Introduction to Computer Simulation

Prof. Dr. Francesco Knechtli*

Dr. Tomasz Korzec† and Dr. Roman Höllwieser‡

Department of Physics, Bergische Universitt Wuppertal, Gaussstr. 20, D-42119 Wuppertal, Germany

January 31, 2022

Abstract

The goal of this lecture and tutorial series is to introduce concepts and applications of computer simulation. These lecture notes are based on the course "Computational Physics I" by Prof. Dr. Ulli Wolff and Dr. B. Bunk [1]. Further literature recommendations are [2–4]. At `http://csis.uni-wuppertal.de/courses/ics17.html`, you can find a sample of programs and scripts discussed in the lecture or in the tutorial. Life recording and lecture notes of the winter term 14/15 are available at `http://www.particle.uni-wuppertal.de/knechtli/`.

*knechtli@physics.uni-wuppertal.de, office D.10.24
†korzec@uni-wuppertal.de, office D.10.03
‡hoellwieser@uni-wuppertal.de, office G.11.37

Contents

1 Error, accuracy and stability

Computers store numbers with a finite number of bits (binary digits: 0 or 1)

Integer representation, exact or $\rightarrow$ floating point representation:

1.1 Range of numbers

- double precision numbers: 64 bits = 8 bytes (1 byte = 8 bits)
- single precision numbers: 32 bits = 4 bytes

number $= \pm\, M \cdot 2^E$ with exponent E and mantissa $M = \sum_i b_i 2^{-i}$, $b_i = 0/1$

	sign	exponent	mantissa	$\Rightarrow$ range
64 bits:	1 bit	11 bits	52 bits	$\approx 10^{-308} \ldots 10^{308}$
32 bits:	1 bit	8 bits	23 bits	$\approx 10^{-38} \ldots 10^{38}$

Remark: range depends on the choice of physical units

1.2 Accuracy/precision

number A in a computer: $A = M \cdot 2^E$

what is the smallest change δA such that $A + \delta A \neq A$?

$\rightarrow \delta A = 2^{-52} \cdot A \approx 2 \times 10^{-16} \cdot A$

fractional accuracy: relative change $\left|\dfrac{\delta A}{A}\right| \geq 2 \times 10^{-16}$

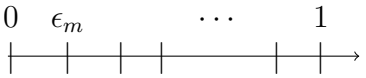
there exists a number $\epsilon_m > 0$, such that

$$\begin{aligned} 1 + z &= 1 \quad \text{for} \quad 0 \leq z \leq \epsilon_m \\ 1 + z &> 1 \quad \text{for} \quad z > \epsilon_m \end{aligned}$$

ϵ_m = machine accuracy/precision = intrinsic representation/roundoff error

the numbers A and $A + \epsilon_m \cdot A$ are numerically indistinguishable

mantissa M defines a grid of representable rational numbers with resolution ϵ_m:

$A = 2^E \times ($  $)$

a real number is approximated by the closest of these representable numbers

ϵ_m is a lower bound for roundoff error...let $A, B > 0$, $A \approx B$, $x = A - B$, *e.g.:*

$$\begin{aligned} A &= 0.375684\ldots \times 10^8 \qquad \text{(16digits)} \\ -B &= 0.375641\ldots \times 10^8 \qquad \text{(16digits)} \\ \hline = x &= 0.000043\ldots \times 10^8 \\ &= 0.43\ldots \times 10^4 \qquad \text{(12digits)} \end{aligned}$$

$\rightarrow$ significance loss...$\dfrac{|x|}{A} \approx 10^{-n}$ (above, $n = 4$)

uncertainty (roundoff error): $\left|\dfrac{\delta A}{A}\right| = \epsilon_m = \left|\dfrac{\delta B}{B}\right|$

$$\begin{aligned} \delta x &\approx |\delta A| + |\delta B| = \epsilon_m A + \epsilon_m B \approx \epsilon_m A \\ &\Rightarrow \left|\frac{\delta x}{x}\right| \approx \frac{\epsilon_m A}{10^{-n} A} = \epsilon_m 10^n \\ &\Rightarrow x \text{ is less precise than } A, n \text{ digits of precision are lost} \end{aligned}$$

Reason: cancellation between A and B (significance loss depends on the organization of computations)

1.3 Numerical derivative

function $f(x)$ is given, compute the derivative $f'(x) = \dfrac{df}{dx}(x)$

Taylor expansion:

$$\begin{aligned} f(x+h) &= f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + O(h^3) \\ f(x-h) &= f(x) - f'(x)h + \frac{1}{2}f''(x)h^2 + O(h^3) \end{aligned}$$

numerical derivative:

- asymmetric derivative: $f'(x) = \dfrac{f(x+h) - f(x)}{h} + O(h)$

- symmetric derivative: $f'(x) = \dfrac{f(x+h) - f(x-h)}{2h} + O(h^2)$

h small $\Rightarrow f(x+h) \approx f(x) \approx f(x-h) \Rightarrow$ significance loss, roundoff error

h large $\Rightarrow$ remainder of Taylor series is large, truncation error

$\Rightarrow$ optimal value of h?

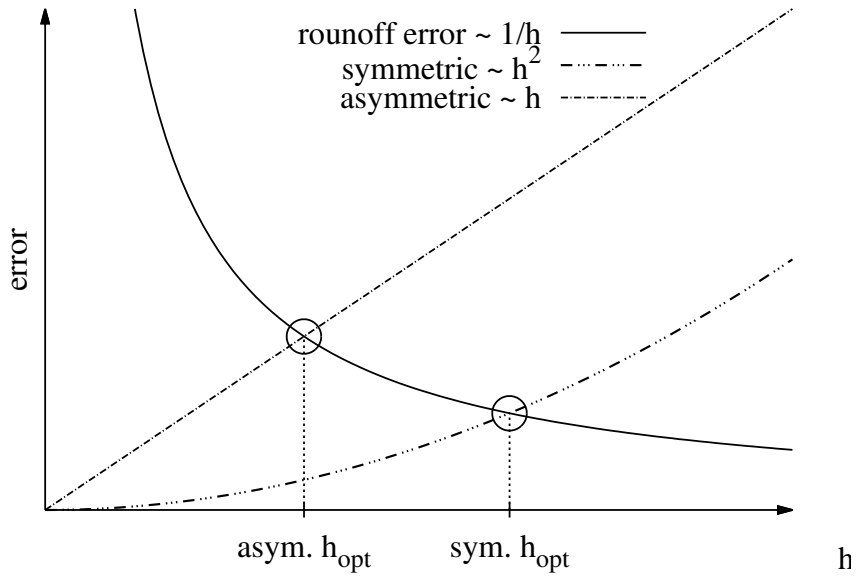
numerical experiment:
`http://csis.uni-wuppertal.de/courses/numerical_accuracy/numdiff.m`

```
% program numdiff
% tabulates the numerical derivatives of sin(x) at x=1
% as a function of the step-size
%
help numdiff
%
h = 10.^(-[1:16]); % step-sizes
x = 1;
dex = cos(1); % exakt
d1 = (sin(x+h)-sin(x))./h;          % asym. formula
d2 = (sin(x+h)-sin(x-h))./(h+h);    % sym. formula
y = [ h ; (d1-dex)/dex ; (d2-dex)/dex ];
%
fprintf('    h          asymm.       symm. \n\n')
fprintf(' %5.0e %10.1e %10.1e \n',y)
```

- asymmetric: $h_{opt} \sim 10^{-8}$, relative error $\sim 10^{-8}$

- symmetric: $h_{opt} \sim 10^{-5}$, relative error $\sim 10^{-11}$

roundoff error: $\delta f'(x) = \delta \left(\dfrac{f(x+h) - f(x)}{h} \right) \sim \dfrac{\epsilon_m f(x)}{h}$



optimal step-size or increment h_{opt}:

- asymmetric: $\frac{\epsilon_m}{h} \sim h \Rightarrow h_{opt} \sim \epsilon_m^{1/2} \sim 10^{-8} \Rightarrow$ error: $\sim 10^{-8}$
- symmetric: $\frac{\epsilon_m}{h} \sim h^2 \Rightarrow h_{opt} \sim \epsilon_m^{1/3} \sim 10^{-5} \Rightarrow$ error: $\sim 10^{-11}$

Remark: in these estimates we assumed $f(x), f'(x), f''(x) \sim O(1)$

1.4 Numerical limit

taking limits numerically is challenging, *e.g.*, $\lim_{h\to 0} \frac{f(x+h)-f(x)}{h}$ can be calculated only to $\Delta^{rel} \sim 10^{-8}$

another example:

$$
\begin{aligned}
e^x &= \lim_{n\to\infty}(1+\frac{x}{n})^n \quad \to \quad \text{approximate} \quad e^x \approx (1+\frac{x}{n})^n \quad \text{for large } n \\
e^x &= (e^{x/n})^n = (1+\frac{x}{n}+\frac{1}{2}\frac{x^2}{n^2}+\ldots)^n \quad | \quad \text{binomial expansion } (a+b)^n \\
&= (1+\frac{x}{n})^n + n(1+\frac{x}{n})^{n-1}\frac{1}{2}\frac{x^2}{n^2}+\ldots)
\end{aligned}
$$

$$= (1+\frac{x}{n})^n(1+n\frac{1}{1+\frac{x}{n}}\frac{1}{2}\frac{x^2}{n^2}+\ldots) \quad | \quad \text{large } n$$

$$= (1+\frac{x}{n})^n(1+\frac{1}{2}\frac{x^2}{n}+O(\frac{1}{n^2}))$$

$\rightarrow$ for large n, leading order correction $\delta \sim \frac{x^2}{2n}$

numerical experiment: $x = 1 \rightarrow e^1 = (1+\frac{1}{n})^n$ for large n

error: $\delta = \frac{e-(1+\frac{1}{n})^n}{e} \approx \frac{1}{2n} \Rightarrow n\delta = \frac{1}{2}$

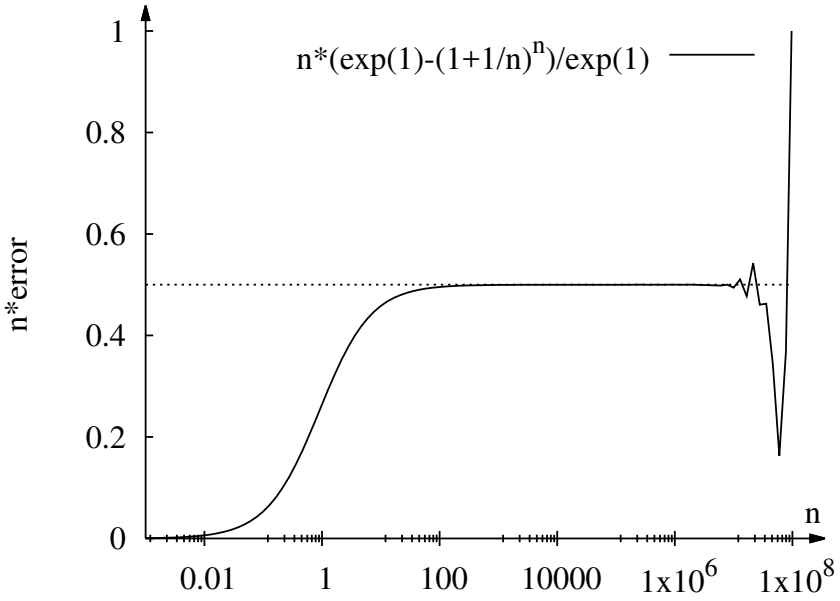
`http://csis.uni-wuppertal.de/courses/numerical_accuracy/test_exp_1.m`

```
1  % the program test_exp_1.m determines exp(x) with the help of
2  % the formula (1+x/n)^n and compares the result
3  % with the exact one.
4  %
5  clear
6  x=1.0;
7  n=10.^[1:12];
8  s=(1+x./n).^n;
9  delta=(s-exp(x))/exp(x);    % relative error
10 ndelta=delta.*n;
11
12 tab=[n;delta;ndelta];
13 fprintf('     n          delta          n*delta \n\n')
14 fprintf('%10.4e %11.4e %11.4e \n',tab)
```

$\Rightarrow$ up to $n \sim 10^7$ error decreases, $\delta \sim 10^{-8}$, $n\delta \sim 1/2$ as expected

$\Rightarrow$ larger n: $(1+1/n)^n$ becomes imprecise and δ does not decrease further

$\Rightarrow$ best achievable relative precision $\delta \sim 10^{-8}$ instead of 10^{-16}



explanation: $a = 1 + \frac{1}{n}$ has an error $\epsilon = 10^{-16}$

$\Rightarrow (a+\epsilon)^n \approx a^n(1 + n\frac{\epsilon}{a}) + O(\epsilon^2) \quad \Rightarrow a^n$ has a rounding error of $O(n\epsilon)$

$\Rightarrow$ balance between roundoff and truncation error $n\epsilon \sim \frac{1}{2n}$

$\Rightarrow$ optimal $n \approx \sqrt{1/\epsilon} \sim 10^8$, error $\delta \sim 10^{-8}$

$\Rightarrow (1 + \frac{x}{n})^n$ cannot give e^x with machine precision 10^{-16}

alternative: Taylor series

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

`http://csis.uni-wuppertal.de/courses/numerical_accuracy/test_exp_2.m`

```
1 % the program exp_test_2.m determines exp(x) with the help of
2 % the Taylor series and compares the result
3 % with the exact one.
```

```
4  %
5  clear
6  % computation of n!
7  %
8  x=1.0;
9  nfac(1)=1;
10 for i=2:20, nfac(i)=nfac(i-1)*i; end
11 %
12 % sum up the exponential series
13 %
14 s(1)=1+x;
15 for i=2:20, s(i)=s(i-1)+x^i/nfac(i); end
16 delta=(s-exp(x))/exp(x);
17 %
18 % printout
19 %
20 fprintf('   i        delta \n\n')
21 for i=1:20,
22     fprintf('%2i  %11.4e \n',i,delta(i))
23 end
```

$\Rightarrow$ for $x = 1$ we reach machine precision with $n = 17$ terms

1.5 Recurrence relations

often a sequence P_n, $n \in \mathbb{N}$, is defined recursively: $P_{n+1} = f(P_n)$, P_1 given, *e.g.:*

$$\begin{aligned} P_n &= \int_0^1 x^n e^x \, dx \\ P_{n+1} &= \int_0^1 x^{n+1} e^x \, dx = [x^{n+1} e^x]_0^1 - (n+1) \int_0^1 x^n e^x \, dx = e - (n+1) P_n \end{aligned}$$

recurrence relation, start with $P_0 = e - 1 \Rightarrow P_1 = 1 \Rightarrow \ldots$

`http://csis.uni-wuppertal.de/courses/numerical_accuracy/recurrence1.m`

```
% recurrence1.m
%
% compute integrals
%           1
%          /
%  p_n = | x^n exp(x) dx
%          /
%          0
%
% by using the recurrence
% p_1    = 1
% p_n+1 = e - (n+1) p_n

p(1) = 1;

for n=1:20
    p(n+1) = exp(1) - (n+1)*p(n);
    fprintf('p_%2d = %0.16f\n',n,p(n))
end
```

interesting behavior: $x^n e^x > 0$ for $x \in [0,1[$, but why is P_{19} negative?

$P_{n+1} < P_n$ because $x^{n+1}e^x = x \cdot x^n e^x$ where $x < 1$, but numerically $P_{18} > P_{17}$!

error propagation: let P_n^* be exact, numerically we have $P_n = P_n^* + \delta_n$

$$\begin{aligned} \Rightarrow P_{n+1} &= \underbrace{e - (n+1)P_n^*}_{\text{exact } P_{n+1}^*} \underbrace{-(n+1)\delta_n}_{\delta_{n+1}} \\ |\delta_{n+1}| &= (n+1)|\delta_n| = (n+1)n|\delta_{n-1}| = \ldots = (n+1)!|\delta_0| \end{aligned}$$

$|\delta_n| \propto n! \Rightarrow$ instable recurrence, even if $\delta_0 \sim 10^{-16}, \delta_{18} \sim 1$!

luckily, we can turn the table around:

backward iteration: $P_n = \dfrac{e - P_{n+1}}{n+1} \Rightarrow |\delta_n| = \dfrac{|\delta_{n+1}|}{n+1}$

$|\delta_n| \propto \frac{1}{n!} \Rightarrow$ stable recurrence, start at arbitrary value for high n

$\Rightarrow$ we get machine precision for P_n with small n, see example code

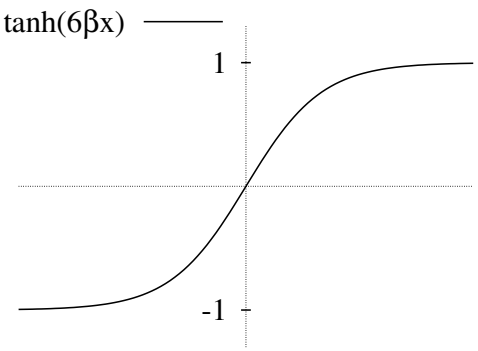
`http://csis.uni-wuppertal.de/courses/numerical_accuracy/recurrence2.m`

```
% recurrence2.m
%
% compute integrals
%           1
%          /
%  p_n = | x^n exp(x) dx
%          /
%          0
%
% by using the backward recurrence
% p_100    = 1
% p_n = (e - p_n+1) / (n+1)

p(100) = 1;

for n=99:-1:1
    p(n) = (exp(1) - p(n+1)) / (n+1);
end

for n=1:20
    fprintf('p_%2d = %0.16f\n',n,p(n))
end
```

2 Root finding

given: real function $x \to f(x)$, we look for roots x^*, *i.e.*, $f(x^*) = 0$

2.1 Example

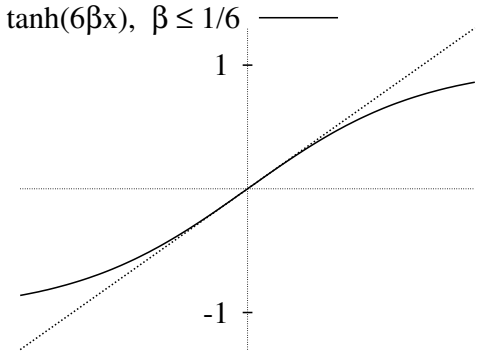


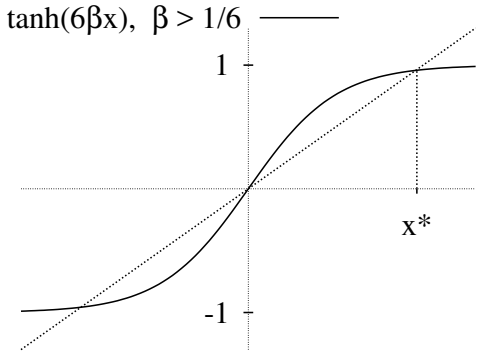
$f(x) = \tanh 6\beta x - x = 0$

$-\beta$ is some parameter

$\tanh z = \dfrac{e^z - e^{-z}}{e^z + e^{-z}}$

$x = 0$ is always a root, do other roots exist? yes, for $\beta > 1/6$!





$\Rightarrow$ transcendental equation, we have to compute the solution numerically:

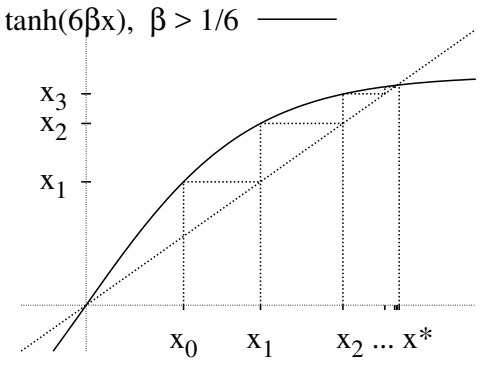
2.2 Special Method

for monotonic, concave functions:

$\to$ start with $x_0 > 0$

$\to x_{n+1} = \tanh 6\beta x_n$

$\lim_{n\to\infty} x_n = x^*$



Convergence Analysis

general case: solve $g(x) = h(x)$ through sequence $x_{n+1} = g^{-1}(h(x_n))$

x^* is the solution $g(x^*) = h(x^*)$

$x_n = x^* + \delta_n$, where δ_n is the distance from the solution

$g(x_{n+1}) = h(x_n) \rightarrow g(x^* + \delta_{n+1}) = h(x^* + \delta_n)$

Taylor: $g(x^*) + \delta_{n+1} g'(x^*) + \ldots = h(x^*) + \delta_n h'(x^*) + \ldots \Rightarrow \delta_{n+1} \approx \delta_n \dfrac{h'(x^*)}{g'(x^*)}$

in order to have $\delta_{n+1} < \delta_n$, we need $\left| \dfrac{h'(x^*)}{g'(x^*)} \right| < 1$

Note, **not** needed for all x, only at x^*; in this case: **linear** convergence:

$|\delta_{n+1}| = c|\delta_n|$, where $0 < c < 1$

$\rightarrow |\delta_n| = c^n |\delta_0| = e^{-n|\ln 1/c|} |\delta_0|$

$\rightarrow$ initial error decreases **exponentially** with n.

How many steps do we need for machine precision, *i.e.*, $|\delta_n| \approx$ `eps` $|\delta_0|$, where the MATLAB constant `eps` $= \epsilon_m \approx 10^{-16}$:

$c^n |\delta_0| =$`eps` $|\delta_0| \Rightarrow \log$ `eps`$/\log c = n \Rightarrow n = -16/\log_{10} c$

convergence depends on c, for $c \rightarrow 1, \log_{10} c \rightarrow 0 \Leftrightarrow n \rightarrow \infty$

$\rightarrow$ `convergence.m` (not available on the web $\rightarrow$ exercise 2):

- inline functions `f=@(x)` $\tanh 6\beta x - x$
- `fzero(f,`x_0`)` - `feval(f,x)`
- semilogy $\rightarrow \ln |\delta_n| = n \ln c + \ln |\delta_0|$

```
- while a > 0
      ⋮
   a = a - 1
end
```

are there better methods? more general? better convergence? less complex?

2.3 Bisection Method

Assumption: $f(x)$ changes sign once in $I = [x_1, x_2]$

$\rightarrow$ divide interval by 2 and continue with interval where $f(x)$ changes sign

$\rightarrow$ repeat until interval shrinks to `eps`

convergence: $\delta_0 = |x_2 - x_1| \rightarrow \delta_1 = |x_2 - x_1|/2 \ldots \delta_n = |x_2 - x_1|/2^n$

$$\delta_{n+1} = \tfrac{1}{2}\delta_n \quad \Rightarrow \quad \text{linear convergence again}$$

to get $\delta_n = \texttt{eps} = |x_2 - x_1|/2^n \Rightarrow n = \log_2 \frac{|x_2 - x_1|}{\texttt{eps}} \overset{|x_2 - x_1| \approx 1}{\approx} \log_2 2^{52} = 52$

$\rightarrow$ `http://csis.uni-wuppertal.de/courses/root_finding/bisect.m`

```
1  % file bisect.m
2  % program for finding zeros with the bisection method
3  % call: x0 = bisect(func,x1,x2,tol)
4  % x1,x2: start interval; x0: root of func
5  % IT MUST BE: x1 < x2, func(x1)*func(x2) <= 0
6  % func: function name as string constant, for example 'sin'
7  % tol: desired absolute accuracy
8  %
9    function [x0] = bisect(func,x1,x2,tol)
10   f1=feval(func,x1); if f1 == 0, x0=x1; return; end
11   f2=feval(func,x2); if f2 == 0, x0=x2; return; end
12   if f1*f2 >= 0 | x1 >= x2
13     error('bad choice of x1, x2');
14   end
15   for i=1:100 % bisection loop:
16     x0 = 0.5*(x2+x1);
17     if x2 - x1 < tol return; end
18     f0=feval(func,x0);
19     if f0*f1 <= 0          % root is between x0 and x1
20       x2=x0; f2=f0;
21     else                   % root is between x0 and x2
22       x1=x0; f1=f0;
```

```
23        end
24      end
25      error(' did not find a zero')
```

`[x0]=bisect(func,x1,x2,tol); func='sin'; func='mffunc';` (file mffunc.m)

`bisect('sin',3,4,1e-15)` $\rightarrow \pi$; `func=f;` (inline funct., no *.m necessary)

2.4 Newton-Raphson Method

root: $f(x^*) = 0$

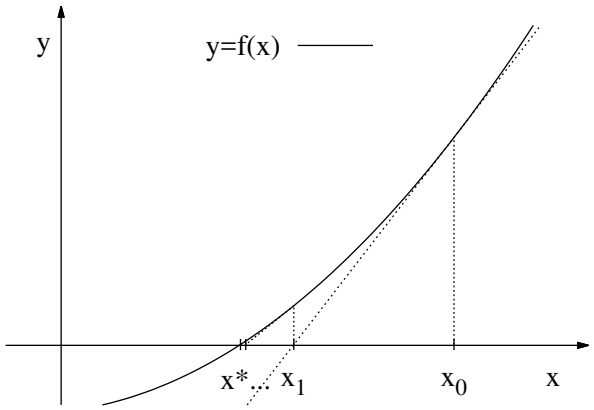
tangent at x_n:

$$f(x) \approx f(x_n) + (x - x_n) f'(x_n)$$

next point x_{n+1}:

$$0 = f(x_n) + (x_{n+1} - x_n) f'(x_n)$$

$$\Rightarrow x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad \rightarrow \quad \text{Newton-Raphson Iteration}$$



Assuming the derivative $f'(x)$ is known analytically; if not, use secant method in 2.5, better than using a numerical approximation for $f'(x)$.

Convergence Analysis

$$x_n = x^* + \delta_n \Rightarrow \overbrace{x_{n+1} - x^*}^{=\delta_{n+1}} = \overbrace{x_n - x^*}^{=\delta_n} - \frac{f(x_n)}{f'(x_n)}$$

$$\begin{aligned} f(x_n) &= \underbrace{f(x^*)} + (x_n - x^*) f'(x^*) + \frac{1}{2}(x_n - x^*)^2 f''(x^*) + O((x_n - x^*)^3) \\ &= \quad 0 \quad + \quad \delta_n f'(x^*) \quad + \quad \frac{1}{2}\delta_n^2 f''(x^*) \quad + \quad O(\delta_n^3) \\ f'(x_n) &= f'(x^*) + \delta_n f''(x^*) + O(\delta_n^2) \end{aligned}$$

$$\begin{aligned}
\delta_{n+1} &= \delta_n - \frac{\delta_n f'(x^*) + \frac{1}{2}\delta_n^2 f''(x^*) + O(\delta_n^3)}{f'(x^*) + \delta_n f''(x^*) + O(\delta_n^2)} \\
&= \delta_n - \delta_n \frac{1 + \frac{1}{2}\delta_n f''(x^*)/f'(x^*) + O(\delta_n^2)}{1 + \underbrace{\delta_n f''(x^*)/f'(x^*) + O(\delta_n^2)}_{=z}}
\end{aligned}$$

geometric series: $\frac{1}{1-z} = \sum_{k=0}^{\infty} z^k, \quad |z| < 1 \quad \Rightarrow \quad \frac{1}{1+z} \approx 1 - z$

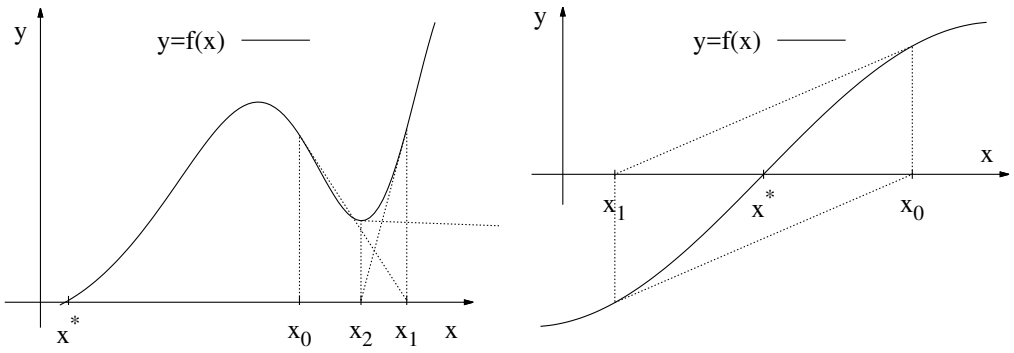
$$\begin{aligned}
\Rightarrow \delta_{n+1} &= \delta_n - \delta_n[1 + \frac{1}{2}\delta_n f''(x^*)/f'(x^*) + O(\delta_n^2)][1 - \delta_n f''(x^*)/f'(x^*) + O(\delta_n^2)] \\
\delta_{n+1} &= \delta_n - \delta_n[1 + \frac{1}{2}\delta_n f''(x^*)/f'(x^*) - \delta_n f''(x^*)/f'(x^*) + O(\delta_n^2)] \\
\Rightarrow \delta_{n+1} &= \frac{1}{2} f''(x^*)/f'(x^*)\delta_n^2 + O(\delta_n^3)
\end{aligned}$$

quadratic convergence $\delta_{n+1} \propto \delta_n^2$

$\Rightarrow$ with each iteration the number of exact digits is doubled!

WARNING: we truncated the Taylor expansion of $f(x)$

$\Rightarrow$ no guarantee of convergence!!! *e.g.*, bad choice of starting value x_0



method of choice close to convergence: bisection $\rightarrow$ close to x^* $\rightarrow$ Newton-Raphson to get x^* with ϵ_m (root polishing)

2.5 The Secant Method

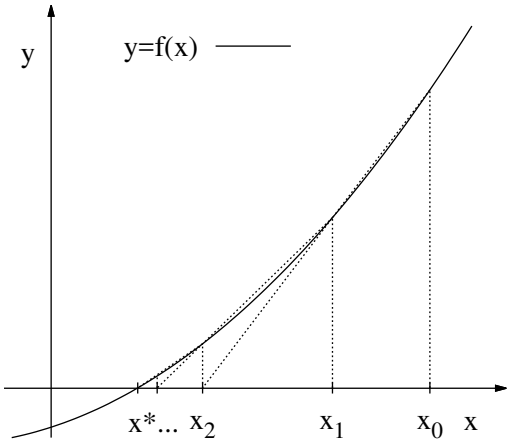
tangent $\rightarrow$ secant

$$f'(x_n) \rightarrow \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$$

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

2-step iteration:

$$(x_{n-1}, x_n) \rightarrow x_{n+1}$$



recall, significance loss in $f(x_n) - f(x_{n-1})$!

stopping condition: $|f(x_n) - f(x_{n-1})| < \tau |f(x_n)|$ with tolerance $\tau > \epsilon_m$

Convergence Analysis

$x_{n-1} = x^* + \delta_{n-1}, x_n = x^* + \delta_n, x_{n+1} = x^* + \delta_{n+1}$

$$\Rightarrow x_{n+1} - x^* = x_n - x^* + f(x_n)\frac{x_n - x^* - (x_{n-1} - x^*)}{f(x_n) - f(x_{n-1})}$$

$$\begin{aligned}
f(x_n) &= \underbrace{f(x^*)}_{=0} + \delta_n f'(x^*) + \frac{1}{2}\delta_n^2 f''(x^*) + O(\delta_n^3) \\
f(x_{n-1}) &= \overbrace{f(x^*)} + \delta_{n-1} f'(x^*) + \frac{1}{2}\delta_{n-1}^2 f''(x^*) + O(\delta_{n-1}^3) \\
\delta_{n+1} &= \delta_n - (\delta_n - \delta_{n-1})\frac{\delta_n f'(x^*) + \frac{1}{2}\delta_n^2 f''(x^*) + O(\delta_n^3)}{(\delta_n - \delta_{n-1}) f'(x^*) + \frac{1}{2}(\delta_n^2 - \delta_{n-1}^2) f''(x^*) + O(\delta^3)} \\
&= \delta_n - \delta_n \frac{f'(x^*) + \frac{1}{2}\delta_n f''(x^*) + O(\delta_n^2)}{f'(x^*) + \frac{1}{2}(\delta_n + \delta_{n-1}) f''(x^*) + O(\delta^2)} \quad / \quad \delta \approx \delta_n \approx \delta_{n-1} \\
&= \delta_n - \delta_n \frac{1 + \frac{1}{2}\delta_n f''(x^*)/f'(x^*) + O(\delta^2)}{1 + \frac{1}{2}(\delta_n + \delta_{n-1}) f''(x^*)/f'(x^*) + O(\delta^2)}
\end{aligned}$$

$\rightarrow$ use geometric series...

$$
\begin{aligned}
\Rightarrow \delta_{n+1} &= \delta_n - \delta_n[1 + \frac{1}{2}\delta_n \frac{f''(x^*)}{f'(x^*)} + O(\delta^2)][1 - \frac{1}{2}(\delta_n + \delta_{n-1})\frac{f''(x^*)}{f'(x^*)} + O(\delta^2)] \\
&= \delta_n - \delta_n[1 + \frac{1}{2}\delta_n \frac{f''(x^*)}{f'(x^*)} - \frac{1}{2}(\delta_n + \delta_{n-1})\frac{f''(x^*)}{f'(x^*)} + O(\delta^2)] \\
\Rightarrow \delta_{n+1} &= \frac{1}{2}\frac{f''(x^*)}{f'(x^*)}\delta_n \delta_{n-1} + O(\delta^3)
\end{aligned}
$$

something between linear and quadratic convergence $\delta_{n+1} \propto \delta_n \delta_{n-1}$

ansatz: $|\delta_{n+1}| \approx c \cdot |\delta_n|^\alpha \Rightarrow |\delta_n|^\alpha \approx |\delta_n| \cdot |\delta_n|^{1/\alpha}$

$\Rightarrow \alpha = 1 + 1/\alpha \Rightarrow \alpha(\alpha - 1) = 1 \Rightarrow \alpha = \frac{1}{2}(1 + \sqrt{5}) \approx 1.62$

Remark: Fibonacci Numbers: $\lambda_0 = 1, \lambda_1 = 1, \lambda_{n+2} = \lambda_{n+1} + \lambda_n$

$\Rightarrow \lambda_2 = 2, \lambda_3 = 3, \lambda_4 = 5, \lambda_5 = 8, \lambda_6 = 13, \ldots$

one can show: $\lambda_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \left(\frac{1-\sqrt{5}}{2}\right)^{n+1}\right]$

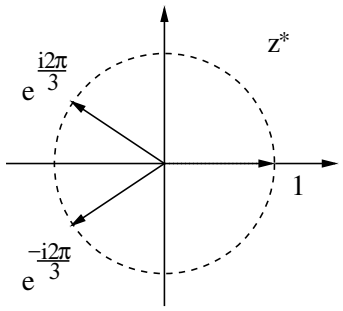
Proof. $\begin{pmatrix} \lambda_{n+2} \\ \lambda_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n+1} \begin{pmatrix} \lambda_1 \\ \lambda_0 \end{pmatrix}$

$\Rightarrow$ eigenvalues λ: $(1 - \lambda)(-\lambda) - 1 = 0$ □

2.6 Newton-Raphson and Fractals

$f(z) = z^3 - 1$, complex function $z \in \mathbb{C}$

$f(z) = 0$ for $z = z^* = 1, e^{i\frac{2\pi}{3}}, e^{-i\frac{2\pi}{3}}$



Newton-Raphson (complex):

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)} = z_n - \frac{z_n^3 - 1}{3z_n^2}$$

Question: given a starting value z_0, to which z^* will z_n converge?

observation: $z_0, z_1, \ldots \to z^*$

$e^{\pm i\frac{2\pi}{3}} z_0, e^{\pm i\frac{2\pi}{3}} z_1, \ldots \to e^{\pm i\frac{2\pi}{3}} z^*$

symmetry: if $z_0 \to e^{\pm i\frac{2\pi}{3}} z_0$ then $z_n \to e^{\pm i\frac{2\pi}{3}} z_n \; \forall \, n$

it is sufficient to study convergence to $z^* = 1$. basin of attraction $\to$ fractal
if z_0 converges to $z^* = 1$, then $e^{\pm i\frac{2\pi}{3}} z_0$ converges to $z^* = e^{\pm i\frac{2\pi}{3}}$

MATLAB programs `nr3frac.m` (main program) and `nr3.m` (returns root):

`http://csis.uni-wuppertal.de/courses/root_finding/nr3frac.m`

```
%
%  file nr3frac.m
%
%  For a given lattice in the complex plane
%  the program decides which points converge
%  with Newton-Raphson to the solution z=1
%  of the equation z^3-1=0
%
global tol; tol=1.e-13; % convergence criterion for N.-R.
tol10=10*tol;
%
t=cputime; % for determining the CPU time
size=199.5;
maxy=2.0;
re=[-size:size]*(maxy/size); % lattice values for the real part
im=re*sqrt(-1);              % lattice values for the imaginary part
%
n=length(re);
m=zeros(n,n); % matrix of zeros is predefined
%
for i=1:n, for j=1:n
   m(i,j) = (abs(nr3(re(j)+im(i))-1) < tol10);
   % m(,)=1 if and only if N.-R. -> 1
end,end
%
```

```
26 cputime-t % determines and outputs the CPU time consumed
27 spy(m);
28 title('Fractal with Newton-Raphson, z^3-1=0');
```

`http://csis.uni-wuppertal.de/courses/root_finding/nr3.m`

```
1 %
2 %  file nr3.m
3 %
4 %  Newton-Raphson iteration for the complex function z^3-1
5 %
6 %  Argument: start value; Answer: zero
7 %  tol is needed as global variable
8 %
9 function [z] = nr3(zn);
10 global tol;
11 zo = inf;
12 del = tol*abs(zn);
13 while abs(zn-zo) > del
14  zo=zn;
15  zn = zo - (zo^3-1.0)/(3.0*zo^2);
16 end
17 z=zn;
```

2.7 MATLAB functions

- `x = fzero('function name',x0)`: also works with function handles, *e.g.*, `@sin`, or inline functions: `f=@(x)tanh(1.1x)-x` ⇒ `x=fzero(f,0.6)`
- `cputime`: CPU time in seconds...

  ```
  t = cputime
    ... (work)
  cputime - t
  ```

- `zeros(n,n)`: initialize matrix with zeros ⇒ improves performance
- `spy`: plot non-zero elements of matrix

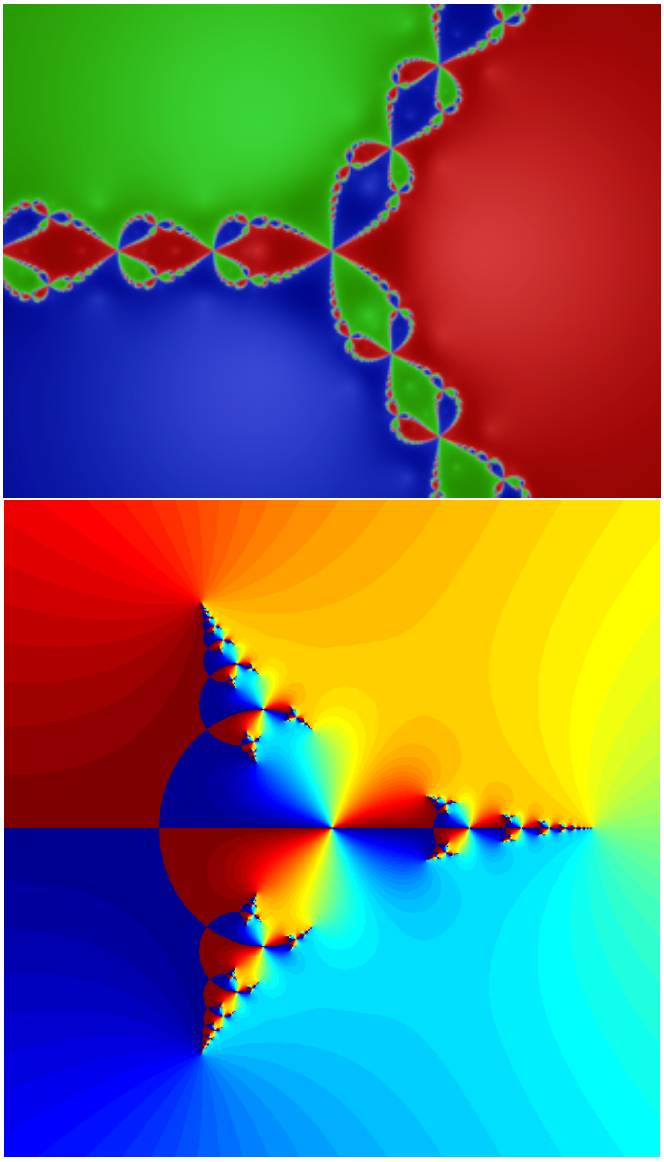
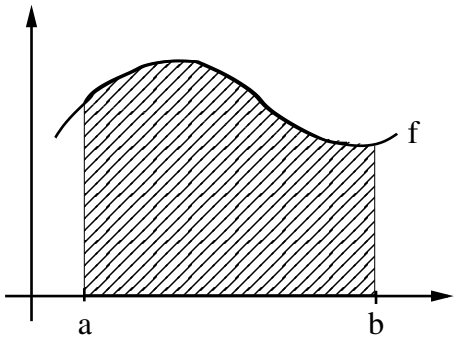


Figure 1: Julia set for the rational function associated to Newton's method and generalized Newton fractal for $f(z) = z^3 - 1$. Courtesy of `https://en.wikipedia.org/wiki/Newton_fractal`.

3 Numerical integration

one-dimensional integral: $I = \int_a^b f(x)\,dx$



higher dimensional integrals can be reduced to the one-dimensional case:

$$I = \int_{x_1}^{x_2} \left(\int_{y_1(x)}^{y_2(x)} f(x,y)\,dy \right) dx$$

3.1 Interpolation polynomials

easy: $\int_a^b x^n\,dx = \frac{x^{n+1}}{n+1}\Big|_a^b = \frac{1}{n+1}(b^{n+1} - a^{n+1})$

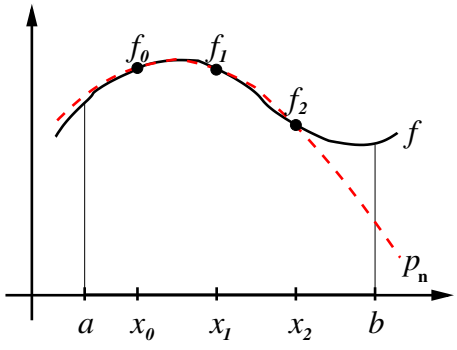
idea: - approximate f by polynomial

- integrate the polynomial exactly $\Rightarrow$ approximation of the integral

- the interval $[a,b]$ can be decomposed into subintervals; consider approximation on each subinterval

Polynomial approximation:

choose sampling points $x_0, x_1, \ldots, x_n$



There exists a polynomial of degree n,

such that

$p_n(x_i) = f(x_i)$ for $i = 0, 1, \ldots, n$

Proof. $p_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n = \sum_{j=0}^n a_j x^j$

require $f_i \equiv f(x_i) = p_n(x_i) = \sum_{j=0}^n a_j (x_i)^j$ for $i = 0, 1, \ldots, n$

$\Rightarrow n{+}1$ linear equations for the coefficients $a_0, a_1, \ldots, a_n$ (see chap. 7) $\square$

Explicit construction: **Lagrange interpolation formula**

$l_i(x) = \prod_{j \neq i} \dfrac{x - x_j}{x_i - x_j}$ polynomial of degree n

property: $l_i(x_k) = \prod_{j \neq i} \dfrac{x_k - x_j}{x_i - x_j} = \begin{cases} 0 & \text{if } k \neq i \\ 1 & \text{if } k = i \end{cases} = \delta_{ik}$

$p_n(x) = \sum_{i=0}^{n} f_i l_i(x)$ since $p_n(x_k) = f_k$.

Approximation of the integral:

$$I = \int_a^b f(x)\,dx \approx \int_a^b p_n(x)\,dx = \sum_{i=0}^{n} f_i w_i \text{ with } w_i = \int_a^b l_i(x)\,dx$$

If $f(x)$ is a polynomial of degree $\leq n$, then $p_n(x) = f(x)$ and it is integrated exactly

Proof. $p_n(x) - f(x)$ is a polynomial of degree $\leq n$

Fundamental theorem of algebra:

A polynomial has as many complex roots as its degree.

By construction: $p_n(x_i) - f(x_i) = 0$ for $i = 0, 1, \ldots, n$.

$\Rightarrow p_n - f$ has $n + 1$ different roots (sampling points)

$\Rightarrow p_n - f = 0$ by the theorem. □

Error estimate of the integral approximation:

Taylor expansion of $f(x)$ in $[a, b]$ in powers of $(x - a)$:

$$f(x) = \underbrace{\sum_{k=0}^{n} \frac{(x-a)^k}{k!} f^{(k)}(a)}_{\text{polynomial of degree } n\text{, integrated exactly}} + R_n(x)$$

$$R_n(x) = \frac{(x-a)^{n+1}}{(n+1)!} f^{(n+1)}(\xi), \qquad \xi \in (a,x)$$

Assuming $|f^{(n+1)}(\xi)| \leq M \,\forall\, \xi \in (a,b)$, we obtain the following error estimate:

$$\left| \int_a^b R_n \, dx \right| \leq M \int_a^b \frac{(x-a)^{n+1}}{(n+1)!} \, dx = M \frac{(b-a)^{n+2}}{(n+2)!}$$

3.2 Trapezoidal and Simpson's rules

equally spaced sampling points:

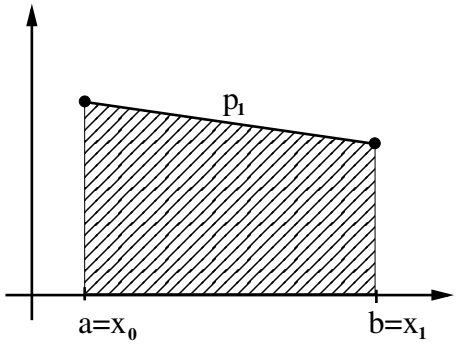
$$x_i = x_0 + ih \qquad i = 0, 1, \ldots, n \qquad x_0 = a \qquad x_n = b \quad \Rightarrow \quad h = \frac{b-a}{n}$$

$n = 1$: linear, $h = b - a$

$$l_0(x) = \frac{x - x_1}{x_0 - x_1} = \frac{x-b}{a-b}$$

$$l_1(x) = \frac{x - x_0}{x_1 - x_0} = \frac{x-a}{b-a}$$

$$\int_a^b l_0(x) \, dx = \frac{b-a}{2} = \frac{h}{2}$$



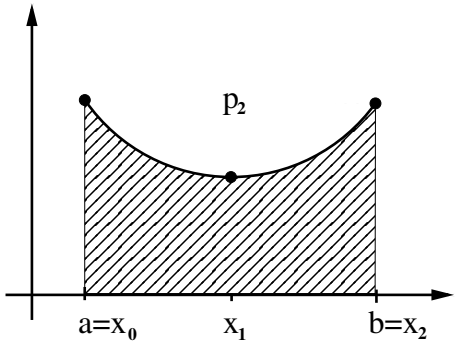
$$\int_a^b l_1(x) \, dx = \frac{h}{2} \Rightarrow \int_a^b f(x) \, dx = \int_{x_0}^{x_1} f(x) \, dx = \underbrace{\frac{h}{2}\Big(f(x_0) + f(x_1) \Big)}_{= \text{surface of trapezoid}} + O(h^3)$$

$\rightarrow$ trapezoidal rule

$n = 2$: quadratic, $h = \dfrac{b-a}{2}$

$$l_0(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}$$

$$l_1(x) = \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}$$



$$l_2(x) = \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}, \qquad \int_a^b l_i(x)\,dx = \begin{cases} \frac{h}{3} & \text{for } i = 0, 2 \\ \frac{4h}{3} & \text{for } i = 1 \end{cases}$$

$$\Rightarrow \int_{x_0}^{x_2} f(x)\,dx = h\left[\frac{1}{3}f(x_0) + \frac{4}{3}f(x_1) + \frac{1}{3}f(x_2)\right] + \underbrace{O(h^5)}$$

one order higher accuracy due to symmetry $\int_{x_0}^{x_2}(x-x_1)^3 = 0$

$\rightarrow$ Simpson's rule

$n > 2$: Newton-Cotes formulae

3.3 Extended trapezoidal/Simpson's rules

Divide interval $[a, b]$ in N contiguous and equal subintervals

$$\Rightarrow h = \frac{b-a}{N} \quad (n = 1)$$

Apply trapezoidal rule to each subinterval:

$$\begin{aligned} \int_a^b f(x)\,dx &= \int_{x_0}^{x_1} f(x)\,dx + \int_{x_1}^{x_2} f(x)\,dx + \ldots + \int_{x_{N-1}}^{x_N} f(x)\,dx \\ &\approx T_N = h\left[\frac{1}{2}f_0 + f_1 + f_2 + \ldots + f_{N-1} + \frac{1}{2}f_N\right] \end{aligned}$$

$\rightarrow$ extended trapezoidal rule

error: $\sim$ (N trapezoids)$\times \left(\frac{b-a}{N}\right)^3 = O(N^{-2})$

$T_1(f_0, f_1) \rightarrow T_2(f_0, f_1, f_2) \rightarrow T_4(f_0, f_1, f_2, f_3, f_4)$

$\rightarrow$ double $N \rightarrow 2N$

$\rightarrow$ always only N new values f_i to be computed

$\rightarrow$ successive improved approximations $T_1 \rightarrow T_2 \rightarrow T_4 \rightarrow \ldots$

Since $T_N - \underbrace{\int_a^b f(x)\,dx}_{=:T_*} = cN^{-2} + O(N^{-3})$

$\Rightarrow 4(T_{2N} - T_*) - (T_N - T_*) = 4c\frac{1}{4N^2} - c\frac{1}{N^2} + O(N^{-3})$

leading error term $\sim N^{-2}$ cancels out

$\Rightarrow T_* = \frac{4}{3}T_{2N} - \frac{1}{3}T_N + O(N^{-3})$

Define: $S_{2N} = \frac{4}{3}T_{2N} - \frac{1}{3}T_N$

$$S_{2N} = h\Big[\underbrace{\frac{1}{3}f_0 + \frac{4}{3}f_1 + \frac{2}{3}}_{\text{Simpson}\int_{x_0}^{x_2}}\underbrace{f_2 + \frac{4}{3}f_3 + \ldots}_{\text{Simpson}\int_{x_2}^{x_4}} + \frac{1}{3}f_{2N}\Big], \qquad h = \frac{b-a}{2N}$$

S_{2N} is the extended Simpson's rule applied to N subintervals!

error: $S_{2N} - T_* = O(Nh^5) = O(N^{-4})$

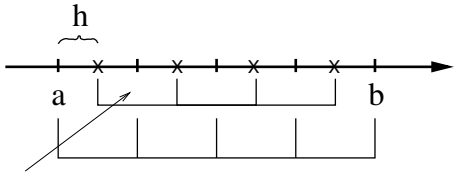
simple and effective integration method!

`http://csis.uni-wuppertal.de/courses/integration/trapez.m`

```
1  %
2  % trapez.m
3  %
4  % numerical integration of func(x) between the limits a and b
5  % computation of approximations using the trapezoidal rule
6  %      for n=0: initialization of T_1
7  %           n>0: T_N -> T_2N , where 2N = 2^n
8  %
9  function [T2N] = trapez(func,a,b,TN,n);
10
11 if n < 0,
12       error('negative n in trapez');
13 elseif n == 0,                    % only one interval
14       T2N = .5*(b-a)*(feval(func,a) + feval(func,b));
```

```
15 else
16         h = (b-a)/2^n;                % new step size
17         T2N = 0;
18         for x=a+h:2*h:b,              % sum over new (internal) points
19                 T2N = T2N + feval(func,x);
20         end
21         T2N = h*T2N + .5*TN;          % new approximation
22 end
```

T_{2N}:



new $h\sum f$ $\qquad$ $0.5T_N$ because h is a factor two smaller

up to now: endpoints a, b are sampling points (closed formulae)

for cases like $\int_0^b \frac{\sin x}{x} dx \to \frac{0}{0}$ (or worse, but the singularity at the endpoints is integrable) there exist open formulae: only sampling points in (a, b)

3.4 Gaussian quadrature

optimize the position of (not equally spaced) sampling points

$$x \in [a,b] \to y = \frac{x - \frac{1}{2}(a+b)}{\frac{1}{2}(b-a)} \in [-1,1]$$

$$f(x) = f(\tfrac{1}{2}(a+b) + \tfrac{1}{2}(b-a)y) = g(y)$$

$g(y)$ is defined in the interval $[-1, 1]$

$$\int_a^b f(x)\,dx = \frac{b-a}{2} \int_{-1}^{1} g(y)\,dy$$

Legendre Polynomials $P_n(x)$ (degree n)

orthogonalization of $x^0, x^1, x^2, x^3, \ldots$ over $[-1, 1]$,

with respect to the scalar product

$$\int_{-1}^{1} w(x)\phi_1(x)\phi_2(x)dx \qquad \text{with} \qquad w(x) \equiv 1$$

normalization: $P_n(1) = 1$

$$P_0(x) = 1, \quad P_1(x) = x, \quad P_2(x) = ax^2 + bx + c \perp P_0, P_1$$

$$\Rightarrow P_2(x) = \frac{3x^2 - 1}{2} \text{ etc. } \Rightarrow P_n(x) \perp x^0, x^1, x^2, x^3, \ldots x^{n-1}$$

P_n fulfill orthogonality relation

$$\int_{-1}^{1} P_m(x)P_n(x)\,dx = \frac{2}{2n+1}\delta_{mn}$$

and recursion formula

$$(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x)$$

$P_n(x)$ is odd/even for n odd/even.

Choose n, sampling points: $P_n(x_i) = 0, \quad i = 1, 2, \ldots, n$

All the roots x_i are real, different, $\in (-1, 1)$, with these $\{x_i\}$

$$\rightarrow l_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j} \text{ degree } n-1, \qquad w_i = \int_{-1}^{1} l_i(x)\,dx$$

$$\int_{-1}^{1} f(x)\,dx = \sum_{i=1}^{n} w_i f(x_i) \text{ is exact for}$$

$f =$ polynomial of degree $\leq 2n - 1$

Proof. $f(x) = \underbrace{q(x)}_{\text{degree } n-1} P_n(x) + \underbrace{r(x)}_{\rightarrow \quad 2n \text{ free parameters}}$

$$\int_{-1}^{1} q(x)P_n(x)\,dx = 0 \text{ because } P_n \perp x^0, x^1, x^2, \ldots, x^{n-1}$$

$$\Rightarrow \int_{-1}^{1} f(x)\,dx = \int_{-1}^{1} r(x)\,dx$$

using the result of subsection 3.1: $r(x) = \sum_{i=1}^{n} l_i(x) r(x_i)$

$\Rightarrow \int_{-1}^{1} f(x)dx = \sum_{i=1}^{n} w_i r(x_i) = \sum_{i=1}^{n} w_i f(x_i)$ because $P_n(x_i) = 0$ $\square$

Interval of length h, remainder of Taylor expansion $x^{2n} \rightarrow$ error $\sim h^{2n+1}$

N subintervals with $h = \dfrac{b-a}{N} \rightarrow$ error $\sim Nh^{2n+1} \sim N^{-2n}$ (*e.g.*, $n = 10$)

Very efficient for smooth functions, otherwise trapezoidal/Simpson's rule OK.

Values of x_i: tables, approximations + Newton-Raphson $\rightarrow$ numerical recipes

3.5 Adaptive step size $\rightarrow$ ODE

more sampling points where f varies strongly

MATLAB: `quad, quadl` (see `help quad`)

well developed methods for solving differential equations:

$$y(x) = \int_{a}^{x} f(\xi)\,d\xi, \quad I = y(b)$$

$\rightarrow$ solve $y'(x) = f(x)$ with initial condition $y(a) = 0$

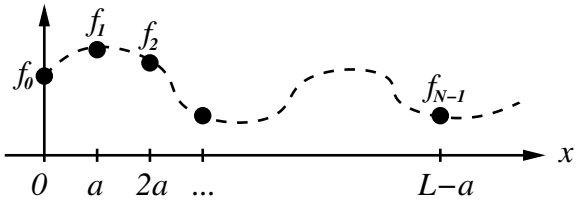
4 Fourier transformation

plane waves play generally an important role in physics → most physical laws described by ordinary differential equations (ODEs) → involves dx/dt

$$\begin{array}{lcll} x(t) = \sum_{\omega_n} \underbrace{\alpha_n}_{\text{Fourier amplitude}} e^{i\omega_n t} & \searrow & \{\alpha_n\} & \nearrow \quad \{\frac{-i}{\omega_n}\alpha'_n\} \\ & & & \omega_n \neq 0 \\ \frac{dx}{dt} = \sum_{\omega_n} i\omega_n \alpha_n e^{i\omega_n t} & \searrow & \{i\omega_n \alpha_n\} & \nearrow \quad \{\alpha'_n\} \end{array}$$

Questions to answer: which ω_n, etc...

4.1 Discrete lattice

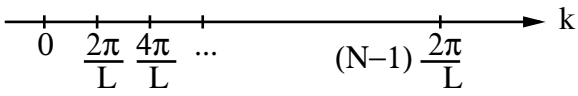


$N = \dfrac{L}{a}$ points $\Leftrightarrow \sum_x \equiv \sum_{\{x_n\}}$, $\{x_n = na, n = 0, 1, \ldots, N-1\}$

consider $f(x) \in \mathbb{C} \rightarrow$ defines a vector on lattice of points $\{f_n \equiv f(x_n)\} \in \mathbb{C}^N$

master formula: $\dfrac{a}{L}\sum_k e^{ik(x-y)} = \delta_{xy} = \begin{cases} 1, \text{if } x = y; \\ 0, \text{otherwise.} \end{cases}$

$\sum_k \equiv \sum_{\{k_n\}}$, $\{k_n = n\dfrac{2\pi}{L}, n = 0, 1, \ldots, N-1\}$



k is called wave number $= \dfrac{2\pi}{\lambda}$, with the wave length λ

Proof. $x = la, y = ma, k = n\dfrac{2\pi}{L}, 0 \leq l, m, n < N \Rightarrow k(x-y) = n(l-m)\dfrac{2\pi}{N}$

$$\frac{1}{N}\sum_{n=0}^{N-1}\left[e^{i2\pi(l-m)/N}\right]^n = \begin{cases} 1, & \text{if } l=m; \\ \frac{1}{N}\underbrace{\frac{1-e^{i2\pi(l-m)}}{1-e^{i2\pi(l-m)/N}}}_{\neq 0} = 0, & \text{if } l\neq m. \end{cases}$$

□

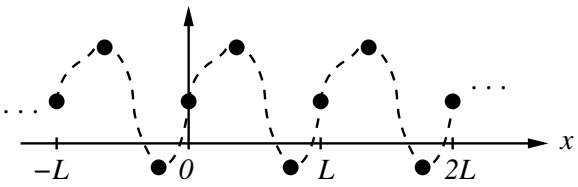
$$\begin{aligned} \Rightarrow f(x) &= \sum_y \delta_{xy} f(y) = \frac{a}{L}\sum_y\sum_k e^{ik(x-y)} f(y) \\ f(x) &\overset{(1)}{=} \frac{1}{L}\sum_k e^{ikx}\tilde{f}(k) \qquad \text{inverse Fourier transform} \\ \tilde{f}(k) &\overset{(2)}{=} a\sum_y e^{-iky} f(y) \qquad \text{Fourier transform} \end{aligned}$$

- $f(x)$ is represented as a superposition of e^{ikx}, for $k = k_n : e^{ik(x\pm L)} = e^{ikx}$

  $\Rightarrow f$ is defined through (1)$\forall x = la, l \in \mathbb{Z}$ with $f(x \pm L) = f(x)$ (periodic boundary conditions)

  $\Rightarrow$ only N independent values

  $\Rightarrow$ think of $f(x)$ as a periodic function from the beginning

  $\Rightarrow$ in (2) : $e^{-iky} f(y)$ is periodic in y

  $\sum_y$: choose arbitrary set of N contiguous points on the x-axis, *e.g.*, for odd $N = 2M + 1 : y = -Ma, -(M-1)a, \ldots, -a, 0, a, \ldots, Ma$

- $e^{-i(k\pm\frac{2\pi}{a})y} = e^{-iky}$ if $y = la, l \in \mathbb{Z}$
  $\overset{(2)}{\Rightarrow} \tilde{f}(k \pm \frac{2\pi}{a}) = \tilde{f}(k), \quad \frac{2\pi}{a} = N\frac{2\pi}{L}, \quad$ periodicity of wave number k is determined by discretization length a
  $\Rightarrow \sum_k$ in (1): freedom to choose interval of N contiguous k-values
- $\delta_{xy} = \delta_{x(y\pm L)}$ periodic continuation of master formula
- $\frac{a}{L}\sum_x e^{i(k-k')x} = \delta_{kk'}$ dual to master formula

Summary of discrete Fourier transformation (DFT):

space coordinate: $x = na,\ n = 0, 1, \ldots, N-1;\ N = L/a$: interval of size L

wave number: $k = m\dfrac{2\pi}{L},\ m = 0, 1, \ldots, N-1$; interval of size $\dfrac{2\pi}{a}$

inverse Fourier transform: $f(x) = \dfrac{1}{L}\sum_k e^{ikx}\tilde{f}(k)$, period $L : f(x \pm L) = f(x)$

Fourier transform: $\tilde{f}(k) = a\sum_x e^{-ikx} f(x)$, period $\dfrac{2\pi}{a} : \tilde{f}(k \pm \frac{2\pi}{a}) = \tilde{f}(k)$

proof: master formula $\delta_{xy} = \dfrac{1}{N}\sum_k e^{ik(x-y)}$, its dual is: $\delta_{kk'} = \dfrac{1}{N}\sum_x e^{i(k-k')x}$

4.2 MATLAB

We introduce the vectors F and $\tilde{F}$:

$$\begin{aligned} F(n) &= f(an) = \frac{1}{L}\sum_{m=0}^{N-1} z^{nm}\tilde{f}(m\frac{2\pi}{L}) = \sum_{m=0}^{N-1} z^{nm}\tilde{F}(m) \\ z &= e^{\frac{2\pi i}{N}} \\ \tilde{F}(m) &= \frac{1}{L}\tilde{f}(m\frac{2\pi}{L}) = \frac{1}{N}\sum_{n=0}^{N-1} z^{-nm}F(n) \end{aligned}$$

MATLAB implementation of $F(n)$ and $\tilde{F}(m)$: indices $n = 0, 1, \ldots, N-1$ have to be mapped to MATLAB vectors with indices $1, 2, \ldots, N$.

has built-in routines `fft` ($\rightarrow$ fast fourier transform) and `ifft`:

$F = N\times$`ifft`$(\tilde{F}), \quad \tilde{F} = (1/N)\times$`fft`$(F)$ (different conventions)

the routines work for any N, for $N = 2^\nu, \nu \in \mathbb{N}$, they are particularly fast

4.3 Example Helmholtz

Helmholtz equation: $(-\Delta + \mu^2) f(x) = \rho(x)$

(1dim, discrete)
$$-\Delta f(x) = [2f(x) - f(x+a) - f(x-a)]/a^2$$
(Poisson: $\rho \to 0$)

periodic boundary conditions (pbc), period $= L$

in Fourier space $\left\{ \begin{array}{c} f(x) \\ \rho(x) \end{array} \right\} = \frac{1}{L} \sum_k e^{ikx} \left\{ \begin{array}{c} \tilde{f}(k) \\ \tilde{\rho}(k) \end{array} \right\}$:

$$(\hat{k}^2 + \mu^2)\tilde{f}(k) = \tilde{\rho}(k), \quad \hat{k} = \frac{2}{a} \sin \frac{ka}{2}, \qquad (3)$$

is diagonal in k, easy to invert: $\tilde{f}(k) = \dfrac{\tilde{\rho}(k)}{\hat{k}^2 + \mu^2}$ and

$$f(x) = a \sum_y K(x,y)\rho(y) \quad \text{with} \quad K(x,y) = \frac{1}{L} \sum_k \frac{e^{ik(x-y)}}{\hat{k}^2 + \mu^2}$$

$$\begin{aligned} a \sum_y e^{-iky} \quad &: \quad \rho(y) \to \tilde{\rho}(k) \\ \text{then} \frac{1}{\hat{k}^2 + \mu^2} \quad &: \quad \text{operator}^{-1} \\ \text{then} \frac{1}{L} \sum_k e^{ikx} \quad &: \quad \tilde{f}(k) \to f(x) \end{aligned}$$

$\hat{k} = \dfrac{2}{a} \sin \dfrac{ka}{2} \approx k$ if $ka << 1$, *i.e.*, modes for which the discretization is a good approximation, $|\hat{k}| \leq \dfrac{2}{a}$ cut-off in the wave number (UV regularization)

if ρ (and f) contains modes $k \sim \dfrac{1}{a}$, result depends on discretization ($k \neq \hat{k}$)

Proof of (3):

$$\begin{aligned}
-\Delta f(x) &= -\Delta \frac{1}{L}\sum_k e^{ikx}\tilde{f}(k) = \frac{1}{L}\sum_k \tilde{f}(k)(-\Delta e^{ikx}) \\
&= \frac{1}{L}\sum_k \tilde{f}(k)\frac{1}{a^2}\left[2e^{ikx} - e^{ik(x+a)} - e^{ik(x-a)}\right] \\
&= \frac{1}{L}\sum_k \tilde{f}(k)e^{ikx}\frac{1}{a^2}\left[2 - e^{ika} - e^{-ika}\right] \\
&= \frac{1}{L}\sum_k \tilde{f}(k)e^{ikx}\frac{1}{a^2}\left[2 - 2\cos(ka)\right] = \frac{1}{L}\sum_k \tilde{f}(k)e^{ikx}\frac{4}{a^2}\sin^2(\frac{ka}{2})
\end{aligned}$$

4.4 Continuum limit

$a \to 0$ with L fixed

$$\begin{cases} N = \dfrac{L}{a} \to \infty : \quad f(x) = \dfrac{1}{L}\sum_k e^{ikx}\tilde{f}(k) \overset{\text{Fourier theorem}}{=} \dfrac{1}{L}\sum_{m=-\infty}^{\infty} e^{im\frac{2\pi}{L}x}\tilde{f}(k) \\ x \in (-\dfrac{L}{2}, \dfrac{L}{2}) \text{ continuous}, \quad k = 0, \pm\dfrac{2\pi}{L}, \pm\dfrac{4\pi}{L} \ldots \infty\text{-many terms} \quad (*) \end{cases}$$

$$\begin{aligned}
\tilde{f}(k) &= \int_0^L e^{-iky}f(y)dy \quad (\text{limit of Riemann sum: } a\sum_y \ldots \to \int dy \ldots) \\
&= \int_{-\frac{L}{2}}^{\frac{L}{2}} e^{-iky}f(y)\,dy \quad (e^{-iky} \text{ and } f(y) \text{ are L-periodic}) \\
\frac{1}{a}\delta_{xy} &= \frac{1}{L}\sum_k e^{ik(x-y)} \to \delta(x-y)
\end{aligned}$$

L-periodic functions (also $\delta(.)$) and infinite mode ($\Leftrightarrow k$) sums

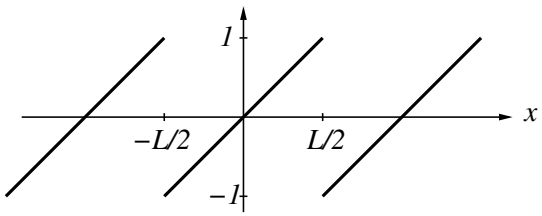
$(*)$ if we choose $k = 0, \dfrac{2\pi}{L}, \dfrac{4\pi}{L}, \ldots, (\dfrac{L}{a} - 1)\dfrac{2\pi}{L}$, the slowly changing periodic function $f(x) = e^{-i\frac{2\pi}{L}x}$ contains only the Fourier component of largest $k = (\dfrac{L}{a} - 1)\dfrac{2\pi}{L}$. In order to avoid a pathological limit $a \to 0$ a slowly changing function should contain only small $|k|$ values

$\to$ choose $-\dfrac{\pi}{a} < k \leq \dfrac{\pi}{a}$ before taking $a \to 0$.

Absolute symmetry with $k = 0$ in the middle holds only for $N =$ odd.

Example:

$$f(x) = \frac{x}{L/2}, \quad -\frac{L}{2} \le x \le \frac{L}{2}$$



$$\tilde{f}(k) = \int_{-L/2}^{L/2} e^{-iky} f(y)\, dy = \frac{2}{L} \int_{-L/2}^{L/2} e^{-iky} \underset{\uparrow}{y}\, \underset{\downarrow}{dy}, \quad k = \frac{2\pi}{L} m,\ m \in \mathbb{Z}$$

$$k \neq 0: \quad = \frac{2}{L}\left[\frac{e^{-iky}}{-ik} y|_{-L/2}^{L/2} - \int_{-L/2}^{L/2} \frac{e^{-iky}}{-ik} dy\right] = \frac{2}{L}\frac{i}{k}\left[e^{-iky} y - \frac{e^{-iky}}{-ik}\right]_{-L/2}^{L/2}$$

$$= \frac{2}{L}\frac{i}{k}\left[e^{-ik\frac{L}{2}}\frac{L}{2} - \frac{i}{k}e^{-ik\frac{L}{2}} + e^{ik\frac{L}{2}}\frac{L}{2} + \frac{i}{k}e^{ik\frac{L}{2}}\right]$$

$$= \frac{2}{L}\frac{i}{k}\left[2\cos(k\frac{L}{2})\frac{L}{2} + \frac{i}{k}2i\sin(k\frac{L}{2})\right]$$

$$= \frac{2}{L}\frac{i}{k}\left[\cos(k\frac{L}{2})L - \frac{2}{k}\sin(k\frac{L}{2})\right]$$

$$= \frac{2i}{2\pi m}\left[\underbrace{\cos(\pi m)}_{(-1)^m} L - \frac{2}{k}\underbrace{\sin(\pi m)}_{=0}\right] = \frac{iL}{\pi m}(-1)^m \qquad (k = \frac{2\pi}{L}m)$$

$$k = 0: \quad = 0$$

$$\Rightarrow f(x) = \frac{1}{L}\sum_{\substack{m=-\infty \\ m\neq 0}}^{\infty} e^{im\frac{2\pi}{L}x}\frac{iL}{\pi m}(-1)^m = \frac{i}{\pi}\sum_{m=1}^{\infty}\overbrace{\left[e^{im\frac{2\pi}{L}x} - e^{-im\frac{2\pi}{L}x}\right]}^{2i\sin(m\frac{2\pi}{L}x)}\frac{(-1)^m}{m}$$

$$f(x) = \frac{2}{\pi}\sum_{m=1}^{\infty}\frac{(-1)^{m+1}}{m}\sin(m\frac{2\pi}{L}x) = \frac{2}{\pi}\left[\sin(\frac{2\pi}{L}x) - \frac{1}{2}\sin(\frac{4\pi}{L}x) + \frac{1}{3}\sin(\frac{6\pi}{L}x) - \ldots\right]$$

Fourier series at $x = \pm\frac{L}{2}$ is $= 0$

Theorem of Dirichlet: $f(x)$ in $\left(-\frac{L}{2}, \frac{L}{2}\right)$

(a) $\left(-\frac{L}{2}, \frac{L}{2}\right)$ can be divided in a finite number of subintervals where $f(x)$ is continous and monotonic

(b) if $f(x)$ is discontinuous at $x = x_0$, then $\lim_{x\to x_0^+}$ and $\lim_{x\to x_0^-}$ exist

$$\Rightarrow \text{Fourier series} = \begin{cases} f(x), & \text{if } f \text{ is continuous in } x \\ \dfrac{\lim_{x\to x_0^+} + \lim_{x\to x_0^-}}{2}, & \text{otherwise.} \end{cases}$$

4.5 Thermodynamic limit ≡ infinite volume limit

$L \to \infty$ with a fixed

$$\begin{cases} N \to \infty: \quad f(x) = \int_{-\pi/a}^{\pi/a} \dfrac{dk}{2\pi} e^{ikx} \tilde{f}(k) & (e^{ikx} \text{ and } \tilde{f}(k) \text{ are } \dfrac{2\pi}{a}\text{-periodic,} \\ \quad \text{Brillouin zone} & \to \text{can shift integration range)} \\ k \in \overbrace{(-\dfrac{\pi}{a}, \dfrac{\pi}{a})} \text{ continuous} & \dfrac{1}{L}\sum_{k=\frac{2\pi}{L}n} \ldots = \dfrac{1}{2\pi}\sum_k \dfrac{2\pi}{L} \ldots \to \dfrac{1}{2\pi}\int dk \ldots \end{cases}$$

$$\begin{aligned} \tilde{f}(k) &= a \sum_{m=-\infty}^{\infty} e^{-ikma} f(ma) \qquad y = ma, m = -\infty \ldots \infty \\ \frac{1}{a}\delta_{xy} &= \int_{-\pi/a}^{\pi/a} \frac{dk}{2\pi} e^{ik(x-y)} \end{aligned}$$

4.6 Fourier integral

double limit $a \to 0$ and $L \to \infty$

choose x and k intervals symmetric about 0 before limits

$$\begin{aligned} f(x) &= \int_{-\infty}^{\infty} \frac{1}{2\pi} e^{ikx} \tilde{f}(k) dk \\ \tilde{f}(k) &= \int_{-\infty}^{\infty} e^{-ikx} f(x) dx \\ \delta(x) &= \int_{-\infty}^{\infty} \frac{1}{2\pi} e^{ikx} dk \qquad \text{master formula} \\ &\downarrow \end{aligned}$$

in the sense of distribution:

$$\int_{-\infty}^{\infty} f(x)\delta(x)dx = f(0) \text{ if } f \text{ converges fast enough}$$

elegant trick:

$$\begin{array}{lcl} \displaystyle\int_{-\infty}^{\infty} \frac{1}{2\pi} e^{ikx-k^2\epsilon^2/2} dk & = & \displaystyle\frac{1}{\epsilon\sqrt{2\pi}} e^{-\frac{x^2}{2\epsilon^2}} \quad \text{Gauss integral} \\ & = & \delta_\epsilon(x) \text{ regularized expression of} \\ \Big\downarrow & & \Big\downarrow \quad \delta\text{-function of width } \epsilon \text{ and} \\ \leftarrow \text{ limit } \epsilon \to 0 \ \rightarrow & & \displaystyle\int_{-\infty}^{\infty} \delta_\epsilon(x)\, dx = 1 \\ \displaystyle\int_{-\infty}^{\infty} \frac{1}{2\pi} e^{ikx} dk & = & \delta(x) \end{array}$$

4.7 Sampling theorem

$(x,k) \Leftrightarrow (t,\omega)$; $a \leftrightarrow \tau$; frequency $f = \dfrac{\omega}{2\pi}$

$f(t)$ is for example a continuous audio signal: $\tilde{f}(\omega) = \int_{-\infty}^{\infty} e^{-i\omega t} f(t)\, dt$

assume: $\tilde{f}(\omega) = 0$ for $|\omega| > \omega_{\max}$
(true for any sound source: it has a finite spectrum)

$\tilde{f}$ on compact ω-interval is related by Fourier trsf. to f at discrete t's:

if we set $\omega_{\max} = \dfrac{\pi}{\tau}$: $\qquad \tilde{f}(\omega) = \tau \displaystyle\sum_{n=-\infty}^{\infty} f(n\tau) e^{-i\omega(n\tau)}$

$\downarrow$ *same* $\tilde{f}$ can be obtained only knowing $f(n\tau),\ n \in \mathbb{Z}$

Nyquist critical frequency: $f_{\text{Ny}} = \dfrac{\pi/\tau}{2\pi} = \dfrac{1}{2\tau}$

$\Rightarrow$ the full signal $f(t)$ can be reconstructed from the discrete samples (exact interpolation):

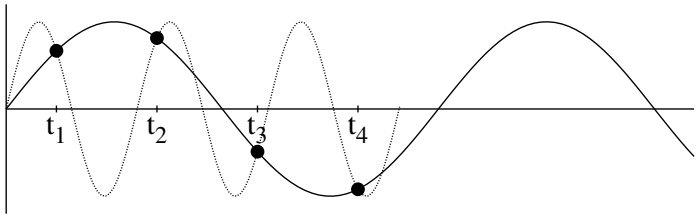
$$\begin{aligned} f(t) &\overset{(*)}{=} \int_{-\pi/\tau}^{\pi/\tau} \frac{d\omega}{2\pi} e^{i\omega t} \tau \sum_{n=-\infty}^{\infty} f(n\tau) e^{-i\omega n\tau} = \tau \sum_{n=-\infty}^{\infty} f(n\tau) \frac{1}{2\pi} \frac{e^{i\omega(t-n\tau)}}{i(t-n\tau)} \Big|_{\omega=-\pi/\tau}^{\omega=\pi/\tau} \\ &= \tau \sum_{n=-\infty}^{\infty} f(n\tau) \frac{\sin[2\pi f_{\mathrm{Ny}}(t-n\tau)]}{(t-n\tau)\pi} \end{aligned}$$

if we choose τ too large, such that $\omega_{\max} > \pi/\tau$, frequencies outside $-f_{\mathrm{Ny}} < f < f_{\mathrm{Ny}}$ are not integrated over in $(*)$. At the same time contribution from the higher frequencies are included in the samples $f(n\tau)$ $\Rightarrow$ **aliasing**: through the reconstruction the frequencies outside $(-f_{\mathrm{Ny}}, f_{\mathrm{Ny}})$ are aliased (falsely translated) into that range and the signal is distorted:

$\sin(\omega_1 t) = \sin(2\pi f_1 t)$

$\sin(\omega_2 t) = \sin(2\pi f_2 t)$

sampling rate: $1/\tau$



$\Rightarrow t = n\tau$ samples at $t_1, t_2, t_3, \ldots$, where the two signals are indistinguishable

$\rightarrow$ DFT misses the high frequency component (the high frequency component has been **aliased** by the low frequency one)

$$\sin(2\pi f_1 n\tau) = \sin(2\pi f_2 n\tau) \text{ if } |f_2 - f_1| = m/\tau = m2f_{\mathrm{Ny}}$$

any frequency component outside the range $(-f_{\mathrm{Ny}}, f_{\mathrm{Ny}})$ is spuriously moved into that range; if $f \notin (-f_{\mathrm{Ny}}, f_{\mathrm{Ny}})$, the sampling is not able to separate it from low frequency components.

4.8 Several dimensions

discrete case:

$$\begin{aligned}
\vec{x} &= (n_1, n_2, n_3)a, \qquad n_i = 0, 1, \ldots, N-1 (i = 1, 2, 3) \\
\vec{k} &= (m_1, m_2, m_3)\frac{2\pi}{L}, \quad m_i = 0, 1, \ldots, N-1 (i = 1, 2, 3) \\
f(\vec{x}) &= \frac{1}{L^3} \sum_{\vec{k}} e^{i\vec{k}\vec{x}} \tilde{f}(\vec{k}) \\
\tilde{f}(\vec{k}) &= a^3 \sum_{\vec{x}} e^{-i\vec{k}\vec{x}} f(\vec{x})
\end{aligned}$$

easy to generalize: $a_i, L_i,\ i = 1, 2, 3$, limits ($a \to 0$, etc.) are similar,

for example: $\displaystyle\int_{-\infty}^{\infty} \frac{d^3k}{(2\pi)^3} e^{i\vec{k}\vec{x}} = \delta^{(3)}(\vec{x})$

4.9 Fast Fourier Transform (FFT)

We follow the derivation of the FFT given in [1].

Numerical evaluation of the discrete Fourier transformation:

$$x = na, \quad k = m\frac{2\pi}{L}, \quad n, m = 0, 1, \ldots, N-1, \quad a > 0, \quad \frac{L}{a} = N \text{ finite}$$

$$\begin{cases} f(x) = \dfrac{1}{L} \displaystyle\sum_k e^{ikx} \tilde{f}(k) \\ \\ \tilde{f}(k) = a \displaystyle\sum_x e^{-ikx} f(k) \end{cases} \Rightarrow \begin{cases} f(an) = F(n) = \displaystyle\sum_{m=0}^{N-1} z^{nm} \tilde{F}(m),\ z = e^{2\pi i/N} \\ \\ \tilde{F}(m) = \dfrac{1}{L}\tilde{f}(m\dfrac{2\pi}{L}) = \dfrac{1}{N} \displaystyle\sum_{n=0}^{N-1} (z^*)^{mn} F(n) \end{cases}$$

The formulae on the right are interpreted as matrix multiplications (left) with $N \times N$ matrices of elements z^{nm} or $(z^*)^{nm}$.

Cost of matrix × vector: N^2 multiplications. What happens in higher, *e.g.*, three dimensions? ⇒ matrix size $N^3 \times N^3$, cost $(N^3)^2$, but we can do better:

given $\tilde{f}(k_1, k_2, k_3) \to f^{(1)}(x_1, \overbrace{k_2, k_3}^{\text{fixed}}) = \frac{1}{L} \sum_{k_1} e^{ik_1 x_1} \tilde{f}(k_1, k_2, k_3)$, *i.e.*, we transform only the first variable $k_1 \to x_1$. This has to be done for each value

of the "spectator" variables k_2, k_3 ⇒ cost is $N^2 \times N^2 = N^4$.

$$f^{(1)}(x_1, k_2, k_3) \;\; \to \;\; f^{(2)}(\overbrace{x_1, x_2}^{\text{fixed}}, k_3) = \frac{1}{L} \sum_{k_2} e^{ik_2 x_2} f^{(1)}(x_1, k_2, k_3)$$

$$f^{(2)}(x_1, x_2, k_3) \;\; \to \;\; f(\overbrace{x_1, x_2}^{\text{fixed}}, x_3) = \frac{1}{L} \sum_{k_3} e^{ik_3 x_3} f^{(2)}(x_1, x_2, k_3)$$

total cost: $3 \times N^4$ for 3 dimensions; in D dimensions: $DN^{D-1}N^2 = DN^{D+1}$ compared to N^{2D} for matrix multiplication.

FFT algorithm: by Cooley and Tukey; but even before in other codes; in 1805 Gauss had something similar to analyze trajectories of asteroids.

one-dimensional case: cost $N^2 \to N \ln N$; we restrict to the case $N = L/a = 2^\nu$, otherwise technical complications arise

z^{nm}, $z = e^{2\pi i/N}$: $z^N = 1$ ⇒ write nm in binary representation and neglect powers $2^\nu = N$ or higher $(2N, 4N, \ldots)$ since $z^N = 1$

$$n = \sum_{i=0}^{\nu-1} n_i 2^i, \;\; F(n) \equiv F(n_0, n_1, \ldots, n_{\nu-1}), \;\; n_i = 0, 1$$

$$0 \le n \le \frac{1 - 2^\nu}{1 - 2} = 2^\nu - 1 = N - 1$$

$$m = \sum_{i=0}^{\nu-1} m_i 2^{\nu-1-i}, \;\; \tilde{F}(m) \equiv \tilde{F}(m_0, m_1, \ldots, m_{\nu-1}), \;\; m_i = 0, 1$$

$0 \le m \le 2^\nu - 1$; it is technically better to write m in bit-reversed order

$\Rightarrow nm|_{\text{truncated}} = m_0 2^{\nu-1} n_0 + m_1 2^{\nu-2}(n_0 + n_1 2) + m_2 2^{\nu-3}(n_0 + n_1 2 + n_2 2^2) + \ldots$

Now we can proceed similarly to the multi-dimensional Fourier transform:

$\tilde{F}(m_0, m_1, \ldots, m_{\nu-1}) \rightarrow$

$F^{(1)}(n_0, m_1, m_2, \ldots, m_{\nu-1}) = \sum_{m_0=0,1} z^{m_0 2^{\nu-1} n_0} \tilde{F}(m_0, m_1, \ldots, m_{\nu-1}) \rightarrow$

$F^{(2)}(n_0, n_1, m_2, \ldots, m_{\nu-1}) = \sum_{m_1=0,1} z^{m_1 2^{\nu-2}(n_0+n_1 2)} F^{(1)}(n_0, m_1, \ldots, m_{\nu-1}) \rightarrow$

$\ldots$

$F^{(\nu)}(n_0, n_1, n_2, \ldots, n_{\nu-1}) = F(n_0, n_1, n_2, \ldots, n_{\nu-1}) = \sum_{m=0}^{2^\nu - 1} z^{nm} \tilde{F}(m)$

"triangular" relationship (different from higher D case) in the exponent of z:

transform $m_0 \rightarrow n_0$: $\quad m_0 \ldots n_0$, need $\tilde{F}$

transform $m_1 \rightarrow n_1$: $\quad m_1 \ldots (n_0 \ldots n_1 \ldots)$, need $F^{(1)}$

transform $m_2 \rightarrow n_2$: $\quad m_2 \ldots (n_0 \ldots n_1 \ldots n_2)$, need $F^{(2)}$

$\ldots$ recursion can be solved in the order

Cost:

- overhead at the beginning:
  sorting $\tilde{F}(m)$ into bit-reversed order
- ν times one multiplication $(F^{()} + z^{1\cdots} F^{()})$ for $2^{\nu-1} = N/2$ values of the "spectator" variables

$\Rightarrow$ overall: $N\nu = N \log_2(N)$!

- the powers of z can be computed through recursion and not many trigonometric function evaluations are needed

4.10 Real functions

$f(x) \in \mathbb{R} \Leftrightarrow \tilde{f}(k)^* = \tilde{f}(-k)$

Proof. $a > 0, k-$sum (or integral) is symmetric about 0 (!):

$$f(x) = \frac{1}{L}\sum_k e^{ikx}\tilde{f}(k); \quad f(x)^* = \frac{1}{L}\sum_k e^{-ikx}\tilde{f}(k)^* \overset{(!)}{=} \frac{1}{L}\sum_k e^{ikx}\tilde{f}(-k)^* \qquad \square$$

We obtain the following manifestly real expression:

$$\begin{aligned} f(x) &= \frac{1}{2}(f(x)+f(x)^*) = \frac{1}{2L}\sum_k (e^{ikx}\tilde{f}(k) + e^{-ikx}\tilde{f}(-k)) \; [\tilde{f}(k)^* = \tilde{f}(-k)] \\ &= \frac{1}{L}\sum_k \Re(e^{ikx}\tilde{f}(k)) = \frac{1}{L}\sum_k \Big[\cos(kx)\Re(\tilde{f}(k)) - \sin(kx)\Im(\tilde{f}(k))\Big] \end{aligned}$$

The sum is over the full range of N $k-$values. But the summand is symmetric under $k \to -k$ $\Rightarrow$ many terms (not all) appear twice

$$\Rightarrow f(x) = \sum_{0\le k\le \pi/a} \alpha(k)\cos(kx) + \sum_{0<k<\pi/a} \beta(k)\sin(kx)$$

Exercise: a) find expression for $\alpha(k), \beta(k)$ as functions of $\tilde{f}(k)$
b) verify that there are always N real parameters
(distinguish N even/odd cases)

$a > 0, N$ odd: $N = 2M+1$

$$\begin{aligned} f(x) &= \frac{1}{L}\sum_{n=-M}^{M} e^{ik_n x}\tilde{f}(k_n), \; f \text{ real } \Rightarrow \tilde{f}(-k) = \tilde{f}(k)^*, \; k_n = \frac{2\pi}{L}n, \; n = -M, ..., 0, ..., M \\ &= \frac{1}{L}\sum_{n=-M}^{M} \Big[\cos(k_n x)\Re(\tilde{f}(k_n)) - \sin(k_n x)\Im(\tilde{f}(k_n))\Big] \\ &= \frac{1}{L}\Bigg\{ \underbrace{Re\tilde{f}(0) + \sum_{n=1}^{M} 2\Big[\cos(k_n x)\Re(\tilde{f}(k_n)) - \sin(k_n x)\Im(\tilde{f}(k_n))\Big]}_{N = 2M+1 \text{ real parameters}} \Bigg\} \end{aligned}$$

$\cos(k_n x)\Re(\tilde{f}(k_n))$ and $\sin(k_n x)\Im(\tilde{f}(k_n))$ are even functions under $n \to -n$

$a > 0, N$ even: $N = 2M$

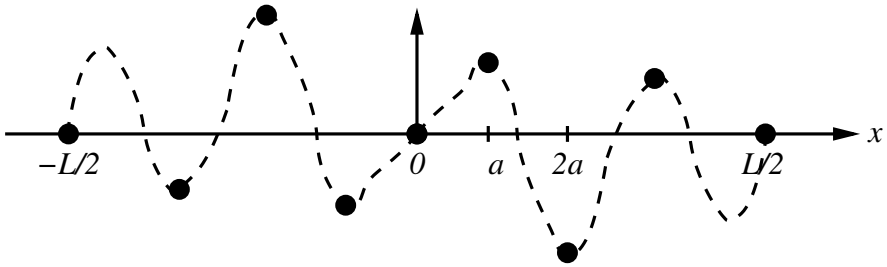
$$
\begin{aligned}
f(x) &= \frac{1}{L}\sum_{n=0}^{2M-1} e^{ik_n x}\tilde{f}(k_n), \quad f \text{ real } \Rightarrow \tilde{f}(k)^* = \tilde{f}(\frac{2\pi}{a} - k)^*, \quad k_n = \frac{2\pi}{L}n = \frac{\pi}{Ma}n \\
&\qquad n = 0, 1, \ldots 2M-1 \;\Rightarrow\; \tilde{f}(k_n)^* = \tilde{f}(k_{2M-n}) \\
&\qquad \frac{2\pi}{a} - k_n = \frac{2\pi}{a} - \frac{\pi}{Ma}n = \frac{\pi}{Ma}(2M-n) \\
&= \frac{1}{2L}\sum_{n=0}^{2M-1}\Big[e^{ik_n x}\tilde{f}(k_n) + \underbrace{e^{ik_{2M-n}x}}_{=e^{-ik_n x}}\underbrace{\tilde{f}(k_{2M-n})}_{\tilde{f}(k_n)^*}\Big] \\
&= \frac{1}{L}\sum_{n=0}^{2M-1}\Big[\cos(k_n x)\Re(\tilde{f}(k_n)) - \sin(k_n x)\Im(\tilde{f}(k_n))\Big] \\
&\qquad \cos(k_n x)\Re(\tilde{f}(k_n)) \text{ and } \sin(k_n x)\Im(\tilde{f}(k_n)) \text{ are symmetric under } n \to 2M-n \\
&= \frac{1}{L}\Bigg\{\Bigg(\sum_{n=0}^{M-1} + \sum_{\substack{n=M \\ \uparrow \\ j=2M-n}}^{2M-1}\Bigg)\Big[\cos(k_n x)\Re(\tilde{f}(k_n)) - \sin(k_n x)\Im(\tilde{f}(k_n))\Big]\Bigg\} \\
&= \frac{1}{L}\sum_{n=0}^{M-1}\Big[\cos(k_n x)\Re(\tilde{f}(k_n)) - \sin(k_n x)\Im(\tilde{f}(k_n))\Big] \\
&\quad + \frac{1}{L}\sum_{j=1}^{M}\Big[\cos(k_j x)\Re(\tilde{f}(k_j)) - \sin(k_j x)\Im(\tilde{f}(k_j))\Big] \\
&= \frac{1}{L}\Bigg\{\Re(\tilde{f}(0)) + \sum_{n=1}^{M-1}\Big[\cos(k_n x)\Re(\tilde{f}(k_n)) - \sin(k_n x)\Im(\tilde{f}(k_n))\Big] \\
&\qquad + \cos(\frac{\pi}{a}x)\Re(\tilde{f}(\frac{\pi}{a}))\Bigg\}
\end{aligned}
$$

4.11 Fourier transformation with specified boundary conditions

real function $f(x), a > 0, 0 \leq x \leq \frac{L}{2}$

case 1: Dirichlet boundary conditions:

$f(0) = 0 = f(\frac{L}{2})$: $N = \frac{L}{a}$ must be even ($\Rightarrow \frac{L}{2}$ is a lattice point)



extension: $[0, L/2] \to [-L/2, L/2]$ through $f(-x) = -f(x)$

periodicity: $\to \mathbb{R}$ through $f(x + L) = f(x)$

$\Rightarrow f(x) = \sum_{n=1}^{N/2-1} \beta_n \sin(\frac{\pi}{L/2} nx)$ (no $\alpha_n \cos()$ due to $f(-x) = -f(x)$)

$a \to 0 : \sum_{n=1}^{\infty}$

$\frac{N}{2} - 1$ values $\beta_n \leftrightarrow$ function values $f(a), f(2a), \ldots, f(\frac{L}{2} - a)$

case 2: Neumann boundary conditions: discrete:

$f(-a) = f(a)$ and $f(\frac{L}{2} - a) = f(\frac{L}{2} + a)$

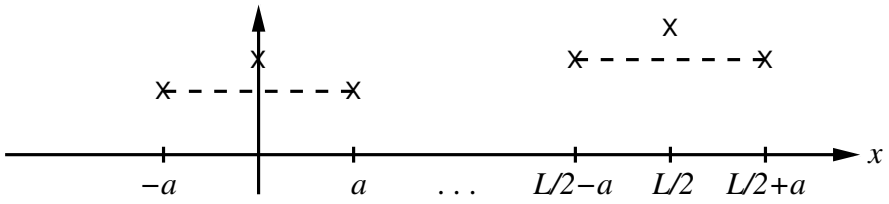
$\Rightarrow$ symmetric, derivatives at $x = 0$ and $x = L/2$ vanish

extension through $f(-x) = f(x)$ symmetry

L-periodicity through $f(x+L) = f(x)$

$$\Rightarrow f(x) = \sum_{n=0}^{N/2} \alpha_n \cos(\frac{\pi}{L/2} nx), \qquad a \to 0 : \sum_{n=0}^{\infty}$$

$\frac{N}{2} + 1$ values $\alpha_n \leftrightarrow$ function values $f(0), f(a), \ldots, f(\frac{L}{2})$



$f(a) = f(-a)$ (symmetry)

$f(\frac{L}{2} + a) = f(-\frac{L}{2} - a) = f(\frac{L}{2} - a)$ (symmetry + L-periodicity)

need $f(0), f(a), \ldots, f(\frac{L}{2})$

5 Initial value problems (ODEs)

Differential equations relate a function $u(t)$ and its derivatives $u'(t), u''(t), \ldots$ to each other at the same time. Ordinary differential equations (ODEs) of order n have the explicit form

$$F(t, u(t), u'(t), u''(t) \ldots) = u^{(n)}(t)\,,$$

where $u^{(n)} \equiv \dfrac{d^n u}{dt^n}$. ODEs are extremely important in mathematics, physics engineering, economics, finance,...

Examples:

- Classical mechanics, Newton's law of motion

$$\begin{array}{rcccc} \vec{f}(\vec{x}(t)) & = & m & \times & \dfrac{d^2\vec{x}}{dt^2} \\ \text{Force} & = & \text{mass} & \times & \text{acceleration} \end{array}$$

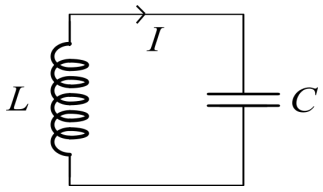
  needs initial values $\vec{x}(t_0), \dot{\vec{x}}(t_0)$

- Electronics, LC-circuit:

  $L \ldots$ inductance (inductor)

  $C \ldots$ capacitance (capacitor)

  $I \ldots$ current:



$$\frac{d^2 I(t)}{d^2 t} + \frac{1}{LC} I(t) = 0$$

  $\Rightarrow$ Harmonic Oscillator with $\omega_0 = 1/\sqrt{LC}$

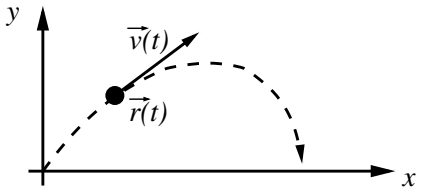
- Mathematics: Legendre Polynomials P_n are solutions to this equation

$$\frac{d}{dt}\left[(1-t^2)\frac{d}{dt}P_n(t)\right] + n(n+1)P_n(t) = 0$$

  with $P_n(1) = 1$, $P_n(-1) = (-1)^n$

5.1 A simple example from physics

trajectory of a flying ball:



$$\text{postion } \vec{r}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$$

$$\text{velocity } \vec{v}(t) = \frac{d\vec{r}(t)}{dt}$$

Newton's equations of motion: $m\dfrac{d^2\vec{r}}{dt^2} = \vec{f}(\vec{r}, \vec{v})$

m: mass of the ball (*e.g.*, 0.145 kg)

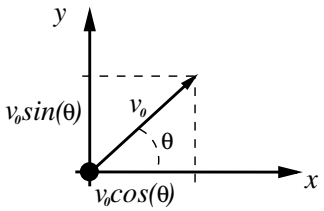
f: forces acting on the ball, *e.g.*, $\vec{f} = \vec{f_g} + \vec{f_f}$

$$\begin{aligned}
\vec{f_g} &= -m\,g \begin{pmatrix} 0 \\ 1 \end{pmatrix}, && \text{gravity} \\
& g \approx 9.81 \frac{\mathrm{m}}{\mathrm{s}^2} && \text{acceleration due to gravity} \\
\vec{f_f} &= -\frac{1}{2} C_d\, \rho\, \pi\, R^2\, |\vec{v}|\, \vec{v}, && \text{friction} \\
& C_d = 0.35, && \text{friction coefficient of a baseball} \\
& R = 3.7\,\mathrm{cm}, && \text{radius of the ball} \\
& \rho = 1.2\,\mathrm{kg/m^3}, && \text{density of air}
\end{aligned}$$

The solution is unique, once the initial values at $t = 0$ are fixed, *e.g.*,

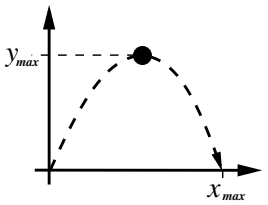
$$\vec{r}(0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\vec{v}(0) = v_0 \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix}$$



without friction ($\vec{f_f} = 0$), solution ⇒ parabola:

$$\vec{r}(t) = \begin{pmatrix} v_0 \cos(\theta) t \\ v_0 \sin(\theta) t - g t^2/2 \end{pmatrix}$$



- the ball returns to the ground ($y(t_f) = 0$) after $t_f = \dfrac{2v_0}{g}\sin(\theta)$
- maximal height at $t_f/2$ due to symmetry: $y_{max} = \dfrac{v_0^2}{2g}\sin^2(\theta)$
- range $x(t_f)$: $x_{max} = \dfrac{v_0^2}{g}\sin(2\theta) \Rightarrow$ maximal at $\theta = 45°$

5.2 Standard Form

for ordinary differential equations (ODEs): $\dfrac{d\vec{y}}{dt} = \vec{f}(t, \vec{y}(t))$

$\vec{y}$ can have many components $\vec{y} = (y_1, y_2, \ldots)^T$

initial conditions $\vec{y}(t = 0)$ completely determine the solution $\vec{y}(t)\ \forall\ t$

ODEs of higher order can be brought into standard form, *e.g.*,

$$F(t, u(t), u'(t)) = u''(t) \qquad \text{(2nd order)},$$

introduce

$$\vec{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} u(t) \\ u'(t) \end{pmatrix} \Rightarrow \frac{d\vec{y}}{dt} = \begin{pmatrix} u'(t) \\ u''(t) \end{pmatrix} = \underbrace{\begin{pmatrix} y_2(t) \\ F(t, y_1(t), y_2(t)) \end{pmatrix}}_{\equiv \vec{f}(t, \vec{y}(t))}$$

ODE of order 2 $\Leftrightarrow$ system of 2 equations of order 1.

Example: ball in two dimensions (x, y) without friction:

$\vec{r} = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}, \quad \vec{v} = \dfrac{d\vec{r}}{dt} = \begin{pmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \end{pmatrix} = \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix}$

Newton's equation (ODE): $\quad m\dfrac{d^2\vec{r}}{dt^2} = -mg\begin{pmatrix} 0 \\ 1 \end{pmatrix}$

$$\vec{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} \vec{r} \\ \vec{v} \end{pmatrix} \Rightarrow \frac{d\vec{y}}{dt} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \ddot{x} \\ \ddot{y} \end{pmatrix} = \begin{pmatrix} y_3 \\ y_4 \\ 0 \\ -g \end{pmatrix} = \vec{f}(t, \vec{y})$$

5.3 Euler method

simplest method for the solution of $\dot{\vec{y}} = \vec{f}(t, \vec{y})$

$\Rightarrow$ use "forward derivative": $\frac{d}{dt}\vec{y}(t) = \frac{1}{\tau}\left[\vec{y}(t+\tau) - \vec{y}(t)\right] + O(\tau)$

$\Rightarrow \vec{y}(t+\tau) = \vec{y}(t) + \tau\vec{f}(t, \vec{y}) + O(\tau^2)$ with step size τ

recursion: given $\vec{y}(t=0)$ we can compute $\vec{y}(\tau) \overset{O(\tau^2)}{\to} \vec{y}(2\tau) \overset{O(\tau^2)}{\to} \vec{y}(3\tau) \overset{O(\tau^2)}{\to} \ldots$

for a trajectory of length T, $N = T/\tau$ steps are necessary, the local errors of $O(\tau^2)$ in the worst case add up to the total error $NO(\tau^2) = \frac{T}{\tau}O(\tau^2) = O(\tau)$.

Example program:
`http://csis.uni-wuppertal.de/courses/initial_value/ball.m`

```
1  % ball.m - program for flying balls
2  % Euler Method, without friction
3  %
4  clear;  help ball;
5  %
6  r = [0 0]; % optional also as input argument
7  v0    = input('Initial velocity v0 (m/sec) ? ');
8  theta = input('Angle theta (degree) ? ')*pi/180;
9  % converted in radians
10 tau   = input('Time step tau (sec) ? ');
11 %
12 v = v0*[cos(theta) sin(theta)]; % v_0
13 g = 9.81; % g in m/sec^2
14 a = [0 -g]; % in this case constant acceleration
15 %
16 maxstep = 100000; % maximum number of steps
17 %
18 % main loop:
19 %
20 y(1,:) = [r(1) r(2) v(1) v(2)];  % initial values
21 t(1) = 0;
```

```
22 for istep=1:maxstep
23    y(istep+1,:) = y(istep,:) + tau * [y(istep,3),\
24                   y(istep,4), 0, -g];
25    if( y(istep+1,2) < 0 )  % stop the loop, impact!
26       break;
27    end
28 end
29
30 fprintf(' Range: %g meters\n',y(istep+1,1));
31 fprintf(' Time of flight: %g seconds\n',(istep+1)*tau)
32 fprintf(' Maximal height: %g m\n',max(y(:,2)));
33
34 %
35 %  Plot:
36 %
37 % Ground line:
38 xground = [0 y(istep+1,1)]; yground = [0  0];
39 % Graph of the trajectory:
40 plot(y(:,1),y(:,2),'+',xground,yground,'-');
41 xlabel('Range (m)')
42 ylabel('Altitude (m)')
43 title('Flying ball')
44 %
45 % Compare with exact solution:
46 %
47 xmaxx = v0^2*sin(2*theta)/g;
48 tfl   = 2*v0*sin(theta)/g;
49 ymax  = v0^2/2/g*sin(theta)^2;
50 fprintf('relative discrepancy in time of flight: %g\n',\
51         ((istep+1)*tau-tfl)/tfl)
52 fprintf('relative discrepancy in range         : %g\n',\
53         (y(istep+1,1)-xmaxx)/xmaxx)
54 fprintf('relative discrepancy in maximal height: %g\n',\
55         (max(y(:,2))-ymax)/ymax)
56 %
57 % End of program
58 %
```

5.4 Runge-Kutta method (second order)

need to cancel the $O(\tau^2)$ term in

$$(I) \quad \vec{y}(t+\tau) = \vec{y}(t) + \tau \overbrace{\dot{\vec{y}}(t)}^{=f(t,\vec{y})} + \frac{\tau^2}{2}\ddot{\vec{y}}(t) + O(\tau^3)$$

$\Rightarrow$ as in finite-difference method: introduce a different shift

$$(II) \quad \vec{y}(t+2\alpha\tau) = \vec{y}(t) + 2\alpha\tau\vec{f}(t,\vec{y}) + \frac{(2\alpha\tau)^2}{2}\ddot{\vec{y}}(t) + O(\tau^3)$$

$\Rightarrow \quad (I) \; - \; \frac{1}{(2\alpha)^2}(II)$ has no $O(\tau^2)$ term:

$$\vec{y}(t+\tau) - \frac{1}{(2\alpha)^2}\vec{y}(t+2\alpha\tau) = \vec{y}(t)\left[1 - \frac{1}{(2\alpha)^2}\right] + \tau\vec{f}(t,\vec{y})\left[1 - \frac{1}{2\alpha}\right] + O(\tau^3)$$

$$\vec{y}(t+\tau) = \vec{y}(t) + \frac{\tau}{2\alpha}\underbrace{\frac{\vec{y}(t+2\alpha\tau) - \vec{y}(t)}{2\alpha\tau}} + \tau\vec{f}(t,\vec{y})\left[1 - \frac{1}{2\alpha}\right] + O(\tau^3)$$

$$\dot{\vec{y}}(t+\alpha\tau) + O(\tau^2) = \vec{f}(t+\alpha\tau, \vec{y}(t+\alpha\tau)) + O(\tau^2)$$

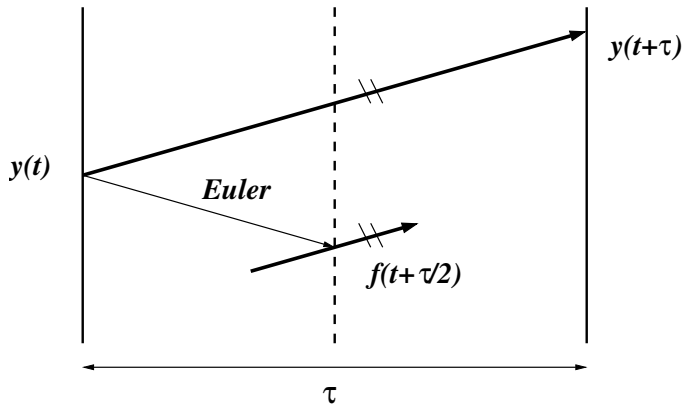
$$\vec{y}(t+\tau) = \vec{y}(t) + \tau\left[\vec{f}(t,\vec{y})\left(1 - \frac{1}{2\alpha}\right) + \frac{1}{2\alpha}\vec{f}(t+\alpha\tau, \vec{y}(t+\alpha\tau))\right] + O(\tau^3)$$

$$\vec{y}(t+\tau) = \vec{y}(t) + \tau\left[w_1\vec{f}(t,\vec{y}) + w_2\vec{f}(t+\alpha\tau, \vec{y}^*)\right] + O(\tau^3)$$

with $w_2 = \dfrac{1}{2\alpha}$, $w_1 = 1 - w_2$, and $\vec{y}^* = \vec{y}(t) + \alpha\tau\vec{f}(t, \vec{y}(t))$,

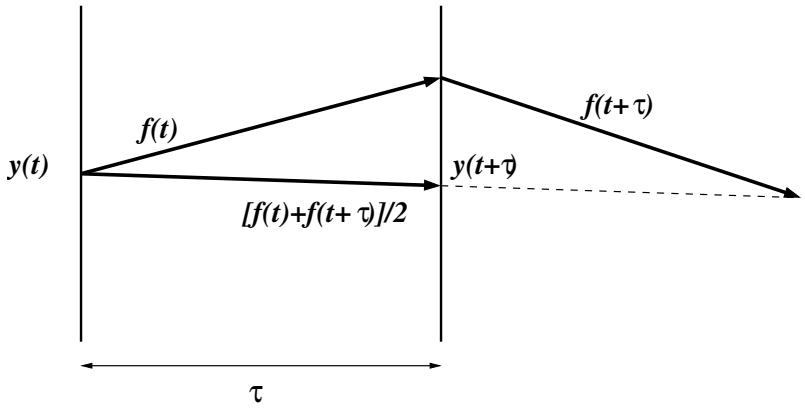
where in the second last line we used $\vec{f}(t+\alpha\tau, \vec{y}(t+\alpha\tau)) = f(t+\alpha\tau, \vec{y}^*) + O(\tau^2)$.
In order to compute one step:

1. compute $\vec{y}^* = \vec{y}(t) + \alpha\tau\vec{f}(t, \vec{y}(t))$ (ordinary Euler)
2. $\vec{y}(t+\tau) = \vec{y}(t) + \tau\left[w_1\vec{f}(t,\vec{y}) + w_2\vec{f}(t+\alpha\tau, \vec{y}^*)\right] + O(\tau^3)$
   popular choices:
   - "midpoint method": $\alpha = 1/2 \;\Rightarrow\; w_1 = 0, w_2 = 1$



- Henn's method: $\alpha = 1 \Rightarrow w_1 = w_2 = 1/2$:

$$\begin{aligned} \vec{y}(t+\tau) &= \vec{y}(t) + \frac{1}{2}\left[\vec{k}_1 + \vec{k}_2\right] + O(\tau^3) \\ \vec{k}_1 &= \tau \vec{f}(t, \vec{y}(t)) \\ \vec{k}_2 &= \tau \vec{f}(t+\tau, \vec{y}(t) + \vec{k}_1) \end{aligned}$$



fixed interval T: global error $\dfrac{T}{\tau} O(\tau^3) = O(\tau^2)$

higher order integrators tedious to derive without some computer algebra help $\Rightarrow$ some results:

5.5 Runge-Kutta method 3rd order

$$\begin{aligned}
\vec{k}_1 &= \tau \vec{f}(t, \vec{y}(t)) \\
\vec{k}_2 &= \tau \vec{f}(t+\tau, \vec{y}(t)+\vec{k}_1) \\
\vec{k}_3 &= \tau \vec{f}(t+\tau/2, \vec{y}(t)+(\vec{k}_1+\vec{k}_2)/4) \\
\vec{y}(t+\tau) &= \vec{y}(t) + \frac{1}{6}\left[\vec{k}_1+\vec{k}_2+4\vec{k}_3\right] + O(\tau^4) \ldots \text{global error: } O(\tau^3)
\end{aligned}$$

5.6 Runge-Kutta method 4th order

$$\begin{aligned}
\vec{k}_1 &= \tau \vec{f}(t, \vec{y}(t)) \\
\vec{k}_2 &= \tau \vec{f}(t+\tau/2, \vec{y}(t)+\vec{k}_1/2) \\
\vec{k}_3 &= \tau \vec{f}(t+\tau/2, \vec{y}(t)+\vec{k}_2/2) \\
\vec{k}_4 &= \tau \vec{f}(t+\tau, \vec{y}(t)+\vec{k}_3) \\
\vec{y}(t+\tau) &= \vec{y}(t) + \frac{1}{6}\left[\vec{k}_1+2\vec{k}_2+2\vec{k}_3+\vec{k}_4\right] + O(\tau^5) \ldots \text{global error: } O(\tau^4)
\end{aligned}$$

5.7 Adaptive step-size control

idea: estimate the numerical integration error and tune the step size to achieve desired accuracy

one way to do this: step doubling

$$\begin{aligned}
\vec{y}(t) &\xrightarrow{\tau} \vec{Y} && \text{Runge-Kutta } n\text{th order} \\
\overset{\frac{\tau}{2}}{\searrow} &\xrightarrow{\frac{\tau}{2}} \vec{y}(t+\tau) && : \text{ expected to be more precise,} \\
& && \quad \text{take to be the solution}
\end{aligned}$$

$\vec{y}_{ex}(t+\tau)$ is the exact solution:

$$\begin{aligned}
\vec{Y} &= \vec{y}_{ex}(t+\tau) + \vec{c}\,\tau^{n+1} && \text{the same } \vec{c}\text{: leading} \\
\vec{y}(t+\tau) &= \vec{y}_{ex}(t+\tau) + 2\,\vec{c}\left(\frac{\tau}{2}\right)^{n+1} && \text{error term only} \\
\vec{Y} - \vec{y}(t+\tau) &= \vec{c}(1-2^{-n})\tau^{n+1} = \vec{c}\,2^{-n}\tau^{n+1}(2^n-1)
\end{aligned}$$

error in $\vec{y}(t+\tau) \Rightarrow \vec{c}\,2^{-n}\tau^{n+1} = \dfrac{\vec{Y} - \vec{y}(t+\tau)}{2^n - 1}$ $\qquad (*)$

define the maximal relative error of the components of solution $\vec{y}(t+\tau)$:

$$\delta = \max_i \frac{|Y_i - y_i(t+\tau)|}{|y_i(t+\tau)|(2^n - 1)}$$

(if $|\vec{y}_i(t+\tau)| \approx 0$: use absolute error or a different scale)

$\overset{(*)}{\Rightarrow}$ δ is proportional to τ^{n+1}, if ϵ is the desired accuracy, we need:

$$\frac{\delta}{\tau^{n+1}} = \frac{\epsilon}{(\tau')^{n+1}} \text{ or } \tau' = \tau \left(\frac{\epsilon}{\delta}\right)^{1/(n+1)} \times 0.9 \text{ (security factor)}$$

algorithm to tune the step size:

`if` $\delta > \epsilon$: repeat the integration step with new step size $\tau' < \tau$

`else` : do the next step with $\tau' \gtrsim \tau$

adaptive step size: smaller step size when solution changes more rapidly

local extrapolation: correct

$$\vec{y}(t+\tau) - \frac{\vec{Y} - \vec{y}(t+\tau)}{2^n - 1} = \frac{2^n \vec{y}(t+\tau) - \vec{Y}}{2^n - 1}$$

leading error is now $O(\tau^{n+2})$, but we do not have an estimate for it

$\Rightarrow$ method nth order **with** error control

other way to estimate the error: compare Runge-Kutta (RK) methods with nth and $(n+1)$th order
Step size control based on RK4 and RK5:

$$
\begin{aligned}
\vec{Y} &= \vec{y}_{ex}(t+\tau) + \vec{c}\tau^5 \quad \text{(RK4)} \\
\vec{y}(t+\tau) &= \vec{y}_{ex}(t+\tau) + \vec{c}'\tau^6 \quad \text{(RK5)} \\
&\approx \vec{y}_{ex}(t+\tau) \\
\Rightarrow \Delta\vec{y} \equiv \vec{Y} - \vec{y}(t+\tau) &= \vec{c}\tau^5 + O(\tau^6) \\
\delta &= \max_i \frac{|\Delta y_i|}{y_i(t+\tau)} \quad \text{relative error}
\end{aligned}
$$

δ is proportional to τ^5

ϵ is the target accuracy: $\dfrac{\epsilon}{(\tau')^5} = \dfrac{\delta}{\tau^5}$

$$\Rightarrow \left(\frac{\tau'}{\tau}\right)^5 = \frac{\epsilon}{\delta} \text{ or } \tau' = \tau\left(\frac{\epsilon}{\delta}\right)^{1/5} \quad (\times 0.9\ldots \text{ security factor})$$

5.8 MATLAB

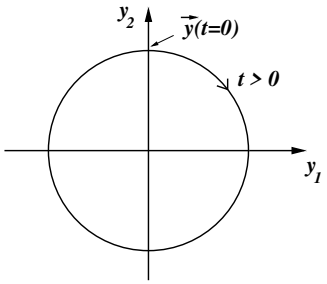
`ode23`: RK 2nd order with 3rd order to control the error and adapt step size

`ode45`: RK 4th order with 5th order to control the error and adapt step size
$\rightarrow$ exercise

Example:

$$\frac{d}{dt}\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} y_2 \\ -y_1 \end{pmatrix}, \ \vec{y}(t=0) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

exact solution is $\vec{y}_{ex}(t) = \begin{pmatrix} \sin(t) \\ \cos(t) \end{pmatrix}$



Example programs `odetest.m` and `ftest.m`

```
[t,y] = ode23( 'ftest' , [ 0 40*pi ] , [ 0 1 ] , opts );
```

`t`: column with times where solution is computed

`y`: matrix; columns correspond to $y_i(t), i = 1, 2$

`opts = odeset('RelTol',2e-4);` $\leftrightarrow$ $\epsilon = 2 \cdot 10^{-4}$ (default $\epsilon = 1 \cdot 10^{-3}$)

function `ftest.m` returns a column vector corresponding to

$$\vec{f}(t, \vec{y}(t)) = \begin{pmatrix} y_2 \\ -y_1 \end{pmatrix}$$

`http://csis.uni-wuppertal.de/courses/ode/ftest.m`

```
1  function yprime = ftest(t,y)
2  % simple test for ode23 and ode45
3  % y: 2-component column
4  yprime = [y(2); -y(1)];
```

`http://csis.uni-wuppertal.de/courses/ode/odetest.m`

```
1  %
2  % odetest.m
3  %
4  % test for ode23;
5  % uses the function ftest
6  t0 = 0;      % initial value for the time
7  y0 = [0 1]; % initial value y0
8  tfinal = 40.*pi;
9  tspan=[t0 tfinal];
10 [t,y] = ode23('ftest',tspan,y0);
11 %
12 opts = odeset('RelTol',2e-4);
13 [t,y] = ode23('ftest',tspan,y0,opts);
14 %
15 %[t,y] = ode45('ftest',tspan,y0);
16 dis = [sin(t) cos(t)]-y; % discrepancy from the exact solution
17 plot(t,dis);
18 title('Discrepancy from the exact solution')
19 pause
20 plot(y(:,1),y(:,2));
21 axis square
22 title('Components of the solution');
```

5.9 A note on Kepler's 3rd law

We consider the gravitational motion in three dimensions of two bodies, 1 and 2. The positions of the bodies are given by the vectors $\vec{r}_1$ and $\vec{r}_2$ and their masses by m_1 and m_2. We also introduce the relative distance vector $\vec{r} = \vec{r}_1 - \vec{r}_2$, see Fig. 2. The relative distance of the bodies is denoted by $r = |\vec{r}|$. According to Newton's gravitational law, the bodies attract each

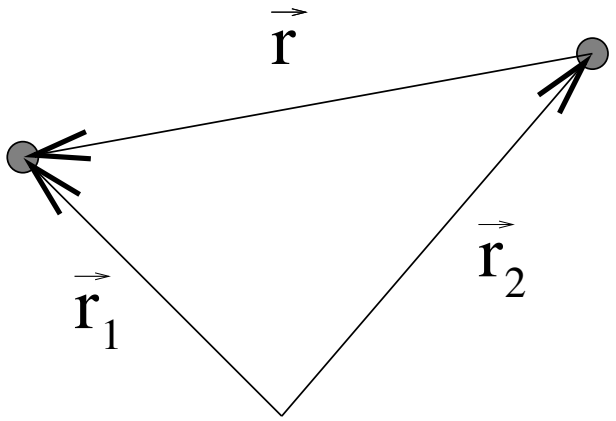


Figure 2: Position vectors of two bodies and their relative distance vector.

other with forces $\vec{F}_1$ acting on body 1 and $\vec{F}_2$ acting on body 2 which are given by

$$\vec{F}_1 = -\frac{Gm_1m_2}{r^3}\vec{r} = -\vec{F}_2\,. \tag{1}$$

$G = 6.67 \times 10^{-11}\frac{\mathrm{m}^3}{\mathrm{kg}\cdot\mathrm{s}^2}$ is Newton's gravitational constant. Newton's second law of motion determines the time evolution of the system. Denoting by a dot one derivate with respect to time t, we have

$$m_1\ddot{\vec{r}}_1 = \vec{F}_1\,, \tag{2}$$
$$m_2\ddot{\vec{r}}_2 = \vec{F}_2\,. \tag{3}$$

Using Eq. (1) we can combine Eq. (2) and Eq. (3) to form an equation for the relative distance vector which reads

$$\ddot{\vec{r}} = -\frac{G(m_1+m_2)}{r^3}\vec{r}. \tag{4}$$

We introduce the reduced mass μ which is defined as

$$\frac{1}{\mu} = \frac{1}{m_1} + \frac{1}{m_2} \quad \Leftrightarrow \quad \mu = \frac{m_1 m_2}{m_1 + m_2}\,. \tag{5}$$

In terms of the reduced mass Eq. (4) assumes the form

$$\mu \ddot{\vec{r}} \;\; = \;\; -\frac{G m_1 m_2}{r^3}\vec{r}, \tag{6}$$

which describes the motion of one body of mass μ with potential energy $U(r) = -Gm_1m_2/r$ and force $\vec{F} = -\vec{\nabla}U = -\frac{\mathrm{d}U}{\mathrm{d}r}\frac{\vec{r}}{r}$.

We choose the center-of-mass frame of the system, which is defined by the following condition

$$m_1\vec{r}_1 + m_2\vec{r}_2 \;\; = \;\; 0\,. \tag{7}$$

Eq. (7) implies

$$\vec{r}_1 = \frac{m_2}{m_1 + m_2}\vec{r} \quad , \quad \vec{r}_2 = -\frac{m_1}{m_1 + m_2}\vec{r}, \tag{8}$$

i.e. in the center-of-mass frame the position vectors are proportional to the distance vector.

Under the assumption (see below) that the solution $\vec{r}(t)$ to Eq. (6) describes a circular orbit of radius R and orbital period T, the acceleration is given by $|\ddot{\vec{r}}| = \omega^2 R$ where $\omega = 2\pi/T$ is the angular frequency. Inserting into Eq. (6) yields the relation $\omega^2 R^3 = G(m_1 + m_2)$. This relation holds also for an elliptical orbit with orbital period T by replacing R with the semi-major axis a of the ellipse:

$$\left(\frac{2\pi}{T}\right)^2 a^3 \;\; = \;\; G(m_1 + m_2)\,. \tag{9}$$

This is Kepler's 3rd law and for a proof we refer for example to [5]. Consider two orbits, one with period T and semi-major axis a and the other with these parameters equal to T' and a'. Eq. (9) implies $(T'/T)^2 = (a'/a)^3$, i.e. the ratio of the squares of the orbital periods is the same as the ratio of the cubes of the orbit sizes.

In the case where one of the body is the Sun ($m_2 = M = 1.99 \times 10^{30}$kg) and the other a planet like the Earth ($m_1 = m = 5.97 \times 10^{24}$kg), since

$m/M \simeq 3.0 \times 10^{-6}$ the sun can be taken approximately as static at the origin of the coordinate system ($\vec{r_2} \approx 0$). In the limit $\mu \approx m$ Eq. (6) becomes

$$\ddot{\vec{r}} = -\frac{GM}{r^3}\vec{r}, \tag{10}$$

where $\vec{r} \approx \vec{r}_1$ descibes the orbit of the planet. The orbits are *conic sections* with the Sun sitting in a focal point. There are three quantities whose values do not change with the time t and are called conserved quantities (or constants of motion): the total energy ($\vec{v} \equiv \dot{\vec{r}}$ is the velocity)

$$E = m\left(\frac{1}{2}\vec{v}^2 - \frac{GM}{r}\right), \tag{11}$$

the angular momentum

$$\vec{L} = m\vec{r} \times \vec{v} \tag{12}$$

and the Runge–Lenz vector (short Lenz vector)

$$\vec{M} = \vec{v} \times \vec{L} - \frac{GMm}{r}\vec{r}. \tag{13}$$

When the total energy E is negative, the orbit is an *ellipse*. For $E < 0$ there is a smallest allowed value of the energy and when the total energy equals this value the ellipse turns into a circle. This situation is represented in Fig. 3. When $E > 0$ the orbit is a hyperbola and when $E = 0$ it is a parabola (with $\vec{v} = 0$ at $r = \infty$). More details can be found in [5].

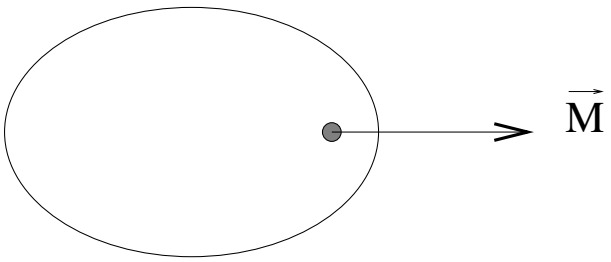


Figure 3: Plane of the elliptical orbit of a planet around the Sun in the focal point (origin). The Lenz vector $\vec{M}$ points to the perihelion.

The angular momentum $\vec{L}$ is by definition perpendicular to the orbital plane spanned by the vectors $\vec{r}$ and $\vec{v}$. Since $\vec{L}$ doesn't change with time, the orbit stays in the same plane.

The Runge–Lenz vector $\vec{M}$ is directed from the origin to the point of the orbit where the planet is closest to the origin (focal point). This point is called *perihelion*. The conservation of $\vec{M}$ ($\dot{\vec{M}} = 0$) means that there is no perihelion precession. In other words, there exist closed orbits. We note that the conservation of the Lenz vector also has implications for the quantum mechanical problem of the hydrogen atom, which exhibits a larger symmetry group SO(4) instead of the usual rotational symmetry group SO(3). This fact was observed by W. Pauli in 1926. Einstein's theory of general relativity provides corrections to the $1/r$ gravitational potential which lead to perihelion precession. The latter has been indeed measured for the planet Mercury.

A final remark concerns the choice of units. For the solar system it is natural to define lengths in Astronomical Units (AU). 1 AU equals to the semi-major axis a of the elliptical orbit of the Earth around the Sun, $a = 1.496 \times 10^{11}$m. For the time it is natural to use units of years (yr) since 1 yr corresponds to the orbital period of the Earth. Using Kepler's 3rd law Eq. (9), the constant GM in Eq. (10) evaluates in these units to $GM = 4\pi^2(\mathrm{AU})^3/(\mathrm{yr})^2$.

6 A note on Molecular Dynamics

6.1 Preparatory considerations

Molecular dynamics deals with the classical motion of many interacting molecules. We follow [3]. Although in principle the problem involves quantum mechanics, the approximation by the classical Newton's equations of motion is good for the following reasons. The binding energy of electrons in atoms or molecules is of the order 10 eV [1] whereas the thermal kinetic energy at room temperature is 26 meV, which is not enough to break the molecule in collisions. The quantum mechanical deBroglie wavelength of molecules at room temperature is smaller than 1 Å (10^{-10} m), which is the typical average spacing of atoms in a solid crystal and we will see that atoms in liquids or gases never get closer than 1 Å during a collision.

The noble gas Argon (Ar) is a popular choice for molecular dynamics. The Ar-Ar interaction is very well described by the Lennard–Jones potential

$$V(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] , \tag{14}$$

where r is the distance between the atoms. It describes a superposition of an attractive Van der Waals force and a repulsive short distance force due to the overlap of the electron clouds of the two atoms. In Fig. 4 we plot the dimensionless potential $V(r)/\epsilon$ and the dimensionless force $F(r)\sigma/\epsilon$ (directed along the line connecting the two atoms), where

$$F(r) = -\frac{\mathrm{d}V}{\mathrm{d}r} = 24\frac{\epsilon}{r} \left[2\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \tag{15}$$

We see that the atoms experience a significant attractive (negative) force in the range $\sim (1.1\text{–}2.0)\,\sigma$. For separations larger than $3\,\sigma$ the force is essentially zero, while for $r \leq 1.1\,\sigma$ the force is very strongly repulsive (positive).

We measure lengths in units of σ, energies in units of ϵ and masses in units of the mass of an argon atom. For argon $\sigma = 3.4\,\text{Å}$, $\epsilon = k_B T$ with $T = 120\,\text{K}$ ($k_B = 1.381 \times 10^{-23}\,\text{m}^2\,\text{kg}\,\text{s}^{-2}\,\text{K}^{-1}$ is the Boltzmann constant) and $m = 6.63 \times 10^{-26}\,\text{kg}$. This fixes all units, in particular the unit of time is $\sqrt{m\sigma^2/\epsilon} \approx 2 \times 10^{-12}\,\text{s}$.

[1] 1 eV (electron volt) is the energy change of one electron which traverses a potential difference of 1 V (Volt). It corresponds to 1.602×10^{-19} J (Joule).

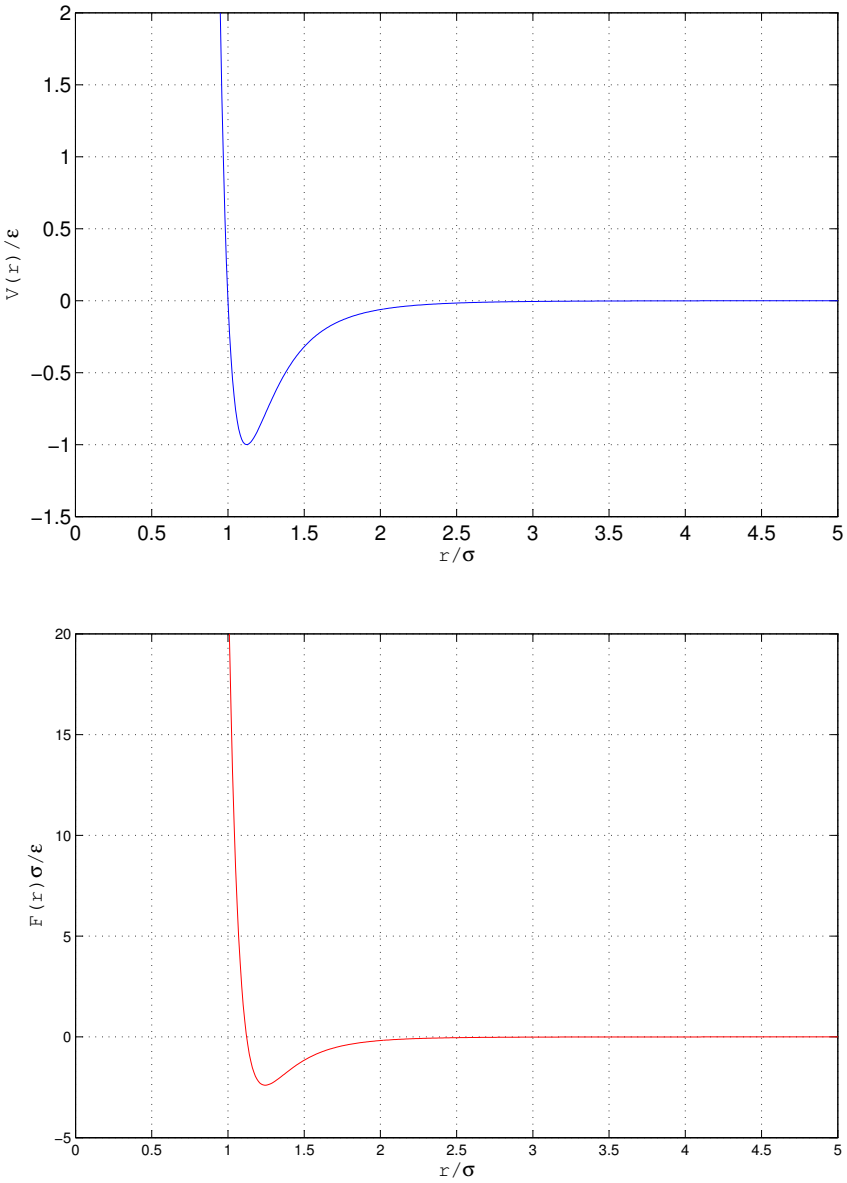


Figure 4: The dimensionless Lennard–Jones potential Eq. (14) and the force Eq. (15).

6.2 Equations of motion and the leapfrog algorithmus

In the following we consider a two-dimensional $L \times L$ box where the argon atoms can move. Their coordinates (x, y) satisfy $0 \leq x, y \leq L$. The numerical simulations in higher dimensions are much more costly, although the algorithms can be extended in a straightforward way. The interactions between $i = 1, 2, \ldots, N$ atoms with coordinates $\vec{r}_i = (x_i, y_i)$ are described by the potential

$$\sum_{i<j} V(|\vec{r}_i - \vec{r}_j|)\,, \tag{16}$$

where the sum extends over all Ar-Ar pairs (i, j). Newton's equations of motion are ($\dot{} \equiv \mathrm{d}/\mathrm{d}t$)

$$\begin{aligned} \ddot{\vec{r}}_i &= -\sum_{j\neq i} \vec{\nabla}_i V(|\vec{r}_i - \vec{r}_j|) \\ &= \sum_{j\neq i} F(|\vec{r}_i - \vec{r}_j|)\frac{\vec{r}_i - \vec{r}_j}{|\vec{r}_i - \vec{r}_j|}\,, \end{aligned} \tag{17}$$

where $i, j = 1, \ldots, N$ and $\vec{\nabla}_i = (\partial/\partial x_i, \partial/\partial y_i)$. The numerical integration could be performed in MATLAB using the built-in 4th order Runge-Kutta integrator `ode45`. For that one builds a $4N$-component vector

$$\vec{y} = (x_1, \ldots, x_N, y_1, \ldots, y_N, v_{x,1}, \ldots, v_{x,N}, v_{y,1}, \ldots v_{y,N})\,, \tag{18}$$

which contains the 2d positions and velocities of the Ar-atoms. Eq. (17) can then be cast into the form

$$\frac{\mathrm{d}}{\mathrm{d}t}\vec{y}(t) = \vec{f}(t, \vec{y})\,, \tag{19}$$

where

$$\vec{f}(\vec{y}) = (v_{x,1}, \ldots, v_{x,N}, v_{y,1}, \ldots v_{y,N}, G_{x,1}, \ldots, G_{x,N}, G_{y,1}, \ldots, G_{y,N}) \tag{20}$$

with

$$\vec{G}_i[\vec{r}_k] = (G_{x,i}, G_{y,i}) = \sum_{j\neq i} F(|\vec{r}_i - \vec{r}_j|)\frac{\vec{r}_i - \vec{r}_j}{|\vec{r}_i - \vec{r}_j|}\,. \tag{21}$$

Here we want instead to implement our "own" integrator and choose the leap-frog integrator. It belongs to the class of symplectic numerical integrators which are area-preserving and time-reversible. The leap-frog algorithm

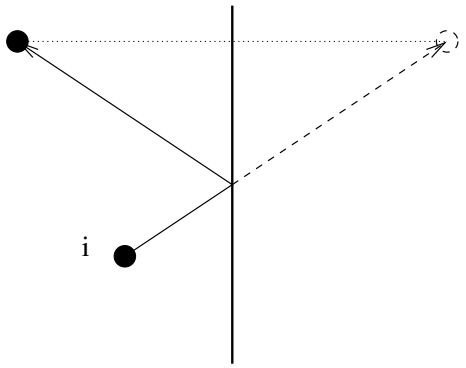


Figure 5: Reflection of Ar-atoms on the box wall.

introduces a time step-size τ. The positions $\vec{r}_i(n\tau)$ are computed at times $n\tau$, $n = 0, 1, 2, \ldots$ and the velocities $\vec{v}_i((n+1/2)\tau)$ in between. The equations for the step $n \longrightarrow n+1$ are

$$\vec{r}_i((n+1)\tau) = \vec{r}_i(n\tau) + \tau\vec{v}_i((n+1/2)\tau)\,, \tag{22}$$

$$\vec{v}_i((n+3/2)\tau) = \vec{v}_i((n+1/2)\tau) + \tau\vec{G}_i[\vec{r}_k((n+1)\tau)]\,. \tag{23}$$

The algorithm is exactly time reversible. This means that changing $\tau \longrightarrow -\tau$ and taking the final values for $\vec{r}$ and $\vec{v}$ as initial values, the leap-frog integration goes backwards through the same values (up to roundoff errors). This property holds for the exact differential equation Eq. (17) and therefore also for the the discretization errors of the leap-frog algorithm. It follows that for each leap-frog step the discretization errors are $\mathrm{O}(\tau^3)$, because even powers (τ^2) are excluded. Please check explicitely the time reversibility of the leap-frog equations. For usual initial value problems, where $\vec{r}_i(0)$ and $\vec{v}_i(0)$ are prescribed, one needs an extra half step at the beginning

$$\vec{v}_i(\tau/2) = \vec{v}_i(0) + \frac{\tau}{2}\vec{G}_i[\vec{r}_k(0)]\,. \tag{24}$$

If the velocities $\vec{v}_i(0)$ are randomly chosen this half step is not necessary.

In a finite box we have to worry about boundary conditions. One possibility is to impose periodic boundary conditions: when an atom hits the wall it "reappears" on the opposite side of the box, which becomes in this case a torus (a bycicle tire with two identical radii). Periodic boundary conditions require a modification of the definition of distance, since distances

are measured "around the torus". We take here a different approach. The leap-frog algorithm divides the motion of particles in small steps of uniform motion (cf. Eq. (22)), where the atoms move independently from each other, and correction steps of the velocities in between (cf. Eq. (23)). If a particle crosses the box we reflect it at the box's walls.[2] If the i-th atom in the *straight* path $\vec{r}_i(n\tau) \longrightarrow \vec{r}_i(n\tau) + \tau\vec{v}_i((n+1/2)\tau)$ (cf. Eq. (22)) crosses one of the box's walls, the atom is elastically reflected inside the box, see Fig. 5. This implies that the velocity component of the i-th atom which is perpendicular to the wall changes its sign

$$v_{i\perp} \longrightarrow -v_{i\perp}\,, \tag{25}$$

and the atom is put back into the box as if the reflection had happened exactly on the wall. For the position component $r_{i\perp}$ perpendicular to the wall this implies

$$r_{i\perp} \longrightarrow \begin{cases} -r_{i\perp}\,, & \text{if } r_{i\perp} < 0 \\ 2L - r_{i\perp}\,, & \text{if } r_{i\perp} > L \end{cases}\,. \tag{26}$$

The reflection is done in the steps of uniform motion and *before* the force computation needed to update the velocities in step Eq. (23).

6.3 MATLAB implementation

The code for the leap-frog integrator is

```
1 function [X,Y,VX,VY]=leapfrog(x,y,vx,vy,L,tau,nstep)
2
3 % number of particles
4 N=length(x);
5 X =zeros(N,nstep+1); Y =zeros(size(X));
6 VX=zeros(N,nstep+1); VY=zeros(size(X));
7 % start values
8 X(:,1)= x;  Y(:,1)= y;
9 VX(:,1)=vx; VY(:,1)=vy;
```

[2] The reflection at the box's walls is equivalent to including during the steps of uniform motion a potential energy term like $V_{\rm Box}(\vec{r}) = B\{\exp(-x/\lambda) + \exp(-(L-x)/\lambda) + (x \leftrightarrow y)\}$ with parameters (λ, B) and considering the idealized limit of an infinitely high and infinitesimally thin wall $\lim_{\lambda\to 0+,\, B>0}$.

```
10
11 for n=1:nstep
12     % positions: n*tau -> (n+1)*tau
13     X(:,n+1)=X(:,n)+tau*VX(:,n);
14     % using v evaluated at (n+1/2)*tau
15     Y(:,n+1)=Y(:,n)+tau*VY(:,n);
16     % reflections of particles at the box boundaries:
17     for i=1:N
18         if X(i,n+1)<0, X(i,n+1)=   -X(i,n+1);
19            VX(i,n)=-VX(i,n); end
20         if X(i,n+1)>L, X(i,n+1)=2*L-X(i,n+1);
21            VX(i,n)=-VX(i,n); end
22         if Y(i,n+1)<0, Y(i,n+1)=   -Y(i,n+1);
23            VY(i,n)=-VY(i,n); end
24         if Y(i,n+1)>L, Y(i,n+1)=2*L-Y(i,n+1);
25            VY(i,n)=-VY(i,n); end
26     end
27
28     [fx,fy]=LJforce(X(:,n+1),Y(:,n+1));
29
30     % momenta: (n+1/2)*tau -> (n+3/2)*tau
31     VX(:,n+1)=VX(:,n)+tau*fx;
32     % using fx,fy evaluated at (n+1)*tau
33     VY(:,n+1)=VY(:,n)+tau*fy;
34 end
```

In the arguments of the function there are the initial values at $t = 0$ for the positions and at $t = \tau/2$ for the velocities. A number `nstep` of leap-frog integration steps is made and the values are stored in a matrix, like with `ode`. In MATLAB there are only integer indices and therefore the components of $\vec{v}_i((n + 1/2)\tau)$ are stored in the same matrix columns as the ones of $\vec{r}_i(n\tau)$.

The force is computed by a call to

```
1 function [fx fy] = LJforce(x,y)
2 % Force from Lennard-Jones potential for N particles (2D)
3 N=length(x);
4 fx=zeros(size(x)); fy=zeros(size(fx)); % reserve storage
5 % loop over pairs of distinct particles
```

```
6  % (each pair counted only one time, i<j !)
7  for i=1:N-1
8      for j=i+1:N
9      r2=(x(i)-x(j))^2+(y(i)-y(j))^2; % distance^2
10     ri2=1/r2;
11     fac=24*ri2^4*(2*ri2^3-1);
12     % contribution of this pair-force acting ON i
13     fx(i)=fx(i)+(x(i)-x(j))*fac;
14     fy(i)=fy(i)+(y(i)-y(j))*fac;
15     % contribution of this pair-force acting ON j
16     fx(j)=fx(j)-(x(i)-x(j))*fac;
17     fy(j)=fy(j)-(y(i)-y(j))*fac;% Sign: actio=reactio
18     end % loop j
19 end      % loop i; all forces are taken into account
```

Additional comments are:

- line 3: the number of atoms N is determined from the argument x
- lines 7,8: double loop over the pairs of atoms $i < j$
- lines 13,14: force on the atom i
- lines 16,17: force on the atom j, opposite sign due to actio = reactio

Please verify that the force terms in Eq. (21) are indeed implemented by this function.

The cost of the force computation grows proportional to N^2 and this limits the number of particles N that can be simulated. The computation of such pair forces is an important problem and its efficient implementation on parallel computers is a topic of algorithmic research. In [6] for example an algorithm is discussed which should improve the scaling behavior at least effectively to $\propto N^{3/2}$.

Finally we consider a test application. The MATLAB script is

```
1  % test program of molecular dynamics
2  clf;hold off; % free the plot window
3
4  L=6;  % side length of the box
```

```
5
6  % Initial configuration:
7  N=5*5;  % Number of particles
8  x=zeros(N,1); y=zeros(N,1); % reserve storage
9  vx=zeros(N,1); vy=zeros(N,1); % reserve storage
10
11 i=0;
12 for yi=1:5
13     for xi=1:5
14         i=i+1;
15         x(i)=xi; y(i)=yi;
16     end
17 end
18
19 % Plot the start configuration:
20 plot(x,y,'*r');axis([0 L 0 L]);hold on;
21 pause;
22
23 % random initial momenta
24 vmax=1; % maximal value
25 % 0 < rand < 1 random
26 vx=vmax*(2*rand(N,1)-1);vy=vmax*(2*rand(N,1)-1);
27 % total momentum is set to zero
28 vx=vx-mean(vx);vy=vy-mean(vy);
29
30 tf=20;                  % final time of the integration
31                         % of equations of motion
32 tau=0.005;              % step size
33 nstep=fix(tf/tau);      % number of steps
34
35 [X,Y,VX,VY]=leapfrog(x,y,vx,vy,L,tau,nstep);
36
37 % plot all the trajectoris with thin points
38 for i=1:N
39     plot(X(i,:),Y(i,:),'MarkerSize',1,'Marker','.',...
40         'LineStyle','none');
41 end
42 pause
```

```
43 print -depsc molecular.eps
44
45 % time series of configurations
46 strob=fix((tf/50)/tau); % frequency of plots
47 for n=1:strob:nstep+1
48     clf
49     plot(X(:,n),Y(:,n),'.');axis([0 L 0 L])
50     pause(0.2)
51 end
52 %print -depsc mol_final.eps
53 % save parameters and last configuration
54 x=X(:,nstep+1);y=Y(:,nstep+1); vx=VX(:,nstep+1);
55           vy=VY(:,nstep+1);
56 save gas.mat N L x y vx vy
```

Additional comments are:

- lines 6–17: the atoms are set to an arbitrary initial configuration with distance 1. If we take a distance significantly smaller than the minimum of the Lennard–Jones potential we get a very high energy
- lines 23–36: here we set random initial velocities in each direction with maximal magnitude 1 and zero total momentum $\sum_i \vec{v}_i = 0$
- lines 54–56: the parameters of the last configuration are saved; these can for example be read in later as initial values to continue the integration

Fig. 6 shows the starting configuration (stars) and the resulting configurations ("trajectories") from the integration (lines). In the last part of the script a sort of movie of the successive configurations is made.

6.4 Interpretation

The energy of the system

$$E = E_{\text{kin}} + E_{\text{pot}} = \frac{1}{2}\sum_{i=1}^{N}(\dot{\vec{r}}_i)^2 + \sum_{i<j} V(|\vec{r}_i - \vec{r}_j|) \tag{27}$$

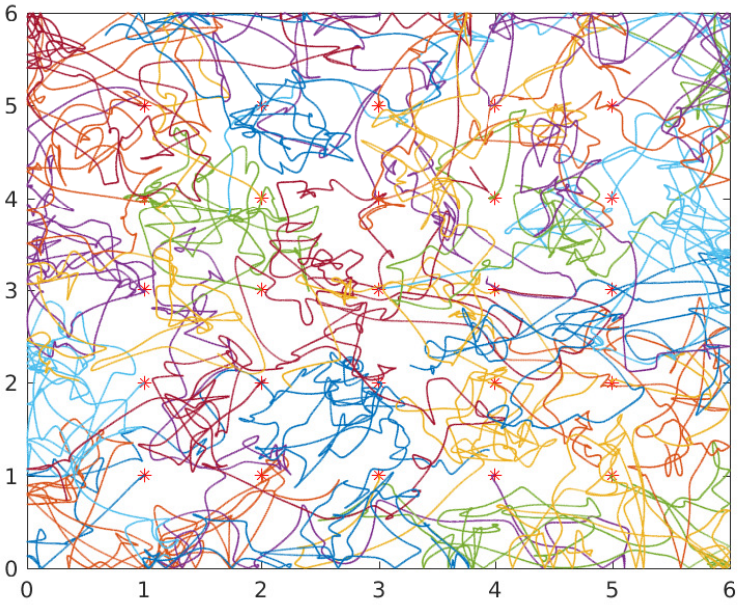


Figure 6: Molecular dynamics motion of Ar-atoms. Initial positions and trajectories, $t = 0, \ldots, 20$ with $\tau = 0.005$.

is exactly conserved if the time integration is exact. Numerically it is conserved up to discretization errors, which are $\propto \tau^2$ using the leap-frog integrator over a fixed time interval T. The arbitrary initial configuration of the atoms, which is given as input, determines the energy. After some molecular dynamics time (thermalization), the system reaches equilibrium and physical thermodynamical quantities like temperature, pressure etc. can be determined. An isolated system with fixed energy is called *microcanonical*. The temperature can be obtained[3] from the average kinetic energy pro particle

$$\frac{1}{2}k_B T = \langle \frac{1}{2}v_x^2 \rangle = \langle \frac{1}{2}v_y^2 \rangle \,. \tag{28}$$

The average can be taken over the N atoms at a certain time but also over consequent (in time) configurations of the atoms after equilibrium. A neces-

[3] This is valid only if the total momentum of the system is zero, $m\sum_i \dot{\vec{r}}_i = 0$. In collisions with the walls the momentum is not conserved. But on average the momentum transfer to the walls is zero, since no direction is preferred for the particle motion.

sary (but not sufficient) condition for reaching equilibrium is that both $E_{\rm kin}$ and $E_{\rm pot}$ fluctuate around their average value (only their sum is fixed). The Boltzmann distribution of velocities is given by

$$P(v_x)P(v_y)\mathrm{d}v_x\mathrm{d}v_y = \frac{1}{2\pi k_B T}\mathrm{e}^{-(v_x^2+v_y^2)/(2k_BT)}\mathrm{d}v_x\mathrm{d}v_y\,. \tag{29}$$

The distribution of speed $v = \sqrt{v_x^2 + v_y^2}$ can be obtained by integrating the velocity distribution over a "shell" between v and $v + \mathrm{d}v$

$$P(v)\mathrm{d}v = \frac{v}{k_B T}\mathrm{e}^{-v^2/(2k_BT)}\mathrm{d}v\,, \tag{30}$$

as it can be seen by changing in Eq. (29) to polar coordinates $v_x = v\cos(\phi)$, $v_y = v\sin(\phi)$ with $\mathrm{d}v_x\mathrm{d}v_y = v\mathrm{d}\phi\mathrm{d}v$ and integrating over the angle ϕ.

With molecular dynamics simulations we can also study non-equilibrium processes, when for example the temperature of the system is increased. The first order melting phase transition from a crystal to a fluid is an example.

7 Linear systems of equations

$Ax = b$ with A and b given, to solve for x

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

$$m \times n \qquad n \qquad m$$

$$\leftrightarrow \sum_{j=1}^{n} a_{ij} x_j = b_i, \qquad i = 1, \dots, m \qquad a_{ij}, b_i \in \mathbb{C} \text{ or } \mathbb{R}$$

here: quadratic case $m = n$, A non-singular $\Leftrightarrow \det(A) \neq 0 \rightarrow \exists A^{-1}$,
unique solution is $x = A^{-1}b$, we consider real case

also: x and b can have more columns $\leftrightarrow$ solve simultaneously for several
right-hand sides b

general case: $m \neq n$ or A (almost) singular for $m = n$
$\rightarrow$ Singular Value Decomposition (SVD)

7.1 Naive Gaussian Elimination

idea: build linear combinations of equations
transform to an equivalent triangular system that can be solved easily

$$\begin{pmatrix} & & \\ & \diagdown & \\ 0 & & \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b'_1 \\ b'_2 \\ \vdots \\ b'_m \end{pmatrix}$$

```
>> load beispiel
>> A
>> b
```

$a_{1j}x_j = b_1$ / multiply by a_{21}/a_{11} and

implicit summation convention subtract from 2nd equation

$a_{2j}x_j = b_2$

$\rightarrow a_{2j}x_j - \frac{a_{21}}{a_{11}}(a_{1j}x_j) = b_2 - \frac{a_{21}}{a_{11}}b_1$ term $a_{21}x_1$ cancels

and similarly for 3rd and 4th equation

```
>> B=[A  b]
>> for  i=2:4,
     B(i,:)=B(i,:)-B(1,:)*B(i,1)/B(1,1);
   end
>> B
```

we generated zeros below diagonal in first column, now subtract 2nd equation from 3rd and 4th to create zeros in the 2nd column

```
>> for  i=3:4,
     B(i,:)=B(i,:)-B(2,:)*B(i,2)/B(2,2);
   end
>> B
```

and finally

```
>> B(4,:)=B(4,:)-B(3,:)*B(4,3)/B(3,3);
>> B
```

the transformed system has zeros below the diagonal

$\rightarrow$ solve by **backsubstitution**

$$\left.\begin{array}{rcl} -3x_4 &=& -3 \\ 2x_3 &=& -9+5x_4 \\ -4x_2 &=& -6-2x_3-2x_4 \\ 6x_1 &=& 16+2x_2-2x_3-4x_4 \end{array}\right\} \Rightarrow \begin{array}{r} x_4 = 1 \\ x_3 = -2 \\ x_2 = 1 \\ x_1 = 3 \end{array}$$

MATLAB: `>> help slash, mldivide (\), mrdivide (/)`

$Ax = b \Leftrightarrow$ `x=A\b` "divide both sides from left by A"

```
>> x=A\b
```

http://csis.uni-wuppertal.de/courses/linear_sys/gaussel.m

```
%
%  file gaussel.m
%
%  x=gaussel(A,b) returns the solution of the set of
%  linear algebraic equations A*x=b using Gauss elimination
%  without pivoting. b and x can have several columns
%
%
function [x] = gaussel(A,b)
[m,n]=size(A);
if m~=n | n~=size(b,1), error('not a square matrix problem'); end;

B=[A b];
N=size(B,2);

% bring the matrix into triangular form (Gauss elimination):

for k=1:n-1,     % loop over columns where the zeros will appear
   fac=1/B(k,k);
   for i=k+1:n   % loop over rows where subtractions take place
      fac1=fac*B(i,k); % factor
      B(i,k)=0; % new zero by construction
      B(i,k+1:N)=B(i,k+1:N)-B(k,k+1:N)*fac1; % subtraction
   end
end

% Solution by backsubstitution :
x=zeros(size(b)); % predefinition of x
for k=n:-1:1
   x(k,:)=B(k,n+1:N);
   for j=k+1:n
      x(k,:)=x(k,:)-B(k,j)*x(j,:);
   end
   x(k,:)=x(k,:)/B(k,k);
end
```

backsubstitution (for one right-hand side):

$$x_k = \frac{B_{k,n+1} - \sum_{j=k+1}^{n} B_{kj} x_j}{B_{kk}}, \qquad k = n, n-1, \ldots, 1$$

`gaussel.m` works for several right-hand sides: `N=size(B,2)`; `N>n+1`.

$\rightarrow b = I_{n\times n}$ identity matrix $\Rightarrow x = A^{-1}$!

```
>> b=eye(4)
>> x=gaussel(A,b)
>> x*A
>> A*x
```

Numerical accuracy?

`A=rand(n,n)` `x=rand(n,1)` $\Rightarrow$ $b = Ax$

Solve $Ay = b$ numerically using `gaussel(A,b)` or `A\b` with MATLAB

Compute $\delta = \max_i |x_i - y_i|$

```
>> timing
```

Semilogarithmic plot of δ vs. n

- numerical errors grow with n
- irregularity $\leftrightarrow$ `rand`
- MATLAB errors are 10-100 time smaller
  $\rightarrow$ Gaussian elimination with pivoting

7.2 Pivoting

Until now:

- ordering of rows $\leftrightarrow$ equations of B plays a role in Gaussian elimination; this is unnatural, since n equations can be written in arbitrary order
- $a_{11} = 0$ is possible in non-singular systems $\rightarrow$ leads to division by zero

This is the motivation for **pivoting**:

- **partial pivoting:** permutation of rows
- **full pivoting:** also permutation of columns means changing order of the components x_i

Partial pivoting:

start the Gaussian elimination with the row j, where $|a_{j1}|$ is maximal $\rightarrow$ the pivot element or **pivot**; this amounts to a permutation of row 1 with row j.

if all $|a_{j1}| = 0 \Rightarrow A$ is singular

then take the row where $|a_{j2}|$ is maximal, etc.

Permutation of rows:

in practice do not swap the rows in memory instead: **bookkeeping** of the permutations:

- initialize a vector `p=[1 2 ... n]`
- exchange the components of `p`!
  this generates a permutation `[p(1) p(2) ... p(n)]`
  address row `i` as `B(p(i),:)`

Scaling (implicit pivoting):

there is an arbitrariness: the first equation can be multiplied by a faction million and it is almost guaranteed that a_{11} (if $\neq 0$) will become the pivot;

implicit pivoting makes use of the freedom to rescale the equations by a scale factor $s(i)$

$a_{ij} \rightarrow s(i)a_{ij} \quad (j = 1, \ldots, n)$ and $b_i \rightarrow s(i)b_i$ such that $\max_j |a_{ij}| = 1$

in praxis this does often not lead to an improvement

7.3 Iterative improvement of the solution

$Ax = b \;\rightarrow$ solution $x^{(0)}$

roundoff errors: numerically $Ax^{(0)} =: \tilde{b} \neq b$

we denote the exact solution by $x^* = A^{-1}b$

the numerical error $\delta x = x^{(0)} - x^*$ satisfies

$$A\delta x = Ax^{(0)} - Ax^* = \tilde{b} - b = \delta b$$

$\rightarrow$ solve for δx and improve the solution $x^{(1)} := x^{(0)} - \delta x$

this procedure can be iterated:

compute $\delta b = Ax^{(n)} - b$

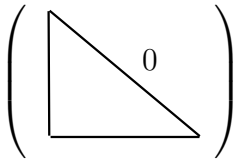
solve $A\delta x = \delta b$ for $\delta x = x^{(n)} - x^*$

improve $x^{(n+1)} = x^{(n)} - \delta x$

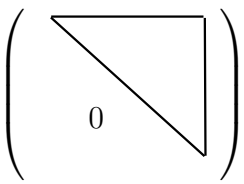
comment: we have to solve linear systems for the same A and different right-hand sides $\rightarrow$ advantage of decomposition methods

7.4 LU Decomposition

$A = L \cdot U$; $\quad$ A is a real or complex $n \times n$ matrix



lower triangular matrix $l_{ij} = 0$ for $i < j$



upper triangular matrix $u_{ij} = 0$ for $i > j$

number of elements of L and U:

$$(\sum_{i=1}^{n} i = \frac{n(n+1)}{2}) \times 2 = n^2 + n \text{ elements}$$

A has n^2 elements $\rightarrow$ too many

Convention: $l_{ii} = 1 \quad i = 1, \ldots, n \ \rightarrow \ n^2$ elements

Proof: construction, Crout's algorithm (see Sect. 7.6 and [2] Sect. 2.3).

$Ax = b$ is equivalent to $L \cdot Ux = b \Rightarrow$ solve $Ly = b$ and then $Ux = y$:

$$\begin{pmatrix} & \\ & \diagdown \\ 0 & & \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \quad \text{forward substitution (begin from top)}$$

$$\begin{pmatrix} & \diagdown & \\ & & \\ 0 & & \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \quad \text{backward substitution (begin from bottom)}$$

LU decomposition is the algebraic description of Gaussian elimination (see [7] Sect. 3.2) with the advantage, when L, U are known, one can solve $Ax = b$ for several right-hand sides independently

7.5 Householder reduction

- reduce real matrix A to triangular form through orthogonal transformations:

$$OA = \begin{pmatrix} & \diagdown & \\ & & \\ 0 & & \end{pmatrix} = R \quad \text{upper triangular}$$

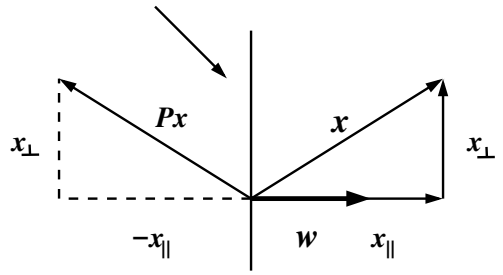
O is orthogonal: $O^T O = OO^T = \mathbb{I}$

- replaces the Gaussian elimination
- since $O^T O = \mathbb{I}$, numerically stable
- $O = \prod$ reflections
- factorization: $A = O^T R$

Reflection through a hyperplane in n dimensions:

unit vector w, $w^T w = 1$

$(n-1)-$dimensional hyperplane orthogonal to w



reflection $x \to Px$:

$$x_{||} = (w^T x)w : \text{ vector projection of } x \text{ onto } w$$

$$x = x_{||} + x_\perp \;\to\; Px = -x_{||} + x_\perp = x - 2x_{||}$$

$$Px = x - 2w(w^T x) = (\mathbb{I} - 2\underbrace{ww^T})x$$

$$M = ww^T \quad n \times n - \text{matrix} : m_{ij} = w_i w_j$$

$$\Rightarrow P = \mathbb{I} - 2ww^T$$

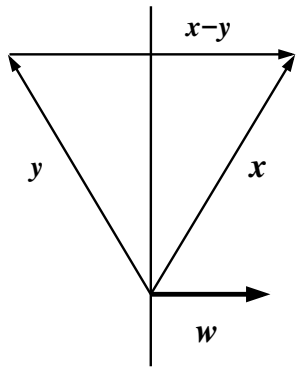
$$P^T = P \quad : \quad P \text{ is symmetric}$$

$$P^2 = (\mathbb{I} - 2ww^T)(\mathbb{I} - 2ww^T) = \mathbb{I} - 4ww^T + 4w\underbrace{w^T w}_{1}w^T = \mathbb{I}$$

$\Rightarrow P$ is orthogonal: $P^TP = PP^T = P^2 = \mathbb{I}$

Lemma: given two vectors $x \neq y$ of same length: $|x|^2 = x^Tx = y^Ty = |y|^2$, there exists a reflection P such that $Px = y (\Rightarrow P^2x = x = Py)$

Proof:



$$w = \frac{x-y}{|x-y|}$$

$$ww^Tx = (x-y)\frac{(x^T-y^T)x}{|x-y|^2} =$$

// $y^Tx = x^Ty$; $x^Tx = y^Ty$ by assumption

$$= (x-y)\frac{\frac{1}{2}(x^Tx + y^Ty - y^Tx - x^Ty)}{|x-y|^2}$$

$$= (x-y)\frac{\frac{1}{2}(x-y)^T(x-y)}{|x-y|^2} = \frac{1}{2}(x-y)\frac{|x-y|^2}{|x-y|^2} = \frac{1}{2}(x-y)$$

$\Rightarrow Px = (\mathbb{I} - 2ww^T)x = x - 2\frac{1}{2}(x-y) = y$ ■

1st step of the Householder reduction: $a^{(1)}$ = first column of A ($n \times n$ real matrix) $\Rightarrow$ construct reflection P_1 such that

$$a^{(1)} = \underbrace{\begin{pmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{pmatrix}}_{x} \xrightarrow{P_1 a^{(1)}} \underbrace{\begin{pmatrix} -\sigma_1 \\ 0 \\ | \\ 0 \end{pmatrix}}_{P_1 x = y} \qquad P_1 \underbrace{a^{(1)}}_{:=x} = \underbrace{-\sigma_1 e_1}_{:=y} \text{ with } \sigma_1 = \pm|a^{(1)}|$$

(choice $\pm$: later)

$$(x-y)^T =: u^T = (a_{11} + \sigma_1, a_{21}, \ldots, a_{n1}), \qquad P_1 = \mathbb{I} - uu^T/H$$

$$H = \tfrac{1}{2}u^Tu = \tfrac{1}{2}(a^{(1)}+\sigma_1 e_1)^T(a^{(1)}+\sigma_1 e_1) = \tfrac{1}{2}(\underbrace{|a^{(1)}|^2}_{=\sigma_1^2} + 2\sigma_1 a_{11} + \sigma_1^2) = \sigma_1(\sigma_1 + a_{11})$$

H is larger, when signσ_1 =signa_{11} : σ_1 =sign$(a_{11})|a^{(1)}|$

$\Rightarrow$ smaller roundoff errors in the addition $\sigma_1 + a_{11}$

No pivoting is required: always $|a^{(1)}|$ on the diagonal, independently on the ordering of the elements a_{i1}. If $|a^{(1)}| = 0 \Rightarrow$ matrix is singular; numerically this happens when $|a^{(1)}| \leq \epsilon_m a$, where a is a typical element a_{ij}

$$
\begin{aligned}
Ax &= b \Rightarrow P_1 Ax = P_1 b \\
P_1 A &= \begin{pmatrix} -\sigma_1 & a'_{12} & & \\ 0 & a'_{22} & & \\ 0 & a'_{32} & \cdots & \\ | & \vdots & & \\ 0 & a'_{n2} & & \end{pmatrix}
\end{aligned}
$$

2nd step of the Householder reduction:

$$
\begin{aligned}
u^T &= (0, a'_{22} + \sigma_2, a'_{32}, \ldots, a'_{n2}) \\
\sigma_2 &= \text{sign}(a'_{22}) \sqrt{\sum_{i=2}^{n} (a'_{i2})^2} \\
P_2 &= \mathbb{I} - \frac{uu^T}{H} = \begin{pmatrix} 1 & 0 \cdots 0 \\ 0 & \\ | & \quad \mathbb{I} \\ 0 & \end{pmatrix} \\
H &= \sigma_2(\sigma_2 + a'_{22}) \\
a'_{ij} &= (P_1 A)_{ij} \\
P_2 P_1 A &= \begin{pmatrix} -\sigma_1 & a'_{12} & \cdots & a'_{1n} \\ 0 & -\sigma_2 & & \\ | & 0 & \cdots & \\ | & | & & \\ 0 & 0 & & \end{pmatrix}
\end{aligned}
$$

$\rightarrow$ first row and column of $P_1 A$ remain unchanged

$$\ldots \text{ until } P_{n-1} \;\Rightarrow\; \underbrace{P_{n-1}P_{n-2}\ldots P_2P_1}_{=:Q^T} A = \begin{pmatrix} & & \\ & & \\ 0 & & \end{pmatrix} = R$$

$$\begin{aligned} Ax = b \;\Rightarrow\; Q^TAx &= Q^Tb \\ Rx &= b';\; b' = Q^Tb \end{aligned}$$

$$\begin{aligned} Q &= (P_{n-1}P_{n-2}\ldots P_2P_1)^T = P_1P_2\ldots P_{n-2}P_{n-1} \\ Q^TQ &= P_{n-1}P_{n-2}\ldots P_2\underbrace{P_1P_1}_{=\mathbb{I}\ldots}P_2\ldots P_{n-2}P_{n-1} = \mathbb{I} \end{aligned}$$

$\Rightarrow Q$ is orthogonal

$\Rightarrow A = QR$ =(orthogonal)×(upper triangular) $\Rightarrow$ "QR decomposition"

$A^{-1} = R^{-1}Q^T$; $\quad R^{-1}$ is simple to compute since triangular:

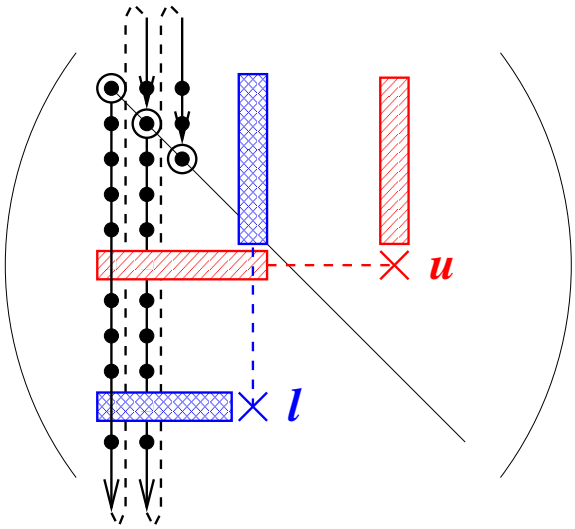
$AX = \mathbb{I} \;\Leftrightarrow\; RX = Q^T$ solve for $X = A^{-1}$ by backsubstitution

QR decomposition needs more operations than Gauss elimination (about twice as many operations than LU decomposition), but is uniform (no pivoting) and numerically stable

7.6 Crout's algorithm for LU decomposition

$$A = L \cdot U \;\Leftrightarrow\; a_{ij} = \sum_{k=1}^{n} l_{ik}u_{kj} = \sum_{k=1}^{\min(i,j)} l_{ik}u_{kj}$$

Crout's algorithm defines an explicit sequence to solve for the elements of L and U:



$$a_{11} = \underbrace{l_{11}}_{=1} u_{11} \;\Rightarrow\; u_{11} = a_{11}$$

$$a_{21} = l_{21}u_{11} \;\Rightarrow\; l_{21} = \frac{a_{21}}{u_{11}}$$

$$\vdots$$

$$a_{n1} = l_{n1}u_{11} \;\Rightarrow\; l_{n1} = \frac{a_{n1}}{u_{11}}$$

then

$$a_{12} = \underbrace{l_{11}}_{=1} u_{12} \;\Rightarrow\; u_{12} = a_{12}$$

$$\begin{aligned}
a_{22} &= l_{21}u_{12} + \underbrace{l_{22}}_{=1} u_{22} \;\Rightarrow\; u_{22} = a_{22} - l_{21}u_{12} \\
a_{32} &= l_{31}u_{12} + l_{32}u_{22} \;\Rightarrow\; l_{32} = \frac{1}{u_{22}}(a_{32} - l_{31}u_{12}) \\
&\vdots \\
a_{n2} &= l_{n1}u_{12} + l_{n2}u_{22} \;\Rightarrow\; l_{n2} = \frac{1}{u_{22}}(a_{n2} - l_{n1}u_{12}) \\
&\vdots
\end{aligned}$$

General formulae:

$$i \le j \quad : \quad a_{ij} = \sum_{k=1}^{i} l_{ik}u_{kj} = \underbrace{l_{ii}}_{=1} u_{ij} + \sum_{k=1}^{i-1} l_{ik}u_{kj} \Rightarrow u_{ij} = a_{ij} - \underbrace{\sum_{k=1}^{i-1} l_{ik}u_{kj}}$$

have already been determined, (see figure above)

$$i > j \quad : \quad a_{ij} = \sum_{k=1}^{j} l_{ik}u_{kj} = l_{ij}u_{jj} + \sum_{k=1}^{j-1} l_{ik}u_{kj} \Rightarrow l_{ij} = \frac{1}{u_{jj}}(a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj})$$

Sequence: $u_{11}, l_{21}, l_{31}, \ldots, l_{n1}, u_{12}, u_{22}, l_{32}, l_{42}, \ldots, l_{n2}, u_{13}, u_{23}, u_{33}, l_{43}, \ldots, u_{nn}$

Pivoting:

assume that we reached the point of

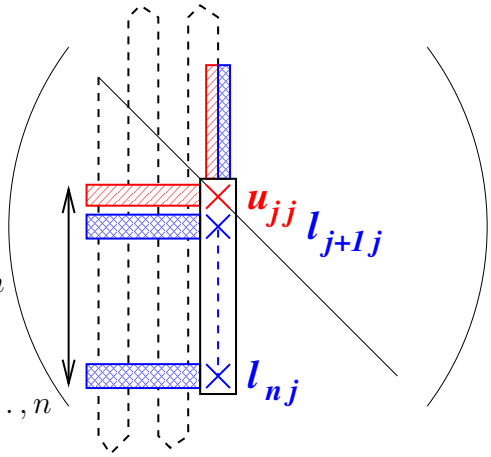
computation for $u_{jj}, l_{j+1,j}, l_{j+2,j}, \ldots, l_{nj}$

we define

$c_i = a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}$ for $i = j, j+1, \ldots, n$

the naive procedure is:

$u_{jj} = c_j,\ l_{ij} = \frac{1}{c_j} c_i$ for $i = j+1, j+2, \ldots, n$



Pivoting: divide by c_k for which $|c_k| = \max_i |c_i|$

This corresponds to interchanging row j and k $(k \geq j)$

$\rightarrow$ interchange c_j, c_k (horizontal strips in the figure)

$\rightarrow$ transposition of j, k is stored in a permutation vector

$\rightarrow$ it affects the part of L which has already been computed and A

Pivoting is equivalent to the LU decomposition of a form of the matrix A where rows have been permuted: $PA = LU$

Storage: replace the elements a_{ij} with the computed u_{ij}, l_{ij}: a_{ij} will not be needed any more (pivoting included)

Cost:

LU decomposition: $\sim n^3/3$ multiplications

Solution of $Ax = b$ afterwards: $\sim n^2$ for each vector b

Inversion $(b = e_1, e_2, \ldots, e_n)$: n^3 (considering the zeros)

MATLAB: `>>[L,U,P]=lu(A)` $\Leftrightarrow$ $PA = LU$

Determinant:

$$\begin{aligned}
\det(A) &= \det(P^{-1}LU) \\
&= \underbrace{\det(P^{-1})}_{\pm 1} \quad \underbrace{\det(L)}_{\prod_{i=1}^{n} l_{ii} = 1} \quad \underbrace{\det(U)}_{\prod_{i=1}^{n} u_{ii}} \\
&= \pm \prod_{i=1}^{n} u_{ii} \\
\text{sign} &= (-1)^{\#\text{ transposition of rows}} \quad \text{(keep track)} \\
&= \text{sign of the permutation}
\end{aligned}$$

7.7 Note on complex matrices

- LU decomposition:

  Crout's algorithm goes through in the obvious way, with complex arithmetic used as needed (in the search of the pivot complex modules substitutes for absolute value)

- complex QR:

  $0 \neq x \in \mathbb{C}^n,\ x_1 = re^{i\Theta}$ with $r, \Theta \in \mathbb{R}$

  if $u = x \pm e^{i\Theta}|x|e_1$ and $P = \mathbb{I} - \dfrac{uu^+}{H},\ H = \dfrac{u^+u}{2}$

  then $Px = \mp e^{i\Theta}|x|e_1 \quad (|x| = x^+x)$

P is unitary:

$$\begin{aligned} P^+P &= (\mathbb{I} - \frac{uu^+}{H})(\mathbb{I} - \frac{uu^+}{H}) \\ &= \mathbb{I} - 2\frac{uu^+}{H} + \frac{uu^+uu^+}{(\frac{1}{2}u^+u)^2} \\ &= \mathbb{I} - \frac{2uu^+}{H} + \frac{2uu^+}{H} \\ &= \mathbb{I} \end{aligned}$$

the sign can be determined to maximize H for the sake of stability

8 Fitting of data

In this chapter we mainly follow [1].

8.1 Example

Y = population in the USA 1900...2000

fit $Y = f(x)$ for models: x = time in years -1900

1. $f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$
   polynomial of degree n; fit parameters $a_0 \ldots a_n$
2. $f(x) = \alpha e^{x/T_0}$
   exponential; fit parameters α, T_0

Given $x_1, x_2, \ldots, x_N$ and the corresponding values Y_i of Y: $y_1 \pm \sigma_1, y_2 \pm \sigma_2, \ldots, y_N \pm \sigma_N$ (measurements at times x_i), where σ_i are the uncertainties (measurement errors) of the values y_i:

determine the fit parameters through maximum likelihood fit

$$\Rightarrow \text{minimization of } \chi^2 = \sum_{i=1}^{N} \frac{(y_i - f(x_i))^2}{\sigma_i^2}$$

8.2 Normal distributed measurements, χ^2

Assumption: errors of the measurements $y_i - Y_i$ are normal or Gaussian distributed

if the true value is Y,
the probability to measure the value y in the interval $[y, y + dy]$ is

$$P_\sigma(y)dy = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y-Y)^2}{2\sigma^2}} dy\,,$$

with maximum at $y = Y$, uncertainty σ, normalization: $\int_{-\infty}^{\infty} dy\, P_\sigma(y) = 1$.

"y is a measurement of Y with error σ"

$$\begin{aligned}
\langle (y-Y)^2 \rangle &= \int_{-\infty}^{\infty} dy\,(y-Y)^2 P_\sigma(y) = \int_{-\infty}^{\infty} du u^2 \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{u^2}{2\sigma^2}} \\
&= 2\int_0^{\infty} du u^2 \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{u^2}{2\sigma^2}} = \frac{2}{\sqrt{2\pi}\sigma}\int_0^{\infty} dt \sigma^2 e^{-t}\sqrt{2\sigma^2 t} \qquad (t = \frac{u^2}{2\sigma^2}) \\
&= \frac{2\sigma^2}{\sqrt{\pi}}\int_0^{\infty} dt e^{-t}\sqrt{t} = \frac{2\sigma^2}{\sqrt{\pi}}\Gamma(\frac{3}{2}) = \frac{2\sigma^2}{\sqrt{\pi}}\frac{1}{2}\underbrace{\Gamma(\frac{1}{2})}_{=\sqrt{\pi}} = \sigma^2
\end{aligned}$$

Remark: if the errors originate from the addition of several statistically fluctuating (random) influences then, according to the central limit theorem, the probability distribution of the sum of these random deviations converges almost always to a normal distribution

Define $p_n := p\{|y-Y| < n\sigma\} = \int_{Y-n\sigma}^{Y+n\sigma} dy\, P_\sigma(y)$

$p_1 = 0.68, p_2 = 0.95, p_3 = 0.99, \ldots$

Suppose: N measurements (data points) (x_i, y_i) with uncertainties σ_i

Define $\chi^2 = \sum_{i=1}^{N} \frac{(y_i - Y_i)^2}{\sigma_i^2}$ with $Y_i = f(x_i)$

What is the probability distribution of χ^2?

Consider N **independent** data points (x_i, y_i):

The probability that $\chi^2 \le C$ is

$$Q(C) = p\{\chi^2 \le C\} = \int d^N y\, P_{\sigma_1}(y_1) \ldots P_{\sigma_N}(y_N)\Theta(C - \chi^2)\,,$$

with normalization $Q(\infty) = 1$. We substitute $z_i = (y_i - Y_i)/\sigma_i$,

$$Q(C) = (2\pi)^{-N/2} \int d^N z\, e^{-\frac{1}{2}z^2}\Theta(C - z^2)\,, \quad z^2 \equiv \sum_{i=1}^{N} z_i^2\,.$$

Changing to polar coordinates in N dimensions

$$Q(C) = \frac{|S_{N-1}|}{(2\pi)^{N/2}} \int_0^{\sqrt{C}} dr\, r^{N-1} e^{-\frac{1}{2}r^2}\,,$$

where $|S_{N-1}|$ is the surface of a ball of unit radius in N dimensions ($|S_1| = 2\pi$, $|S_2| = 4\pi$). It can be shown

$$|S_{N-1}| = \frac{2\pi^{N/2}}{\Gamma(N/2)}\,, \quad \Gamma(z) = \int_0^\infty dt\, t^{z-1} e^{-t}\,.$$

Changing variable $t = r^2/2$

$$Q(C) = \frac{|S_{N-1}|}{2\pi^{N/2}} \int_0^{C/2} dt\, t^{\frac{N}{2}-1} e^{-t} = \Gamma_{\rm inc}(\frac{C}{2}, \frac{N}{2})\,, \tag{31}$$

where

$$\Gamma_{\rm inc}(A, b) = \frac{\int_0^A dt\, t^{b-1} e^{-t}}{\int_0^\infty dt\, t^{b-1} e^{-t}}$$

is the incomplete gamma function (MATLAB: `gammainc(A,b)`).
The probability $1 - Q(C)$ that $\chi^2 > C = N$ vanishes like a step function for N large, see Fig. 7.

8.3 Fits

Fits of data points (x_i, y_i) with uncertainties σ_i. The functional form $Y(x)$ is known but it contains a number of free parameters:

$$Y = f(x; \underbrace{c_1, \ldots, c_R}_{\text{free parameters}}) \rightarrow \text{determine } \{c_1 \ldots c_R\}\text{: goodness of the fit?}$$

8.3.1 Least Squares

We construct $\chi^2 = \sum_{i=1}^{N} \frac{(y_i - f(x_i; c_\alpha))^2}{\sigma_i^2}$

Least squares fit: minimize χ^2

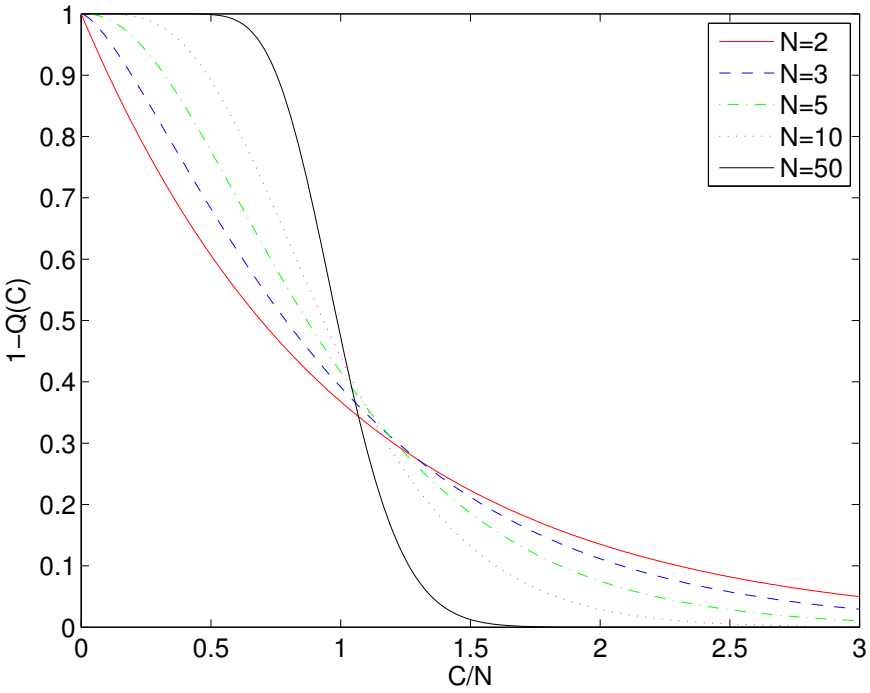


Figure 7: Plot of $1-Q(C)$ (= probability for $\chi^2 > C$) as a function of C/N.

Theory: probability for χ^2 follows the curves $Q(C)$ with N replaced by

$$N_{\rm dof} = N - R = \#\text{ degrees of freedom}$$

($N = \#$ data points and $R = \#$ parameters)

If after minimization we find $\dfrac{\chi^2}{N_{\rm dof}} >> 1$, the fit is **not** good!

8.3.2 Linear fits

We follow [2]. Cosider a fit function which is a *linear combination* of arbitrary fixed functions $g_1(x), \ldots, g_R(x)$ of x, called the basis functions

$$f(x; c_1, \ldots, c_R) = \sum_{k=1}^{R} c_k g_k(x) \,.$$

For example $g_k(x) = x^{k-1} \Rightarrow f$ is a polynomial of degree $R-1$. Define a merit function

$$\chi^2 = \sum_{i=1}^{N} \frac{(y_i - \sum_{k=1}^{R} c_k g_k(x_i))^2}{\sigma_i^2} \tag{32}$$

The best parameters $c_1, \ldots, c_R$ are those that minimize χ^2. Define a $N \times R$ matrix $\mathbf{A}$ whose components are

$$A_{ij} = \frac{g_j(x_i)}{\sigma_i} \,.$$

It is called the design matrix. Notice that $N \geq R$. Also define a vector $\mathbf{b}$ of length N

$$b_i = \frac{y_i}{\sigma_i}$$

and a vector $\mathbf{c}$ of length R whose components are the parameters c_i to be fitted.

The minimum of Eq. 32 occurs when $\frac{\partial \chi^2}{\partial c_k} \stackrel{!}{=} 0$ for $k = 1, \ldots, R$:

$$0 \stackrel{!}{=} \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \left[y_i - \sum_{j=1}^{R} c_j g_j(x_i) \right] g_k(x_i) \qquad k = 1, \ldots, R$$

This can be written as a matrix equation

$$(\mathbf{A}^T \cdot \mathbf{A}) \cdot \mathbf{c} = \mathbf{A}^T \cdot \mathbf{b} \,. \tag{33}$$

Notice that $\mathbf{A}^T \cdot \mathbf{A}$ is a $R \times R$ positive symmetric matrix. Eq.33 are called the *normal equations* of the least squares fit. They can be solved for the parameters $\mathbf{c}$ by standard methods, e.g. LU decomposition and backsubstitution

$$c_j = \sum_{k=1}^{R} (\mathbf{A}^T \cdot \mathbf{A})^{-1}_{jk} (\mathbf{A}^T \cdot \mathbf{b})_k = \sum_{k=1}^{R} (\mathbf{A}^T \cdot \mathbf{A})^{-1}_{jk} \left[\sum_{i=1}^{N} \frac{y_i g_k(x_i)}{\sigma_i^2} \right] \tag{34}$$

The variances (squared standard uncertainties) $\sigma^2_{c_j}$ of the parameters $\mathbf{c}$ can be computed by error propagation:

$$\sigma^2_{c_j} = \sum_{i=1}^{N} \sigma_i^2 \left(\frac{\partial c_j}{\partial y_i} \right)^2$$

and from Eq. 33. Noticing that $\mathbf{A}^T \cdot \mathbf{A}$ is independent of y_i we get

$$\frac{\partial c_j}{\partial y_i} = \sum_{k=1}^{R} (\mathbf{A}^T \cdot \mathbf{A})^{-1}_{jk} g_k(x_i)/\sigma_i^2$$

end finally

$$\begin{aligned} \sigma^2_{c_j} &= \sum_{k=1}^{R}\sum_{l=1}^{R} (\mathbf{A}^T \cdot \mathbf{A})^{-1}_{jk} (\mathbf{A}^T \cdot \mathbf{A})^{-1}_{jl} \left[\sum_{i=1}^{N} \frac{g_k(x_i) g_l(x_i)}{\sigma_i^2} \right] \\ &= \sum_{k=1}^{R}\sum_{l=1}^{R} (\mathbf{A}^T \cdot \mathbf{A})^{-1}_{jk} (\mathbf{A}^T \cdot \mathbf{A})^{-1}_{jl} (\mathbf{A}^T \cdot \mathbf{A})_{kl} \\ &= (\mathbf{A}^T \cdot \mathbf{A})^{-1}_{jj} \end{aligned} \tag{35}$$

So the errors squared of the fit parameters are the diagonal elements $(\mathbf{A}^T \cdot \mathbf{A})^{-1}_{jj}$. The off-diagonal elements $(\mathbf{A}^T \cdot \mathbf{A})^{-1}_{jk}$, $j \neq k$ are the covariances between c_j and c_k. The affect the errors of the fit function $f(x; c_1, \ldots, c_R) = \sum_{k=1}^{R} c_k g_k(x)$. Error propagation gives

$$\sigma^2_{f(x)} = \sum_{i=1}^{N} \left(\frac{\partial f(x)}{\partial y_i} \right)^2 \sigma_i^2 = \sum_{j,k=1}^{R} \frac{\partial f(x)}{\partial c_j} (\mathbf{A}^T \cdot \mathbf{A})^{-1}_{jk} \frac{\partial f(x)}{\partial c_k} \tag{36}$$

where we used $\frac{\partial f(x)}{\partial y_i} = \sum_{j=1}^{R} \frac{\partial f(x)}{\partial c_j} \frac{\partial c_j}{\partial y_i}$ (and $\frac{\partial f(x)}{\partial c_j} = g_j(x)$). The result in Eq. 36 is different from the "naive" expression $\sum_{k=1}^{R} \left(\frac{\partial f(x)}{\partial c_k} \right)^2 \sigma^2_{c_k}$, see Eq. 35.

8.3.3 Fits with a non-linear parameter

If all fit parameters enter f in a non-linear way (*e.g.*, $\cos(c_1 x)$) we have a difficult optimization problem...here some special cases:

1. in physics we often have $f(x) = ce^{-\lambda x}$, λ appears to be non-linear, but consider instead $\ln f(x) = \ln(c) - \lambda x$
   $\Rightarrow$ linear fit parameters $c_1 = \ln(c), c_2 = -\lambda$.
   take logarithm of data, *i.e.*, $y_i \to \ln(y_i),\ \sigma_i \to \sigma_i^{\ln} = \dfrac{\sigma_i}{|y_i|}$ $(\delta(\ln y) = \dfrac{\delta y}{y})$

2. $f(x) = a + ce^{-\lambda x}$ with constant term a; general case with one non-linear fit parameter λ:

$$f(x; c_1, \ldots, c_{R-1}; \lambda) = \sum_{\alpha=1}^{R-1} c_\alpha g_\alpha(x; \lambda)$$

   For any fixed value of λ solve the linear problem according to sect. 8.3.2, get values $\overline{c}_\alpha(\lambda)$ for $\alpha = 1, \ldots, R-1$. Write a pogram to plot

$$F(\lambda) = \sum_{i=1}^{N} \frac{(y_i - \sum_{\alpha=1}^{R-1} c_\alpha g_\alpha(x_i; \lambda))^2}{\sigma_i^2}\bigg|_{c_\alpha = \overline{c}_\alpha(\lambda)}$$

   and find the minimum (it can be identified by eye). If there $F = \chi^2$ is not $>> N_{\rm dof}$ then the fit is plausible. Error estimate of λ: take interval around minimum where at the interval ends F is augmented by 1 ($\Delta F = 1$). With this procedure one can check if there are multiple almost degenerate minima. Knowledge of the problem considered helps to identify the "right" minimum.

8.4 Practical considerations

How do we know the uncertainty σ_i of the data point (x_i, y_i)?

One possibility is to perform a series of measurements, *i.e.*, at each $x = x_i$ we repeat the measurement with results $y_{i,a},\ a = 1, \ldots, n_i$.
From the variance of the results we get

$$\sigma_i^2 = \frac{1}{n_i - 1} \sum_{\alpha=1}^{n_1} (y_{i,a} - \overline{y}_i)^2, \qquad \overline{y}_i = \frac{1}{n_i} \sum_{a=1}^{n_i} y_{i,a}\,.$$

The best estimate is the mean value $\overline{y}_i$. Its uncertainty is given by

$$\overline{\sigma}_i = \frac{\sigma_i}{\sqrt{n_i}}\,,$$

assuming that each measurement is Gaussian distributed.

$$
\begin{aligned}
\overline{y} &= \frac{1}{n}\sum_{i=1}^{n} y_i, \qquad P_\sigma(y_i)dy_i = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_i - Y)^2}{2\sigma^2}} dy_i \\
\overline{\sigma}^2 &= \langle (\frac{1}{n}\sum_{i=1}^{n} y_i - Y)^2 \rangle = \langle [\frac{1}{n}\sum_{i=1}^{n} (y_i - Y)]^2 \rangle \\
&= \int_{-\infty}^{\infty} dy_1 \ldots dy_n [\frac{1}{n}\sum_{i=1}^{n} (y_i - Y)]^2 P_\sigma(y_1) \ldots P_\sigma(y_n) \\
&= \int_{-\infty}^{\infty} du_1 \ldots du_n (\frac{1}{n}\sum_{i=1}^{n} u_i)^2 \frac{1}{(2\pi)^{n/2}\sigma^n} e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{n} u_i^2} \\
&\qquad\qquad \text{mixed terms } u_i u_j \;\; i \neq j \text{ vanish} \\
&= \int_{-\infty}^{\infty} du_1 \ldots du_n \frac{1}{n^2}\sum_{i=1}^{n} u_i^2 \ldots = \frac{1}{n^2}\sum_{i=1}^{n} \sigma^2 = \frac{\sigma^2}{n}
\end{aligned}
$$

References

[1] U. Wolff and B. Bunk, *Computational Physics I.* http://www.physik.hu-berlin.de/com/teachingandseminars

[2] Press, Teukolsky, Vetterling and Flannery, *Numerical Recipes*, Camebridge University Press

[3] N. J. Giordano and H. Nakanishi,
Computational Physics, 2nd edition, Pearson Prentice Hall, 2006.

[4] Landau, Pez and Bordeianu, *A Survey of Computational Physics*, Princeton University Press

[5] L. D. Landau and E. M. Lifshitz, Mechanics.

[6] T. Lippert, A. Seyfried, A. Bode and K. Schilling, "Hypersystolic parallel computing", http://de.arxiv.org/abs/hep-lat/9507021.

[7] G. H. Golub and C. F. Van Loan, *Matrix computations*, Johns Hopkins University Press Baltimore, MD, USA (1996) http://web.mit.edu/ehliu/Public/sclark/Golub%20G.H.,%20Van%20Loan%20C.F.-%20Matrix%20Computations.pdf