RUHR
UNIVERSITÄT
BOCHUM

RUB

Institut für Theoretische Physik

---

Notes on

# Computational Physics I

WS 2025/26

Jürgen Dreher & Kevin Schoeffler

January 19, 2026

Based on the lecture "Computational Physics I" from WS 2020/21 and WS 2022/23.

# Contents

# 1 Fundamentals

Why Computational Physics? Physics is concerned with exploring and untangling natural phenomena through observation, experiment, and theoretical modeling. *Models* are formulated by means of mathematical representations of physical concepts like entities, quantities, interactions and the like, and are subsequently investigated with respect to their capabilities and restrictions. Model equations are often difficult to handle analytically, and even fairly simple systems require *numerical computation* to be solved. This becomes even more pressing when modifications are made to idealized theoretical setups in order to provide results of practial relevance.

The numerical treatment of model equations will in general introduce the additional aspect of *numerical errors* by using methods and algorithms that provide only approximations to the true solutions of model equations, if they exist at all. Hence, we must be able to assess and control the quality and characteristics of numerical methods if we want to judge the numerical computations' outcome in their power to represent physics.

We must not forget this *two-step* abstraction from actual physics to a theoretical model and then to a numerical approximation: If the former is poor, even the most powerful supercomputer can't help.

## 1.1 Number representation

The computer encodes numbers as a sequence of *bits* in a pre-defined way. A bit is the smallest unit of information that is in one of two possible states, typically called 0 and 1, respectively, and is realized by means of electrical circuits in modern computers. Assigning, for instance, a value to a variable in our computer program will assign the corresponding bit pattern at the variable's location in computer memory.

The way in which a bit sequence is interpreted as a number is called a *data type*. This comprises both the set of numbers to be represented and their actual mapping to bit patterns. Typically, different data types are used in a computer program for different purposes, and their definition might depend on the programming language in use.

Let's take our well-known ten-digit decimal system as reference: A natural number like "seventhousand-fifty-eight" in $\mathbb{N}$ is represented

by the sequence

$$7058 = (7, 0, 5, 8)_{(10)} = 7 \cdot 10^3 + 0 \cdot 10^2 + 5 \cdot 10^1 + 8 \cdot 10^0$$

where the lower index $(10)$ reminds us to interpret these digits in our **decimal system**. Here, four digits give us $10000 = 10^4$ different possibilities to represent a number, hence, $N$ digits will give us $10^N$ such possibilities. Inidicate the **digits** as $d_i \in \{0, 1, 2, ..., 9\}$, a number $z$ in *decimal representation* naturally follows from its digit sequence as

$$z = (d_{N-1}, d_{N-2}, ..., d_1, d_0)_{(10)} = \sum_{j=0}^{N-1} d_j \cdot 10^j$$

*Binary representation* of a natural number works exactly like this, with the only difference that the available "digits" are only bits $b_j \in \{0, 1\}$, and the "base value" of bit number $j$ now is $2^j$ instead of $10^j$. Hence, encoding in the **binary system** based on bits is

$$z = (b_{N-1}, b_{N-2}, ..., b_1, b_0)_{(2)} = \sum_{j=0}^{N-1} b_j \cdot 2^j$$

Four bits allow for 16 different numbers to be represented, for example the natural numbers from 0 bis 15. Examples for this encoding are $(1, 1, 0, 1)_{(2)} = 13$, and $(0, 1, 1, 0)_{(2)} = 6$. This is a *sign-free* interpretation of the bit sequence as an integer number, we might call it a 4-bit **unsigned integer** data type, in short tech-talk an **unsigned int** or even **uint**.

What about **negative numbers**? We extend the representation to **signed** data types: A simple possiblity would be to sacrifice on bit to indicate $\pm$, together with the additional agreement for zero to have the sign bit always positive or negative. For ease of technical implementation, negative numbers are encoded in the so-called *two-complement* representation (for details, see e.g. Wikipedia). Here, the "highest" bit is "1" in negative numbers and "0" in non-negative numbers, while the resulting bits "count" upwards from the smallest possible value. With this system, one byte (= 8 bits) allows for the representation of the signed numbers $(-128, ..., +127)$, e.g., $(0, 1, 1, 0, 0, 1, 0, 1)_{(2)} = 101$ und $(1, 1, 1, 0, 0, 1, 0, 1)_{(2)} = -27$.

We're not satisfied with integer numbers: We know **fixed-point numbers** with a defined number of digits on either side of the decimal

point, e.g. from bank transaction forms. Such things can easily, be implemented by appropriate definitions like, e.g.,

$$50.92 = (0,5,0).(9,2)_{(10)} = 0 \cdot 10^2 + 5 \cdot 10^1 + 0 \cdot 10^0 + 9 \cdot 10^{-1} + 2 \cdot 10^{-2} = \sum_{j=-2}^{2} d_j \cdot 10^j$$

in the decimal representation, or analogously in the binary system as

$$3.25 = (0,1,1).(0,1)_{(2)} = 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = \sum_{j=-2}^{2} b_j \cdot 2^j$$

again with two-complement representation of signed data types.

In physics, we will often work with numbers that are really big or small (by magnitude). Take, for instance, the elementary charge, measured in Coulomb, as approximately

$$\texttt{1.6022e-19} = 1.6022 \cdot 10^{-19} = M \cdot 10^E$$

What has happened? We're using - even before resorting to computers or binary representation - a **floating-number** representation to avoid writing down 25 or so digits, most of them being zero: The **mantissa** $M = 1.6022$, a fixed-point with a certain number of digits that determine the number's **precision**, gets multiplied with a scaling factor that "shifts" the point as specified by the integer signed exponent $E = -19$, hence the name *floating-point* representation. In addition to the number of digits in the mantissa, the size of the exponent - two digits in the example above - determines the *dynamical range* of the representation, i.e., the ratio of largest to smallest magnitude that can be encoded. With two decimals in the exponent and six digits in the mantissa, we arrive at approximately $10^{205}$ for this dynamical range. The *resolution*, i.e., the difference between "neighboring" numbers, varies from number to number. For instance, `-6.224e55` and `-6.223e55` are by $10^{52}$ apart from each other! And remember: How many different numbers can be represented with the $5 + 2$-digit representation above?

In computer science, data types are usually based on bytes, i.e. bunches of eight bits, so that we typically work with 16, 32, 64, oder 128 bit types. In particular, floating-point types usually make use of standardized definitions that are termed **32-bit** or **"single precision"** and **64-Bit** or **"double precision"**, and wich are precisely defined in the standard **IEEE 754** (details in e.g. `http://en.wikipedia.org/wiki/IEEE754`). The "double precision" type, for instance, defines a 53-bit mantissa (signed fixed-point) and an 11-bit exponent (signed integer). Thus, the exponent alone provides a dyanamical range of $2^{-1022}...2^{1023} \approx 10^{-308}...10^{308}$, the

mantissa's precision is approximately $2^{-53} \approx 10^{-16}$. In addition to these arrangements, special bit patterns are reserved to indicate special conditions that can occur while operating with these numbers, and these are defined in IEEE754 as well. Examples are arithmetic overflow conditions indicated by **Inf**, akin to "infinity", and arithmetic failures like **NaN**, short for "not-a-number". Arithmetic operations themselves, including **rounding** of results, are also well-defined, and identical across computing architectures and programming languages, thereby giving *exactly* identical results in terms of the bit patterns, as long as the standard is adhered to.

When working in the **Matlab** environment, the silently defined standard type for numbers is "double" (precision), so that an assignment like **a = 5.5** will make **a** a variable of type "double." Use of, and conversion into, other types like single precision must be done explicitly, e.g. by using conversion functions like **single**. Integer types can be instantiated with, e.g. **uint32**.

In contrast to the standard floating-point data types, the implementation of **integer data types** is usually **not standardized**. Their characteristics, as well as integer **arithmetics** can **differ** between programming environments and computer hardware.

## 1.2 Discretization

In physics, we're often working with functions defined on continuous sets, e.g., the time dependent position of a particle, $\vec{r}(t)$, or a measured quantity $V(t)$, or physical "fields" that are functions of space and time. In the following, we will optimistically assume that such functions are "well-behaved," that is, continous and sufficiently smooth, as we do in physics in general. When working analytically, or with symbolic calculus in the computer, we describe those functions' characteristics on an abstract level. Whenever this comes to an end, because things start getting complicated and cease to be treatable by analytical means, we might resort to **numerics** where those **functions** are **approximated** by a finite number of **numbers** (typically real-valued) in the computer. This approximate representation of a continous set of continous functions by a finite set of discrete numbers, which typically implies the restriction to a finite set of representable functions, is called a **discretization**. It will often happen automatically in practice, for example if a lab instrument delivers a sequence of measurement values with given cadence.

Such discretization $f(x) \to f_i, i = 1, 2, 3, ...$ with numbers $f_i$ can be done in many different ways: The $f_i$ could, for example, be

- the values $f$ at some predefined positions $x_i$, hence $f_i = f(x_i)$
- the averages of $f$ in predefined intervals $I_i$ of $x$
- the Fourier coefficients of $f$, or coefficients of some other expansion of $f$, e.g. a polynomial expansion

In this section, we will address the first case, that is, we are provided with values of $f$ in some regular or irregular intervals. We will not discuss the fact that in practice, such data from an apparatus in the lab or similar, will already come with measurement errors of various origins.

We operate frequently upon functions, like taking derivatives or integrals and much more, and we will want to find corresponding **discrete operators** that act on the function values $f_i$ to give us, hopefully "good", approximations to the original operators' outcome. The challenge now is not only to find such discrete operators, but also to quantify their "qualities" and behavior. We discuss this first by addressing the derivatives of our function $f(x)$ and approximate it by *finite differences*.

## 1.3 Finite differences

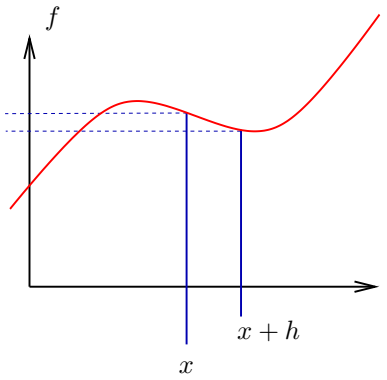
Given $f(x)$, we want to find a discrete approximation to its first derivative $f' =: D[f]$, that is, a *formula* to compute the value of $f'$ at some position $x$ from $f$'s own discrete values $f_i$ at points $x_i$ (we should be precise and call that position like $x_D$, but we're not).

Remenber the definition of the right-sided derivative as

$$\frac{df}{dx}(x) = f'(x) := \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}$$

and stay "before of the limit" with finite $h$ to postulate the approximation

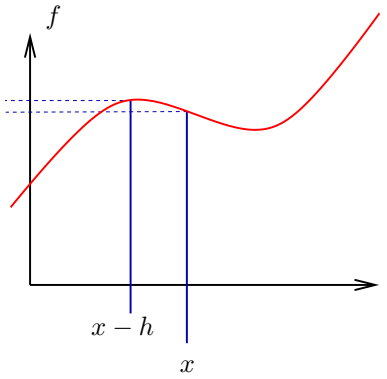
$$f'(x) \approx \frac{f(x+h) - f(x)}{h} =: D_h^+ f$$




This is the *right-sided* or *forward* finite difference, which corresponds to the slope of the graph of $f$ approximated from the right-sided triangle.

Same to the left gives us the *left-sided* or *backward* finite difference,

$$f'(x) \approx \frac{f(x) - f(x-h)}{h} =: D_h^- f$$




It's different from the one above, and both are approximations only.

Yet another approximation, without figure given, is the *central difference*,

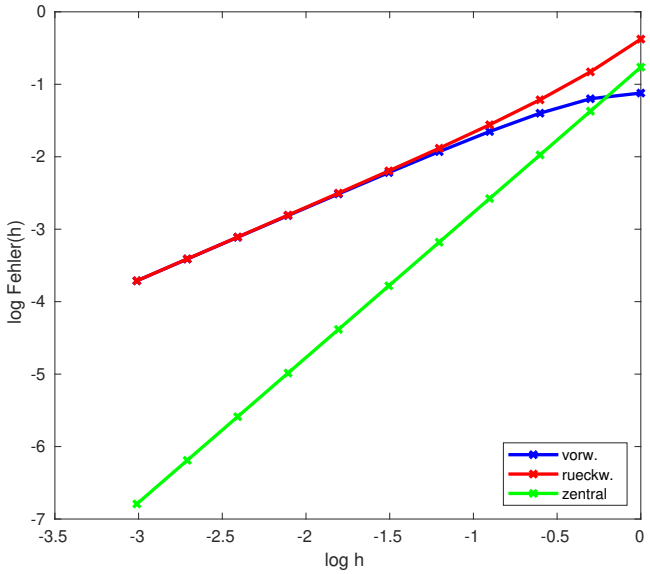
$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} =: D_h^c f$$

Thus, we "found" three *different* discrete operators, $D_h^+$, $D_h^-$ und $D_h^c$, for the *same* continous $D := (\cdot)' = \frac{d}{dx}$, and each of them contains an "artificial" numerical parameter $h$, that is the "width" in $x$ of the finite difference. We hope that all these approximations will eventually get better and approximate $D$ in the limits $h \to 0$ as closely as we might wish.
We say that such a discrete operator $D_h$ to the original $D := \frac{d}{dx}$ is

- **convergent**, if, for all differentiable $f$ and at every $x$, the limit $\lim_{h\to 0} D_h f$ exists,
- **consistent** with $D$, if it converges towards the "correct" value $f'(x)$,
- consistent to $n$-**th order** in $h$, short $\mathcal{O}(h^n)$, if the *discretization error* $E_h := D_h f - f'$ approaches zero at least as $h^n$ with $h \to 0$.

We test our discrete operators "experimentally" by carrying out a **convergence test**: We

- choose a "suitable" *test function* $f(x)$ with known $f'(x)$
- choose a reference point $x$ and various difference widths $h > 0$
- compute the approximation $(D_h f)(x)$ for decreasing values of $h$
- compute the discretization error $E_h := (D_h f - f')(x)$




The graphs display the error behavior in this "almost"-limit $h \to 0$: $D_h^+$ and $D_h^-$ are $\mathcal{O}(h)$ consistent to $D$, i.e. first order, while $D_h^c$ is even second order consistent. We might have expected that $D_c$ would be "better". But is it "better"? And can we understand this from the various finite difference formula?

The key technique to derive and quantify such finite difference formulae is Taylor-expanding $f$ around the reference point $x$ (remember that we assumed $f$ to be "sufficiently" smooth):

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{h^2 f''(x)}{2} + \frac{h^3 f'''(x)}{6} + \mathcal{O}(h^4) \\ f(x-h) &= f(x) - hf'(x) + \frac{h^2 f''(x)}{2} - \frac{h^3 f'''(x)}{6} + \mathcal{O}(h^4) \end{aligned}$$

Now, re-arrange and divide by $h$ (take care of the $\mathcal{O}()$-terms!), and we get our formulae, supplemented with additional information about the error terms:

$$\begin{aligned} D_h^+ f(x) := \frac{f(x+h) - f(x)}{h} &= f'(x) + \frac{hf''(x)}{2} + \frac{h^2 f'''(x)}{6} + \mathcal{O}(h^3) = f'(x) + \mathcal{O}(h) \\ D_h^- f(x) := \frac{f(x) - f(x-h)}{h} &= f'(x) - \frac{hf''(x)}{2} + \frac{h^2 f'''(x)}{6} + \mathcal{O}(h^3) = f'(x) + \mathcal{O}(h) \\ D_h^c f(x) := \frac{f(x+h) - f(x-h)}{2h} &= f'(x) + \frac{h^2 f'''(x)}{6} + \mathcal{O}(h^3) = f'(x) + \mathcal{O}(h^2) \end{aligned}$$

The crucial point about the central formula is obvious: Due to the symmetry of the formula, error contributions of $\mathcal{O}(h)$ (and all higher odd orders in $h$) cancel, and we obtain a second order formula.

Writing down the difference formulae can be tedious, although they don't contain too much substance. We can abbreviate the formalism and write down the difference schemes, or "stencils" as they're commonly named, in several more compact ways. They're not standard, but often useful and introduced ad-hoc.

One way, for a fixed increment $h$, is to simply write down the coefficients that fall on the function values $f_i$, in brackets, and mark the central coefficient of $f_i$ with an underbar, for example

$$\begin{aligned}
\frac{1}{h}\left[0 \;\; \underline{-1} \;\; 1\right] f(x) &:= \frac{f(x+h)-f(x)}{h} = D_h^+ f(x) \\
\frac{1}{h}\left[-1 \;\; \underline{1} \;\; 0\right] f(x) &:= \frac{f(x)-f(x-h)}{h} = D_h^- f(x) \\
\frac{1}{2h}\left[-1 \;\; \underline{0} \;\; 1\right] f(x) &:= \frac{f(x+h)-f(x-h)}{2h} = D_h^c f(x)
\end{aligned}$$

As the schemes act linearly on the $f_i$, we can do calculations with those stencils.
For example, we immediately see that $D_h^c = (D_h^+ + D_h^-)/2$.

---

With this technique, we found a systematic way to derive advanced formulae for other operators, or/and of even higher consistency order, that we would no more "guess" or "expect." As an example, an approximation formula for the second derivative $f''(x)$ that employs the function values $f(x-h)$, $f(x)$ und $f(x+h)$ can be derived from again expanding $f$ as

$$\begin{aligned}
f(x+h) &= f(x) + hf'(x) + \frac{h^2 f''(x)}{2} + \frac{h^3 f'''(x)}{6} + \mathcal{O}(h^4) \\
f(x-h) &= f(x) - hf'(x) + \frac{h^2 f''(x)}{2} - \frac{h^3 f'''(x)}{6} + \mathcal{O}(h^4)
\end{aligned}$$

and re-arranging this to the **standard 3-point stencil** with second order accuracy for $f''(x)$,

$$f''(x) = \frac{f(x+h)+f(x-h)-2f(x)}{h^2} + \mathcal{O}(h^2) = \frac{1}{h^2}\left[1 \;\; \underline{-2} \;\; 1\right] f(x) + \mathcal{O}(h^2)$$

## 1.4 Richardson's Extrapolation

A simple method to obtain a higher-order finite difference formula from one of lower order is Richardson's "extrapolation to the limit." It is a fairly general method for an Operator $D_h$ with error $\mathcal{O}(h^n)$, as long as the formula's coefficients are constant, which is the case if it's based on simple Taylor expansion. In this case, the leading error term can be written as $Ch^n$ with some usually *unknown* constant $C$. To give an example, for the right-sided difference $D_h^+$ to approximate $Df(x) = f'(x)$, this constant is $C = \frac{f''(x)}{2}$, and obviously $n = 1$.

Here's the trick: We write down

$$Df(x) = D_h f(x) + Ch^n + \mathcal{O}(h^{n+1})$$

without knowing $C$ itself. The crucial point is that the Taylor coefficient $C$ is *independent of* $h$. Now, apply the same operator again, using the enhanced width $2h$ instead of $h$ for differencing. This gives another discrete operator $D_{2h}$, and

$$Df(x) = D_{2h} f(x) + \underbrace{C(2h)^n}_{=2^n Ch^n} + \mathcal{O}(h^{n+1})$$

(remember: using the order-of notation, constant factors inside $\mathcal{O}()$ play no role, because the statement made is only about the *order* of convergence).

That's it! Eliminating the unknown constant $C$ yields

$$(2^n - 1)Df(x) = 2^n D_h f(x) - D_{2h} f(x) + \mathcal{O}(h^{n+1})$$

and we have the new operator

$$\tilde{D}_h := \frac{2^n D_h - D_{2h}}{2^n - 1}$$

as an $O(h^{n+1})$-consistent approximation to $D$!

Try this for the first order $D_h^+$ approximation to $Df = f'$ from above: Write down

$$D_h^+ f(x) = \frac{f(x+h) - f(x)}{h} = \frac{1}{h}\left[\underline{-1} \quad 1 \quad \right] f(x) = f'(x) + \mathcal{O}(h)$$

and, just replacing the differencing width by $2h$,

$$D_{2h}^+ f(x) = \frac{f(x+2h) - f(x)}{2h} = \frac{1}{2h}\left[\underline{-1} \quad 0 \quad 1\right] f(x) = f'(x) + \mathcal{O}(h)$$

Now, with these two $\mathcal{O}(h^1)$-operators, instead of doing a new messy Taylor expansion, we know that

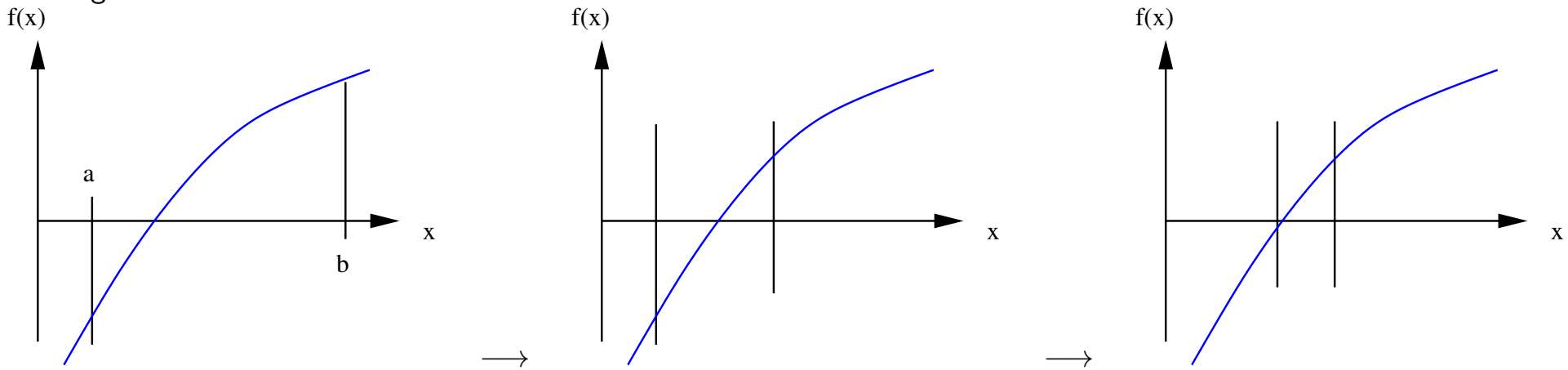
$$\tilde{D}_h^+ f(x) := \frac{2^1 D_h^+ - D_{2h}^+}{2^1 - 1} f(x) = \frac{1}{2h}\left[\underline{-3} \quad 4 \quad -1\right] f(x) = \frac{-f(x+2h) + 4f(x+h) - 3f(x)}{2h} = f'(x) + \mathcal{O}(h^2)$$

is a $\mathcal{O}(h^2)$-approximation to the first derivative $f'(x)$.

## 1.5 Root-Finding in One Dimension

We mention the most fundamental methods to find roots of a given function $f(x)$ of one independent variable $x$, i.e., the points/ values/ positions $x_\star$ with $f(x_\star) = 0$. To be more precise, we assume that $f$ is *continous* on a closed interval $I = [a, b]$, and we assume that $f(a)f(b) < 0$, which taken together implies that there exists (at least one) root $x_\star \in I$ of $f$. In practice, one often knows from additional considerations, or from careful specification of $I$, that there's actually only one unique root in $I$.
The methods to find (a) $x_\star$ are all *iterative* in nature:

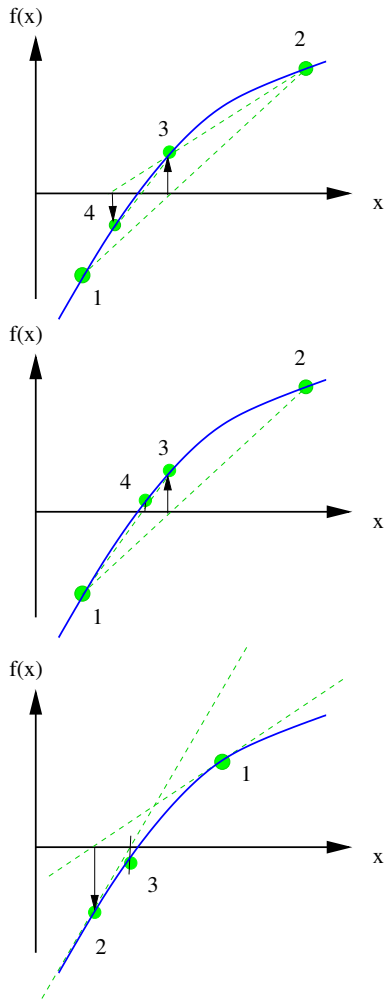
- **Bi-sectioning** is widely used, **simple**, and **robust**: Inspect $f$'s value in the interval's center, $f_M := f(\frac{a+b}{2})$, and decide according to its sign: If $f_M$ and $f(a)$ are of same sign, then there's a root in the right half of $I$, and we replace $a \leftarrow \frac{a+b}{2}$. If, on the other hand, $f_M$ and $f(b)$ have the same sign, the we replace $b \leftarrow \frac{a+b}{2}$. And if, by chance, $f_M$ is zero, then we're done!
  This recipe is used to recursively cut the interval into halves and thereby enclosing $x_\star$ by smaller and smaller intervals, until the uncertainty of the value of $x_\star$ is acceptable. Only the *signs* of $f$'s values in the respective interval centers are used in this method, not their magnitudes.



- The **secant method** follows a slightly different philosophy to construct a sequence $x_i$, with $i = 1, 2, 3, ...$, to approach a root. A new value $x_{i+1}$ of this sequence is computed as the root of the secant spanned by its two predecessors $x_i$ and $x_{i-1}$, starting with $x_1 = a$ und $x_2 = b$.
  As an advantage, the method converges faster than bi-sectioning once we are close to a root. However, convergence is not guaranteed when the $x_i$ are "far" away from any root, and the method can even leave the interval $[a, b]$.

- **Regula falsi** combines ideas of both previous methods:. $x_{i+1}$ gain is computed as a secant's root, but that secant is based on the two most recent $x_i$ that have $f$-values with opposite signs.

- An other simple algorithm that provides **rapid** quadratic convergence in the vicinity of roots is the **Newton-Raphson** method. It's based on the tangent slope of $f$ and presumes that the derivative $f'(x)$ can be determined analytically. Here, we iterate $x_i$ according to $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$, which of course presumes $f'(x_i) \neq 0$. This method can fail, for example if $f$ has a local extremum between the current $x_i$ and the closest root.




Secant method (top), Regula Falsi (center), Newton-Raphson method (bottom).

In practice, different methods can be combined for optimal performance. Often, a few bi-sectioning steps are used to find a narrow suitable interval $I$ first, followed by few Newton iterations to fixed the solution within that $I$ up to the desired accuracy.

## 1.6 Numerical Integration Using Interval Methods

Given a univariate function $f(x)$ again, this time we want to approximate its integral's value $A := \int_a^b f(x)dx$ with $a < b$. Needless to say that $f$ must be integrable, we assume that it's also smooth and hence can be approximated by Taylor polynomials so that we can perform an accuracy analysis similar to what we did before.

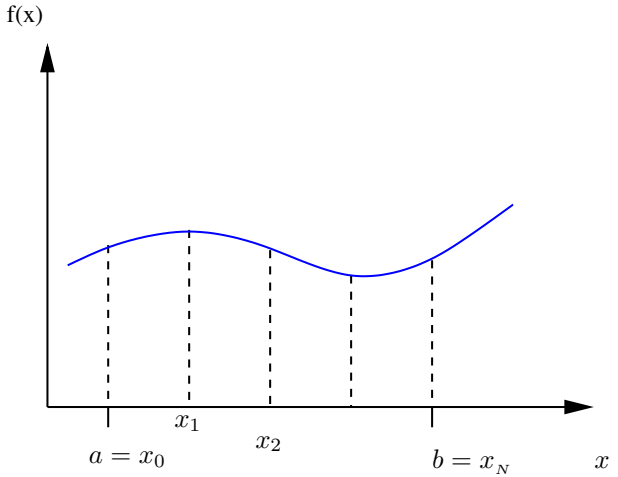
We divide $I := [a, b]$ into $N$ equally sized subintervals of width $h$ as

$$I_i = [x_i, x_{i+1}], \quad x_0 = a, \quad x_N = b, \quad x_{i+1} = x_i + h$$

and write $A$ as the (exact!) sum

$$A = \sum_{i=0}^{N-1} \int_{I_i} f(x)dx = \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} f(x)dx,$$

and now need a formula to integrate over any of the subintervals. We use the compact notation $f_i := f(x_i)$, $f_{i+1/2} := f(x_i + \frac{h}{2})$ etc.




Some preparatory work: In each subinterval, carry out a Taylor expansion of $f$ up to third order around the respective subinterval's center $x_{i+1/2} = x_i + \frac{h}{2}$ of $I_i$,

$$\begin{aligned} f(x) &= f_{i+1/2} + f'_{i+1/2} \cdot (x - x_{i+1/2}) + \frac{1}{2} f''_{i+1/2} \cdot (x - x_{i+1/2})^2 + \frac{1}{6} f^{(3)}_{i+1/2} \cdot (x - x_{i+1/2})^3 + \mathcal{O}\left((x - x_{i+1/2})^4\right) \\ &= f_{i+1/2} + f'_{i+1/2}\xi + \frac{1}{2} f''_{i+1/2}\xi^2 + \frac{1}{6} f^{(3)}_{i+1/2}\xi^3 + \mathcal{O}(\xi^4) \end{aligned}$$

where the substitution $\xi := x - x_{i+1/2}$ was introduced. Integrating this polynomial over $I_i$ yields

$$\int_{x_i}^{x_{i+1}} f(x)dx = \int_{-h/2}^{h/2} f(x_{i+1/2} + \xi)\, d\xi = hf_{i+1/2} + 0 + \frac{h^3}{24} f''_{i+1/2} + 0 + \mathcal{O}(h^5) = hf_{i+1/2} + \mathcal{O}(h^3)$$

which means that we have obtained a $\mathcal{O}(h^3)$ accurate approximation to the integral for each subinterval.

Taking the sum, we get

$$A = \sum_{i=0}^{N-1} h f_{i+1/2} + \mathcal{O}(h^2)$$

as a *second order* approximation of the desired entire integral $A$.

The reason why the *global* accuracy of $A$ is one order less than the *local* accuracy of integrating any subinterval is the following: We address the limit $N \to \infty$, or, according to $h = (b-a)/N$, the equivalent limit $h \to 0$. Now, with $N \to \infty$, the number of terms in the $A$-sum increases $\sim N$ while each term's error decreases as $\sim h^3 \sim 1/N^3$. Taking this together, the global error behaves like $\sim h^2 \sim 1/N^2$ only.

---

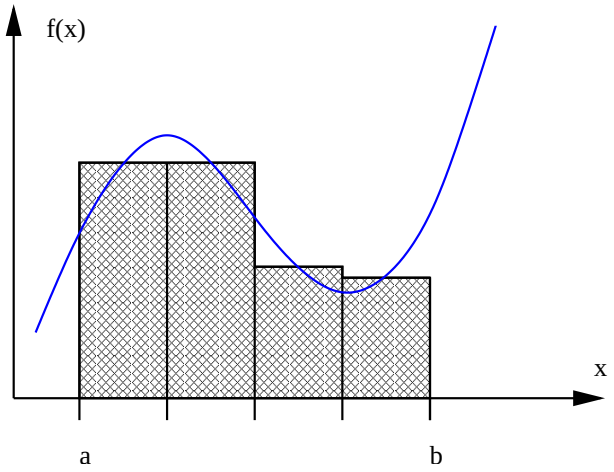
With these considerations, we arrive at the important **mipoint rule** of integration:

$$\begin{aligned} \int_a^b f(x)\,dx &= h \sum_{i=0}^{N-1} f_{i+1/2} + \underbrace{\mathcal{O}(h^2)}_{E_M} \\ &\approx h\left[f_{1/2} + f_{3/2} + f_{5/2} + ... + f_{N-1/2}\right] \end{aligned}$$

which incorpates $f$'s values in the **interval centers**.
We write down the (unknown) leading error term in more detail as (s. above)

$$E_M = \frac{h^3}{24} \sum_{i=0}^{N-1} f''_{i+1/2} + O(h^4)$$

fest. Again, the *local* error of the $\mathcal{O}(h^5)$-term is reduced to *global* $\mathcal{O}(h^4)$ in order.



In the same setup as above, the **trapezoidal rule** builds upon $f$-values at the intervals' boundaries, i.e., function values $f_i$ with integer indices $i$.

We start with approximating

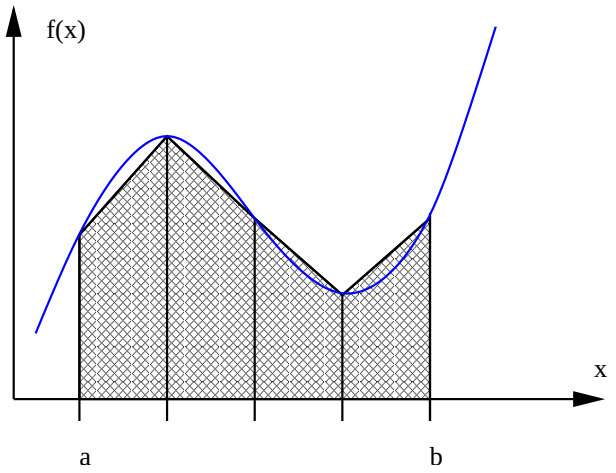
$$f_i + f_{i+1} = 2f_{i+1/2} + \frac{h^2}{4} f''_{i+1/2} + O(h^4)$$

and thus, for the midpoint values,

$$f_{i+1/2} = \frac{f_i + f_{i+1}}{2} - \frac{h^2}{8} f''_{i+1/2} + O(h^4)$$

Inserting this into the midpoint rule yields the **trapezoidal rule**

$$\begin{aligned} \int_a^b f(x)\, dx &= h\frac{f_0 + f_N}{2} + h\sum_{i=1}^{N-1} f_i + \underbrace{\mathcal{O}(h^2)}_{E_T} \\ &\approx h\left[\frac{f_0}{2} + f_1 + f_2 + ... + f_{N-1} + \frac{f_N}{2}\right] \end{aligned}$$




We can see that function values taken at the inner interval boundaries enter with weight $h$ into the sum, and the two external boundary values each carry the factor $\frac{h}{2}$. Again, we look at the error term in more detail:

$$E_T = -\frac{h^3}{12}\sum_{i=0}^{N-1} f''_{i+1/2} + O(h^4)$$

The results found for these two very fundamental integration rules are

- **Midpoint and trapezoidal** rule are **both second order** accurate (globally) with respect to $h \sim 1/N$.
- The leading error of the trapezoidal rule is, by magnitude, twice that of the midpoint rule!
- The midpoint rule is extremely simple to implement, as no calculations of $f$ are needed at the external boundaries $a$ und $b$, where $f$ could show, i.e., singular behavior.
- The trapezoidal rule is easily refined recursively from $N$ to $2N$ subintervals.
- In practice, the choice of which rule to use often follows from the working context, e.g. existing datasets or a previously existing discretization of $f$.

Another very powerful method on intervals is **Simpson's rule**. It assumes the number of intervals $N$ to be even, and is **fourth order** accurate:

$$\int_a^b f(x)\,dx = \frac{h}{3}\left[f_0 + 4f_1 + 2f_2 + 4f_3 + ... + 4f_{N-1} + f_N\right] + \mathcal{O}(h^4)$$

As for the trapezoidal rule, values of the integrand are evaluated at the interval boundaries, with those at internal boundaries entering with alternating weights.

# 2 Ordinary Differential Equations

Differential equations play a central role in physics. We will first focus on **ordinary differential equations (ODEs)**, which involve only one independent variable. One elementary example is Newton's equation with time $t$ as independent quantity and an expression $\vec{F}(\vec{r})$ as a force field, in which case the trajectories $\vec{r}(t)$ of particles typically are solutions to be determined.

Limiting the discussion further to **explicit** ODEs, we can formally write such an $k$-th order ODE with solution $f(t)$ as

$$f^{(k)}(t) = \frac{d^k f(t)}{dt^k} = G(f, f', ..., f^{(k-1)}, t)$$

The ODE itself is encoded in the expression $G$ that combines $f$ and its first $k-1$ derivatives $f'$, $f''$, ..., $f^{(k-1)}$ into a term equal to the $k$th derivative. In addition to $G$, **initial values** of $f..., f^{(k-1)}$ at some starting point $t_0$ might be specified in order to fix one particular function $f(t)$ as the (hopefully) unique solution to this initial value problem (IVP). Or, the **general solution** will contain $k$ degrees of freedom to allow for later specialization to initial conditions.

The $k$th order equation given above can be re-written as a **system** of $k$ first order ODEs by interpreting $f$'s derivatives as functions on their own, like

$$\frac{df(t)}{dt} = f'(t), \quad \frac{df'(t)}{dt} = f''(t), \quad \frac{df''(t)}{dt} = f^{(3)}(t), \quad \frac{df^{(k-2)}(t)}{dt} = f^{(k-1)}(t), \quad ..., \quad \frac{df^{(k-1)}(t)}{dt} = G(f, f', ..., f^{(k-1)}, t)$$

and again potentially specifying $k$ initial conditions for a particluar solution. With this consideration, we just have to address the solution of (systems of) *first order* ODEs.

The basic approach is to discretize with repsect to the independent variable $t$: We introduce a *step size* $\Delta t$ (i.e., a time step) and write function values of the sought solution $f$ at discrete times $t_n := t_0 + n\Delta t$ (assumed equidistant here) as $f_n := f(t_n) = f(t_0 + n\Delta t)$ with $n = 0, 1, 2, ....$ Hence, the task consists in the recursive calculation of the evolving $f_1, f_2, ...$ from a given initial value $f_0 = f(t_0)$ using the ODE.
Formally, we can write this as integrating $f'(t) = G(f, t)$ up to

$$f_{n+1} = f_n + \underbrace{\int_{t_n}^{t_{n+1}} G(f(t), t)\, dt}_{=:\, I}$$

where, in contrast to the integration of a given function as discussed in the previous section, the unknown itself now enters into the right hand side. Different approximation to the integral lead to different methods of (approximately) solving the ODE, we will discuss some of them.

## 2.1 "Foward-Euler"-Method

The algorithmically simplest way is to approximate the integral as $I = \Delta t\ G(f_n, t_n) + \mathcal{O}(\Delta t^2)$, where we easily use the "old" known value $f_n$ to evaluate the equation's RHS and add this increment, multiplied by $\Delta t$, to get the "new" value $f_{n+1}$. Omitting the discretization error term, we write this **forward-Euler** scheme as

$$f_{n+1} = f_n + \Delta t\ G(f_n, t_n)$$

understanding that this $f_{n+1}$ now is merely an approximation to the real $f(t_{n+1})$. It is an **explicit method** because the increment can be evaluated directly from the already known $f_n$ (at least when discarding potential complications in the function evaluation of $G$). The **local error** (i.e., per integration "time" step), is **second order** in $\Delta t$, but as with simple function integration, these local errors sum up to a **first order global error** with $\Delta t \to 0$, making the method $\mathcal{O}(\Delta t)$ globally.

As a concrete working example, we look at the probably simplest meaningful first-order ODE

$$\frac{df(t)}{dt} = -\gamma f(t)$$

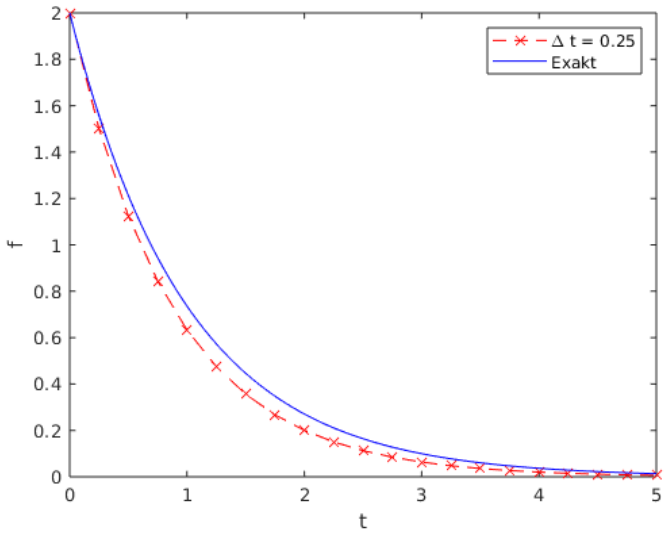
with constant $\gamma > 0$. It is explicit and linear, and in the formal notation $G(f,t) = -\gamma f$. This prototype equation can desribe numerous simple physical settings, like radioactive decay, mechanical friction, (de-)charging of an electrical capacitor, to name a few. The analytical solution for some $f(t_0) = f_0$ is well-known, and we use the equation only as a test case for our numerical schemes at this point.

Dicretizing the ODE with forward-Euler results in

$$f_{n+1} = f_n - \Delta t \gamma f_n$$

where each iteration step involves only as simple multiplication by an **iteration eigenvalue** $\lambda = (1 - \gamma\Delta t)$, owing to the ODE's linearity.
From this eigenvalue, we see immediately that

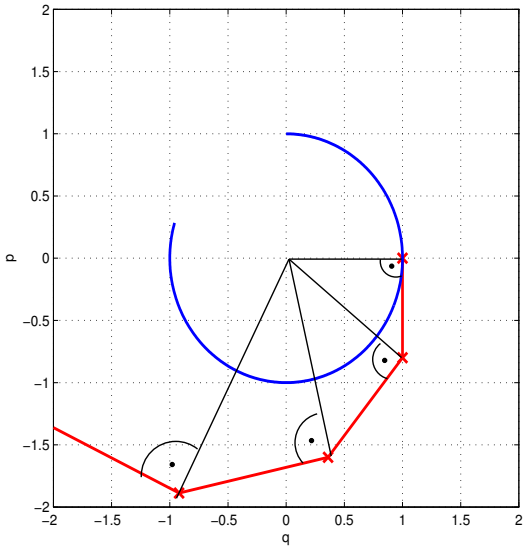
- $\lambda$ turns negative whenever $\gamma\Delta t > 1$ (artificial oscillations).
- the method is **unstable** when $\gamma\Delta t > 2$ because $|\lambda| > 1$ in that case.
- the method is **absolutely stable** when $\gamma\Delta t < 2$ because $|f_{n+1}/f_n| < 1$ then.



Given the experience from the first ODE example, we might get the expression that the forward-Euler method is appropriate for solving ODEs. This impression diminishes when we try our second favorite system, the harmonic oscillator.

Writing it down in symmetric form

$$\begin{aligned} q'(t) &= \omega p(t) \\ p'(t) &= -\omega q(t) \end{aligned}$$




as a system of two first-order ODEs for functions $q(t)$ and $p(t)$ (we could even combine them to a single complex-valued $z(t) = q(t) + ip(t)$ to get $z'(t) = -i\omega z(t)$).
Discretizing according to forward-Euler results in

$$\begin{aligned} q_{n+1} &= q_n + \omega \Delta t p_n \\ p_{n+1} &= p_n - \omega \Delta t q_n \end{aligned}$$

which can be written in matrix form as

$$\begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix} = \begin{pmatrix} q_n \\ p_n \end{pmatrix} + \omega \Delta t \begin{pmatrix} p_n \\ -q_n \end{pmatrix} = \begin{pmatrix} 1 & \omega \Delta t \\ -\omega \Delta t & 1 \end{pmatrix} \begin{pmatrix} q_n \\ p_n \end{pmatrix} = \underline{\underline{M}} \begin{pmatrix} q_n \\ p_n \end{pmatrix}$$

Looking at the graphical sketch, we see that this method will give us some headache: The exact solution $(q(t), p(t))$ moves along a circle in phase space, whereas the forward-Euler methods proceeds by following the tangent at $(q_n, p_n)$ which makes the numerical approximation to spiral outward and artificaclly lead to a continous increase of the "energy" $(q^2 + p^2)/2$.

This can also be deduced by addressing the iteration matrix $\underline{\underline{M}}$: Its eigenvalues are $\lambda_{1,2} = 1 \pm i\omega\Delta t$ and indicate, because of $|\lambda_{1,2}| > 1$ **unconditional instability** of the method for this ODE system.

## 2.2 "Backward-Euler"-Method

The term "backward-Euler" might be somewhat misleading: We still iterate like $f_n \to f_{n+1} \to f_{n+2} \to ...$, but this time, we evaluate the RHS increment from $G(f_{n+1}, t_{n+1})$, i.e. by using the "new" values. This is equivalent to solving the time-reversed equation by forward-Euler, hence the name.

Writing down

$$f_{n+1} = f_n + \Delta t G(f_{n+1}, t_{n+1})$$

we see that the method will, in general, be **implicit**: The new $f_{n+1}$ that we're out for is used to compute itself, so that we need some means to find this $f_{n+1}$ from its implicit definition.

As with forward-Euler, this method is obviously also $\mathcal{O}(\Delta t)$ globally, but as a rule-of-thumb, implicit schemes give **better stability properties** as compared to explicit schemes, so that the additional effort so solve the iteration rule for $f_{n+1}$, usually numerically, might pay off.

Try this again for the linear decay equation $\frac{df(t)}{dt} = -\gamma f(t)$: We find the rule $f_{n+1} = f_n - \Delta t \gamma f_{n+1}$, which can be rearranged into

$$f_{n+1} = \frac{f_n}{1 + \Delta t \gamma}$$

The iteration eigenvalue is $\lambda = 1/(1 + \Delta t \gamma) < 1$ (assuming $\gamma > 0$) which indicates **unconditional stability** for this system.

For the oscillator $q'(t) = \omega p(t), p'(t) = -\omega q(t)$, we get

$$\begin{aligned} q_{n+1} &= q_n + \omega \Delta t p_{n+1} \\ p_{n+1} &= p_n - \omega \Delta t q_{n+1} \end{aligned}$$

or, in matrix notation,

$$\begin{pmatrix} 1 & -\omega\Delta t \\ \omega\Delta t & 1 \end{pmatrix} \begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix} = \begin{pmatrix} q_n \\ p_n \end{pmatrix}$$

It's easy to invert this 2x2 matrix to get

$$\begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix} = \underbrace{\frac{1}{1+\omega^2\Delta t^2}\begin{pmatrix} 1 & \omega\Delta t \\ -\omega\Delta t & 1 \end{pmatrix}}_{\underline{\underline{M}}} \begin{pmatrix} q_n \\ p_n \end{pmatrix}$$

with eigenvalues

$$\lambda_{1,2} = \frac{1 \pm i\omega\Delta t}{1+\omega^2\Delta t^2}$$

Again, we find $|\lambda_{1,2}| < 1$, so that the backward iteration is **unconditionally stable** even for this system.

---

The fact that we could (explicitly) solve for $f_{n+1}$ in these examples, although we started with an implicit discretization formula, goes back to the linearity of the ODEs. This is fine for the elementary considerations made here, however, we will typically need a computer to solve non-linear ODEs. Then, numerical methods to find $f_{n+1}$ will be necessary.

## 2.3 $\vartheta$-, Trapezoidal-, Crank-Nicholson-Method

While the backward discretization gives some stability improvement over forward-Euler, it's still only first order accurate, and also results in considerable artificial dissipation or damping.

We can mix the forward- and backward approaches in any relation to each other by introducing a numerical parameter $0 \leq \vartheta \leq 1$ and write

$$f_{n+1} = f_n + \Delta t \left[\vartheta G(f_n, t_n) + (1-\vartheta) G(f_{n+1}, t_{n+1})\right]$$

which gives us a continous transition between

- $\vartheta = 1 \Rightarrow$ forward-Euler method
- $\vartheta = 0 \Rightarrow$ backward-Euler method
- $\vartheta = \frac{1}{2} \Rightarrow$ trapezoidal method

The last one is new: Here, we use the **trapezoidal** rule to approximate the integral $\int_{t_n}^{t_{n+1}} G(f(t), t)\, dt$ from above. As a consequence, this time-symmetric formula gives us **improved** $\mathcal{O}(\Delta t^2)$ **accuracy** as compared to the first-order accuracy with all other values of $\vartheta$.

Applying the trapezoidal method to the harmonic oscillator, we again start writing

$$q_{n+1} = q_n + \frac{\omega \Delta t}{2}(p_n + p_{n+1})$$
$$p_{n+1} = p_n - \frac{\omega \Delta t}{2}(q_n + q_{n+1})$$

or

$$\begin{pmatrix} 1 & -\frac{\omega\Delta t}{2} \\ \frac{\omega\Delta t}{2} & 1 \end{pmatrix} \begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & \frac{\omega\Delta t}{2} \\ -\frac{\omega\Delta t}{2} & 1 \end{pmatrix} \begin{pmatrix} q_n \\ p_n \end{pmatrix}$$

Inverting the left matrix and rearranging for the "new" state results in

$$\begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix} = \underbrace{\frac{1}{1+\left(\frac{\omega\Delta t}{2}\right)^2}\begin{pmatrix} 1-\left(\frac{\omega\Delta t}{2}\right)^2 & \omega\Delta t \\ -\omega\Delta t & 1-\left(\frac{\omega\Delta t}{2}\right)^2 \end{pmatrix}}_{\underline{\underline{M}}} \begin{pmatrix} q_n \\ p_n \end{pmatrix}$$

form which $\underline{\underline{M}}$'s eigenvalues follow as

$$\lambda_{1,2} = \frac{\left(1 \pm i\frac{\omega\Delta t}{2}\right)^2}{1+\left(\frac{\omega\Delta t}{2}\right)^2} = \frac{1 \pm i\frac{\omega\Delta t}{2}}{1 \mp i\frac{\omega\Delta t}{2}}$$

We see that $|\lambda_{1,2}| = 1$ for all $\Delta t$, hence the method is unconditionally stable with neither amplification nor dissipation of the solution $(q, p)$.

---

Another second-order, implicit method that is widely used in physics is the **Crank-Nicholson method**, where $G$ is evaluated using the average of $f_n$ and $f_{n+1}$,

$$f_{n+1} = f_n + \Delta t G\left(\frac{f_n + f_{n+1}}{2}, \frac{t_n + t_{n+1}}{2}\right)$$

It is equivalent to the trapezoidal method for linear problems (as the examples addressed above), and while the two differ from each other for the case of non-linear $G$, they usually have similar stability properties.

## 2.4 Multilevel and Multistage Methods

Forward and backward Euler methods as discussed above are only $\mathcal{O}(\Delta t)$ accurate, which in practice is often inacceptable. Yet, they are often used as building blocks for the creation of higher-order linear ODE solvers (not to be confused with solvers for linear ODEs!), and a zoo of such methods exists, of which a few are only considered below:

- **Multi-level methods** make use of "older" function values $f_{n-1}, f_{n-2,\ldots}$ in addition to the "current" $f_n$ in order to compute the "new" $f_{n+1}$. A often used class of such methods are the **explicit**, and hence easily implemented, **Adams-Bashforth methods**,

$$\begin{aligned}
\mathcal{O}(\Delta t): \quad f_{n+1} &= f_n + \Delta t G(f_n, t_n) \quad \text{(identical to forward-Euler)} \\
\mathcal{O}(\Delta t^2): \quad f_{n+1} &= f_n + \Delta t \left[\frac{3}{2} G(f_n, t_n) - \frac{1}{2} G(f_{n-1}, t_{n-1})\right] \\
\mathcal{O}(\Delta t^3): \quad f_{n+1} &= f_n + \Delta t \left[\frac{23}{12} G(f_n, t_n) - \frac{4}{3} G(f_{n-1}, t_{n-1}) + \frac{5}{12} G(f_{n-2}, t_{n-2})\right] \\
&\vdots
\end{aligned}$$

  The implicit counterparts, which again typically have better stability properties and therefore allow for a larger time step $\Delta t$ on the cost of more intricate implementation, are the **Adams-Moulton methods**,

$$\begin{aligned}
\mathcal{O}(\Delta t): \quad f_{n+1} &= f_n + \Delta t \left[\frac{1}{2} G(f_{n+1}, t_{n+1}) + \frac{1}{2} G(f_n, t_n)\right] \quad \text{(identical to trapezoidal integration)} \\
\mathcal{O}(\Delta t^2): \quad f_{n+1} &= f_n + \Delta t \left[\frac{5}{12} G(f_{n+1}, t_{n+1}) + \frac{2}{3} G(f_n, t_n) - \frac{1}{12} G(f_{n-1}, t_{n-1})\right] \\
\mathcal{O}(\Delta t^3): \quad f_{n+1} &= f_n + \Delta t \left[\frac{9}{24} G(f_{n+1}, t_{n+1}) + \frac{19}{24} G(f_n, t_n) - \frac{5}{24} G(f_{n-1}, t_{n-1}) + \frac{1}{24} G(f_{n-2}, t_{n-2})\right] \\
&\vdots
\end{aligned}$$

Both families have the advantage that the computed $G_n$ can be re-used in subsequent steps, which can potentially save computing time, in particlular if the computation of $G$ involves costly operations, but require the intermediate storage of the old $G$-values. On the other hand, their use requires all integration steps to be carried out with the same $\Delta t$, and furthermore, the startup to create the first data levels must be done with another method.

- **Multi-stage methods** still follow the old philosophy of creating all levels $f_n$ independently of each other, allowing for varying $\Delta t$ and easy startup. However, each **step** is split into several **stages**, each of which usually involve one $G$-computation, and these stages are designed to merge into the desired $f_{n+1}$. The most commonly used class here are the **Runge-Kutta methods**, which again come in a variety of specific realisations. As with the Adams methods, they result from Taylor expansion of $f(t)$ and careful combination to obtain the desired accuracy.

We demonstrate the design of such methods for the explicit second-order Runge-Kutta methods and start with the ODE

$$\frac{df}{dt} = G(f(t), t)$$

and its derivative wrt. $t$,

$$\frac{d^2 f}{dt^2} = \frac{d}{dt} G(f(t), t) = \frac{\partial G}{\partial t} + \frac{\partial G}{\partial f} \underbrace{\frac{df}{dt}}_{=G} = G_t + G G_f$$

Indices in the symbols $G_t$ and $G_f$ abbreviate the partial derivatives wrt. $t$ and $f$, respectively, and all expressions are understood to be taken at (unspecified) values $t$ und $f(t)$. The third derivative follows as (according to product und chain rules)

$$\frac{d^3 f}{dt^3} = G_{tt} + G_{tf} G + G_t G_f + G G_f^2 + G G_{tf} + G^2 G_{ff}$$

Next, we write down Taylor expansion of $f$ wrt. $t_n$ in its generic form as

$$f(t_n + \Delta t) = f_n + \Delta t G + \frac{\Delta t^2}{2}(G_t + G G_f) + \mathcal{O}(\Delta t^3)$$

with $G, G_t, G_f$ understood to be evaluated using the current $t_n, f_n$. Now, the Runge-Kutta design is based on the idea to use separate incremets $k_i$ to be added to $f_n$, and for the second order explicit schemes, the two increments are supposed to be of the form

$$\begin{aligned} k_1 &= \Delta t G(f_n, t_n) \\ k_2 &= \Delta t G(f_n + \nu_{21} k_1, t_n + \nu_{21}\Delta t) \end{aligned}$$

with some numerical parameter $\nu_{21}$. The update $f_n \to f_{n+1}$ shall be

$$f_{n+1} = f_n + \alpha_1 k_1 + \alpha_2 k_2$$

again with numerical parameters $\alpha_1$ and $\alpha_2$. This is the design ansatz for the methods. Expanding once more, we can re-write

$$k_2 = \Delta t G(f_n + \nu_{21} k_1, t_n + \nu_{21}\Delta t) = \Delta t G(f_n, t_n) + \nu_{21}\Delta t^2 \left(G_t + G_f G\right) + \mathcal{O}(\Delta t^3)$$

where all expressions are again taken at $t_n$ and $f_n$.

Requiring $\mathcal{O}(\Delta t^3)$ accuracy means that the generic expansion of $f(t_n + \Delta t)$ and the designed $f_{n+1}$ must coincide up to second order, i.e., $f_{n+1} = f(t_n + \Delta t) + \mathcal{O}(\Delta t^3)$, which, comparing coefficients, yields the conditions

$$\alpha_1 + \alpha_2 = 1, \quad \alpha_2 \nu_{21} = \frac{1}{2}$$

which couple the three yet unspecified parameters. With three parameters and two conditions, we obtain a single-parametrized family of second-order Runge-Kutta schemes.

Special choices often used are

- $\alpha_1 = \alpha_2 = \frac{1}{2}$ and $\nu_{21} = 1$: **Heun**'s method, which, when written out, is

$$\begin{aligned} k_1 &= \Delta t G(f_n, t_n) \quad \text{(forward-Euler as first stage)} \\ k_2 &= \Delta t G(f_n + k_1, t_n + \Delta t) \quad \text{(second stage)} \\ f_{n+1} &= f_n + \frac{k_1 + k_2}{2} \quad \text{(update)} \end{aligned}$$

- $\alpha_1 = 0, \alpha_2 = 1$ and $\nu_{21} = \frac{1}{2}$, resulting in

$$k_1 = \Delta t G(f_n, t_n), \quad k_2 = \Delta t G\left(f_n + \frac{k_1}{2}, t_n + \frac{\Delta t}{2}\right), \quad f_{n+1} = f_n + k_2$$

The derivation is systematically generalized to higher orders, where the $l$-order methods involve the computation of $l$ increments in the subsequent stages. The results are very universal self-starting methods that are widely used. They are studied in applied mathematics and also implemented in many software packages, where additional features like adaptive time step control and error monitoring are included.

Originally proposed by Runge and Kutta is a fourth order method like

$$k_1 = \Delta t G(f_n, t), \quad k_2 = \Delta t G\left(f_n + \frac{k_1}{2}, t + \frac{\Delta t}{2}\right), \quad k_3 = \Delta t G\left(f_n + \frac{k_2}{2}, t + \frac{\Delta t}{2}\right), \quad k_4 = \Delta t G(f_n + k_3, t + \Delta t)$$

$$f_{n+1} = f_n + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}$$

which, while being explicit, also provides good stability for many problems.

## 2.5 Leapfrog and Verlet Method

In contrast to the very general methods from above, the so-called symplectic methods are tuned specifically to Hamiltonian systems, encountered in e.g. celestial or molecular dynamics.

A short recapitulation: A Hamiltonian $H$, formulated in canonical pairs of variables $q_i(t)$ and $p_i(t)$, generates the equations of motion according to

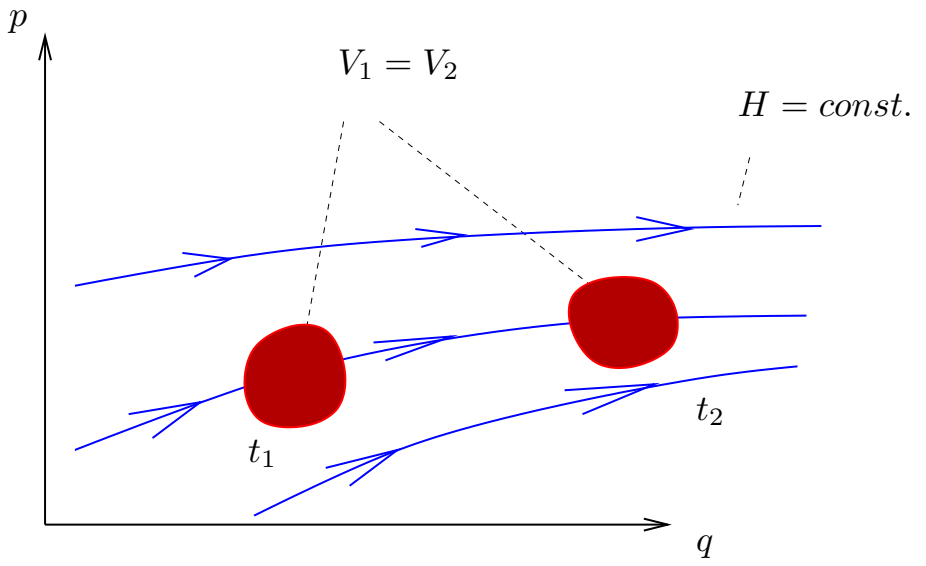
$$\dot{q}_i = \frac{\partial H(q_i, p_i, t)}{\partial p_i}, \quad \dot{p}_i = -\frac{\partial H(q_i, p_i, t)}{\partial q_i}$$

As an exmaple, the harmonic oscillator with one degree of freedom results from $H = \frac{\omega}{2}(q^2 + p^2)$.

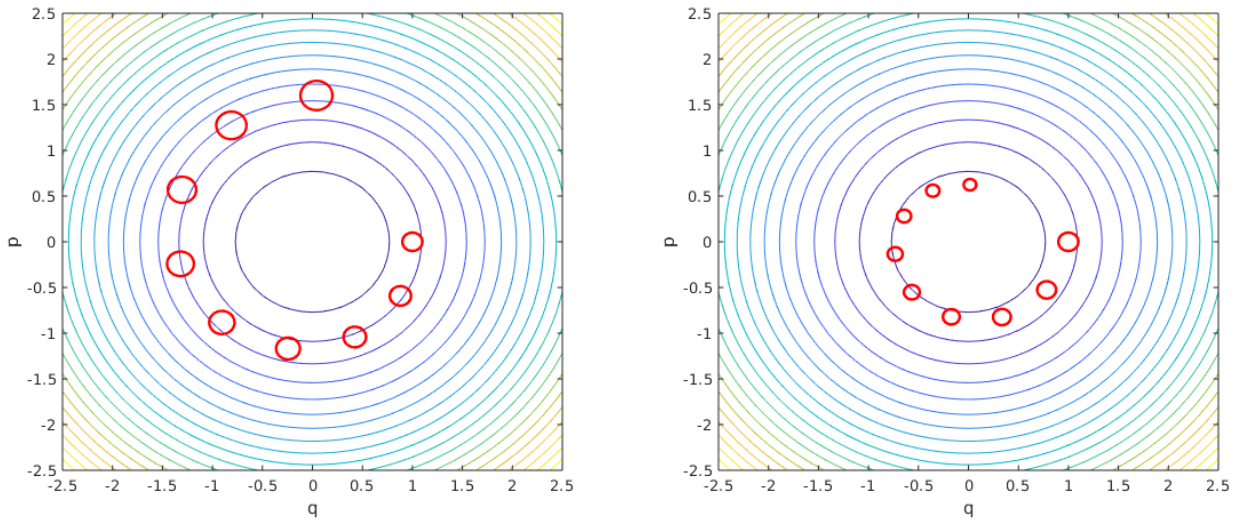
An important geometrical consequence of these laws is that the corresponding phase flow $\vec{v} := \sum_i (\dot{q}_i \vec{e}_{q_i} + \dot{p}_i \vec{e}_{p_i})$ is divergence-free or *incompressibel*, which can be seen from

$$\nabla_{(q_i,p_i)} \cdot \vec{v} = \sum_i \left( \frac{\partial \dot{q}_i}{\partial q_i} + \frac{\partial \dot{p}_i}{\partial p_i} \right) = 0$$

This means that an ensemble of systems that at some time $t_1$ occupy a volume of size $V_1$ in phase space will be transported in this "incompressible" flow and later, at $t_2 > t_1$ say, occupy the *same* volume size $V_2 = V_1$: Phase space volume is conserved under Hamiltonian dynamics (Liouville's theorem), although the volume itself may get distorted with time. Further, for any $H$ that does *not explicitly* depend on time, points in phase space will stay on energy shells, i.e., equimanifolds of $H$, because $H$ is conserved in this case.



Let's come back to the simple harmonic oscillator: We know that system points travel on circles in phase space with angular velocity $\omega$ – these are just the oscillations in $q$ und $p$. What happens if we integrate the system with Euler's method? Let's test:




The pictures show the integration of $\dot{q} = \omega p$, $\dot{p} = -\omega q$ using foward (left) and backward (right) Euler methods, respectively, using $t \in [0, 5]$, $\omega = 1$, and $\Delta t = 0.2$ and an ensemble of systems starting around $q_0 = 1, p_0 = 0$.

We observed this before: Forward-Euler blows up the solution, oscillations are amplified, energy grows, all artificially. And: The phase space volume increases as well!. Backward-Euler show exactly opposite behavior, again artificially: Oscillations get damped, losing energy, and phase space volume shrinks.

The simple idea: We create a method that exactly conserves the phase space volume! Ideally, this will tie the systems to the correct energy shell, because the volume can neither expand nor collapse.

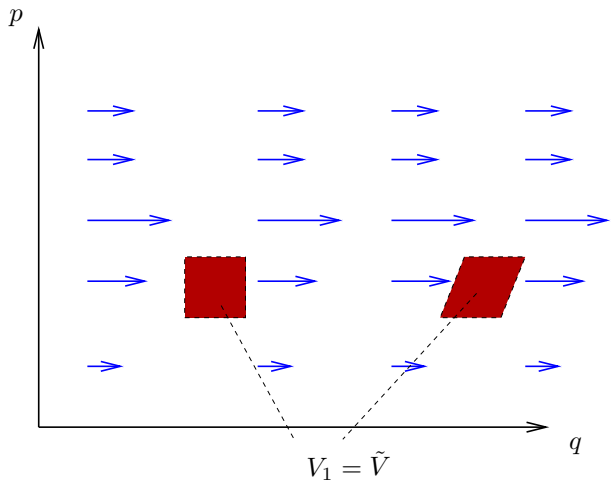
Sounds like a difficult programming task, but it's actually fairly simple. At least, if $H$ separates into a kinetic part (depending only on the $p_i$) and a potential part, like

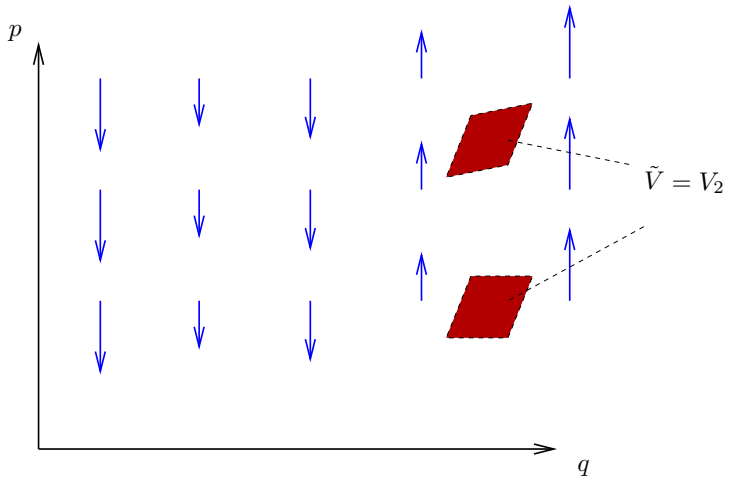
$$H(q_i, p_i) = T(p_i) + V(q_i)$$

which is often the case. Then, $\frac{\partial}{\partial q_i}\dot{q}_i = \frac{\partial}{\partial q_i}\frac{\partial H}{\partial p_i} = 0$ and $\frac{\partial}{\partial p_i}\dot{p}_i = -\frac{\partial}{\partial p_i}\frac{\partial H}{\partial q_i} = 0$, which means that the "velocity" component $\dot{q}_i$, at which phase space points are travelling in $q_i$-direction is *independent* of $q_i$ itself, and analogously the same for the $p_i$.

All that we have to do is to apply the update $t_n \to t_{n+1}$ for the $q_i$ and $p_i$ *after each other*!

Why does this work? Let's have a look at the evolution:




In the first step on the left, only the $q_i$ are updated, so that phase space points move along the $q_i$-directions, this is the **drift**. Important: The "velocity" doesn't depend on the $q_i$ themselves, the volume gets sheared but neither compressed nor elongated. It's content remains unchained. The same in $p_i$-directions – that's the **kick** because momenta change. Again, the volume is sheared, its size unchanged.

Let's write this down for one degree of freedom:

$$q_{n+1} = q_n + \Delta t \left(\frac{\partial H}{\partial p}\right)_{p=p_n} \quad \to \quad p_{n+1} = p_n - \Delta t \left(\frac{\partial H}{\partial q}\right)_{q=q_{n+1}}$$

The important detail is the fact that the kick already uses the *new* $q_{n+1}$! Matlab-code for this example might be like

```
q = q + dt*omega*p;
p = p - dt*omega*q;
```

Looks like forward-Euler, but it's not!

Obviously, the order of drift and kick can be changed, doing "kick-drift" instead of "drift-kick." In both cases, the accuracy is $\mathcal{O}(\Delta t)$. A simple way to achieve second order accuracy is, once more, to do a symmetric update, e.g. "1/2-drift → kick → 1/2-drift":

$$q_{n+1/2} = q_n + \frac{\Delta t}{2}\left(\frac{\partial H}{\partial p}\right)_{p=p_n} \quad \rightarrow \quad p_{n+1} = p_n - \Delta t\left(\frac{\partial H}{\partial q}\right)_{q=q_{n+1/2}} \quad \rightarrow \quad q_{n+1} = q_{n+1/2} + \frac{\Delta t}{2}\left(\frac{\partial H}{\partial p}\right)_{p=p_{n+1}}$$

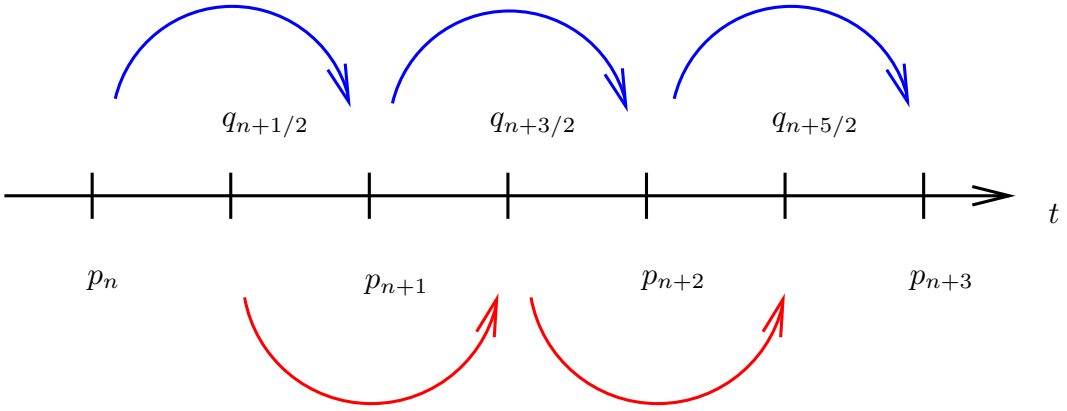
When continuing this scheme to the following $t_{n+2}$, the next semi-drift would be

$$q_{n+3/2} = q_{n+1} + \frac{\Delta t}{2}\left(\frac{\partial H}{\partial p}\right)_{p=p_{n+1}}$$

However, this can be combined with the previous semidrift, as they both involve the same $p_{n+1}$,

$$q_{n+3/2} = q_{n+1/2} + \Delta t\left(\frac{\partial H}{\partial p}\right)_{p=p_{n+1}}$$

so that we fall into a cycle in which the $q_i$ are arranged at "half" time levels $t_{i+1/2}$, and the conjugate $p_i$ at "integer" time levels $t_i$ (or vice versa), while all steps are carried out with "full" $\Delta t$. Written this way, it's called the **leapfrog** scheme (red: drifts, blue: kicks):




- usually good stability
- $\mathcal{O}(\Delta t^2)$ accurate
- explicit, easy to implement
- specific to Hamiltonian systems
- $q$ and $p$ are staggered in time, this must be taken into account at start/ stop/ diagnostics

## 2.6 Verlet formulation

Many applications come with equations of motions in the second-order Newtonian form

$$m\ddot{x} = F(x,t)$$

with purely space-dependent forces $F$. Examples are celestial and molecular dynamics. In this case, the leapfrog scheme looks like

$$\begin{aligned} x_n &= x_{n-1} + \Delta t\; v_{n-1/2} \\ v_{n+1/2} &= v_{n-1/2} + \Delta t\; \frac{F(x_n,t_n)}{m} \\ x_{n+1} &= x_n + \Delta t\; v_{n+1/2} \end{aligned}$$

with positions $x$ now arranged at integer time indices $t_n$.

Eliminating the velocities with their half-integer indices, we arrive at a leapfrog-formulation that goes under the widely-known name **Verlet method**,

$$x_{n+1} = 2x_n - x_{n-1} + \Delta t^2 \frac{F(x_n,t_n)}{m}$$

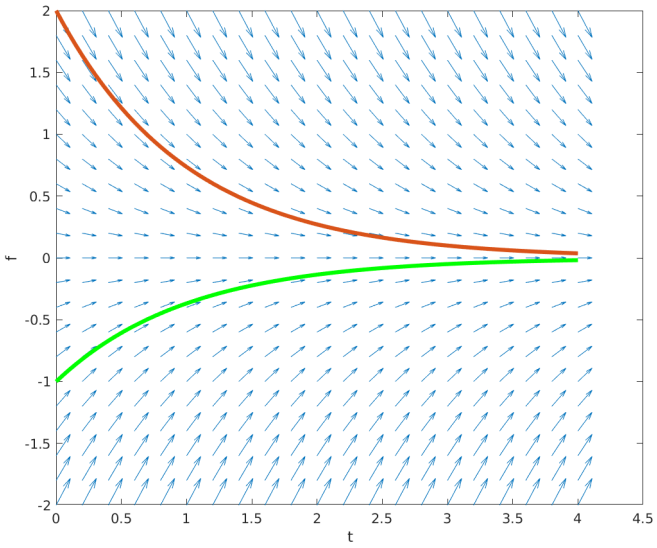
and which, of course, still shares the characterstics from the explicit leapfrog and drift-kick-formulations. Now, we're left with a **two-level** scheme for the $x_n$ only, so that the same complications wrt. startup as in the previous multilevel schemes come into play. An alternative interpretation of Verlet's method is the direct discretization of the second derivative $\ddot{x}$ by meand of the standard three-point finite difference stencil,

$$\frac{d^2x}{dt^2} = \frac{F(x,t)}{m} = \frac{x_{n+1} - 2x_n + x_{n-1}}{\Delta t^2} + O(\Delta t^2)$$

## 2.7 Additional Remarks on ODE-Solvers

The discussion so far has been fairly optimistic: Given an ODE and initial values, we devised methods to compute the solution $f(t)$ on a given interval in $t$, tacitly assuming that such a solution actually exists. Moreover, inspired from experience with physical modeling, we assume that solution to be unique. Although this approach proves successful for many applications, there might be cases, even in physics, where those presumptions are not met. From mathematics, we know that a unique solution is guaranteed to exists in a finite interval around the initial value if the ODE fulfills a **Lipschitz condition**.

The picture to the right displays the portrait of the (linear) decay equation $f'(t) = -f$ (arrows), and two particular solutions with initial values $f(0) = 2$ and $f(0) = -1$ are plotted in red an green, respectively. For each initial value, this ODE provides a unique global solution for $t \in \mathbb{R}$, this is the "best case" scenario.




Two standard examples that show different behavior wrt. their solutions are the following:

- The ODE $f'(t) = f^2$ with initial value $f(t_0) = f_0$ is solved by $f(t) = \frac{f_0}{1-(t-t_0)f_0}$. While being unique, the solution becomes singular at $t_\star = t_0 + \frac{1}{f_0}$ whenever $f_0 > 0$, so that the solution only exists in a finite interval around $t_0$.
- Equation $f'(x) = \sqrt{f}$ doesn't fulfill the Lipschitz condition at $f = 0$. Given the initial value $f(0) = 0$, it has the solution $f(t) = x^2/4$. However, the solution is not unique!

So, while the *differential equation* might look innocent, its *solution* can give us trouble!

Assuming that the ODE itself is well-behaved, i.e. has a unique solution, we want to address the **error** and **stability** properties of a given numerical scheme. First, the **discretization error** shall be defined more specifically: We expand the (generally unknown) exact solution of $\frac{df}{dt} = G(f,t)$ around $t_n$ as

$$f(t_{n+1}) = f(t_n) + \Delta t G\left(f(t_n), t_n\right) + \frac{1}{2}\Delta t^2 f''(t_n) + \mathcal{O}(\Delta t^3)$$

and then compare this expression with the approximation $f_{n+1}$ that we get from our numerical method under consideration, using $f_n = f(t_n)$ as initial data. The forward-Euler method, as an example, results in a **local truncation error (LTE)** of

$$L_n := f(t_{n+1}) - f_{n+1} = f(t_n) + \Delta t G\left(f(t_n), t_n\right) + \frac{1}{2}\Delta t^2 f''(t_n) + \mathcal{O}(\Delta t^3) - \underbrace{\left(f_n + \Delta t G\left(f(t_n), t_n\right)\right)}_{f_{n+1}} = \frac{1}{2}\Delta t^2 f''(t_n) + \mathcal{O}(\Delta t^3)$$

from which we see that $L_n = \mathcal{O}(\Delta t^2)$. The method is **consistent** with a **global error** $\mathcal{O}(\Delta t)$, if we advance the solution to a given terminal time/ position $t_E$, in analogy to the integration of functions.

However, there's a problem here: In function integration, the integrand $f(t_n)$, and resulting error coefficients like $f''(t_n)$, are given beforehand, so that the error is limited and can be estimated in each subinterval $[t_n, t_{n+1}]$, independently to result in an estimate for the global error by just summing the individual contributions. In the context of ODE integration, the initial data for the next interval results from the previous integration step, and we have to ask, how **errors propagate** from one step to the next, and if it's possible at all to compute the solution with any desired tolerance.

A solution method is called **convergent**, if, for any $t_E > t_0$, the approximation $f_N$ that is calculated by making $N$ steps of size $\Delta t = (t_E - t_0)/N$, converges to the "correct" value $f(t_E)$,

$$\lim_{\Delta t \to 0} f_N = f(t_E)$$

A necessary condition is that the method must be *consistent* (see above), but it also has to be **zero-stable**. Essentially, this means that it must be possible to limit the *global error* in terms of the *local error* in order to carry the consistency property, i.e. an estimate of the local error, over onto the global error.

The stability property (and hence the convergence) will in principle depend on the method *and the differential equation* under consideration, so that a method might be convergent and usable for one equation, but not for another one.

We use the equation

$$f'(t) = \lambda f(t) + g(t)$$

with a constant $\lambda$, an inhomogeneity $g$, and an initial value $f_0 = f(t_0)$ as an example: Discretizing with forward-Euler gives us

$$f_{n+1} = (1 + \lambda\Delta t) f_n + \Delta t\, g(t_n),$$

while the exact solution is still

$$f(t_{n+1}) = (1 + \lambda\Delta t) f(t_n) + \Delta t\, g(t_n) + L_n.$$

In order to quantify the error propagation, we define for every $n$ the difference between exact and approximate solution as the **error** $E_n := f_n - f(t_n)$ to obtain, by taking the difference of the above expressions,

$$E_{n+1} = (1 + \lambda\Delta t) E_n - L_n$$

This means that the error will be multiplied by $(1 + \lambda\Delta t)$ in each step, and in addition will be enhanced by the local error $-L_n$ hinzukommt. In recursion, this gives

$$E_n = (1 + \lambda\Delta t)^n E_0 - \sum_{m=1}^{n} (1 + \lambda\Delta t)^{n-m} L_{m-1}$$

With $|1 + \lambda\Delta t| \leq e^{|\lambda|\Delta t}$ and hence $(1 + \lambda\Delta t)^{n-m} \leq e^{(n-m)|\lambda|\Delta t} \leq e^{n|\lambda|\Delta t} = e^{|\lambda|(t_E - t_0)}$ we can make an estimate like

$$|E_n| \leq e^{|\lambda|(t_E-t_0)} \left( |E_0| + \sum_{m=1}^{n} |L_{m-1}| \right) \leq e^{|\lambda|(t_E-t_0)} \left( |E_0| + n \max_{1\leq m\leq n} |L_{m-1}| \right) \leq e^{|\lambda|(t_E-t_0)} \left( |E_0| + N \|L\|_\infty \right)$$

where $\|L\|_\infty = \max_{1\leq m\leq N} |L_{m-1}| = \mathcal{O}(\Delta t^2)$ denotes the largest local truncation error in the interval $[t_0, t_E]$.

Assuming that we start with the "exact" initial value, i.e., $E_0 = 0$, and realizing that $\lambda, t_0, t_E$ are all constant, and that $N = (t_E - t_0)/\Delta t$, we have shown that: *For any* $n$, a local truncation error $L_n = \mathcal{O}(\Delta t^2)$ results in a *global* error $E_n = \mathcal{O}(\Delta t)$. Hence, if we only choose $\Delta t$ small enough, we can approximate any $f(t_n)$, and in particular $f(t_E)$, to any desired accuracy.

Every **single step method** provides **zero-stability**, which allows us to limit the global error by the LTE, and hence deduce **convergence** from **consistency**. For the case of systems of ODEs, the analysis follows similar lines based on the iteration matrix' eigenvalues, supplemented with Lipschitz estimates for non-linear systems.

To give an example for in **unstable method**, and to understand the naming "zero-stability," let's consider the linear multi-level method

$$f_{n+2} - 3f_{n+1} + 2f_n = -\Delta t G(f_n, t_n)$$

to compute $f_{n+2}$ from $f_{n+1}$ and $f_n$ (we renamed the indices for convenience).

After Taylor expansion of the exact solution's values $f(t_{n+2})$ and $f(t_{n+1})$, and insertion into the scheme, we see that also this method is a globally $\mathcal{O}(\Delta t)$-**consistent** discretization of $f' = G(f,t)$ with a local truncation error $L_n = \frac{1}{2}\Delta t^2 f''$. However, the scheme is obviously **zero-unstable**: Considering the trivial IVP

$$f'(t) = 0, \quad f(0) = 0$$

(that is, $G = 0$), we arrive at the iteration

$$f_{n+2} = 3f_{n+1} - 2f_n$$

and with initial values $f_0 = 0$ and $f_1 = 0$ (we need two of them because it's a two-level scheme), we actually find the correct solution $f(t) = 0$. However: If an intermediate value, say $f_1$, deviates by some $C\Delta t$ from the correct solution (with some constant $C$), we get $f_2 = 3C\Delta t$, $f_3 = 7C\Delta t$, ..., $f_n = (C2^n - 1)\Delta t$ (the general solution to the recurrence equation is $f_n = 2f_0 - f_1 + 2^n(f_1 - f_0)$), so that any small LTE grows exponentially, and the "result" becomes *worse*, the *smaller* we choose $\Delta t$, as $N \propto 1/\Delta t$.
Important: The initial "perturbation" $f_1 = C\Delta t$ is $\mathcal{O}(\Delta t)$ and therefore consistent with the method's order. That means that "zero" can't be computed in a stable manner, the method is not "zero-stable." Of course, it will fail for any non-trivial $G$ as well.

This definition of "zero-stability" is borrowed from the numerical analysis of boundary value problems. However, the statement that we can reach the exact solution in the limit $\Delta t \to 0$ might be of little help in practice. From the example of the harmonic oscillator, we have already learned that, although the forward-Euler method is zero-stable and hence formally convergent, meaningful results require tiny $\Delta t$ to follow only few oscillator periods, because the errors $E_n$ are still amplified from step to step.

A more stringent concept is **absolute stability**, which requires all iteration eigenvalues to be $|\lambda| \leq 1$, so that the error $E_n$ won't be amplified at all. Applying this to the lineare ODE from above, again using forward-Euler, this condition turns into $|1 + \lambda\Delta t| \leq 1$, meaning the method is absolutely stable for $\lambda\Delta t \in [-2, 0]$.

In the more general form, the eigenvalues $\lambda$ will be complex-valued, and those regions in the complex plane that feature $|1+\lambda\Delta t| \leq 1$ are termed the **regions of absolute stability**. For multi-level schemes, methods exist to investigate these regions in terms of characteristic polynomials of the iteration coefficients. For ODE systems, again, the analysis involves the eigenvalues of the iteration matrix.

Even this stability term is not helpful in all cases: Consider, for example, an ODE that describes some exponential growth process. Here, we wouldn't get far with absolute stability, which motivates more elaborated concepts like "A-stability", "relative stability", etc. In any case, this shows that the choice of a specific numerical scheme must always be seen in the context of the application that is addressed.

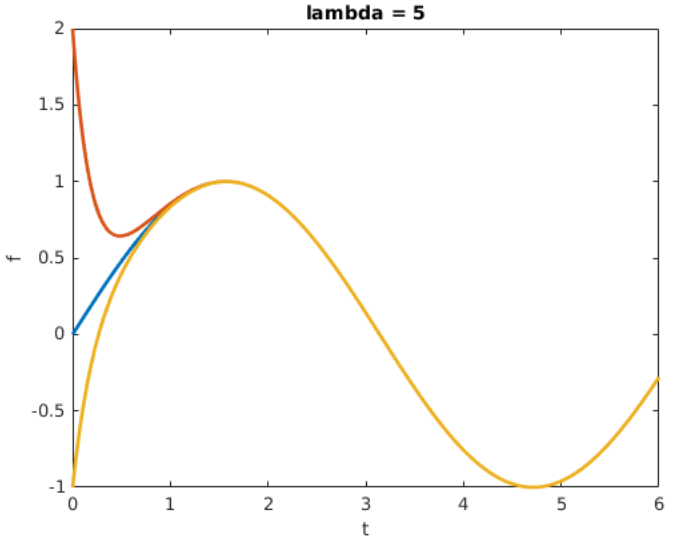
Finally, the term *stiffness* for ODE shall be mentioned. It originates from mechanical systems (e.g., stiff springs) and occurs in many branches of physics and related fields. In layman's terms, it means that the equation(s) contain one or more terms that alone give rise to a "fast" evolution althogh the actual solution evolves only "slowly." One is then confronted with the dilemma that the slow evolution would admit large $\Delta t$, i.e. few steps, to obtain the desired *accuracy*, while *stability* requires much smaller $\Delta t$ in order to prevent the fast terms from blowing up.

An example is this: The equation $f' = \cos t$ with initial value $f(0) = 0$ is "solved" by $f = \sin t$. We make this a "true" ODE by writing

$$f' = -\lambda(f - \sin t) + \cos t$$

with some constant $\lambda$ and require the same initial $f(0) = 0$ to get the same $f = \sin t$.

Inspecting the first term on the RHS, we see that $\lambda > 0$ describes a decay or relaxation of $f$ towards $\sin t$ on the time scale $1/\lambda$. The second term is still the inhomogeneity which contains the "driving" time scale $\sim 1$. Starting with an initial value $f(0) \neq 0$, the solution $f$ will approach the sine function to follow it afterwards.
With $\lambda \gg 1$, this relaxation is fast, and the integration step must obey the restrictive condition $\lambda \Delta t \leq 1$ to guarantee stability, even when we're only interested in the "slow" solution $f = \sin t$ zu $f(0) = 0$ and not so much in the fast initial relaxation phase. sind.




A number of methods exists for such stiff systems, which in practice come in a more disguised setting, and many of these schemes combine explicit and implicit methods, where the latter are included for the sake of stabilizing the "fast" terms.

# 3 Partial Differential Equations (PDEs)

Following the discussion of ordinary differential equations, we now turn to PDEs, i.e., situations in which functions that are sought as solutions depend on several independent variables, and where the PDE(s) combine the corresponding partial derivatives. Well-known examples in physics are fields that depend on several space coordinates, like cartesian $x, z, z$ or cylindrical $\rho, \varphi, z$, and possibly also on time $t$.

Traditionally, second order PDEs (i.e., those containing second derivatives as highest) are classified as elliptic, parabolic and hyperbolic, based on which derivatives they involve, and which coefficients appear with those derivatives: The generic form, for the simplest case of a real-valued function $f(x, y)$ in two variables, is

$$p\frac{\partial^2 f}{\partial x^2} + q\frac{\partial^2 f}{\partial x \partial y} + r\frac{\partial^2 f}{\partial y^2} + s\frac{\partial f}{\partial x} + t\frac{\partial f}{\partial y} + uf + v = 0$$

with coefficients $p, q, r, s, t, u, v$, which in turn might be functions of $x$ und $y$. In analogy to the various conic sections, they are called **elliptic** if $q^2 < 4pr$, **parabolic** for $q^2 = 4pr$, and **hyperbolic** if $q^2 > 4pr$. We want to discuss prototypes of these types based on their appearance in physics:

1. One example of an **elliptic** equation is **Poisson's equation** $\Delta f = \rho$, known from, e.g., electrostatics. In two-dimensional space with cartesian coordinates, it's written out as

   $$\frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2} = \rho(x,y)$$

   and the homogeneous specialization, $\Delta f = 0$, is called **Laplace's equation**. In the standard problem from electrostatics, the charge density $\rho$ will be a given function of space, and the task is to calculate the corresponding electric potential $f$ (usually called $\Phi$). Further, the gradient would be the electric field, $\vec{E} = -\nabla f$, where we omitted gauge constants like $4\pi\varepsilon_0$ for clarity. Poisson's equation also appears in incompressible hydrodynamics in order to compute the fluid pressure $p$ from the velocity field $\vec{v}$.

The physical situation here is a **boundary value problem (BVP)**: The PDE itself contains information about the spatial variation of the desired solution $f$, and in order to close the problem, additional **boundary conditions** for $f$ on the entire boundary of the solution's domain must be given. Examples here are **Dirichlet conditions**, which prescribe the **value** of $f$ itself on the domain boundary, or **von-Neumann conditions**, which prescribe the normal derivative $\partial f/\partial n = \nabla f \cdot \vec{n}$ with respect to the local normal direction $\vec{n}$.
Thus, elliptic equations are of **global** type: We are looking for a solution $f$ in the entire domain under consideration, given values on the boundary, and the PDE communicates information about $f$ between all domain points by virtue of $f$'s derivatives.

2. The **diffusion equation** or **heat equation, Fourier's law, ...** is of **parabolic** type: It combines space- and time dependencies of $f(\vec{r}, t)$ like

$$\frac{\partial f}{\partial t} = \nabla \cdot (\kappa \nabla f)$$

where $\kappa$ might be a diffusion coefficient, heat conductivity or the like, which in general might depend on space and time as well. With constant $\kappa$, we get the special case $\frac{\partial f}{\partial t} = \kappa \Delta f$. If we're dealing with only one spatial coordinate, $x$ say, the equation is written out as $\frac{\partial f(x,t)}{\partial t} = \frac{\partial}{\partial x}\left(\kappa \frac{\partial f(x,t)}{\partial x}\right)$ or, with $\kappa = const.$, as $\frac{\partial f(x,t)}{\partial t} = \kappa \frac{\partial^2 f(x,t)}{\partial x^2}$.

Usually, parabolic equations come as **initial value problems (IVP)**: Initial conditions are given for some time $t_0$ as $f(x, t_0) = f_0(x)$ in the entire domain, and we look for the solution $f(x, t)$ for $t > t_0$. Again, we have to provide **boundary conditions** at the domain boundaries for all $t \geq t_0$, which encode the physical interaction with the "outside world."

A physical interpretation is that some physical quantity (mass, electric charge, heat) gets transported through space according to Fick's law with a corresponding current density $\vec{q} = -\kappa \nabla f$ or flux density, which then would be the mass flux density, electric current density or the heat flux density. This transport will then change the local concentration $f$ of the underlying quantity according the divergence of the flux density, $\partial_t f = -\nabla \cdot \vec{q}$, so that we can read the equation as a **balance equation** with diffusive flux.

3. An example for a **hyperbolic equation** is the **convection/ advection equation**

$$\frac{\partial f}{\partial t} = -\vec{c} \cdot \nabla f$$

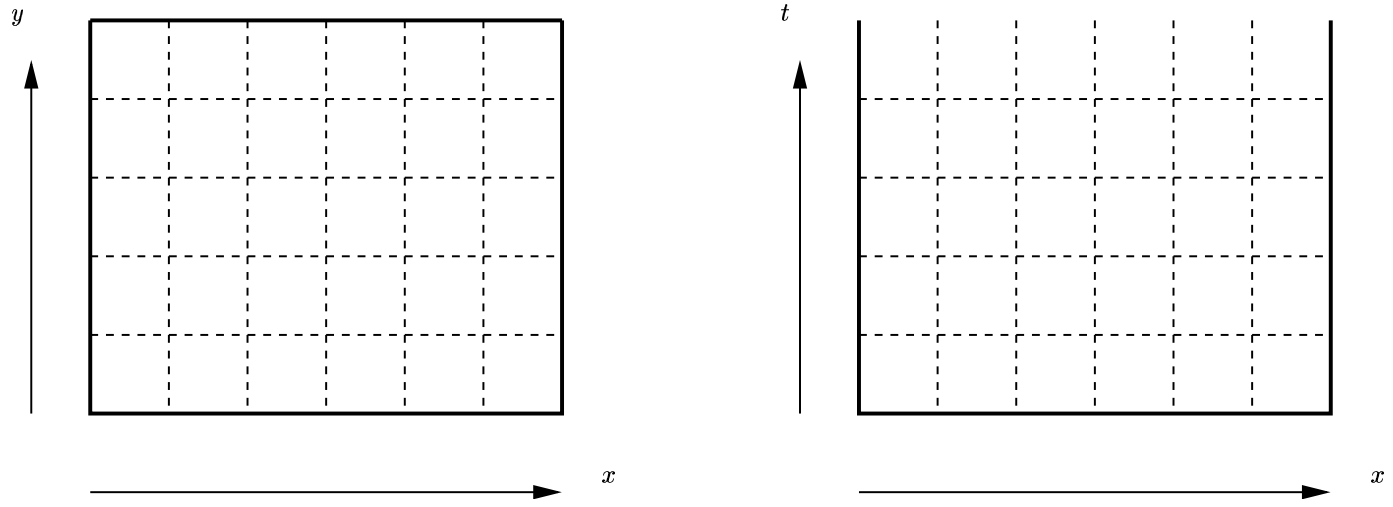
which describes the (passive) transport of a quantity $f$ through space with a given velocity $\vec{c}$. This is sometimes written as $\frac{Df}{Dt} = 0$, where $\frac{D(\cdot)}{Dt} = \frac{\partial(\cdot)}{\partial t} + \vec{c} \cdot \nabla(\cdot)$ denotes the a "**total/material derivative**" wrt. $\vec{c}$, that is the rate of change as seen by an observer moving with velocity $\vec{c}$. Again, we're dealing with with an **initial value problem** that requires additional specification of initial- and boundary conditions.

In the special case of only one cartesian spatial coordinate $x$, we have $\frac{\partial f(x,t)}{\partial t} = -c\frac{\partial f(x,t)}{\partial x}$. By taking the derivatives wrt. both $x$ and $t$, we arrive, for constant $c$, at the **wave equation**

$$\frac{\partial^2 f(x,t)}{\partial t^2} - c^2 \frac{\partial^2 f(x,y)}{\partial x^2} = 0$$

Physical examples often involve coupled quantities, like with acoustic sound waves or with electromagentic waves.

All examples above are **linear** equations.




Boundary conditions for an (elliptic) BVP in $x, y$ (left) and for an IVP (parabolic, hyperbolic) in $x, t$ (right).

## 3.1 Elementary Solution Using Finite Differences

We can easily discretize the PDEs mentioned above by means of Finite Differences as we did before with ODEs. All we have to take into account is that the solution $f$ is now a multivariate function, so that we must discretize for all the independent variables $x$, $y$, $t$ that appear.

Take, for instance, a function $f(x,t)$ and discretize space and time as $x_j = j\Delta x$ and $t_n = n\Delta t$, respectively. Then, the function values at these discrete points are

$$f_j^n := f(x_j, t_n) = f(j\Delta x, n\Delta t)$$

where we follow the widely used notation to write the time index $n$ as upper and all spatial indices $j, k, l$ as lower indices.

Let's take the diffusion equation in its simplest form,

$$\frac{\partial f}{\partial t} = \kappa \frac{\partial^2 f}{\partial x^2}$$

We can approximate the $x$-derivative at each time $t_n$ and space position $x_j$ by the standard three-point stencil like

$$\left.\frac{\partial^2 f}{\partial x^2}\right|_{(x_j,t_n)} = \frac{f_{j+1}^n - 2f_j^n + f_{j-1}^n}{\Delta x^2} + \mathcal{O}(\Delta x^2)$$

(ahead of boundaries, we'll have to implement the appropriate conditions by adjusting the formula there).

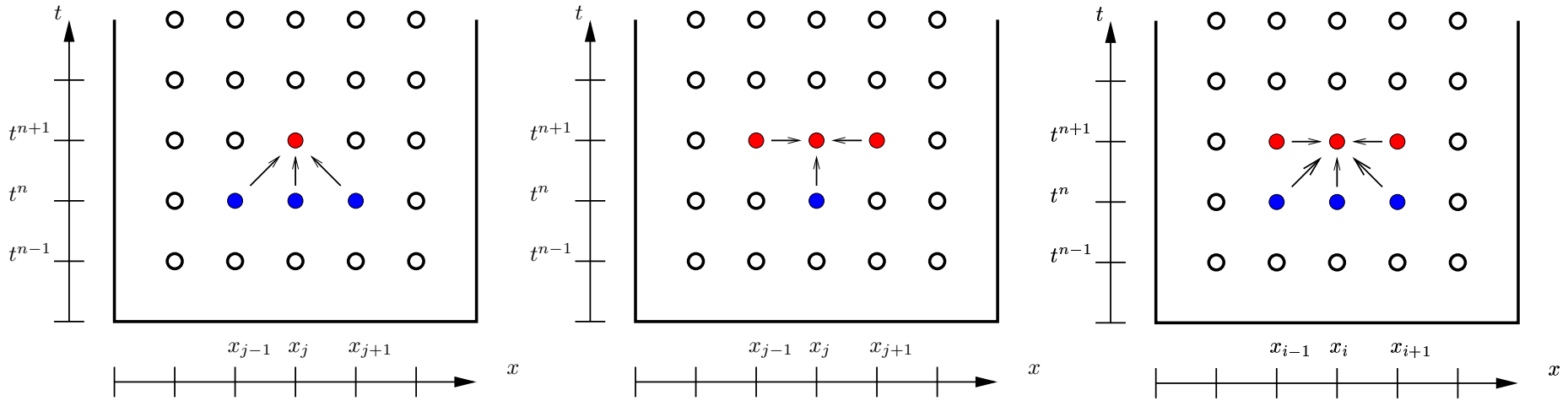
As we're dealing with a time-dependent initial value problem, we can resort to one of the known methods for time integrations, e.g. Runge-Kutta or the like. The simplest, however, is once more the Euler method, either forward like

$$f_j^{n+1} = f_j^n + \frac{\kappa\Delta t}{\Delta x^2}\left(f_{j+1}^n - 2f_j^n + f_{j-1}^n\right)$$

or backward

$$f_j^{n+1} = f_j^n + \frac{\kappa\Delta t}{\Delta x^2}\left(f_{j+1}^{n+1} - 2f_j^{n+1} + f_{j-1}^{n+1}\right)$$

And as before, the implicit backward formula comes with the complication that unknown neighboring values of $f$ are needed for the update $f_j^{n+1}$, as illustrated by this sketch that displays $f$ in the $x$-$t$-plane: To the left, we see the dependencies when using forward integration, the center and right pictures demonstrate the implicit coupling for backward-, $\vartheta-$ or trapezoidal methods.




Before trying the actual numerical implementation, we'd like to know what we have to expect from the diffusion equation: Take a constant $\kappa$, disregard effects from the boundaries, and assume the function $f$ to be a wave component

$$f(x,t) = \hat{f}_k e^{i(kx-\omega t)}$$

with real wave number $k$ and an angular frequency $\omega$ (remember that, given the linearity of the equation, we can decompose each appropriate $f$ into spectral components and treat each of them separately).

Inserting this ansatz into the equation yields the **dispersion relation** $-i\omega = -\kappa k^2$, which states that a mode with wave number $k \in \mathbb{R}$ will evolve $\propto e^{-i\omega t} = e^{-\kappa k^2 t} = e^{-\gamma(k)t}$, that is, it decays at rate $\gamma(k) = \kappa k^2$: Long wave length modes decay slowly, short wave lengths decay fast. Addressing again an arbitrary $f$ as a superposition of spectral components with different wave numbers $k$, we expect the **diffusion** process to **smoothen** $f$ with time.

## 3.2 Discretization Matrix, Time-Implicit Discretization

Explicit discretization of the diffusion equation wil typically lead to a rather restrictive condition on the time step to use in order to guarantee numerical stability. This can be seen from the so-called **von-Neumann** analysis, the discrete counterpart of the normal mode analysis given above: Insering the ansatz

$$f_j^n = \hat{f}_k^n e^{ikj\Delta x}$$

into the forward-Euler step and looking at the resulting ratio

$$\lambda := f_j^{n+1}/f_j^n = \hat{f}_k^{n+1}/\hat{f}_k^n$$

we see that the solution will remain bounded only if we choose $\Delta t < C\Delta x^2$, with a constant $C$ that incorporates the phyical parameters. This means that the allowable time step decreases quadratically with $\Delta x$ when improving the spatial resolution, so that a large number of small time steps must be carried out when using fine grids. While we made this analysis only for the forward-Euler method, it generally applies to other explicit time discretizations as well, i.e. to higher-order explicit Runge-Kutta methods. Using our experience gained from ODEs, we expect that implicit discretizations will alleviate this problem to allow larger $\Delta t$.

How to implement an implicit time scheme for finite differences in space? Let's put the time dependency aside for a moment and look at the discrete spatial derivative $\partial_{xx} f \approx \frac{f_{j+1}^n - 2f_j^n + f_{j-1}^n}{\Delta x^2}$. We can write this operation as a matrix product when arranging the $f_j$-data on the grid as a column vector $\underline{f}$ and set up a **discretization matrix** for the $\partial_{xx}$-differences as

$$(\partial_{xx} f)_j \to \frac{1}{\Delta x^2} \begin{pmatrix} -2 & 1 & & & & 1 \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \dots & \dots & \dots & \\ & & & 1 & -2 & 1 \\ 1 & & & & 1 & -2 \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \dots \\ f_{m-1} \\ f_m \end{pmatrix} = \underline{\underline{D}} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \dots \\ f_{m-1} \\ f_m \end{pmatrix} = \underline{\underline{D}}\, \underline{f}$$

One easily verifies that the two entries in the upper-right and lower-left corners represent periodic boundary conditions assumed here. Further, we have saved us from writing out the zero-entries for clarity. A matrix like this can easily be set up in Matlab or Python.

Re-introducing time dependency by adding the upper index $n$ to $\underline{f}^n$, the **forward-Euler** scheme is written compactly as

$$\underline{f}^{n+1} = \underline{f}^n + \Delta t \kappa \underline{\underline{D}}\, \underline{f}^n = (\underline{\underline{1}} + \Delta t \kappa \underline{\underline{D}}) \underline{f}^n = \underline{\underline{V}}\, \underline{f}^n$$

That's for the forward-Euler – however, **backward**-Euler is just as simple,

$$\underline{f}^{n+1} = \underline{f}^n + \Delta t \kappa \underline{\underline{D}}\, \underline{f}^{n+1} \quad \text{or} \quad (\underline{\underline{1}} - \Delta t \kappa \underline{\underline{D}}) \underline{f}^{n+1} = \underline{\underline{R}}\, \underline{f}^{n+1} = \underline{f}^n$$

where we see that carrying out one time step here amounts to solving a linear system of equations, or, in loose terms, inverting the matrix $\underline{\underline{R}}$.

Naively, we can do exactly that: Numerically compute the full inverse matrix $\underline{\underline{R}}^{-1}$ once, and then simply keep on multiplying $\underline{f}^n$ from the left for each step. In Matlab, we would write something like `Rinv = inv( R )` and then carry out the step $\underline{f}^n \to \underline{f}^{n+1}$ as `f = Rinv * f`. This approach comes with several caveats: First of all, in contrast to $\underline{\underline{R}}$ itself, the inverse will in general not be sparse, so that it might occupy a large space in computer memory for larger problems. Then, computing the inverse is numerically tricky and often amplyfies roundoff errors. Finally, the inverse might actually not exist! This is true for the $\underline{\underline{R}}$ from above and is a direct consequence of the periodic boundary conditions assumed here.

The **better** alternative is **not** to **invert the matrix** as a whole, but rather solve the matrix equation as a linear system, where we would even be satisfied with *one* solution if the system is ill-conditioned. Standard numerical software packages used for this purpose usually employ algorithms that are stable and fast to compute the "new" $\underline{f}^{n+1}$. Again, in Matlab we would simply use the \-Operator to write the iteration step as `f = R \ f;` .

To cope with the aspect of memory efficiency, we even don't need to save the entire of the dicretization matrix $\underline{\underline{D}}$, as software packages usually offer the possibility to save **sparse matrices** in a spectial format that only stores the non-zero elements and omit all the zeros from memory. This sparse format then "survives" the computation, i.e. products of sparse matrices are gain stored as such, and multiplications of zeros are omitted. Matlab offers the function `sparse` for this purpose.

Up to now, we have used periodic boundary conditions when setting up the matrix of the discretized $\partial_{xx}$ operator. Several ways exists to implement **Dirichlet**-conditions instead, and an easy way to achieve this is setting the first and last row of $\underline{\underline{D}}$ to zero, i.e. in Matlab `D(1, :) = 0` and `D(end, :) = 0`. We can see that this will leave $f_0$ and $f_m$ untouched under the iteration, keeping their initial Dirirchlet values. With minor modifications, we can also implement time-dependent Dirichlet conditions this way.

## 3.3 Flux Form, Conservative Discretization

In more realistic applications, the diffusion coefficient $\kappa$ won't be a constant, but might depend on position $x$ and time $t$, so that the equations takes the more general form

$$\frac{\partial f(x,t)}{\partial t} = \frac{\partial}{\partial x}\left(\kappa(x,t)\frac{\partial f(x,t)}{\partial x}\right) = -\frac{\partial}{\partial x}q(x,t)$$

where $q(x,t) = -\kappa(x)\frac{\partial f(x,t)}{\partial x}$ is the **diffusive flux density** associated with quantity $f$ (the two minus signs are convention in physics). Written in this **divergence form**, the concept of **conservation** of $f$ becomes more obvious: Integrating the equation over a finite interval $x \in [a,b]$ in space, we get

$$\frac{d}{dt}\int_a^b f(x,t)\,dx = -\left(q(b,t) - q(a,t)\right)$$

that is, the **change** of the total integral of $f$ in the interval/ domain is nothing but the **transport** across the boundaries, quantified by the surface fluxes $q$. This means that the underlying quantity is **conserved**, as it is only transported into/ out of the domain, and re-distributed within. This important physical property can be reflected in the numerical discretization by using a **conservative discretization** in terms of **numerical fluxes** that are computed and used to update $f$ from the **flux differences**. Even if the flux computation introduces truncation errors from $\Delta x$ and $\Delta t$, the conservation is guaranteed up to round-off error.

To apply this idea to the diffusion equation with forward-Euler, we would approximate the (diffusive) fluxes at half-integer grid positions $x_{j+1/2}$ as

$$q^n_{j+1/2} = -\kappa_{j+1/2}\frac{f^n_{j+1} - f^n_j}{\Delta x}$$

to then do the $f$-update as

$$f^{n+1}_j = f^n_j - \Delta t\frac{q^n_{j+1/2} - q^n_{j-1/2}}{\Delta x}$$

For constant $\kappa$, this is identical to the standard discretization of $\kappa\partial_{xx}$ from above, but even for arbitrary $\kappa(x,t)$, this conservative discretization will ensure that the sum $\sum_j f_j$ stays constant, up to the effect of boundary contributions. Depending on the context, the half-integer values $\kappa_{j+1/2}$ will either be directly available, or they can be interpolated as $\kappa_{j+1/2} = \frac{1}{2}(\kappa_j + \kappa_{j+1})$.

## 3.4 Advection Equation

A prototype of hyperbolic PDEs is the advection equation, which in its simplest form reads

$$\frac{\partial f(x,t)}{\partial t} + c\frac{\partial f(x,t)}{\partial x} = 0$$

with $c = \text{const}$. Given initial conditions as $f(x, t = 0) = f_0(x)$, it is solved by

$$f(x,t) = f_0(x - ct),$$

in words: Initial values of $f_0$ are transported through space at vecolity $c$. If we were tempted to use the Fourier approach like $f(x,t) \sim e^{i(kx-\omega t)}$, we would get the simple dispersion relation $\omega = ck$, telling us that all spectral components move at the same speed $c$, and every superposition is again simply shifted with this velocity. Thus, we don't even need a computer to obtain the solution to this equation. We will need one, however, if the equation becomes slightly more complex, and we therefore should be able to solve even the simplest case numerically.

It is surprisingly difficult to produce good numerical solutions to this simple equation with methods that are based on spatial grids, i.e., finite differences. Trying, as for the diffusion equation before, the forward-Euler and central differences, also called **FTCS** ("Forward in time, centered in space") here,

$$f_j^{n+1} = f_j^n - \frac{c\Delta t}{2\Delta x}(f_{j+1}^n - f_{j-1}^n)$$

gives us unconditional (absolute) instability and very poor approximations.

Some elementary methods, among many others, to improve the behavior, are

- the **leapfrog** method

$$f_j^{n+1} = f_j^{n-1} - \frac{c\Delta t}{\Delta x}(f_{j+1}^n - f_{j-1}^n)$$

  which separates the spatial grid into one "even" and one "odd" subgrid, resulting in a two-level scheme.

- **Upwind** methods are based on the follwing idea: The (physical) "transport" of quantity $f$ with velocity $c$ corresponds to the flow of information into one "downwind" direction, as given by the sign/ direction of $c$ (or $\vec{c}$ in the multidimendional generalization). It is therefore natural use a one-sided difference into the "upwind" direction to capture the inflow of information, rather than using central differences. In first order, this would be like

$$f_j^{n+1} = \begin{cases} f_j^n - \frac{c\Delta t}{\Delta x}(f_j^n - f_{j-1}^n), & c \geq 0 \\ f_j^n - \frac{c\Delta t}{\Delta x}(f_{j+1}^n - f_j^n), & c < 0 \end{cases}$$

- The **Lax-Friedrichs** method

$$f_j^{n+1} = \frac{f_{j-1}^n + f_{j+1}^n}{2} - \frac{c\Delta t}{2\Delta x}(f_{j+1}^n - f_{j-1}^n)$$

  is based on the FCTS, but it introduces considerable numerical diffusion by averaging $f$ wrt. $x$.

- The **Lax-Wendroff** method starts with a half Lax-Friedrichs step to produce intermediate values on a staggered mesh (*predictor*),

$$f_{j+1/2}^{n+1/2} = \frac{f_j^n + f_{j+1}^n}{2} - \frac{c\Delta t}{2\Delta x}(f_{j+1}^n - f_j^n)$$

  followed by a full step with central differences (*corrector*),

$$f_j^{n+1} = f_j^n - \frac{c\Delta t}{\Delta x}(f_{j+1/2}^{n+1/2} - f_{j-1/2}^{n+1/2})$$

  which results in a second order accurate scheme in $x$ and $t$.

Comparing these schemes, it becomes obvious that they differ significantly with respect to their inherent **numerical diffusion**. In fact, it turns out that, except for the leapfrog, they can be written as

$$f_j^{n+1} = f_j^n - \underbrace{\frac{c\Delta t}{2\Delta x}(f_{j+1}^n - f_{j-1}^n)}_{c\Delta t\partial_x u + \mathcal{O}(\Delta x^2)} + \underbrace{\frac{\kappa\Delta t}{\Delta x^2}(f_{j+1}^n - 2f_j^n + f_{j-1}^n)}_{\kappa\Delta t\partial_{xx} u + \mathcal{O}(\Delta x^2)}$$

that is a $\mathcal{O}(\Delta x^2)$ forward step in the advection term, supplemented with a stabilizing diffusion, so that we're effectively using a FTCS-like discretization of the *advection-diffusions equation* $\partial_t u = -c\partial_x u + \kappa \partial_{xx} u$, where the individual schemes differ only by the amount of intrinsic diffusivity $\kappa$ according to

- FTCS: $\kappa = 0$, unstable forward discretization.
- First order upwinding: $\kappa = \frac{c\Delta x}{2}$, employs a stabilizing diffusivity that depends on $\Delta x$.
- Lax-Friedrichs: Here, $\kappa = \frac{\Delta x^2}{2\Delta t}$, and we must take some care when using it, because it doesn't allow for the *semi-discrete limit* $\Delta t \to 0$ with $\Delta x$ fixed!
- Lax-Wendroff: $\kappa = \frac{c^2 \Delta t}{2}$ enhances FTCS to $\mathcal{O}(\Delta t^2)$ accuracy and stabilizes at the same time.

For the last example, we can use some analysis of the underlying PDE to understand why Lax-Wendroff choose this particular $\kappa$: Derive the PDE $\partial_t f = -c\partial_x f$ wrt. $t$ to get the wave form $\partial_{tt} f = c^2 \partial_{xx} f$ and insert both into the Taylor expansion to obtain the **modified equation**

$$f_j^{n+1} = f_j^n + (\partial_t f)_j^n \, \Delta t + \frac{1}{2} (\partial_{tt} f)_j^n \, \Delta t^2 + \mathcal{O}(\Delta t^3) = f_j^n - c \, (\partial_x f)_j^n \, \Delta t + \frac{c^2}{2} (\partial_{xx} f)_j^n \, \Delta t^2 + \mathcal{O}(\Delta t^3)$$

While the FTCS methods neglects the $\mathcal{O}(\Delta t^2)$ term, getting unstable, the Lax-Wendroff method approximates it by virtue of an explicit discretization of $\partial_{xx} f$ wrt. $x$ to obtain a $\mathcal{O}(\Delta t^3)$ consistent approximation.

## 3.5 Extension to Multiple Space Dimensions

Up to now, we have only considered one space dimension with coordinate $x$. While multi-dimensional systems hugely increase the variety of physical phenomena and complexity, the extension of the methods disussed so far is mostly straight-forward and primarily concerns the increased programming efforts and computational costs.

To give an example, a function $f(x, y, t)$ in two space dimensions and time would be discretized like

$$f^n_{j,k} = f(x_j, y_k, t_n) = f(j\Delta x, k\Delta y, n\Delta t)$$

and in three dimensions analogously as $f^n_{j,k,l}$. Partial derivatives follow naturally as $\partial_y f = (f^n_{j,k+1} - f^n_{j,k-1})/(2\Delta y) + \mathcal{O}(\Delta y^2)$ etc.,

In implementation environments like python or matlab, one will arrange the data at some time level $t_n$ in matrices or higher-dimensional data arrays with indices $j, k$, where $j$ counts the matrix' rows, and $k$ its coloumns. "Looking" at the matrix (e.g. when printing it out for debugging purposes), we will "see" the first direction ($\sim x$) positive downwards, while the second direction ($\sim y$) goes positive to the right, which is different from our standard orientation. To fix this, we might transpose the matrix where appropriate, or choose a different index sequence.

A conservative forward-Euler discretization of $\partial_t f = -\nabla \cdot \vec{q}$ in two space dimensions would follow as

$$f^{n+1}_{j,k} = f^n_{j,k} - \Delta t \left( \frac{q^n_{x,j+1/2,k} - q^n_{x,j-1/2,k}}{\Delta x} + \frac{q^n_{y,j,k+1/2} - q^n_{y,j,k-1/2}}{\Delta y} \right)$$

with staggered cartesian components of the flux density $\vec{q} = (q_x, q_y)$.

Applying this to the diffusion equation with $\vec{q} = -\kappa \nabla f$, we would in turn compute those fluxes in a staggered approach, i.e.,

$$q^n_{x,j+1/2,k} = -\kappa_{j+1/2,k} \frac{f^n_{j+1,k} - f^n_{j,k}}{\Delta x}, \quad q^n_{y,j,k+1/2} = -\kappa_{j,k+1/2} \frac{f^n_{j,k+1} - f^n_{j,k}}{\Delta y}$$

# 4 Discrete Fourier Transform

The **Fourier transformation** of a function $f(x)$, together with its inverse (back-)transformation, are well-known operations

$$\hat{f}(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-ikx} dx, \quad f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{f}(k) e^{ikx} dk$$

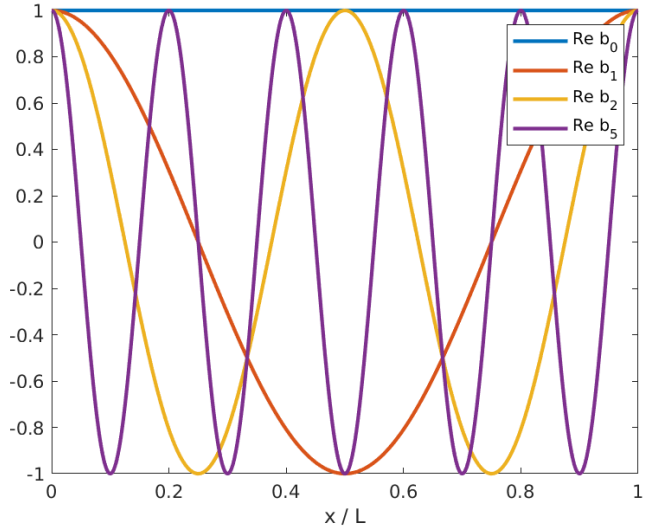
and frequently used in physics in order to analyze (Fourier-)spectral aspects of $f$ and its behavior. In analogy to these transformations, the **discrete Fourier transformation** (DFT) acts on a *discrete* and finite-sized sequence of numbers $f_j$ with $j = 0..(m-1)$, which often are the values of given function $f$ at discrete points, $f_j = f(j\Delta x)$, i.e. on a regular mesh with spacing $\Delta x$.

These transforms can be understood as writing $f(x)$, or the discrete set $f_j$, as a linear *superposition* of harmonic base functions with individual wave numbers $k$, for which the derivation with respect to $x$ becomes trivial. As a consequence, because the discrete $f_j$ naturally cover only a limited period length $L = m\Delta x$, it is implicitly assumed that $f$ is periodic when extended beyond a corresponding interval, i.e., $f(x + L) = f(x)$.

We start with some fundamentals of **sampling** a continous function, which come actually before discussing the DFT itself: Representing $f(x)$ on interval $x \in [0, L]$ by $m$ function values $f_j$ with fixed **sample interval** (or "sample rate, grid spacing, ...") $\Delta x = L/m$, we write down the corresponding **base functions** with wave numbers $k_\beta = \beta \frac{2\pi}{L} = \beta \Delta k$, $\beta = 0, \pm 1, \pm 2, ...$ as

$$b_\beta(x) = e^{ik_\beta x}$$

These are later combined to make up $f$. As the $b_\beta(x)$ must fit into the $x$-interval, every $k_\beta$ is a (integer) multiple of $\Delta k = \frac{2\pi}{L}$, so that $\beta$ is just the number of oscillations that $b_\beta$ undergoes in $[0, L]$, and we identify the base function with smallest $k$ (or "lowest frequency" in a time-frequency setting) as $b_1(x) = e^{i\,2\pi x/L}$, while $b_0(x) = 1$ corresponds to the average (non-oscillatory or "DC" part) of $f$.



With the *longest* wave length mode $\beta_1$ identified, we might ask for the *shortest* wave length (largest $k$) that can be represented by a sampling process with interval $\Delta x$?

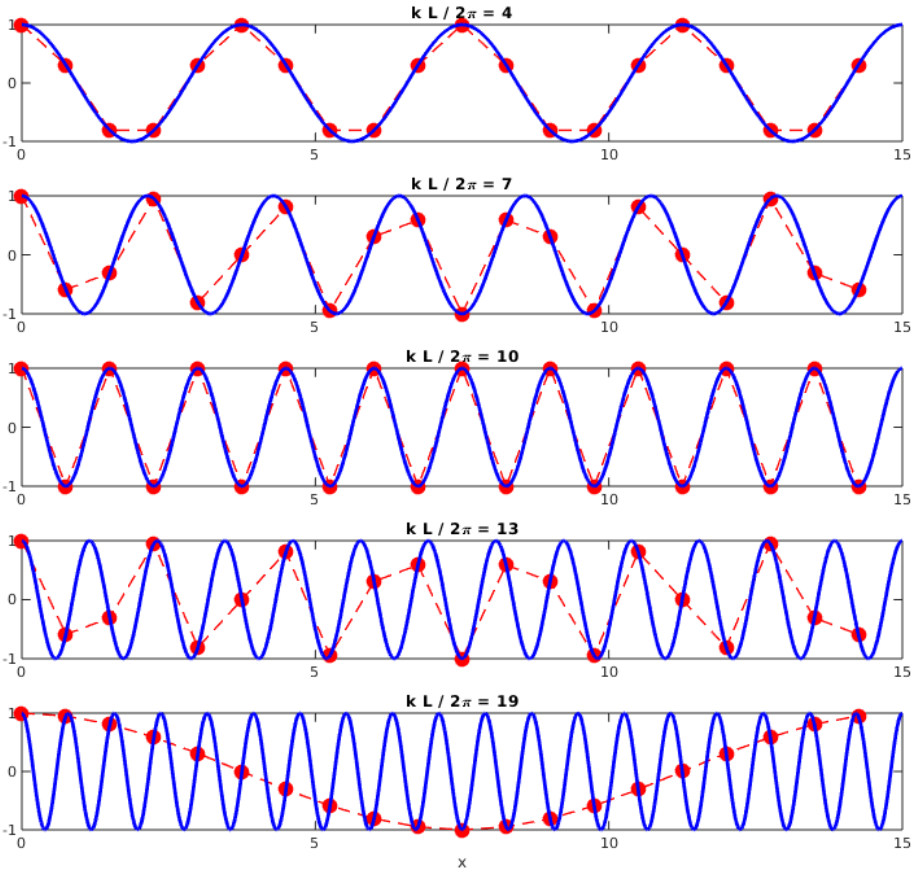
In order to represent an oscillation, we need at least two function values of $f$, and if these are sepatated by $\Delta x$, the mode with shortest possible wave length, or largest wave number (frequency, ...), has wave length $\lambda_{\min} = 2\Delta x$ or **Nyquist wave number** (**Nyquist frequency**, ...)

$$k_{\text{Nyq.}} = \frac{\pi}{\Delta x} = \frac{m\pi}{L}$$

The crucial question is: What will happen, if function $f$, about to be sampled with $\Delta x$, contains spectral contributions beyond $k_{\text{Nyq.}}$, that is, wave lengths that are not "resolved" by the chosen $\Delta x$ ?

It would be nice if those parts of the "signal" $f$ would simply get lost during the sampling process. This, unfortunately, is not the case: Those parts of the signal that can't be represented by $k_{\text{Nyq.}}$ get **aliased** into the sampled values $f_j$, which means, they are "**mirrored**" at the upper limit $k_{\text{Nyq.}}$ into the representable spectrum, so that the $f_j$ contain **fake oscillations** at smaller $k < k_{\text{Nyq.}}$ which are not present in the original function $f$.

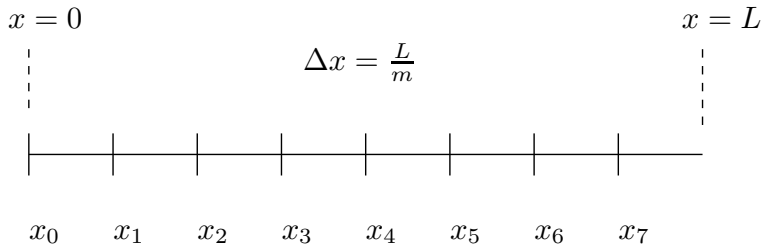
This effect is demonstrated in the graphic: Successively shorter wave length functions (blue) in $x \in [0, 15]$ are sampled using the same $m = 20$ positions (red circles). With $k_{\text{Nyq.}} = 20\pi/L$, a maximum of 10 oscillations can be represented using the sampling positions $x_j$. Then, sampling a function that contains, e.g. 13 (and hence "too many") oscillations, we get **the same** sample valued $f_j$ that we get from the $m = 7 = 10 - 3$ mode. Simlilarly, the sample values of $\sin\left((10+9)\frac{2\pi}{L}x\right)$ are *identical* to those of $\sin\left((10-9)\frac{2\pi}{L}x\right)$.
Important to note: All this has nothing to do (yet) with the Fourier transformation itself, it is simply a consequence of sampling a continous function $f(x)$ at discrete positions $x_j$ with sample intervals $\Delta x$.
It's the **Nyquist-Theorem**: If $f(x)$ contains wave lengths above $k_{\text{Nyq.}}$, then these will be aliased into the discrete spectrum *by the sampling process*. The same applies to sampling with repsect to time with interval $\Delta t$ and the correspoding **Nyquist frequency** $\omega_{\text{Nyq.}} = \frac{\pi}{\Delta t}$. This means, that the signal must be **filtered** ("**de-aliased**") **prior to the sampling process** in order to remove spectral components beyond $k_{\text{Nyq.}}$ from $f$.




With these consideration, the good news is the **sampling theorem**: If a continous function $f(x)$ (or $f(t)$) is **bandwidth limited** with $k_{\text{Nyq.}}$ (or $\omega_{\text{Nyq.}}$), which means that is doesn't contain spectral components beyond that limit, then $f$ is completly determined by its sample values $f_j$ and we have the full information about $f$ in terms of the $m$ sample values.

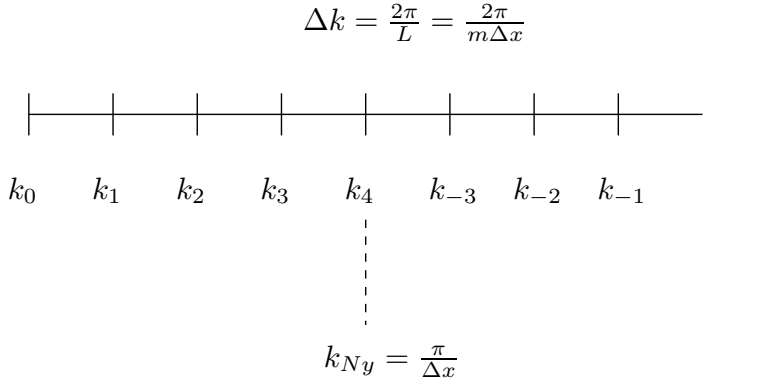
Then, this full information is equivalently contained in the $m$ Fourier amplitudes $\hat{f}_\beta$ that we compute by transforming the $f_j$.

The setting is this: We assume that we have the $m$ sample values $f_j$, taken in $x$ at positions $x_0 = 0$ to $x_{m-1} = (m-1)\frac{L}{m}$ with spacing $\Delta x = \frac{L}{m}$. These positions $x_j$ correspond wave numbers $k_\beta$ with spacing $\Delta k = \frac{2\pi}{m\Delta x}$ in wave number or Fourier space, running from $k_0 = 0$ to $k_{\text{Nyq.}} = \frac{\pi}{\Delta x}$ in the center of the spectrum. Beyond the Nyquist wave number, we can interpret the wave numbers as "negative" and rapidly oscillating base functions, then increasing towards longer wave lengths, i.e. slower oscillating base funtions.




The transformation from function values $f_j$ to spectral amplitudes $\hat{f}_\beta$ and back, is performed accoring to

$$
\begin{aligned}
f_j &= \frac{1}{m}\sum_{\beta=0}^{m-1} \hat{f}_\beta e^{\frac{2\pi i}{m} j\beta} = \frac{1}{m}\sum_{\beta=0}^{N-1} \hat{f}_\beta b_\beta(x_j) \\
\hat{f}_\beta &= \sum_{j=0}^{m-1} f_j e^{-\frac{2\pi i}{m} j\beta}
\end{aligned}
$$




where the normalization factor $\frac{1}{m}$ is, as above, ususally attributed to the backward transformation only (it could as well be split into two factors $\frac{1}{\sqrt{m}}$ for both forward- and back-transformation).

With this, we interpret the $f_j$ as a superposition of oscillating base functions $b_\beta(x_j)$, each weighted with Fourier amplitudes (or coefficients) $\hat{f}_\beta$. In particular, $\hat{f}_0/m$ is the average (DC part) of the $f_j$.

This computation of the $\hat{f}_\beta$ is nothing but a "projection" of the $f_j$ onto the individual base functions, and the DFT is a unitary transformation in $\mathbb{C}^m$. Both sets of the $f_j$ and the $\hat{f}_\beta$, respectively, contain identical information.

From this definition, we get a number of conclusions:

- As $f$ is periodic with period length $L$, i.e., $b_\beta(x+L) = b_\beta(x)$, so is its spectrum $\hat{f}$ is with period length $\hat{L} = \frac{2\pi}{\Delta x}$, simply because $\hat{f}_{\beta+m} = \hat{f}_\beta$, as is easily (?) confirmed.
  This, however, means that we can "cut" the upper half of the spectrum and treat it as being located at "negative" wave numbers.
- Real valued $f_j \in \mathbb{R}$ usually have complex valued spectra $\hat{f}$, but they obey $\hat{f}_{-\beta} = \overline{\hat{f}_\beta}$ (complex conjugate).
- Taking the derivative wrt. $x$ of $f$ is a trivial operation on the $\hat{f}_\beta$: With $x_j = j\Delta x$, we get $\frac{df(x)}{dx} = \frac{1}{\Delta x}\frac{df(x)}{dj}$, (of course, we have only discrete values of $f$ available). Carrying this out in Fourier-representation yields

$$\frac{df}{dx} = \frac{1}{\Delta x}\frac{d}{dj}\left(\frac{1}{m}\sum_{\beta=0}^{N-1}\hat{f}_\beta e^{\frac{2\pi i}{m}j\beta}\right) = \frac{1}{\Delta x}\frac{1}{m}\sum_{\beta=0}^{m-1}\frac{2\pi i}{m}\beta\hat{f}_\beta e^{\frac{2\pi i}{m}j\beta} = \frac{1}{m}\sum_{\beta=0}^{m-1} ik_\beta \hat{f}_\beta e^{\frac{2\pi i}{m}j\beta}$$

The last term is a Fourier-representation on its own, and it tells us that the spectrum of $\frac{df(x)}{dx}$ is that of $f$ itself, multiplied (in $k$-space) by $ik_\beta$. So, taking the derivative in $x$-space corresponds *exactly* to the (simple) multuplication by $ik_\beta$ in $k$-space.

From this, we get the basic idea of a **Fourier-spectral** method in computations that involve derivatives in $x$: Compute these derivatives by performing a DFT on $f$, multiplying the spectrum by $ik_\beta$ and back-transform (if desired, at all) into $x$-space through the *inverse* Fourier transform.

Naturally, the second derivative $\frac{d^2}{dx^2}$ translates into multiplying by $-k_\beta^2\hat{f}_\beta$ in $k$-space, etc.

How "expensive" is it to perform a DFT on a set of $m$ function values $f_j$ ?

For each of the $m$ spectral amplitudes $\hat{f}_\beta$ with $\beta = 0..(m-1)$, we have to carry out the summation over the $m$ function values $f_j$, weighted with the $e^{-\frac{2\pi i}{m} j\beta}$-factors according to the DFT formula. Hence, we have a total of $m^2$ kernel operations, which predicts high computational costs for large data sets.

Luckily, the number of operations can be reduced significantly into a more benign $m \log m$ complexity, "simply" by rearranging the formula, identifying operations that occur repeatedly for different $\hat{f}_\beta$, and carrying out the transformation simulataneously for all $\beta$. This re-arrangement of the transformation formula was invented by Cooley & Tukey in 1965, became famous under the name "fast Fourier-transformation" or FFT, and is efficiently implemented in all standard numerical libraries. It works most efficiently if $m$ is a power of $2$, so that large and costly computations based on the FFT and its inverse (usually termed IFFT) regularly use mesh sizes of $2^?$ whenever possible.

The FFT/ IFFT are exactly the DFT formulae from above (up to number representation), just implemented in a "smart" economic way.

# 5 Conservation Laws / Finite Volume Methods

Time-dependent **conservation law** in one dimension: $\partial_t u(x,t) + \partial_x F(u(x,t)) = 0$

- $u(x,t)$: conserved variable, $F(u)$: "flux function." Simplest case: $u : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ scalar, $F : \mathbb{R} \to \mathbb{R}$.
- Generalizations: System of conserved variables, e.g., Euler equations of gas dynamics,

$$\partial_t \begin{pmatrix} \rho \\ \rho v \\ E \end{pmatrix} + \partial_x \begin{pmatrix} \rho v \\ \rho v^2 + p \\ v(E+p) \end{pmatrix} = 0$$

  where $u = (\rho, v, E)$ are mass density, velocity, total energy density. Equation of state asumed, e.g., $E = \frac{p}{\gamma - 1} + \frac{1}{2}\rho v^2$ with adiabatic index $\gamma$. $F : \mathbb{R}^3 \to \mathbb{R}^3$. System is nonlinear.
- Hyperbolic conservation law: Jacobian $F' = \frac{dF}{du}$ is diagonizable with real eigenvalues.
- Higher dimensions e.g. $u(x,y,t)$ or $u(\vec{r},t)$ with vector valued $F(u) = (F_x(u), F_y(u))$, i.e.,

$$\partial_t u + \partial_x F_x(u) + \partial_y F_y(u) = \partial_t u + \nabla \cdot F(u) = 0$$

Conservation property for scalar equation:

$$\frac{d}{dt} \int_{x_1}^{x_2} u(x,t)\, dx = -f(u(x_2,t)) + f(u(x_1,t)) \qquad \text{(1-dim.)}$$

$$\frac{d}{dt} \int_{\Omega} u\, d\Omega = -\oint_{\partial\Omega} F(u) \cdot d\vec{o} \qquad \text{(n-dim.)}$$

Consider simplest example: **Linear advection equation** $\partial_t u + c\partial_x u = \partial_t u + \partial_x(cu) = 0$ with $c = \text{const.}$.

Solution: $u(x,t) = u_0(x - ct)$. Lines $x - ct = x_0$ are **characteristics** of equation, $u$ is constant along these lines $x(t) = x_0 + ct$:

$$\frac{d}{dt}u(x_0 + ct, t) = u_t + cu_x = 0$$

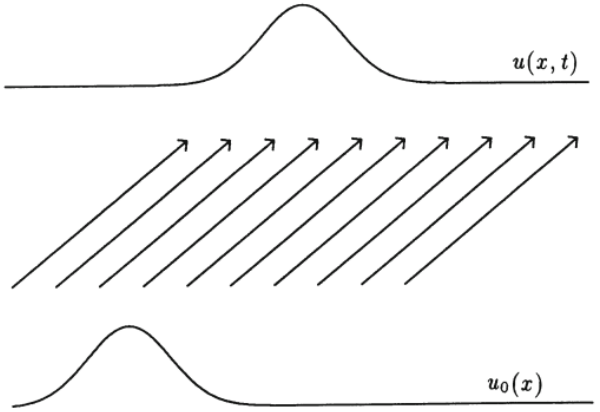



*Figure 3.1. Characteristics and solution for the advection equation.*

More general: Speed $c$ not constant, but given as $c(x)$:

- $\partial_t u + \partial_x(c(x)u) = 0$ or $(\partial_t + c(x)\partial_x)\, u(x,t) = -c'(x)u(x,t)$.
- **characteristics** are solutions to $x'(t) = c(x(t))$ with $x(0) = x_0$.
- solution $u$ is no more constant on characteristics, but fulfills ODE $\frac{d}{dt}u(x(t),t) = -c'(x(t))\, u(x(t),t)$ there.

### Non-smooth solutions, shocks

Simplest **nonlinear** example: Inviscous Burgers' equation, $\partial_t u + u\partial_x u = 0$ has flux function $F(u) = \frac{1}{2}u^2$.

Characteristics are $x'(t) = u(x(t), t)$ and $u =$ const. along characteristics. Shock at $T_b = -1/\min u_0'(x)$.

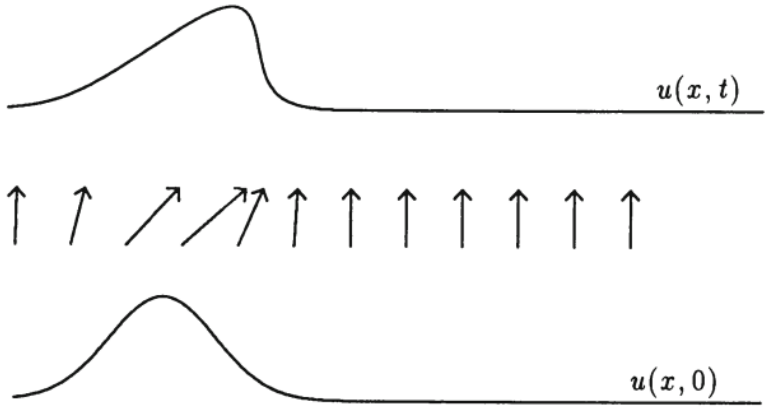



*Figure 3.3. Characteristics and solution for Burgers' equation (small t).*

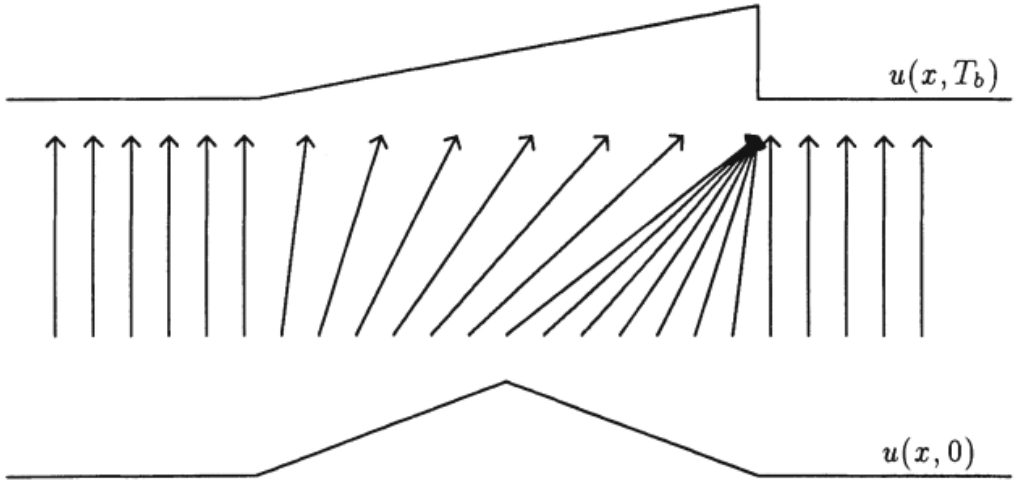



*Figure 3.4. Shock formation in Burgers' equation.*

In reality, there will be some kind of "diffusion" (viscosity, resistivity, ...), hence $\partial_t u + u\partial_x u = \varepsilon u_{xx}$ (viscous Burger's equation).

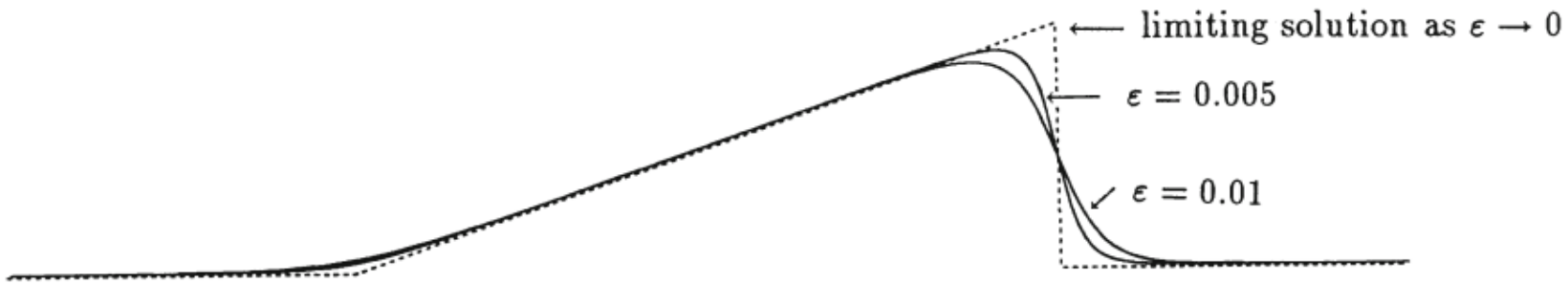



*Figure 3.7. Solution to the viscous Burgers' equation for two different values of $\epsilon$.*

Can define **non-smooth solution** with shocks etc. to PDE as **limiting case** $\lim_{\varepsilon\to 0}$. That's **difficult** in general.

New approach: Define **weak solution** to PDE: For any **test function** $\phi(x,t) \in C_0^1(\mathbb{R}\times\mathbb{R}^+)$ (differentiable, compact support), integrate PDE:

$$\int_0^\infty \int_{-\infty}^\infty (\phi\ \partial_t u + \phi\ \partial_x F(u))\ dx\, dt = 0$$

Integrate by parts: $u$ is said to be **weak solution** to PDE if for all $\phi \in C_0^1(\mathbb{R}\times\mathbb{R}^+)$

$$\int_0^\infty \int_{-\infty}^\infty ((\partial_t\phi)\ u + (\partial_x\phi)\ F(u))\ dx\, dt = -\int_{-\infty}^\infty \phi(x,0)\, u(x,0) dx$$

Differentiability condition moved over from $u$ to $\phi$ !

Every weak solution fulfills the PDE.

But: Weak solutions are not unique ! Find correct one by imposing additional **entropy condition**.

**Riemann problem**: Piecewise constant initial data,

$$u(x,0) = \begin{cases} u_l, & x < 0 \\ u_r, & x > 0 \end{cases}$$

Example Burgers' equation, $\partial_t u + u\partial_x u = \partial_t u + \partial_x \frac{u^2}{2} = 0$.

i) $u_l > u_r > 0$: Unique weak solution $u(x,0) = \begin{cases} u_l, & x < st \\ u_r, & x > st \end{cases}$ with shock speed $s = (u_l + u_r)/2$

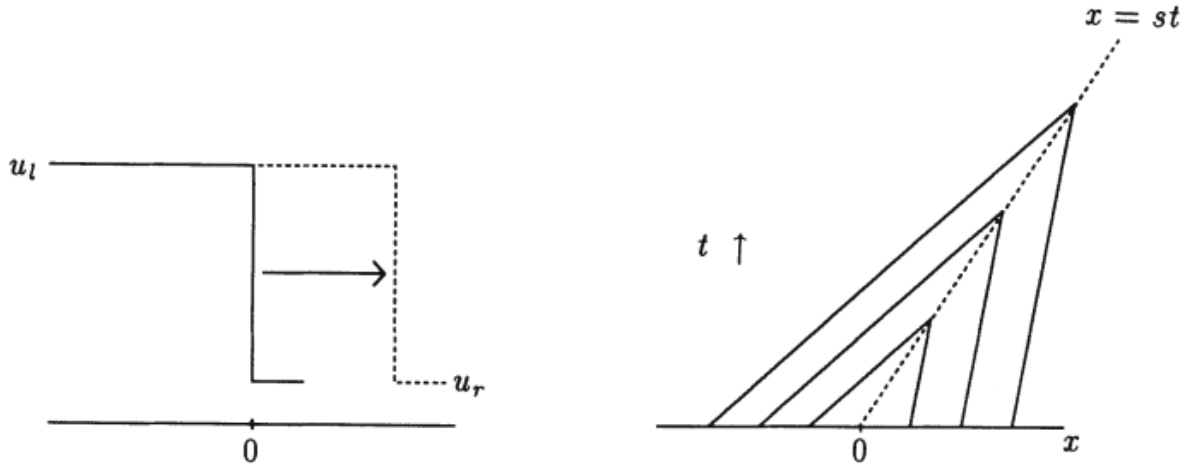



*Figure 3.8. Shock wave.*

Charactersitics must go **into** shock !

ii) $0 < u_l < u_r$: Infinitely many weak solutions, e.g.:

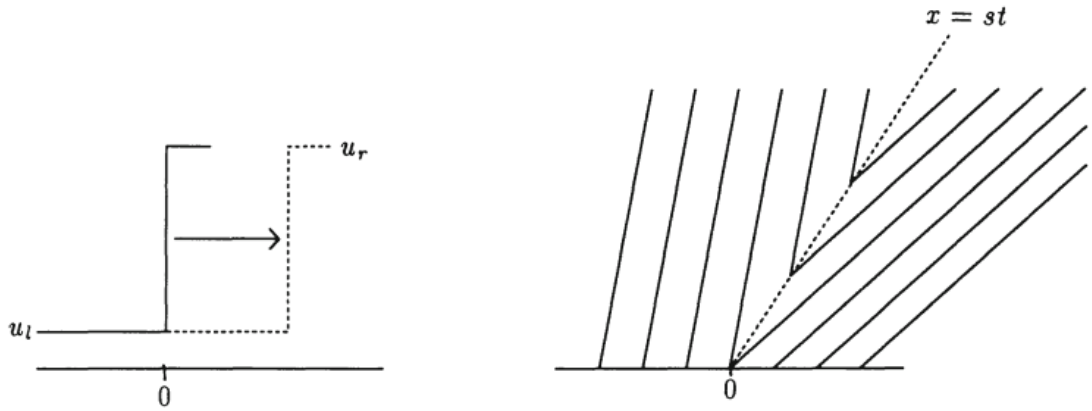



*Figure 3.9. Entropy-violating shock.*

**Wrong**, entropy-violating (characteristics going out of shock) !

---

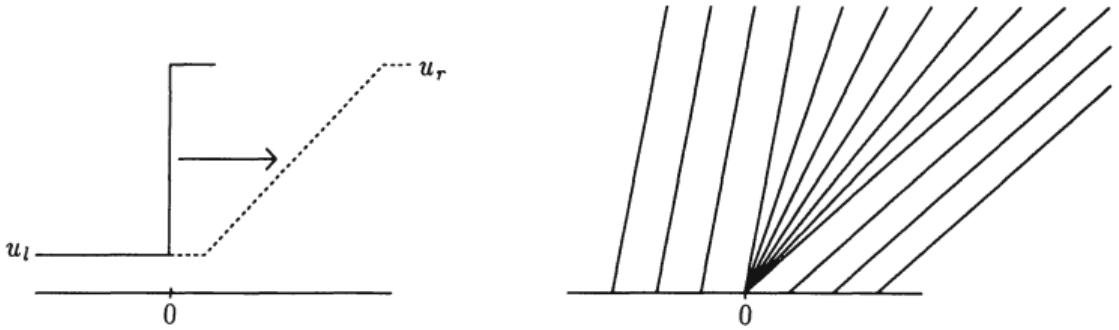



*Figure 3.10. Rarefaction wave.*

**Correct** rarefaction wave, no shock.

Challenge for "shock-capturing" numerical schemes: Reproduce/ approx. correct weak solutions!

### Rankine-Hugoniot conditions

**Shock speed** for scalar equation: Consider shock position $x_s$. Then,

$$\frac{d}{dt}\int_{x_s-\xi}^{x_s+\xi} u(x,t)\,dx = -F(x_s+\xi) + F(x_s-\xi)$$

On the other hand,

$$\frac{d}{dt}\int_{x_s-\xi}^{x_s+\xi} u(x,t)\,dx \approx -s(u(x_s+\xi) - u(x_s-\xi))$$

Limit $\xi \to 0$ gives **Rankine-Hugoniot jump condition** with shock speed $s$ as

$$s = \frac{F(u_r) - F(u_l)}{u_r - u_l} = \frac{[[F]]}{[[u]]}$$

For **systems**: $u$ and $F$ are vectors, $s$ is scalar. Linearized system $F(u) = Au$ with $A = \frac{dF}{du}$ yields condition

$$A(u_r - u_l) = s(u_r - u_l)$$

Shock speeds $s$ are eigenvalues of matrix $A$, jumps $(u_r - u_l)$ are associated eigenvectors.

Shock structure of adiabatic Euler equations. Riemann problem with initial jumps at $x = 0$.

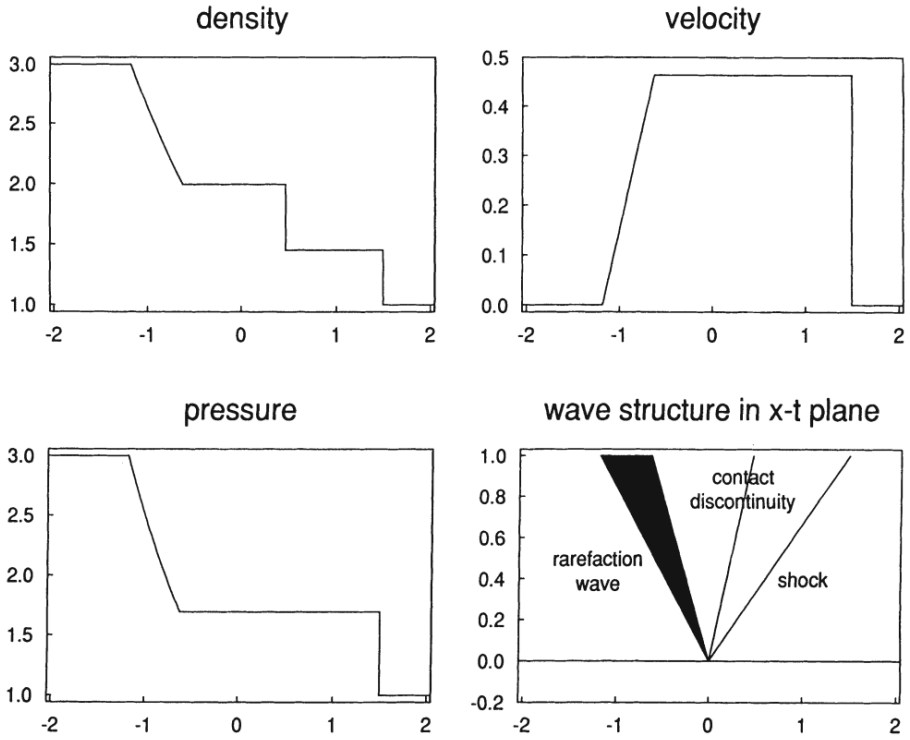



*Figure 1.1. Solution to a shock tube problem for the one-dimensional Euler equations.*

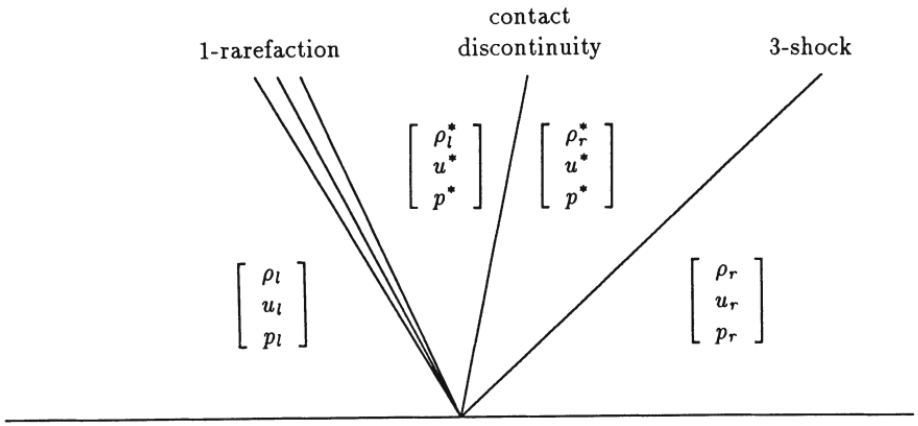
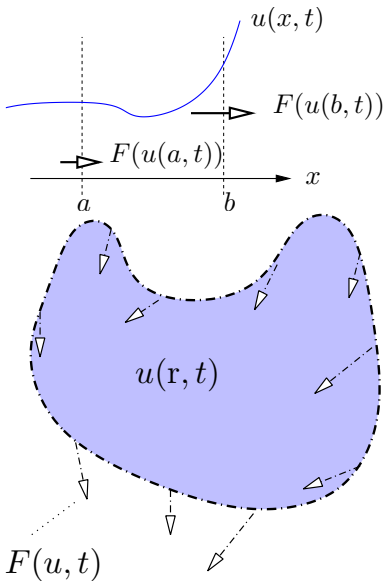



*Figure 9.1. Typical solution to the Riemann problem for the Euler equations.*

We know: Discontinous solutions are genuine in nonlinear systems, simplest case is Burgers' equation. How to handle shocks in general, ensure (correct) weak solution ?

Carry idea of weak solution over to numerical method: No differentiation (Taylor), use **integral** form instead !

**Finite Volume** approach for conservation laws $\partial_t u(x,t) = -\partial_x F(u,t)$. $F$ = flux function.




**Integrate** over $x \in [a,b]$ to get

$$\frac{d}{dt}\int_a^b u\,dx = -\{F(u(b,t),t) - F(u(a,t),t)\}$$

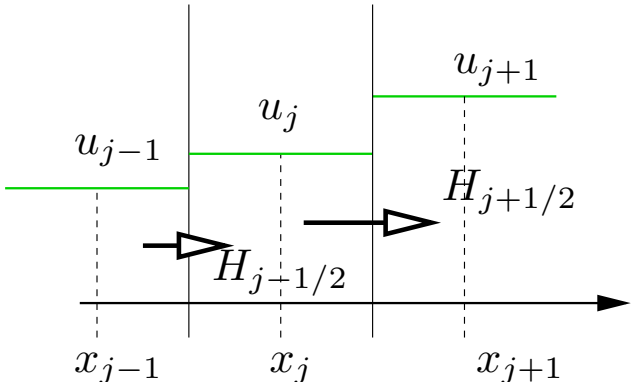
Higher dimensions:

$$\partial_t u = -\nabla\cdot\vec{F}(u,t) \Rightarrow \frac{d}{dt}\int_V u\,dV = -\oint_{\partial V}\vec{F}\cdot d\vec{s}$$

Finite Volume discretization applied to grid cell $I_j := [x_j - \frac{\Delta x}{2}, x_j + \frac{\Delta x}{2}]$:

$$u_j^n = \frac{1}{\Delta x}\int_{I_j} u(x,t^n)\,dx \quad \textbf{(cell average)}$$

$$H_{j+1/2}^n = \frac{1}{\Delta t}\int_{t^n}^{t^{n+1}} H(u(x_{j+1/2},t))\,dt \quad \text{(time averaged } \textbf{interface flux}\text{)}$$



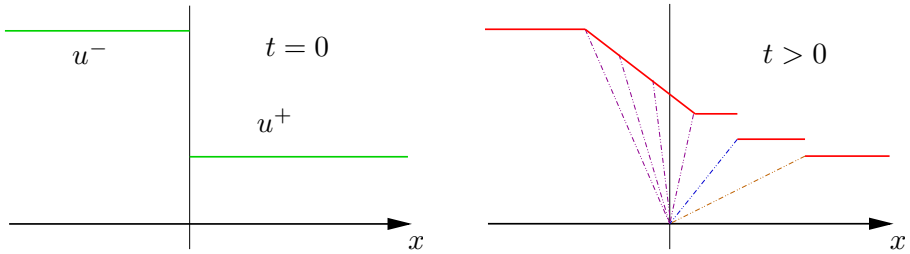

Finite Volume approach must catch weak solutions. Then, art is to "good" numerical fluxes $H$ at cell interfaces.

**Godunov theorem**: *Every linear scheme which preserves monotonicity independent of the intial conditions is at most* $O(\Delta x)$ *accurate.*

Linear scheme: $u_j^{n+1} = \sum_k \alpha_k u_{j-k}^n$ with $\alpha_k$=const., Examples: Finite difference schemes from above (FTCS, Lax-Wendroff, ...)

Looks like we can't get more than first order without producing oscillations. Unless we allow the scheme to be non-linear! How ?

Godunov's approach: Numerical flux is computed by solving 1-dimensional *Riemann problems* at each time step at each cell interface.




Gas dynamics: 3 "waves."

Initial constant left ($\vec{u}^-$) and right ($\vec{u}^+$) states at interface, fluctuations propagate as "Riemann fan".

Hence: At each cell interface $x_{j+1/2}$:

1. reconstruct $\vec{u}^-_{j+1/2}$ and $\vec{u}^+_{j+1/2}$ from $\vec{u}^n_j$, $\vec{u}^n_{j+1}$ (e.g. piecewise constant or linear w. limiter)
2. compute $\vec{u}^\star(x,t)$ as Riemann-solution
3. use Godunov interface flux $\vec{H}^{n+1/2}_{j+1/2} = \vec{F}(\vec{u}^\star(x_{j+1/2}, t^n + \frac{\Delta t}{2}))$

But: Non-trivial and *non-linear* function of cell averages $u^n_j$. Step 2 impossible for intricate systems.

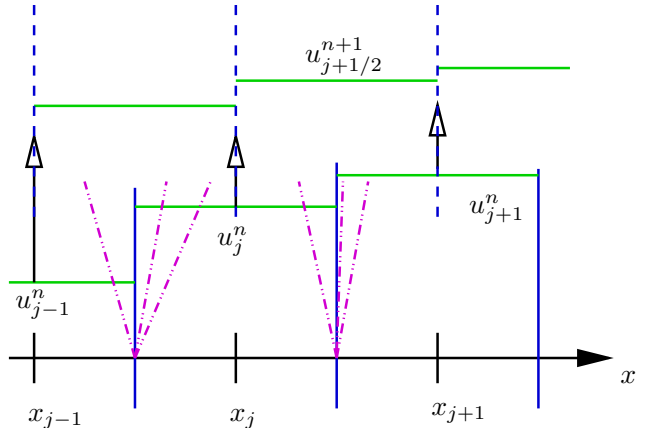
Many **approximate Riemann solvers** have been designed (see textbook by Toro).

**One** alternative to Riemann solvers: "Central schemes." Adopt Godunov's approach, but **integrate** (unknown) $\vec{u}^*$ over Riemann fan.

Simplest approach:

- piecewise constant reconstruction in each cell $u_j^n$.
- use new control volume on *staggered* mesh.

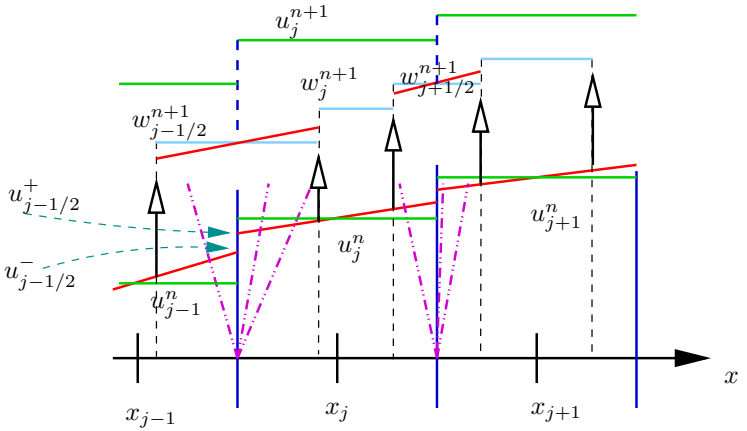
$$\begin{aligned}\vec{u}_{j+1/2}^{n+1} &= \frac{1}{\Delta x}\int_{x_j}^{x_{j+1}} u^n(x)\,dx - \frac{1}{\Delta x}\int_{t^n}^{t^{n+1}} F(\vec{u}_{j+1}) - F(\vec{u}_j)\,dt \\ &= \frac{\vec{u}_j^n + \vec{u}_{j+1}^n}{2} - \frac{\Delta t}{\Delta x}\left(F(\vec{u}_{j+1}^n) - F(\vec{u}_j^n)\right) + O(\Delta t^2)\end{aligned}$$




- $\Rightarrow$Lax-Friedrichs scheme on a staggered mesh! Excessive diffusion again.

Kurganov & Tadmor (2000) scheme:

1. Linear reconstruction of $\vec{u}$ from cell averages. Use limiters. Yields $\vec{u}_{j+1/2}^{\pm}$ at interfaces.
2. Integration over Riemann fans only, yields oversampled, non-equidistant solution $\vec{w}$ (smooth vs. non-smooth regions).
3. Linear reconstruction on oversampled grid
4. Integration back to original grid (no flux contribution)



Take limit $\Delta t \to 0$ to obtain *semi-discrete* scheme by Kurganov & Tadmor:

$$
\begin{aligned}
\frac{d\vec{u}_j}{dt} &= \lim_{\Delta t \to 0} \frac{\vec{u}_j^{n+1} - \vec{u}_j^n}{\Delta t} = -\frac{\vec{H}_{j+1/2}(t) - \vec{H}_{j-1/2}(t)}{\Delta x} \\
\vec{H}_{j+1/2} &= \frac{\vec{F}(\vec{u}^+_{j+1/2}) + \vec{F}(\vec{u}^-_{j+1/2})}{2} - \frac{a_{j+1/2}(t)}{2}\left[\vec{u}^+_{j+1/2} - \vec{u}^-_{j+1/2}\right] \\
a_{j+1/2} &= \max\left[\sigma(\vec{u}^+_{j+1/2}), \sigma(\vec{u}^-_{j+1/2})\right] \\
\sigma(\vec{u}) &= \text{maximum wave speed} \\
\vec{u}^+_{j+1/2}(t) &= u_{j+1}(t) - \frac{\Delta x}{2} g_{j+1}(t) \quad \text{reconstructed value from right} \\
\vec{u}^-_{j+1/2}(t) &= u_j(t) + \frac{\Delta x}{2} g_j(t) \quad \text{reconstructed value from left} \\
g_j &= \text{limited discrete derivative in cell (e.g. } \textit{MinMod}\text{)}
\end{aligned}
$$

- Apparently single grid (built-in oversampling and re-map), cell-centered
- Simple and robust. Only major ingredient: Estimate for phase speeds
- Transparent construction, room for "tuning"
- Simple addition of non-hyperbolic fluxes, source terms
- Straight-forward extension to multi-dimensional systems
- inherits 2nd order accuracy of reconstruction in smooth regions

Semi-discrete scheme can be integrated with ODE integrator, e.g. Runge-Kutta.

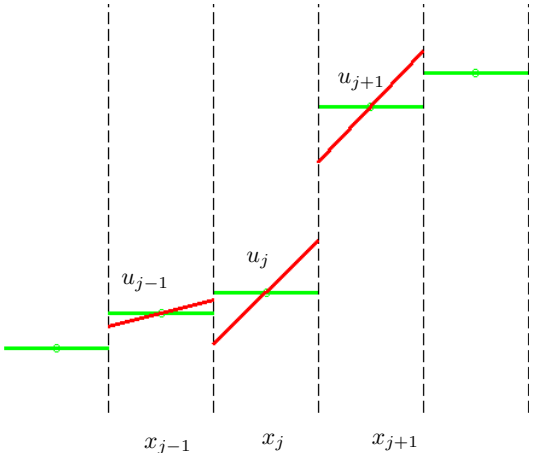
Favorable 3rd order low-storage scheme with good stability properties (Shu & Osher, 1988):

$$
\begin{aligned}
u^* &= u^n + \Delta t G(u^n, t) \\
u^{**} &= \frac{3}{4}u^n + \frac{1}{4}\left[u^* + \Delta t G(u^*, t^n + \Delta t)\right] \\
u^{n+1} &= \frac{1}{3}u^n + \frac{2}{3}\left[u^{**} + \Delta t G(u^{**}, t^n + \frac{\Delta t}{2})\right]
\end{aligned}
$$

Here: System of equations, conserved quantities $\vec{u}$ in each grid cell $j$: $\vec{G}_j(\vec{u}) = -\frac{\vec{H}_{j+1/2}(\vec{u}) - \vec{H}_{j-1/2}(\vec{u})}{\Delta x}$ with numerical flux $\vec{H}$ computed from $u$ using Kurganov-Tadmor scheme.

In each cell $I_j$, **reconstruct** $\tilde{u}_j(x)$ from cell averages. Piecewise linear reconstruction with slope limiters: $\tilde{u}_j(x) = u_j + g_j(x - x_j)$.

Unlimited central difference: $g_j = g_c := \frac{u_{j+1} - u_{j-1}}{2\Delta x}$.
Introduces oscillations (not monotone).

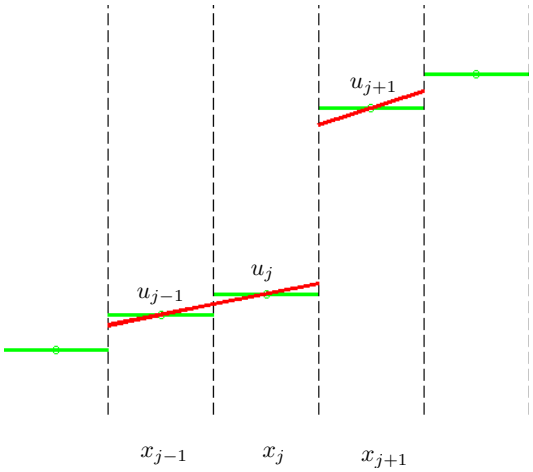



MinMod-limiter:

$$g^R := \frac{u_{j+1} - u_j}{\Delta x}, \quad g^L := \frac{u_j - u_{j-1}}{\Delta x}$$

$$g_j = \begin{cases} \operatorname{sign}(g^R) \min(|g^R|, |g^L|), & g^R g^L > 0 \\ 0, & g^L g^R < 0 \end{cases}$$

Yields monotone reconstruction.




There are many "sharper" limiter types than MinMod, but not strictly monotone (MinMod2, SuperBee, ...)

*Central weighted essentially non-oscillatory*. Levy, Puppo und Russo, 1999+: Piecewise parabolic reconstruction with CWENO-limiter, $O(\Delta x^3)$ accurate in smooth regions.

Simple central scheme time step summary: Integration time step $t^n \to t^{n+1}$ with central scheme,

- do 3 Runge-Kutta stages
- in each stage,
  1. reconstruct data at cell interfaces from left/right: $u^+$, $u^-$
  2. compute physical fluxes: $F(u^+)$, $F(u^-)$
  3. compute max. wave speed: $a = \max(\sigma(u^+), \sigma(u^-))$
  4. mix numerical flux $H$ according to formula

$$\vec{H}_{j+1/2} = \frac{\vec{F}(\vec{u}^+_{j+1/2}) + \vec{F}(\vec{u}^-_{j+1/2})}{2} - \frac{a_{j+1/2}(t)}{2}\left[\vec{u}^+_{j+1/2} - \vec{u}^-_{j+1/2}\right]$$

  5. apply flux to cell average u in each cell (Gauss)
  6. apply boundary conditions

Extension to higher dimensions: Compute fluxes in each direction separately, apply together. Or: Use multi-dimensional reconstruction.

# 6 Jacobi/ Gauß-Seidel Iteration, Multigrid Method

System of linear equations: $\underline{\underline{A}}\,\underline{x} = \underline{b}$. Example: 1-dim Poisson's equation,

$$\Delta f = \rho.$$

Finite Differences:

$$\frac{1}{\Delta x^2}\begin{pmatrix} -2 & 1 & & & & 1 \\ 1 & -2 & 1 & & & \\ & & \dots & \dots & \dots & \\ & & & 1 & -2 & 1 \\ 1 & & & & 1 & -2 \end{pmatrix}\begin{pmatrix} f_1 \\ f_2 \\ \dots \\ f_{m-1} \\ f_m \end{pmatrix} = \underline{\underline{A}}\,\underline{f} = \begin{pmatrix} \rho_1 \\ \rho_2 \\ \dots \\ \rho_{m-1} \\ \rho_m \end{pmatrix} = \underline{\rho}$$

- Standard problem in applied mathematics, solvers implemented in many software products
- Typically from discretization of linear PDEs with Finite Differences/ Elements/ Volumina
- **Direct solvers** based on elimination/ permutation (Gauß elimination, tridiagonal matrix algor., LU/ Cholesky decompositions & friends, ...)
- **(Fast) Fourier method**: $\hat{f} = -\hat{\rho}/k^2$ (only for simple settings).
- **Iterative solvers** construct (hopefully!) convergent sequence of approximations (gradient descent, conjugate gradients, GMRES, Jacobi & Gauß-Seidel, Multigrid)
- Selection of appropriate method depends on matrix properties (symmetry, spectral properties/ definiteness, band-/ block structure, sparsity, ...), matrix size, allowed tolerance, computing platform (parallel ?), ...

**Jacobi Iteration**: Split diagonal part $\underline{\underline{D}} = \mathrm{diag}(\underline{\underline{A}})$ to

$$(\underline{\underline{A}} - \underline{\underline{D}})\underline{f} - \underline{\rho} = -\underline{\underline{D}}\ \underline{f}$$

and iterate $\underline{f}^0 \to \underline{f}^1 \to \underline{f}^2 \to \ldots$ to **fix point** as

$$\underline{f}^{n+1} = \underline{\underline{D}}^{-1}\left[\underline{\rho} - (\underline{\underline{A}} - \underline{\underline{D}})\underline{f}^n\right]$$

Works if $\underline{\underline{A}}$ is diagonally dominant. Apply this to system above, with $\underline{\underline{D}}^{-1} = -\frac{\Delta x^2}{2}\,\underline{\underline{1}}$,

$$f_j^{n+1} = \frac{\Delta x^2}{2}\left[\frac{f_{j+1}^n + f_{j-1}^n}{\Delta x^2} - \rho_j\right] = f_j^n + \frac{\Delta x^2}{2}\left[\frac{f_{j+1}^n - 2f_j^n + f_{j-1}^n}{\Delta x^2} - \rho_j\right]$$

That's forward Euler applied to the diffusion equation

$$\frac{\partial f}{\partial t} = \Delta f - \rho$$

with largest stable (pseudo-)time step $\Delta t = \Delta x^2/2$ !

Hence, Jacobi iteration "diffuses" the error away towards stationary solution $\Delta f - \rho = 0$.

Numerical test: Provide some $\rho$, monitor **error** $\|\underline{f}^n - \underline{f}\|$ with known solution $\underline{f}$.

Or: As *discrete* solution is usually unknown, monitor (norm of) the **residual** or **defect** $\underline{d}^n = \underline{\rho} - \underline{\underline{A}}\,\underline{f}^n$.
Test reveals:

1. method works in principal, easy to generalize to more elaborate setups like 2-dim
2. but "Nyquist mode" in error/ residual survives (grid oscillations)
3. convergence is prohibitively slow for large matrices

Point 2: Iteration eigenvalue for Nyquist mode is $-1$. Can be resolved by modification to $\omega$-Jacobi method

$$\underline{f}^{n+1} = \underline{\underline{D}}^{-1}\left[\omega\underline{\rho} - (\omega\underline{\underline{A}} - \underline{\underline{D}})\underline{f}^n\right]$$

with parameter $0 < \omega \leq 1$, typically $\omega \approx 0.8$.
Apply to above problem:

$$\underline{f}^{n+1} = \underline{f}^n + \omega\frac{\Delta x^2}{2}\left(\underline{\underline{A}}\,\underline{f}^n - \underline{\rho}\right)$$

Hence, pseudo-time step is reduced to $\Delta t = \frac{\Delta x^2}{2}\,\omega$.

---

Similar related method, often easier to implement: **Gauß-Seidel** iteration,

$$f_j^{n+1} = \frac{\Delta x^2}{2}\left[\frac{f_{j+1}^n + f_{j-1}^{n+1}}{\Delta x^2} - \rho_j\right]$$

(in different flavors: $\omega$-GS, Red-Black-GS, ...).

Still open: Point 3, the slow convergence ! To understand, recall the interpretation of iterative solution as diffusion process, go back to continous formulation:

- exact (unknown) solution fulfills $\Delta f(\vec{r}) = \rho(\vec{r})$.
- (pseudo-)iterative solution $\tilde{f}(\vec{r}, t)$ follows $\frac{\partial \tilde{f}(\vec{r},t)}{\partial t} = \Delta \tilde{f}(\vec{r}, t) - \rho$
- then, (unknown) error $E(\vec{r}, t) := \tilde{f}(\vec{r}, t) - f(\vec{r})$ fulfills **defect equation**

$$\Delta E(\vec{r}, t) = -d(\vec{r}, t)$$

- error diffuses to zero according to homogeneous pendant $\frac{\partial E(\vec{r},t)}{\partial t} = \Delta E(\vec{r}, t)$
- error's Fourier components decay as $\sim e^{-k^2 t}$, i.e., by factor $e^{-k^2 \Delta t}$ per iteration step

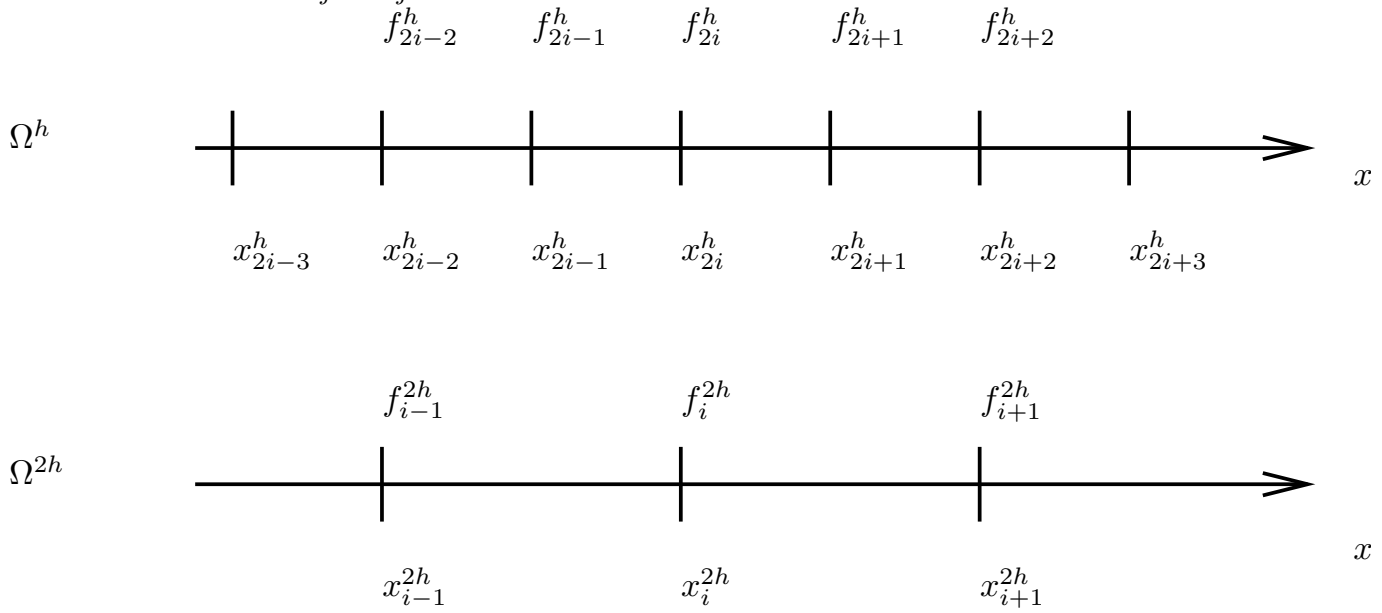
That's the dilemma: Stability dictates $\Delta t \leq \Delta x^2/2$ from *largest* $k_{=}\pi/\Delta x$.
Then, the longest wave length with *smallest* $k_1 = \pi/L = \pi/N\Delta x$ decays only with factor

$$\gamma_L = \exp\left\{-\frac{\pi^2 \Delta t}{N^2 \Delta x^2}\right\} = \exp\left\{-\frac{\pi^2}{2N^2}\right\}$$

per step, which becomes $\approx 1$ for $N \gg 1$ !
($\omega$-)Jacobi-/ Gauß-Seidel are good **smoothers** not good *solvers* !

Multigrid approach: Break the multi-scale dilemma by using multiple grids with different spacings $h \sim \Delta x$ to treat each error component appropriately. Here: **Nod-centered** grids.

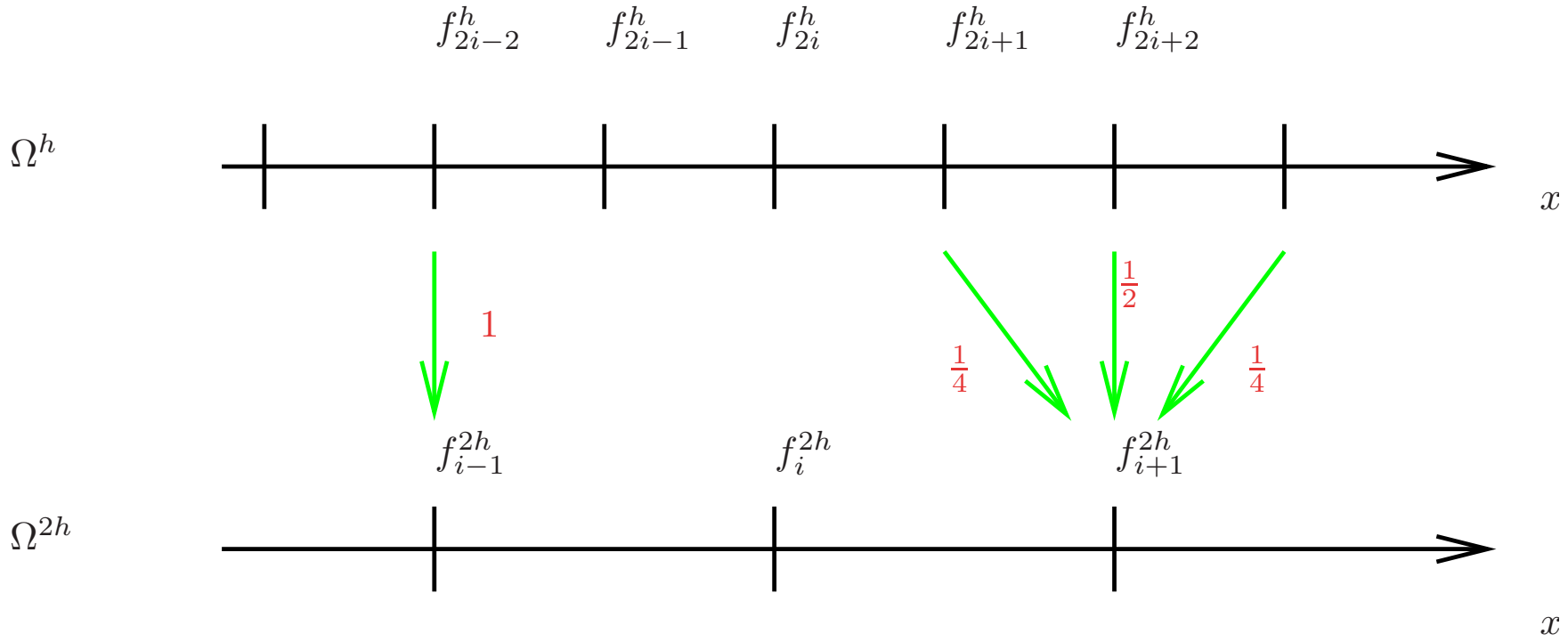
- original "fine" grid $\Omega^h$, spacing $h = \Delta x$, grid data $f_j^h$ (resp. $f_j^{h,n}$), $\rho_j^h$
- coarsened grid $\Omega^{2h}$, spacing $2h$, grid data $f_j^{2h}$, $\rho_j^{2h}$




- data transfer $\Omega^h \to \Omega^{2h}$: coarsening, **restriction**
- data transfer $\Omega^{2h} \to \Omega^h$: interpolation, **prolongation**
- "smoother" (Jac, GS) on all grids

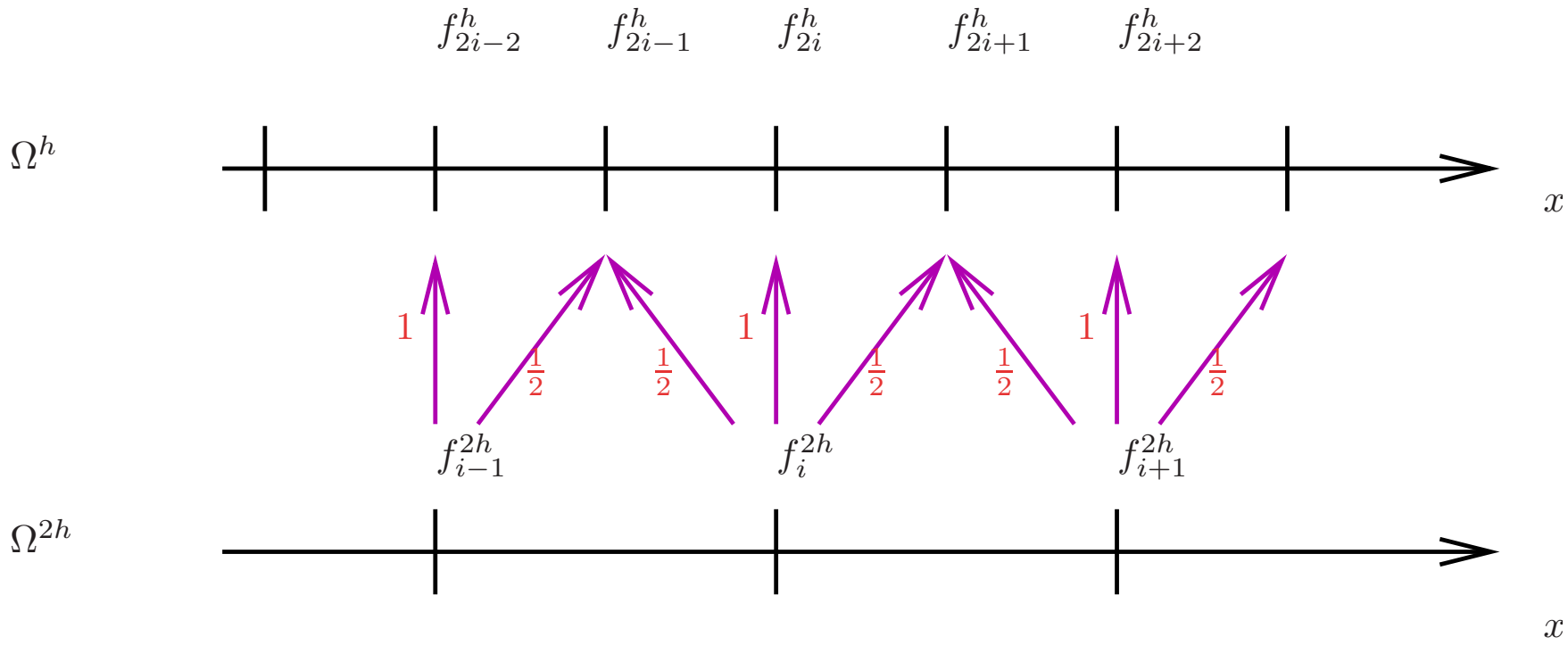
Examples for 1-dim restriction operators $\Omega^h \to \Omega^{2h}$, fine to coarse:

- Injection $f_i^{2h} = f_{2i}^h$ ("subsampling").
- Full-weighting $f_i^{2h} = \frac{1}{4} f_{2i-1}^h + \frac{1}{2} f_{2i}^h + \frac{1}{4} f_{2i+1}^h$ ("moving average").



Examples for 1-dim prolongation operators $\Omega^{2h} \to \Omega^h$, coarse to fine:

- Linear interpolation: $f^h_{2i} = f^{2h}_i$ and $f^h_{2i+1} = \frac{1}{2}(f^{2h}_i + f^{2h}_{i+1})$



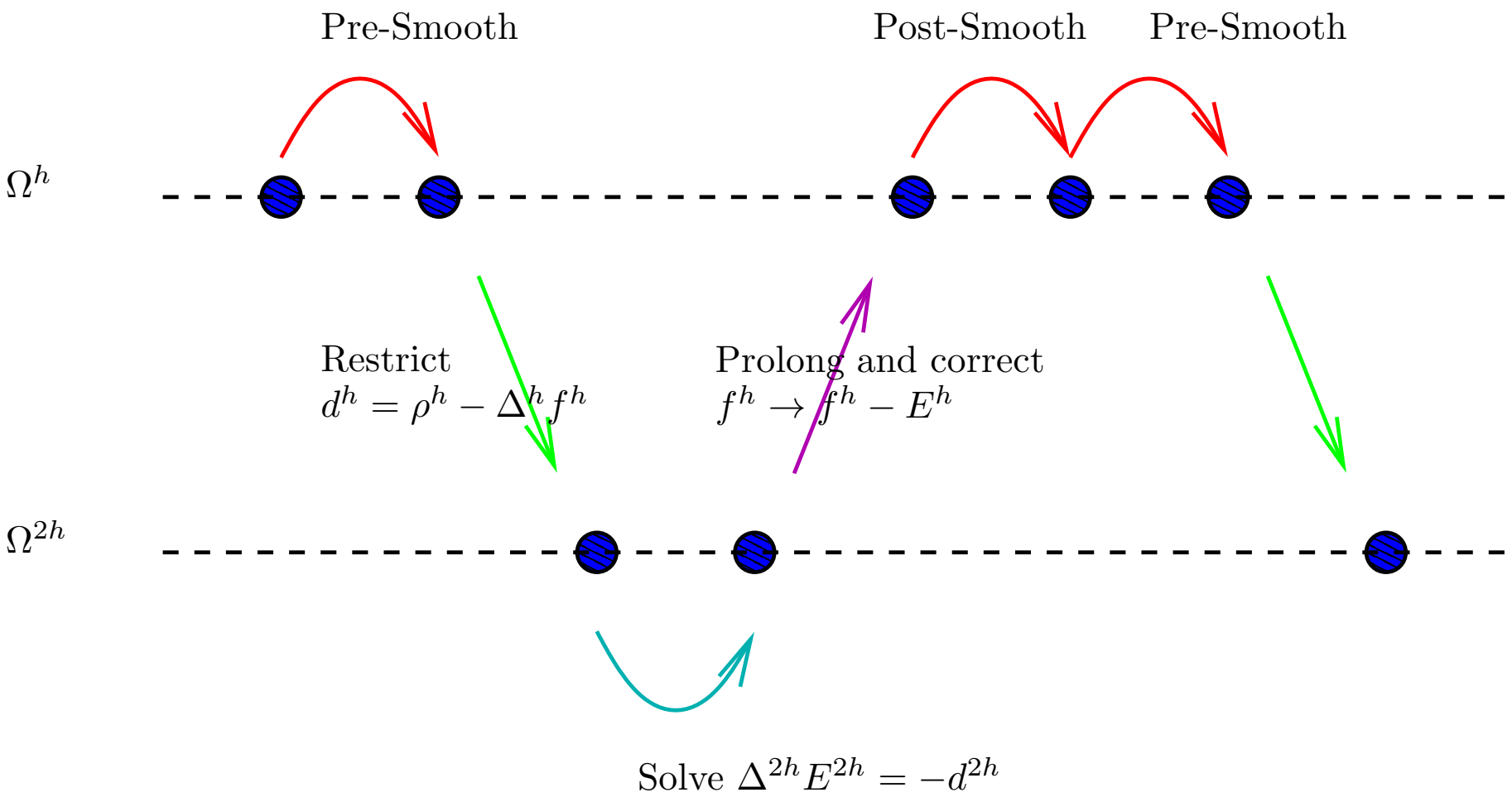

Remark: **Cell-centered** grid data will require other operators.

## Coarse grid correction

Recall: $\Delta f = \rho$, i.e., $\Delta E = -d$ with error $E := \tilde{f} - f$ and defect $\vec{d} := \rho - \Delta f$:

1. **One/ few** smoothing step(s) on $\Omega^h$, *pre-smoothing*.
   $\Rightarrow$ short wave length error modes are strongly dampened.
2. Compute defect $d = \rho - \Delta f$ on $\Omega^h$. Contains error modes w. long wave lengths $> \pi/h$.
3. Transfer (restrict) defect to coarse grid $\Omega^{2h}$.
4. Solve defect equation $\Delta E = -d$ on **coarse** grid.
   "Solution" is error $E$, contains long wave length modes of *error*.
5. Prolong *error* to fine grid.
6. Correct $f$ on fine grid: $f \to f - E$, **coarse grid correction**.
7. **One/ few** *post-smoothing* steps on $\Omega^h$.
8. Repeat entire scheme until defect has fallen below desired accuracy limit.

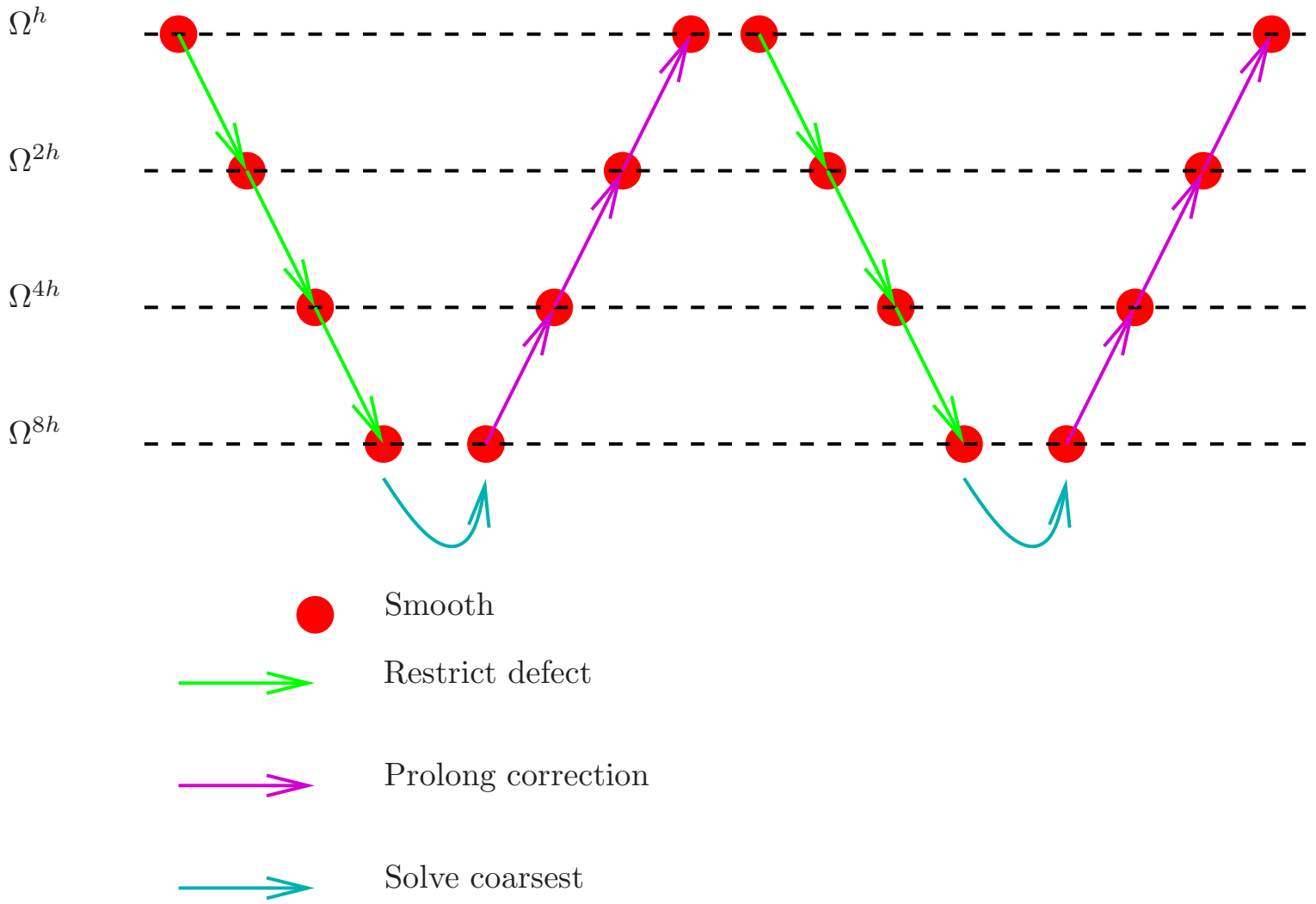
Important: Only errors/ defects are transferred between grids, not full solution !



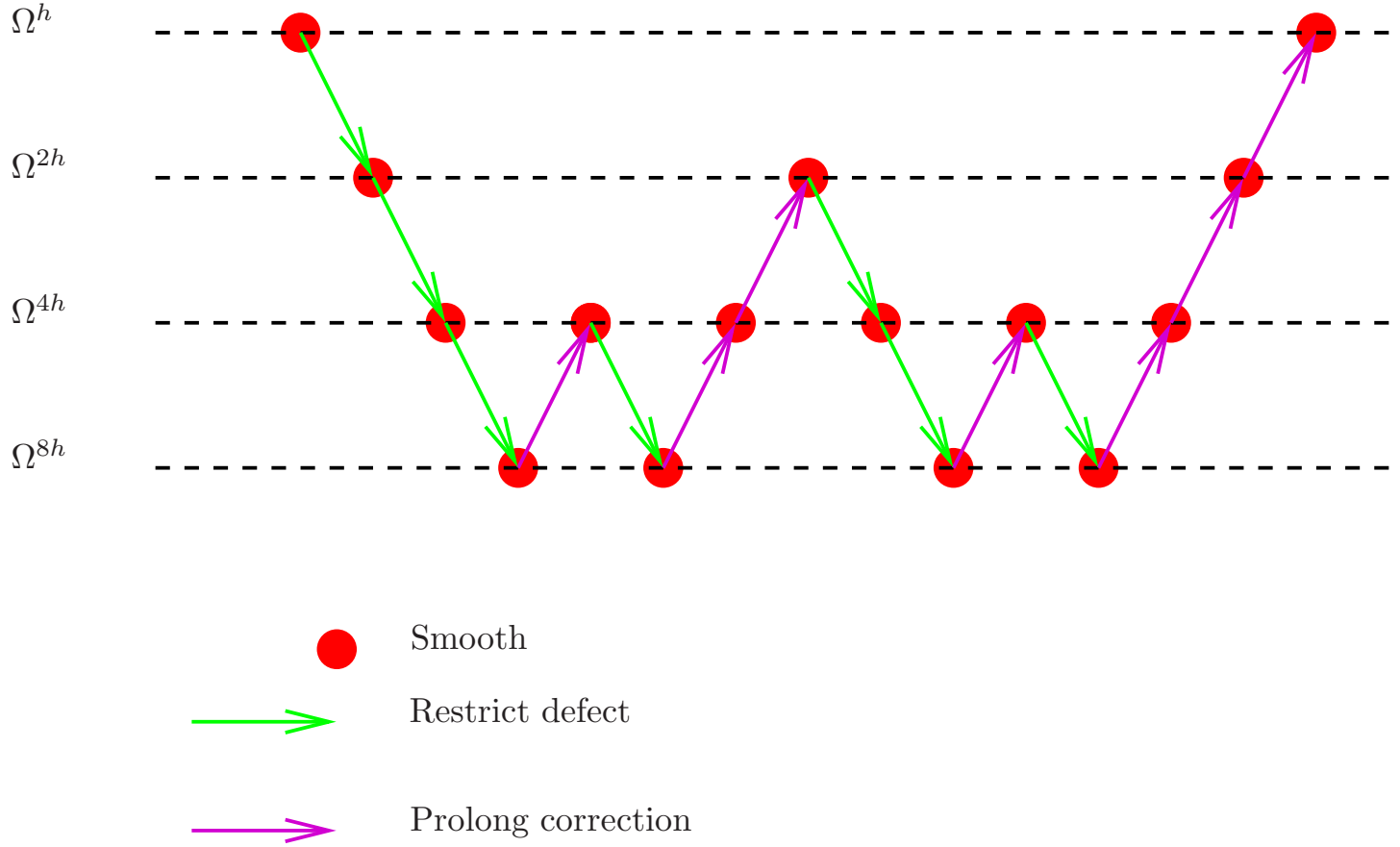

One question remains: How to solve defect equation $\Delta E = -d$ on $\Omega^{2h}$ ?

Defect equation $\Delta E = -d$ on $\Omega^{2h}$ is just another Poisson equation, this time on $\Omega^h$, hence solve again by coarse grid correction $\Omega^{2h} \to \Omega^{4h} \to \Omega^{2h}$ !

Recursive application up to some really coarse grid (small $N$), then use direct solver or a couple of (cheap) smoothing steps there.
**V-cycle**:



Alternatives: **W-cycle**, "full"-multigrid.




Multigrid in general:

- Overall convergence rate is *independent* of base resolution, complexity $O(N)$.
- Typical defect reduction by factor $\approx 10 - 20$ per cycle.