Numerical Methods

Lev Leontev

February 17, 2023

Contents

***Definition* 1** (Numerical Methods)**.** Numerical Methods are algorithmic approaches to numerically solve mathematical problems. We use them often when it is hard/difficult/impossible to solve a problem analytically.

1 Taylor series

Given a function $f : \mathbb{R} \to \mathbb{R}$ (that is hard to evaluate for some $x \in \mathbb{R}$), but f and $f^{(n)}$ are known for a value c, which is close to x. Can we use this information to approximate $f(x)$?

We know values for $\cos^{(n)}(0)$.

$$\begin{cases} f(0) = \cos(0) = 1 \\ f'(0) = -\sin(0) = 0 \\ f''(0) = -\cos(0) = -1 \end{cases} \qquad \text{for } c = 0$$

Can we get $\cos(0.1)$ from this?

***Definition* 1** (Taylor series)**.** Let $f : \mathbb{R} \to \mathbb{R}$, differentiable infinitely many times at $c \in \mathbb{R}$. So we have $f^{(k)}(c),\ k = 1, 2, \ldots$. Then the Taylor series of f at c is:

$$f(x) \approx f(c) + \frac{f(c)}{1!}(x-c)^1 + \frac{f''(c)}{2!}(x-c)^2 + \cdots = \sum_{k=0}^{\infty} \frac{f^{(k)}}{k!}(x-c)^k$$

Remark. Taylor series is a power series.

Remark. For $c = 0$ also known as Maclaurin series

Remark. A power series has an interval/radius of convergence. You can only evaluate the series if $x \in$ interval of convergence.

Example 1. What is the Taylor series for $f(x) = e^x$ at $c = 0$? We have $f^{(k)}(x) = e^x$, so $f^{(k)}(0) = 1$. Thus:

$$\sum_{k=0}^{\infty} \frac{1}{k!} x^k$$

and the radius of convergence is ∞.

I.e. for any $x \in \mathbb{R}$:

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

For an algorithm we need a finite amount of terms. For example,

$$e^x \approx \frac{1}{0!}x^0 + \frac{1}{1!}x^1 + \frac{1}{2!}x^2 = 1 + x + \frac{x^2}{2}$$

This is a polynomial!

Example 2. Let's calculate Taylor series of a polynomial.

$$f(x) = 4x^2 + 5x + 7,\ c = 2$$
$$f(2) = 33,\ f'(2) = 8x + 5\Big|_{x=2} = 21,\ f''(2) = 8$$

Taylor series:

$$33 + 21(x-2) + \frac{8}{2}(x-2)^2 = 4x^2 + 5x + 7 = f(x)$$

Taylor series of a polynomial is itself.

Theorem 1 (Taylor theorem)**.** Let $f \in \mathrm{C}^{n+1}([a,b])$ (i.e. f is $(n+1)$-times continuously differentiable). Then for any $x \in [a,b]$ we have that

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(c)}{k!}(x-c)^k + \frac{f^{(n+1)}(\xi_x)}{(n+1)!}(x-c)^{n+1}$$

where ξ_x is a point that depends on x and which is between c and x.

The first sum is called *truncated Taylor series*, the remainder is called *the error*.

Example. For $n = 0$:

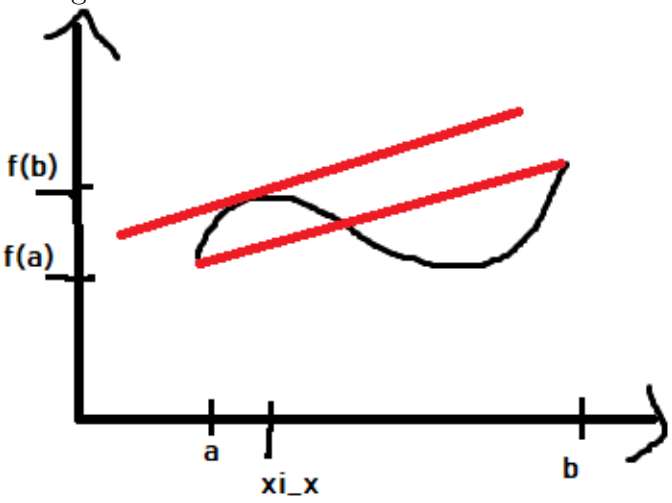
$$f(x) = f(c) + f'(\xi_x)(x-c)$$

Choose $c = a$, $x = b$:

$$f(b) = f(a) + f'(\xi_x)(b-a) \iff f'(\xi_x) = \frac{f(b)-f(a)}{b-a}$$

This is the mean value theorem!

Figure 1: Mean value theorem illustration

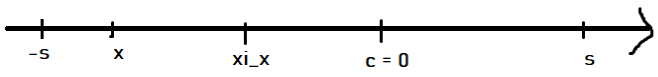


***Definition* 2.** We say that the Taylor series *represents* the function f at x if the Taylor series converges at that point, i.e. the remainder tends to zero as $n \to \infty$.

Example 1. Back to e^x: $f(x) = e^x$, $c = 0$, ξ_x is between c and x.

$$e^x = \sum_{k=0}^{n} \frac{x^k}{k!} + \frac{e^{\xi_x}}{(n+1)!}x^{n+1}$$

For any $x \in \mathbb{R}$ we find $s \in \mathbb{R}_0^+$ ($\mathbb{R}_0^+$ are all real, positive numbers including 0) so that $|x| \leqslant s$, and $|\xi_x| \leqslant s$ because ξ_x is between c and x.



Because e^x is monotone increasing, we have $e^{\xi_x} \leqslant e^s$, thus

$$\lim_{n\to\infty}\left|\frac{e^{\xi^x}}{(n+1)!}x^{n+1}\right| \leqslant \lim_{n\to\infty}\left|\frac{e^s}{(n+1)!}\right|s^{n+1} = e^s \lim_{n\to\infty}\frac{s^{n+1}}{(n+1)!} = 0$$

Because $(n+1)!$ will grow faster than any power of $s \implies \lim_{n\to\infty}\left|\frac{e^{\xi x}}{(n+1)!}x^{n+1}\right| = 0$.

Thus e^x is *represented* by its Taylor series.

Example 2.

$$\begin{aligned}
&f(x) = \log(1+x),\ c = 0\\
&f'(x) = \frac{1}{1+x} = (1+x)^{-1}\\
&f''(x) = -(1+x)^{-2}\\
&f'''(x) = +2(1+x)^{-3}\\
&f^{(k)}(x) = (-1)^{k+1}(k-1)!\frac{1}{(1+x)^k}
\end{aligned}$$

So $f^{(k)}(0) = (-1)^{k-1}(k-1)!$ for $k \geqslant 1$, $f(0) = \log(1) = 0$.

Taylor series:

$$f(x) = \sum_{k=1}^{n}\frac{(-1)^{k-1}}{k}x^k + \frac{(-1)^k}{n+1}\frac{1}{(1+\xi_x)^{n+1}}\cdot x^{n+1} \quad \left(\frac{n!}{(n+1)!} = \frac{1}{n+1}\right)$$

$$E_n(x) = \frac{(-1)^k}{n+1}\frac{1}{(1+\xi_x)^{n+1}}\cdot x^{n+1} \text{ — the remainder}$$

Question: for which x does $\lim_{n\to\infty} E_n(x) = 0$?

$$\lim_{n\to\infty} E_n(x) = \lim_{n\to\infty}\frac{(-1)^n}{n+1}\left(\frac{x}{\xi_x+1}\right)^{n+1} \text{ for } \xi_x \in (c,x)\ (c=0)$$

Such a limit converges to 0, if the fraction is less than 1.

$$0 < \frac{x}{\xi_x+1} < 1 \iff x < \xi_x + 1 \iff x - \xi_x < 1 \text{ with } \xi_x \in (0,x) \iff x \leqslant 1$$

Consequence. $\lim_{n\to\infty} E_n(x) = 0$ if $0 < x \leqslant 1$. This means that the Taylor series represents $\log(x+1)$ for $x \in [0,1]$. We can extend this to show $x \in (-1,1]$.

Example 3. Let's compute $\cos(0.1)$. Let's approximate it with Taylor series with $c = 0$ (around zero).

$$\cos(x) = 1 - \frac{x^2}{2} + \frac{x^4}{4!} - \frac{x^6}{6!} \pm \cdots + \text{remainder}$$

Consequence.

$$\left|\cos(x) - \sum_{k=0}^{n}(-1)^k\frac{x^{2k}}{(2k!)}\right| = \left|(-1)^{n+1}\cos(\xi_x)\frac{x^{2(n+1)}}{\big(2(n+1)\big)!}\right| \leqslant \frac{0.1^{2(n+1)}}{2(n+1)!} \underset{n\to\infty}{\longrightarrow} 0$$

n	Taylor polynomial	$\lvert error\rvert \leqslant$
0	1	$\frac{(0.1)^2}{2} = 0.0005$
1	0.995	$\frac{0.0001}{24}$
2	0.99500416	$\frac{0.000001}{6!}$

Error depends on choice of $|x - c|$ and n.

Example 4. Compute $\log(2)$ using $f(x) = \log(x + 1)$

$$\log(2) = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \frac{1}{6} + \dots$$

Keeping 8 terms (until $n = 8$) we get $\log(2) \approx 0.63452$, the actual solution is $\log(2) = 0.693147$. Not so accurate. Can we improve?

We can use Taylor series of $\log\left(\frac{1+x}{1-x}\right)$ instead, since $\log\left(\frac{1+x}{1-x}\right) = \log(1 + x) - \log(1 - x)$. We choose $x = \frac{1}{3}$ instead of $x = 1$. Since x is closer to zero, both of the logarithms converge quicker.

$$\left(\log\left(\frac{1 + 1/3}{1 - 1/3}\right) = \log(2)\right)$$

We then get

$$\log(2) = 2 \cdot \left(\frac{1}{3} + \frac{1}{3^3 \cdot 5} + \dots\right)$$

We only need 4 terms to get $\log 2 \approx 0.69313$.

Theorem 2 (Reformulation of Taylor's theorem)**.** $f \in \mathrm{C}^{n+1}([a, b])$. We change c to x and the old x to $x + h$ from previous version $\implies$ get for $x,\ x + h \in [a, b]$:

$$f(x + h) = \sum_{k=0}^{n} \frac{f^{(k)}(x)}{k!} h^k + \frac{f^{(n+1)}(\xi_x)}{(n + 1)!} h^{n+1} \text{ where } \xi_x \in (x, x + h),\ h > 0$$

We can write error term as

$$f(x + h) - \sum_{k=0}^{n} \frac{f^{(k)}(x)}{k!} h^k = \mathcal{O}(h^{n+1})$$

Remark. Let's recall what the $\mathcal{O}$-notation means. $a(h) = \mathcal{O}\big(b(h)\big)$ if $\exists c > 0$ such that $\frac{a(j)}{b(j)} \leqslant c$ as $h \to 0$. So, for $n = 1$ the error decreases with h^2 (quadratic convergence).
$n = 2$: error decreases cubically, i.e. h^3, etc.

Summary of Taylor series:

- Problem: Evaluate $f(x)$ with a given error bound.
- Required: $f \in \mathrm{C}^{n+1}$, values of derivatives $f^{(k)}(a)$.
- Check interval of convergence: does Taylor series expansion work?
- Estimate the maximum error for n terms of the Taylor polynomial.
- Choose n, such that the error bound is low enough.
- Evaluate the Taylor polynomial.

2 Number representation

2.1 Errors

There are different error types:

1. An error in data (partly due to roundoff).
2. Roundoff errors (during computation). For example, multiplication increases the amount of needed significant digits, and we can't store them all on a computer.
3. Truncation error, that is inherent to numeric methods. For example, if we take a finite number of terms in our Taylor series.

***Definition* 1.** Let $\tilde{a}$ be an approximation of a. Then $|\tilde{a} - a|$ is the *absolute error*, and $\left|\frac{\tilde{a}-a}{a}\right|$ is the *relative error*. The *error bound* is the magnitude of admissable error.

Example. 0.00123 with error $0.000004 = 0.4 \cdot 10^{-5}$. The error is below $\frac{1}{2} \cdot 10^{-t}$ with $t = 5$, so there's 5 correct digits and 3 significant digits (the number of non-leading zeros).

Example. 0.00123 with error $0.000006 = 0.06 \cdot 10^{-4} > \frac{1}{2} \cdot 10^{-5}$. Only has 2 significant digits, because we have to round the error up.

Theorem 1. In addition/subtraction the bounds for absolute errors are added, in multiplication/division the relative errors are added.

Example. Solve $x^2 - 56x + 1 = 0$:

$$x = 28 - \sqrt{783} \approx 28 - 27.982 \text{ (5 significant digits)} = 0.018 \pm \frac{1}{2} \cdot 10^{-3}$$

We end up with 2 significant digits in the answer, despite that we used to have 5. That's why computers use floating point numbers, as leading zeros are bad.

***Definition* 2** (Error propagation)**.** If $y(x)$ is smooth, than the derivative $|y(x)'|$ can be interpreted as the sensitivity of $y(x)$ to errors in x. We can generalize this to functions of multiple variables:

$$|\Delta y| \leqslant \sum \left|\frac{\partial y}{\partial x_i}\right| \cdot |\Delta x_i|, \text{ where } \Delta y = \tilde{y} - y,\ \Delta x_i = \tilde{x_i} - x_i$$

This is an empirical inequality that is only valid for small Δx_i. It is used a lot in physics.

2.2 Base representation

***Definition* 3** (Base representation)**.** Every number $x \in \mathbb{N}$ can be written in the following form as a unique expansion with respect to the base b, where $b \in \mathbb{N} \setminus \{1\}$:

$$x = a_0 b^0 + a_1 b^1 + a_2 b^2 + \cdots + a_n b_n = \sum_{i=0}^{n} a_i b^i$$
$$a \in \mathbb{N}_0,\ a_i < b,\ a_i \in \{0, \ldots, b-1\}$$

Here b is called the *base*, a_i are called the *digits*. Humans usually use base 10. But, for example, computers can use base 2.

For a real number $x \in \mathbb{R}$ we can write:

$$x = \sum_{i=0}^{n} a_i b^i + \sum_{i=1}^{\infty} a_{-i} b^{-i}$$

Example. $b = 2:\ 1011 = 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 = (11)_{10}$

There are different algorithms that convert number systems.

2.3 Euclid's algorithm

Euclid's algorithm converts $(x)_{10}$ to $(y)_b$.

1. Input $(x)_{10}$.
2. Determine the smallest n, such that $x < b^{n+1}$.
3. For $i = n$ to 0 do:

$$a_i \coloneqq x \text{ div } b^i \text{ (integer division)}$$
$$x \coloneqq x \bmod b^i \text{ (the remainder)}$$

4. Output result $a_n a_{n-1} a_{n-2} \dots a_0 = (y)_b$.

Example. 1. $(x)_{10} = (13)_{10} \to (y)_2$

2. $n = 3$ since $13 < 2^4$.
3.

$$i = 3:\ a_3 = 13 \text{ div } 2^3 = 1,\ x = 13 \bmod 2^3 = 5$$
$$i = 2:\ a_2 = 5 \text{ div } 2^2 = 1,\ x = 5 \bmod 2^2 = 1$$
$$i = 1:\ a_1 = 1 \text{ div } 2^1 = 0,\ x = 1 \bmod 2^1 = 1$$
$$i = 0:\ a_0 = 1 \text{ div } 2^0 = 1,\ x = 1 \bmod 2^0 = 0$$

4. Output: $(1101)_2 = (13)_{10}$.

Two problems of the Euclid's algorithm:

1. Step 2 is inefficient
2. Division by large numbers can be problematic.

2.4 Horner's scheme

Horner's scheme is a more efficient algorithm. The idea is to represent the number as follows:

$$(a_n a_{n-1} \dots a_0)_b = a_0 + b\big(a_1 + b(a_2 + b(a_3 + \dots + b(a_n))) \dots\big)$$

The algorithm is the following:

1. Input $(x)_{10}$.
2. $i \coloneqq 0$.

3. While $x > 0$ do:

$$a_i \coloneqq x \bmod b$$
$$x \coloneqq x \operatorname{div} b$$
$$i \coloneqq i + 1$$

4. Output result $a_n a_{n-1} a_{n-2} \dots a_0 = (y)_b$.

Remark. The algorithm is very similar to the Euclid's algorithm — the difference is that we execute it in reverse. We no longer have divisions by large numbers, and thus it runs faster.

General remarks:

- A number with simple representation in one base may be complicated to represent in another base. For example, $(0.1)_{10} = (0.0001100110011\dots)_2$.
- Base 2 is called *binary*, base 8 is *octal*, base 16 is *hexadecimal*.
- To convert from a base b to base 10 we can just perform the following computation:

$$(42)_8 = 4 \cdot 8^1 + 2 \cdot 8^0 = (34)_{10}$$

- Conversion 2 and 8. $8 = 2^3$: three consecutive bits represent one octal digit, e.g.

$$(551.624)_8 = (101101001.110010100)_2$$

- Conversion 2 and $16 = 2^4$: just four bits to one hexadecimal digit.
- Horner's scheme algorithm does not need estimate of n and does not divide by large numbers.
- It is applicable to real numbers, but one needs a critireon to stop if the representation with the new base is infinite.
- On computers we only have finite precision (the number of digits/bits).

2.5 Floating point representation

***Definition* 4.** Normalized floating point representation with respect to a base b stores any number x as follows:

$$x = 0.a_1 a_2 \dots a_k \cdot b^n \text{ where } a_i \in \{0, 1, \dots, b-1\}$$

a_i are called *digits*, k is called *precision*, n is called *exponent*, $a_1 \dots a_k$ is called *mantissa*, $a_1 \neq 0$ is called *normalization*, which makes the representation unique.

Remark. Leading zeros are essentially a waste of space. That's why floating points are useful when you add/subtract/divide numbers — as you're able to move the decimal point.

Example 1. Base 10: $32.213 = 0.32213 \cdot 10^2$.

Base 2: $x = \pm 0.b_1 b_2 \dots b_k \cdot 2^n$ We need another bit to define the sign of the number.

Example 2. For *single precision* floating-point numbers: 4 bytes = 32 bits.

- 1 bit sign mantissa.

- 1 bit sign exponent.
- 7 bits for exponent (integer).
- 23 bits for mantissa (24 effectively, since the first digit is always 1).

7 bits allow for the largest exponent of 127, i.e. $2^{127} \approx 10^{38}$. Anything above that is infinity. So, we have the range of $10^{-38} < |x| < 10^{38}$.

Since $2^{-24} \approx 10^{-7}$, we can represent 7 significant digits.

We can have a better representation, e.g. with *double precision* (8 bytes).

Issues with floating-point numbers:

- Adding numbers is not commutative, i.e. $x + y$ does not necessarily equal to $y + x$.
- It's not associative, i.e. $(x + y) + z$ does not necessarily equal to $x + (y + z)$. You usually try to add small numbers together first, in hopes that they will become big enough to become significant.

Example 1. Take $x = y = 0.00000033$, $z = 0.00000034$, $w = 1.00000000$. We compute $x + y + z + w$, and in that order we get 1.000001. Reversed order gets 1.000000 with base 10. Why is that? That's because we only have 7 significant digits.

Example 2.

$$\sum_{1}^{10^7} 1 + 10^7 = 1 + \cdots + 1 + 10^7$$

The order of summation will make a difference. You will either get 10^7 or $2 \cdot 10^7$, because $1 + 1 = 2$, but $1 + 10^7 = 10^7$ due to roundoff.

Consequence. Avoid adding numbers of different order of magnitude. Add numbers in increasing order of their size.

Example 3. Compute $x - \sin x$ for x close to 0, e.g. $x = 1/15$. Assume $k = 10$ precision.

$$\begin{aligned} x &= 0.6666666667 \cdot 10^{-1} \\ \sin x &= 0.6661729492 \cdot 10^{-1} \\ x - \sin x &= 0.0004937175 \cdot 10^{-1} = 0.4937175\underline{000} \cdot 10^{-4} \end{aligned}$$

Notice the three zeros at the end — that is a sign of a precision loss (unless the number actually ends with zeros).

Consequence. Avoid subtracting numbers of similar size, because it leads to a loss of precision.

A potential solution in this case would be to use Taylor series expanison:

$$\begin{aligned} \sin x &= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots \\ x - \sin x &= +\frac{x^3}{3!} - \frac{x^5}{5!} + \dots \end{aligned}$$

Using 3 terms we get: $0.4937174328 \cdot 10^{-4}$. The error of the Taylor series with 3 terms will be $\leqslant 10^{-13}$ (which is needed for "full" precision for $0.\ldots \cdot 10^{-4}$).

Theorem 2. Let x, y be two normalized floating point numbers with $x > y > 0$ and base $b = 2$. If there exist $p, q \in \mathbb{N}_0$ such that

$$2^{-p} \leqslant 1 - \frac{y}{x} \leqslant 2^{-q}$$

Then at most p and at least q significant bits (digits base 2) are lost during subtraction.

3 Linear Systems of Equations

***Definition* 1.** A linear system of equations is given by

$$Ax = b,\ A \in \mathbb{R}^{m\times n},\ x \in \mathbb{R}^n, b \in \mathbb{R}^m$$

I.e. the matrix A has m rows and n columns, x is a vector with n unknowns, b has m entries, thus the system has m equations.

Remark. The system of equations is called *linear*, because the degree of all x_i is equal to one.

Remark. If $n = m$, the system is called *square*. (As the matrix is square).

Remark. We can also write the system as a sum:

$$\sum_{j=1}^{n} a_{ij}x_j = b_i,\ i = 1, \dots, m$$

Example.

$$\begin{cases} a_{11}x_1 + a_{12}x_2 = b_1 \\ a_{21}x_1 + a_{22}x_2 = b_2 \end{cases} \qquad n = 2,\ m = 2$$

Linear systems of equations arise in a lot of problems:

- Geometrical problems (coordinate transforms, 3D matrices).
- Electrical circuits, Kirchhoff's laws/Ohm's laws.
- Solving differential equations.
- GPS.

3.1 Gaussian Elimination (GE)

Assume that $m = n$ (square system). The idea of Gaussian Elimination: do row operations to produce an upper triangular matrix (echelon form). Then do backward substitution to solve the system.

Allowed row operations:

1. Swap rows.
2. Scale rows, i.e. multiply a row by a scaler.
3. Add multiples of one row to another.

Example.

$$A = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix}, \quad b = \begin{bmatrix} 16 \\ 26 \\ -19 \\ -34 \end{bmatrix}$$

Step 1. Do GE in a systematic way:

$$\text{Augmented matrix} = \left[\begin{array}{cccc|c} 6 & -2 & 2 & 4 & 16 \\ 12 & -8 & 6 & 10 & 26 \\ 3 & -13 & 9 & 3 & -19 \\ -6 & 4 & 1 & -18 & -34 \end{array}\right]$$

The 6 here is the pivot element, and the first row is the pivot row.

$$\left[\begin{array}{cccc|c} 6 & -2 & 2 & 4 & 16 \\ 12 & -8 & 6 & 10 & 26 \\ 3 & -13 & 9 & 3 & -19 \\ -6 & 4 & 1 & -18 & -34 \end{array}\right] \begin{array}{l} \leftarrow \text{ pivot row} \\ \leftarrow (-2)\cdot R_1 + R_2 \\ \leftarrow (-1/2)\cdot R_1 + R_3 \\ \leftarrow 1\cdot R_1 + R_4 \end{array}$$

$$\hookrightarrow \left[\begin{array}{cccc|c} 6 & -2 & 2 & 4 & 16 \\ 0 & -4 & 2 & 2 & -6 \\ 0 & -12 & 8 & 1 & -27 \\ 0 & 2 & 3 & -14 & -18 \end{array}\right] \begin{array}{l} \\ \leftarrow \text{ pivot row} \\ \leftarrow (-3)\cdot R_2 + R_3 \\ \leftarrow (1/2)\cdot R_2 + R_4 \end{array}$$

Always consider the factor, e.g.

$$-3 = -\left(\frac{-12}{-4}\right)$$
$$\frac{1}{2} = -\left(\frac{2}{-4}\right)$$

Eventually, we end up with a triangular form (using diagonal elements as pivots).

$$\left[\begin{array}{cccc|c} 6 & -2 & 2 & 4 & 16 \\ 0 & -4 & 2 & 2 & -6 \\ 0 & 0 & 2 & -5 & -9 \\ 0 & 0 & 0 & -3 & -3 \end{array}\right]$$

Step 2. Backward substitution:

- Last row: $-3x_4 = -3 \Longleftrightarrow x_4 = 1$.
- Second last row:

$$2x_3 - 5x_4 = -9$$
$$2x_3 - 5 = -9 \Longleftrightarrow x_3 = -2$$

- ... finally: $x_1 = 3,\ x_2 = 1,\ x_3 = -2,\ x_4 = 1$.

The algorithm again:

1. Input $A \in \mathbb{R}^{n\times n},\ b \in \mathbb{R}^n$.

   Forward substitution:

2. For $k = 1, \ldots, n-1$ (for all pivot rows, except the last one):

3. For $i = k+1, \ldots, n$ (for all rows below the pivot row):

4. For $j = k, \ldots, n$ (for all columns from the pivot one):
   $a_{ij} := a_{ij} - \frac{a_{ik}}{a_{kk}} a_{kj}$
   End for.
   $b_i := b_i - \frac{a_{ik}}{a_{kk}} b_k$

$\overline{3}$. End for.

$\overline{2}$. End for.

Backward substitution:

5. $x_n = \frac{b_n}{a_{nn}}$ (last unknown)

6. For $i = n-1, \ldots, 1$ (return back row by row)
   rhs $\coloneqq b_i$

7. For $j = n, \ldots, i+1$ (for all columns up to the pivot element)
   rhs $\coloneqq$ rhs $- a_{ij}x_i$ (x_j are already known)

$\overline{7}$. End for. $x_i \coloneqq \frac{\text{rhs}}{a_{ii}}$

$\overline{6}$. End for.

GE can be used whenever the pivots don't vanish.

Example.

$$\begin{cases} x_1 + x_2 + x_3 = 1 \\ x_1 + x_2 + 2x_3 = 2 \\ x_1 + 2x_2 + 2x_3 = 1 \end{cases} \implies \begin{cases} x_1 = 1 \\ x_2 = -1 \\ x_3 = 1 \end{cases}$$

But addition of rows will give us:

$$\begin{cases} x_3 = 1 \text{ – here we have a missing pivot} \\ x_2 + x_3 = 0 \end{cases} \qquad \left(\begin{array}{ccc|c} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 2 & 2 & 1 \end{array}\right)$$

We already get into trouble with very small pivot elements.

Example. Let $\varepsilon > 0$ and consider

$$\begin{cases} \varepsilon x_1 + x_2 = 1 \\ x_1 + x_2 = 2 \end{cases} \iff \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix} \vec{x} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

For $\varepsilon \ll 1$, the actual solution is $x_1 \approx x_2 \approx 1$. However, GE yields

$$x_2 = \frac{2 - \frac{1}{\varepsilon}}{1 - \frac{1}{\varepsilon}} \overset{\varepsilon \ll 1}{\approx} \frac{-\frac{1}{\varepsilon}}{-\frac{1}{\varepsilon}} = 1$$

WIth finite precision we will get through backward substitution: $x_2 = 1$ and $x_1 = \frac{1-x_2}{\varepsilon} = 0$ which is wrong. The pivot is too small. But change order of equations.

$$\begin{cases} x_1 + x_2 = 2 \\ \varepsilon x_1 + x_2 = 1 \end{cases} \overset{\text{GE}}{\implies} \begin{cases} x_2 = \frac{1-2\varepsilon}{1-\varepsilon} \\ x_1 = 2 - x_2 \end{cases}$$

Now the answer is correct. The reason why the first one was incorrect is error amplification of x_2 by multiplication. $\frac{1}{\varepsilon}$ leads in the first case to a wrong result.

3.2 Scaled partial pivoting

***Definition* 2.** *Pivoting* means that the pivot element is chosen appropriately, and not just row by row.

***Definition* 3.** *Partial pivoting* means we will reorder rows (not columns, otherwise it would be full pivoting).

***Definition* 4.** *Scaled* means we look for best *relative* pivot, i.e. best ratio between pivot element and maximal entry of row (all in absolute values).

Remark. This will lead to minimal error propagation.

The algorithm:

1. Input $A \in \mathbb{R}^{n\times n},\ b \in \mathbb{R}^m$.
2. Find maximal absolute values of entries in rows $s \in \mathbb{R}^n$, such that $s_i = \max_{j=1}^{n} |a_{ij}|$.
   Forward elimination:
3. For $k = 1, \ldots, n-1$ (for all pivot rows).
4. For $i = k, \ldots, n$ (for all rows below pivot row)
   compute $\left|\frac{a_{ik}}{s_i}\right|$.

$\overline{4}$. End for.

5. Find row with the largest relative pivot element, name it row j.
6. Swap k with j.
7. Swap entries k and j in vector s.
8. Do skip of forward elimination in row k.

$\overline{3}$. End for.

Backward substitution is done as before, but with updated order.

Example.

$$\left[\begin{array}{cccc|c} 3 & -13 & 9 & 3 & -19 \\ -6 & 4 & 1 & -18 & -32 \\ 6 & -2 & 2 & 4 & 16 \\ -12 & -8 & 6 & 10 & 26 \end{array}\right]$$

Initial $s = (13, 18, 6, 12)$. Iterations:

1. - Relative pivots:
   $$\left(\frac{3}{13}, \frac{6}{18}, \frac{6}{6}, \frac{12}{12}\right) = \left(\left|\frac{a_{ik}}{s_i}\right|\right)$$
   - Rows 3 and 4 have pivot 1 greater than all others. Select for swapping rows 1 and 3.
   $$\left[\begin{array}{cccc|c} 6 & -2 & 2 & 4 & 16 \\ -6 & 4 & 1 & -18 & -32 \\ 3 & -13 & 9 & 3 & -19 \\ 12 & -8 & 6 & 10 & 26 \end{array}\right]$$
   - Swap entries $3 \leftrightarrow 1$ in s: $(6, 18, 13, 12)$.
   - Forward elimination step (like in GE):
   $$\left[\begin{array}{cccc|c} 6 & -2 & 2 & 4 & 16 \\ 0 & 2 & 3 & -14 & -18 \\ 0 & -12 & 8 & 1 & -27 \\ 0 & -4 & 2 & 2 & -6 \end{array}\right]$$

2. On the second iterations, $k = 2$.

   - Relative pivots (we don't care about the first row anymore, so just three rows left):

     $$\left(\left| \frac{2}{18} \right|, \left| \frac{12}{13} \right|, \frac{4}{12} \right)$$

     The second ratio is the largest, and it corresponds to the third row.
   - So, we swap row 3 with row $k = 2$.
   - Swap entries in s.
   - Forward elimination. Then backward substitution on updated matrix as before.

Remarks:

- In efficient implementations, the step of row swapping can be omitted, just a permutation vector l needs to be stored to keep track of matrix rearrangements. This will result in "echelon form" that will look like e.g.

$$\begin{array}{cc} 2 \rightarrow \\ 4 \rightarrow \\ 1 \rightarrow \\ 3 \rightarrow \end{array} \left[\begin{array}{cccc|c} 0 & * & * & * & * \\ 0 & 0 & 0 & * & * \\ * & * & * & * & * \\ 0 & 0 & * & * & * \end{array}\right]$$

- GE with scaled partial pivoting always works when matrix is invertible, i.e. there exists a A^{-1}, such that $AA^{-1} = I$.

  It will fail for a singular (i.e. not invertible) matrix, because eventually a division by 0 will occur.

- Doing Gaussian elimination has computational complexity of $\mathcal{O}(n^3)$, because we have three nested for-loops. Cubic behaviour n^3 is problematic for large n!

- Traditionally, only the multiplication/division operations were counted in the number of operations C. (Since addition is very cheap). On present-day hardware, however, the costs are nearly as "cheap" as addition or subtraction.

- We are missing costs due to exchange with memory. Therefore, estimates of time complexity and reality may diverge substantially.

- Backward substitution has order n^2, which does not affect the general estimate of n^3.

- Scaled partial pivoting leads to an increase in cost, but order stays n^3.