff error of the system. It is obviously not possible to get a better precision than the precision at which the computer is working, so when our error gets close to it, there is no point in more iterations.

24. Error in matrix calculations, condition number

In this chapter we will ask what impact does a small change on input have on the outcome of matrix calculation, this will also clarify the situation about checking solutions that we left unsolved. But before we do it, we have to introduce some concepts. If we want to talk about small changes in vectors and in matrices, we first need to have a way to measure them.

24a. Matrix norms

We start with vectors. The reader surely knows how to find a magnitude of a vector using squares and square root, but this is not the only possible way to do that. Depending on applications, other approaches might be more feasible. One particular motivation to look elsewhere is the fact that the usual Euclidean norm features square root, which sometimes causes trouble.

When we want to come up with some new way to express the size of a vector, we have to make sure that this new way is practical and that it feels like a size (or distance, we know that with vectors we easily pass from one to another). Around the beginning of the 20th century mathematicians formulated the properties needed for a notion of size to be reasonable. One popular type of measuring size (and thus distance) is called a norm.

Definition 24a.1.
Let V be a vector space. A mapping $\|\!|\cdot|\!\|\colon V \mapsto \mathbb{R}$ is called a **norm** if it has the following properties:

- $\|\!|\vec{x}|\!\| \geq 0$ for all $\vec{x} \in \mathbb{R}^n$;
- $\|\!|\vec{x}|\!\| = 0$ if and only if $\vec{x} = \vec{0}$;
- $\|\!|c\vec{x}|\!\| = |c| \cdot \|\!|\vec{x}|\!\|$ for all $\vec{x} \in \mathbb{R}^n$ and $c \in \mathbb{R}$;
- $\|\!|\vec{x} + \vec{y}|\!\| \leq \|\!|\vec{x}|\!\| + \|\!|\vec{y}|\!\|$ for all $\vec{x}, \vec{y} \in \mathbb{R}^n$ (triangle inequality).

The properties seem reasonable. We definitely do not expect "size" to be negative. The second property allows us to recognize the zero vector. The third property makes sure that scaling works well, and the fourth one tells us that this notion of size cannot distort our feeling of space too much. The name comes from the interpretation that in a triangle, one side cannot be longer than the other two put together. In other words, a straight trip cannot be longer than when we take a detour through some other place.

Note that in a vector space there are two operation–addition and multiplication by a scalar–and a norm offers some way to work with both of them. The most popular norms for real vectors are these:

$$\|\vec{x}\| = \sqrt{\sum_{k=1}^{n} |x_k|^2} \qquad \text{(Euclidean norm)},$$

$$\|\vec{x}\|_\infty = \max_{k=1,\dots,n} |x_k| \qquad \text{(max norm)},$$

$$\|\vec{x}\|_1 = \sum_{k=1}^{n} |x_k| \qquad \text{(sum norm)}.$$

They also work for complex vectors. All of them somehow express how we feel about vectors. The third norm can be thought of as a "taxi driver distance". If he has to drive from one place to another in a city with a square grid of streets, then the distance taken is calculated in exactly this way.

The second norm seems natural for numerical analysis. A vector represents a collection of some values that we want to approximate. If we want to say that our approximation as a whole has a small error, then it is reasonably to interpret it as saying that all those values have this small error.

It should be noted that information can pass from one norm to another, because if we work in $\mathbb{R}^n$ for a fixed $n \in \mathbb{N}$, then the information supplied by these norms cannot differ substantially. To be precise, we have the following estimates.

Fact 24a.2.
Let $n \in \mathbb{N}$. Then for every vector $\vec{x} \in \mathbb{R}^n$ the following estimates are true:
(i) $\|\vec{c}\|_\infty \leq \|\vec{x}\|_2 \leq \|\vec{x}\|_1$,
(ii) $\|\vec{c}\|_2 \leq \sqrt{n}\|\vec{x}\|_\infty$, $\|\vec{x}\|_1 \leq \sqrt{n}\|\vec{c}\|_2$, $\|\vec{x}\|_1 \leq n\|\vec{c}\|_\infty$.

By the way, we can see the largest possible difference on the vector $(1, 1, \dots, 1)$.

This fact shows that our choice of a particular norm should not change the substance of things, therefore it is mostly a matter of convenience. There are more norms (even infinitely many), but that belongs to another part of mathematics, we here look at those that are most popular in numerical analysis.

Now we would like to invent some notion of size for matrices. We could just ask for a norm, but with matrices we have an extra operation, that is, multiplication, and we would like to have a rule for it as well.

Definition 24a.3.
Let V be a vector space of matrices. A mapping $|||\cdot|||\colon V \mapsto \mathbb{R}$ is called a **matrix norm** if it is a norm and also satisfies
• $|||AB||| \leq |||A||| \cdot |||B|||$ for all $A, B \in M_{n\times n}$.

It is useful to note what are not matrix norms. In many applications people use the spectral radius as an indication of influence of a matrix. However, it is not a matrix norm. It is interesting that it actually satisfies all the properties related to operations, like $\varrho(AB) \leq \varrho(A) \cdot \varrho(B)$, it fails only one condition, the one of zero element. One can have a matrix that is not zero but has zero spectral radius, for instance $\begin{pmatrix} 0 & 13 \\ 0 & 0 \end{pmatrix}$.

Another parameter that tells us something about a matrix is the determinant, but it fails the very first condition. This can be fixed using absolute value, but $|\det(A)|$ is still not a norm, because it also fails the zero condition. The matrix above has determinant equal to zero, yet it is not a zero matrix.

To get matrix norms we have to try something new, actually it is not completely new, inspiration from vectors is obvious. The following norms are most popular in numerical analysis.

$$\|A\|_\infty = \|A\|_R = \max_{i=1,\dots,n} \sum_{j=1}^{n} |a_{i,j}| \qquad \text{(row-sum norm)},$$

$$\|A\|_1 = \|A\|_C = \max_{j=1,\dots,n} \sum_{i=1}^{n} |a_{i,j}| \qquad \text{(column-sum norm)},$$

$$\|A\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} |a_{i,j}|^2} \qquad \text{(Frobenius norm)}.$$

Example 24a.a: Consider the $n\times n$ matrix A that has numbers 1 in the first row and 0 elsewhere. We easily find that

$$\|A\|_\infty = n, \qquad \|A\|_F = \sqrt{n}, \qquad \|A\|_1 = 1.$$

On the other hand, the matrix A^T has those 1s in the fisrt column and we see that

$$\|A^T\|_\infty = 1, \qquad \|A^T\|_F = \sqrt{n}, \qquad \|A^T\|_1 = n.$$

△

This example shows that unlike the popular norms for vectors, here we have no hierarchy, it is not true that one norm would always be bounded by another. Depending which matrix we look at, one norm can be larger than the other or the other way around (or they are equal). However, just like with norms in $\mathbb{R}^n$, these matrix norms cannot differ too much, in fact the example above shows the worst case scenario.

Fact 24a.4.
Let $n \in \mathbb{N}$. For any $n \times n$ matrix A we have the following.

$$\begin{aligned} \|A\|_1 &\leq n\,\|A\|_\infty, & \|A\|_\infty &\leq n\,\|A\|_1, \\ \|A\|_F &\leq \sqrt{n}\,\|A\|_1, & \|A\|_F &\leq \sqrt{n}\,\|A\|_\infty, \\ \|A\|_1 &\leq \sqrt{n}\,\|A\|_F, & \|A\|_\infty &\leq \sqrt{n}\,\|A\|_F. \end{aligned}$$

One thing this example shows s that the row-sum and column-sum norms are sensitive to transposition. The Frobenius norm does not care, which is obvious from the formula, the actual position of entries in a matrix are not taken into account in that sum.

There is a whole theory about matrix norms, but we do not have time for it here, we'll just need basic facts that will help us in our work. Just as an example of what can be done we show one observation.

Fact 24a.5.
For every $n \in \mathbb{N}$ and for every matrix norm $\|\!|\cdot|\!\|_M$ we have $\|\!|E_n|\!\| = 1$.

Indeed, according to the last property we have

$$\|\!|E_n|\!\| = \|\!|E_n E_n|\!\| \leq \|\!|E_n|\!\| \cdot \|\!|E_n|\!\|.$$

Sine $\|\!|E_n|\!\| \neq 0$ (see the second property), we can divide the inequality and obtain what we need.

We have norms, what do we want to do with them? Recall our motivation from chapter 23. We would like to know whether we can use the information that residual $\vec{r} = A\vec{E}_x$ is small to conclude that also the error $\vec{E}_x$ is small. We are in a position now to make the "small" precise, we choose some vector norm $\|\!|\cdot|\!\|$ and we assume that $\|\!|\vec{r}|\!\| < \varepsilon$. Can we deduce some upper bound on $\|\!|\vec{E}_x|\!\|$?

In case of real numbers we would change $|r| = |aE_x|$ into $|r| = |a| \cdot |E_x|$ and take it from there. It would be really nice if we could take some matrix norm $\|\!|\cdot|\!\|_M$ and split $\|\!|A\vec{E}_x|\!\| = \|\!|A|\!\|_M \cdot \|\!|\vec{E}_x|\!\|$, but this is not possible in general. One reason is obvious, if we just choose some norm for vectors and another for matrices, then there is no reason why they should collaborate. We will therefore have to choose smartly, but even then i would be too much to ask for equality in that formula we desire.

Definition 24a.6.
Consider a norm $\|\!|\vec{x}|\!\|$ for vectors from $\mathbb{R}^n$ and a norm $\|\!|A|\!\|_M$ for matrices from $M_{n\times n}$. We say that these norms are **compatible** if $\|\!|A\vec{x}|\!\| \leq \|\!|A|\!\|_M \cdot \|\!|\vec{x}|\!\|$ for all $A \in M_{n\times n}$ and $\vec{x} \in \mathbb{R}^n$.

Now we could go through lists of norms for vectors and matrix norms and start investigating which pairs work and which do not, but usually people prefer a different approach. It turns out that for every norm for vectors there is a matrix norm that is a perfect mate.

Theorem 24a.7.
(i) Let $|||\cdot|||$ be a vector norm. The number defined for $A \in M_{n\times n}$ by the formula

$$|||A|||_M = \sup\Big\{\frac{|||A\vec{x}|||}{|||\vec{x}|||};\ \vec{x} \in \mathbb{R}^n \setminus \{\vec{0}\}\Big\} = \sup\{|||A\vec{x}|||;\ \vec{x} \in \mathbb{R}^n \wedge |||\vec{x}||| \leq 1\}$$

determines a matrix norm compatible with $|||\cdot|||$. We call it the matrix norm **induced** by $|||\cdot|||$.
(ii) Let $|||\cdot|||_M$ be a matrix norm. The number defined for $\vec{x} \in \mathbb{R}^n$ by the formula

$$|||\vec{x}||| = \left|\left|\left| \begin{pmatrix} x_1 & 0 & \dots & 0 \\ x_2 & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ x_n & 0 & \dots & 0 \end{pmatrix} \right|\right|\right|_M$$

is a norm on $\mathbb{R}^n$ compatible with $|||\cdot|||_M$. It is called the norm induced by $|||\cdot|||_M$.

We see that also every matrix norm can find its mate, but this time the relationship is not as close. If we take the vector norm induced by $|||\cdot|||_M$ and induce a matrix norm by it, we need not arrive back at $|||\cdot|||_M$.

This is related to the fact that matrix norms that are induced by someone are in some way special. Here is one observation.

Fact 24a.8.
For every $n \in \mathbb{N}$ and for every matrix norm $|||\cdot|||_M$ induced by some norm on $\mathbb{R}^n$ we have $|||E_n||| = 1$.

This follows right from the definition.

Now what can we say about popular norms? The notation is helful, the matrix norm $\|A\|_1$ is induced by the norm $\|\vec{x}\|_1$, in particular they are compatible; similarly, the matrix norm $\|A\|_\infty$ is induced by the norm $\|\vec{x}\|_\infty$. This is one of the reasons we like them a lot, all these norms are easy to evaluate and form natural pairs. As usual there are some drawbacks, one is the sensitivity of these two matrix norms to taking transposition, another disadvantage is that they are not the best to use in theory, they do not fit very well with other notions.

What about the Frobenius norm? Since $\|E_n\|_F = \sqrt{n}$, this norm cannot be induced by any vector norm. Note that according to the theorem, the Frobenius norm induces a certain vector norm, and it is easy to se that it is the Euclidean norm $\|\vec{x}\|_2$. Unfortunately, the norm induced by $\|\vec{x}\|_2$ is not (and cannot be) the Frobenius norm. This explains why the Frobeinus norm does not work all that well in theory applications. On the other hand, this norm has many nice properties, for instance it can be calculated easily, is indifferent to taking transposes, in many situations it is more favourable (on average) then the sum norms. Moreover, it is compatible with the natural Euclidean norm for vectors, so they can be used in various estimates.

This brings us to the natural question: What matrix norm gets induced by the Euclidean norm $\|\vec{x}\|_2$? It is one that we did not meet before, we call it the **spectral norm** and it is given by the formula $\|A\|_2 = \rho(A^*A)$. This norm is perhaps the best for theory, but it is a real pain to evaluate it in practice, because we would have to find the largest eigenvalue of the matrix A^*A and there is no direct formula for it.

We thus arrived at one prominent difference between norms for vectors and matrix norms. When it comes to vectors from $\mathbb{R}^n$, there is a default norm. When one says "norm of a vector", everybody assumes that the Euclidean norm $\|\cdot\|_2$ is meant unless we specify otherwise. The Euclidean norm is easy to evaluate, works well in theory and also works reasonably well in applications (even though we sometimes prefer different norms for convenience).

In contrast, there is nothing like "the matrix norm". There is no matrix norm that would satisfy most needs, people pick specific norms for specific needs.

Now we have norms and we can do something about our errors.

24b. Error of a solution

Recall out motivating problem: We have $A\vec{E}_x = \vec{r}$ and we are asking whether we can argue that $\vec{E}_x$ is small assuming that $\vec{r}$ is small. We can do it as follows.

$$A\vec{E}_x = \vec{r} \implies \vec{E}_x = A^{-1}\vec{r} \implies |||\vec{E}_x||| \leq |||A^{-1}|||_M |||\vec{r}|||.$$

It would be natural to consider row-sum norms here, but we can use any pair of compatible norms. This is interesting, but we want to get a bit more. In numerical analysis we often prefer information about relative error, so we will try to deduce such a formula. However, before we do it we change our setup.

We have a perfect system $A\vec{x}_0 = \vec{b}_0$, where $\vec{b}_0$ is the given right hand side and $\vec{x}_0$ is the precise solution. Instead we obtained some other vector $\vec{x}_a$ and we can calculate $A\vec{x}_a = \vec{b}_a$. In this setting, the residual is $\vec{b}_0 - \vec{b}_a$. The interesting part about this new setting is that we can interpret it differently: We have two systems of equations and we are trying to see some ties between them, namely understand what is going on with relationships between $\vec{b}$'s and $\vec{x}$'s. We have the following result.

Theorem 24b.1.
Consider a vector norm $|||\cdot|||$ and a compatible matrix norm $|||\cdot|||_M$.
Assume that vectors $\vec{x}_0, \vec{x}_a$ a $\vec{b}_0, \vec{b}_a$ are related by the formulas $A\vec{x}_0 = \vec{b}_0$ and $A\vec{x} = \vec{b}$. Denote $\vec{E}_x = \vec{x}_0 - \vec{x}$ and $\vec{E}_b = \vec{b}_0 - \vec{b}$. Then we have the following estimates.

$$|||\vec{E}_b||| \leq |||A|||_M \cdot |||\vec{E}_x|||,$$
$$\frac{|||\vec{E}_x|||}{|||\vec{x}|||} \leq |||A|||_M \cdot |||A^{-1}|||_M \frac{|||\vec{E}_b|||}{|||\vec{b}|||}.$$

Proof: Subtracting the two systems we obtain

$$A(\vec{x}_0 - \vec{x}_a) = \vec{b}_0 - \vec{b}_a \implies A\vec{E}_x = \vec{E}_b,$$

hence $|||\vec{E}_b||| \leq |||A|||_M \cdot |||\vec{E}_x|||$.

We solve the first equality for $\vec{x} = A^{-1}\vec{b}$ and then estimate $|||\vec{x}||| \leq |||A^{-1}|||_M \cdot |||\vec{b}|||$. Now we put the two estimates together,

$$\frac{|||\vec{E}_x|||}{|||\vec{x}|||} \leq \frac{|||A^{-1}|||_M \cdot |||\vec{E}_b|||}{|||\vec{x}|||} \leq \frac{|||A^{-1}|||_M \cdot |||\vec{E}_b|||}{\frac{1}{|||A|||_M}|||\vec{b}|||} = |||A^{-1}|||_M \cdot |||A|||_M \frac{|||\vec{E}_b|||}{|||\vec{b}|||}.$$

Done.

□

The number that appears in the estimates is an important characterisation of a matrix.

Definition 24b.2.
For an $n \times n$ matrix A we define its **condition number** as $\mathrm{cond}_{|||\cdot|||}(A) = |||A||| \cdot |||A^{-1}|||$.

If we introduce the notation ε_x for the relative error $\frac{|||\vec{E}_x|||}{|||\vec{x}|||}$ of a vector, we can now rewrite the second inequality above as follows:

$$\varepsilon_x \leq \mathrm{cond}_{|||\cdot|||}(A)\varepsilon_b.$$

Note that the condition number depends on the matrix norm we use. Obviously we always match norms so that all bits of information that come into our estimates fit together.

We return to our first interpretaion, where $\vec{E}_b$ is actually the residual $\vec{r}$ according to our older notation. We obtain the following fact:

• Assume that solving a system $A\vec{x} = \vec{b}$ we obtain an approximate solution $\vec{x}_a$. Let $\vec{r} = \vec{b} - A\vec{x}_a$ be the residual. Then we have the following estimates for the absolute and relative error of $\vec{x}$:

$$|||\vec{E}_x \leq |||A^{-1}|||_M |||\vec{r}|||,$$
$$\varepsilon_x \leq \mathrm{cond}_{|||\cdot|||}(A) \frac{|||\vec{r}|||}{|||\vec{b}|||}.$$

Here is a corollary in practical terms.

Fact 24b.3.
Let $\vec{x}_a$ be a numerical solution to a system $A\vec{x} = \vec{b}$, let $\vec{r}$ be its residual. If $\|\vec{r}\|_\infty > 10^{-k}$, then the solution $\vec{x}_a$ has $k - \log_{10}(\mathrm{cond}_\infty(A))$ correct digits.

It is obvious that we prefer matrices with small condition number. How small can it get?

Fact 24b.4.
For every matrix norm $|||\cdot|||$ and for every matrix A we have $\mathrm{cond}_\infty(A) \geq 1$.

The proof is simple, we use one of the defining properties of matrix norms.

$$\mathrm{cond}_\infty(A) = |||A||| \cdot |||A^{-1}||| \geq |||AA^{-1}||| = |||E_n||| \geq 1.$$

That's about all concerning residual. However, the result we derived in theorem 24b.1 can tell us much more.

24c. Error propagation in systems of equations

With every numerical method we investigate its numerical stability, that is, how it reacts to errors that appear in numbers for one reason or another. This is done under the assumption that calculations are performed precisely, but for systems of equations we have finite methods, that is, when performed precisely, they provide a true solution. Thus in fact we will not be analyzing any particular method, but stability properties of equations themselves!

We already have one useful answer ready, we just look at the setup of theorem 24b.1 in a different way. We may interpret it as follows: There is a system $A\vec{x} = \vec{b}_0$ that was supposed to be solved with expected solution $\vec{x}_0$, so $A\vec{x}_0 = \vec{b}_0$. Unfortunately, the vector of right hand sides got somehow corrupted, so we solved a different system $A\vec{x}\vec{b}_a$ instead, obtaining (precise) solution $\vec{x}_a$, so $A\vec{x}_a = \vec{b}_a$ is really true. How does the error of $\vec{b}$ influence the error of $\vec{x}$?

The theorem gives an answer to this question. The larger the condition number is, the more the error may magnify. Note the conditional. The inequality provides an upper bound for the error of $\vec{x}$, and often it happens that things are much better. However, we cannot rely on that, so we prefer to work with matrices with small condition number.

Of course, if the right hand sides came with errors (for instance because we work in floating point format), then also the matrix very likely has some errors in it and instead of the real one A_0 we get some other matrix A_a (hopefully close). Solving $A_a\vec{x} = \vec{b}_a$ we obtain the precise solution $\vec{x}_a$ WE denote $E_A = A_0 - A_a$, $\vec{E}_x = \vec{x}_0 - \vec{x}_a$, $\vec{E}_b = \vec{b}_0 - \vec{b}_a$, and also introduce the notion of relative error for matrices $\varepsilon_A = \frac{|||E_A|||_M}{|||A|||_M}$. After some work get an answer.

Theorem 24c.1.
Assume that matrices A_0, A_a, vectors $\vec{b}_0, \vec{b}_a$ and solutions $\vec{x}_0, \vec{x}_a$ are related by the formulas $A_0\vec{x}_0 = \vec{b}_0$ and $A_a\vec{x}_a = \vec{b}_a$. Then

$$\varepsilon_x \leq \operatorname{cond}(A)\Big(\varepsilon_b + \varepsilon_A \cdot \frac{|||\vec{x}_a|||}{|||\vec{x}_0|||}\Big).$$

If it happened that $\vec{x}_a$ is about the same in norm as $\vec{x}_0$, we would get (roughly)

$$\varepsilon_x \leq \operatorname{cond}(A)_{\|\cdot\|}(\varepsilon_b + \varepsilon_A).$$

In other words, if $\operatorname{cond}_{\|\cdot\|}(A)$ is large, then the error on input may be blown up. We see that there is a potential for trouble and we do not have to go far to find it.

Example 24c.a: Consider the matrix $A = \begin{pmatrix} 50 & 25 \\ 51 & 25 \end{pmatrix}$ of a system with the right hand side $\vec{b} = \begin{pmatrix} 250 \\ 254 \end{pmatrix}$. We easily find the solution $x = 4$, $y = 2$.

We try the condition number:

$$\operatorname{cond}_\infty(A) = \| \begin{smallmatrix} 50 & 25 \\ 51 & 25 \end{smallmatrix} \|_\infty \cdot \| \begin{smallmatrix} -1 & 1 \\ \frac{51}{50} & 2 \end{smallmatrix} \|_\infty$$
$$= 76 \cdot \frac{101}{25} \approx 307.$$

That's quite a lot, if we are unlucky, relative errors on input are magnified up to 300 times. Let's see.

We will make a small permutation in $\vec{b}$, we take $\begin{pmatrix} 250 \\ 256 \end{pmatrix}$ instead. In the row-sum norm this represents a relative error of about 0.008, less than one percent. However, the new system has the solution $x = 6$, $y = -2$, which is totally off.

By the way, we see that after we changed $\vec{b}$, only two digits in its entries it can be trusted. According to Fact 24b.3, we should trust two less digits of our result, which leaves nothing, exactly what we got here.

Now we try a small change in the matrix, with relative error under a percent again. The system $\left(\begin{array}{cc|c} 50 & 25 & 250 \\ 50.5 & 25 & 254 \end{array}\right)$ has the solution $x = 8$, $y = -6$, this is again very far from the real solution.
△

Our example shows that a bad error estimate can easily happen, our matrix wasn't really monstrous. Note that the way that we derived the solution did not matter at all. These values for x and y are not given by some procedure, but by algebraic equations and they are determined uniquely. Thus the huge growth in error is not caused by some method but it is encoded into the nature of the problem itself.

It follows that there are problems (systems of linear equations) that do not cause trouble by its nature (for instance with small condition number) and systems that are dangerous by themselves. There is a traditional terminology used is such context. We say that a matrix is **well-conditioned** if it behaves well with respect to error propagation. One way to recognize a well-conditioned matrix is to show that its condition number is small. It is not quite clear what this means, the smallest possible condition number is one, so let's say that a well-conditioned matrix should not be significantly worse. Please don't ask us about specific border value.

Matrices that cause troubles (those with huge condition numbers are hot candidates) are called **ill-conditioned**. The matrix in the above example is obviously ill-conditioned. This terminology carries over to other types of problems, for instance we may say that a system of equations is well or ill-conditioned, but also a differential equation may be well or ill-conditioned and so on.

To appreciate the difference we look at a simple example.

Example 24c.b: Consider a system of two linear equations. Each equation represent a line in $\mathbb{R}^2$ and the solution of the system is the intersection of these two lines. On the right we see two such systems (full lines), they have the same right hand sides. In both equations we change the right hand side in the first equation by the same amount (dashed line) Note that howe the new solution compare to the original ones. You can see that in the first system, the change in the solution is much smaller than in the second one.

We see where the problem is: In the second system, the directions of these lines are almost the same. When things become fuzzy (small errors in the matrix or in the right hand side), the intersection can move around a lot. Translating this into the language of matrices, a system whose matrix has rows that are almost linearly dependent will be sensitive to changes. Note that the rows in our A in example 24c.a has lines that are almost identical.
△

The observation about rows of matrices being nearly linearly dependent is good, but unfortunately we do not have a tool (some norm or another parameter) that could recognize it. What we know is that the conditional number can forse a matrix (or system of equations) to be well conditioned if it is small. If a matrix has large condition number, then it actually may be quite well behaved, but we cannot know.

Now the bad part. To determine the condition number we first have to find the inverse A^{-1}, but that is a lot of work and if a matrix is ill-conditioned then we could not trust this inverse anyway. People found some ways to estimate the condition number, they also developed some ways that are not precise but with a bit of luck drop useful hints.

• After solving a system $A\vec{x} = \vec{b}$, try to solve it again with slightly permuted A and $\vec{b}$. If the solution changes significantly, it is a warning sign.

• Divide each row of a matrix by its maximal number, so that all terms in A are (in absolute value) at most 1. Find A^{-1}. If it has some very large entries, it is a warning sign.

Of course, if these tests work out well then it does not really mean that our matrix is well conditioned, we might have been lucky.

24d. Numerical stability of elimination

When we do elimination, we create a sequence of matrices

$$A = A_1 \mapsto A_2 \mapsto A_3 \mapsto \cdots \mapsto A_N = U.$$

Each matrix is obtained from the previous one using a row operation that can be represented by a special matrix L, which is essentially the unit matrix with appropriate $l_{i,k}$ added as one entry off the diagonal. That is, $A_{m+1} = L_m A_m$, and we are interested in what happens to errors in A_m after such step. We can use similar approach as above and deduce that the growth of the error can be controlled if the condition numbers of L_m and A_m are small.

At this point it will be useful to focus on the row-sum norm, because due to the specific form of L_m we can easily evaluate

$$\|L_m\|_\infty = \|L_m^{-1}\|_\infty = 1 + |l_{i,k}|.$$

We see that it is in our best interest to keep this as small as possible, which brings us back to benefits of the partial pivoting we already discussed in chapter 22.

Considering the condition number of A_m, it depends on the conditional number of the original matrix A and also on what we do with it. We see that we should be trying to keep condition numbers of all matrices created during elimination as small as possible. Note that pivoting itself does not change the condition number, because the norms that we are using do not care about the order of rows. However, it does influence the condition number of the resulting matrix after elimination.

It would be nice if we could say that when we do one stage of elimination and we chose the pivot wisely, then the condition number does not increase. Unfortunately, this is not true; however, pivoting in most cases does not allow the condition number to grow to much, which is also nice.

Numerical analysts studied elimination for almost a hundred years now (it really is important in applications) and there are deep results on what happens when we do elimination on computers with floating point arithmetics, but they are beyond the scope of this book. What we can take from those results is that pivoting is good, not only bevause of error propagation, they may even help alleviate roundoff errors in matrix calculations. But despite all this, we should be careful anyway because things can still go wrong.

We will illustrate some points made here in the following example.

Example 24d.a: Consider the matrix $A = \begin{pmatrix} \varepsilon & 1 \\ 1 & 1 \end{pmatrix}$, where ε is a very small number. We do not really like such an unsuitable candidate for the pivot, but we try it anyway and do the first stage of Gaussian elimination, obtaining the matrix $A_2 = \begin{pmatrix} \varepsilon & 1 \\ 0 & -\frac{1-\varepsilon}{\varepsilon} \end{pmatrix}$. How much do we like the new matrix?

We easily find that $A^{-1} = \begin{pmatrix} \frac{-1}{1-\varepsilon} & \frac{1}{1-\varepsilon} \\ \frac{1}{1-\varepsilon} & \frac{-\varepsilon}{1-\varepsilon} \end{pmatrix}$ and $A_2^{-1} = \begin{pmatrix} 1 & -\frac{\varepsilon}{1-\varepsilon} \\ 0 & \frac{\varepsilon}{1-\varepsilon} \end{pmatrix}$ and we are ready to look at condition numbers. We will show how they work for the three most popular norms.

	A	A_2
cond_∞	$\frac{4}{1-\varepsilon} \approx 4$	$\frac{1}{\varepsilon}$
cond_1	$\frac{4}{1-\varepsilon} \approx 4$	$\frac{1}{\varepsilon}$
cond_F	$\frac{3+\varepsilon^2}{1-\varepsilon} \approx 3$	$\sqrt{1+\varepsilon^2+\frac{(1-\varepsilon)^2}{\varepsilon^2}}\sqrt{1+\frac{2\varepsilon^2}{(1-\varepsilon)^2}} \approx \frac{1}{\varepsilon}$

We see that no matter what norm we try, row elimination made the condition number much bigger, it goes to infinity as $\varepsilon \to 0^+$.

We take the good advice and try partial pivoting, so we start our elimination with the matrix $B = \begin{pmatrix} 1 & 1 \\ \varepsilon & 1 \end{pmatrix}$ and obtain $B_2 = \begin{pmatrix} 1 & 1 \\ 0 & 1-\varepsilon \end{pmatrix}$.

Inverse matrices are $B^{-1} = \begin{pmatrix} \frac{1}{1-\varepsilon} & \frac{-1}{1-\varepsilon} \\ \frac{-\varepsilon}{1-\varepsilon} & \frac{1}{1-\varepsilon} \end{pmatrix}$ and $B_2^{-1} = \begin{pmatrix} 1 & \frac{-1}{1-\varepsilon} \\ 0 & \frac{1}{1-\varepsilon} \end{pmatrix}$, hence

	B	B_2
cond_R	$\frac{4}{1-\varepsilon} \approx 4$	$\frac{2(2-\varepsilon)}{1-\varepsilon} \approx 4$
cond_S	$\frac{4}{1-\varepsilon} \approx 4$	$\frac{2(2-\varepsilon)}{1-\varepsilon} \approx 4$
cond_F	$\frac{3+\varepsilon^2}{1-\varepsilon} \approx 3$	$\frac{2+(1-\varepsilon)^2}{1-\varepsilon} \approx 3$

As expected, B has the same condition number as A. However, the outcome of elimination couldn't have been more different, with A it grew beyond control, here the condition number actually slightly improved.

Now we look at the other claim, about alleviating roundoff errors. We will attempt to solve the system $A = \left(\begin{array}{cc|c} \varepsilon & 1 & 1+\varepsilon \\ 1 & 1 & 2 \end{array}\right)$ whose obvious solution is $x = y = 1$.

We will analyze what happens when we do operations with k digit precision and $\varepsilon < 10^{-k}$. Note that in this case the computer does not have a problem storing ε, but the number $1+\varepsilon$ is seen as 1. The computer therefore thinks that it is supposed to solve the system $\left(\begin{array}{cc|c} \varepsilon & 1 & 1 \\ 1 & 1 & 2 \end{array}\right)$

Elimination without pivoting should lead to the matrix $\left(\begin{array}{cc|c} \varepsilon & 1 & 1 \\ 0 & 1-1/\varepsilon & 2-1/\varepsilon \end{array}\right)$, but computer sees it differently. Since $\varepsilon < 10^{-k}$, we have $\frac{1}{\varepsilon} > 10^k$, so according to computer $1 - \frac{1}{\varepsilon} = \frac{1}{\varepsilon}$. This the

computer actually sees the matrix $\left(\begin{array}{cc|c} \varepsilon & 1 & 1 \\ 0 & -1/\varepsilon & -1/\varepsilon \end{array}\right)$ and using back substitution easily concludes that the solution is $x = 0$, $y = 1$, which is obviously wrong.

If we allow pivoting, computer starts with the matrix $\left(\begin{array}{cc|c} 1 & 1 & 2 \\ \varepsilon & 1 & 1 \end{array}\right)$, which should theoretically reduce to $\left(\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & 1-\varepsilon & 1-2\varepsilon \end{array}\right)$, but the computer sees $\left(\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & 1 & 1 \end{array}\right)$. Back substitution yields the correct solution. The pivoting really helped (this time).

$\triangle$

25. Solving systems of linear equations by iteration

In chapter 20 we transformed the task of solving an equation into a fixed point problem. We can do the same with systems of linear equations. Given a system $A\vec{x} = \vec{b}$, we can do

$$A\vec{x} = \vec{b} \iff A\vec{x} + \vec{x} = \vec{b} + \vec{x} \iff (A + E_n)\vec{x} - \vec{b} = \vec{x}.$$

Then we could define a mapping $\mathbb{R}^n \mapsto \mathbb{R}^n$ by the formula $\vec{x} \mapsto (A + E_n)\vec{x} - \vec{b}$ and try iteration. As a motivational example it wasn't bad, but we will look at better ways. Before we do it, we will look closer at that iterating business.

Inspired by the above formula, we will look at iteration schemes of the form $\vec{x}_{k+1} = B\vec{x} + \vec{c}$. Such a scheme produces a sequence of vectors $\{\vec{x}_k\}$ and we need to learn how to work with them. In particular, we need to know what convergence means for sequences of vectors. We we follow inspiration from sequences of real numbers, where the convergence $x_k \to x$ is equivalent to the fact that the distance between x_k and x goes to zero. We know how to do distances in vector spaces and distances are numbers for which convergence is known, so this should be easy. Similarly we work with sequences of matrices.

Definition 25.1.
Let $n \in \mathbb{N}$, let $\{\vec{x}_k\}$ be a sequence of vectors from $\mathbb{R}^n$. We say that it **converges** to $\vec{x} \in \mathbb{R}^n$ with respect to a vector norm $|||\cdot|||$ if $|||\vec{x}_k - \vec{x}||| \to 0$.
Let $n \in \mathbb{N}$, let $\{A_k\}$ be a sequence of $n \times n$ matrices. We say that it **converges** to an $n \times n$ matrix A with respect to a matrix norm $|||\cdot|||$ if $|||A_k - A||| \to 0$.

It can be easily shown that a sequence of vectors $\vec{x}_k$ converges to a vector $\vec{x}$ if and only if for every k, the kth coordinate of vectors $\vec{x}_k$ converge to the kth coordinate of $vecx$. It is also easy to show that because we have mutual comparison between norms in $\mathbb{R}^n$, the notion of convergence does not actually depend on the choice of a norm, so we can use any norm we feel like. In numerical analysis we like to use the max norm $\|\cdot\|_\infty$.

Analogous facts are also true about matrices, so from now one we just say that a sequence of vecros or matrices converges to some element without referring to the norm used to check it.

25a. Convergence of iteration scheme

If a sequence of vectors comes from the iteration scheme that we are interested in, then there is a complete answer to the problem of convergence.

Theorem 25a.1.
An iterative method $\vec{x}_{k+1} = B\vec{x}_k + \vec{c}$ converges if and only if $\varrho(B) < 1$.

Unfortunately, finding the spectral radius is not so simple, so we would prefer another criterion, perhaps not so good, but more convenient. We get help from the following theorem.

Fact 25a.2.
Consider an $n \times n$ matrix B.
Every induced matrix norm $|||\cdot|||$ satisfies $\varrho(B) \leq |||B|||$.
Conversely, for every $\varepsilon > 0$ there is an induced matrix norm $|||\cdot|||$ such that $|||B||| - \varepsilon < \rho(B)$.

The first statement is the one that we want, it allows us to force the spectral radius to be small. It is actually simple to prove. We take any eigenvalue λ of B with eigenvector $\vec{x}$ and estimate

$$|\lambda| \cdot |||\vec{x}||| = |||\lambda\vec{x}||| = |||B\vec{x}||| \leq |||B||| \cdot |||\vec{x}|||.$$

We cancel and obtain $|\lambda| \leq |||B|||$ for any eigenvalue, in particular it is true for the largest eigenvalue that gives the spectral radius.

As a corollary we obtain a practical statement.

Theorem 25a.3.
If a matrix B satisfies $|||B|||_M < 1$ for some compatible matrix norm, then the corresponding iterative method $\vec{x}_{k+1} = B\vec{x}_k + \vec{c}$ converges to $\vec{x}_f$ for arbitrary choice of $\vec{x}_0$ and we have

$$|||\vec{x}_f - \vec{x}_{k+1}||| \leq \frac{|||B|||_M}{1 - |||B|||_M} |||\vec{x}_{k+1} - \vec{x}_k|||.$$

We can obtain this result also in a different way. Recall that in chapter 20 we mentioned that the Banach contraction theorem works in many general settings, in particular it works in vector spaces with a norm. Thus one way to show convergence of our iteration scheme is to prove that the mapping $\varphi(\vec{x}) = B\vec{x} + \vec{c}$ is a contraction. Indeed, we can estimate

$$\begin{aligned} |||\varphi(\vec{x}) - \varphi(\vec{y})||| &= |||(B\vec{x} + \vec{c}) - (B\vec{y} + \vec{c})||| \\ &= |||B(\vec{x} - B\vec{y})||| \leq |||B||| \cdot |||\vec{x} - \vec{y}|||. \end{aligned}$$

The condition $q = |||B||| < 1$ shows that we indeed have a contraction.

Now we are ready.

25b. Iteration for systems of equations

We transform equations into fixed point problems by creating an expression of the form $x = \cdots$. To get an inspiration we look at a small system.

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 &= b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 &= b_2 \end{aligned}$$

We want some formulas for x_1 and x_2, why not using the two equations for it? We obtain

$$\begin{aligned} x_1 &= \frac{1}{a_{1,1}}\big(b_1 - a_{1,2}x_2\big) \\ x_2 &= \frac{1}{a_{2,2}}\big(b_2 - a_{2,1}x_1\big) \end{aligned}$$

Generalizing this idea to larger systems seems obvious. Iteration works as follows. Given a vector $\vec{x}_k$, we substitute its coordinates into the expressions on the right and obtain a new vector $\vec{x}_{k+1}$.

We will need to access coordinates of these vectors, which is a bit awkward, the ith coordinate of a vector $\vec{x}$ is usually denoted x_i, but now we use subscript to index the sequence. To avoid confusion, we will use the notation $(\vec{x}_k)_i$ for the coordinate here.

Algorithm 25b.1.
⟨JIM: Jacobi iteration method⟩
Given: a system $A\vec{x} = \vec{b}$ of linear equations with a square $n \times n$ matrix, tolerance ε, arbitrary initial vector $\vec{x}_0$.

0. Set $k = 0$.

1. For all $i = 1, \ldots, n$ compute

$$(\vec{x}_{k+1})_i = \frac{b_i}{a_{i,i}} - \frac{1}{a_{i,i}}\Big(\sum_{j=1}^{i-1} a_{i,j}(\vec{x}_k)_j + \sum_{j=i+1}^{n} a_{i,j}(\vec{x}_k)_j\Big).$$

If $\|\vec{x}_{k+1} - \vec{x}_k\|_\infty \geq \varepsilon$, increase k by one and go back to step **1**.
△

As usual, in the stopping condition we could also use condition with relative change, that is, $\frac{\|\vec{x}_{k+1}-\vec{x}_k\|_\infty}{\|\vec{x}_k\|_\infty} \geq \varepsilon$. We use the infinity norm, because it is traditional in this setting, it controls directly sizes of all coordinates. It is also wise to set a condition on maximal number of iterations.

It is obvious that the Jacobi iteration can be used only if the diagonal entries in A are nor zero. That's life. In case of trouble we may try to switch rows, but in practical applications where iterative methods are used we usually need not worry.

To learn something about convergence we need to rephrase this iteration into the form we studied above, that is, with matrices. It will be easier if we decompose A into three prominent parts, that is, we separate its diagonal, entries under the diagonal, and entries above the diagonal. Mathematically, we write

$$A = D + L + U,$$

where D is a diagonal matrix, L is lower triangular, U is upper triangular and both L and U have diagonal entries equal to zero. This decomposition obviously always exists and is unique. Note that if A has non-zero diagonal entries $d_{i,i}$, then D^{-1} exists, it is a diagonal matrix and has $\frac{1}{d_{i,i}}$ on the diagonal.

Now it is easy to check that Jacobi's iteration scheme can be written as follows:

$$\vec{x}_{k+1} = D^{-1}\vec{b} - D^{-1}(L+U)\vec{x}_k.$$

In this form we easily check that if the sequence $\vec{x}_k$ converges to a fixed point $\vec{x}_f$, then this fixed point solves the original system.

$$\vec{x}_f = -D^{-1}(L+U)\vec{x}_f + D^{-1}\vec{b} \iff D\vec{x}_f = -(L+U)\vec{x}_f + \vec{b} \iff (D+L+U)\vec{x}_f = \vec{b}.$$

Written in matrix form, the Jacobi iteration fits with the general setup discussed in the previous section, with $B_{\mathrm{JIM}} = -D^{-1}(L+U)$ and $\vec{c} = D^{-1}\vec{b}$. However, we cannot say anything about the norm of B_J in general, so the scheme converges only sometimes. We will look at it below, after we introduce our second iteration method.

The key observation for our improved method can be made in our simple example of two equations.

$$x_1 = \frac{1}{a_{1,1}}\big(b_1 - a_{1,2}x_2\big)$$
$$x_2 = \frac{1}{a_{2,2}}\big(b_2 - a_{2,1}x_1\big)$$

Jacobi says: When you have a vector $\vec{x}_k$, our it into the formulas on the right and obtain $\vec{x}_k$. Gauss with Saidel say: Isn't it a bit conservative? When we calculate x_2, we actually already know the new, improved value of x_1, so why don't we use it?

This sounds like something that could speed things up and we easily generalize this to more equations.

Algorithm 25b.2.
⟨GSM: Gauss-Seidel iteration⟩
Given: a system $A\vec{x} = \vec{b}$ of linear equations with a square $n \times n$ matrix, tolerance ε, arbitrary initial vector $\vec{x}_0$.

0. Set $k = 0$.

1. For all $i = 1, \dots, n$ compute

$$(\vec{x}_{k+1})_i = \frac{b_i}{a_{i,i}} - \frac{1}{a_{i,i}}\Big(\sum_{j=1}^{i-1} a_{i,j}(\vec{x}_{k+1})_j + \sum_{j=j+1}^{n} a_{i,j}(\vec{x}_k)_j\Big).$$

If $\|\vec{x}_{k+1} - \vec{x}_k\|_\infty \geq \varepsilon$, increase k by one and go back to step **1**.
△

Again, relative change can be used as a stopping condition and a maximal allowed number of iterations should be specified.

In order to obtain a matrix form we again decompose A as above and write

$$\vec{x}_{k+1} = -D^{-1}(L\vec{x}_{k+1} + U\vec{x}_k) + D^{-1}\vec{b}.$$

This is the formula as used in the algorithm, but in order to use the general setup we would need to have $\vec{x}_{k+1}$ all by itself on the lfet. No problem, we solve the equation. We multiply both sides by D and then sort it out.

$$\begin{aligned} D\vec{x}_{k+1} = -L\vec{x}_{k+1} - U\vec{x}_k + \vec{b} &\implies D\vec{x}_{k+1} + L\vec{x}_{k+1} = -U\vec{x}_k + \vec{b} \\ &\implies (D+L)\vec{x}_{k+1} = -U\vec{x}_k + \vec{b} \\ &\implies \vec{x}_{k+1} = -(D+L)^{-1}U\vec{x}_k + (D+L)^{-1}\vec{b}. \end{aligned}$$

Thus the matrices as in the general iteration are $B_{\mathrm{GSM}} = -(D+L)^{-1}U$ and $\vec{c} = (D+L)^{-1}\vec{b}$.

We check that if we obtain a fixed point, then we have a solution of our equation.

$$\vec{x} = -(D+L)^{-1}U\vec{x} + (D+L)^{-1}\vec{b} \iff (D+L)\vec{x} = -U\vec{x} + \vec{b} \iff A\vec{x} = \vec{b}.$$

Yes it does.

We have two iterative schemes for solving systems of linear equations and it is time to ask whether they are any good.

We know that convergence depends on the spectral radius $\varrho(B)$. Unfortunately, it takes some work to find it, so we will try a different approach and identify types of matrices for which these two iterations are known to work. But before we do it, we do mention some results. It is known that matrices coming from certain applications do have small eigenvalues because of the nature of the thing that is studies. Then iteration works well. It would seem that Gauss-Seidel iteration is better, but this is not exactly true, it can easily happen that one converges and the other does not. So the Jacobi iteration does have its uses, one interesting aspect is that it is immune to row permutations, whereas the Gauss-Seidel iteration is sensitive to it, switching a few rows (that is, changing the order of equations) can cause it do stop converging.

An interesting result is that if $\rho(B_{\mathrm{JIM}}) = 0$ or $\rho(B_{\mathrm{GSM}}) = 0$, then we should get the precise answer after a finite number of steps. However, this does not happen too often.

For some matrices it is known that GSM works better. For instance, if a matrix A is three-diagonal (non-zero numbers are allowed only on the diagonal and next to it, very useful matrix in differential equations), then $\rho(B_{\mathrm{GSM}}) = \rho(B_{\mathrm{JIM}})^2$. If these numbers are smaller than 1 then GSM is significantly faster, for $\vec{x}_k$ close to $\vec{x}_f$ it determines correct digits twice as fast compared to JIM.

Now we look at other ways to recognize "good" matrices for iteration. We introduce two notions, the first is easy to see, the second is more involved and understanding its meaning requires a deeper dip into the matrix theory. Let's just say that again, it is something useful in many applications.

Definition 25b.3.
Consider an $n \times n$ matrix A.
We say that A is **strictly diagonally dominant** if

$$|a_{i,i}| > \sum_{j \neq i} |a_{i,j}|$$

for all $i = 1, \ldots, n$.
We say that A is **positive definite** if $\vec{x}^T A\vec{x} > 0$ for all non-zero vectors $\vec{x} \in \mathbb{R}^n$.

Here is the good news.

Theorem 25b.4.
If A is strictly diagonally dominant, then both JIM and GSM converge for arbitrary choice of initial vector.
If A is symmetric and positive definite, then GSM converges for arbitrary choice of initial vector.

That is a very good news.

How does iteration compare to elimination? We know that elimination requires about $\frac{2}{3}n^3$ operations. On the other hand, we easily conclude that both the Jacobi and the Gauss-Seidel iterations require $2n^2 + O(n)$ operations for one iteration, which is very nice. However, this iteration has to be repeated, say N times before we obtain a reasonably precise answer. The total number of poperations is therefore about $2n^2N$ and we obviously do not know N.

What we do see is that iteration will be better than iteration in case that N is significantly smaller than n. For some matrices this is known to be true. If n is about a million (or even billion), then we can easily afford a hundred iterations and we still get the solution ten thousand times faster than using elimination. By the way, under a hunderd iterations to get a good approximation of a solution is fairly typical. So if iteration works, it can be very helpful.

Thing can get even better. If a matrix is sparse (most of the entries are zero) and the non-zeros are spaces regularly, we can actually code it into the formula for $(\vec{x}_k)_i$ and add only the terms that matter. The case of a three-diagonal matrix deserves another mention. Since there are only three non-zero numbers in every row, determining $(\vec{x}_k)_i$ requires six operations, so one whole iteration takes measly $6n$ operations. This is an incredible difference.

We see that we really want to have convergent iterations, and we hope for quick convergence. In chapter 20 we learned about a method that sometimes speeds up iteration, we called it relaxation and the trick was that we were looking for a suitable compromise between the iteration we want and constant iteration that had guaranteed and fast convergence. The same idea can be used with iteration for matrices, we will apply it to Gauss-Seidel iteration.

We introduce a parameter $\lambda > 0$ indicating how much we trust GSM. Let $\vec{x}^G_{k+1}$ be the vector obtained from $\vec{x}_k$ using Gauss-Seidel iteration. We decide to use instead the vector $\vec{x}_{k+1} = (1 - \lambda)\vec{x}_k + \lambda\vec{x}^G_{k+1}$. It is obvious that taking $\lambda = 1$ we get GSM. When we look what it does to individual coordinates, we arrive at the following procedure.

Algorithm 25b.5.
⟨SOR, Successive OverRelaxation method⟩
Given: a system $A\vec{x} = \vec{b}$ of linear equations with a square $n \times n$ matrix, parameter of relaxation λ, arbitrary initial vector $\vec{x}_0$.

0. Set $k = 0$.

1. For all $i = 1, \dots, n$ compute

$$(\vec{x}_{k+1})_i = (1 - \lambda)(\vec{x}_k)_i - \frac{\lambda}{a_{i,i}}\Big(\sum_{j=1}^{i-1} a_{i,j}(\vec{x}_{k+1})_j + \sum_{j=j+1}^{n} a_{i,j}(\vec{x}_k)_j\Big) + \frac{\lambda b_i}{a_{i,i}}.$$

If $\|\vec{x}_{k+1} - \vec{x}_k\|_\infty \geq \varepsilon$, increase k by one and go back to step **1**.
△

We used to translate this into matrix form, but we will do it differently this time, we start from the given equation and rearrange it using λ, just like we did in chapter 20. This will also confirm that the scheme solves the given equation. We start by rearranging the given equation in the way

we used when working our GSM above.

$$A\vec{x} = \vec{b} \iff \vec{x} = -D^{-1}(L\vec{x} + U\vec{x}) + D^{-1}\vec{b} \iff \lambda\vec{x} = -\lambda D^{-1}(L\vec{x} + U\vec{x}) + \lambda D^{-1}\vec{b}$$
$$\iff \vec{x} = (1-\lambda)\vec{x} - \lambda D^{-1}(L\vec{x} + U\vec{x}) + \lambda D^{-1}\vec{b}.$$

That's the blueprint for the scheme, now we make a proper iteration out of it.

$$\begin{aligned}\vec{x}_{k+1} &= (1-\lambda)\vec{x}_k - \lambda D^{-1}L\vec{x}_{k+1} - \lambda D^{-1}U\vec{x}_k + \lambda D^{-1}\vec{b}\\ &\implies (E_n + \lambda D^{-1}L)\vec{x}_{k+1} = (1-\lambda)E_n\vec{x}_k - \lambda D^{-1}U\vec{x}_k + \lambda D^{-1}\vec{b}\\ &\implies (D + \lambda L)\vec{x}_{k+1} = (1-\lambda)D\vec{x}_k - \lambda U\vec{x}_k + \lambda\vec{b}\\ &\implies \vec{x}_{k+1} = (D+\lambda L)^{-1}[(1-\lambda)D - \lambda U]\vec{x}_k + \lambda(D+\lambda L)^{-1}\vec{b}.\end{aligned}$$

This fits the general iteration setup with $B_\lambda = (D+\lambda L)^{-1}[(1-\lambda)D - \lambda U]$ and $\vec{c}_\lambda = \lambda(D+\lambda L)^{-1}\vec{b}$. We can see that as we send λ to zero, the diagonal becomes dominant, which is a good thing. However, one shoukd not overdo this.

How do we find a good relaxation parameter? We usually fish beween numbers 0 and 2, the following statement shows why.

Theorem 25b.6.
(Ostrovsky)
Let A be a symmetric $n \times n$ matrix with positive diagonal entries. Then $\varrho(B_\lambda) < 1$ if and only if A is positive definite and $0 < \lambda < 2$.

That's about the best help one gets, good λ must be discovered experimantally. Obviously, this is not worth the trouble when solving just one system. However, in some applications people end up solving again and again a system that changes little each time, so one can expect that its convergence will behave in analogous ways. Then it makes sense to try a slightly different λ each time and see what happens.

We are near the end. We conclude by asking about numerical stability. Of sourse, calculating $\vec{x}_{k+1}$ in floating point introduces errors. However, this is an iterative scheme that is (with a bit of luck) trying to get us closer to the real solution, so it fixes these errors for us.

Which brings us to some recommendations. When do we want to try iteration? For instance when the matrix of the system is ill conditioned, because then we cannot trust elimination. Another good reason is if the given system is really really huge. And yet another good reason is when the system has a sparse matrix with structure that can be used to efficiently program the iteration step.